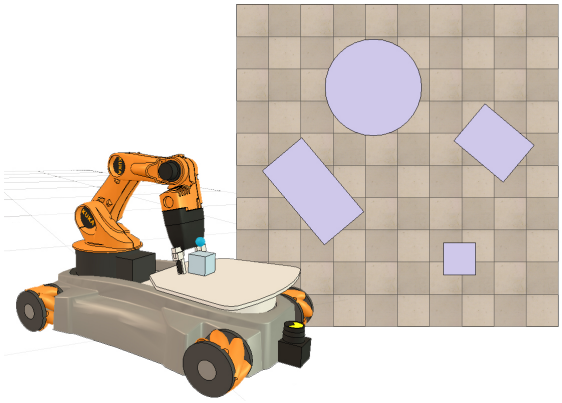
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia de
Sistema de Potência e Automação



Planeamento de Rota em Ambiente Desconhecido

João Paulo Gouveia Garrinhas
(Licenciado em Engenharia Eletrotécnica)

Dissertação para a obtenção do grau de Mestre
em Engenharia Eletrotécnica – Ramo Automação e Eletrónica Industrial

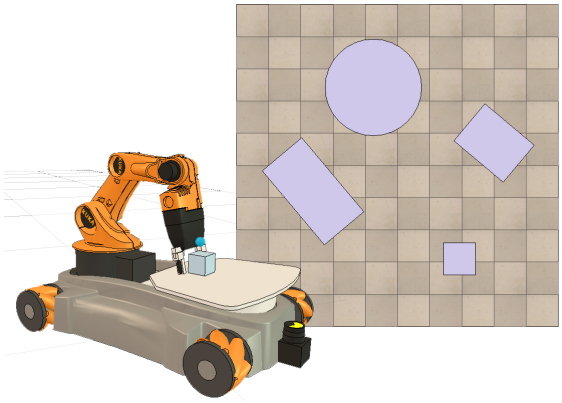
Orientadores:
Prof. Dr. Fernando Manuel Fernandes Melício

Júri:
Presidente: José Manuel Prista do Valle Cardoso Igreja
1º Vogal: Fernando Manuel Fernandes Melício
2º Vogal: Maria da Graça Vieira Brito Almeida

Abril de 2015

INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia de
Sistema de Potência e Automação



Planeamento de Rota em Ambiente Desconhecido

João Paulo Gouveia Garrinhas
(Licenciado em Engenharia Eletrotécnica)

Dissertação para a obtenção do grau de Mestre
em Engenharia Eletrotécnica – Ramo Automação e Eletrónica Industrial

Orientadores:
Prof. Dr. Fernando Manuel Fernandes Melício

Júri:
Presidente: José Manuel Prista do Valle Cardoso Igreja
1º Vogal: Fernando Manuel Fernandes Melício
2º Vogal: Maria da Graça Vieira Brito Almeida

Abril de 2015

Agradecimentos

A realização desta tese não seria possível sem a existência do meu orientador Fernando Melício , que dispôs de todo o conhecimento e ajuda no decorrer do trabalho.

Um agradecimento especial a toda a minha família, pela dedicação, apoio e confiança que depositaram em mim. A minha gratidão vai também para todos aqueles que contribuíram de forma direta ou indireta para a conclusão deste trabalho, que sempre se disponibilizaram para me ajudar.

Resumo

A robótica é o ramo da ciência que tenta desenvolver métodos para que seja possível realizar ações de forma autónoma, sem recorrer à mão humana.

Neste sentido, este trabalho tem como objetivo o estudo do planeamento de um caminho possível sem se conhecer à partida o mapa onde um robot omnidireccional é inserido dando apenas as coordenadas do ponto final. Neste trabalho tanto o robot utilizado como os sensores que se escolheram são dispositivos comuns e que se podem encontrar facilmente. O sistema terá de ser capaz de mapear o ambiente e projetar a sua rota de modo a não existir qualquer colisão entre o robot e qualquer objeto que exista no meio ambiente.

Para tal o robot possuirá apenas um sensor distância e um sensor GPS ("Global Positioning System"), que juntamente com o código de navegação será possível ao robot deslocar-se no ambiente sem qualquer intervenção externa.

Palavras Chave

- Robótica
- Robot móvel
- Algoritmo
- GPS
- Planeamento de Rota

Abstract

Robotics is the branch of science that attempts to develop methods so that robots can perform actions independently, without resorting to human hand.

The primary goal of this work is to find a route in an unknown landscape where it is only known the GPS coordinates of an objective. In this thesis, the omnidirectional robot and used sensors are common devices which can be found easily. The system must be able to map the environment and design your route, so that there is no collision between the robot and any object that exists in the environment.

For this purpose, the robot will have only one distance sensor and GPS sensor, together with the navigation code will be possible to move the robot in environment without any external intervention.

Keywords

- Robotics
- Mobile Robot
- Algorithm
- GPS
- Motion Planning

Índice

Lista de Figuras

Lista de Tabelas

1

Introdução

Desde a invenção da roda que o Homem desenvolve técnicas, utensílios e mecanismos para desempenhar tarefas, aumentando a sua segurança e ajudando-o na realização das mesmas. Focado nessa ideia, criou várias ciências para tentar antecipar, controlar e realizar ações, como o caso da robótica, que desenvolve processos para a realização de tarefas.

A robótica é um dos principais ramos da ciência responsável pelo desenvolvimento de técnicas e mecanismos que ajudam, ou fazem na íntegra diversas funções. O principal objetivo desta área é ter o melhor desempenho e assegurar a segurança de todos os intervenientes no processo.

1.1 Motivação e Enquadramento

A área da robótica, e tudo o que ela engloba, é muito abrangente, com bastante potencial para investigação. Devido às suas potencialidades, há um leque gigante de opções que se podem tomar. Podendo dar asas à imaginação para criar novos projetos nas mais diversas áreas.

Junto destes aspetos, está, ainda, o meu interesse na automação de sistemas, tanto móveis (condução autónoma) como fixos (linhas de montagens). Querendo aprofundar os conhecimentos de como se processa estes mecanismos, com inteligência artificial. Assim, torna-se inevitável a escolha deste tema.

Este trabalho será concebido sempre tendo uma base de simulação, porém foi sempre tida em conta a sua conceção e realização em termos práticos. Pretendendo neste trabalho, usar dispositivos recentes existentes no mercado, de forma a que o projeto esteja atualizado e possa ser utilizado para possíveis desenvolvimentos futuros.

A dissertação em causa visa usar condução autónoma num robot/veículo móvel terrestre, de forma a conseguir navegar num ambiente totalmente desconhecido. De modo a que seja capaz de ir de determinado ponto a outro sem que no caminho haja qualquer colisão.

1.2 Objetivos

Este trabalho tem como principal objetivo equipar um robot omnidireccional com GPS e sensores laser, que o tornem apto a navegar num meio ambiente sem que ocorra qualquer colisão entre ele e os objectos que aí se encontrem.

1.3 Metodologia de Trabalho

Este trabalho foi desenvolvido por etapas, das quais se pode destacar as em baixo mencionadas, sendo estas as mais representativas para os avanços deste projeto.

- Pesquisa por plataformas de simulação e sua exploração;
- Pesquisa bibliográfica;
- Estudo sobre o funcionamento dos robots móveis, dando relevância aos robots móveis omnidirecionais;
- Criação do ambiente de simulação e seleção dos dispositivos;
- Avaliação das formas de controlo para desempenharem a função pretendida;
- Comunicação das plataformas;
- Desenvolvimento do sistema de navegação autónoma em linha reta;
- Expansão do sistema com a capacidade de deteção de objetos;
- Expansão do Sistema com a capacidade de retroação;
- Testes, conclusões e análise das possibilidades de desenvolvimento futuro.

1.4 Estrutura

Este documento está dividido em seis capítulos.

No presente capitulo, **Capítulo 1 - Introdução** para além desta secção, é realizada uma introdução ao trabalho, assim como, a apresentação das motivações, objetivos e a metodologia de trabalho.

Capítulo 2 - Estado da Arte Neste capítulo é apresentada a evolução e o estado da arte das áreas científicas envolvidas neste trabalho.

Capítulo 3 - Planeamento Descreve os métodos usados e os respetivos conceitos teóricos associados, necessários ao desenvolvimento e implementação do sistema.

Capítulo 4 - Implementação Apresenta os equipamentos usados e descreve a implementação imposta no sistema de controlo.

Capítulo 5 - Resultados São apresentados e discutidos os resultados obtidos nos testes efetuados.

Capítulo 6 - Conclusões Finalmente, neste capítulo são apresentadas as principais conclusões sobre os resultados obtidos, é feita um balanço das limitações da metodologia adotada e são avaliadas possíveis perspetivas de desenvolvimento futuro do trabalho realizado.

2

Estado da Arte

Neste capitulo será apresentado todo o conhecimento teórico necessário para compreender este projeto.

Primeiramente irá ser abordada a temática da robótica, realizando uma breve introdução e enunciado o seu surgimento e factos históricos. Seguidamente serão expostos os vários tipos de robot e suas principais características e explorados em pormenor os factos associados aos robots móveis terrestres com rodas.
De seguida, dar-se-á conhecimento de vários algoritmos usados para controlar os robos móveis, enunciando os mais utilizados. Nesta fase, será exposto o modo de funcionamento de cada algoritmo e a sua funcionalidade.

2.1 Robótica

O termo robot foi utilizado pela primeira vez por um escritor checo, Karel Čapek, na sua peça *Rossum's Universal Robots* (R.U.R.) de 1921 [30]. Este termo provém da palavra checa *robota* cujo significado é "trabalho forçado", porém com a evolução da linguagem e da tecnologia o termo robot tornou-se uma palavra universal e com significado bastante abrangente. Houve, por isso, necessidade de formalizar a aceção da palavra. Assim, de acordo com a *International Organization for Standardization (ISO)* definiu-se "robot" como sendo: "um mecanismo programável em dois ou mais eixos de liberdade, movendo-se dentro de um ambiente para executar tarefas. O robot tem de incluir, por isso, um sistema de controlo e uma interface de controlo"[14].

O uso de mecanismos e dispositivos para ajudar ou substituir o Homem a desempenhar as suas ações data das eras antes de Cristo. Os egípcios e os gregos desenvolveram mecanismos acionados pelo homem que desempenhavam as suas funções exigindo menores capacidades por parte de quem executava as tarefas, porém esses mecanismos não eram programáveis, não sendo por isso robots.

O primeiro robot industrial é utilizado na década de 40, com a utilização dos robots cartesianos, após a Segunda Guerra Mundial. A perigosidade associada aos produtos radioativos fez com que houvesse necessidade da sua utilização. Contudo, o surgimento do que se pode chamar o primeiro robot móvel só aparece na década de 50, por meio de um curioso neurofisiologista. Com a criação destes últimos, verifica-se a existência de dois tipos de robots, segundo a *ISO* distingue-se: robot manipulador ("*industrial robot* ") e robot móvel ("*service robot* ")[14].

2.1.1 Robot Manipuladores

Os robots manipuladores foram desenvolvidos com o pressuposto de replicar os movimentos de um braço humano. A construção de um robot deste tipo é formado apenas por elos e juntas, sendo que na junção de dois elos existirá sempre uma junta, tal como acontece no braço humano, com os ossos e articulações. Desta forma, A *International Organization for Standardization* definiu um robot manipulador como: "um robot manipulador multiuso programável em três ou mais eixos que podem ser tanto fixos no local ou móvel para uso em aplicações de automação industrial. É controlado automaticamente , podendo ser reprogramável. Incluído no robot manipulador estão os atuadores e o sistema de controlo, bem como todos os eixos adicionais" [14].

Os primeiros robots manipuladores a serem comercializados foram os robot cartesianos, estes robots desenvolvidos após a segunda guerra mundial, eram de grandes dimensões, tinham como principal objetivo manipular materiais perigosos, como os materiais radioativos. Uma das empresas a explorar este negócio foi sem dúvida a *General Mills Corporation - USA*, com o seu robot *Gantry-robot*, em 1950. Mais tarde,em 1957, aparece o *Planetboy* que se rege por coordenadas polares[26].

Todavia, o primeiro braço robótico a ser utilizado para fins industriais foi o *Unimate* (Fig. 2.1). Este robot foi desenvolvido por George Devol e Joseph Engelberger na empresa americana *Unimation Inc.*, em 1959. O robot apresentava seis eixos programáveis acionados por atuadores hidráulicos. Introduzido na industria em 1962, sendo a sua principal função soldar a carroçaria dos automóveis, na *General Motors*[13]. Contudo, este robot, tal como todos os robots, poderia ser reprogramado para fazer outras tarefas, usando um órgão terminal adequado.

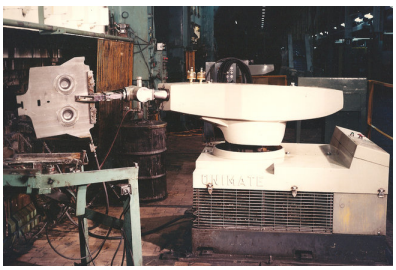


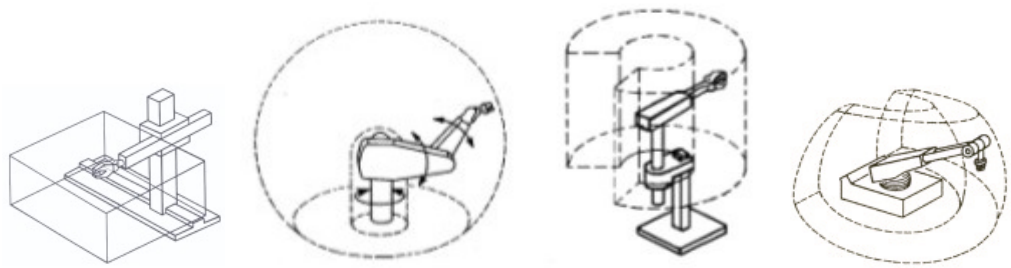
Figura 2.1: Robot *Unimate*

Após a primeira utilização desta ferramenta verificou-se a sua potencialidade. Assim sendo, nos anos 60 e 70 desenvolveram-se vários trabalhos de investigação por parte da *Cincinatti Millacron*, introduzindo um micro-controlador e o eixo de revolução, criando o T^3 -*The Tomorrow Tool* comercializado a partir de 1973[13]. A partir deste, outros surgiram combinando várias juntas, utilizando mais ou menos sensores de interação com o ambiente, dando origem a braços robóticos cada vez mais complexos, com maior mobilidade, rapidez e precisão.

Por norma, os robots manipuladores são atuados por motores elétricos, ou por sistemas hidráulicos quando é necessário maiores esforços. Juntamente com o sistema de atuação é imposto o sistema de controlo, que lê não só os sensores responsáveis pela leitura do ambiente, mas também lê os sensores internos do robot.

Apesar da sua sofisticação e elevado potencial estes robots são sobretudo usados em linhas de montagem, em meios industriais, para desenvolverem tarefas precisas mas repetitivas. A sua utilização nesses ambientes deve-se sobretudo devido à sua limitação espacial. Visto que, normalmente, os robots são construidos a partir de uma base fixa, ficando limitados ao comprimento máximo do braço. Em processos industriais, as tarefas realizadas são repetitivas e precisas, por isso a limitação do braço é tida em conta na construção da linha de montagem.

Atendendo à limitação de cada robot manipulador e observando o seu volume de trabalho podemos designá-los de maneira diferente. (Fig.2.2)



(a) cartesiano (b) articulado (esférico) (c) cilíndrica (d) polar

Figura 2.2: Volumes de trabalho

2.1.2 Robot Móveis

A potencialidade dos robots manipuladores é inegável, porém a sua limitação espacial devido à sua incapacidade de se deslocarem torna-se um fator que induz ao desenvolvimento dos robots móveis. Os robots móveis por sua vez, são um equipamento bastante útil, dado que podem viajar no ambiente para executar as suas tarefas. Podemos ainda acoplar um robot manipulador e as

suas potencialidades podem ser ainda maiores.

Estes robots apareceram nos anos 50 com o neurofisiologista, William Grey Walter, que fascinado pela robótica inventa o *Elmer and Elsie* (abreviatura de *ELectro MEchanical Robots, Light Sensitive*). Eram robots simples que tinham como principal função explorar o ambiente utilizando um sensor de luz, um sensor de contacto, um motor propulsor, uma roda de direção e um computador analógico com dois tubos de vácuo[20][24].

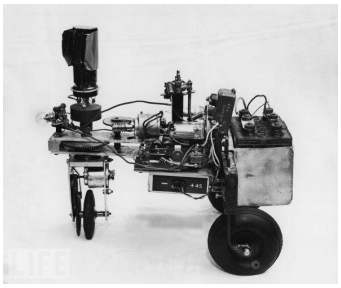


Figura 2.3: Elsie

Em 1969, aparece o primeiro robot controlado por sensores de visão. Este robot criado pela universidade de *Stanford* ,denominado por *Shakey*, tinha como objetivos reconhecer um objeto através da visão, encontrar um caminho para o objeto e podia executar algumas ações com o objeto (empurrá-lo)[24]. Na mesma universidade, em 1967, começou-se a desenvolver um novo robot, *Stanford Cart*. Este foi capaz de atravessar uma sala cheia de cadeiras sem a intervenção humana em 1979. Equipado com *stereo vision* cujas imagens eram processadas num computador que dava a distância do robot para os objetos.
Em França, no Laboratório de Arquitetura e de Análise de Sistemas - LAAS, em 1977, surge o primeiro *Hilare*, robot composto por duas rodas motrizes e uma roda livre. Este robot foi evoluindo ao longo dos tempos até 1992, já era composto por duas rodas motrizes e 4 rodas livres, sonares, câmera e podia ser acoplado um braço robótico [24].

Os robots moveis com rodas sempre foram muito populares. Todavia, em 1989, aparece o primeiro robot com patas. O *Ghenghis* robot hexapod criado no *Massachusetts Institute of Technology - MIT* usava 4 microprocessadores, 22 sensores e 12 servo motores, não pesando mais que 1kg[8]. Este robot deslocava-se de forma semelhante a um inseto, estando mais próximo de movimentos que podemos encontrar no mundo real. Após este, outros surgiram , como o *Dante II* desenvolvido pela *Carnegie Mellon University*, entre 1993-1994, usado para explorar vulcões na Antártida e no Alasca. Composto por 8 patas, foi o primeiro robot a usar o rapel. Realizava comunicação por intermédio de fibra ótica, e tinha câmaras e sensores nas pernas e disponibili-

zava ainda de um sensor de gás e um Computador em VME bus (Sparc and MC68000)[9][24].

Estes últimos desenvolvimentos foram importantes para a criação dos robots antropomorfos, robots com fisionomia parecida com o Homem.

Apesar do elevado desenvolvimento no meio terrestre, existem ainda outras gamas de robots capazes de se deslocarem noutros meios. Os robots móveis são construidos a pensar no ambiente onde se movem e suas funções, podendo ser distinguidos em diferentes categorias:

- Terrestre;
  - Rodas (Fig. 2.4a)
  - Pernas
  - Esteiras
- Aquáticos;
  - submersiveis
  - flutuantes (Fig. 2.4c)
- Aéreos. (Fig. 2.4b)

(a) Terrestre-KUKA Youbot (b) Aério-Parrot A.R.Drone 2.0 (c) Aquático-Ziphius

Figura 2.4: Exemplos de robot moveis

Tendo em conta todos os tipos de robots, para este projeto será utilizado um robot terrestre com rodas. Assim, posteriormente, seguir-se-á o desenvolvimento das principais características destes tipos de robots.

Tipos de Rodas

Os robots móveis terrestres podem mover-se de diferentes maneiras. A utilização das rodas é bastante usual, visto que é a forma de locomoção mais rápida e com menos consumo de energia, comparando-a com outros meios mecânicos terrestres (pernas ou esteiras). Atendendo a esse facto o robot deste trabalho mover-se-á por intermédio de rodas. Neste ponto, serão abordados

os vários tipos de rodas utilizados em robótica .

Nas futuras descrições irá assumir-se que as rodas giram sempre em torno dos seus eixos e estarão sempre na perpendicular ao chão, descrevendo um ângulo de 90º entre o plano da roda e o solo. Assume-se ainda que a sua tração é total, não havendo escorregamento associado, o que significa que a velocidade é nula no ponto de contacto com o chão.

Primeiramente, é necessário ter em consideração se as rodas concedem tração ou não ao veículo, indicando se são:

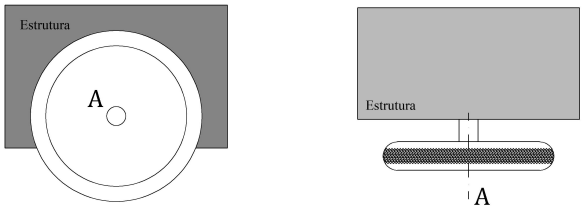
Rodas Ativas são todas as rodas motoras ou responsáveis pela direção do robot.

Rodas Passivas responsáveis pela estabilidade do veículo, denominadas também de rodas livres.

Quanto à direção, as rodas podem ser:

Rodas unidirecionais As rodas unidirecionais, como o nome indica, apenas se deslocam numa direção. Deste modo, houve necessidade de refazer a roda para que esta pudesse tomar todas as direções. Existem várias soluções que se podem ter em conta para responder o mais eficazmente às adversidades do ambiente. Sendo as mais utilizadas em robótica descritas a baixo:

- Roda Fixa - Este tipo de roda apenas dispõe de um eixo (representado na fig. 2.5 com a letra "A"), fixo na estrutura do veículo. São, sobretudo, usadas como rodas motrizes, pois podem ser acopladas diretamente ao motor.



(a) Vista Lateral (b) Vista de Cima

Figura 2.5: Roda Fixa

- Roda Orientada Centrada - As rodas centradas encontram-se destacadas debaixo do veículo. Estas rodas possuem dois eixos, um fixo na estrutura do veículo (eixo "A"), que confere a direção. A roda, por sua vez, gira em torno de um eixo (eixo "B") que concede o movimento ao veículo. (Fig.2.6)

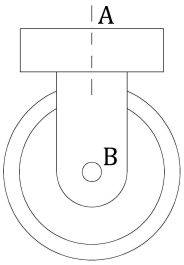


Figura 2.6: Roda Orientada Centrada

- Roda Orientada não Centrada - À semelhança da roda orientada centrada, também este tipo de roda tem dois eixos, é igualmente colocada por baixo do veículo. Ambos os eixos têm a mesma função, contudo existe um "*offset*" ou uma deslocação do eixo B, estas rodas são designadas também de "*caster wheel*". (Fig.2.7)

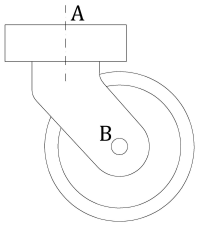


Figura 2.7: Roda Orientada Não Centrada

Rodas Omnidirecionais Este tipo de rodas surgiram para colmatar o problema da direção nas rodas unidirecionais, uma vez que estas podem deslocar-se em qualquer direção sem necessidade de serem reorientadas.

Em robótica este tipo de rodas exige um maior conhecimento de controlo, pois é necessário uma maior coordenação da velocidade das rodas. Neste grupo de rodas existe dois conjuntos fundamentais:

- Rodas omnidirecionais tradicionais - são compostas por pequenas rodas passivas em torno da circunferência que forma a roda propriamente dita, cujo o eixo principal é uma junta ativa. As pequenas rodas passivas podem ser dispostas de dois tipos diferentes, originando dois tipos de rodas omnidirecionais:

  – Rodas Omnidirecionais Convencionais: Neste tipo, as pequenas rodas passivas estão dispostas de maneira a que o seu eixo seja paralelo ao plano da roda (fig. 2.8a);

– *Mecanum Wheel* as pequenas rodas passivas são concorrentes ao plano da roda, formando um ângulo de 45º entre o eixo das rodas passivas e o plano da roda (fig. 2.8b).

(a) Roda Omnidirecionais Convencionais

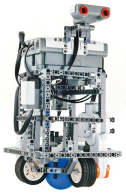
(b) *Mecanum wheel*

Figura 2.8: Rodas Omnidirecionais

Os dois tipos de rodas apresentados anteriormente são as mais usadas em termos de rodas omnidirecionais, porém surge ainda outro tipo de roda que se pode inserir nesta categoria. As rodas esféricas, apesar de não serem propriamente uma roda com um eixo fixo, são responsáveis por movimentar o robot em todas as direções, estando por isso categorizadas como rodas omnidirecionais.

- Rodas Esféricas - Estas rodas perfeitamente esféricas podem movimentar-se em todas as direções, podendo ser passivas ou ativas. Enquanto roda passiva a sua função é conferir estabilidade ao robot.(fig.2.9a) Enquanto ativa, estas rodas são a junção, de pelo menos três rodas unidirecionais, ou três rodas omnidirecionais, com uma esfera (fig. 2.9b). O seu funcionamento é semelhante ao funcionamento das bolas de rato de computador, todavia as rodas usadas não servem para ler o movimento realizado pela bola, mas sim conceder-lhe o movimento.
  Este tipo de rodas, enquanto roda ativa, é uma solução pouco explorada, apesar de permitir que com rodas unidireccionais seja possível construir uma roda que possa deslocar-se em todos os sentidos. Esta solução necessita de uma complexidade de funções para controlar o seu movimento, pois tem de acionar pelo menos três rodas para desempenhar o correto movimento. Por outro lado, é uma solução que tem pouca tração em comparação com os restantes tipos de rodas, visto que é perdida energia por escorregamento na ação das rodas para a esfera e novamente dissipada energia entre a esfera e o chão.

(a) Roda Esférica Passiva



(b) Roda Esférica Ativa - *Ballbot lego*

Figura 2.9: Rodas Esféricas

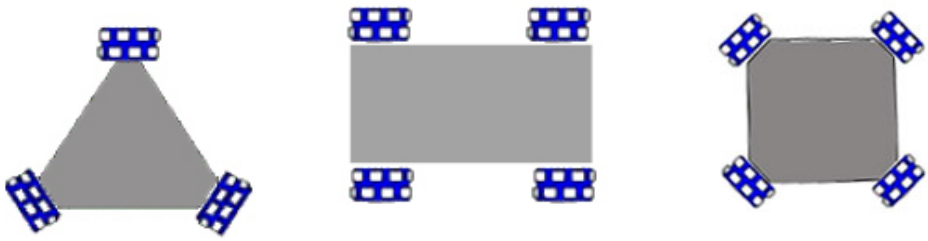
Disposição das Rodas

Na construção de robots terrestres, não é considerado apenas o meio de locomoção, mas também a forma como este se dispõe, neste caso as rodas.

A disposição das rodas é selecionada tendo em conta as rodas usadas, a forma como o robot irá mudar de direção, o meio onde o robot será inserido e a estabilidade pretendida. Assim, podemos considerar diversas opções que são normalmente usadas.

Robots Omnidirecionais Os robots omnidirecionais são todos os robots que se podem deslocar em qualquer direção sem necessitarem que as suas rodas sejam reorientadas. Este tipo de robots, para o seu correto funcionamento, exigem um controlo complexo de modo a sincronizar todas as rodas.

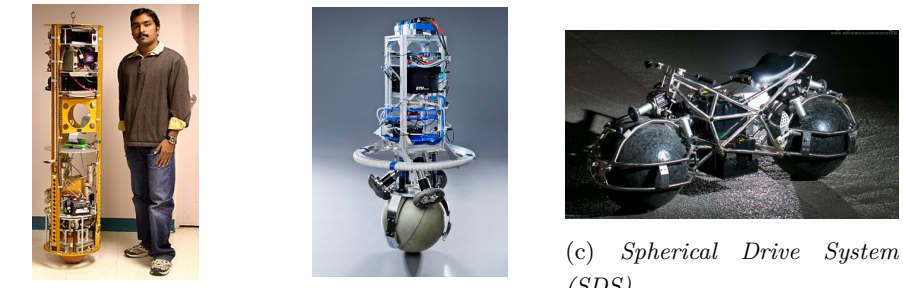
- Robot com rodas omnidirecionais tradicionais - Os robots compostos por rodas omnidirecionais tradicionais, rodas omnidirecionais convencionais ou Mecanum Wheel, têm o problema da direção resolvido, visto que este tipo de rodas pode adotar qualquer direção. Assim, apenas é tido em conta a disposição das rodas. Neste tipo de robot, normalmente, utilizam-se três ou quatro rodas dispostas como apresenta a figura 2.10 .
  Na disposição de três rodas (fig. 2.10a) a solução é mais barata e o controlo das rodas é menos complexo, porém esta solução apresenta menor estabilidade e menos tração que as restantes opções que conjugam quatro rodas. As duas opções a quatro rodas (fig. 2.10b e 2.10c) são as mais usadas e tem um desempenho semelhante, apenas diferem na forma com é realizado o controlo das rodas, pois a sua orientação na estrutura é diferente .



(a) três rodas omnidirecionais (b) quatro rodas omnidirecionais (c) quatro rodas omnidirecionais

Figura 2.10: Disposição das rodas Omnidirecionais

- Rodas Esféricas - Apesar deste tipo de rodas não ser muito explorado devido às razões mencionadas anteriormente, existem soluções de disposição das rodas que desempenham corretamente as funções exigidas. Estas rodas podem configurar-se no controlo, usando apenas uma roda, denominado de ballbot. O seu funcionamento, neste caso, baseia-se na ideia de pêndulo invertido, sendo que o primeiro modelo funcional surge em 2005, desenvolvido na Carnegie Mellon University pelo prof. Professor Ralph Hollis[27]. Todavia, pode ser conjugado o números de rodas que se achar pertinente, tendo em atenção que quantas mais bolas adicionadas à estrutura, mais estável será e menos complexo será o controlo do robot.



(a) Primeiro Ballbot (b) *Rezeco*[28] - Ballbot (c) *Spherical Drive System (SDS)*

Figura 2.11: Disposição das rodas esféricas

Robots com rodas unidirecionais Estes robots são formados por rodas unidirecionais em que estas precisam de ser orientadas para que o robot possa mudar de direção. Assim, houve a criação de diversas formas de executar o processo de orientação e as rodas podem ser dispostas de diferentes formas:

- Robot diferencial - Estes robots são por norma os mais utilizados, visto serem os mais simples em termos de funcionamento e controlo.
  Composto por duas rodas unidirecionais acionadas por motores independentes, e uma ou duas rodas passivas para conferir estabilidade ao robot, na maioria dos casos essas rodas passivas são unidirecionais ou esféricas. É estritamente necessário que estes tenham as rodas motrizes independentes, porque a sua direção é dada recorrendo à diferença de velocidades entre as rodas motrizes. Em curvatura, as rodas exteriores percorrem uma maior distância que as interiores, portanto as rodas exteriores devem ter uma velocidade maior, sendo que a roda interior é a que está mais perto do centro da curva.
  Assim, se as rodas tiverem iguais velocidades, o robot desloca-se em linha reta. Caso contrário, se houver diferença entre as velocidades das rodas, o robot desloca-se para o lado que terá menor velocidade.

(a) Robot diferencial três rodas (b) Robot diferencial quatro rodas

Figura 2.12: Orientação das rodas em Robot Diferenciais

- Robot Triciclo - Estes dispositivos apresentam uma estrutura muito semelhante aos robots diferenciais com três rodas, figura 2.12a. Contudo, as rodas que desempenhavam tração nos robots diferencias são, neste caso, passivas, tendo muitas das vezes o eixo acoplado. Nos robots triciclos apenas existe uma roda de tração que simultaneamente é a responsável por conferir a direção ao veículo, estas duas tarefas ficam a cargo da roda que se encontra isolada.
  Nestes robots já não há a necessidade de variar a velocidade para curvar, podendo obter velocidades constantes iguais em curvatura e em linha reta. Contudo, estes robots não podem apresentar velocidades muito elevadas, porque o seu centro de massa está muito próximo da zona limite de estabilidade definida pelas três rodas. Nas curvas, se a velocidade for muito elevada, pode-se dar o caso do centro de massa do robot se deslocar para fora da superfície de equilíbrio, e dar-se um capotamento.

- Robot Quadriciclo - Esta disposição baseia-se na ideia de triciclo, sendo formada também por rodas unidirecionais. Todavia, esta opção soluciona o problema da estabilidade que existia no caso anterior. Este sistema recorre ao uso de mais uma roda para melhorar o

desempenho e o equilíbrio dos robots. Ao adicionar esta roda o robot será mais estável, no entanto acresce o problema de controlar a direção das rodas.
O controlo do equipamento passa por usar a geometria Ackerman. Este mecanismo apoia-se na disposição das quatro rodas par a par, paralelamente, duas à frente e duas atrás. Esta geometria é muito familiar, podendo ser encontrada sobretudo nos carros convencionais. Normalmente, apenas duas rodas são de direção, sendo na maioria dos casos as rodas frontais. Estas rodas apresentam os eixos de direção ligados, de forma a que nas curvas a roda interior descreva um ângulo maior que a roda exterior. Isto origina duas circunferências com diferentes raios, geradas a partir do movimento de cada uma das rodas, acabando ambas por terem o mesmo centro. (Fig.2.13)

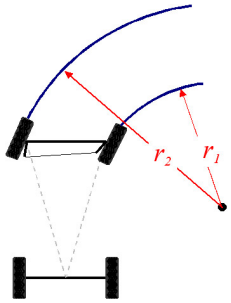


Figura 2.13: Geometria de Ackerman em curvatura

Com este método, torna-se mais fácil o controlo da direção do veículo, não necessitando de controlar cada roda independentemente.

Sensores

Os robots são dispositivos de execução de tarefas autónomos ou semi-autónomos e para tal há necessidade de uma leitura do meio. Os sensores são o processo de interação entre o meio físico e o digital. Existem vários tipos de sensores para se adaptarem aos diferentes processos e ambientes. A utilização dos sensores é indispensável para o correto funcionamento de um robot, podendo precaver algumas situações indesejáveis.

A variedade de sensores utilizados, em robótica, para ler os mais diversos tipos de parâmetros são infindáveis. O robot realiza a leitura com a frequência necessária de modo a ficar com a perceção do meio, ou de si mesmo. Assim sendo, teremos, à partida duas grandes categorias de sensores: internos e externos [29].

- **Sensores Internos** dizem respeito a parâmetros internos do robot, como por exemplo a indicação da bateria;

- **Sensores Externos** têm como objetivos dar informação do meio envolvente, por exemplo sensores de proximidade.

Os sensores podem ser ainda definidos pela emissão ou não de energia sendo classificados por: ativos ou passivos[29].

- **Sensores Passivos** projetados para receber e medir a energia existente no ambiente, como por exemplo medição de temperaturas ou sensores fotossensíveis;
- **Sensores Ativos** emitem energia e medem o impacto com o meio por forma a entender o que os rodeia, por exemplo Radares.

Após dar conhecimento das principais categorias existentes dos sensores, será enunciado diversos sensores os mais comuns em robótica.

Encoders

Estes sensores internos são utilizados nas juntas ou nas rodas dos robots, permitindo medir a velocidade angular ou a posição angular de um eixo. Os encoders são compostos por um feixe de luz e seu recetor, entre eles está um disco perfurado acoplado ao eixo. Consoante o formato do disco poderemos ter encoders absolutos ou relativos [29].

Encoders Absolutos os discos são perfurados de maneira a que cada posição da junta represente determinada palavra binária, assim com este tipo podemos saber exatamente a posição angular da junta.

Encoders Relativos apresentam ranhuras no seu disco de dentro para fora, todas idênticas, de modo a que a uma velocidade constante surja um sinal periódico com frequência constante. Quanto maior a velocidade maior a frequência do sinal.

(a) Disco de Encoder Absoluto

(b) Disco de Encoder Relativo

Figura 2.14: Discos de Encoder

Sensor Distância

Os sensores de distância são os mais usados em robótica. Estes permitem medir a distância entre o robot e os objetos de forma fácil, precisa e barata. Este tipo de sensores são usados

sobretudo para mapeamento do ambiente. Eles emitem uma forma de energia que será refletida e enviada de volta ao sensor, estando bastantes suscetíveis a interferências e dependendo bastante das superfícies dos objetos. A emissão de energia pode ocorrer sobre duas formas.

- **Ultra-Som** baseia-se na emissão de ondas de pressão, este processo é um pouco mais disperso que o laser;
- **Laser** baseia-se na emissão de ondas eletromagnéticas, estes sistemas são mais precisos que os ultrassónicos.

Podemos classificar estes sensores em três grupos[17]:

Tempo de Vôo Este tipo de sensores, designados em inglês de *Time-of-Flight* (*TOF*), baseiam-se na reflexão. É emitido um pulso de onda, esta por sua vez bate no objeto e é refletido de novo ao sensor. A distância é dada pelo cálculo entre a velocidade de propagação da onda e o tempo de vôo.

Medição Mudança de Fase À semelhança dos sensores de distância por tempo de vôo, estes sensores baseiam-se, também, na reflexão da onda. Todavia, estes sensores enviam um espectro continuo, em vez de pulsado. O sinal refletido e recebido é comparado com a referência, gerada com o sinal enviado, e verifica-se a desfasagem do sinal. O cálculo com o comprimento de onda e a desfasagem dão a distância a que o objeto se encontra.

Radar de Frequência Modulada Neste tipo de sensores é enviado um sinal triangular contínuo com frequência variável em torno de um valor médio. À semelhança dos sensores de distância anteriores, o sinal enviado é refletido pelo objeto retornando ao recetor do sensor. O sinal é comparado com o sinal de referência para determinar há quanto tempo a frequência recebida foi enviada. Sabendo que a energia emitida viaja à velocidade da luz podemos saber com precisão a distância a que se encontra o objeto.

Global Positioning System (GPS)

O sistema de GPS foi desenvolvido pelo Departamento de Defesa Americano, inicialmente servindo apenas para fins militares. Porém, em 1995, este sistema ficou disponível para meios civis.
Este sistema de localização, utiliza 24 satélites artificiais que enviam continuamente sinais de rádio para a terra. Ao instalar o recetor de GPS, este processa os sinais de cada um dos satélites no seu alcance calculando a sua coordenada terrestre através do método de triangulação. Sendo necessário para a correta leitura de posição 4 pontos fundamentais[17]:

- Sincronização entre o recetor de GPS e os satélites;
- Localização exata de cada satélite;
- Uma medição precisa do sinal propagado;
- Proporção sinal ruído suficiente para o funcionamento fiável.

Este sistema bastante preciso de várias utilizações foi revolucionário, sendo utilizado em robótica. Todavia, o seu uso fica restrito a meios exteriores, pois no interior de edifícios é impossível ao recetor receber o sinal de GPS.

Sensores de Movimento e Velocidade

Este tipo de sensores utilizam o principio do efeito de Doppler, que consiste numa variação da frequência de um sinal quando este embate num objeto em movimento. Portanto, este tipo de sensores apenas detetam movimentos e velocidades. São utilizados sobretudo para a eliminação de eventuais erros.

Sensores Orientação

Os sensores de orientação são sensores importantes, visto que estes são responsáveis por saber a orientação do Robot, importantes para a determinação de objetos e mapeamento do ambiente. Podemos detetar a orientação do robot com dois tipos de sensores distintos:

- **Giroscópios** - são sensores que se baseiam nas leis da física para determinar a orientação do robot. Deste modo, podemos ter giroscópios mecânicos ou óticos.
  - Giroscópios mecânicos - baseia-se na conservação de energia. Verifica se existe desequilibro mecânico num giroscópio medindo a velocidade angular de cada junta.
  - Giroscópios ótico - usa a velocidade da luz para ler a orientação do robot. Este sensor são diferenciados em dois tipos: fibra ótica e anel de laser. Ambos emitem dois feixes de luz, que devido ao efeito de Sagnac, o feixe de luz que viaja contra a rotação sofre um atraso de percurso. Ao medir a desfasagem do sinal é possível saber a velocidade angular a que o robot foi submetido. Enquanto que os giroscópios de anel ótico usam espelhos para direcionar o feixe, no outro caso a fibra ótica faz esse processo.

- **Bússolas** - As bússolas têm como função saber a orientação do robot lendo o campo magnético terrestre. Este processo é realizado por intermédio de bobines e um núcleo bastante permeável. As bobines medem o campo magnético da terra que atravessa o núcleo.

Sensores de Visão

Os sensores de visão são utilizados quando é necessário ter uma maior noção do ambiente que rodeia o robot. Com este tipo de sensor é ainda possível saber a distância exata a que os objetos se encontram. Utilizando duas câmaras conseguimos ter uma perspetiva 3D do ambiente, conseguindo calcular a distância dos objetos visíveis na imagem. Este tipo de sensores são formados

por uma matriz de semicondutores fotossensíveis, processados essencialmente de duas formas[12]:

- **Charge-Coupled Device (CCD)** - foi um dos primeiros sensores de visão, este sensores tem por base o efeito de carga e descarga de um condensador realizado por intermédio do semicondutor fotossensível (pixel). A descodificação é feita lendo a carga elétrica de cada condensador, esta função está a cabo de um conjunto de circuitos que convertem a informação para poder ser lida pelos restantes dispositivos.
- **Complementary metal–oxide–semiconductor (CMOS)** - este tipo de sistema tem acoplado a cada pixel um conjunto de transístor que processa imediatamente a informação recebida. Assim, torna-se mais simples a descodificação da informação, o que implica a existência de menos eletrónica. O facto de não haver condensadores ligados ao semicondutor fotossensível, compreende-se que os ciclos de descarga sejam inexistentes, sendo bastante mais rápido e simples que o sensor CCD.

2.2 Planeamento de Rotas

Desde a invenção dos robots móveis houve sempre a preocupação de evitar as colisões entre os robots e os objetos do ambiente. Por isso, houve a necessidade de criar algoritmos de modo a que fosse possível que os robots conseguissem planear a sua rota tendo em conta os objetos que estão no seu caminho.

Enquadramento

Considerando que o robot se desloca num determinado espaço $W \in R^2$ onde:

- A é o Robot;
- $B_1, B_2, ..., B_\infty$ são os objetos estacionários.

Sendo o objetivo do robot navegar do ponto "S"(start) ao ponto "G"(goal) evitando qualquer obstáculo B_n.

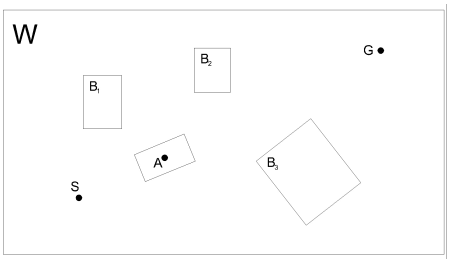


Figura 2.15: Ambiente robot móvel

Assumindo os dados anteriores„ existem vários processos de controlar o robot para resolver o problema. O processo é escolhido dependendo da sua eficácia e o conhecimento do meio.

2.2.1 Algoritmo Bug

Este tipo de algoritmo consiste na navegação do robot no espaço, contornando os objetos que sejam colocados na sua rota, este método foi baseado no comportamento dos insetos.

Neste tipo de código, o robot está programado para mover-se sempre sobre uma linha reta imaginária entre si e o ponto alvo. Porém na sua deslocação podem surgir objetos no seu caminho. Nestes casos, o robot toma uma decisão de contornar o objeto, pela esquerda ou pela direita. Findo o contorno dos objetos, o robot volta a traçar a reta até ao ponto objetivo.

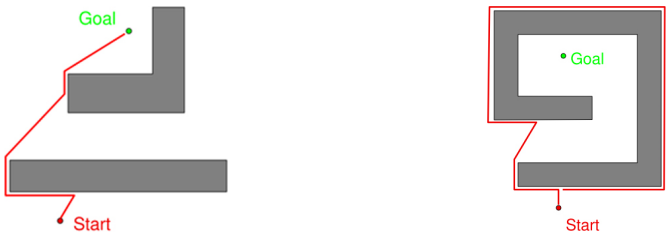
O código deste algoritmo pode ser variado, devido a haver vários processos de ultrapassar os objetos, mas resume-se em três fases:

- Objetivo ponto final;
- Contornar o objeto;
- Repetição dos dois últimos pontos até chegar ao ponto de chegada;

Dependendo da forma como é feita a escolha de contornar os objetos, este tipo de algoritmo pode ser classificado em vários tipos[16].

BUG 0

A primeira opção para o código é contornar o objeto até conseguir seguir de novo o ponto final. Porém este código suscita um problema, atendendo ao facto de este algoritmo virar o robot sempre para o mesmo lado, pode-se dar o caso de fechar a trajetória sem que o robot chegue ao ponto final, mesmo sendo possível executar um percurso.(Fig.2.16b)



(a) Bug0 completo, virar a esquerda (b) Bug0 incompleto

Figura 2.16: Bug 0

BUG 1

Atendendo ao problema existente no código anterior, surge o "Bug 1". Este é semelhante ao anterior, todavia quando o robot encontra um objeto na sua frente, contorna-o na totalidade. Após saber o contorno completo do objeto, volta ao ponto em que esteve mais perto do seu objetivo. (Fig. 2.17) Este algoritmo soluciona o problema anterior, contudo este processo acresce mais tempo e distância decorrido.

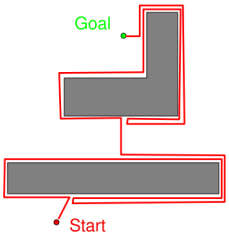


Figura 2.17: Bug 1

BUG 2

Neste processo é considerada uma recta do ponto "start" até ao ponto de chegada "goal". O objeto é ultrapassado da mesma maneira que a anterior, circundando o obstáculo. Mas só é contornado até encontrar de novo a reta que definiu de início antes de colidir com o objeto.(Fig.2.18)Este processo resolve os problemas existentes nos códigos anteriores, melhorando ambos os algoritmos.

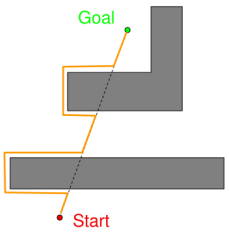


Figura 2.18: Bug 2

Tangent Bug

Este algoritmo é semelhante aos outros algoritmos *Bug*, no entanto este é mais eficiente que os anteriores, pois conseguimos detetar os obstáculos com maior antecedência. Durante o percurso, o robot está constantemente a receber as distâncias obtidas por parte do sensor de alcance, este sensor de alcance tem como função ler a posição dos objetos de forma radial. (Fig. 2.19a)[33][35]. Detetada uma colisão, o algoritmo tenta minimizar a distância que o robot irá precorrer, recorrendo ao *Tangent Bug Algorithm (TBA)*, este algoritmo faz com que o robot se dirija para uma

das extremidades do objeto, a mais próxima, dada por $\rho(x, \theta)$.

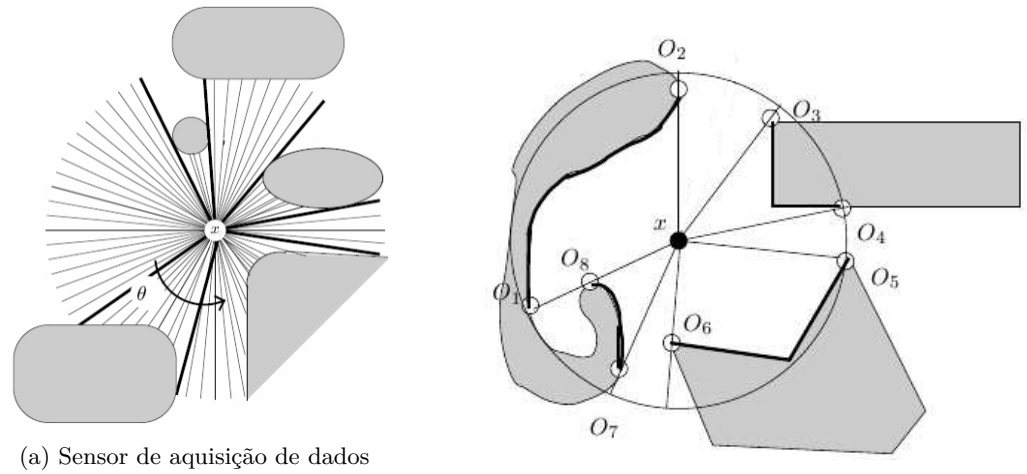
$$\rho(x, \theta) = \min_{\lambda \in [0, \infty[} d(x, x + \lambda[\cos(\theta), \sin(\theta)]^T) \tag{2.1}$$

sujeito a:

$$x + \lambda[\cos(\theta), \sin(\theta)] \in \bigcup_i O_i \tag{2.2}$$

x : é a posição atual do robot;
θ : ângulo entre o robot e o novo ponto
d : distância estimada desde o ponto atual do robot até ao ponto de destino obtido pelo *tagente bug*;
O_i : pontos descontínuos obtidos por *TBA*.

Como neste caso se consegue antecipar as extremidades dos objetos, é o processo mais eficaz de ultrapassar os obstáculos.



(a) Sensor de aquisição de dados

(b) Seleção de pontos O_i

Figura 2.19: Tangent bug algorithm

2.2.2 Planeamento com Base em Amostragem

Este é um conjunto de algoritmos que necessita de conhecer a totalidade do mapa. Estes algoritmos têm em conta todos os obstáculos existentes no meio e calculam a rota, de forma a ser o mais eficiente possível para fazer o robot chegar ao seu destino.

Na geometria computacional, não será considerado o mapa o obtido na fig. 2.15. Antes de qualquer cálculo de rota, é necessário executar um código de forma a dilatar os objetos, obtendo um novo mapa proveniente do anterior com menos espaço livre. O novo mapa é obtido deslocando o robot(A) sobre as extremidades dos obstáculos (B), criando obstáculos com maior volume. A forma como se originam as extremidades do novo objeto no mapa pode ser obtido através de inúmeros processos, baseando-se sempre na deslocação do robot sobre as extremidades do objeto. Na fig. 2.20 é considerado um dos vértices do robot para originar as novas extremidades do objeto[19].

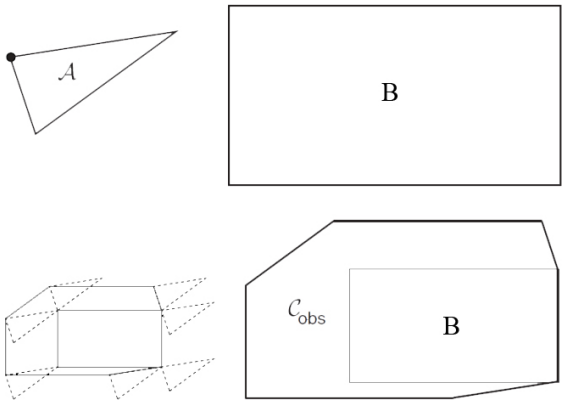


Figura 2.20: Obstáculo C_{obs}

Efetuado a estrutura do novo mapa poderemos aplicar qualquer um dos processos descritos a seguir, dos quais fazem parte: Visibility Graph; Voronoi Diagram; Decomposição celular; Probabilistic Road Maps (PRM); Rapidly-exploring Random Trees (RRT).

Visibility Graph

O cálculo deste algoritmo tem em conta a localização de todos os obstáculos existentes no mapa, é realizada uma linearização entre os vértices dos obstáculos,o robot e o ponto objetivo, obtendo todos os caminhos possíveis de realizar.
Inicialmente, é criados vários caminhos a partir do robot até todos os vértices visíveis deste ponto. O mesmo acontece no ponto de chegada[25].

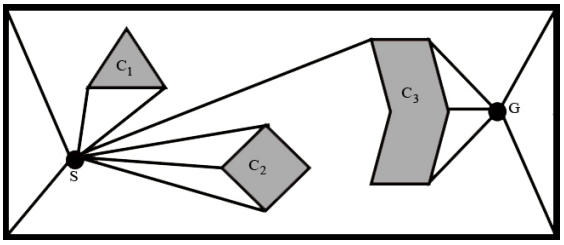
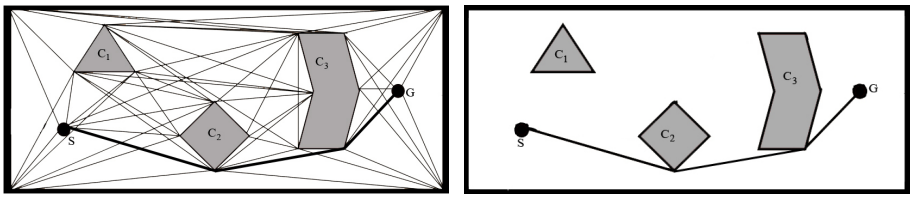


Figura 2.21: Primeiro passo visibility graph

De seguida é executado o mesmo processo, mas agora para todos os vértices dos objetos, tendo em conta quais os vértices visíveis. Obtendo todas as combinações possíveis:



(a) Toda as combinações de caminhos (b) Caminho solução do problema

Figura 2.22: Exemplo de visibility graph

Por fim, faz-se a escolha dos troços que o robot irá seguir. Pode-se ainda otimizar o algoritmo fazendo com que o robot escolha o caminho mais curto(fig.2.22b). Este tipo de algoritmo tem um inconveniente, pois ao considerar os vértices dos obstáculos, o robot passará sempre próximo dos mesmos, o que acresce perigo de colisão.

Diagrama de Voronoi

Diagrama de Voronoi é um método da decomposição do espaço. Este algoritmo baseia-se na decomposição do espaço vazio segundo caminhos que estejam equidistantes das arestas dos objetos. Este processo pode ser visualizado utilizando circunferências traçadas de maneira a que sejam tangentes às arestas dos objetos, o seu centro será um ponto do caminho[25](fig. 2.23).

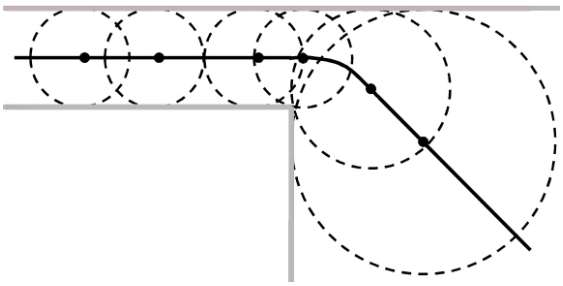


Figura 2.23: Caminhos diagrama Voronoi

No código, por vezes, não é possível ter uma grande definição dos objetos, e matematicamente é difícil traçar circunferências. Assim, no limiar, os objetos podem ser representados por um conjunto de pontos nas suas arestas. Para obtenção do diagrama de Voronoi teremos de traçar bissetrizes entre os pontos detetados. Existirá um ponto comum entre pelo menos 3 retas, esse será um vértice de Voronoi que estará equidistante de todos os objetos. Este ponto vai ser fulcral para eliminação de parte das retas que não fazem parte do diagrama.

Sendo o vértice equidistante dos objetos é possível traçar uma circunferência onde os pontos estão contidos, a circunferência será formada por arcos numerados conforme os pontos. Os arcos, por sua vezes, são intercetados pelas retas, contudo estas terão de passar a semirretas. O processo é realizado eliminando as semirretas que têm origem no vértice e passam no arco correspondente[32], figura 2.24.

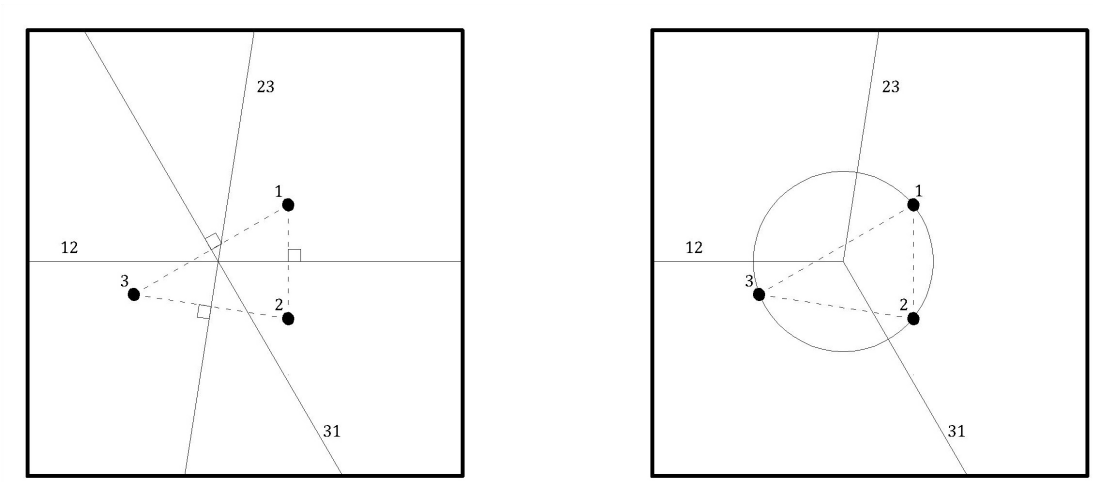
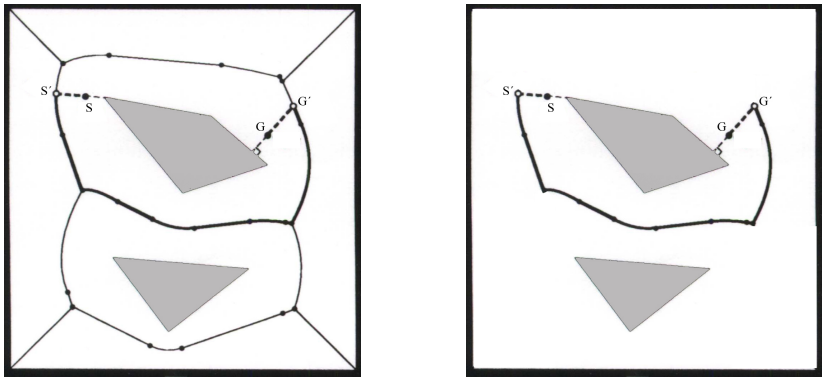


Figura 2.24: Criação de caminhos diagrama Voronoi [6]

Ao aplicar este teorema a todo o mapa, pode-se usar o algoritmo de Fortune [22][32] para obter um mapa semelhante ao apresentado na figura 2.25a.



(a) Mapa diagrama Voronoi (b) Solução do diagrama Voronoi

Figura 2.25: Diagrama de Voronoi

Com base no mapa de Voronoi obtido, é só necessário definir um caminho e indicar ao robot os segmentos do diagrama que deve seguir, figura 2.25b.

Decomposição celular

Este algoritmo consiste na divisão do espaço livre em células, que irão definir o caminho. Pode-se realizar este processo através de dois métodos diferentes[25]:

- **Decomposição Exata** - Esta primeira opção de decomposição é realizada de modo a que as células não tenham mais de 4 faces. Este algoritmo pode ser dividido em duas fases distintas. Uma primeira, de divisão do mapa seguida de uma segunda em que se escolhe as divisões a tomar.
  Inicialmente, no mapa são criados segmentos na vertical cada vez que exista uma extremidade ou vértice no mapa. Este processo de divisão pode ser realizado com o algoritmo *Boustrophedon decomposition*, usualmente utilizado[25][11], através deste método o espaço livre ficará dividido em células irregulares tendo sempre 3 ou 4 faces.

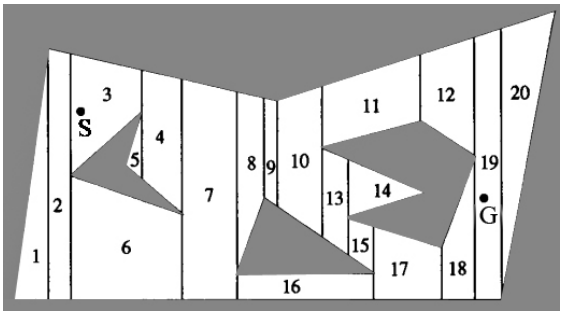
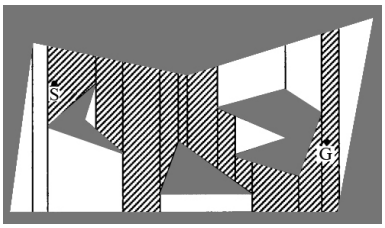
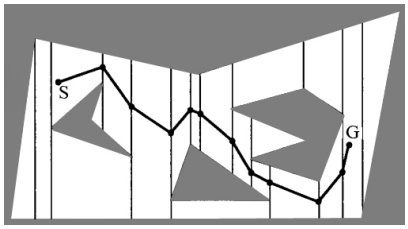


Figura 2.26: Decomposição exata [25]

Os segmentos verticais, criados, que dividem as diversas áreas são responsáveis por definir os pontos que vão pertencer ao caminho. Através da numeração das áreas seleciona-se o conjunto de pontos a tomar, estes são colocados no meio de cada segmento selecionado. (Fig.2.28b)



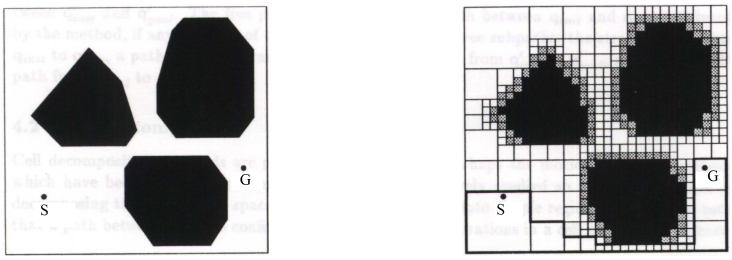
(a) Seleção de áreas



(b) Caminho com decomposição exata

Figura 2.27: Decomposição exata por células

- **Decomposição por aproximação** - Neste processo de decomposição do espaço o procedimento resume-se na divisão recursiva do espaço em retângulos. Também neste método podem observar-se as duas fases mencionadas no algoritmo anterior, a primeira de divisão e uma segunda de decisão/seleção.
  O algoritmo começa pela divisão do ambiente em células, é utilizado o *Barnes–Hut simulation*, que divide o mapa em células regulares de 4 faces até determinada resolução, caso haja existência de objetos dentro da divisão. Para ser mais fácil a organização das células é gerada uma estrutura de dados em árvore. Seguidamente existe uma análise aos diferentes retângulos, verificando se fazem ou não parte do espaço livre. Por fim, é selecionada a sequência de retângulos a seguir pelo robot[25].



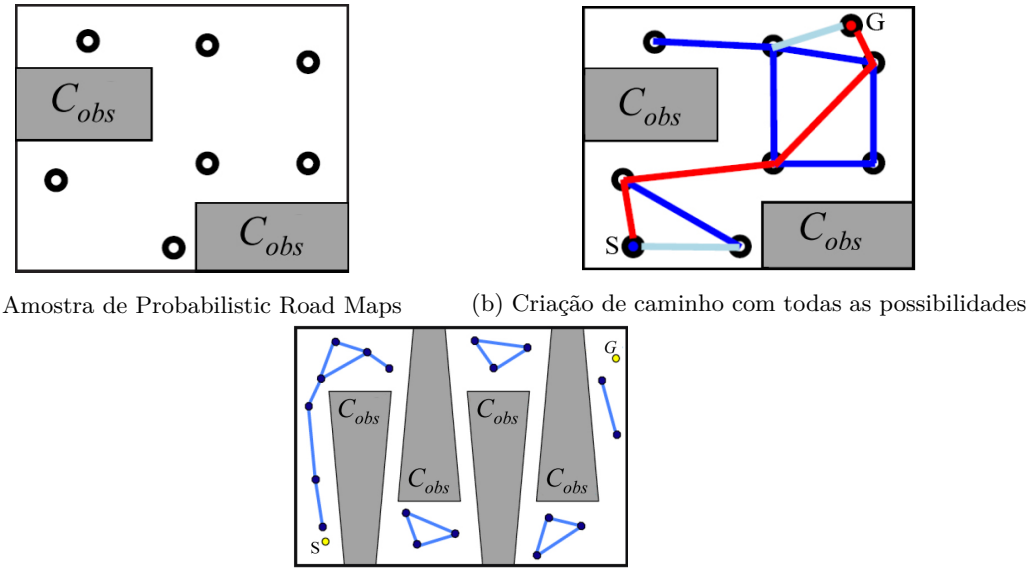
(a) Divisão do Mapa (b) Seleção de areas para criar o caminho

Figura 2.28: Decomposição aproximada por células

Probabilistic Road Maps (PRM)

O algoritmo em questão, tem como função escolher um determinado número de amostras, pontos, pertencentes ao espaço livre de forma aleatória. *A posteriori* é realizada a ligação entre as amostras próximas umas das outras, de modo a que não haja colisões entre os segmentos criados e os objetos, criando assim uma rede de caminhos. Após a criação da rede de caminhos, escolhe-se aquele que tem melhor desempenho.

Apesar deste código ser bastante eficiente, se houver passagens estreitas no mapa, por vezes torna-se impossível realizar a deslocação do robot. Uma vez que do ponto de vista do algoritmo, é impossível unir os pontos da amostra sem colisões, como demonstrado na figura 2.29c[34].



(a) Amostra de Probabilistic Road Maps (b) Criação de caminho com todas as possibilidades

(c) Problema do Algoritmo

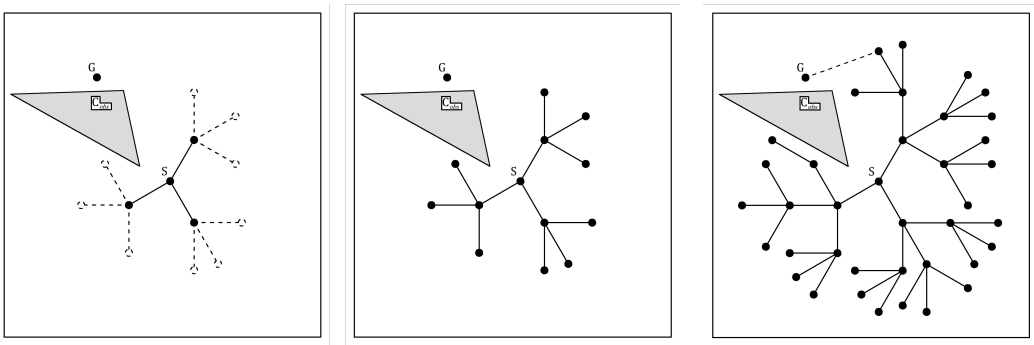
Figura 2.29: Probabilistic Road Maps

Rapidly-exploring Random Trees (RRT)

Este tipo de algoritmo também tem em consideração pontos escolhidos aleatoriamente. Contudo, neste código, os segmentos são criados a partir do ponto de partida até à chegada.

Inicialmente, tem-se em conta o ponto de partida e recorre-se a uma função que dará um determinado número de amostras dispostas em torno desse ponto. São criados segmentos do ponto inicial para os novos pontos criados, verificando sempre a inexistência de colisões. O processo realizado no ponto de partida é repetido nos novos pontos criados, sendo agora o ponto de referencia esses mesmos pontos e assim sucessivamente até atingirmos o ponto objetivo.

A forma como a rede de caminhos é criada origina uma forma semelhante a uma árvore, a obtenção dos caminhos desta forma corrige o problema que existia no *PRM*.



(a) Primeiros pontos de RRT (b) Novos Ramos do RRT (c) Fase final de RRT

Figura 2.30: Criação de pontos com RRT

2.2.3 Localização e Mapeamento Simultâneos (SLAM)

Nesta solução, o robot não disponibiliza de qualquer informação externa. Deslocar-se-á no mapa, de maneira a realizar o seu próprio mapa ao mesmo tempo que se localiza, de forma a conseguir completar o seu objetivo.

Neste tipo de ambiente, o robot conta apenas com as leituras dos seus sensores. Todavia estes têm sempre um erro, acrescendo a este facto está a dificuldade de contabilização de escorregamentos e atritos em juntas e rodas, o que inviabiliza a criação correta de um mapa.

Enquadramento

Usaremos nesta secção a nomenclatura igual à *IEEE* [7] de modo a que:

- $\mathbf{x}_k$ é a posição do robot no instante k;
- $\mathbf{u}_k$ é o vetor responsável pela translação do robot da posição $\mathbf{x}_{k-1}$ para $\mathbf{x}_k$;
- $\mathbf{m}_i$ é a localização do objeto i;
- $\mathbf{z}_{ik}$ é a distância do robot para o objeto i no instante k.

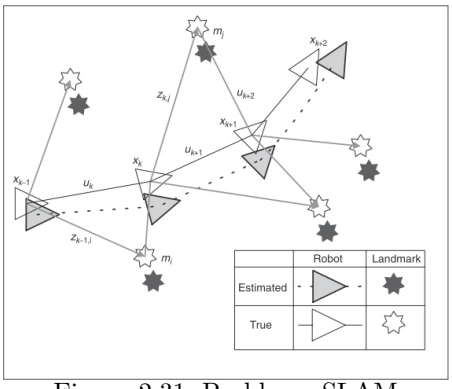


Figura 2.31: Problema SLAM

Este problema é construido sobre uma distribuição probabilística, usando as medições efetuadas e dispondo-as numa função distribuição, o robot consegue fazer uma estimativa da sua posição relativamente ao mapa. Com as medições efetuadas, o robot executa filtros EFK(extended Kalman filter)[7], de modo a saber a sua localização e dos objetos encontrados, colocando-os no mapa de forma mais correta que sem este método.

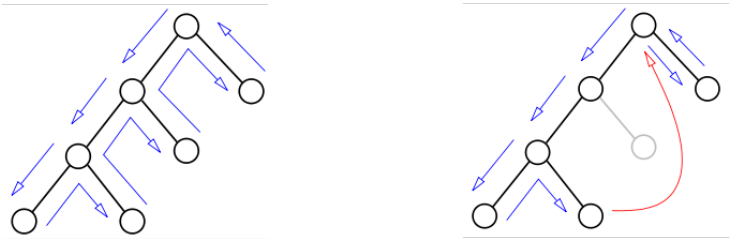
2.3 Retroação

A retroação, ou *backtracking*, veio solucionar inúmeros problemas do mundo real que não tinham solução.

Esta solução passa pela busca sucessiva de pontos de forma a tomar uma escolha de pontos que acaba por ser a solução ao problema. O *backtracking* será um conjunto de funções que serão recursivamente chamadas de forma a chegarmos à solução. Este baseia-se nos dois casos que podem acontecer: por um lado a decisão é acertada e portanto o ponto faz parte do caminho; por outro lado pode-se dar o caso da decisão ser um caminho fechado ou uma escolha incorreta, assim teremos de voltar ao ponto de decisão anterior ou tomar outro caminho[31].

A melhor forma de pensar em backtracking é sob a forma de árvore, este algoritmo parte de um ponto inicial, a raiz, que a partir deste é formado vários ramos. Caso os ramos não sejam a solução podem ser criados outros ramos e assim sucessivamente até chegar à solução.

O backtracking pode ser configurado de duas maneiras, uma onde os pontos são sempre todos introduzidos na árvore e a outra somente alguns pontos são inseridos - processo com backjumping.



(a) Backtracking sem backjumping (b) Backtracking com backjumping

Figura 2.32: Estrutura Árvore - Backtracking

3

Planeamento

Neste capitulo é explicado o planeamento bem como o desenvolvimento dos métodos usados para a realização deste projeto, apresentando as razões e as motivações das escolhas realizadas, bem como a explicação teórica de certos conceitos utilizados.

Esta dissertação tem como objetivo controlar um robot móvel de forma a que este consiga deslocar-se autonomamente num ambiente desconhecido. Deste modo, para a realização deste projeto, optou-se por usar um simulador, que sem qualquer custo associado permite a realização de testes aos diversos elementos.

Apesar do projeto ser puramente de simulação, contemplou-se sempre a sua implementação. Assim, foi escolhida uma plataforma que utilizasse ou tivesse em consideração elementos físicos reais. Depois de uma busca pelos diversos simuladores robóticos, pôde-se encontrar diversas plataformas de simulação, destacando-se: o MORSE [1], S.T.D.R. Simulator [4], OpenSim [2], Stage [3] e o V-REP [5].

Após uma análise detalhada aos vários programas, avaliando todos os prós e contras, foi escolhida a plataforma V-REP. Esta ferramenta, bastante poderosa, desenvolvida pela *Coppelia Robotics*, publicada pela primeira vez em 2010, oferece um ambiente a três dimensões, regido por leis fundamentais da física, aproximando-se bastante da realidade. A sua versatilidade de comunicação com outros programas e a existência de dispositivos reais pré-construidos na plataforma para simulação, foram as principais motivações para o uso deste programa. Contudo o seu elevado desempenho em termos gráficos pode acrescer problemas de processamento, exigindo mais capacidade por parte da máquina que opera este programa.

Escolhida a plataforma de simulação que pretende simular os acontecimentos físicos deste trabalho, constata-se que esta pode entrar em comunicação com outros programas. Este facto motiva a realização do código de controlo numa outra plataforma, não só para não sobrecarregar apenas um programa, mas, sobretudo, para ter um meio de representação do programa de controlo. Desenvolvido o código de controlo numa outra plataforma torna mais fácil descarregar

o algoritmo de controlo para um robot real.

Tendo em conta a gama de programas/linguagens que podem entrar em comunicação com o V-REP e as funções que podem ser utilizadas nas diferentes plataformas/linguagens, foi selecionado o MATLAB para resolver o processo de controlo do robot. Esta plataforma é uma ferramenta bastante utilizada em engenharia, bem como no ramo da robótica. Por trabalhar sobretudo com processamento de sinais, imagem e controlo de sistemas, é a ferramenta ideal para desenvolver todo o processo de controlo.

3.1 Robot

A plataforma de simulação apresentava inúmeras soluções existentes no mercado, porém optou-se por usar um robot do fabricante *KUKA*, o *YouBot*. O *YouBot* é um robot móvel com rodas omnidirecionais, que pode ser equipado com diferentes acessórios. Por esta razão tem forte potencial para desempenhar tarefas, sendo, por isso, um robot utilizado sobretudo para fins de exploração e investigação. A junção de todos estes fatores levaram à escolha deste robot.

O robot em causa é composto por uma plataforma móvel responsável pelo deslocamento do robot e por um braço robótico para desempenhar determinadas funções.

3.1.1 Plataforma YouBot

Conforme citado, este robot pode ser equipado por vários acessórios, porém o objeto de estudo nesta dissertação é a plataforma móvel *Youbot*, que será o foco principal do projeto, apresentando as especificações, de acordo com o manual de utilização do fabricante [21].

Figura 3.1: Plataforma *Youbot*

Tabela 3.1: Especificações da Plataforma móvel

Comprimento:	580 mm
Largura:	380mm
Altura:	140mm
Peso da plataforma:	20kg
Peso máximo transportado:	20kg
Velocidade máxima:	0.8 ms^{-1}

A plataforma dispõe apenas de rodas omnidirecionais, pelo que é necessário que todas sejam ativas. Existe, então, um motor acoplado a cada roda, bem como um encoder relativo e uma caixa de redução, com as características apresentadas em baixo:

Tabela 3.2: Especificações dos motores das rodas

Motores	
Tensão Nominal:	24 V
Corrente Nominal:	2.32 A
Binário Nominal:	82.7 mNm
Indutância:	0.573 mH
Resistência:	0.978 mΩ
Momento de Inércia:	13.5 $kgmm^2$
Velocidade Nominal:	5250 RPM
Peso:	110 g
Caixa de Redução	
Rácio de redução:	26
Momento de Inércia:	0.14 $kg * mm^2$
Peso:	224 g
Encoder	
Contagens por revolução:	4000

A plataforma dispõe ainda de um computador interno, cujas características são:

Tabela 3.3: Especificações do Computador

Computador	
Tipo:	Mini-ITX
CPU:	Intel Atom D510 Dual Core 1.66 GHz
RAM:	2 GB; SO-DDR2-667 200PIN
Disco Rigido:	32 GB SSD
Portas:	6 x USB 2.0, 1 x VGA, 2 x LAN (1 available)
Sistema Operativo:	Remastered Ubuntu Linux (preinstalled)
DC Input :	12 V
Comunicação:	EtherCat
Bateria:	chumbo-ácido recarregável 24V/5A

A bateria referida na Tabela 3.3, alimentará não só o computador, mas também todos os equipamentos do robot.

3.1.2 Braço Robótico

O *YouBot* traz, muitas das vezes, acoplado o braço robótico aumentando as potencialidades do dispositivo. Apesar de não estar a ser usado de forma a desempenhar alguma função, o robot está equipado com este utensílio, sendo necessário a sua descrição para futuras utilizações. O braço

robótico é composto por 5 juntas tendo um volume de trabalho do tipo esférico. A geometria das juntas e elos estão dispostas conforme a figura 3.2.

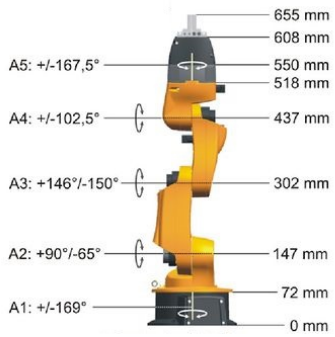


Figura 3.2: Geometria Braço Robótico [18]

Tabela 3.4: Especificações do Braço Robótico

Numero de Eixos:	5
Altura:	655 mm
Volume de Trabalho:	0.513 m^3
Peso:	5.3 kg
Carga Máxima:	0.5 kg
Estrutura:	Liga de Magnésio
Repetibilidade:	0.1 mm
Comunicação:	EtherCAT
Tensão nominal:	24V DC
Potencia máxima:	80 W
Velocidade máxima do eixo:	90 deg/s

O braço robótico dispõe de cinco graus de liberdade. Estes variam acionando os cinco motores elétricos, para dispor a extremidade na posição desejada. As característica dos motores estão em baixo descrita.

Tabela 3.5: Especificações dos Motores do Braço Robótico [18]

Atuadores	Joint 1	Joint 2	Joint 3	Joint 4	Joint 5
Motor					
Tensão Nominal (V)	24	24	24	24	24
Corrente Nominal (A)	2.32	2.32	2.32	1.07	0.49
Binário Nominal (mNm)	82.7	82.7	82.7	58.8	
Indutância eq. (mH)	0.573	0.573	0.573	2.24	7.73
Resistência eq. (Ω)	0.978	0.978	0.978	4.48	13.7
Momento de Inercia ($kg * mm^2$)	13.5	13.5	13.5	9.25	3.5
Velocidade Nominal (RPM)	5250	5250	5250	2850	2800
Peso (g)	110	110	110	75	46
Caixa Redutora					
Rácio de Redução	156	156	100	71	71
Momento de inércia ($kg * mm^2$)	0.409	0.409	0.071	0.07	0.07
Encoder					
Contagens por Revolução	4000	4000	4000	4000	4000

Na extremidade do braço podemos usar diversos dispositivos, contudo neste trabalho foi usado a garra de dois dedos que vem por defeito equipada no braço do *Youbot*.

3.2 Sensores

Este projeto para além do equipamento responsável pela mobilidade do robot, terá de contar com equipamentos que possam obter informação do meio - sensores externos.

Pretende-se desenvolver este trabalho juntando dois sensores para colaborarem em conjunto, de forma a conseguir obter o mapa onde o robot se encontra. Para tal, foi usado um sensor de distância e um sensor de posição/orientação.

3.2.1 Hokuyo URG-04LX-UG01

De modo a ser possível desempenhar corretamente a sua função, é necessário que o robot faça uma leitura do meio envolvente, para que consiga definir a localização dos objetos. Para este processo, optou-se por empregar um sensor de distância, escolhendo o equipamento conforme o aconselhado pelo fabricante do *YouBot*. O sensor aconselhado, o *Hokuyo* cujo o modelo é *URG-04LX-UG01*, é composto por um semicondutor díodo laser de infravermelhos com um diâmetro inferior a 20mm, que faz um varrimento de 240º. Tem como princípio de funcionamento a medição por mudança de fases (descrito em 2.1.2) e apresenta as seguintes especificações[23]:

Figura 3.3: *Hokuyo URG-04LX-UG01*

Tabela 3.6: Especificações do Hokuyo URG-04LX-UG01

Comprimento de onda do laser:	785 nm
Segurança Laser:	class 1
Distância de deteção:	20mm - 4000mm
Precisão:	±3% da medição
Resolução:	1 mm
Ângulo de varrimento:	240°
Resolução Angular:	0.36°
Frequência de varrimento:	100 $msec/scan$
Interface:	USB 2.0
Ambiente(temp./hum.):	-10 - 50°C/85 % ou menos
Luz existente no ambiente:	10000Lx ou menos
Consumo:	0.5W a 5V
Dimensões:	50 x 50 x 70 mm
Peso:	160 g

3.2.2 Sensor Posição/Orientação

Qualquer robot para funcionar corretamente tem de ter uma referência. Os robot móveis, não fogem à regra, exigindo, muitas vezes, estarem equipados com um sistema de geolocalização, para determinar a sua posição. Este sistema será implementado para colmatar o erro que possa existir através dos encoders dos motores e o escorregamento das rodas, aumentando a fiabilidade

do sistema.

Este trabalho está previsto para ser implementado no exterior de edifícios, portanto o robot pode ser equipado com um sistema de *Global Position System - GPS*. Caso se preveja o uso do robot no interior de edifícios terão de ser desenvolvidas novas formas de registar a posição do robot, ou recorrer apenas à posição dos encoders.

Este projeto tem em conta, também, um sensor de orientação, por forma que o robot consiga saber a sua orientação no mapa.

A seleção do sensor a implementar na simulação passa pela escolha de um dispositivo que junta o sensor de orientação e o sensor GPS, na mesma placa de circuito. O sensor do fabricante *CH Robotics* modelo *GP9 GPS-Aided AHRS* poderá ser uma opção viável para a leitura da posição e orientação do robot. Este sensor compreende um giroscópio e um terminal para ligar uma antena GPS, para receção do sinal.

Tabela 3.7: Especificações sensor [10]

Especificações Gerais	
Taxa de atualização:	500 Hz
Taxa de dados de saída:	0 Hz to 255 Hz, dados selecionáveis
Comunicação:	3.3V TTL UART
Dados de saída:	Aceleração, ângulos(quaternion, Euler Angle), campo magnético, pressão barométrica, pressão baseado na altitude, GPS altitude, posição, velocidade, velocidade do ar
Vin nominal:	5.0V
Transmissões Suportadas:	9600, 14400, 19200, 38400, 57600, 115200, 128000, 153600,230400, 256000, 460800, 921600
Consumo de Potência:	< 150mA at 5.0V com GPS a procurar, < 100mA at 5.0V sem GPS a procurar
Temperaturas operacionais:	-40C to +85C
Dimensões:	1.5" x 1.3" x 0.5"
Peso:	11 grams
Especificações de GPS	
Precisão típica da posição:	2.5 m CEP(probabilidade de erro circular)
Precisão típica da velocidade:	0.1m/s
Precisão do tempo:	60ns
Max GPS dinamico:	< 4G
Limites Operacionais:	Altitude < 18,000 m Velocidade < 515 m/s
Open Sky TTFF:	29 second cold start, 1 second hot start
Especificações de Giroscópio	
Precisão em modo estático:	±1 graus
Precisão em modo Dinâmico:	±3 graus
Gama de medição do giroscópio:	± 2000 graus/s
Gama de medição do acelerómetro:	± 8G
Repetibilidade:	0.5 graus
Resolução:	< 0.01 graus
Mudança da sensibilidade com a temperatura:	± 2%
Densidade da taxa do ruído:	0.03 deg/s/rtHz
Não-linearidade:	0.2 % FS

Este sensor será projetado para devolver as coordenadas GPS na escala de graus decimais, conforme a equação 3.1. Nesta forma de coordenadas, Por sua vez, a matriz rotação obtida por parte do giroscópio virá sob a forma de ângulos de Euler (eq. 3.2), que será explicado em capítulos mais à frente a forma de obter os campos da matriz.

$$\begin{aligned} &\text{Longitude:} \quad P_x = \pm DD.DDDD \\ &\text{Latitude:} \quad P_y = \pm DD.DDDD \end{aligned} \tag{3.1}$$

$$R = \begin{bmatrix} X_x & Y_x & Z_x \\ X_y & Y_y & Z_y \\ X_z & Y_z & Z_z \end{bmatrix} \tag{3.2}$$

3.3 Mapeamento

O trabalho irá ser desenvolvido com o pressuposto do robot se deslocar no ambiente, de modo a conseguir chegar a um ponto final. No seu deslocamento recorre-se à posição lida por parte de um sensor GPS e à informação proveniente do sensor de distância, de modo a ser possível guardar o local dos objetos, para elaboração de um mapa.

A realização do mapa foi planeada para ser feita utilizando uma matriz criada no ambiente programático, de forma a representar o meio ambiente. O tamanho da matriz será tanto maior quanto o ambiente onde o robot está a operar e o fator de escala selecionado.

Uma posição da matriz representa um determinado ponto no mapa, onde a latitude e longitude, correspondem a um conjunto de linhas e colunas. Essas posições serão preenchidas com "0" caso o ambiente não tenha objetos, ou "1" se nas coordenadas correspondentes existir um objeto no mapa.

Ao gerar o mapa desta forma, há necessidade de escolher previamente o sistema de eixos de todos os dispositivos, assim como o referencial base assumido para as coordenadas GPS.

3.3.1 Sistema de eixos

Os sistemas de eixos, num projeto de robótica, são bastante importantes, visto que é através dos referenciais que se toma conhecimento do meio. Houve o cuidado de escolher o sistema de eixos antecipadamente.

O robot irá reger-se pelas coordenadas terrestres, estas poderão ser positivas e negativas, quer em longitude ou latitude. Por esse motivo, houve a preocupação de selecionar um conjunto de ambiente/referencial que pudesse, também, compreender estes dois tipos de coordenadas. Acabando por escolher um ambiente quadrado, cujo o referencial estivesse no meio do plano, assim, são testadas todas as possibilidades, podendo o robot ser implementado em qualquer parte do mundo.

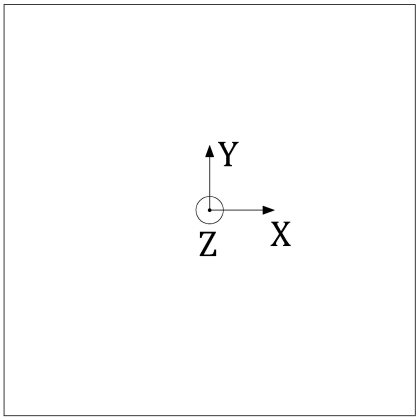


Figura 3.4: Exemplo sistema de eixos

O sistema de eixos do robot (sensor de posição), bem como o sistema de eixos do sensor Hokuyo URG-04LX-UG01 não foram deixados de parte. Estes dois dispositivos serão acoplados, portanto vão ter a sua disposição sempre igual em relação um ao outro.

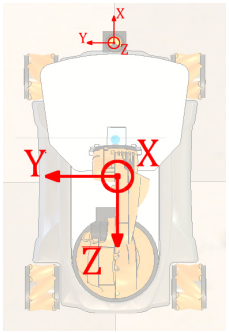


Figura 3.5: Sistema de eixos do Robot

3.4 Rota

O Planeamento de rota baseou-se nos processos mais comuns usados nesta área. Apoiando-se sobretudo nas estratégias de algoritmo *bug* 2.2.1, diagrama de Voronoi 2.2.2, *visibility graph* 2.2.2 e SLAM 2.2.3. A pesquisa realizada para conhecer os conceitos associados a estes tipos de planeamento de rota originaram uma ideia que englobava todos os processos mencionados.

Focado, sobretudo, na implementação de um robot com total desconhecimento do mundo de modo a descobrir o mapa do ambiente, surge a solução de SLAM. Esta ideia originou o começo deste projeto. Embora não se tenha implementado o código de SLAM, aproveitou-se o conceito que está subjacente, a ideia da construção do mapa à medida que o robot se movimenta. Neste caso será coadjuvado pelo sensor de GPS e pelo sensor distância.

A navegação por algoritmo *bug* mostra-se um recurso válido e simples de implementação. À semelhança do que acontece neste tipo de algoritmos, é traçada uma linha reta entre o ponto inicial e o ponto de destino e assim que é detetado uma colisão entre a rota e qualquer obstáculo é necessário ultrapassar o objeto. No entanto, para contornar os objetos não foi selecionado nenhum dos métodos mencionados na secção 2.2.1-Algoritmo Bug.

Quando é detetada uma colisão, recorre-se ao mapa que está à disposição no momento da colisão, de modo a obter os vértices do objeto em causa, um dos pontos será tomado pelo robot como sendo o próximo alvo. Esta ideia surge na pesquisa do algoritmo *visibility graph*. Contudo, como não se tem o conhecimento de todo o mapa torna-se impossível a implementação direta do código. Assim a obtenção dos vértices não é realizada fazendo um varrimento em torno dos objetos, será executado máscaras de processamento de imagem (bwmorph) existentes no MATLAB de forma a dilatar os objetos e por sua vez obter o diagrama de Voronoi dos mesmos. Com o diagrama de Voronoi dos objetos é possível obter os seus vértices, dado que estes são representados pelas extremidades de cada um dos ramos no diagrama de Voronoi.

De referir, também, que para o planeamento de rota será bastante importante o processo de backtracking. Este será o método para guardar as posições calculadas de modo a podermos ter acesso novamente às posições sem precisar de recalcular posições. Assim, se o caminho escolhido pelo robot for o errado poderá ser executado o caminho inverso.

3.5 Backtracking

Este processo foi o selecionado para evitar que o robot ficasse incapacitado de chegar ao ponto final caso escolhesse o caminho errado. Teve-se em consideração o referido numa das alíneas anteriores (2.3-Backtracking), e uma vez que o MATLAB não disponibiliza uma biblioteca ou forma de criar árvores, a sua implementação foi projetada sob forma matricial, composta por células sob forma de estrutura.

A árvore, criada sob forma de matriz, foi selecionada de modo a ser uma árvore binária com o máximo de profundidade de 4, ou seja o robot faz a busca contornando no máximo 4 objetos consecutivos. A árvore é composta por níveis e em cada nível existe as diferentes seleções. Será necessário saber a posição da matriz que o robot está a seguir, para tal é usado um "apontador"

que define o nível e a seleção do próximo ponto alvo do robot.

A posição inicial do robot é guardada na posição (1,1) da matriz sob forma de estrutura, representando a raiz da árvore. A partir desta poderão surgir novos ramos, caso haja conflito entre os objetos e a rota tomada. Nestes casos são definidas novas posições intermédias que são colocadas nos níveis seguintes (2,1) e (2,2). O processo será executado sucessivamente conforme a fig. 3.6. Atendendo ao facto de estar a ser usada uma árvore binária de profundidade 4 a matriz irá ser no máximo uma matriz 4x8. Contudo, esta matriz começa com o tamanho 1x1 e irá crescendo à medida que são introduzidas novas posições. Consoante é completada e são introduzidos os novos níveis à matriz, não é necessário usar toda a sua extensão, ficando algumas posições vazias, equação 3.4 .

Na execução deste processo é necessário guardar o nível (`level`) e a seleção (`selec`) onde se encontra o "apontador" da árvore, recorrendo a apontadores ou a variáveis para memorizar a localização na matriz.
As seleções são numeradas em cada nível de 1 até "n". Se observarmos, constata-se que a árvore segue uma sequência, os descendentes de qualquer um dos pontos $btree(level, selec)$ serão: $btree(level+1, selec \times 2-1)$ e $btree(level+1, selec \times 2)$. Assim, sempre que é para inserir valores na árvore basta incrementar um nível e introduzi-los nas devidas secções. Para retroceder níveis na matriz, basta fazer o processo inverso, detetando primeiro se a seleção em causa é par ou ímpar.

Os termos a adicionar em cada célula da matriz serão do tipo estrutura, apresentando dois campos: posição e confirmação. O campo de posição é preenchido com uma matriz linha com três colunas, representando as coordenadas geográficas x, y e z. O campo confirmação representa o número de possibilidades ou a impossibilidade do ponto, sendo preenchido com os números:

- 0 - Ainda não foi verificado, chegado a este ponto será necessário calcular novos pontos na árvore;
- 1 - Foi verificado havendo duas possibilidades no próximo nível;
- 2 - Foi verificado, há apenas uma possibilidade no nível a seguir;
- 3 - O ponto é impossível.

Ao desenvolver este sistema, há certas ocorrências que necessitam de uma atenção especial. O robot ao escolher novos pontos no mapa, poderá selecionar pontos pré-existentes na matriz, pois o mesmo ponto pode ser visto de várias ângulos. Se o ponto a inserir já existir na matriz árvore não há necessidade de adicionar duas vezes, estamos num caso de backtracking com backjumping. Assim, sempre que são introduzidos novos pontos na matriz teremos de recorrer a uma busca para determinar se a posição a inserir já existe. Caso se confirme, a estrutura a inserir na matriz passa a ser composta pelo campo de posição-vazia e o campo de confirmação passa ao valor 3, indicando que o caminho é impossível e vazio.

O formato da estrutura que se insere nas células da matriz (eq.3.3), bem como a árvore (fig.3.6) e a matriz `btree` (eq.3.4) que representa a árvore binária no MATLAB estão representadas em baixo.

$$btree(1,1) = \begin{cases} \text{Position} = \begin{bmatrix} x_i & y_i & z_i \end{bmatrix} \\ \text{confir} = value \end{cases} \tag{3.3}$$

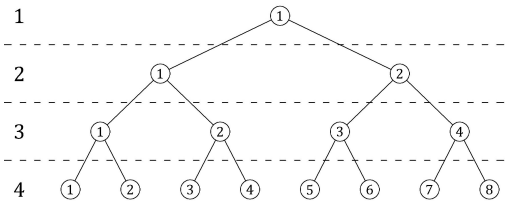


Figura 3.6: Árvore binária

$$btree = \begin{bmatrix} btree(1,1) & vazio & vazio & vazio & vazio & vazio & vazio & vazio \\ btree(2,1) & btree(2,2) & vazio & vazio & vazio & vazio & vazio & vazio \\ btree(3,1) & btree(3,2) & btree(3,3) & btree(3,4) & vazio & vazio & vazio & vazio \\ btree(4,1) & btree(4,2) & btree(4,3) & btree(4,4) & btree(4,5) & btree(4,6) & btree(4,7) & btree(4,8) \end{bmatrix} \tag{3.4}$$

4

Implementação

Este projeto conta na sua implementação com uma base de simulação para comprovar os resultados, para tal teve-se sempre em atenção a sua execução prática. Assim, houve a necessidade de projetar não só o código do programa, mas também a implementação física, descrevendo o modo como foi implementado o hardware selecionado no capitulo anterior.

A simulação do projeto foi concebida conjugando duas plataformas: O V-REP e o MATLAB. O projeto foi construido de maneira a que uma das plataformas representasse o meio físico, V-REP, e a outra desempenhasse a função do ambiente programático do robot, MATLAB.

4.1 Estrutura e Composição do ambiente

O V-REP tem um ambiente de simulação em modo três dimensões, onde é possível simular com algum detalhe a realidade. Neste programa está presente um ambiente que pode ser criado de diversas formas, onde se aplicam as leis fundamentais da física. Nesta plataforma está presente o robot e todo o hardware nele patente, bem como a consola de comando, onde é possível introduzir as coordenadas finais pretendidas.

O meio ambiente escolhido para simulação pretende representar a realidade, cujo o solo é uma superfície completamente plana e pouco rugosa com as dimensões de 10x10, cujo o referencial localiza-se no centro da superfície. Ou seja, estará a trabalhar com coordenadas num intervalo de -5 a 5, tanto em latitude como longitude. Na superfície podem ser introduzidos múltiplos objetos, contudo terão de ser sempre objetos estáticos.

O robot implementado é um robot da *KUKA*, descrito em 3.1, equipado com um braço robótico. Assume-se que este dispõe de um computador interno responsável pelo controlo das rodas e a leitura dos sensores.

O processo de controlo de velocidade de cada roda foi desenvolvido em linguagem lua, na plataforma V-REP. Associando um ficheiro ao conjunto robot móvel cujo conteúdo é, simples-

mente, o código do controlo das rodas. Este é executado lendo a posição do robot e a posição de destino, um ponto criado no V-REP denominado `middle_point`. O robot adequará a velocidade de cada roda tendo em conta as distâncias em x, y e ainda o valor das velocidades nos instantes anteriores. Desta forma o robot criará uma rota em linha recta até ao ponto desejado, designado por `middle_point` no V-REP. O processo de controlo das rodas não será um ponto muito debatido nesta dissertação, visto o foco do projeto ser a implementação de algoritmos de navegação.

A consola de comandos para inserção das coordenadas foi assumida como sendo um programa disponível no computador do robot, sendo por isso apresentada no V-REP.

Acoplado ao robot está o sensor responsável pelas medições aos objetos. Este sensor colocado na dianteira do robot, faz um varrimento horizontal obtendo um conjunto de pontos que correspondem à face dos objetos. A obtenção destes dados ficou a cargo do sensor da *Hokuyo* cujo o modelo é *URG-04lX-UG01*, descrito no ponto 3.2.1, que por sua vez estará conectado ao robot por USB. É de referir que poderá ser usado outro dispositivo de leitura, mas já que o sensor projetado tem uma taxa de atualização de 100ms/scan e visto a velocidade máxima do robot ser 0.8m/s, na pior das hipóteses será realizada uma medição completa de 0.08m em 0.08m, este será o rácio de leitura que está projetado. Caso seja usado outro dispositivo terá de manter-se a razão referida anteriormente.

No V-REP, existe, ainda, um sensor GPS e um sensor orientação. Estes sensores são bastante importantes, pois são os responsáveis por saber a posição e a orientação do robot, respetivamente. Os dados obtidos por estes sensores são processados, juntamente com os dados do sensor de distância, no computador interno do robot, de modo a poderem ser enviados, posteriormente, para o MATLAB. Os dados deste equipamento são gerados sob a forma de UART 3.3V, todavia pode ser implementado diretamente no computador do robot, utilizando um dispositivo para converter o sinal para USB.

O sinal a enviar para o MATLAB será composto pelos sinais provenientes do sensores (distância, orientação e posição), construído sob forma de matriz coluna. Seguir-se-á a disposição dos dados dos sensores e a descrição da realização da matriz que é enviada para o MATLAB.

4.1.1 Matriz Transformação

Os dados recebidos através dos sensores de posição e orientação terão de ser tratados pelo computador interno do robot, de modo a conseguir formar uma matriz transformação, sempre com a referência à origem do referencial base, no centro do plano. Analisando a matriz transformação, esta será uma matriz quadrada 4x4 que pode ser dividida em quatro partes:

$$\begin{array}{c|c} R & T \\ \hline 0 & E \end{array} \tag{4.1}$$

- R - componentes de rotação(submatriz 3x3);
- T - componentes de translação(submatriz 3x1);
- E - componentes de escala(unitário);

A matriz rotação, provém do sensor de orientação. Os cálculos internos na máquina V-REP são feitos em sistema de quaterniões, porém será sempre apresentado ao usuário os ângulos regendo-se pelos ângulos *Euler* (ψ, θ, ϕ)[15]. Assim sendo, a matriz de rotação é dada por:

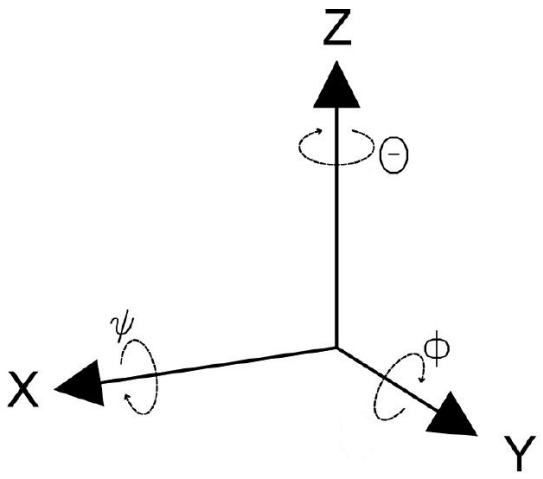


Figura 4.1: Ângulos *Euler*

$$R_x(\psi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\psi & -\sin\psi \\ 0 & \sin\psi & \cos\psi \end{bmatrix} \tag{4.2}$$

$$R_y(\phi) = \begin{bmatrix} \cos\phi & 0 & \sin\phi \\ 0 & 1 & 0 \\ -\sin\phi & 0 & \cos\phi \end{bmatrix} \tag{4.3}$$

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.4}$$

$$R(\psi, \phi, \theta) = R_x(\psi).R_y(\phi).R_z(theta) \tag{4.5}$$

A matriz translação é composta por um vetor com três componentes em x, y e z, a posição do robot em relação à origem, derivado do GPS.

$$T = \begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix} \tag{4.6}$$

O fator de escala "E"será sempre unitário, visto não estar a ser usado este campo para obtenção da posição.

No final, com os dados da matriz rotação "R"e a matriz translação "T"descrita anteriormente, obtém-se a matriz transformação:

$$P = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.7}$$

- $a_{11} = \cos(\theta)\cos(\phi)$
- $a_{12} = -\sin(\theta)\cos(\phi)$
- $a_{13} = \sin(\phi)$
- $a_{14} = P_x$
- $a_{21} = \sin(\psi)\sin(\phi)\cos(\theta) + \cos(\psi)\sin(\theta)$
- $a_{22} = -\sin(\psi)\sin(\phi)\sin(\theta) + \cos(\psi)\cos(\theta)$
- $a_{23} = -\sin(\psi)\sin(\phi)$
- $a_{24} = P_y$
- $a_{31} = -\sin(\phi)\cos(\psi)\cos(\theta) + \sin(\psi)\sin(\theta)$
- $a_{32} = \sin(\psi)\sin(\phi)\sin(\theta) + \sin(\psi)\cos(\theta)$
- $a_{33} = \cos(\psi)\cos(\phi)$
- $a_{34} = P_z$

4.1.2 Matriz sensor distância

O sensor Hokuyo efetua várias medições a cada ciclo de scan, varrendo na horizontal um ângulo de 240º, para no final devolver uma matriz com todas as medições efetuadas, sendo todas referenciadas à posição do sensor.

É pertinente informar que o sensor usado para a simulação é uma versão do sensor Hokuyo. Esta versão disponibilizada pela plataforma V-REP, tem como objetivo acelerar o processamento dos dados. Enquanto que o sensor real devolve todos os 666 valores, de 0.36º em 0.36º, o sensor da

simulação apenas devolve os valores quando o sensor fica ativo, ou seja, nas posições de medida onde o espectro é recebido de volta, o laser embate, dentro da área de deteção, num objeto e regressa ao recetor. No caso do espectro não regressar não é inserido o valor à matriz.

A matriz formada pelos dados do sensor distância será do tipo matriz coluna de tamanho variável, com o número de linhas sempre múltipla de três. Composta pela sequência de posições em x, y e z a que se encontram os objetos do sensor.

Definindo as medições com a letra m, numeradas de 1 ao número máximo de medições detetadas, designadas por n. Sendo cada medição composta pelas coordenadas x, y e z a que se encontram os objetos do sensor. Teremos então n medições definidas como:

- $m_1 = \begin{bmatrix} m_{x1} & m_{y1} & m_{z1} \end{bmatrix}$
- $m_2 = \begin{bmatrix} m_{x2} & m_{y2} & m_{z2} \end{bmatrix}$
  $\cdots$
- $m_n = \begin{bmatrix} m_{xn} & m_{yn} & m_{zn} \end{bmatrix}$

A matriz do sensor será a junção de todas as medições realizadas, dispostas do seguinte modo:

$$M^T = \begin{bmatrix} m_{x1} & m_{y1} & m_{z1} & m_{x2} & m_{y2} & m_{z2} & \cdots & m_{xn} & m_{yn} & m_{zn} \end{bmatrix} \tag{4.8}$$

4.1.3 Matriz/Sinal MATLAB

A construção da matriz a ser enviada para o MATLAB é realizada recorrendo às matrizes citadas anteriormente, juntando a matriz transformação do robot (eq. 4.7), proveniente do sensor de orientação/posição e a matriz do sensor distância (eq.4.8). Dispondo, primeiramente, a matriz transformação, com dimensão constante, excluindo a última linha de fator de escala, e seguidamente a matriz do sensor de distância supracitada.

$$\begin{aligned} M^T_{VREP} = [&a_{11}\ a_{12}\ a_{13}\ a_{14}\ a_{21}\ a_{22}\ a_{23}\ a_{24} \\ &a_{31}\ a_{32}\ a_{33}\ a_{34}\ m_{x1}\ m_{y1}\ m_{z1} \\ &m_{x2}\ m_{y2}\ m_{z2}\ \cdots\ m_{xn}\ m_{yn}\ m_{zn}] \end{aligned} \tag{4.9}$$

Para a criação da matriz é executado um ficheiro Lua, agregado ao dispositivo "fasthokuyo", sensor presente no V-REP a versão do sensor hokuyo. Neste ficheiro, encontra-se ainda a criação

do canal fullduplex de forma a ser realizada a comunicação com o MATLAB, enviando constantemente a matriz M_{VREP}, para o canal.

4.2 Algoritmo de Controlo

4.2.1 Estrutura Geral

O desenvolvimento do algoritmo de controlo do robot foi realizado na íntegra no MATLAB. Contando, nesta plataforma, com um programa cuja estrutura é semelhante a ambientes de programação procedimental, nele encontrar-se-á uma função principal (main) executada em *loop* e a partir desta função serão chamadas todas as funções necessárias para controlar corretamente o robot.

O programa começa pela conexão dos programas, V-REP com MATLAB, estabelecendo a ligação com o canal full-duplex criado no V-REP, de maneira a ser possível realizar a comunicação entre as duas plataformas. Após a ligação dos programas, o processo entra em *loop* continuo até que haja quebra da ligação. Dentro do *loop* pode-se contar com a atualização das variáveis e sua interpretação, seguida do mapeamento e planeamento de rota.

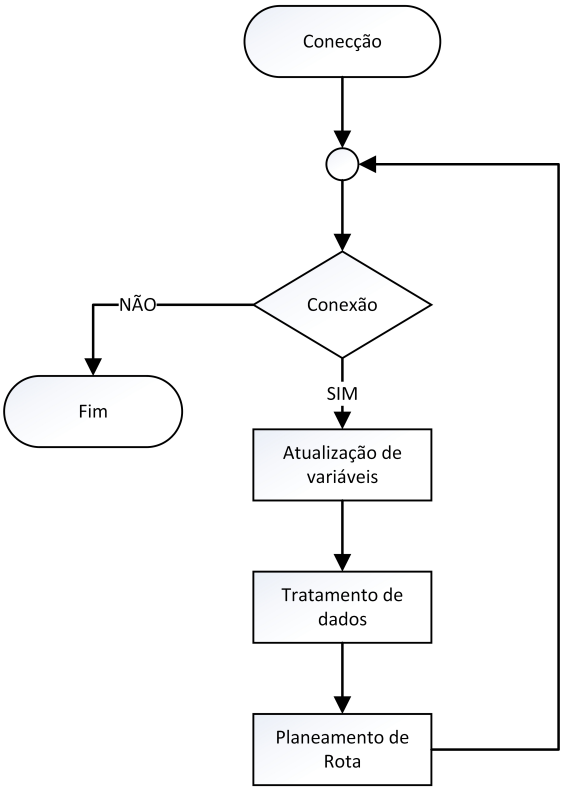


Figura 4.2: Fluxograma da função “main”

4.2.2 Atualização de variáveis

Este conjunto de funções foi definido para ler as variáveis provenientes do V-REP. Recorrendo à comunicação existente, é executado código especifico que faz parte do pacote API-Remote do V-REP, sendo atualizadas as variáveis sempre que um ciclo é executado.

4.2.3 Tratamento de dados

As variáveis provenientes do V-REP, obtidas na fase anterior terão de ser tratadas de modo a poderem ser utilizadas no resto do código. Por conseguinte, ter-se-á de descompactar os dados provenientes do V-REP de modo a poder obter as duas variaveis desejadas. Para tal, recorreu-se a três funções essenciais no descompactamento e tratamento de dados: `decompose`, `mapa` e `ceal_matrix`.

decompose

Esta função recebe o sinal proveniente do V-REP (M_{VREP}) e descompacta o sinal de forma a obter a matriz transformação do robot e a posição de todos os objetos medidos pelo sensor de distância.

O processo de descompactar os dados será parecido ao executado no V-REP, mas o processo é invertido. Inicialmente obtém-se a matriz transformação do robot recorrendo às primeiras 12 posições da matriz recebida (M_{VREP}), todavia com as 12 posições é impossível formar a matriz homogénea de transformação. Posto isto, é necessário introduzir a última linha de fator de escala, sendo a matriz homogénea guardada numa outra variável (`position`). As posições utilizadas são eliminadas da matriz M_{VREP}, ficando apenas uma matriz composta pelas distâncias do sensor aos diversos objetos.

A matriz M_{VREP} encontra-se agora com os valores que dizem respeito apenas aos dados do sensor Hokuyo. Os dados são agrupados três a três formando as várias medidas aos objetos detetados. Uma vez que o sensor de distância está muito perto do robot serão detetados pontos pertencentes ao robot, sendo necessário anulá-los. Analisando ao pormenor o acontecimento (fig. 4.3), terão de ser retirados todos os pontos no interior da caixa negra, ou seja, todas as coordenadas que estejam compreendidas: $-0,0677 < x_{sensor} < 0$ e $-0,2 < y_{sensor} < 0,2$ do referencial do sensor.

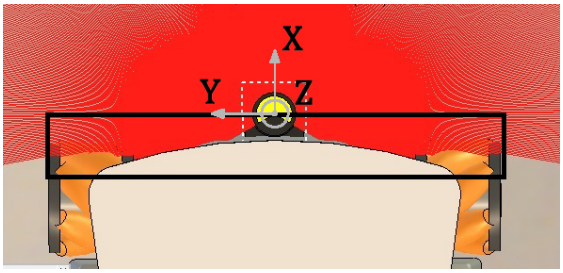


Figura 4.3: Pormenor sensor VREP

No final obtém-se a matriz transformação descrita em 4.1.1 e uma matriz formada pelas posições dos objetos, ainda com a sua referência no sensor de distância, matriz 4.10.

$$\texttt{sensorcorr} = \begin{bmatrix} O_{x1} & O_{y1} & O_{z1} \\ O_{x2} & O_{y2} & O_{z2} \\ & \vdots & \\ O_{xn} & O_{yn} & O_{zn} \end{bmatrix} \tag{4.10}$$

Pseudocódigo

```
function [posição,sensorcorr] = decompose(M_{VREP})

 posição=[M_{VREP}(1) M_{VREP}(2)  M_{VREP}(3)  M_{VREP}(4);
          M_{VREP}(5) M_{VREP}(6)  M_{VREP}(7)  M_{VREP}(8);
          M_{VREP}(9) M_{VREP}(10) M_{VREP}(11) M_{VREP}(12);
               0           0           0           1      ];
 M_{VREP}=M_{VREP}([13:length(M_{VREP})]);  %% Anular os 12 dados anteriores
    for i=1:tamanho(M_{VREP})/3
        sensor(i,1)=M_{VREP}(i*3-2);
        sensor(i,2)=M_{VREP}(i*3-1);
        sensor(i,3)=M_{VREP}(i*3);
    end

    %% Eliminar posições pretencentes ao robot
    for i=1:tamanho(sensor)
        if (fora da caixa)
            sensorcorr(j,:)=sensor(i,:);
            j=j+1;
        end
    end
 end
```

Após a obtenção/atualização dos dados, antes de mais, é necessário guardar a posição inicial do robot. Esta ação vai ser fulcral para a construção da matriz de backtracking. A posição inicial do robot, obtida pela variável `position` é guardada sob forma de estrutura conforme a equação 3.3, na $btree(1,1)$ - a raiz da árvore. Guardada a posição inicial do robot e adquiridos todos os dados, pode proceder-se à construção do mapa.

mapa

De modo a poder realizar a construção do mapa, terá de se determinar a localização exata dos objetos. Para esse procedimento recorre-se à matriz do sensor de distância `sensorcorr`, todavia esta matriz é constituída por um conjunto de pontos cuja referência está no sensor. Por esse motivo é impossível determinar a posição exata dos objetos no mapa, uma vez que o referencial das medições é móvel. Posto isto, terão de ser resolvidos cálculos de transformações de matrizes para determinar a localização dos objetos na referência base do sistema GPS. Para tal, vamos tomar em consideração as matrizes definidas como:

Tabela 4.1: Lista de referenciais

Definição	Programação	Comentário
${}^{0}P_{robot}$	`BtoR`	Robot no referencial base do sistema
${}^{robot}P_{sensor}$	`RtoTV`	Sensor no referencial do robot
${}^{sensor}P_{obj}$	`TVtoO`	Objeto no referencial do sensor Hokuyo
${}^{0}P_{obj}$	`BtoO`	Objeto no referencial base

As variáveis acima são matrizes homogéneas, que correspondem às transformações das matrizes de um referencial para outro.

Observando todas as variáveis, constata-se que estão disponíveis para utilização: ${}^{0}P_{robot}$ proveniente do sensor posição/orientação; ${}^{sensor}P_{obj}$ derivado do sensor Hokuyo. Pode ainda obter-se através de medições previamente realizadas, a matriz transformação entre o centro de massa do robot e o sensor de distância ${}^{robot}P_{sensor}$, sendo que o sensor está aparafusado ao robot, manterá sempre a mesma distância e orientação para com o centro de massa do robot.

$$ {}^{0}P_{robot} = \begin{bmatrix} X_x & Y_x & Z_x & P_x \\ X_y & Y_y & Z_y & P_y \\ X_z & Y_z & Z_z & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^{robot}P_{sensor} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & -0.3005 \\ 0 & 0 & 1 & -0.0154 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^{sensor}P_{obj} = \begin{bmatrix} 1 & 0 & 0 & O_{xn} \\ 0 & 1 & 0 & O_{yn} \\ 0 & 0 & 1 & O_{zn} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.11} $$

X_x, X_y, X_z - projeção do eixo x no referencial base;
Y_x, Y_y, Y_z - projeção do eixo y no referencial base;
Z_x, Z_y, Z_z - projeção do eixo z no referencial base;
P_x, P_y, P_z - posição do robot no referencial base;
O_x, O_y, O_z - distância do objeto ao sensor;

Recorrendo às matrizes supracitadas, é possível obter ${}^{0}P_{obj}$, que corresponde à posição dos objetos no referencial base. Calculando o produto interno das matrizes conhecidas consegue-se definir o local exato dos objetos no referencial do sistema.

$$ {}^{0}P_{obj} = {}^{0}\ P_{robot}.{}^{robot}P_{sensor}.{}^{sensor}P_{obj} \tag{4.12} $$

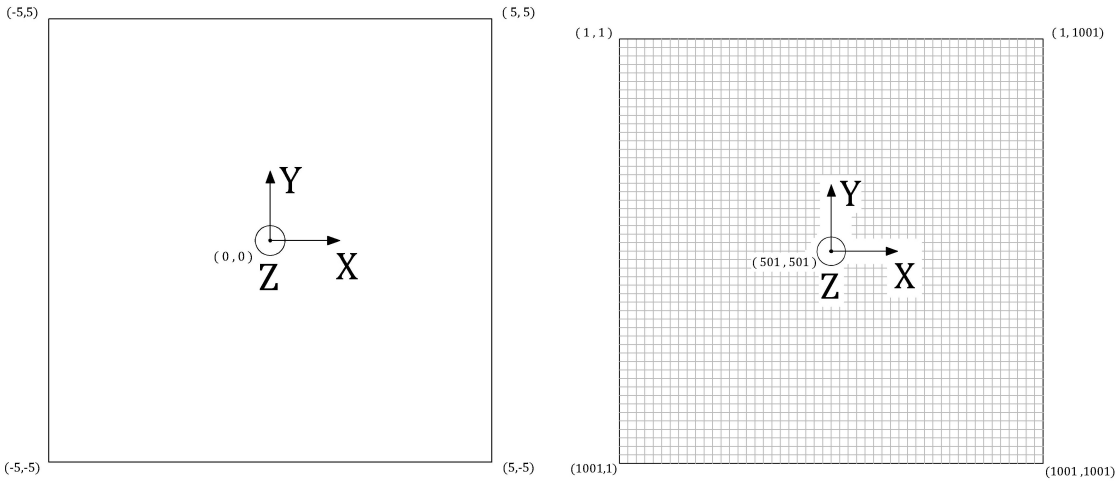
As diversas posições encontradas através do calculo supracitado, não podem ser imediatamente guardadas, na medida em que exigiria uma enorme capacidade de memória para armazenar todo o mapa. Apesar de neste caso o mapa estar restrito a uma dimensão (10x10), na situação real o mapa poderá ser bem maior exigindo por isso uma grande capacidade de armazenamento.

Assim, é conveniente recorrer-se a um fator de escala para guardar o mapa. O mapa será guardado sob forma de matriz, semelhante a uma imagem binária, assim com este procedimento é possível usar máscaras e filtros de imagem, existentes no MATLAB, que serão úteis no cálculo das trajetórias.

Neste projeto é considerada uma matriz 1001x1001, `maps`, que irá representar o ambiente em 2D, onde cada uma das células da matriz, que forma o mapa, representa 0.01m no ambiente de simulação. Com este processo perde-se algum detalhe da posição do robot e dos objetos, porém é a unidade suficiente para termos o correto funcionamento de todo o processo.

Ao contrário das coordenadas GPS, a matriz `maps` não pode conter posições negativas. Assim, para além do fator de escala a aplicar, terá de se recorrer a uma translação, para que o zero do referencial seja no centro da matriz. Pode-se observar através da imagem 4.4 que as linhas da matriz representarão as coordenadas em y e as colunas as coordenadas em x. Logo, a transformação a aplicar para passar qualquer posição do mapa para a matriz será:

$$sim(\,x\,,\,y\,) \quad = \quad maps(\,-Posição_y * 100 + 501\,,\,Posição_x * 100 + 501\,) \tag{4.13}$$



(a) Ambiente de simulação(sim)

(b) Matriz mapa (`maps`)

Figura 4.4: Ambiente e matriz programação

Cada posição da matriz será preenchida por zeros, que correspondem a espaços não ocupados por objetos. Sempre que exista posições na `sensorcorr`, estas sofrem a transformação indicada acima e a posição da matriz `maps` passa a valer "1" indicando que essa posição está ocupada por um objeto.

Pseudocódigo

```
function mapa=mapa(posição, sensorcorr)

 BtoR=posição;          %%Posição do robot

 RtoTV=[0  1 0    0   ;              %%Matriz centro de massa robot ao sensor
        -1 0 0 -0.3005;
        0  0 1 -0.0154;
        0  0 0    1  ];


 for i=1:tamanho(sensorcorr)

     TVtoO=[1 0 0 sensorcorr(i,1);
            0 1 0 sensorcorr(i,2);
            0 0 1 sensorcorr(i,3);
            0 0 0     1      ];

     BtoO=BtoR*RtoTV*TVtoO;

     x=fix(-(BtoO(2,4))*100+501);
     y=fix((BtoO(1,4))*100+501);

    Confirmar casos extremos, limites do mapa

    maps(x,y)=1;
 end
end
```

clean_matrix

Este troço de programa é responsável pela limpeza e tratamento da matriz mapa (`maps`), de forma a evidenciar e tornar mais densos os objetos. Os processos aqui utilizados são filtros e máscaras pré existentes no MATLAB.

Os objetos na matriz são definidos por pontos que foram detetados pelo laser do sensor Hokuyo. Se fosse utilizado um mapa deste tipo, constituído apenas por pontos, seria impossível determinar novos pontos intermédios segundo o critério que está planeado. Então, para o cálculo dos novos pontos objetivo teve de se considerar um novo mapa. Este será uma modificação do

mapa original, construído apenas por pontos, e será obtido através de uma máscara de dilatação de imagem (4.5).

0	0	0	0	0
0	0	0	0	0
0	0	1	0	0
0	0	0	0	0
0	0	0	0	0

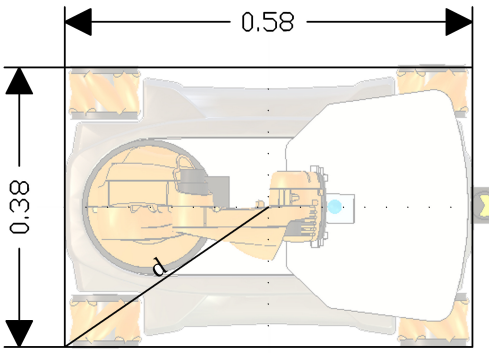
(a) Matriz sem dilatação

0	0	0	0	0
0	1	1	1	0
0	1	1	1	0
0	1	1	1	0
0	0	0	0	0

(b) Matriz com dilatação

Figura 4.5: Máscara dilatação

Este processo vai agregar todos os pontos detetados que pertençam ao mesmo objeto e será determinante para o cálculo de novos pontos. Por isso, o processo não será executado apenas uma vez, ter-se-á em conta a realização duma dilatação, de maneira a que o robot ao definir novos pontos utilizando os vértices dos novos objetos dilatados, consiga evitar as colisões ao rodar sobre si mesmo sobre os pontos intermédios calculados. Observando a figura 4.6, a dilatação terá de ser maior que a distância "d", a maior distância existente do centro de massa do robot à extremidade do robot.



$$d = \sqrt{(0.58/2)^2 + (0.38/2)^2} = 0.3467 \tag{4.14}$$

Figura 4.6: Distância de colisão

Atendendo a que uma célula da matriz `maps` equivale 0.01 no mapa real, do ambiente V-REP, terão de ser executadas 35 dilatações, correspondentes ao cálculo 4.14 da distância de segurança deixada para a rotação do robot. Após a dilatação da imagem, é executado agora o filtro para obter o diagrama de Voronoi do mapa. Com este novo mapa saberemos onde estão os vértices dos objetos, importantes para a determinação dos pontos intermédios.

Por fim, é executado um comando para diferenciar os diversos objetos, numerando os vários objetos detetados, por forma a quando uma colisão saber o objeto em causa.

Pseudocódigo

```
function clean_matrix()

  dil=bwmorph(maps,'dilate',35);
  cormap=bwmorph(dil,'skel','inf');
  cormap=bwlabel(cormap,8);

end
```

4.2.4 Planeamento de rota

Desenvolvido o tratamento e atualização dos dados, proceder-se-á à interpretação dos mesmos.

Esta fase do programa, está dividida em três partes: uma primeira que corresponde às decisões, onde é avaliada a rota e tomado conhecimento da localização do robot; uma segunda fase de criação de novas rotas, onde é executado um conjunto de funções de forma a criar novos pontos objetivos bem como a inserção desses pontos na matriz árvore; a última parte diz respeito à orientação e deslocamento do robot, em que se verifica a orientação do robot. Importante de referir que o robot apenas se dirige para o ponto objetivo quando a sua orientação for a indicada.

Decisões do Planeamento

Nesta primeira fase do planeamento de rota, terá de se começar por averiguar a localização do robot. Este apenas compreenderá três estados distintos: o robot encontra-se na posição final, ou numa das posições intermédias definidas na árvore binária, ou está a deslocar-se.

Quando detetado que o robot atinge o seu ponto objetivo, ele permanecerá no ponto final esperando a quebra de ligação. Assim que termina a ligação, o MATLAB devolve amostras dos mapas obtidos, assim como os pontos intermédios e os pontos de passagem.

Eventualmente, a localização do robot pode ser um dos pontos intermédios calculados. Aqui, o robot terá de verificar se se confirma uma rota até ao ponto final. Na hipótese de existir uma rota plausível, essa será a tomada pelo dispositivo, porém se não se confirmar uma rota terá de se calcular um novo ponto intermédio, de forma a conseguir ultrapassar o obstáculo que existe entre o robot e a coordenada final.

Se por acaso o robot não está em nenhum dos casos anteriores, ou seja, encontra-se a cumprir uma rota. Também, nestes casos é verificada a colisão com os objetos. Atendendo ao facto de o sensor de distância ter um alcance máximo e visto não ser conhecida a totalidade do mapa, é necessário uma constante confirmação de colisão no deslocamento do robot, pois à medida que este se movimenta pode descobrir novos objetos, que poderão estar em rota de colisão.

A confirmação de qualquer rota é efetuada usando a função `configuration_lane`.

Pseudocódigo

```
if (~= ponto final)
%% Confirmação se existe caminho
   configuration_lane(rob, newpos);
   if collision==0
        if (ponto intermédio)
        %% verifica se há caminho para o ponto final
            configuration_lane(rob,goal)
            if collision==0
                newpos=goal;
            end
        end
   end
```

configuration_lane

Esta função recebe dois pontos, o ponto inicial sendo sempre as coordenadas do robot atualizadas e o ponto objetivo, um dos pontos intermédios da matriz árvore ou ponto final. O objetivo desta função é configurar 6 pontos para confirmar se a rota é possível de ser executada. Os pontos são criados de maneira que ao serem unidos par a par, criem segmentos de retas paralelas. Os segmentos vão desde o centro de massa do robot até ao ponto de destino, contudo, devem ainda ser criados outros dois segmentos de reta nas extremidades laterais do robot. Posteriormente terá de ser confirmado se todos os segmentos de reta pertencem ao espaço livre do mapa, pois só assim se confirma que existe uma rota até ao ponto destino.

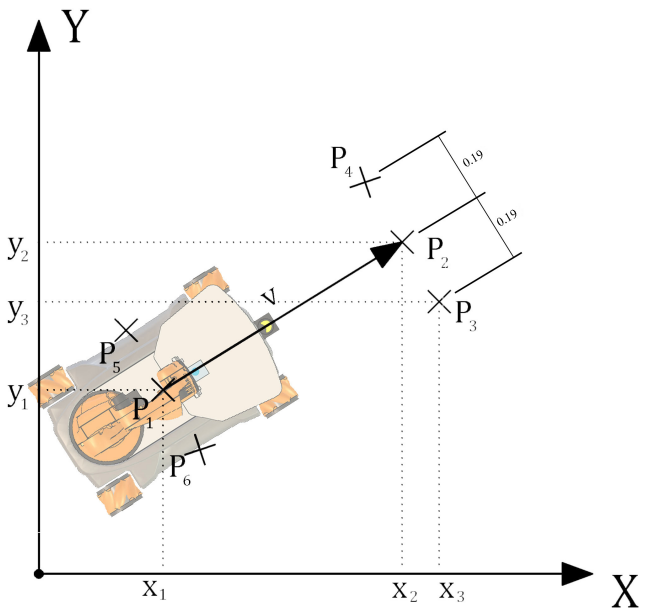


Figura 4.7: Estudo de vectores

Na realização do algoritmo ter-se-á de considerar a distância entre os dois pontos, ponto inicial ($P_1 = [x_1; y_1]$) e ponto final($P_2 = [x_2; y_2]$).

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2)}$$

Procede-se de seguida, à criação dos pontos nas extremidades laterais do robot. Estes pontos serão colocados paralelamente à rota. Analisando a fig. 4.7, pode-se afirmar que o vetor v é a rota tomada pelo robot, e define-se o vetor unitário (v'), representado por:

$$v' = \left(\frac{x_2 - x_1}{d}; \frac{y_2 - y_1}{d}\right)$$

Será um vetor perpendicular a v' o vetor unitário n', estes serão importantes para originar os pontos P_3,P_4,P_5 e P_6.

$$n' = \left(\frac{y_2 - y_1}{d}; \pm\frac{x_1 - x_2}{d}\right)$$

Recorrendo aos dois possíveis vetores unitários perpendiculares à rota e multiplicando pela distância do centro de massa à extremidade lateral do robot (0.195), obtém-se um vetor que terá o comprimento desejado e será perpendicular à rota. Posteriormente, é realizada uma translação de forma a que os vetores se desloquem para os pontos P_1 e P_2, assim tem-se os pontos P_3, P_4, laterais ao ponto destino e P_5 e P_6, nas extremidades laterais do robot.

$$P_3 = \left(x2 + \frac{y_1 - y_2}{d} \times 0.195; y2 + \frac{x_2 - x_1}{d} \times 0.195 \right) \tag{4.15a}$$

$$P_4 = \left(x2 + \frac{y_1 - y_2}{d} \times 0.195; y2 + \frac{x_1 - x_2}{d} \times 0.195 \right) \tag{4.15b}$$

$$P_5 = \left(x1 + \frac{y_1 - y_2}{d} \times 0.195; y1 + \frac{x_2 - x_1}{d} \times 0.195 \right) \tag{4.15c}$$

$$P_6 = \left(x1 + \frac{y_1 - y_2}{d} \times 0.195; y1 + \frac{x_1 - x_2}{d} \times 0.195 \right) \tag{4.15d}$$

Com os pontos encontrados é necessário verificar se os segmentos pertencem ao espaço vazio, cujo processo é realizado recorrendo à função `testobject`. Portanto, esta função será chamada três vezes, confirmando os pares:

P_1-P_2;
P_6-P_3;
P_5-P_4.

Por fim, a função retorna o valor da função `testobject`, correspondendo ao objeto que está em colisão.

Pseudocódigo

```
function configuration_lane(inicial_point, end_point)

d=sqrt((inicial_point(1)-end_point(1))^2+(inicial_point(2)-end_point(2))^2);

rro=[inicial_point(1)+(-(inicial_point(2)-end_point(2))/d)*0.195
         inicial_point(2)+((inicial_point(1)-end_point(1))/d)*0.195 0 ];
lro=[inicial_point(1)+((inicial_point(2)-end_point(2))/d)*0.195
         inicial_point(2)+(-(inicial_point(1)-end_point(1))/d)*0.195 0 ];

rta=[end_point(1)+(-(inicial_point(2)-end_point(2))/d)*0.195
         end_point(2)+((inicial_point(1)-end_point(1))/d)*0.195 0 ];
lta=[end_point(1)+((inicial_point(2)-end_point(2))/d)*0.195
         end_point(2)+(-(inicial_point(1)-end_point(1))/d)*0.195 0 ];

    collision=testobject(rro,rta);
    if(collision==0)
        collision=testobject(lro,lta);
            if(collision==0)
                collision=testobject(inicial_point, end_point);
                           end
        end
end
```

testobject

Esta função tem como objetivo verificar as colisões entre dois pontos. A função recebe dois pontos e traça uma reta entre eles, verifica cada posição da matriz (`maps`) pertencente à reta, de forma a saber se existe algum objeto contido na reta criada.

Primeiramente, será executada uma interpolação linear entre os dois pontos, de modo a descobrir o declive e a coordenada no eixo das ordenadas da reta. De seguida, procede-se ao varrimento da reta no eixo das abcissas, usando a interpolação calculada e testa-se se os pontos da matriz `maps` estão ocupados por objetos. Caso qualquer ponto sobre a reta seja diferente de zero, é parado o ciclo de varrimento e retornando o valor do objeto.

Em casos extremos de linearização, como é o caso dos dois pontos estarem na vertical, o declive será infinito, requerendo por isso uma restrição particular. Nestes casos, será feito um varrimento nas ordenadas mantendo o mesmo valor de "x", verificando célula a célula a matriz

"maps".

Ao executar o ciclo "for" há que ter em atenção quais dos pontos recebidos é maior e qual é menor, devendo o ciclo ser realizado de maneira a ser incremental, assim sendo é necessário executar um teste a verificar esse aspeto.

Pseudocódigo

```
function collision=testobject(begin,goal)

collision=0;

%% linear parameters
m=(yg-yr)/(xg-xr);
b=yg-m*xg;

        if (declive infinito)

                varrimento da reta

        else
        %% declive infinito, reta vertical

            varrimento da reta vertical

        end
end
```

Definição de Novos Pontos

Este processo só será executado caso se confirmem colisões no traçado do robot. Assim que seja detetado um objeto, será analisada a árvore binária, responsável pela memorização dos pontos intermédios, a fim de saber o estado da estrutura.

Na criação de novos pontos e incrementação desses pontos na árvore de backtracking (`btree`) é estritamente necessário ter conhecimento das variáveis nível (`level`) e a seleção (`select`) do "apontador" da matriz estrutura. Tendo em vista a inserção dos pontos na matriz foi criada uma função para desempenhar esta função e uma outra para o decremento do nível de maneira a ser mais fácil a navegação pela árvore.

ins_tree

Este procedimento diz respeito à matriz estrutura, encarregando-se de introduzir coordenadas na árvore binária.

Esta função recebe os dois pontos intermédios a adicionar à matriz, `position1` e `position2`, ambos vêm sob a forma de coordenadas, já mencionadas ($[\,x\ y\ z\,]$). Primeiramente, o algoritmo faz uma busca completa pela matriz estrutura, com a finalidade de verificar a existência ou não destes pontos na matriz `btree`, pois o mesmo ponto pode ser calculado de diferentes perspetivas. Os pontos intermédios calculados pelo robot podem ter uma gama de variação devido às atualizações do mapa. Assim, tem-se de testar se `position1` e `position2` estão próximos de algum dos pontos contidos na matriz. Recorrendo ao teorema de Pitágoras terá de se calcular a distância entre pontos recebidos e qualquer ponto da matriz, da forma que aparece na equação 4.16. Sempre que a distância seja inferior a 0.2 os pontos são considerados o mesmo, não sendo necessário introduzi-los.

$$dist(btree(i,j);\texttt{position1}) = \sqrt{(btree(i,j)_x - \texttt{position1}_x))^2 + (btree(i,j)_y - \texttt{position1}_y))^2} \tag{4.16}$$

A busca consiste em dois ciclos `for` de modo a varrer as linhas e colunas da matriz, é executado o cálculo mencionado anteriormente em cada posição da matriz verificando se algum dos pontos a introduzir já existe na matriz. Caso se verifique a existência de um dos pontos é acionado as variáveis `n1` ou `n2`, estas variáveis binárias afirmam se as posições `position1` ou `position2` já estão na matriz. Deste modo, teremos 4 condições plausíveis, que foram tidas em conta na realização do algoritmo.

`n1`	`n2`	Comentário
0	0	ambas as variaveis são válidas, adicionadas ambas à matriz
0	1	apenas `position2` é válida `position1` já está presente na matriz, não é adicionada
1	0	apenas `position1` é válida `position2` já está presente na matriz, não é adicionada
1	1	ambas as posições já estão presentes na matriz, não é adicionada nenhuma posição. retorna ao nível anterior

Testada a existência das coordenadas na matriz, procede-se à inserção das coordenadas conforme a combinação obtida por `n1` e `n2`, no nível e posição adequadas.

Pseudocódigo

```
function ins_tree(position1, position2)  %% select position1

    n1=0;
    n2=0;

    if level==0
        btree(1,1).pos=position1;
        level=1;
    else

        ciclo for para varrer a matriz
               dist1=pitágoras(btree(i,j).pos,position1);
               dist2=pitágoras(btree(i,j).pos,position2);

               if (dist1<0.2)
                   n1=1;
               end
               if (dist2<0.2)
                   n2=1;
               end

        end

        if and(n1==0,n2==0)
            level=level+1;
            select=select*2-1;
            btree(level,select)=struct('pos',position1,'confir',1);
            btree(level,select+1)=struct('pos',position2,'confir',0);
        elseif and(n1==0,n2==1)
            btree(level,select).confir=2; %% só há uma hipotese no prox nivel
            level=level+1;
            select=select*2-1;
            btree(level,select)=struct('pos',position1,'confir',1);
            btree(level,select+1)=struct('pos',[],'confir',3)
        elseif and(n1==1,n2==0)
            btree(level,select).confir=2; %%só há uma hipotese no prox nivel
            level=level+1;
            select=select*2;
            btree(level,select-1)=struct('pos',[],'confir',3);
            btree(level,select)=struct('pos',position2,'confir',1);
        elseif and(n1==1,n2==1)
            btree(level,select).confir=3; %%não há hipoteses no nivel seguinte
            btree(level+1,select*2-1)=struct('pos',[],'confir',3);
            btree(level+1,select*2)=struct('pos',[],'confir',3);
            remove_tree();
        end
    end
end
```

remove_tree

Esta função será chamada sempre que seja necessário decrementar níveis na árvore binária. Toma em consideração o nível(`level`) e a seleção(`select`) do "apontador", responsável por guardar a posição da árvore. Sempre que é decrementado o nível, é executado um teste de forma a saber se a seleção onde está o "apontador" é par ou ímpar. Se seleção par, decrementa o nível e a variável seleção ficará a valer o quociente da select/2. Se a seleção for ímpar, decrementa-se na mesma o nível e a seleção ficará com o valor (select+1)/2, este processo foi baseado na regularidade da árvore, descrito na secção 3.5. Por fim, na posição selecionada é incrementado o campo de confirmação, visto que foi confirmado um dos caminhos.

Pseudocódigo

```
function remove_tree()

    level=level-1;

    if mod(select,2)
        select=(select+1)/2;
    else
        select=select/2;
    end

    btree(level,select).confir=btree(level,select).confir+1;

end
```

Main

Na função principal para além da atualização das variáveis e do recursivo acionamento de outras funções serão executados diversos testes para perceber a necessidade de criação de novos pontos. Após a confirmação de uma colisão entre a rota e os objetos, terá de se passar por uma avaliação para saber o estado da posição selecionada da matriz estrutura, indicada pelo "apontador". Verificando o nível e analisando o campo de confirmação da estrutura, pode-se saber se é efetivamente necessário definir novos pontos, ou apenas avançar ou recuar na árvore.

Inicialmente, analisa-se a variável nível (`level`), esta terá de estar compreendida entre 1 e 4, isto porque a árvore foi projetada para ser de profundidade 4. A incrementação de nível e inserção de pontos à matriz será executada até ao nível 4, chegado a esse nível a solução é decrementar

níveis até obter uma solução que possa ser novamente explorada.

Entre o nível 1 e o nível 4, terá de ser averiguado qual o valor do campo `confir` da estrutura que se encontra selecionada. Este campo toma 3 valores, como explicado no capítulo 3.5. Será executado um `switch` para verificar este campo, podendo-se dar três casos:

- Chegado ao ponto intermédio pela primeira vez (`confir=1`), o robot não conhece o resto dos elementos da árvore, nestas situações será necessária a criação de novos pontos, para tal recorre-se a uma função `newmiddle` que produz dois pontos intermédios no nível seguinte.

- No caso 2 (`confir`=2), sabe-se que apenas existe uma possibilidade, deixando o robot sem qualquer tipo de decisão, nesta opção apenas é mudado o local para onde aponta o "apontador" na matriz, enviando o robot diretamente para a coordenada previamente calculada.

- Se se confirmar o caso 3 o caminho é impossível, portanto nesta opção o robot terá de refazer o caminho, realizando a sucessiva decrementação de níveis até achar uma solução que possa ser novamente explorada.

Finalmente, após a análise à estrutura e os novos pontos, será consagrado o novo objetivo do robot, igualando as coordenadas do "apontador" da matriz estrutura à variável `newpos`.

Pseudocódigo

```
if(collision)

  if and(level>0,level<4)
     switch btree(level,select).confir
       case 1
          newmiddle();
       case 2
          select=select*2;
          level=level+1;
          btree(level,select).confir=btree(level,select).confir+1;
       case 3
          remove_tree();
       end
  else
     btree(level,select).confir=3;
     remove_tree();
  end

  newpos=btree(level,select).pos;
end
```

newmiddle

Esta foi a função criada com o intuito de descobrir novos pontos intermédios. A função recebe a variável `collision` respeitante ao objeto em colisão, e determina a forma de o transpor.

Começa-se por definir os limites do objeto em colisão, restringindo uma imagem ao tamanho do objeto dilatado. Analisando o diagrama de Voronoi da nova imagem criada, utilizando mais uma vez máscaras, ficamos a saber todos os vértices que podem ser encontrados no objeto. Contudo nem todos os vértices detetados serão um bom partido para o robot, devido ao facto da rota entre eles ser impossível. Posto isto, é imprescindível analisar quais os vértices são possíveis de executar (vértices visíveis representados na fig. 4.8 a pontilhado). O processo de seleção é realizado usando mais uma vez a função `configuration_lane`, envia-se a coordenada do robot e cada um dos vértices detetados do objeto, de modo a confirmar a existência de rota entre os dois pontos. Neste processo, ignora-se o vértice onde o robot se encontra, dado que muito provavelmente este poderá fazer parte do objeto.

Sempre que a análise aos vértices dê uma matriz linha, ou seja, existe apenas um vértice possível de ser executado, esse será o ponto a considerar como próximo destino, neste caso apenas

é introduzida na árvore binária uma coordenada.

Quando existe mais do que um ponto, será necessário definir quais os vértices mais à esquerda, mais à direita, mais em cima e mais em baixo no mapa. Após se ter selecionado os quatro pontos importantes, terá de se proceder à escolha de quais os pontos a inserir na matriz estrutura. Terá de ser analisadas as distâncias em x e em y do robot até ao ponto final, estas serão as responsáveis pela decisão do robot. Sempre que a distância em x é maior que em y os pontos a escolher serão superior e o inferior do objeto (fig. 4.8), ignorando os pontos da esquerda e da direita. Os pontos superiores e inferiores serão pontos intermédios do robot, têm de ser inseridos na árvore binária através da `ins_tree`. Todavia, quando a distância em y é maior que x é escolhido o ponto mais à esquerda e mais à direita do objeto, neste caso os pontos a serem ignorados são os pontos superiores e inferiores do objeto. Sendo explorado primeiro o lado direito ou o ponto mais a cima.

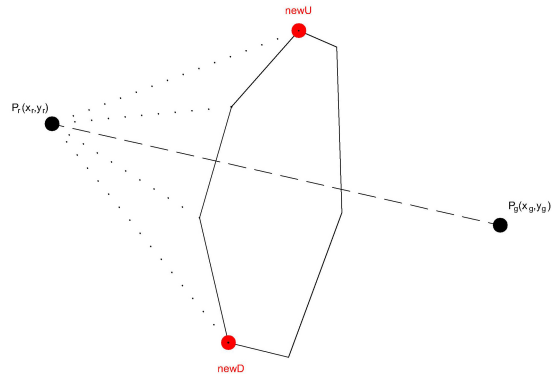


Figura 4.8: Exemplo ultrapassar objetos

Por fim, é executada a função `ins_tree` supracitada para introduzir os pontos convenientemente na matriz estrutura.

Pseudocódigo

```
function newmiddle()


global cormap goal position m b target C rob collision

xg=(-(goal(2))*100+501);
yg=((goal(1))*100+501);

xr=(-(position(2,4))*100+501);
yr=((position(1,4))*100+501);
```

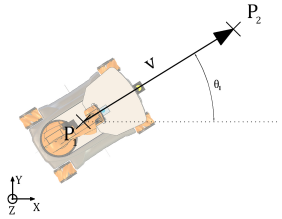
```
        limitar o mapa ao objeto;


    endImg= bwmorph(object, 'endpoints'); %%%%find corner
        C= todos os vértices;


    for i=1:length(C) %%%%find visible edge
      point=[(C(i,2)-501)/100 (501-C(i,1))/100 ];

    %% quais os vertices visiveis
        configuration_lane(rob, point);

    %% testar se é o ponto onde o robot está
      distx=point(1)-rob(1);
      disty=point(2)-rob(2);

      if(and(and(-0.2<distx,distx<0.2),and(-0.2<disty,disty<0.2)))
        collision=1;
      end

        %%%% It is visible
        if collision==0
            cor(j,:)=C(i,:); %%%% add to cor (corners)
            j=j+1;
        end
    end

    tam=size(cor);

if (tam(1)==1)
    newpos=[(cor(2)-501)/100 (501-cor(1))/100 0];
    ins_tree(newpos,[])

else

    [a,maxloc]=max(cor);
    [a,minloc]=min(cor);
    newD=cor(maxloc(1),:); %%%% point above the object
    newU=cor(minloc(1),:); %%%% point below the object
```

```
    newR=cor(maxloc(2),:); %%%% the left point of the object
    newL=cor(minloc(2),:); %%%% the right point of the object


    if (abs(xg-xr)>abs(yg-yr))

         newU=[(newU(2)-501)/100 (501-newU(1))/100 0];
         newD=[(newD(2)-501)/100 (501-newD(1))/100 0];

             ins_tree(newU,newD);
     else
%          %% Varrimento Lateral
         newR=[(newR(2)-501)/100 (501-newR(1))/100 0];
         newL=[(newL(2)-501)/100 (501-newL(1))/100 0];

             ins_tree(newR,newL);

    end

end
```

Orientação e Movimento do robot

Por último, ficou a orientação do robot. Após a realização de todos os processos para saber qual o novo ponto alvo, executa-se um conjunto de métodos para definir a nova orientação.

A orientação do robot é dada pelo sensor de orientação, que através da matriz homogénea ${}^0P_{obj}$ obtemos o ângulo ϕ, descrito em torno do eixo Z. A orientação do robot tenderá sempre para um ângulo calculado com a localização do robot (`rob`) e o ponto objetivo (`newpos`). Com estes dois pontos pode-se definir um ângulo θ_1 (`theta1`), que será:



$$\theta_1 = \arctan 2\left(\frac{\texttt{newpos}_y - \texttt{rob}_y}{\texttt{newpos}_x - \texttt{rob}_x}\right) + \frac{\pi}{2}$$

Figura 4.9: Localização de θ_1

O ângulo calculado é composto por dois parâmetros: o primeiro dado obtido pela diferença das distâncias entre o robot e o ponto alvo e o segundo parâmetro deriva das coordenadas em

repouso do robot, que por defeito apresenta uma desfasagem de 90º em relação ao sistema de eixos de referência.

Assim que calculado o ângulo é imediatamente imposto ao robot, sem que o este se dirija para as novas coordenadas. O robot sendo omnidirecional rodará sobre si mesmo para tomar a orientação pretendida. Com as várias execuções do programa, a orientação do robot é atualizada aproximando-se cada vez mais do valor de referência. O momento em que o valor em módulo da diferença entre o ângulo do robot e o de referência seja inferior a 0.05 radianos, e não sendo confirmada qualquer colisão, o código envia as coordenadas da variável `newpos` para o VREP, definindo o ponto `middle_point` com essas coordenadas. Garantindo assim que o robot só sai de um ponto depois de estar devidamente orientado e confirmadas as colisões com os objetos.

Durante a deslocação é necessária a constante confirmação do ângulo, verificando se o robot está apontado ao alvo. Nesta situação, pode ser detetada uma colisão, que era invisível quando o robot saiu do último ponto, devido à limitação do sensor. Nestes casos, é imediatamente estipulado o novo ponto alvo, sendo que a orientação do robot irá ser otimizada enquanto o robot se desloca, pois existirá sempre uma distância segura para que o robot consiga rodar e deslocar-se ao mesmo tempo, sem colisões.

Pseudocódigo

```
%% Rectify orientation robot

dnex=newpos(1)-rob(1);
dney=newpos(2)-rob(2);

tetalido=-atan2(position(1,1),position(2,1))+pi/2;
if and(abs(dnex)<0.05,abs(dney)<0.05)
    teta1=tetalido;
else
    teta1=atan2(dney,dnex)+(pi()/2);
end


Middle_point no VREP, toma o ângulo calculado
ori=abs(tetalido-teta1);


    if or(abs(ori)<0.05,(or(abs(vel(1))>0.001,abs(vel(2))>0.001)))
    middle_point passa a ter as coordenadas do newpos;
end
```

5

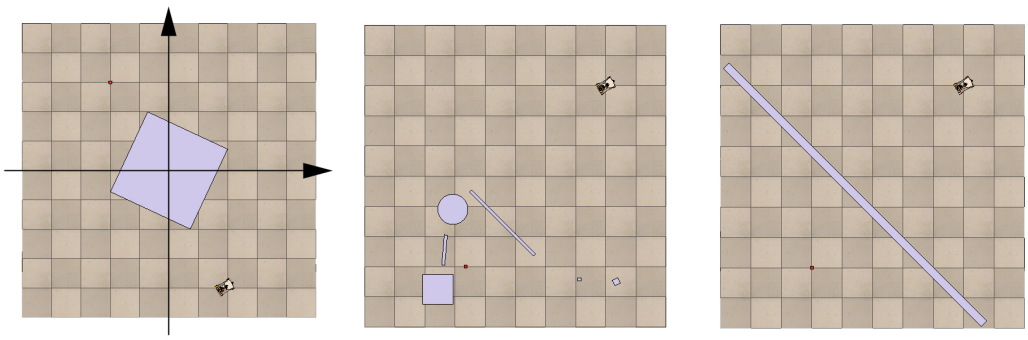
Resultados

Este capítulo visa analisar vários testes para comprovar o correto funcionamento do sistema, para tal serão realizados três testes cuja diferença entre eles é a mudança no cenário e do ponto de destino, provando o funcionamento de várias possibilidades de configuração do ambiente.

Primeiramente, será apreciado um teste simples apenas com um objeto. De seguida testar-se-á um caso mais complexo e por último um caso impossível.

5.1 Ambientes escolhidos

Os ambientes escolhidos foram projetados de forma aleatória com vários tipos de polímeros em diversas disposições, todavia todos os objetos presentes no ambiente são estáticos e para poderem ser detetados por parte do sensor de distância terão de ter uma altura maior que 0.78, caso contrário o espectro do laser passará por cima dos objetos tornando a sua localização impossível. Em baixo nas figuras 5.1, são apresentados todas as configurações dos vários sistemas escolhidos para representar os diversos casos. Como referido em capítulos anteriores, nos sistemas foi tomado como referencial base do GPS o centro do plano, cujas dimensões são 10x10, como demonstrado na figura 5.1a. Sendo que cada azulejo do chão do ambiente apresenta as dimensões 1x1.



(a) Ambiente de teste simples (b) Ambiente teste complexo (c) Ambiente teste impossível

Figura 5.1: Configuração dos ambientes

5.2 Análise Pormenorizada

Esta secção do projeto, conta aclarar o processo de criação dos pontos alvos criados para ultrapassar os objetos. Para tal, usar-se-á um mapa muito simples, conforme a fig.5.1a e será disponibilizado passo a passo a informação obtida através do código.

Conforme citado, o mapa da fig. 5.1a foi tomado para este teste, sendo o ponto inicial as coordenadas do robot na figura e o ponto final as coordenadas (-2,3), ponto vermelho na imagem. Ao observar a imagem, constata-se que entre os dois pontos existe um objeto que estará em colisão com o robot. Assim, a função do robot será ultrapassar esse objeto.

O robot deslocar-se-á resolvendo sempre uma linha reta entre si e o ponto destino `middle_point`. Este ponto vai tomar diversos valores, no VREP, e será sempre perseguido pelo robot. Como tal, o `middle_point` inicialmente começa com as coordenadas do robot, de forma a que este não saia do seu lugar sem que seja tomada uma decisão por parte do utilizador. O `middle_point` tem uma correspondência no MATLAB sendo ela o `newpos`.

Ignorando a fase do código responsável pela atualização dos dados e após ser introduzido as coordenadas objetivo, ter-se-á de proceder primeiramente a um teste de modo a verificar a existência de um caminho entre o robot e o `newpos`, o ponto a atingir. Neste primeiro ciclo os pontos estão no mesmo local, por isso confirma-se o caminho, este processo é irrelevante neste primeiro ciclo, porém será pertinente quando o robot se encontrar a deslocar.
De seguida, testa-se a existência de uma rota até ao ponto final, inicialmente é confirmado uma rota válida, uma vez que o robot está de costas para o objeto. É dado ordem para o robot se orientar para a coordenada final, após o robot estar apontado ao ponto indicado, é usado o mapa obtido por parte do sensor e confirma-se a existência de rota do centro de massa do robot ao ponto final e igualmente as laterais do robot como explicado no capítulo anterior 4.2.4, a confir-

mação é executada com a função `configuration_lane`.

Observa-se através da fig. 5.2 que no traçado da reta entre os dois pontos o trajeto é impossível, visto que na execução da confirmação é detetada uma colisão, sendo parado o ciclo não se completando a reta de um ponto ao outro, nem sendo testado as restantes retas.

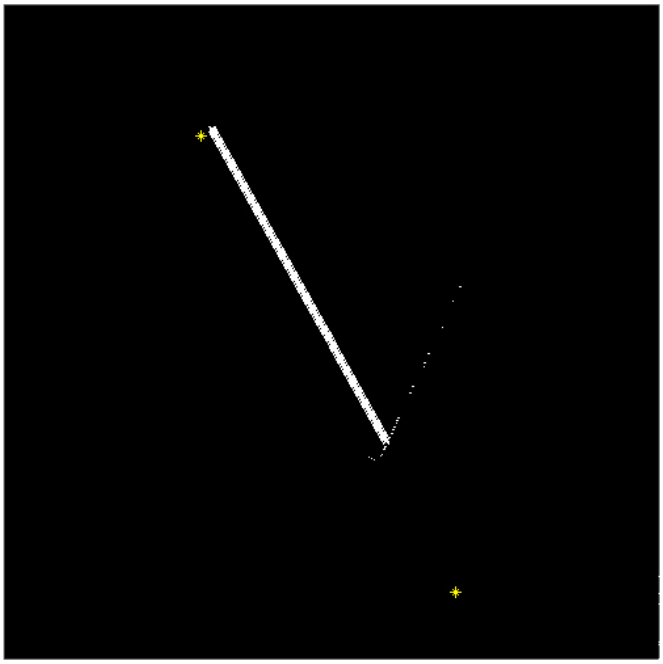


Figura 5.2: Teste ao ponto final

Após ser detetada a colisão, é impraticável o robot tomar uma rota direta ao ponto final, sendo necessário calcular um ponto intermédio para ultrapassar o obstáculo. Para tal, confirma-se a matriz binária, estando no primeiro nível e a posição `confir` preenchida com o número 1, indicando que não conhece os ramos dos níveis seguintes da árvore, é preciso recorrer a uma função para cálculo dos novos pontos (função `newmiddle`). Esta função usa o diagrama de Voronoi atualizado do ambiente, por forma a descobrir todos os vértices do objeto, representados a verde na figura 5.3.

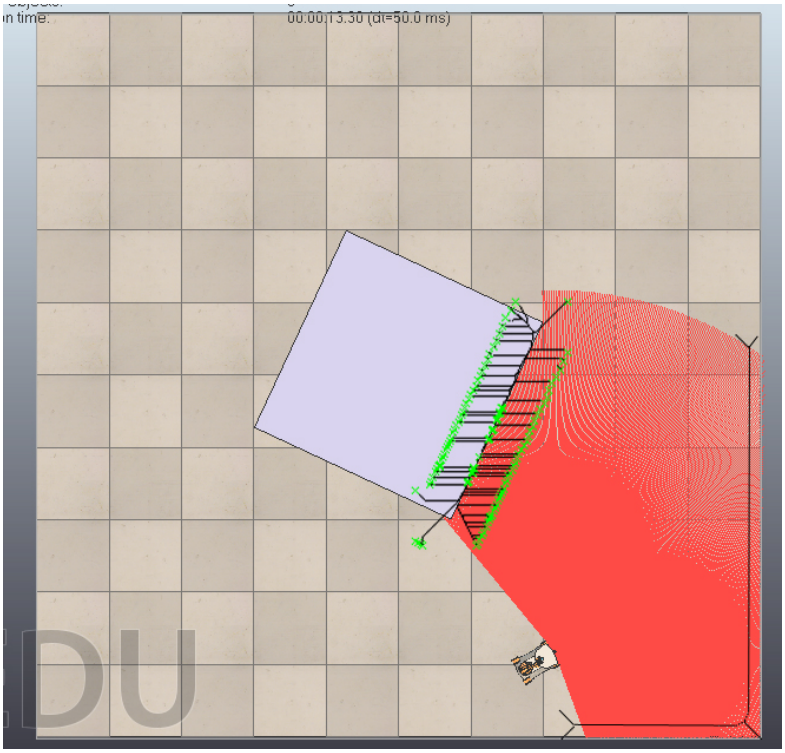


Figura 5.3: Analise Voronoi

Constata-se através da observação da imagem 5.3 que são assumidos demasiados pontos como vértices do objeto, este facto sucede-se devido à pouca informação obtida inicialmente. Como o objeto, nesta fase, é constituído apenas por alguns pontos de medição obtidos pelo sensor, quando executado a dilatação e posteriormente realizado a função de obtenção do diagrama de Voronoi vão ser assumidos demasiados pontos como vértices do objeto. Todavia, à medida que o robot se movimenta no mapa a definição dos objetos irá aumentando e portanto número de pontos assumidos como vértices vai diminuir.

Descoberto todos os vértices do objeto, terá de se determinar quais deles são visíveis pelo robot. Para tal, executa-se mais uma vez a `configuration_lane`. Esta função desenvolve três segmentos de retas se o vértice for visível(figura 5.4), caso algum dos segmentos não seja completado até ao vértice selecionado esse não exequível. Com o teste aos diferentes pontos obtém uma matriz composta por todos os vértices possíveis.

Figura 5.4: Exemplo de vértice visível

A seleção do destino intermédio fica a cargo de um conjunto de decisões já enunciadas, quando descrito a função `configuration_lane`. Recorre-se à distância entre o robot e o ponto final para fazer a seleção dos pontos intermédios. Sendo que neste caso a distância em x maior que em y, escolhe-se os vértices superior e inferior para introduzir na árvore binária e o robot dirigir-se-á primeiro para o ponto mais superior.

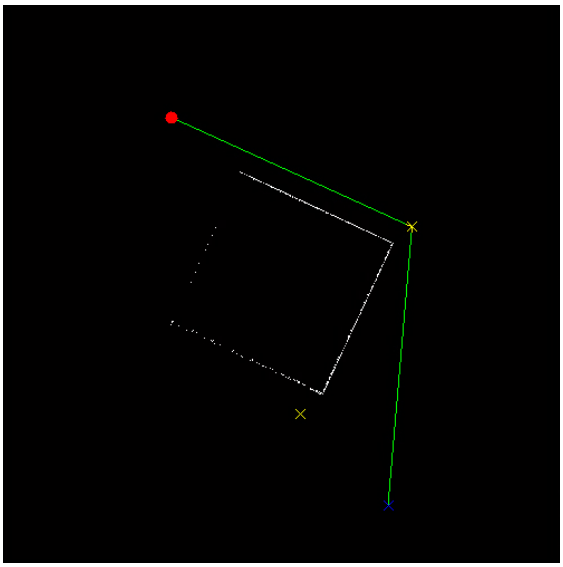
Com o novo ponto destino determinado, há que reorientar o robot, de forma a que este fique apontado ao novo ponto, é feito o cálculo do ângulo a tomar e enviado ao robot. Após alguns ciclos de scan, já com o robot devidamente orientado são verificadas novamente as colisões. Não sendo detetada qualquer colisão, é definido o `middle_point` no VREP na nova posição calculada e selecionada, tomando o valor de `newpos`, variável do MATLAB. O robot dirige-se, então, para as novas coordenadas verificando sempre a colisão com um eventual objeto que poderá não ter sido detetado.

Chegado às coordenadas intermédias, o robot testa o caminho até ao ponto objetivo. No caso em estudo, existe um caminho possível, sendo por isso o próximo ponto alvo do robot. Porém, antes de qualquer deslocação, primeiro tem que se reorientar o robot e testar uma última vez a inexistência de colisões.

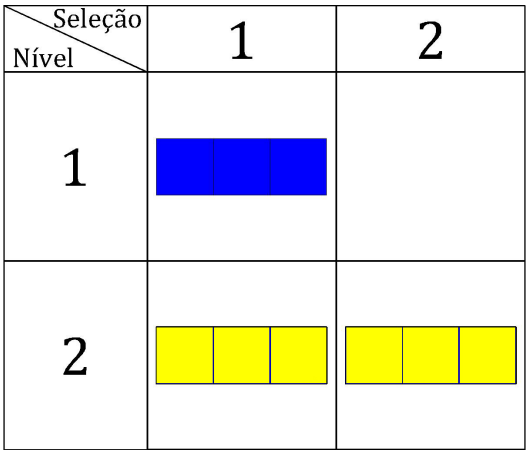
Contudo, se não se tivesse verificado uma rota até ao ponto final, teria de ser calculado dois novos pontos e estes serem introduzidos no nível 3 da árvore.

Neste processo, obtém-se uma árvore binária que terá no máximo dois níveis. Na figura abaixo 5.5a, está representados a solução que o robot descreveu para atingir o ponto final, pode ainda observar-se na figura 5.5b a matriz representativa da variável `btree` responsável por guardar a

árvore binária. Cada célula da matriz representa uma estrutura, que tem o campo confirmação e o campo de coordenadas como explicado em capítulos anteriores (na figura 5.5b apenas está representado o campo da estrutura referente às coordenadas). Esta imagem será informativa e bastante útil nos testes seguintes, afim de verificar as posições na matriz preenchidas com coordenadas.



(a) Mapa e solução da trajetoria



(b) Matriz binária `btree` da solução simples

Figura 5.5: Solução do teste simples

5.3 Análise testes complexos

Os testes aqui apresentados serão resumidamente descritos, apresentando imagens do ambiente e caminho escolhido ou realizado. Esta secção visa verificar a exatidão do código, bem como de todo o projeto. Deste modo exige-se uma maior complexidade em termos de mapa verificando diversos testes de forma a confirmar o correto funcionamento do programa.

Teste Complexo

Esta verificação é realizada utilizando uma mistura de objetos, prismas quadrangulares e cilindros. Inicialmente, após a devida orientação para o ponto final, o robot não tem alcance suficiente para detetar qualquer um dos obstáculos. Assim, irá dirigir-se para o último ponto a atingir.

Conforme o robot se vai deslocando, descobrindo assim o mapa, é confirmado uma colisão entre a rota e os objetos. O robot, nesta situação, calcula dois novos pontos recorrendo ao mapa

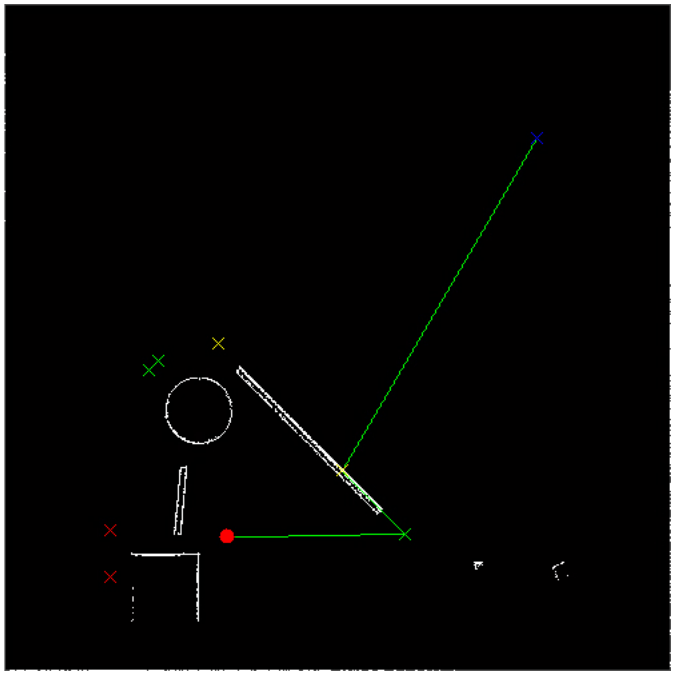
que dispõe, optando pelo ponto mais à sua direita (nível 2 na árvore binária, cruz a amarelo no mapa na figura 5.6a). Chegado a este ponto, terá de ser confirmada a existência de um caminho para o ponto final.

Neste caso, o robot não consegue definir um caminho até ao ponto alvo, por isso tem de definir novos pontos no mapa, inserindo-os na matriz estrutura, para tentar ultrapassar o obstáculo. Este processo será executado até atingirmos o nível 4 na árvore binária. Desta forma testam-se os pontos mais à direita do robot. Pode ser observado na árvore binária, figura 5.6b, que as posições (4,3) e (4,4) estão vazias. Estes pontos apesar de terem sido calculados não foram introduzidos na matriz, uma vez que já se encontravam presentes na matriz estrutura, assim sendo estas posições ficaram vazias.

Estando inteiramente calculado todos os pontos do lado esquerdo da matriz, ou seja, todos os pontos descendentes da posição (2,1). Todos estes pontos estão preenchidos com número 3 no campo `confir` em todas as células, representativo de caminho impossível. Conclui-se que todas as soluções deste lado da matriz são impossíveis. O robot retorna então à posição inicial para explorar o ponto definido na posição (2,2) da árvore binária.

O ponto que o robot pretende explorar é um ponto que foi calculado juntamente com o (2,1). Nessa fase do programa, quando se calculou os dois pontos intermédios, o conhecimento que o robot tinha do objeto era incompleto, devido ao alcance do sensor. Com a deslocação do robot foi realizada a atualização do mapa e por isso o robot concluiu que o ponto definido anteriormente é impossível, uma vez que está colocado sobre o objeto. Uma vez que o ponto é impossível de atingir, será ignorado pelo robot (figura5.6b posição (2,2) da matriz) e calculado dois novos pontos que são adicionados nas posições seguintes, (3,3) e (3,4). Ao examinar a matriz final obtida constata-se que a posição (3,3) está vazia. Igualmente ao que aconteceu quando se tentou adicionar as posições (4,3) e (4,4), também o ponto calculado para a posição (3,3) já existe na matriz, não sendo inserida qualquer coordenada nesta posição.

Por fim, o robot dirige-se para o ponto calculado. A partir deste ponto é possível a realização de um troço para as coordenadas finais concluindo assim o programa.



(a) Trajetória definida pelo robot

Seleção / Nível	1	2	3	4
1				
2				
3				
4				

(b) Matriz binária da solução complexa

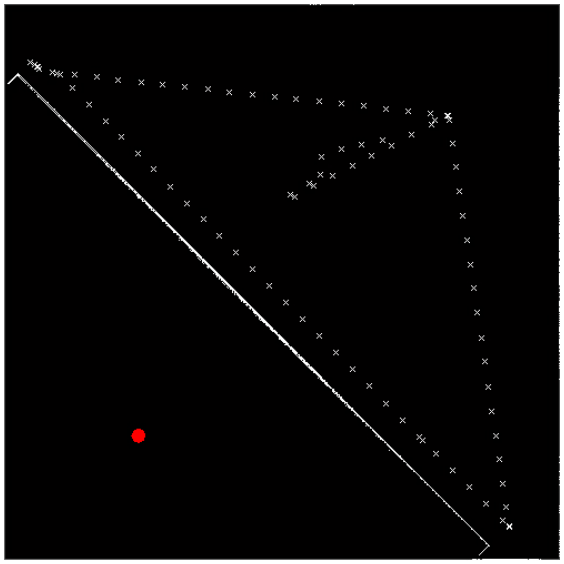
Figura 5.6: Solução do teste complexo

Teste Impossível

Inicialmente, o robot começa por se dirigir para o ponto final, contudo apercebe-se que essa trajetória é impossível, então calcula dois novos pontos com a `new_middle` recorrendo ao mapa que dispõe, chegando depois à conclusão que também esses pontos são impossíveis, sendo necessário calcular novamente novos pontos intermédios. O robot andará a deslocar-se no mapa um bocado perdido a calcular pontos impossíveis, contudo vai descobrindo cada vez mais o mapa. Chegando ao ponto em que se preenche a parte direita da matriz e tem de se retornar ao ponto inicial.

Nesta fase, já com mais dados do mapa é calculado um ponto intermédio, originado no canto superior esquerdo. Verifica o ponto não sendo confirmado qualquer rota até ao ponto final, segue então para um novo ponto agora ao canto inferior direito. O robot acaba por completar a matriz binária, confirmando que todas as soluções escolhidas são impossíveis.

Por fim, retorna ao ponto inicial e apresenta uma mensagem no MATLAB, “SOLUÇÃO IMPOSSÍVEL”, aguardando o fim da conexão.



Seleção / Nível	1	2	3	4	5	6	7	8
1	■							
2	■	■						
3	■		■	■				
4	■	■			■	■	■	

(a) Mapa e trajetória do teste impossível

(b) Matriz binária da solução impossível

Figura 5.7: Solução do teste impossível

6

Conclusão e Futuros Trabalhos

Este trabalho é o culminar de um estudo minucioso que exigiu uma análise e uma reflexão profunda sobre a matéria.

Sumariamente, serão expostas as principais conclusões obtidas e averiguadas eventuais limitações do projeto. Será apresentada, por fim, uma perspetiva de futuros trabalhos com base neste projeto.

6.1 Conclusão

O desenvolvimento de um projeto de condução autónoma é algo complexo e exige forte conhecimento de diferentes áreas. Este carece do estabelecimento de etapas, sendo constituído por quatro módulos fundamentais: obtenção de dados; tratamento dos sinais; deteção de obstáculos; sistema de controlo.

O robot omnidirecional usado, o *Youbot*, é um robot bastante versátil e ágil, desempenhando as suas funções de forma exímia. A possibilidade do robot rodar sobre si mesmo e deslocar-se para todos os sentidos foi uma opção excelente para desempenhar as funções pretendidas.

A obtenção dos dados por parte de um sensor de distância, memorizando as posições dos objetos numa matriz é uma forma eficaz para realizar o processo, contudo, devido à limitação do sensor, os dados disponibilizados são sempre insuficientes para executar uma otimização do método de controlo, sendo impossível determinar qual o melhor caminho a tomar pelo robot.
A limitação do sensor, por vezes, torna a deteção dos obstáculos incompleta, o que origina a criação de novos pontos intermédios definidos sobre os objetos. Todavia, a colisão é detetada enquanto é realizada a deslocação, pois é atualizado o mapa. Nestes casos o robot tem de calcular novas posições ignorando a posição intermédia impossível, evitando a colisão que existia.

A forma de guardar os pontos intermédios sob forma de árvore binária recorrendo a uma formação matricial construída por estruturas, é muito eficaz na seleção e memorização dos pontos intermédios. Esta estratégia usada para a navegação é bastante útil caso se escolha uma hipótese impossível. Pois detetada a impossibilidade de uma opção, poderá sempre ser realizado o retrocesso na matriz, de forma a que o robot refaça o caminho para avaliar outras possibilidades. Contudo a profundidade máxima da árvore poderá, por vezes, incapacitar o robot de chegar às coordenadas finais. Se forem inseridos muitos objetos, ou objetos cilíndricos de grandes dimensões, pode suceder que sejam preenchidas todas as posições da árvore sem que seja completada a missão do robot, porém o caminho poderá ser realizado.

No geral, foi um estudo realmente interessante e definitivamente instrutivo, elevando o grau de conhecimento da matéria estudada. Conforme apresentado, verifica-se nos vários testes disponibilizados que o programa comporta-se de forma correta, devolvendo respostas e trajetórias de acordo com o pretendido. É de referir que além dos testes realizados, foram executados outros ensaios de modo a verificar outras disposições, todos os objetivos propostos inicialmente foram cumpridos, à exceção da execução dos testes realizados com cilindros, neste tipo de mapas os vértices dos objetos são inexistentes, o que torna difícil passar os objetos, já que o cálculo dos pontos é realizado usando os vértices dos obstáculos. Nestes casos o robot, define muitos pontos para ultrapassar os cilindros ocupando muitas células da matriz estrutura, podendo dar-se o caso de preencher a matriz, indicando caso impossível.

6.2 Trabalhos Futuros

Os resultados obtidos neste projeto foram satisfatórios, apesar disso há desenvolvimentos que poderão ser realizados para melhorar o desempenho do conjunto. Reutilizando os recursos e o código inerente neste trabalho, poderão ser desenvolvidos futuros trabalhos para expandir as interações entre o robot e o meio, bem como o desenvolvimento de melhorias no desempenho das funções do robot.

Uma das principais propostas passa por resolver a questão da inserção de coordenadas inválidas na matriz. Como explicado em secções anteriores, este acontecimento surge porque não se conhece a totalidade do mapa, assim o robot poderá selecionar pontos de destino intermédios que possam estar contidos no objeto. Apesar desta situação ser corrigida pelo robot, ainda assim o ponto ficará na matriz estrutura, ocupando uma posição que poderia ser preenchida por uma posição válida.

Avaliando a questão energética do projeto, uma vez que o robot dispõe de uma bateria que lhe confere uma autonomia, seria proveitoso em futuros trabalhos desenvolver um algoritmo que desliga-se o sensor de distância em deslocamento sempre que o ponto fica-se dentro da sua àrea de deteção. Sendo que todos os testes são realizados com objetos estáticos, após o robot estar

devidamente orientado e se a distância ao ponto alvo fosse inferior a 4m, o alcance máximo do sensor de distância, não há necessidade deste sensor permanecer ligado no deslocamento do robot, evitando assim gastos de energia desnecessários e aumentando a autonomia do robot comparativamente com o sensor sempre ligado.

Em termos de hardware, considerando o *Youbot* e uma vez que este está equipado por um braço robótico, é incontornável não o usar em futuros trabalhos. Este poderá interagir com objetos de forma a carregar o equipamento com objetos para poderem ser transportados. Nesta etapa teremos de obter uma imagem 3D do meio envolvente. Para tal, poderá ser usado o sensor de distância supracitado, porém terá de ser acoplado entre o robot e o sensor um motor de modo a ser possível fazer um varrimento vertical com o sensor.

Referências

[1] URL http://www.openrobots.org/wiki/morse/.

[2] URL http://opensimulator.sourceforge.net/.

[3] URL http://playerstage.sourceforge.net/index.php?src=stage.

[4] URL http://stdr-simulator-ros-pkg.github.io/.

[5] URL http://www.coppeliarobotics.com/index.html.

[6] Introduction to voronoi diagrams. URL http://dasl.mem.drexel.edu/Hing/VoronoiTutorial.htm.

[7] Simultaneous localization and mapping: Part I. IEEE Robotics & Automation Magazine, June 2006.

[8] Colin Angle. Genghis, a six legged autonomous walking robot. Massachusetts Institute of Technology, March 1989. URL http://dspace.mit.edu/bitstream/handle/1721.1/14531/20978065.pdf?sequence=1.

[9] John Bares and William (Red) L. Whittaker. Dante II. Robotic thesis, dante ii, Carnegie Mellon University, 1994. URL http://www.ri.cmu.edu/research_project_detail.html?project_id=163&menu_id=261.

[10] *GP9 GPS-Aided AHRS Datasheet*. CH Robotics, 1.3v edition, outubro 2013. URL http://www.chrobotics.com/docs/GP9Datasheet.pdf.

[11] Howie Choset and Philippe Pignon. Coverage path planning: The boustrophedon cellular decomposition. In Alexander Zelinsky, editor, *Field and Service Robotics*, pages 203–209. Springer London, 1998. URL http://dx.doi.org/10.1007/978-1-4471-1273-0_32.

[12] Axis communication. Ccd and cmos sensor technology, 2010. URL www.axis.com/products/video/about_networkvideo/image_sensors.htm.

[13] International federation of robotics. History of industrial robots, 2012. URL http://www.ifr.org/fileadmin/user_upload/downloads/forms___info/History_of_Industrial_Robots_online_brochure_by_IFR_2012.pdf.

[14] International Organization for Standardization. Robots and robotic devices - vocabulary. standard. URL https://www.iso.org/obp/ui/#iso:std:iso:8373:ed-2:v1:en.

[15] Hans J. Weber George B. Arfken. *MATHEMATICAL METHODS FOR PHYSICISTS*, chapter Chapter 3, pages 202–203. Sixth edition edition, 2005. URL http://mathworld.wolfram.com/RotationMatrix.html.

[16] Seth Hutchinson George Kantor Wolfram Burgard Lydia E. Kavraki e Sebastian Thrun Howie Choset, Kevin M. Lynch. *Principles of Robot Motion: Theory, Algorithms, and Implementations*, chapter 2, pages 17–31. massachusetts institute of technology, 2005.

[17] H. R. Everett J. Borenstein, L. Feng Contributing authors: S. W. Lee, and R. H. Byrne. *Where am I? Sensors and Methods for Mobile Robot Positioning*. the University of Michigan, 1996. URL http://www-personal.umich.edu/~johannb/Papers/pos96rep.pdf.

[18] KUKA. Youbot detailed specifications, Julho 2014. URL ftp://ftp.youbot-store.com/manuals/KUKA-youBot_UserManual.pdf.

[19] Steven M. LaValle. *Planning Algorithms*, chapter 4. Cambridge University Press, 2006.

[20] Arthur Ed LeBouthillier. W. grey walter and his turtle robots. *The Robot Builder*, 1999. URL www.rssc.org/content/w-grey-walter-and-his-turtle-robots.

[21] Locomotec. *KUKA youBot User Manual*. KUKA, version 1.02 edition, February 2013. URL http://www.youbot-store.com/wiki/index.php?title=YouBot_Detailed_Specifications&hmswSSOID=df36edd574042b776f117d573324deceafa897c8{#}Arm_Actuatorhttp://www.youbot-store.com/wiki/index.php?title=YouBot_Detailed_Specifications&hmswSSOID=df36edd574042b776f117d573324deceafa897c8{#}Arm_Actuator.

[22] Marc van Kreveld Mark de Berg, Otfried Cheong and Mark Overmars. *Computational Geometry*, chapter 7, pages 151–160. Springer, 2008.

[23] MORI. *Scanning Laser Range Finder URG-04LX-UG01 (Simple-URG) Specifications*. Hokuyo Automatic.co, Agosto 2009. URL http://www.hokuyo-aut.jp/02sensor/07scanner/download/pdf/URG-04LX_UG01_spec_en.pdf.

[24] Pedro Lima, Maria Isabel Ribeiro. Mobile robotics. Instituto Superior Técnico, 2002. URL http://users.isr.ist.utl.pt/~mir/cadeiras/robmovel/Introduction.pdf.

[25] Pedro Lima, Maria Isabel Ribeiro. Motion planning. Instituto Superior Técnico (IST), Instituto de Sistemas e Robótica (ISR), Av.Rovisco Pais, 1, 1049-001 Lisboa PORTUGAL, April 2002. URL http://users.isr.ist.utl.pt/~mir/cadeiras/robmovel/Motion.pdf.

[26] J. NORBERTO PIRES. Robôs manipuladores industriais. *Publico*, 2002. URL http://www.publico.pt/noticias/jornal/robos-manipuladores-industriais-172517.

[27] RedOrbit. Ballbot. URL http://www.redorbit.com/education/reference_library/technology_1/robotics-technology_1/1112956375/ballbot/.

[28] rezero. URL http://www.rezero.ethz.ch/.

[29] Maria Isabel Ribeiro. Sensores em robótica. URL http://users.isr.ist.utl.pt/~mir/pub/sensores.pdf.

[30] Maria Isabel Ribeiro. Uma viagem ao mundo dos robots. Ciclo de Colóquios Despertar para a Ciência 2004, Instituto Superior Técnico, 2004. URL http://users.isr.ist.utl.pt/~mir/pub/ViagemRobots-IsabelRibeiro05.pdf.

[31] Eric Roberts. *CS 106B. Programming Abstractions (Chapter 7 - Backtracking Algorithms)*. Stanford University, 2008-2009. URL http://cs.stanford.edu/people/eroberts/courses/cs106b/chapters/07-backtracking-algorithms.pdf.

[32] Robert Sedgewick. *Algorithms in C*, chapter 28, pages 407–411. 2nd edition edition, 1990.

[33] Kadir Firat Uyanik. A study on tangent bug algorithm. Technical report. URL https://www.academia.edu/2007423/A_study_on_Tangent_Bug_Algorithm.

[34] Maren Bennewitz Kai Arras Wolfram Burgard, Cyrill Stachniss. Introduction to mobile robotics. University of Freiburg, July 2011.

[35] Hasan Ýhsan Turhan. Implementation of tangent bug algorithm with a non-holonomic wheeled mobile robot pioneer 3-dx. URL http://www.eee.metu.edu.tr/~hturhan/IMPLEMENTATION%20OF%20TANGENT%20BUG%20ALGORITHM.pdf.