Universidade do Minho
Escola de Engenharia

Tiago João Fernandes Maia

Desenvolvimento de Sistema de Controlo de Movimento para Plataformas Móveis Autónomas

Desenvolvimento de Sistema de Controlo de Movimento para Plataformas Móveis Autónomas
Tiago João Fernandes Maia
UMinho | 2019

outubro de 2019

Universidade do Minho
Escola de Engenharia

Tiago João Fernandes Maia

Desenvolvimento de Sistema de Controlo de Movimento para Plataformas Móveis Autónomas

Dissertação de Mestrado
Ciclo de Estudos Integrados Conducentes ao Grau de Mestre em Engenharia Eletrónica Industrial e Computadores

Trabalho realizado sob a orientação do
Professor Doutor António Fernando Macedo Ribeiro

outubro de 2019

DIREITOS DE AUTOR E CONDIÇÕES DE UTILIZAÇÃO DO TRABALHO POR TERCEIROS

AGRADECIMENTOS

Esta secção é dedicada a todas as pessoas que de uma forma, direta ou indireta, contribuíram e apoiaram-me ao longo do curso, que termina com esta dissertação. A todos deixo aqui os meus sinceros agradecimentos.

Em primeiro lugar quero agradecer ao meu orientador Professor Dr. Fernando Ribeiro e ao Professor Dr. Gil Lopes, por me terem dado a oportunidade de fazer parte do LAR e do projeto Minho Team MSL. Oportunidade essa que me ajudou a evoluir como pessoa e profissional, dando-me a possibilidade de aprender a trabalhar em equipa e de pôr em prática alguns conhecimentos adquiridos no curso. Obrigado por todo o apoio e confiança que depositaram em mim.

Agradeço aos meus amigos e colegas que encontrei no LAR, pela partilha de conhecimento e pelos dias e noites que passamos a trabalhar para alcançar o objetivo mais desejado, fazer um jogo completo na MSL.

Às minhas três irmãs, agradeço muito por toda a ajuda e incentivo quando mais necessitei.

Por último, e mais importante, um agradecimento muito especial ao meu pai, João Maia e à minha mãe, Maria da Conceição pela oportunidade que me deram para continuar a estudar, sem vocês nada disto era possível. Obrigado por tudo.

A todos, muito obrigado.

DECLARAÇÃO DE INTEGRIDADE

Declaro ter atuado com integridade na elaboração do presente trabalho académico e confirmo que não recorri à prática de plágio nem a qualquer forma de utilização indevida ou falsificação de informações ou resultados em nenhuma das etapas conducente à sua elaboração.

Mais declaro que conheço e que respeitei o Código de Conduta Ética da Universidade do Minho.

DESENVOLVIMENTO DE SISTEMA DE CONTROLO DE MOVIMENTO PARA PLATAFORMAS MÓVEIS AUTÓNOMAS

RESUMO

Nas últimas décadas tem-se verificado um crescimento acentuado em projetos na área da robótica móvel e autónoma, nomeadamente para acompanhamento de pessoas idosas ou com deficiências, ou mesmo para tarefas domésticas. Este crescimento deve-se em boa medida à evolução de áreas como a eletrónica e a informática que permitiram uma redução dos custos de desenvolvimento e um considerável aumento na velocidade de processamento. Isso tem permitido um desenvolvimento considerável em algoritmos de controlo, inteligência artificial e visão por computador.

Esta dissertação tem como objetivo a implementação e validação de uma arquitetura de controlo de movimento, baseada no planeamento da trajetória, com aplicação prática na equipa de futebol robótico do Laboratório de Automação e Robótica (LAR) da Universidade do Minho, denominada Minho Team. A equipa é constituída por 5 robôs futebolistas de tamanho médio (até 80 cm de altura e 40 kg de peso) que foram desenvolvidos com todos os requisitos para participar no RoboCup, mais propriamente numa liga denominada de Middle Size League (MSL). Estes robôs são totalmente autónomos e operam num ambiente extremamente dinâmico, constituído por obstáculos dinâmicos que se deslocam a velocidades elevadas (por vezes atingindo os 4 m/s).

Cada um dos robôs da Minho Team faz o planeamento da sua trajetória entre a posição atual e uma posição pretendida. A trajetória é planeada de forma a obter o melhor compromisso entre a trajetória mais curta e a mais suave, evitando colisões com obstáculos em campo. Para isso, foram implementados métodos associados a um planeamento global e adicionados outros algoritmos a estes métodos, formando um modelo de planeamento da trajetória eficiente e robusto. Para transformar a representação contínua do espaço de jogo num mapa discreto foi utilizado o método do diagrama de Voronoi, sendo a pesquisa pela trajetória mais curta realizada pelo algoritmo de Dijkstra. Para construir uma trajetória suave foi utilizado o método de curvas paramétricas polinomiais, e para determinar o melhor movimento ao longo de uma trajetória foi implementado o controlador PID em malha fechada. O principal objetivo desta solução é permitir que os robôs em campo consigam jogar futebol com a máxima velocidade possível, obedecendo às regras de jogo, nomeadamente, não colidir com os robôs adversários nem robôs cooperativos (robôs da Minho Team).

Palavras-Chave: Controlo, MSL, Planeamento da Trajetória, Robótica.

DEVELOPMENT OF MOTION CONTROL SYSTEM FOR AUTONOMOUS MOBILE PLATFORMS

ABSTRACT

In the recent decades, there has been a pronounced growth in projects in the area of autonomous mobile robotics, particularly for assisting elderly or incapacitated people as well as helping with domestic tasks. This growth is due to the evolution of areas such as electronics and advanced computing, that allowed a reduction of development costs and a considerable increase in processing speed. This has allowed considerable development in control algorithms, artificial intelligence and computer vision.

This dissertation aims at the implementation and validation of a motion control architecture, based on trajectory planning, with practical application in the robotic soccer team of the Laboratory of Automation and Robotics at the University of Minho, called Minho Team. The team consists of 5 medium-sized soccer robots (up to 80 cm in height and 40 kg in weight) which were developed with all the requirements to participate in RoboCup, more specifically in a league called Middle Size League (MSL). These robots are fully autonomous and operate in an extremely dynamic environment, constituted by dynamic obstacles that move at high speeds (sometimes up to 4 m/s).

Each robot of the Minho Team makes planning for its trajectory between the current position and the desired position. The trajectory is planned in such a way as to obtain the best balance between the shortest and the smoothest trajectories in a way to also avoid collisions with obstacles in the field. For this matter, methods associated with global planning were implemented and other algorithms were added to these methods forming an efficient and robust trajectory planning model. The Voronoi diagram's method was used to transform the continuous representation of the game space into a discrete map being the search for the shortest trajectory performed by the Dijkstra algorithm. To build a smooth trajectory, the parametric polynomial curves method was used. And to determine the best movement along a trajectory, the closed-loop PID controller was implemented. The main objective of this solution is to allow robots to play soccer with the possible maximum speed, obeying the rules of the game, namely, not to collide with the enemy robots or the cooperative robots (Minho Team robots).

Keywords: Control, MSL, Path Planning, Robotics.

ÍNDICE

LISTA DE FIGURAS

LISTA DE ACRÓNIMOS

ISO	*International Organization for Standardization*
FIFA	*Fédération Internationale de Football Association*
2D	Duas Dimensões
3D	Três Dimensões
MSL	*Middle Size League*
LAR	Laboratório de Automação e Robótica
FNR	Festival Nacional de Robótica
A*	*A Star*
API	*Application Programming Interface*
CGAL	*Computational Geometry Algorithms Library*
EC	Espaço de Configuração
DOF	*Degrees Of Freedom*
ROS	*Robot Operating System*
GPS	*Global Positioning System*
CAD	*Computer-Aided Design*
IDE	*Integrated Development Environment*
PID	*Proportional Integral Derivative*
ADC	*Analog-to-Digital Converter*
IMU	*Inertial Measurement Unit*

Capítulo 1

INTRODUÇÃO

Esta dissertação surge no âmbito do Mestrado Integrado em Engenharia Eletrónica Industrial e Computadores da Universidade do Minho, com o objetivo de documentar o trabalho desenvolvido durante o projeto de dissertação denominado *Desenvolvimento de Sistema de Controlo de Movimento para Plataformas Móveis Autónomas*.

Este capítulo inicia com o enquadramento deste projeto na robótica em geral e a motivação para a realização do mesmo. Posteriormente, é apresentado o problema que se pretende resolver e também são definidos os objetivos para esta dissertação.

1.1. ENQUADRAMENTO E MOTIVAÇÃO

A robótica surgiu com a intenção de oferecer sistemas que permitam auxiliar os humanos em algumas tarefas e substituí-los noutras com maior velocidade, precisão e segurança [1]. Estes sistemas, constituídos por plataformas mecânicas, *hardware* e *software*, são desenvolvidos com a aplicação de conceitos de áreas como a matemática, física, química, biologia e de áreas que estudam o corpo humano, tal como, a neurologia [1].

A evolução tecnológica verificada nas últimas décadas, permitiu o desenvolvimento de robôs móveis que se deslocam autonomamente em diversos ambientes. Contudo não existe uma solução otimizada que consiga abranger a enorme variedade de aplicações que existe atualmente para robôs móveis. Por esse facto, a robótica móvel continua em constante evolução. Para além da importância da evolução tecnológica, o crescimento da robótica deve-se ao esforço da comunidade científica no desenvolvimento em algoritmos de controlo, inteligência artificial e visão por computador. A presença em competições robóticas, como exemplo o RoboCup, tem um peso muito importante para este desenvolvimento, sendo uma forma dos investigadores apresentarem e partilharem ideias.

Esta crescente evolução tem contribuído de forma significativa para o desenvolvimento de plataformas, que posteriormente resultam em novos produtos para o mercado. Esses produtos estão focados em dois tipos de robôs, os industriais e os de serviços. Os robôs de serviço, tipicamente móveis, são utilizados no auxílio a tarefas realizadas por humanos, sendo capazes de navegar e de se adaptarem

aos ambientes onde atuam. Os robôs assistentes em hospitais como o HOSPI da Panasonic [2] (ver *Figura 1.1*) são um bom exemplo de robôs de serviço. Estes realizam diferentes tarefas, desde a entrega de medicamentos, amostras médicas para análise e anotações médicas de casos de pacientes, proporcionando assim aos profissionais responsáveis pelos cuidados médicos mais tempo na assistência aos pacientes. O HOSPI, através do conhecimento prévio do mapa do hospital, tem a capacidade de escolher a melhor trajetória para se deslocar para um determinado local e utilizar sensores para se desviar de obstáculos, caso seja necessário deve calcular um novo percurso. Localização, desvio de obstáculos e planeamento da trajetória são tarefas indispensáveis para o funcionamento deste tipo de robôs.

Figura 1.1 – Robô de entrega autónomo Panasonic-HOSPI [2]. Auxilia nas tarefas hospitalares.

Um robô industrial é definido pela norma ISO 8373 [3] como sendo um manipulador de três ou mais eixos, controlado automaticamente e programável, pode ser fixo ou móvel para o uso em aplicações de automação industrial. As indústrias de produção de eletrodomésticos, automóvel, alimentar, semicondutores e plásticos são algumas das áreas da aplicação destes robôs, pois o aumento de produtividade, a qualidade dos produtos e a operação em ambientes difíceis e perigosos para os humanos são algumas das vantagens que a indústria obtém com estes robôs. Alguns robôs são programados para realizarem ações repetidamente sem nenhuma variação, estas ações são determinadas por rotinas pré-programadas que especificam a posição, força e aceleração dos movimentos [4]. Outros são mais flexíveis nas linhas de produção recorrendo a câmaras, que através de algoritmos de visão por computador e inteligência artificial conseguem se adaptar a possíveis variações na produção.

1.1.1. ROBOCUP

O RoboCup [5] é um encontro científico internacional promovido pela RoboCup Federation, que tem como objetivo promover o estudo e o desenvolvimento da inteligência artificial, robótica e de áreas relacionadas. Na sua criação foi definida uma meta, o principal objetivo a longo prazo: "*No ano de 2050, uma equipa de robôs totalmente autónomos humanoides, ser capaz de vencer um jogo de futebol contra a equipa campeã do mundo de futebol humano, cumprindo as regras oficiais da FIFA*" [5].

A primeira edição decorreu em 1997 em Nagoya no Japão, sendo realizado anualmente em várias cidades por todo o mundo, como em Lisboa, em 2004 [6]. Cada edição é constituída por competições de diferentes categorias e um simpósio, onde é possível aos participantes apresentarem e discutirem os seus trabalhos de investigação.

Atualmente o RoboCup é constituído pelas seguintes categorias: RoboCupSoccer, RoboCupRescue, RoboCup@Home, RoboCupIndustrial e RoboCupJunior. Cada uma destas categorias é constituída pelas seguintes ligas:

- RoboCupSoccer
    - Humanoid League
        - KidSize
        - TeenSize
        - AdultSize
    - Standard Platform League
    - Middle Size League
    - Small Size League
    - Simulation League
        - Simulation 2D
        - Simulation 3D
- RoboCupRescue
    - Rescue Robot League
    - Rescue Simulation League
        - Agent Simulation and Infrastructure
        - Virtual Robot
- RoboCup@Home
    - Open Platform League
    - Domestic Standard Platform League

    - Social Standard Platform League
- RoboCupIndustrial
    - RoboCup@Work League
    - RoboCupLogistics League
- RoboCupJunior
    - Soccer
    - OnStage
    - Rescue
        - CoSpace

1.1.2. MIDDLE SIZE LEAGUE

A Middle Size League (MSL) foi uma das ligas criadas no início do RoboCup, em 1997. Nesta liga os robôs devem jogar futebol de acordo com as regras do RoboCup, que se baseiam nas regras oficiais da FIFA, com algumas alterações necessárias para um jogo de futebol com robôs [7].

Um jogo tem duas partes de 15 minutos cada e um intervalo que não deve exceder os 10 minutos. Disputado por duas equipas, cada equipa pode jogar com 5 robôs que devem ter no máximo 80 cm de altura, 52 cm × 52 cm na base e um peso máximo de 40 kg, tendo estes de apresentar cor maioritariamente preta. Cada um dos robôs tem de jogar autonomamente, contendo os seus próprios sensores. Através do *wireless* podem partilhar informação em tempo real entre os elementos da equipa e com um computador externo, denominado de *Base Station*, que atua como um treinador e não tem qualquer tipo de sensor.

À semelhança de um jogo de futebol humano, na MSL também existem regras que devem ser cumpridas durante o jogo e para isso é atribuído a humanos a tarefa de fazer valer essas regras, normalmente é nomeado um árbitro e um ou mais assistentes. Como os robôs ainda não detetam sons, as faltas ou as ações marcadas pelo árbitro são entendidas pelos robôs através da utilização de um computador com uma interface denominada de *Referee Box*. Esta interface envia comandos sobre as situações de jogo para a *Base Station* de cada uma das equipas, como o *start*, *stop*, *kick-off* e entre outras decisões do árbitro.

O campo de jogo é verde, tem no máximo 18 m × 12 m e as linhas são brancas. As balizas têm 2 m de largura e 1 m de altura.

A MSL desde a sua existência tem oferecido grandes desafios a nível científico, pois o ambiente de jogo é extremamente dinâmico e os robôs deslocam-se a velocidades elevadas, por vezes atingindo os 4 m/s. Este ambiente é ainda mais complexo pelo facto de não existir uma observação total do espaço de jogo, por isso, cada robô tem o seu sistema de visão. Para que seja possível jogar é necessário a perceção do espaço de jogo por parte de cada um dos robôs, fusão sensorial, algoritmos para o controlo do movimento e ainda a colaboração entre os múltiplos robôs, através de estratégias de jogo.

Figura 1.2 - Jogo da MSL no RoboCup European Open 2016.

1.1.3. MINHO TEAM

A Minho Team é a equipa de futebol robótico do Laboratório de Automação e Robótica (LAR) da Universidade do Minho. Esta surgiu em 1998 e tem participado ativamente na Middle Size League, em eventos nacionais e internacionais, tais como o Festival Nacional de Robótica (FNR) e o RoboCup. Fez a sua primeira participação no RoboCup em 1999, sendo uma das primeiras equipas a surgir nestas provas, onde tem colaborado ao longo destes anos para a evolução do estado da arte desta competição.

A equipa desde que iniciou a sua atividade tem desenvolvido várias versões de robôs para competir na MSL, tal como ilustrado na *Figura 1.3*. Nessas versões foram implementadas algumas soluções inovadoras que ainda hoje são utilizadas por outras equipas, tais como, o sistema de chuto eletromagnético e o dribbler [8], [9], [10], [11], [12].

A Minho Team nem sempre competiu devido à saída dos seus elementos, pois esta contou com vários alunos que quando concluíam o curso terminavam a sua ligação com a equipa e não existia continuidade do trabalho desenvolvido. Alguns alunos que atualmente estão na equipa retomaram o

projeto em 2015, dando início à remodelação da plataforma anterior desenvolvida em 2011. No *Capítulo 4* será apresentada a versão mais atual dos robôs da Minho Team.

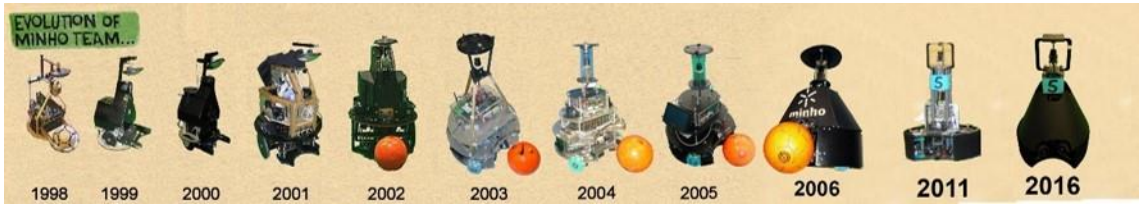


Figura 1.3 – Evolução da Minho Team.

1.2. DESCRIÇÃO DO PROBLEMA

O RoboCup apresenta atualmente um elevado nível de competitividade e a MSL não é exceção. Para este nível de exigência é necessário que *software*, *hardware* e mecânica estejam bem estruturados. Visão por computador, inteligência artificial, gestão das comunicações, planeamento da trajetória e controlo são algoritmos importantes para a coordenação destes robôs. Na MSL cada uma das duas equipas em jogo é constituída por 5 robôs futebolistas de tamanho médio (até 80 cm de altura e 40 kg de peso), são robôs totalmente autónomos e operam num ambiente extremamente dinâmico, constituído por obstáculos dinâmicos que se deslocam a velocidades elevadas (por vezes atingindo os 4 m/s). Neste ambiente dinâmico cada robô necessita de se deslocar de um ponto A para um ponto B sem colidir com os obstáculos em campo, exceto a bola, caso contrário pode ser assinalado falta. Para isso, terá de planear constantemente uma trajetória, de preferência a mais curta e a mais suave, e percorrer essa trajetória no menor tempo possível.

1.3. OBJETIVOS

O principal objetivo desta dissertação passa pela implementação e validação de uma arquitetura de controlo de movimento, baseada no planeamento da trajetória, com aplicação prática na equipa de futebol robótico da Universidade do Minho, denominada Minho Team. O objetivo inicial para todos os elementos da equipa foi a reconstrução e melhoramento da plataforma de cada robô, desenvolvida em 2011. Para isso, foram atribuídas tarefas a cada um dos elementos, no que diz respeito à estrutura mecânica, ao *hardware* e ao desenvolvimento de *software*.

Outro objetivo é publicar o *software* desenvolvido no trabalho desta dissertação, de modo a contribuir para o crescente estado da arte da robótica, do RoboCup e mais especificamente da MSL.

Para alcançar os objetivos supracitados, será necessário trabalhar nas seguintes tarefas:

- Estudo do estado da arte em algoritmos de planeamento da trajetória utilizados em robótica e em algumas equipas que participam na MSL;
- Estudo dos algoritmos a implementar nesta dissertação para o planeamento da trajetória e para o controlo do movimento;
- Implementação de todos os algoritmos num módulo individual do restante *software* da Minho Team utilizando o ROS (*Robot Operating System*);
- Implementação de ferramentas/*software* que possibilite a configuração dos parâmetros que necessitam de ajuste;
- Testar todo o *software* desenvolvido utilizando o simulador da Minho Team;
- Testar todo o *software* no campo de pequenas dimensões disponível no LAR;
- Testes em ambiente de competição/jogar um jogo da MSL;

É também um objetivo desta dissertação que o *software* desenvolvido possa ser reutilizado noutros tipos de robôs.

1.4. PUBLICAÇÕES

H. Ribeiro, P. Silva, R. Roriz, T. Maia, R. Saraiva, G. Lopes, and A. F. Ribeiro, "Fast Computational Processing for Mobile Robots' Self-Localization," *Proc. - 2016 Int. Conf. Auton. Robot Syst. Compet. ICARSC 2016*, no. March, pp. 168–173, 2016.

Capítulo 2

ESTADO DA ARTE

Neste capítulo é feita uma análise à arquitetura de controlo de movimento, mais especificamente, o planeamento da trajetória para os robôs de algumas equipas que participam na MSL. CAMBADA, Carpe Noctem Cassel e Tech United Eindhoven são as equipas descritas neste capítulo, pois são as que publicam a informação sobre este tema com maior detalhe.

2.1. CAMBADA

CAMBADA é a equipa de futebol robótico da Universidade de Aveiro [13]. Este projeto começou em 2003, contando com várias participações na Middle Size League em eventos nacionais e internacionais. Nas suas participações destaca-se o primeiro lugar no RoboCup em 2008.

Esta equipa faz a discretização da representação do espaço de jogo, ou de trabalho, em células, utilizando o método aproximado de decomposição em células. Cada célula tem uma resolução de 50 cm × 50 cm e pode ser do tipo: posição atual (azul), posição pretendida (vermelho), obstáculo (preto), espaço livre (verde). Na *Figura 2.1* é apresentado um exemplo da discretização da representação do espaço de jogo com as cores de cada célula.

Para a pesquisa pela trajetória entre a posição atual e a posição pretendida é implementado o algoritmo A* (*A Star*). Este algoritmo foi modificado para evitar colisões com os adversários, como também, manter uma distância segura para que estes não consigam tirar a bola. Este A* modificado consiste em não considerar células a uma distância n de células com obstáculo. Um exemplo de duas trajetórias obtidas pelo A* e A* modificado podem ser observadas na *Figura 2.1*. As células com a cruz magenta formam a trajetória obtida pelo algoritmo A* e as células com a cor amarela formam a trajetória obtida pelo A* modificado [14].

O *software* desta equipa é implementado com a linguagem de programação C++. A implementação do algoritmo para a discretização da representação do espaço de jogo e para a pesquisa da trajetória é auxiliado pela API Libtcod [15]. Esta API é bastante utilizada por desenvolvedores de jogos e está disponível em diferentes linguagens de programação.

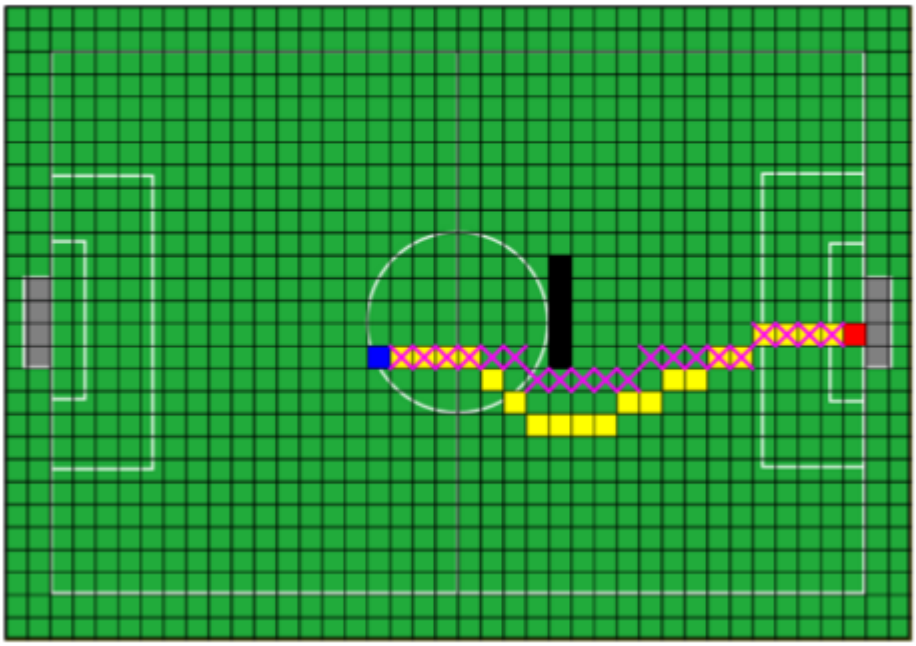


Figura 2.1 – Discretização da representação do espaço de jogo em células e um exemplo de duas trajetórias obtidas por A* e A* modificado. Algoritmo implementado pela equipa CAMBADA [14].

2.2. CARPE NOCTEM CASSEL

Carpe Noctem Cassel é uma equipa de futebol robótico da Universidade de Kassel, Alemanha. Desde 2005 que esta equipa participa na Middle Size League, em eventos do RoboCup e também no Festival Nacional de Robótica em Portugal [16].

Esta equipa, como todas as outras, não utiliza nenhuma unidade de planeamento central da trajetória, ou seja, cada robô faz o planeamento da sua trajetória. O algoritmo para obter a melhor trajetória (ver *Figura 2.2*) está dividido em duas fases: *Data Fusion* e *Planning*. A fusão de dados (*Data Fusion*) é constituída por três etapas: agrupamento e validação (*Clustering & Validation*), observador/filtro (*Object Tracker*) e construção do diagrama de Voronoi (*Voronoi Construction*). A primeira etapa, agrupamento e validação, recebe a posição e velocidade de cada um dos robôs cooperativos (robôs da equipa) e ainda a posição dos obstáculos na área de visão de cada robô, estes dados são recebidos a cada 100 ms. Após receber os dados, cada robô funde a sua própria posição e seus obstáculos percebidos com as posições e os obstáculos de todos os robôs cooperativos. Concluído o agrupamento e validação de todos os dados, segue-se a construção do diagrama de Voronoi. Para a construção do diagrama são consideradas as posições de todos os obstáculos, que podem ser robôs cooperativos e/ou robôs adversários. As posições de robôs adversários não são muito precisas, por isso não são enviadas diretamente para a construção do diagrama, sendo enviadas primeiro para a etapa observador/filtro.

Esta etapa remove falsos positivos, compensa posições que possam faltar e também estima velocidades [17].

A fase planeamento (*Planning*) é constituída por duas etapas: planeamento da trajetória (*Path Planning*) e correção da trajetória (*Path Correction*). A etapa de planeamento da trajetória recebe o diagrama de Voronoi calculado na fase fusão de dados e recebe também a posição pretendida, dada por um comportamento. Nesta etapa é utilizado o algoritmo A* para a pesquisa da melhor trajetória, sendo esta trajetória constituída por uma sequência de arestas de Voronoi. As funções heurísticas do algoritmo A* são aplicadas para estimar o custo de uma sequência de arestas, de modo a encontrar a trajetória de acordo com determinados fatores, por exemplo, distância da trajetória e a diferença angular entre arestas. A etapa seguinte, correção da trajetória, baseia-se no trabalho de Fujimori (2000) que consiste num comportamento anti-colisão reativo, ou seja, consiste num planeamento local da trajetória. Esta etapa é processada quando uma colisão entre dois robôs é iminente, sendo utilizada a direção global de cada robô calculada na etapa anterior, e desta forma, verificar se existe ou não uma possível colisão. Como em determinados casos mudar de direção não chega para evitar a colisão entre os robôs da equipa, são consideradas as velocidades destes, sendo que, a velocidade para o robô com maior prioridade é aumentada e a velocidade para o que tem menor prioridade é reduzida. No final é enviado para o subsistema do atuador o comando de condução resultante. Todo o processo para obter a melhor trajetória é repetido a cada 30 ms [17].

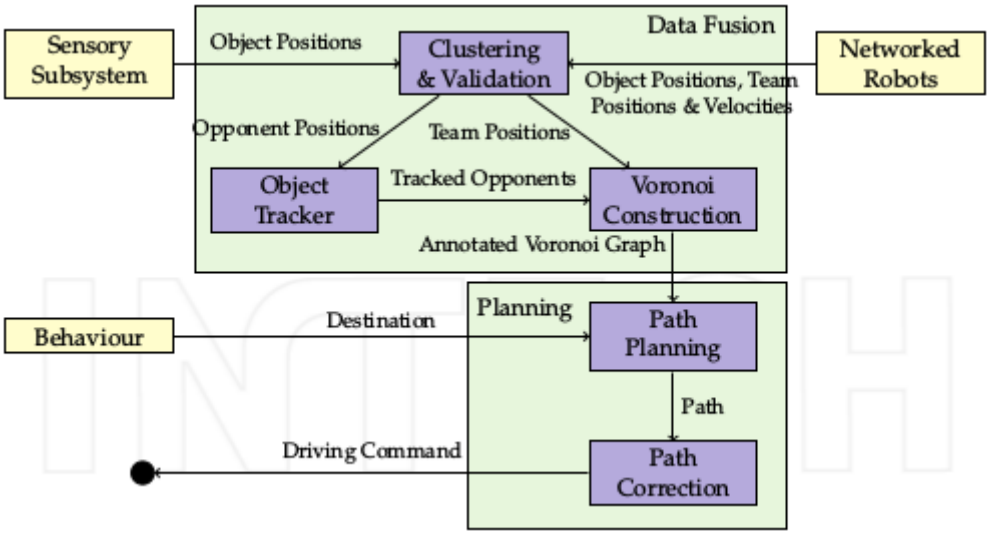


Figura 2.2 - Visão geral da arquitetura implementada pela equipa Carpe Noctem Cassel para obtenção da melhor trajetória [17].

No passado, o *software* desta equipa era implementado com a linguagem de programação C#, mais tarde transitaram todo o *software* para C++. Com esta transição, foi possível a integração da

biblioteca de algoritmos de geometria computacional CGAL, tendo melhorado o desempenho do processo para obtenção da melhor trajetória [18].

2.3. TECH UNITED EINDHOVEN

Tech United Eindhoven é uma equipa da Universidade de Tecnologia de Eindhoven, Holanda. Fundada em 2005, participa desde 2006 em vários eventos da Middle Size League [19]. Nas suas participações destaca-se o primeiro lugar obtido no RoboCup em 2012, 2014 e 2016. Com o conhecimento adquirido no futebol robótico, criaram em 2011 um robô de serviço para competir na liga RoboCup@Home [20].

Esta equipa tem um planeamento da trajetória bastante eficaz, devido à excelente precisão da representação do espaço de jogo, que contém as informações sobre as posições e velocidades dos robôs adversários. A discretização da representação do espaço de jogo é feita pelo diagrama de Voronoi e a pesquisa pela trajetória mais curta é feita pelo algoritmo de Dijkstra. Tal como todas as outras equipas, cada robô faz o planeamento da sua trajetória.

Na *Figura 2.3* é apresentado um exemplo do primeiro passo para o planeamento da trajetória. Os números presentes na figura representam os robôs adversários (obstáculos), e os círculos vermelhos representam a área onde eles poderiam ir no tempo que é necessário para um robô desta equipa deslocar-se desde a posição atual até à posição pretendida. O diagrama de Voronoi é representado pelas linhas a preto, porém, as linhas do diagrama que intersetam os círculos não são consideradas na pesquisa pela trajetória.

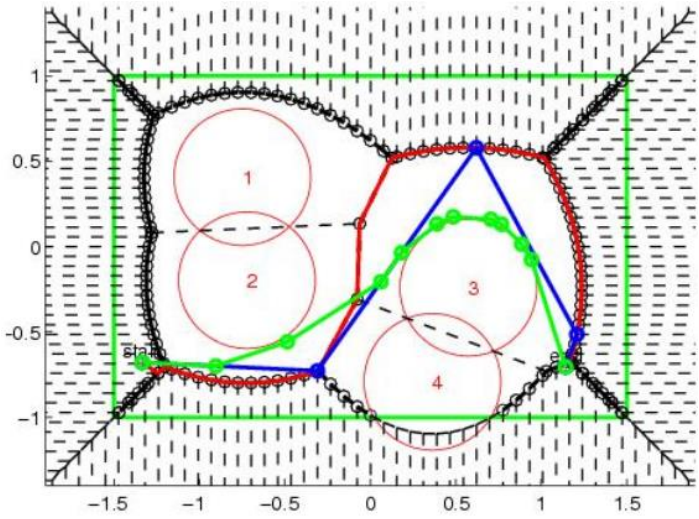


Figura 2.3 - Diagrama de Voronoi com as trajetórias obtidas através do algoritmo implementado pela equipa Tech United Eindhoven [19].

A trajetória mais curta obtida pelo algoritmo de Dijkstra, no exemplo da *Figura 2.3*, é constituída pela sequência de segmentos a vermelho. De seguida, um algoritmo é implementado para encurtar mais esta trajetória, que resulta na sequência de segmentos azuis. Esta é transformada noutra trajetória ainda mais curta, representada pela sequência de segmentos verdes.

O segundo passo do planeamento da trajetória consiste na criação de uma trajetória de tempo (ver *Figura 2.4*), para que o robô percorra a trajetória planeada no primeiro passo (segmentos verdes) o mais rápido possível, considerando também as limitações do atuador, que neste caso, é o conjunto de motores que fazem mover o robô. Curvas de Bézier são calculadas para obter a trajetória de tempo, tendo como base a trajetória obtida no primeiro passo. As limitações do atuador, ou seja, limitações cinemáticas como a aceleração máxima, velocidade máxima, velocidade inicial e velocidade máxima final são incluídas no cálculo das curvas.

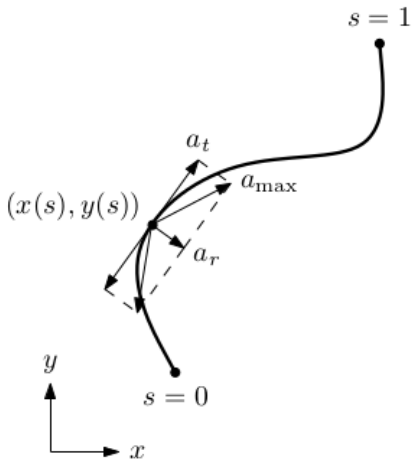


Figura 2.4 - Trajetória de tempo implementada pela equipa Tech United Eindhoven [19].

O algoritmo para o planeamento da trajetória, como todo o *software* desta equipa, é implementado com a linguagem de programação C em MATLAB.

2.4. DISCUSSÃO DO ESTADO DA ARTE

As três equipas apresentadas neste capítulo são muito competitivas, sendo que nos últimos anos a CAMBADA e Tech United Eindhoven têm conquistado constantemente um lugar entre as três melhores equipas da MSL. Este grande nível de competitividade é o resultado de muitos anos de trabalho contínuo, em investigação e desenvolvimento de *software* para a implementação de novos algoritmos, e também no desenvolvimento constante do *hardware* e da mecânica dos robôs.

Analisando as três equipas, é possível tirar algumas conclusões sobre os algoritmos mais utilizados para o planeamento da trajetória. A primeira conclusão tem a ver com a escolha do modo que o planeamento da trajetória é implementado, pois este pode ser centralizado ou descentralizado. As três equipas supracitadas, tal como todas as outras equipas da MSL, usam o planeamento descentralizado, isto significa que cada robô de uma equipa faz o planeamento da sua trajetória. Esta escolha deve-se ao facto de o espaço de jogo ser extremamente dinâmico, e com a existência de mais que um robô a controlar por cada equipa, no caso da MSL, 5 robôs por cada equipa. Por outro lado, no planeamento centralizado é planeada a trajetória de todos os robôs, usando um único algoritmo e processado na mesma máquina. Desta forma, este modo de planeamento não é opção para as equipas da MSL, pois o processamento é bastante elevado e demorado, sendo necessário uma elevada capacidade de memória.

Para a discretização da representação do espaço de jogo, as equipas Carpe Noctem Cassel e Tech United Eindhoven utilizam o diagrama de Voronoi, enquanto que a equipa CAMBADA utiliza o método aproximado de decomposição em células.

O planeamento da trajetória pode ser global, local ou uma combinação dos dois. Analisando a arquitetura de planeamento das três equipas, é possível afirmar que a equipa Carpe Noctem Cassel implementa o planeamento global e local, ou seja, uma combinação dos dois. O diagrama de Voronoi para a discretização da representação do espaço e o algoritmo A* para a pesquisa da melhor trajetória são implementados para criar o planeamento global, enquanto que a correção da trajetória com base no trabalho de Fujimori (2000), que consiste num comportamento anti-colisão reativo, é o planeamento local. Com esta combinação, é possível obter um modelo de planeamento da trajetória eficiente e robusto. Por outro lado, as equipas CAMBADA e Tech United Eindhoven implementam apenas o planeamento global. Isto não significa que seja menos eficiente e menos robusto, pois os métodos ou algoritmos escolhidos por estas duas equipas, para a discretização da representação do espaço de jogo e para a pesquisa da trajetória, são adaptados e adicionados outros algoritmos desenvolvidos por estas equipas. O resultado final é de um planeamento da trajetória com eficiência e robustez igual ou superior da equipa Carpe Noctem Cassel.

Relativamente aos algoritmos para a pesquisa da trajetória, duas equipas implementam o A* e a outra implementa o algoritmo de Dijkstra. A escolha destes algoritmos depende muito da arquitetura de planeamento de cada equipa e da estratégia adotada. Existindo, como é óbvio, vantagens e desvantagens por parte de cada algoritmo, sendo que estas não são mencionadas nas publicações disponíveis pelas equipas.

Para finalizar, a linguagem de programação utilizada por CAMBADA e Carpe Noctem Cassel é C++, enquanto que a Tech United Eindhoven utiliza C em MATLAB, que juntamente com o Simulink utilizam como ferramenta para simular todo o espaço de jogo.

Capítulo 3

FUNDAMENTOS TEÓRICOS

Neste capítulo serão apresentados os métodos e respetivos conceitos para o desenvolvimento desta dissertação. O primeiro subcapítulo é sobre navegação autónoma, que inicia com uma breve introdução, e de seguida são apresentados alguns métodos e respetivos conceitos sobre o planeamento da trajetória e controlo do movimento, sendo estes descritos com a intenção de serem implementados em robôs móveis autónomos. O segundo subcapítulo é uma introdução ao ROS (*Robot Operating System*) e à sua arquitetura.

3.1. NAVEGAÇÃO AUTÓNOMA

Os robôs ou veículos móveis autónomos têm ganho cada vez mais importância na sociedade, devido ao crescente investimento na investigação e desenvolvimento por parte de diversas entidades, de forma a desenvolver soluções para aplicações industriais, hospitalares, domésticas, militares, exploração espacial e mais recentemente nos transportes. Com esta expansão e exigência, foi indispensável ao longo dos últimos anos o desenvolvimento de sistemas móveis autónomos robustos e adaptáveis a qualquer tipo de ambiente.

Um dos problemas fundamentais num sistema móvel autónomo é a navegação. Um robô ou veículo móvel autónomo tem de se mover entre a posição atual e a posição pretendida, de modo eficaz. Para que tal seja possível são necessários os seguintes processos: (a) perceção do espaço de trabalho: através da recolha de informação de todos os sensores, (b) localização e construção de um mapa: com os dados obtidos na perceção do espaço de trabalho é construído um mapa onde os obstáculos são mapeados. É também determinada a localização (posição e orientação) do robô no mapa, (c) planeamento da trajetória: encontra a melhor trajetória a ser percorrida pelo robô, e (d) controlo do movimento: calcula os valores para o controlo dos motores, de modo a que o robô percorra a trajetória obtida no processo anterior. Na *Figura 3.1* são apresentados de uma forma geral os processos de navegação [21].

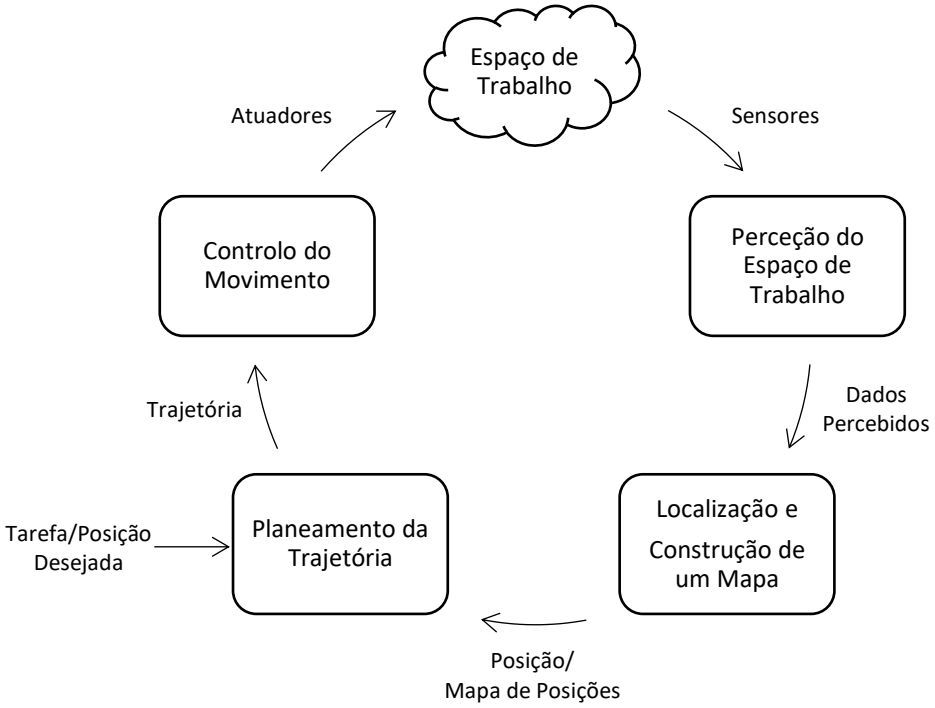


Figura 3.1 - Organização geral dos processos de navegação, adaptado de S. G. Tzafestas (2013) [21].

Planeamento da trajetória e controlo do movimento são os processos de navegação investigados e implementados neste trabalho de dissertação. Tal como já foi mencionado no início deste capítulo, todos os métodos e respetivos conceitos apresentados no atual subcapítulo, sobre estes dois processos de navegação, serão descritos com a intenção de serem implementados em robôs móveis autónomos. Contudo, é importante referir que estes podem ser implementados noutros tipos de robôs, tais como robôs manipuladores (braço robótico), robôs móveis/manipuladores e robôs humanoides.

Para estes dois processos de navegação existem diversos métodos, por isso serão apresentados apenas aqueles que foram implementados no trabalho desta dissertação. O motivo para a utilização de cada método será descrito no *Capítulo 5 IMPLEMENTAÇÃO*.

3.1.1. PLANEAMENTO DA TRAJETÓRIA

O planeamento da trajetória ou como é mais conhecido em inglês, *Path Planning*, é um dos processos de navegação supracitados no atual subcapítulo *NAVEGAÇÃO AUTÓNOMA*. Este processo tem como objetivo, determinar uma trajetória ótima para o sistema autónomo. Com essa trajetória ótima o robô ou veículo móvel autónomo deve ser capaz de se deslocar da posição atual para outro local e, eventualmente, para a posição pretendida final, ao mesmo tempo evitando quaisquer obstáculos. Uma trajetória tem também um determinado custo que deve ser minimizado, esse custo pode ser em relação: (a) comprimento: encontrar a trajetória mais curta, (b) tempo: a trajetória a percorrer no menor tempo

possível, (c) esforço: minimizar o consumo de energia do robô, (d) espaço: aumentar o espaço de manobra para o robô, e/ou, (e) segurança: minimizar a possibilidade de qualquer colisão com obstáculos.

Para resolver um problema de planeamento da trajetória é indispensável analisar o espaço de trabalho ou ambiente antes de optar por qualquer tipo de planeamento (global, local ou uma combinação dos dois), ou por qualquer método. Desta forma, é necessário analisar qual o conhecimento disponível que o robô móvel autónomo tem sobre o espaço de trabalho e saber qual o tipo de obstáculos que possam existir.

O espaço de trabalho onde se encontra o robô pode ser totalmente conhecido, parcialmente conhecido ou totalmente desconhecido. Na maioria dos casos é parcialmente conhecido, isto significa que o robô não tem um conhecimento completo de todos os obstáculos no espaço de trabalho. Os obstáculos podem ser, obstáculos estáticos ou obstáculos em movimento.

3.1.1.1. TIPOS DE PLANEAMENTO DA TRAJETÓRIA

O planeamento da trajetória pode ser global, local ou uma combinação dos dois, criando um sistema de planeamento híbrido [22], [23]. O planeamento global está inserido numa arquitetura de controlo/navegação deliberativa (ver *Figura 3.2*), este tipo de planeamento exige um mapa com a representação total do espaço de trabalho para obter uma trajetória ótima com uma posição precisa até à posição pretendida. Porém, é difícil obter um mapa completo, por isso, este planeamento não é adequado para um espaço de trabalho dinâmico (obstáculos em movimento).

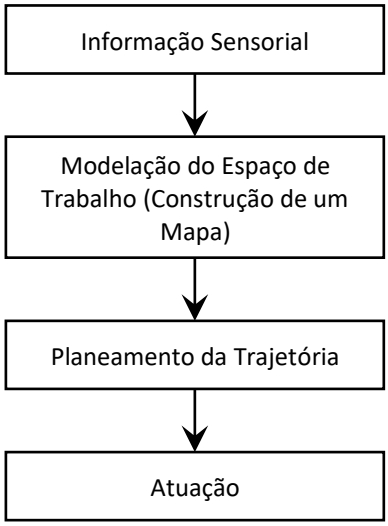


Figura 3.2 - Arquitetura de controlo/navegação deliberativa.

Os algoritmos para o planeamento global exigem uma elevada capacidade de processamento e memória. O tempo de resposta é relativamente mais lento, que pode originar atraso na navegação.

O planeamento local está inserido numa arquitetura de controlo/navegação reativa (ver *Figura 3.3*), esta surgiu depois da arquitetura deliberativa com o objetivo de superar os problemas de navegação em espaços dinâmicos e totalmente desconhecidos. Desta forma, a arquitetura reativa é baseada em comportamentos, que dependem da informação sensorial. Isto é, o robô obtém informação local do espaço que o rodeia através de sensores, posteriormente, uma função de transferência recebe a informação e seleciona um comportamento. Para terminar, é enviado os comandos necessários para os atuadores com base no comportamento selecionado.

Assim, com o planeamento local da trajetória já não é necessário construir um mapa completo do espaço de trabalho, sendo executado com o robô em movimento [24] e com a informação obtida localmente. Este tipo de planeamento também oferece uma resposta rápida para gerar uma nova trajetória no espaço dinâmico e exige pouco processamento. Contudo, não produz uma trajetória tão eficaz como o planeamento global, não sendo adequado para tarefas mais complexas.

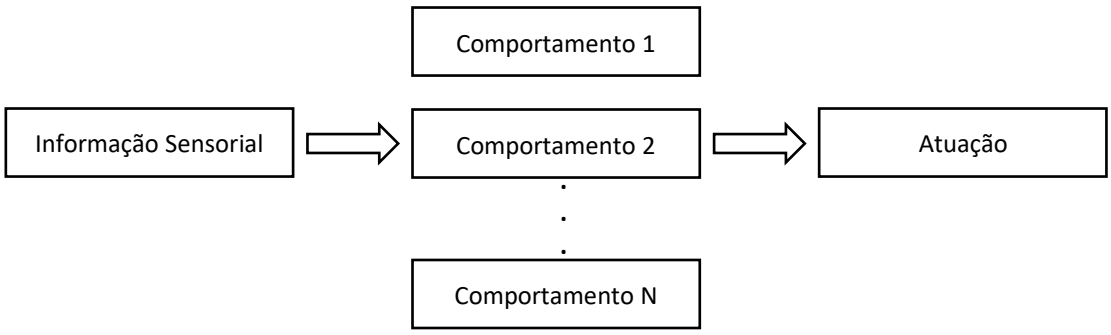


Figura 3.3 - Arquitetura de controlo/navegação reativa.

Cada uma das duas arquiteturas já apresentadas tem as suas vantagens e desvantagens, não existindo por isso uma solução ótima. Para resolver melhor o problema de navegação surgiu a arquitetura de controlo/navegação híbrida, esta é uma combinação das caraterísticas da deliberativa com a reativa, que dá origem a um planeamento da trajetória bastante robusto.

Na *Figura 3.4* é apresentada a estrutura mais comum para a arquitetura híbrida, sendo esta constituída por três camadas: deliberativa, controlo de execução e reativa. A camada deliberativa é responsável pela fusão sensorial, construção do mapa com a representação do espaço de trabalho e do planeamento global da trajetória. A camada de controlo de execução faz a conexão entre a camada deliberativa e a camada reativa, com a tarefa de coordenar os comportamentos, de modo a que o robô

percorra a trajetória planeada na camada deliberativa. A camada reativa define um comportamento e envia os comandos necessários para os atuadores.

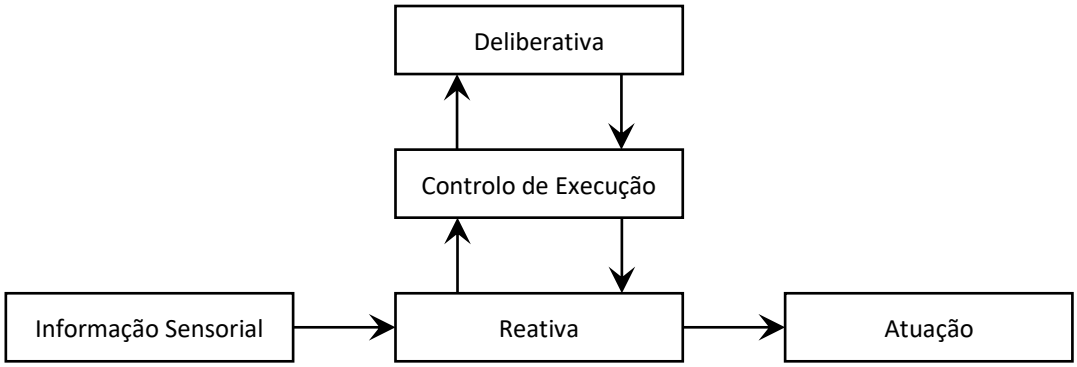


Figura 3.4 - Arquitetura de controlo/navegação híbrida.

O tipo de planeamento da trajetória implementado neste trabalho de dissertação é o planeamento global, sendo que os métodos e respetivos conceitos apresentados nas secções seguintes fazem parte deste tipo de planeamento. O motivo para a escolha do planeamento global da trajetória será descrito no *Capítulo 5 IMPLEMENTAÇÃO*.

3.1.1.2. ESPAÇO DE CONFIGURAÇÃO

O primeiro passo para obter uma solução para o problema do planeamento da trajetória de um robô é, criar uma representação do espaço de trabalho no qual o robô está inserido [25]. Essa representação a que se dá o nome de espaço de configuração (EC) é aplicado em robôs manipuladores (braços robóticos) e na maioria dos robôs móveis [26]. No trabalho desenvolvido nesta dissertação, é considerado o espaço de configuração apenas para o espaço de trabalho de robôs móveis.

No espaço de trabalho existem locais ocupados (obstáculos) e locais não ocupados (espaço livre) [27]. Desta forma, para planear uma trajetória, o robô e o espaço de trabalho devem ser modelados por um modelo matemático adequado. Por isso, um espaço de estados é utilizado na configuração do robô (espaço) [22].

O espaço de estados é constituído por um vetor de dimensão N, que deve descrever com precisão o estado do robô no seu ambiente. A dimensão do vetor determina o número de graus de liberdade (DOF). No caso de um robô móvel sem braço robótico, DOF corresponde ao número de parâmetros independentes necessários para especificar o movimento [22]. Assim, o espaço de estados é definido pelos parâmetros: posição e orientação do robô no espaço de trabalho.

No cálculo da trajetória, quanto maior o número de graus de liberdade maior é a complexidade computacional, aumentando assim exponencialmente o tempo de cálculo. Por isso, o objetivo é simplificar ao máximo o problema, de modo a aumentar a eficácia dos algoritmos. Um exemplo, é de um robô móvel sem braço robótico que pode ter no máximo seis graus de liberdade (6 DOF), definidos pela posição e orientação nos eixos coordenados x, y e z, isto significa, que pode mover-se em qualquer direção no espaço tridimensional (3D). Para simplificar, a abordagem mais comum é considerar o movimento num espaço a duas dimensões (ver *Figura 3.5*), restringindo assim o número de graus de liberdade para três: posição do robô nas coordenadas (x, y) e a orientação.

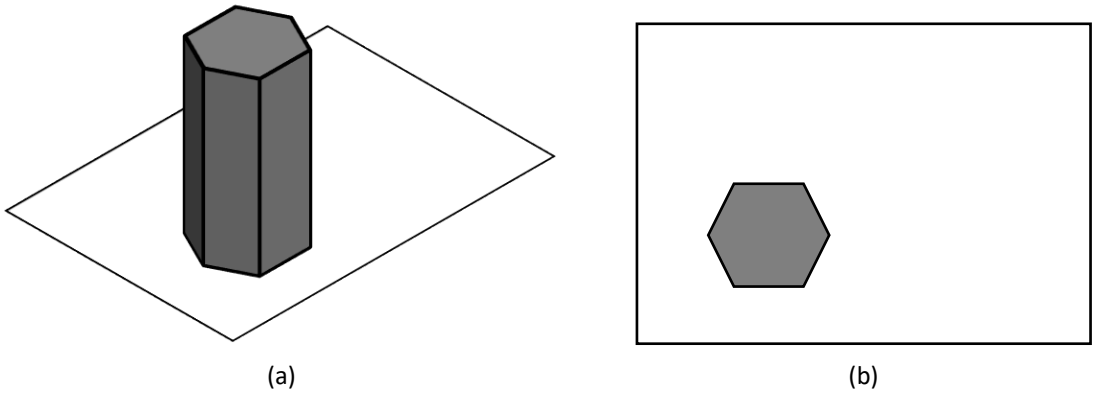


Figura 3.5 - (a) Representação de um objeto ou robô no espaço de trabalho (espaço a três dimensões). (b) Espaço de configuração com o mesmo objeto ou robô (espaço a duas dimensões).

Para o planeamento da trajetória geralmente assume-se que o robô é holonómico (não apresenta restrições no movimento, ou seja, pode mover-se em qualquer direção com uma qualquer orientação) [26]. A orientação pode ser assim excluída do planeamento, como também é possível assumir que o robô tem uma forma circular no espaço de configuração (ver *Figura 3.6(a)*). Portanto, o robô é representado apenas pela sua posição (x, y).

A forma mais prática e simples na implementação dos algoritmos para o planeamento da trajetória, é representar o robô como um ponto e aumentar o tamanho dos obstáculos de acordo com o raio do robô, para obter um espaço de configuração equivalente (ver *Figura 3.6(b)*) [22], [27].

Na *Figura 3.6(b)* é possível observar que o espaço de configuração (EC) é constituído por dois tipos de locais: EC_{obst} e EC_{livre}. Os locais com obstáculos (EC_{obst}) é espaço ocupado que o robô tem de evitar, enquanto que os locais não ocupados (EC_{livre}) é espaço livre onde o ponto que representa a posição do robô se pode mover. Assim, o problema do planeamento da trajetória é, encontrar uma trajetória entre a posição atual e uma posição pretendida no espaço de configuração livre [21].

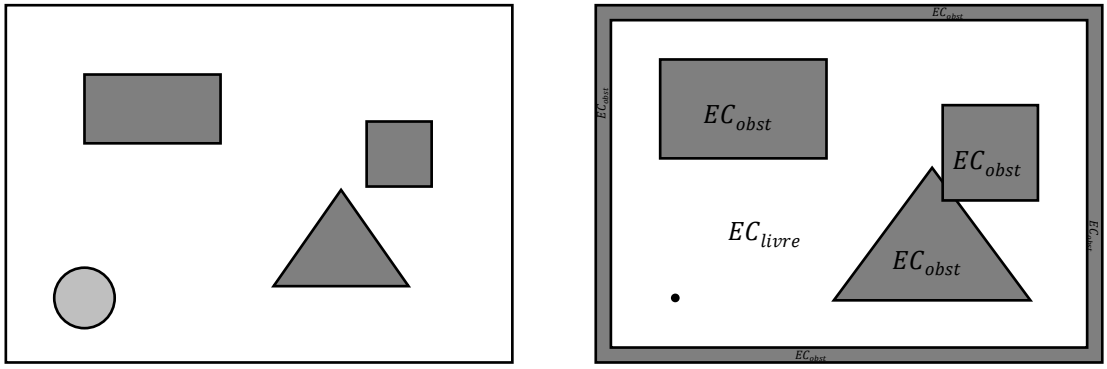


Figura 3.6 – Exemplo de um espaço de configuração. (a) Representação de três obstáculos com formas diferentes (cor mais escura) e um robô com uma forma circular. (b) O mesmo espaço de (a) mas com os obstáculos e limites do espaço expandidos de acordo com o raio do robô, pois este é representado por um ponto. $\boldsymbol{EC_{livre}}$ é o espaço livre e $\boldsymbol{EC_{obst}}$ é o espaço ocupado por obstáculos. Esta figura é adaptada de R. Jitendra (2007) [22] e de M. Queirós (2014) [28].

3.1.1.3. DIAGRAMA DE VORONOI

Como já citado na secção *ESPAÇO DE CONFIGURAÇÃO*, o problema do planeamento da trajetória é, encontrar uma trajetória entre a posição atual e uma posição pretendida no espaço de configuração livre (EC_{livre}). O espaço de configuração é um mapa que faz a representação geométrica contínua do espaço de trabalho do robô, porém, este mapa deve ser transformado num mapa discreto, adequado ao algoritmo de pesquisa da trajetória a ser implementado [21].

Uma das abordagens para obter um mapa discreto tem o nome de *roadmap* [27]. Esta abordagem consiste em conectar o espaço de configuração livre através de uma rede de curvas, tendo cada um dos métodos de *roadmap* existentes uma configuração de mapa diferente. Um desses métodos é o diagrama de Voronoi, que será descrito nesta secção.

O diagrama de Voronoi é uma estrutura de geometria computacional utilizada em diversas aplicações. Este diagrama produz uma rede de curvas (ou arestas) que dão forma a possíveis trajetórias no espaço de configuração livre e que maximizam a distância entre o robô e os obstáculos [27]. No entanto, uma trajetória obtida diretamente do diagrama de Voronoi pode não ser ótima, no caso em que se pretende a trajetória mais curta, pois em alguns locais a distância pode ser longa entre os obstáculos e as arestas que compõem a trajetória [29]. Neste caso, existem estratégias/algoritmos que podem ser adicionadas a este diagrama, ou então, a implementação de outro método de *roadmap*, como até mesmo, a utilização de outro tipo de abordagem para obter um mapa discreto. Para uma trajetória segura, longe dos obstáculos, o diagrama de Voronoi é ótimo.

Este diagrama, para um conjunto de pontos no plano, consiste na subdivisão desse plano em regiões, em que cada ponto tem uma região associada. Normalmente, estes pontos denominam-se por *sítios* e as regiões por *regiões de Voronoi* [30]. Cada região de Voronoi é um polígono convexo limitado ou ilimitado. As arestas do diagrama podem ser segmentos de reta ou semirretas.

Seja S um conjunto de n sítios do diagrama de Voronoi $DV(S)$. Uma aresta é comum a duas regiões adjacentes de dois sítios ($s_i, s_j \in S$), se cada ponto da aresta é equidistante de s_i e s_j. Assim, os pontos da aresta estão mais próximos destes dois sítios do que qualquer outro sítio de $DV(S)$. Cada vértice do diagrama é equidistante de três ou mais sítios e três ou mais arestas têm um vértice em comum [31]. Qualquer ponto (ou local) dentro de uma determinada região de Voronoi, está mais próximo do sítio dessa região do que qualquer outro sítio de $DV(S)$.

Na *Figura 3.7* é apresentado um exemplo de um diagrama de Voronoi de 14 sítios no plano.

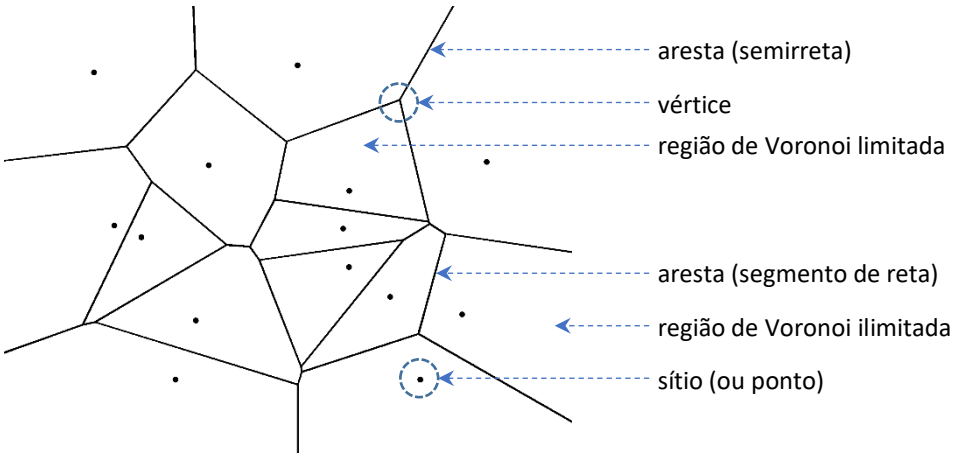


Figura 3.7 - Diagrama de Voronoi de 14 sítios (ou pontos) no plano.

3.1.1.4. TRIANGULAÇÃO DE DELAUNAY

Uma triangulação de um conjunto de pontos no plano é um grafo $G(V, E)$, onde os vértices V correspondem ao conjunto de pontos e E é o conjunto de arestas formadas por pares de vértices distintos. Um grafo de uma triangulação tem como propriedades: (a) conter um conjunto de vértices $V = \{v_1, \dots, v_n\}$, com $n \geq 3$ vértices distintos no plano e não todos colineares e (b) as arestas não se intercetam, exceto nos extremos, ou seja, nos vértices. Portanto, podem ser adicionados pontos, desde que nenhuma aresta nova intersete numa outra aresta já existente. Com estas propriedades é garantido que cada uma das faces, exceto as externas ao grafo, é um triângulo [32], [33].

Para a construção de uma triangulação de um conjunto de pontos, existem diversas soluções possíveis. Uma triangulação muito utilizada e descrita nesta secção é a triangulação de Delaunay, pois

está relacionada diretamente com o diagrama de Voronoi, existindo assim uma dualidade entre as duas estruturas.

A triangulação de Delaunay de um conjunto de pontos V é um grafo denotado por $TD(V)$. Esta triangulação tem como propriedades: (a) a circunferência circunscrita a um triângulo $\Delta v_i v_j v_k \in TD(V)$, não contém nenhum outro vértice (ponto) de V no seu interior, (b) uma aresta de $TD(V)$ é formada por dois vértices ($v_i, v_j \in V$) se, e somente se, existe uma circunferência que passa apenas por v_i e v_j e não contém nenhum outro vértice de V no seu interior, e (c) o ângulo mínimo interno de cada triângulo é maximizado, de forma a aproximar-se de um triângulo equilátero [33], [34]. Na *Figura 3.8* é apresentado um exemplo de uma triangulação de Delaunay de 14 pontos no plano e ainda a representação de uma circunferência circunscrita a um triângulo, sem qualquer vértice no seu interior.

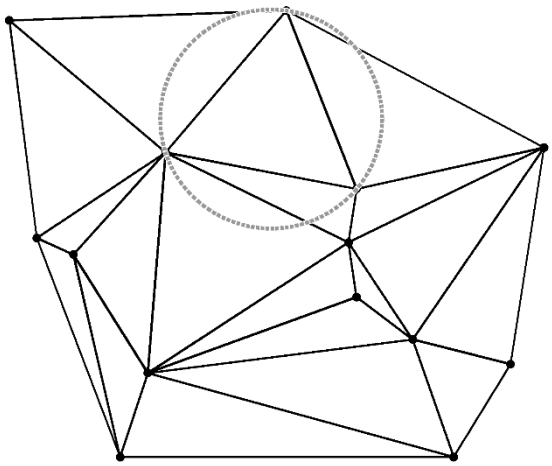


Figura 3.8 - Triangulação de Delaunay de 14 pontos no plano e circunferência circunscrita a um triângulo.

3.1.1.5. DUALIDADE ENTRE VORONOI E DELAUNAY

A triangulação de Delaunay de um conjunto de pontos S é o grafo dual $TD(S)$ do diagrama de Voronoi $DV(S)$ [33]. A *Figura 3.9* ilustra a dualidade das duas estruturas.

Nesta secção são apresentadas algumas propriedades sobre esta dualidade. Uma dessas propriedades é direcionada ao conjunto de pontos (no exemplo da *Figura 3.9* são os pontos a preto), sendo estes comuns a TD e DV, e tal como já foi mencionado nas duas secções anteriores, para TD os pontos correspondem aos vértices dos triângulos e para DV correspondem aos sítios das regiões de Voronoi. É também possível verificar que duas regiões de Voronoi adjacentes, têm os respetivos sítios conectados por uma aresta de TD [35].

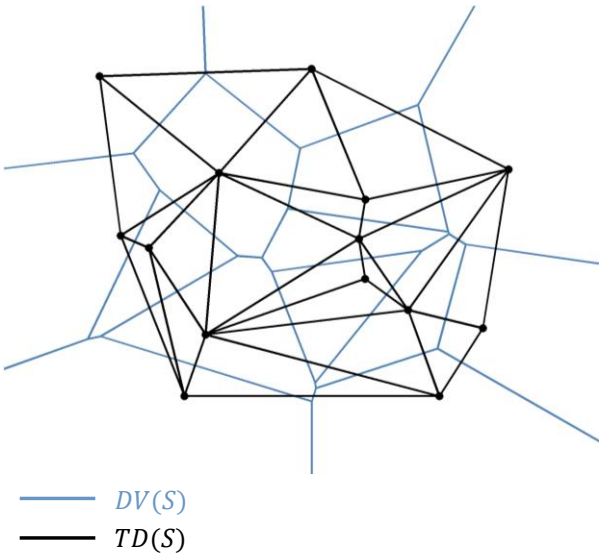


Figura 3.9 - Diagrama de Voronoi $\boldsymbol{DV(S)}$ e Triangulação de Delaunay $\boldsymbol{TD(S)}$.

Na *Figura 3.10* é possível observar outra propriedade, que deve ser interpretada da seguinte forma: a circunferência circunscrita a um triângulo $\Delta v_i v_j v_k \in TD(S)$, não contém nenhum outro ponto de S no seu interior [33], e o circuncentro (centro da circunferência) corresponde a um vértice de $DV(S)$.

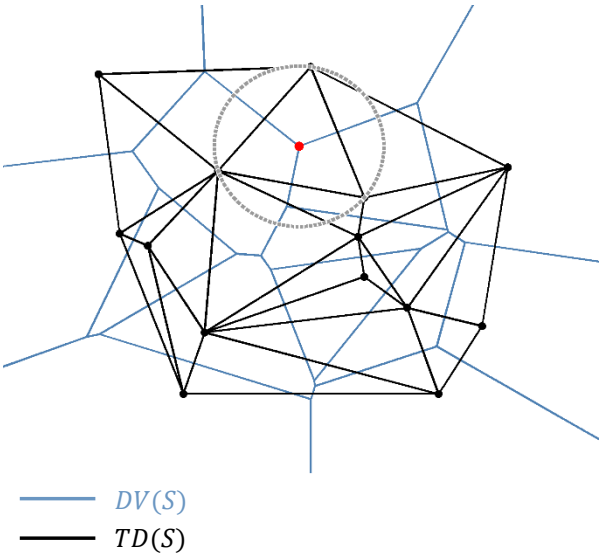


Figura 3.10 - Diagrama de Voronoi $\boldsymbol{DV(S)}$, Triangulação de Delaunay $\boldsymbol{TD(S)}$, e ainda a representação de uma circunferência circunscrita a um triângulo de $\boldsymbol{TD(S)}$, com o circuncentro (ponto a vermelho) a corresponder a um vértice de $\boldsymbol{DV(S)}$. Isto é uma das propriedades da dualidade entre $\boldsymbol{TD(S)}$ e $\boldsymbol{DV(S)}$.

Os algoritmos existentes para construir um diagrama de Voronoi podem ser demorados para determinadas aplicações. Como a triangulação de Delaunay de um conjunto de pontos é o grafo dual do diagrama de Voronoi e os algoritmos existentes para construir a triangulação são menos demorados,

normalmente utiliza-se a triangulação de Delaunay para construir o diagrama de Voronoi do mesmo conjunto de pontos.

3.1.1.6. ALGORITMO DE DIJKSTRA

Um mapa discreto de um espaço de configuração, concebido por um diagrama de Voronoi, é constituído por um conjunto de arestas e por um conjunto de vértices que dão forma a possíveis trajetórias no espaço de configuração livre. No entanto, o objetivo é encontrar apenas uma trajetória entre uma posição inicial e uma posição pretendida, sendo por isso necessário implementar um algoritmo para encontrar a melhor trajetória. A escolha do algoritmo deve ser de acordo com o método que constrói o mapa discreto e com o custo ou critério que deve ser minimizado. Por exemplo, no trabalho desta dissertação é implementado o diagrama de Voronoi para criar o mapa, e o custo é encontrar a trajetória mais curta. Desta forma, o algoritmo mais adequado para este problema e utilizado em diversas aplicações é o algoritmo de Dijkstra.

O algoritmo de Dijkstra foi criado em 1956 e publicado em 1959 pelo cientista da área da computação Edsger Dijkstra [36], [37]. Este algoritmo faz a pesquisa pela trajetória mais curta, ou de custo mínimo, entre um determinado vértice inicial e todos os outros vértices de um grafo $G(V, E)$ direcionado ou não direcionado, onde V corresponde ao conjunto de vértices e E corresponde ao conjunto de arestas. Neste algoritmo as arestas devem possuir um peso/custo não negativo.

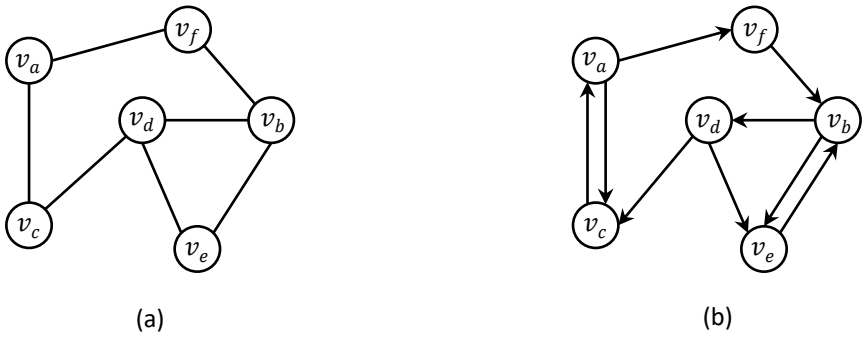


Figura 3.11 – Exemplo de dois grafos, sendo (a) um grafo não direcionado e (b) um grafo direcionado. Os círculos representam os vértices que são ligados por arestas direcionais em (b) e não direcionais em (a).

O algoritmo faz a estimativa do custo mínimo entre um determinado vértice inicial e cada um dos restantes vértices de um grafo, sendo essa estimativa atualizada durante a execução do algoritmo. Os vértices são visitados por ordem crescente do custo estimado para cada vértice, sendo que um vértice será considerado como visitado, quando o algoritmo encontrar uma trajetória de custo mínimo, desde o vértice inicial até este, caso contrário, continua como vértice não visitado. O algoritmo termina quando

todos os vértices tiverem sido visitados. Contudo, este algoritmo também permite encontrar uma trajetória de custo mínimo entre um vértice inicial e um vértice pretendido, sem ser necessário visitar todos os vértices do grafo, pois a execução do algoritmo pode ser interrompida, quando o vértice pretendido for visitado.

Uma trajetória obtida a partir de um grafo, é constituída por uma sequência de vértices. Neste algoritmo essa sequência é guardada por cada vértice, isto é, durante a execução do algoritmo cada vértice guarda o seu vértice antecessor, que pertence à trajetória de custo mínimo.

O pseudocódigo do algoritmo de Dijkstra [38] é apresentado em *Algoritmo 3.1*. A *Figura 3.12* representa um exemplo da execução deste algoritmo, que será explicado em seguida juntamente com o pseudocódigo.

O algoritmo (*Algoritmo 3.1*) tem como entradas, um grafo $G(V, E)$ direcionado ou não direcionado, um vértice inicial $s \in V$ e o custo exato de cada aresta. Cada vértice é denotado por v ou u, sendo que cada aresta conecta dois vértices e é denotada por (u, v), e o seu custo por $w(u, v)$. Tal como já foi mencionado, para este algoritmo as arestas devem possuir um custo não negativo, $w(u, v) \geq 0$.

Na *Figura 3.12* está representado um exemplo de um grafo $G(V, E)$ não direcionado, no qual é apresentado na proximidade de cada aresta o custo exato da respetiva aresta, e no interior do círculo que representa cada vértice é apresentada a estimativa do custo mínimo, $estim$, para o respetivo vértice. Tal como no algoritmo, s é o vértice inicial.

A estimativa do custo mínimo é inicializada para todos os vértices, sendo atribuído zero para o vértice inicial (ver linha *5* do algoritmo) e um valor mais próximo de infinito para os restantes vértices do grafo (ver linhas *2* e *3* do algoritmo). Na linha *4* é inicializado o antecessor de cada vértice. Este primeiro passo do algoritmo está representado na *Figura 3.12(a)*.

O algoritmo utiliza o conjunto S de vértices para saber quais os vértices de V que já foram visitados, ou seja, os vértices com a estimativa final do custo mínimo já determinada. A cada iteração do ciclo principal do algoritmo é adicionado no conjunto S o vértice extraído da fila de prioridade mínima Q, isto significa que esta fila tem todos os vértices de V não visitados e que o vértice a ser extraído é sempre aquele que tem maior prioridade de todos os vértices da fila, ou seja, aquele que tem o menor custo estimado. A fila de prioridade Q é inicializada com todos os vértices de V (ver linha *7*) e o conjunto S é inicializado para conjunto vazio (ver linha *6*).

Depois de todas as inicializações segue-se o ciclo principal (ver da linha *8* a *14*), onde é feita a pesquisa pela trajetória de custo mínimo entre o vértice inicial $s \in V$ e cada um dos restantes vértices

de V. O número de iterações deste ciclo depende do número inicial de vértices da fila de prioridade Q, que corresponde ao número de vértices de V. Este ciclo e a execução do algoritmo terminam quando não existirem mais vértices na fila de prioridade Q.

No início de cada iteração do ciclo principal é extraído da fila de prioridade Q o vértice com o menor custo estimado, $estim$, e atribuído a u (ver linha *9*). Posteriormente, é adicionado no conjunto S o vértice u (ver linha *10*). Na *Figura 3.12* é possível observar que o preenchimento do círculo que representa cada vértice é: branco para os vértices presentes na fila de prioridade Q, preto para os vértices já adicionados no conjunto S, e cinzento claro para o vértice u.

Algoritmo 3.1 Algoritmo de Dijkstra

```
1   DIJKSTRA (G, s, w)
2       para cada vértice v ∈ V fazer
3           estim[v] ← ∞
4           antec[v] ← v
5       estim[s] ← 0
6       S ← ∅
7       Q ← V
8       enquanto Q ≠ ∅ fazer
9           u ← EXTRAIR_MÍNIMO(Q)
10          S ← S ∪ {u}
11          para cada vértice v ∈ Q e adjacente de u fazer
12              se estim[v] > estim[u] + w(u, v) então
13                  estim[v] ← estim[u] + w(u, v)
14                  antec[v] ← u
```

De seguida, é atualizada a estimativa do custo mínimo e o antecessor, $antec$, para cada vértice v presente na fila de prioridade Q e adjacente ao vértice u. Isto, se a estimativa final do custo mínimo do vértice u, mais o custo exato da aresta (u, v), for menor que a estimativa do custo mínimo do vértice v. Estas condições e atualizações podem ser observadas da linha *11* a *14* do algoritmo, e de *(b)* a *(e)* na *Figura 3.12*. De notar que as arestas sombreadas no grafo da figura representam os vértices antecessores.

Na *Figura 3.12(f)* estão representados os resultados finais da pesquisa pela trajetória mais curta, ou de custo mínimo, deste exemplo.

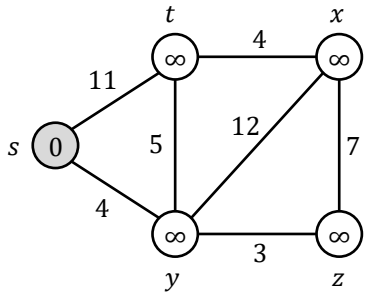
(a)

vértices $v \in V$	s	t	x	y	z
$estim[v]$	0	∞	∞	∞	∞
$antec[v]$	s	t	x	y	z

$S = \{\,\}$

$Q =$ | t | x | y | z | − |

$u \leftarrow s$



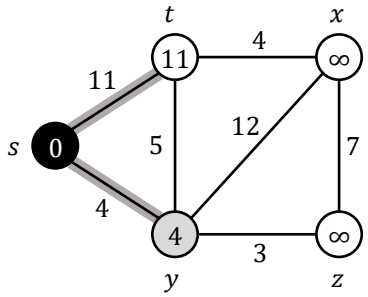
(b)

$S = \{s\}$

vértices $v \in V$	s	t	x	y	z
$estim[v]$	0	11	∞	4	∞
$antec[v]$	s	s	x	s	z

$Q =$ | t | x | z | − | − |

$u \leftarrow y$



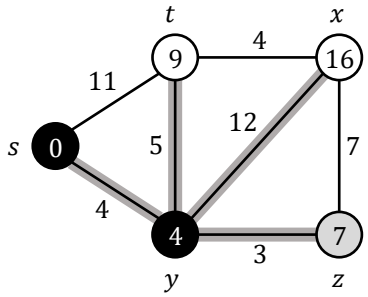
(c)

$S = \{s, y\}$

vértices $v \in V$	s	t	x	y	z
$estim[v]$	0	9	16	4	7
$antec[v]$	s	y	y	s	y

$Q =$ | t | x | − | − | − |

$u \leftarrow z$



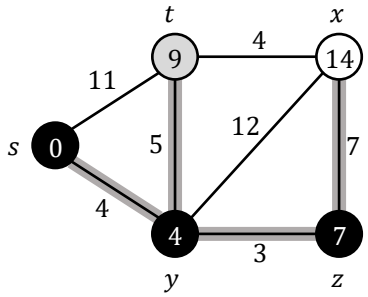
(d)

$S = \{s, y, z\}$

vértices $v \in V$	s	t	x	y	z
$estim[v]$	0	9	14	4	7
$antec[v]$	s	y	z	s	y

$Q =$ | x | − | − | − | − |

$u \leftarrow t$



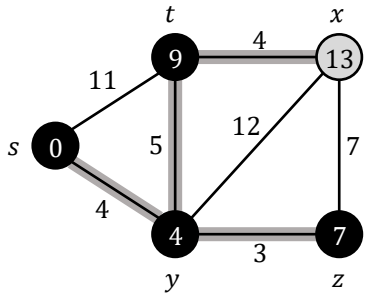
(e)

$S = \{s, y, z, t\}$

vértices $v \in V$	s	t	x	y	z
$estim[v]$	0	9	13	4	7
$antec[v]$	s	y	t	s	y

$Q =$ | − | − | − | − | − |

$u \leftarrow x$



(f)

$S = \{s, y, z, t, x\}$

vértices $v \in V$	s	t	x	y	z
$estim[v]$	0	9	13	4	7
$antec[v]$	s	y	t	s	y

$Q =$ | − | − | − | − | −

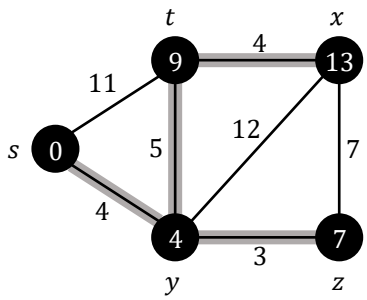


Figura 3.12 – Execução do algoritmo de Dijkstra para o exemplo de um grafo $\boldsymbol{G(V, E)}$ não direcionado. Esta figura é adaptada de Thomas H. Cormen (2001) [38] e de [39].

Através da análise dos resultados finais do exemplo (ver *Figura 3.12(f)*), é possível determinar a sequência de vértices da trajetória mais curta, ou de custo mínimo, entre o vértice inicial s e cada um dos restantes vértices (t, x, y ou z). Por exemplo, a sequência de vértices da trajetória de custo mínimo entre s e x é a seguinte: $s - y - t - x$. Esta sequência é determinada pelo antecessor, $antec$, de cada vértice, ou seja, pela tabela é possível observar que o antecessor do vértice x é t, o antecessor de t é y, e por último, o antecessor de y é s.

Um exemplo prático onde pode ser implementado este algoritmo, é a pesquisa pelo percurso mais curto entre duas cidades, sendo que os vértices de um grafo representam as cidades e as arestas representam as distâncias das estradas entre as cidades.

Os dispositivos móveis e aplicações de navegação com GPS têm este algoritmo integrado. Estes possuem mapas das estradas e utilizam o GPS para estimar a posição atual.

3.1.1.7. CURVAS PARAMÉTRICAS POLINOMIAIS

Não sendo citado na literatura como um método de planeamento da trajetória, a representação ou geração de curvas é implementado em diversas aplicações para a obtenção da melhor trajetória, sendo por isso, um elemento fundamental para o trabalho desta dissertação.

Uma curva pode ser representada por um conjunto de pontos, ou então, conectar esse conjunto de pontos com segmentos de reta (ver *Figura 3.13(a)*). Porém, esta representação pode não ser a ideal para curvas acentuadas, se o número de pontos não for suficiente para obter uma curva suave. Neste caso, é necessário obter mais pontos na proximidade dos pontos dados para a geração de uma curva [40]. Na *Figura 3.13*, os 7 pontos representados em cada curva são os pontos dados, normalmente denominados *pontos de controlo* [41].

Posto isto, com base nos pontos de controlo, o número de pontos de uma curva pode ser aumentado por interpolação ou por aproximação [40], [41]. Na *Figura 3.13(b)* é possível observar uma

nova curva obtida por interpolação e na *Figura 3.13(c)* por aproximação. É de salientar que nestas duas curvas, como os pontos que as representam são muitos, estes não são apresentados na figura, exceto os 7 pontos de controlo da curva *(b)* e os 2 (inicial e final) da curva *(c)*, que também pertencem a estas curvas.

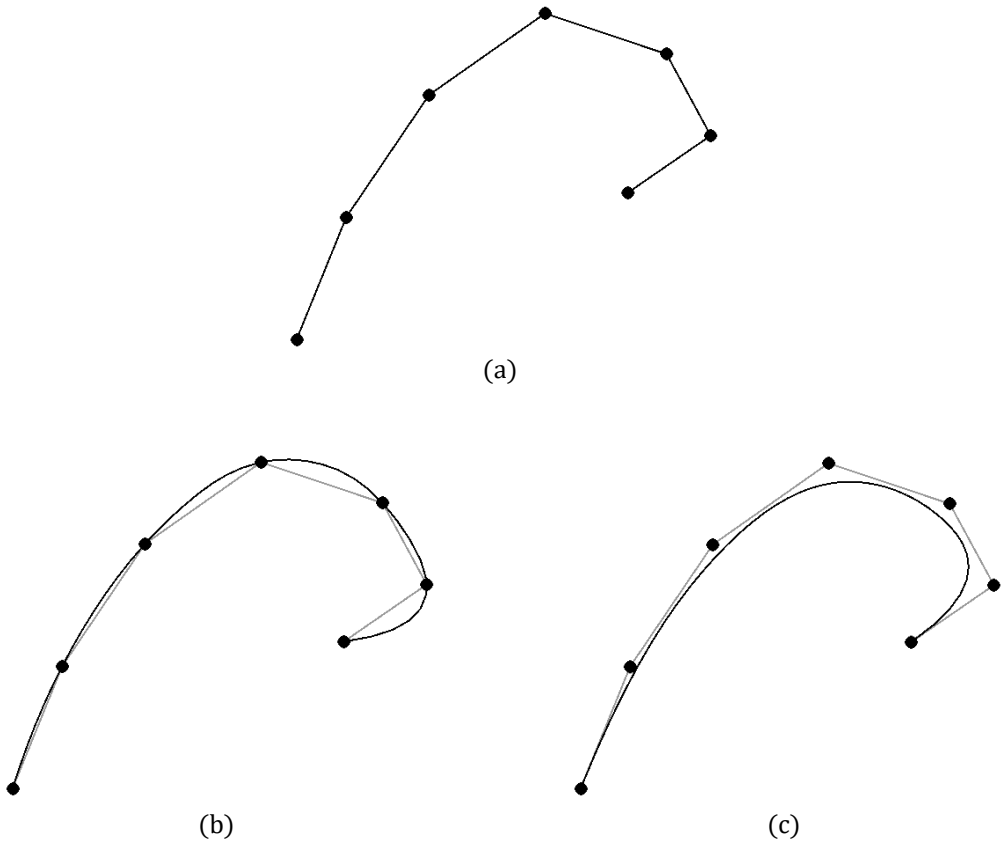


Figura 3.13 – (a) Exemplo de uma curva com 7 pontos conectados por segmentos de reta. (b) Curva obtida por interpolação dos pontos da curva (a). (c) Curva de aproximação aos pontos da curva (a). Os segmentos de reta a cinzento claro em (b) e (c) são os segmentos de reta da curva (a).

Assim, na interpolação a curva obtida passa por todos os pontos de controlo, enquanto que na aproximação a curva passa próximo dos pontos de controlo, existindo a possibilidade de não passar por nenhum desses pontos.

Uma curva obtida por interpolação ou por aproximação, é normalmente representada analiticamente na forma paramétrica através de polinómios, onde as coordenadas dos pontos da curva resultante são obtidas em função de um único parâmetro, sendo por isso, independente do sistema de coordenadas. Assim, com a representação paramétrica é possível representar curvas fechadas, ou seja, para um valor no eixo x é possível ter vários valores de y da mesma curva, tal como acontece nas curvas *(b)* e *(c)* da *Figura 3.13*.

A representação analítica de uma curva, geralmente, é dividida por partes, isto é, uma curva como a de *(b)* ou *(c)* da *Figura 3.13* pode ser constituída por uma união de pequenas curvas, pois não é possível representar analiticamente a curva completa. Esta divisão depende de alguns fatores, tais como, o número de pontos de controlo, o tipo de curva a implementar e o grau do polinómio.

Posto isto, uma curva paramétrica polinomial tem para cada dimensão ou eixo coordenado, a seguinte equação:

$$P(t) = c_0 + c_1 t + c_2 t^2 + \cdots + c_k t^k \tag{3.1}$$

Nesta equação (*Equação 3.1*), c_k são os coeficientes polinomiais, k é o grau do polinómio e t é o parâmetro supracitado, que geralmente é limitado entre 0 e 1. O grau do polinómio define se a curva é linear, quadrática, cúbica e entre outras, sendo as curvas cúbicas as mais utilizadas.

Para uma curva cúbica o grau do polinómio é igual a 3, oferecendo um resultado bastante satisfatório para a maioria das aplicações, pois curvas com o grau do polinómio inferior a 3 têm uma forma pouco flexível, enquanto que as curvas com o grau do polinómio superior a 3 têm um custo computacional superior e também podem surgir oscilações no formato da curva, não sendo por isso o ideal para a maioria das aplicações.

As curvas paramétricas polinomiais podem existir em qualquer espaço (dimensão). Como neste trabalho de dissertação as curvas são implementadas num espaço bidimensional (2D), será apresentada nas *Equações 3.2*, *3.3* e *3.4* a forma geral de uma curva paramétrica polinomial cúbica 2D, sendo um ponto da curva para um determinado t dado por $P(t) = [x(t) \quad y(t)]$.

$$\begin{aligned} x(t) &= c_{3x} t^3 + c_{2x} t^2 + c_{1x} t + c_{0x} \\ y(t) &= c_{3y} t^3 + c_{2y} t^2 + c_{1y} t + c_{0y} \end{aligned} \tag{3.2}$$

$$P(t) = T \cdot C = [t^3 \quad t^2 \quad t \quad 1] \begin{bmatrix} c_{3x} & c_{3y} \\ c_{2x} & c_{2y} \\ c_{1x} & c_{1y} \\ c_{0x} & c_{0y} \end{bmatrix} \tag{3.3}$$

A *matriz de coeficientes* C da *Equação 3.3* é definida pela *matriz base* M e pela *matriz geométrica* G, onde $C = M \cdot G$. A matriz base contém os coeficientes de alguns polinómios que definem o tipo de curva, enquanto que a matriz geométrica contém os pontos de controlo (quatro para curvas cúbicas) definidos pelo tipo de curva [40], [41], [42].

$$P(t) = T \cdot M \cdot G \tag{3.4}$$

As curvas paramétricas polinomiais mais utilizadas são as curvas de Hermite, Bézier e B-Spline, sendo a curva de Hermite, uma curva de interpolação, e a de Bézier e B-Spline são curvas de aproximação, no entanto, a curva de Bézier cúbica faz a interpolação do primeiro e último ponto de controlo.

3.1.2. CONTROLO DO MOVIMENTO

O controlo do movimento é um dos processos de navegação supracitados no atual subcapítulo *NAVEGAÇÃO AUTÓNOMA*. Este processo tem como objetivo, calcular os valores necessários para o controlo dos motores, de modo a que o robô ou veículo móvel autónomo percorra a trajetória obtida no processo *Planeamento da Trajetória*.

Normalmente, o tempo que um robô demora a percorrer uma trajetória é um fator muito importante, no entanto, no caso de um robô se mover a uma velocidade excessiva para uma determinada trajetória, ou numa parte dessa trajetória, por exemplo, uma curva fechada, este pode afastar-se da trajetória pretendida e possivelmente colidir com obstáculos.

Para um robô com movimento holonómico, tal como os robôs utilizados no trabalho desta dissertação, é necessário determinar valores para controlar os movimentos de translação e rotação, e também a direção do movimento de translação. Assim, existem vários movimentos possíveis para este tipo de robô, sendo por isso necessário determinar o melhor movimento ao longo de uma trajetória. Além disso, é possível que uma determinada tarefa exija um movimento específico, por exemplo, um robô que tem um objeto na sua posse e tem de percorrer a trajetória em segurança.

Posto isto, uma solução para o problema de controlo do movimento é implementar o controlador PID em malha fechada. Com este controlador é possível que um sistema, no caso deste trabalho, o robô ou o conjunto de sistemas que este possui, seja levado do estado atual para o estado pretendido de modo eficaz.

3.1.2.1. SISTEMA DE CONTROLO EM MALHA FECHADA

Um sistema de controlo em malha fechada, ou como também é frequentemente citado na literatura, sistema de controlo por realimentação, permite que a resposta do sistema não seja afetada por perturbações externas ou por alterações inesperadas no valor dos parâmetros do sistema [43]. Na *Figura 3.14* é possível observar o diagrama de um sistema de controlo com realimentação negativa [44].

Neste tipo de sistema de controlo, o sensor ou o conjunto de sensores medem a saída do sistema a controlar, isto é, medem a variável a controlar y. Desta forma, o controlador recebe o valor do erro, que consiste na diferença entre o valor de referência y_{ref} (valor desejado para y) e o valor medido pelo sensor y_f. Posteriormente, o algoritmo implementado para o controlador gera o valor da variável de comando u que irá para o atuador, alterando assim o estado atual do sistema caso o erro seja diferente de zero. É de salientar que o atuador no diagrama apresentado faz parte do bloco *Sistema a Controlar*.

A principal desvantagem deste sistema de controlo é na estabilidade, pois é possível existirem oscilações causadas pelo ajuste excessivo dos parâmetros do controlador.

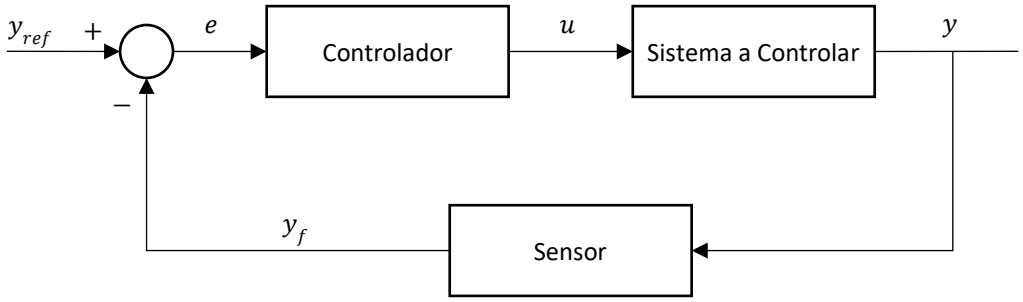


Figura 3.14 – Diagrama de um sistema de controlo com realimentação negativa.

3.1.2.2. CONTROLADOR PID

O controlador PID, aplicado em sistemas de controlo em malha fechada, é bastante utilizado na indústria e noutras aplicações, por exemplo, robôs de serviços. O algoritmo deste controlador faz uma soma de três ações: Proporcional Integral Derivativa (PID) [44]. Desta forma, o algoritmo pode ser descrito pela seguinte equação:

$$u(t) = K\left(e(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau + T_d\frac{de(t)}{dt}\right) \quad (3.5)$$

Nesta equação e tal como apresentado na secção anterior, u é a variável de comando e a variável e representa o erro. K, T_i e T_d são parâmetros, sendo K o ganho proporcional do controlador, T_i a constante de tempo da ação integral e T_d a constante de tempo da ação derivativa. De seguida, será apresentada a influência que cada uma das três ações pode ter no sistema de controlo.

Ação Proporcional. Esta ação de controlo, tal como é possível verificar pela *Equação 3.6*, é proporcional ao erro. Desta forma, quanto maior for o ganho K, mais rápida é a resposta do sistema a

controlar e o erro em regime permanente diminui. No entanto, um ganho muito elevado pode aumentar a oscilação da resposta do sistema e com isso demorar mais tempo a estabilizar no valor pretendido para este.

$$u(t) = K\, e(t) \tag{3.6}$$

Ação Integral. Esta ação permite eliminar o error em regime permanente, pois só com a ação proporcional, geralmente, não é possível eliminá-lo. Na *Equação 3.7* é possível verificar que esta ação consiste no cálculo integral do erro ao longo do tempo. Isso leva a que o valor do erro seja somado. Desta forma, enquanto o valor do erro for diferente de zero, o valor da ação integral irá aumentar ao longo do tempo, até que o erro em regime permanente seja zero. A constante de tempo T_i tem impacto na estabilidade e no tempo de resposta do sistema. Assim, para um valor baixo desta constante, mais rápida será a resposta. No entanto, como desvantagem, a oscilação aumenta e com isso o tempo que o sistema demora a estabilizar no valor de referência pretendido também aumenta. Por outro lado, quando a constante tende para um valor elevado, a resposta será mais lenta e com menos oscilação.

$$u(t) = K\left(\frac{1}{T_i}\int_0^t e(\tau)d\tau\right) \tag{3.7}$$

A ação integral, na prática, não é implementada isoladamente, sendo acompanhada pela ação proporcional que dá origem a um controlador PI, ou então pode ser adicionada também a ação derivativa, para criar um controlador PID. Para o controlador PI a variável de comando tem a seguinte equação:

$$u(t) = K\left(e(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau\right) \tag{3.8}$$

Ação Derivativa. Esta ação melhora a estabilidade e diminui o tempo de resposta do sistema. Na *Equação 3.9* é possível verificar que esta ação consiste no cálculo da derivada do erro ao longo do tempo. Desta forma, a variável de comando tem um controlo mais previsível, sendo mais rápida a resposta do sistema para uma alteração no valor desta variável. A constante de tempo T_d tem impacto no ajuste desta ação de controlo. Assim, para um valor elevado desta constante, mais rápida será a resposta do sistema para uma alteração no valor da variável de comando. No entanto, como desvantagem, o sistema pode ficar instável se o valor do erro oscilar muito num curto espaço de tempo.

$$u(t) = K\left(T_d \frac{de(t)}{dt}\right) \tag{3.9}$$

Tal como a ação integral, na prática, a ação derivativa não é implementada isoladamente, sendo acompanhada pela ação proporcional que dá origem a um controlador PD, ou então adicionar ainda a ação integral, para criar um controlador PID.

3.1.2.3. CONTROLADOR PID DIGITAL

Na secção anterior foi apresentado o controlador PID analógico ou contínuo, tendo sido este algoritmo implementado inicialmente para o controlo de diversos sistemas. Contudo, com o desenvolvimento dos microprocessadores e microcontroladores, onde sistemas conectados a estes funcionam por amostragem, surgiu a necessidade de adaptar este algoritmo para a forma discreta. Posto isto, o algoritmo para o controlador PID na forma discreta ou digital, pode ser descrito pela seguinte equação:

$$u(t_n) = P(t_n) + I(t_n) + D(t_n) \tag{3.10}$$

Nesta equação, t_n representa o momento em que é feita a amostragem, sendo apresentadas nas *Equações 3.11*, *3.12* e *3.14* as três ações deste controlador na forma discreta. Assim, a ação proporcional pode ser descrita pela *Equação 3.11*, sendo que K_p, denominado *ganho proporcional*, é equivalente ao K apresentado no PID contínuo.

$$P(t_n) = K_p\, e(t_n) \tag{3.11}$$

Na *Equação 3.12* é possível verificar a ação integral na forma discreta, sendo que esta equação recursiva consiste em multiplicar o erro no instante de tempo t_n, pelo ganho integral K_i e pelo tempo T entre amostras, ou seja, o tempo entre t_{n-1} e t_n. Ao valor desta multiplicação é somado o valor da ação integral obtido em t_{n-1}.

$$I(t_n) = I(t_{n-1}) + K_i\, T\, e(t_n) \tag{3.12}$$

O ganho integral é descrito pela seguinte equação:

$$K_i = \frac{K}{T_i} \tag{3.13}$$

A ação derivativa na forma discreta é dada pela *Equação 3.14*. Tal como é possível verificar, o ganho derivativo K_d é multiplicado pelo valor da diferença entre o erro nos instantes t_n e t_{n-1}. O valor desta multiplicação é dividido pelo tempo T entre amostras.

$$D(t_n) = K_d \frac{e(t_n) - e(t_{n-1})}{T} \tag{3.14}$$

O ganho derivativo é descrito pela seguinte equação:

$$K_d = K\, T_d \tag{3.15}$$

3.2. ROBOT OPERATING SYSTEM

O Robot Operating System (ROS) é uma *framework open-source* que atualmente é bastante utilizada na área da robótica [45]. Qualquer aplicação robótica engloba um conjunto de diferentes tarefas. O trabalho em equipa e o desenvolvimento de módulos individuais para cada uma das tarefas, pode tornar-se num processo muito difícil, devido à complexidade de algumas aplicações robóticas. Foi por este motivo que surgiu o ROS, que contribui significativamente para o desenvolvimento de plataformas robóticas, pois tem a vantagem de simplificar a comunicação entre módulos e desenvolver *software* robusto.

O objetivo do ROS, é incentivar o desenvolvimento colaborativo de *software* para aplicações robóticas [45]. Por exemplo, um laboratório tem um grupo de programadores que são especialistas em mapeamento de ambientes e contribuem com o *software* para ser integrado num robô de auxílio em tarefas hospitalares. Outro grupo domina visão por computador e desenvolve um *software* para a deteção de obstáculos para o mesmo robô. Desta forma, o ROS promove a reutilização de *software*, ou seja, os módulos de *software* desenvolvidos para um robô de auxílio em tarefas hospitalares podem ser integrados em diferentes plataformas robóticas. Com este conceito de reutilização e com ajuda de muitos colaboradores, são criadas bibliotecas em áreas como a navegação, visão por computador, simulação, perceção e controlo.

O ROS foi desenvolvido em diversas instituições e para diversos robôs. No ano 2000, a Universidade de Stanford, com dois projetos envolvendo inteligência artificial, criou sistemas de *software* dinâmicos com a intenção de os usar em robôs. Em 2007, o laboratório de investigação robótica Willow

Garage, deu continuidade ao desenvolvimento do ROS, com a criação de pacotes de *software* já testados em robôs [45].

Apesar do seu nome, o Robot Operating System (ROS) não é um sistema operativo convencional, mas sim uma *framework*. Contudo, este fornece alguns serviços de um sistema operativo [46], que são: a abstração de *hardware*, comunicação entre processos, controlo de baixo nível e gestão de pacotes.

3.2.1. ARQUITETURA ROS

A arquitetura do ROS é baseada em nós, sendo que um nó é um processo que executa uma tarefa para a qual foi criado. Uma aplicação robótica que é constituída por um conjunto de tarefas e usa o ROS, contém um conjunto de nós. Esta abordagem permite: (a) um *software* modular, (b) se existir algum problema com um dos nós ou "desativar" propositadamente um nó, o programa pode continuar a ser executado, pois os restantes nós são individuais e (c) a complexidade do *software* é menor em comparação com outras estruturas.

O ROS cria internamente uma rede, em que os nós estão conectados diretamente a outros nós. A rede é configurada pelo ROS *Master*, que permite aos nós se localizarem entre eles. Sem o *Master*, não é possível a comunicação entre os nós. O protocolo normalmente usado para a comunicação é chamado TCPROS, que assegura a gestão de mensagens e serviços ROS. Este protocolo usa *sockets* TCP/IP.

A comunicação entre os nós pode ser feita por duas formas distintas: mensagens e serviços. Uma mensagem é uma estrutura de dados, que contém uma ou mais "variáveis" de diferentes tipos de dados (char, int, float, bool, entre outros). Para a gestão da troca de mensagens entre os nós, existem os tópicos. Os tópicos oferecem uma comunicação unidirecional, onde os nós podem publicar ou subscrever as mensagens. Isto é, um nó se enviar uma mensagem, está a publicar no tópico, por outro lado, se um nó subscrever um ou mais tópicos, recebe as mensagens que circulam no(s) tópico(s) que subscreveu. É também de salientar que vários nós podem publicar e subscrever num único tópico, e um nó pode publicar em vários tópicos.

Os serviços oferecem uma comunicação bidirecional entre dois nós para a troca de mensagens, usando o paradigma pedido/resposta entre cliente/servidor. Este tipo de comunicação funciona da seguinte forma: o nó cliente envia um pedido ao nó servidor através de uma mensagem e aguarda pela resposta do nó servidor, que irá enviar uma ou mais mensagens. As mensagens trocadas entre os nós, são definidas na configuração do serviço. Na *Figura 3.15* é possível observar a arquitetura de um exemplo de uma rede ROS.

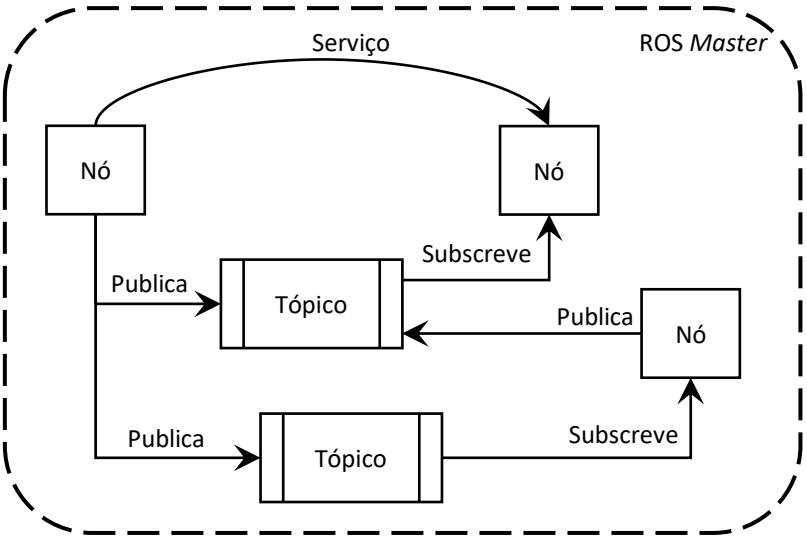


Figura 3.15 – Exemplo de uma rede ROS.

Capítulo 4

MINHO TEAM

Conforme mencionado na secção *1.1.3* do *Capítulo 1*, a equipa Minho Team tem participado ativamente na Middle Size League (MSL), em eventos nacionais e internacionais. A primeira participação no RoboCup foi em 1999, sendo uma das primeiras equipas a surgir nesta prova, contribuindo ao longo destes anos para a evolução do estado da arte desta competição. A equipa desde que iniciou a sua atividade, em 1998, tem desenvolvido várias versões de robôs, de modo a acompanhar a constante evolução tecnológica e, consequentemente, melhorar o desempenho para competir ao mais alto nível na MSL.

A plataforma anterior, desenvolvida em 2011, não tinha a estrutura mecânica muito estável e também existiam alguns problemas com o *hardware*, que não era muito eficiente e tinha algumas falhas intermitentes. Por isso, a equipa decidiu reconstruir e melhorar a plataforma, sendo que todo o processo de reconstrução foi desenvolvido no Laboratório de Automação e Robótica (LAR) da Universidade do Minho [47]. Foi também desenvolvido todo o *software* e implementado novos algoritmos.

Neste capítulo será apresentada a estrutura mecânica da plataforma após a sua reconstrução. Também será apresentada a nova arquitetura de *hardware*, sendo que alguns componentes da plataforma anterior foram reaproveitados. Posteriormente, será feita uma breve descrição da arquitetura de *software* implementada nos robôs, e por último, será apresentado o simulador da Minho Team.

Os subcapítulos *ESTRUTURA MECÂNICA*, *HARDWARE* e *ARQUITETURA DE SOFTWARE* serão descritos apenas para um único robô, uma vez que os outros são uma cópia dele, exceto o guarda-redes que tem algumas partes diferentes.

4.1. ESTRUTURA MECÂNICA

Na MSL existem regras para as dimensões, peso e cor dos robôs, porém não é exigido qualquer formato para a estrutura mecânica. Por este motivo, a estrutura desenvolvida por cada uma das equipas é variável, sendo as formas circular, triangular e quadrada as mais usadas para a base dos robôs. Na Minho Team a base da plataforma atual é circular com 50 cm de diâmetro, que desta forma cumpre o limite máximo de 52 cm × 52 cm determinado pelas regras. É na base que estão incorporados os

atuadores, tais como: motores para a locomoção, chuto eletromagnético e o sistema de manipulação de bola ou *dribbler*.

A manipulação da bola e o chuto são ações executadas por dois sistemas independentes. De seguida, será apresentada a mecânica, ou mecanismos, destes dois sistemas, sendo que na secção seguinte será apresentado o *hardware* incluído nestes mesmos sistemas.

O sistema de manipulação de bola tem como objetivos reter e driblar a bola de forma a mantê-la na posse do robô (ver *Figura 4.1(g)*) e, ao mesmo tempo, exercer movimento na bola, respeitando assim a regra que não permite o bloqueio da bola quando esta se encontra na posse do robô. O mecanismo e o *hardware* deste sistema, integrados na plataforma anterior, foram substituídos, permitindo ao robô mover-se em qualquer direção com uma qualquer orientação sem perder a bola. O novo mecanismo (ver *Figura 4.1(f)*) é um protótipo com uma estrutura idêntica a outras equipas, sendo constituído por dois braços de controlo e duas rodas com dois pneus de borracha que oferecem a fricção necessária para reter a bola.

O sistema de chuto é fundamental para efetuar o passe e a marcação de golos. Na plataforma atual, este sistema contém um mecanismo em forma de alavanca (ver *Figura 4.1(e)*) que impulsiona a bola, sendo esta alavanca impulsionada com pouca força para efetuar um passe ou com bastante força para fazer a bola descrever uma trajetória parabólica. Na secção *4.2.1.1 CHUTO ELETROMAGNÉTICO* será apresentado todo o *hardware* responsável pelo impulso na alavanca.

No que diz respeito ao sistema de visão utilizado pelos robôs da Minho Team, exceto o guarda-redes, este é semelhante a todas as outras equipas, sendo um sistema de visão omnidirecional catadióptrico (ver *Figura 4.1(b)*), que consiste na utilização de uma câmara a apontar para o centro de um espelho convexo [48]. Para adquirir uma imagem com uma qualidade razoável, sem deformações, é necessário um suporte capaz de ajustar a distância entre a câmara e o espelho, como também ajustar o alinhamento entre o centro da câmara e o centro do espelho. No trabalho desenvolvido pelo elemento da equipa André Pereira, no sistema de visão, este optou por desenvolver um novo suporte, uma vez que o da plataforma anterior não oferecia a possibilidade de ajustar a distância pretendida e também não era possível ajustar o centro da câmara com o espelho.

Para obter a maior área de visão possível, o suporte com a câmara e o espelho é fixado a uma torre instalada no centro da base do robô que, desta forma, posiciona o sistema de visão no topo do robô. Porém, a altura a que este fica é limitada pela altura máxima da plataforma, que de acordo com as regras é de 80 cm.

A estabilidade da torre instalada no centro do robô, também foi melhorada com a fixação de quatro barras de aço que unem a placa superior da base à torre (ver *Figura 4.1(c)*). No exterior destas barras foi fixada uma cobertura de plástico (ver *Figura 4.1(d)*) para proteger todo o *hardware* situado nesta zona do robô.

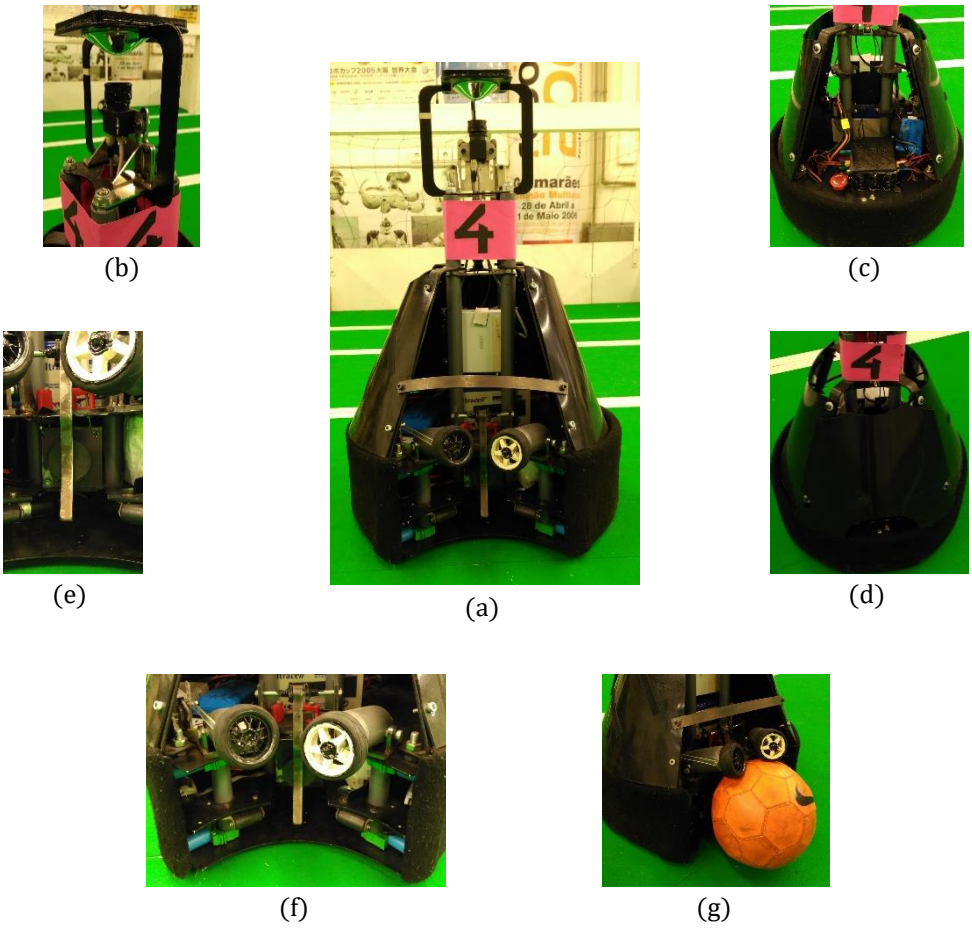


Figura 4.1 – (a) Robô da Minho Team (plataforma atual). (b) Sistema de visão. (c) Torre no centro da base do robô, barras de aço e algum *hardware*. (d) Proteção com cobertura de plástico. (e) Alavanca do sistema de chuto. (f) Mecanismo do sistema de manipulação de bola. (g) Robô com a posse de bola.

4.2. *HARDWARE*

Para desenvolver a nova arquitetura de *hardware* implementada nos robôs, foram considerados alguns fatores, tais como, a eficiência, a estabilidade e o orçamento.

Nesta nova arquitetura de *hardware*, cada robô possui uma unidade de processamento principal, o mini-PC MSI Cubi com o processador Intel Pentium 3805U. Este mini-PC faz o processamento dos algoritmos de localização (visão por computador e fusão sensorial), do planeamento da trajetória e de controlo. Este também é responsável pela comunicação com a rede externa através do *wireless*, de forma

a enviar e a receber informação da *Base Station* e de qualquer outro robô da equipa. A captura de imagem é feita pela câmara BlackFly PGE 13S2C, posicionada no topo do robô e conectada ao barramento Gigabit Ethernet do mini-PC.

Alguns algoritmos processados no mini-PC necessitam da informação sensorial, e no final de cada ciclo do processamento é necessário enviar os comandos de ação de alto nível para o sistema de controlo de cada atuador, ou seja, para o baixo nível. Desta forma, um sistema de controlo principal do baixo nível, que está conectado ao mini-PC via USB, é responsável por enviar a informação dos sensores para o mini-PC e receber do mini-PC os comandos de ação de alto nível para os interpretar e, de seguida, enviar os comandos específicos para o sistema de controlo de cada atuador.

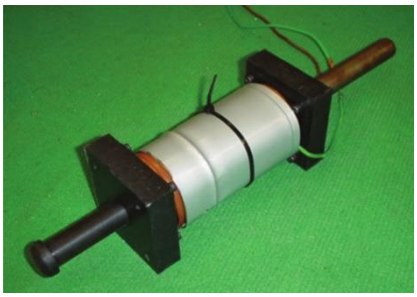
O sistema de controlo principal do baixo nível foi desenvolvido pelo elemento da equipa Hélder Ribeiro, que inclui uma placa com o microcontrolador ATmega2560, conhecida por Arduino Mega. Esta placa será substituída por um novo sistema que inclui a RaspberryPi, sendo que o desenvolvimento e teste deste novo sistema está a cargo do elemento da equipa, Pedro Osório.

4.2.1. ATUADORES

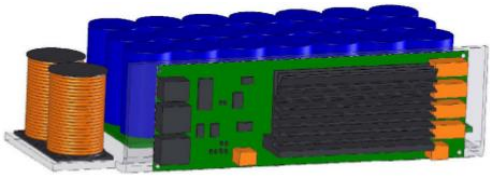
Os atuadores presentes em cada robô são: chuto eletromagnético, três motores para a locomoção e o sistema de manipulação de bola ou *dribbler*. De seguida, será feita uma breve descrição de cada atuador e do respetivo sistema de controlo.

4.2.1.1. CHUTO ELETROMAGNÉTICO

O chuto eletromagnético, sem incluir o mecanismo em forma de alavanca que impulsiona a bola, é constituído por uma bobina eletromagnética, uma placa controladora, uma bateria de condensadores com um total de $4.3\ mF$ e $441\ V$ de tensão máxima, e duas bobinas com $3.8\ mH$ de indutância conjunta, usadas para recarregar a bateria de condensadores. Na *Figura 4.2(a)* é possível observar uma bobina eletromagnética com um núcleo móvel constituído por uma haste, sendo metade da haste em ferro e a outra metade em nylon. Quando uma corrente elétrica percorre a bobina eletromagnética, a haste desloca-se para o lado do nylon, que impulsiona a alavanca que está em contacto com a bola. A força aplicada pela haste na alavanca é proporcional à quantidade de energia que percorre a bobina. Esta quantidade de energia, armazenada na bateria de condensadores, é controlada por uma placa desenvolvida por um ex-elemento da equipa, que contém circuitos de controlo e de potência [49]. Assim, com esta placa é possível controlar a duração da descarga de energia para a bobina. Na *Figura 4.2(b)* é possível observar um módulo em CAD com a bateria de condensadores, as bobinas de recarga e a placa controladora.



(a)



(b)

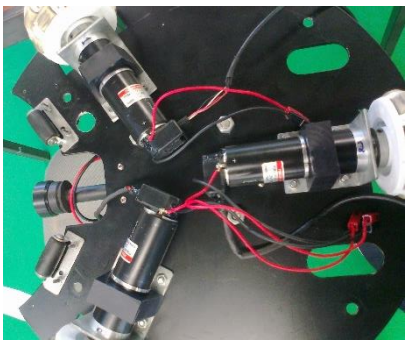
Figura 4.2 – Chuto eletromagnético. (a) Bobina eletromagnética [12]. (b) Representação CAD da bateria de condensadores (cilindros azuis), das bobinas de recarga (os dois cilindros à esquerda) e da placa controladora (a verde) [11].

4.2.1.2. MOTORES PARA A LOCOMOÇÃO DO ROBÔ

Para a locomoção de cada robô estão dispostos na parte inferior da base, três motores Maxon de 150 W e 24 V DC, com uma caixa redutora e com um encoder acoplados a cada motor (ver *Figura 4.3(a)*). Os três motores estão fixados à base com um desfasamento de 120° entre si e com a mesma distância ao centro da base (ver *Figura 4.3(c)*).

(a)

(b)



(c)

Figura 4.3 – (a) Motor Maxon de 150 W e 24 V DC, com caixa redutora e encoder. (b) Roda omnidirecional. (c) Disposição dos três motores na parte inferior da base da plataforma.

No eixo de cada motor, ou mais especificamente, no eixo de cada caixa redutora, está acoplado uma roda omnidirecional (ver *Figura 4.3(b)*) que garante um movimento holonómico (ou omnidirecional). Este tipo de movimento não apresenta qualquer restrição, ou seja, o robô ou plataforma pode mover-se

em qualquer direção com uma qualquer orientação, sendo possível movimentos de translação e rotação em simultâneo.

Para o controlo dos três motores é usada a placa OMNI3D-MAX (ver *Figura 4.4*), desenvolvida pela empresa SAR (Soluções de Automação e Robótica) [50]. A OMNI3D-MAX é capaz de controlar de forma independente três motores DC de 24 V e até 50 A de corrente por motor. A comunicação entre esta placa e o sistema de controlo principal do baixo nível é efetuada via barramento I2C.

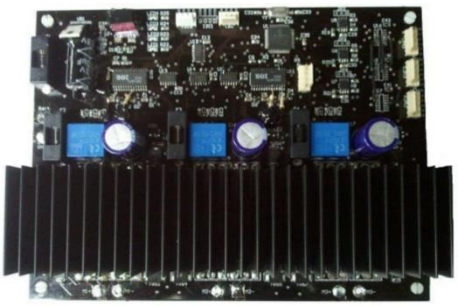


Figura 4.4 – Placa OMNI3D-MAX para o controlo dos três motores.

Para simplificar a interação do utilizador com a placa OMNI3D-MAX, é disponibilizada uma biblioteca *open-source* para programação em C e direcionada para o Arduino IDE. Esta biblioteca contém funções de configuração, de leitura e de movimentação.

A OMNI3D-MAX permite a movimentação dos motores em vários modos, em particular:

- Movimentação omnidirecional de três motores com ou sem controlo PID;
- Movimentação linear de três motores com ou sem controlo PID;
- Movimentação posicional de três motores com ou sem controlo PID;

Nos robôs da Minho Team, é implementado o modo de movimentação omnidirecional de três motores com controlo PID. Neste modo, o controlo é em malha fechada com controlador PID, sendo necessária a leitura do valor da contagem incremental dos encoders dos três motores. Na biblioteca disponível existe uma função com três parâmetros, que envia uma ordem de movimentação omnidirecional. Estes três parâmetros são: velocidade linear, velocidade angular e a direção do movimento de translação. Quando os parâmetros são enviados, o tempo para a execução da ordem de movimentação é de 25 ms, 50 ms ou 100 ms, dependendo dos valores dos ganhos proporcional, integral e derivativo do controlador PID, que são configuráveis pelo utilizador. Também é possível configurar a rampa de aceleração através de um parâmetro que define a inclinação da rampa, permitindo o ajuste para um arranque suave e que minimize o deslizamento das rodas.

4.2.1.3. SISTEMA DE MANIPULAÇÃO DE BOLA OU *DRIBBLER*

O sistema de manipulação de bola ou *dribbler*, tal como já foi mencionado no subcapítulo *4.1 ESTRUTURA MECÂNICA*, é constituído por dois braços de controlo e duas rodas com dois pneus de borracha que oferecem a fricção necessária para reter a bola. A rotação das rodas para exercer movimento na bola, é efetuada por dois motores de 12 V DC, um em cada braço de controlo, sendo que cada roda está acoplada diretamente ao eixo do respetivo motor. No que diz respeito ao controlo dos dois motores, este é um controlo on/off efetuado por um simples circuito de controlo e de potência.

A deteção da bola no *dribbler*, ou seja, a informação que o robô recebe para saber se tem ou não a posse de bola, é garantida por um potenciómetro adaptado num dos braços, sendo enviado o sinal analógico para um dos pinos do ADC do sistema de controlo principal do baixo nível.

4.2.2. SENSORES

Os sensores presentes em cada robô são: IMU, *Inertial Measurement Unit*, de 10 DOF da Adafruit, e os já mencionados, câmara, três encoders e o potenciómetro do *dribbler* para a deteção da bola. A comunicação entre o IMU e o sistema de controlo principal do baixo nível é efetuada via barramento I2C.

4.3. ARQUITETURA DE *SOFTWARE*

Tal como já foi mencionado no *Capítulo 1*, na MSL cada um dos robôs tem de jogar autonomamente, partilhando apenas informação em tempo real entre os elementos da equipa e também com um computador externo denominado de *Base Station*, que atua como um treinador. Para que tal seja possível, o *software* de um robô da MSL tem de incluir determinados processos, que são normalmente usados em todos os robôs móveis autónomos. Esses processos são baseados em algoritmos de visão por computador, inteligência artificial, gestão das comunicações, planeamento da trajetória e controlo.

Desta forma, o *software* de um robô da Minho Team é constituído pelos seguintes processos:

- **Controlo do *Hardware***: controlar todos os atuadores e adquirir leituras dos sensores, exceto da câmara;
- **Localização**: com recurso ao sistema de visão e da fusão sensorial, este processo: (a) faz a representação do estado do mundo, isto é, obter um mapeamento dos obstáculos e da bola, e (b) determina a posição e orientação do robô no espaço de jogo;

- **Comunicação**: responsável pelo envio e receção de informação entre a rede externa e a rede interna do robô;
- **Inteligência Artificial**: atribui papeis aos robôs (defesa, avançado e entre outros), com base na tática de jogo. O papel atribuído define a posição pretendida para o robô, que pode ser a posição atual ou outra posição. Este processo também é responsável pela decisão do passe e remate;
- **Planeamento da Trajetória**: determina uma trajetória entre uma posição inicial e uma posição pretendida no espaço de configuração livre (espaço sem obstáculos);
- **Controlo do Movimento**: calcula os valores necessários para o controlo dos motores.

Estes processos partilham informação entre si, sendo que a comunicação entre todos os processos é efetuada pelo ROS. Na *Figura 4.5* é possível observar a arquitetura da rede ROS implementada em cada robô. É de salientar que o nó ***Controlo*** é responsável pelos dois processos: ***Planeamento da Trajetória*** e ***Controlo do Movimento***.

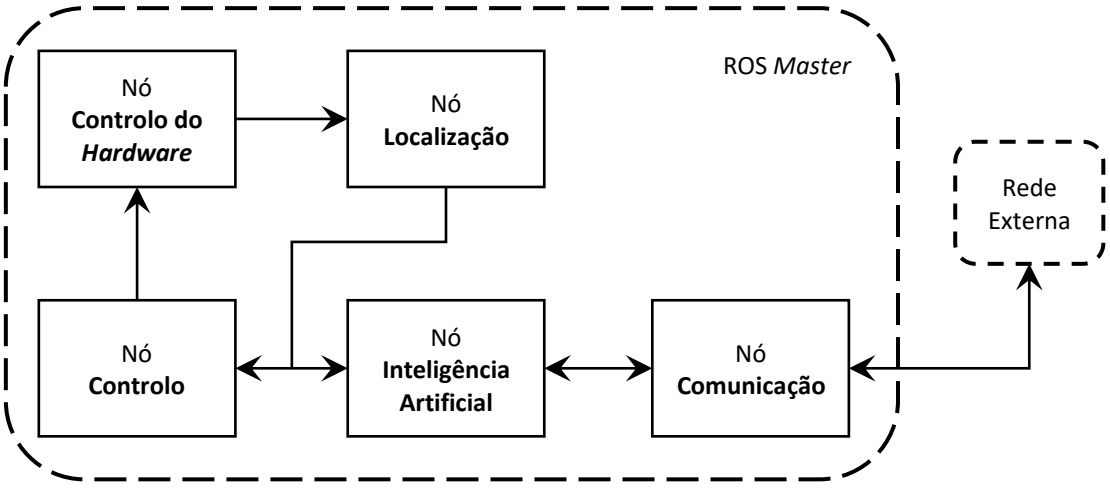


Figura 4.5 - Arquitetura da rede ROS de um robô da Minho Team.

4.4. SIMULADOR

O simulador da Minho Team, denominado *Minho Simulator*, foi desenvolvido pelo elemento da equipa, Pedro Osório. Este simulador contém um conjunto de ferramentas que permitem simular algumas competições do RoboCup, sendo obviamente a simulação para a MSL a que interessa para o trabalho desta dissertação.

A interface gráfica do *Minho Simulator* foi criada em Qt, sendo o Gazebo responsável pelos cálculos de toda a física necessária numa simulação. O simulador funciona numa arquitetura cliente-servidor,

sendo o cliente um *widget* Qt responsável por fazer a interface entre o Qt e o Gazebo (servidor), isto de forma a possibilitar a interação do utilizador com o espaço simulado e oferecer uma melhor performance. O ambiente e toda a dinâmica de simulação é a três dimensões.

Na *Figura 4.6* é possível observar a interface gráfica do simulador para a MSL e o espaço de jogo em duas perspetivas diferentes. Através de movimentos e cliques do rato, é possível ao utilizador alterar a perspetiva de visualização do espaço e manipular os modelos (robôs, bola, entre outros) existentes no espaço de simulação.

Figura 4.6 – Interface gráfica do simulador para a MSL e o espaço de jogo em duas perspetivas diferentes.

Capítulo 5

IMPLEMENTAÇÃO

Este capítulo descreve todos os detalhes da implementação deste trabalho de dissertação. O primeiro subcapítulo *5.1 PLANEAMENTO DA TRAJETÓRIA*, descreve todos os passos para obter a melhor trajetória, sendo que o segundo subcapítulo *5.2 CONTROLO DO MOVIMENTO*, descreve todo o processo para obter os valores a enviar para a placa de controlo dos motores.

Antes de iniciar a descrição dos dois processos supracitados, será apresentada a linguagem de programação usada, as ferramentas de apoio para o desenvolvimento deste trabalho, a visão geral do nó *Controlo* da rede ROS de um robô da Minho Team, as bibliotecas utilizadas no desenvolvimento deste trabalho, e para finalizar será apresentada a visão geral do sistema de controlo de movimento, desenvolvido neste trabalho.

Linguagem de Programação. O *software* desenvolvido no trabalho desta dissertação, foi implementado com a linguagem de programação C++, tal como o *software* dos restantes processos (exceto *Controlo do Hardware*), que constituem a arquitetura de *software* de um robô da Minho Team. A ferramenta desenvolvida para configuração, que será apresentada posteriormente neste capítulo, foi desenvolvida em Qt5.

Ferramenta de Simulação. Para testar todo o *software* foi utilizado, inicialmente, o simulador da Minho Team, mencionado no subcapítulo *4.4 SIMULADOR*. Esta ferramenta tem várias vantagens em relação aos robôs reais, uma delas é o tempo, pois é possível testar várias vezes em diversas situações num curto espaço de tempo. Outra vantagem é a possibilidade de utilizar menos vezes as baterias dos robôs e assim aumentar a vida útil destas. Pelo facto de a equipa não ter baterias suplentes, dificulta o teste com os robôs reais, sendo este, outro motivo, para a utilização do simulador.

O *software* usado num robô real, para ser testado no simulador, exige apenas a calibração de alguns parâmetros que serão mencionados neste capítulo.

Ferramenta de Configuração. Esta ferramenta foi desenvolvida para ser utilizada no computador pessoal dos elementos da equipa, pois o ROS permite a comunicação entre diferentes computadores

que estejam na mesma rede ROS, ou seja, é possível ter uma rede ROS com vários nós em diferentes computadores, desde que todos os nós comuniquem com o *Master* dessa rede. Desta forma, esta ferramenta de configuração é um nó que comunica com o *Master* da rede ROS de um robô real ou de um robô simulado, tendo cada robô real ou simulado uma rede ROS e o respetivo *Master*.

Relativamente às funcionalidades desta ferramenta de configuração (ver *Figura 5.1*), esta permite para cada robô, ajustar os ganhos dos controladores PID das velocidades linear e angular, e ajustar as velocidades máximas linear e angular. Para efetuar alguns testes é possível definir a posição e orientação pretendida para o robô, definir a posição (x, y) pretendida para a bola, selecionar entre fazer um passe ou um remate, e definir a força aplicada pelo chuto na bola. Por último, é disponibilizado uma série de botões, que permitem selecionar a informação a enviar do nó *Controlo* para a ferramenta de visualização, que será apresentada de seguida.

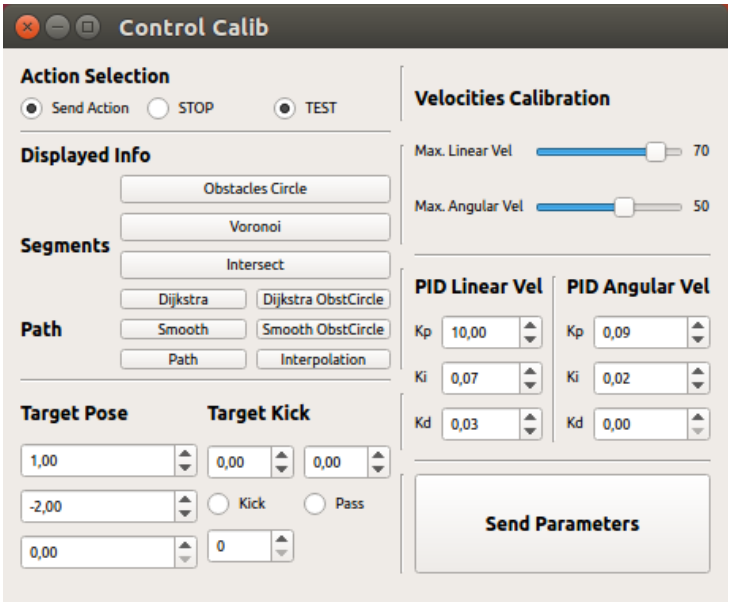


Figura 5.1 – Ferramenta de configuração.

Ferramenta de Visualização. Esta ferramenta, denominada *visualizer*, permite visualizar em 2D toda a informação adquirida por um robô do espaço de jogo, tal como, a sua posição e orientação, posição da bola, posições de possíveis obstáculos e entre outras informações. Na *Figura 5.2* é possível observar o *visualizer*, onde o robô é representado pela forma com a cor ciano, os obstáculos pelos círculos a preto e a bola pelo círculo ou ponto laranja.

O *visualizer* foi também uma ferramenta fundamental para o desenvolvimento do processo de planeamento da trajetória, pois permite a visualização de todos os passos deste processo. Tal como a ferramenta de configuração, esta também é um nó ROS.

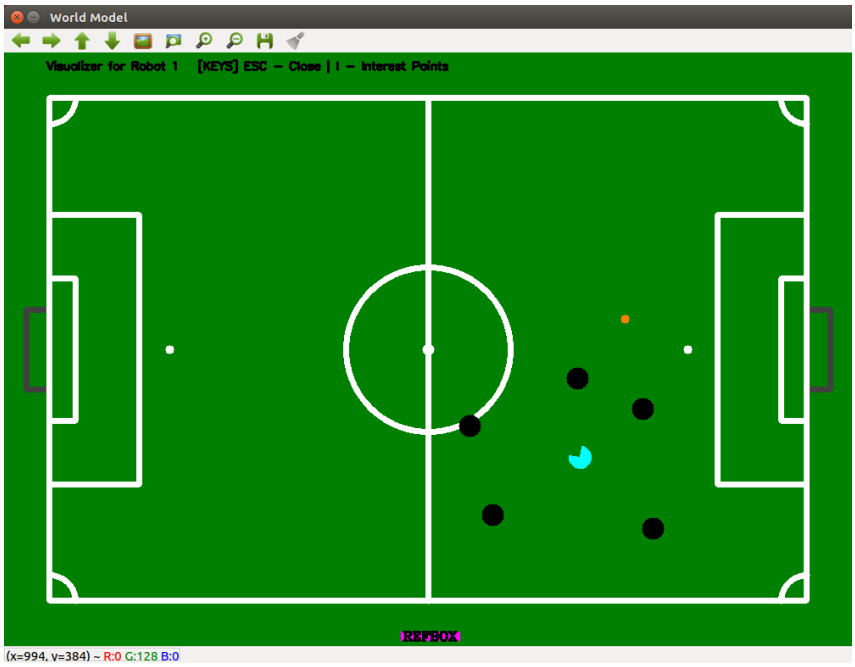


Figura 5.2 - Ferramenta de visualização ou *visualizer*.

Nó *Controlo*. O nó *Controlo* é responsável pelos dois processos *Planeamento da Trajetória* e *Controlo do Movimento*. Este nó, tal como os outros nós da rede ROS de um robô da Minho Team, subscreve e publica mensagens através de tópicos ROS, e envia/recebe mensagens através de serviços ROS. Uma explicação mais detalhada sobre o ROS é apresentada no subcapítulo *3.2*. Na *Figura 5.3* é possível observar todas as mensagens que o nó *Controlo* publica e subscreve, e que envia/recebe.

Agora é apresentada uma breve descrição de cada mensagem:

- **robotInfo**: contém informação sobre o espaço de jogo, tal como a posição (x, y) e orientação do robô, velocidade do robô e da bola, posições de possíveis obstáculos, posição da bola, uma variável que indica se o robô vê a bola e outra variável que indica se o robô tem a bola;
- **controlInfo**: o nó *Controlo* é o único nó a publicar esta mensagem, que contém os valores para as velocidades linear e angular e a direção do movimento de translação do robô, sendo estes valores obtidos no processo *Controlo do Movimento*. Esta mensagem contém também uma variável para o controlo on/off do *dribbler*;
- **controlConfig**: esta é a única mensagem que o nó *Controlo* recebe através de um serviço ROS, sendo enviada pelo nó responsável pela ferramenta de configuração. A mensagem contém os valores dos ganhos dos controladores PID para as velocidades linear e angular, os valores das velocidades máximas linear e angular, e um conjunto de variáveis que permitem selecionar a informação a enviar do nó *Controlo* para o *visualizer*. O nó da ferramenta de configuração

também recebe uma mensagem, que é enviada apenas uma vez pelo nó *Controlo* no momento da inicialização do programa, sendo que esta mensagem contém os valores dos ganhos e das velocidades máximas, que se encontram guardados num ficheiro. Quando é feita alguma alteração destes valores pela ferramenta de configuração, estes são guardados/atualizados no ficheiro;

- **pathData**: contém a informação obtida nos passos do processo *Planeamento da Trajetória*, sendo esta mensagem publicada apenas pelo nó *Controlo* e subscrita apenas pelo *visualizer*, isto é, o nó desta ferramenta de visualização é o único nó a subscrever o tópico desta mensagem;

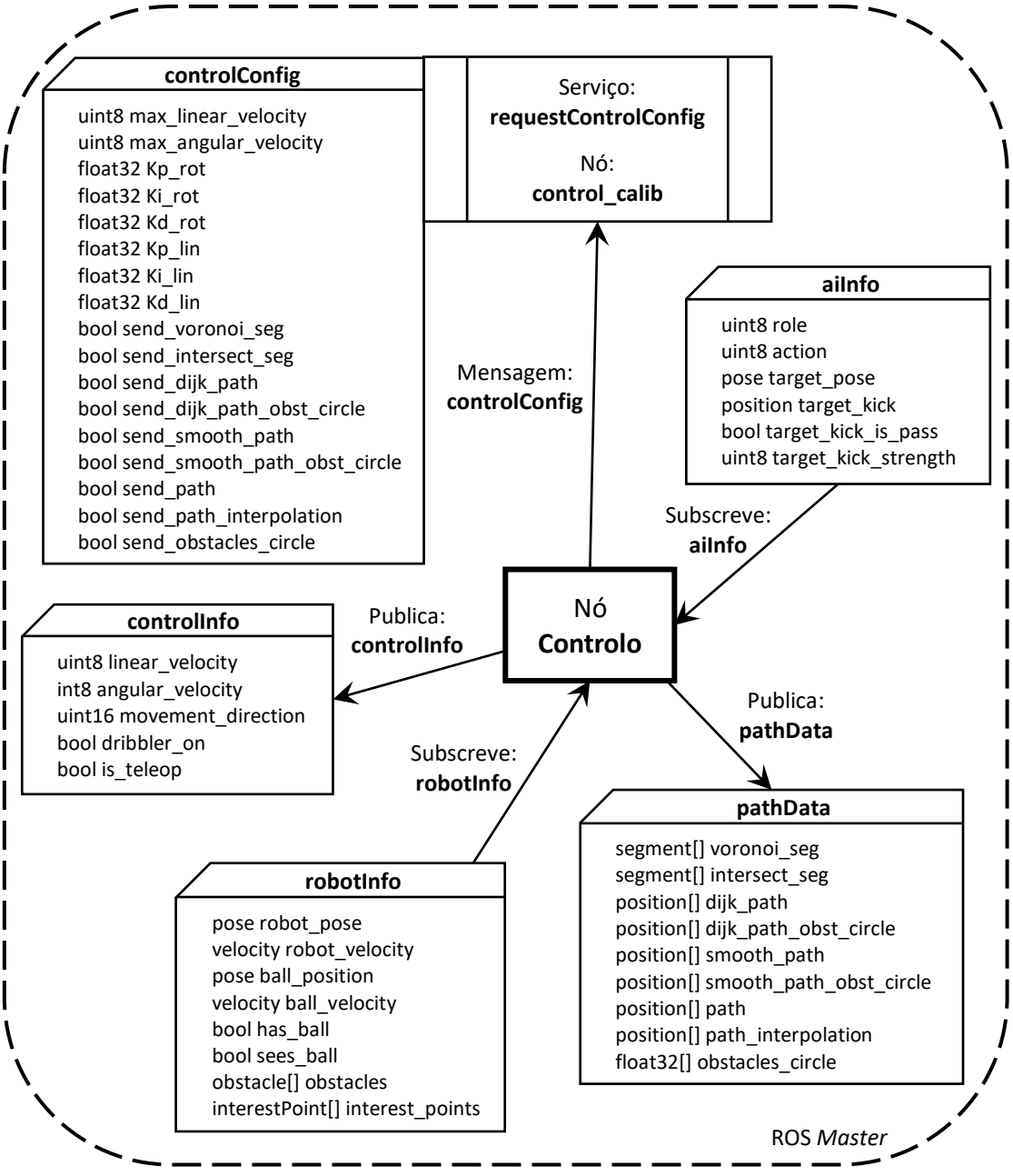


Figura 5.3 – Mensagens que o nó *Controlo* publica e subscreve, e que envia/recebe.

- **aiInfo**: o nó *Controlo* subscreve esta mensagem, que contém informação que deve ser executada pelo robô, tal como o seu papel (defesa, avançado ou entre outros), a ação (*stop*, *slow*, *engage ball*, *fast move* ou entre outras), a posição e orientação pretendida, a posição (x, y) pretendida para a bola, uma variável que indica se o robô deve fazer um passe ou um remate, e a força aplicada pelo chuto na bola. Esta mensagem é publicada pelo nó *Inteligência Artificial* ou pelo nó da ferramenta de configuração no caso de um teste (este último nó não define o papel a ser executado pelo robô).

CGAL. A *Computational Geometry Algorithms Library* (CGAL) [51] é uma biblioteca em C++ de estruturas de dados e algoritmos de geometria computacional. Em 1995, um grupo de universidades e centros de investigação iniciaram o desenvolvimento da CGAL, com o objetivo de disponibilizar gratuitamente algoritmos geométricos eficientes, flexíveis e robustos [52]. Esta biblioteca é aplicada em diversas áreas, tais como, a robótica, visão por computador, sistemas de informação geográfica, design assistido por computador, biologia molecular, imagens médicas e computação gráfica.

A biblioteca CGAL é constituída por um núcleo geométrico ou *kernel*, pela *biblioteca básica* e por uma *biblioteca de suporte*. O *kernel* contém objetos geométricos não modificáveis, tais como, ponto, segmento, linha, raio, retângulo orientado e entre outros. Para estes objetos existem dois conjuntos de funções, um denominado *predicates* e o outro *constructions*. As funções de *predicates* permitem testar se um ponto está no interior de um círculo ou de uma esfera, comparar distâncias entre objetos e entre outras funções em que o tipo de dados de retorno é bool (verdadeiro ou falso) ou enum (enumeração). Por outro lado, as funções de *constructions* permitem transformações geométricas, calcular e detetar uma interseção entre objetos, calcular distâncias, entre outras funções.

Esta biblioteca oferece diversos modelos de *kernel*, que permitem optar por uma implementação com resultados exatos ou por uma implementação onde a velocidade para obter resultados é mais importante.

A biblioteca básica é constituída por várias estruturas de dados e por vários algoritmos, sendo implementados neste trabalho a triangulação de Delaunay (*2D Triangulation*), o diagrama de Voronoi (*2D Voronoi Diagram Adaptor*) e arranjos (*2D Arrangements*).

No que diz respeito à biblioteca de suporte, esta contém estruturas de dados não geométricas para a interface com algoritmos de outras bibliotecas, tipos de números, I/O para depuração e para a interface com várias ferramentas de visualização.

Sistema de Controlo de Movimento. O sistema de controlo de movimento implementado neste trabalho (ver *Figura 5.4*) é constituído pelos dois processos já mencionados *Planeamento da Trajetória* e *Controlo do Movimento*, sendo o nó *Controlo* responsável por integrar estes dois processos na rede ROS de um robô da Minho Team. A comunicação entre estes dois processos é feita diretamente, sendo enviado do processo *Planeamento da Trajetória* para o processo *Controlo do Movimento*, um vetor de posições com a trajetória obtida no primeiro processo.

Nos próximos dois subcapítulos *5.1* e *5.2* serão apresentados todos os detalhes da implementação destes dois processos.

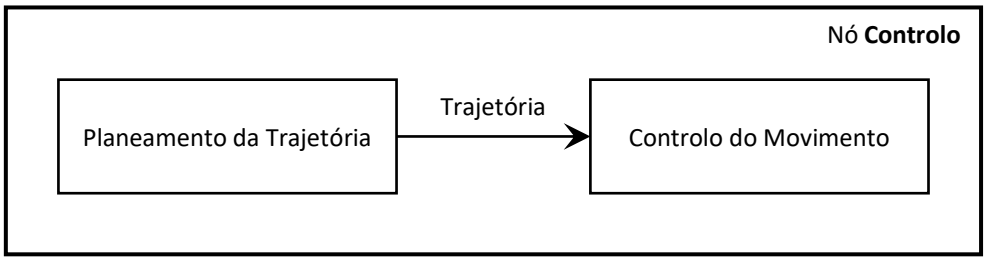


Figura 5.4 - Visão geral do sistema de controlo de movimento.

5.1. PLANEAMENTO DA TRAJETÓRIA

O primeiro passo para a implementação deste processo, foi a análise do espaço de trabalho onde o robô se encontra e o conhecimento que o robô tem sobre o espaço de trabalho. Após esta análise foi feita a escolha do tipo de planeamento da trajetória e dos métodos associados ao tipo de planeamento escolhido. A estes métodos são adicionadas algumas estratégias/algoritmos, criando um modelo de planeamento da trajetória eficiente e robusto.

Tal como já foi mencionado em capítulos anteriores, o espaço de trabalho ou de jogo da MSL é extramente dinâmico, onde os robôs deslocam-se a velocidades elevadas, por vezes atingindo os 4 m/s. Isto significa que no espaço de jogo existem obstáculos dinâmicos, sendo estes obstáculos, robôs adversários e robôs cooperativos (robôs da Minho Team).

Na MSL, cada robô possui o seu sistema de visão e alguns sensores para a perceção do espaço de jogo á sua volta, não existindo uma observação total deste espaço, tornando assim tudo mais complexo. Desta forma, o *hardware* de um robô não permite um conhecimento completo do espaço de jogo, sendo por isso, um espaço parcialmente conhecido. Para obter um conhecimento completo do espaço de jogo, as equipas, normalmente, fazem a fusão da informação obtida de todos os robôs da equipa. Essa informação, geralmente, é constituída pela posição (x, y) de cada robô, as posições de

possíveis obstáculos percebidos por cada robô, a posição da bola, e as velocidades dos robôs adversários e da bola. A fusão de toda esta informação permite obter uma representação total do espaço de jogo com excelente precisão. A Minho Team também faz a fusão da informação, exceto das velocidades dos robôs adversários, tendo por isso de momento uma representação do espaço de jogo com pouca precisão em relação a outras equipas.

Após esta análise foi escolhido o tipo de planeamento da trajetória a ser implementado nos robôs. De acordo com o que foi apresentado na secção *3.1.1.1* do *Capítulo 3*, existem três tipos: global, local e a combinação dos dois (global e local). O planeamento local, sendo adequado para espaços dinâmicos e totalmente desconhecidos, não produz uma trajetória muito eficaz em relação ao planeamento global. Por isso, visto que é possível através da fusão da informação obter um espaço totalmente conhecido, mesmo com pouca precisão, o planeamento global foi, desta forma, o tipo de planeamento escolhido. O planeamento global e local, ou seja, a combinação dos dois, seria teoricamente a melhor opção, mas com os métodos escolhidos, associados ao planeamento global, e com as estratégias adicionadas a estes métodos, o planeamento da trajetória desenvolvido também se revela eficiente e robusto. O modelo de planeamento da trajetória desenvolvido neste trabalho é semelhante ao da equipa Tech United Eindhoven, apresentado no *Capítulo 2*.

Relativamente ao modo que o planeamento da trajetória é implementado, este pode ser centralizado ou descentralizado. Tal como já foi mencionado no subcapítulo *2.4 DISCUSSÃO DO ESTADO DA ARTE*, todas as equipas da MSL, incluindo a Minho Team, usam o planeamento descentralizado. Isto significa que cada robô da Minho Team, ou de qualquer outra equipa, faz o planeamento da sua trajetória. Por isso, todos os passos que serão apresentados de seguida são para o planeamento da trajetória de um único robô, sendo igual para os restantes.

Posto isto, a trajetória planeada deve permitir ao robô deslocar-se da posição atual para a posição pretendida, evitando colisões com possíveis obstáculos. Além disso, a trajetória tem um determinado custo que deve ser minimizado. Neste trabalho o custo é dividido entre encontrar a trajetória mais curta e a mais suave.

5.1.1. ESPAÇO DE CONFIGURAÇÃO

A representação ou espaço de configuração do espaço de jogo da MSL é simples, pois trata-se de um espaço com robôs móveis sem braço robótico que se deslocam no plano (x, y). No planeamento da trajetória, a orientação do robô não é considerada pelo facto do robô ter um movimento holonómico, sendo o robô representado apenas pela sua posição no plano (x, y). O formato usado para representar

os obstáculos (robôs adversários e robôs cooperativos) no espaço de configuração é circular, pois é o mesmo formato da base dos robôs da Minho Team e o mais próximo das outras equipas, sendo as formas circular, triangular e quadrada as mais usadas por estas. Assim, o formato circular com o diâmetro configurável permite a melhor aproximação a todos os formatos, sendo também o mais adequado para as estratégias e os métodos implementados no planeamento da trajetória, tal como se poderá verificar posteriormente. A posição (x, y) de cada obstáculo fica situada, como é óbvio, no centro do respetivo círculo.

Na discretização do espaço de configuração, que será apresentada na secção *5.1.3*, não será considerada para o espaço de configuração qualquer área circular para os obstáculos, mas apenas os pontos, ou seja, as posições destes. No entanto, neste passo, estes serão representados no *visualizer* pela área circular da base, apenas para ser mais representativo, pois não será usada qualquer área no algoritmo deste passo. Nos restantes passos do planeamento, incluindo a *TRAJETÓRIA RETILÍNEA*, cada obstáculo será representado no espaço de configuração por uma área circular com diâmetro configurável ou pela sua posição se o diâmetro da área for igual a zero. Esta área denominada *área do obstáculo* será explicada posteriormente. Relativamente à representação no *visualizer*, cada obstáculo será representado pela área circular da base e pela área do obstáculo. No que diz respeito ao robô e tal como já foi mencionado, este será representado no espaço de configuração pela sua posição e no *visualizer* pela área circular da base.

Posto isto, o espaço de configuração (EC), tal como é possível observar pela *Figura 5.5*, é constituído por dois tipos de locais: EC_{obst} e EC_{livre}.

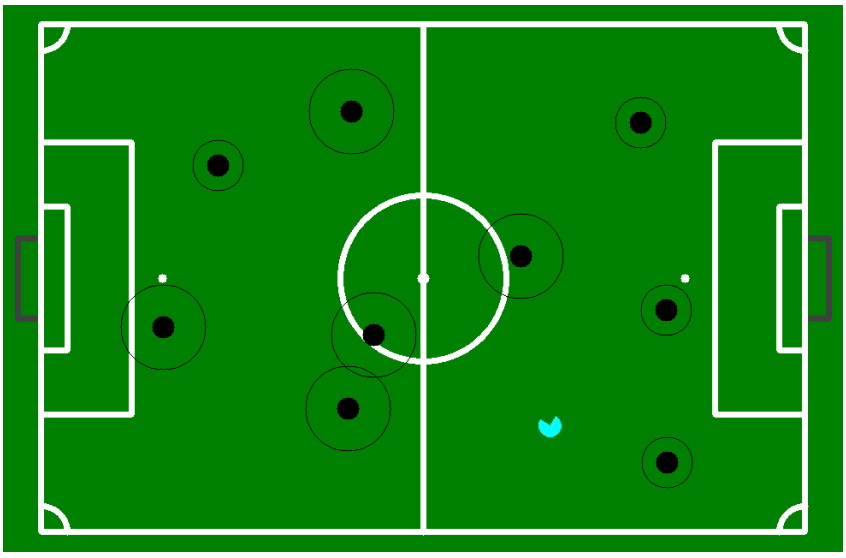


Figura 5.5 - Espaço de configuração do espaço de jogo. Os círculos a preto representam a área da base dos obstáculos e a circunferência em cada obstáculo, representa a área do obstáculo. O robô é representado nesta ferramenta de visualização pela forma com a cor ciano.

A área do obstáculo de todos os obstáculos (EC_{obst}) é espaço ocupado que o robô tem de evitar, enquanto que os locais não ocupados por obstáculos (EC_{livre}) é espaço livre onde o robô pode mover-se. Assim, o problema do planeamento da trajetória é encontrar a melhor trajetória entre a posição atual e uma posição pretendida no espaço de configuração livre (EC_{livre}).

5.1.2. TRAJETÓRIA RETILÍNEA

A trajetória retilínea é obtida quando não existem obstáculos a intersetarem com a linha reta entre a posição atual do robô e a posição pretendida para o mesmo, e quando o diâmetro da área do obstáculo, de todos os obstáculos, for superior a duas vezes o diâmetro da área circular da base.

Quando existe esta trajetória, os próximos passos do planeamento apresentados de seguida, não são executados, sendo esta a trajetória final. Por outro lado, quando existem obstáculos na linha reta entre as duas posições, não existe trajetória retilínea e os próximos passos do planeamento são executados de forma a obter a melhor trajetória.

Na *Figura 5.6* é possível observar a trajetória retilínea para o exemplo do espaço de configuração apresentado.

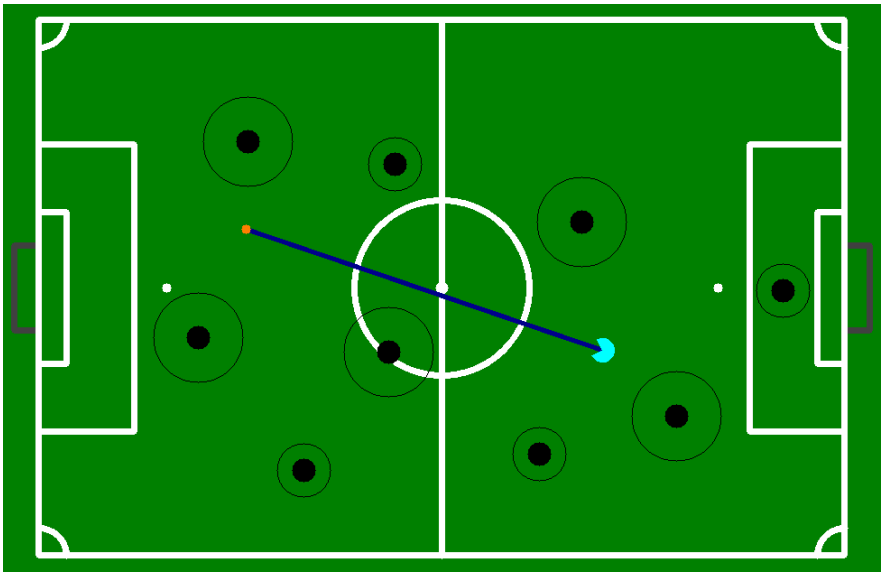


Figura 5.6 – Trajetória retilínea representada pelo segmento de reta azul escuro.

5.1.3. DISCRETIZAÇÃO DO ESPAÇO DE CONFIGURAÇÃO

O espaço de configuração, consiste num mapa que faz a representação geométrica contínua do espaço de jogo. Este mapa é transformado num mapa discreto de forma a ser possível implementar o algoritmo escolhido para a pesquisa da trajetória. Para obter o mapa discreto foi implementado o

diagrama de Voronoi, apresentado na secção *3.1.1.3*. Este diagrama produz uma rede de curvas (ou arestas) que dão forma a possíveis trajetórias no espaço de configuração livre e que se encontram à distância máxima das posições dos obstáculos.

Na *Figura 5.7* é possível observar o diagrama de Voronoi de 9 obstáculos, no entanto, existem neste caso, 7 regiões de Voronoi ilimitadas, não existindo arestas (segmentos de reta) entre os obstáculos dessas regiões e os limites do espaço de configuração. A solução encontrada para este problema foi adicionar obstáculos artificiais sobrepostos às linhas dos limites do campo (ver *Figura 5.8*). O número de obstáculos artificiais depende das dimensões do campo de jogo e do diâmetro dos robôs da Minho Team. Tal como é apresentado na *Figura 5.8*, as posições dos obstáculos artificiais são determinadas de forma a que estes fiquem igualmente espaçados. Na *Figura 5.9* estão representadas as etapas para a discretização do espaço de configuração.

Para a construção do diagrama de Voronoi foi utilizada a biblioteca CGAL, mais especificamente, as estruturas de dados e algoritmos de *2D Voronoi Diagram Adaptor* e de *2D Triangulation*. Na CGAL, o diagrama de Voronoi de um conjunto de pontos (posições de obstáculos) é obtido implicitamente a partir do seu grafo dual, que é o grafo da triangulação de Delaunay do mesmo conjunto de pontos. Isto é, para obter um diagrama de Voronoi, a biblioteca CGAL, através do *2D Voronoi Diagram Adaptor*, calcula a triangulação de Delaunay e depois através de uma estrutura de dados DCEL (*Doubly Connected Edge List*) produz o diagrama de Voronoi. A triangulação de Delaunay é apresentada na secção *3.1.1.4* e a dualidade entre Voronoi e Delaunay na secção *3.1.1.5*.

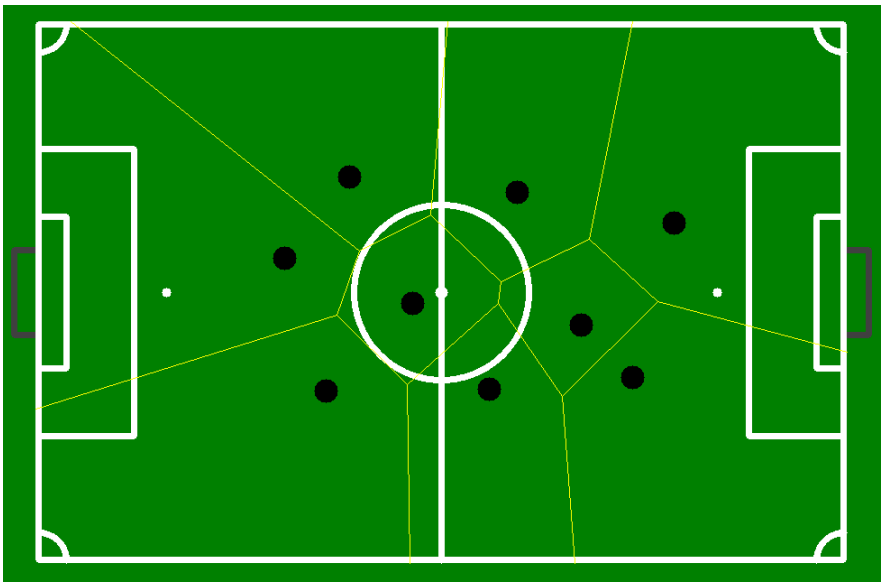


Figura 5.7 - Diagrama de Voronoi de 9 obstáculos.

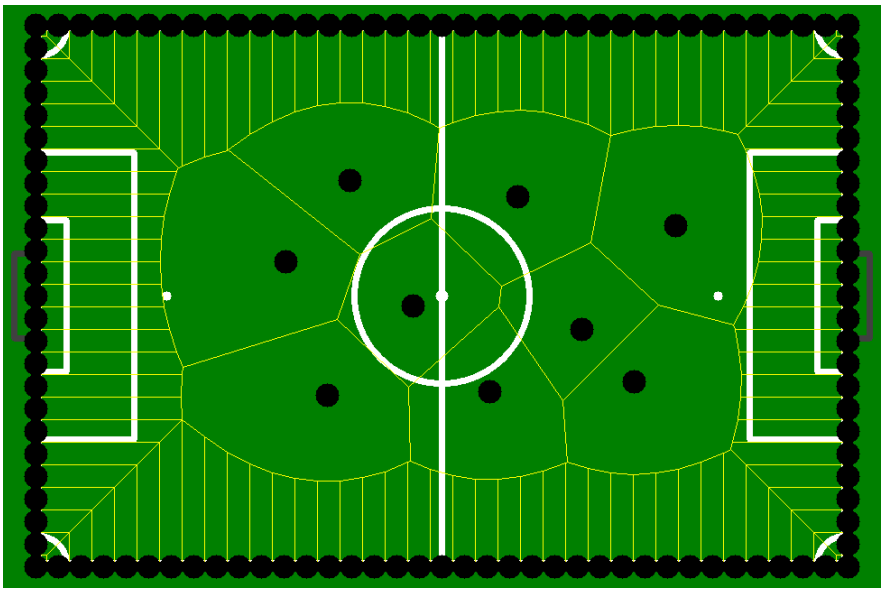


Figura 5.8 - Diagrama de Voronoi com os obstáculos reais e artificiais.

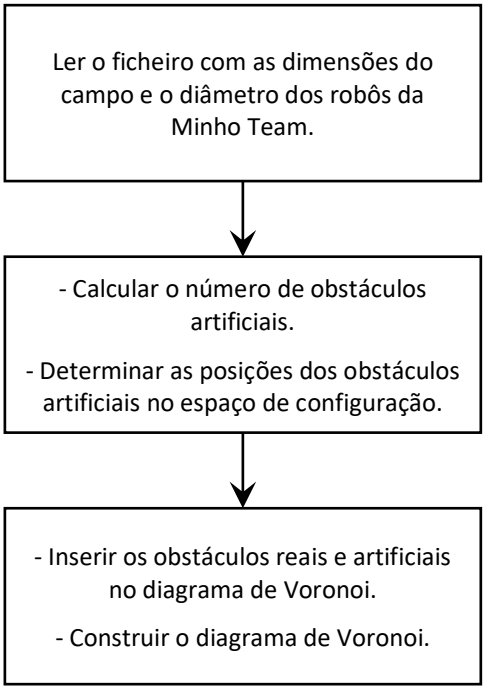


Figura 5.9 – Etapas para a discretização do espaço de configuração.

Na CGAL os algoritmos são parametrizados, sendo o modelo de *kernel* o primeiro parâmetro a ser escolhido para a construção do diagrama de Voronoi e para os objetos geométricos utilizados na discretização do espaço de configuração. O *kernel* denominado *Exact_predicates_inexact_constructions_kernel*, foi o escolhido por ter o menor tempo de execução de todos os *kernels* e também por ser o mais usual nestes algoritmos. As construções geométricas com

este modelo de *kernel* podem ser ocasionalmente inexatas, devido a erros de arredondamento. Contudo, neste tipo de aplicações não representa um problema.

A classe de *2D Voronoi Diagram Adaptor*, que adapta um grafo da triangulação de Delaunay ao diagrama de Voronoi, possui três parâmetros. O primeiro parâmetro está relacionado com o comportamento da triangulação de Delaunay, sendo que dos modelos existentes para este parâmetro, foi atribuído o modelo que permite a adaptação ao diagrama de Voronoi e que permite também a inserção e remoção dinâmica de obstáculos, sem ser necessário reconstruir toda a triangulação. No entanto, neste trabalho os obstáculos não são inseridos nem removidos dinamicamente, mas sim inseridos todos os obstáculos em cada ciclo do programa principal e, portanto, é construída uma nova triangulação em cada ciclo. Esta é a melhor opção, pois tal como já foi mencionado, o espaço de jogo da MSL é extremamente dinâmico e por isso as posições dos obstáculos podem alterar-se a cada momento. O segundo parâmetro é responsável pelo acesso a informações geométricas na triangulação de Delaunay e também por permitir ou não a consulta do sítio (posição de um obstáculo no diagrama de Voronoi) mais próximo de um determinado ponto no diagrama de Voronoi. O terceiro parâmetro determina o que deve ser feito com as arestas e regiões degeneradas do diagrama de Voronoi. Isto é, na adaptação de uma triangulação de Delaunay a um diagrama de Voronoi, podem existir arestas de Voronoi de comprimento zero e regiões de Voronoi de área zero. Esta situação pode acontecer, quando existe um subconjunto de pontos na triangulação de Delaunay em configurações degeneradas e por isso o grafo não é completamente triangulado. Desta forma, foi escolhido para este parâmetro um modelo que remove estas degenerações, oferecendo um diagrama de Voronoi sem qualquer aresta ou região degenerada.

5.1.4. ÁREA DO OBSTÁCULO

Depois da discretização do espaço de configuração, onde os obstáculos são representados pelas suas posições, nos próximos passos estes serão representados pela área do obstáculo, já ilustrada na *Figura 5.5*.

A área do obstáculo tem como objetivo principal, garantir um espaço mínimo entre o obstáculo e o robô para evitar uma colisão. Desta forma, o diâmetro da área do obstáculo, atribuído inicialmente, é superior ao diâmetro da área circular da base.

Na *Figura 5.5* é possível observar pela área de cada obstáculo, a existência de dois grupos de obstáculos. Os obstáculos com a área mais pequena são robôs cooperativos e os outros são robôs

adversários. Esta diferença deve-se à baixa precisão da representação do espaço de jogo, com as posições dos robôs adversários a terem menor precisão em relação às posições dos robôs cooperativos.

O diâmetro da área do obstáculo atribuído inicialmente a cada obstáculo, com base na abordagem apresentada, pode ter de ser alterado posteriormente. Esta alteração depende de algumas situações que podem ocorrer no espaço de jogo, ou então, é necessário alterar para obter um espaço de configuração mais adequado para o robô executar determinados comportamentos de jogo. A alteração consiste na diminuição do diâmetro, como até diâmetro zero, ou seja, área do obstáculo igual a zero. Esta possível alteração será apresentada com maior detalhe no próximo passo com a ajuda de um fluxograma.

5.1.5. PESQUISA PELA TRAJETÓRIA MAIS CURTA

Na discretização do espaço de configuração, o diagrama de Voronoi produz um conjunto de arestas, que dão forma a possíveis trajetórias no espaço de configuração livre. No entanto, o objetivo é implementar um algoritmo para encontrar apenas uma trajetória entre todas as trajetórias possíveis, sendo conveniente para este trabalho a pesquisa da trajetória mais curta entre a posição atual do robô e uma posição pretendida. Desta forma, foi escolhido o algoritmo de Dijkstra, apresentado na secção *3.1.1.6*.

Para obter a trajetória mais curta através do algoritmo de Dijkstra, foi implementada a função *dijkstra_shortest_paths*, disponibilizada pela *Boost Graph Library* (BGL). Esta função tem dois parâmetros de entrada obrigatórios e os restantes são parâmetros de entrada e saída nomeados. Na BGL, a maioria das funções possuem vários parâmetros, existindo por isso uma classe que permite inserir em qualquer ordem, todos ou apenas alguns parâmetros nomeados, de forma a obter determinados tipos de resultados. Na função *dijkstra_shortest_paths* o primeiro parâmetro obrigatório é o objeto do grafo direcionado ou não direcionado onde o algoritmo é aplicado, e o segundo parâmetro obrigatório é o vértice inicial. Em relação aos parâmetros de entrada nomeados, foram usados para este trabalho o mapeamento de todos os vértices do grafo e o custo de cada aresta do grafo. Em relação aos parâmetros de saída nomeados foram usados o *predecessor_map* para obter o antecessor de cada vértice e o *distance_map* para obter o custo total da trajetória mais curta entre o vértice inicial e cada um dos restantes vértices do grafo. Isto é, o custo total da trajetória mais curta entre um determinado vértice do grafo e o vértice inicial, é a soma do custo exato de cada aresta dessa trajetória.

O objeto do grafo, primeiro parâmetro da função, é construído através da estrutura de dados *Arrangement* da biblioteca CGAL, sendo que esta estrutura permite através da inserção de segmentos de reta, construir um grafo compatível com os conceitos de grafos da BGL. Desta forma, o diagrama de

Voronoi obtido em cada ciclo do programa principal, é transformado num grafo de segmentos de reta do tipo *Arrangement*. No entanto, dos dois tipos de arestas existentes no diagrama de Voronoi, segmentos de reta e semirretas, apenas são considerados para o novo grafo os segmentos de reta. Relativamente ao tipo de grafo, foi usado para este trabalho o grafo não direcionado.

Um segmento de reta do diagrama de Voronoi é inserido no novo grafo se não intercetar, em todo o seu comprimento, com nenhum obstáculo existente no espaço de jogo. Na *Figura 5.10* é possível observar a interseção de segmentos de reta do diagrama de Voronoi com alguns obstáculos, sendo estes segmentos de reta representados a cinzento claro. Nesta figura e nas próximas não serão representados os obstáculos artificiais no *visualizer*. Porém, as suas posições serão sempre incluídas na construção do diagrama, tal como é possível comparar pelos diagramas de Voronoi das figuras *5.8* e *5.10*.

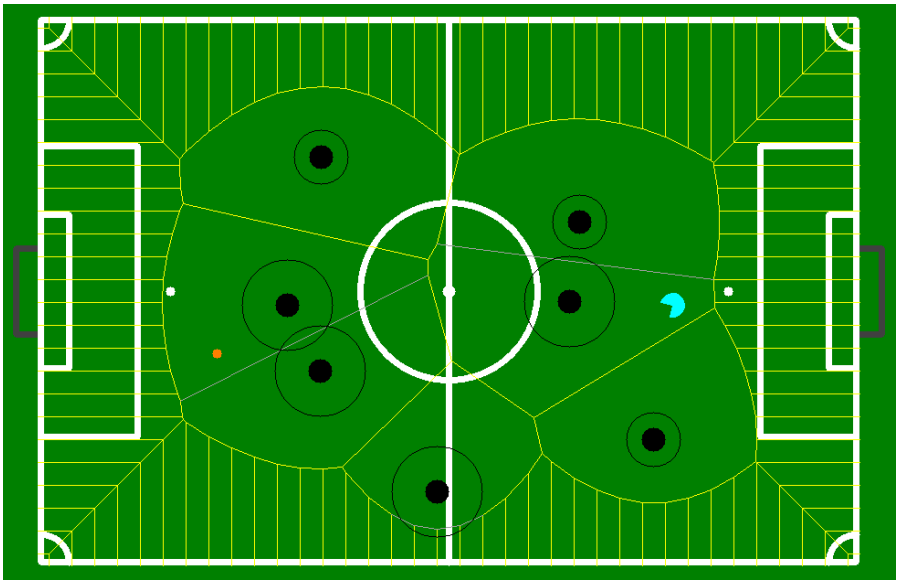


Figura 5.10 – Interseção de alguns segmentos de reta do diagrama de Voronoi com alguns obstáculos. Estes segmentos de reta estão representados a cinzento claro.

De seguida, são feitas as conexões para a posição atual do robô e para posição pretendida para o mesmo com o diagrama de Voronoi. A conexão para cada uma destas duas posições é feita por segmentos de reta, conectados entre a posição e os vértices da região de Voronoi limitada ou ilimitada, onde se encontra a posição. Posto isto, foram desenvolvidos dois tipos de abordagens de conexão, disponíveis para as duas posições. Na *Figura 5.11* é possível observar um exemplo das conexões das duas posições com o diagrama de Voronoi, onde cada posição tem uma abordagem de conexão diferente.

A abordagem usada pela posição atual do robô, apenas pode ser usada no caso em que a posição (atual ou pretendida), não se encontre no interior da área de qualquer obstáculo. Esta abordagem faz a pesquisa de todos os vértices da região de Voronoi onde se encontra a posição e verifica, posteriormente,

a interseção dos segmentos que fazem a conexão, com todos os obstáculos existentes no espaço de jogo. Os segmentos que não intersetam com qualquer obstáculo são inseridos no novo grafo.

Relativamente à abordagem usada pela posição pretendida para o robô (ponto laranja), esta pode ser usada em qualquer caso, isto é, internamente ou externamente à área de um obstáculo. Nesta abordagem, é criada uma linha na proximidade da posição (ver linha vermelha *(a)* da *Figura 5.12*) e perpendicular à linha entre a posição e o sítio (posição do obstáculo) da região de Voronoi onde se encontra a posição (ver linha vermelha *(b)* da *Figura 5.12*). Posteriormente, é feita a pesquisa pelos vértices, tal como na outra abordagem, e verificada a interseção dos segmentos de reta da conexão com a linha *(a)*, sendo inseridos no novo grafo os segmentos de reta que não intersetam com esta linha.

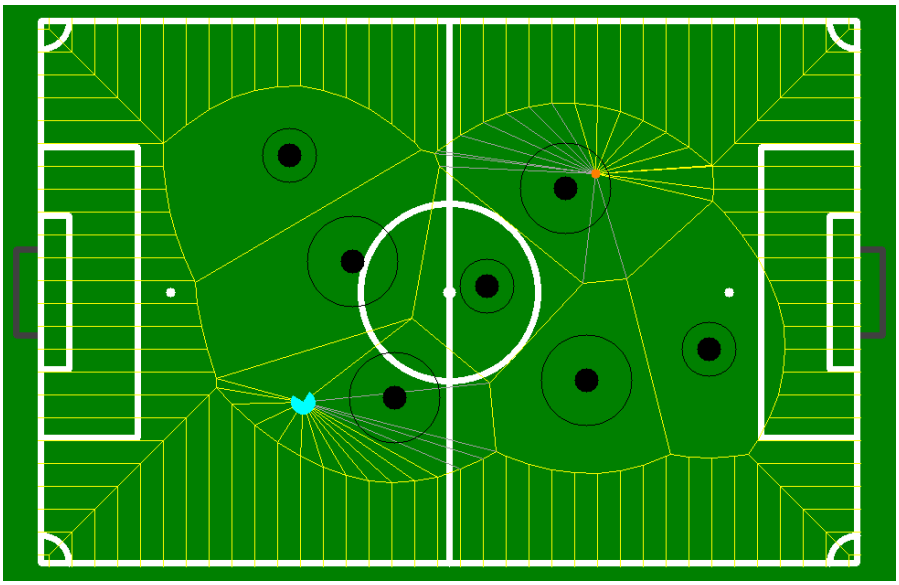


Figura 5.11 – Conexões através de segmentos de reta para a posição atual do robô e para a posição pretendida para o mesmo (ponto laranja) com o diagrama de Voronoi.

Após a construção do objeto do novo grafo, foram definidos os restantes parâmetros da função, entre os quais, o vértice inicial que corresponde à posição atual do robô, o mapeamento de todos os vértices do grafo para índices, onde cada vértice é identificado por um número inteiro, e o custo de cada aresta (segmento de reta) do grafo. Depois destes parâmetros estarem definidos a função é executada.

O algoritmo da função é igual ao algoritmo apresentado na secção *3.1.1.6*, em que permite obter a trajetória mais curta entre o vértice inicial e cada um dos restantes vértices do grafo. Desta forma, a função *dijkstra_shortest_paths* contém o parâmetro de saída nomeado *predecessor_map*, que contém o antecessor de cada vértice após terminar a execução da função. Através deste parâmetro é possível determinar a sequência de vértices para a trajetória mais curta entre o vértice inicial e outro vértice do grafo, que corresponde, neste caso, à posição pretendida para o robô. Assim, a sequência de vértices é

obtida da mesma forma como apresentado na secção *3.1.1.6* e guardada numa variável, que será utilizada no próximo passo do planeamento da trajetória.

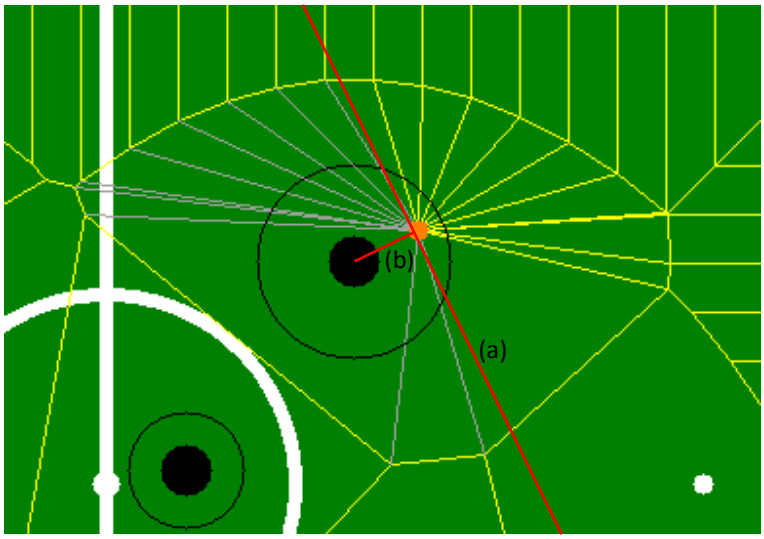


Figura 5.12 – Tipo de abordagem de conexão entre uma posição e o diagrama de Voronoi. As linhas vermelhas *(a)* e *(b)* representam o método usado nesta abordagem.

Na *Figura 5.14* é possível observar a trajetória mais curta para o exemplo do espaço de configuração apresentado.

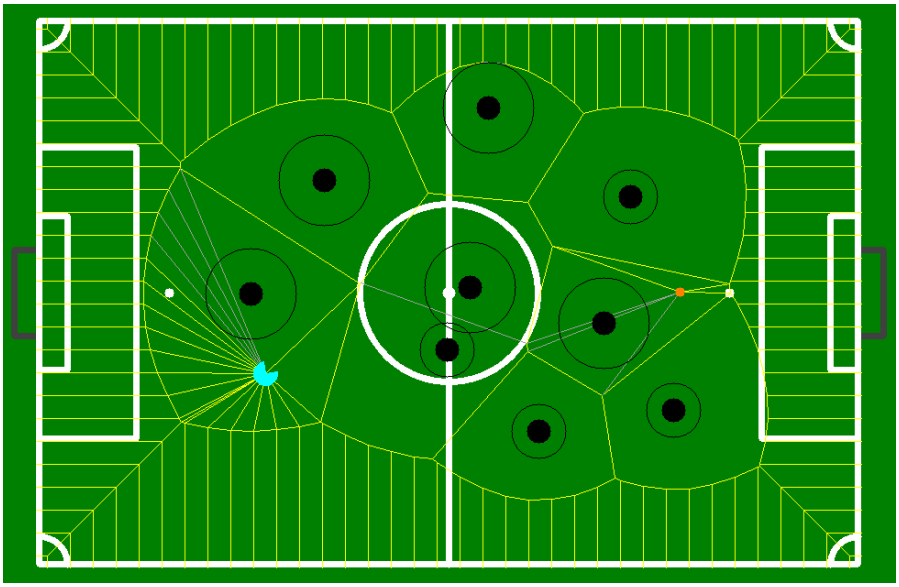


Figura 5.13 – Espaço de configuração igual ao da *Figura 5.14*, sem a representação da trajetória mais curta.

No modelo de planeamento da trajetória apresentado até ao momento, é possível que em algumas situações não exista qualquer trajetória entre a posição atual do robô e a posição pretendida para o mesmo. Isto pode acontecer, se não existir uma sequência de vértices entre estas duas posições. Na

Figura 5.15 é possível observar um exemplo deste problema, sendo que o posicionamento dos três obstáculos mais próximos do ponto laranja (posição pretendida para o robô) e as respetivas áreas, não permitem qualquer conexão do ponto com o diagrama de Voronoi.

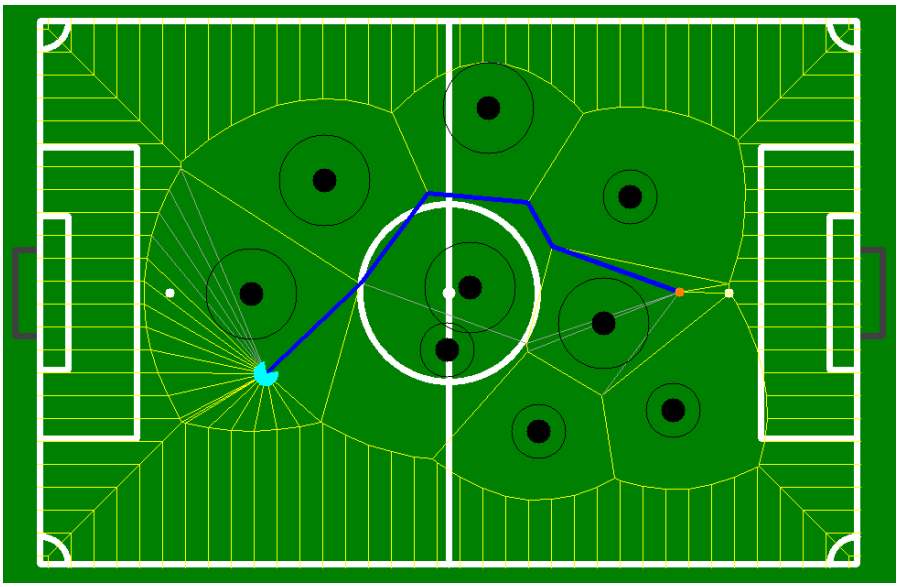


Figura 5.14 – Trajetória mais curta entre a posição atual do robô e a posição pretendida para o mesmo, representada pelos segmentos de reta azuis.

Este tipo de problema foi resolvido com a diminuição do diâmetro da área do obstáculo, de alguns ou de todos os obstáculos existentes no espaço de jogo, ou caso não seja suficiente, pode ser excluída a área, sendo os obstáculos representados no espaço de configuração pelas suas posições. Isto é feito, se a função de pesquisa da trajetória mais curta, após ser executada, não retornar uma sequência de vértices da trajetória mais curta. Esta abordagem é apresentada no fluxograma da *Figura 5.16*.

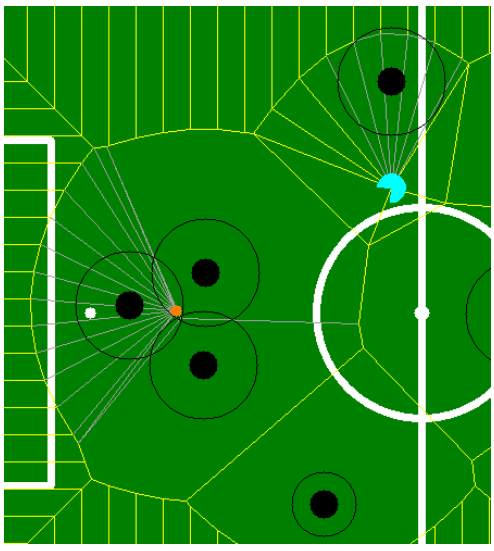


Figura 5.15 – Exemplo de um problema do modelo de planeamento da trajetória apresentado até ao momento.

Através da análise do fluxograma, possibilita perceber que após a diminuição do diâmetro da área, é feita novamente a pesquisa pela trajetória mais curta. Como a diminuição do diâmetro é feita de forma progressiva, o ciclo é repetido até existir uma trajetória. Assim, a função *Calcular o Diâmetro da Área de Cada Obstáculo* atribui o diâmetro inicial para a área do obstáculo, posteriormente e se necessário, diminui o diâmetro. Esta função permite também outras funcionalidades que serão apresentadas posteriormente.

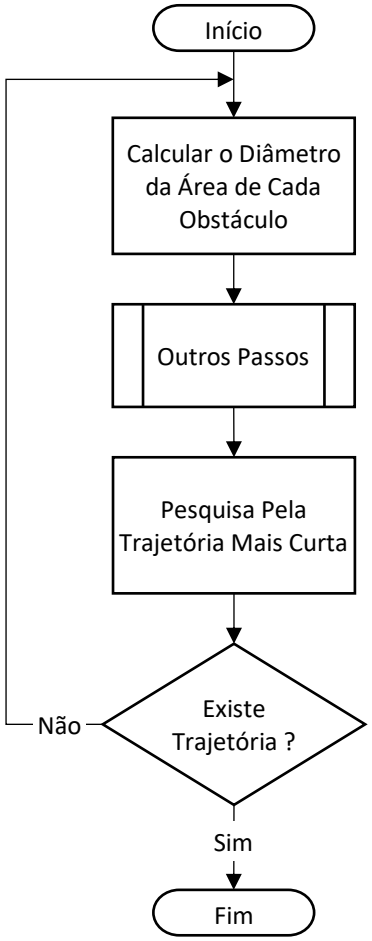
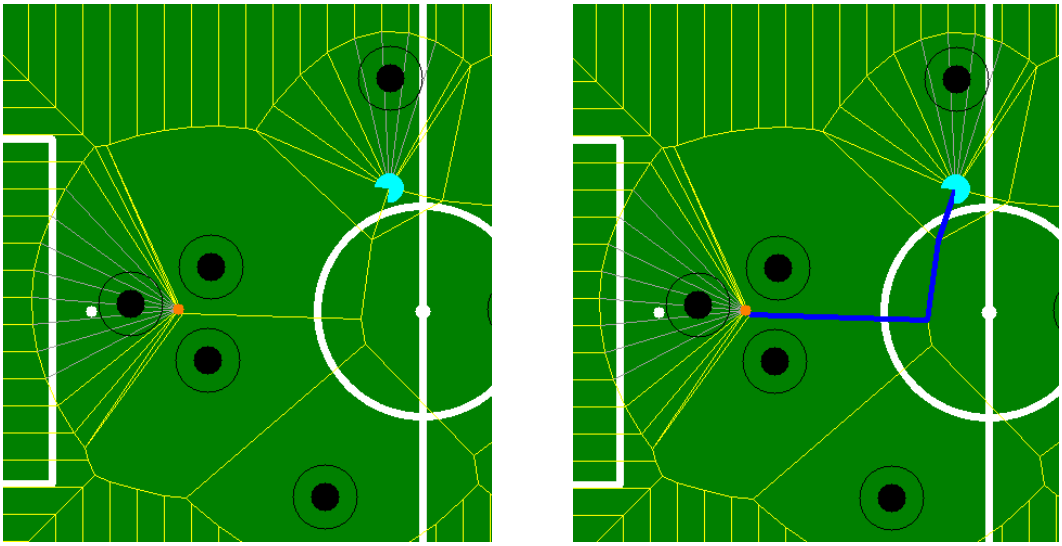


Figura 5.16 - Fluxograma que permite diminuir a área de cada obstáculo.

Na *Figura 5.17(a)* é possível observar o resultado obtido após o algoritmo apresentado ser executado, neste caso, para o exemplo da *Figura 5.15*. O mesmo resultado, mas com a representação da trajetória mais curta, é apresentado na *Figura 5.17(b)*. Através da análise das figuras *5.15* e *5.17*, é possível perceber que foi diminuída a área do obstáculo, apenas para os robôs adversários, pois o algoritmo verifica se existem obstáculos com o diâmetro da área do obstáculo superior a duas vezes o diâmetro da área circular da base. Esta é a primeira condição deste algoritmo, se não for suficiente, será diminuído ainda mais o diâmetro para todos os obstáculos, até existir uma trajetória. É de salientar que

o motivo por ser duas vezes o diâmetro da área circular da base, deve-se por este ser o mínimo para evitar uma colisão.

Diminuir ou aumentar o diâmetro da área do obstáculo, permite também obter um espaço de configuração mais adequado para o robô executar determinados comportamentos de jogo. Esta funcionalidade é controlada pela função apresentada anteriormente.



(a) (b)

Figura 5.17 – Resultado obtido após o algoritmo ser executado para o exemplo da *Figura 5.15*.

Na *Figura 5.18* é possível observar dois tipos de comportamentos que podem ser atribuídos pelo processo *Inteligência Artificial*, sendo que *(a)* e *(c)* representam um comportamento, e *(b)* e *(d)* representam outro. Relativamente às duas trajetórias apresentadas para cada comportamento, a trajetória com os segmentos de reta azuis escuros é a trajetória mais curta, enquanto que a de segmentos de reta vermelhos é a trajetória mais curta e suave, que será apresentada posteriormente.

Posto isto, a diferença entre os dois comportamentos tem a ver com a "agressividade" com que o robô se move para a posição pretendida, que neste exemplo, é a bola. Desta forma, o comportamento do robô para *(a) (c)* é mais "agressivo" e mais rápido a atingir a posição pretendida do que em *(b) (d)*. No entanto, considerando que neste exemplo é um robô adversário que tem bola, a probabilidade de o robô colidir para o primeiro comportamento é maior e assim, como está especificado nas regras de jogo da MSL, é marcada falta.

É de salientar que o diâmetro da área do obstáculo em *(a) (c)* é igual a zero, sendo o obstáculo representado apenas pela sua posição no espaço de configuração.

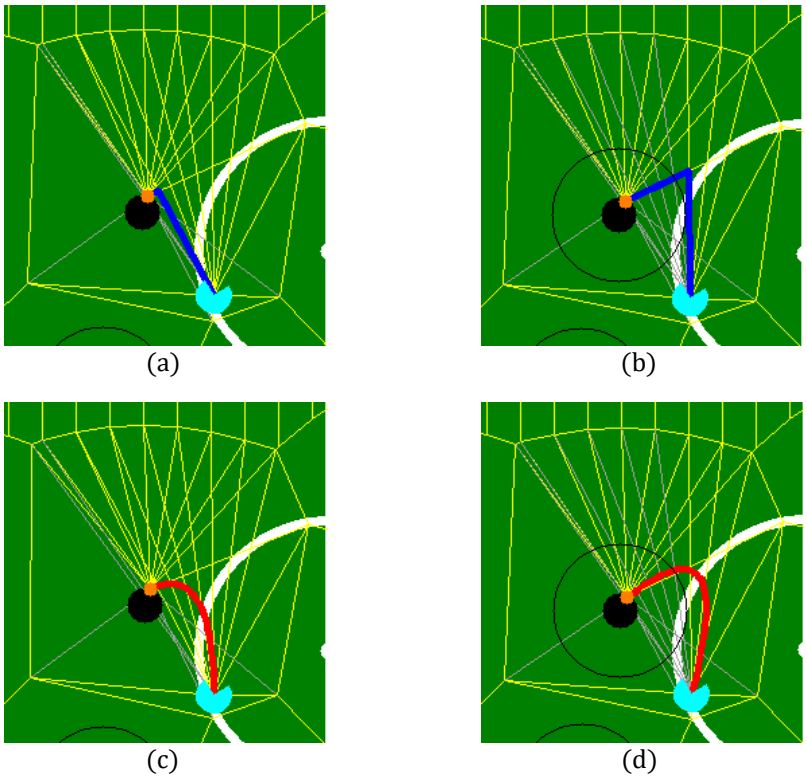


Figura 5.18 – Exemplo de dois tipos de comportamentos. As trajetórias (a) e (c) representam um tipo de comportamento, as (b) e (d) representam outro.

5.1.6. TRAJETÓRIA MAIS CURTA E SUAVE

A trajetória obtida pelo algoritmo de Dijkstra, sendo a mais curta de todas as trajetórias possíveis no mapa discreto implementado, não é a melhor trajetória por dois motivos. O primeiro, é a possibilidade de existirem alguns casos em que a trajetória é a mais curta no mapa discreto e não no espaço de configuração, isto é, como a trajetória obtida é constituída pelos segmentos de reta do diagrama de Voronoi, estes podem estar em alguns locais demasiado distantes dos obstáculos. O segundo motivo, é a possibilidade de a trajetória obtida conter cantos afiados, dificultando o movimento do robô.

Posto isto, a solução é implementar estratégias/algoritmos para suavizar a trajetória e obter, ao mesmo tempo, a trajetória mais curta no espaço de configuração livre. Nas secções seguintes *5.1.6.1* a *5.1.6.4*, será apresentada a implementação para resolver este problema.

5.1.6.1. TRAJETÓRIA MAIS CURTA SIMPLIFICADA

Neste passo, o objetivo é remover segmentos de reta desnecessários da trajetória obtida anteriormente, de forma a construir uma nova trajetória mais simples. Na *Figura 5.21* é possível observar o resultado da implementação deste passo para o exemplo apresentado, sendo que a trajetória obtida anteriormente é constituída pelos 5 segmentos de reta azuis escuros e a nova trajetória simplificada

pelos 2 segmentos de reta azuis claros. O algoritmo desenvolvido neste passo será representado no fluxograma da *Figura 5.20*, e para melhor se compreender o fluxograma, a *Figura 5.19* representa uma trajetória a ser simplificada, com os pontos das extremidades dos segmentos de reta identificados por números. As posições (x, y) dos pontos encontram-se guardadas num vetor.

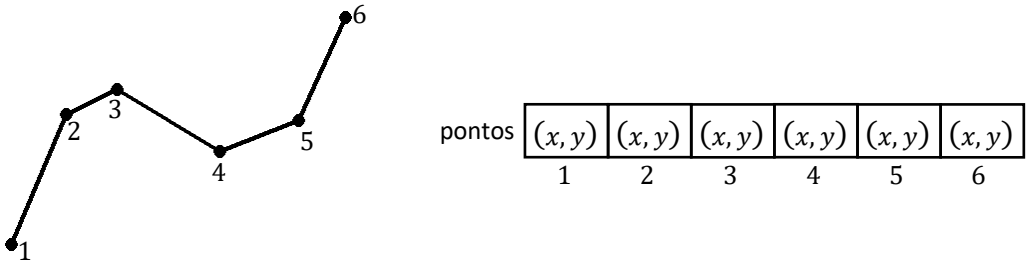


Figura 5.19 - Exemplo de uma trajetória a ser simplificada e a representação do vetor de posições dos pontos da mesma.

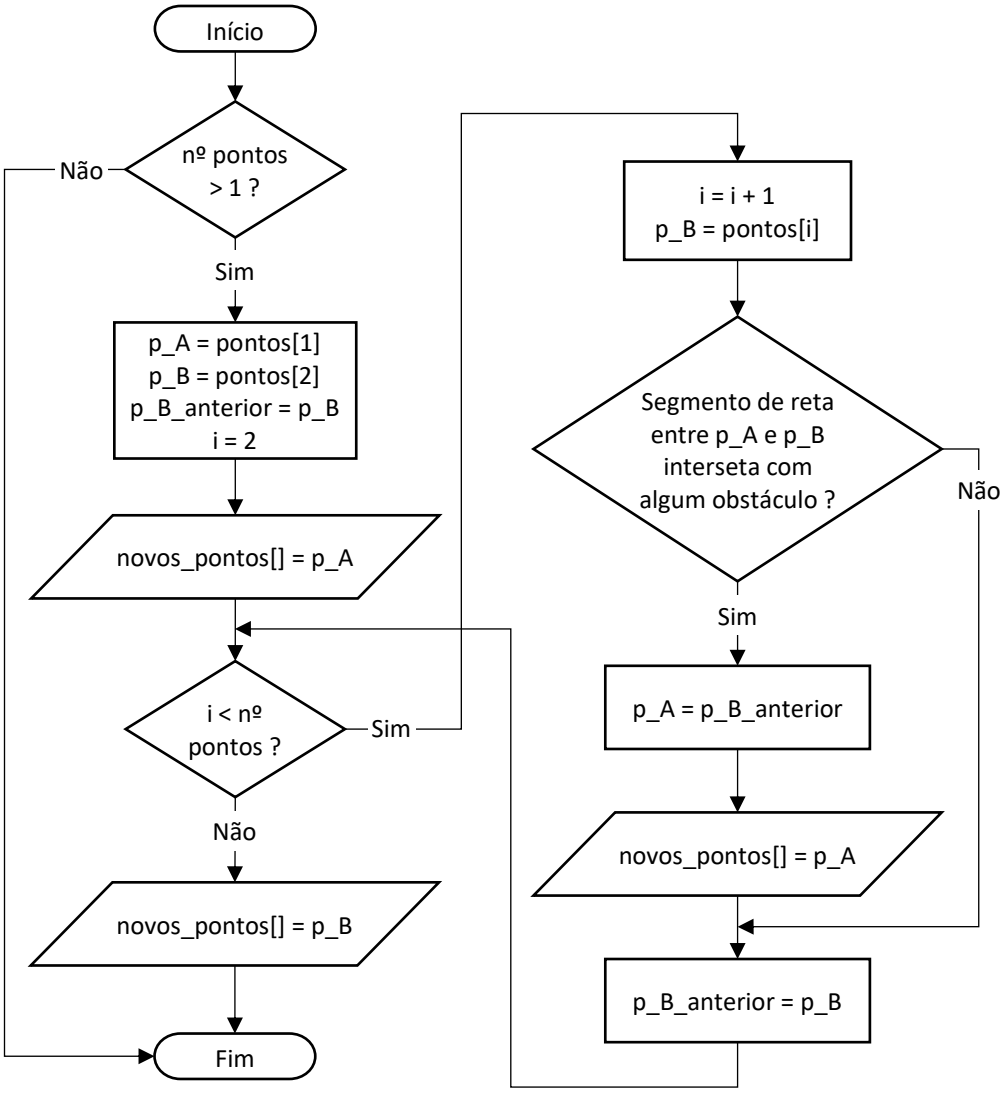


Figura 5.20 – Fluxograma para obter uma trajetória ainda mais curta e simplificada.

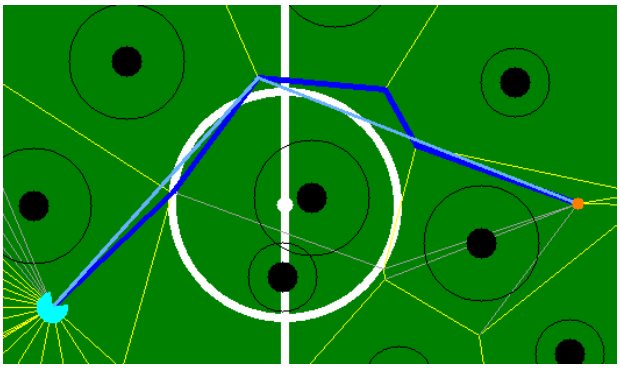


Figura 5.21 - Trajetória ainda mais curta e simplificada, representada pelos 2 segmentos de reta azuis claros.

5.1.6.2. TRAJETÓRIA SUAVE

Neste passo, o objetivo é suavizar a trajetória obtida no passo anterior, pois existe a possibilidade de esta conter cantos afiados. No exemplo apresentado (ver *Figura 5.21* ou *Figura 5.22*) a trajetória simplificada, tem aproximadamente um ângulo de 90º entre os dois segmentos de reta azuis claros, dificultando o movimento do robô para velocidades elevadas.

Para suavizar a trajetória é utilizado o método de curvas paramétricas polinomiais apresentado na secção *3.1.1.7*, sendo escolhida a curva do tipo B-Spline. Este tipo de curva permite obter uma nova trajetória por aproximação aos pontos da trajetória simplificada. Desta forma, a nova trajetória passa apenas pelos pontos inicial e final da trajetória simplificada, sendo aproximada para os restantes. Na *Figura 5.22* é possível observar o resultado da implementação deste passo para o exemplo apresentado, sendo a trajetória suave representada pelos segmentos de reta roxos.

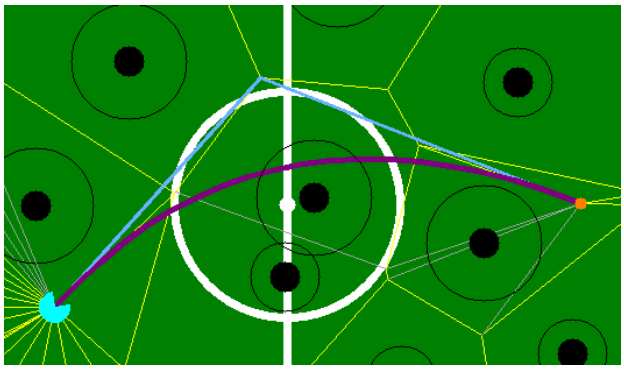


Figura 5.22 – Trajetória suave representada pelos segmentos de reta roxos.

Para se obter a trajetória suave é necessário considerar alguns fatores para a construção da curva, tais como, o grau do polinómio e o número de pontos de amostragem. O grau do polinómio igual a 3, que corresponde a uma curva cúbica, é o que oferece melhores resultados e por isso é o grau utilizado,

no entanto, se o número de pontos da trajetória simplificada for inferior a 4, o grau terá de ser inferior a 3. Isto é, se o número de pontos for igual a 3, é atribuído o grau 2, que corresponde a uma curva quadrática, mas se o número de pontos for igual a 2, é atribuído o grau 1, que corresponde a uma curva linear. No exemplo da *Figura 5.22*, a trajetória suave é uma curva quadrática, pois a trajetória simplificada contém apenas 3 pontos.

Relativamente ao número de pontos de amostragem, este número define a suavidade da curva, pois tal como já foi mencionado na secção *3.1.1.7*, uma curva é representada analiticamente, sendo necessário determinar o número de pontos a serem extraídos da curva calculada, ou seja, a quantidade de amostras. Desta forma, para poucos pontos de amostragem a curva é pouco suave com possíveis cantos afiados, enquanto que para bastantes pontos é possível uma curva suave.

É de salientar que na obtenção da trajetória suave não são considerados os obstáculos.

5.1.6.3. AJUSTAR TRAJETÓRIA

Este passo tem como objetivos ajustar e simplificar a trajetória suave, obtida no passo anterior. O ajuste é feito apenas para o caso em que a trajetória interseta com algum obstáculo, tal como no exemplo apresentado na *Figura 5.23*. Nesta figura, a trajetória com os segmentos de reta cor-de-rosa representa o ajuste feito à trajetória suave, representada pelos segmentos de reta roxos. Desta forma, o ajuste consiste em desviar a trajetória suave para fora da área do obstáculo caso esta intersete com algum. Este ajuste é necessário, pois não interessa ter uma trajetória que faça o robô colidir com os obstáculos.

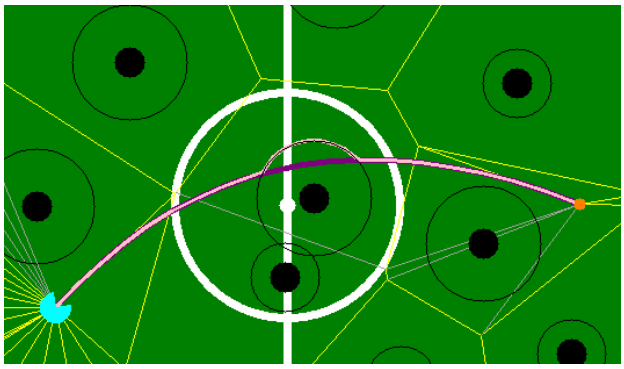


Figura 5.23 – Trajetória ajustada, representada pelos segmentos de reta cor-de-rosa.

Na *Figura 5.24* está representado um desenho para melhor se compreender o algoritmo de ajuste. Este algoritmo numa primeira fase, verifica se a trajetória suave interseta com algum obstáculo, se existir interseção, é feita a pesquisa pelos dois pontos de interseção da trajetória com a circunferência que representa o obstáculo. Estes dois pontos são representados na figura com as cores verde e vermelho, enquanto que a trajetória é representada pelos segmentos de reta roxos.

De seguida, é feita a pesquisa pelos pontos dos segmentos de reta da trajetória no interior do círculo do obstáculo. Em cada ponto, passa uma linha com origem num dos dois pontos de interseção. Desta forma, o conjunto de pontos no interior do círculo é dividido em duas partes. Na primeira metade, passam as linhas com origem no segundo ponto de interseção (ponto e linhas a vermelho na figura), enquanto que na segunda metade, passam as linhas com origem no primeiro ponto de interseção (ponto e linhas a verde na figura). Estas linhas intersetam com a circunferência do obstáculo, existindo assim um ponto de interseção para cada linha. No espaço onde não existem linhas, são adicionados alguns pontos. Contudo, é necessário fazer para cada ponto uma pequena translação para não intersetar com o obstáculo. Com estes novos pontos é construída a trajetória ajustada, sendo esta representada na figura pelos pontos e segmentos de reta a cor-de-rosa.

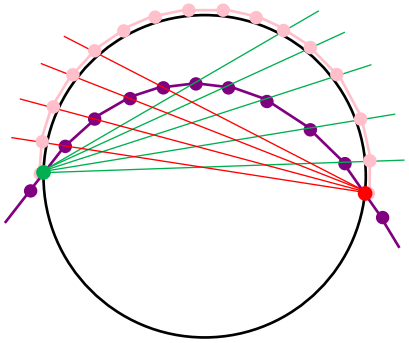


Figura 5.24 – Desenho ilustrativo do ajuste da trajetória, sendo que a trajetória a ser ajustada é representada pelos segmentos de reta roxos e a trajetória ajustada pelos segmentos de reta cor-de-rosa.

O outro objetivo deste passo é simplificar a trajetória suave, independentemente de esta ter sido ajustada ou não. Para isso, foi implementado o mesmo algoritmo apresentado na secção *5.1.6.1*. Na *Figura 5.25* é possível observar o resultado da implementação deste algoritmo para o exemplo apresentado, sendo a trajetória ajustada constituída pelos segmentos de reta cor-de-rosa e a trajetória simplificada pelos segmentos de reta castanhos.

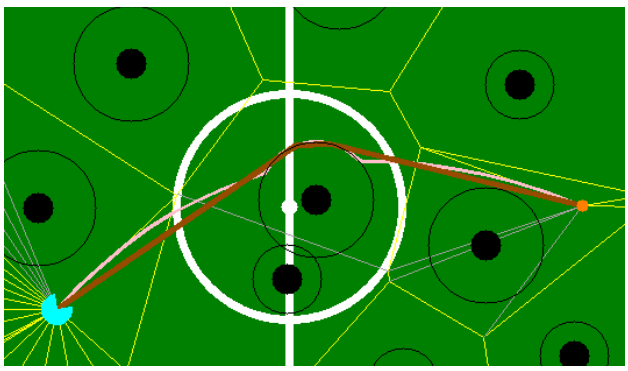


Figura 5.25 – Trajetória simplificada, representada pelos segmentos de reta castanhos.

5.1.6.4. TRAJETÓRIA SUAVE - TRAJETÓRIA MAIS CURTA E SUAVE

Após a construção de algumas trajetórias, a simplificada obtida no passo anterior é a trajetória mais curta possível no espaço de configuração. No entanto, existe a possibilidade de esta conter cantos afiados, dificultando o movimento do robô para velocidades elevadas. Desta forma, é necessário suavizar esta trajetória com a implementação do método de curvas paramétricas polinomiais, tal como na secção *5.1.6.2*. Contudo, foi escolhido outro tipo de curva, que permite obter uma nova trajetória por interpolação aos pontos da trajetória simplificada. Assim, a nova trajetória passa por todos os pontos da trajetória simplificada e os restantes pontos que a constituem são distribuídos com o intuito de formarem uma trajetória suave. Na *Figura 5.26* é possível observar o resultado da implementação deste último passo para o exemplo apresentado, sendo a trajetória mais curta e suave representada pelos segmentos de reta vermelhos.

Tal como na obtenção da trajetória suave na secção *5.1.6.2*, é necessário considerar os fatores, grau do polinómio e o número de pontos de amostragem para a construção da curva. A influência destes dois fatores na construção de uma curva por interpolação é equivalente a uma curva por aproximação.

É de salientar, que na obtenção desta trajetória não são considerados os obstáculos, pois esta segue de perto a trajetória simplificada.

Posto isto, esta trajetória é o melhor compromisso entre a trajetória mais curta e a trajetória mais suave.

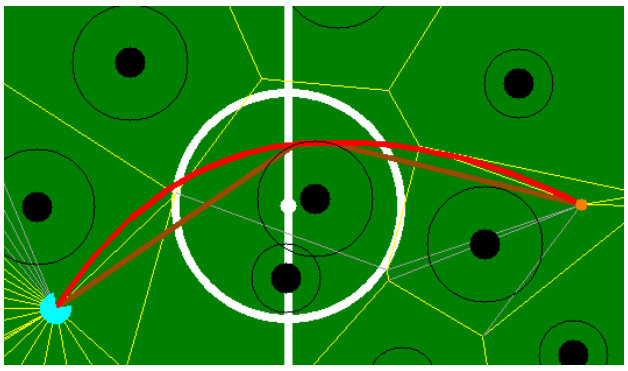


Figura 5.26 – Trajetória mais curta e suave, representada pelos segmentos de reta vermelhos.

5.1.7. VISÃO GERAL DO PLANEAMENTO DA TRAJETÓRIA

Após a apresentação de todos os passos da implementação do processo *Planeamento da Trajetória*, é apresentado nesta secção o fluxograma da visão geral deste processo (ver *Figura 5.27*). Todas as condições e passos apresentados no fluxograma, foram descritos neste subcapítulo. É de

salientar, que a forma entrada/saída *Trajetória Final* representa o vetor com as posições (x, y) que o robô deve percorrer.

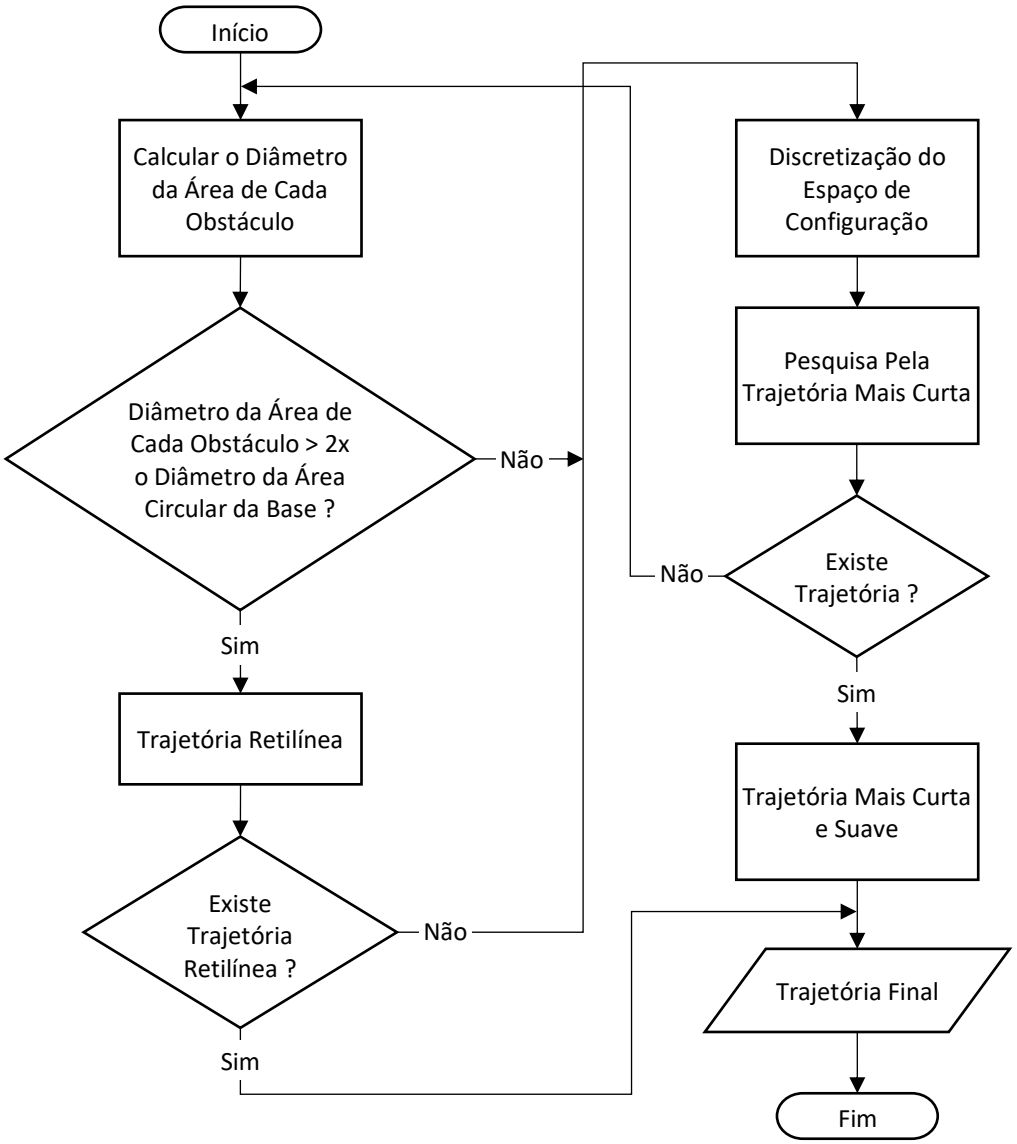


Figura 5.27 – Fluxograma da visão geral do processo *Planeamento da Trajetória*.

5.2. CONTROLO DO MOVIMENTO

Neste subcapítulo serão apresentados os detalhes da implementação deste processo, sendo que o objetivo é calcular os valores a enviar para a placa de controlo dos motores, de modo a que o robô percorra a trajetória obtida no processo *Planeamento da Trajetória*.

Tal como já foi apresentado no *Capítulo 4*, a estrutura mecânica, o *hardware* e o *software* do baixo nível de um robô da Minho Team, garantem um movimento holonómico. Com a placa OMNI3D-MAX é

possível controlar os motores para este movimento, sendo necessário enviar para esta os valores das velocidades linear e angular e também a direção do movimento de translação. Para determinar em tempo real estas três variáveis, foram implementados algoritmos que serão apresentados nas secções seguintes do atual subcapítulo.

Antes de iniciar a descrição dos algoritmos, é necessário ter em consideração o sistema de eixos coordenados absoluto utilizado para o espaço de jogo. As posições dos obstáculos, do robô e da bola são em relação a este referencial. Para o robô também foi definido um sistema de eixos coordenados, sendo este apenas uma translação do anterior para a posição do robô no espaço de jogo, ou seja, é um sistema de eixos coordenados relativo ao robô. A rotação do robô, ou o ângulo do robô, dado pelo nó *Localização* é desde o eixo y do sistema de eixos coordenados do robô, até à parte da frente do robô. Na *Figura 5.28* estão ilustrados os dois sistemas de eixos coordenados.

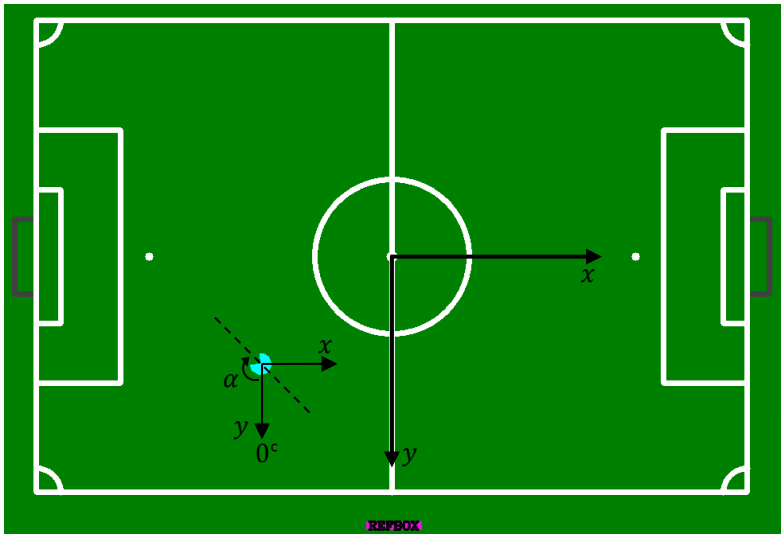


Figura 5.28 - Sistemas de eixos coordenados do espaço de jogo e do robô.

5.2.1. VELOCIDADE ANGULAR

A velocidade angular é determinada para um valor em percentagem da velocidade máxima angular permitida pelo *hardware*, neste caso, pela placa OMNI3D-MAX e motores. Este valor pode ser entre -100 e 100, sendo que na *Figura 5.30* está ilustrado o sentido de rotação do robô para uma velocidade angular menor ou maior que zero.

O algoritmo desenvolvido para determinar o valor da velocidade angular em tempo real, consiste num controlador PID em malha fechada. Este algoritmo tem como entradas o ângulo atual do robô, o ângulo pretendido para o mesmo, a velocidade máxima angular e os valores dos ganhos PID. Na *Figura*

5.29 está representado o fluxograma do algoritmo desenvolvido. É de salientar que o valor do erro resulta do cálculo da diferença entre o ângulo atual do robô e o ângulo pretendido para o mesmo.

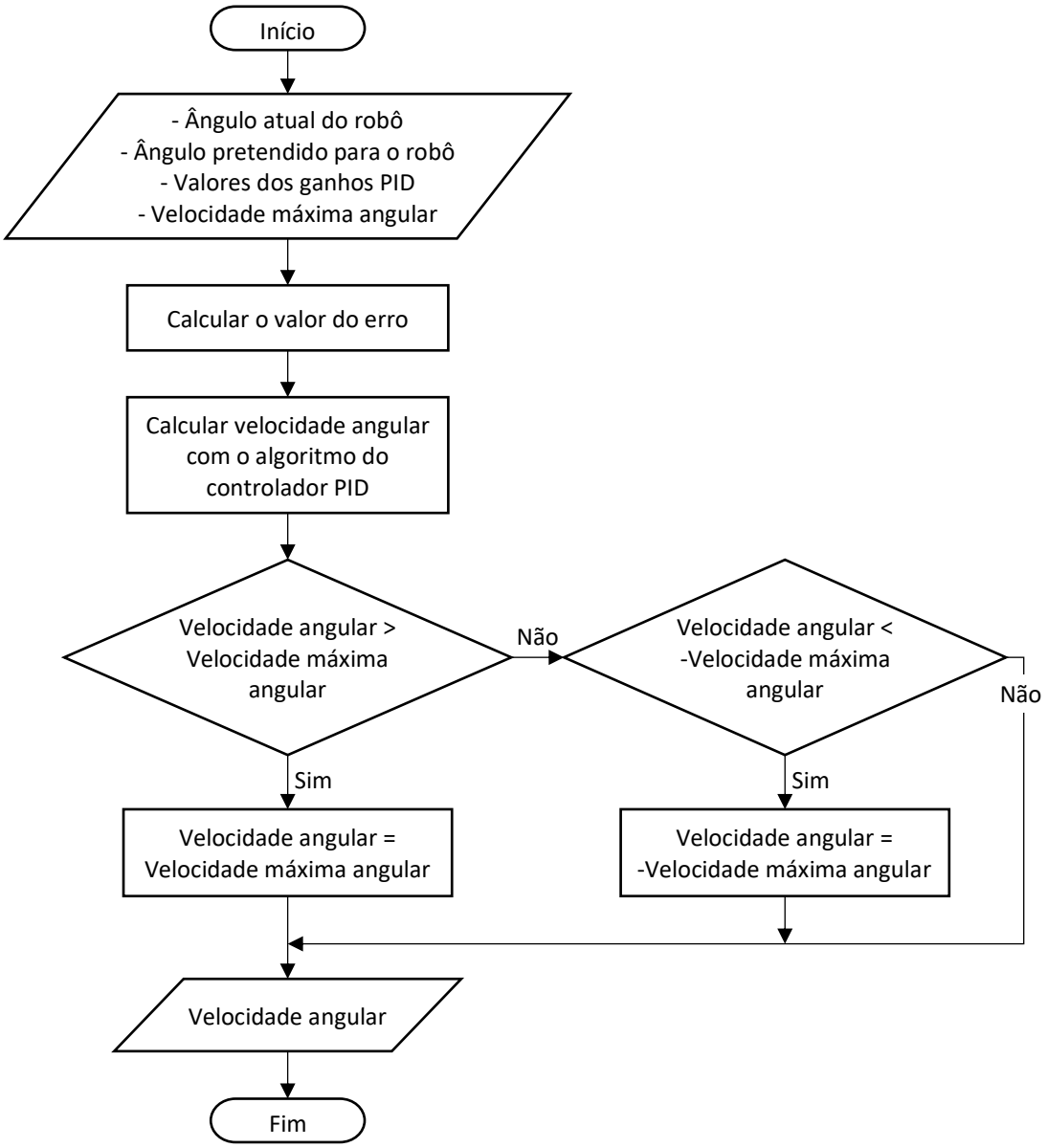


Figura 5.29 - Fluxograma para o cálculo da velocidade angular em tempo real.

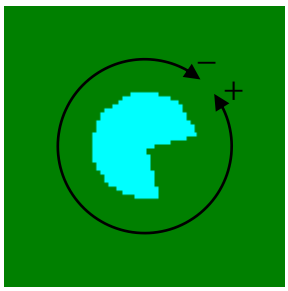


Figura 5.30 - Desenho ilustrativo do sentido de rotação do robô para uma velocidade angular menor ou maior que zero.

5.2.2. VELOCIDADE LINEAR

A velocidade linear é determinada para um valor em percentagem da velocidade máxima linear permitida pelo *hardware* supracitado. A função da biblioteca da placa OMNI3D-MAX, que envia a ordem de movimentação omnidirecional, permite um valor para a velocidade linear entre 0 e 100, sendo que o valor máximo depois de alguns testes realizados com os robôs, foi limitado a 80 no *software* da Minho Team. Isto, de forma a que o movimento numa determinada direção seja mais próximo a uma linha reta.

Para determinar o valor da velocidade linear em tempo real, foram desenvolvidos e testados três algoritmos, sendo usado apenas um. O primeiro algoritmo desenvolvido é com base no cálculo da tangente hiperbólica, o segundo é com base no cálculo do logaritmo e o terceiro consiste num controlador PID em malha fechada. Estes algoritmos têm em comum as fases inicial e final, tal como é possível observar pelo fluxograma apresentado na *Figura 5.31*. Assim, este fluxograma é comum aos três algoritmos, sendo que as diferenças são nos parâmetros de entrada e no passo *Calcular velocidade linear*. Desta forma, nos parâmetros de entrada, os valores dos ganhos PID são apenas para o terceiro algoritmo, enquanto que o passo *Calcular velocidade linear* é onde difere o modo como é calculado o valor da velocidade linear para cada algoritmo desenvolvido.

Nas entradas deste algoritmo, o *Fator velocidade máxima linear* é um valor entre 0 e 1, que é multiplicado pela *Velocidade máxima linear*, sendo definido um valor deste fator para cada ação que o robô tenha de executar. No fluxograma a multiplicação deste fator é feita no passo *Calcular velocidade máxima linear*. Relativamente ao passo *Calcular o valor do erro*, neste é verificada em primeiro lugar a ação a ser executada pelo robô, e de seguida, se a ação implica aproximar e reter a bola com o sistema de manipulação de bola, é subtraída à distância total da trajetória, a distância definida entre o centro da base do robô e da bola. Assim, o valor do erro corresponde ao resultado desta subtração, ou então, se a ação não implica aproximar e reter a bola, o valor do erro corresponderá apenas à distância total da trajetória.

As *Equações 5.1* e *5.2* apresentadas de seguida, são usadas nos dois primeiros algoritmos desenvolvidos. Tal como as equações do terceiro algoritmo, estas são processadas no passo *Calcular velocidade linear*. Assim, no primeiro algoritmo é usada a *Equação 5.1* que corresponde ao cálculo da tangente hiperbólica, enquanto que no segundo é usada a *Equação 5.2* que corresponde ao cálculo do logaritmo. Relativamente às equações usadas no terceiro algoritmo desenvolvido, estas foram descritas na secção *3.1.2*. É de salientar que o valor da velocidade máxima linear, usado neste e nos próximos passos, corresponde ao valor calculado no passo *Calcular velocidade máxima linear*.

No subcapítulo *6.2* será feita a comparação entre os três algoritmos, sendo apresentado os resultados obtidos de algumas trajetórias percorridas pelo robô.

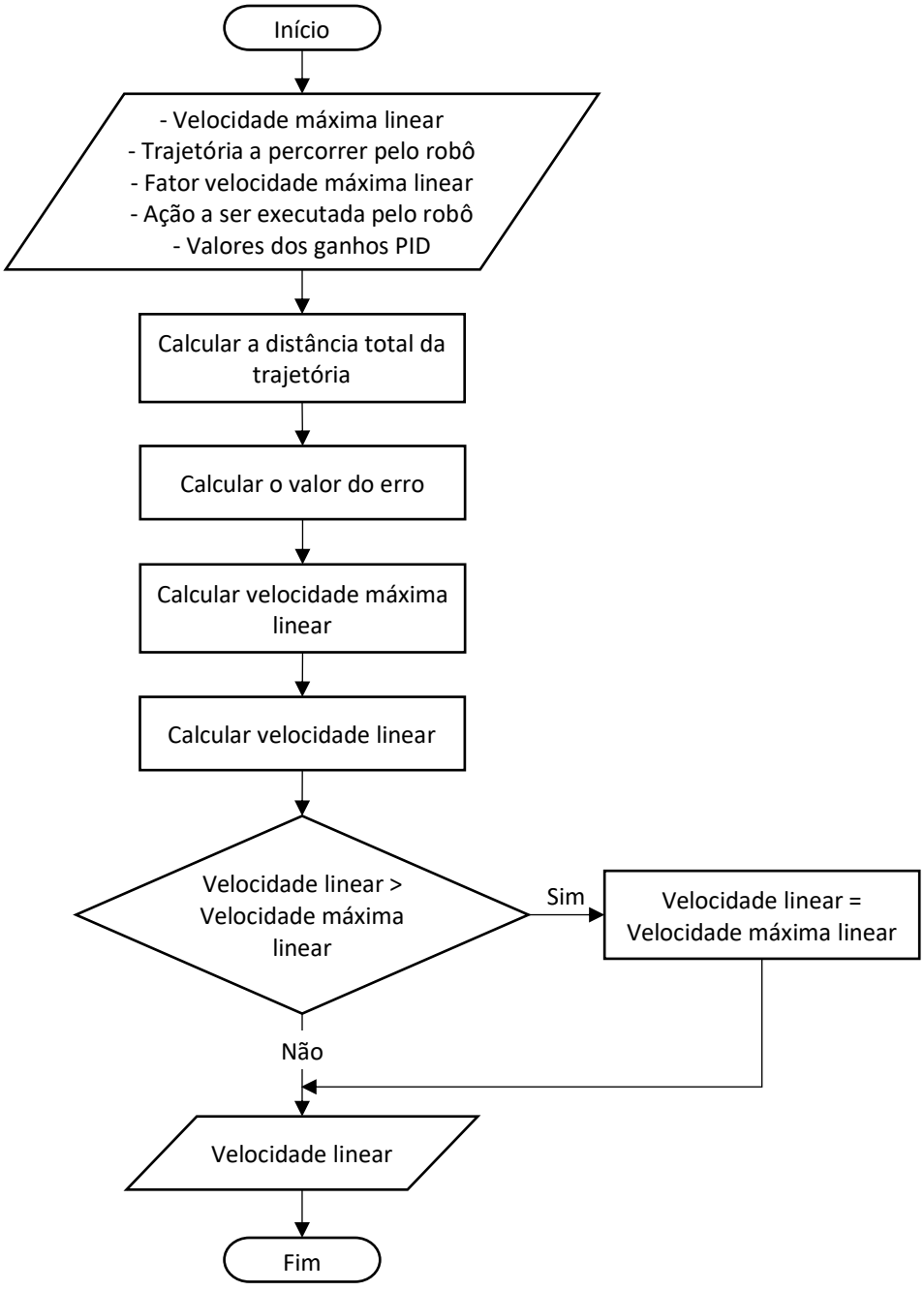


Figura 5.31 – Fluxograma para o cálculo da velocidade linear em tempo real.

Equação usada no primeiro algoritmo desenvolvido:

$$Velocidade\ linear = Velocidade\ máxima\ linear * \tanh(valor\ do\ erro) \quad (5.1)$$

Equação usada no segundo algoritmo desenvolvido:

$$Velocidade\ linear = Velocidade\ máxima\ linear * \log(valor\ do\ erro + 1.0) \quad (5.2)$$

5.2.3. DIREÇÃO DO MOVIMENTO DE TRANSLAÇÃO

A direção do movimento de translação é determinada para um valor entre 0° e 360°, sendo este valor relativo à parte da frente do robô.

Na *Figura 5.32* está ilustrado o sistema de eixos coordenados relativo ao robô e os ângulos azuis escuros correspondentes à direção do movimento de translação. Como é possível observar pela figura, para o robô se mover numa direção relativa ao sistema de eixos coordenados do robô, sem qualquer rotação para direcionar a parte da frente do robô nessa direção, é necessário calcular o ângulo da direção do movimento de translação. Um exemplo para compreender melhor, é considerar que na figura o robô está desfasado 300° em relação ao eixo y, e a direção pretendida para o movimento, em relação ao eixo y tem um ângulo de 90°. Neste caso, o ângulo para a direção do movimento de translação do robô é de 150°.

O algoritmo desenvolvido para calcular a direção do movimento de translação em tempo real, está representado no fluxograma da *Figura 5.33*. Nas entradas deste algoritmo, o *Ângulo pretendido para o movimento* corresponde à direção pretendida para o movimento em relação ao eixo y. Os cálculos e a condição apresentados no fluxograma efetuam a normalização do ângulo, de modo a obter um valor entre 0° e 360°.

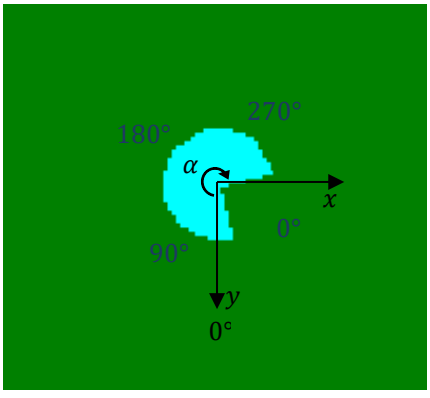


Figura 5.32 - Desenho ilustrativo do sistema de eixos coordenados para o robô e representação dos ângulos azuis escuros, correspondentes à direção do movimento de translação.

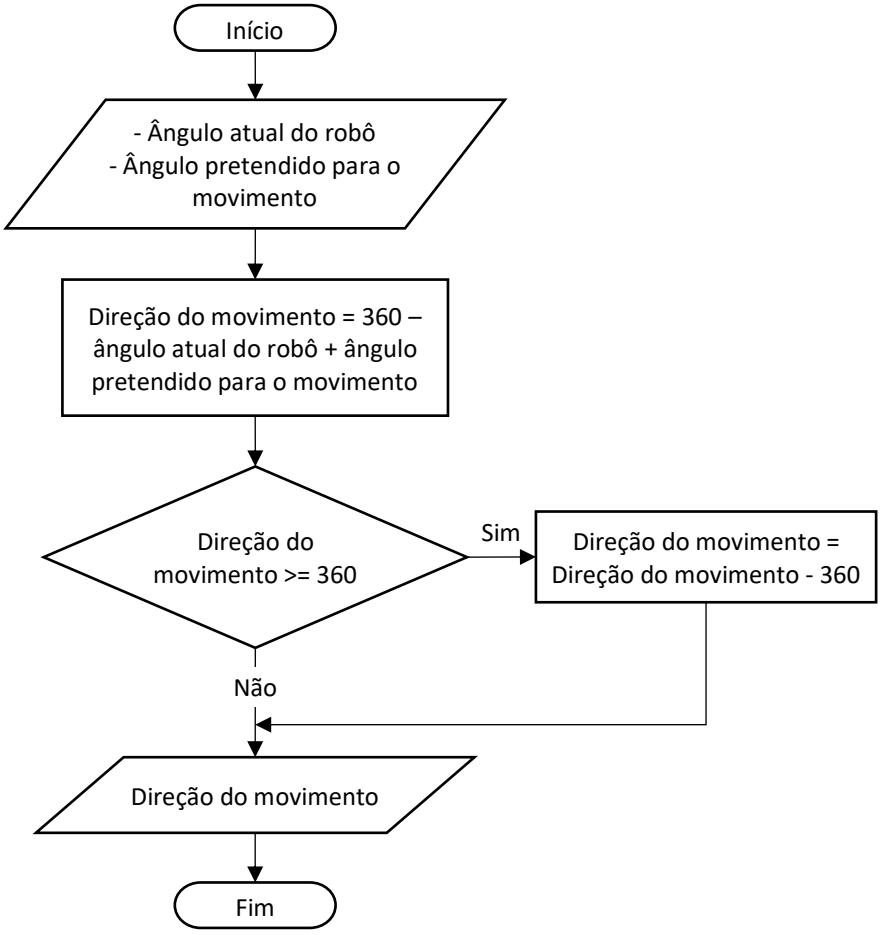


Figura 5.33 - Fluxograma para o cálculo da direção do movimento de translação em tempo real.

Capítulo 6

RESULTADOS

Este capítulo é constituído por quatro subcapítulos, onde serão apresentados os resultados obtidos após a implementação de todos os passos dos dois processos já apresentados. Assim, no primeiro subcapítulo é feita uma análise ao desempenho do planeamento da trajetória, sendo que no segundo subcapítulo serão apresentados exemplos da evolução do erro para os movimentos de rotação e de translação do robô, após o ajuste das variáveis dos algoritmos implementados. Posteriormente, no terceiro subcapítulo será apresentada a trajetória percorrida pelo robô em alguns exemplos do espaço de configuração do espaço de jogo, e também por dois robôs em simultâneo. Para finalizar, no quarto subcapítulo será feita a análise da participação no Festival Nacional de Robótica 2017.

Os resultados apresentados nos três primeiros subcapítulos foram obtidos usando o simulador da Minho Team, num sistema com um processador Intel Core i5 a 2,4 GHz, 4 GB de memória RAM, e com o sistema operativo Ubuntu 16.04 de 64 bits.

6.1. DESEMPENHO DO PLANEAMENTO DA TRAJETÓRIA

Como foi implementado e testado apenas um único algoritmo para cada passo deste processo, não sendo possível desta forma comparar com outros algoritmos existentes, o desempenho será assim analisado pelo tempo médio de 100 amostras.

As dimensões do campo de jogo são as oficiais, já apresentadas no primeiro capítulo, e os obstáculos são 9 no total, sendo que destes, 5 são obstáculos de robôs adversários com 2 m de diâmetro na área de cada obstáculo e 4 são obstáculos de robôs cooperativos com 1,2 m de diâmetro.

De seguida, será apresentado um exemplo para cada passo do planeamento da trajetória, onde é possível observar o espaço de configuração e o resultado obtido após a implementação do respetivo algoritmo. O tempo médio de 100 amostras, registado em cada passo, corresponde ao resultado do respetivo exemplo apresentado.

Na *Figura 6.1* é possível observar a trajetória retilínea obtida para o exemplo do espaço de configuração apresentado.

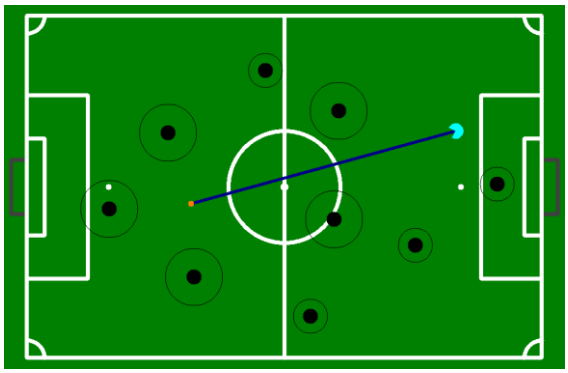


Figura 6.1 - Trajetória retilínea.

Neste passo, para este exemplo do espaço de configuração, o tempo médio de processamento para obter a trajetória retilínea foi de 0,292 ms. Considerando a complexidade dos algoritmos implementados nos próximos passos, este tempo é bastante baixo.

De seguida, é apresentado na *Figura 6.2* um exemplo da discretização do espaço de configuração, onde é possível observar o diagrama de Voronoi para as posições de 9 obstáculos reais e dos obstáculos artificiais, sobrepostos às linhas dos limites do campo. Tal como apresentado na secção *5.1.5*, os obstáculos artificiais não são apresentados no *visualizer*, porém as posições destes são sempre incluídas na construção do diagrama de Voronoi.

O tempo médio de processamento obtido no exemplo apresentado foi de 5,156 ms. Este tempo poderia ser muito menor se não tivesse os obstáculos artificiais, que são em grande número e que fazem o diagrama ter muitas mais arestas e vértices. Contudo, estes são necessários para que existam segmentos de reta à volta de cada obstáculo.

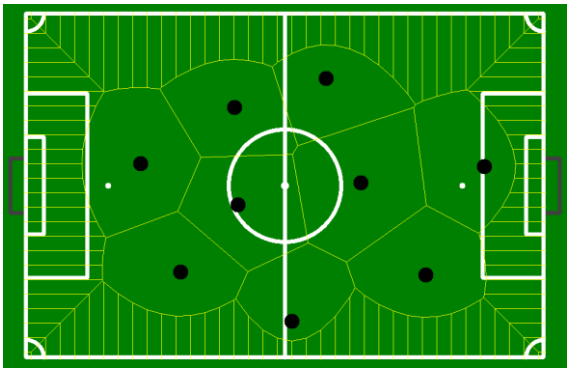
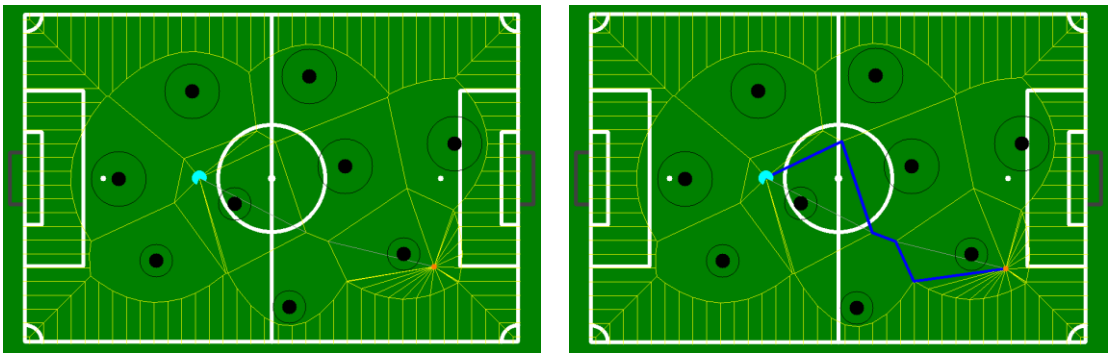


Figura 6.2 - Discretização do espaço de configuração através do diagrama de Voronoi.

O algoritmo implementado no próximo passo do planeamento, que faz a pesquisa pela trajetória mais curta, foi analisado em duas etapas. A primeira etapa corresponde à construção de um grafo não direcionado, já mencionado na secção *5.1.5*, enquanto que a segunda etapa consiste na pesquisa pela

trajetória mais curta através do algoritmo de Dijkstra. Posto isto, na *Figura 6.3(a)* é possível observar o resultado de um exemplo da primeira etapa, onde o tempo médio de processamento obtido foi de 88,057 ms. Este algoritmo tem um tempo médio de processamento muito mais elevado que todos os outros algoritmos implementados neste trabalho. Isto deve-se ao facto de este verificar para cada segmento de reta, se existe interseção com algum obstáculo, e também do tempo que demora para construir o objeto do grafo.

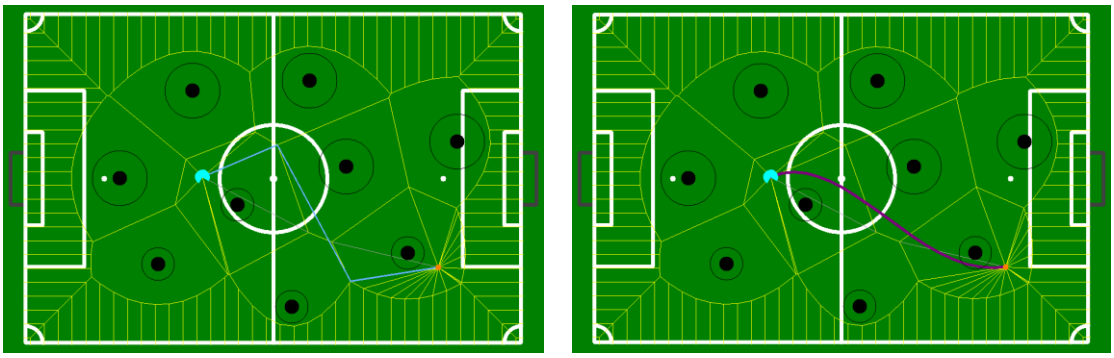
Relativamente à segunda etapa, na *Figura 6.3(b)* é possível observar a trajetória mais curta entre a posição atual do robô e a posição pretendida para o mesmo, no exemplo do espaço de configuração apresentado. O tempo médio de processamento obtido nesta etapa foi de 12,367 ms.



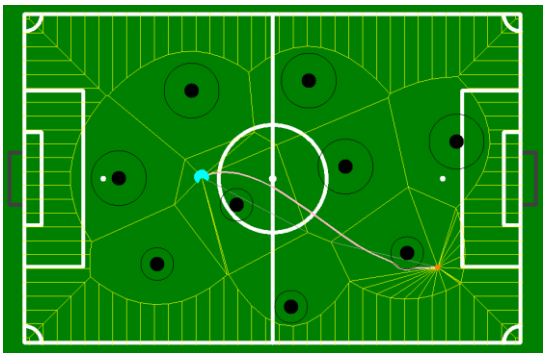
(a) (b)

Figura 6.3 - Pesquisa pela trajetória mais curta. (a) Grafo com verificação da interseção dos segmentos de reta com os obstáculos. (b) Trajetória mais curta representada pelos segmentos de reta azuis.

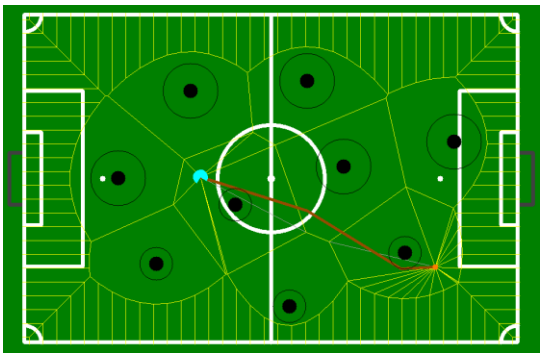
O último passo do planeamento, tal como já foi apresentado na secção *5.1.6*, tem como objetivo suavizar a trajetória encontrada pelo algoritmo de Dijkstra e obter, ao mesmo tempo, a trajetória mais curta no espaço de configuração livre. Na *Figura 6.4* é possível observar as cinco trajetórias obtidas neste último passo, para o mesmo exemplo do espaço de configuração do passo anterior e para as mesmas posições (do robô e da pretendida para este). O tempo médio de processamento obtido neste último passo do planeamento, para o exemplo apresentado na figura, foi de 3,279 ms.



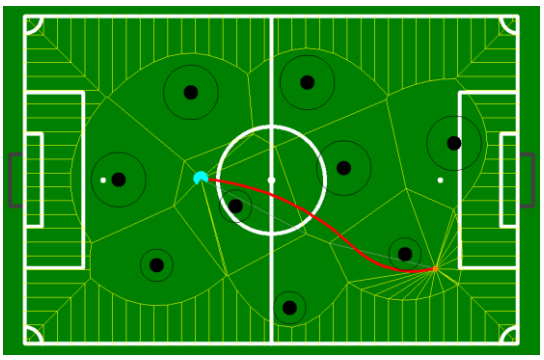
(a) (b)



(c)



(d)



(e)

Figura 6.4 - Trajetórias dos algoritmos implementados, para obter a trajetória mais curta e suave. (a) Trajetória mais curta simplificada. (b) Trajetória suave. (c) Trajetória ajustada. (d) Trajetória simplificada. (e) Trajetória mais curta e suave.

De seguida, será apresentado na *Tabela 6.1* um resumo dos tempos médios de processamento para os exemplos apresentados, e também para a função que calcula o diâmetro da área de cada obstáculo.

Tabela 6.1 – Tempos médios de processamento do planeamento da trajetória para os exemplos apresentados.

Algoritmo/Passo do Planeamento da Trajetória	**Tempo Médio de Processamento (ms)**
Calcular o Diâmetro da Área de Cada Obstáculo	0,015
Trajetória Retilínea	0,292
Discretização do Espaço de Configuração	5,156
Pesquisa Pela Trajetória Mais Curta (1ª etapa)	88,057
Pesquisa Pela Trajetória Mais Curta (2ª etapa)	12,367
Trajetória Mais Curta e Suave	3,279
Tempo Total:	109,166

Na *Figura 6.5* é possível observar a trajetória mais curta e suave para outro exemplo do espaço de configuração, onde as diferenças para o exemplo anterior é nas posições dos obstáculos, na posição do robô e na posição pretendida para este. O tempo médio de processamento obtido em todos os passos do planeamento da trajetória, para este exemplo, foi de 108,187 ms.

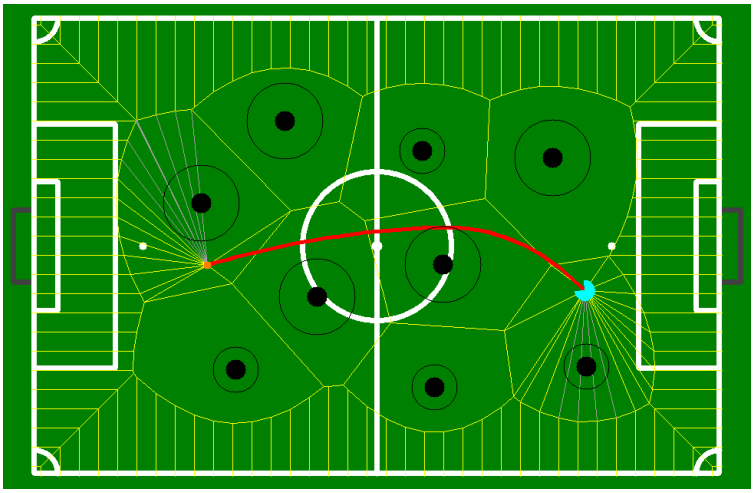


Figura 6.5 – Trajetória mais curta e suave.

Tal como é possível verificar pelos tempos médios obtidos nos dois exemplos, estes são muito próximos. Pois o que muda de um exemplo para outro é apenas as posições dos obstáculos, do robô e da posição pretendida. A dispersão dos obstáculos no espaço de configuração é idêntica, o que faz com que a quantidade de segmentos de reta também seja muito próxima. Com isto a complexidade de processamento no passo *Pesquisa Pela Trajetória Mais Curta* é equivalente.

Noutros testes efetuados, com menos ou mais obstáculos, com outras dispersões ou com diâmetros das áreas dos obstáculos diferentes, verificaram-se tempos menores, mas também se verificou o oposto, pois depende totalmente do espaço de configuração.

Os primeiros testes efetuados no simulador da Minho Team, logo após a implementação de todos os algoritmos, levantaram algumas dúvidas relativamente ao tempo médio de processamento do planeamento da trajetória, que poderia ser elevado para o controlo do movimento. Contudo, após testes com os robôs reais verificou-se que este seria o caminho a seguir. Pois tal como foi apresentado na secção *4.2.1.2*, o tempo necessário para a OMNI3D-MAX executar uma ordem de movimentação pode ir até 100 ms. No entanto, no futuro deve-se ter em conta uma possível melhoria na 1ª etapa do passo *Pesquisa Pela Trajetória Mais Curta*.

6.2. MOVIMENTO APÓS AJUSTES DOS PARÂMETROS

Os gráficos apresentados de seguida, correspondem ao valor do erro durante o teste realizado a cada algoritmo apresentado no subcapítulo *5.2*. Estes algoritmos fazem o cálculo das três variáveis necessárias para o movimento holonómico, enviadas em tempo real para a placa de controlo dos motores. Desta forma, são controlados os movimentos de rotação e de translação, e também a direção do movimento de translação.

Na *Figura 6.6* é possível observar o valor do erro angular durante a rotação do robô. Isto significa que o valor do erro num instante de tempo, corresponde à diferença entre o ângulo do robô nesse instante e o ângulo pretendido para o robô. Através da análise deste gráfico, é possível afirmar que o erro inicial neste teste era de aproximadamente -170°. Com o valor da velocidade angular determinado em tempo real, o robô roda até que fique direcionado no ângulo pretendido, ou seja, até que o erro seja igual a zero ou muito próximo. Neste teste, os ganhos do controlador PID foram ajustados, de modo a que o robô rodasse o mais rápido possível e sem grandes oscilações.

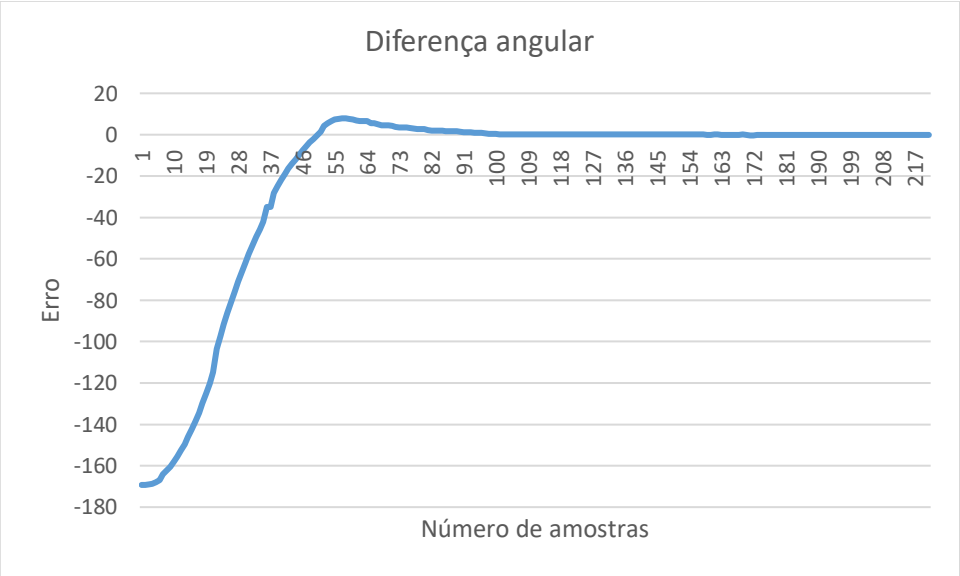


Figura 6.6 – Valor do erro angular durante a rotação do robô.

Nos gráficos apresentados nas figuras *6.7*, *6.8* e *6.9*, é possível observar a distância da trajetória durante a translação do robô até à posição pretendida. Assim, o valor do erro num instante de tempo, corresponde à distância da trajetória entre a posição do robô nesse instante e a posição pretendida para o robô. Através da análise destes três gráficos, é possível afirmar que o erro inicial, nestes testes, era de aproximadamente 7,5 m. Com o valor da velocidade linear determinado em tempo real, o robô move-se até chegar à posição pretendida, ou seja, até que o erro seja igual a zero ou muito próximo.

Para determinar o valor da velocidade linear nestes testes, foram usados os algoritmos apresentados na secção *5.2.2*. Assim, a linha tracejada azul do gráfico da *Figura 6.7* corresponde ao algoritmo com base no cálculo da tangente hiperbólica, a linha de pontos laranja do mesmo gráfico corresponde ao algoritmo com base no cálculo do logaritmo, e as duas linhas do gráfico da *Figura 6.8* correspondem ao algoritmo com base num controlador PID em malha fechada.

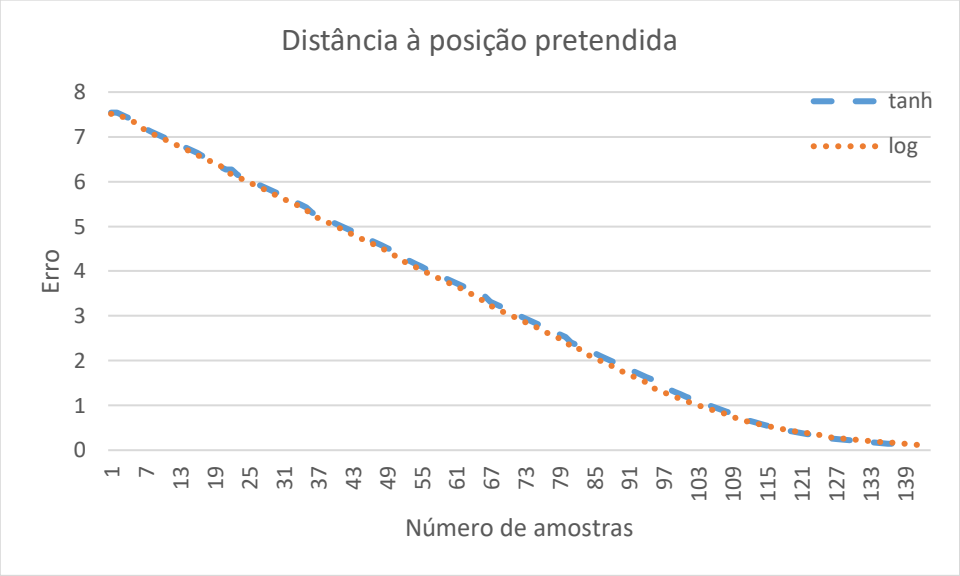


Figura 6.7 – Distância da trajetória durante a translação do robô até à posição pretendida, sendo que *Erro* corresponde a essa distância. A linha *tanh* corresponde ao algoritmo com base no cálculo da tangente hiperbólica e a linha *log*, ao algoritmo com base no cálculo do logaritmo.

Considerando que para estes testes foi usada a mesma velocidade máxima linear e o mesmo fator multiplicativo para a velocidade máxima linear, é possível analisar a evolução do valor do erro entre gráficos e consequente comparação entre algoritmos.

Analisando a evolução do valor do erro entre os dois algoritmos do gráfico da *Figura 6.7*, é possível afirmar que os resultados são muito idênticos. Os tempos obtidos nestes dois testes também comprovam esta análise, pois no teste correspondente a *tanh* o robô demorou 4,63 s a percorrer a trajetória, e no teste correspondente a *log* demorou 4,79 s. Relativamente à evolução do valor do erro entre os dois testes do gráfico da *Figura 6.8*, é possível afirmar que têm um resultado diferente entre eles. Pois o algoritmo com base num controlador PID em malha fechada, usado nestes dois testes, permite ajustar a velocidade para um comportamento de jogo, mais ou menos "agressivo". Desta forma, no teste correspondente à linha *PID_Teste1*, o robô moveu-se mais rapidamente para a posição pretendida, que no teste correspondente à linha *PID_Teste2*. Os tempos obtidos também comprovam esta análise, pois no teste correspondente à linha *PID_Teste1* o robô demorou 4,43 s a percorrer a trajetória e no teste

correspondente à linha *PID_Teste2* demorou 5,39 s. Esta diferença deve-se aos ajustes feitos aos ganhos do controlador PID, sendo também possível obter resultados diferentes, através da alteração do fator velocidade máxima linear.

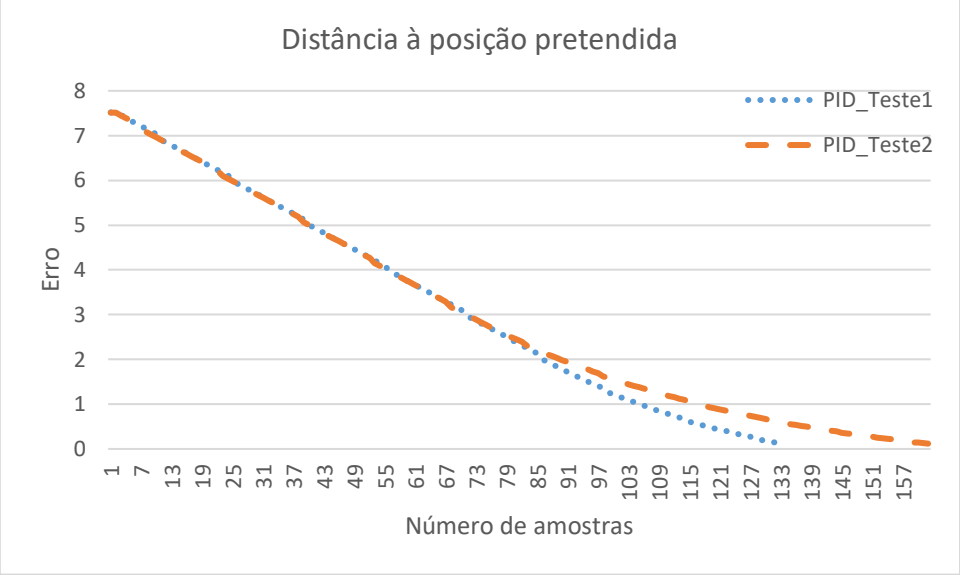


Figura 6.8 – Distância da trajetória durante a translação do robô até à posição pretendida, sendo que *Erro* corresponde a essa distância. As duas linhas apresentadas, correspondem a dois testes efetuados ao algoritmo com base num controlador PID em malha fechada.

Na *Figura 6.9* é possível observar num único gráfico os mesmos resultados já apresentados, correspondentes a *tanh*, *PID_Teste1* e *PID_Teste2*.

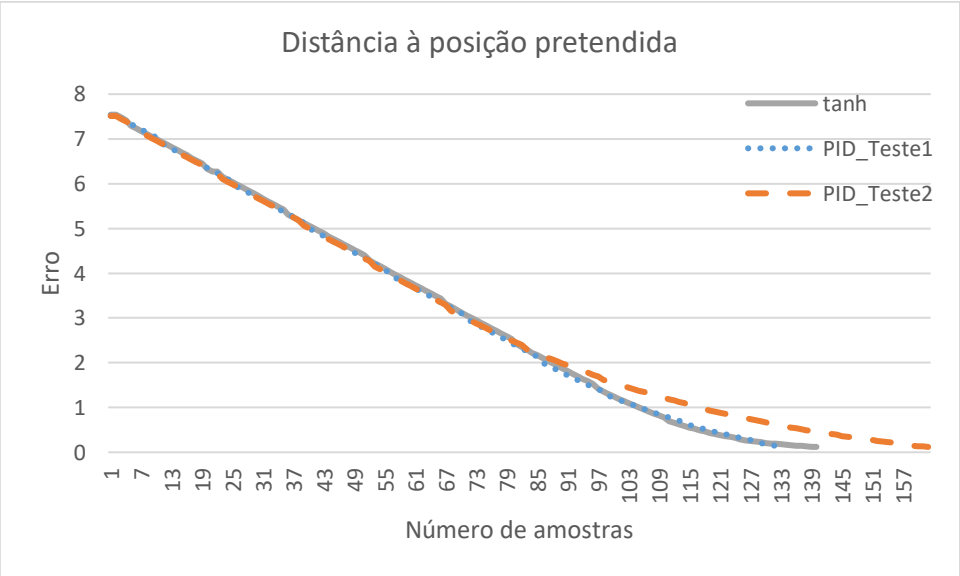


Figura 6.9 – Gráfico com os mesmos resultados já apresentados nos dois gráficos anteriores. Apenas a linha *log* não é apresentada.

Posto isto, o algoritmo com base num controlador PID em malha fechada, foi o algoritmo usado na implementação final para o cálculo do valor da velocidade linear. Pois, dos três algoritmos este é bastante flexível no ajuste, permitindo ajustar para diferentes comportamentos de jogo. Nos testes realizados no simulador e nos robôs reais, este foi o algoritmo mais eficaz.

6.3. TRAJETÓRIAS PERCORRIDAS

Neste subcapítulo serão apresentadas as trajetórias percorridas pelo robô em diferentes exemplos do espaço de configuração, e também as trajetórias percorridas em simultâneo por dois robôs no mesmo espaço de configuração.

No algoritmo desenvolvido para o controlo do movimento, foram determinados dois modos de movimento que dependem da ação que o robô tenha de executar. Desta forma, um dos modos é quando a ação implica mover o robô para se aproximar da bola e retê-la, o outro é quando a ação implica mover o robô para uma posição e direcionar a parte da frente numa determinada direção.

Na *Figura 6.10* é possível observar a trajetória mais curta e suave entre a posição atual do robô e a posição pretendida para o mesmo, no exemplo do espaço de configuração apresentado.

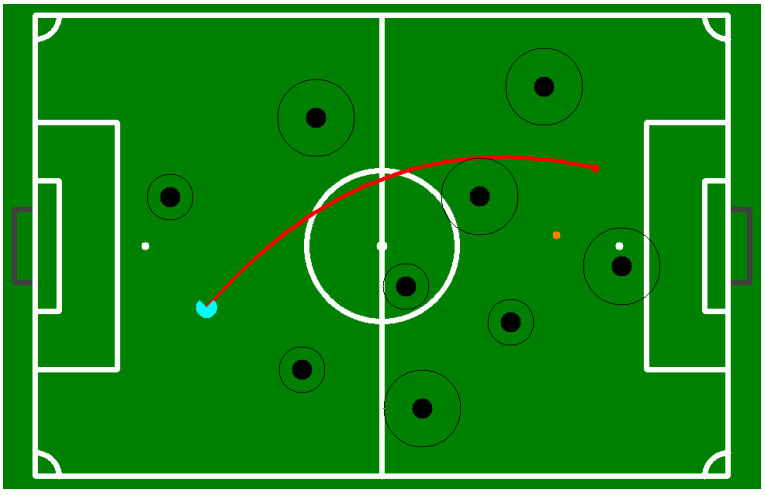


Figura 6.10 – Trajetória mais curta e suave. A posição pretendida para o robô é representada pelo ponto vermelho e a direção pelo ponto laranja.

No teste efetuado, o robô percorreu esta trajetória tal como apresentado na *Figura 6.11*. O modo de movimento usado foi o modo "sem bola", por isso o robô para percorrer a trajetória não fez qualquer movimento de rotação durante o movimento de translação até à posição pretendida (ponto vermelho na figura). Quando o robô atingiu a posição, o valor da velocidade linear foi a zero, e da velocidade de rotação diferente de zero até que ficasse direcionado para o ponto laranja representado na figura. Também é

possível observar um pequeno desvio na trajetória percorrida em relação à trajetória obtida inicialmente. Este desvio deve-se ao facto de a trajetória ser planeada constantemente em tempo real, e assim, ser atualizada durante o movimento de translação. Pela forma da trajetória percorrida, é possível afirmar que a trajetória passou a ser retilínea, no momento em que deixou de existir qualquer obstáculo a intersetar com a linha reta entre a posição atual do robô e a posição pretendida. O movimento dos obstáculos, não tendo sido verificado neste teste, também faz alterar a trajetória.

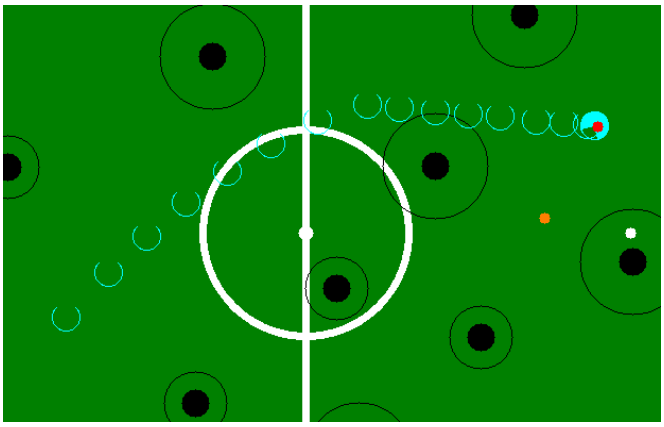


Figura 6.11 – Trajetória percorrida pelo robô relativa à trajetória inicial apresentada na *Figura 6.10*.

Na *Figura 6.13* é apresentado o resultado de dois testes efetuados, onde o robô percorreu a trajetória apresentada na *Figura 6.12*. Nestes dois testes foi usado o modo de movimento "com bola" e por isso o robô durante o movimento de translação até à posição pretendida, neste caso a posição da bola (ponto laranja na figura), rodou nos dois sentidos, direcionando a parte da frente na direção do movimento de translação. Através da análise das trajetórias percorridas, é possível afirmar que no teste correspondente à *Figura 6.13(b)*, o robô fez um pequeno desvio em relação à trajetória planeada inicialmente, não tendo sido verificado este desvio no teste correspondente à *Figura 6.13(a)*. Esta diferença deve-se ao facto de ter sido atribuído 80 ao valor da velocidade máxima linear no teste *(b)* e 50 no teste *(a)*. Também foi verificado, que a execução dos movimentos de rotação e de translação em simultâneo, contribui significativamente para desvios na trajetória.

Nas figuras *6.14* e *6.15*, são apresentados os resultados dos dois testes efetuados para os dois tipos de comportamentos, já apresentados na secção *5.1.5*. Através da análise das trajetórias percorridas, é possível afirmar que a trajetória do comportamento correspondente à *Figura 6.15*, é mais curta que a trajetória do comportamento correspondente à *Figura 6.14*. No entanto, a probabilidade de o robô colidir com os obstáculos é maior. É de salientar que a posição pretendida para o robô nestes dois testes corresponde à posição da bola, que se encontra na posse de um robô adversário.

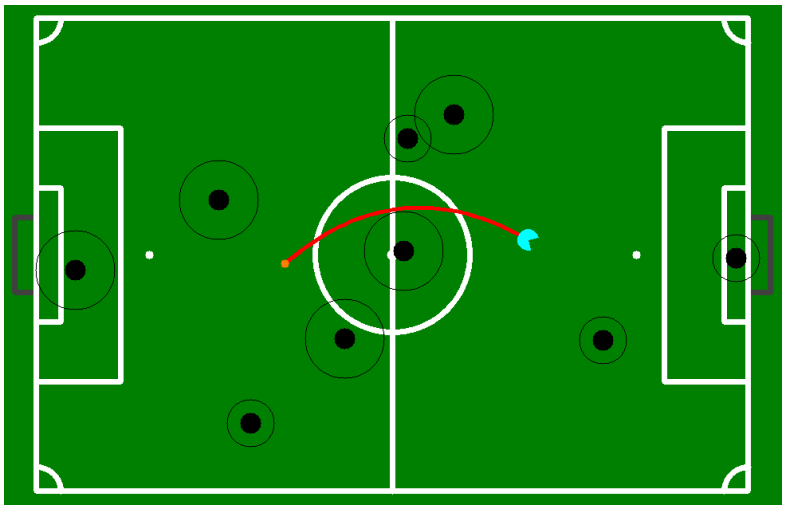
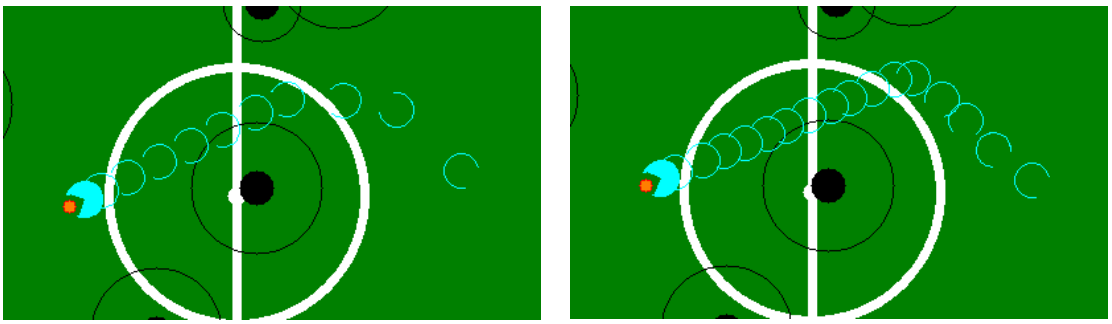
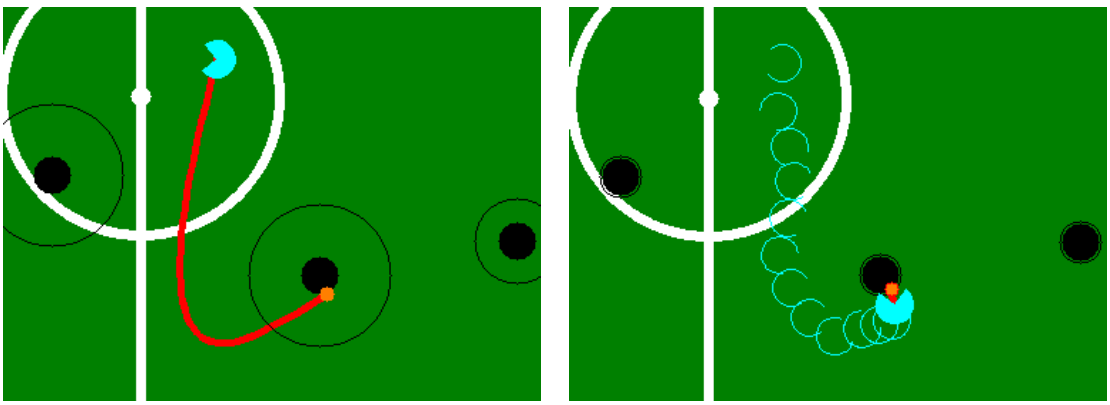


Figura 6.12 – Trajetória mais curta e suave. O ponto laranja representa a posição da bola.



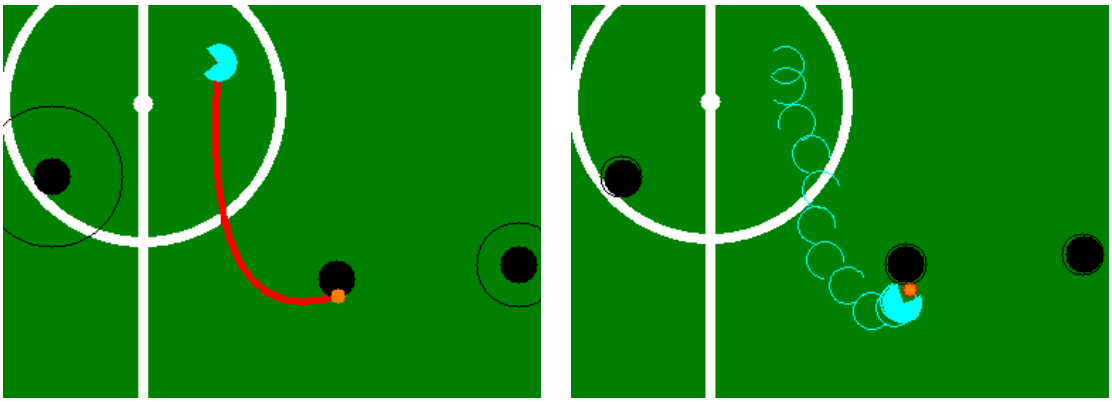
(a) (b)

Figura 6.13 – Trajetórias percorridas pelo robô relativas à trajetória inicial apresentada na *Figura 6.12*.



(a) (b)

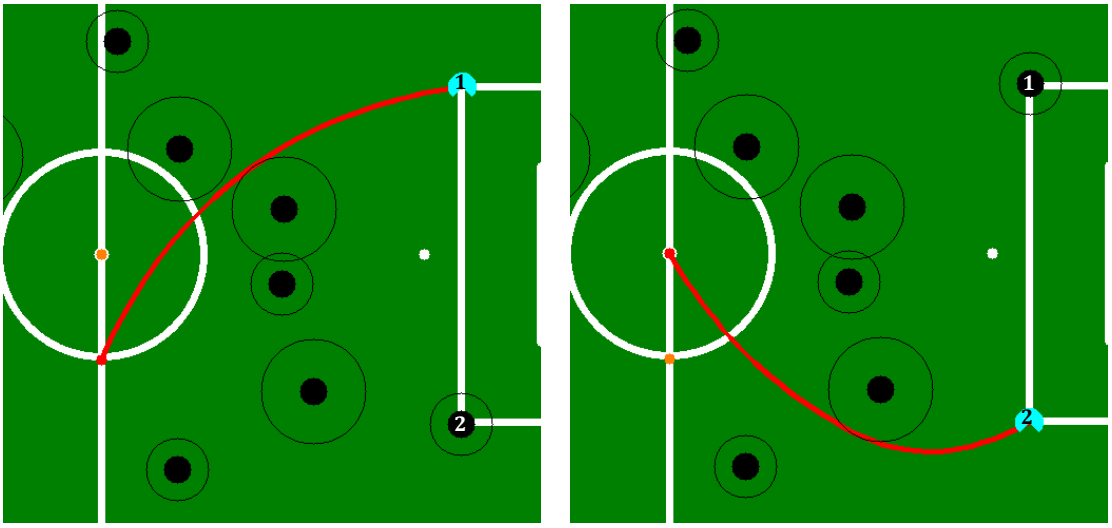
Figura 6.14 – Tipo de comportamento do robô. (a) Trajetória mais curta e suave. (b) Trajetória percorrida pelo robô.



(a) (b)

Figura 6.15 – Tipo de comportamento do robô. (a) Trajetória mais curta e suave. (b) Trajetória percorrida pelo robô.

De seguida, serão apresentados os resultados de três testes efetuados, sendo que em cada teste, dois robôs percorreram duas trajetórias em simultâneo, ou seja, cada robô percorreu a sua trajetória no mesmo espaço de configuração. Na *Figura 6.16* é possível observar as duas trajetórias (uma para cada robô), e os robôs *1* e *2*. É de salientar que o robô *2* é um obstáculo para o robô *1* (ver *Figura 6.16(a)*) e o robô *1* é um obstáculo para o robô *2* (ver *Figura 6.16(b)*). O modo de movimento usado nos três testes foi o modo "sem bola".



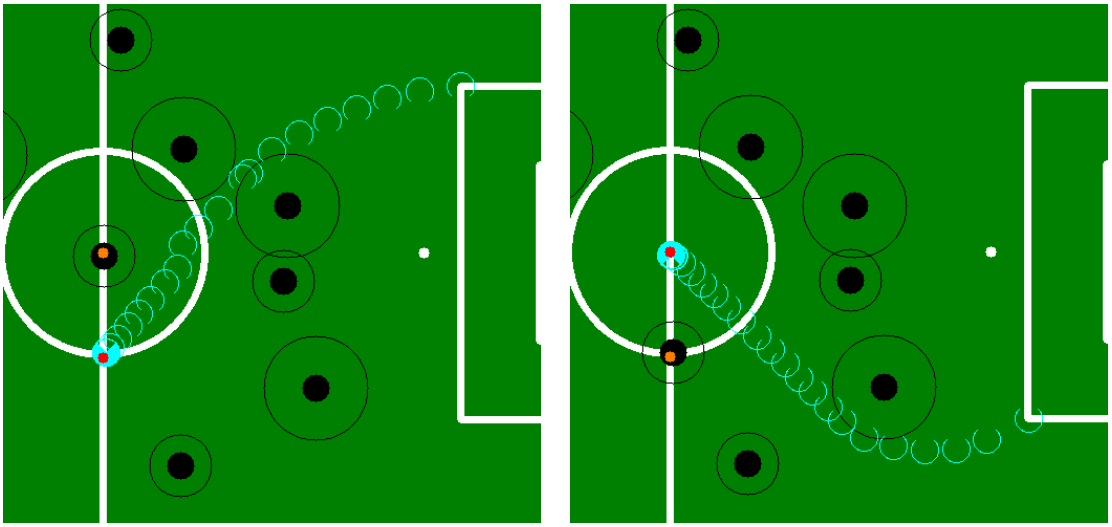
(a) (b)

Figura 6.16 – Trajetória mais curta e suave para cada robô (robôs *1* e *2*).

O resultado do primeiro teste é apresentado na *Figura 6.17*. Para este teste foi atribuído 50 ao valor da velocidade máxima linear dos dois robôs.

Através da análise das trajetórias percorridas, é possível afirmar que o robô *1* fez um pequeno desvio quando entrou no círculo situado no centro do campo. Este desvio foi originado pela passagem

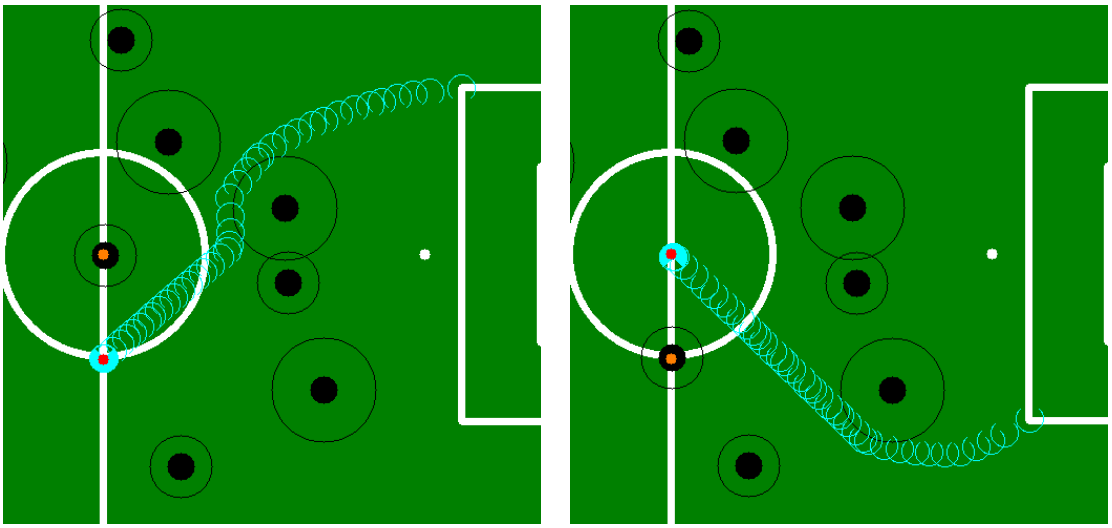
do robô *2* naquela zona do campo, pois o robô *1* considerou o robô *2* um obstáculo. As trajetórias apresentadas na *Figura 6.16*, têm uma distância semelhante, no entanto, o robô *2* chegou primeiro aquela zona do campo, pois o robô *1* iniciou o movimento aproximadamente dois segundos depois do robô *2*.



(a) (b)

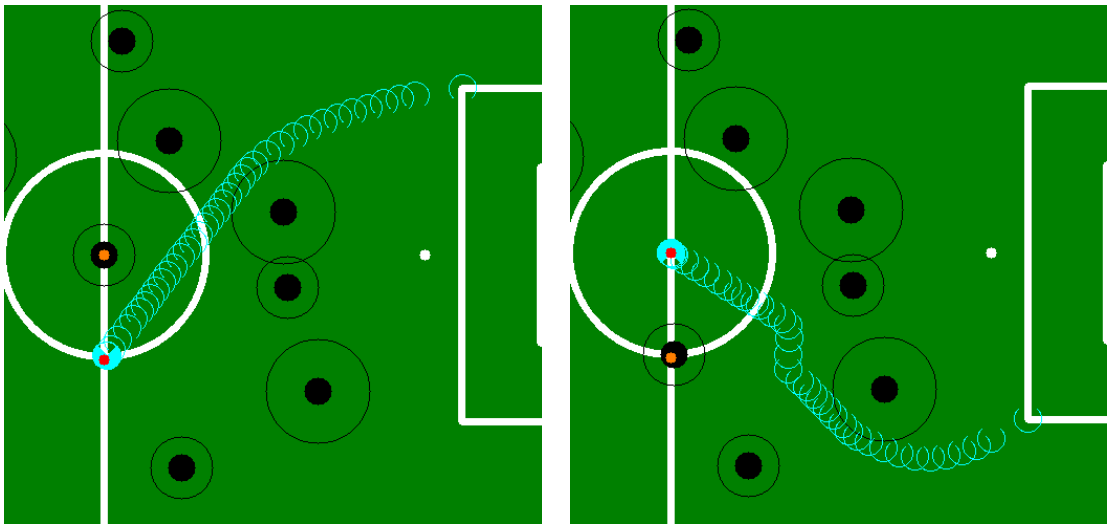
Figura 6.17 – Trajetórias percorridas pelos robôs *1* e *2*, relativas às trajetórias iniciais apresentadas na *Figura 6.16*.

Os resultados dos dois últimos testes são apresentados nas figuras *6.18* e *6.19*. Para estes dois testes foi atribuído 80 ao valor da velocidade máxima linear dos dois robôs.



(a) (b)

Figura 6.18 – Trajetórias percorridas pelos robôs *1* e *2*, relativas às trajetórias iniciais apresentadas na *Figura 6.16*.



(a) (b)

Figura 6.19 – Trajetórias percorridas pelos robôs *1* e *2*, relativas às trajetórias iniciais apresentadas na *Figura 6.16*.

Através da análise das trajetórias percorridas, é possível afirmar que o aumento da velocidade máxima linear não afetou a performance dos dois robôs (no seguimento das respetivas trajetórias). Os desvios verificados à entrada para o círculo situado no centro do campo foram originados pelos mesmos motivos do primeiro teste, exceto no terceiro teste, onde o robô *1* chegou primeiro ao círculo. Pois o robô *2* iniciou o movimento aproximadamente dois segundos depois do robô *1*. Por isso, o desvio no terceiro teste é verificado na trajetória percorrida pelo robô *2* (ver *Figura 6.19(b)*).

6.4. FESTIVAL NACIONAL DE ROBÓTICA 2017

O Festival Nacional de Robótica 2017 que ocorreu em Coimbra, contou com a presença de algumas das melhores equipas da MSL e que competem regularmente no RoboCup. Este evento era aguardado com muita expectativa por parte dos elementos da Minho Team, pois o principal objetivo era realizar pelo menos um jogo do início ao fim de acordo com as regras, executando algumas ações de jogo, tais como, um *kick-off* ou um *free-kick*. Assim, este evento era de grande importância para concretizar estes objetivos, pois nas competições de anos anteriores não foram alcançados, devido ao tempo despendido na reconstrução da plataforma anterior e de alguns problemas com a arquitetura principal do *software* dos robôs, que tinha a tarefa de trocar informação entre os vários módulos de *software*. Também existiam problemas com alguns módulos em específico, incluindo o módulo de controlo, onde o conceito adotado inicialmente não foi bem sucedido. Contudo, com as participações nos eventos de 2015 e 2016, a equipa ganhou experiência e adquiriu bastante conhecimento, mesmo que

não tenha realizado qualquer jogo nesses eventos. Após o FNR de 2016 a equipa decidiu abandonar a arquitetura principal existente, para implementar uma arquitetura com base no ROS. O conceito adotado para o módulo de controlo foi também abandonado, sendo implementados os conceitos apresentados nesta dissertação.

A equipa teve assim um ano de muito trabalho e de completa dedicação para cumprir os objetivos a que se propôs. Foi então no primeiro jogo do FNR 2017 que a equipa tentou colocar os robôs a jogar pela primeira vez, mas sem sucesso, pois deparou-se com problemas de rede. No entanto, a probabilidade deste tipo de problemas acontecer era grande, visto que era a primeira tentativa de iniciar um jogo.

Com os problemas de rede resolvidos, seguiu-se uma nova tentativa, desta vez com sucesso. Um momento de grande satisfação, sendo que a derrota por 3-0 foi o menos importante, pois os robôs foram capazes de efetuar algumas ações de jogo e de se moverem sem grandes colisões com os obstáculos em campo. Posteriormente a este jogo e antes do próximo, foram melhoradas algumas ações e táticas a serem executadas pelos robôs, por exemplo, marcar um canto, defender ou atacar.

O *software* desenvolvido durante o evento era testado no simulador, pois eram feitos continuamente jogos e por isso a disponibilidade do campo era muito limitada. No entanto, apesar das dificuldades as alterações implementadas resultaram, tendo sido alcançado no segundo jogo realizado um empate 0-0 e uma grande oportunidade de golo que só não aconteceu devido a problemas com o sistema de manipulação de bola.

Este evento foi bastante importante, não só pelos objetivos que a equipa tinha e que foram alcançados, mas também por ter sido possível a cada elemento da equipa, validar numa competição o trabalho desenvolvido.

No que diz respeito ao trabalho desenvolvido nesta dissertação, foi possível neste evento validar todo o *software*. Os resultados obtidos no planeamento da trajetória e no controlo do movimento foram bastante positivos, pois os robôs nos dois jogos realizados moveram-se para as posições pretendidas e sem grandes colisões com os obstáculos em campo. É de salientar que apenas foi necessário verificar antes do primeiro jogo, se os valores dos ganhos dos controladores PID do processo *Controlo do Movimento* estavam bem ajustados. O facto de não ter sido necessário fazer mais ajustes e alterações ao *software* já testado no LAR e no simulador, foi muito importante, visto que era quase impossível usar o campo para fazer testes e registar alguns resultados.

Relativamente aos pontos fracos na estrutura mecânica, *hardware* e *software*, neste evento foram encontrados alguns que podem ser melhorados no futuro. Por exemplo, na estrutura mecânica o sistema de manipulação de bola é pouco eficiente.

Figura 6.20 - Robôs da Minho Team no Festival Nacional de Robótica 2017 em Coimbra.

Capítulo 7

Conclusão e Trabalho Futuro

Este capítulo descreve as conclusões sobre o trabalho desenvolvido nesta dissertação e algumas ideias que poderiam ser implementadas no futuro.

7.1. Conclusão

No trabalho desta dissertação foi desenvolvido um sistema de controlo de movimento, aplicado nos robôs da Minho Team. Neste sistema, constituído pelos dois processos *Planeamento da Trajetória* e *Controlo do Movimento*, foram implementados os métodos e respetivos conceitos apresentados no *Capítulo 3*. O *software* desenvolvido neste trabalho foi testado no simulador da Minho Team, e com os robôs reais no campo de pequenas dimensões do LAR e em ambiente de competição (FNR 2017).

Nos testes realizados verificou-se que a diferença entre utilizar o simulador e os robôs reais é apenas nos ajustes aos ganhos dos controladores PID para o controlo do movimento. Por isso, e por outras razões mencionadas em capítulos anteriores, o simulador foi uma ferramenta fundamental para o desenvolvimento do *software*. O ROS foi também essencial para a construção de uma arquitetura de *software* robusta, que permitiu a comunicação entre todos os módulos de *software* de um robô.

No sistema de controlo de movimento, nomeadamente, no processo *Planeamento da Trajetória*, foram implementados métodos associados ao tipo de planeamento global com algumas estratégias/algoritmos adicionadas a esses métodos. Nos testes efetuados verificou-se que o modelo de planeamento desenvolvido é eficiente e robusto. No entanto, este depende muito da precisão do espaço de configuração para obter a melhor trajetória possível. Relativamente aos algoritmos implementados no processo *Controlo do Movimento*, foram obtidos os resultados pretendidos no que diz respeito ao seguimento das trajetórias. Por isso, pelos resultados de todos os testes efetuados, incluindo os jogos realizados no FNR 2017, conclui-se que o sistema de controlo de movimento cumpre os objetivos planeados.

Para finalizar, este trabalho de dissertação é apenas um módulo de um conjunto de módulos de *software* desenvolvidos pelos elementos da Minho Team, sendo que alguns desses módulos incluem também ferramentas de configuração e simulação. Com todos os módulos de *software* e com o tempo

despendido na reconstrução do *hardware* e da estrutura mecânica de cada robô, foi possível alcançar os objetivos delineados para este trabalho e também os objetivos da equipa.

7.2. TRABALHO FUTURO

Durante o desenvolvimento deste trabalho e após os testes efetuados, surgiram algumas ideias para possíveis melhorias, não só no sistema de controlo de movimento, mas também na estrutura mecânica dos robôs.

Uma primeira melhoria seria desenvolver um algoritmo na *Base Station* para o cálculo do diâmetro da área de cada obstáculo. Este algoritmo iria incluir a informação relativa à previsão das velocidades dos robôs adversários e dos robôs cooperativos. O algoritmo de previsão também teria de ser desenvolvido, pois este ainda não existe nos robôs da Minho Team.

Outra possível melhoria seria no algoritmo da primeira etapa da pesquisa pela trajetória mais curta, que corresponde à construção do grafo não direcionado. Neste algoritmo poderia ser estudada outra estratégia, ou melhorar a existente, de modo que não fosse necessário verificar para cada segmento de reta se existe interseção com algum obstáculo. Pois esta verificação e posterior construção do objeto grafo, demora algum tempo em comparação com os restantes algoritmos do planeamento da trajetória implementados neste trabalho.

Adicionar um método de planeamento local da trajetória, também poderia representar uma melhoria considerável, no caso em que o espaço de configuração tem pouca precisão, e também para diminuir a hipótese de uma colisão quando os robôs se deslocam para a mesma zona do campo a velocidades elevadas, ou quando as trajetórias se intersetam numa zona e estes passam nessa zona no mesmo momento.

Para finalizar, o desenvolvimento de um algoritmo para controlar o movimento do robô quando tem a bola, também seria uma boa melhoria. No entanto, seria indispensável melhorar a estrutura mecânica e *hardware* do sistema de manipulação de bola, para reter e driblar a bola durante os movimentos de translação e rotação do robô.

BIBLIOGRAFIA

[1] C. V. R. Coutinho, "Robótica Móvel - Sistema de Condução Autónoma," Instituto Superior de Engenharia de Lisboa, 2014.

[2] Business Wire, "Robôs de entrega autônomos da Panasonic – HOSPI – auxiliam operações hospitalares." [Online]. Available: http://www.businesswire.com/news/home/20150724005247/pt/. [Accessed: 04-Nov-2016].

[3] International Federation of Robotics, "Industrial Robots." [Online]. Available: http://www.ifr.org/industrial-robots/. [Accessed: 05-Nov-2016].

[4] J. A. M. F. De Souza, "Robótica." [Online]. Available: http://webx.ubi.pt/~felippe/main_pgs/mat_didp.htm. [Accessed: 23-Jul-2019].

[5] RoboCup Federation, "RoboCup." [Online]. Available: http://www.robocup.org/. [Accessed: 07-Nov-2016].

[6] "RoboCup 2004 – Portugal." [Online]. Available: http://www.robocup2004.pt/. [Accessed: 07-Nov-2016].

[7] M. Asada *et al.*, "Middle Size Robot League - Rules and Regulations," 2016.

[8] F. Ribeiro, P. Braga, J. Monteiro, I. Moutinho, P. Silva, and V. Silva, "New improvements of minho team for robocup middle size league in 2003," *Proc. CD-ROM, Rob.*, 2003.

[9] F. Ribeiro, I. Moutinho, P. Silva, C. Fraga, and N. Pereira, "Three Omni-Directional Wheels Control on a Mobile Robot," *Control 2004*, 2004.

[10] F. Ribeiro, I. Moutinho, N. Pereira, and F. Oliveira, "Cooperative Behaviour of specific tasks in multi-agent systems and robot control using dynamic approach," 2006.

[11] A. Ribeiro, G. Lopes, and J. Costa, "Minho MSL: a new generation of soccer robots," 2011.

[12] F. Ribeiro, P. Braga, I. Moutinho, P. Silva, and B. Martins, "Magnetically Impelled Kicker for Robotic Football in MSL RoboCup," *Most*, pp. 2–5, 2004.

[13] "CAMBADA - RoboCup MSL Soccer Team - Home." [Online]. Available: http://robotica.ua.pt/CAMBADA/index.php?a=106a6c241b8797f52e1e77317b96a201. [Accessed: 20-Jun-2017].

[14] G. Corrente, "Arquitectura de controlo/coordenação de uma equipa de Futebol Robótico," Universidade de Aveiro, 2008.

[15] "libtcod." [Online]. Available: http://roguecentral.org/doryen/libtcod/. [Accessed: 11-Jul-2017].

[16] "Carpe Noctem Cassel - Robotic Soccer: Home." [Online]. Available: https://www.uni-kassel.de/eecs/carpe-noctem-cassel/home.html. [Accessed: 22-Jun-2017].

[17] S. Opfer, H. Skubch, and K. Geihs, "Cooperative Path Planning for Multi-Robot Systems in Dynamic Domains," *Cdn.Intechopen.Com*, pp. 237–258, 2011.

[18] D. Bachmann *et al.*, "Carpe Noctem Cassel Team Description 2016," 2016.

[19] J. J. T. H. De Best, D. J. H. Bruijnen, R. Hoogendijk, R. J. M. Janssen, and K. J. Meessen, "Tech United Eindhoven Team Description 2010," vol. 1, 2010.

[20] C. Lopez *et al.*, "Tech United Eindhoven Team Description 2015," 2015.

[21] S. G. Tzafestas, *Introduction to mobile robot control*. 2013.

[22] R. Jitendra and K. Ajith, *Mobile Intelligent Autonomous Systems*. 2007.

[23] D. Nakhaeinia, S. H. Tang, S. B. M. Noor, and O. Motlagh, "A review of control architectures for autonomous navigation of mobile robots," *Int. J. Phys. Sci.*, vol. 6, no. 2, pp. 169–174, 2011.

[24] K. H. Sedighi, K. Ashenayi, T. W. Manikas, R. L. Wainwright, and H.-M. Heng-Ming Tai, "Autonomous local path planning for a mobile robot using a genetic algorithm," *Proc. 2004 Congr. Evol. Comput. (IEEE Cat. No.04TH8753)*, vol. 2, pp. 1338–1345, 2004.

[25] S. Russell and P. Norvig, *Artificial Intelligence A Modern Approach*. 2009.

[26] R. Siegwart and I. R. Nourbakhsh, *Introduction to Autonomous Mobile Robots*, vol. 23. 2004.

[27] J.-C. Latombe, *Robot motion planning*. 1991.

[28] M. F. R. Queirós, "Planeamento de Caminhos para Robôs Móveis Autónomos em Ambientes Conhecidos e Estruturados," Universidade do Minho, 2014.

[29] B. Y. P. Bhattacharya and M. L. Gavrilova, "Roadmap-Based Path Planning," no. June, 2008.

[30] S. Fortune, "A Sweepline Algorithm for Voronoi Diagrams," *Algorithmica*, vol. 2, no. 2, pp. 153–174, 1987.

[31] F. Aurenhammer, "Voronoi Diagrams — A Survey of a Fundamental Geometric Data Structure," *ACM Comput. Surv.*, vol. 23, no. 3, pp. 345–405, 1991.

[32] D. T. Lee and A. K. Lin, "Generalized delaunay triangulation for planar graphs," *Discrete Comput. Geom.*, vol. 1, no. 1, pp. 201–217, 1986.

[33] D. T. Lee and B. J. Schachter, "Two algorithms for constructing a Delaunay triangulation," *Int. J. Comput. Inf. Sci.*, vol. 9, no. 3, pp. 219–242, 1980.

[34] M. A. PITERI, A. G. DOS JUNIOR, MESSIAS MENEGUETTE SANTOS, and F. F. OLIVEIRA, "Triangulação De Delaunay E O Princípio De Inserção Randomizado," *Simpósio Bras. Geomática*, no. 1999, pp. 655–663, 2007.

[35] M. De Berg and M. Van Kreveld, "Voronoi Diagrams," *Comput. ...*, 1997.

[36] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numer. Math.*, vol. 1, no. 1,

pp. 269–271, 1959.

[37] T. J. Misa, "An interview with Edsger W. Dijkstra," *Commun. ACM*, vol. 53, no. 8, p. 41, 2010.

[38] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Second Edition*. 2001.

[39] "Caminho de Custo Mínimo - Algoritmo de Dijkstra." [Online]. Available: http://www.inf.ufsc.br/grafos/temas/custo-minimo/dijkstra.html. [Accessed: 22-Dec-2017].

[40] A. Conci and E. Azevedo, *Computação Gráfica - Teoria e Prática*. 2003.

[41] M. E. Mortenson, *Geometric Modeling*, 2nd ed. 1997.

[42] J. F. HUGHES *et al.*, *Computer Graphics: Principles and Practice*, 3rd ed. 2014.

[43] K. Ogata, *Modern Control Engineering 4th edtion By Katsuhiko Ogata: Modern Control Engineering*. 2002.

[44] J. Karl and T. Hagglund, *PID Controllers, 2nd Edition*. 1995.

[45] Ros.org, "ROS.org | Powering the world's robots," *website*. [Online]. Available: http://www.ros.org/. [Accessed: 11-Sep-2017].

[46] M. Quigley *et al.*, "ROS: an open-source Robot Operating System," *Icra*, vol. 3, no. Figure 1, p. 5, 2009.

[47] F. Ribeiro *et al.*, "MinhoTeam '2016 : Team Description Paper," 2016.

[48] H. Ribeiro *et al.*, "Fast Computational Processing for Mobile Robots' Self-Localization," *Proc. - 2016 Int. Conf. Auton. Robot Syst. Compet. ICARSC 2016*, no. March, pp. 168–173, 2016.

[49] J. L. Fernandes, "Desenvolvimento de hardware e software para robôs móveis," Universidade do Minho, 2006.

[50] "Omni-3MD." [Online]. Available: http://botnroll.com/omni3md/. [Accessed: 05-Mar-2018].

[51] The CGAL Project, "The Computational Geometry Algorithms Library," *Web*. [Online]. Available: https://www.cgal.org/. [Accessed: 19-Sep-2017].

[52] A. Fabri, "CGAL- The computational Geometry algorithm library," *Proc. 10th Annu. Int. Meshing Roundtable*, pp. 7–10, 2001.