Universidade de Aveiro Departamento de Engenharia Mecânica
2018

Ricardo Luís
Fernandes Silva

Navegação Local do ATLASCAR2 para Condução Autónoma e Assistência ao Condutor

ATLASCAR2 Local Navigation for Autonomous Driving and Driver Assistance

Universidade de Aveiro Departamento de Engenharia Mecânica
2018

Ricardo Luís
Fernandes Silva

Navegação Local do ATLASCAR2 para Condução Autónoma e Assistência ao Condutor

ATLASCAR2 Local Navigation for Autonomous Driving and Driver Assistance

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Mecânica, realizada sob orientação científica de Vítor Manuel Ferreira dos Santos, Professor Associado c/ Agregação do Departamento de Engenharia Mecânica da Universidade de Aveiro e de Paulo Miguel de Jesus Dias, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

O júri / The jury

Presidente / President

Professor Doutor Miguel Armando Riem de Oliveira
Professor Auxiliar do Departamento de Engenharia Mecânica da Universidade de Aveiro

Vogais / Committee

Professor Doutor Pedro Mariano Simões Neto
Professor Auxiliar do Departamento de Engenharia Mecânica da Faculdade de Ciências e Tecnologia da Universidade de Coimbra (arguente)

Professor Doutor Vítor Manuel Ferreira dos Santos
Professor Associado c/ Agregação do Departamento de Engenharia Mecânica da Universidade de Aveiro (orientador)

Agradecimentos / Acknowledgements

Em primeiro lugar deixo um agradecimento mais que especial à minha família, principalmente aos meus pais e irmã, pelo apoio e compreensão durante os cinco anos desta longa caminhada.

Agradeço ao Professor Vítor Santos e ao Professor Paulo Dias, não só pela orientação e acompanhamento nos trabalhos da dissertação, mas também pela motivação e entusiasmo transmitidos.

Quero agradecer aos meus colegas e amigos que partilharam o percurso académico comigo, nomeadamente ao Filipe, ao Miguel e ao Francisco, e aos aventureiros do ATLASCAR2, o Pedro e o Tiago.

Tenho ainda que agradecer aos velhos amigos de sempre, a Roxanne, o Tiago, o Luís e o Pedro, pela recarga das energias aos fins de semana.

E por fim, mas não menos importante, agradeço à Doutora Cláudia Fernandes que, com as sugestões certas a uma mente inconsciente, mas sábia, assegurou e cumpriu o objetivo. A conclusão desta dissertação e do Mestrado Integrado em Engenharia Mecânica.

Palavras-chave

ATLASCAR2; Condução Autónoma; Veículos Autónomos; Algoritmos de planeamento; Navegação Local; ROS; Gazebo; LIDAR; Sistemas Avançados de Apoio à Condução; Trajetórias

Resumo

A Condução Autónoma (AD) é uma das maiores áreas de investigação e com maior desenvolvimento realizado pelo setor automóvel mundial. O número de Sistemas Avançados de Apoio à Condução (ADAS) incorporados nos automóveis produzidos atualmente tem vindo a aumentar exponencialmente e é constante a pressão para o desenvolvimento de automóveis totalmente autónomos.

Inserida no projeto ATLASCAR2, que está na linha da frente na investigação da AD a nível nacional, esta dissertação tem como objetivo o desenvolvimento de um módulo de navegação local para assistência ao condutor na sua tomada de decisão imediata.

Para o cumprimento deste objetivo foi realizado um estudo dos algoritmos de planeamento local de trajetórias existentes, quais as suas vantagens e desvantagens e quais as abordagens com mais sucesso aplicadas na navegação local em projetos reais de AD. Deste estudo foi possível identificar uma solução baseada num algoritmo de abordagem por múltiplas hipóteses, a qual foi adaptada ao ATLASCAR2.

Foram realizadas adaptações ao *software* responsável pela implementação do algoritmo, desenvolvido em *Robot Operating System* (ROS), de forma a complementar as lacunas apresentadas pelo mesmo e melhorar o seu desempenho e representação gráfica. Foi ainda pesada a influência das linhas da estrada na decisão da escolha da trajetória.

Para a realização de testes, com vista a identificar as falhas e a avaliar o desempenho da solução implementada, desenvolveu-se um ambiente de simulação em *Gazebo*. Os testes foram estendidos a algumas aplicações reais, passiveis de serem realizadas tendo em conta o estado inicial da plataforma ATLASCAR2.

Keywords

ATLASCAR2; Autonomous Driving; Autonomous Vehicles; Planning Algorithms; Local Navigation; ROS; Gazebo; LIDAR; Advanced Driver Assistance Systems; Trajectories

Abstract

Autonomous Driving (AD) is one of the largest and most developed areas of research in today's world automotive sector. The number of Advanced Driver Assistance Systems (ADAS) incorporated into cars produced today has been increasing exponentially and there exists a constant pressure for the development of fully autonomous vehicles.

Inserted in the ATLASCAR2 project, which is at the forefront of AD research at the national level, this dissertation aims to develop the local navigation module for driver assistance in the immediate decision making.

In order to accomplish this objective, a study of the existing local path planning algorithms was made, their advantages and disadvantages were weighed and the most successful approaches applied to local navigation in real AD projects were taken into account. From this study, it was possible to identify a solution based on a multiple hypothesis approach that was applied to ATLASCAR2.

Adaptations were made to the software responsible for implementing the algorithm (developed in Robot Operating System (ROS)), in order to complement the gaps presented by it and to improve its own performance and graphic representation. The influence of the road lines in the trajectory choice decision was also studied.

For evaluation tests, in order to identify faults and evaluate the performance of the implemented solution, a simulation environment was developed in Gazebo. Those tests were extended to some real applications, within the limits of the initial state of the ATLASCAR2 platform.

Conteúdo

Lista de Tabelas

Lista de Figuras

Lista de Excertos de Código

Lista de Acrónimos

AD Condução Autónoma. v, 1, 2, 4–9, 91, 93

ADAP Diferença Angular ao Ponto Atrator. ii, iii, vii–ix, xi, 50, 52, 54, 73–84, 86, 89, 99

ADAS Sistemas Avançados de Apoio à Condução. 1

CL Linha Central. ii, iii, vii–ix, 50–52, 56, 57, 69, 81–84, 86, 89, 99

DAP Distância ao Ponto Atrator. ii, iii, vi–ix, xi, 50, 52–54, 73–84, 86, 89, 99

DLO Distância Mínima aos Obstáculos. ii, iii, vii–ix, xi, 50, 52, 55, 56, 66, 73–84, 86, 89, 92, 99

FNR Festival Nacional de Robótica. 2

FS Espaço Livre. 50, 55

GNSS *Global Navigation Satellite System*. 3, 4, 28

GPS *Global Positioning System*. 5, 7, 49

ICR Centro de Rotação Instantâneo. 42

JPS *Jump Point Search*. i, v, 13, 15, 16

LAR Laboratório de Automação e Robótica. i, v, 1–3, 8, 9, 21, 27, 35, 41, 93

LIDAR *Light Detection And Ranging*. vi, xi, 2–5, 7, 28–30, 33, 34, 37, 38, 41, 59, 62, 63, 65, 67, 69, 84, 85, 87, 100

OBD *On-Board Diagnostic*. 93

PRM *Prababilistic Roadmap Method*. i, v, 13, 18–20

ROS *Robot Operating System*. i, ii, vi, 9, 31–35, 37, 38, 41, 49, 59, 60, 66, 67, 91–93, 99–101

RRT *Rapidly-Exploring Random Tree*. i, v, 13, 20, 21, 41

TF *Transform Frame*. 33, 59, 67

UPS *Uninterruptible Power Supply*. 3

URDF *Unified Robot Description Format.* vi, vii, 34, 60, 61, 64

VFH *Vector Field Histogram.* i, v, 13, 16–18

WN *Winding Number.* vi, 47, 48

Capítulo 1

Introdução

O paradigma dos veículos autónomos tem vindo a gerar uma larga discussão acerca da implementação, robustez e fiabilidade dos mesmos (Gruyer et al., 2017). Muita da argumentação empregue nessa discussão baseia-se na elevada mortalidade rodoviária a que se assiste nos dias de hoje. Segundo dados da Organização Mundial de Saúde, em 2013, ocorreram cerca de 1,25 milhões de mortes nas estradas de todo o mundo, e espera-se que este número venha a aumentar na próxima década (WHO, 2015).

Desde as décadas de 60 e 70 do século passado que existe a preocupação, por parte da indústria automóvel, em aumentar a segurança dos veículos com o surgimento dos primeiros Sistemas Avançados de Apoio à Condução (ADAS). Quando se fala em ADAS, não se está apenas a dirigir aos veículos autónomos, capazes de realizar rotas sem qualquer intervenção humana, mas também a sistemas mais simples como, por exemplo, de auxílio à travagem como o ABS (*Anti-lock Braking System*). Mais tarde começaram a emergir soluções de controlo de estabilidade, de velocidade e de distância de travagem (Gruyer et al., 2017). Estes sistemas são considerados passivos pois só atuam quando solicitados pelo condutor. No entanto, com a evolução tecnológica, a investigação nesta área expandiu-se a instituições académicas e empresas que não estão diretamente relacionadas com a produção automóvel. Isto deu lugar a projetos de condução totalmente livre da intervenção humana compostos por sistemas ativos, que não necessitam de ações por parte do condutor para atuarem (Hu et al., 2018).

Desafios de Condução Autónoma (AD), como o *DARPA Grand Challenge* e o *DARPA Urban Challenge*, organizados pela Agência de Projetos de Investigação Avançada de Defesa dos Estados Unidos, têm estimulado o surgimento de novos projetos de AD e, atualmente, os carros autónomos desenvolvidos pela *Google* já realizaram mais de 5 milhões de quilómetros de testes nas estradas americanas. Construtores automóveis como a *Tesla* também já começaram a equipar os seus veículos com sistemas de piloto automático capazes de realizar percursos em ambiente urbano e extraurbano sem a intervenção do condutor (Hu et al., 2018).

O Laboratório de Automação e Robótica (LAR), do Departamento de Engenharia Mecânica da Universidade de Aveiro, também possui um projeto de AD, o ATLAS, que iniciou o seu percurso à escala real com o ATLASCAR1, um veículo instrumentado para a recolha de dados e execução de algumas manobras, como estacionamento e seguimento de pedestres. Desde 2016, fruto de parcerias com outros departamentos, o projeto iniciou a migração para uma nova plataforma, o ATLASCAR2. Ainda numa fase inicial, este projeto apresenta inúmeros desafios, nomeadamente a navegação local.

Segundo Katrakazas et al., (2015), a navegação local é responsável pelo guiamento do veículo, isto é, pelo planeamento da direção e velocidade a tomar, num espaço próximo da posição atual com base em informação exclusivamente obtida pelos sensores a bordo. O guiamento deve ser planeado de forma a garantir o deslocamento, desde o estado atual até ao objetivo, sem colisões.

A definição anterior é o ponto de partida desta dissertação. Assim, durante o decorrer dos trabalhos, foi desenvolvido um sistema de navegação local e assistência à condução implementado no ATLASCAR2. Este sistema recorre a um algoritmo de planeamento de trajetórias por múltiplas hipóteses para realizar a navegação e tem como objetivo indicar ao condutor qual a direção na qual este deve orientar o automóvel, com base, exclusivamente, em dados de sensores com tecnologia *Light Detection And Ranging* (LIDAR). O alvo da direção dada pelo algoritmo é a orientação do veículo para um ponto de destino sem a presença de colisões durante a navegação.

Neste capítulo introdutório, o projeto ATLAS é apresentado com mais detalhe na secção 1.1, sendo de seguida apresentados exemplos de projetos de AD, (secção 1.2). O enquadramento desta dissertação é realizado na secção 1.3 e os objetivos a atingir e a organização do documento são alvo de escrutínio nas secções 1.4 e 1.5, respetivamente.

1.1 O projeto ATLAS

O projeto ATLAS foi criado em 2003, pelo Grupo de Automação e Robótica, no LAR, com o objetivo de estudar e desenvolver sensores avançados e sistemas ativos projetados para a instalação em automóveis, possibilitando assim a proliferação de conhecimentos acerca dos mesmos no ambiente académico (LARlabs, 2011). Os primeiros projetos na área da AD centraram-se em modelos à escala (subsecção 1.1.1), para participações no Festival Nacional de Robótica (FNR). O sucesso e a experiência adquirida com estes modelos permitiu a evolução do projeto para veículos à escala real e assim surgiram o ATLASCAR1 (subsecção 1.1.2), em 2010 e o ATLASCAR2 (subsecção 1.1.3), em 2016.

1.1.1 Modelos ATLAS

O primeiro robô desenvolvido (figura 1.1a), assentava numa estrutura de alumínio e madeira. O movimento de tração era assegurado por um diferencial mecânico acoplado às rodas traseiras e o movimento de direção era dado por uma única roda dianteira. Neste protótipo estava instalada apenas uma câmara apontada a um espelho em `v` para permitir a visualização completa da pista onde o robô circulava.

Numa tentativa de criar um modelo semelhante a um automóvel comum, a equipa do projeto ATLAS criou o ATLAS 2000 (figura 1.1b), à escala de 1:4 e com ele ganhou pela primeira vez a prova de AD do FNR, em 2006. Depois de várias melhorias efetuadas no ATLAS 2000, em 2008 foi criada uma nova plataforma, o ATLAS MV (figura 1.1c). Este robô foi projetado a uma escala menor (1:5), com o intuito de ser mais leve e rápido. A bordo foram instalados um novo sistema de direção, travagem hidráulica e uma unidade de perceção ativa. Este robô permitiu a conquista de novas vitórias na prova de AD, perfazendo um total de 6.

(a) Primeiro protótipo ATLAS. (b) ATLAS 2000. (c) ATLAS MV.

Figura 1.1: Modelos ATLAS à escala desenvolvidos pelo Grupo de Automação e Robótica no LAR, (adaptado de J. F. Pereira, (2012)).

1.1.2 ATLASCAR1

Impulsionado pelos resultados positivos, alcançados com os modelos à escala e pelos anos de experiência na navegação em ambientes controlados, em 2010, o Grupo de Automação e Robótica decidiu investir num projeto à escala real, o ATLASCAR1 (figura 1.2). A plataforma utilizada para este projeto foi uma carrinha *Ford Escort* de 1998 alimentada por um motor de combustão interna a gasolina, onde foram instalados múltiplos sensores, unidades de processamento e atuadores. Aos sensores a bordo competia a recolha de dados do veículo e do ambiente em redor, com vários LIDARs para a deteção de obstáculos e reconstrução do ambiente, câmaras para a deteção de peões e uma unidade de *Global Navigation Satellite System* (GNSS) para a localização do veículo e planeamento de rotas. Depois de passarem pelas unidades de processamento, estes dados eram enviados para os atuadores que permitiam a movimentação e execução de manobras de uma forma totalmente autónoma por parte o veículo. Para alimentar todo o equipamento era utilizada uma *Uninterruptible Power Supply* (UPS), carregada a partir de um alternador auxiliar.

No decorrer deste projeto foram desenvolvidos inúmeros trabalhos no LAR, muitos deles originando dissertações de mestrado. A título de exemplo, Cabral de Azevedo, (2014), desenvolveu um módulo de deteção peões por fusão sensorial de dados LIDAR e de visão e Vieira da Silva, (2016), criou um módulo de calibração multissensorial que foi exportado para projetos posteriores.

1.1.3 ATLASCAR2

Depois de inúmeras modificações, e já com idade avançada, o ATLASCAR1 atingiu vários limites. De forma a dar continuidade ao projeto, em 2016, foi adquirido um novo veículo, o ATLASCAR2 (figura 1.3). Desta vez a escolha da plataforma recaiu num *Mitsubishi i-MiEV*, um automóvel elétrico com autonomia para cerca de 100 km. Para além de custos de utilização mais reduzidos, o facto do ATLASCAR2 ser elétrico permite utilizar a energia armazenada nas suas baterias, facilitando a alimentação dos sensores instalados.

Apesar do pouco tempo de existência, a bordo do ATLASCAR2 já se encontram

Figura 1.2: ATLASCAR1, baseado numa *Ford Escort Station Wagon* de 1998, (adaptado de J. F. Pereira, (2012)).

instalados 3 LIDARs, uma câmara, sensores de inclinometria e a uma unidade de GNSS. A maioria destes sensores foram transferidos do ATLASCAR1 para este projeto durante os trabalhos da dissertação de Madureira Correia, (2017), onde foi também desenvolvido um módulo de deteção do espaço livre em redor do carro.

Figura 1.3: ATLASCAR2, baseado num automóvel elétrico, um *Mitsubishi i-MiEV* de 2015.

1.2 Veículos autónomos

Os projetos de AD contemporâneos têm surgido de diversos meios. Apesar de empresas de construção automóvel como a *General Motors*, a *Audi*, a *Mercedes-Benz*, a

Daimler, a *Toyota*, a *Nissan* e a *Tesla* estarem bem posicionadas em termos tecnológicos, os maiores avanços neste plano têm sido apresentados por empresas tecnológicas com a *Google* e a *Uber*. Existem ainda alguns projetos de investigação ligados a universidades e centros de investigação, como o *VisLab*, que têm demonstrado excelentes resultados em situações de condução totalmente autónoma (Bimbraw, 2015).

Desde 2009 que a *Google* tem vindo a liderar na investigação relacionada com a AD. O objetivo inicial do projeto era conseguir desenvolver um automóvel capaz navegar em autoestradas com a mínima intervenção humana. No entanto, o projeto foi tão bem sucedido que, em 2012, avançaram para testes em ambiente urbano. Em 2016, a *Google* isolou a investigação da AD num projeto designado de *Waymo* e no ano seguinte lançou o seu primeiro automóvel totalmente autónomo, uma carrinha *Chrysler Pacifica* (figura 1.4), (Waymo, 2018a).

Figura 1.4: Veículo autónomo *Waymo*, baseado numa *Chrysler Pacifica*, (adaptado de Waymo, (2018a)).

Equipado com 3 LIDARs, 9 câmaras e outros sensores de localização, o *Waymo* consegue detetar peões, ciclistas e veículos até 3 campos de futebol num ângulo de 360°. Com mais de 5 milhões de quilómetros percorridos em testes, o *software* desenvolvido consegue prever o comportamento dos outros utilizadores da estrada como, por exemplo, a intenção de um ciclista mudar de via quando estica um braço (Waymo, 2018b).

A *Uber* também realizou fortes investimentos no desenvolvimento do seu próprio veículo autónomo, um *Ford Fusion* com 20 câmaras montadas a toda a volta. No topo do veículo está montado um LIDAR com abertura de 360° e uma câmara auxiliar para colorir a nuvem de pontos do LIDAR com o intuito de detetar mudanças nos sinais de trânsito luminosos, permitindo assim identificar quando o automóvel pode ou não avançar. Estão ainda instalados mais 6 LIDARs, 7 radares e uma unidade de posicionamento por *Global Positioning System* (GPS).

Este veículo foi usado numa série de testes em Pittsburgh, não só para testar a sua capacidade de AD, mas também para recolha de mapas. Recentemente a *Uber* adquiriu novos carros de testes baseados em veículos *Volvo*. Nestes novos automóveis o número de câmaras foi reduzido para metade, mas o número de radares aumentou para também garantir a existência de dados deste tipo de tecnologia a toda a volta no veículo (Uber, 2018b).

Figura 1.5: Veículo autónomo da *Uber*, baseado num *Ford Fusion*, (adaptado de Uber, (2018a)).

Comercialmente, os veículos de produção em série com maior autonomia de condução são os modelos *S* e *X* (figura 1.6), da *Tesla*, equipados com *Autopilot*. Estes automóveis trazem instaladas 8 câmaras, 12 sensores ultrassónicos e um radar capazes de receber dados a toda a volta do veículo e com um alcance de mais de 250 metros. A gestão desta informação fica a cargo de uma rede neuronal desenvolvida pela própria *Tesla*. Toda esta capacidade de perceção e processamento de dados permite que, através de um comando do utilizador, o automóvel calcule a rota e a execute quase sem intervenção humana. Para além da capacidade de circular em autoestradas a elevadas velocidades, estes carros já conseguem circular autonomamente em estradas urbanas, mesmo sem marcações de faixa, cruzamentos complexos com semáforos ou sinais de *STOP* e rotundas. Resta ainda referir que manobras de estacionamento também podem ser executadas autonomamente (Tesla, 2018b).

Figura 1.6: *Tesla Model S* equipado com *Autopilot* e capaz de realizar tarefas de AD, (adaptado de Tesla, (2018a)).

Por fim, um bom exemplo de um automóvel autónomo desenvolvido por um laboratório de investigação é o *BRAiVE* (figura 1.7). Este protótipo foi criado pelo *VisLab* com o objetivo de ser um projeto de baixo custo, utilizando maioritariamente sensores de visão, como câmaras, em detrimento de LIDARs. A unidade de perceção do *BRAiVE* é constituída por 10 câmaras, 4 sensores laser, 1 radar e uma unidade inercial e de GPS. Todos estes sensores permitem a existência de diferentes níveis de automação a bordo do veículo, capazes do reconhecimento de sinalização de trânsito, de assistência para circulação dentro da faixa de rodagem, seguimento de outros veículos e da tarefa de AD propriamente dita, com o recurso a algoritmos de planeamento por múltiplas hipóteses. Devido a problemas de circulação em estradas não pavimentadas toda a unidade de perceção e processamento deste protótipo foi replicada para outros veículos que participaram no *VisLab Intercontinental Autonomous Challenge*, uma prova de AD que ligou Parma, em Itália, a Shanghai, na China num total de mais de 214 horas de viagem (Alberto Broggi et al., 2013).

Figura 1.7: Protótipo *BRAiVE* desenvolvido pelo *VisLab*, baseado num *Hyundai Sonata*, (adaptado de Alberto Broggi et al., (2013)).

1.3 Enquadramento e motivação

Nos dias que correm, o mercado automóvel é constantemente inundado por novos modelos capazes de assistir o condutor na tarefa de condução, sendo que, alguns veículos conseguem mesmo substituir a ação do condutor em determinadas circunstâncias. A investigação e desenvolvimento que os construtores automóveis têm vindo a investir na esfera da AD e assistência ao condutor faz com que os veículos de gamas mais baixas venham já equipados com sistemas de aviso de ângulo morto ou de mudança de faixa. À medida que se vai subindo para modelos topo de gama, ou marcas *premium*, já se encontram carros com tecnologia de seguimento do veículo da frente e começam a surgir no mercado os primeiros carros capazes de realizar rotas sem a ação do condutor. Construtores mais direcionados para a produção de veículos pesados de transporte de mercadorias também já entraram nesta onda tecnológica e integram nos seus modelos sistemas de auxílio ao condutor para facilitar e tornar mais segura a tarefa de condução.

A este ritmo, é bem possível que nas próximas décadas os nossos automóveis não passem de robôs, operados por complexos algoritmos e com comunicação entre si, que desempenham a tarefa de motoristas. O condutor passa a ser designado de utilizador e, para além de tarefas de manutenção e abastecimento, apenas necessita de chamar o automóvel e ditar a instrução do local para onde se pretende deslocar.

A existência de um veículo no LAR capaz de recolher dados sensoriais que caracterizam a sua envolvente possibilita que também a Universidade de Aveiro se possa aventurar no complexo mundo da AD. Integrado nesta temática, e vocacionado para a assistência ao condutor, devido à impossibilidade de atuação no ATLASCAR2, surgiu o tema desta dissertação. Este projeto propõe o desenvolvimento de um sistema de apoio ao condutor que, através da implementação de um algoritmo de navegação local, sugere ao condutor qual a direção que este deve tomar.

Não para já, mas num futuro próximo, espera-se que o projeto ATLASCAR2 se venha a tornar no primeiro veículo autónomo desenvolvido por uma instituição de ensino em Portugal, tal como os apresentados na secção anterior.

1.4 Objetivos

Os objetivos iniciais que esta proposta de dissertação pretende cumprir são:

Identificação e seleção das estruturas de dados mais adequadas aos algoritmos de navegação local

O primeiro objetivo pode ser decomposto em duas tarefas principais, a compreensão das diferentes estruturas de dados para representação do espaço livre, e a escolha da mais apropriada para a implementação do algoritmo escolhido. Para tal, em primeira instância, deverão ser avaliadas as diferentes representações do espaço livre, como grelhas de ocupação, polígonos de espaço livre, espaços de obstáculos, etc. Este objetivo deverá ser desenvolvido em simultâneo com a seleção dos algoritmos de navegação local pois, diferentes algoritmos requerem diferentes representações do espaço de trabalho. No entanto, o algoritmo de navegação não pode ser o único fator a ter em conta na escolha da estrutura de dados. Importará também, nesta fase, ter um conhecimento *a priori* dos custos computacionais das diferentes soluções e se os sensores instalados no ATLASCAR2 conseguem fornecer dados que possam ser convertidos na estrutura de dados requerida pelo algoritmo.

Identificação e seleção dos algoritmos de navegação local mais adequados aos ambientes esperados para o ATLASCAR2

Para o cumprimento deste objetivo deverá ser realizado um levantamento teórico dos algoritmos de navegação local existentes na literatura, bem como as suas restrições. Depois do estudo das diferentes opções, será ainda importante investigar a aplicabilidade dos algoritmos a situações de navegação em AD, tendo em conta as condicionantes que a mesma apresenta, a título de exemplo, o ambiente extremamente dinâmico e tempos de decisão reduzidos. Após avaliar as limitações que os algoritmos apresentam e as introduzidas pela AD restará selecionar um algoritmo para implementar como solução de navegação local para o ATLASCAR2. Esta etapa será fundamental para o corolário

da dissertação pois todos os resultados obtidos, bem como o sucesso da navegação, irão depender do algoritmo aqui selecionado.

Implementação do algoritmo selecionado para fazer a navegação local

Uma vez selecionado o algoritmo de navegação local, este deverá ser implementado e integrado na arquitetura de *software* já instalada no ATLASCAR2. Assim, todo o *software* terá que ser desenvolvido em *Robot Operating System* (ROS), permitindo a comunicação direta com os sensores ou subscrevendo informação já publicada por trabalhos anteriores, e o algoritmo deve ser capaz de prever qual a direção e velocidade que o veículo deve tomar a cada instante. Há ainda que considerar as características cinemáticas e dimensionais do ATLASCAR2 para o cálculo da trajetória e da velocidade. A implementação do algoritmo implicará também a integração de uma estrutura de dados adequada ao mesmo, podendo ser utilizadas as já existentes ou existir a necessidade de desenvolver uma nova estrutura.

Como complemento à implementação, e de forma a avaliar o algoritmo, este deve ainda ser testado para que se possa validar o seu comportamento. Os testes podem ocorrer em ambiente real ou simulado.

Desenvolvimento de uma aplicação de monitorização do algoritmo de navegação local

Como forma de acompanhar a ação do algoritmo de navegação local, no último objetivo desta dissertação, é proposto o desenvolvimento de uma aplicação que permitirá ao utilizador visualizar o espaço livre para a navegação e a trajetória calculada pelo algoritmo. Esta aplicação servirá ainda para sugerir ao utilizador a direção que o veículo deve tomar a cada instante.

1.5 Estrutura da dissertação

A presente dissertação encontra-se estruturada em sete capítulos, a saber: "Introdução", "Revisão da literatura", "Infraestrutura experimental", "Solução de navegação local", "Ambiente de simulação", "Resultados"e "Conclusões e trabalhos futuros".

No presente capítulo, ("Introdução"), é feita uma breve apresentação teórica do paradigma da AD e do projeto ATLAS, do qual esta dissertação faz parte. São ainda descritos o enquadramento e a motivação, para a realização deste projeto, e enumerados os objetivos aos quais esta dissertação se propõe.

O capítulo 2 foca alguns dos aspetos mais teóricos desta dissertação com uma breve introdução ao planeamento, uma revisão da literatura relacionada com algoritmos de navegação local e uma noção introdutória à cinemática de *Ackermann*.

Depois, no capítulo 3, é apresentada toda a infraestrutura experimental utilizada durante o decorrer dos trabalhos. É descrito o *hardware* integrado no ATLASCAR2, diretamente relacionado com este trabalho, e é realizado um apanhado do *software* de terceiros e desenvolvido no LAR que serviu como ferramenta a esta dissertação.

Segue-se a descrição da solução de navegação local implementada, no capítulo 4. Começando com a justificação da escolha da solução, neste capítulo é detalhada a forma como o algoritmo de navegação local é implementado e de que forma são obtidos os

outputs do mesmo.

Devido à inexistência de atuação no ATLASCAR2 foi necessária a criação de um simulador para testar e afinar o algoritmo de navegação. A criação no simulador, desde o modelo do ATLASCAR2 até à aplicação proposta para a realização das simulações, é descrita no capítulo 5.

Por fim, são apresentados e discutidos os resultados obtidos nas simulações no capítulo 6 e as conclusões e propostas de trabalhos futuros no capítulo 7.

Capítulo 2

Revisão da literatura

Neste capítulo é apresentada a revisão da literatura realizada para suportar as decisões tomadas ao longo da dissertação, principalmente a escolha do algoritmo de navegação local implementado.

Na secção 2.1 são explicadas e diferenciadas as componentes que o planeamento executado por um veículo autónomo possuí, desde a informação global até à informação local da direção a seguir. Nas secções 2.2 e 2.3 são apresentados os algoritmos de planeamento cuja implementação no ATLASCAR2 foi ponderada. A secção 2.4 introduz a cinemática de *Ackermann* que rege a direção do ATLASCAR2 e cujo modelo foi utilizado no simulador.

2.1 Planeamento

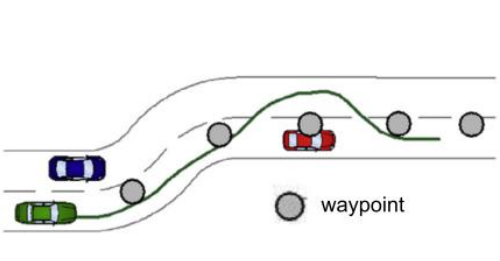
No contexto da condução autónoma, onde se enquadra esta dissertação, o planeamento é realizado em dois níveis bem distintos. O alto nível, ou planeamento/navegação global, que é responsável pela determinação da rota que o veículo deve seguir, desde o ponto de partida até ao destino, tendo em consideração exclusivamente mapas digitais com a representação das estradas. E o baixo nível, ou planeamento/navegação local, que é responsável pelo traçado de um caminho possível de ser seguido pelo veículo, pelo planeamento de manobras e pelo planeamento de trajetórias (Katrakazas et al., 2015).

O planeamento do caminho entre dois pontos consecutivos da rota é encarado como um problema complexo, onde são levados em conta os obstáculos físicos, os obstáculos virtuais, impostos pelas regras de trânsito, e a dinâmica do veículo de forma garantir a ausência de colisões. Para planear o caminho, pode ainda ser fundida informação fornecida por outros elementos ativos no espaço de planeamento (Ganesha Perumal et al., 2016), como, por exemplo, informação de outros veículos. Este tipo de planeamento necessita de uma visão mais abrangente do espaço e de um ponto de vista superior (figura 2.1a), comparada a uma vista de satélite atualizada sempre que o planeamento é executado.

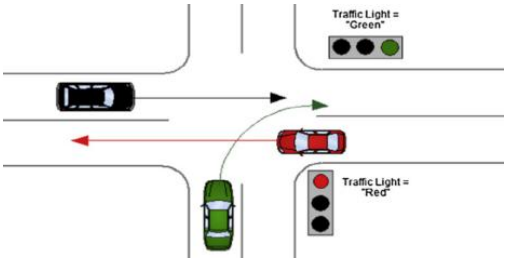
As manobras são decisões de alto nível cujo planeamento deve controlar a direção e a velocidade instantânea do veículo como, por exemplo, em estacionamento, cruzamentos com sinalização luminosa (figura 2.1b), ou ultrapassagens, tendo sempre em consideração o caminho anteriormente definido.

Ainda assim podem surgir obstáculos inesperados, como peões, ou situações não

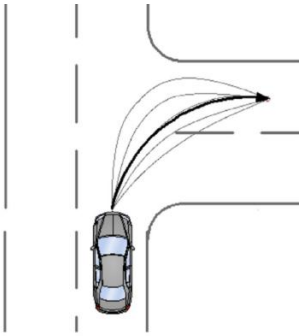
contempladas pelos planeamentos de caminhos e manobras que requerem uma atuação instantânea na direção. Como tal, deve ser executado um planeamento da direção a tomar com base em informação exclusivamente obtida pelos sensores a bordo do veículo que permita o deslocamento entre estados livre de colisões e confortável para os ocupantes do veículo, mesmo que isto implique um desvio do caminho já traçado (Katrakazas et al., 2015). O planeamento de trajetórias, ponto central desta dissertação, deve ser atualizado à maior taxa possível. Sempre que são recebidos dados dos sensores a bordo, deve ser planeada ou escolhida uma trajetória que guie o veículo com o objetivo de seguir o caminho proposto (figura 2.1c).



(a) Planeamento de caminhos.



(b) Planeameto de manobras.



(c) Planeamento de trajetórias.

Figura 2.1: Representação dos três diferentes tipos de planeamento/navegação local, (adaptado de Katrakazas et al., (2015)).

A conjugação do planeamento global com as três dimensões na navegação local (planeamento de caminhos, de manobras e de trajetórias), é essencial para o projeto de um veículo autónomo seguro e confortável.

2.2 Algoritmos de planeamento de trajetória

Por algoritmos de planeamento de trajetória entendem-se algoritmos que calculam uma trajetória que une dois pontos, a posição atual e o próximo objetivo a atingir (Werling et al., 2010). Estes algoritmos objetivam encontrar a trajetória ótima, de acordo com o espaço de planeamento disponível, no entanto, para navegação em ambientes com elevada imprevisibilidade, requerem cálculos complexos constantes o que se pode revelar ineficiente para aplicações em tempo real. Esta aplicação fica ainda mais comprometida

no caso de planeamento a elevadas velocidades, como por exemplo o planeamento em autoestradas (Hu et al., 2018).

Nesta secção são apresentados alguns dos algoritmos mais utilizados para o planeamento local de trajetórias, a saber: campos de potencial, A*, *Jump Point Search* (JPS), *Vector Field Histogram* (VFH), *Prababilistic Roadmap Method* (PRM) e *Rapidly-Exploring Random Tree* (RRT).

2.2.1 Campos de potencial

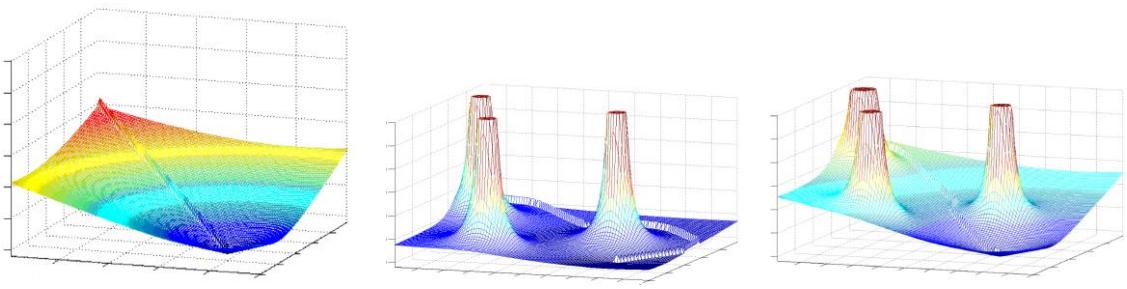
O conceito de uma partícula com carga elétrica a deslocar-se num campo magnético potencial pode ser exportado para o planeamento de trajetórias (Goodrich et al., 2002). O algoritmo dos campos de potencial parte dessa representação e combina dois campos, um negativo para o ponto atrator e outro positivo para os obstáculos presentes no espaço operacional, para planear trajetórias entre dois pontos conhecidos. Numa representação simplista, o potencial atrator — negativo — gerado pelo objetivo é uma formulação quadrática da distância dos pontos do espaço a esse ponto (equação 2.1). O campo gerado pelos obstáculos — positivo — é descrito por uma função hiperbólica que tende para infinito quando a distância aos obstáculos é mínima (equação 2.2).

$$U_{obj}(q) = \mathrm{d}(q, obj)^2 \tag{2.1}$$

$$U_{obst}(q) = \mathrm{d}(q, obst)^{-1} \tag{2.2}$$

A representação combinada do potencial atrator com a soma dos campos repulsores, equação (2.3), é utilizada para planear a trajetória a executar. O planeamento é então realizado na direção do maior gradiente decrescente, e a figura 2.2 apresenta um exemplo de planeamento utilizando o algoritmo dos campos de potencial.

$$U(q) = U_{obj}(q) + \sum U_{obst}(q) \tag{2.3}$$



(a) Campo atrator, $U_{obj}(q)$. (b) Campo repulsor, $U_{obst}(q)$. (c) Campo total, $U(q)$.

Figura 2.2: Exemplo de planeamento de trajetória utilizando o algoritmo dos campos de potencial, (adaptado de Andrade da Costa, (2012)).

Este algoritmo tem sofrido várias evoluções deste o seu aparecimento e em Moreau et al., (2017), pode-se encontrar uma dessas evoluções numa aplicação deste método a veículos reais. Numa tentativa de tornar a curva de potencial menos restrita e permitir uma navegação mais próxima dos obstáculos, sempre sem colisão, Moreau et al., (2017),

utiliza campos de potencial fracionais que limitam o potencial repulsor a um raio pré-definido do obstáculo (equação 2.4). r_{min} e r_{max} são os raios de atuação do campo repulsor e o n pode ser ajustado consoante o perigo que o obstáculo apresenta, quanto maior, menor é o decaimento inicial do campo.

$$U_{obst}(q) = \frac{r^n - r_{max}^n}{r_{min}^n - r_{max}^n}, r_{min} < r < r_{max} \tag{2.4}$$

Outra limitação dos campos de potencial a que os campos fracionais tentam responder é a elevada aceleração quando o robô ainda se encontra muito distante do objetivo. Através da colocação de *way points* entre o estado atual e o objetivo, o potencial atrator é diminuído, atenuando assim as acelerações calculadas.

2.2.2 A*

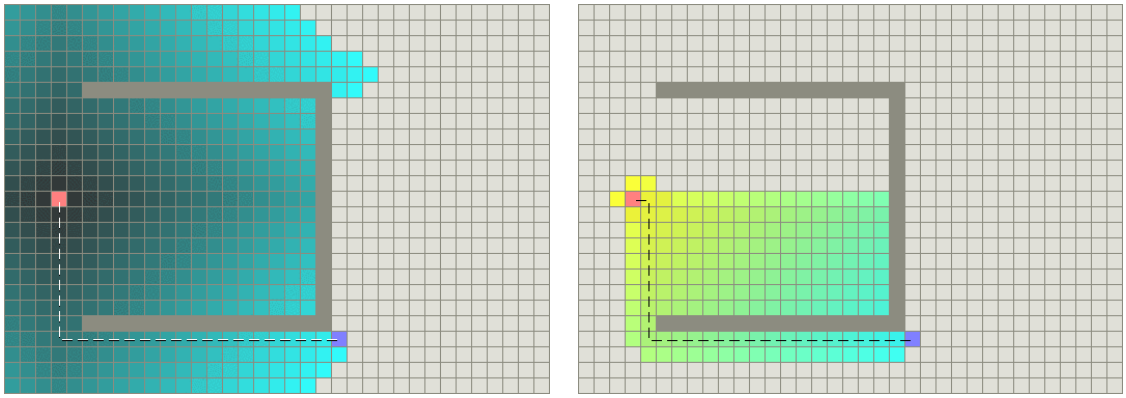
O A* (Hart et al., 1968), é um algoritmo de procura do caminho com menor custo entre dois pontos baseado em grelhas de ocupação que discretizam o espaço de planeamento em células ocupadas e células livres. Este algoritmo é uma evolução do algoritmo de *Dijkstra* (Cormen et al., 2001), que, a partir do ponto inicial, analisa todos os pontos até atingir o ponto final. Sempre que um ponto é analisado, os seus vizinhos são adicionados à lista de pontos a analisar, garantindo assim o varrimento de todo o espaço até se atingir o objetivo e que a trajetória encontrada é a mais curta entre os pontos de entrada.

Numa tentativa de limitar a expansão da análise do algoritmo a todos os pontos da grelha, reduzindo assim custos computacionais, este deve tomar uma decisão informada acerca dos nodos para os quais se deve expandir (Hart et al., 1968). Se a expansão não ocorrer ao longo da trajetória mais curta estão-se a desperdiçar cálculos, mas se esta expansão ignorar as outras direções, o objetivo pode não ser atingido. Desta forma o A* utiliza a função de avaliação $\hat{f}(n)$, uma estimativa da função de custo $f(n)$, para determinar quais os nodos que devem ser avaliados no passo seguinte. A função de custo (equação 2.5), é dividida em duas partes, $g(n)$ e $h(n)$, sendo $g(n)$ o custo da trajetória ideal desde o ponto inicial até ao nodo n e $h(n)$ o custo da trajetória ótima desde o nodo n até ao objetivo.

$$f(n) = g(n) + h(n) \tag{2.5}$$

O custo das trajetórias é muitas vezes definido como a distância entre os pontos, podendo esta ser euclidiana, diagonal ou de *Manhattan*. Depois da análise atingir o ponto final, a trajetória é definida pelo conjunto de pontos com menor $f(n)$. A redução do número de pontos, resultado da utilização desta função de avaliação, face ao de *Dijkstra* é bem evidente na figura 2.3.

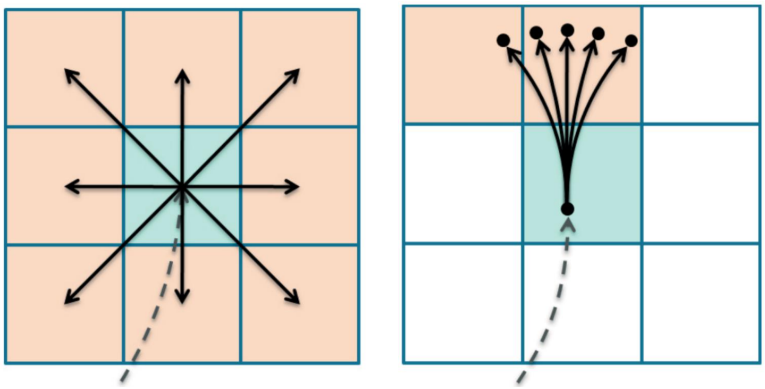
A aplicação direta do A* à navegação de veículos com comportamento não holonómico não é possível devido a variações não controladas de direção; surgiu por isso o A* híbrido. Ao contrário do A* convencional, a sua versão híbrida é capaz de realizar um planeamento contínuo de acordo com as limitações de um veículo real (Petereit et al., 2012). Isto é conseguido com equações de movimento pré-determinadas, respeitando a cinemática característica do veículo, que são utilizadas para determinar quais as células do espaço de planeamento que são atingíveis num determinado estado. Para além do custo, são também armazenados os parâmetros da equação que permite chegar a uma



(a) Algoritmo de *Dijkstra*. (b) Algoritmo A*.

Figura 2.3: Comparação do espaço da grelha de planeamento analisado pelo o algoritmo de *Dijkstra* e pelo A*, (Patel, 2018).

dada célula, utilizados posteriormente no traçado da melhor trajetória. A figura 2.4 dá as possibilidades de vizinhança de uma célula para a utilização do A* convencional e do A* híbrido.



(a) A* convencional. (b) A* híbrido.

Figura 2.4: Possibilidades de vizinhança de uma célula, dadas pelas duas formulações do A*, (adaptado de Petereit et al., (2012)).

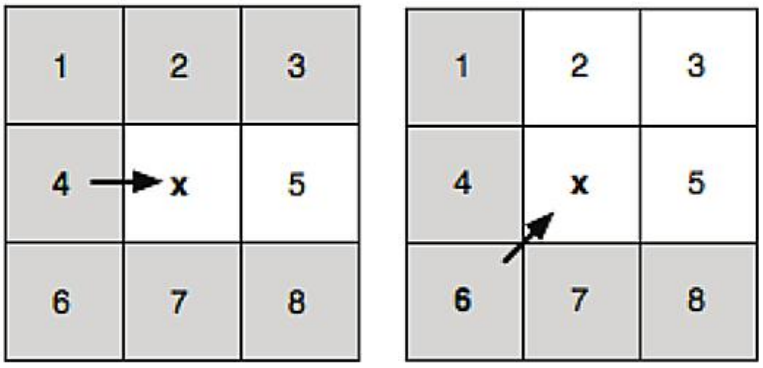
2.2.3 *Jump Point Search* (JPS)

Como forma de otimizar o A* em grelhas com custo uniforme surge o JPS (Harabor et al., 2011). A ideia por detrás deste algoritmo é eliminar a expansão a certos nodos da grelha com base na informação dos seus vizinhos permitindo assim maiores saltos na grelha em vez dos saltos unitários permitidos pelo A*. A eliminação de vizinhos depende da direção de chegada ao estado atual, sendo que, no ponto inicial não podem ser eliminados quaisquer vizinhos. Para deslocações na horizontal ou vertical são eliminados todos os vizinhos, n, que verifiquem a equação 2.6 e na diagonal todos os que verifiquem a equação 2.7.

$$len(\langle p(x), ..., n\rangle \backslash x) \leq len(\langle p(x), x, n\rangle) \tag{2.6}$$

$$len(\langle p(x), ..., n\rangle \backslash x) < len(\langle p(x), x, n\rangle) \tag{2.7}$$

Partindo do exemplo da figura 2.5a, num deslocamento horizontal vindo da ponto 4, $p(4)$, são excluídos todos os vinhos expecto o 5, $n = 5$, pois para todos os outros o custo do caminho $\langle 4, n\rangle$ não passando por x é menor ou igual ao custo do caminho $\langle 4, x, n\rangle$. No caso de deslocamentos diagonais (figura 2.5b), a única diferença é que são eliminados os vizinhos apenas com um caminho de custo menor.



(a) *Jump points* horizontais. (b) *Jump points* diagonais.

Figura 2.5: Vizinhos excluídos da busca na grelha pelo algoritmo JPS (marcados a cinzento), (adaptado de Harabor et al., (2011)).

Os restantes procedimentos de funcionamento do JPS são idênticos ao A* descrito na sub-secção 2.2.2. Tal como o A*, este algoritmo também é capaz de encontrar a melhor trajetória e segundo Harabor et al., (2011), possuí uma velocidade de expansão cerca de uma ordem de grandeza superior ao A*.

2.2.4 *Vector Field Histogram* (VFH)

A formulação original do VFH (Borenstein e Koren, 1991), utiliza duas etapas de redução de dados para determinar a orientação e velocidade no robô a partir do seu mapa envolvente, discretizado numa grelha de confiança. A primeira etapa engloba a descrição do espaço com recurso a sensores de distância, que se encontram a bordo do robô, e a conversão dos dados sensoriais num histograma cartesiano bidimensional. É então atribuído um valor $m_{i,j}$ a cada célula de acordo com a equação 2.8, onde a e b são constantes, $c_{i,j}$ traduz o grau de confiança da presença de obstáculos, obtido através da deteção do obstáculo durante as sucessivas leituras dos sensores, e $d_{i,j}$ a distância desses obstáculos ao ponto central do veículo, dada pelos sensores.

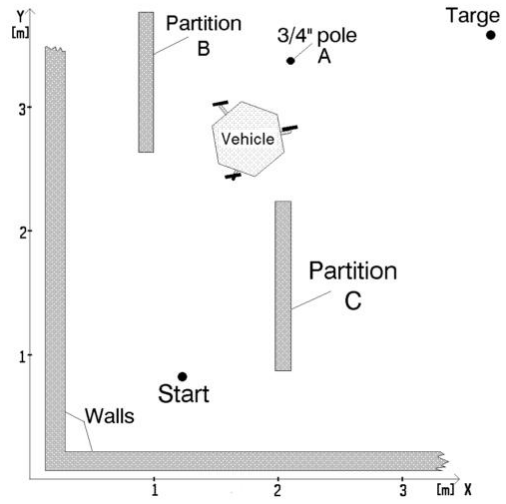
$$m_{i,j} = (c_{i,j})^2 \cdot (a - b \cdot d_{i,j}) \tag{2.8}$$

De seguida é feita então a conversão do histograma cartesiano bidimensional num histograma polar unidimensional. Este histograma é construído em torno do ponto central do veículo com a divisão do espaço em setores compreendidos entre intervalos angulares

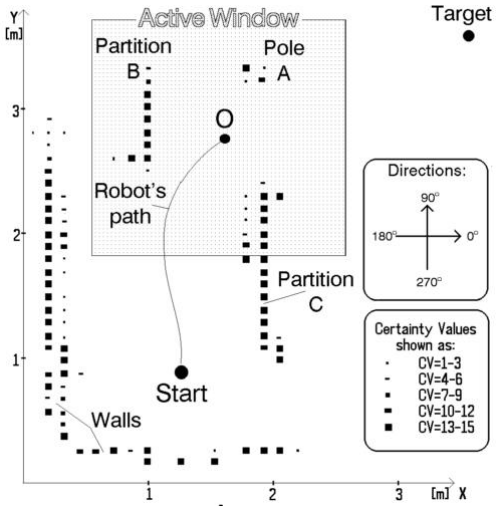
pré-definidos. Para cada setor é determinada a densidade de obstáculos h_k através do somatório das magnitudes das células cartesianas compreendidas nesse sector, (equação 2.9).

$$h_k = \sum_{j,i}^{N} m_{i,j} \tag{2.9}$$

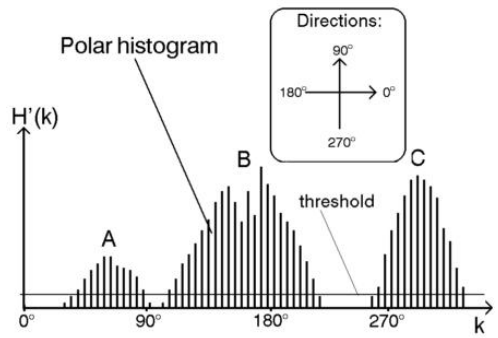
As representações do espaço nos diferentes histogramas podem-se encontrar na figura 2.6 onde é possível visualizar as diferentes etapas do algoritmo para o mesmo espaço de trabalho.



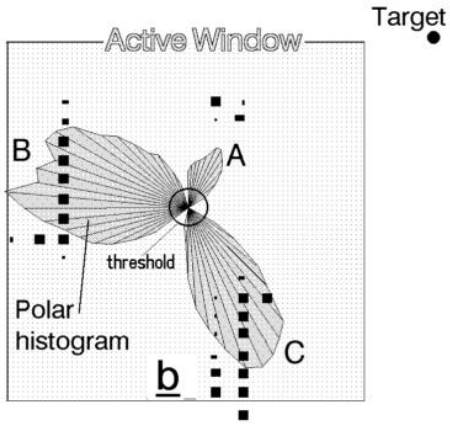
(a) Exemplo de espaço de obstáculos.



(b) Grelha de confiança cartesiana ou histrograma bidimensional.



(c) Histrograma polar unidimensional.



(d) Representação do histrograma polar unidimensional no espaço.

Figura 2.6: Representação do espaço livre ao longo das etapas de redução de dados segundo o algoritmo VFH, (adaptado de Borenstein e Koren, (1991)).

O VFH estabelece um limiar de densidade abaixo do qual os setores são candidatos à direção a seguir. Conjuntos de setores consecutivos abaixo do limiar são agrupados

em vales e desses vales é selecionado aquele cuja direção se encontra mais próxima do objetivo a atingir. Depois é calculado o seu ângulo médio que é tomado como a direção que robô deve seguir. A figura 2.7 ilustra esta escolha.

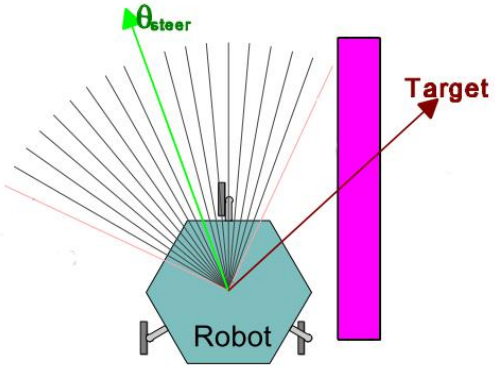


Figura 2.7: Direção escolhida pelo VFH, dentro do setor candidato, (adaptado de Borenstein e Koren, (1991)).

A velocidade é sempre assumida como a máxima pré-definida, a não ser que sejam determinadas variações bruscas de direção. Neste caso a velocidade é ajustada de acordo com a equação 2.10 onde h_c é a densidade de obstáculos na direção atual e h_m é uma constante empírica.

$$v = v_{max} \cdot \left(1 - \frac{\min(h_c, h_m)}{h_m}\right) \tag{2.10}$$

Segundo Borenstein e Koren, (1991), este algoritmo apresenta algumas limitações. Uma das quais é o facto de o robô poder ser conduzido para passagens demasiado estreitas em virtude de uma escolha errada do limiar de densidade de obstáculos. Outra assenta no planeamento local reativo do algoritmo incapaz de delinear uma trajetória ótima.

Posteriormente surgiram novas formulações. O VFH+ (Borenstein e Ulrich, 1998), que passou a utilizar quatro etapas de redução de dados e introduziu uma função de custo para a escolha da direção em vez de assumir imediatamente o valor médio do vale selecionado. O VFH* (Ulrich et al., 2000), assenta no VFH+ e verifica se uma dada direção candidata leva o robô a contornar os obstáculos. Esta verificação recorre ao A* produzindo uma trajetória global mais suave, isto é, com menos variações bruscas de direção.

2.2.5 *Prababilistic Roadmap Method* (PRM)

A ideia base do PRM (Kavraki et al., 1996), é a marcação de pontos aleatórios no espaço operacional, testar os que se encontram no espaço livre e depois, através de uma heurística, encontrar a conexão de pontos mais vantajosa. Este método está dividido em duas fases. A primeira é a marcação dos pontos e a segunda consiste na aplicação da heurística para encontrar o caminho mais curto.

A primeira fase, também designada de fase de aprendizagem, tem como objetivo formar uma rede de pontos uniformemente espaçados no espaço livre, C_{free}. De acordo com uma heurística são detetados os pontos mais próximos de obstáculos e é realizada

a sua expansão, que conduz a uma densificação da rede de pontos nesses locais. Os nós c, pertencentes ao espaço livre, são adicionados, um a um, ao espaço de vértices, V, e para cada vértice é selecionado um conjunto de vizinhos do espaço livre, N_c, composto por pontos c', distados a uma distância inferior à pré-definida, do ponto c. Posteriormente é testada a conectividade dos pontos c', pertencentes ao espaço de vizinhos, com o ponto principal, c, para desta forma se criar os espaço de ligações, E. Este processo de construção do PRM é explicado em Geraerts et al., (2006), de acordo com o algoritmo 1.

Algoritmo 1: Construção do PRM: (V, E), (adaptado de Geraerts et al., (2006)).

```
1: V ← ∅ ; E ← ∅;
2: for ∀c ∈ C_free do
     c ← um ponto de C_free ;
     V ← V ∪ c ;
     N_c ← um conjunto de nós candidatos c', vizinhos de c e pertencentes a V ;
     for ∀c' ∈ N_c do
       if c' e c não estiverem ligados then
         if o planeador local encontrar um caminho entre c' e c then
           E ← E ∪ c'c ;
         end
       end
     end
   end
3: return V; E
```

Nesta conceção, o algoritmo apenas adiciona uma ligação ao espaço de caminhos, se c' e c não estiverem já ligados por um segmento de reta, reduzindo assim tempo de implementação. Este efeito pode ser observado na figura 2.8.

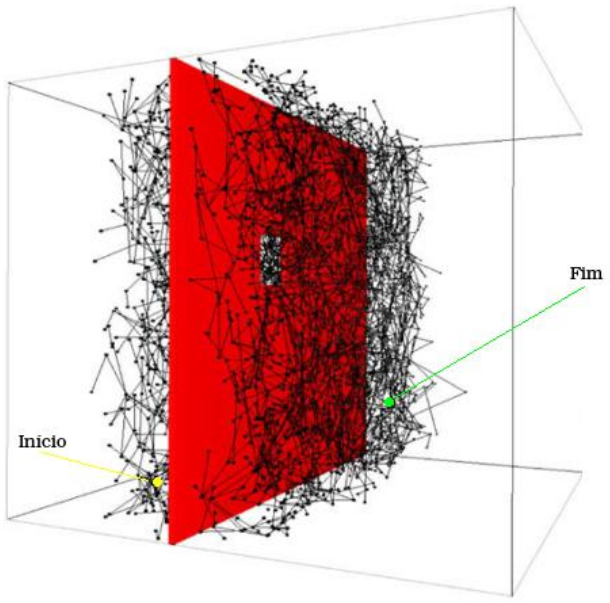


Figura 2.8: Pontos e ligações geradas pelo PRM, (adaptado de Geraerts et al., (2006)).

A segunda fase do algoritmo aplica outra heurística, tipicamente *Dijkstra*, para a escolha da trajetória mais vantajosa. Para robôs holonómicos esta trajetória pode ser executada diretamente, porém, quando na presença de restrições de movimentos, é necessária a sua suavização (Kavraki et al., 1996). A suavização da trajetória acarreta um problema, esta pode sofrer variações de tal ordem que conduza a colisões até então não detetadas.

2.2.6 *Rapidly-Exploring Random Tree* (RRT)

O RRT foi concebido, inicialmente (LaValle, 1998), para aproveitar os benefícios dos algoritmos de planeamento aleatórios, respeitando as restrições cinemáticas impostas pelo veículo utilizado. Outros algoritmos, como o PRM, já realizam planeamento aleatório, no entanto, a ligação entre estados não contempla cinemáticas complexas como a de veículos com direção não holonómica. O RRT utiliza o mesmo tipo de planeamento entre estados com a particularidade de os ligar com equações de transição que garantem sempre o cumprimento das restrições cinemáticas do robô. Para o planeamento são considerados dois espaços complementares, o espaço livre e o de obstáculos, sendo que, em nenhum caso, a árvore de trajetórias, τ, pode intercetar o espaço de obstáculos. Sendo um processo iterativo, o planeamento da trajetória desde o ponto inicial até ao final é realizado de acordo com o algoritmo 2:

Algoritmo 2: Geração do RRT: (x_{init}, K, Δt), (adaptado de LaValle, (1998)).

1: τ.init(x_{init})
2: **for** $k = 1$; $k \leftarrow k + 1$; $k < K$ **do**
  $x_{rand} \leftarrow$ estado_aleatório() ;
  $x_{near} \leftarrow$ estado_mais_próximo(x_{rand},τ) ;
  $u \leftarrow$ seleção_entrada(x_{rand},x_{near}) ;
  $x_{new} \leftarrow$ novo_estado(x_{near}, K, Δt) ;
  τ.adiciona_estado(x_{new}) ;
  τ.adiciona_ligação(x_{near}, x_{nw}, u) ;
  end
3: **return** τ

A partir do ponto inicial da árvore, x_{init}, são marcados pontos pontos aleatórios, x_{rand}, no espaço livre e é escolhido aquele cuja distância linear ao estado atual é menor, x_{near}. A ligação entre os vértices é realizada de maneira a minimizar a distância percorrida de acordo com a cinemática definida, u, criando assim um novo vértice, x_{new}. Esse vértice e o respetivo caminho são adicionados à árvore de trajetórias com K estados até atingir o objetivo.

Em Hess et al., (2013), pode ser encontrada uma aplicação deste algoritmo a robôs com cinemática idêntica aos automóveis. Nesta aplicação a distância entre estados é definida com a distância euclidiana e para controlar o aumento exagerado do número de estados apenas são considerados 10 novos estados a cada iteração. O resultado desta aplicação encontra-se na figura 2.9.

Segundo Katrakazas et al., (2015), apesar dos RRTs serem probabilisticamente completos, garantirem a execução cinemática das trajetórias e explorarem rapidamente o espaço livre, apresentam algumas limitações. A principal desvantagem é a geração de trajetórias irregulares e a forte dependência das mesmas da métrica utilizada para determinar o ponto mais próximo em cada estado. Também é necessária uma verificação

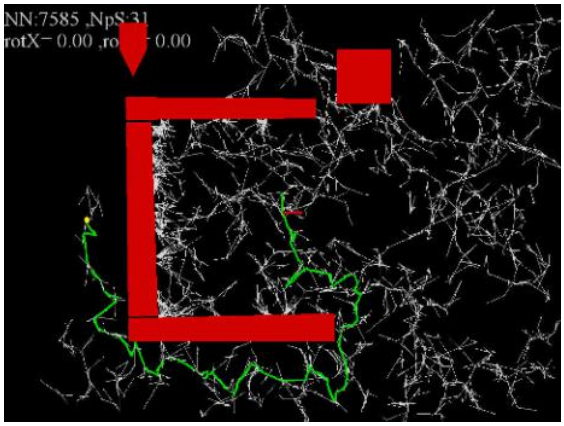


Figura 2.9: Exemplo de trajetória planeada pelo RRT, (Hess et al., 2013).

posterior de possíveis colisões quando a trajetória atravessa passagens estreitas, de acordo com as dimensões do veículo.

2.3 Algoritmos de planeamento por hipóteses

Outras abordagens ao planeamento de trajetórias incluem otimização discreta (Werling et al., 2010). Por otimização discreta entende-se a escolha da melhor trajetória de um conjunto discreto de trajetórias hipotéticas calculadas com base em equações que descrevem a dinâmica do veículo. A escolha da melhor trajetória é realizada com base numa função de custo determinada pelo algoritmo implementado.

Ao contrário dos algoritmos de planeamento de trajetória, os algoritmos de planeamento por hipóteses não determinam a trajetória ótima, no entanto, têm alcançado grande sucesso na área da condução autónoma pois permitem uma implementação em tempo real (Hu et al., 2018).

Nesta secção são apresentados dois algoritmos de planeamento discretos. As grelhas evidenciais e a abordagem por múltiplas hipóteses, um algoritmo com origem no LAR.

2.3.1 Grelhas evidenciais

Muitos dos algoritmos apresentados na secção 2.2 utilizam grelhas de ocupação onde o estado de cada célula varia entre ocupada e livre. Em Mouhagir et al., (2017), é proposto um conceito diferente de avaliação do espaço de planeamento designado de grelhas evidenciais. Esta representação tem como base a fusão da leitura atual dos sensores com a leitura anterior e condensa essa informação em quatro campos para cada célula, $[m(\emptyset); m(F); m(O); m(\Omega)]$, cuja soma é unitária: $m(\emptyset)$ representa o conflito de determinada célula como por exemplo células ocupadas pelo próprio veículo, $m(F)$ representa a confiança da célula estar livre, $m(O)$ a confiança da ocupação e $m(\Omega)$ a incerteza que caracteriza as células atrás de obstáculos. A figura 2.10 apresenta um exemplo de uma grelha evidencial.

Em cima da grelha evidencial são traçadas trajetórias pré-definidas em forma de tentáculo ou clotóide, equação 2.11, onde a sua curvatura, ρ, depende da abcissa curvilínea,

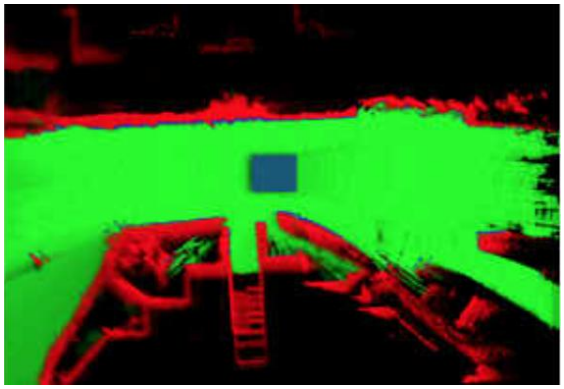


Figura 2.10: Discretização do espaço de navegação numa grelha evidencial, (adaptado de Mouhagir et al., (2017)).

s, e do parâmetro da clotóide, k. Cada tentáculo é avaliado individualmente segundo a equação 2.12 e é escolhido o que apresentar maior classificação.

$$\rho = \frac{2}{k^2} \cdot s \tag{2.11}$$

$$R_{tent} = \sum_{k=0}^{n_s} \gamma_0 \cdot R_{s_{k|ocup}} + \sum_{k=0}^{n_s} \gamma_1 \cdot R_{s_{k|traj}} + R_{ultrap} \tag{2.12}$$

Para cada estado de um tentáculo, n_s, a que corresponde um circulo com o diâmetro igual à largura do veículo, é calculado o critério de ocupação, $R_{s_{k|ocup}}$, com base no estado das células da grelha intercetadas e o critério da trajetória, $R_{s_{k|traj}}$, com base na distância ao objetivo. Para o planeamento de ultrapassagens é acrescentado o parâmetro R_{ultrap} às trajetórias da esquerda, fazendo com que estas apresentem uma classificação superior às da direita e centrais.

A aplicação das trajetórias em forma de clotóide à grelha evidencial pode ser visualizada na figura 2.11.

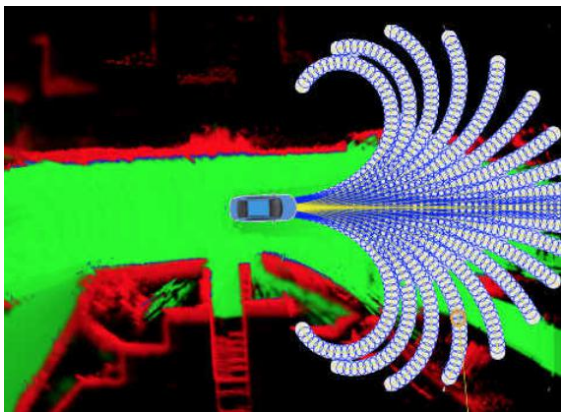


Figura 2.11: Planeamento com clotóides, baseado em grelhas evidenciais, (adaptado de Mouhagir et al., (2017)).

2.3.2 Abordagem por múltiplas hipóteses

Com o objetivo de planear a direção imediata que um robô deve seguir, Oliveira et al., (2012), propõe uma abordagem de planeamento por múltiplas hipóteses. Testado em modelos de automóveis à escala, este algoritmo seleciona a trajetória mais vantajosa de entre um conjunto de trajetórias pré-definidas. Essas trajetórias são arcos de circunferência gerados tendo em conta comportamento não holonómico do veículo e as suas dimensões. Com base na figura 2.12, as coordenadas cartesianas de cada trajetória, $(\prod_x, \prod_y)$ são determinadas pela equação 2.13, onde A é o comprimento do arco, D é a distância do eixo de direção ao centro de rotação e α é o ângulo de direção. As trajetórias são discretizadas num número pré-definido de nodos unidos por segmentos de reta para facilitar a aplicação do algoritmo.

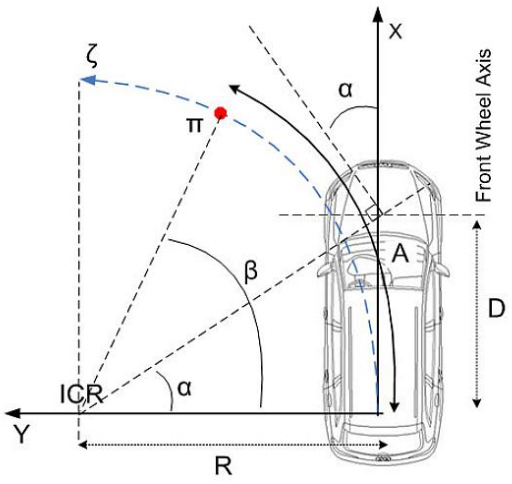


Figura 2.12: Traçado de trajetórias circulares com base num modelo não holonómico de um veículo, (adaptado de Oliveira et al., (2012)).

$$\prod = \begin{bmatrix} \prod_x \\ \prod_y \end{bmatrix} = \begin{bmatrix} \frac{D}{\tan(\alpha)} \cdot \sin\left(\frac{A \cdot \tan(\alpha)}{D}\right) \\ \frac{D}{\tan(\alpha)} \cdot \left(1 - \cos\left(\frac{A \cdot \tan(\alpha)}{D}\right)\right) \end{bmatrix} \tag{2.13}$$

Em oposição aos outros algoritmos aqui apresentados, este algoritmo não utiliza grelhas de qualquer tipo para efetuar o planeamento, mas sim conjuntos de pontos, designados de marcadores, que delimitam os obstáculos físicos, U^k, detetados e as linhas da estrada reconhecidas, R^k, L^k e C^k. Os pontos de cada obstáculo são unidos por segmentos de reta que são utilizados para detetar interseções com os segmentos de reta em que as trajetórias estão divididas. Como representado na figura 2.13, também são utilizados pontos atratores, A^k, como objetivos do planeamento.

Para cada trajetória é calculada a sua pontuação, ζ, com base na equação 2.14. Ω_1 é a média das distâncias lineares dos nodos da trajetória ao ponto atrator, Ω_2 é a média das distâncias angulares dos nodos da trajetória ao ponto atrator e Ω_3 é a distância média dos nodos da trajetória aos obstáculos. Estas distâncias são normalizadas e são-lhes atribuídos pesos, w_1, w_2 e w_3, para alterar o comportamento do algoritmo consoante a aplicação. Ω_4 é um critério adicional que é unitário quando a trajetória não

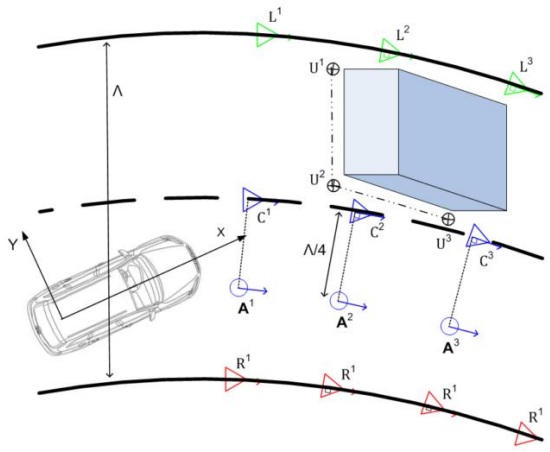


Figura 2.13: Representação dos marcadores dos obstáculos físicos (U^k), das linhas da estrada (R^k, L^k e C^k), e dos pontos atratores (A^k), (adaptado de Oliveira et al., (2012)).

interceta nenhum obstáculo e nulo quando existe interceção, anulando assim a pontuação da trajetória de forma a esta não ser escolhida.

$$\zeta = \Omega_4 \cdot \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}^\top \cdot \begin{bmatrix} \Omega_1 \\ \Omega_2 \\ \Omega_3 \end{bmatrix} \tag{2.14}$$

O algoritmo seleciona a trajetória que apresentar uma pontuação mais elevada.

2.4 Cinemática de *Ackermann*

A principal característica de um veículo é a sua capacidade para ser conduzido de uma maneira precisa. A tarefa de condução não é mais do que ajustar as variações da velocidade e da direção do veículo, mediante as condições percecionadas. Para variar a magnitude da velocidade o condutor tem que atuar nos pedais de travão e acelerador, enquanto que para alterar a direção deve atuar no volante (Guiggiani, 2014). Para além do volante, o sistema de direção é composto pela coluna de direção, engrenagens responsáveis por converter o movimento de rotação em translação e a articulação responsável por direcionar as rodas (figura 2.14).

Para garantir a estabilidade do veículo, quando este descreve uma curva, todas as rodas devem rolar sobre o piso sem escorregamento (De-Juan et al., 2012). Segundo a lei de *Ackermann*, a ausência de escorregamento é conseguida quando o eixo de rotação de todas as rodas possui um ponto comum. Para veículos com direção apenas no eixo dianteiro, este ponto tem que estar localizado no prolongamento do eixo traseiro. Matematicamente, esta condição é dada pela equação 2.15, onde δ_{od} e δ_{in} são os ângulos da roda de dentro e da roda de fora, respetivamente, z_1 é a distância entre os eixos de rotação da direção e L é a distância entre eixos.

$$\delta_{od} = \arctan\left(\frac{1}{\cot\left(\delta_{in}\right) + \frac{z_1}{L}}\right) \tag{2.15}$$

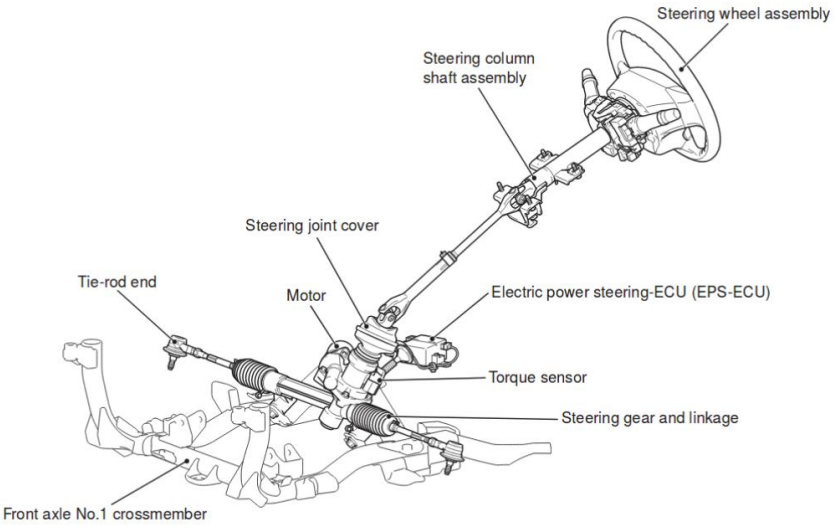


Figura 2.14: Componentes do sistema de direção do *Mitsubishi i-MiEV*, (adaptado de MMC, (2018a)).

A figura 2.15 esquematiza o modelo de direção do tipo *Ackermann* e, no caso do veículo virar à esquerda, a roda de dentro é a roda esquerda e a de fora é a direita, e para uma viragem à direita, a roda de dentro passa a ser a da direita e de fora a da esquerda.

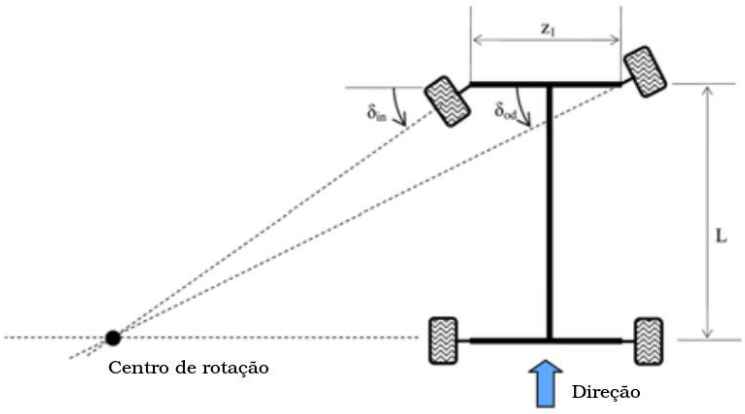


Figura 2.15: Modelo cinemático de uma direção do tipo *Ackermann*, (adaptado de De-Juan et al., (2012)).

Capítulo 3

Infraestrutura experimental

Este capítulo apresenta as infraestruturas de *hardware* e *software* utilizadas na realização dos trabalhos esta dissertação.

Primeiramente, nas secções 3.1, 3.2 e 3.3, são apresentados e descritos os equipamentos que compõem a plataforma ATLASCAR2 e dos quais este trabalho se serviu. Numa segunda parte, é descrito o *software* (secção 3.4), e as extensões de *software* criadas no LAR (secção 3.5), utilizadas como ferramentas de trabalho ao longo deste projeto.

3.1 *Mitsubishi i-MiEV*

A base de todo o projeto ATLASCAR2 assenta num carro elétrico, o *Mitsubishi i-MiEV* (figura 3.1). É nesta plataforma que se encontram instalados todos os sensores mencionados na secção 3.2, e a unidade de processamento (secção 3.3), bem como o seu sistema de alimentação.

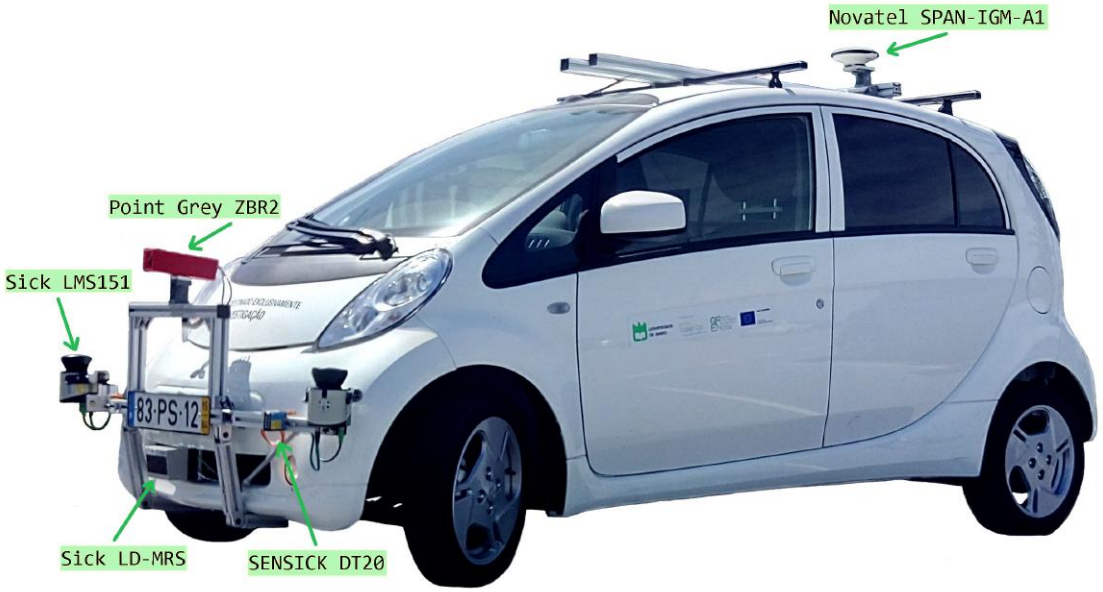


Figura 3.1: Plataforma ATLASCAR2, baseada num *Mitsubishi i-MiEV*.

O *Mitsubishi i-MiEV* possui um motor elétrico com uma potência de 49 kW e uma bateria de iões de lítio capaz de armazenar 16 kWh de energia elétrica, o que se traduz numa autonomia estima de cerca de 100 km (MMC, 2018b). A bateria pode ser totalmente carregada em cerca de 10 horas quando ligada a uma tomada convencional de 220 V ou em 30 minutos numa estação de carregamento rápido. O *Mitsubishi i-MiEV* também possui um sistema de regeneração de energia capaz de fornecer energia suplementar à bateria em caso de travagem.

Outras características importantes como as dimensões do chassi, da carroceria e direção podem ser encontradas na tabela 3.1.

Tabela 3.1: Principais características técnicas do *Mitsubishi i-MiEV*, (MMC, 2018b).

Motor	Potência	49 kW
	Binário	180 N.m
Bateria	Energia total	16 kWh
	Autonomia	100 km
Carroceria	Largura	1,475 m
	Comprimento	3,475 m
	Distância entre-eixos	2,550 m
	Distância entre rodas (frente/trás)	1,310 m / 1,270 m
	Peso	1450 kg
Direção	Ângulo máximo (dentro/fora)	45° / 38°
	Raio mínimo de viragem	4,5 m
Pneus	Frente	145/65R15 72S
	Trás	175/55R15 77V

3.2 Sensores

A bordo da plataforma ATLASCAR2 estão instalados três LIDARs, quatro sensores de medição de distância, *SENSICK DT20 Hi*, uma unidade de GNSS, *Novatel SPAN-IGM-A1* e uma câmara para obtenção de imagem, *Point Grey ZBR2-PGEHD-20S4C* (figura 3.1). Os quatro sensores de medição de distância integram a unidade de inclinometria que tem como objetivo medir as variações de *roll* e *pitch* ao longo do trajeto realizado. Estes dados são depois combinados com os dados obtidos pelos LIDARs para deteção dos limites da estrada e deformações do pavimento. A unidade de GNSS é responsável pelo fornecimento de dados inerciais e posição global do ATLASCAR2 que é utilizada para o planeamento global de rotas.

Nesta secção são apenas descritos os sensores LIDAR *Sick LMS151* e *Sick LD-MRS400001* pois são a fonte da nuvem de pontos utilizada na navegação local.

3.2.1 *Sick LMS151*

Atualmente, o ATLASCAR2 possui instalados dois sensores LIDAR 2D *Sick LMS151* (figura 3.2), projetados para aplicações de navegação, deteção e medição em ambientes internos e externos. São sensores com alcance até 50 m e devido à tecnologia *multi-echo* é possível a obtenção de dados fiáveis em ambientes com poeiras, nevoeiro e mesmo

através de vidro (Sick, 2018b). Com um elevado ângulo de abertura do feixe, 270°, estes sensores laser são utilizados para a deteção de obstáculos dentro e fora de estrada e nas laterais do veículo. Encontram-se instalados nos extremos do para-choques dianteiro e a comunicação é realizada através do protocolo *Ethernet*.

Figura 3.2: Sensor LIDAR *Sick LMS151*, (Sick, 2018b).

A tabela 3.2 lista algumas das principais características deste sensor.

Tabela 3.2: Especificações técnicas do sensor *Sick LMS151*, (Sick, 2018b).

Características	Fonte luminosa	Infravermelha (905 nm)
	Classe laser	1 (EN 60825-1:2014)
	Ângulo de abertura	270°
	Frequência	25 Hz / 50 Hz
	Resolução angular	0,25° / 0,50°
	Alcance	0,5 m a 50 m
	Alcance máx. com refletividade de 10%	18 m
Desempenho	Tempo de resposta	$\geq$ 20 ms
	Objetos detetáveis	Quase todos
	Erro sistemático	$\pm$ 30 mm
	Erro estatístico	12 mm
Alimentação	Tensão de operação	10,8 V DC até 30 V DC
	Consumo elétrico	8 W
Comunicação	*Ethernet*	TCP/IP 10/100 MBit/s

3.2.2 *Sick LD-MRS400001*

O terceiro LIDAR que se encontra no ATLASCAR2 é um *Sick LD-MRS400001* (figura 3.3). Este sensor realiza varrimento a três dimensões através da combinação dos dados de quatro feixes planares distados, angularmente, de 0,8°. Além da tecnologia *multi-echo*, este LIDAR pode ser configurado para apresentar diferentes resoluções ao longo dos feixes planares o que permite a densificação da nuvem de pontos nas zonas mais importantes

e a redução nas zonas de menor interesse (Sick, 2018a). No ATLASCAR2 este sensor é utilizado em duas funções distintas, a reconstrução da estrada para deteção dos seus limites físicos e irregularidades no piso, e a deteção de objetos para a navegação local devido ao seu alcance de 300 m. A sua localização é no centro do para-choques dianteiro e também comunica através do protocolo *Ethernet*.

Figura 3.3: Sensor LIDAR *Sick LD-MRS400001*, (Sick, 2018a).

A tabela 3.3 lista algumas das principais características deste sensor.

Tabela 3.3: Especificações técnicas do sensor *Sick LMS151*, (Sick, 2018a).

Características	Fonte luminosa	Infravermelha
	Classe laser	1 (EN 60825-1:2014)
	Ângulo de abertura (2 / 4 feixes)	110° / 85°
	Frequência	12,5 Hz a 50 Hz
	Resolução angular	0,125° / 0,25° / 0,50°
	Alcance	0,5 m a 300 m
	Alcance máx. com refletividade de 10%	50 m
Desempenho	Objetos detetáveis	Quase todos
	Erro sistemático	± 300 mm
	Erro estatístico	100 mm
Alimentação	Tensão de operação	9 V DC até 27 V DC
	Consumo elétrico	8 W
Comunicação	*Ethernet*	TCP/IP 100 MBit/s

3.3 Unidade de processamento

O processamento de toda a informação proveniente dos sensores instalados é realizado pelo servidor *Nexus P-2308H4/HR4*, instalado no ATLASCAR2 (figura 3.4). Os dois processadores *Intel XEON* permitem a execução simultânea, em tempo real, das várias aplicações de tratamento de informação dos vários contribuintes do projeto ATLASCAR2. Para a representação gráfica dos dados é utilizada uma placa gráfica *Nvidia Quadro*.

De todo o *hardware* presente na plataforma ATLASCAR2, este equipamento foi o único que sofreu intervenções no âmbito desta dissertação. Devido ao baixo desempenho gráfico demonstrado, concluiu-se que a placa gráfica não estava 100% funcional. Foi então necessária a instalação de *drivers* compatíveis com o sistema operativo em funcionamento.

Figura 3.4: Servidor *Nexus P-2308H4/HR4*, (Nexus, 2016).

A tabela 3.4 apresenta as principais características do servidor.

Tabela 3.4: Especificações técnicas do servidor *Nexus P-2308H4/HR4*, (Nexus, 2016).

Processador	2x *Intel XEON E5-2600V4*
Memória	32 Gb DDR4 2400 MHz
Armazenamento	6 Tb HDD 7200 rpm
Gráficos	*Nvidia Quadro K4200*
Rede	2x *Intel Gigabit onboard*
Sistema operativo	*Ubuntu Server 16.04.4 LTS (Xenial Xerus)*

3.4 *Software*

Seguindo a filosofia da arquitetura do *software* já instalado da plataforma ATLAS-CAR2, todo o *software* desenvolvido nesta dissertação é baseado em ROS. Esta *framework* é utilizada no desenvolvimento de aplicações para robôs e, para além de permitir a programação e integração de código em várias linguagens, possui ferramentas de visualização de dados a 3 dimensões, como o *Rviz*, e integração com *softwares* de simulação, como o *Gazebo*.

Esta secção começa por apresentar a arquitetura do ROS e algumas das suas ferramentas, utilizadas nesta dissertação. São ainda descritos o *Rviz* e o *Gazebo*, utilizado na realização de simulações para teste do algoritmo implementado.

3.4.1 *Robot Operating System* (ROS)

A plataforma de *software* ROS surgiu em 2007, pelas mãos do Laboratório de Inteligência Artificial de Stanford, como suporte aos seus projetos de robótica. Nos anos seguintes sofreu um forte desenvolvimento, especialmente devido ao laboratório de robótica *Willow Garage*, e, atualmente, é uma *framework* amplamente utilizada em projetos e aplicações para robôs, desde manipuladores a projetos complexos de condução autónoma (Fernández et al., 2015).

A filosofia base do ROS é a produção de *software* que possa ser executado em diferentes robôs, apenas com pequenas alterações no código. Desta forma, é possível criar

funcionalidades que podem ser partilhadas e utilizadas por outros projetos sem grande esforço. Ao nível da programação, o ROS também se mostra bastante flexível ao suportar três linguagens de programação, *C++*, *Phython* e *LISP* e bibliotecas em *Java* e *LUA*, sendo possível a comunicação de nodos desenvolvidos em linguagens diferentes (Fernández et al., 2015). Para além desta vantagem, o facto de ser um *software open source*, com suporte para uma vasta gama de sensores e possuir uma grande comunidade ativa, faz com que seja uma ferramenta de eleição para o desenvolvimento de *software* para robôs.

No que diz respeito à arquitetura, e segundo Fernández et al., (2015), esta encontra-se dividida em três grandes níveis: o *Filesystem level*, o *Computation Graph level* e o *Community level* (figura 3.5).

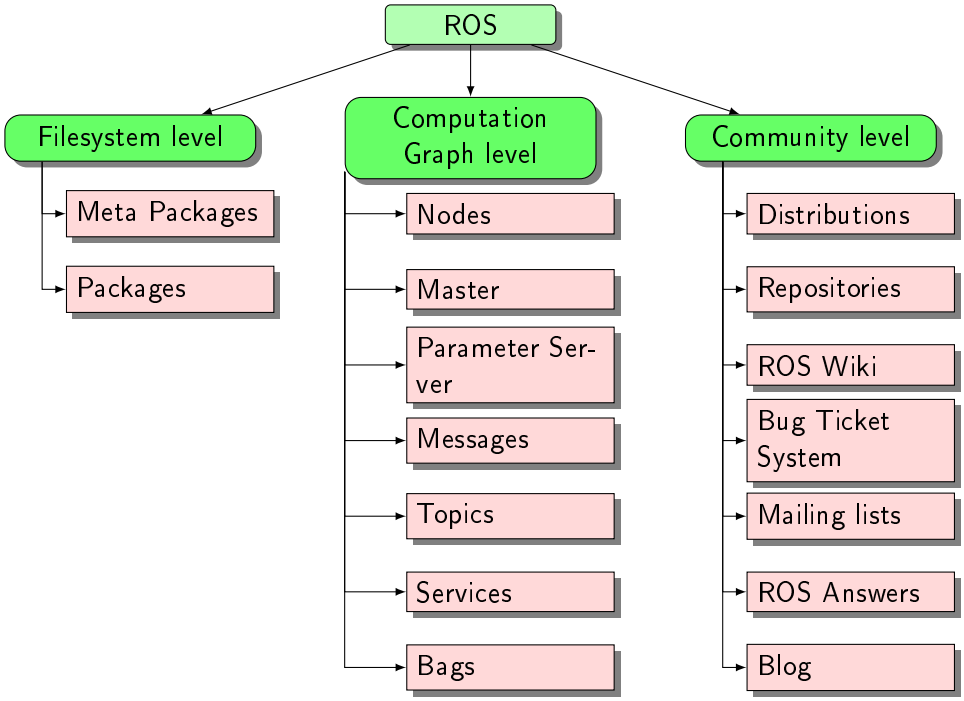


Figura 3.5: Arquitetura ROS segundo Fernández et al., (2015).

Filesystem level

Ao nível do sistema de ficheiros, o ROS está organizado em *Meta Packages* e *Packages*. As *Meta Packages* não são mais que um conjunto de *Packages* que foi agrupado para determinada aplicação. As *Packages* são a unidade mais básica da arquitetura ROS e contêm o essencial para criar um programa ROS. Nelas estão contidos os ficheiros de código e de configuração, os processos e outros elementos necessários à execução do programa.

Computation Graph level

Quando iniciado, o ROS cria uma rede à qual todos os processos estão ligados. Qualquer nodo do sistema pode aceder a esta rede, interagir com outros nodos, visualizar a informação que eles lançam na rede e enviar dados para a rede. Alguns conceitos como

nodos, mensagens, tópicos, serviços e *bags* são esclarecidos de seguida:

- **Nodos**: Os nodos são os processos ROS onde o processamento é realizado. Normalmente, um nodo é responsável por uma só função, assim, quantas mais funções forem implementadas num *package* maior será o número de nodos existentes. Para comunicarem entre si, os nodos podem recorrer à publicação e subscrição de tópicos ou a serviços quando esperam uma resposta de outro nodo. Os nodos são escritos com recurso a bibliotecas ROS como o `roscpp` e o `rospy`.
- **Mensagens**: Os pacotes através dos quais os nodos ROS enviam e recebem informação designam-se de mensagens. Possuem uma estrutura bem definida e conhecida pelos nós que as subscrevem e publicam e, apesar do elevado número de mensagens já definidas, é possível a criação de estruturas personalizadas.
- **Tópicos**: Os tópicos podem ser descritos como os canais de comunicação através dos quais os nodos enviam e recebem mensagens. Quando um nodo envia ou recebe uma mensagem pode-se dizer que publicou ou subscreveu um tópico. Um tópico pode ser acedido por vários nodos.
- **Serviços**: Quando em ROS se pretende uma comunicação do tipo cliente/servidor, isto é, espera-se uma resposta após o envio de dados, utilizam-se serviços. Após o envio de um serviço, o nodo servidor responde ao nodo cliente com a informação da conclusão da tarefa ou com os dados pedidos pelo cliente.
- ***Bags*** - As mensagens que circulam nos tópicos podem ser gravadas em ficheiros, *bags*, para posterior reprodução ou análise. Esta ferramenta permite armazenar dados de um sensor, por exemplo, e desenvolver um algoritmo sem que seja necessária a recolha de novos dados a cada teste.

Community level

É neste nível que se encontra todo o suporte do ROS, desde as distribuições aos repositórios oficiais. Uma das ferramentas mais importantes é o fórum *ROS Answers* onde a comunidade de utilizadores ROS pode colocar dúvidas e/ou sugerir resoluções para problemas de outros utilizadores.

Outras ferramentas do ROS, utilizadas nesta dissertação, foram o *roslaunch* e a *Transform Frame* (TF). O *roslaunch* é um *package* que permite a execução de múltiplos nodos localmente ou remotamente, a configuração dos seus parâmetros e estabelecer se a sua execução é ou não obrigatória para a aplicação se manter operacional. É ainda possível lançar e carregar ficheiros de configuração de aplicações como o *Rviz*.

A livraria TF é um *package* ROS que permite a existência de múltiplos sistemas de coordenadas e publica as transformações geométricas entre os mesmos ao longo do tempo. Entre outras funções, também tem ferramentas para determinar as transformações entre os diferentes referenciais existentes.

3.4.2 *Rviz*

A ferramenta utilizada para visualizar as nuvens de pontos provenientes dos LIDARs e outra informação gerada pelo algoritmo foi o *Rviz*. Esta ferramenta é um *package*

ROS que integra uma interface *OpenGL* capaz de representar a três dimensões dados provenientes de vários sensores e conjugá-los no mesmo referencial utilizando os sistemas de coordenadas de cada sensor para posicionar as suas leituras corretamente face aos outros sensores (Fernández et al., 2015).

Como se pode observar na figura 3.6, o *Rviz* também permite a visualização de marcadores, quer estáticos, quer interativos, e de modelos *Unified Robot Description Format* (URDF). Os marcadores estáticos são apenas formas geométricas ou texto que ajudam a visualizar o tratamento da informação. Já os marcadores dinâmicos permitem interação e os seus dados, como posicionamento face a um referencial ou rotação, são publicados num tópico que pode ser subscrito pela aplicação em execução. Os modelos URDF são ficheiros *XML* com a representação do modelo de um robô.

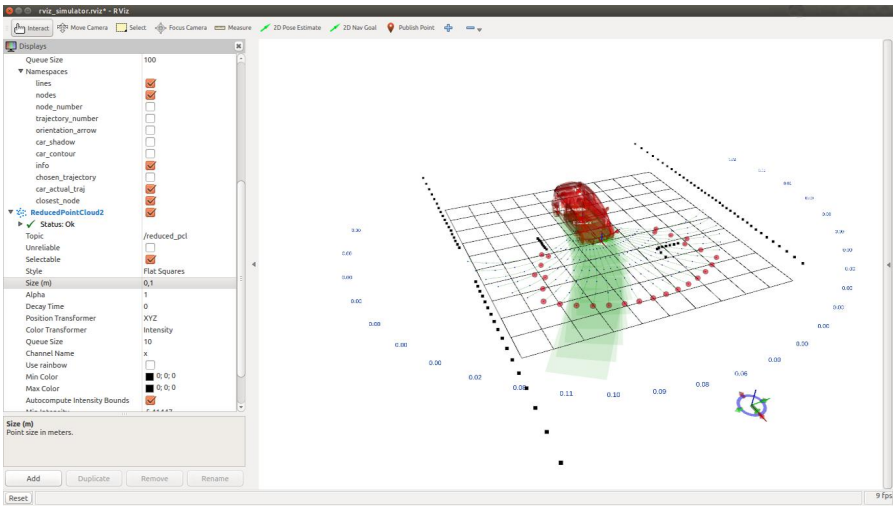


Figura 3.6: Exemplo de interface gráfica do *Rviz* com a representação de tópicos, marcadores visuais e modelo URDF.

O conjunto de tópicos subscritos, os parâmetros visíveis em determinada aplicação e a posição de visualização podem ser guardados em ficheiros de extensão `.rviz` para carregamento futuro.

3.4.3 *Gazebo*

O *Gazebo* foi criado em 2002, na Universidade do Sul da Califórnia, como um simulador de alta fiabilidade para simulação de robôs em ambientes exteriores sobre várias condições e só a partir de 2009 foi integrado com o ROS (OSRF, 2014).

Atualmente, o *Gazebo* pode simular múltiplos robôs em simultâneo, quer em ambientes interiores, quer exteriores, e permite a incorporação de vários tipos de sensores existentes no mercado, como, por exemplo, câmaras, LIDARs, radares, entre outros. Em simulação, este *software* é capaz de gerar informação realista quer dos dados sensoriais, quer das interações físicas entre os diferentes modelos simulados (Fernández et al., 2015).

Devido à integração com o ROS, este simulador foi utilizado nesta dissertação para realizar simulações controladas que permitissem avaliar o algoritmo de navegação local em diversas configurações. Na figura 3.7 está representado um ambiente tipo do *Gazebo*. Neste simulador os modelos de robôs, padrão ou criados pelo utilizador, são carregadas

num ambiente de simulação designado de "mundo". Os sensores, escolhidos de uma lista de *plugins* existente, devem ser integrados nos modelos de forma a criar simulações dinâmicas e representativas da realidade. Podem ainda ser adicionados objetos estáticos para a criação de complexidade na simulação.

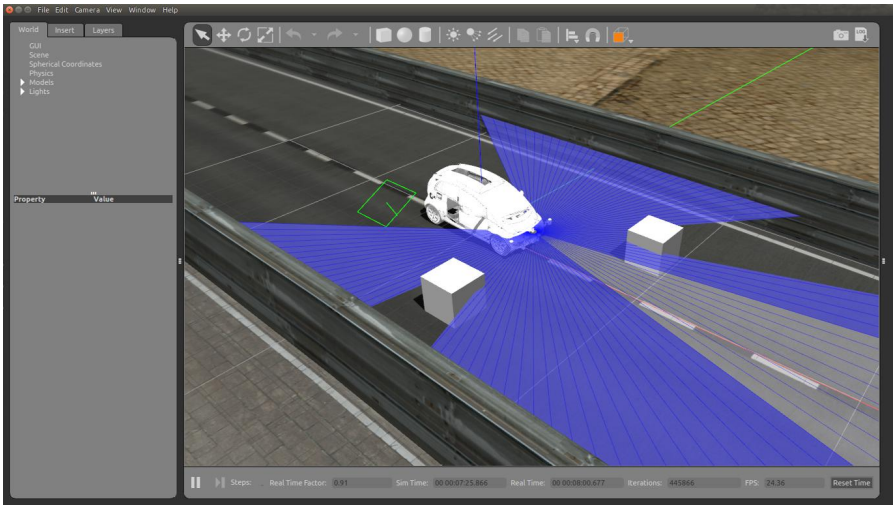


Figura 3.7: Exemplo de ambiente de simulação personalizado no *Gazebo* e de um robô com sensores integrados.

3.5 Pacotes de *software* existentes no LAR

Devido à longevidade do projeto ATLAS, existe um grande historial de *software* criado no LAR que foi útil para a realização desta dissertação. Pacotes ROS como o *Trajectory planner*, uma extensão do LAR *toolkit*, o *Multisensor calibration* e o *Free space detection* foram utilizados e adaptados para a realização da navegação local pelo que são alvo de uma descrição ao longo desta secção.

3.5.1 *Trajectory planner*

De 2009 a 2015, o *software* desenvolvido no LAR foi organizado num repositório denominado de LAR *toolkit* destinado a apoiar o desenvolvimento de *software* dos projetos existentes no laboratório. As versões iniciais (versões 1 e 2), foram baseadas na *framework* CARMEN, enquanto as últimas versões já foram escritas em ROS (Santos, 2018). Dos múltiplos *packages* ROS existentes no repositório, nesta dissertação foi utilizado o `trajectory_planner`.

O *package* `trajectory_planner` foi desenvolvido em ROS por J. F. Pereira, (2012), no âmbito da sua dissertação de mestrado como aplicação do planeamento por múltiplas hipóteses, descrito em Oliveira et al., (2012), ao planeamento da manobra de estacionamento nos robôs ATLAS à escala. Tal como referido na subsecção 2.3.2, o planeamento é baseado na avaliação de múltiplas trajetórias pré-definidas que descrevem a manobra de estacionamento (figura 3.8). Como neste tipo de manobra a orientação com a qual o robô se enquadra com o alvo é determinante para o sucesso do planeamento, na equação de avaliação das trajetórias J. F. Pereira, (2012), atribuiu um peso de 35% ao desvio angular

com o objetivo, 40% à distância linear e os restantes 25% à distância ao obstáculo mais próximo.

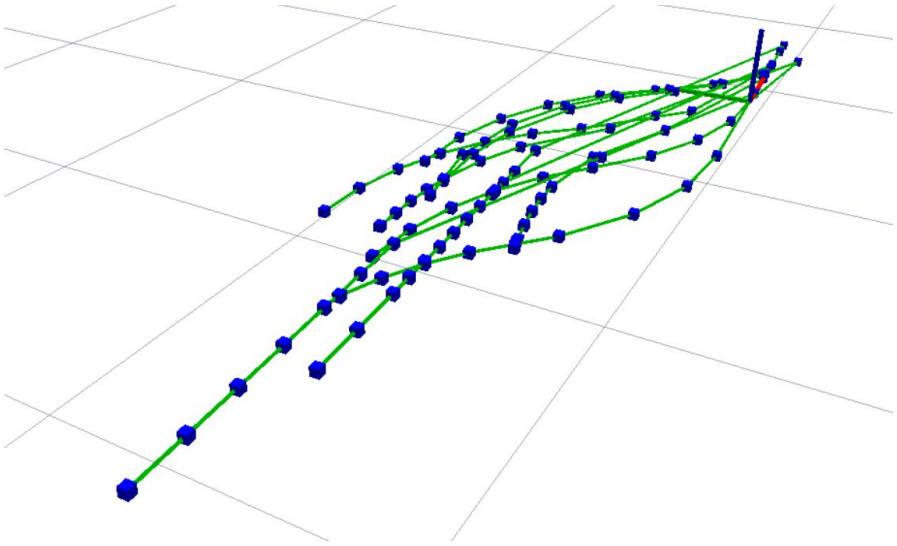


Figura 3.8: Exemplo de trajetórias utilizadas na manobra de estacionamento em série por J. F. Pereira, (2012).

Nesta aplicação também é possível simular o comportamento do algoritmo de seleção de trajetórias com a adição de obstáculos virtuais através de marcadores no *Rviz*. Estes marcadores simulam segmentos de reta que são utilizados no cálculo de colisões através da intercessão com os segmentos das trajetórias (figura 3.9).

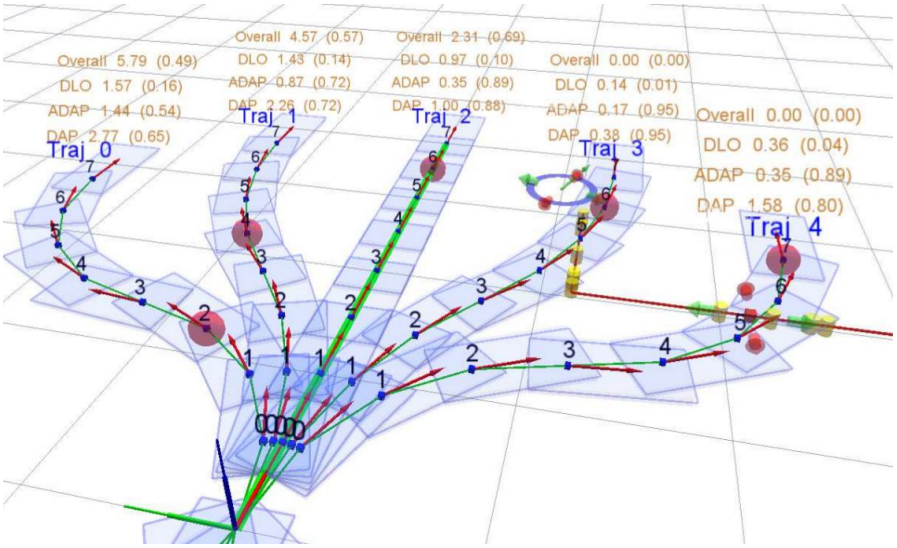


Figura 3.9: Exemplo de deteção de colisões (cilindros amarelos), com obstáculos virtuais, (segmentos de reta vermelhos), (J. F. Pereira, 2012).

3.5.2 *Multisensor calibration*

Para a calibração dos LIDARs instalados no ATLASCAR2 foi utilizado o *package* ROS `multisensor_calibration`, desenvolvido por Vieira da Silva, (2016), no âmbito do projeto ATLASCAR1. Este *package* possui uma interface gráfica através da qual é possível escolher se a calibração é automática ou manual, qual o sensor de referência e quais os sensores a calibrar (figura 3.10). Este *software* está preparado para a calibração de câmaras e LIDARs, 2D e 3D, e através do movimento de uma bola em frente dos mesmos são gerados ficheiros de texto (`.txt`), que contêm as matrizes da transformação geométrica entre o sensor a calibrar e o sensor de referência.

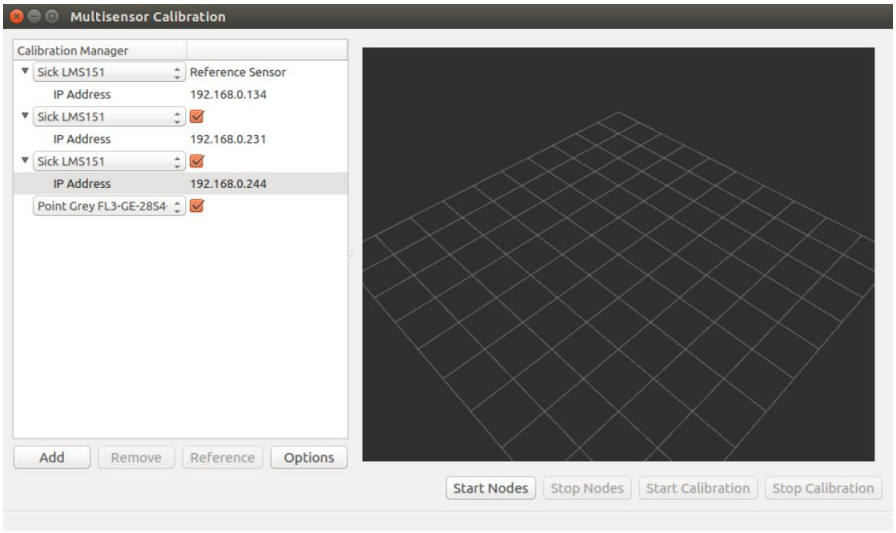
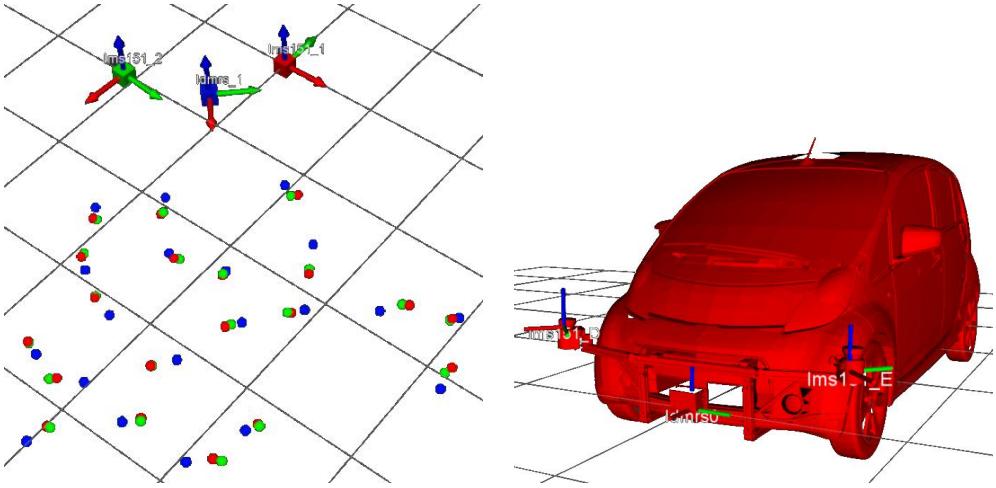


Figura 3.10: Interface gráfica do *package* de calibração multissensorial desenvolvido por Vieira da Silva, (2016).

Para determinar a matriz da transformação entre dois sensores é necessário determinar a posição do centro da bola vista dos referenciais dos dois sensores para posteriormente determinar a matriz da transformação extrínseca. A determinação do centro da bola para sensores de visão está implementa em *OpenCV* com recurso às transformadas de *Hough* ou com um algoritmo de deteção do contorno aproximado da bola. Inicialmente é determinada a posição do centro da bola na imagem que, através da matriz intrínseca da câmara, é convertida em coordenadas reais no referencial da câmara. Nos LIDARs 2D o centro é determinado segmentando a nuvem de pontos em *clusters*, isto é, conjuntos de pontos com grande probabilidade de pertencer ao mesmo objeto. Posteriormente, os *clusters* são analisados com o intuito de detetar círculos, aos quais corresponde a bola. No caso de LIDARs 3D, com múltiplos feixes, a posição do centro da bola é dada pela média das posições calculadas para cada feixe. Em M. Pereira et al., (2016), pode-se encontrar a descrição completa do funcionamento do algoritmo de calibração.

Mais tarde, em Madureira Correia, (2017), utilizou-se este trabalho para a calibração dos sensores instalados no ATLASCAR2 (figura 3.11).



(a) Posições do centro da bola nos referenciais dos LIDARs.

(b) Localização dos LIDARs no ATLASCAR2.

Figura 3.11: Calibração dos três sensores LIDAR no ATLASCAR2, através do *package* de calibração multissensorial, (adaptado de Madureira Correia, (2017)).

3.5.3 *Free space detection*

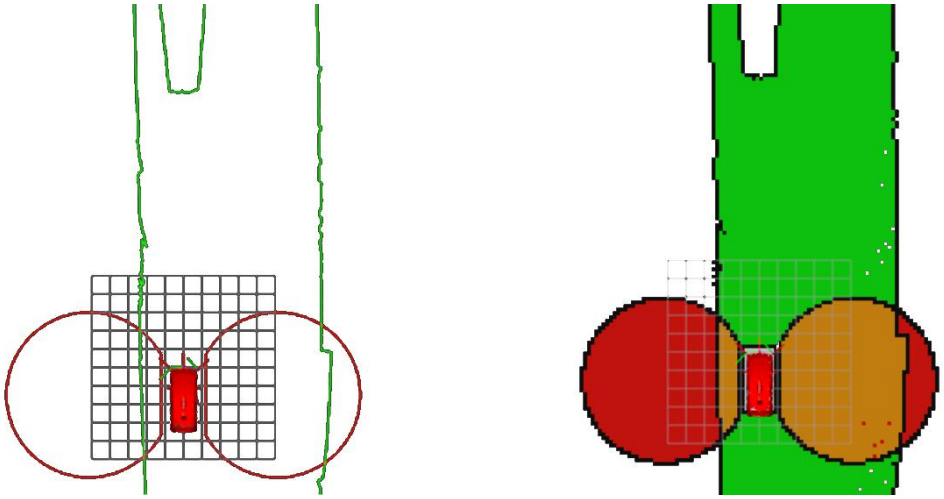
Outro *package* ROS desenvolvido dentro do projeto ATLAS é o `free_space_detection`. Esta aplicação foi criada por Madureira Correia, (2017), com o objetivo de fornecer uma representação do espaço livre navegável pelo ATLASCAR2. Nesta dissertação, este *package* é o responsável pelo fornecimento de dados, sejam estes a nuvem de pontos com a fusão dos dados dos sensores ou o polígono do espaço livre.

É da responsabilidade do *package* `free_space_detection` a comunicação direta com os sensores através da subscrição dos tópicos onde os *scans* laser são publicados. Depois de transformados com as matrizes de transformação presentes nos ficheiros de calibração, obtidos com o *package* `multisensor_calibration`, os *scans* são fundidos numa nuvem de pontos cujo referencial se designa de `/map`. Os pontos da nuvem de pontos são projetados no plano `z=0`, convertidos em coordenadas polares e ordenados por azimute. Esta ordenação visa a remoção de pontos referentes ao mesmo obstáculo detetado por sensores diferentes. Os pontos presentes no mesmo intervalo de azimute — neste caso 0,5° — e mais afastados da origem do referencial são eliminados, sendo os restantes novamente convertidos em coordenadas cartesianas. Esta nuvem de pontos é publicada no tópico `/reduced_pcl` e é utilizada com *input* no algoritmo de navegação local.

Como referido acima, o objetivo deste *package* é representar o espaço navegável pelo ATLASCAR2 e essa representação é realizada de duas formas distintas, um polígono de espaço livre e uma grelha de ocupação. O polígono, representado a verde na figura 3.12a, é obtido diretamente da nuvem de pontos ordenada através da união de pontos consecutivos com segmentos de reta. São ainda adicionadas duas circunferências que não são mais que as restrições cinemáticas do veículo em função do seu raio mínimo de direção.

A grelha de ocupação (figura 3.12b), resulta da discretização do espaço livre em células onde cada uma assume uma cor em função do seu estado. Células dentro do polígono, mas fora das circunferências, são consideradas como livres e representadas a

verde, células dentro das circunferências são consideradas inavegáveis e representadas a vermelho e laranja no caso de estarem fora ou dentro do polígono do espaço livre, respetivamente.



(a) Espaço livre dado por um polígono de espaço livre.

(b) Espaço livre dado por uma grelha de ocupação.

Figura 3.12: Representações do espaço navegável pelo ATLASCAR2, (adaptado de Madureira Correia, (2017)).

Capítulo 4

Solução de navegação local

Ao longo deste capítulo, vai ser descrita a aplicação do algoritmo de planeamento por múltiplas hipótese, bem como as alterações realizadas ao *package* ROS `trajectory_planner` do LAR *toolkit*, descrito na subsecção 3.5.1. Depois de uma breve introdução, na secção 4.1 é explicada a forma de criação das trajetórias, na secção 4.2 está descrito o modelo utilizado para avaliar colisões com as trajetórias, na secção 4.3 é apresentado o alvo do planeamento local e, por fim, na secção 4.4 são detalhados os parâmetros de avaliação das trajetórias.

Da revisão da literatura sobre algoritmos de planeamento, realizada no capítulo 2, existem três algoritmos que se destacam pela sua aplicação em soluções de condução autónoma, o RRT e suas variantes, em Hess et al., (2013), o A* híbrido, em Jo et al., (2013), e os algoritmos de múltiplas hipóteses de trajetória em A. Broggi et al., (2012), Li et al., (2016), e Oliveira et al., (2012).

Como referido na subsecção 2.2.6, o facto dos RRTs lidarem facilmente com problemas de planeamento, como as restrições cinemáticas e não holonómicas dos veículos, e explorarem rapidamente do espaço de trabalho, torna-os bastante apetecíveis para aplicações de condução autónoma reais. No entanto, segundo Hu et al., (2018), a forte dependência deste tipo de algoritmos da heurística de amostragem, isto é, da heurística de marcação aleatória dos pontos candidatos, e da métrica utilizada para determinar o próximo estado, faz com que a trajetória gerada nem sempre seja a mais eficiente. Outras desvantagens residem na natureza dos dados obtidos pelos LIDARs do ATLASCAR2 e na constante atualização do mapa de planeamento. As leituras dos sensores são traduzidas em espaços poligonais, e não mapas gerais com obstáculos bem definidos e limitados no seu interior, pelo que, a capacidade dos RRTs em contornar obstáculos não é totalmente aproveitada. Já o ambiente real extremamente dinâmico faz com que nenhuma trajetória seja totalmente executada pelo veículo, desperdiçando assim o custo computacional associado ao planeamento completo da trajetória.

A resposta do A* híbrido às limitações que o A* convencional apresenta no planeamento de trajetórias executáveis por veículos não holonómicos e a natureza ótima das trajetórias geradas são dois argumentos de peso na escolha deste algoritmo para tarefas de planeamento local. Planear trajetórias em condução autónoma exige uma taxa de processamento que garanta uma atualização rápida do planeamento devido, não só à velocidade de deslocamento do automóvel, mas também à rapidez com que novos obstáculos podem surgir. É aqui que a aplicação de algoritmos de planeamento baseados

numa procura incremental em grelhas de ocupação pode ser colocada em causa pois o custo computacional apresenta um crescimento exponencial com aumento do tamanho da grelha (Hu et al., 2018). O planeamento com recurso a estes algoritmos apenas é considerado viável em situações de condução a baixa velocidade ou utilizando grelhas extremamente grosseiras.

Os algoritmos de planeamento por múltiplas hipóteses são aqueles que, à partida, despertam menos confiança pois nunca é executado um planeamento da trajetória, apenas é escolhida a trajetória mais vantajosa. No planeamento em ambientes pouco dinâmicos e cuja trajetória é escrupulosamente cumprida pelo veículo a utilização deste tipo de algoritmos nem se equaciona. Contudo, nos veículos autónomos, existem taxas de atualização do mapa de planeamento superiores a 20 Hz e, idealmente, o planeamento deve ser executado à mesma taxa que os dados sensoriais são recebidos. Isto exige uma grande capacidade computacional e/ou um algoritmo extremamente eficiente a nível computacional. Como nesta abordagem as trajetórias apenas são avaliadas, a capacidade de computação economizada pode ser transferida para aumentar a velocidade de atualização do planeamento. Esta capacidade de operação em tempo real juntamente com o facto da trajetória planeada não ser executada na totalidade tem ganho terreno em aplicações da condução autónoma nos últimos anos (Hu et al., 2018).

Considerando os aspetos aqui apresentados e o bom desempenho que o algoritmo de abordagem por múltiplas hipóteses, descrito em Oliveira et al., (2012), apresentou na sua aplicação em veículos à escala, com várias competições ganhas pelos projetos ATLAS (Riem de Oliveira, 2014), decidiu-se aplicar esta algoritmo como solução de navegação local do ATLASCAR2.

4.1 Geração de trajetórias

A trajetória descrita por um veículo é o conjunto das sucessivas posições que este ocupa no espaço, em relação ao um sistema de coordenadas pré-definido. No caso de veículos não holonómicos, é possível determinar todos os pontos que compõem a trajetória descrita, sabendo o ângulo das rodas que orientam o veículo e o espaço percorrido com essa orientação. Considerando que o ATLASCAR2 segue o modelo de *Ackermann*, o sistema de coordenadas no qual as trajetórias são definidas encontra-se localizado no centro do eixo traseiro e o Centro de Rotação Instantâneo (ICR) no prolongamento deste eixo. As expressões matemáticas que definem os pontos da trajetória para um modelo deste tipo podem ser encontradas em Oliveira et al., (2012).

Através da observação da figura 4.1 consegue-se determinar a ordenada do ICR sabendo o ângulo de direção, α, e a distância entre eixos, D (equação 4.1). Tendo em conta que a direção do ATLASCAR2 é do tipo *Ackermann*, o ângulo das duas rodas dianteiras não é o mesmo. No entanto, para efeitos de cálculo das trajetórias é considerado o ângulo médio das duas rodas da frente.

$$R = \frac{D}{\tan(\alpha)} \tag{4.1}$$

Depois de conhecida a localização do ICR, segue-se o cálculo da posição angular do veículo, β, ao longo do arco de comprimento A (equação 4.2).

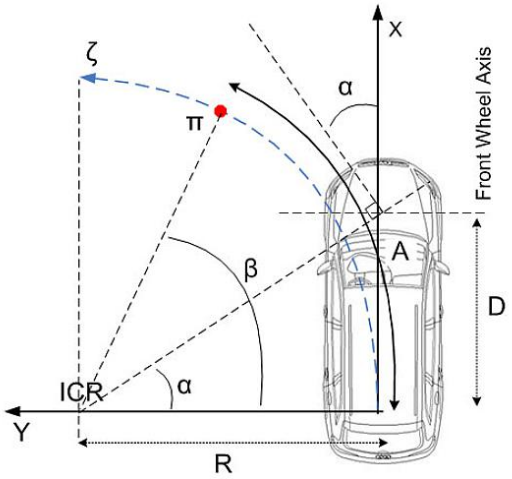


Figura 4.1: Modelo não holonómico de um veículo com direção de *Ackermann*, (adaptado de Oliveira et al., (2012)).

$$\beta = \frac{A}{R} \tag{4.2}$$

Por fim, as coordenadas cartesianas da posição do veículo, $(\prod_x, \prod_y)$ são dadas pela equação 4.3.

$$\begin{bmatrix} \prod_x \\ \prod_y \end{bmatrix} = \begin{bmatrix} R \cdot \sin(\beta) \\ R - R \cdot \cos(\beta) \end{bmatrix} = \begin{bmatrix} \frac{D}{\tan(\alpha)} \cdot \sin\left(\frac{A \cdot \tan(\alpha)}{D}\right) \\ \frac{D}{\tan(\alpha)} \cdot \left(1 - \cos\left(\frac{A \cdot \tan(\alpha)}{D}\right)\right) \end{bmatrix} \tag{4.3}$$

Para facilitar a aplicação do algoritmo na seleção da melhor trajetória, os arcos de circunferência são divididos em troços por nodos igualmente espaçados e cada troço é aproximado a um segmento de reta. Antes das intervenções realizadas durante esta dissertação, o *package* `trajectory_planner` utilizava um ficheiro com duas matrizes de ângulos de direção e comprimentos de arco para, através da equação 4.3, calcular as posições dos nodos. Esta metodologia não se revelava eficiente, pois cada vez que era necessário alterar as trajetórias o ficheiro tinha que ser integralmente reescrito e o código recompilado. Deste modo, foi adotada uma abordagem de cálculo automático das trajetórias em função da velocidade.

Numa abordagem simplificada, a energia cinética de um veículo em movimento é toda dissipada pela fricção no pavimento durante a travagem e é dada em função da massa, m, e velocidade, v, do veículo. O trabalho dissipativo é realizado por uma força tangencial ao movimento, F_t, ao longo da distância de travagem, D, e, segundo o princípio da conservação da energia é possível obter a equação 4.4.

$$E_{dissipada} = E_c \Leftrightarrow D \times F_t = \frac{1}{2} \cdot m \cdot v^2 \Leftrightarrow D = \frac{m \cdot v^2}{2 \cdot F_t} \tag{4.4}$$

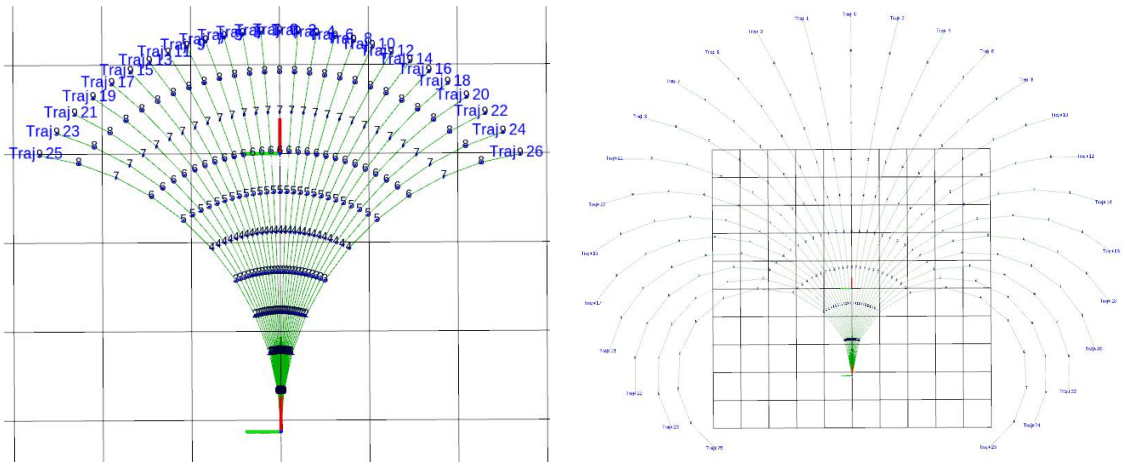
Admitindo que a força tangencial (equação 4.5), responsável pela dissipação da energia, é proporcional à força normal ao pavimento (equivalente ao peso do automóvel), e ao coeficiente de atrito, μ, pode-se determinar a distância de travagem, em função da velocidade, v, e da aceleração da gravidade, g, de acordo com a equação 4.6.

$$F_t = \mu \cdot m \cdot g \tag{4.5}$$

$$D = \frac{v^2}{2 \cdot \mu \cdot g} \tag{4.6}$$

O número de nodos, de trajetórias e o espaçamento angular entre as mesmas passaram também a ser configurados através dos parâmetros `_NUM_NODES_`, `_NUM_TRAJ_` e `_TRAJECTORY_ANGLE_`, respetivamente. A parametrização das trajetórias e modificações na classe `c_manage_trajectory` permitiram a geração dinâmica de trajetórias durante a implementação do algoritmo de avaliação.

Esta geração dinâmica de trajetórias, de acordo com a velocidade do veículo, pode ser observada na figura 4.2 onde se encontram dois exemplos de conjuntos de trajetórias para duas velocidades distintas, 6 m/s e 10 m/s, considerando um coeficiente de atrito típico para piso molhado de 0,4. O referencial de origem das trajetórias é designado de `vehicle_odometry` e tem a sua origem no centro do eixo traseiro do ATLASCAR2.



(a) Velocidade de 6 m/s com 4,67 m de comprimento.

(b) Velocidade de 10 m/s com 12,96 m de comprimento.

Figura 4.2: Trajetórias definidas pelo algoritmo para diferentes velocidades.

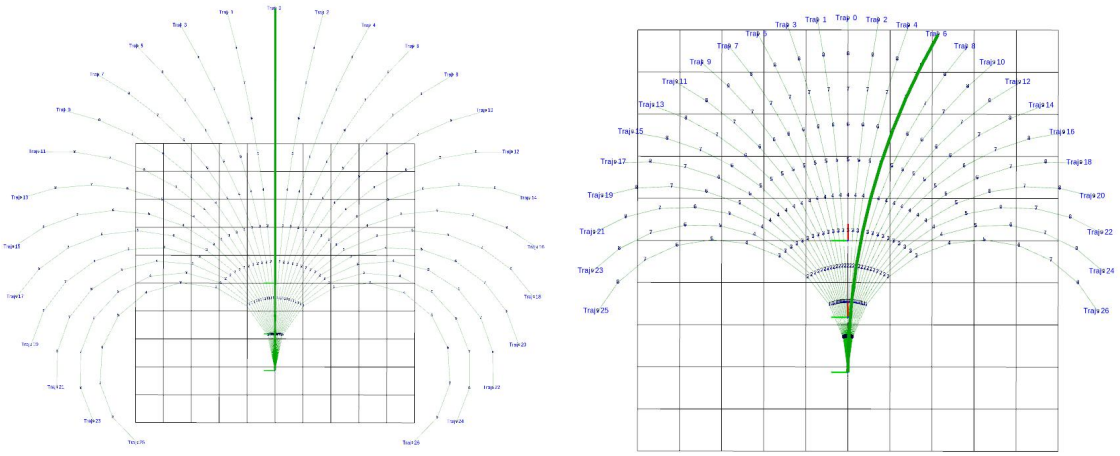
Numa fase de testes mais avançada decidiu-se variar a velocidade, até então estática, consoante o ângulo da trajetória escolhida. Quanto maior o ângulo de direção determinado pela trajetória selecionada no algoritmo de planeamento, mais baixa será velocidade de deslocamento do veículo. Esta variação é realizada linearmente entre duas velocidades pré-estabelecidas, a velocidade máxima admissível e a velocidade de deslocamento segura, configuradas pelos parâmetros `_SPEED_REQUIRED_` e `_SPEED_SAFFETY_`, respetivamente. O excerto de código 4.1 apresenta a função responsável por esta variação, onde `_MAX_STEERING_ANGLE_` é o ângulo máximo de direção retirado da tabela 3.1 como a média dos ângulos máximos da roda de dentro e da roda de fora.

Excerto de código 4.1: Função que devolve a velocidade em função do ângulo da trajetória a seguir.

```
double angle_to_speed(double angle)
```

```
{
        double m = (_SPEED_SAFFETY_ - _SPEED_REQUIRED_) / (_MAX_STEERING_ANGLE_ *
            M_PI / 180);
        return (m * abs(angle) + _SPEED_REQUIRED_);
}
```

Esta variação da velocidade acrescentou um problema ao algoritmo. No caso da velocidade mínima ser baixa o comprimento das trajetórias diminui de tal forma que o algoritmo fica instável, gerando inclusive colisões. A solução encontrada estabelece um comprimento mínimo para as trajetórias igual ao raio mínimo de viragem do ATLASCAR2, 4,5 m, permitindo inversões de marcha quando na presença de obstáculos frontais sem a possibilidade de contorno dos mesmos. Na figura 4.3 é bem visível a variação do comprimento das trajetórias em função do ângulo escolhido.



(a) Direção a 0 graus com velocidade de 10 m/s e comprimento de 12,96 m.

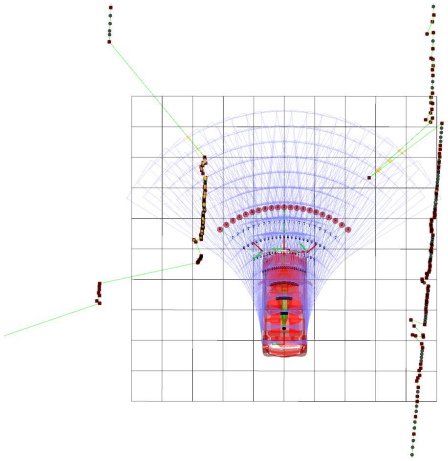
(b) Direção 9 graus para a direita com velocidade de 8,05 m/s e comprimento de 8,39 m.

Figura 4.3: Trajetórias definidas pelo algoritmo em função da direção escolhida para um intervalo de velocidades de 1 m/s a 10 m/s.

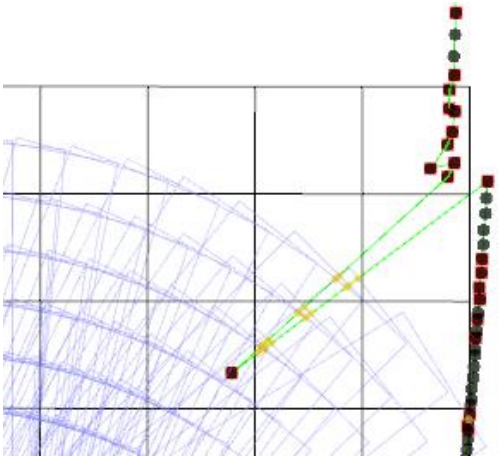
4.2 Deteção de colisões

A deteção de colisões das trajetórias traçadas não pode ser realizada considerando exclusivamente a linha da trajetória pois esta não representa o espaço ocupado pelo carro. Tal como em Oliveira et al., (2012), o *package* `trajectory_planner` utiliza um retângulo para representar o espaço ocupado pelo veículo em cada nodo das múltiplas trajetórias. Este retângulo deve possuir as mesmas dimensões que o veículo onde o algoritmo de planeamento é aplicado pois as colisões serão diretamente calculadas entre os retângulos e os obstáculos. Nesta abordagem diz-se que existe uma colisão quando duas linhas se intercetam, isto é, quando uma das quatro linhas do contorno do retângulo que representa o veículo se cruza com uma linha que une dois pontos do obstáculo. Na figura 4.4 são visíveis os retângulos representativos do ATLASCAR2 e os pontos de colisão, representados por cilindros amarelos.

Quando na presença de curvas acentuadas, este método de deteção de colisões não se revelou eficaz pois a nuvem de pontos fornecida pelo *package* `free_space_detection`



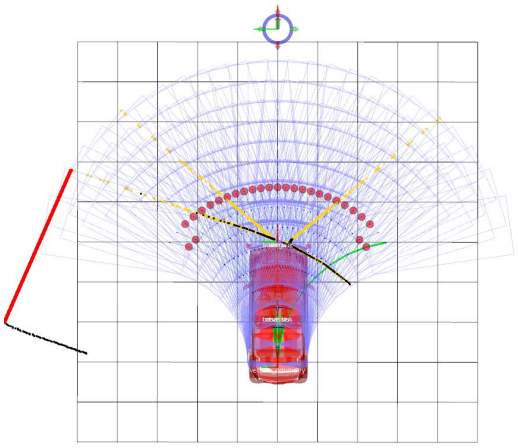
(a) Representação do veículo em rectângulos.



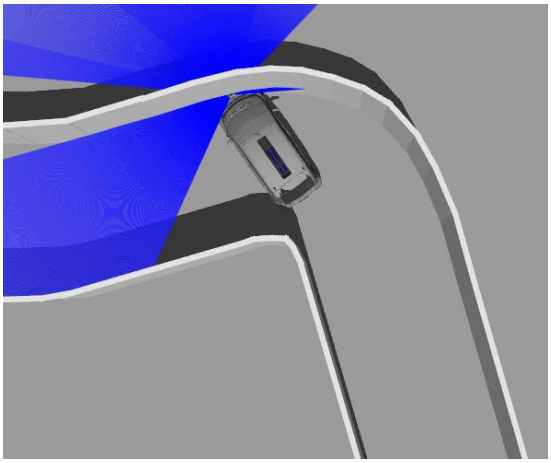
(b) Pormenor de deteção de colisões (cilindros amarelos).

Figura 4.4: Exemplo de deteção de colisões de acordo com o método das interseções entre segmentos de reta, adotado no *package* original.

não está segmentada em obstáculos. O facto do algoritmo receber apenas uma nuvem de pontos faz com que este considere todos os pontos da nuvem pertencentes ao mesmo obstáculo, o que conduz à união dos limites da estrada em curvas que possuem grandes ângulos de viragem, criando assim uma barreira virtual e eliminando as trajetórias nessa direção. Na figura 4.5a este fenómeno está representado por uma linha vermelha, e perante tais condições, o algoritmo não consegue evitar colisões (figura 4.5b), devido a uma tomada de decisão tardia.



(a) Representação da união das duas margens da estrada (linha vermelha).



(b) Situação de colisão visualizada no simulador *Gazebo*.

Figura 4.5: Situação de simulação de curva acentuada onde o método de deteção de colisões falha, provocando uma colisão.

Foram identificadas duas abordagens para a resolução deste problema. A segmentação

da nuvem de pontos proveniente do *package* `free_space_detection` ou a alteração do método de deteção de colisões no *package* `trajectory_planner`. Numa tentativa de solucionar o problema do lado deste *package*, optou-se por alterar o método de cálculo das colisões. Em vez de interseções de segmentos de reta passou-se a identificar os pontos da nuvem que se encontram dentro dos retângulos. Desta forma, sempre que um retângulo tem uma inclusão, isso é sinónimo que a trajetória possui pontos de colisão.

Para determinar se um ponto está dentro ou fora do retângulo é utilizado o *Winding Number* (WN). Na sua forma mais simplista, o cálculo do WN percorre todos os lados de um polígono no sentido anti-horário e se o ponto a testar se encontrar à esquerda do vetor definido por um lado do polígono é adicionada uma unidade ao WN do ponto, se se encontrar à direita é subtraída uma unidade ao WN do ponto. Um ponto está dentro do polígono se o seu WN for diferente de zero (Sunday, 2012). A figura 4.6 apresenta a ilustração deste método de cálculo num ponto em três situações distintas para uma melhor compreensão do conceito.

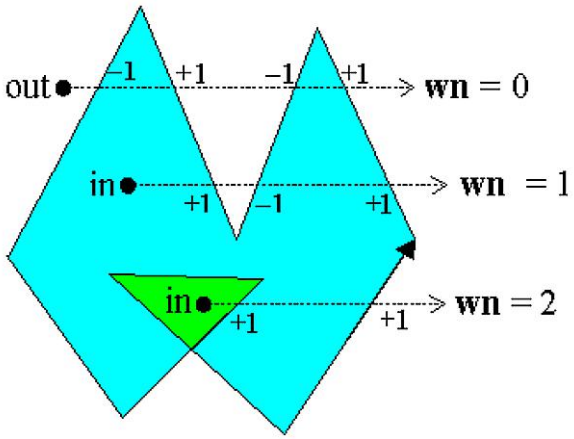


Figura 4.6: Representação gráfica da metodologia utilizada para o cálculo do WN, (Sunday, 2012).

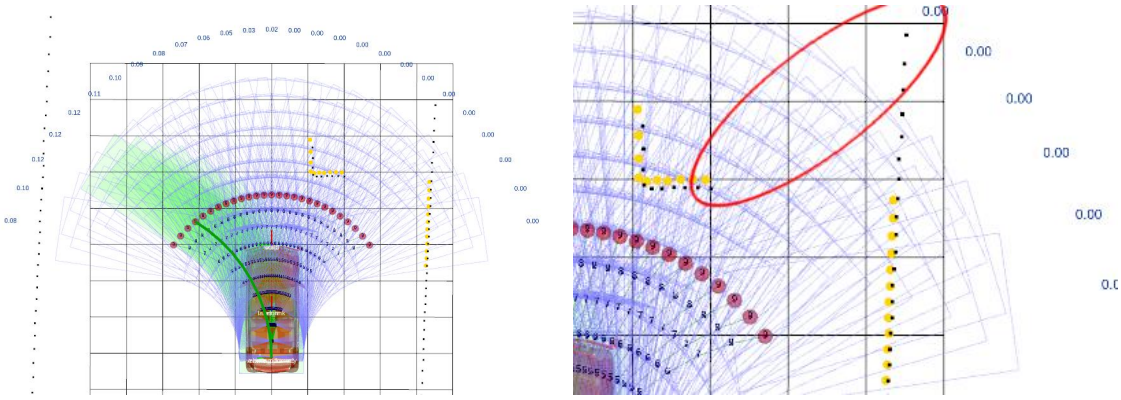
O excerto de código 4.2 resulta da alteração realizada ao *package* `trajectory_planner` e resume o teste de colisões efetuado com recurso ao WN.

Excerto de código 4.2: Resumo do cálculo de colisões das trajetórias através do WN.

```
// testa todos os nós até ao mais próximo
for (int n = 0; n <= trajectory->closest_node; ++n)
{
        (...)
        // testa todas as linhas do retângulo
        for (size_t l = 0; l < trajectory->v_lines[n].size(); ++l)
        {
                (...)
                // testa todos os obstáculos
                for (size_t o = 0; o < vo.size(); ++o)
                {
                        // testa todos os pontos do obstáculo
                        for (size_t lo = 0; lo < vo[o].x.size(); ++lo)
                        {
                                (...)
```

```
                                int wn = wn_PnPoly(P, &car_polygon, car_polygon_size);
                                if (wn != 0)
                                {
                                        (...)
                                        trajectory->collision_pts.push_back(p);
                                        trajectory->score.FS *= 0;
                                }
                        }
                }
        }
}
```

O resultado da modificação do método de deteção de colisões é bem visível na figura 4.7. Ao contrário da figura 4.4b, na área circundada a vermelho da figura 4.7b não existe a união dos pontos da nuvem com segmentos de reta e as colisões deixam de ser detetadas ao longo desses segmentos. Neste caso, são bem visíveis os novos pontos de colisão, representados por cilindros amarelos, coincidentes com os pontos da nuvem.



(a) Exemplo de simulação de obstáculos.

(b) Ampliação da simulação de obstáculos onde é visível a não união de osbtáculos diferentes (elipse a vermelho).

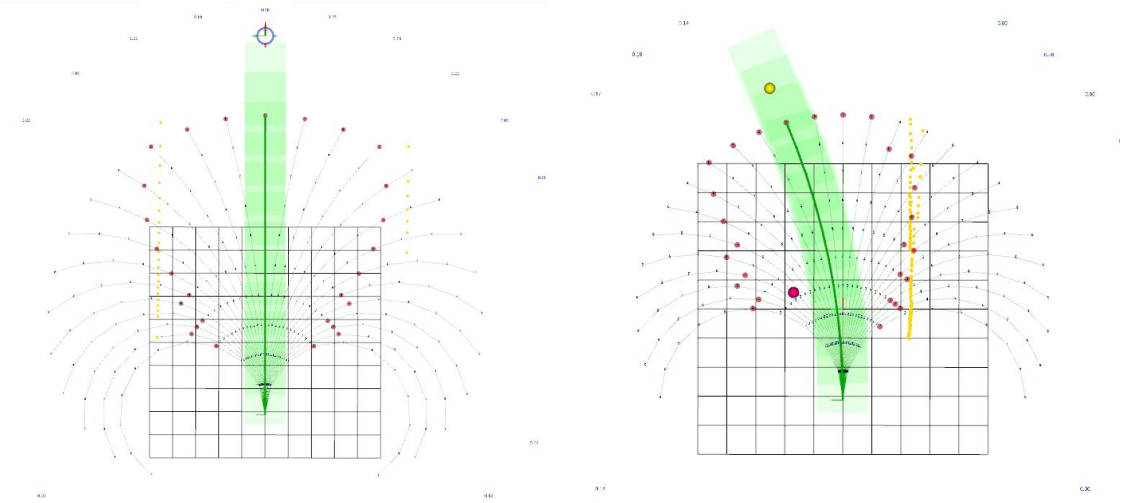
Figura 4.7: Exemplo de deteção de colisões através de inclusões, utilizando a metodologia do WN.

Para a deteção de interseções com obstáculos não físicos, isto é, as linhas da estrada, recorreu-se à mesma metodologia do WN. Como este tipo de obstáculos não são fisicamente intransponíveis, são passados ao algoritmo numa nuvem de pontos distinta que é analisada da mesma forma da que representa os obstáculos físicos, mas o resultada da análise de colisões é armazenado num parâmetro diferente da trajetória. Esta distinção mantém a independência entre os diferentes tipos de obstáculos.

4.3 Ponto atrator

É da responsabilidade do algoritmo de navegação local escolher a trajetória que o veículo deve tomar em cada instante, até o objetivo final da navegação ser alcançado. Como tal, deve existir um ponto, final ou intermédio, que sirva de atrator ao planeamento, isto é, o objetivo. O *package* `trajectory_planner` já possui um nodo responsável pela publicação deste ponto, o `APgenerator`. Este nodo publica um marcador visual interativo em

Rviz que pode ser posicionado manualmente pelo utilizador com três graus de liberdade, translações em x e y e rotação em z (figura 4.8a). Durante a execução do programa ROS, sempre que este marcador é reposicionado, o tópico com a sua posição e orientação é atualizado e o algoritmo passa a ter um novo objetivo. Nas situações de simulação, descritas no capítulo 6, este ponto é mantido fixo ao longo do tempo para não influenciar os resultados obtidos.



(a) Ponto atrator definido pelo marcador visual interativo.

(b) Ponto atrator definido pelos *way points*, a amarelo.

Figura 4.8: Diferentes tipos de pontos atratores e sua representação.

Para um planeamento em condições reais a utilização deste ponto atrator deve ser dispensada. Nessa situação devem ser considerados os *way points*, fornecidos pelo módulo GPS de navegação global, como pontos atratores durante a execução da rota. Como tal, foi desenvolvido um novo nodo, designado de `APwaypoints`, responsável pela subscrição dos pontos anterior e seguinte da rota. A subscrição dos dois pontos serve para garantir que o ponto atrator está dentro de um raio que influencie o algoritmo. No caso da distância ao próximo ponto da rota ser superior do limiar de normalização, explicado na subsecção 4.4.1, este não vai influenciar a escolha da trajetória. Para garantir que o ponto atrator se encontra sempre a uma distância capaz de influenciar a escolha da trajetória, é traçada uma reta que une os dois *way points* mais próximos e sobre ela é marcado o ponto atrator a uma distância fixa da posição do veículo (figura 4.8b).

Partindo da resolução em ordem a x do sistema de equações 4.7, no excerto de código 4.3 pode ser encontrada a função que determina o ponto atrator de acordo com os *way points* fornecidos pelo módulo GPS de navegação global. Depois de definidos os parâmetros da equação da reta que une os dois pontos da rota, m e b, o ponto atrator é marcado na interceção da reta com uma circunferência de raio r, centrada no referencial de origem das trajetórias.

$$\begin{matrix} y = m \cdot x + b \\ x^2 + y^2 = r^2 \end{matrix} \Leftrightarrow x = \frac{-2 \cdot m \cdot b + \sqrt{(2 \cdot m \cdot b)^2 - 4 \cdot (1 + m^2) \cdot (b^2 - r^2)}}{2 \cdot (1 + m^2)} \tag{4.7}$$

Excerto de código 4.3: Função de criação do ponto atrator de acordo com os dois *way points* recebidos.

```
geometry_msgs::PointStamped waypointsAnalise::computeAP(const std_msgs::
    Float64MultiArray& waypoints_in)
{
        double r = 12;
        geometry_msgs::PointStamped point;
        double m = (waypoints_in.data[3] - waypoints_in.data[2]) / (waypoints_in.data
            [1] - waypoints_in.data[0]);
        double b = waypoints_in.data[3] - m * waypoints_in.data[1];
        double x = (-2 * m * b + pow((pow((2 * m * b), 2) - 4 * ((pow(m, 2) + 1) * (
            pow(b, 2) - pow(r, 2)))), 0.5)) / (2 * (pow(m, 2) + 1));
        point.point.x = x;
        point.point.y = (m * x + b);
        point.point.z = 0;
        return point;
}
```

4.4 Parâmetros da trajetória

O módulo de cálculo da pontuação e seleção das trajetórias implementado no *package* `trajectory_planner` por J. F. Pereira, (2012), não sofreu alterações de maior no decorrer dos trabalhos desta dissertação. Ainda assim foram ajustados os parâmetros de avaliação das trajetórias e de normalização, e foi acrescentado um parâmetro que adiciona a informação da linha central da estrada ao algoritmo, pelo que se passa a explicar o método de cálculo e avaliação das trajetórias.

O cálculo da pontuação de uma trajetória depende da Distância ao Ponto Atrator (DAP), da Diferença Angular ao Ponto Atrator (ADAP), da Distância Mínima aos Obstáculos (DLO), da Linha Central (CL) e do Espaço Livre (FS). Numa tentativa dos três primeiros parâmetros entrarem na equação 4.8 com a mesma ordem de grandeza, garantindo assim uma avaliação dos resultados mais fiável, é realizada a sua normalização. Depois de normalizados, estes parâmetros são afetados de uma constante, m_k, que determina o seu peso na equação de classificação das trajetórias. O parâmetro CL considera a interseção das trajetórias com as linhas da estrada, neste caso a central. Este parâmetro possui um valor unitário para as trajetórias que não intercetam a linha e um valor pré-definido, de acordo com o tipo de linha e o planeamento que se pretende efetuar, para as trajetórias que se sobrepõem à linha. O parâmetro FS funciona com um parâmetro exclusivo, isto é, indica se uma trajetória possui ou não pontuação. A afetação do FS ocorre durante o processo do cálculo de colisões, excerto de código 4.2, onde o seu valor unitário é anulado quando é detetada um colisão, eliminando assim a pontuação da trajetória.

$$P_{traj} = (m_1 \cdot DAP_{norm} + m_2 \cdot ADAP_{norm} + m_3 \cdot DLO_{norm}) \cdot CL \cdot FS \tag{4.8}$$

Em situações de simulação, onde o ponto atrator força o veículo a deslocar-se sempre em frente e o objetivo é uma navegação segura evitando colisões, os pesos, m_k, dos parâmetros normalizados, DAP, ADAP e DLO, e o peso do parâmetro CL são definidos segundo a tabela 4.1. O peso dado à DAP é muito reduzido pois, do ponto de vista

da navegação, a forma como o ponto atrator está definido não é vantajosa perante as curvas do traçado. O parâmetro m_2 é nulo pela mesma razão, sendo que em situações de navegação local reais estes parâmetros devem ser testados e ajustados de acordo com o ambiente de navegação. O parâmetro CL deve ser pré-definido como 1, no caso de não se pretender incorporar a informação da linha central, e como 0,2, quando o objetivo é a circulação do modelo na via da direita. Um exemplo da escolha de trajetórias segundo a equação 4.8 e com os parâmetros da tabela 4.1 pode ser encontrado na figura 4.9.

Tabela 4.1: Pesos dos parâmetros da trajetória em situação de simulação.

Parâmetro	DAP	ADAP	DLO	CL
m_k	m_1	m_2	m_3	—
Peso	0,1	0,0	0,9	1(s. simples) / 0,2(s. linhas)

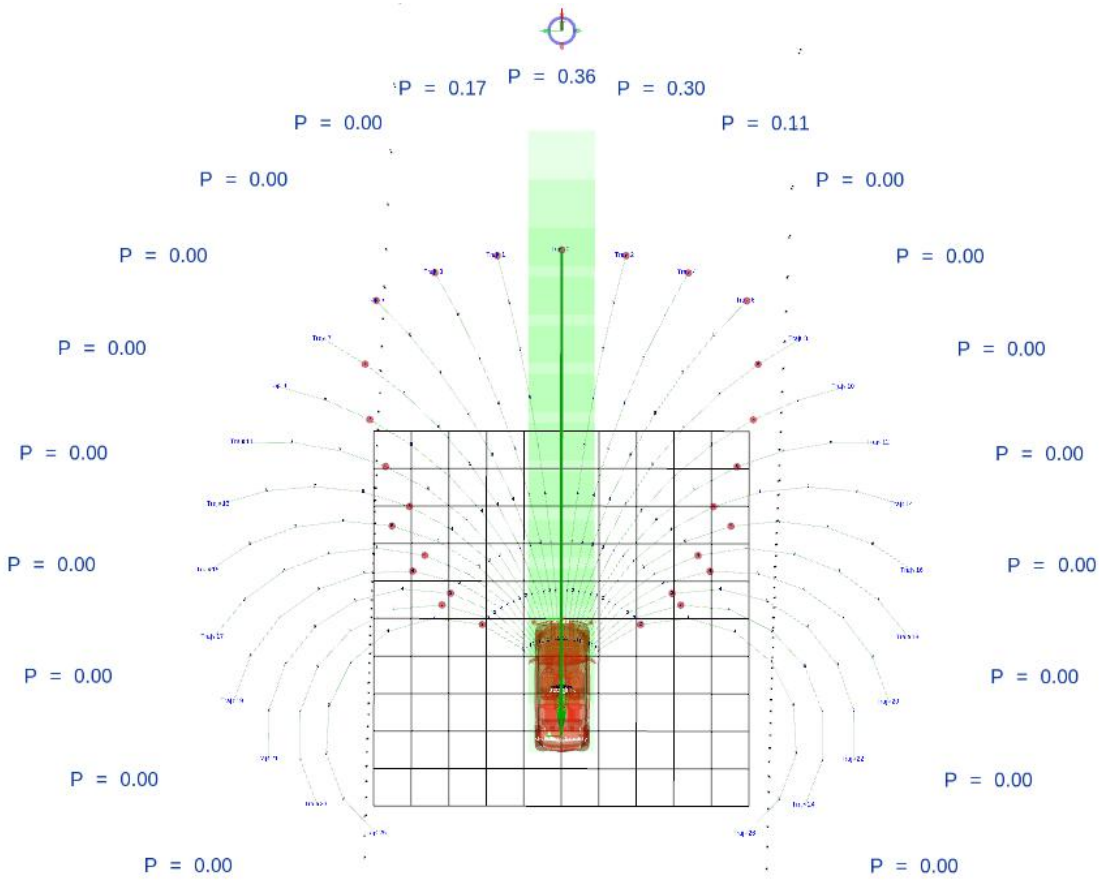


Figura 4.9: Exemplo de escolha de trajetórias por análise da sua pontuação numa situação de simulação simples.

Quando em execução, o *package* `trajectory_planner` apresenta o diagrama de nodos e tópicos da figura 4.10. O nodo responsável pelo cálculo das pontuações das trajetórias é o `trajectory_planner_nodelet` e o nodo `APgenerator` dá a posição do ponto atrator nas situações de simulação. A nuvem de pontos representativa dos obstáculos físicos está publicada no tópico `/reduced_pcl` e a que representa as linhas do tópico `/line_pcl`. A

direção e velocidade, resultantes do planeamento, são publicadas no tópico `/cmd_vel`.

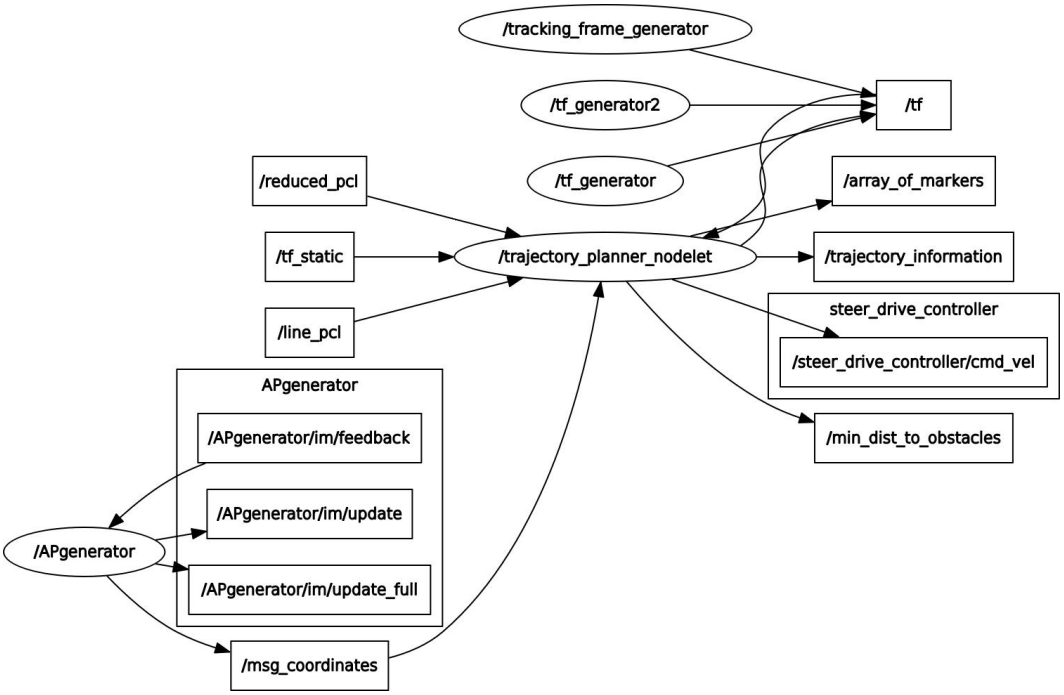


Figura 4.10: Diagrama dos nodos e tópicos executados pelo algoritmo de planeamento.

Nas quatro subsecções seguintes é explicado o cálculo dos três parâmetros DAP, ADAP e DLO, e a afetação do parâmetro CL.

4.4.1 Distância ao Ponto Atrator (DAP)

Em situações de planeamento real, onde o ponto atrator representa o objetivo a atingir, devem ser priorizadas as trajetórias que conduzem o veículo a esse mesmo ponto. No caso dos *way points*, as suas coordenadas no referencial de planeamento traduzem o ponto para o qual o veículo se deve direcionar de forma a cumprir uma rota pré-definida. O parâmetro DAP é a distância a que um nodo da trajetória, isto é, um ponto resultante da discretização dos arcos de circunferência, se encontra do ponto atrator. A DAP é calculada para todos os nodos da trajetória, sendo selecionada a mais baixa e o nodo onde que verifica a distância mínima é designado do nodo mais próximo do objetivo, sendo o planeamento de cada trajetória executado até esse nodo. Esta distância é a distância euclidiana medida em metros onde (Π_x, Π_y) são as coordenadas do nodo mais próximo do ponto atrator e (AP_x, AP_y) as coordenadas do ponto atrator. A DAP é calculada com a equação 4.9 e graficamente representada na figura 4.11.

$$DAP = \sqrt{(\Pi_x - AP_x)^2 + (\Pi_y - AP_y)^2} \tag{4.9}$$

Posteriormente, o valor da DAP é normalizado pela equação 4.10, onde o parâmetro de normalização, Δ_{DAP}, é a distância máxima à qual o ponto atrator exerce influência na escolha da trajetória, `maximum_admissible_to_DAP`. Pontos mais distantes possuem um valor de DAP nulo.

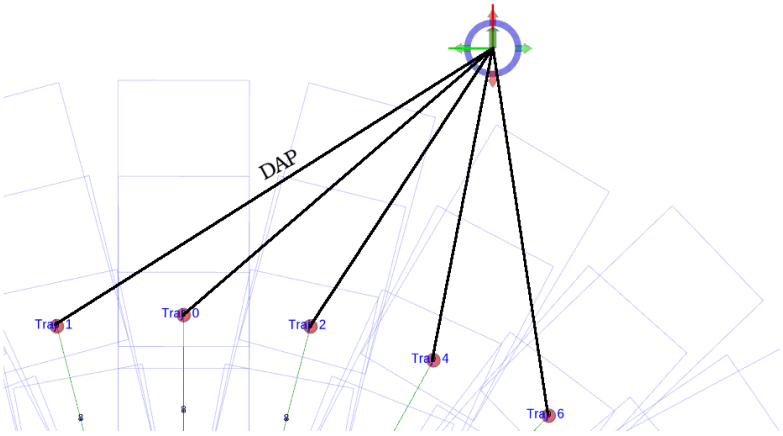


Figura 4.11: Cálculo da DAP desde os nodos das trajetórias mais próximos do objetivo ao ponto atrator.

$$DAP_{norm} = \max\left(0, 1 - \frac{DAP}{\Delta_{DAP}}\right) \tag{4.10}$$

O código responsável pela implementação destes cálculos pode ser encontrado no excerto 4.4. Foi necessário alterar o código original e limitar o valor normalizado da DAP a zero para não existirem trajetórias com pontuações negativas.

Excerto de código 4.4: Cálculo do valor da DAP e seleção do nodo mais próximo do ponto atrator.

```
double maximum_admissible_to_DAP = 8.0;
for (int i = 0; i < (int)vt.size(); ++i)
{
        // Cálculo DAP
        compute_DAP(vt[i], AP);
        // Normalização DAP
        vt[i]->score.DAPnorm = max(0.0, (1 - (vt[i]->score.DAP) /
            maximum_admissible_to_DAP));
}

t_func_output c_manage_trajectory::compute_DAP(c_trajectoryPtr& trajectory,
    t_desired_coordinates& AP)
{
        trajectory->score.DAP = 10e6;
        trajectory->closest_node = -1;
        for (size_t i = 0; i < trajectory->x.size(); ++i)
        {
                double DAP_prev = sqrt(pow(trajectory->x[i] - AP.x, 2) + pow(
                    trajectory->y[i] - AP.y, 2));
                if (DAP_prev < trajectory->score.DAP)
                {
                        trajectory->score.DAP = DAP_prev;
                        trajectory->closest_node = i;
                }
        }
}
```

4.4.2 Diferença Angular ao Ponto Atrator (ADAP)

Na navegação local, a orientação com que o veículo atinge determinado objetivo, seja ele final ou intermédio (*way point*), não tem a mesma importância que em manobras específicas como o estacionamento, por exemplo. Ainda assim o *package* `trajectory_planner` está preparado para determinar a diferença angular entre o veículo e o ponto atrator, ADAP. Esta distância é determinada pela diferença angular entre a orientação do veículo, Π_θ, no ponto da trajetória mais próximo do objetivo e a orientação do ponto atrator, AP_θ. A ADAP é determinada, em valor absoluto, pela equação 4.11 para facilitar a sua normalização final. Na figura 4.12 está representada a diferença angular entre o ponto final de uma trajetória e o ponto atrator.

$$ADAP = |\Pi_\theta - AP_\theta| \tag{4.11}$$

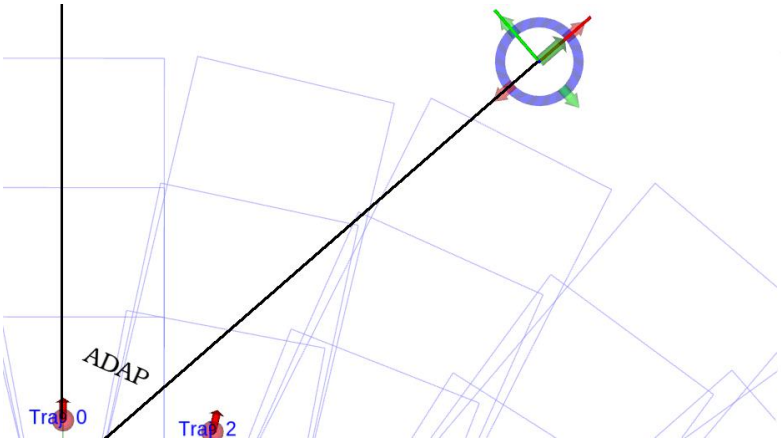


Figura 4.12: Cálculo da ADAP entre o nodo final de uma trajetória e o ponto atrator.

Antes de entrar no cálculo final da pontuação da trajetória, e já em valor absoluto, a ADAP é normalizada pela equação 4.12. Para a normalização da ADAP, a constante de normalização utilizada é π.

$$ADAP_{norm} = \max\left(0, 1 - \frac{ADAP}{\pi}\right) \tag{4.12}$$

À semelhança da DAP, também aqui se alterou o código, excerto 4.5, para garantir que a ADAP normalizada é sempre positiva.

Excerto de código 4.5: Cálculo do valor da ADAP.

```
for (int i = 0; i < (int)vt.size(); ++i)
{
        // Cálculo ADAP
        trajectory->score.ADAP = compute_ADAP(trajectory, AP, trajectory->
            closest_node);
        // Normalização ADAP
        vt[i]->score.ADAPnorm = max(0.0, (1 - (vt[i]->score.ADAP / (M_PI))));
}
```

```
double c_manage_trajectory::compute_ADAP(c_trajectoryPtr& trajectory,
    t_desired_coordinates& AP, int i)
{
       double adap = abs(trajectory->theta[i] - AP.theta);
       if (adap > M_PI)
       {
              adap = 2 * M_PI - adap;
       }
       return adap;
}
```

4.4.3 Distância Mínima aos Obstáculos (DLO)

A pontuação das trajetórias com colisões é automaticamente anulada pelo parâmetro FS. No entanto, de entre as restantes trajetórias, pode não ser seguro selecionar a trajetória que conduz o veículo para o local mais próximo do objetivo. Caso esta trajetória possua obstáculos demasiado próximos pode ser perigosa, quer para o veículo, quer para os obstáculos, podendo estes representarem seres vivos. Há ainda que considerar que as trajetórias se encontram discretizadas em segmentos de reta e caso esta aproximação seja demasiado grosseira podem existir colisões não detetadas. A DLO pode ser utilizada para evitar estas situações pois ao representar a distância a que o obstáculo mais próximo se encontra da trajetória irá favorecer as trajetórias com mais espaço livre ao seu redor.

Para o cálculo da DLO pode ser utilizada a equação 4.9, substituindo as coordenadas do ponto atrator pelas coordenadas dos pontos da nuvem de pontos que representam os obstáculos. A figura 4.13 ilustra o cálculo da DLO para a trajetória número 2.

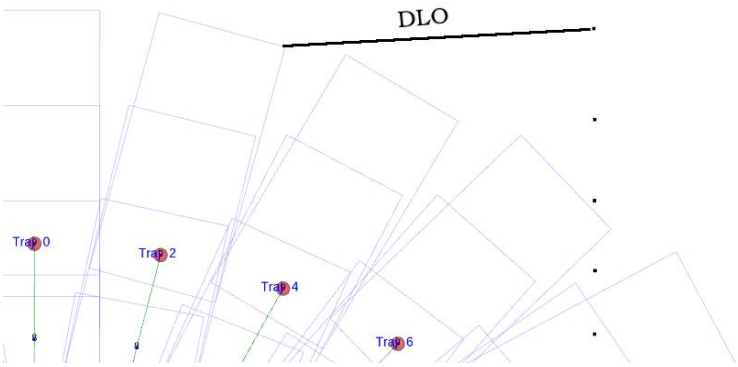


Figura 4.13: Cálculo da DLO para o nodo da trajetória número 2 que se encontra mais próximo dos obstáculos.

A normalização da DLO é realizada pela equação 4.13, onde o parâmetro de normalização, Δ_{DLO}, é a distância de referência de influência dos obstáculos, configurada pelo parâmetro `maximum_admissible_to_DLO`. Ao contrário dos outros parâmetros da trajetória, a DLO pode assumir valores superiores a 1, caso os obstáculos se encontrem acima do limiar de normalização.

$$DLO_{norm} = \frac{DLO}{\Delta_{DLO}} \tag{4.13}$$

Neste parâmetro o código do *package* `trajectory_planner` manteve-se inalterado (excerto 4.6).

Excerto de código 4.6: Cálculo do valor da DLO.

```
double maximum_admissible_to_DLO = 10.0;
for (int i = 0; i < (int)vt.size(); ++i)
{
        // Cálculo DLO
        compute_DLO(vt[i], vo);
        // Normalização DLO
        vt[i]->score.DLOnorm = (vt[i]->score.DLO) / maximum_admissible_to_DLO;
}

t_func_output c_manage_trajectory::compute_DLO(c_trajectoryPtr& trajectory, std::
    vector<t_obstacle>& vo)
{
        trajectory->score.DLO = 10.0; // Igual ao maximum_admissible_to_DLO
        (...)
        for (size_t lo = 0; lo < vo[o].x.size(); ++lo)
        {
                double DLOprev = sqrt(pow(trajectory->v_lines[n][1].x[0] - vo[o].x[lo
                    ], 2) + pow(trajectory->v_lines[n][1].y[0] - vo[o].y[lo], 2));
                if (trajectory->score.DLO > DLOprev)
                {
                        trajectory->score.DLO = DLOprev;
                }
                (...)
        }
        (...)
        return SUCCESS;
}
```

4.4.4 Linha Central (CL)

Quanto à introdução da informação da deteção da linha central no algoritmo, não foi realizado nenhum cálculo aritmético, contrariamente aos outros parâmetros descritos. Neste caso, o fator CL possui um valor inicial igual à unidade que pode ser alterado, ou não, consoante a deteção de inclusões de pontos pertencentes à linha central nas trajetórias. Inicialmente foi pensado considerar a informação das linhas como um termo adicional ao somatório, no entanto, optou-se por um fator multiplicativo que pudesse permitir a anulação da pontuação da trajetória. Podem existir situações, como o caso de uma dupla linha contínua, onde ultrapassar a linha pode ser tão ou mais perigoso que ir em direção a um obstáculo, pois põe em causa a segurança de outros veículos que circulam em sentido contrário. Desta forma o valor de CL pode ser adaptado ao tipo de linha detetada, consoante o grau de permissibilidade que esta apresenta em ser transposta, alterando o valor pré-definido de `W_CL`.

Um exemplo de planeamento executado de acordo com a deteção da linha central pode ser encontrado na figura 4.14, onde se pode observar que, apesar da primeira trajetória da esquerda conduzir a um ponto mais afastado dos obstáculos físicos, a trajetória que apresenta maior pontuação é a central, mesmo implicando uma navegação mais próxima dos obstáculos.

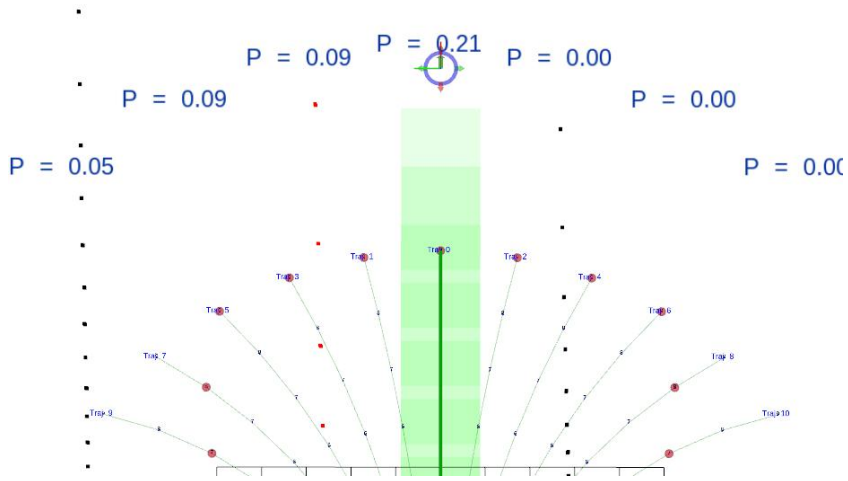


Figura 4.14: Exemplo da afetação da pontuação das trajetórias à esquerda, que intercetam a linha central (pontos a vermelho), pelo parâmetro CL.

O código apresentado no excerto 4.7 é responsável pela afetação do parâmetro CL, com a particularidade de apenas as trajetórias da esquerda e central poderem ser afetadas por este parâmetro. Desta forma permite-se que o algoritmo possa conduzir o veículo novamente à via de circulação da direita, depois deste se deslocar temporariamente à esquerda da linha.

Excerto de código 4.7: Resumo da atribução de valores ao parâmetro CL recorrenndo ao WN.

```
// testa todos os nós até ao mais próximo
for (int n = 0; n <= trajectory->closest_node; ++n)
{
        (...)
        // testa todas as linhas do retângulo
        for (size_t l = 0; l < trajectory->v_lines[n].size(); ++l)
        {
                (...)
                // testa todos os obstáculos
                for (size_t oo = 0; oo < vv.size(); ++oo)
                {
                        // testa todos os pontos do obstáculo
                        for (size_t ll = 0; ll < vl[oo].x.size(); ++ll)
                        {
                                (...)
                                int wn = wn_PnPoly(P, &car_polygon, car_polygon_size);
                                if (wn != 0 && trajectory->alpha[0] >= 0)
                                {
                                        trajectory->score.CL = W_CL;
                                }
                        }
                }
        }
}
```

Capítulo 5

Ambiente de simulação

O objetivo deste capítulo é a descrição do ambiente de simulação utilizado para a realização dos testes e afinação do algoritmo, iniciando-se essa descrição com as definições do modelo do robô utilizado e o sensor LIDAR a ele acoplado, secções 5.1 e 5.2. É depois apresentada a integração destes modelos no *Gazebo* (secção 5.3), e o ambiente de testes (secção 5.4). Por fim, na secção 5.5, é ilustrado o resultado final do simulador.

Na fase inicial de implementação da solução de navegação local foram utilizados *bags* com dados reais dos sensores LIDAR para comprovar as alterações realizadas ao *package* `trajectory_planner`. Depois de realizadas as adaptações aos parâmetros característicos do ATLASCAR2 utilizados pelo algoritmo, as modificações nos marcadores visuais do *Rviz*, as alterações de referenciais e a criação de nodos capazes de publicar as TFs entre os referenciais existentes, decidiu-se testar o comportamento da solução de navegação local com a variação dos parâmetros das trajetórias. Foi nesta fase que os dados sensoriais armazenados se tornaram obsoletos. A avaliação de qualquer algoritmo de planeamento é uma tarefa impossível de realizar *offline* devido ao papel ativo que o algoritmo possui na tarefa de navegação. Não se podem retirar quaisquer conclusões acerca do desempenho de um algoritmo quando não são cumpridas as indicações por ele fornecidas em função dos dados percecionados, ainda mais quando esses dados dependem diretamente das últimas trajetórias escolhidas pelo algoritmo (Ansuategui et al., 2014).

Perante a impossibilidade de atuação direta do algoritmo de navegação local no controlo do ATLASCAR2 e de forma a garantir repetibilidade no ambiente de teste, optou-se por avaliar o desempenho do algoritmo recorrendo à simulação. Para a realização das simulações foi escolhido o *software Gazebo*, apresentado no capítulo 3, devido à sua capacidade de simular dados de sensores LIDAR e interações físicas entre os modelos, como por exemplo, colisões. A estas funcionalidades acresce ainda a vantagem da sua integração com o ROS, permitindo a interação do simulador com o *package* de navegação local.

5.1 Definição do modelo

Antes da realização das simulações, foi necessária a criação de um modelo virtual do ATLASCAR2 que respeitasse as suas dimensões físicas e descrevesse com algum rigor as restrições cinemáticas associadas à direção do tipo *Ackermann*. Apesar do *Gazebo* possuir

uma extensa lista de *plugins* que permitem o controlo de diversos robôs e plataformas, não se encontra disponível um *plugin* que permita controlar um robô com direção do tipo *Ackermann*. Numa tentativa de encontrar um modelo do controlador da direção recorreu-se à ROS *Wiki*. Aqui foi encontrado o robô *CIR-KIT-Unit03* (CIR-KIT, 2016a), utilizado na competição japonesa de condução autónoma *Tsukuba Challenge*, e que possui um controlador de direção do tipo *Ackermann*.

Apesar da grande complexidade deste modelo, com sete *meta packages* capazes de realizar tarefas de navegação e de deteção de humanos, para o simulador descrito neste capítulo apenas foram utilizados alguns *packages* de três *meta packages*, organizados de acordo com a figura 5.1.

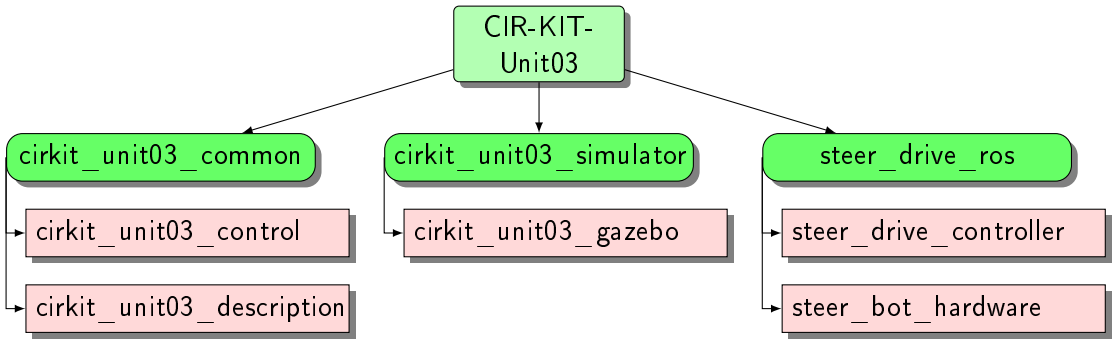


Figura 5.1: Arquitetura do robô *CIR-KIT-Unit03* utilizada para a realização do simulador do ATLASCAR2.

O *meta package* `cirkit_unit03_common` possui dois pacotes ROS responsáveis pelas configurações do robô. No *package* `cirkit_unit03_control` são configurados os parâmetros dos controladores da direção, `steer_drive_controller` e `steer_bot_hardware`, e do controlador da posição das juntas do robô utilizado, `joint_state_controller`, de acordo com o modelo que se encontra descrito no *package* `cirkit_unit03_description`. Enquanto que os parâmetros dos controladores permaneceram inalterados, a descrição do robô teve que sofrer adaptações para respeitar as dimensões do ATLASCAR2. São três os ficheiros responsáveis pela descrição no modelo:

- ***cirkit_unit03.xacro***: É o ficheiro principal, onde são incluídos os dois ficheiros abaixo. Aqui são dimensionadas as restantes propriedades do modelo como a massa, o modelo URDF que representa o ATLASCAR2, a posição dos sensores que integram o robô e a posição dos eixos do robô em relação ao referencial global.

- ***cirkit_unit03_macro.xacro***: Neste ficheiro são criadas as representações das rodas, a partir das suas dimensões gerais, e são posicionadas de acordo com as medidas do veículo. É inicialmente definida uma roda padrão e as restantes são reposicionadas de acordo com planos de simetria. Também são definidas as juntas das rodas que permitem movimentos de direção e de rotação, responsáveis pelo deslocamento do robô.

- ***materials.xacro***: Aqui são definidas as propriedades visuais dos elementos que compõem o modelo.

Depois da configuração, o resultado final do modelo, representado no *Gazebo*, pode ser visualizado na figura 5.2.

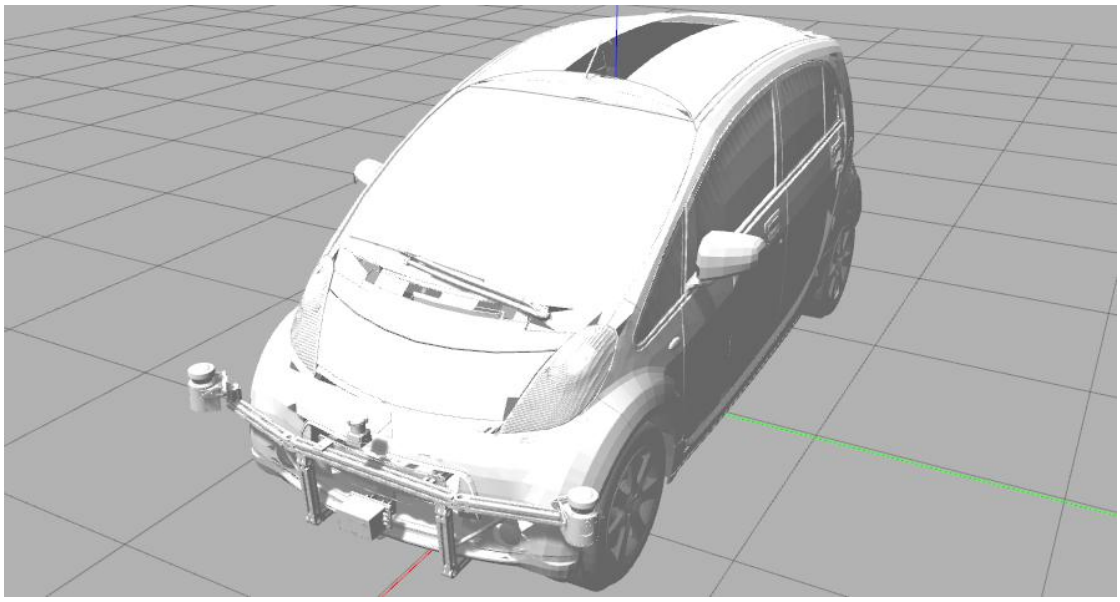


Figura 5.2: Modelo URDF do ATLASCAR2 representado no *Gazebo* e utilizado para a realização das simulações.

O *meta package* `cirkit_unit03_simulator` é responsável pela integração do modelo criado no *Gazebo* e pelo envio de informação do sensor para o planeador de trajetórias, sendo descrito mais pormenorizadamente na secção 5.3.

Por fim, os controladores estão definidos no *meta package* `steer_drive_ros` e o seu funcionamento pode ser explicado com recurso à figura 5.3. Depois do algoritmo de planeamento receber e processar os dados do sensor a bordo do modelo, publica a orientação e a velocidade no veículo no tópico `/cmd_vel` do tipo `geometry_msgs::Twist`. Este tópico vai ser subscrito pelo controlador primário da direção, `steer_drive_controller`, que converte a orientação que o veículo deve seguir no ângulo de direção de uma roda virtual, situada no centro do eixo dianteiro, e a velocidade linear de deslocamento em velocidade de rotação de uma roda traseira. Numa segunda etapa, e quando configurado para tal, o controlador `steer_bot_hardware` aplica os resultados simplificados do controlador anterior a uma configuração do tipo *Ackermann* com velocidades de rotação diferentes nas rodas traseiras e ângulos de direção distintos nas rodas dianteiras. Esta conversão é realizada com base nos parâmetros de configuração apresentados no código 5.1, e de acordo com as dimensões reais do ATLASCAR2 (tabela 3.1). A informação é então enviada para o controlador do sistema, `joint_state_controller`, que publica as posições das juntas do robô, posteriormente interpretadas pelo *Gazebo*.

Excerto de código 5.1: Parâmetros de configuração do controlador da direção.

```
steer_bot_hardware_gazebo:
rear_wheel : 'rear_wheel_joint'
front_steer : 'front_steer_joint'
virtual_rear_wheels : ['base_to_right_rear_wheel', 'base_to_left_rear_wheel']
```

```
virtual_front_wheels : ['base_to_right_front_wheel', 'base_to_left_front_wheel']
virtual_front_steers : ['base_to_right_front_steer', 'base_to_left_front_steer']

enable_ackermann_link: true
wheel_separation_w : 1.310
wheel_separation_h : 2.55
```

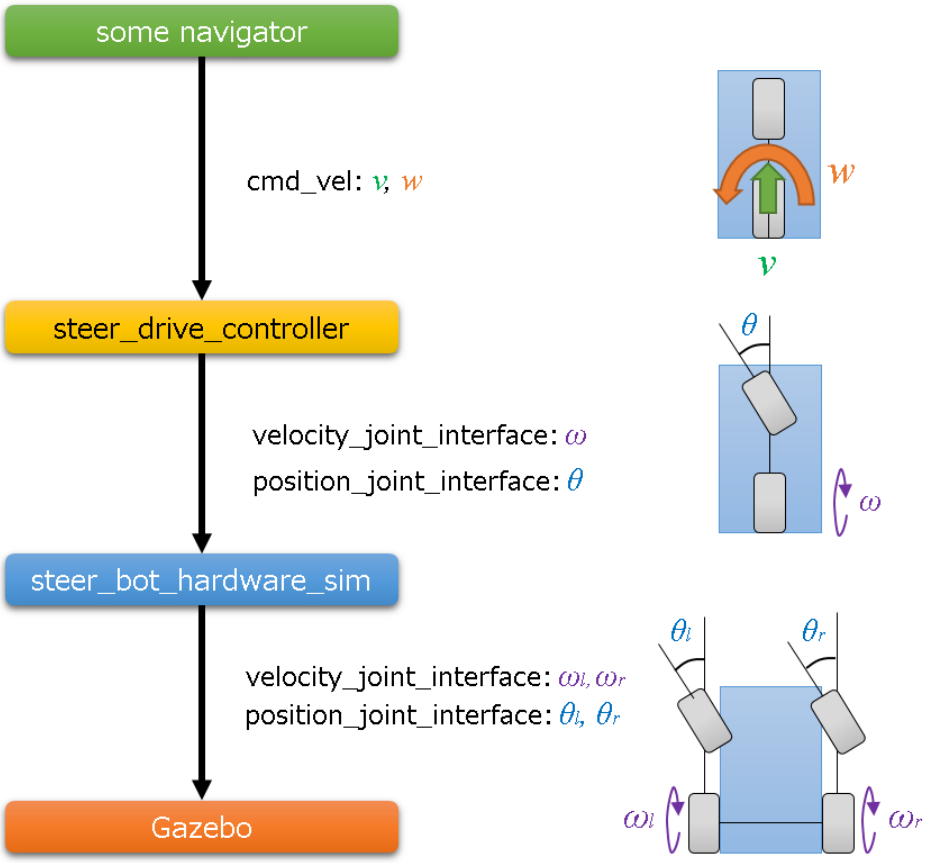


Figura 5.3: Esquema de funcionamento do controlador de direção e velocidade do modelo do robô, (CIR-KIT, 2016b).

5.2 Definição de sensores

Para simular as nuvens de pontos geradas pelos sensores instalados no ATLASCAR2 foi integrado um sensor LIDAR no modelo do robô. Inicialmente, foi ponderada a simulação com dois sensores nas extremidades no para-choques da frente, com posicionamento semelhante aos *Sick LMS151*, para obter dados da frente e das laterais do robô. No entanto, devido à distinção entre elementos visuais e elementos com colisões, realizada pelo *Gazebo*, a incorporação de um sensor com um ângulo de abertura maior foi suficiente para geração de dados frontais e laterais, com cobertura idêntica à conseguida pelos dois sensores reais. Este sensor foi posicionado logo acima do *Sick LD-MRS400001* para facilitar a publicação de transformações entre os referenciais.

A configuração do sensor simulado está descrita no excerto 5.2. Foi utilizado o *plugin* `gazebo_ros_head_hokuyo_controller` para simular o LIDAR 2D e a nuvem de pontos publicada no tópico `/simulator_laser_scan`. O alcance do sensor foi definido como 50 m, à semelhança dos LIDARs 2D instalados no ATLASCAR2. Como referido acima, foi utilizada uma abertura do feixe de 5 rad para permitir a recolha de dados laterais apenas com um sensor.

Excerto de código 5.2: Parâmetros de configuração do sensor LIDAR incorporado no modelo do robô.

```
<gazebo reference="front_bottom_lrf">
       <sensor type="ray" name="front_bottom_lrf_hokuyo_sensor">
              <pose>0 0 0 0 0 0</pose>
              <visualize>true</visualize>
              <update_rate>40</update_rate>
              <ray>
                     <scan>
                            <horizontal>
                                   <samples>1080</samples>
                                   <resolution>0.25</resolution>
                                   <min_angle>-2.50</min_angle>
                                   <max_angle>2.50</max_angle>
                            </horizontal>
                     </scan>
                     <range>
                            <min>0.50</min>
                            <max>50.0</max>
                            <resolution>0.01</resolution>
                     </range>
                     <noise>
                            <type>gaussian</type>
                            <mean>0.0</mean>
                            <stddev>0.01</stddev>
                     </noise>
              </ray>
              <plugin name="gazebo_ros_head_hokuyo_controller" filename="
                  libgazebo_ros_laser.so">
                     <topicName>/simulator_laser_scan</topicName>
                     <frameName>front_bottom_lrf</frameName>
              </plugin>
       </sensor>
</gazebo>
```

5.3 Integração no *Gazebo*

Depois de criado o modelo virtual do ATLASCAR2 e configurado o sensor LIDAR, é necessária a sua integração no simulador. Essa tarefa é da responsabilidade do *meta package* `cirkit_unit03_simulator` e o excerto de código 5.3 é referente ao ficheiro `.launch` que inicializa o simulador. O primeiro ficheiro a ser carregado é o espaço da simulação, que pode ser inicializado como um espaço vazio ou conter já os elementos referentes a uma dada aplicação. Quando é carregado o espaço, são enviados parâmetros que configuram a inicialização do *Gazebo* como por exemplo, se a simulação se encontra

em pausa, (`arg name="paused"`), ou se é apresentada a interface gráfica de simulação, (`arg name="gui"`).

De seguida são incluídos os ficheiros com a descrição do robô e com os controladores e é lançado o nodo `urdf_spawner`, que é responsável pela visualização do modelo URDF no ambiente de simulação.

Os últimos nodos a serem lançados na simulação são o `cirkit_unit03_gazebo` e o `cirkit_unit03_gazebo_line`. Estes nodos foram criados e adicionados a este *meta package* para permitir a integração direta do simulador com o algoritmo de navegação local. O primeiro é responsável pela conversão do *scan* laser, com os dados do limite da pista, numa nuvem de pontos do tipo `sensor_msgs::PointCloud2` e pela publicação da mesma num tópico já subscrito pelo algoritmo, permitindo a execução do mesmo, quer em ambiente real, quer em ambiente de simulação, sem que seja necessária nenhuma alteração. O segundo executa uma tarefa idêntica, mas para a linha central da estrada. Esta é considerada como uma parede virtual e detetada por um segundo laser, cujo *scan* também é convertido numa nuvem de pontos.

Excerto de código 5.3: Ficheiro de lançamento e inicialização do simulador em *Gazebo*.

```
<launch>
        (...)
        <include file="$(find gazebo_ros)/launch/empty_world.launch">
                <arg name="world_name" value="worlds/monaco.world"/>
                <arg name="paused" value="true"/>
                <arg name="gui" value="true"/>
                <arg name="use_sim_time" value="true"/>
                <arg name="debug" value="false"/>
                <arg name="headless" value="false"/>
        </include>
        <include file="$(find cirkit_unit03_gazebo)/launch/description.gazebo.launch"
            />
        <include file="$(find cirkit_unit03_control)/launch/control_gazebo.launch"/>
        (...)
        <node name="urdf_spawner" pkg="gazebo_ros" type="spawn_model" respawn="false"
             output="screen" args="-urdf -model cirkit_unit03 -param
            robot_description"/>
        <node name="cirkit_unit03_gazebo" pkg="cirkit_unit03_gazebo" type="
            cirkit_unit03_gazebo_node" output="screen"/>
        <node name="cirkit_unit03_gazebo_line" pkg="cirkit_unit03_gazebo" type="
            cirkit_unit03_gazebo_line_node" />
        (...)
</launch>
```

5.4 Ambiente de testes

Para a realização de simulações com validade comparativa foi necessário desenvolver uma aplicação que mantivesse o ambiente sempre controlado (subsecção 5.4.1), e um parâmetro que conseguisse quantificar a qualidade do percurso realizado pelo robô no simulador (subsecção 5.4.2).

5.4.1 Experiência proposta

A experiência proposta para a realização das simulações foi um circuito fechado comparado a uma estrada real. Como o algoritmo de navegação, em situações reais, apenas deteta objetos físicos através dos LIDARs, foram criadas paredes que simulam os limites da estrada. Na figura 5.4 é visível o primeiro modelo criado para os testes no simulador, onde apenas existem paredes, dentro das quais o robô tem que circular. Este modelo foi criado no *Building editor* do *Gazebo* com base numa pista de *rally-cross*. No entanto, esta ferramenta do *Gazebo* está otimizada para a criação de espaços de edifícios o que impossibilitou a criação de um modelo da pista com dimensões suficientes para a circulação do robô à escala real.

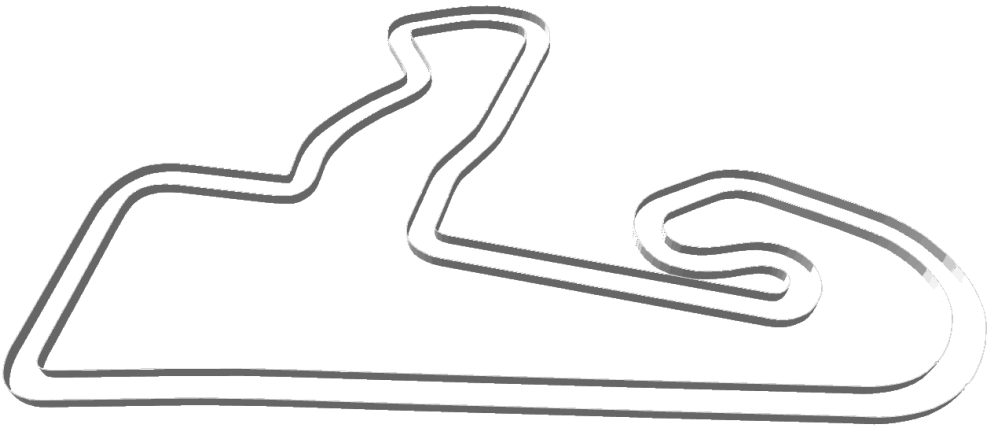


Figura 5.4: Primeiro ambiente de simulação desenvolvido no *Building editor* do *Gazebo*, pista de *rally-cross*.

Numa segunda abordagem, foi utilizada uma representação tridimensional do Circuito do Mónaco sem variações da cota de altitude, desenvolvida por Villamil Vuelta, (2018) (figura 5.5). A escolha deste circuito para a aplicação de simulação teve como base dois

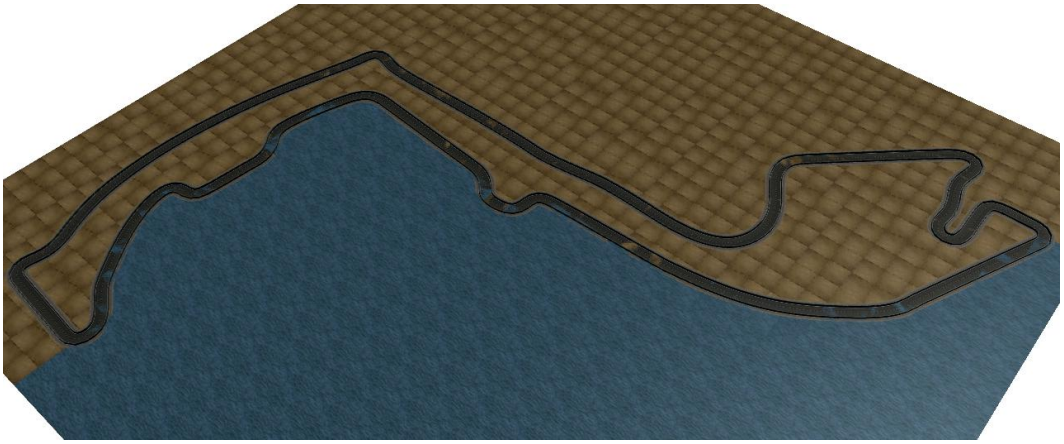


Figura 5.5: Ambiente de simulação final adaptado de Villamil Vuelta, (2018), pista de Fórmula 1.

fatores: o circuito ser desenhado em ambiente urbano e possuir a menor extensão de pista de todos os circuitos do campeonato de Fórmula 1. Ao simular uma estrada real garante-se que as dimensões são adequadas à circulação do robô à escala real e o reduzido tamanho possibilita a realização de simulações mais curtas.

5.4.2 Avaliação do planeamento

Estando concluída a construção do ambiente de simulação, passou-se à definição de um critério que quantificasse a qualidade do planeamento. Para além de garantir a inexistência de colisões, é necessária uma avaliação do percurso descrito pelo robô em função dos parâmetros da trajetória e das alterações realizadas ao *package* `trajectory_planner`. Como não é garantido o cumprimento total das trajetórias testadas, é difícil avaliar a qualidade da escolha realizada pelo algoritmo de navegação local. Assim, nas simulações, e priorizando a segurança, o critério encontrado avalia o espaço livre em redor do robô.

O espaço livre é determinado de forma idêntica à DLO, descrita no capítulo 4. Cada vez que o planeamento é realizado é determinada a distância linear a que o obstáculo mais próximo se encontra da posição atual robô, `trajectory->score.EVAL` (excerto de código 5.4), e publicada no tópico ROS, `/min_dist_to_obstacles`. Durante o decurso das simulações este tópico é armazenado num *bag* para que os dados do percurso completo possam ser tratados e possa ser realizada a análise estatística dos mesmos. Desta forma, e tendo conhecimento da largura da pista, é possível diferenciar a navegação com diferentes configurações dos parâmetros das trajetórias e selecionar a combinação mais segura.

Excerto de código 5.4: Cálculo da distância do obstáculo mais próximo ao modelo durante a avaliação da DLO.

```
t_func_output c_manage_trajectory::compute_DLO(c_trajectoryPtr& trajectory, std::
    vector<t_obstacle>& vo)
{
       (..)
       // testa todos os nós até ao mais próximo
       for (int n = 0; n <= trajectory->closest_node; ++n)
       {
              (...)
              // testa todas as linhas do retângulo
              for (size_t l = 0; l < trajectory->v_lines[n].size(); ++l)
              {
                     // testa todos os obstáculos
                     for (size_t o = 0; o < vo.size(); ++o)
                     {
                            // testa todos os pontos do obstáculo
                            for (size_t lo = 0; lo < vo[o].x.size(); ++lo)
                            {
                                   double DLOprev = sqrt(pow(trajectory->v_lines[n
                                       ][l].x[0] - vo[o].x[lo], 2) + pow(trajectory
                                       ->v_lines[n][l].y[0] - vo[o].y[lo], 2));
                                   (...)
                                   // avaliação da simulação
                                   if (n==0 && trajectory->score.EVAL>DLOprev)
                                   {
                                          trajectory->score.EVAL = DLOprev;
                                   }
                                   (...)
```

```
                        }
                    }
                }
            }
            (...)
}
```

5.5 Resultado final

Com o modelo do robô já integrado no *Gazebo* e a aplicação de simulação concluída, o ambiente de simulação está completo. O resultado desta junção encontra-se na figura 5.6. Aqui é possível visualizar os três grandes blocos da simulação, o modelo do ATLASCAR2, o modelo do sensor e o ambiente de testes. É de destacar a representação dos feixes do LIDAR, simulados a azul mais saturado quando é detetado um objeto e a azul menos saturado quando não é intersetado nenhum obstáculo dentro do alcance do sensor. Os muros que limitam a estrada são identificados pelo sensor como obstáculos, permitindo assim que o algoritmo de navegação guie o modelo sempre dentro dos limites da faixa de rodagem.

Quando se pretende que a navegação seja realizada na via da direta, a informação da linha central é tratada e injetada no algoritmo de navegação local.

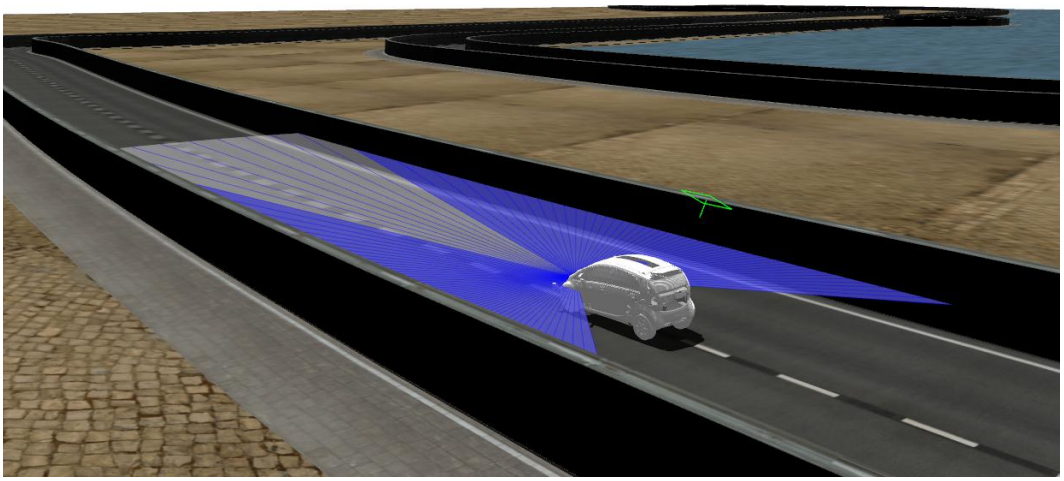


Figura 5.6: Ambiente de simulação simples final visualizado através da interface gráfica do *Gazebo* e com todos os elementos da simulação ativos.

Durante a execução do ambiente de simulação estão ativos os nodos e tópicos ROS presentes no esquema da figura 5.7. Facilmente se destaca a publicação do tópico `/simulator_laser_scan`, pelo *Gazebo*, com os dados do sensor LIDAR, correspondentes aos limites da estrada, que são subscritos e tratados pelo nodo `cirkit_unit03_gazebo`. O tópico `/line_laser_scan` contém a informação da linha central, que é convertida em dados legíveis pelo algoritmo pelo nodo `cirkit_unit03_gazebo_line`. Também se encontram em execução os nodos responsáveis pelas TFs entre os diferentes referenciais, `tf_generator4`, `tf_generator5` e `tf_generator6`, e o controlador da direção do modelo, através da subscrição do tópico `/steer_drive_controller`.

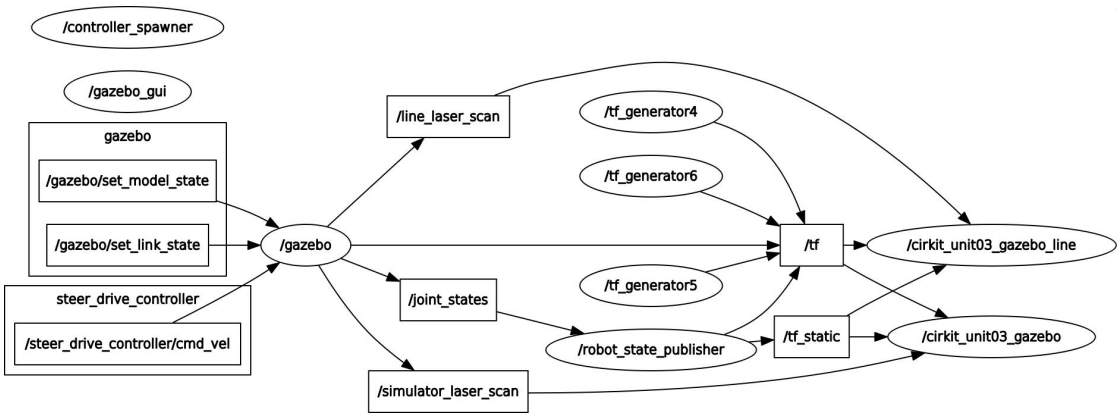


Figura 5.7: Diagrama dos nodos e tópicos executados pelo simulador.

Na subsecção 5.5.1 é descrita a simulação geral utilizada para a realização de testes comparativos. Na subsecção 5.5.2 é apresentada a situação de simulação em que se considera a linha central para efeitos de planeamento.

5.5.1 Simulação simples

As simulações designadas de "simples"são simulações onde o modelo do robô descreve uma volta completa à pista sem a presença de nenhum elemento que cause perturbação à navegação. Nestas simulações a única condicionante ao movimento do modelo são as paredes laterais da pista, que equivalem aos limites da faixa de rodagem. O percurso tem início e fim na marca verde da figura 5.8 e pode ser realizado a várias velocidades e com trajetórias de planeamento estáticas ou dinâmicas.

Figura 5.8: Ambiente de simulação simples onde a simulação é iniciada e terminada na marca verde da pista.

Durante a simulação, o funcionamento do algoritmo pode ser acompanhado e controlado visualmente através da representação da nuvem de pontos, obtida pelo sensor, e dos marcadores visuais no *Rviz* (figura 5.9). Os marcadores visuais permitem a visuali-

zação das pontuações das trajetórias e das sucessivas posições que o robô ocuparia, caso executasse por completo a trajetória escolhida.

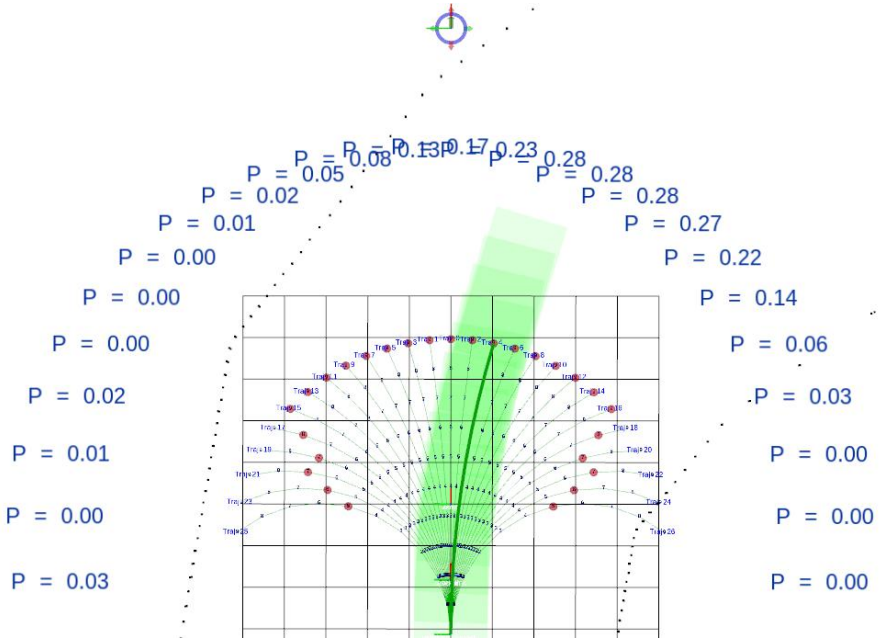


Figura 5.9: Exemplo de planeamento e pontuação de trajetórias em ambiente de simulação simples.

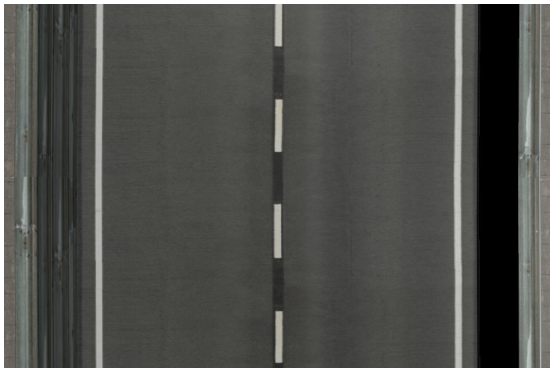
Quando lançados em simultâneo, o simulador e o planeador de trajetórias executam os nodos e os tópicos presentes no diagrama da figura 5.12. Em simulação simples, os dois tópicos que unem estes dois *packages* são o `/reduced_pcl`, onde é publicada a nuvem de pontos adquirida pelo sensor, e o `/cmd_vel`, onde são publicadas as informações de velocidade e direção determinadas pelo algoritmo, posteriormente interpretadas pelos controladores da direção.

5.5.2 Simulação com linhas

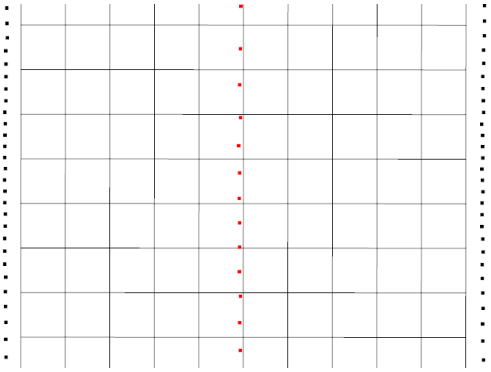
Para testar o comportamento do algoritmo de planeamento perante outro tipo de obstáculos que não físicos, foram realizadas simulações onde a linha central foi considerada como uma barreira, com maior ou menor transponibilidade consoante o peso do parâmetro CL. Como ainda não estão disponíveis dados de visão no ATLASCAR2, que permitam a identificação das linhas, e o tratamento de dados visuais vai para além do âmbito desta dissertação, o posicionamento das linhas no simulador é obtido através de dados laser que são convertidos numa nuvem de pontos para tratamento no algoritmo.

A nuvem de pontos da linha central (figura 5.10b), é obtida através da criação de uma parede virtual situada no centro da pista da simulação. Um sensor LIDAR, semelhante ao utilizado para a deteção das paredes laterais, é utilizado para recolher dados exclusivamente da parede externa da pista. Esses dados laser são convertidos para uma nuvem de pontos que é translacionada em tempo real para o centro da faixa de rodagem, isto é, para a posição da linha central.

A agregação da linha central ao planeamento de trajetórias pode ser encontrada na



(a) Linha central e paredes laterais visualizadas no simulador.



(b) Nuvens de pontos representativas da linha central (a vermelho), e paredes laterais (a preto).

Figura 5.10: Diferentes representações da linha central e das paredes da pista.

figura 5.11, onde esta se encontra representada pelos pontos da nuvem a vermelho. É claramente visível a diferença no planeamento com e sem a linha. Agora, o algoritmo não seleciona a trajetória que conduz à posição de maior espaço livre (segunda trajetória à esquerda), mas sim aquela que guia o modelo para o local mais afastado dos obstáculos dentro da via da direita (primeira trajetória à esquerda).

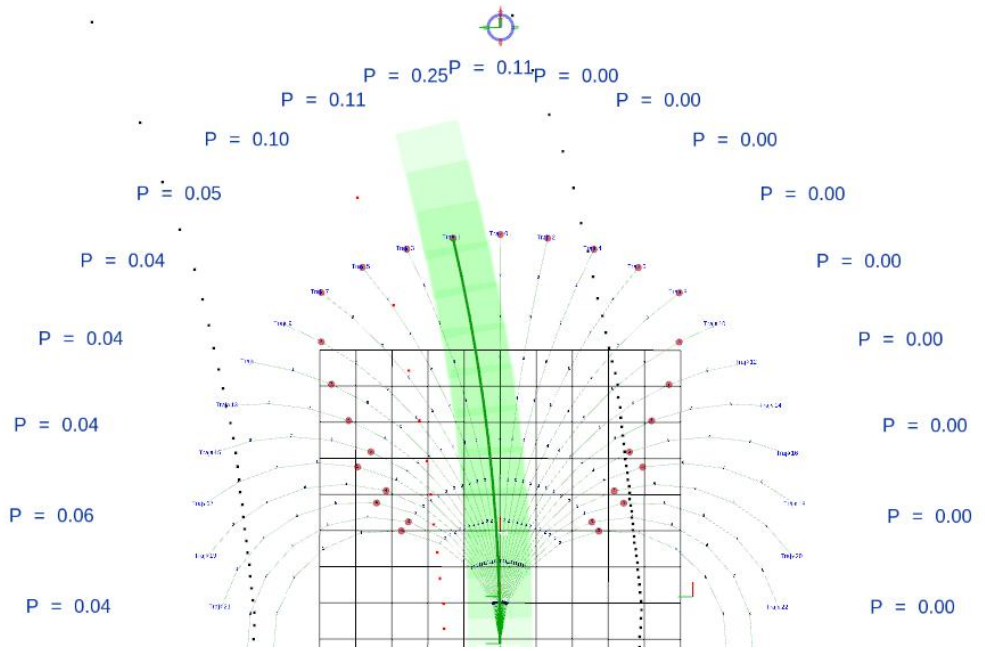


Figura 5.11: Exemplo de planeamento e pontuação de trajetórias em ambiente de simulação com linhas.

Nestas simulações, para além dos tópicos `/reduced_pcl` e `/cmd_vel`, o simulador e *package* de planeamento local comunicam ainda através do tópico `/line_pcl`, onde é feita a publicação da nuvem de pontos representativa da linha central (diagrama da figura 5.12).

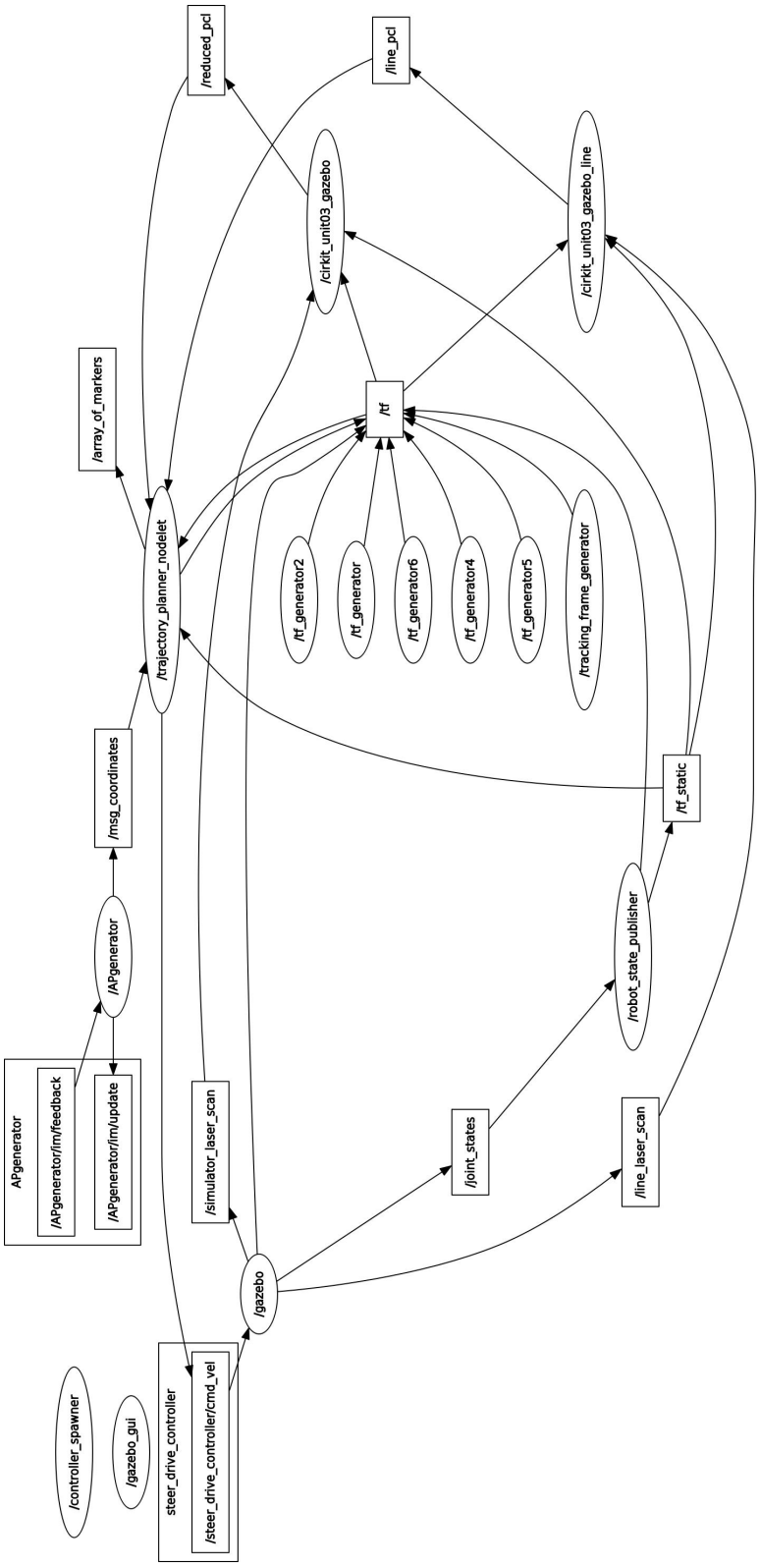


Figura 5.12: Diagrama dos nodos e tópicos executados em simultâneo pelo simulador e pelo planeador de trajetórias.

Capítulo 6

Resultados

O objetivo deste capítulo é a apresentação e discussão dos resultados obtidos nas simulações realizadas e em ambiente real. Na secção 6.1 estão descritos os primeiros testes realizados para verificar a influência dos parâmetros das trajetórias no comportamento do algoritmo. Depois, foi testada a influência do comprimento das trajetórias na qualidade do planeamento (secção 6.2). A secção 6.3 contém os resultados das simulações depois de modificações no algoritmo que permitiram a geração dinâmica de trajetórias e na secção 6.4 aparecem as simulações com introdução da linha central. Numa fase final foi implementado de um filtro de média à direção, e os resultados estão expressos na secção 6.5. Foram ainda realizados testes em ambiente real para identificar algumas das possíveis falhas que o algoritmo pode apresentar (secção 6.6).

6.1 Simulações padrão

A primeira série de testes realizada no simulador teve como função a avaliação da influência que a variação dos parâmetros Distância ao Ponto Atrator (DAP), Diferença Angular ao Ponto Atrator (ADAP) e Distância Mínima aos Obstáculos (DLO) exerce no comportamento do algoritmo de planeamento local. Durante as simulações, devido à ausência de *way points* fidedignos, o ponto atrator foi colocado 13 m à frente do veículo e com a orientação do mesmo, o que se traduz num objetivo de andar em frente quando não existem obstáculos nessa direção. Assim foram realizadas três simulações onde os parâmetros DAP e DLO variaram entre os valores {0,1; 0,5; 0,9} e {0,9; 0,5; 0,1}, respetivamente. O parâmetro ADAP foi mantido nulo durante as simulações pois apenas iria dar mais peso ao ponto atrator.

A velocidade definida para a realização destas simulações foi de 5 m/s, sendo que, nesta fase, o algoritmo ainda executava o planeamento com trajetórias estáticas, isto é, sem variação do comprimento em função da velocidade, o que se traduz em trajetórias de 3,24 m de comprimento. O ponto atrator também foi definido a uma distância fixa, mas abaixo do limiar de normalização definido na secção 4.3, de forma a influenciar a escolha da trajetória. Sempre com o objetivo de uma navegação segura em mente, foi realizada a análise estatística — histograma e distribuição normal (equação 6.1) — da distância mínima do obstáculo mais próximo ao modelo durante as simulações, neste caso as paredes da pista. Esta análise pode ser encontrada nos gráficos das figuras 6.1, 6.2 e 6.3.

$$f(x) = \frac{1}{\sigma \cdot \sqrt{2 \cdot \pi}} \cdot e^{-\frac{(x-\mu)^2}{2 \cdot \sigma^2}} \tag{6.1}$$

Tendo em conta que a distância mínima a que o carro se encontra das paredes no início da simulação, numa zona reta, é de 4,45 m pode-se verificar que os valores médios obtidos durante as simulações se aproximam deste valor à medida que se aumenta o peso da distância as obstáculos, DLO, como seria de esperar (figuras 6.1 e 6.2). No entanto, esta aproximação ocorre a um ritmo elevado pois quando se igualam os pesos dos parâmetros DAP e DLO a média da distância mínima aos obstáculos já é de 4,37 m, o que significa que o modelo se mantém a maior parte do tempo numa posição muito próxima do centro da pista. Quando utilizados parâmetros de 0,1 e 0,9 para DAP e DLO, respetivamente, o modelo nunca de aproxima mais de 3,83 m das paredes durante toda a simulação, mesmo em zonas de curva.

Quando se faz pesar mais o ponto atrator (figura 6.3), os resultados do planeamento deixam de ser positivos. Como o objetivo é andar em frente, o modelo só muda a direção quando está na iminência de colidir com as paredes. Isto resulta numa simulação onde o veículo se desloca em toda a largura da estrada simulada, encostando-se a um dos lados da estrada assim que sai de uma curva. A média da distância às paredes, para valores dos parâmetros DAP e DLO de 0,9 e 0,1, respetivamente, reduz para os 2,26 m com um desvio padrão de 1,70 m, que é representativo dos movimentos ao longo de toda a largura da pista. Um fator alarmante é que, neste caso, o modelo chega a estar a apenas 0,01 m de colidir com as paredes.

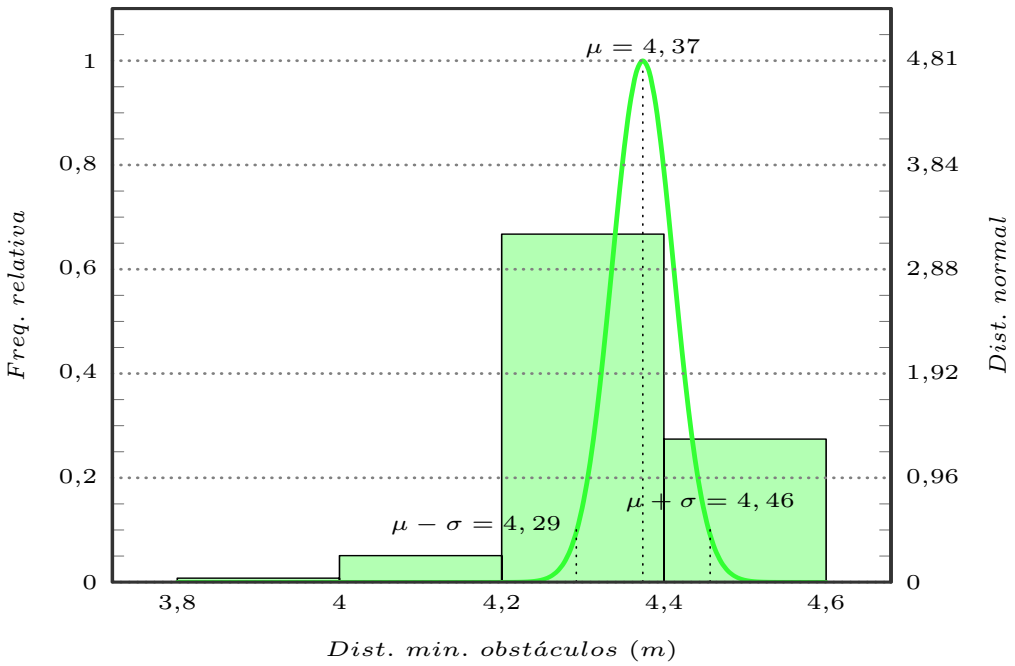


Figura 6.1: Histograma e distribuição normal da distância mínima aos obstáculos para velocidade de 5 m/s, DAP=0,1, ADAP=0 e DLO=0,9. Valores estatísticos: média(μ)=4,37, desvio padrão(σ)=0,08 e mínimo=3,83 m.

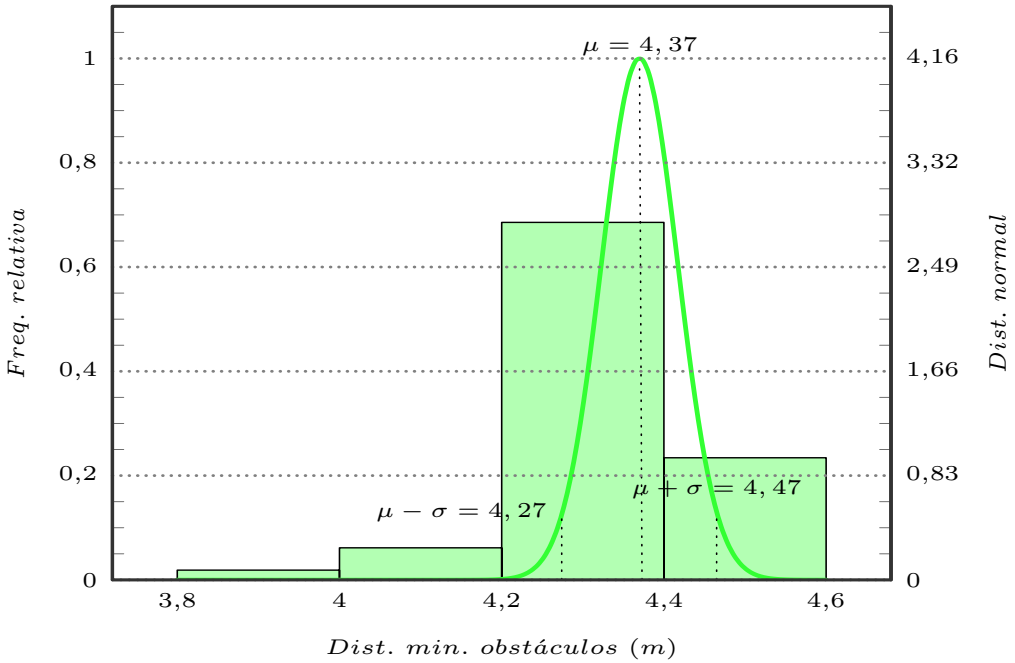


Figura 6.2: Histograma e distribuição normal da distância mínima aos obstáculos para velocidade de 5 m/s, DAP=0,5, ADAP=0 e DLO=0,5. Valores estatísticos: média(μ)=4,37, desvio padrão(σ)=0,10 e mínimo=3,82 m.

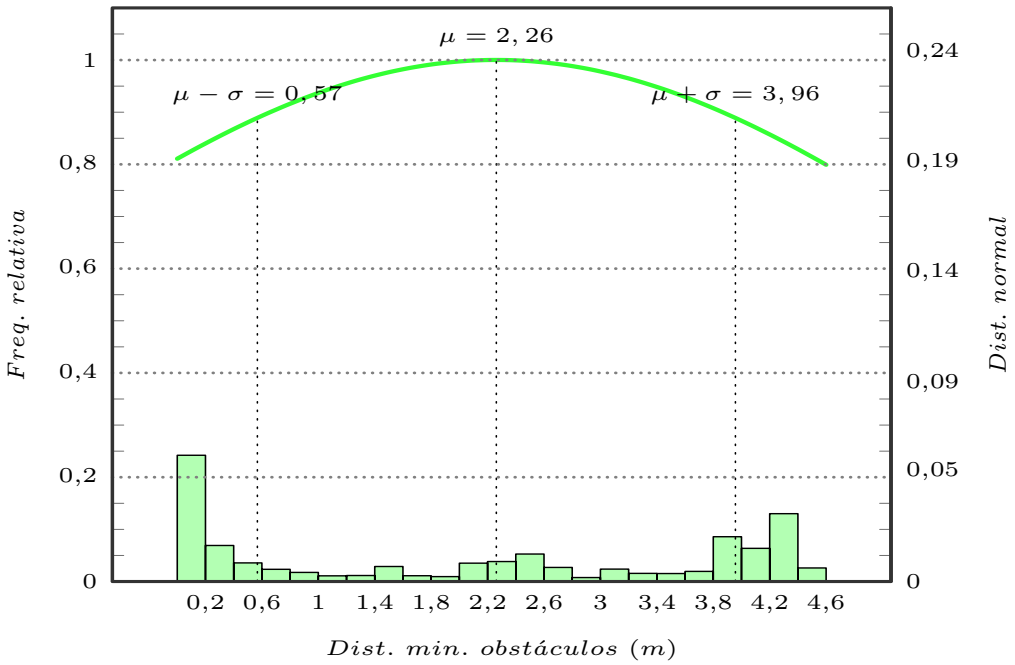


Figura 6.3: Histograma e distribuição normal da distância mínima aos obstáculos para velocidade de 5 m/s, DAP=0,9, ADAP=0 e DLO=0,1. Valores estatísticos: média(μ)=2,26, desvio padrão(σ)=1,70 e mínimo=0,01 m.

6.2 Simulações com trajetórias de comprimento aumentado

Outro conjunto de testes realizado consistiu na alteração do comprimento das trajetórias para tentar aferir a sua influência na qualidade do planeamento. Para poderem ser estabelecidos paralelismos entre estas simulações e as apresentadas na secção 6.1, os parâmetros DAP e DLO variaram entre os valores {0,1; 0,5; 0,9} e {0,9; 0,5; 0,1}, respetivamente, e o parâmetro ADAP foi mantido nulo. A velocidade também foi mantida constante e igual a 5 m/s e o ponto atrator definido da mesma posição.

O comprimento das trajetórias foi determinado para uma velocidade de deslocamento de 10 m/s, isto é, o dobro da utilizada nos testes apresentados na secção 6.1, o que se traduz em trajetórias com 12,96 m de comprimento. Apesar dos cálculos internos serem realizados considerando a velocidade de 10 m/s, o algoritmo foi curto-circuitado de forma a continuar a enviar mensagens de velocidade de 5 m/s para o simulador, permitindo avaliar a regra utilizada para calcular o comprimento das trajetórias. Os resultados das três simulações encontram-se simplificados nos gráficos das figuras 6.4, 6.5 e 6.6.

Mantendo a tendência das simulações padrão, quanto maior o peso do parâmetro DLO, melhores são os resultados do planeamento, tendo o valor médio da distância ao obstáculo mais próximo ficado nos 3,77 m para valores de 0,1 e 0,9 nos parâmetros DAP e DLO, respetivamente (figura 6.4). Apesar deste facto, o valor médio ficou aquém do registado nas simulações padrão (tabela 6.1). A explicação para a diminuição do desempenho reside em dois fatores: a antecipação do planeamento em curva e a maior distância entre os pontos finais das trajetórias. Como o comprimento das trajetórias é superior, e a influência do ponto atrator é baixa, o algoritmo começa a planear a curva mais cedo e o planeamento aproxima-se mais do limite interno da curva, diminuindo a distância à parede interior. Por outro lado, nas saídas das curvas e em curvas de raio elevado, como o comprimento do arco das trajetórias é superior às simulações padrão e o ângulo entre elas se mantém, os pontos finais estão mais espaçados. Isto requer uma aproximação superior a um obstáculo para de seja possível o algoritmo selecionar outra trajetória, mantendo assim o modelo com mesmo sentido, durante mais tempo.

Tabela 6.1: Resultados estatísticos das simulações padrão e com trajetórias aumentadas.

Simulação	DAP/DLO	Média(μ) (m)	Desvio padrão(σ) (m)	Mínimo (m)
Padrão	0,1/0,9	4,37	0,08	3,83
	0,5/0,5	4,37	0,10	3,82
	0,9/0,1	2,26	1,70	0,01
Traj. aumentada	0,1/0,9	3,77	0,30	2,81
	0,5/0,5	2,11	1,26	0,01
	0,9/0,1	0,58	1,23	0,01

Conforme se vai reduzindo a influência da DLO e se vai aumentando a da DAP (gráficos das figuras 6.5 e 6.6), o planeamento começa a ser realizado cada vez mais junto aos limites da pista dado que o peso da distância ao ponto atrator é superior ao da distância aos obstáculos e as trajetórias terminam mais junto do ponto atrator, então o modelo só muda de direção quando está em risco de colisão. Tal como nas simulações padrão, o modelo chega a estar a menos de 0,01 m de um obstáculo, mas a distância média cai para 0,58 m pois, nas saídas nas curvas, o modelo mantém a direção do ponto atrator, mesmo que isto implique proximidade aos obstáculos.

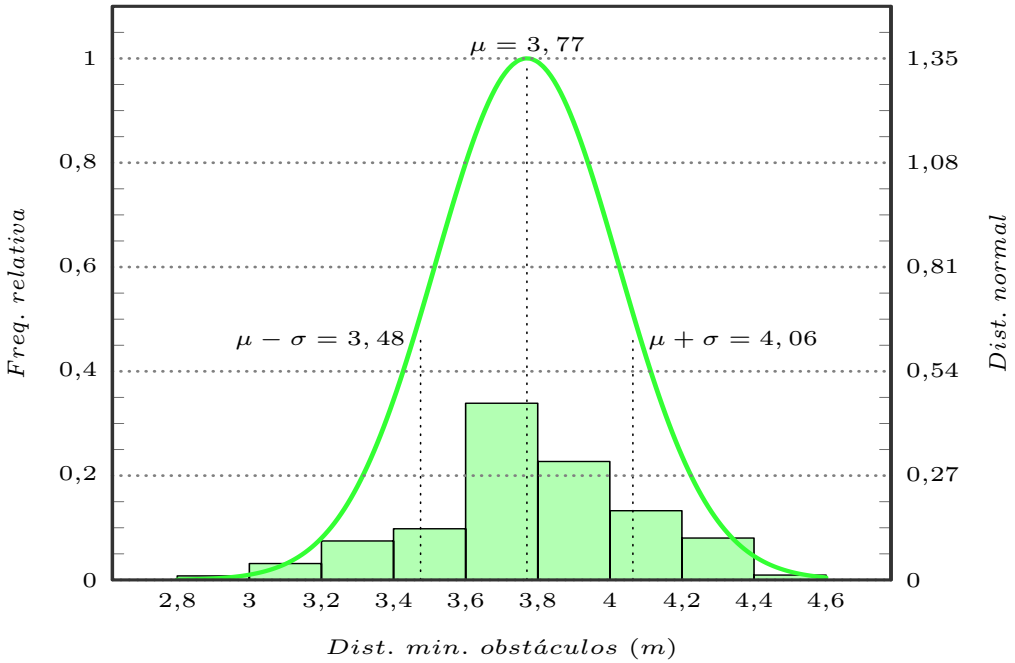


Figura 6.4: Histograma e distribuição normal da distância mínima aos obstáculos para velocidade de 5 m/s e trajetórias aumentadas, DAP=0,1, ADAP=0 e DLO=0,9. Valores estatísticos: média(μ)=3,77, desvio padrão(σ)=0,30 e mínimo=2,81 m.

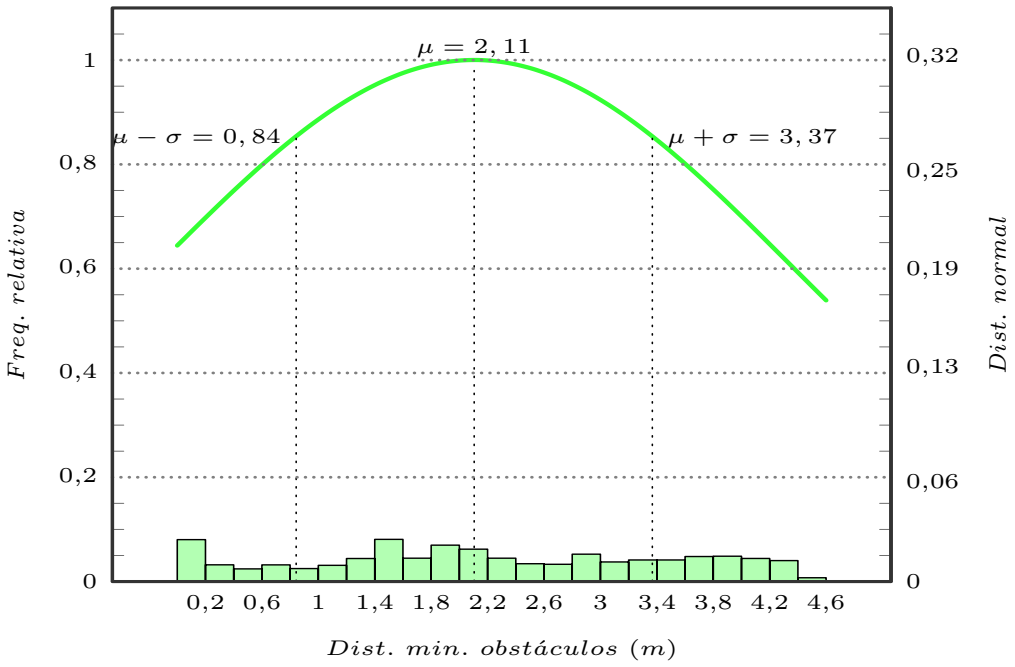


Figura 6.5: Histograma e distribuição normal da distância mínima aos obstáculos para velocidade de 5 m/s e trajetórias aumentadas, DAP=0,5, ADAP=0 e DLO=0,5. Valores estatísticos: média(μ)=2,11, desvio padrão(σ)=1,26 e mínimo=0,01 m.

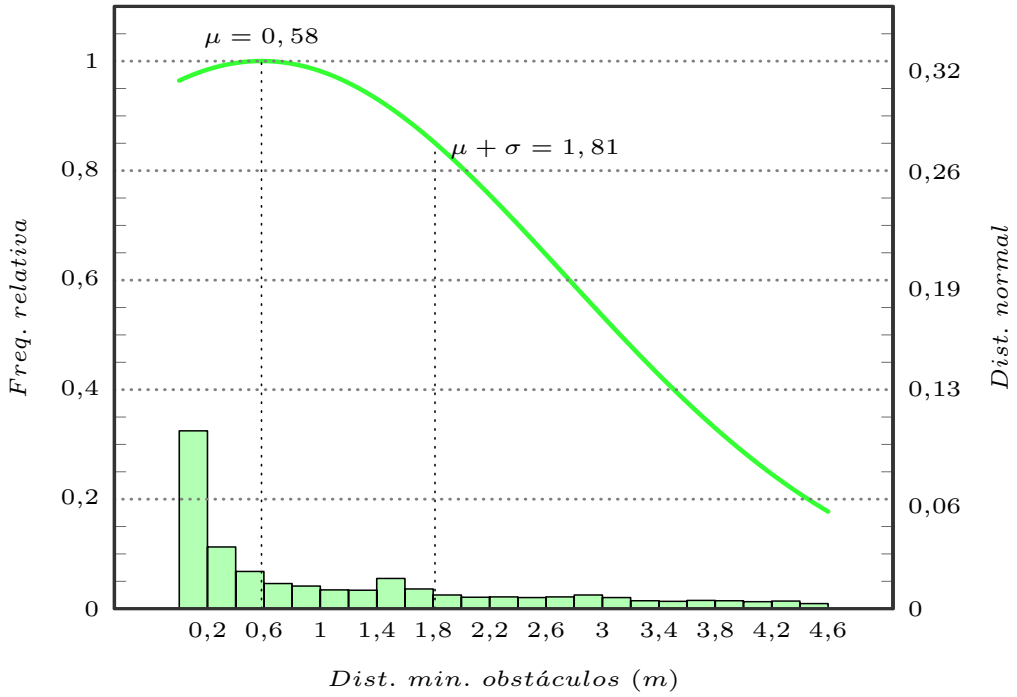


Figura 6.6: Histograma e distribuição normal da distância mínima aos obstáculos para velocidade de 5 m/s e trajetórias aumentadas, DAP=0,9, ADAP=0 e DLO=0,1. Valores estatísticos: média(μ)=0,58, desvio padrão(σ)=1,23 e mínimo=0,01 m.

6.3 Simulações com trajetórias de comprimento variável

Depois de implementar a variação dinâmica das trajetórias com a velocidade, sendo esta dada em função do ângulo de direção selecionado pelo algoritmo, voltaram a realizar-se mais simulações de forma a garantir que o planeamento não levava a colisões e a avaliar a qualidade do mesmo. Mais uma vez, foram retirados resultados para os parâmetros DAP e DLO a variarem entre os valores {0,1; 0,5; 0,9} e {0,9; 0,5; 0,1}, respetivamente, e o parâmetro ADAP nulo. A definição do ponto atrator foi realizada de igual forma às simulações descritas nas secções 6.1 e 6.2.

No que respeita à velocidade, para a realização destas simulações, foram escolhidos os limiares máximo e mínimo de 10 m/s e 1 m/s. Desta forma, o comprimento das trajetórias varia entre 12,96 m e 4,5 m pois foi estabelecido um comprimento mínimo para o planeamento de forma a evitar colisões com obstáculos inesperados (secção 4.1). O tratamento estatístico dos resultados obtidos para a distância mínima a que o modelo se encontra dos obstáculos, durante as três simulações, pode ser encontrado nos gráficos das figuras 6.7, 6.8 e 6.9.

Seguindo o padrão verificado nas simulações anteriores, quanto maior for a influência do parâmetro DLO na equação da pontuação geral da trajetória, melhores são os resultados do planeamento. Com os parâmetros DAP e DLO definidos em 0,1 e 0,9, respetivamente, o valor médio da distância ao obstáculo mais próximo situa-se nos 3,81 m e o desvio padrão nos 0,24 m, (figura 6.7). Durante a volta completa, o modelo nunca se aproximou mais do que 2,74 m das paredes da pista.

À medida que o peso do ponto atrator foi aumentando (gráficos das figuras 6.8 e 6.9), o planeamento foi-se aproximando dos obstáculos, caracterizando-se por uma média de 0,70 m para os parâmetros DAP e DLO a valerem 0,9 e 0,1, respetivamente. Neste caso, o planeamento foi realizado a menos de 0,20 m dos limites da pista em mais de vinte por cento do tempo.

Como estas simulações são dinâmicas, não podem ser estabelecidas comparações diretas com simulações realizadas a velocidade constante pois não foi estudado o comportamento do planeamento com a evolução da velocidade. No entanto, os resultados apresentam um perfil aproximado aos obtidos nas simulações com trajetórias de comprimento aumentado. Devido às baixas velocidades de deslocamento das simulações realizadas, sempre que o algoritmo determina que a direção a seguir implica virar mais de 18,94° para um dos lados está-se a planear com trajetórias aumentadas. Nestes casos, a velocidade selecionada pelo algoritmo é inferior a 5,89 m/s o que se traduz em trajetórias com comprimento inferior a 4,5 m. Como o limite mínimo para o comprimento das trajetórias é ultrapassado, as trajetórias são estendidas, o que influencia o comportamento do planeamento. As trajetórias dinâmicas apresentam vantagens em situações de mudanças de direção acentuadas pois, ao reduzirem a velocidade, permitem que o modelo descreva a trajetória com uma mudança de direção mais confortável e para o mesmo espaço percorrido o planeamento é realizado mais vezes.

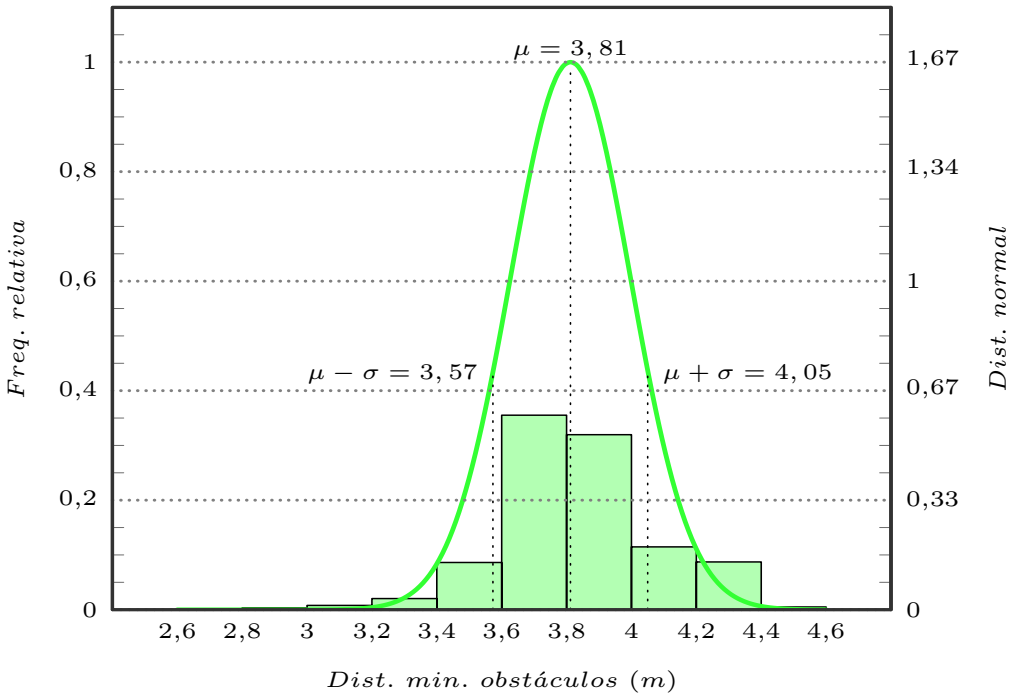


Figura 6.7: Histograma e distribuição normal da distância mínima aos obstáculos para trajetórias dinâmicas com velocidades entre 1 m/s e 10m/s, DAP=0,1, ADAP=0 e DLO =0,9. Valores estatísticos: média(μ)=3,81, desvio padrão(σ)=0,24 e mínimo=2,74 m.

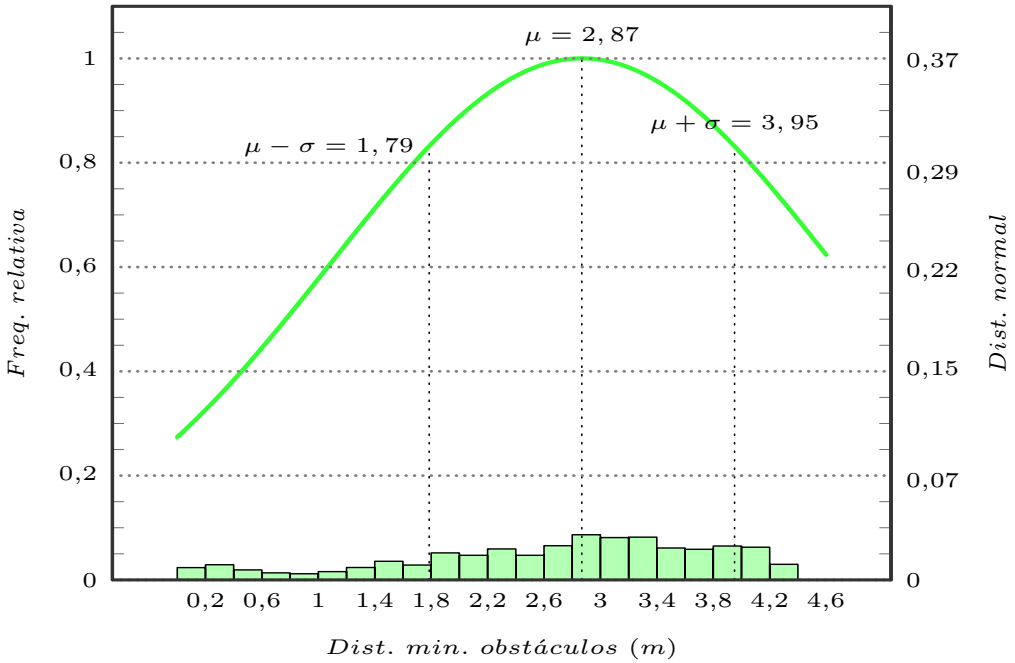


Figura 6.8: Histograma e distribuição normal da distância mínima aos obstáculos para trajetórias dinâmicas com velocidades entre 1 m/s e 10m/s, DAP=0,5, ADAP=0 e DLO =0,5. Valores estatísticos: média(μ)=2,87, desvio padrão(σ)=1,08 e mínimo=0,07 m.

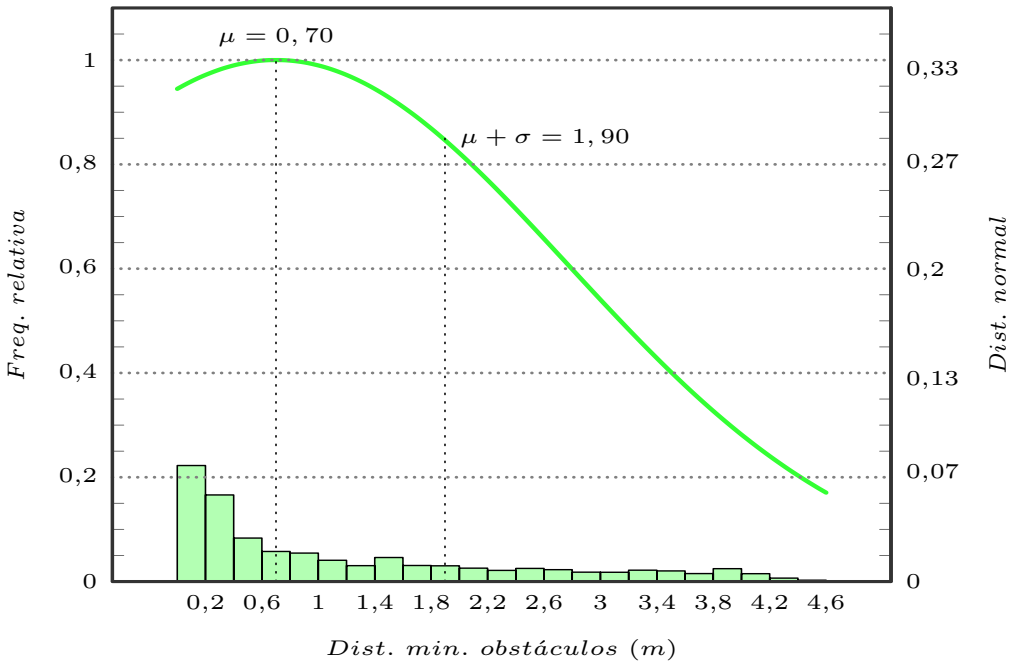


Figura 6.9: Histograma e distribuição normal da distância mínima aos obstáculos para trajetórias dinâmicas com velocidades entre 1 m/s e 10m/s, DAP=0,9, ADAP=0 e DLO =0,1. Valores estatísticos: média(μ)=0,70, desvio padrão(σ)=1,20 e mínimo=0,01 m.

6.4 Simulações considerando a linha central

De forma a avaliar a influência da linha central no algoritmo de planeamento foram realizadas simulações para aferir o comportamento do algoritmo com a variação do parâmetro Linha Central (CL). Para a realização destes testes foram escolhidos os parâmetros mais satisfatórios do conjunto de simulações apresentadas na secção 6.3. Assim, foi mantida a variação dinâmica das trajetórias entre as velocidades de 10 m/s e 1 m/s e fixados os parâmetros DAP, ADAP e DLO nos valores respetivos, {0,1; 0,0; 0,9}.

Para analisar as variações no planeamento, foram atribuídos valores de 0,8 e 0,2 ao parâmetro CL (gráficos das figuras 6.10 e 6.11). A análise destas simulações deve estar focada não só nos resultados estatísticos mas também na posição do modelo ao longo da simulação. Para que o planeamento seja realizado com sucesso, o modelo simulado deve circular o máximo de tempo possível dentro da via da direita. Considera-se que o modelo se encontra do lado direito da linha central se a distância ao obstáculo mais próximo for inferior a 3,57 m.

Tal como expectável, quanto mais baixo for o parâmetro CL mais próximo do limite direito da pista se realiza o planeamento pois as trajetórias com direção à esquerda, e que intercetam a linha central, possuem uma pontuação inferior. Com CL a valer 0,8, em 31,6% do tempo o modelo está na via da direita. Já para CL a valer 0,2 esta percentagem aumenta para 87,7%, demonstrando claramente este comportamento padrão.

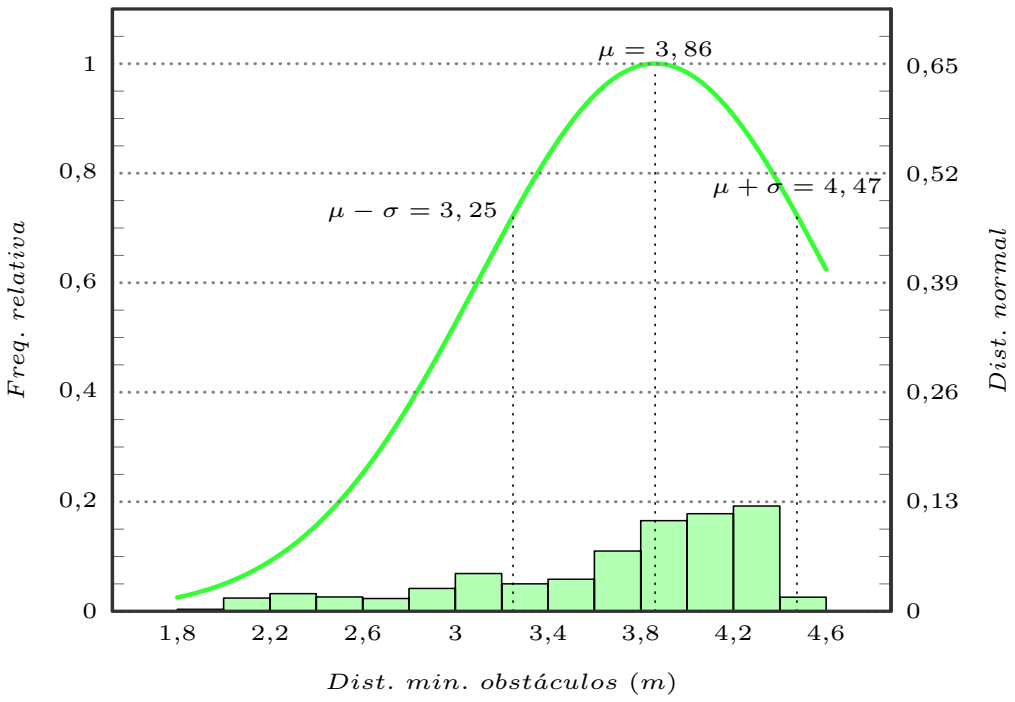


Figura 6.10: Histograma e distribuição normal da distância mínima aos obstáculos para trajetórias dinâmicas com velocidades variáveis de 1 m/s a 10m/s, considerando a linha central, DAP=0,1, ADAP=0, DLO=0,9 e CL=0,8. Valores estatísticos: média(μ)=3,86, desvio padrão(σ)=0,61 e mínimo=1,94 m.

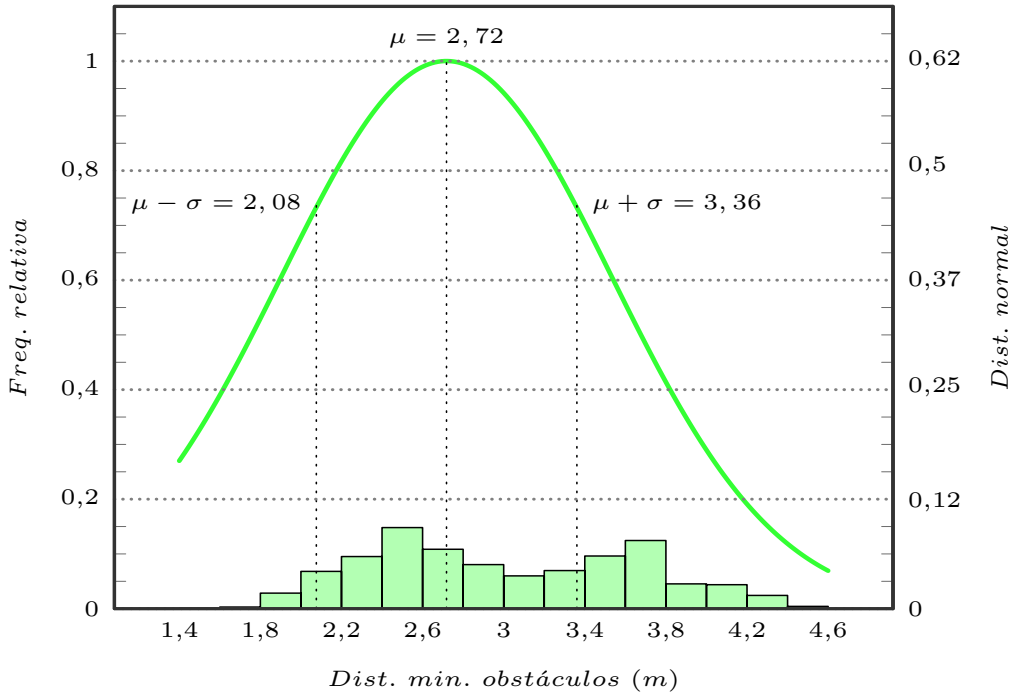


Figura 6.11: Histograma e distribuição normal da distância mínima aos obstáculos para trajetórias dinâmicas com velocidades variáveis de 1 m/s a 10m/s, considerando a linha central, DAP=0,1, ADAP=0, DLO=0,9 e CL=0,2. Valores estatísticos: média(μ)=2,72, desvio padrão(σ)=0,64 e mínimo=1,33 m.

6.5 Simulações com filtro de média da direção

Numa tentativa de resolver o problema de os pontos finais das trajetórias serem mais distantes para trajetórias de maior comprimento, sem aumentar o número de trajetórias, e minimizar algumas oscilações do algoritmo durante o planeamento, foi criado um filtro de média para a direção a seguir. Esta deixa de ser a selecionada diretamente pelo algoritmo e passa a assumir o valor da média da direção, θ, das últimas 5 trajetórias selecionadas (equação 6.2). Este filtro foi aplicado em condições iguais à primeira e última simulações apresentadas nas secções 6.3 (para simulações com trajetórias dinâmicas), e 6.4 (para simulações com a linha central), respetivamente, de forma a poderem ser realizadas comparações. Os resultados da aplicação do filtro encontram-se nos gráficos das figuras 6.12 e 6.13.

$$\bar{\theta}(t) = \frac{\sum_{i=0}^{4} \theta(t-i)}{5} \tag{6.2}$$

Como se pode verificar, pela comparação dos gráficos das figuras 6.7 e 6.12 e pela análise da tabela 6.2, existe uma aproximação de 2,2% do modelo ao centro da estrada, com a aplicação do filtro, para simulações com geração dinâmica de trajetórias. A par deste benefício surge ainda uma melhoria no desvio padrão e na distância mínima aos obstáculos ao longo de toda a simulação. Também, através da analogia dos gráficos das figuras 6.11 e 6.13 e da consulta da tabela 6.2, se comprova que o tempo que o

modelo se desloca à direita da linha central aumenta de 87,7% para 88,9%, depois de utilizado o filtro. Com isto pode-se dizer que a aplicação de um simples filtro melhora o desempenho do planeamento sem ser necessário um acréscimo do custo computacional associado à análise de mais trajetórias. No entanto, não se deve estender o filtro a uma grande quantidade de trajetórias, que podem representar uma grande amplitude de direções, sob pena de a direção filtrada já não se encontrar livre ou o planeamento se tornar pouco reativo. Considerando as 5 últimas trajetórias, a uma taxa de atualização de cerca de 10 Hz, está-se a filtrar o último meio segundo de planeamento.

Tabela 6.2: Resumo dos resultados estatísticos das simulações com e sem filtro de direção para trajetórias dinâmicas e parâmetros DAP=0,1, ADAP=0 e DLO=0,9, e CL=0,2 para as simulações com linha central (gráficos das figuras 6.7 e 6.12, e 6.11 e 6.13).

Simulação	Trajetória dinâmica		Linha central	
Filtro	•		•	
Média(μ) (m)	3,91	3,81	2,88	2,72
Desvio padrão(σ) (m)	0,21	0,24	0,56	0,64
Mínimo (m)	3,02	2,74	1,72	1,33
Tempo dir. linha (%)	—	—	88,9	87,7

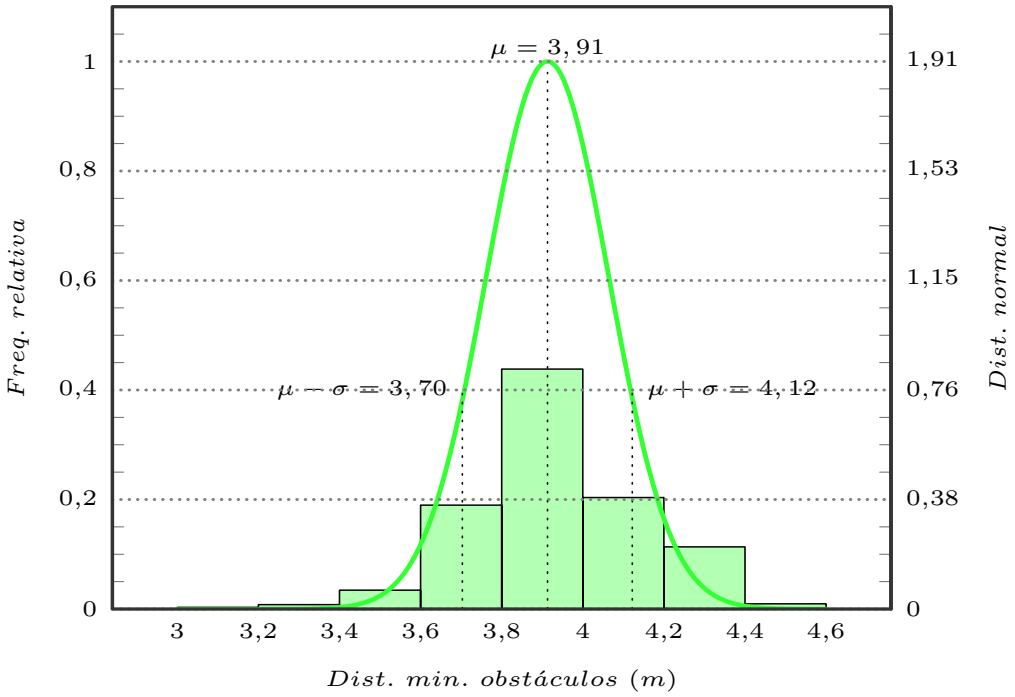


Figura 6.12: Histograma e distribuição normal da distância mínima aos obstáculos para trajetórias dinâmicas e filtro de direção com velocidades variáveis de 1 m/s a 10m/s, DAP=0,1, ADAP=0 e DLO=0,9. Valores estatísticos: média(μ)=3,91, desvio padrão(σ)=0,21 e mínimo=3,02 m.

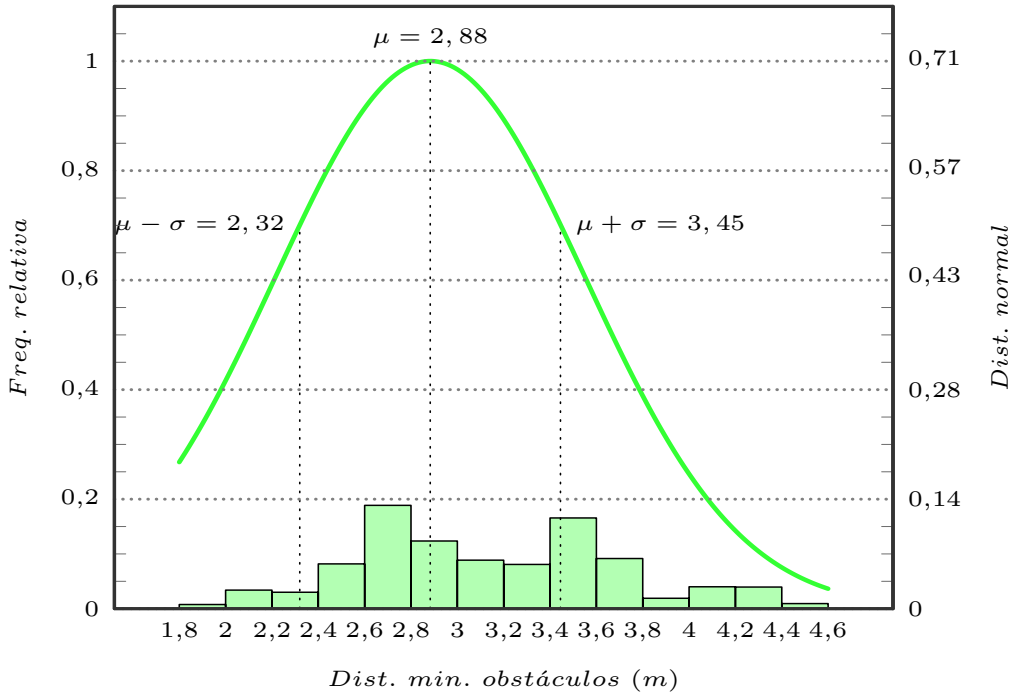


Figura 6.13: Histograma e distribuição normal da distância mínima aos obstáculos para trajetórias dinâmicas e filtro de direção com velocidades variáveis de 1 m/s a 10m/s, considerando a linha central, DAP=0,1, ADAP=0, DLO=0,9 e CL=0,2. Valores estatísticos: média(μ)=2,88, desvio padrão(σ)=0,56 e mínimo=1,72 m.

6.6 Aplicações reais

A realização de testes em ambientes reais teve que ser bem ponderada pois a informação da velocidade e direção do ATLASCAR2 não está disponível diretamente. Desta forma os testes foram realizados sabendo, *a priori*, que estas duas variáveis são, única e exclusivamente, controladas e monitorizadas pelo condutor. Isto condiciona o tipo de testes pois é muito difícil para o condutor distinguir entre rodar o volante de 0° a 18° ou de 0° a 21°, por exemplo. Outro fato importante é a impossibilidade de deteção das linhas da estrada por parte do ATLASCAR2, fazendo com que o espaço livre para a navegação seja dado apenas pelos obstáculos físicos, identificados pelos sensores LIDAR.

Tendo em conta estas limitações, os testes definidos possuem obstáculos físicos que limitam a estrada o mais junto às linhas possível e foram realizados com velocidade e direção constantes de forma a identificar as hipotéticas variações da direção determinadas pelo algoritmo de navegação local. Estes testes foram realizados com os parâmetros de avaliação das trajetórias DAP, ADAP, DLO e CL definidos como 0,1, 0, 0,9 e 1, respetivamente. A influência do ponto atrator foi mantida baixa porque nem sempre os *way points* fornecidos estão posicionados corretamente na estrada e o *way point* virtual — a amarelo nas figuras — está sempre dentro do alcance do planeamento. Já a influência das linhas da estrada é nula devido à falta de deteção das mesmas em ambiente real. Devido à contribuição positiva do filtro de média da direção, este foi mantido ativo durante estas experiências.

Para a realização do primeiro teste foi escolhido o túnel de Santa Joana, no centro de Aveiro (figura 6.14), pois possui uma única via em cada sentido e um separador central físico descontínuo. Os limites laterais também são barreiras físicas, facilmente detetáveis pelos LIDARs, e estão bastante próximos do limite da via. Como é possível percorrer todo o túnel com a direção alinhada em frente e não existe qualquer interação com outros veículos pôde-se assim aferir o número de vezes que o algoritmo selecionou uma trajetória que não a que permitia seguir em frente. No que diz respeito à velocidade, tentou-se manter uma velocidade estável de 36 km/h, o equivalente a 10 m/s.

Figura 6.14: Vista de satélite do túnel de Santa Joana, no centro de Aveiro, local escolhido para a realização do primeiro teste real (troço de recolha de dados identificado pela reta a vermelho).

Como esperado, na presença de barreiras físicas, o algoritmo indica a trajetória que guia o automóvel ao espaço livre em frente, mesmo quando o *way point* fornecido se encontra totalmente fora da via de circulação (figura 6.15). Durante os 33 segundos de circulação dentro do túnel, em 93,04% das vezes em que o planeamento foi realizado o algoritmo selecionou a trajetória central (gráfico da figura 6.16). Apesar de simples, este teste permite demonstrar a influência da localização do ponto atrator no planeamento da trajetória. Da análise do gráfico pode-se verificar que o algoritmo seleciona periodicamente trajetórias com direção à direita, isto é, que conduzem o veículo ao outro lado da estrada, onde está localizado o *way point*. Este fenómeno periódico ocorre nas descontinuidades do separador central. Sempre que existe uma interrupção no separador central o algoritmo deteta espaço livre mais próximo do ponto atrator e seleciona momentaneamente trajetórias que conduzem a esse espaço.

Aqui está identificada uma situação de navegação em que a posição errada do ponto atrator prejudica o planeamento. Neste caso, a ausência do ponto atrator dentro do limiar de normalização, apresentado no capítulo 4, traria vantagens pois estando o ponto atrator foram desse limiar o algoritmo iria selecionar a trajetória que conduziria ao maior espaço livre, sem qualquer influência do ponto atrator na seleção da mesma. Outra solução possível para tentar solucionar este problema, da parte do algoritmo, seria atribuir ainda menos peso ao ponto trator na equação de pontuação.

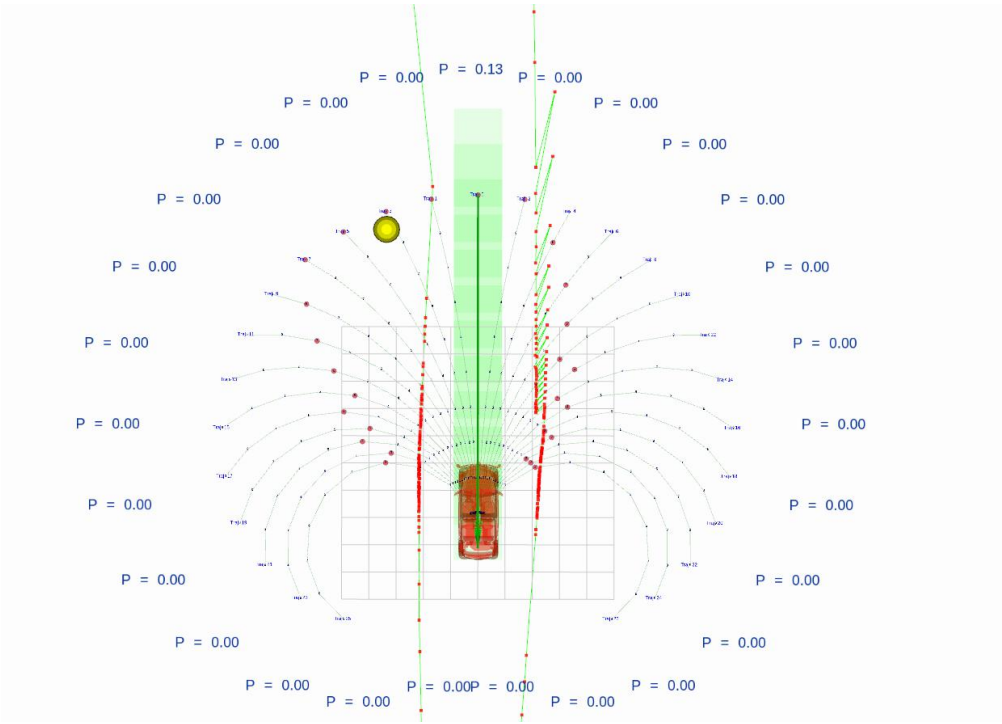


Figura 6.15: Exemplo de escolha de trajetória por parte do algoritmo de navegação local com o *way point* fora da via de circulação.

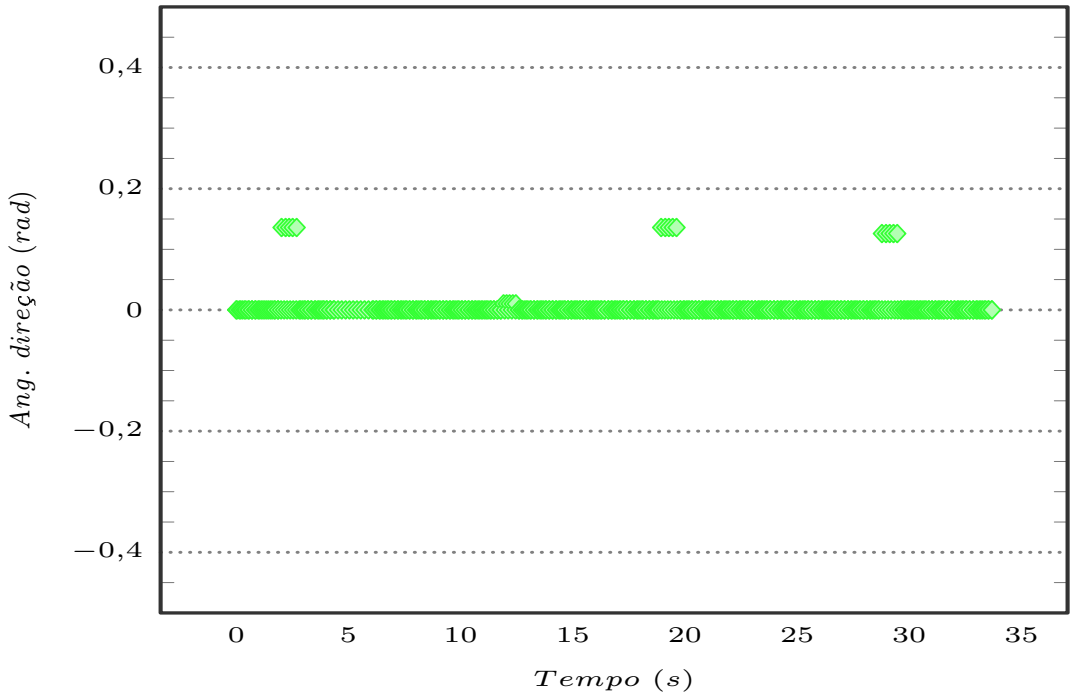


Figura 6.16: Histórico da direção do veículo calculada pelo algoritmo de navegação local durante a realização do primeiro teste real, DAP=0,1, ADAP=0, DLO=0,9 e CL=1.

Um segundo teste idealizado consistiu na circulação num troço em linha reta de autoestrada. Devido ao posicionamento dos LIDARs 2D é possível detetar os *railes* de proteção que limitam as laterais da faixa de rodagem. Como a berma possui uma largura quase idêntica a uma via, ao circular na via da direita possui-se espaço livre à direita do veículo o que permite a seleção de trajetórias em frente. Na ausência deste espaço livre o algoritmo apresentaria um comportamento tendencioso de seleção de trajetórias à esquerda devido ao espaço livre muito superior desse lado do veículo. Esta experiência pretendeu avaliar a interação do algoritmo com obstáculos dinâmicos que surgiram em situações de ultrapassagem. Sendo o troço reto, foi tentado manter um ângulo de direção nulo para, mais uma vez, avaliar as diferenças de direção fornecidas pelo algoritmo. O local identificado para a realização deste teste foi o troço da A25 situado entre o nó 2 e a área de serviço no sentido este/oeste (figura 6.17). Devido à tipologia da estrada foi mantida uma velocidade de circulação estável de 65 km/h, cerca de 18,06 m/s.

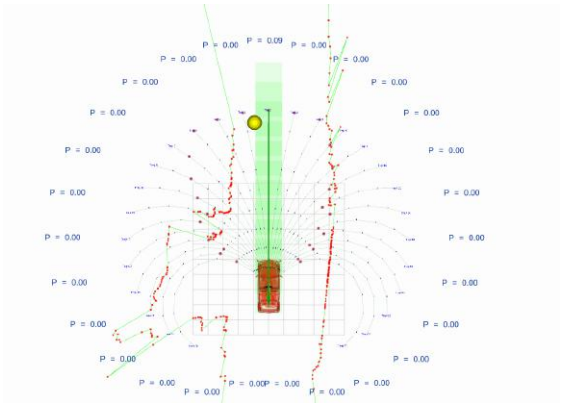
Figura 6.17: Vista de satélite do troço da A25, desde o nó 2 até à área de serviço, no sentido este/oeste, local escolhido para a realização do segundo teste real (troço de recolha de dados identificado pela reta a vermelho).

Esta experiência foi realizada com uma vantagem face à anterior, os *way points* fornecidos pela navegação global encontravam-se quase sempre na via de circulação da direita (figura 6.18), atraindo assim o planeamento para o local correto da estrada. Desta forma, 90,62% das vezes que o planeamento foi realizado o algoritmo selecionou a direção correta, seguir em frente (gráfico da figura 6.19).

Ao contrário do teste no túnel, aqui existem oscilações de direção para os dois lados devido à existência de espaço livre em ambos. As oscilações para a direita (ângulo negativo), ocorrem em situações de ultrapassarem onde a berma apresenta uma largura superior e o automóvel que ultrapassa se desloca mais próximo do ATLASCAR2. Nestes momentos, apesar do *way point* puxar o planeamento para a frente, o algoritmo deteta mais espaço livre à direita e indica que o condutor deve conduzir o veículo para esse espaço. Neste contexto, o aumento do peso do ponto atrator na equação de pontuação das trajetórias seria vantajoso.

Trajetórias com direção positiva, isto é, para a esquerda, tem origem noutro fenómeno.

(a) Mapa de navegação global com rota a seguir identificada pela linha a preto.



(b) Seleção da trajetória a seguir por parte do algoritmo de navegação local.

Figura 6.18: Exemplo de integração das navegações local e global durante a realização da segunda experiência real.

Depois da ultrapassagem quase concluída, existem veículos que se encostam à via da direita quase de imediato, ainda dentro do espaço de planeamento do algoritmo, o que invalida o seguimento de trajetórias em frente. Como o algoritmo não pode selecionar trajetórias em frente, o maior espaço livre encontra-se agora à esquerda e o ponto atrator também se localiza mais à esquerda, o planeamento é puxado para a esquerda. Em várias situações, passiveis de serem identificadas no gráfico de ângulos de direção, no início da ultrapassagem, quando o outro veículo se encontra na via da esquerda, o algoritmo tende a selecionar trajetórias mais à direita, à procura do espaço livre, na parte final, quando o veículo se encosta à esquerda, a tendência é selecionar as trajetórias à direita pois as trajetórias em frente encontram-se condicionadas.

Numa tentativa de solucionar parte deste problema poderia-se pensar em diminuir o comprimento do planeamento, isto é, das trajetórias. Esta abordagem não seria segura pois em caso de travagem do veículo da frente o ATLASCAR2 não teria espaço para se imobilizar sem colisão. Neste caso a melhor opção estaria em técnicas de identificação de obstáculos e excluir estes veículos do espaço de planeamento.

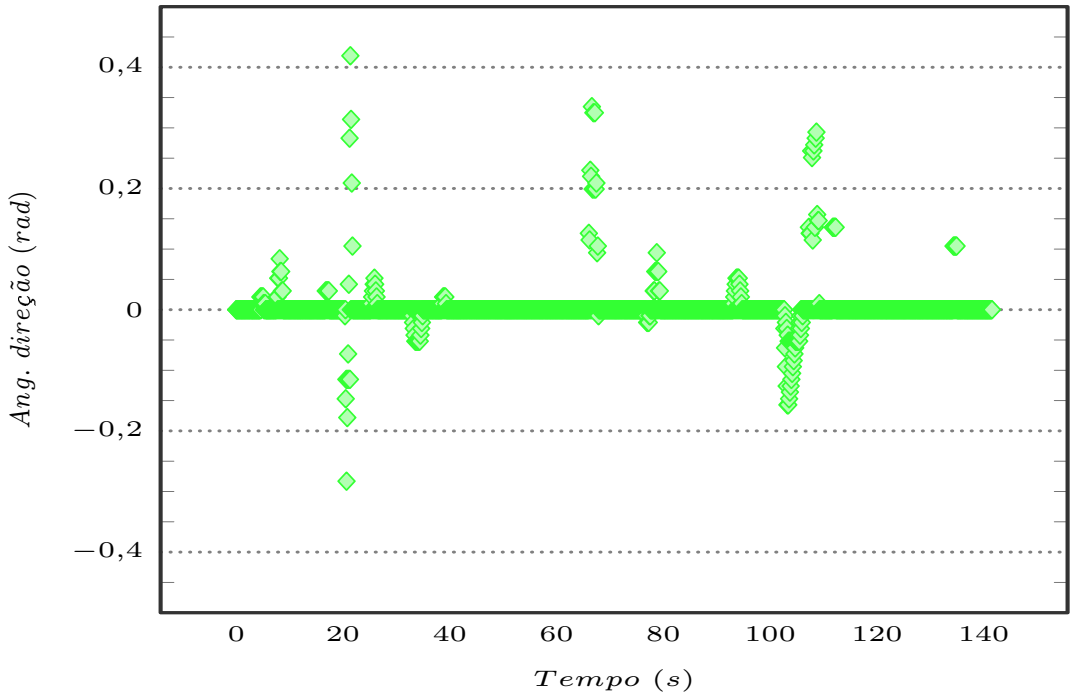


Figura 6.19: Histórico da direção do veículo calculada pelo algoritmo de navegação local durante a realização do segundo teste real, DAP=0,1, ADAP=0, DLO=0,9 e CL=1.

Capítulo 7

Conclusões e trabalhos futuros

Este capítulo final da dissertação apresenta um resumo do trabalho de investigação e de desenvolvimento de *software* realizado, juntamente com as conclusões retiradas durante o cumprimento desses mesmos trabalhos (secção 7.1). São ainda apresentadas algumas propostas de trabalhos futuros a realizar no projeto ATLASCAR2 que possibilitam quer a valorização desta dissertação, quer a valorização do próprio projeto (secção 7.2).

7.1 Contribuições e conclusões

A navegação local é um dos pilares fundamentais da AD, sendo alvo de investigação e desenvolvimento por parte de grandes empresas automóveis e tecnológicas bem com instituições de ensino. No âmbito esta dissertação foram seguidos todos os passos necessários à incorporação de um módulo de navegação local e assistência ao condutor no ATLASCAR2.

Antes de partir para a resolução de um problema é importante a contextualização com o mesmo e a definição de uma estratégia de resolução. Para o enquadramento com o tema da AD, em particular da navegação local na AD, foi realizada uma revisão da literatura com a pesquisa e estudo de algoritmos de planeamento de trajetória e de planeamento por múltiplas hipóteses. Este levantamento de algoritmos permitiu perceber quais as diferentes abordagens que se podem adotar na realização do planeamento local e quais as que melhor se enquadram em tarefas de planeamento real, em veículos autónomos. Depois desta consulta bibliográfica ficou bem presente a ideia que, para um planeamento eficaz a velocidades relativamente elevadas, é necessária a escolha de um algoritmo computacionalmente rápido e com uma representação simplificada do espaço de planeamento. Quando o objetivo é uma navegação mais lenta, podem ser empregues algoritmos baseados em grelhas de ocupação que determinam a trajetória ótima segundo vários parâmetros. A ideia original era a implementação de um algoritmo tradicional de planeamento em grelhas. No entanto, esta opinião foi reavaliada e substituída por um algoritmo de abordagem por múltiplas hipóteses depois de identificados exemplos de sucesso deste tipo de planeamento na literatura.

Identificada a estratégia de resolução do problema da navegação local, iniciou-se a fase de implementação da mesma com a ambientação à estrutura de *software* ROS, já incorporada no ATLASCAR2 por projetos anteriores. Foi colocado em funcionamento o

package ROS `free_space_detection`, já existente, que fornece a informação do espaço livre e habilitou-se o *package* `trajectory_planner`, onde o algoritmo já se encontrava implementado, para interagir com a informação do espaço livre publicada. Foi nesta fase que surgiram três grandes obstáculos: o algoritmo não estava preparado para o planeamento dinâmico de trajetórias, mas sim para o planeamento de trajetórias fixas de estacionamento, os obstáculos eram fornecidos ao algoritmo como linhas e não era possível o teste do funcionamento em tempo real do algoritmo no ATLASCAR2. O primeiro obstáculo foi ultrapassado com a criação e manipulação de *software* que permitiu gerar trajetórias em arco de circunferência com comprimento variável em função da velocidade. Esta resolução possibilitou ainda a modelação da velocidade de saída do algoritmo em função da direção selecionada pelo mesmo, quanto maior o ângulo de direção menor a velocidade. Foram ainda acrescentados parâmetros que permitem a alteração do ângulo entre as trajetórias desenhadas. Para solucionar os problemas gerados pela forma como os obstáculos eram introduzidos no algoritmo, converteu-se a metodologia de deteção de colisões do cálculo de interseções entre linhas, para a identificação de inclusões poligonais. Esta mudança eliminou por completo a criação de barreiras virtuais pelo mecanismo de deteção de colisões originalmente implementado pelo algoritmo. Entre outras alterações de melhoria do funcionamento do algoritmo e visualização da direção a seguir, foi criado um simulador para testar o desempenho do mesmo perante as modificações e parametrizações realizadas.

A criação de um simulador permitiu realizar testes, impossíveis de praticar no ATLASCAR2, com repetibilidade do ambiente de forma a comparar resultados entre as várias configurações do algoritmo de planeamento. Ao escolher o *Gazebo* para realizar as simulações, foi conseguida uma integração direta com o ROS e ficou criada uma plataforma que permite testar este ou outro algoritmo que venha a ser implementado no ATLASCAR2. O modelo do veículo está descrito de acordo com as principais características dimensionais e cinemáticas, como a direção do tipo *Ackermann*, do veículo real. Apesar da identificação das linhas da estrada não estar ainda disponível no ATLASCAR2, durante as simulações surgiu a possibilidade de incorporar a informação da linha central no algoritmo de planeamento e realizaram-se reajustes no mesmo para acrescentar estes dados ao cálculo das pontuações das trajetórias.

Conclusões

Da análise dos resultados obtidos podem ser retiradas algumas propriedades acerca do comportamento do algoritmo de navegação local. A primeira, e mais óbvia, é que quanto maior peso, no cálculo das pontuações das trajetórias, for dado ao parâmetro DLO (ponto atrator), mais segura será a navegação pois o algoritmo tende a escolher trajetórias que conduzam o modelo ao local mais afastado dos obstáculos. Também se pode afirmar que o comprimento das trajetórias potenciais influencia a qualidade do planeamento. Trajetórias demasiado longas proporcionam uma navegação menos segura e quando se reduz drasticamente o seu comprimento o algoritmo deixa de conseguir evitar colisões com objetos colocados frontalmente ao modelo. A utilização de trajetórias dinâmicas, isto é, trajetórias com comprimento variável em função da velocidade, também não se revelou tão vantajosa como o esperado. A explicação encontra-se no facto da distância entre os pontos finais das trajetórias aumentar com o aumento do seu comprimento e, desta forma, ser mais custosa a troca entre duas trajetórias. Como o final das trajetórias

é mais afastado é preciso que a trajetória atual conduza a uma maior proximidade aos obstáculos para que a trajetória adjacente se desloque para o centro do espaço livre e o algoritmo a selecione. Como forma de solucionar este problema, pode ser implementado um controlador de direção ou pode-se densificar as trajetórias na zona central para velocidades superiores, mas com prejuízo da eficiência computacional do algoritmo.

Nos testes em ambiente real ficou demonstrada a falta de informação exterior para se realizar um planeamento com sucesso e a grande diversidade que os parâmetros de cálculo da pontuação das trajetórias devem assumir perante diferentes cenários. Por fim, concluiu-se que a introdução das linhas da estrada é um fator preponderante para um planeamento em situações reais, onde se pretende manter o veículo dentro da faixa de rodagem.

Numa visão global do trabalho realizado ao longo desta dissertação pode-se dizer que os principais objetivos foram atingidos. Foram ainda atingidos objetivos suplementares, mas essenciais ao cumprimento dos propostos como, por exemplo, a criação do simulador para testar e avaliar o desempenho do algoritmo. Depois da análise dos resultados obtidos, facilmente se percebe que o algoritmo de navegação local está ainda longe de poder ser aplicado em ambiente real sem a incorporação de nova informação na escolha da trajetória.

Em resumo, para além do conhecimento adquirido, esta dissertação deixa três importantes contribuições para o LAR. Um resumo dos algoritmos de navegação local, com as suas principais vantagens e desvantagens. Um algoritmo de planeamento de trajetórias por múltiplas hipóteses mais completo e de personalização mais fácil, face ao originalmente utilizado para o planeamento de estacionamento. Sendo que este algoritmo fica ainda totalmente integrado com o *software* já existente na plataforma ATLASCAR2. E um simulador que permitirá testar novos algoritmos, numa fase inicial de implementação, cuja utilização se possa vir a revelar vantajosa em outras situações de planeamento local.

7.2 Trabalhos futuros

Ficaram a cargo desta dissertação os primeiros trabalhos de algoritmia de navegação local para AD e assistência ao condutor instalados no ATLASCAR2; como tal, é possível abrir novas frentes de trabalho que tiveram origem em dificuldades detetadas durante as atividades realizadas.

A primeira dificuldade foi encontrada na realização de testes reais. É impossível transpor situações reais de condução para um simulador devido ao elevado grau de complexidade do ambiente real. Assim, o teste e adequação do algoritmo devem ser realizados com base em dados reais. Esse teste pode ser realizado de três formas distintas: o condutor segue as orientações do algoritmo e é avaliado o resultado da condução seguida pelo condutor; o condutor age por si e a sua ação é comparada com os resultados que algoritmo calcula paralelamente; ou o algoritmo controla o veículo, estando o condutor responsável por atuar em casos de falha iminente do algoritmo. Para a realização de testes de acordo com as duas primeiras metodologias é necessário aceder aos valores de posição do volante, velocidade, hodometria, e estado dos pedais de acelerador e travão através porta *On-Board Diagnostic* (OBD) II do veículo. Estes dados seriam posteriormente publicados em tópicos ROS para visualização e seguimento por parte do condutor

ou para comparação com os resultados do algoritmo. No caso de ser o algoritmo a controlar diretamente o carro, seriam necessárias grandes intervenções a nível de atuação e controlo dos sistemas de travagem, aceleração e direção. Havia que ser estudada a viabilidade de atuação direta, via eletrónica, nos diferentes sistemas ou a necessidade de instalação de servomotores auxiliares para realizar a atuação. Adicionalmente teriam de ser desenvolvidos controladores para que a atuação proporcionasse uma condução que cumprisse com parâmetros de conforto previamente estabelecidos.

Outra lacuna identificada foi a falta da deteção dos limites visuais da estrada. Na grande maioria das estradas principais existem linhas a limitar as vias, que condicionam a atuação do condutor. A identificação destas linhas para inserção no algoritmo de planeamento seria um fator decisivo para a realização de testes realistas e uma correta parametrização do algoritmo de navegação local do ATLASCAR2. Ainda no âmbito da perceção com recurso a dados visuais, seria de extremo valor o desenvolvimento de *software* que identificasse e segmentasse objetos, e conseguisse prever se estes se iram deslocar para o espaço livre de navegação, permitindo a alteração preventiva da trajetória selecionada.

Por fim, da revisão da literatura e do conhecimento adquirido, é previsível que este algoritmo de planeamento não cubra todas as situações de navegação. Para casos específicos como cruzamentos e ambientes com grande densidade de trânsito seria interessante considerar a possibilidade de ajuste dos parâmetros do algoritmo por múltiplas hipóteses em função do ambiente ou o teste de outros algoritmos de planeamento de forma a comparar a sua atuação com o algoritmo já implementado.

Referências bibliográficas

Andrade da Costa, Eugénio Paulo (2012). "Navegação em Ambientes Exteriores do Robô da Série Atlas 2000". Dissertação de mestrado. Departamento de Engenharia Mecânica, Universidade de Aveiro. URL: http://hdl.handle.net/10773/9678.

Ansuategui, A., A. Arruti, L. Susperregi, Y. Yurramendi, E. Jauregi, E. Lazkano e B. Sierra (2014). "Robot trajectories comparison: A statistical approach". Em: *The Scientific World Journal* 2014. DOI: 10.1155/2014/298462.

Bimbraw, Keshav (2015). "Autonomous Cars: Past, Present and Future - A Review of the Developments in the Last Century, the Present Scenario and the Expected Future of Autonomous Vehicle Technology". Em: *Proceedings of the 12th International Conference on Informatics in Control, Automation and Robotics* August, pp. 191–198. DOI: 10.5220/0005540501910198.

Borenstein, Johann e Yoram Koren (1991). "The Vector Field Histogram - Fast Obstacle Avoidance for Mobile Robots". Em: *IEEE Journal of Robotics and Automation* 7.3, pp. 278–288. DOI: 10.1109/70.88137.

Borenstein, Johann e Iwan Ulrich (1998). "VFH+: reliable obstacle avoidance for fast mobile robots". Em: *IEEE International Conference on Robotics and Automation* April, pp. 1572–1577. DOI: 10.1109/ROBOT.1998.677362.

Broggi, A., P. Medici, P. Zani, A. Coati e M. Panciroli (2012). "Autonomous vehicles control in the VisLab Intercontinental Autonomous Challenge". Em: *Annual Reviews in Control* 36.1, pp. 161–171. DOI: 10.1016/j.arcontrol.2012.03.012.

Broggi, Alberto, Michele Buzzoni, Stefano Debattisti, Paolo Grisleri, Maria Chiara Laghi, Paolo Medici e Pietro Versari (2013). "Extensive tests of autonomous driving technologies". Em: *IEEE Transactions on Intelligent Transportation Systems* 14.3, pp. 1403–1415. DOI: 10.1109/TITS.2013.2262331.

Cabral de Azevedo, Rui Filipe (2014). "Sensor Fusion of LASER and Vision in Active Pedestrian Detection". Dissertação de mestrado. Departamento de Engenharia Mecânica, Universidade de Aveiro. URL: http://hdl.handle.net/10773/14414.

Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest e Clifford Stein (2001). Em: *Introduction to Algorithms, Second ed.* Ed. por MIT Press. Massachusetts, USA: McGraw Hill. Cap. 24, 595 à 601.

Fernández, Enrique, Luis Sánchez Crespo, Anil Mahtani e Aaron Martinez (2015). *Learning ROS for Robotics Programming, Second Edition.* Nonserial Publication. Birmingham, UK: Packt Publishing Ltd. ISBN: 978-1-78398-758-0.

Ganesha Perumal, D., B. Subathra, G. Saravanakumar e Seshadhri Srinivasan (2016). "Extended kalman filter based path-planning algorithm for autonomous vehicles with I2V communication". Em: *IFAC-PapersOnLine* 49.1, pp. 652–657. DOI: 10.1016/j.ifacol.2016.03.130.

Geraerts, Roland e Mark H Overmars (2006). "Sampling and node adding in probabilistic roadmap planners". Em: *Robotics and Autonomous Systems* 54, pp. 165–173. DOI: 10.1016/j.robot.2005.09.026.

Goodrich, Michael A. e Rodney Brooks (2002). "Potential Fields Tutorial". Em: *Class Notes 157*.

Gruyer, Dominique, Valentin Magnier, Karima Hamdi, Laurène Claussmann, Olivier Orfila e Andry Rakotonirainy (2017). "Perception, information processing and modeling: Critical stages for autonomous driving applications". Em: *Annual Reviews in Control* 44, pp. 323–341. DOI: 10.1016/j.arcontrol.2017.09.012.

Guiggiani, Massimo (2014). Em: *The Science of Vehicle Dynamics*. Ed. por Springer Science+Business Media. London, UK: Springer. Cap. 1, 1 à 6.

Harabor, Dd e Alban Grastien (2011). "Online Graph Pruning for Pathfinding On Grid Maps." Em: *AAAI Conference on Artificial Intelligence* 25, pp. 1114–1119.

Hart, Peter E., Nils J. Nilsson e Bertram Raphael (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". Em: *IEEE Transactions of Sistems Science and Cybernetics* 2.37, pp. 100–107. DOI: 10.1145/1056777.1056779.

Hess, Robin, Florian Kempf e Klaus Schilling (2013). "Trajectory Planning for Car-Like Robots using Rapidly Exploring Random Trees*". Em: *IFAC Symposium on Telematics Applications* 3.11-13, pp. 44–49. DOI: 10.3182/20131111-3-KR-2043.00018.

Hu, Xuemin, Long Chen, Bo Tang, Dongpu Cao e Haibo He (2018). "Dynamic path planning for autonomous driving on various roads with avoidance of static and moving obstacles". Em: *Mechanical Systems and Signal Processing* 100, pp. 482–500. DOI: 10.1016/j.ymssp.2017.07.019.

Jo, Kichun, Minchae Lee, Dongchul Kim, Junsoo Kim, Chulhoon Jang, Euiyun Kim, Sangkwon Kim, Donghwi Lee, Changsup Kim, Seungki Kim, Kunsoo Huh e Myoungho Sunwoo (2013). "Overall reviews of autonomous vehicle A1 - System architecture and algorithms". Em: *IFAC Intelligent Autonomous Vehicles Symposium* 8.26-28, pp. 114–119. DOI: 10.3182/20130626-3-AU-2035.00052.

De-Juan, A., R. Sancibrian e F. Viadero (2012). "Optimal Synthesis Of Function Generation In Steering Linkages". Em: *International Journal of Automotive Technology* 13.7, pp. 1033–1046. DOI: 10.1007/s12239-012-0106-4.

Katrakazas, Christos, Mohammed Quddus, Wen Hua Chen e Lipika Deka (2015). "Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions". Em: *Transportation Research Part C: Emerging Technologies* 60, pp. 416–442. DOI: 10.1016/j.trc.2015.09.011.

Kavraki, Lydia E., Petr Svestka, Jean-Claude Latombe e Mark H. Overmars (1996). "Probabilistic Roadmaps for Path Planning in High-Dimensional Configuratoin Spaces". Em: *IEEE Transactions on Robotics and Automation* 12.4, pp. 566–580. DOI: 10.1109/ROBOT.2000.846405.

LaValle, Steven (1998). "Rapidly-Exploring Random Trees: A New Tool for Path Planning". Em: *In* 129, pp. 98–11. DOI: 10.1.1.35.1853.

Li, Xiaohui, Zhenping Sun, Dongpu Cao, Zhen He e Qi Zhu (2016). "Real-time trajectory planning for autonomous urban driving: Framework, algorithms, and verifications". Em: *IEEE/ASME Transactions on Mechatronics* 21.2, pp. 740–753. DOI: 10.1109/TMECH.2015.2493980.

Madureira Correia, José Diogo (2017). "Unidade de Perceção Visual e de Profundidade Para o Atlascar2". Dissertação de mestrado. Departamento de Engenharia Mecânica, Universidade de Aveiro. URL: http://hdl.handle.net/10773/22498.

Moreau, Julien, Pierre Melchior St, Stéphane Victor, François Aioun e Franck Guillermard (2017). "Path planning with fractional potential fields for autonomous vehicles". Em: *IFAC Proceedings Volumes (IFAC-PapersOnline)* 50-1, pp. 15098–15103. DOI: 10.1016/j.ifacol.2017.08.2076.

Mouhagir, Hafida, Véronique Cherfaoui, Reine Talj, François Aioun e Franck Guillemard (2017). "Trajectory Planning for Autonomous Vehicle in Uncertain Environment Using Evidential Grid". Em: *IFAC Proceedings Volumes (IFAC-PapersOnline)* 50-1, pp. 12545–12550. DOI: 10.1016/j.ifacol.2017.08.2193.

Oliveira, Miguel, Vitor Santos e Angel D. Sappa (2012). "Short term path planning using a multiple hypothesis evaluation approach for an autonomous driving competition". Em: *IROS Workshop on Planning, Perception and Navigation for Intelligent Vehicles* Outubro.

Pereira, Joel Filipe (2012). "Estacionamento autónomo usando perceção 3D Autonomous parking using 3D perception". Dissertação de mestrado. Departamento de Engenharia Mecânica, Universidade de Aveiro. URL: http://hdl.handle.net/10773/9870.

Pereira, Marcelo, David Silva, Vítor Santos e Paulo Dias (2016). "Self calibration of multiple LIDARs and cameras on autonomous vehicles". Em: *Robotics and Autonomous Systems* 83, pp. 326–337. DOI: 10.1016/j.robot.2016.05.010.

Petereit, Janko, Thomas Emter, Christian W Frey, Thomas Kopfstedt e Andreas Beutel (2012). "Application of Hybrid A* to an Autonomous Mobile Robot for Path Planning in Unstructured Outdoor Environments". Em: *Proceedings of ROBOTIK 2012* 1, pp. 227–232.

Riem de Oliveira, Miguel Armando (2014). "Sistemas Automáticos de Informação e Segurança para Apoio na Condução de Veículos". Tese de doutoramento. Departamento de Engenharia Mecânica, Universidade de Aveiro. URL: http://hdl.handle.net/10773/10658.

Ulrich, Iwan e Jahann Borenstein (2000). "VFH*: local obstacle avoidance with lookaheadverification". Em: *Proceedings of the 2000 IEEE Intl. Conf. on Robotics and Automation (ICRA)* 24-28, pp. 2505–2511. DOI: 10.1109/ROBOT.2000.846405.

Vieira da Silva, David Tiago (2016). "Calibração Multissensorial e Fusão de Dados Utilizando LIDAR e Visão". Dissertação de mestrado. Departamento de Engenharia Mecânica, Universidade de Aveiro. URL: http://hdl.handle.net/10773/17967.

Werling, Moritz, Julius Ziegler, S Kammel e Sebastian Thrun (2010). "Optimal trajectory generation for dynamic street scenarios in a frenet frame". Em: *IEEE International Conference on Robotics and Automation* 8.3, pp. 987–993. DOI: 978-1-4244-5040-4/10/$26.00.

WHO, World Health Organization (2015). *Global Status Report on Road Safety 2015*. Nonserial Publication. World Health Organization. URL: apps.who.int/iris/bitstream/10665/189242/1/9789241565066_eng.pdf?ua=1.

Referências bibliográficas eletrónicas

CIR-KIT (2016a). *ROS Wiki: Robots/CIR-KIT-Unit03*. URL: http://wiki.ros.org/Robots/CIR-KIT-Unit03 (acedido em 14/05/2018).

— (2016b). *ROS Wiki: steer_bot_hardware_gazebo*. URL: http://wiki.ros.org/steer_bot_hardware_gazebo (acedido em 14/05/2018).

LARlabs (2011). *Projeto ATLAS - Universidade de Aveiro*. URL: http://atlas.web.ua.pt/index.html (acedido em 16/05/2018).

MMC, Mitsubishi Motors Corporation (2018a). *MMC-Manuals*. URL: http://mmc-manuals.ru/manuals/i-miev/online/Service_Manual/img/37/ACA02126AB00ENG.pdf (acedido em 01/05/2018).

— (2018b). *MMC-Manuals*. URL: http://mmc-manuals.ru/manuals/i-miev/online/Service_Manual/2013/index_M2.htm (acedido em 02/05/2018).

Nexus (2016). *Nexus, Servidor P-2308H4/HR4*. URL: https://nexus-solutions.pt/wp-content/uploads/2015/05/NEXUS-SERVIDOR-P-2308H4-HR4.pdf (acedido em 03/05/2018).

OSRF, Open Source Robotics Foundation (2014). *Gazebo, Robot simulation made easy*. URL: http://gazebosim.org/ (acedido em 04/05/2018).

Patel, Amit (2018). *Introduction to A**. URL: http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html (acedido em 30/04/2018).

Santos, Vítor (2018). *LAR toolkit - version 5*. URL: https://github.com/vitoruapt/lartkv5 (acedido em 07/05/2018).

Sick (2018a). *Sick LD-MRS400001 Online Data Sheet*. URL: https://www.sick.com/media/pdf/5/55/355/dataSheet_LD-MRS400001_1045046_en.pdf (acedido em 02/05/2018).

— (2018b). *Sick LMS151-10100 Online Data Sheet*. URL: https://www.sick.com/media/pdf/0/40/840/dataSheet_LMS151-10100_1047607_en.pdf (acedido em 02/05/2018).

Sunday, Dan (2012). *Inclusion of a Point in a Polygon*. URL: http://geomalgorithms.com/a03-_inclusion.html (acedido em 08/05/2018).

Tesla (2018a). *Tesla - Model S*. URL: https://www.tesla.com/pt_PT/models (acedido em 17/05/2018).

— (2018b). *Tesla - Sistema de Piloto Totalmente Automático em Todos os Carros*. URL: https://www.tesla.com/pt_PT/autopilot?utm&redirect=no (acedido em 17/05/2018).

Uber (2018a). *Uber - Advanced Technologies Group*. URL: https://www.uber.com/info/atg/car/ (acedido em 17/05/2018).

— (2018b). *Uber - Advanced Technologies Group*. URL: https://www.uber.com/info/atg/ (acedido em 17/05/2018).

Villamil Vuelta, Alvaro (2018). *Avillamil-tfg: Prácticas docentes en JdeRobot, circuito realista F1 y brazo manipulador*. URL: http://jderobot.org/Avillamil-tfg (acedido em 14/05/2018).

Waymo (2018a). *Waymo - Journey*. URL: https://waymo.com/journey/ (acedido em 17/05/2018).

— (2018b). *Waymo - Technology*. URL: https://waymo.com/tech/ (acedido em 17/05/2018).

Apêndice A

Packages ROS

A.1 *Package* de planeamento

O *package* `trajectory_planner` é responsável pela implementação do algoritmo de planeamento e de seguida é apresentada a configuração dos parâmetros do algoritmo, a inicialização do mesmo e o seu repositório *online*.

Configuração

Dependendo das circunstâncias de navegação, os parâmetros do algoritmo podem ser alterados de forma a cumprir os objetivos do planeamento. A lista de parâmetros das trajetórias que é apresentada de seguida pode ser configurada no ficheiro `c_manage_trajectory.h`:

- `W_DAP` – peso do parâmetro Distância ao Ponto Atrator (DAP) na equação de pontuação das trajetórias.
- `W_ADAP` – peso do parâmetro Diferença Angular ao Ponto Atrator (ADAP) na equação de pontuação das trajetórias.
- `W_DLO` – peso do parâmetro Distância Mínima aos Obstáculos (DLO) na equação de pontuação das trajetórias.
- `W_CL` – influência do parâmetro Linha Central (CL) na equação de pontuação das trajetórias.
- `_SPEED_SAFFETY_` – velocidade mínima a que o veículo deve circular.
- `_SPEED_REQUIRED_` – velocidade máxima a que o veículo deve circular.

Outros parâmetros, como o número de trajetórias que o planeamento avalia e o número de nodos em que estas são divididas, são definidos no ficheiro `trajectory_planner_nodelet.h`:

- `_NUM_TRAJ_` – número de trajetórias utilizadas para a realização do planeamento.
- `_NUM_NODES_` – número de nodos em que as trajetórias são discretizadas para a avaliação de colisões.

Inicialização

A inicialização do *package* é realizada com o `roslaunch`. No entanto, o planeamento pode ser realizado com base em dados reais, obtidos do sensores a bordo do ATLASCAR2, ou a partir do ambiente simulado no *Gazebo*.

Para a utilização de dados reais é necessário que o *package* `free_space_detection` esteja em execução de forma a publicar a nuvem de pontos, resultante da fusão dos dados dos sensores, no tópico `/reduced_pcl`. O lançamento do programa nestas condições é realizado com o comando:

```
roslaunch trajectory_planner trajectory_planner_way_points.launch
```

Quando se pretende utilizar o ambiente de simulação para a obtenção dos dados LIDAR, o simulador deve estar em execução simultânea. É dada a possibilidade de incorporar a informação da linha central no planeamento com o parâmetro `sim:=true`, que pode ser alterado no ficheiro de lançamento ou adicionado ao comando de lançamento:

```
roslaunch trajectory_planner trajectory_planner_simulated_simulator.launch
```

GitHub

Devido ao múltiplos contribuintes do projeto ATLASCAR2, o *package* ROS está disponível *online* no repositório:

– `https://github.com/ricardolfsilva/trajectory_planner.git`

A.2 *Package* de simulação

O *package* `cirkit_unit03_gazebo` é responsável não só pelo lançamento do simulador, mas também pelo tratamento de dados de forma a uma interegração imendiata com o *package* de planeamento local. As dependências do *package* de simulação, a sua inicialização e o repositório de alojamento *online* são apresentados de seguida.

Dependências

Devido à grande complexidade que a criação de um simulador com um modelo personalizado, controlado por um controlador não padronizado, apresenta o *package* de simulação foi adaptado do *package* `cirkit_unit03_gazebo`. Originalmente, a lista de depêndencias deste pacote ROS era extensa, mas para a aplicação de simulação criada apenas são necessárias as seguintes:

- `cirkit_unit03_control` – *package* onde a configuração dos controladores do modelo é realizada.
- `cirkit_unit03_description` – *package* com a descrição do modelo do robô e dos sensores simulados.
- `steer_drive_controller` – *package* de implementação inicial do controlador de direção, após correta configuração.

- `steer_bot_hardware` – *package* de passagem do modelo simples do controlador para um modelo do tipo *Ackermann*, com direção independente nas rodas dianteiras e rotação diferenciada nas rodas traseiras.

Inicialização

À semelhança do *package* de planeamento local, o simulador também é inicializado com o `roslaunch`. O comando seguinte permite a inicialização conjunta dos dois pacotes ROS pois o modelo do ATLASCAR2 desloca-se de acordo com os comandos do planeador local:

```
roslaunch cirkit_unit03_gazebo ackermann_vehicle_simulator.launch
```

A simulação está configurada pelo ficheiro de lançamento para começar pausada. A alteração desta e outras definições da interface gráfica podem ser realizadas no ficheiro de inicialização, sem compilação posterior.

GitHub

Para utilização futura, em trabalhos do projeto ATLASCAR2, o *package* ROS e as suas dependências estão disponíveis *online* no repositório:

– `https://github.com/ricardolfsilva/trajectory_planner_simulator.git`