



**Manuel Teles  
Ferreira**

**Planeamento Dinâmico de Trajetórias Locais  
para o ATLASCAR2 em Ambientes com  
Múltiplos Veículos**





**Manuel Teles  
Ferreira**

**Planeamento Dinâmico de Trajetórias Locais  
para o ATLASCAR2 em Ambientes com  
Múltiplos Veículos**

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestrado 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.



**O júri / The jury**

Presidente / President

**Prof. Doutor Miguel Armando Riem de Oliveira**

Professor Auxiliar da Universidade de Aveiro

Vogais / Committee

**Prof. Doutor Vítor Manuel Ferreira dos Santos**

Professor Associado C/ Agregação da Universidade de Aveiro (orientador)

**Prof. Doutor Artur José Carneiro Pereira**

Professor Auxiliar da Universidade do Aveiro



## **Agradecimentos / Acknowledgements**

Em primeiro lugar gostaria de dedicar este espaço à minha família que sempre me apoiou, quer com a sua presença e apoio, quer com conselhos fundamentais para ter cumprido os meus objetivos. Quero agradecer ao professor Vítor Santos pela excelente orientação durante esta dissertação, que procurou sempre estar atualizado com o desenvolvimento do projeto, criando motivação e boa disposição quando os resultados não agradavam. Gostaria também de agradecer aos meus colegas do LAR pela ajuda prestada e pelo ambiente amigável proporcionado, com destaque ao Bernardo Lourenço que sempre se mostrou disponível para ajudar quando o software estava de mau humor. Obrigado ao Tiago Almeida e à Daniela Rato pela boa disposição criada durante estes meses. Por fim quero agradecer à Ana Sousa pelo apoio e paciência demonstrada para aturar uma pessoa tão ansiosa e por ter fornecido um computador que fosse capaz de realizar as simulações necessárias. Muito obrigado!





**Palavras-chave**

Planeador de Trajetórias; Condução Autónoma; ROS; Gazebo; LIDAR; Manobras-tipo

**Resumo**

Um dos maiores desafios no desenvolvimento da condução autónoma corresponde ao planeamento da trajetória a percorrer, sendo necessário a recolha de dados provenientes de sensores instalados no veículo, obtendo informação sobre os obstáculos na vizinhança. Deste modo, esta dissertação tem como objetivo a adaptação do planeador do ATLASCAR2 proveniente de trabalhos realizados em anos anteriores, de modo a que este algoritmo seja capaz de responder a situações comuns durante uma viagem, como por exemplo a ultrapassagem ou o cruzamento entre veículos. Para tal foi necessário reformular o ambiente de simulação existente de modo a permitir a utilização de múltiplos veículos no simulador Gazebo tendo cada um o seu próprio planeador de trajetórias, tendo sido aperfeiçoado a interação entre o programador e a simulação, possibilitando a realização de múltiplos testes sem a necessidade de alterações profundas ao algoritmo. Foram também realizadas alterações ao planeador de trajetórias geral com foco na condução pela via da direita e na utilização de um ponto atrator dinâmico capaz de alterar o comportamento do veículo consoante a manobra a realizar. Desta forma foi possível a parametrização de manobras-tipo especificando o comportamento que o veículo autónomo deve desempenhar em cada situação, dando destaque à parametrização da ultrapassagem. O desenvolvimento desta dissertação utilizou o *Robot Operating System* (ROS) com o auxílio do simulador Gazebo para avaliar o desempenho do planeador de trajetórias desenvolvido.



**Keywords**

Trajectory Planner; Autonomous Driving; ROS; Gazebo; LIDAR; Standard maneuvers

**Abstract**

One of the greatest challenges in the development of autonomous driving corresponds to the planning of the trajectory to be executed, requiring the collection of data acquired by sensors installed in the vehicle, therefore obtaining information about the obstacles in its periphery. With this in mind, this dissertation aims at adapting the trajectory planner of the ATLASCAR2 developed in previous projects, so that this algorithm is able to respond to common situations during a trip, such as overtaking or an intersection between vehicles. In order to do so, it was necessary to reformulate the existing simulation environment in order to allow the use of multiple vehicles in the Gazebo simulator, each having its own trajectory planner, improving the interaction between the programmer and the simulation, allowing multiple tests without the need for profound alterations to the algorithm. Changes were also made to the general trajectory planner focusing on the right lane driving and the use of a dynamic attractor point capable of altering the vehicle's behavior depending on the maneuver to be performed. In this way it was possible to parameterize different maneuvers specifying the behavior that the autonomous vehicle should perform in each situation, with emphasis in the parametrization of the overtaking maneuver. The development of this dissertation used the *Robot Operating System* (ROS) with the aid of the Gazebo simulator to evaluate the performance of the developed trajectory planner.



# Conteúdo

<b>1</b>	<b>Introdução</b>	<b>1</b>
1.1	Enquadramento . . . . .	1
1.2	Apresentação do Problema . . . . .	2
1.3	Objetivos . . . . .	2
1.3.1	Implementação da infraestrutura de simulação computacional em ambientes com múltiplos veículos . . . . .	2
1.3.2	Definição, estudo e parametrização de trajetórias e das métricas de avaliação e seleção em contexto de múltiplos veículos. . . . .	2
1.3.3	Testes de verificação, e se possível avaliação, das técnicas desenvolvidas em situação real a bordo do carro. . . . .	3
1.4	Importância do Simulador . . . . .	3
1.5	Estrutura do Documento . . . . .	4
<b>2</b>	<b>Revisão da literatura</b>	<b>5</b>
2.1	Planeadores Locais . . . . .	5
2.2	Simuladores . . . . .	7
2.2.1	Gazebo . . . . .	7
2.2.2	CARLA . . . . .	8
2.2.3	LGSVL Simulator . . . . .	9
2.2.4	Outros Simuladores . . . . .	10
2.3	Projetos existentes sobre múltiplos veículos . . . . .	10
2.3.1	TurtleBot3 . . . . .	10
2.3.2	Husky . . . . .	12
2.3.3	YouBot e RBCAR . . . . .	13
<b>3</b>	<b>Infraestrutura experimental</b>	<b>15</b>
3.1	Hardware . . . . .	15
3.1.1	ATLASCAR2 . . . . .	15
3.1.2	LIDAR . . . . .	16
3.2	Software . . . . .	16
3.2.1	ROS . . . . .	16
3.2.2	Software CAD 3D . . . . .	18
3.3	Ferramentas de Simulação . . . . .	18
3.3.1	Rqt . . . . .	18
3.3.2	Rviz . . . . .	20
3.3.3	Execução da Simulação . . . . .	21

<b>4</b>	<b>Reformulação do ambiente de simulação</b>	<b>23</b>
4.1	Infraestrutura do Algoritmo de Simulação . . . . .	23
4.1.1	Launch files e ficheiros respetivos . . . . .	23
4.1.2	Descrição dos ficheiros do algoritmo . . . . .	25
4.2	Nós e Tópicos . . . . .	27
4.3	Limitações da infraestrutura inicial . . . . .	29
4.4	Alterações ao programa existente . . . . .	29
4.5	Mapa de Simulação . . . . .	30
4.6	Namespaces . . . . .	31
4.7	Alterações ao ficheiro de lançamento do modelo . . . . .	32
4.8	Alterações restantes . . . . .	35
<b>5</b>	<b>Planeamento em manobras-tipo</b>	<b>37</b>
5.1	Parâmetros da simulação . . . . .	37
5.2	Cálculo da trajetória . . . . .	38
5.2.1	Distância ao ponto atrator (DAP) . . . . .	39
5.2.2	Diferença Angular ao Ponto Atrator (ADAP) . . . . .	40
5.2.3	Distância Mínima aos Obstáculos (DLO) . . . . .	41
5.2.4	Espaço livre (FS) . . . . .	42
5.2.5	Cálculo final da trajetória . . . . .	42
5.3	Disposição das diferentes trajetórias . . . . .	43
5.4	Cálculo da velocidade dependente do ângulo . . . . .	44
5.5	Condução pela via da direita . . . . .	45
5.6	Ponto Atrator . . . . .	46
5.6.1	Ponto Atrator utilizando Waypoits . . . . .	47
5.6.2	Ponto Atrator utilizando a linha central . . . . .	48
5.7	Zona de deteção de obstáculos . . . . .	50
5.8	Zona de deteção de obstáculos traseira . . . . .	56
5.9	Cruzamento de dois veículos . . . . .	57
5.10	Ultrapassagem . . . . .	58
<b>6</b>	<b>Testes e resultados</b>	<b>71</b>
6.1	Simulação base da análise . . . . .	71
6.2	Utilização da nova disposição de trajetórias . . . . .	73
6.3	Análise ao ponto atrator . . . . .	76
6.3.1	Ponto atrator utilizando waypoints . . . . .	76
6.3.2	Ponto atrator utilizando a linha central . . . . .	77
6.4	Consequências do novo ponto atrator . . . . .	77
6.5	Zona de deteção de obstáculos . . . . .	79
6.5.1	Falha na criação do espaço de deteção . . . . .	81
6.6	Testes às manobras-tipo . . . . .	83
6.6.1	Cruzamento de dois veículos . . . . .	83
6.6.2	Ultrapassagem . . . . .	84
6.6.3	Comportamento do veículo ultrapassado . . . . .	88
6.6.4	Ultrapassagem a múltiplos veículos . . . . .	88

<b>7</b>	<b>Conclusões e trabalhos futuros</b>	<b>93</b>
7.1	Problemas e objetivos iniciais . . . . .	93
7.2	Metodologia . . . . .	93
7.3	Avaliação do trabalho realizado e Contribuições . . . . .	94
7.4	Trabalhos futuros . . . . .	95
	<b>Referências</b>	<b>95</b>





# Lista de Figuras

1.1	Planeamento de ultrapassagem . . . . .	1
1.2	Ilustração de uma situação de ultrapassagem . . . . .	3
2.1	rqt-graph do ambiente de simulação . . . . .	6
2.2	Logo do simulador Gazebo . . . . .	7
2.3	Ambiente de simulação do simulador CARLA . . . . .	9
2.4	Ambiente de simulação do simulador LGSVL . . . . .	10
2.5	3 versões do TurtleBot3 . . . . .	11
2.6	Componentes do TurtleBot3 Burger . . . . .	11
2.7	Robô Husky . . . . .	12
2.8	Simulação e controlo dos Huskys . . . . .	12
2.9	Veículos utilizados nos respetivos projetos . . . . .	13
3.1	ATLASCAR2 no exterior do Departamento de Engenharia Mecânica . . . . .	15
3.2	LIDARs presentes no ATLASCAR2 . . . . .	16
3.3	Versões ROS mais utilizadas atualmente . . . . .	17
3.4	Software CAD 3D utilizados . . . . .	18
3.5	Exemplo de um rqt_graph . . . . .	19
3.6	rqt_graph com namespaces . . . . .	19
3.7	Exemplo de um rqt_console . . . . .	19
3.8	Exemplo de um rqt_reconfigure . . . . .	20
3.9	Ambiente inicial do Rviz . . . . .	20
4.1	Diagrama do ficheiro ackermann_vehicle_simulator.launch . . . . .	24
4.2	Exemplo de diagrama entre ficheiro cpp e nó executado . . . . .	24
4.3	Diagrama do ficheiro ackermann_single_model.launch lançado para cada veículo . . . . .	25
4.4	Arquitetura do nó Gazebo . . . . .	27
4.5	Arquitetura do nó cirkit_unit03_gazebo . . . . .	27
4.6	Arquitetura do nó cirkit_unit03_gazebo_line . . . . .	27
4.7	Arquitetura do nó APgenerator . . . . .	28
4.8	Arquitetura do nó RetrieveInfo . . . . .	28
4.9	Arquitetura do nó Rviz . . . . .	28
4.10	Arquitetura do nó trajectory_planner_nodelet . . . . .	29
4.11	Simulação no mapa Monaco . . . . .	30
4.12	Diferentes opções do mapa utilizado . . . . .	30
4.13	Visualização do rqt_graph durante a simulação de dois veículos . . . . .	33
4.14	Bloco de colisão . . . . .	35

4.15	Visualização das colisões . . . . .	35
4.16	<code>rqt_reconfigure</code> durante a simulação de um veículo . . . . .	36
5.1	Demonstração visual do DAP com recurso ao Rviz . . . . .	39
5.2	Demonstração visual do ADAP com recurso ao Rviz . . . . .	41
5.3	Demonstração visual do DLO de uma trajetória com recurso ao Rviz . . . . .	42
5.4	Disposição das trajetórias potenciais visualizadas no Rviz . . . . .	44
5.5	Deteção de obstáculos da linha central . . . . .	46
5.6	Deteção de obstáculos da linha central e parede lateral . . . . .	46
5.7	Colisão iminente com o limite da via . . . . .	47
5.8	<i>Waypoints</i> ao longo do percurso . . . . .	47
5.9	Ângulo gerado por duas linhas de <i>waypoints</i> . . . . .	48
5.10	<i>Waypoints</i> mais concentrados nas curvas . . . . .	48
5.11	Rviz com o novo Ponto Atrator . . . . .	49
5.12	Espaço de análise . . . . .	51
5.13	Exemplificação do espaço detetável da curva da direita . . . . .	52
5.14	Exemplificação do espaço detetável da curva da esquerda . . . . .	53
5.15	Visualização no Rviz da zona de deteção . . . . .	55
5.16	Visualização no Rviz da zona de deteção traseira . . . . .	56
5.17	Cruzamento de dois veículos . . . . .	57
5.18	Deteção do veículo em sentido contrário de cada veículo visualizados no Rviz . . . . .	57
5.19	Diagrama total da ultrapassagem . . . . .	58
5.20	Diagrama do início da deteção de obstáculos . . . . .	59
5.21	Início da deteção de obstáculos . . . . .	60
5.22	Diagrama da primeira fase da ultrapassagem . . . . .	61
5.23	Deteção de obstáculos . . . . .	62
5.24	Diagrama da segunda fase da ultrapassagem . . . . .	62
5.25	Início da ultrapassagem . . . . .	63
5.26	Diagrama da terceira fase da ultrapassagem . . . . .	64
5.27	Ultrapassagem e deteção de obstáculos na via da direita . . . . .	64
5.28	Diagrama da quarta fase da ultrapassagem . . . . .	65
5.29	Diagrama da quinta fase da ultrapassagem . . . . .	66
5.30	Fim da deteção de obstáculos . . . . .	67
5.31	Desvia para a via da direita . . . . .	68
5.32	Diagrama da última fase da ultrapassagem . . . . .	68
5.33	Conclusão da ultrapassagem . . . . .	69
6.1	Visualização no simulador Gazebo dos dois veículos nas suas posições iniciais . . . . .	71
6.2	Disposição inicial das trajetórias . . . . .	73
6.3	Disposição final das trajetórias . . . . .	73
6.4	Diferença entre valores do parâmetro <code>MIN_STEERING_ANGLE</code> . . . . .	74
6.5	Diferença entre valores do parâmetro <code>NUM_TRAJ</code> . . . . .	74
6.6	Gráfico de comparação entre a velocidade e o ângulo máximo das trajetórias geradas com o parâmetro <code>Vel_Ang</code> ativo . . . . .	75
6.7	Gráfico de comparação entre a velocidade e o ângulo máximo das trajetórias geradas com o parâmetro <code>Vel_Ang</code> desativo . . . . .	75
6.8	Histograma do número de trajetórias para cada intervalo de ângulos . . . . .	76

6.9	Comparação entre a velocidade e a distância do ponto atrator ao longo da ultrapassagem . . . . .	78
6.10	Histograma da distância dos pontos atratores gerados durante a ultrapassagem . . . . .	78
6.11	Gráfico dos pontos detetados pelas duas zonas de detecção . . . . .	79
6.12	Gráfico dos pontos totais detetados pelas duas zonas de detecção . . . . .	80
6.13	Bloqueio da detecção da parede lateral direita ao veículo a ultrapassar por parte do veículo da direita . . . . .	81
6.14	Visualização da falha de detecção da parede lateral direita . . . . .	82
6.15	Zona onde ocorre a falha de detecção da parede lateral direita . . . . .	83
6.16	Distância mínima aos obstáculos durante um cruzamento de 2 veículos . . . . .	84
6.17	Distância mínima aos obstáculos durante uma ultrapassagem . . . . .	85
6.18	Relação entre a distância mínima aos obstáculos e a ativação do parâmetro LINES . . . . .	86
6.19	Ativação do parâmetro LINES de acordo com a distância à linha central . . . . .	87
6.20	Distância mínima ao veículo a ser ultrapassado . . . . .	87
6.21	Comportamento do veículo ultrapassado . . . . .	88
6.22	Ultrapassagem a múltiplos veículos . . . . .	89
6.23	Ultrapassagem do primeiro veículo . . . . .	89
6.24	Ultrapassagem do segundo veículo . . . . .	90
6.25	Distância mínima aos obstáculos durante a ultrapassagem de três veículos . . . . .	90
6.26	Relação entre a distância mínima aos obstáculos e a ativação do parâmetro LINES durante a ultrapassagem de três veículos . . . . .	91
6.27	Distância à linha central consoante os pontos detetados . . . . .	91
6.28	Distância mínima aos veículos a serem ultrapassados . . . . .	92



# Lista de Excertos de Código

2.1	Comando de lançamento de uma simulação Gazebo . . . . .	8
2.2	Controlo de um YouBot indicando o namespace desejado . . . . .	13
4.1	Comando de inicialização da simulação . . . . .	23
4.2	Ficheiro launch básico para um namespace . . . . .	31
4.3	Remap de tópicos para ficarem inseridos num namespace específico (excerto presente no ficheiro ackermann_single_model.launch) . . . . .	32
5.1	Cálculo do DAP para uma trajetória . . . . .	39
5.2	Normalização do valor DAP da trajetória em questão . . . . .	40
5.3	Cálculo do ADAP para uma trajetória . . . . .	40
5.4	Normalização do valor ADAP da trajetória em análise . . . . .	40
5.5	Criação das trajetórias . . . . .	43
5.6	Cálculo do ponto atrator . . . . .	50
5.7	Cálculo do $y_{min\_l\_right}$ . . . . .	52
5.8	Cálculo do $y_{max\_w\_right}$ . . . . .	53
5.9	Cálculo do $y_{min\_l\_left}$ . . . . .	54
5.10	Cálculo do $y_{max\_w\_left}$ . . . . .	54
5.11	Cálculo dos limites no eixo Oy do espaço de deteção de obstáculos . . . . .	54
5.12	Início da deteção de obstáculos . . . . .	59
5.13	Início da deteção de obstáculos . . . . .	61
5.14	Aproximação à linha central . . . . .	63
5.15	Veículo na via da esquerda . . . . .	64
5.16	Deteção de obstáculos na via da direita . . . . .	65
5.17	Segunda aproximação à linha central . . . . .	67
5.18	Segunda aproximação à linha central . . . . .	69



# Capítulo 1

## Introdução

Este primeiro capítulo contempla uma introdução à temática da dissertação onde é exposto o enquadramento e problemas/objetivos, e também a estruturação do documento.

### 1.1 Enquadramento

Dado que os veículos autónomos prometem ser o modo predominante de mobilidade da próxima geração, as empresas estão apressadas para serem as primeiras a comercializar tecnologias de condução autónoma. Os influenciadores deste rápido progresso incluem os fabricantes de veículos (por exemplo, Ford, GM, Tesla), os desenvolvedores de sistemas autónomos (Waymo, MobilEye, Tesla, TomTom), empresas de redes de transporte (Uber, Lyft, Didi Chuxing) e a infraestrutura na qual os veículos se encontram (por exemplo, municípios, estados, autoestradas com portagem privada) [1].

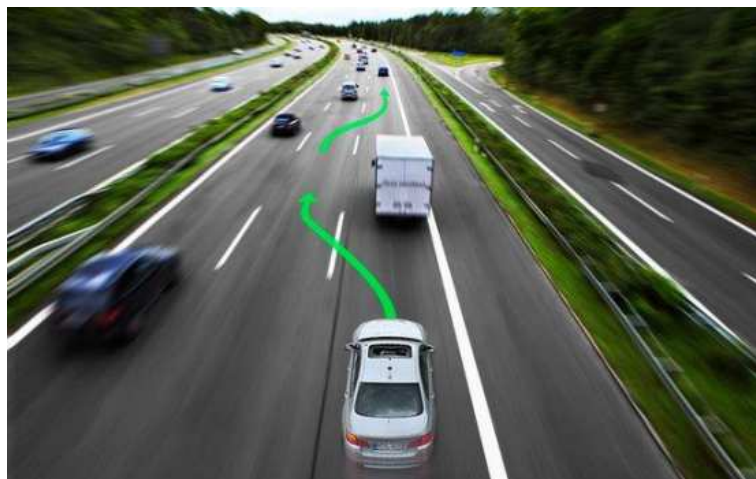


Figura 1.1: Planeamento de ultrapassagem [2]

A condução autónoma envolve grandes desafios sendo um dos principais a toma de decisão, isto é, o planeamento da trajetória a percorrer. Para tal é necessário, não só uma perceção dos limites do espaço navegável, como também a capacidade de detetar outros agentes que percorram este mesmo espaço. Estes agentes englobam tanto obstáculos como também outros veículos,

quer estes estejam ou não em movimento, isto é, quer sejam estáticos ou dinâmicos. Estas características são muito importantes para determinar o correto comportamento do veículo, sendo necessário uma caracterização dos agentes como, por exemplo, as dimensões e a velocidade, de modo a identificar quer o possível obstáculo, quer o tipo de situação em que o veículo se encontra como exemplificado na figura 1.1 (ultrapassagem, travagem do veículo da frente, etc.).

## **1.2 Apresentação do Problema**

Em projetos anteriores, foram desenvolvidos algoritmos de planeamento de trajetórias para veículos autónomos, que visam determinar a melhor trajetória recorrendo-se à informação recolhida pelos sensores LIDAR, de modo a evitar colisões com obstáculos. Tendo em conta que estes obstáculos, quer sejam objetos ou veículos, podem estar em movimento, cria-se a necessidade de contemplar a sua dinâmica. Para tal, recorrendo a ambientes de simulação, é fundamental definir situações que possam ocorrer durante o percurso do veículo, e deste modo adaptar o seu comportamento.

Tratando-se de controlo de veículos, a análise em ambiente real implica riscos elevados sendo necessário a utilização de simuladores que permitam a análise do algoritmo em ambientes pré-configurados.

## **1.3 Objetivos**

Esta dissertação tem três objetivos principais a atingir, de modo a consolidar o planeador de trajetórias existente: a simulação de múltiplos veículos autónomos; a adição de eventuais situações que possam provocar um mau comportamento do veículo; e a análise do algoritmo em ambiente real.

### **1.3.1 Implementação da infraestrutura de simulação computacional em ambientes com múltiplos veículos**

De modo a permitir a simulação de múltiplos veículos autónomos em simultâneo, é necessário reformular o ambiente de simulação para que este possibilite a utilização de planeadores e controladores de informação iguais, isto é, para que cada veículo possua o seu próprio planeador, com todas as ferramentas necessárias agrupadas num único conjunto, sendo distinguível dos planeadores dos restantes veículos. Este objetivo é fundamental tendo em conta que é desejado que cada veículo utilize as mesmas regras de seleção de trajetórias, permitindo uma melhor análise do algoritmo desenvolvido.

Este ambiente de simulação terá de conter os modelos dos veículos com as suas respetivas definições (modelo de colisões, juntas, etc.), os sensores utilizados (neste caso os sensores LIDAR), e ainda um modelo do "mundo", isto é, o percurso que os veículos terão de percorrer.

### **1.3.2 Definição, estudo e parametrização de trajetórias e das métricas de avaliação e seleção em contexto de múltiplos veículos.**

Com a existência de agentes dinâmicos na simulação, torna-se inevitável a criação de manobras previstas, ou seja, de situações que possam ocorrer durante o percurso do veículo em que o plano de trajetória terá de ser alterado, de modo a que o veículo se comporte corretamente. O



caso de o veículo autónomo ser ultrapassado como ilustrado na figura 1.2 é um bom exemplo visto que a distância mínima ao obstáculo mais próximo será bastante reduzida, sendo necessário contemplar esta situação permitindo que este não se desvie do seu trajeto.

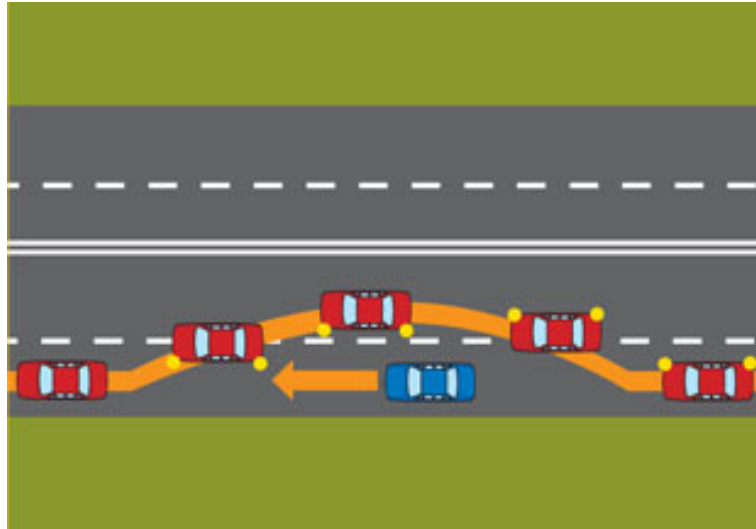


Figura 1.2: Ilustração de uma situação de ultrapassagem [3]

Deste modo, é necessário definir uma métrica geral para a seleção da melhor trajetória em função dos parâmetros já existentes e de outros a definir, com o objetivo de contemplar a percepção de outros veículos, dando ponderações variáveis de modo a estas serem adequadas às diferentes situações em que o veículo se encontra.

Nesta métrica de seleção de trajetórias poderão ainda ser incluídas as propriedades conhecidas dos alvos em movimento, nomeadamente a sua posição e velocidade relativa.

### 1.3.3 Testes de verificação, e se possível avaliação, das técnicas desenvolvidas em situação real a bordo do carro.

Como último objetivo, tendo em consideração que o algoritmo desenvolvido tem como finalidade ser utilizado em ambiente real, o projeto será testado no ATLASCAR2 analisando assim o seu desempenho com dados reais. Este objetivo visa apenas uma consolidação do trabalho realizado, não sendo pretendido uma grande alteração/correção do algoritmo desenvolvido consoante os resultados obtidos através desta análise.

Tendo em conta a dificuldade de analisar o desempenho do planeador, sendo apenas possível comparar o comportamento real às trajetórias calculadas (por questões de segurança não será dado controlo do veículo ao planeador), esta análise poderá ser realizada através de *bags* que consistem num conjunto de dados recolhidos previamente dos sensores do ATLASCAR2 durante uma viagem, agrupados num único ficheiro. Deste modo, estes dados podem ser analisados repetidamente permitindo evitar múltiplos testes a bordo do ATLASCAR2.

## 1.4 Importância do Simulador

A simulação permite a análise do comportamento de veículos autónomos num número enorme de cenários, ambientes, configurações de sistemas e características de condução. Este facto não

torna os testes em ambiente real desnecessários, mas permite verificar quais destes testes serão realmente essenciais para a validação dos resultados da simulação.

Testes em ambientes reais irão contribuir para uma melhor validação do algoritmo, tendo em conta as situações inesperadas que podem ocorrer durante a condução real. Além disto, a simulação permite realizar um número elevado de testes num período de tempo muito inferior quando comparado com a análise em ambientes reais [4].

Além do tempo necessário, a simulação permite adicionar maior incerteza nos dados recolhidos pelos sensores. É possível produzir situações nas quais o veículo conduz em condições atmosféricas extremas, ambientes de trânsito adversos, entre outros, sem que ponha em causa a segurança dos condutores [5].

Apesar de todos os pontos positivos da simulação, as suas limitações não devem ser desprezadas. Atualmente existem dois pontos negativos sendo estes a falta de cenários de emergência e as potenciais diferenças entre os dados reais e simulados [5].

## 1.5 Estrutura do Documento

Esta dissertação encontra-se estruturada em oito capítulos: "Introdução", "Revisão da literatura", "Infraestrutura experimental", "Reformulação do ambiente de simulação", "Planeamento em manobras-tipo", "Testes e resultados" e "Conclusões e trabalhos futuros".

No atual capítulo é feita uma breve introdução à problemática deste projeto e o seu enquadramento no tema de veículos autónomos. São ainda explicados os objetivos principais e a importância do simulador.

No segundo capítulo, "Revisão da literatura", são expostos o planeador local que já se encontrava desenvolvido e que se tornou a base deste projeto, os diferentes simuladores que foram analisados para a problemática da simulação de múltiplos veículos, e por fim são referidos outros projetos que abordaram a problemática da simulação com múltiplos agentes.

No capítulo três, "Infraestrutura experimental", é dado destaque às diferentes ferramentas utilizadas nesta dissertação, sendo distinguido entre Hardware e Software. Na secção de Hardware é referido o ATLASCAR2, veículo presente no departamento de engenharia mecânica onde será testado o algoritmo no ambiente real, e os sensores LIDAR presentes neste veículo. Na secção de Software é exposto a ferramenta ROS e os softwares CAD utilizados na criação dos modelos. Este capítulo é concluído com a explicação das limitações encontradas no planeador de trajetórias existente quando exposto à utilização de múltiplos veículos.

No quarto capítulo denominado "Reformulação do ambiente de simulação" são descritos todos os ficheiros e tópicos presentes no *package* deste projeto, com vista a permitir uma maior facilidade de compreensão da estrutura deste. De seguida é descrito as alterações que foram executadas ao projeto existente. Estas alterações contemplam a necessidade de simular múltiplos veículos tendo sido necessário reformular o ambiente de simulação.

Conforme exposto nos objetivos desta dissertação, no capítulo cinco, "Planeamento em manobras-tipo", são descritos as alterações realizadas ao algoritmo e os comportamentos criados para permitir uma correta trajetória do veículo em situações específicas como a ultrapassagem e a travagem do veículo da frente, entre outros.

Por fim os dois últimos capítulos referem-se aos resultados obtidos com o novo algoritmo e as conclusões retiradas destes. É também referido possibilidades de trabalhos futuros que utilizem este projeto.

## Capítulo 2

# Revisão da literatura

De modo a apresentar as ferramentas e projetos utilizados no desenvolvimento desta dissertação, neste capítulo estes são descritos de uma forma geral, permitindo ao leitor o enquadramento necessário para a compreensão da sua utilização nos capítulos seguintes. Neste capítulo serão explicados os planeadores locais já existentes, os simuladores possíveis de ser utilizados para o teste do algoritmo e os projetos existentes que ajudaram no desenvolvimento do ambiente de simulação.

### 2.1 Planeadores Locais

Esta dissertação tem como base trabalhos realizados no Laboratório de Automação e Robótica da Universidade de Aveiro, nomeadamente as dissertações de Joel Pereira [6] e Ricardo Silva [7], tendo este último o objetivo de desenvolver um algoritmo de navegação local com recurso ao simulador Gazebo, incorporado no software de desenvolvimento ROS. Este algoritmo permite que um veículo autónomo (neste caso o ATLASCAR2) consiga, através de sensores LIDAR, calcular a trajetória que deve percorrer de modo a evitar colisões com objetos detetados por estes sensores.

Tendo em conta que esta dissertação tem como base trabalhos anteriores, de seguida é apresentado um resumo do trabalho de Ricardo Silva [7], tratando-se do projeto mais recente que foi utilizado e reformulado.

#### **Navegação Local do ATLASCAR2 para Condução Autónoma e Assistência ao Condutor**

Esta dissertação [7] teve como objetivo o desenvolvimento de um módulo de navegação local para assistência ao condutor na sua tomada de decisão imediata. Para tal foi utilizado o simulador Gazebo onde, através de sensores LIDAR posicionados no modelo 3D do ATLASCAR2, recolhia *scans* laser que posteriormente eram convertidos em *PointClouds*. Através destes dados era então selecionada a melhor trajetória utilizando os seguintes parâmetros: a Distância ao Ponto Atrator (DAP), a Diferença Angular ao Ponto Atrator (ADAP), a Distância Mínima aos Obstáculos (DLO), a Linha Central (CL) e do Espaço Livre (FS).

A trajetória calculada é enviada para o Gazebo e assim sucessivamente como pode ser visualizado na figura 2.1.

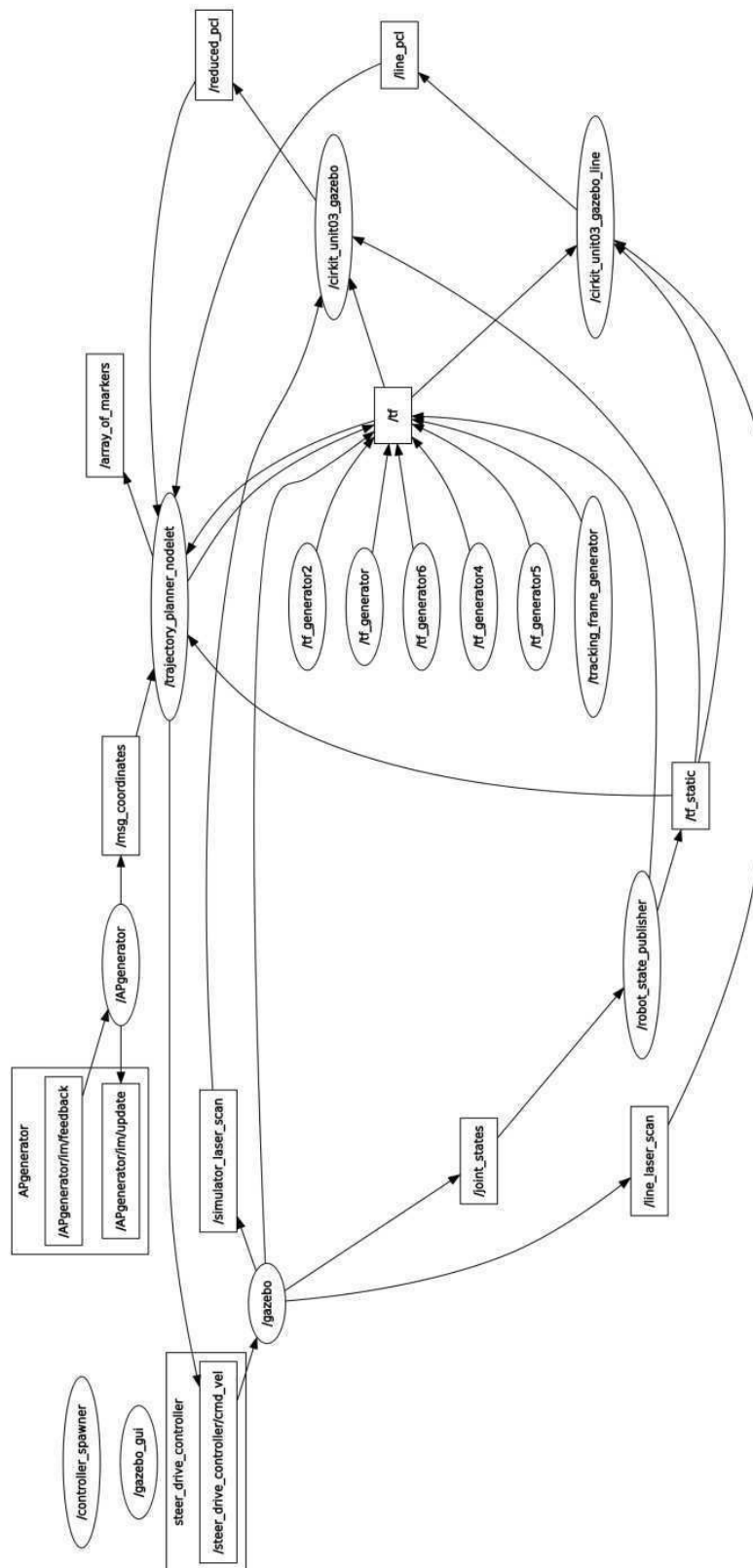


Figura 2.1: rqt-graph do ambiente de simulação [7]

## 2.2 Simuladores

Foram analisados diferentes simuladores com o objetivo de simular múltiplos veículos autônomos, isto é, que permitisse a distinção de múltiplos planejadores de trajetórias com diferentes sensores LIDAR. Esta análise foi necessária devido à dificuldade encontrada de criar um ambiente de simulação com múltiplos agentes no simulador Gazebo. Apesar desta dificuldade, o Gazebo acabou por ser o simulador escolhido, tendo maior destaque neste capítulo. Esta decisão foi tomada tendo em conta a familiaridade já obtida com este simulador, ao contrário dos restantes, que implicaria uma estudo sobre o funcionamento destes, sem certezas de que iriam funcionar para os objetivos deste projeto. De seguida este simulador é apresentado juntamente com outros dois simuladores, que têm como objetivo a simulação de sistemas de condução autónoma, com a principal característica de ter compatibilidade com a execução de múltiplos veículos. Além destes, serão referidos outros simuladores que, apesar de não se focarem em múltiplos agentes, são bastante utilizados pela comunidade de desenvolvimento desta área de investigação.

### 2.2.1 Gazebo

Sendo o simulador de preferência quando se trata de desenvolvimento e teste de algoritmos utilizados em robôs, o Gazebo (figura 2.2) destaca-se com a sua robustez quer na simulação física, na qualidade gráfica ou até mesmo na facilidade de programação [8]. Este último ponto deve-se à sua integração com o ambiente ROS e permite avaliar os algoritmos desenvolvidos utilizando as ferramentas disponíveis com este ambiente.



Figura 2.2: Logo do simulador Gazebo [8]

Como referido anteriormente, os simuladores têm grande importância, sendo o Gazebo um bom exemplo. Visto que este trabalho requer a existência de múltiplos veículos, é tido em conta na seleção do simulador a utilizar o facto de este simulador permitir a utilização de diversos tipos de sensores, gerando dados sensoriais e interações entre os diversos modelos bastante realistas.

#### Ficheiros World

O ficheiro de descrição `world` contém todos os elementos da simulação, incluindo os robôs, luzes, sensores e objetos estáticos. Este ficheiro é formatado utilizando SDF (*Simulation Description Format*), e tipicamente tem a extensão `.world`.

#### Ficheiros Model

Os ficheiros `Model`, tal como os ficheiros `world`, utilizam o formato SDF devendo conter apenas um único modelo. Estes ficheiros têm o propósito de facilitar a reutilização de modelos e simplificar os ficheiros `world`.

## Variáveis do Ambiente

O simulador Gazebo utiliza variáveis com o objetivo de localizar ficheiros e configurar as comunicações cliente-servidor.

Estas variáveis são:

- GAZEBO\_MODEL\_PATH - Diretórios onde o Gazebo irá procurar os modelos
- GAZEBO\_RESOURCE\_PATH - Diretórios onde o Gazebo irá procurar outros recursos como por exemplo ficheiros `world`.
- GAZEBO\_MASTER\_URI - *URI* do master do Gazebo. Este especifica o IP e porta onde o servidor será inicializado e indica ao cliente onde se conectar
- GAZEBO\_PLUGIN\_PATH - Diretórios onde o Gazebo irá procurar os *plugins*.
- GAZEBO\_MODEL\_DATABASE\_URI - *URI* da base de dados de modelos online onde o Gazebo irá executar o download destes.

## Servidor Gazebo

O servidor Gazebo (`gzserver`) analisa e simula um ficheiro `world` recebido pela linha de comandos.

## Cliente Gazebo

O cliente gráfico (`gzclient`) conecta-se a um servidor `gzserver` em funcionamento e visualiza os elementos. Esta ferramenta também permite modificar a simulação em execução.

## Servidor + Cliente Gráfico num só

De modo a executar o servidor e cliente num único comando, é executado o comando `gazebo`, podendo ser especificado o ficheiro `world` a utilizar.

```
1 gazebo worlds/empty_sky.world
```

Listagem 2.1: Comando de lançamento de uma simulação Gazebo

## Plugins

*Plugins* fornecem um mecanismo fácil e conveniente de interagir com o Gazebo. Estes *plugins* são úteis pois permitem ao desenvolvedor controlar quase todos os aspetos do Gazebo e podem ser inseridos e removidos de um sistema em funcionamento.

Estes *plugins* podem ser de seis tipos: World; Modelo; Sensor; Sistema; Visual; e GUI.

### 2.2.2 CARLA

O simulador CARLA foi criado com o objetivo de permitir o desenvolvimento de sistemas autónomos de navegação. Este simulador tem como vantagens não só a existência de código e protocolos *open-source*, como também o fornecimento de modelos de edifícios, veículos, etc. como é ilustrado na figura 2.3.

Para esta dissertação, a principal vantagem deste simulador é a compatibilidade de utilização de múltiplos clientes nos mesmos nós permitindo controlar diferentes veículos [9].

Este simulador destaca-se pela possibilidade da existência de múltiplos clientes, nos mesmos ou diferentes nós, possam controlar diferentes atuadores; a possibilidade de controlar todos os aspetos relativos à simulação como por exemplo o tráfego, o comportamento dos pedestres, as condições meteorológicas e os sensores; a possibilidade da utilização de diferentes sensores; a opção de desativar o modo render permitindo uma simulação rápida onde os gráficos não são necessários; a fácil criação de mapas; a existência de cenários de tráfego pré configurados; e a integração ROS.



Figura 2.3: Ambiente de simulação do simulador CARLA [10]

Este simulador encontra-se em desenvolvimento tendo até ao momento atingido 6 dos 8 objetivos principais propostos.

Os objetivos atingidos passam pelo suporte à simulação de cenários de tráfego; suporte à interface ROS; facilidade de importar e editar mapas; controlo de todos os veículos pelo cliente; controlo de todos os pedestres pelo cliente; e a desativação do modo de renderização para simulações de elevado desempenho.

Os dois objetivos ainda por atingir referem-se ao suporte de simulações em paralelo de cenários de tráfego na cloud e simulações RADAR.

### 2.2.3 LGSVL Simulator

Este simulador desenvolvido por *LG Silicon Valley Lab* tem como objetivo a simulação de sistemas autónomos em ambiente urbano (figura 2.4), correspondendo à necessidade de múltiplos veículos presente neste trabalho. Apesar deste ponto positivo, este simulador foi descartado devido à maior dificuldade de utilização do ambiente ROS e da pequena comunidade existente que utiliza este simulador [11].



Figura 2.4: Ambiente de simulação do simulador LGSVL [11]

Este simulador tem como características: a integração total com os programas Apollo, AutoWare e Duckietown; integração com o ROS/ROS2; a inclusão de modelos de sensores LIDAR, RADAR, GPS, etc.; e suporte a mapas HD.

#### 2.2.4 Outros Simuladores

Foram analisados outros simuladores, através de um estudo sobre os mesmos [12], como por exemplo DeepDrive [13] e Udacity [14], não tendo sido considerados devido à dificuldade de encontrar online projetos sobre múltiplos agentes utilizando estes simuladores, não conseguindo deste modo comprovar a possibilidade de realizar as simulações necessárias para este trabalho.

### 2.3 Projetos existentes sobre múltiplos veículos

Durante o decorrer desta dissertação foram identificados projetos que ajudaram a desenvolver e compreender o ambiente de simulação com múltiplos veículos. Nesta secção são apresentados os projetos mais importantes, quer por ajudarem na compreensão do funcionamento do ROS, quer por terem fornecido ideias de como organizar e desenvolver o ambiente de simulação.

#### 2.3.1 TurtleBot3

O TurtleBot3 é a terceira versão da plataforma TurtleBot, que consiste num robô desenvolvido para ensinar facilmente a programação em ROS. Este robô acabou por ser a plataforma mais popular entre alunos e desenvolvedores, tornando-se a plataforma base do ROS [15]. Existem três modelos de TurtleBots no mercado (figura 2.5), cada um com diferentes características, tendo sido utilizado para os tutoriais a versão *Burger* (figura 2.6).



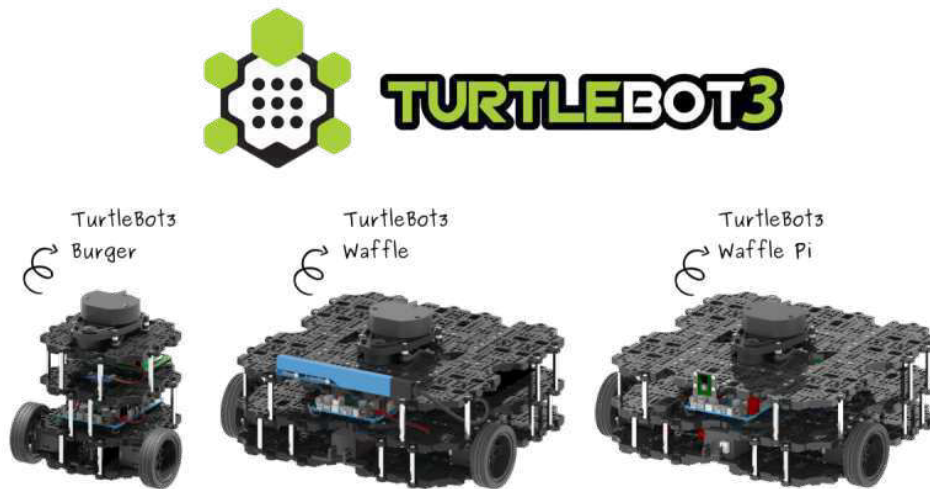


Figura 2.5: 3 versões do TurtleBot3 [16]

Este robô foi fundamental na aprendizagem da simulação no Gazebo, sendo este o exemplo dos tutoriais, existindo uma grande quantidade de informação e perguntas-respostas sobre o mesmo e os seus projetos.

## TurtleBot3 Burger

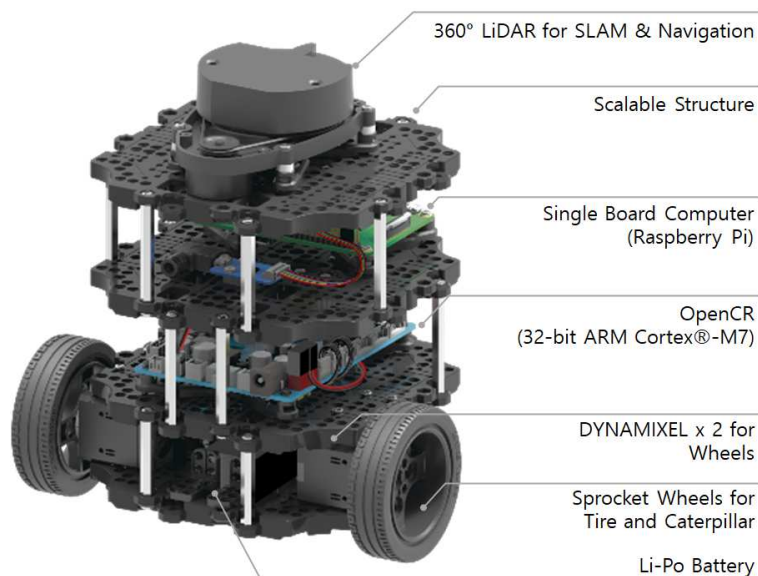


Figura 2.6: Componentes do TurtleBot3 Burger [16]

Contido nos tutoriais desta plataforma (mais especificamente na secção 17.5, "Load Multiple TurtleBot3s") encontra-se uma simulação onde o objetivo é simular múltiplos TurtleBots no mesmo ambiente Gazebo. Nesta simulação são lançados três robôs tendo a desvantagem de ser apenas possível controlar um de cada vez através da aplicação `teleop` (controlo através do teclado).

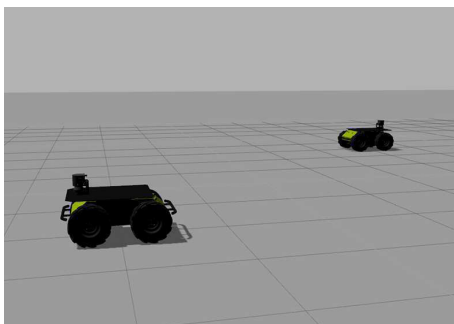
### 2.3.2 Husky

Outro projeto importante no desenvolvimento do ambiente de simulação de múltiplos veículos trata-se do *nre\_simmultihusky* desenvolvido por Brian Bingham. Este projeto utiliza o robô Husky (figura 2.7) que consiste num robô todo-o-terreno utilizado para investigação e aplicações de prototipagem rápida. Tal como o TurtleBot3, este robô é totalmente compatível com ROS [17].

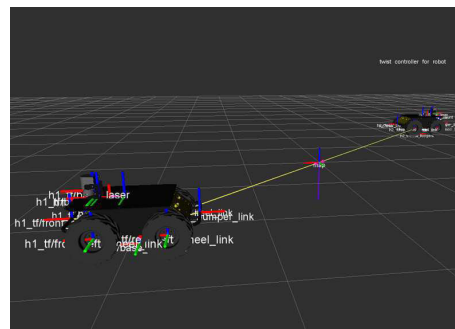


Figura 2.7: Robô Husky [17]

O projeto *nre\_simmultihusky* tem como objetivo a criação de um ambiente de simulação em Gazebo com múltiplos Huskys. Este projeto, apesar de não estar concluído, encontra-se mais adequado à problemática desta dissertação devido à existência de controlo de todos os robôs presentes na simulação através de marcadores interativos na ferramenta Rviz (figura 2.8), publicando diferentes velocidades para cada Husky. Apesar de não conter sensores nem planeadores de trajetórias, fornece ideias interessantes sobre a organização dos nós e tópicos que se tornaram a base da arquitetura desenvolvida, tendo uma estrutura bastante parecida quando comparada com o projeto atual [18].



(a) Huskys no simulador Gazebo



(b) Huskys no Rviz

Figura 2.8: Simulação e controlo dos Huskys [18]

### 2.3.3 YouBot e RBCAR

Os projetos que utilizam o YouBot e o RBCAR (figura 2.9), apesar de não contribuírem com novas ideias para esta dissertação, permitiram comprovar a metodologia dos projetos anteriores.

Ambos os projetos *youbot\_ros\_tools* e *rbcarsim*, tal como o *turtlebot3*, lançam dois modelos no mesmo ambiente Gazebo controlando-os, através da ferramenta *teleop* no caso do YouBot ou através da utilização de um *joystick* no caso do RBCAR, tendo estes projetos a vantagem de conter uma estrutura de código semelhante ao projeto Husky, isto é, utilizam a *launch file* para alterar dinamicamente o namespace de cada robô [19].

O controlo de cada agente é realizado executando a seguinte linha de código, onde é alterado o namespace (ns) de modo a colocar o tópico */teleop* na namespace do robô desejado:

```
1 roslaunch youbot_teleop youbot_teleop.launch ns:=/youbot0
```

Listagem 2.2: Controlo de um YouBot indicando o namespace desejado

Não existindo sensores nem planeadores de trajetórias, estes projetos não responde às questões iniciais da utilização de múltiplos agentes, permitindo contudo consolidar ideias de como separar os nós e tópicos dos mesmos.

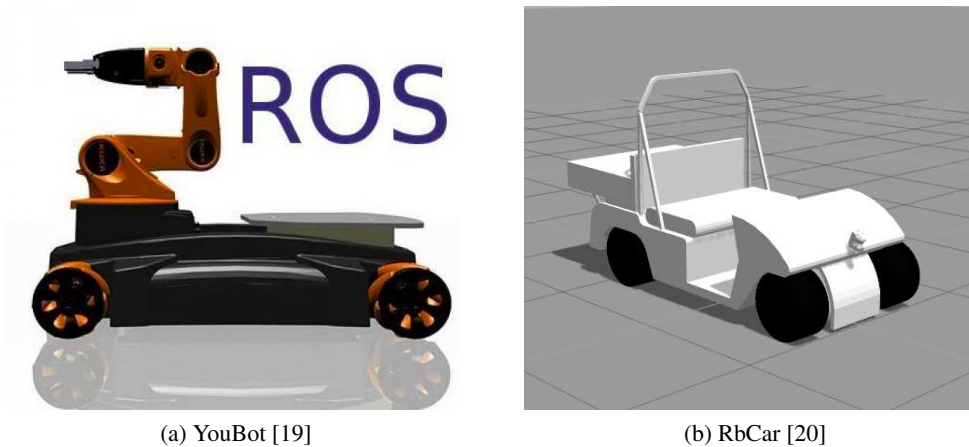


Figura 2.9: Veículos utilizados nos respetivos projetos



## Capítulo 3

# Infraestrutura experimental

### 3.1 Hardware

Esta dissertação tem como objetivo o planeamento de trajetórias de um veículo autónomo. Deste modo é importante apresentar o ATLASCAR2, veículo central deste projeto e o sensores utilizados na recolha de dados.

#### 3.1.1 ATLASCAR2

O ATLASCAR2, demonstrado na figura 3.1, é o veículo elétrico presente na Universidade de Aveiro utilizado nos projetos de automação e robótica. Este é um Mitsubishi i-MiEV com uma autonomia de 100Km utilizando uma bateria de 16KWh, tendo a vantagem de permitir utilizar esta sua bateria para alimentar os sensores ou outros dispositivos necessários.

Neste veículo encontram-se instalados 3 LIDARs, 3 câmaras, sensores de inclinometria e uma unidade de GNSS.



Figura 3.1: ATLASCAR2 no exterior do Departamento de Engenharia Mecânica [21]

### 3.1.2 LIDAR

A tecnologia LIDAR (Light Detection And Ranging) é uma tecnologia ótica de detecção remota que mede propriedades da luz refletida de modo a obter a distância e/ou outra informação a respeito um determinado objeto distante. Nos casos mais habituais, a distância a um objeto é determinada medindo a diferença de tempo entre a emissão de um pulso laser e a detecção do sinal refletido, de forma semelhante à tecnologia do radar, que utiliza ondas de rádio.

No ALTASCAR2 estão presentes dois sensores LIDAR 2D *Sick LMS151* e um sensor *Sick LD-MRS400001* (figura 3.2).

Os sensores *Sick LMS151* [22] têm um alcance de até 50m e um ângulo de abertura de feixe de 270°. Estes dois sensores comunicam utilizando o protocolo Ethernet, encontrando-se instalados nas extremidades do para-choques dianteiro.

O sensor 3D *Sick LD-MRS400001* [23] tem um alcance de trabalho de 300m com um alcance de *scan* de 50m. Este sensor permite a configuração da resolução dos seus quatro feixes planares permitindo densificar a nuvem de pontos na zona mais importante. Este sensor encontra-se no centro do para-choques dianteiro comunicando também através do protocolo Ethernet.



Figura 3.2: LIDARs presentes no ATLASCAR2

## 3.2 Software

Em termos de software é necessário obter conhecimentos do sistema utilizado, das aplicações CAD e das ferramentas de simulação.

### 3.2.1 ROS

ROS é um sistema *open-source* que fornece serviços de sistema operativo para o desenvolvimento de robôs. Estes serviços passam por abstração de hardware (camada entre o hardware físico de um computador e o software que corre neste), controlo de dispositivos de baixo nível, implementação de funcionalidades comumente utilizadas, passagem de mensagens entre processos e gestão de pacotes. Também disponibiliza bibliotecas e ferramentas para a obtenção, construção, escrita e execução de código através de múltiplos computadores. [24]

No momento da escrita desta dissertação, a versão Melodic é a versão mais atualizada do ROS, sendo que muitos dos projetos existentes ainda utilizam a versão Kinetic (figura 3.3).



(a) ROS Kinetic [25]



(b) ROS Melodic [26]

Figura 3.3: Versões ROS mais utilizadas atualmente

De seguida serão explicados resumidamente diferentes componentes base do ROS necessárias para a compreensão do projeto:

- **Nós** - Um nó é um executável que utiliza o ROS para comunicar com outros nós. Estes operam independentemente, podendo não influenciar os restantes nós na eventualidade de um erro (*crash*) [27];
- **Mensagens** - Tipo de dados ROS utilizados quando se subscreve ou publica a um tópico. Estas são estruturas de dados simples suportando tipos primitivos padrão (inteiros, booleanos, etc.) como também matrizes destes [28];
- **Tópicos** - Nós podem utilizar um tópico para enviar ou receber mensagens. Estes tópicos estão concebidos de forma a poderem ser interpretados por qualquer nó, isto é, têm uma semântica anónima de publicação/subscrição [29];
- **Namespace** - Um namespace consiste num agrupamento de nós e tópicos, permitindo que todos estes fiquem anexados com uma única referencia (ao criar um namespace h1, todos os nós e tópicos estarão inseridos em h1/). Deste modo é possível a coexistência de múltiplos nós e tópicos com o mesmo nome, existindo diferenciação entre os mesmos (Cada nó está contido num namespace diferente) [30].
- **Serviços** - O ROS suporta o conceito de "chamada de sistemas remotos" na forma de serviços ROS. Deste modo utilizar uma função de outro nó é tão simples como utilizar funções locais [31].
- **Parâmetros** - `roscpp` permite guardar e manipular dados no ROS Parameter Server onde é possível guardar desde números inteiros a listas [31].

### 3.2.2 Software CAD 3D

#### SolidWorks

O SolidWorks (figura 3.4.a) baseia-se em computação paramétrica, criando formas tridimensionais a partir de operações geométricas elementares. No ambiente do programa, a criação de um sólido ou superfície tipicamente começa com a definição de um *sketch* 2D que depois é transformado através de uma operação num modelo tridimensional.

O SolidWorks dispõe de um amplo leque de funcionalidades, incluindo funções específicas para chapa metálica, construção soldada e moldes [32].

Este software foi importante na criação de modelos 3D necessários na simulação.

#### Blender

O Blender (figura 3.4.b) implementa ferramentas similares às de outros programas proprietários, que incluem avançadas ferramentas de simulação, tais como: dinâmica de corpo rígido, dinâmica de corpo macio e dinâmica de fluidos, ferramentas de modelagem baseadas em modificadores, ferramentas de animação de personagens, um sistema de composição baseado em “nós” de texturas, cenas e imagens, e um editor de imagem e vídeo, com suporte a pós-produção.

O Blender foi utilizado de modo a converter os ficheiros STL provenientes do SolidWorks para ficheiros DAE, que são melhor suportados pelo sistema ROS e o simulador Gazebo [33].



(a) Logótipo SolidWorks [34]



(b) Logótipo Blender [35]

Figura 3.4: Software CAD 3D utilizados

## 3.3 Ferramentas de Simulação

Dentro do sistema ROS existem muitas ferramentas úteis para o desenvolvimento deste projeto. Por esta razão, é importante referir as ferramentas que foram utilizadas para melhor compreender o que foi realizado nesta dissertação.

### 3.3.1 Rqt

A ferramenta `rqt` é uma estrutura de software do ROS que implementa várias ferramentas GUI na forma de *plugins*. Estas ferramentas podem ser utilizadas dentro do ambiente `rqt` ou de forma independente.

#### `rqt_graph`

O `rqt_graph` disponibiliza um *plugin* GUI para a visualização do gráfico computacional ROS. Neste é possível ver a relação entre nós (círculos) e tópicos (retângulos), como representado na figura 3.5, como também as diferentes namespaces existentes.



Esta ferramenta é essencial na resolução de problemas relacionados com a má organização do algoritmo ou de falhas na comunicação entre os diferentes nós e tópicos.



Figura 3.5: Exemplo de um rqt\_graph [31]

Além de nós e tópicos na figura 3.6 é possível visualizar a organização por namespaces, agrupando estes num único cabeçalho.

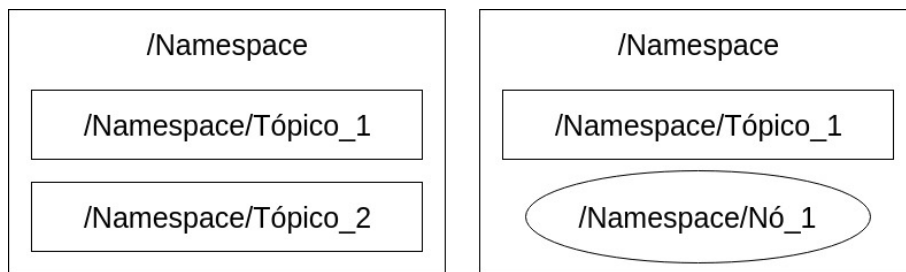


Figura 3.6: rqt\_graph com namespaces

### rqt\_console

O rqt\_console (figura 3.7) é um visualizador que exibe as mensagens a serem publicadas para o rosout, colecionando-as ao longo do tempo, permitindo a sua filtragem e análise com maior detalhe.

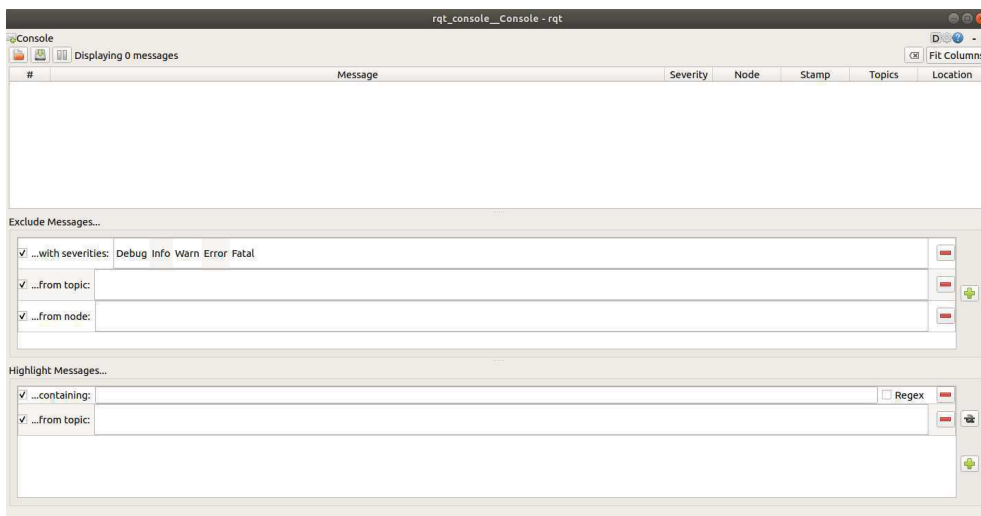


Figura 3.7: Exemplo de um rqt\_console

### rqt\_reconfigure

Este *rqt plugin* disponibiliza uma maneira de visualizar e editar os parâmetros que são acessíveis através do *dynamic\_reconfigure*. Esta ferramenta apenas consegue aceder aos parâmetros associados a nós, não sendo possível visualizar nem editar parâmetros inicializados em ficheiros *launch*. Na figura 3.8 é possível visualizar a utilização desta ferramenta na execução do projeto.

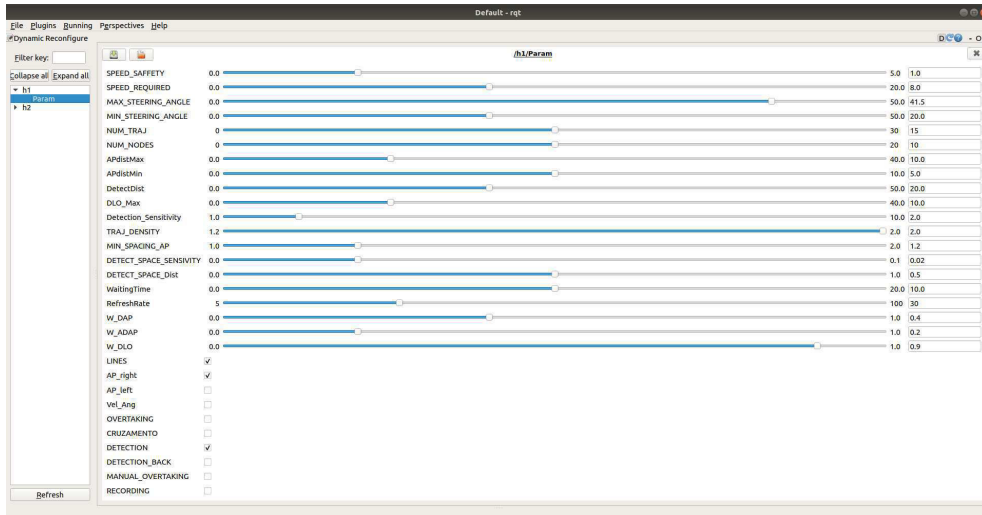


Figura 3.8: Exemplo de um *rqt\_reconfigure*

### 3.3.2 Rviz

A ferramenta Rviz (ROS Visualization) é um visualizador 3D (figura 3.9) *open-source* do sistema ROS. O desenvolvimento de um robô sem saber a informação que este recebe e apenas verificando os números recebidos é uma tarefa quase impossível. O Rviz permite ver a informação que este robô recebe através de câmaras, lasers ou codificadores de junta.

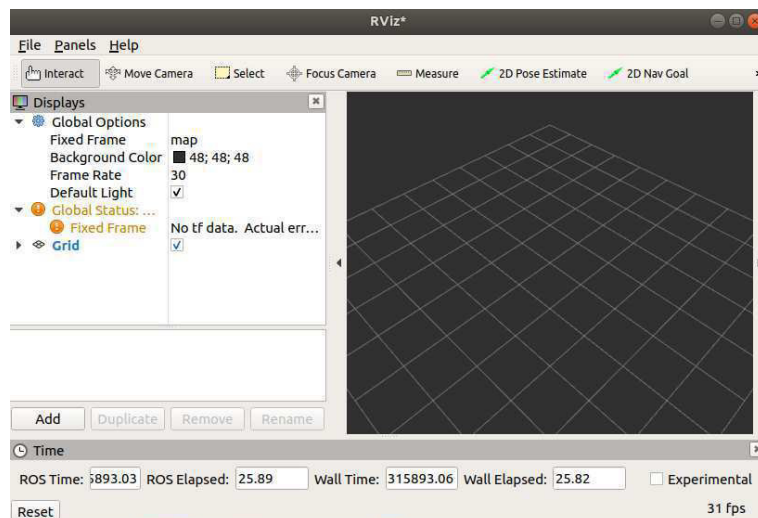


Figura 3.9: Ambiente inicial do Rviz

Existem duas maneiras de visualizar informação no Rviz. Primeiro, esta ferramenta interpreta informação sensorial como *scans* laser, nuvens de pontos, câmaras ou *frames* de coordenadas, onde é possível especificar como se deseja ver esta informação. Em segundo lugar, existem marcadores visuais que permitem ao programador enviar cubos, setas ou linhas, da maneira que desejar. A combinação de dados sensoriais e marcadores visuais tornam o Rviz uma ferramenta poderosa no desenvolvimento e investigação de robôs.

### 3.3.3 Execução da Simulação

Nesta secção, de modo a permitir uma melhor compreensão das alterações realizadas sobre o sistema existente, é descrito o funcionamento do simulador inicial, isto é, o funcionamento antes das alterações.

Para lançar a simulação é utilizado o seguinte comando `roslaunch` no terminal:

```
roslaunch cirkit_unit03_gazebo ackermann_vehicle_simulator.launch
```

Este comando lança o ficheiro `ackermann_vehicle_simulator.launch` do package `cirkit_unit03_gazebo`. O ficheiro `launch` tem acoplado a si outros ficheiros `launch`, como também ficheiros `yaml` e `xacro` com configurações, entre outros.

Após execução do comando no terminal, abre-se uma janela Gazebo e uma janela Rviz. No Gazebo encontra-se o "mundo" (neste caso a pista de Fórmula1 do Mónaco), o modelo do ATLASCAR2 e dos sensores, e as configurações destes.

O Rviz está encarregado de mostrar visualmente o resultado do planeador de trajetórias, com a pontuação de cada uma destas e o trajeto selecionado. Além disto, está representado o ponto atrator interativo, sendo este necessário de ser movido para iniciar a simulação.

Com a simulação iniciada, o veículo percorrerá a pista evitando colidir com as paredes e possíveis obstáculos estáticos que podem ser colocados nesta durante a simulação.



## Capítulo 4

# Reformulação do ambiente de simulação

Neste capítulo descrevem-se os passos que foram necessários para a alteração do ambiente de simulação de modo a permitir a integração de múltiplos veículos autónomos sem existir conflitos entre nós e tópicos. Este problema advém da necessidade de cada tópico e nó reconhecer devidamente com quem tem que comunicar, o que se torna um desafio quando se gera múltiplas instâncias com o mesmo nome, sendo que cada veículo terá de conter o seu próprio planeador de trajetórias. Além disto, o código está configurado para aceder a tópicos e parâmetros específicos, isto é, é fornecido a cada nó o nome específico de cada tópico com que este necessita de interagir, impedindo alteração do seu nome sem comprometer a sua execução.

### 4.1 Infraestrutura do Algoritmo de Simulação

Existindo neste projeto um número elevado de ficheiros, nós e tópicos para a execução das simulações, este subcapítulo descreve-os de modo a permitir uma maior compreensão da tarefa e importância de cada um. Para tal, será feito uma listagem seguindo-se um breve resumo da sua função. Esta listagem está organizada por tipos de ficheiros.

#### 4.1.1 Launch files e ficheiros respetivos

##### **ackermann\_vehicle\_simulator.launch**

Ficheiro launch principal utilizado para lançamento da simulação. Como exemplificado na figura 4.1 este ficheiro contém o lançamento do "mundo", das variáveis do Gazebo, a criação dos namespaces (vehicle\_1, vehicle\_2, etc.) e lançamento do ficheiro launch para cada veículo (ackermann\_single\_model.launch);

```
1 roslaunch cirkit_unit03_gazebo ackermann_vehicle_simulator.launch
```

Listagem 4.1: Comando de inicialização da simulação

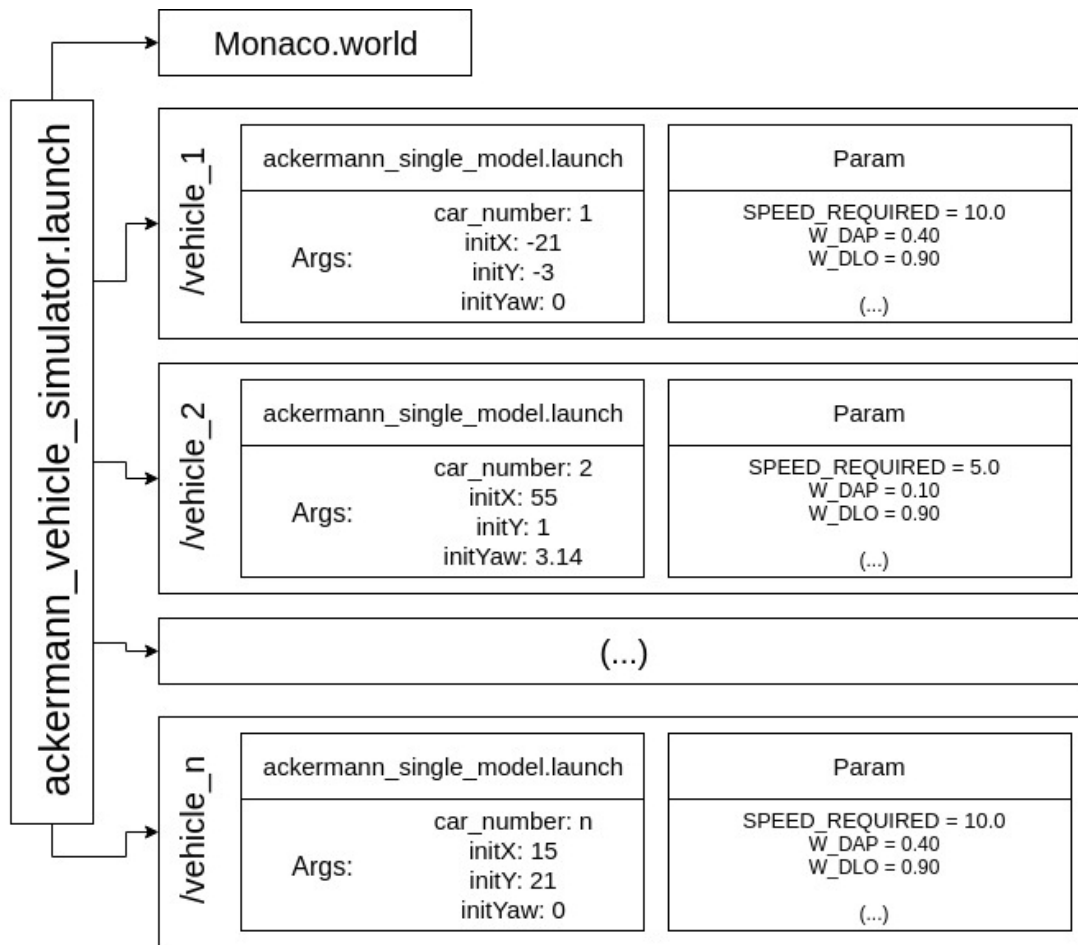


Figura 4.1: Diagrama do ficheiro ackermann\_vehicle\_simulator.launch

Este é o único ficheiro necessário de alterar, contendo os parâmetros iniciais de cada veículo, também sendo possível de serem alterados antes do início da simulação ou durante a mesma, através da aplicação `rqt_reconfigure` que é inicializada juntamente com as restantes ferramentas.

No final desta launch file é possível configurar o lançamento de ferramentas do sistema ROS úteis durante a análise da simulação como o `rqt_graph` ou o `rqt_console`.

### ackermann\_single\_model.launch

Ficheiro launch representado no diagrama da figura 4.3 que inicializa todos os nós e tópicos necessários para criar um veículo autónomo dentro do Gazebo. Neste diagrama é possível visualizar todos os ficheiros pertencentes a esta launch file, separados em cada saída por tipos de ficheiros. Foi incluído destaque aos nós (círculos) referentes a cada ficheiros cpp como exemplificado na figura 4.2.

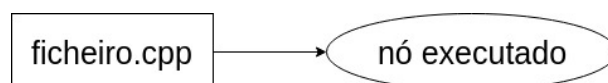


Figura 4.2: Exemplo de diagrama entre ficheiro cpp e nó executado

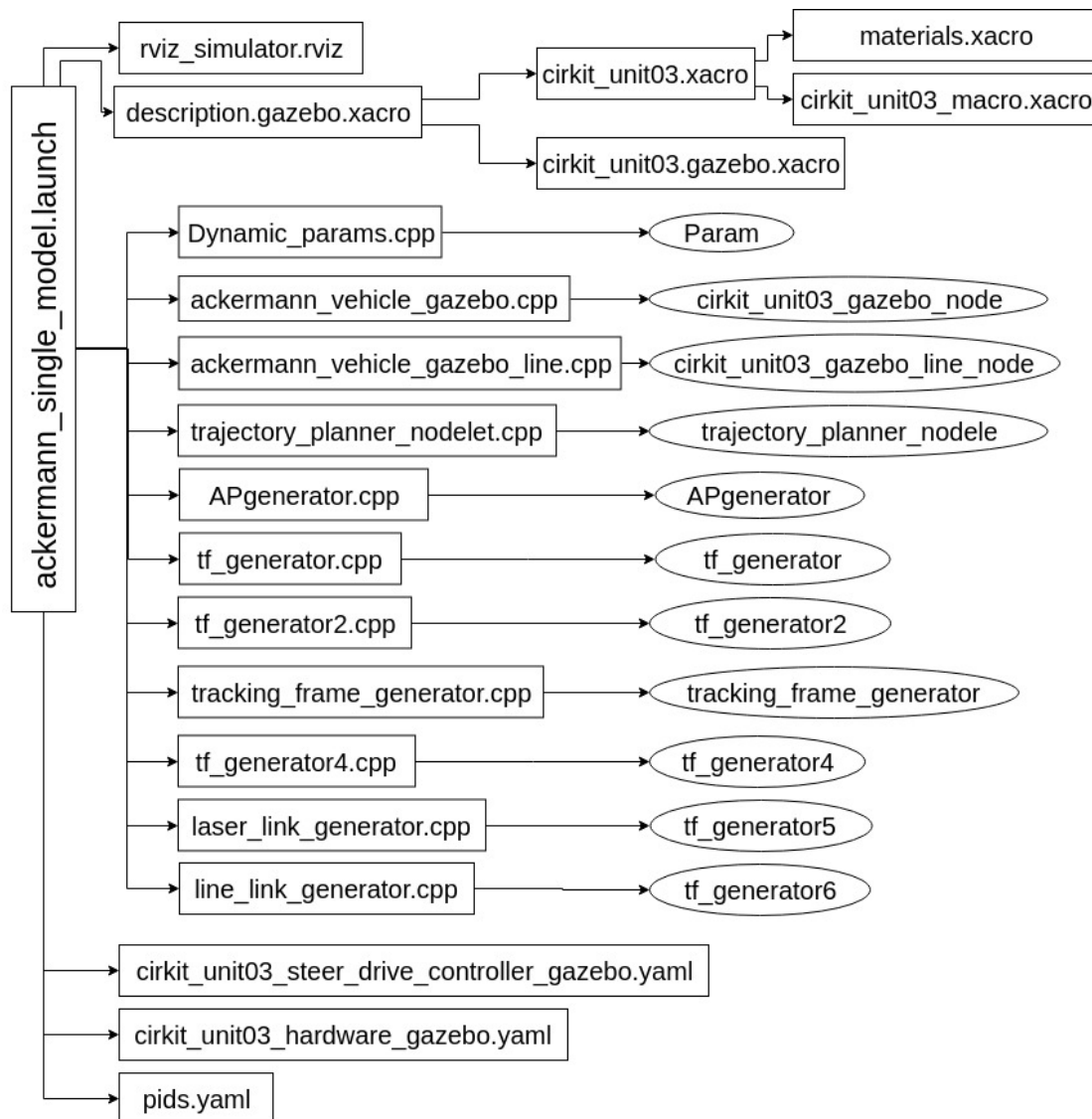


Figura 4.3: Diagrama do ficheiro `ackermann_single_model.launch` lançado para cada veículo

#### 4.1.2 Descrição dos ficheiros do algoritmo

Neste subcapítulo serão descritos os ficheiros pertencentes ao algoritmo de simulação auxiliado pelo diagrama da figura 4.3

##### Ficheiro Rviz

- **rviz\_simulator.rviz** - Configurador do Rviz com os tópicos que utiliza alterados com a ferramenta `remap` para acederem à namespace correta;

### Ficheiros Xacro

- **description.gazebo.xacro** - Ficheiro responsável por configurar o Gazebo, lançando os ficheiros `cirkit_unit03.xacro` e `cirkit_unit03.gazebo.xacro` e executando o *plugin* para o ROS Control dentro do namespace definido;
- **cirkit\_unit03.xacro** - Inclui os ficheiros `materials.xacro` e `cirkit_unit03_macro.xacro` e recolhe e configura as *meshes* necessárias;
- **cirkit\_unit03.gazebo.xacro** - Criação dos *plugins* referentes aos LIDARs com os tópicos `simulator_laser_scan` e `line_laser_scan`; Este LIDARs são configurados em detalhe como o alcance, resolução, ângulos e taxa de atualização.
- **materials.xacro** - Informação sobre diferentes cores;
- **cirkit\_unit03\_macro.xacro** - Contém as dimensões do veículo e as macros das rodas;

### Ficheiros Yaml

- **cirkit\_unit03\_steer\_drive\_controller\_gazebo.yaml** - Dados sobre os controladores `joint_state_controller` e `steer_drive_controller`;
- **cirkit\_unit03\_hardware\_gazebo.yaml** - Configuração do parâmetro `steer_bot_hardware_gazebo`;
- **pids.yaml** - Configuração do parâmetro `gains`;

### Ficheiros C++

Sendo que os nós são executáveis de ficheiros `cpp`, cada um destes ficheiros serão explicados do seguinte modo:

- **exemplo.cpp** - [Nó (executável do exemplo.cpp)] - Descrição do exemplo.

No subcapítulo seguinte estes nós serão descritos com base nos tópicos que publicam e subscrevem.

- **Dynamic\_params.cpp** - [Param] - Ficheiro responsável por publicar os parâmetros dinâmicos da simulação;
- **ackermann\_vehicle\_gazebo.cpp** - [cirkit\_unit03\_gazebo\_node] - Ficheiro responsável por converter o `laser_scan` de objetos em `Point Clouds`;
- **ackermann\_vehicle\_gazebo\_line.cpp** - [cirkit\_unit03\_gazebo\_line\_node] - Ficheiro responsável por converter o `laser_scan` da linha central em `Point Clouds`;
- **trajectory\_planner\_nodelet.cpp** - [trajectory\_planner\_nodelet] - Ficheiro principal do planeamento de trajetórias. Este é responsável por calcular a trajetória a realizar. No lançamento deste nó o ficheiro é alterado com a ferramenta `remap` de modo a permitir que subscreva e publique dentro do namespace desejado;
- **APgenerator.cpp** - [APgenerator] - Gerador do ponto atrator. Este ficheiro, pelas mesmas razões, também é alterado com a ferramenta `remap`;



- **RetrieveInfo.cpp** - [RetrieveInfo] - Subscritor de vários tópicos de modo a recolher informação necessária para a análise do algoritmo e gerador de ficheiros csv;
- **tf\_generator.cpp** - [tf\_generator] - Gera uma *frame* mais alta para publicar a origem das trajetórias;
- **tf\_generator2.cpp** - [tf\_generator2] - Publicador das tfs entre `"/map"` (`free_space_detection`) e `"/world"` (`trajectory_planner_nodelet`);
- **tracking\_frame\_generator.cpp** - [tracking\_frame\_generator] - Gera uma *frame* mais alta que a *frame* do veículo para publicar a Point Cloud ao obstáculo mtt;
- **tf\_generator4.cpp** - [tf\_generator4] - Gerador TF entre `"/map"` e `"/odom"`;
- **laser\_link\_generator.cpp** - [tf\_generator5] - Gerador TF entre `"/world"` e `"/laser_link"`;
- **line\_link\_generator.cpp** - [tf\_generator6] - Gerador TF entre `"/world"` e `"/line_link"`;

## 4.2 Nós e Tópicos

Nesta secção descrevem-se os nós mais importantes da simulação recorrendo à sua visualização, de acordo com o *layout* da ferramenta `rqt_graph`.

- **Gazebo** - Nó principal do Gazebo (ver figura 4.4);

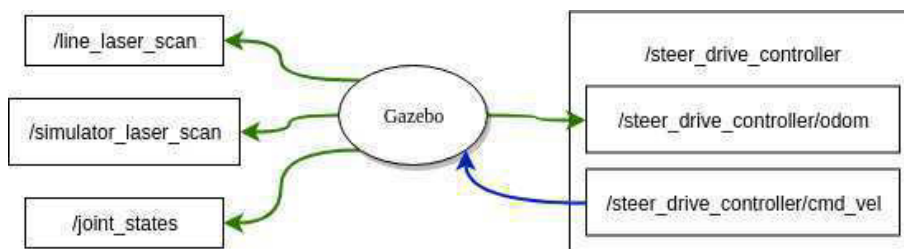


Figura 4.4: Arquitetura do nó Gazebo

- **cirkit\_unit03\_gazebo** - Nó que recebe um laser scan dos obstáculos e a converte numa PointCloud (ver figura 4.5);



Figura 4.5: Arquitetura do nó cirkit\_unit03\_gazebo

- **cirkit\_unit03\_gazebo\_line** - Nó que recebe um laser scan da linha central e a converte numa PointCloud (ver figura 4.6);

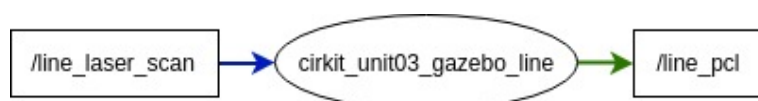


Figura 4.6: Arquitetura do nó cirkit\_unit03\_gazebo\_line

- **APgenerator** - Publicador do ponto atrator que subscrive aos estados dos modelos do Gazebo e à PointCloud da linha central e publica a posição deste ponto para influenciar o planeamento da trajetória e ser publicado na aplicação Rviz através de marcadores (ver figura 4.7);

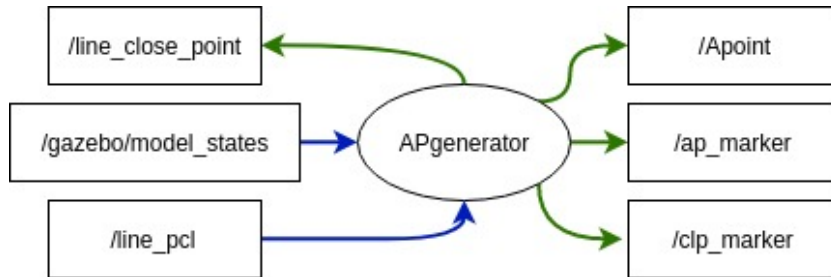


Figura 4.7: Arquitetura do nó APgenerator

- **RetrieveInfo** - Nó responsável pela aquisição de dados durante as simulações e posteriormente pela escrita destes dados em ficheiros csv (ver figura 4.8);

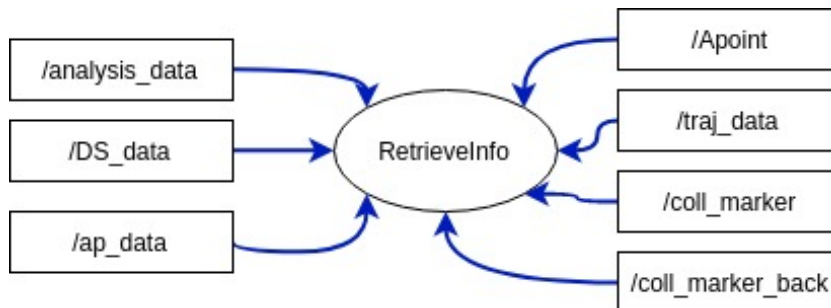


Figura 4.8: Arquitetura do nó RetrieveInfo

- **Rviz** - Nó da ferramenta Rviz que permite visualizar o cálculo das trajetórias, as nuvens de pontos e a deteção de obstáculos (ver figura 4.9);

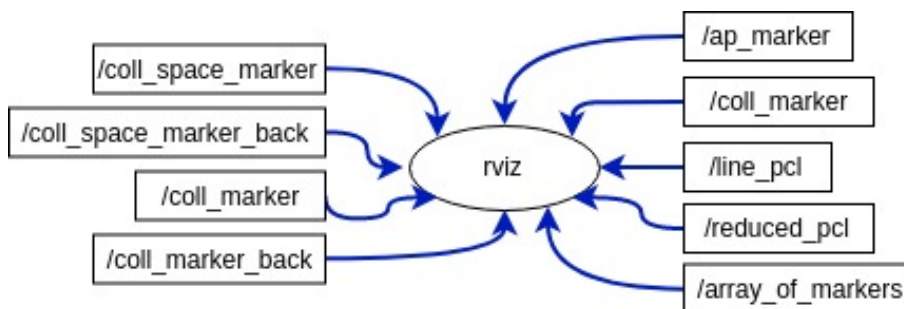


Figura 4.9: Arquitetura do nó Rviz

- **trajectory\_planner\_nodelet** - Nó do planeador de trajetórias que publica no tópico `/cmd_vel` a informação necessária a ser enviada ao simulador Gazebo (ver figura 4.10);

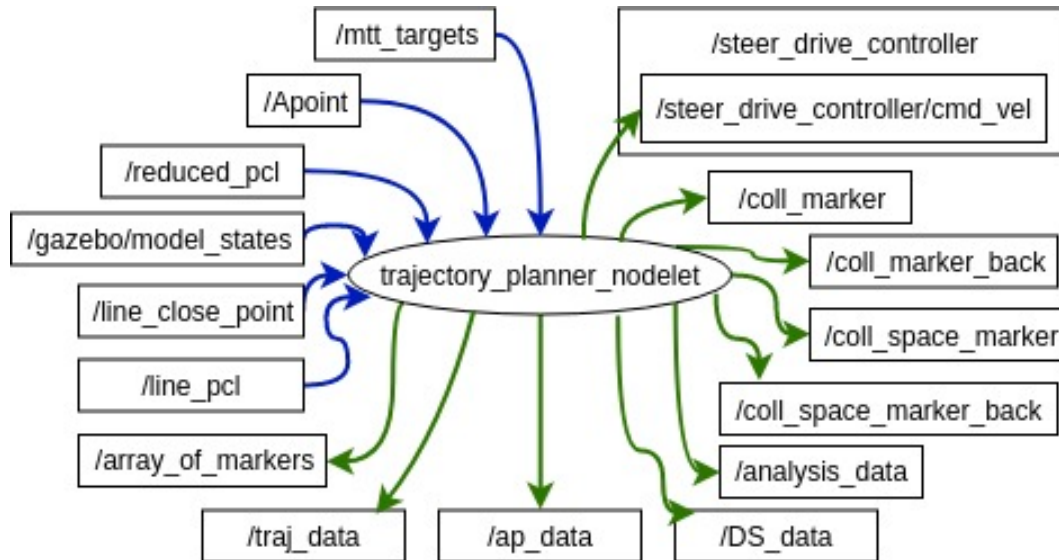


Figura 4.10: Arquitetura do nó `trajectory_planner_nodelet`

### 4.3 Limitações da infraestrutura inicial

Esta dissertação teve como base um ambiente de simulação iniciado por outros trabalhos, nomeadamente Joel Pereira [6] e Ricardo Silva [7]. Este ambiente de simulação, apesar de bastante promissor na simulação de um veículo autónomo, apresentava alguns problemas quando o objetivo passa por inicializar mais do que um veículo, tendo sido necessário fazer algumas alterações aos ficheiros do projeto.

### 4.4 Alterações ao programa existente

Tendo esta dissertação como base o projeto desenvolvido por Ricardo Silva [7], antes de ser possível realizar a simulação corretamente, foi necessário fazer alguns ajustes aos ficheiros existentes.

A primeira alteração a ser realizada teve como objetivo a atualização do código da versão Kinetic para a versão Melodic do ROS. Só após estas alterações é que foi possível compilar e executar a simulação, tendo sido ainda necessário alterar o mapa (figura 4.11) para uma versão mais recente [36]. Esta alteração gerou problemas sendo que alguns detalhes do código do projeto eram baseados em valores fixos medidos do mapa original. Após esta correção e de modo a simplificar toda a ramificação de ficheiros presente, foi criada uma launch file, que agrupa todos os outros ficheiros launch e os seus conteúdos, com o objetivo de simplificar a compreensão e permitir uma maior facilidade em alterações futuras.

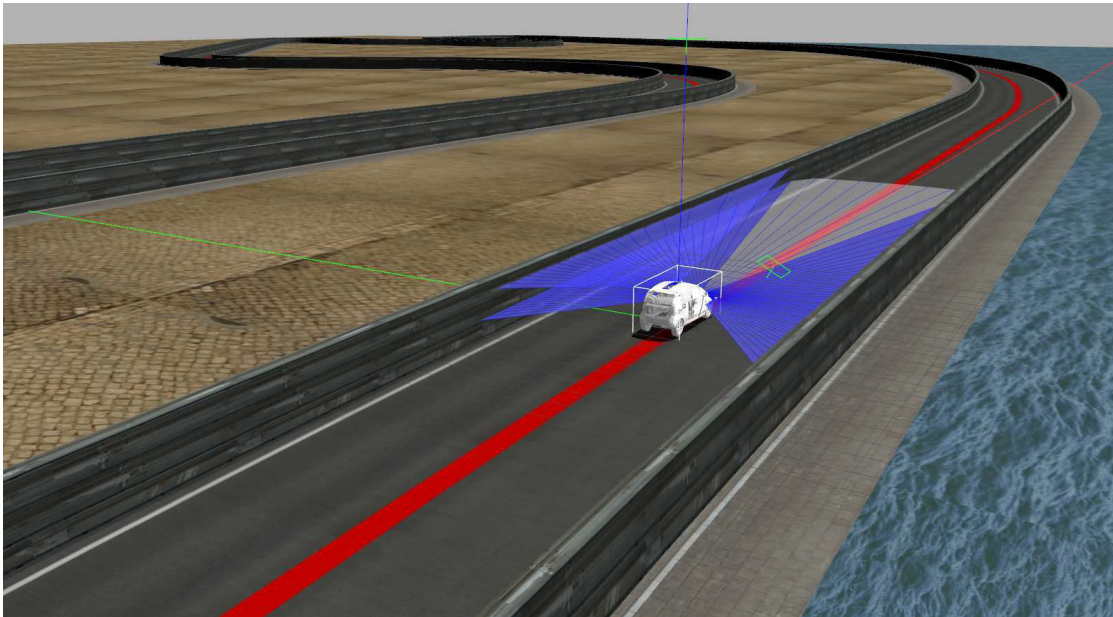
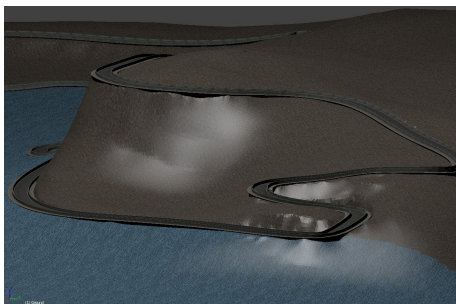


Figura 4.11: Simulação no mapa Monaco

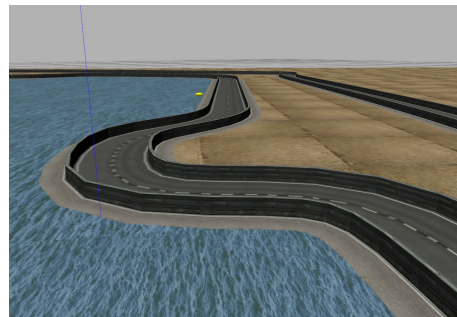
## 4.5 Mapa de Simulação

Não existindo percepção visual na estrada no atual planeamento das trajetórias, é importante utilizar um mapa que permita a utilização dos sensores LIDAR na deteção dos limite da estrada. Para este fim foi utilizado um mapa desenvolvido por Alvaro Villamil [37] que consiste no percurso de Fórmula1 da cidade do Mónaco (figura 4.12).

Este mapa contém como alvos de colisões (objetos reais que interagem com os modelos) a pista e as paredes laterais situadas junto aos limites da faixa de rodagem. Optou-se por utilizar a versão plana deste mapa, visto que a utilização de elevações da pista implicava um custo elevado de processamento gráfico não necessário para os objetivos desta dissertação, tendo apenas mantido as texturas do terreno.



(a) Mapa com elevações



(b) Mapa sem elevações

Figura 4.12: Diferentes opções do mapa utilizado [37]

## 4.6 Namespaces

Após análise dos projetos sobre múltiplos robôs no Gazebo, ficou claro que para ser possível a coexistência destes robôs na mesma simulação, cada um necessita dos seus próprios nós e tópicos, isto é, é necessário que cada um esteja agrupado e seja distinguível dos restantes. Para este fim, utiliza-se os namespaces, permitindo assim a execução de múltiplos agentes, cada um com o seu namespace.

Deste modo, no ficheiro inicial `ackermann_vehicle_simulator.launch` é feito a clonagem do lançamento do `ackermann_single_model.launch`, apenas necessitando alterar os argumentos iniciais. Com esta abordagem, para utilizar N veículos, não é necessário fazer alterações mais profundas evitando constantes alterações ao código. Optou-se por utilizar estes argumentos iniciais sendo que simplificam o código quando comparados com a utilização da função `remap`.

Estes argumentos consistem no número do carro que posteriormente será convertido no namespace correspondente, permitindo a correta alteração realizada com as funções `remap`, entre outros, na posição inicial x e y e ângulo em relação ao eixo Oz para permitir a correta colocação do modelo em causa no "mundo" do Gazebo, e nos parâmetros dinâmicos correspondentes ao veículo em questão.

```

1  <group ns="h1">
2
3  <include file="$(find cirkuit_unit03_gazebo)/launch/ackermann_single_model
4  .launch">
5    <arg name="car_number" default="1" />
6    <arg name="initX" value="-21" />
7    <arg name="initY" value="-3" />
8    <arg name="initYaw" value="0" />
9  </include>
10
11 <param name="Param/MAX_STEERING_ANGLE" value="41.5" />
12 <param name="Param/MIN_STEERING_ANGLE" value="20" />
13 <param name="Param/NUM_NODES" value="10" />
14 <param name="Param/NUM_TRAJ" value="15" />
15 <param name="Param/TRAJ_DENSITY" value="2" />
16
17 <param name="Param/SPEED_REQUIRED" value="3.0" />
18 <param name="Param/SPEED_SAFFETY" value="1.0" />
19
20 <param name="Param/DetectDist" value="20" />
21 <param name="Param/APdistMin" value="5" />
22 <param name="Param/APdistMax" value="10" />
23
24 <param name="Param/W_ADAP" value="0.20" />
25 <param name="Param/W_DAP" value="0.40" />
26 <param name="Param/W_DLO" value="0.90" />
27
28 <param name="Param/LINES" value="true" />
29 <param name="Param/OVERTAKING" value="false" />
30 <param name="Param/DETECTION" value="true" />
31 <param name="Param/AP_right" value="true" />
32 <param name="Param/AP_left" value="false" />
33 <param name="Param/Vel_Ang" value="false" />
</group>

```

Listagem 4.2: Ficheiro launch básico para um namespace

De modo a criar um segundo veículo no simulador Gazebo apenas é necessário copiar este excerto de código alterando os valores para os desejados, tendo o segundo veículo os seus próprios nós, tópicos, parâmetros referentes ao planeador e uma aplicação Rviz. É importante referir que ao criar um segundo veículo é obrigatório alterar o nome do grupo ns, o valor do argumento car\_number e a sua posição inicial de modo a que não exista colisão inicial na simulação.

## 4.7 Alterações ao ficheiro de lançamento do modelo

Tendo agrupado todo o código para o lançamento de um modelo, procedeu-se à alteração dos diversos ficheiros de modo a permitir um correto funcionamento do simulador.

Primeiramente foram adicionados os argumentos namespace e tfpre de modo a permitir a distinção de cada veículo. Tal como foi explicado anteriormente, o argumento namespace é alterado através do valor do argumento car\_number designado no ficheiro principal, enquanto o argumento tfpre é derivado do nome do namespace. De seguida são extraídos os argumentos relativos à posição, para permitir alterar a posição inicial do modelo, visto que dois veículos não podem ser colocados no mesmo local.

Ao lançamento do parâmetro robot\_description, que invoca o ficheiro description.gazebo.xacro, foi adicionado como argumentos a especificação do namespace e tfpre. Com estes torna-se possível alterar o namespace dos plugins e tópicos invocados, nomeadamente as nuvens de pontos que passarão a ter frame\_ids específicos para cada veículo, permitindo que o simulador Gazebo as distribua corretamente e que sejam identificadas pelos nós respetivos. Para o mesmo efeito o base\_frame\_id do steer\_drive\_controller é alterado contendo agora o prefixo tfpre.

De seguida e de modo a permitir a correta colocação dos modelos no "mundo" do Gazebo, os argumentos referentes à posição mencionados anteriormente são especificados no lançamento do nó urdf\_spawnder, sendo aqui também modificado o nome do modelo consoante o namespace, permitindo a sua distinção na interface gráfica do simulador Gazebo.

A ferramenta remap foi utilizada para facilitar as alterações ao namespace, permitindo que todas as referencias a tópicos sejam alteradas para incluir o namespace respetivo. Através desta ferramenta, ao iniciar um nó, todas as instâncias em que o argumento aparece no código são alteradas para o texto desejado (tópico com namespace incluído).

```

1 <node name="circuit" pkg="circuit" type="node" output="screen">
2   <remap from="/scan" to="/$(arg namespace)/scan"/>
3   <remap from="/pcl" to="/$(arg namespace)/pcl"/>
4 </node>
```

Listagem 4.3: Remap de tópicos para ficarem inseridos num namespace específico (excerto presente no ficheiro ackermann\_single\_model.launch)

Com estas alterações finalmente foi possível executar 2 veículos em simultâneo no mesmo ambiente Gazebo. Na figura 4.13 é possível visualizar o rqt\_graph final.

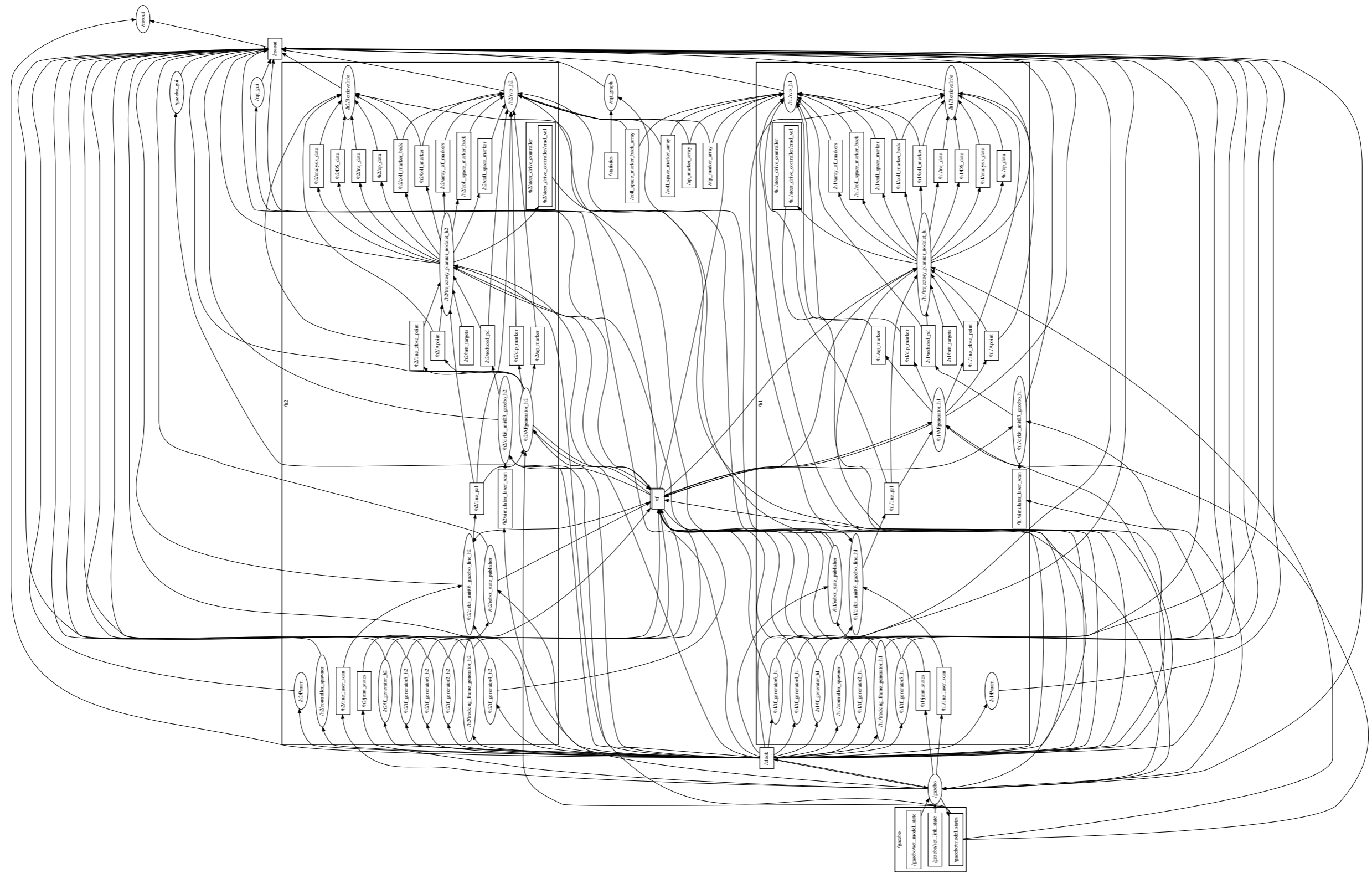


Figura 4.13: Visualização do rqt\_graph durante a simulação de dois veículos





## 4.8 Alterações restantes

Tendo em conta que existia a necessidade de alterar o ponto atrator no Rviz para dar início ao movimento de um veículo, foi alterado o APgenerator de modo a que o ponto atrator fosse definido inicialmente numa posição pré-estabelecida. Esta necessidade da utilização da aplicação Rviz tinha como principal desvantagem a impossibilidade de inicializar todos os veículos ao mesmo tempo, impedindo análises utilizando simulações idênticas. Assim, com esta alteração, a simulação inicia-se ao clicar no botão *play* do Gazebo facilitando a execução da simulação.

Durante testes de colisão frontal entre dois veículos foi detetado que a configuração de colisões do modelo se encontrava com dimensões muito reduzidas não sendo suficientemente visível aos sensores LIDAR. Para corrigir este problema a *hitbox* do veículo foi configurada para ser a combinação de um paralelepípedo e um prisma triangular com as dimensões do ATLASCAR2 tal como representado na figura 4.14, colocados numa posição mais recuada (figura 4.15), evitando deste modo interferir com os dados recolhidos do seu próprio sensor LIDAR.

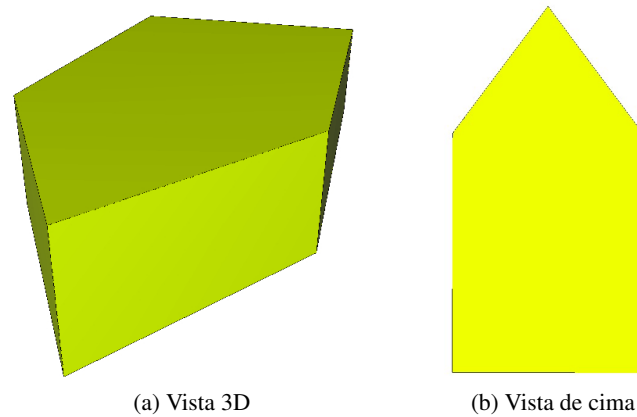


Figura 4.14: Bloco de colisão

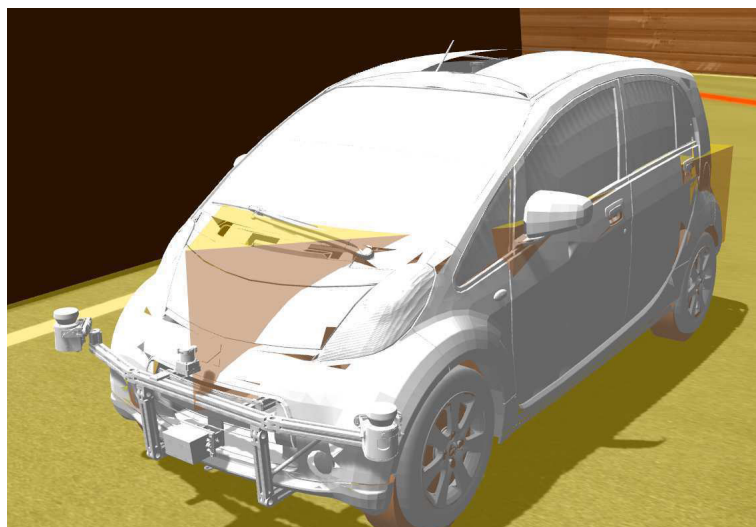


Figura 4.15: Visualização das colisões

Por último e de modo a permitir uma melhor análise da importância de cada parâmetro influente na simulação, foi criado um nó publicador de parâmetros com o nome Param. Este nó está responsável pela publicação constante destes parâmetros sendo possível alterá-los durante a simulação através da ferramenta `rqt_reconfigure` (figura 4.16). Esta opção não se verifica quando estes mesmos parâmetros são inicializados através de um ficheiro `launch`, dado que esta ferramenta apenas deteta parâmetros que são inicializados pelos nós.

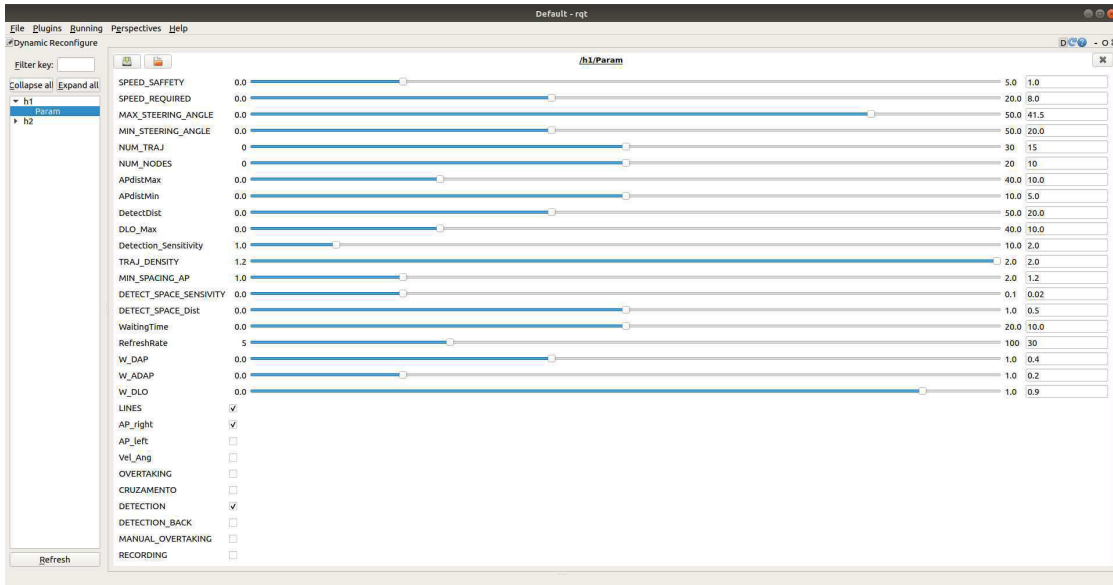


Figura 4.16: `rqt_reconfigure` durante a simulação de um veículo

## Capítulo 5

# Planeamento em manobras-tipo

Com o simulador operacional para a utilização de múltiplos veículos, a próxima tarefa incide na configuração de comportamentos para situações previstas que possam ocorrer durante o percurso do veículo autónomo. Estes comportamentos serão complementos ao planeador geral, e serão ativados consoante dados específicos recebidos pelos sensores.

Neste capítulo serão descritos os passos necessários para a implementação de manobras-tipo, passando pelo ajuste do cálculo da trajetória e os fatores que a influenciam, a problemática da condução pela via da direita e pela criação do ponto atrator. Serão também expostos as manobras-tipo criadas referindo as dificuldades e as suas especificações.

### 5.1 Parâmetros da simulação

De modo a ser possível a alteração da simulação consoante a necessidade, foram criados vários parâmetros que permitem estudar a sua influência no planeamento de trajetórias, sendo possíveis de serem alterados, quer inicialmente através do ficheiro `launch`, quer durante a simulação com a utilização da ferramenta `dynamic_reconfigure`.

- **SPEED\_SAFFETY** - Velocidade mínima que o planeador de trajetórias enviará ao Gazebo através do tópico `/cmd_vel`;
- **SPEED\_REQUIRED** - Velocidade máxima que o planeador de trajetórias enviará ao Gazebo através do tópico `/cmd_vel`;
- **MAX\_STEERING\_ANGLE** - Ângulo máximo do volante;
- **MIN\_STEERING\_ANGLE** - Ângulo mínimo obrigatório, isto é, existirá sempre uma trajetória com, no mínimo, este ângulo;
- **NUM\_TRAJ** - Número de trajetórias a serem geradas para cada lado (direita e esquerda do veículo);
- **NUM\_NODES** - Número de nós de cada trajetória calculados;
- **DetectDist** - Distância de deteção de obstáculos (utilizado nas manobras tipo);
- **APdistMax** - Distância máxima do Ponto Atrator, isto é, a distância deste ao referencial do veículo não ultrapassará este valor;

- **APdistMin** - Distância mínima do Ponto Atrator, isto é, a distância deste ao referencial do veículo será, no mínimo, este valor;
- **MIN\_SPACING\_AP** - Espaçamento mínimo entre os valores máximo e mínimo da distância do ponto atrator, de modo a permitir uma quantidade suficiente de pontos no cálculo da posição do ponto atrator.
- **DLO\_Max** - Distância de referência de influência dos obstáculos, utilizado na normalização do valor DLO.
- **Detection\_Sensitivity** - Número de pontos detetados pelos sensores necessários para ativar manobras.
- **TRAJ\_DENSITY** - Densidade das trajetórias. Quanto mais próximo do valor 1, maior a uniformidade das trajetórias (em relação ao ângulo de cada uma).
- **DETECT\_SPACE\_SENSIVITY** - Distância de segurança da zona de detecção dos objetos, utilizada para evitar falsos positivos.
- **DETECT\_SPACE\_Dist** - distância máxima no eixo Oy utilizado para restringir os pontos utilizados no espaço de detecção. Deste modo pontos muito distantes (obstáculos) dos restantes pontos detetados (limite da estrada) não são considerados na criação da zona de detecção.
- **WaitingTime** - Tempo de espera antes de voltar para a via da direita após uma ultrapassagem.
- **RefreshRate** - Número máximo de publicações do nó por segundo, evitando sobrecarga do sistema.
- **LINES** - Parâmetro booleano que ativa ou desativa a linha central como obstáculo;
- **OVERTAKING** - Parâmetro booleano que controla a ação de ultrapassagem;
- **DETECTION** - Parâmetro booleano que ativa a detecção frontal de obstáculos;
- **DETECTION\_BACK** - Parâmetro booleano que ativa a detecção traseira de obstáculos;
- **RECORDING** - Parâmetro booleano que escreve em ficheiros csv os dados necessários para a análise do algoritmo.
- **AP\_right** - Parâmetro booleano que coloca o ponto atrator na via da direita;
- **AP\_left** - Parâmetro booleano que coloca o ponto atrator na via da esquerda;
- **Vel\_Ang** - Parâmetro booleano que ativa a redução de velocidade do veículo consoante o ângulo da trajetória;

## 5.2 Cálculo da trajetória

Durante o percurso do ATLASCAR2, utilizando a nuvem de pontos obtidas pelos sensores LIDAR e pela utilização de um ponto atrator, o cálculo da trajetória é influenciado por quatro fatores adaptados do projeto anterior [7]: A Distância ao Ponto Atrator (DAP); a Diferença Angular ao Ponto Atrator (ADAP); a Distância Mínima aos Obstáculos (DLO); e o Espaço Livre (FS).

### 5.2.1 Distância ao ponto atrator (DAP)

No cálculo da distância ao ponto atrator (equação 5.1) cada trajetória potencial é analisada, verificando para cada nó da trajetória (figura 5.1.(a)), a distância a que este se encontra do ponto atrator.

Após analisados todos os nós de uma trajetória, para cada uma destas trajetórias é armazenado o seu nó mais próximo ao ponto atrator e a sua pontuação DAP respectiva (figura 5.1.(b)).

$$DAP = \sqrt{(node_x - AP_x)^2 + (node_y - AP_y)^2} \quad (5.1)$$

No excerto de código 5.1 é possível visualizar o processo de seleção do nó mais próximo de cada trajetória e a sua respetiva pontuação DAP.

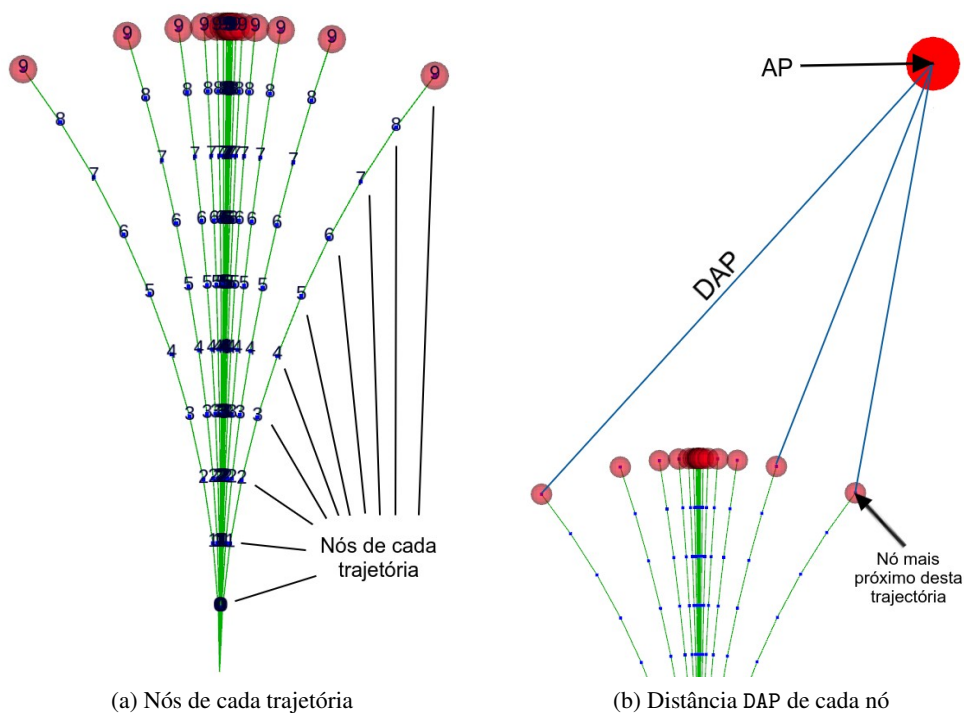


Figura 5.1: Demonstração visual do DAP com recurso ao Rviz

```

1 //verifica todos os nós pelo mais próximo
2 for (size_t i = 0; i < trajectory->x.size(); ++i)
3 {
4     //distância do nó ao AP
5     double DAP_prev = sqrt(pow(trajectory->x[i] - AP.x, 2) + pow(trajectory->
6     y[i] - AP.y, 2));
7
8     if (DAP_prev < trajectory->score.DAP) //se for mais próximo
9     {
10        trajectory->score.DAP = DAP_prev; //nova pontuação
11        trajectory->closest_node = i; //novo nó mais próximo
12    }
13 }

```

Listagem 5.1: Cálculo do DAP para uma trajetória

Na equação 5.2, com o objetivo de normalizar o valor DAP, é executada a comparação com o valor máximo admissível à distância do Ponto Atrator, valor dado pelo parâmetro  $APdistMax$  alterável durante a simulação, sendo que, caso o valor DAP seja superior ao valor máximo definido, este passa a valer 0, isto é, deixa de influenciar o cálculo da trajetória, utilizando para tal a função  $\max()$  como é possível verificar na linha de código 5.2.

$$DAP_{norm} = 1 - \frac{DAP}{APdistMax} \quad (5.2)$$

```
1 // normalização do valor DAP
2 vt[i]->score.DAPnorm = max(0.0 , (1 - (vt[i]->score.DAP) / APdistMax));
```

Listagem 5.2: Normalização do valor DAP da trajetória em questão

### 5.2.2 Diferença Angular ao Ponto Atrator (ADAP)

Com a obtenção do nó mais próximo do ponto atrator para cada trajetória, é calculada a diferença angular entre o ângulo de cada um destes nós com o ângulo do ponto atrator. Para este cálculo é fornecido a trajetória em análise, o nó mais próximo do ponto atrator extraído do cálculo do DAP e a posição do ponto atrator relativamente ao referencial do veículo.

Calculando em radianos o ângulo do nó em análise ( $ang\_node$ ) representado na figura 5.2.(a) e o ângulo do ponto atrator ( $ang\_AP$ ) representado na figura 5.2.(b), é calculada a diferença absoluta entre os dois (equação 5.3), obtendo o valor ADAP final desta trajetória, como demonstrado na listagem 5.3.

$$ADAP = |ang\_node - ang\_AP| \quad (5.3)$$

```
1 double ang_AP = atan2(AP.y, AP.x); // ângulo do ponto atrator
2
3 double ang_node = atan2(trajjectory->y[i], trajectory->x[i]); // ângulo do nó
4
5 double adap = abs(ang_node - ang_AP); //resultado ADAP da trajetória
```

Listagem 5.3: Cálculo do ADAP para uma trajetória

Tal como acontece com o valor DAP, a pontuação ADAP de cada trajetória é normalizada de acordo com a equação 5.4 em função do valor  $\pi$ .

$$ADAP_{norm} = 1 - \frac{ADAP}{\pi} \quad (5.4)$$

```
1 vt[i]->score.ADAPnorm = max(0.0, (1 - (vt[i]->score.ADAP / (M_PI))));
```

Listagem 5.4: Normalização do valor ADAP da trajetória em análise

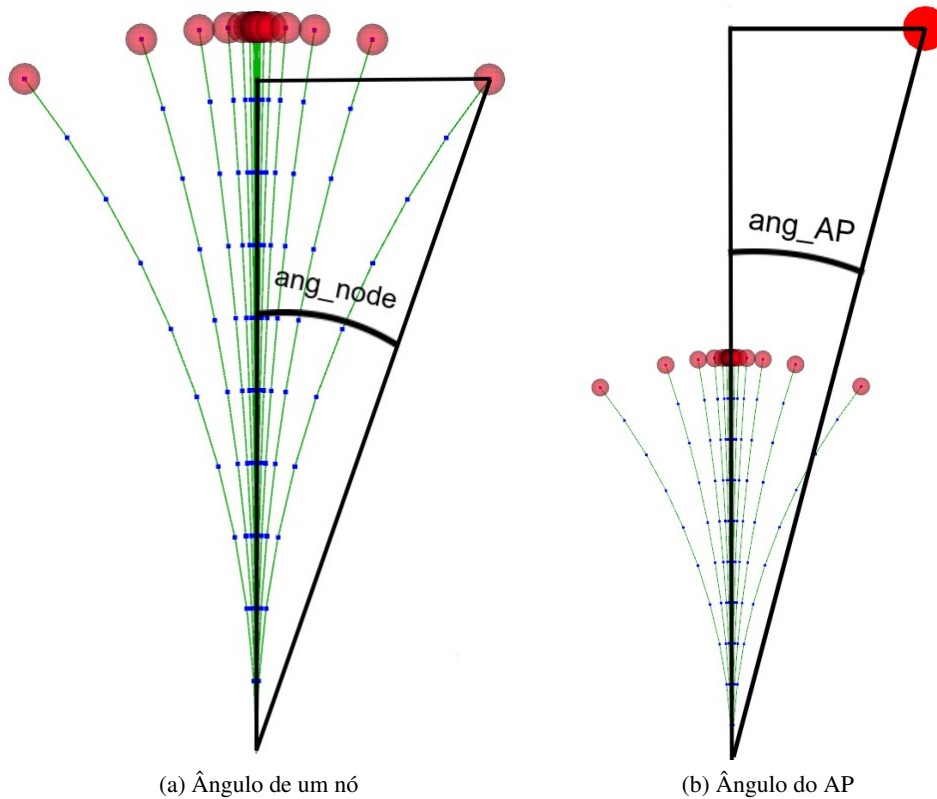


Figura 5.2: Demonstração visual do ADAP com recurso ao Rviz

### 5.2.3 Distância Mínima aos Obstáculos (DLO)

De modo a executar um cálculo preventivo, é considerado um retângulo com as dimensões do ATLASCAR2 centrado no nó em análise de modo a refletir as dimensões reais do veículo, permitindo medir a distância mínima a que este se encontrará dos obstáculos detetados pelos sensores LIDAR.

Apenas são analisados os nós da trajetória que se encontram entre a posição do veículo e o nó mais próximo do ponto atrator, tendo em conta o facto de este último ponto ser o objetivo atual da trajetória.

Assim, para cada nó, é medida a distância entre os vértices do seu retângulo com cada obstáculo através da equação 5.5, sendo o valor DLO final da trajetória o menor resultado entre todos os nós.

$$DLO = \sqrt{(vertice_x - obstaculo_x)^2 + (vertice_y - obstaculo_y)^2} \quad (5.5)$$

Na figura 5.3 é possível visualizar estes retângulos para cada nó de uma única trajetória (escolhida durante a captura da imagem através do Rviz de modo a uma melhor compreensão), e a distância DLO de cada vértice ao ponto mais próximo. Na figura 5.3.(a) o cálculo da distância DLO considera a linha central (pontos vermelhos) enquanto que na figura 5.3.(b) é considerado a parede lateral direita (pontos pretos).

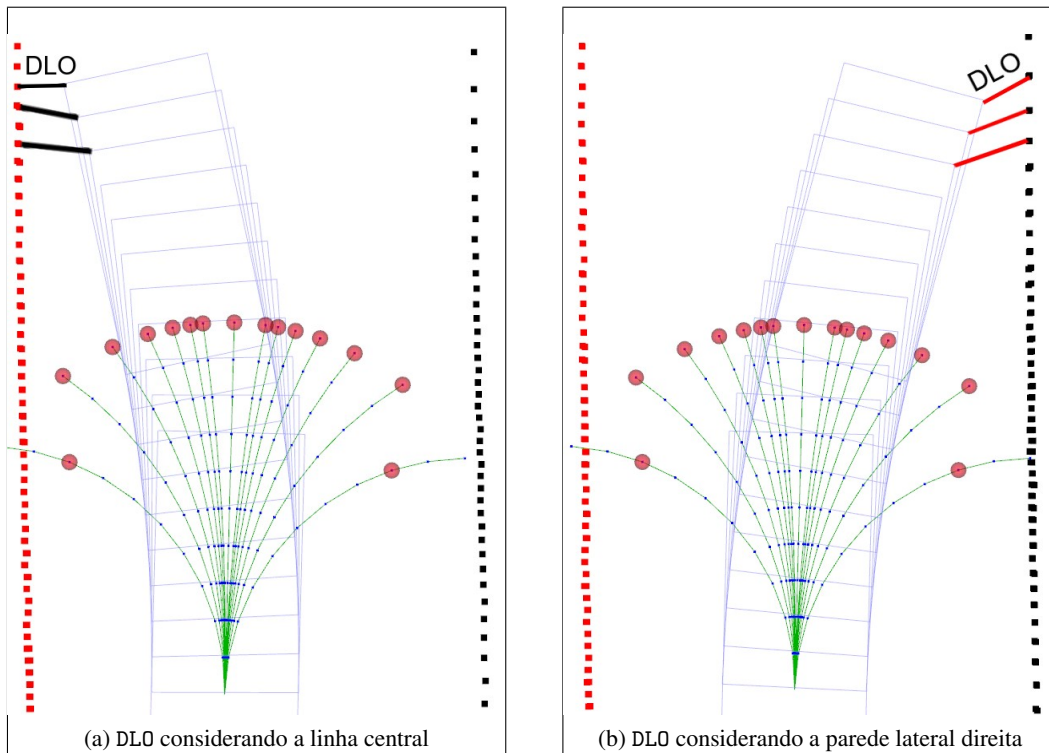


Figura 5.3: Demonstração visual do DLO de uma trajetória com recurso ao Rviz

Na equação 5.6 este valor final é normalizado dividindo-o pelo valor máximo definido inicialmente através do parâmetro  $DLO\_Max$ ;

$$DLO_{norm} = \frac{DLO}{DLO\_Max} \quad (5.6)$$

#### 5.2.4 Espaço livre (FS)

A utilização da variável FS permite anular a pontuação das trajetórias que correspondam a colisões, isto é, trajetórias cujos nós não se encontrem contidos no espaço navegável. Assim, quando esta condição é verificada, a variável FS iguala a 0, invalidando a trajetória em questão.

#### 5.2.5 Cálculo final da trajetória

De modo a permitir a alteração dos pesos das variáveis referidas anteriormente que influenciam o cálculo da pontuação de cada trajetória, existem três parâmetros iniciais (com valores entre 0 e 1) que serão multiplicados pela variável respetiva, sendo fornecidos através do ficheiro `launch`, existindo também a possibilidade da sua alteração durante a simulação através da ferramenta `dynamic_reconfigure`. Estes parâmetros são:

- **W\_DAP** - Distância ao ponto atrator.
- **W\_ADAP** - Diferença angular ao ponto atrator.
- **W\_DLO** - Distância Mínima aos Obstáculos.



Assim, com a utilização destes parâmetros é possível não só alterar a ponderação de cada variável para diferentes veículos, como também analisar a importância que cada um destes tem na pontuação de cada trajetória, e posteriormente no comportamento dos veículos em simulação.

Finalmente a pontuação final da trajetória é calculada através da equação 5.7:

$$Ptraj = (W\_DAP \cdot DAPnorm + W\_ADAP \cdot ADAPnorm + W\_DLO \cdot DLOnorm) \cdot FS \quad (5.7)$$

### 5.3 Disposição das diferentes trajetórias

As curvas realizadas pelo veículo autónomo durante o percurso devem ter em conta a sua velocidade, sendo que em velocidades elevadas, uma curva apertada proporciona uma manobra perigosa e possivelmente com consequências graves.

Deste modo, através da equação 5.8, foram implementadas condições de modo a restringir o ângulo máximo possível de ser realizado em cada momento, tendo em conta os ângulos máximo e mínimo definidos pelos parâmetros MAX\_STEERING\_ANGLE ( $\alpha_{max}$ ) e MIN\_STEERING\_ANGLE ( $\alpha_{min}$ ) respetivamente, a velocidade extraída e calculada através da informação publicada pelo Gazebo no tópico /gazebo/model\_states (velocidade\_atual) e a velocidade máxima definida no parâmetro SPEED\_REQUIRED.

$$\alpha_{max} = \alpha_{max} - (\alpha_{max} - \alpha_{min}) \cdot \frac{velocidade\_atual}{SPEED\_REQUIRED} \quad (5.8)$$

Tendo em conta que as trajetórias estavam espaçadas de forma uniforme, foi criada a necessidade de obter uma maior densidade de trajetórias na orientação do veículo. A existência de uma maior densidade de trajetórias permitiria ao veículo executar pequenas correções na sua orientação, evitando deste modo curvas acentuadas por consequência da inexistência de uma trajetória intermédia.

Para tal, foi calculado o valor `iter_value` através da equação 5.9 utilizando o ângulo máximo calculado anteriormente ( $\alpha_{max}$ ) e os parâmetros NUM\_TRAJ que indica o número de trajetórias desejadas a calcular para cada direção (esquerda e direita) e TRAJ\_DENSITY que indica a densidade desejada das trajetórias. Com a redução do valor do parâmetro TRAJ\_DENSITY, mais uniforme será o espaçamento das diversas trajetórias.

$$iter\_value = \frac{\alpha_{max}}{TRAJ\_DENSITY^{NUM\_TRAJ}} \quad (5.9)$$

Além destas trajetórias também será analisada uma última trajetória correspondente ao movimento frontal.

Esta equação permite obter o espaçamento angular que será multiplicado pelo seu valor a cada iteração da geração das trajetórias, como exemplificado no excerto de código 5.5 e na figura 5.4.

```

1 double iter_value = max_angle / (pow(TRAJ_DENSITY, NUM_TRAJ));
2
3 while (i < max_angle) {
4
5     vector<double> v_a1;
6     vector<double> v_arcl;
7     for (int j = 0; j < NUM_NODES; ++j)
8     {

```

```

9     v_a1.push_back(M_PI / 180. * i);
10    v_arc1.push_back(max_dist / NUM_NODES);
11    }
12    manage_vt->update_trajectory(v_a1, v_arc1, v_a1, num_trajec); //gera uma
13    nova trajetória
14
15    num_trajec = num_trajec + 1;
16
17    //angulo na outra direcao
18    vector<double> v_a2;
19    vector<double> v_arc2;
20    for (int j = 0; j < NUM_NODES; ++j)
21    {
22        v_a2.push_back(M_PI / 180. * (-i));
23        v_arc2.push_back(max_dist / NUM_NODES);
24    }
25    manage_vt->update_trajectory(v_a2, v_arc2, v_a2, num_trajec); //gera uma
26    nova trajetória
27
28    num_trajec = num_trajec + 1;
29
30    i = i * TRAJ_DENSITY; //multiplicação do ângulo
31 }

```

Listagem 5.5: Criação das trajetórias

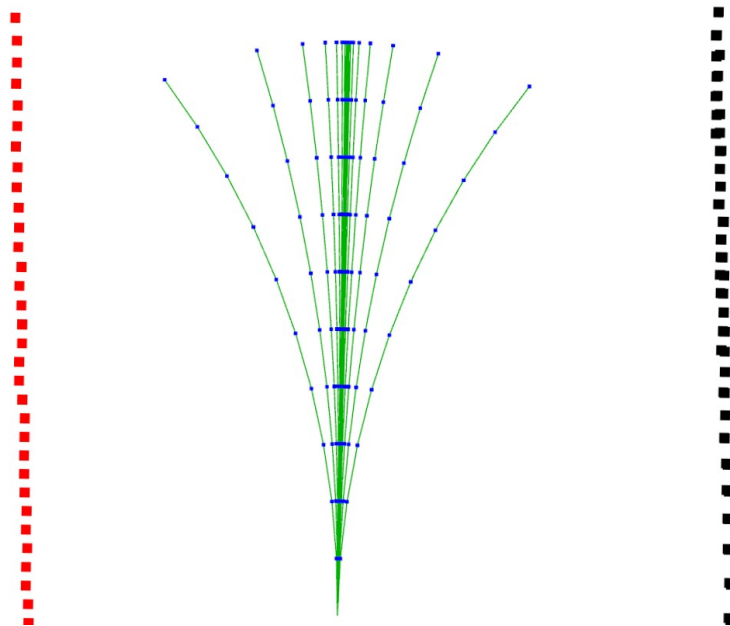


Figura 5.4: Disposição das trajetórias potenciais visualizadas no Rviz

## 5.4 Cálculo da velocidade dependente do ângulo

Durante o percurso do veículo, em casos gerais, dependendo da magnitude da variação da direção pela nova trajetória, é importante que a velocidade seja compensada para assegurar a sua segurança, como por exemplo no caso de uma curva apertada. Para tal e aproveitando o cálculo

já desenvolvido por trabalhos anteriores, este foi reformulado (equação 5.10) utilizando o ângulo em valor absoluto da nova trajetória ( $\alpha$ ). Neste cálculo é utilizado o ângulo máximo calculado anteriormente `max_angle` ( $\alpha_{max}$ ) e os parâmetros `SPEED_REQUIRED` e `SPEED_SAFETY`.

$$vel = SPEED\_REQUIRED - \frac{(SPEED\_REQUIRED - SPEED\_SAFFETY)}{\alpha_{max} \cdot \frac{\pi}{180} \cdot |\alpha|} \quad (5.10)$$

Este cálculo está dependente do parâmetro `Vel_Ang` que é possível de desativar caso não seja desejado esta compensação.

## 5.5 Condução pela via da direita

De modo a ser possível desenvolver e testar diferentes situações durante a condução, é necessário que o veículo circule pela via da direita.

Inicialmente foi analisada a solução proposta pelo projeto anterior [7] onde era utilizado um novo parâmetro `CL` (Central line). Este parâmetro (configurável entre 0 e 1) era multiplicado pelo resultado final de cada trajetória que tivesse um ângulo maior ou igual a 0 (trajetórias de curvas para a esquerda ou em linha reta). Assim, sendo o valor deste parâmetro menor ou igual a 1, iria diminuir a pontuação das trajetórias referidas, resultando em pontuações superiores para as trajetórias que curvavam para a direita.

Esta abordagem implica que o veículo tenha um comportamento instável, visto que circulava evitando a colisão com a parede da direita desviando-se dela (parâmetro `DLO`) e quando a orientação do veículo já não resulta numa colisão, o veículo voltava a curvar para a direita, e assim sucessivamente. Como esta abordagem implicava um resultado indesejado, demonstrando um comportamento oscilante, foi concluído que seria necessário uma nova solução.

Assim, e aproveitando a nuvem de pontos gerada no centro da estrada (`/line_pc1`), esta foi adicionada como um obstáculo (figura 5.5 e 5.6), obrigando o veículo a circular entre a linha central e a parede mais próxima (que no caso geral é a direita). Com esta solução elimina-se o problema anterior, sendo possível realizar manobras-tipo ao deixar de subscrever temporariamente à nuvem de pontos da linha central.

É de referir que, apesar deste novo obstáculo, continua a ser possível a distinção entre as duas nuvens de pontos existentes (a nuvem de pontos detetadas pelos sensores e a nuvem de pontos gerada para a linha central).

Nas figuras seguintes é possível visualizar os pontos considerados como obstáculos (pontos amarelos), isto é, pontos que poderão causar colisões das diferentes trajetórias existentes. A linha vermelha representa a linha central e as linhas pretas representam os limites da estrada (paredes laterais). Assim na figura 5.5.(a) é representada a linha central e na figura 5.5.(b) os obstáculos considerados desta linha. Na figura 5.6 é representado os obstáculos totais (pontos à esquerda representam a linha central e os pontos da direita representam a parede lateral direita).

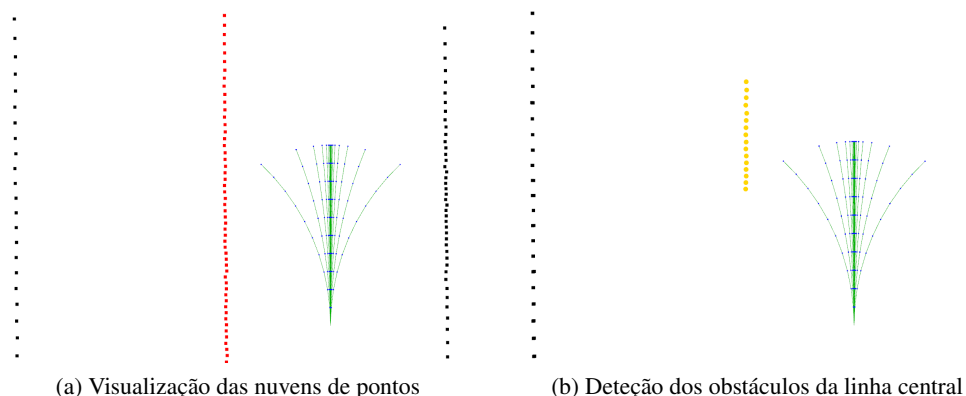


Figura 5.5: Detecção de obstáculos (pontos amarelos) da linha central (pontos vermelhos)

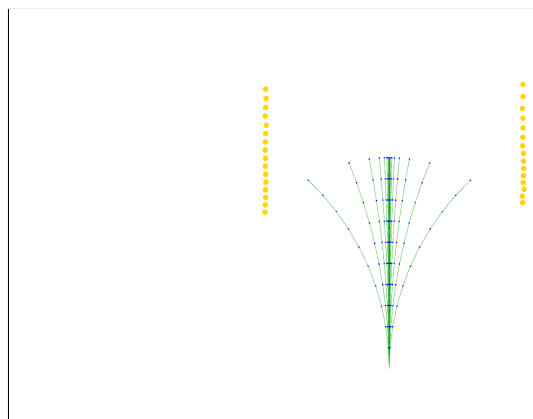


Figura 5.6: Detecção de obstáculos (pontos amarelos) da linha central e parede lateral

## 5.6 Ponto Atrator

Outro problema evidente do movimento instável do veículo surgia devido à aplicação do ponto atrator à sua frente, melhorando a pontuação das trajetórias que seguiam a orientação deste veículo. Esta utilização do ponto atrator obteve bons resultados em situações normais em estradas simples, mas caso algo imprevisto acontecesse (obstáculo na via, necessidade de desviar de outro veículo, etc.) o veículo em análise continuaria a dar prioridade a trajetórias na sua própria direção, mesmo que implicasse uma aproximação indesejada aos limites da estrada, como é possível verificar na figura 5.7. Na figura 5.7.b verifica-se que o ponto atrator continua colocado na orientação do veículo e não na orientação desejada (linha verde) de modo a evitar a colisão iminente com a parede lateral. Deste modo, apesar da trajetória estar correta devido ao peso elevado do valor DLO, as trajetórias na direção do veículo têm pontuações demasiado elevadas devido ao ponto atrator ainda se encontrar na sua direção.

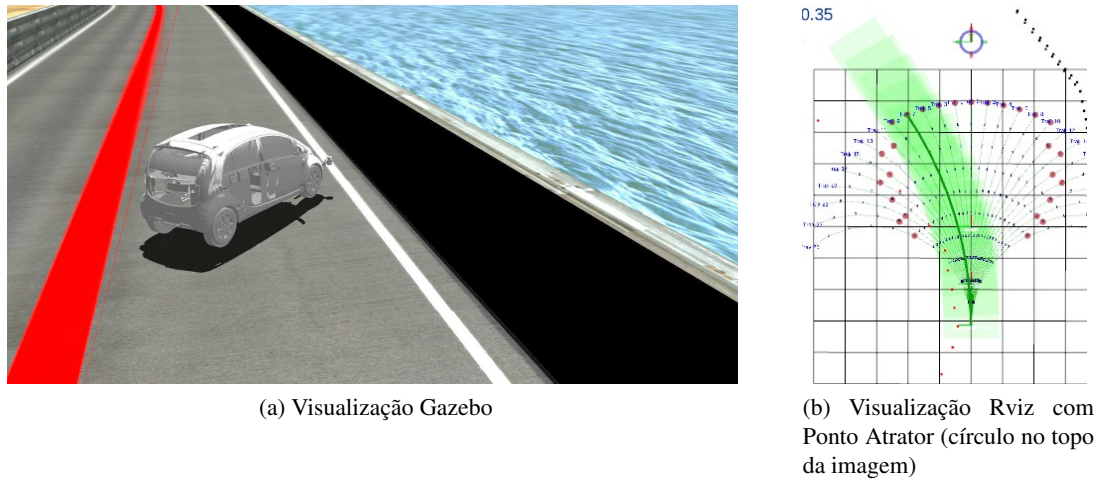


Figura 5.7: Colisão iminente com o limite da via

### 5.6.1 Ponto Atrator utilizando Waypoits

Numa primeira tentativa de criar um ponto atrator que refletisse a sua utilização no ambiente real, foi criada uma lista de *waypoints* ao longo do pista com o objetivo de simular os dados recebidos por um GPS. Para tal foi executada uma simulação onde o veículo percorre o trajeto completo gravando a sua posição ao longo do tempo através da análise do tópico `/gazebo/model_states`, que fornece informações sobre a posição e orientação face ao referencial global do "mundo" e a velocidade atual do veículo. Devido à grande quantidade de posições recolhidas durante o percurso, estes dados foram reduzidos criando vários ficheiros csv com diferentes intervalos, tendo no final resultado em intervalos de 10, 100 e 1000 posições. Com esta informação, e recolhendo, durante a simulação, a posição atual do veículo, foi possível determinar o *waypoint* desejado.

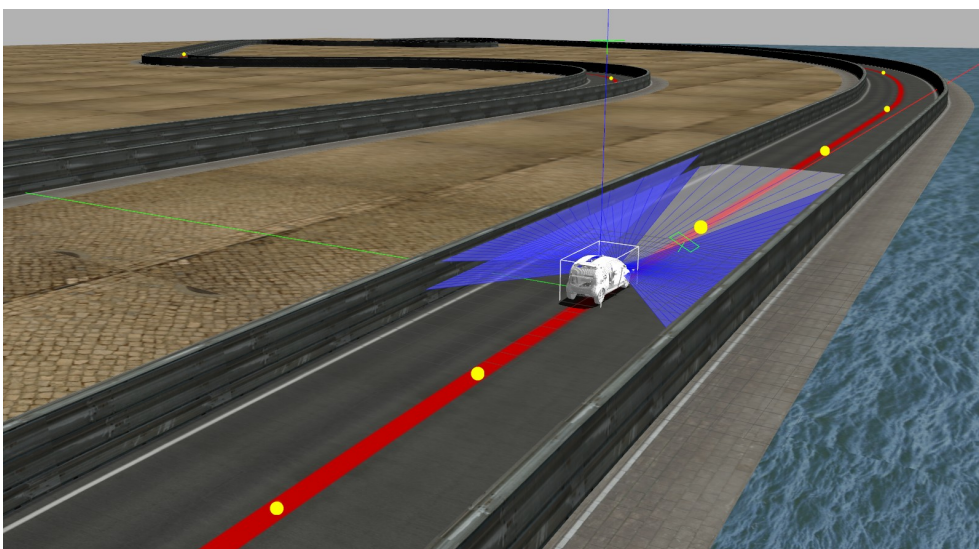


Figura 5.8: *Waypoits* (amarelo) ao longo do percurso

Tendo em conta que o espaçamento entre *waypoints* não variava ao longo do percurso, como é possível visualizar na figura 5.8, numa curva estes *waypoints* tornavam-se escassos, sendo necessário uma maior densidade consoante a curva existente. Para tal a lista de *waypoints* foi recreada de modo a permitir um maior número destes nas curvas, e uma redução dos mesmos em secções em linha reta. Este novo conjunto de *waypoints* foi construído utilizando os ângulos das linhas retas criadas pelos *waypoints* adjacentes, exemplificado na figura 5.9.

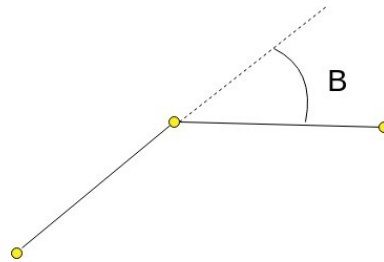


Figura 5.9: Ângulo gerado por duas linhas de *waypoints*

Assim, quanto mais apertada for a curva, maior o número de *waypoints* na mesma, resultando num comportamento mais estável pelo veículo.

Sendo possível subscrever à posição atual do veículo durante a simulação e a todos os *waypoint* existentes, foi criado um gerador de pontos atratores que calculam a distância entre a localização no "mundo" do veículo e todos os *waypoints* existentes, como também o ângulo entre a orientação do veículo com o novo *waypoint*, permitindo este último cálculo prevenir a utilização de *waypoints* localizados fora do percurso atual (estradas paralelas, etc.).

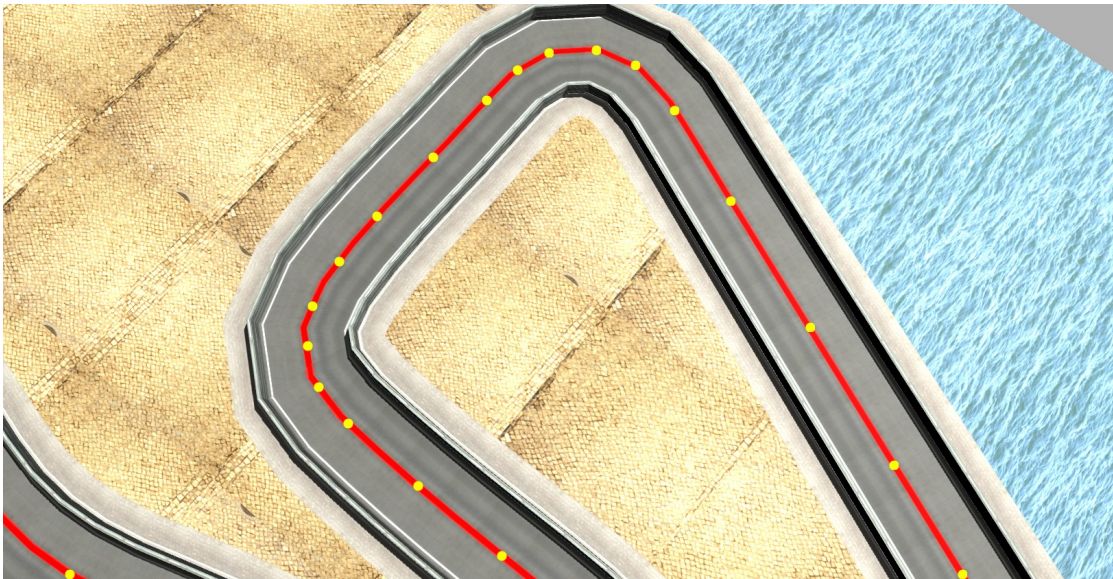


Figura 5.10: *Waypoints* (amarelo) mais concentrados nas curvas

### 5.6.2 Ponto Atrator utilizando a linha central

Após o sucesso na utilização de *waypoints* na obtenção do ponto atrator, era necessário criar uma alternativa que permitisse ao veículo percorrer um circuito sem a utilização destes *waypoints*,

visto que estes terão de ser gerados para cada percurso e se podem tornar imprecisos ou até inexistentes (utilização do GPS).

Para este fim, e de modo a complementar a necessidade do movimento do veículo pela faixa da direita, o ficheiro gerador do ponto atrator `APgenerator.cpp` foi recriado, utilizando a nuvem de pontos da linha central já processada no tópico /line\_pcl como guia da sua colocação. Aqui cada ponto é analisado sendo a distância ao veículo influenciada pela velocidade (tendo valores máximos e mínimos definidos nos parâmetros `APdistMax` e `APdistMin` respetivamente) e lateralmente dependendo da manobra a desempenhar. Em casos gerais em que o veículo circula pela via da direita sem obstáculos (veículos, objetos, curvas) no seu caminho, o ponto atrator é colocado no centro da via da direita à distância máxima definida (`APdistMax`).

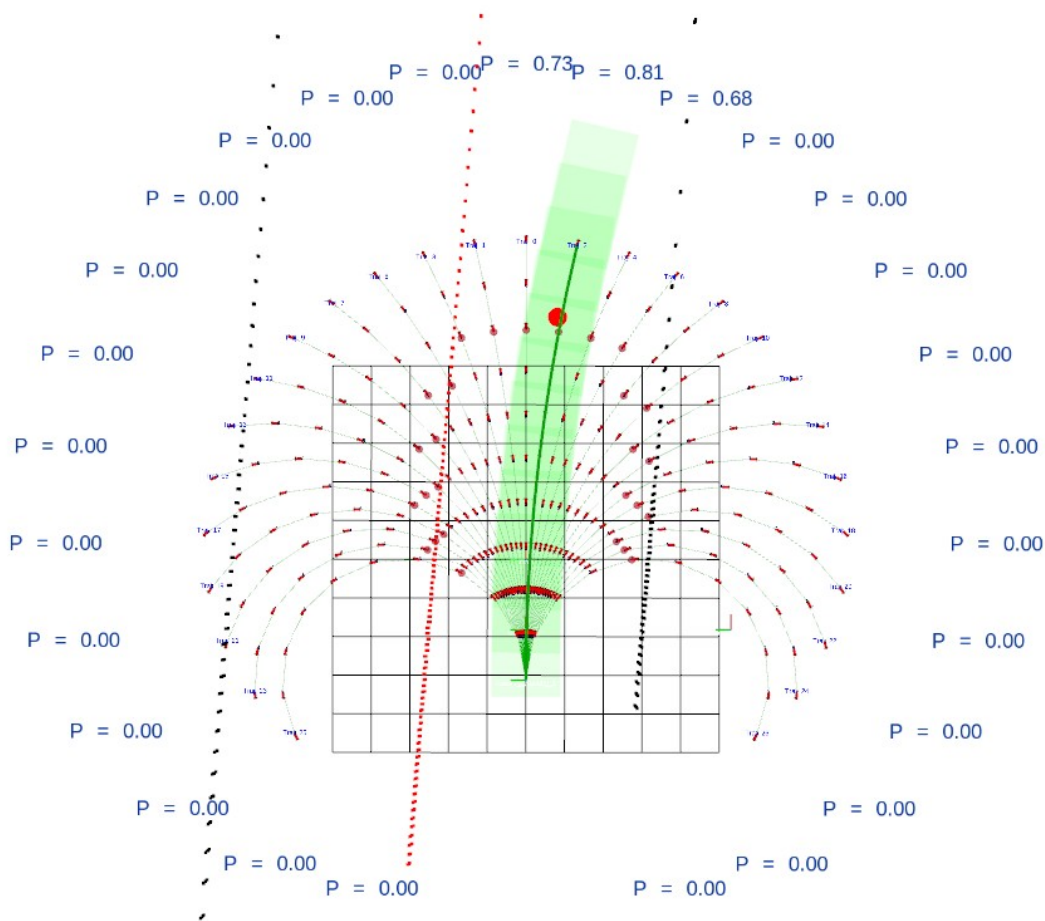


Figura 5.11: Rviz com o novo Ponto Atrator (bola vermelha)

O cálculo da distância máxima do novo ponto atrator (`max_dist_AP`) é realizado em função da velocidade do veículo através da equação 5.11, sendo adicionado a distância mínima definida no parâmetro `APdistMin`. Caso a velocidade seja nula esta distância tem o valor mínimo dado por `APdistMin`.

$$max\_dist\_AP = (APdistMax - APdistMin) \cdot \frac{speed}{SPEED\_REQUIRED} + APdistMin \quad (5.11)$$

Após este cálculo, caso a variável `max_dist_AP` seja inferior à multiplicação entre o parâmetro `MIN_SPACING_AP` e a distância mínima definida no parâmetro `APdistMin`, este valor é alterado de modo a permitir um espaçamento suficiente para o cálculo seguinte, isto é, de modo a que existam pontos suficientes para a colocação do ponto atrator.

Utilizando as posições no eixo `Ox` maiores que 0 dos pontos da linha central, é verificado qual destes se aproxima mais do limite máximo calculado anteriormente.

De seguida, no excerto de código 5.6 é apresentado o código principal destes cálculos:

```

1  if (this_speed_new != 0)
2  {
3      max_dist_AP = (APdistMax - APdistMin) * (this_speed_new / SPEED_REQUIRED)
4      ;
5      max_dist_AP = APdistMin + max_dist_AP;
6  }
7  else
8  {
9      max_dist_AP = APdistMin;
10 }
11
12 if (max_dist_AP < APdistMin * MIN_SPACING_AP)
13 {
14     max_dist_AP = APdistMin * MIN_SPACING_AP;
15 }
16 (... )
17
18 if (pct2.points[i].x > 0){
19
20     double point_dist = sqrt(pow(pct2.points[i].x, 2) + pow(pct2.points[i].y,
21     2));
22
23     if (point_dist < max_dist_AP && point_dist > dist_max_reached)
24     {
25         dist_max_reached = point_dist;
26         point_chosen_AP = i;
27
28         ap_x = pct2.points[i].x;
29         ap_y = pct2.points[i].y;
30     }
31 }

```

Listagem 5.6: Cálculo do ponto atrator

## 5.7 Zona de deteção de obstáculos

Com o objetivo de detetar obstáculos na estrada, é necessário distinguir dos pontos detetados pelos sensores LIDAR os que realmente são obstáculos (veículos, objetos, etc.). Estão também contidos nesta nuvem de pontos as paredes do circuito e a linha central.

Para tal foi desenvolvido um algoritmo que designa um espaço de análise tal como demonstrado na figura 5.12, evitando a inclusão das paredes e da linha central.



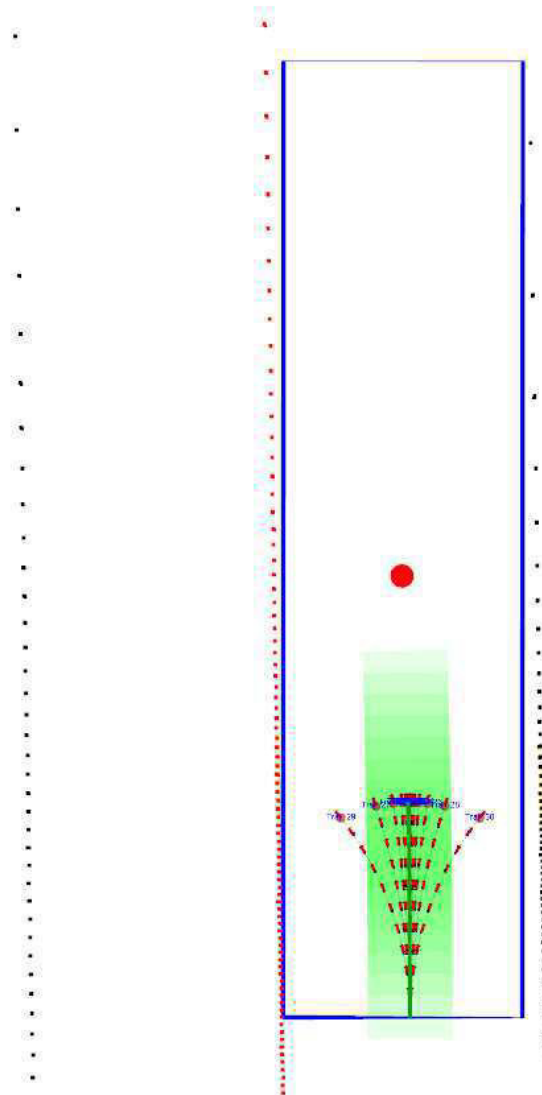


Figura 5.12: Espaço de análise (retângulo azul)

De modo a extrair os vértices do retângulo de análise, foi criado um cálculo que determina os pontos desejados, dentro da função geradora de obstáculos (função que gera o vetor de obstáculos através da nuvem de pontos), quer das paredes e restantes objetos, quer da linha central.

De modo a impedir a inclusão nas paredes e da linha da estrada, foram analisados todos os pontos fornecidos como obstáculos, resultando em coordenadas que serão utilizadas como vértices na construção do espaço de detecção de obstáculos.

Sendo possível a distinção entre paredes e linha central, a criação do espaço de detecção foi dividido em duas partes: uma curva para a direita e uma curva para a esquerda. Isto deve-se à existência de valores discrepantes dos pontos das paredes visto que para cada intervalo no eixo  $Ox$  existem dois intervalos de valores no eixo  $Oy$ .

### Curva à direita

Analisando a curva à direita foram criadas duas variáveis  $y_{\min\_l\_right}$  e  $y_{\max\_w\_right}$  como exemplificado na figura 5.13. Nesta figura é possível visualizar a pista a percorrer, com o veículo representado por um cubo vermelho e o espaço de detecção representado por um retângulo verde.

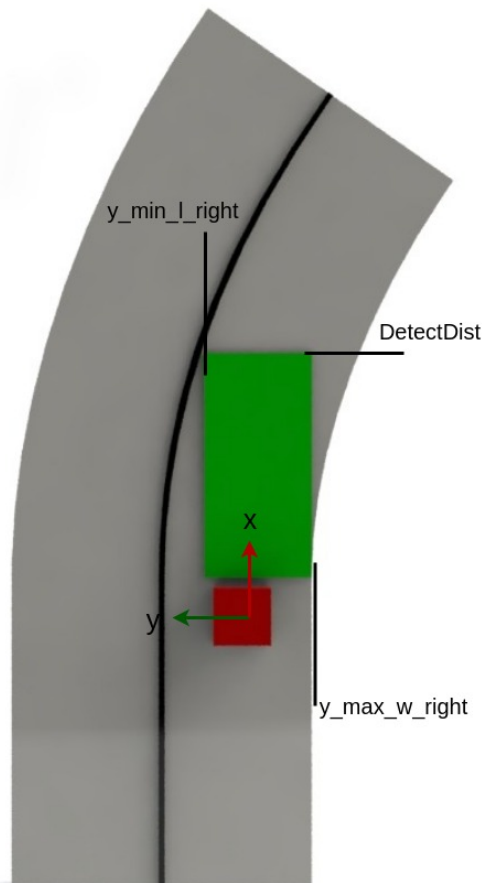


Figura 5.13: Exemplificação do espaço detetável da curva da direita

- **$y_{\min\_l\_right}$**  - (Y-mínimo-linha-curva-direita) - Valor mínimo dos pontos da linha central no eixo Oy calculado em função na distância máxima de detecção DetectDist;

```

1 y_min_l_right = 500;
2
3 (...)
4
5 if (pc.points[j].y < y_min_l_right &&
6     pc.points[j].x > (DetectDist - DetectDist / 10) &&
7     pc.points[j].x < DetectDist)
8     {
9         y_min_l_right = pc.points[j].y;
10    }

```

Listagem 5.7: Cálculo do  $y_{\min\_l\_right}$

- **y\_max\_w\_right** - (Y-máximo-parede-curva-direita) - Valor máximo dos pontos da parede no eixo Oy calculado em função da menor posição positiva no eixo Ox, tendo em consideração apenas valores negativos no eixo Oy, visto que é apenas desejados pontos da parede direita;

```
1 y_max_w_right = -500;
2
3 (...)
4
5 if (pc.points[j].y > y_max_w_right &&
6     pc.points[j].x > 0 &&
7     pc.points[j].x < (DetectDist / 10) &&
8     pc.points[j].y < 0)
9     {
10        y_max_w_right = pc.points[j].y;
11    }
```

Listagem 5.8: Cálculo do y\_max\_w\_right

### Curva à esquerda

Em relação à curva à esquerda foram criadas duas variáveis `y_min_l_left` e `y_max_w_left` como exemplificado na figura 5.14.

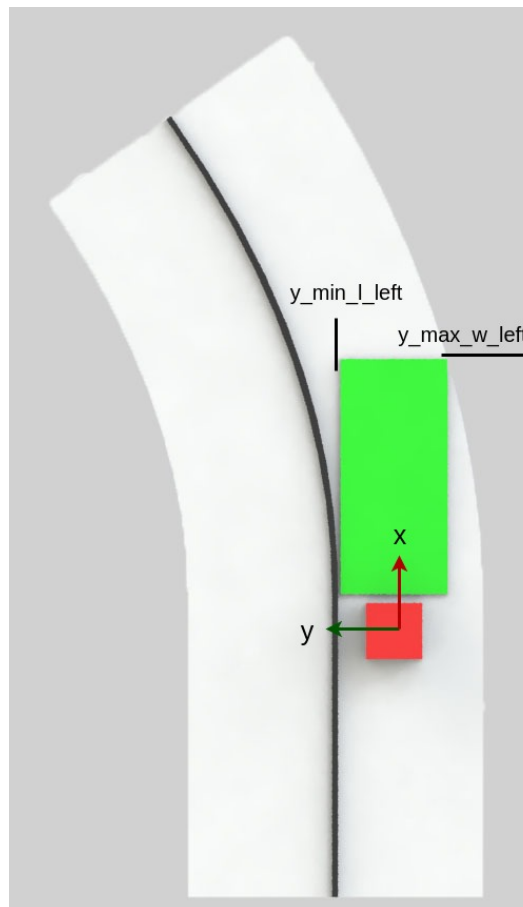


Figura 5.14: Exemplificação do espaço detetável da curva da esquerda

- **y\_min\_l\_left** -(Y-mínimo-linha-curva-esquerda) - Valor mínimo dos pontos da linha central no eixo Oy apenas calculado em função dos valores positivos no eixo Ox;

```

1 y_min_l_left = 500;
2
3 (...)
4
5 if (pc.points[j].y < y_min_l_left &&
6     pc.points[j].x > 0)
7     {
8         y_min_l_left = pc.points[j].y;
9     }

```

Listagem 5.9: Cálculo do y\_min\_l\_left

- **y\_max\_w\_left** - Valor máximo dos pontos da parede lateral direita no eixo Oy calculado em função na distância máxima de detecção DetectDist e de valores negativos em Oy;

```

1 y_max_w_left = -500;
2
3 (...)
4
5 if (pc.points[j].y > y_max_w_left &&
6     pc.points[j].y < 0 &&
7     pc.points[j].x > (DetectDist - DetectDist / 2) &&
8     pc.points[j].x < DetectDist)
9     {
10        y_max_w_left = pc.points[j].y;
11    }

```

Listagem 5.10: Cálculo do y\_max\_w\_left

Após determinadas estas quatro variáveis, é efetuada a comparação entre os valores relativos a cada lado da estrada:

```

1 double limit_left;
2 double limit_right;
3
4 //Limite lateral esquerdo
5 if (y_min_l_left < y_min_l_right)
6     {
7         limit_left = y_min_l_left;
8     }
9 else
10    {
11        limit_left = y_min_l_right;
12    }
13
14 //Limite lateral direito
15 if (y_max_w_right > y_max_w_left)
16     {
17        limit_right = y_max_w_right;
18    }
19 else
20    {
21        limit_right = y_max_w_left;
22    }

```

Listagem 5.11: Cálculo dos limites no eixo Oy do espaço de detecção de obstáculos

Esta zona de análise é ativada consoante o parâmetro booleano DETECTION seja verdadeiro. Com os dois limites referidos anteriormente é definido o espaço detetável, sendo analisados todos os obstáculos e adicionados a uma nova PointCloud.

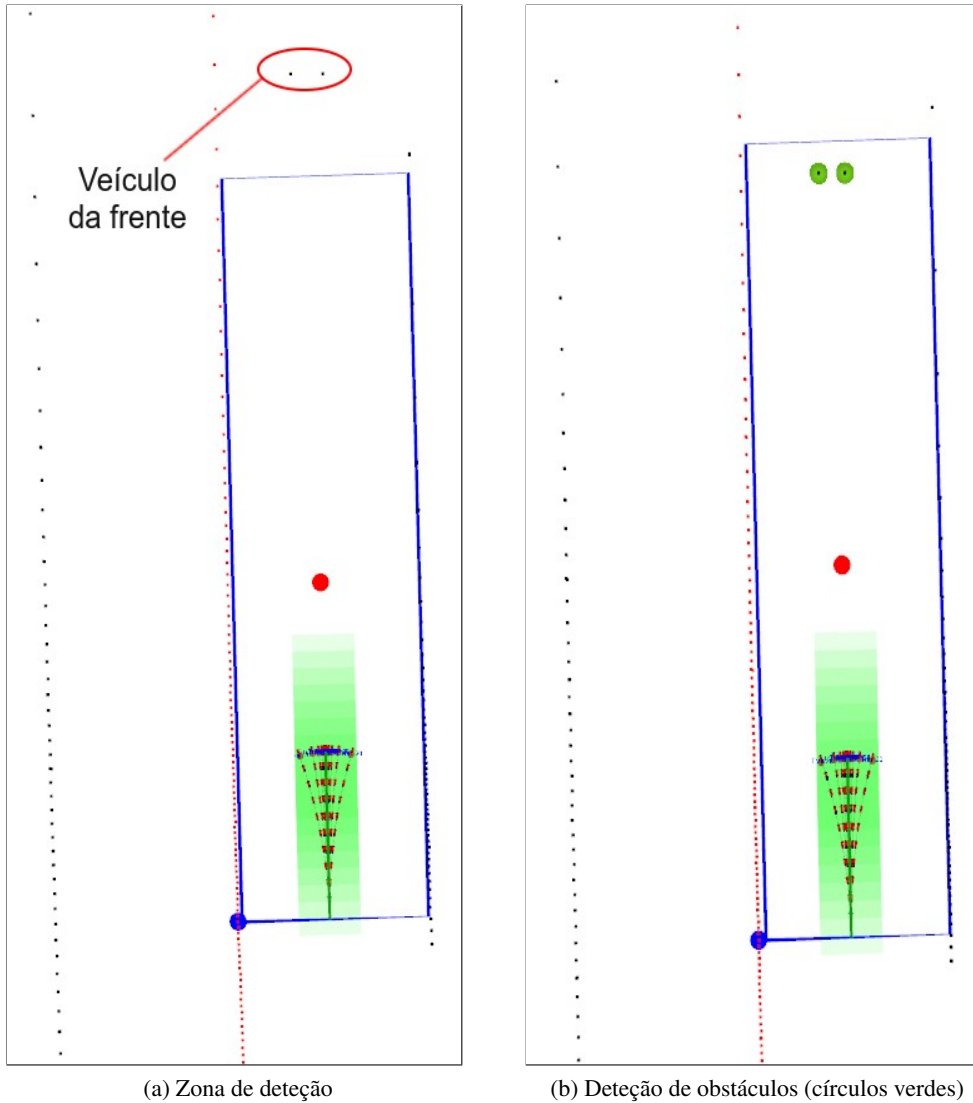


Figura 5.15: Visualização no Rviz da zona de deteção

## 5.8 Zona de deteção de obstáculos traseira

Com o mesmo método utilizado para calcular a zona de análise, foi criada uma segunda zona com o objetivo da deteção de obstáculos na retaguarda do veículo como exemplificado na figura 5.16. Esta tem como objetivo permitir a deteção de obstáculos (veículos) que circulem na via da direita quando o veículo executa uma manobra que o coloca na via da esquerda, como é exemplo a ultrapassagem.

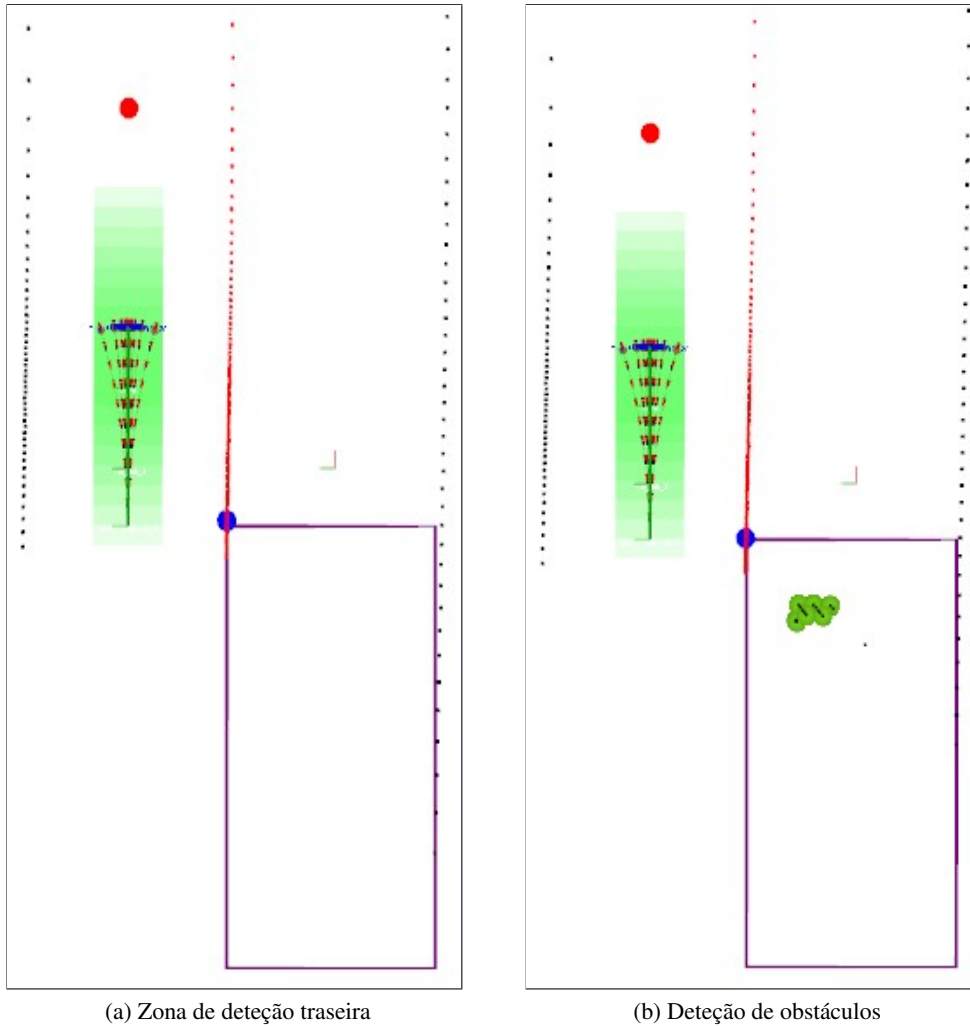


Figura 5.16: Visualização no Rviz da zona de deteção traseira

## 5.9 Cruzamento de dois veículos

Com a utilização da linha central como obstáculo, o cruzamento entre dois veículos tornou-se bastante simples, ignorando-se mutuamente como demonstrado na figura 5.17, desde que não ultrapassem a linha central.

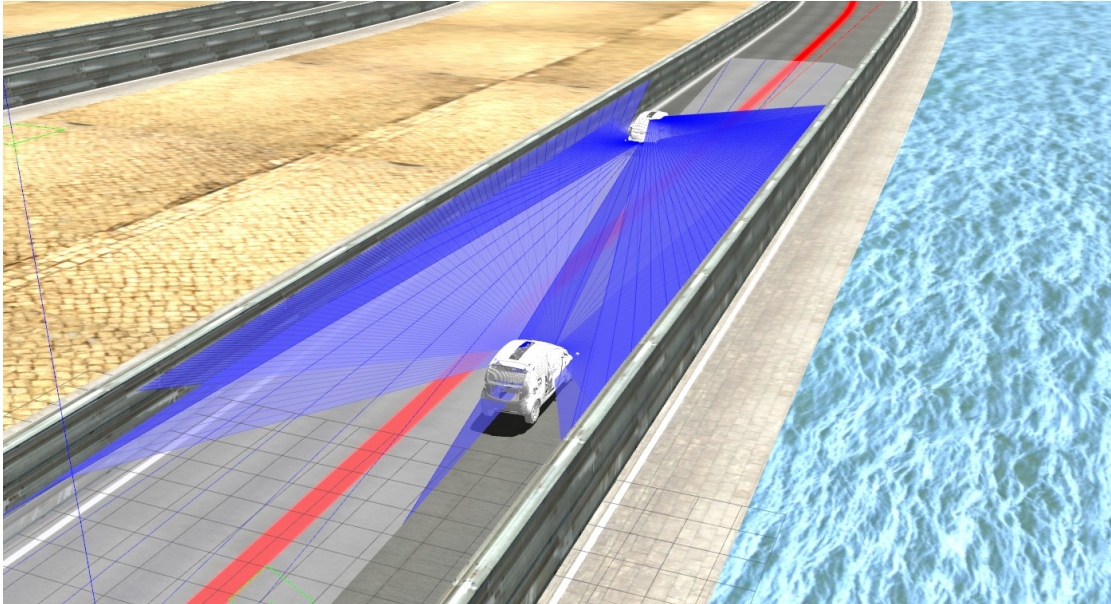


Figura 5.17: Cruzamento de dois veículos

Apesar de se ignorarem, os veículos ainda se detetam um ao outro (figura 5.18), permitindo a eventual criação de manobras utilizando esta solução.

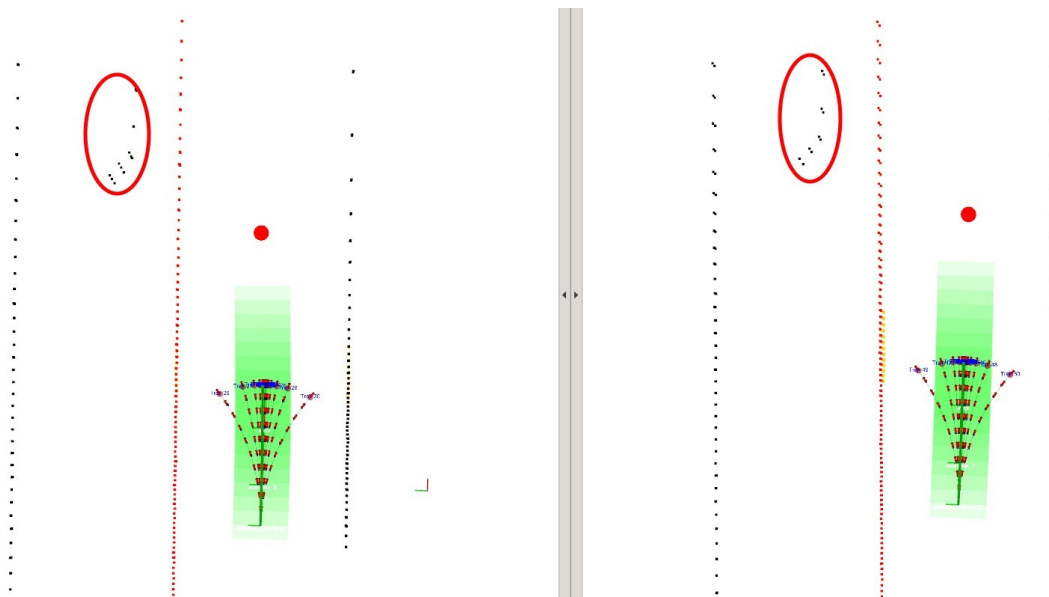


Figura 5.18: Detecção do veículo em sentido contrário (pontos dentro do círculo vermelho) de cada veículo visualizados no Rviz

## 5.10 Ultrapassagem

Sendo a manobra mais desafiadora, a ultrapassagem requer a utilização de diversas ferramentas até agora desenvolvidas. Foi utilizada uma resolução de (0,5°) nos sensores LIDAR influenciando o número de pontos detetados durante esta manobra permitindo desta forma uma simulação mais fluida devido à redução de pontos recolhidos mas tornando a deteção menos fiável. Nesta secção será referido a desativação e ativação de parâmetros booleanos, equivalendo ao valores false e true respetivamente.

Na figura 5.19 está representado o diagrama total de uma ultrapassagem. Este diagrama será analisado em detalhe de modo a explicar cada fase desta manobra utilizando para tal excertos deste diagrama e do código desenvolvido, de modo a auxiliar a compreensão do processo.

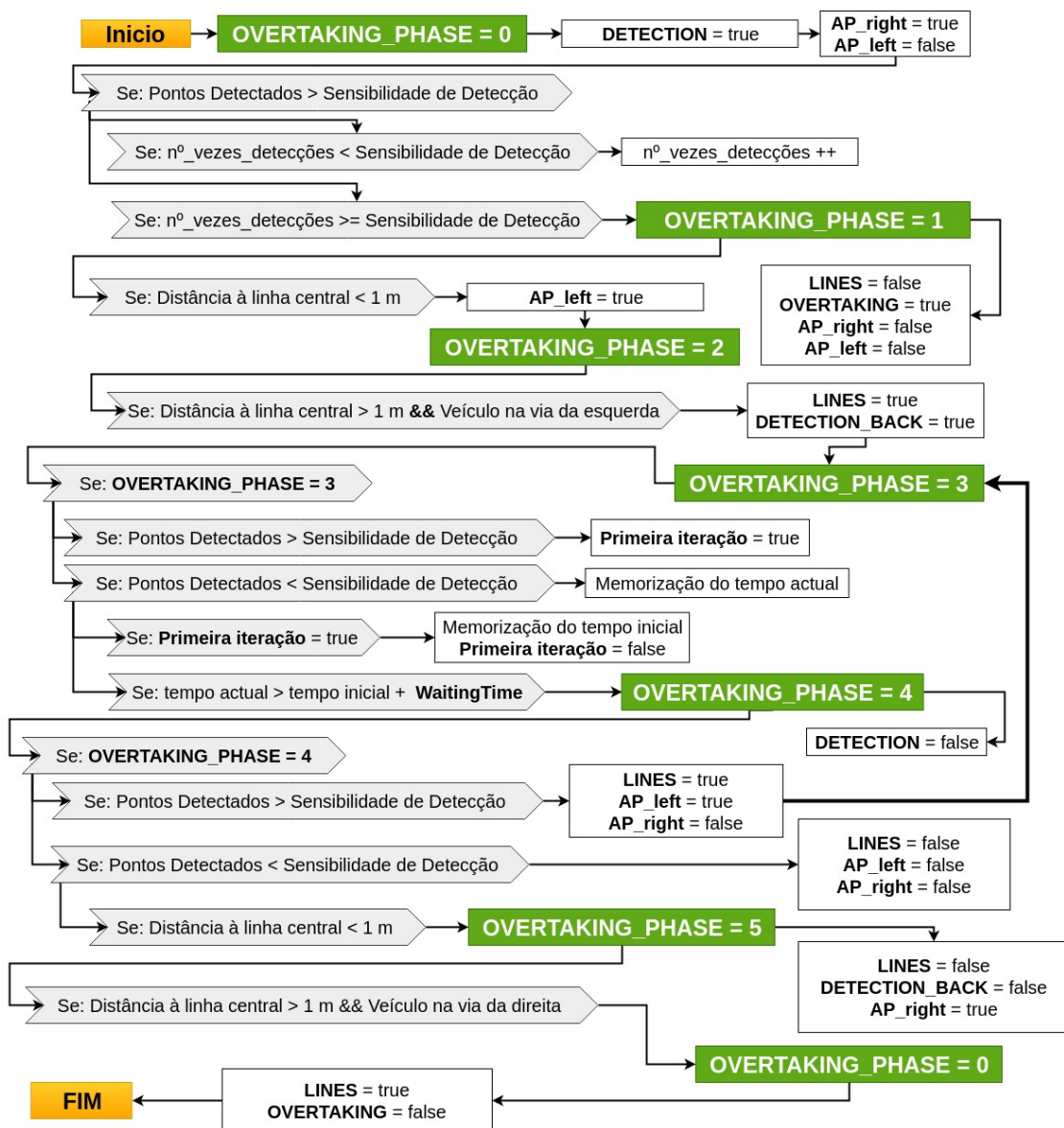


Figura 5.19: Diagrama total da ultrapassagem



No diagrama da figura 5.19 estão representadas várias fases da ultrapassagem. Estas consistem em:

- **OVERTAKING\_PHASE = 0** - Início/Fim da ultrapassagem;
- **OVERTAKING\_PHASE = 1** - Detecção de obstáculos e início de mudança de via;
- **OVERTAKING\_PHASE = 2** - Mudança para a via da esquerda;
- **OVERTAKING\_PHASE = 3** - Movimento pela via da esquerda detetando os obstáculos presentes na via da direita;
- **OVERTAKING\_PHASE = 4** - Detecção de obstáculos de modo a obter "permissão" para voltar para a via da direita;
- **OVERTAKING\_PHASE = 5** - Mudança para a via da direita verificando possíveis obstáculos ainda existentes nesta via;

### Início da deteção

Inicialmente o veículo começará com o parâmetro booleano DETECTION ativo, isto é, com valor verdadeiro, permitindo a deteção do veículo a ultrapassar. Este parâmetro irá criar o espaço de deteção frontal como exemplificado na figura 5.21. No passo inicial é assegurado que o ponto atrator se encontra na via da direita.

Na figura 5.20 é possível visualizar o excerto do diagrama referente ao início da ultrapassagem com o auxílio do excerto de código 5.12.

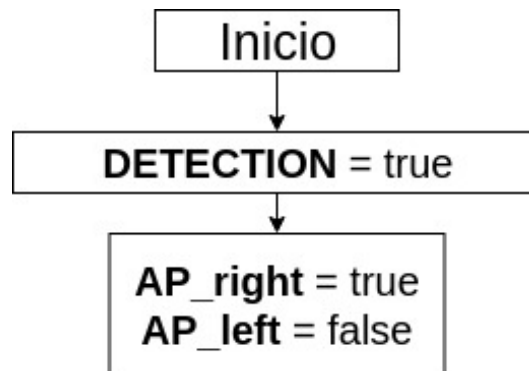


Figura 5.20: Diagrama do início da deteção de obstáculos

```

1  if (DETECTION == true || OVERTAKING == true || CRUZAMENTO == true)
2  {
3    if (OVERTAKING == false)
4    {
5      n.setParam("Param/AP_right", true);
6      n.setParam("Param/AP_left", false);
  
```

Listagem 5.12: Início da deteção de obstáculos

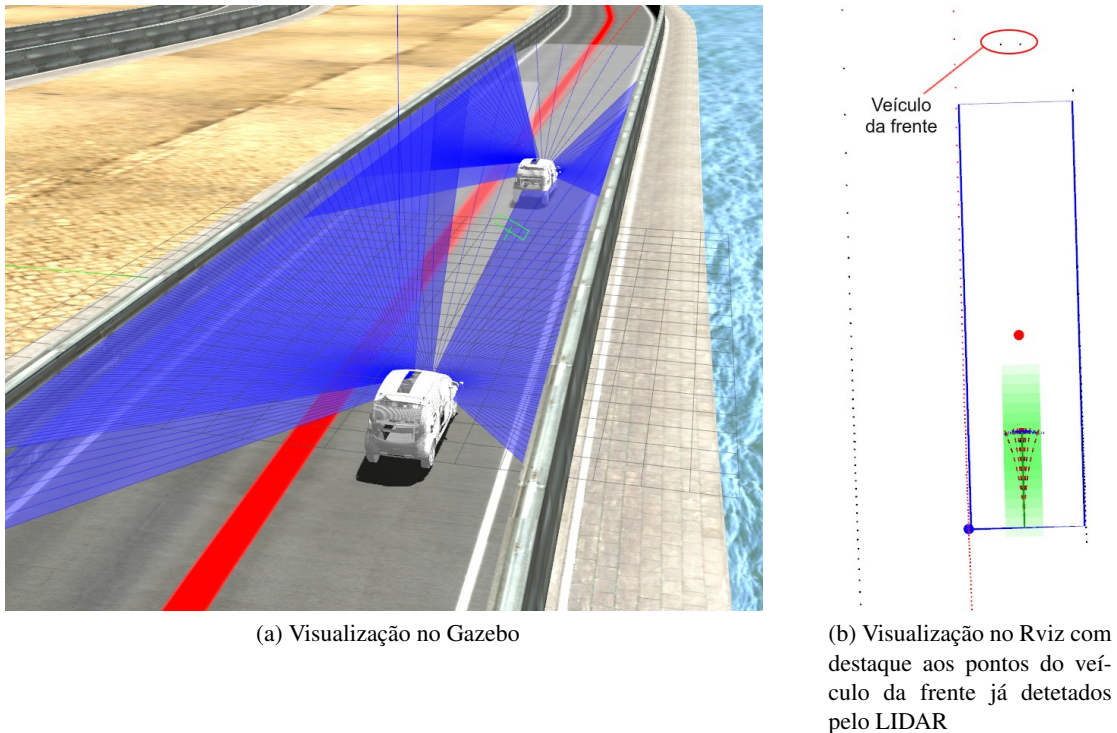


Figura 5.21: Início da detecção de obstáculos

### Primeira Fase

Ao detetar um obstáculo (veículo) tal como ilustrado na figura 5.23 onde é demonstrado a necessidade de existirem um número mínimo de pontos para que seja considerado um verdadeiro obstáculo (no caso do exemplo este valor iguala a 2) definido no parâmetro dinâmico `Detection_Sensitivity`, é executada uma contagem (`detection_times`) de modo a evitar eventuais erros esporádicos de deteções. Quando ocorrem um número de deteções seguidas definido pelo mesmo parâmetro `Detection_Sensitivity`, é então ativado a primeira fase da ultrapassagem, onde são desativados os parâmetros `LINES`, `AP_right` e `AP_left`, e ativado o parâmetro `OVERTAKING`.

Desativando o parâmetro `LINES` o veículo deixa de reconhecer a linha central como um obstáculo, permitindo a sua aproximação ou até transposição. A ativação do parâmetro `OVERTAKING` permite ao planeador prosseguir com planeamento desta manobra. Este parâmetro é utilizado de modo a permitir a utilização do método de deteção sem a obrigatoriedade de uma eventual ultrapassagem. Por fim é desativado o parâmetro `AP_right` e `AP_left`, estando este último já com valor falso se não tiver existido problemas. Com a definição destes dois últimos parâmetros o ponto atrator irá situar-se no centro da via, isto é, sob a linha central. Optou-se por colocar o ponto atrator nesta posição de modo a evitar movimentos "bruscos" por parte do veículo ao mudar de via.

Na figura 5.22 é possível visualizar o excerto do diagrama referente a esta primeira fase da ultrapassagem com o auxílio do excerto de código 5.13.

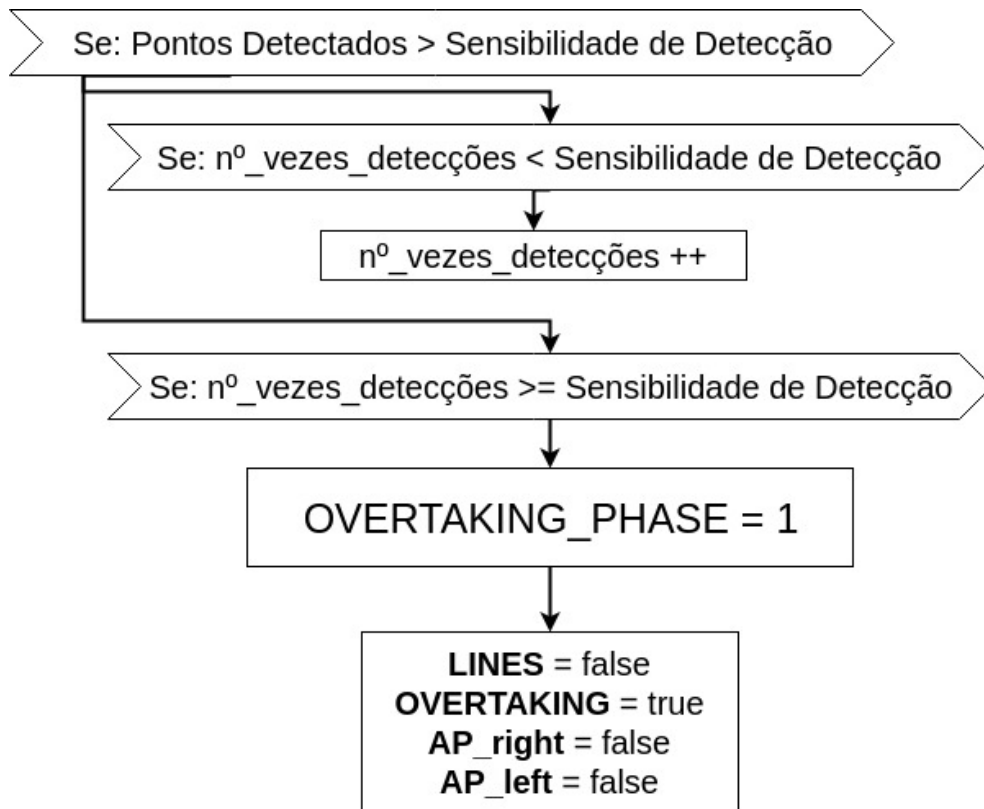


Figura 5.22: Diagrama da primeira fase da ultrapassagem

```

1  if (count_points_detected > Detection_Sensitivity)
2  {
3    if (detection_times < Detection_Sensitivity)
4    {
5      detection_times++;
6    }
7    else
8    {
9      detection_times = 0;
10     n.setParam("Param/LINES", false);
11     n.setParam("Param/OVERTAKING", true);
12     n.setParam("Param/AP_right", false);
13     n.setParam("Param/AP_left", false);
14     overtaking_phase = 1;
15 }

```

Listagem 5.13: Início da detecção de obstáculos

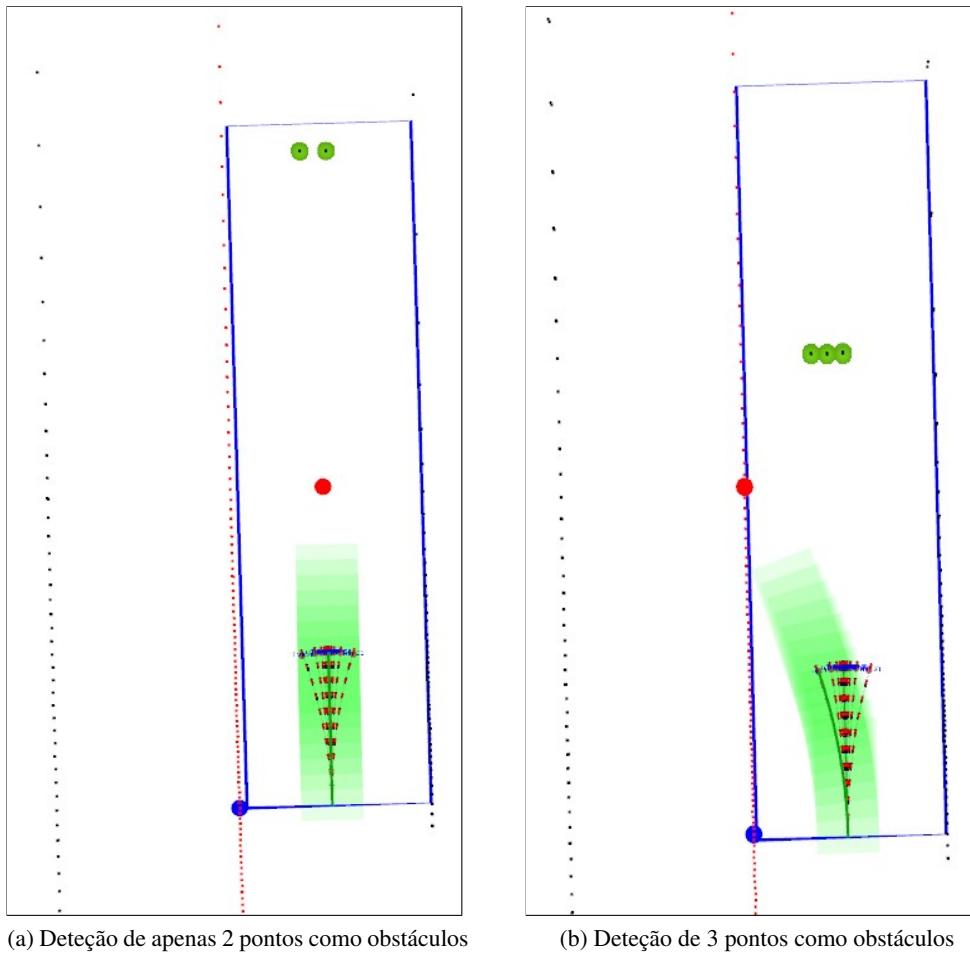


Figura 5.23: Detecção de obstáculos

### Segunda Fase

A segunda fase da ultrapassagem envolve a aproximação do veículo à linha central, sendo que ao atingir uma distância menor que 1 metro desta, o parâmetro `AP_left` é ativado, colocando o ponto atrator na via da esquerda como demonstrado na figura 5.25.

Na figura 5.24 é possível visualizar o excerto do diagrama referente a esta segunda fase da ultrapassagem com o auxílio do excerto de código 5.14.

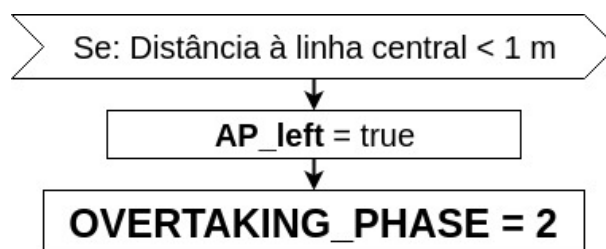


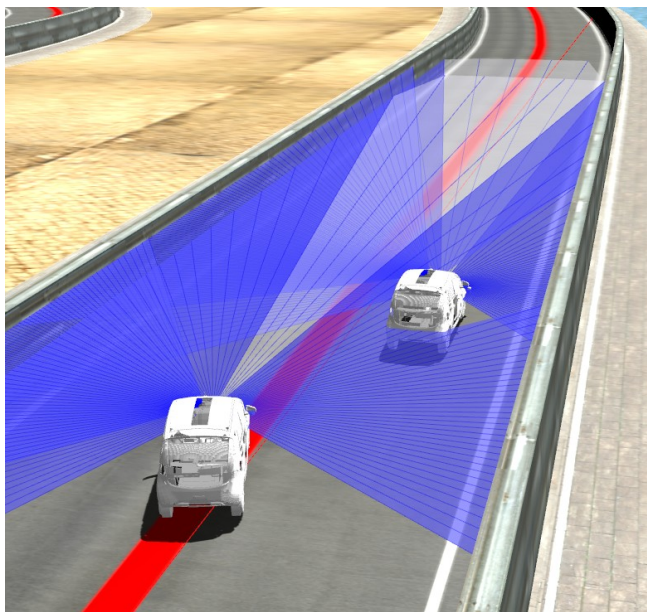
Figura 5.24: Diagrama da segunda fase da ultrapassagem

```

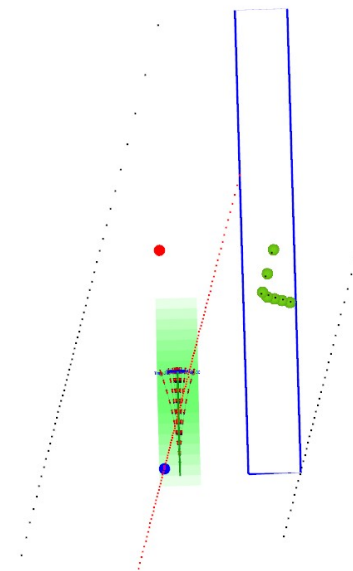
1  if (dist_clp < 1 && overtaking_phase == 1) // começou a curvar e perto da
    linha
2  {
3    n.setParam("Param/AP_left", true);
4    overtaking_phase = 2;
5  }

```

Listagem 5.14: Aproximação à linha central



(a) Visualização no Gazebo



(b) Colocação do AP na via da esquerda

Figura 5.25: Início da ultrapassagem

### Terceira Fase

Com o veículo já na via da esquerda é calculada a distância deste veículo à linha central, ativando o parâmetro `LINES` e `DETECTION_BACK` quando este se encontra a uma distância superior a 1 metro. O parâmetro `LINES` é ativado de modo a estabilizar o veículo na via da esquerda, tornando novamente a linha central um obstáculo, e é iniciada a detecção de obstáculos na retaguarda do veículo na via da direita com o parâmetro `DETECTION_BACK` como demonstrado na figura 5.27. Assim é salvaguardado o surgimento de eventuais veículos que circulem a velocidades superiores ao veículo em análise e que bloqueiem a via da direita, tornando-se um eventual perigo no retorno deste à sua via original.

Na figura 5.26 é possível visualizar o excerto do diagrama referente a esta terceira fase da ultrapassagem com o auxílio do excerto de código 5.15.

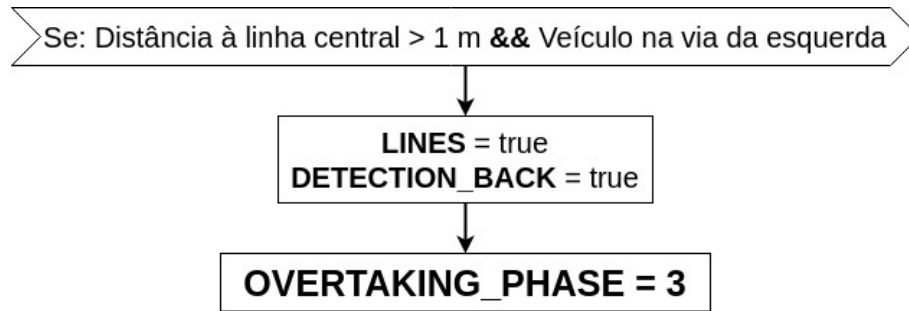
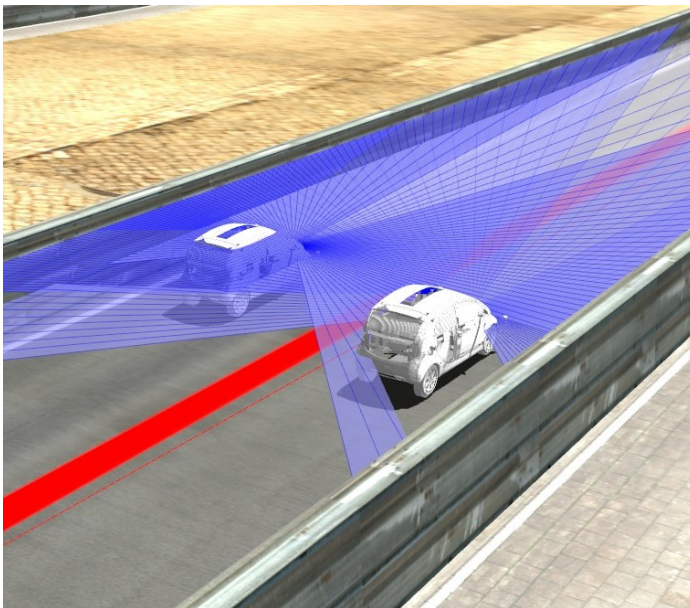


Figura 5.26: Diagrama da terceira fase da ultrapassagem

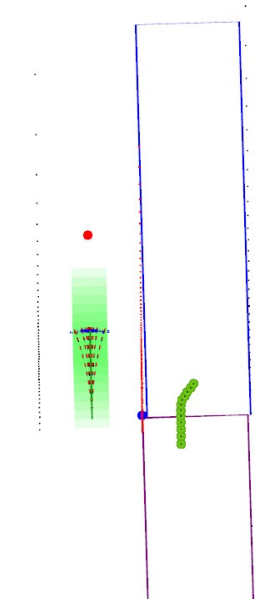
```

1  if (dist_clp > 1 && pos_clp_y < 0 && overtaking_phase == 2) // o veículo
2     esta a esquerda da linha
3     {
4       n.setParam("Param/LINES", true);
5       n.setParam("Param/DETECTION_BACK", true);
6       overtaking_phase = 3;
7     }
  
```

Listagem 5.15: Veículo na via da esquerda



(a) Visualização no Gazebo



(b) Detecção do veículo da via da direita

Figura 5.27: Ultrapassagem e detecção de obstáculos na via da direita

### Quarta Fase

Na quarta fase da ultrapassagem é determinado um tempo de espera através do parâmetro dinâmico `WaitingTime` para uma eventual detecção de obstáculos na via da direita. Este tempo de espera permite obter informação de veículos da via da direita que possam surgir já depois do fim da detecção do veículo ultrapassado, tal como exemplificado anteriormente.

Deste modo é criado um *Time Stamp* no momento em que deixou de ser detetado qualquer obstáculo. Caso seja detetado um novo obstáculo, o veículo mantém o seu comportamento. Se no fim do período definido pelo parâmetro `WaitingTime` não tiver sido detetado mais nenhum obstáculo, prossegue-se para a próxima fase da ultrapassagem.

Na figura 5.28 é possível visualizar o excerto do diagrama referente a esta quarta fase da ultrapassagem com o auxílio do excerto de código 5.16.

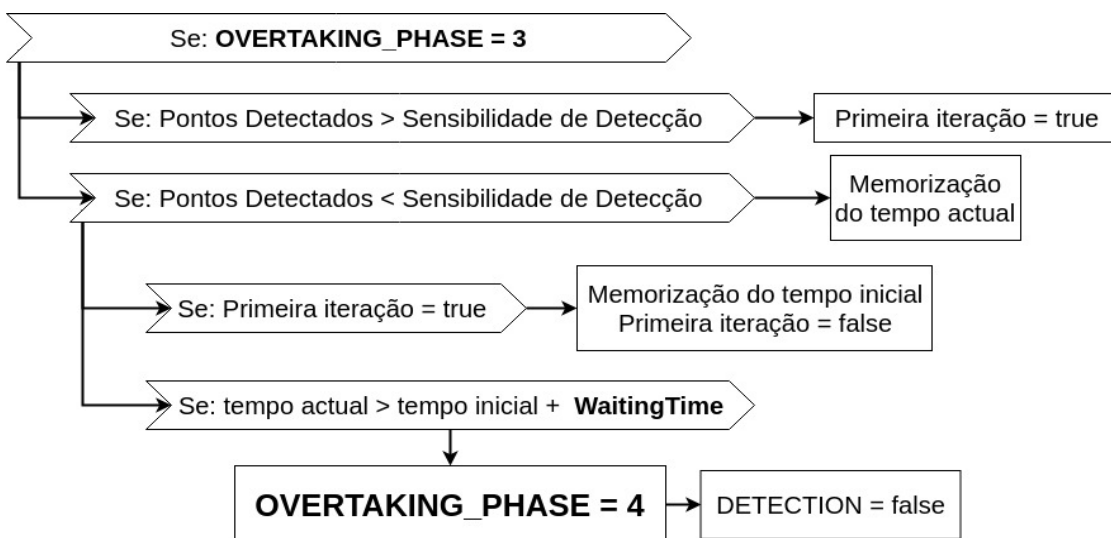


Figura 5.28: Diagrama da quarta fase da ultrapassagem

```

1 bool begin_time = true;
2 if (overtaking_phase == 3)
3 {
4     if (count_points_detected < Detection_Sensitivity)
5     {
6         ros::Time currtime = ros::Time::now();
7         if (begin_time == true)
8         {
9             begin_3 = ros::Time::now();
10            begin_time = false;
11        }
12        else if (currtime > begin_3 + ros::Duration(WaitingTime))
13        {
14            overtaking_phase = 4;
15            n.setParam("Param/DETECTION", false);
16        }
17    }else{
18        begin_time = true;
19    }
20 }
  
```

Listagem 5.16: Detecção de obstáculos na via da direita

### Quinta Fase

Com a inexistência de obstáculos na via da direita os parâmetros `LINES`, `AP_left` e `AP_right` são desativados. Tal como aconteceu na primeira fase, ao tornar o parâmetro `LINES` falso, a linha central deixa de ser um obstáculo, e com o auxílio da colocação do ponto atrator sobre a linha central através dos parâmetros `AP_left` e `AP_right`, o veículo irá deslocar-se para o centro da estrada como demonstrado na figura 5.30.

Se durante a deslocação do veículo para o centro da estrada um objeto for detetado, é acionado a fase 3 da ultrapassagem, sendo ativados de novo os parâmetros `LINES` e `AP_left`. Deste modo o veículo volta a circular pela via da esquerda até deixar de detetar o obstáculo demonstrado na figura 5.31.

Caso não exista nenhuma deteção e o veículo fique a uma distância inferior a 1 metro da linha central, é ativada a fase 5 da ultrapassagem, colocando o ponto atrator na via da direita através da ativação do parâmetro `AP_right` e desativando a deteção de obstáculos na retaguarda.

Na figura 5.29 é possível visualizar o excerto do diagrama referente a esta quinta fase da ultrapassagem com o auxílio do excerto de código 5.17.

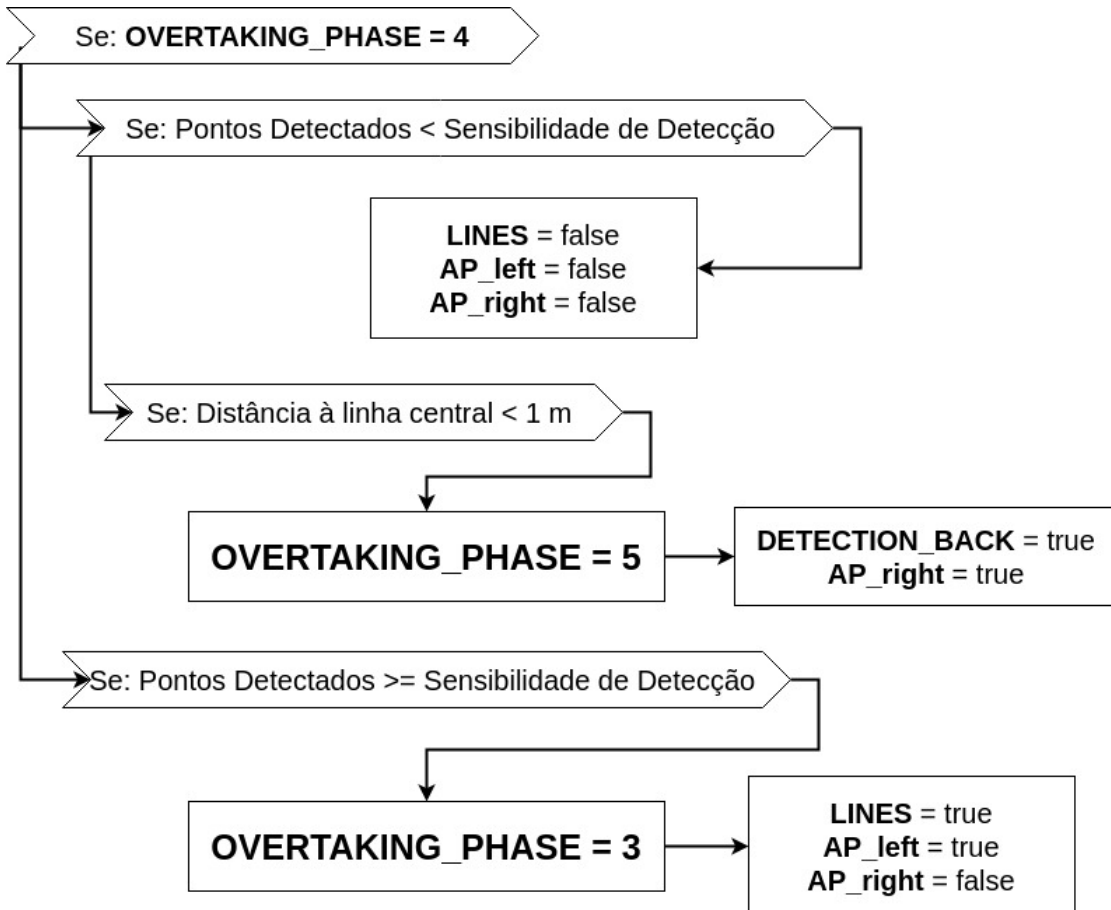


Figura 5.29: Diagrama da quinta fase da ultrapassagem

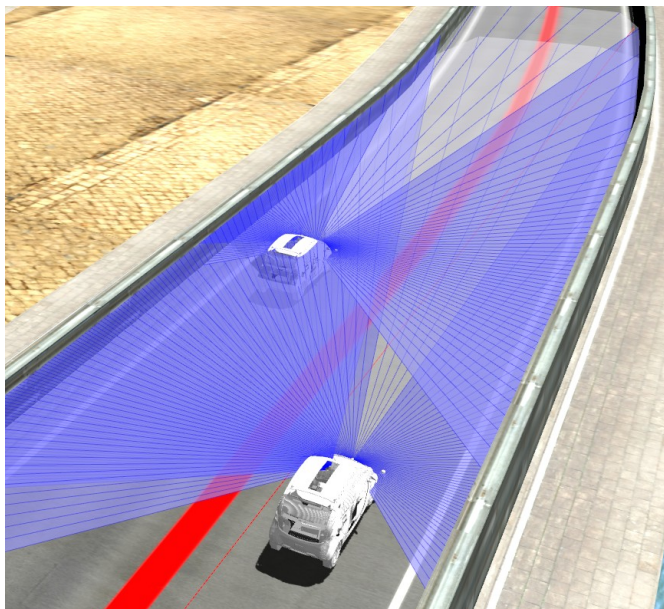


```

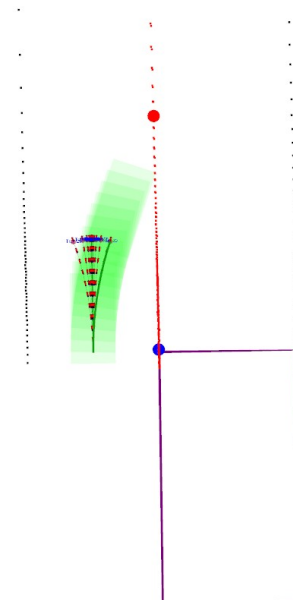
1  if (overtaking_phase == 4)
2  {
3
4      if (count_points_detected < Detection_Sensitivity)
5      {
6          n.setParam("Param/LINES", false);
7          n.setParam("Param/AP_left", false);
8          n.setParam("Param/AP_right", false);
9
10         if (dist_clp < 1) // começou a curvar e perto da linha
11         {
12             n.setParam("Param/AP_right", true);
13             n.setParam("Param/DETECTION_BACK", false);
14             overtaking_phase = 5;
15         }
16     }
17     else //objetos detetados
18     {
19         n.setParam("Param/LINES", true);
20         n.setParam("Param/AP_left", true);
21         n.setParam("Param/AP_right", false);
22
23         overtaking_phase = 3;
24     }
25 }

```

Listagem 5.17: Segunda aproximação à linha central



(a) Visualização no Gazebo



(b) Colocação do AP no centro da via

Figura 5.30: Fim da detecção de obstáculos

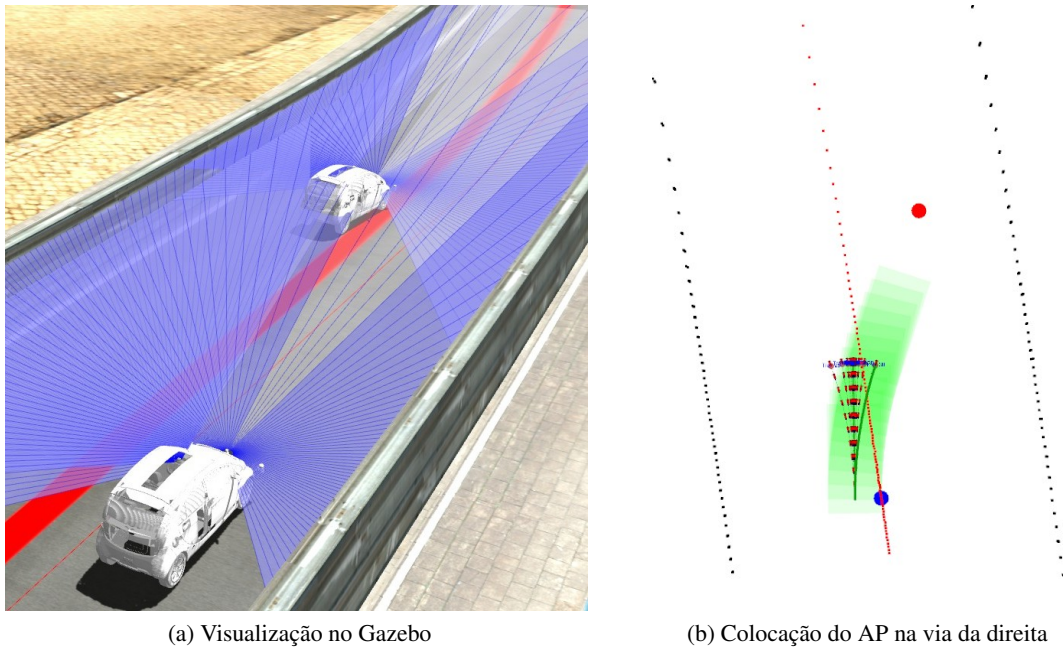


Figura 5.31: Desvia para a via da direita

### Última Fase

Por fim, é verificada a distância do veículo à linha central e caso esta seja superior a 1 metro (e esteja na via da direita) a ultrapassagem está concluída, retomando os parâmetros iniciais, isto é, ativando a linha central como obstáculo através do parâmetro `LINES`, desativando o modo de ultrapassagem com o parâmetro `OVERTAKING` como demonstrado na figura 5.33.

Na figura 5.32 é possível visualizar o excerto do diagrama referente a esta última fase da ultrapassagem com o auxílio do excerto de código 5.18.

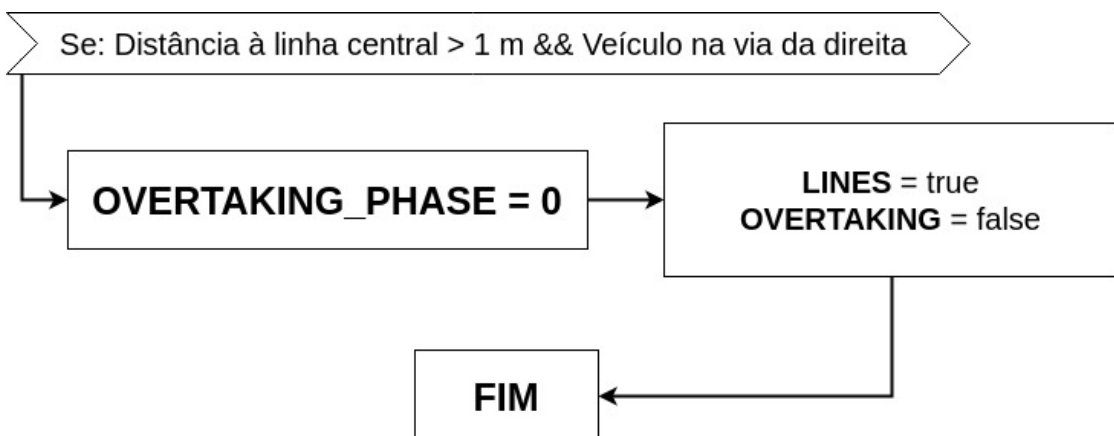
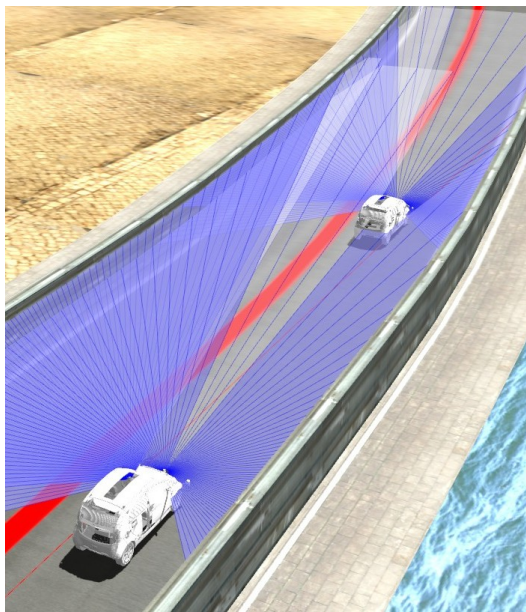


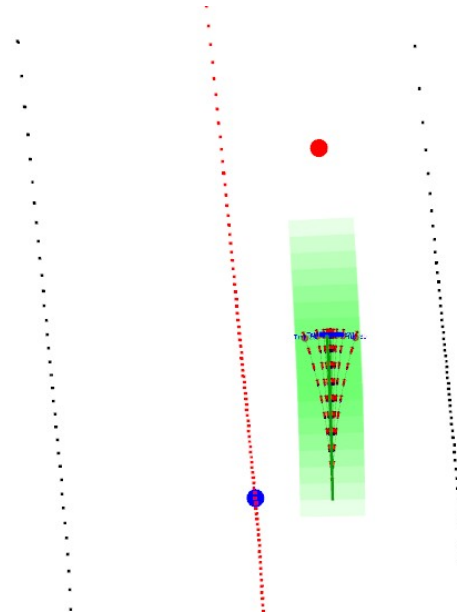
Figura 5.32: Diagrama da última fase da ultrapassagem

```
1  if (overtaking_phase == 5)
2  {
3    if (dist_clp > 1 && pos_clp_y > 0) // o veículo esta a direita da linha
4    {
5      n.setParam("Param/LINES", true);
6      n.setParam("Param/OVERTAKING", false);
7    }
8  }
```

Listagem 5.18: Segunda aproximação à linha central



(a) Visualização no Gazebo



(b) Retoma dos parâmetros iniciais

Figura 5.33: Conclusão da ultrapassagem



## Capítulo 6

# Testes e resultados

Neste capítulo apresenta-se os resultados obtidos das diferentes alterações realizadas ao projeto inicial como também a discussão da qualidade das manobras configuradas. Os resultados retirados nestas análises utilizam a manobra de ultrapassagem como base de estudo.

### 6.1 Simulação base da análise

Sendo que a manobra de ultrapassagem aborda todas as ferramentas desenvolvidas, a análise destas ferramentas utiliza uma simulação pré configurada de uma ultrapassagem entre dois veículos autónomos como apresentado na figura 6.1.

Tendo em conta as diferenças significativas dos resultados obtidos com a utilização do parâmetro  $Vel\_Ang$  (parâmetro que ativa a redução da velocidade do veículo consoante o ângulo da trajetória), alguns dos resultados seguintes serão diferenciados dependendo da sua utilização.

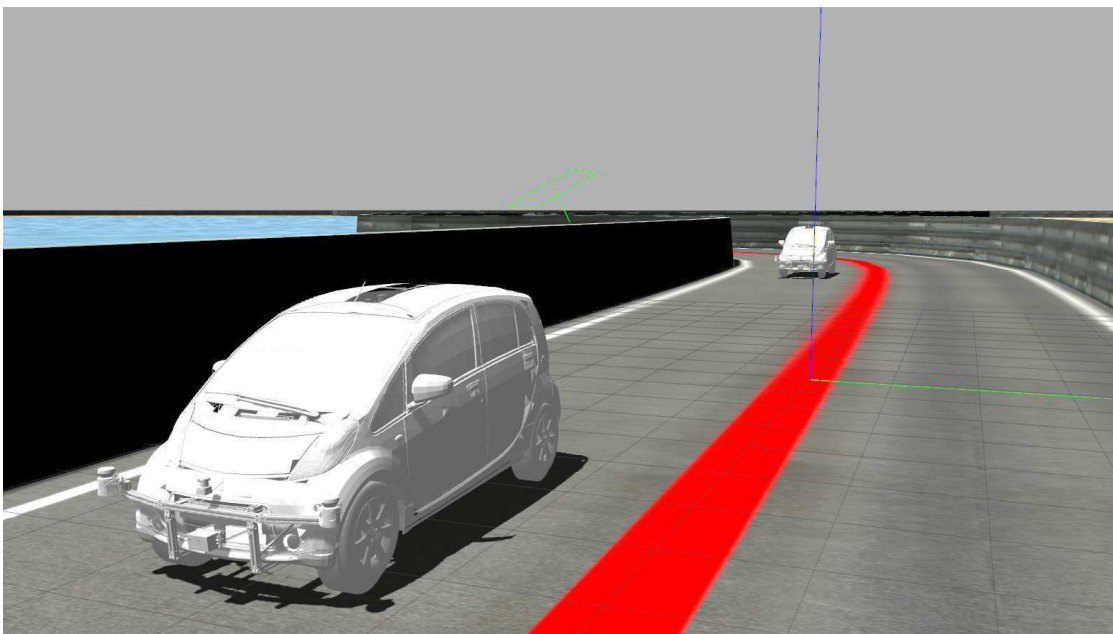


Figura 6.1: Visualização no simulador Gazebo dos dois veículos nas suas posições iniciais

Os dois veículos incluídos na simulação contêm igualmente os seguintes parâmetros iniciais:

- **MAX\_STEERING\_ANGLE = 41.5;**
- **MIN\_STEERING\_ANGLE = 20;**
- **NUM\_NODES = 10;**
- **NUM\_TRAJ = 15;**
- **TRAJ\_DENSITY = 2;**
- **DetectDist = 20;**
- **APdistMin = 5;**
- **APdistMax = 10;**
- **W\_ADAP = 0.20;**
- **W\_DAP = 0.40;**
- **W\_DLO = 0.90;**
- **LINES = true;**
- **OVERTAKING = false;**
- **AP\_right = true;**
- **AP\_left = false;**

As únicas diferenças nos parâmetros iniciais são a velocidade e a ativação do método de detecção, método este que não foi ativado no veículo a ser ultrapassado, sendo considerado desnecessário e reduzindo o impacto computacional durante a simulação. Deste modo os dois veículos têm os seguintes parâmetros:

**Veículo 1** ( veículo que executará a ultrapassagem):

- **SPEED\_REQUIRED = 8.0;**
- **SPEED\_SAFETY = 1;**
- **DETECTION = true;**

**Veículo 2** ( veículo que será ultrapassado):

- **SPEED\_REQUIRED = 5.0;**
- **SPEED\_SAFETY = 1;**
- **DETECTION = false;**

## 6.2 Utilização da nova disposição de trajetórias

Com a utilização da nova disposição de trajetórias influenciadas não só pela velocidade do veículo, mas também pela densidade desejada, como é possível visualizar nas figuras 6.2 e 6.3 onde são comparadas as duas diferentes abordagens, foi possível obter uma maior densidade de trajetórias na direção do veículo, permitindo que este tenha a possibilidade de alterar a sua trajetória de forma mais precisa.

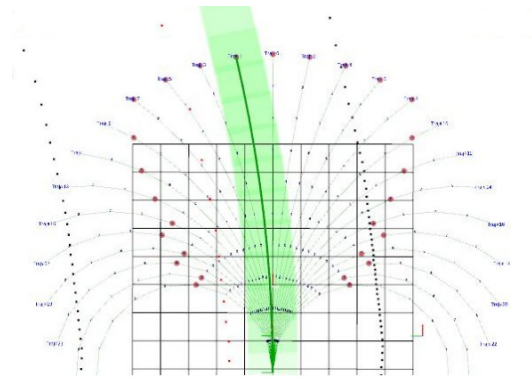


Figura 6.2: Disposição inicial das trajetórias [7]

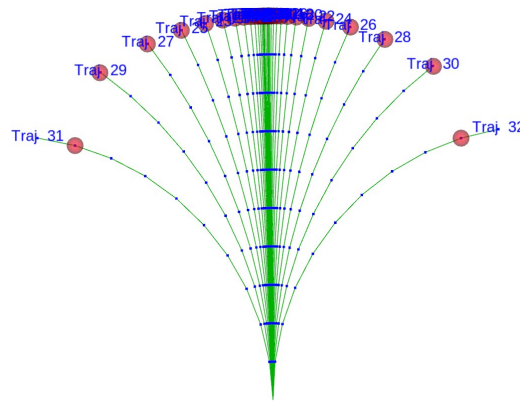


Figura 6.3: Disposição final das trajetórias

Com a variação do ângulo máximo das trajetórias consoante a velocidade, é possível controlar os comportamentos dos veículos, não permitindo grandes alterações da trajetória a velocidades elevadas. Também é possível definir um ângulo mínimo de modo a garantir trajetórias com o ângulo desejado, isto é, existirá uma trajetória com este ângulo mínimo. Na figura 6.4 é possível comparar, para a mesma velocidade, a diferença da disposição das trajetórias consoante o valor mínimo desejado.

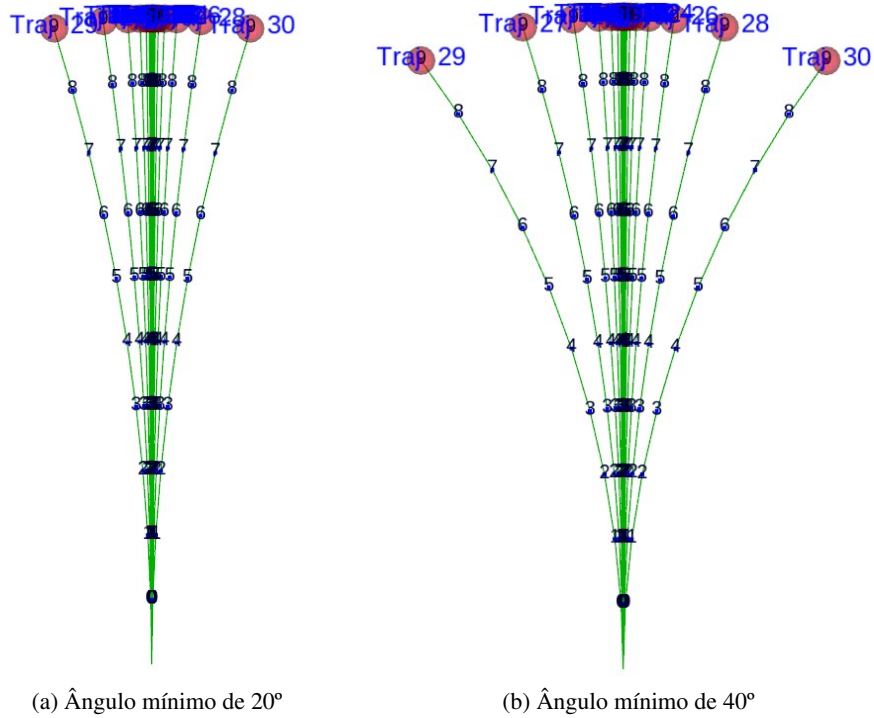


Figura 6.4: Diferença entre valores do parâmetro MIN\_STEERING\_ANGLE

Da mesma maneira, na figura 6.5 é possível verificar a diferença entre a densidade de trajetórias configuradas pelo parâmetro NUM\_TRAJ, sendo que a maior quantidade de trajetórias continua na direção do veículo.

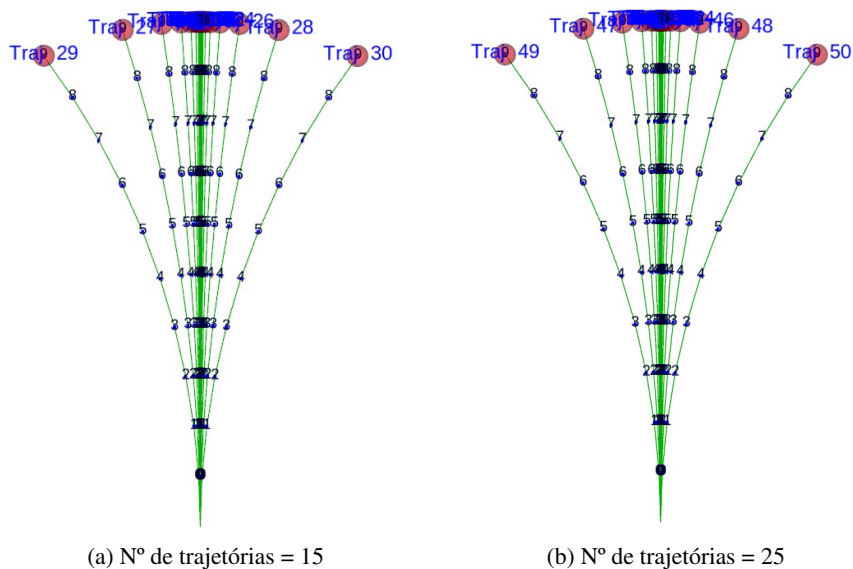


Figura 6.5: Diferença entre valores do parâmetro NUM\_TRAJ



Através do gráfico da figura 6.6 é possível verificar que, durante a ultrapassagem em análise, o ângulo máximo aumenta com a diminuição da velocidade do veículo, tal como esperado, resultados obtidos com a utilização dos parâmetros dinâmicos MAX\_STEERING\_ANGLE e MIN\_STEERING\_ANGLE com um valor de ângulo máximo de  $41,5^\circ$  e ângulo mínimo de  $20^\circ$  e o parâmetro Vel\_Ang ativo.

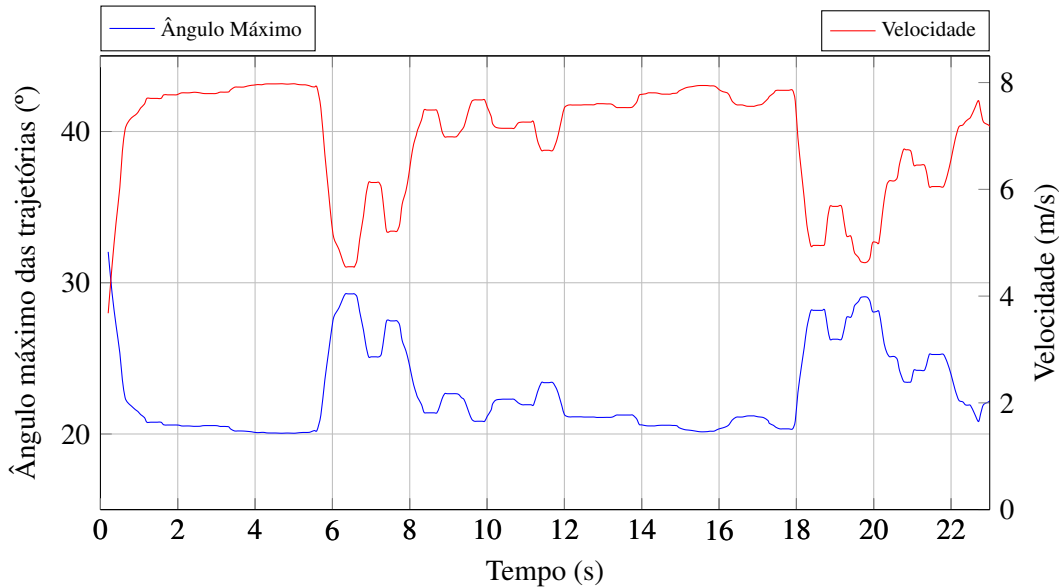


Figura 6.6: Gráfico de comparação entre a velocidade e o ângulo máximo das trajetórias geradas com o parâmetro Vel\_Ang ativo

Com a desativação do parâmetro Vel\_Ang, o veículo raramente abranda, gerando os resultados expostos no gráfico da figura 6.7. É de notar que com a desativação do parâmetro booleano Vel\_Ang, como era previsto, a ultrapassagem decorre num menor intervalo de tempo.

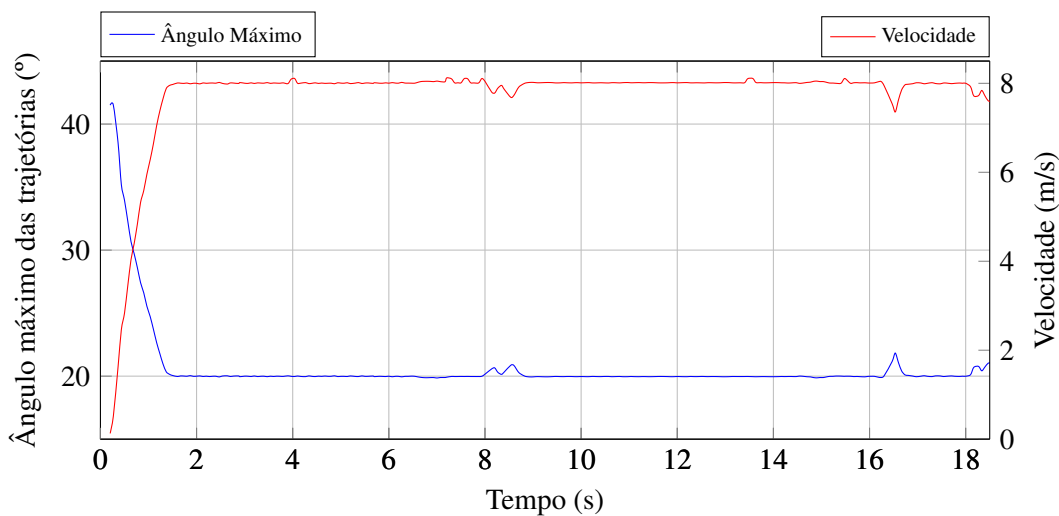


Figura 6.7: Gráfico de comparação entre a velocidade e o ângulo máximo das trajetórias geradas com o parâmetro Vel\_Ang desativo

Em relação à influência da densidade de trajetórias na direção do veículo, através do histograma da figura 6.8, com o parâmetro `Vel_Ang` ativo, é possível verificar que o ângulo da maioria das trajetórias escolhidas não ultrapassava os  $3^\circ$ , recorrendo maioritariamente a ângulos menores a  $1^\circ$ . Deste modo, e após a visualização dos resultados durante a simulação, é possível verificar que o veículo tem um comportamento muito mais estável, apenas recorrendo às trajetórias de ângulos elevados quando necessita de mudar de via ou evitar um obstáculo. É de destacar que, como referido anteriormente, o planeador de trajetórias tem sempre à sua disposição uma trajetória com o ângulo  $20^\circ$ , ângulo mínimo referente ao parâmetro `MIN_STEERING_ANGLE`.

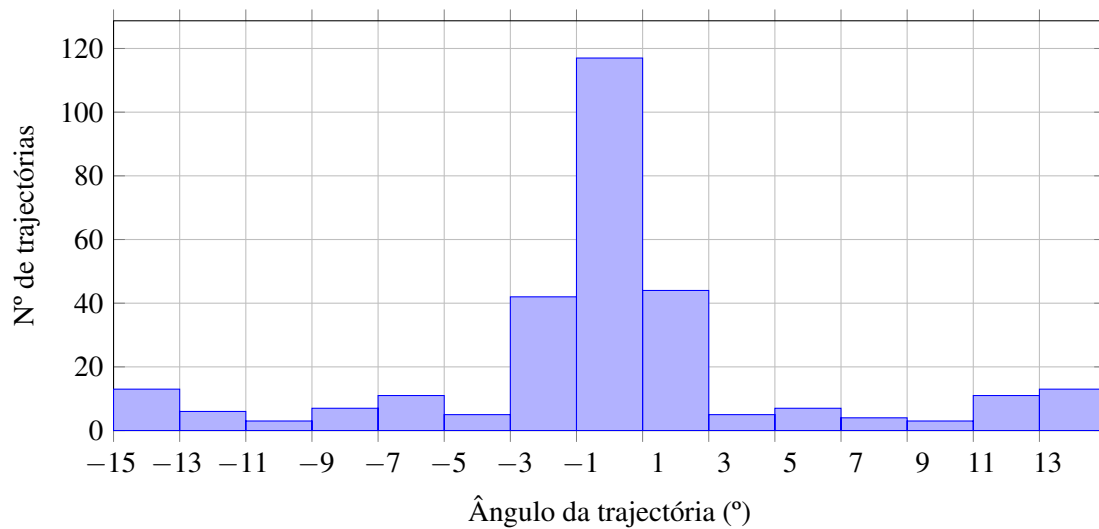


Figura 6.8: Histograma do número de trajetórias para cada intervalo de ângulos

### 6.3 Análise ao ponto atrator

Sendo o ponto atrator o "objetivo" do veículo autónomo, é importante que este seja definido de forma correta, de tal maneira que o veículo circule pelo trajeto desejado. Tendo sido desenvolvido dois sistemas diferentes da utilização do ponto atrator (utilização de *waypoints* e da linha central), estes serão analisados e comparados tendo em consideração a possível utilização de diferentes ambientes.

Tendo em conta que o método da utilização de *waypoints* não se verificou compatível com os resultados desejados, este será analisado tendo em conta os resultados obtidos durante os testes efetuados no seu desenvolvimento.

#### 6.3.1 Ponto atrator utilizando waypoints

Com a utilização de *waypoints* os pontos atratores são estáticos em relação ao referencial global, tal como aconteceria num sistema GPS. Assim, e apesar deste sistema ser mais realista, a utilização destes pontos exige a sua criação para o mapa os utilizar, criando problemas de imprecisão na sua colocação e na distância entre cada um deles, como foi possível verificar no capítulo anterior.

Mesmo com um maior número de *waypoints* nas curvas, o algoritmo torna-se complexo na decisão do *waypoint* a utilizar, sendo necessário cálculos de modo a permitir uma transição

suave entre pontos atratores sem influenciar em demasia a trajetória escolhida, isto é, sem que haja uma grande variação da posição do ponto atrator em relação ao anterior.

Como ponto positivo, este método não necessita da "deteção" da linha central da via, tendo em conta que esta, por enquanto, é apenas possível de ser gerada através da existência de paredes laterais.

Concluindo, a utilização de *waypoints* foi abandonada não só pela dificuldade de processar o local onde colocar o próximo ponto atrator, isto é, a criação da sua metodologia, como também pelas vantagens adquiridas pelo segundo método utilizado.

### 6.3.2 Ponto atrator utilizando a linha central

A criação de um ponto atrator dinâmico que permite a alteração da sua localização dependendo da situação possibilita não só um movimento mais preciso, visto que o objetivo geral é a condução no centro da via, como também permite influenciar o trajeto do veículo auxiliando as manobras a realizar.

Comparando com o ponto atrator inicial que se mantinha a uma distância fixa na direção do veículo, este novo ponto atrator não é influenciado pela orientação do veículo, mantendo-se no centro da via qualquer que seja o comportamento do veículo. Deste modo é evitado o problema inicial dos movimentos oscilantes, onde o veículo seguia em frente apenas se desviando dos obstáculos, tendo agora um objetivo realista a alcançar.

A possibilidade da alteração da sua posição após este ponto atrator ser gerado pelo nó APgenerator permite que tenha grande utilidade nas manobras a realizar, como por exemplo a mudança de via durante a ultrapassagem. Tendo em conta que este ponto atrator tem as suas coordenadas em relação ao referencial do veículo, a alteração no eixo Oy é o suficiente para variar a direção do veículo.

Como ponto negativo, com apenas a utilização de sensores LIDAR, este método torna-se algo irrealista quando o objetivo é a sua utilização do ambiente real. Apesar disto, com a inclusão de câmaras e subsequentemente deteção das linhas da estrada, este método poderá ser replicado facilmente.

## 6.4 Consequências do novo ponto atrator

Tendo sido escolhido o método da geração do ponto atrator através da utilização da linha central, a sua distância ao veículo será analisada em relação à sua velocidade. Para tal, foi utilizado a simulação da ultrapassagem de modo a verificar as alterações da distância em função à mudança de via. Tendo em conta a maior flutuação de velocidades com a ativação do parâmetro *Vel\_Ang*, este parâmetro foi utilizado nas análises seguintes.

Na figura 6.9 é possível visualizar a distância do ponto atrator ao referencial do veículo (azul) e a sua velocidade (vermelho). A tracejado amarelo está representado a mudança de via do ponto atrator, isto é, o valor 0 representa o ponto atrator na via da esquerda, o valor 1 representa o ponto atrator sob a linha central e, por fim, o valor 2 representa o ponto atrator na via da direita.

Através deste gráfico é possível verificar que a velocidade do veículo realmente influencia a distância do ponto atrator, existindo uma redução de velocidade quando o ponto atrator se desloca entre vias, sendo que o ângulo da trajetória escolhida aumenta significativamente.

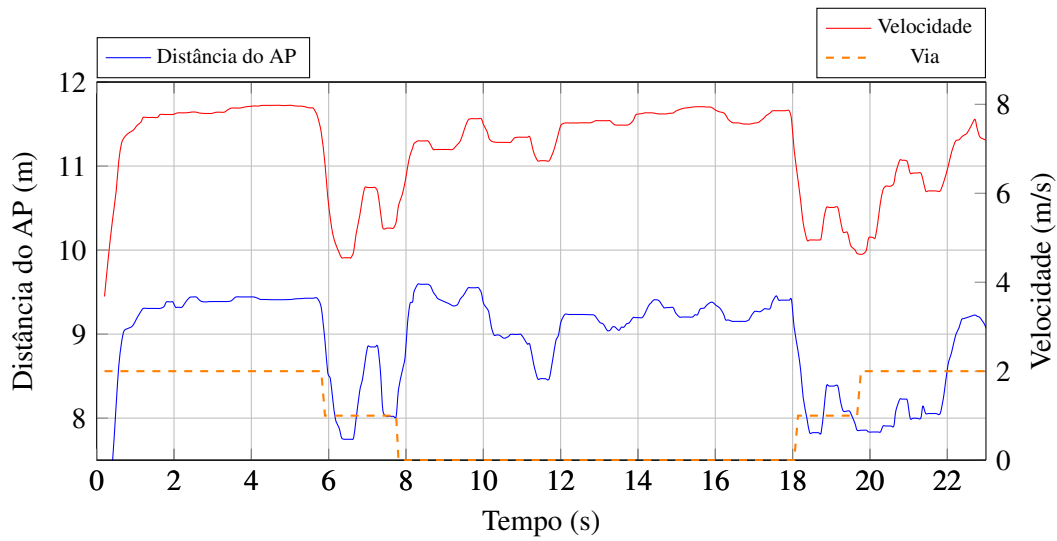


Figura 6.9: Comparação entre a velocidade e a distância do ponto atrator ao longo da ultrapassagem

Tendo sido definido um limite mínimo e máximo para a distância do ponto atrator  $AP_{distMin}$  de 5 metros e  $AP_{distMax}$  de 10 metros respetivamente (parâmetros pertencentes aos parâmetros dinâmicos alteráveis durante a simulação), através do histograma da figura 6.10 é possível visualizar que esta distância se manteve entre os valores 9 e 10, tendo existido reduções desta devido à mudança de via tal como verificado no gráfico anterior.

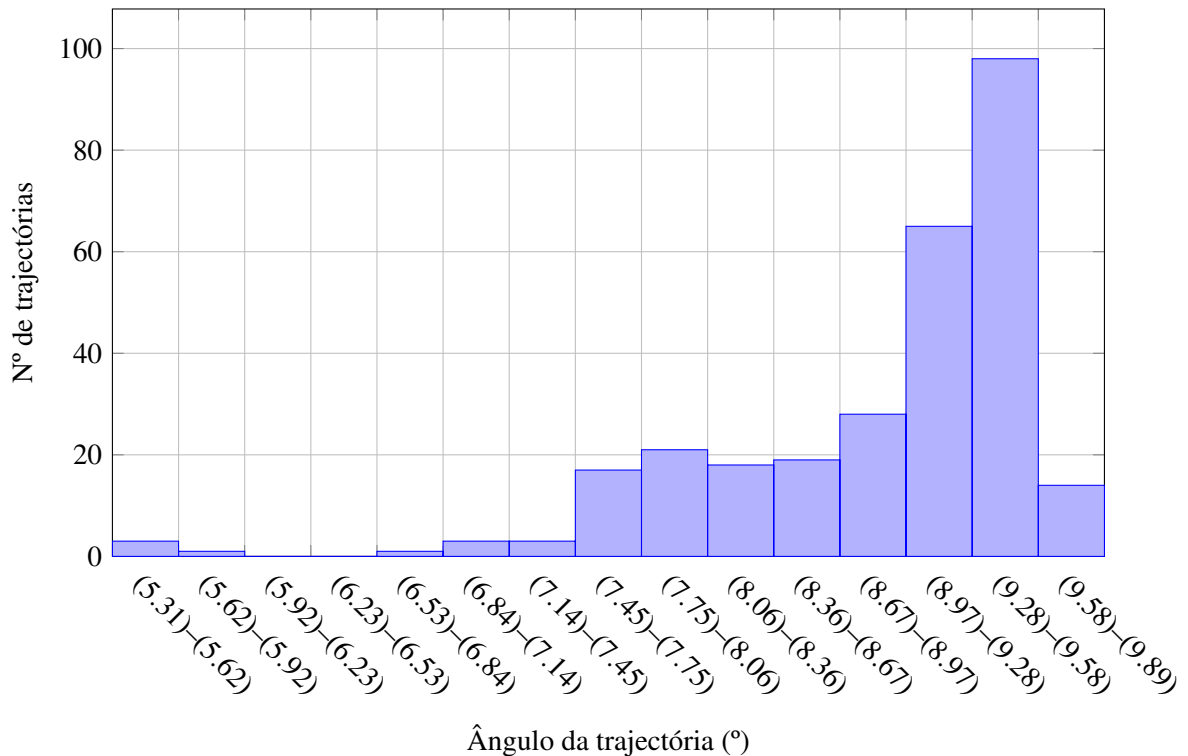


Figura 6.10: Histograma da distância dos pontos atratores gerados durante a ultrapassagem

## 6.5 Zona de deteção de obstáculos

A zona de deteção de obstáculos, apesar de não se revelar uma ferramenta robusta como seria de desejar, permite a análise das manobras criadas. De modo a analisar a eficácia deste método, foram extraídos durante a ultrapassagem, os número de pontos detetados por cada zona de deteção (frontal e traseira).

Na figura 6.11 é apresentado a azul os pontos detetados pela zona de deteção frontal e a vermelho os pontos detetados pela zona de deteção traseira. Ainda é representado neste gráfico o intervalo de tempo em que cada zona de deteção está ativa, sendo a linha vertical laranja o final da utilização da deteção frontal, e as linhas verticais verdes a ativação e desativação da deteção traseira.

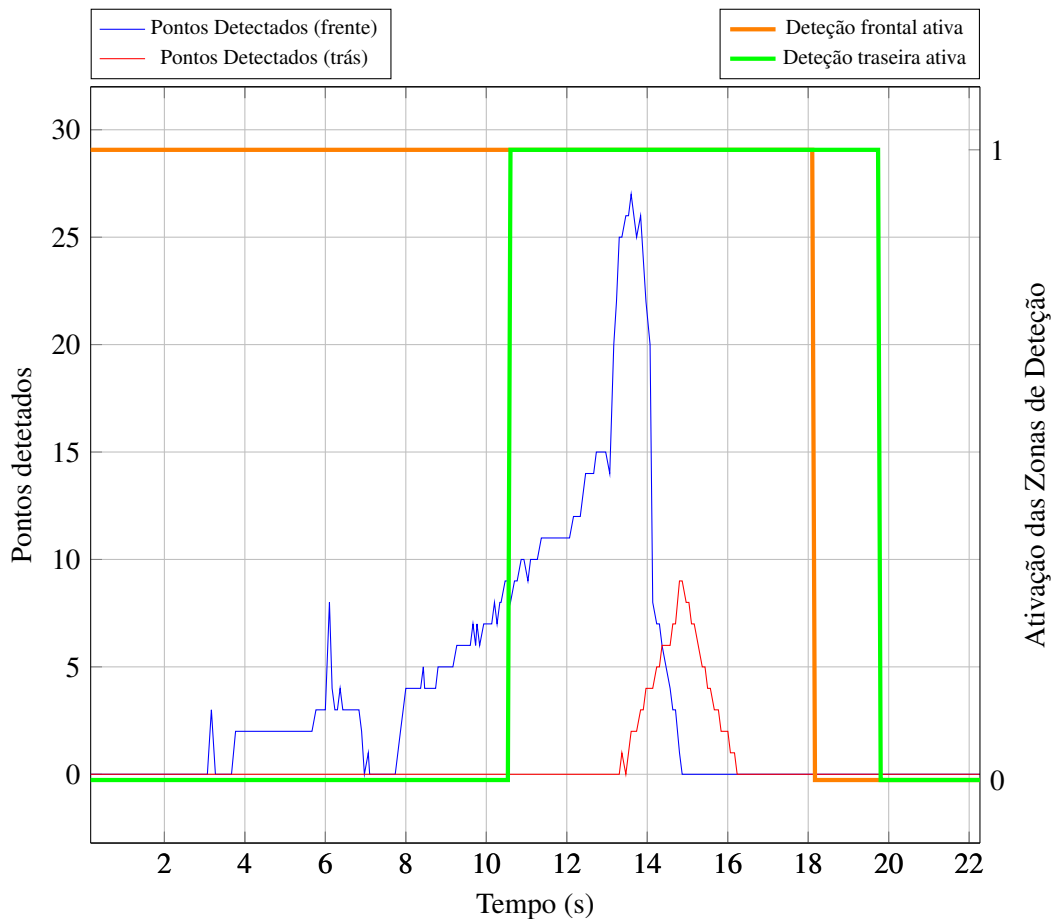


Figura 6.11: Gráfico dos pontos detetados pelas duas zonas de deteção

Por fim, os pontos detetados por ambas as zonas de deteção são agrupados resultando no gráfico da figura 6.12.

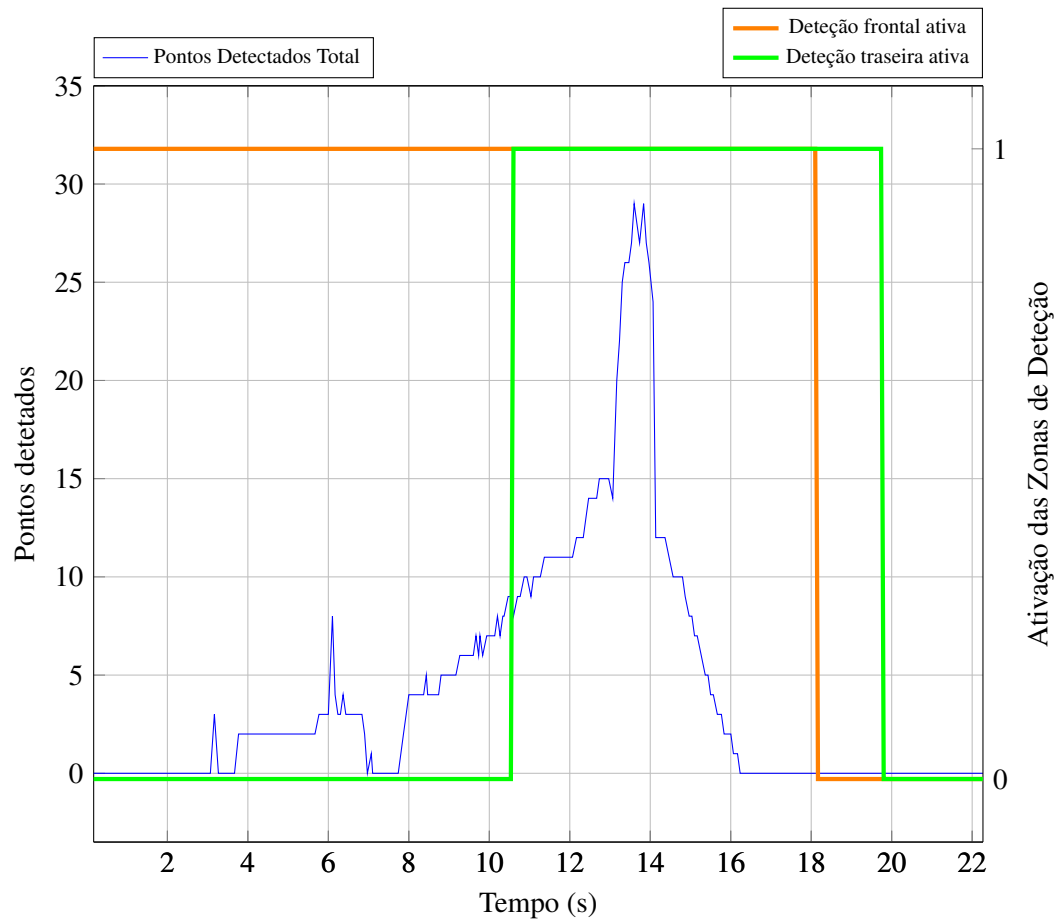


Figura 6.12: Gráfico dos pontos totais detetados pelas duas zonas de deteção

### 6.5.1 Falha na criação do espaço de deteção

Como é possível verificar no gráfico da figura 6.11, a deteção frontal obtém um aumento significativo de pontos detetados. Estes pontos ocorrem, como demonstrado na figura 6.13, quando os dois veículos se encontram paralelos, sendo que o veículo em estudo (veículo a ultrapassar) tem a deteção da parede lateral direita bloqueada pelo segundo veículo.

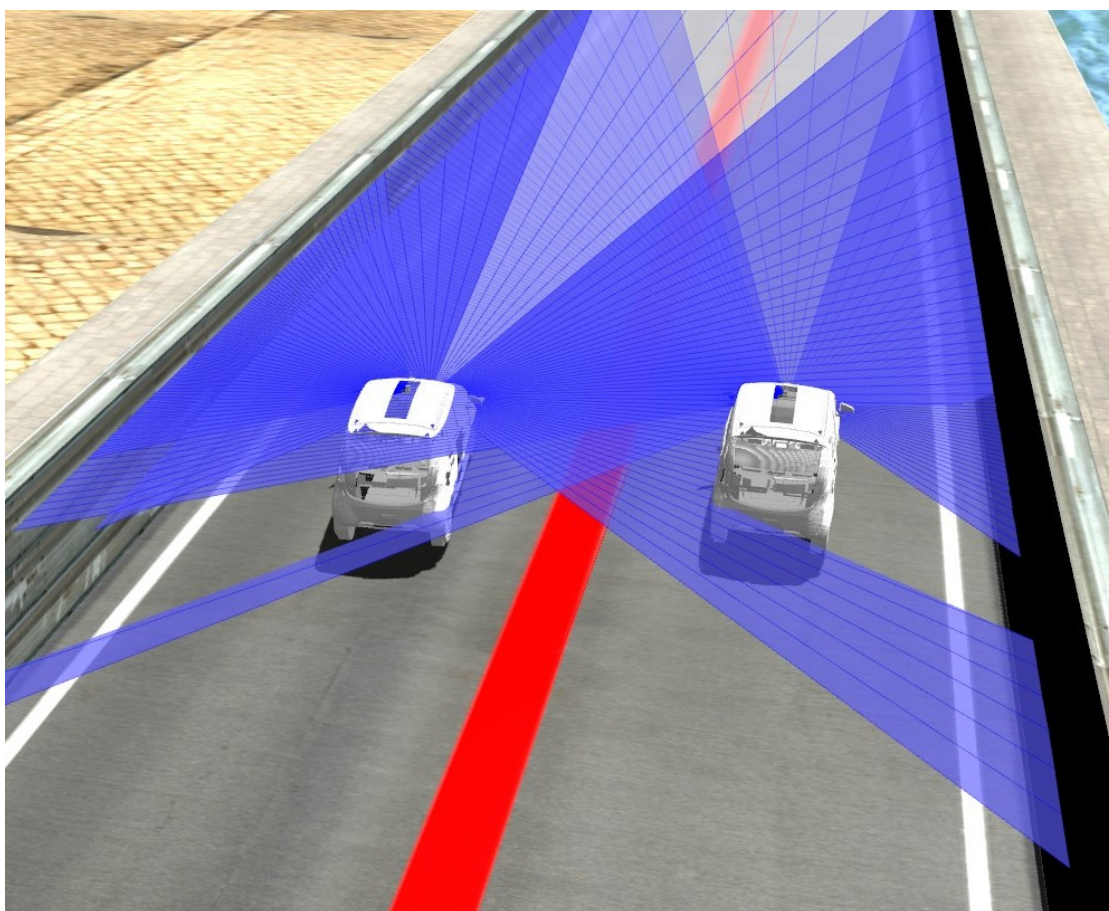


Figura 6.13: Bloqueio da deteção da parede lateral direita ao veículo a ultrapassar por parte do veículo da direita

Devido à metodologia utilizada no cálculo do espaço de deteção, este bloqueio dos sensores por parte do segundo veículo leva a que este cálculo obtenha valores errados, criando um espaço suficiente largo para ser possível de considerar parte dos pontos detetados pelos sensores da parede lateral direita como obstáculos (figura 6.14 e figura 6.15).

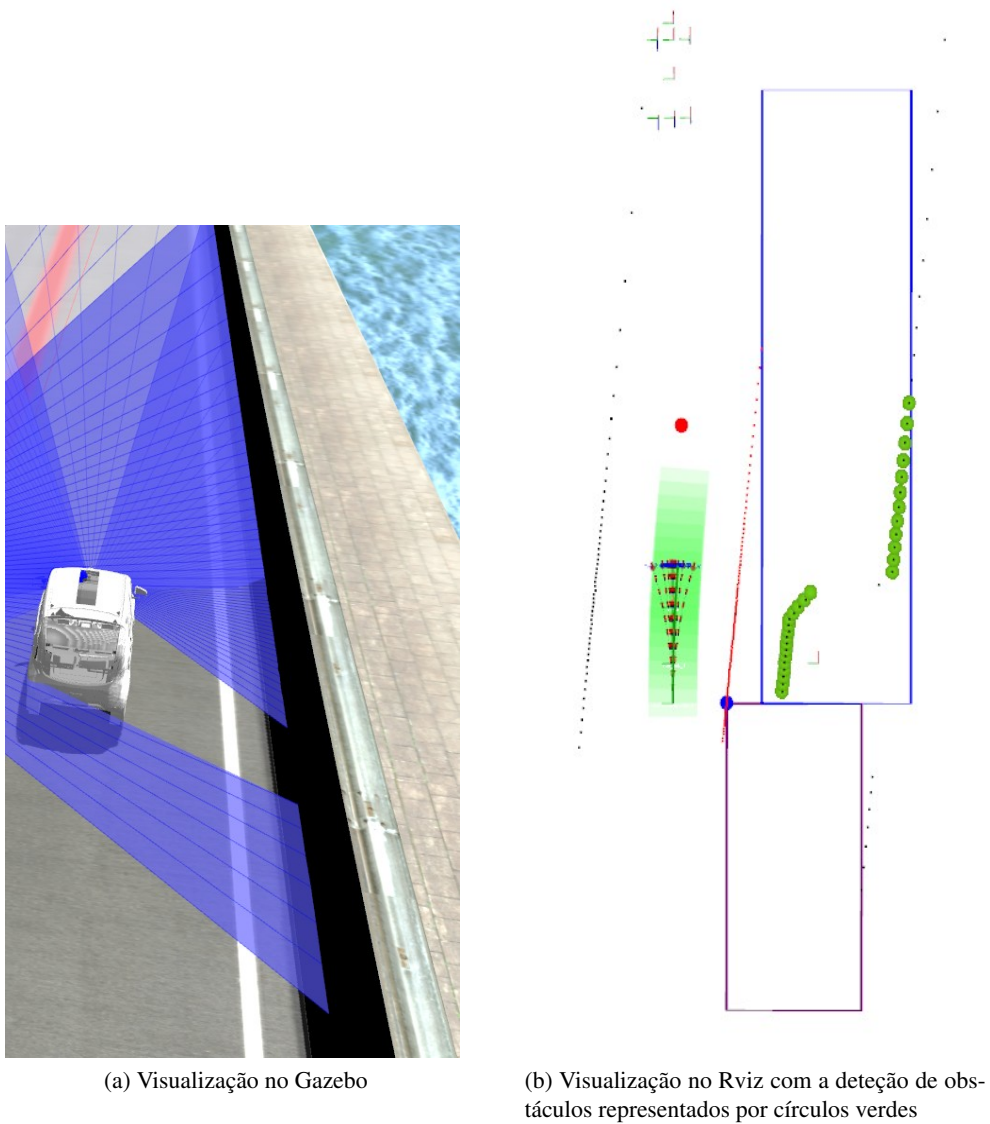


Figura 6.14: Visualização da falha de detecção da parede lateral direita

Este erro, apesar de influenciar o número de pontos detetados, apenas ocorre enquanto os veículos estão paralelos, não comprometendo a manobra a executar, visto que esta recolha de informação apenas serve para informar o veículo a ultrapassar que existem obstáculos na via da direita. Na continuação da manobra a parede volta a ser detetável, retomando a correta criação da zona de deteção.



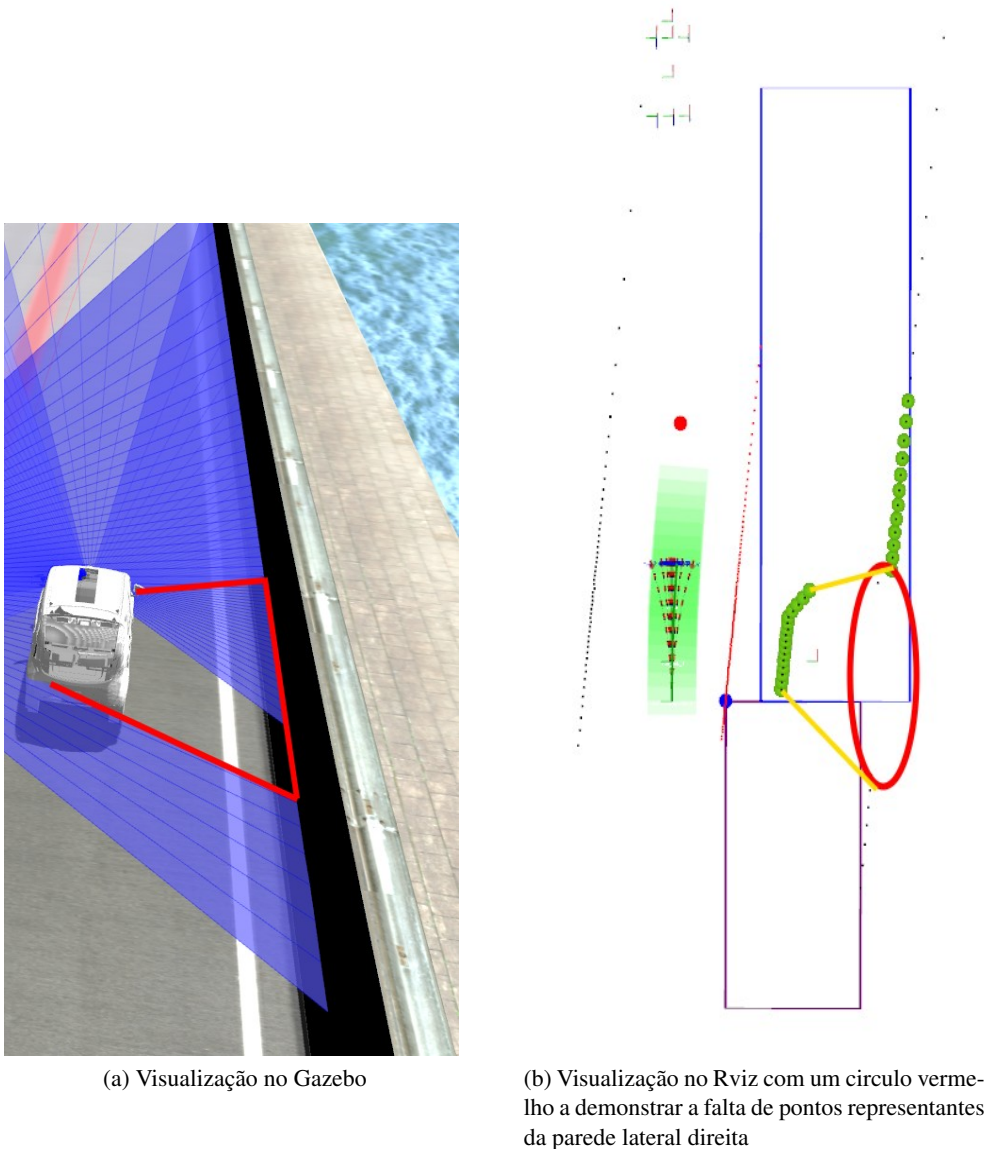


Figura 6.15: Zona onde ocorre a falha de detecção da parede lateral direita

## 6.6 Testes às manobras-tipo

Com as ferramentas criadas e definidas, é agora possível analisar o comportamento do veículo nas manobras pré-definidas.

### 6.6.1 Cruzamento de dois veículos

Na análise de um cruzamento de veículos, ambos possuem as mesmas parametrizações, apenas alterando as suas posições iniciais no "mundo".

Sendo utilizado a linha central como obstáculo durante a circulação geral de um veículo, o cruzamento entre dois veículos em diferentes vias de trânsito decorre sem quaisquer consequên-

cias. Este caso é demonstrado no gráfico da figura 6.16 onde, a linha azul representa a distância mínima ao obstáculo mais próximo do veículo nº1 e a linha vermelha representa a distância mínima ao obstáculo mais próximo do veículo nº2. Nas linhas verde e amarela estão representados os pontos detetados pelos veículos 1 e 2 respetivamente. Como é possível verificar, os veículos são mutuamente detetados, não resultando em nenhum desvio por parte dos mesmos.

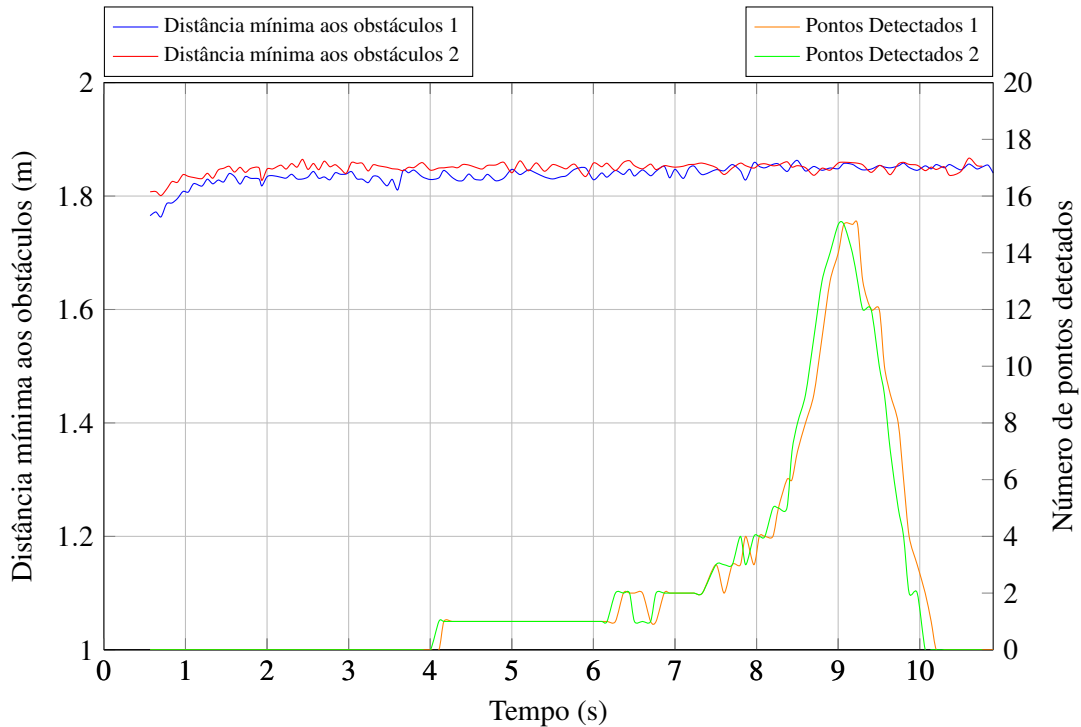


Figura 6.16: Distância mínima aos obstáculos durante um cruzamento de 2 veículos

### 6.6.2 Ultrapassagem

Para a análise da ultrapassagem, foi utilizada a mesma simulação das análises anteriores:

**Veículo 1** ( veículo que executará a ultrapassagem):

- **SPEED\_REQUIRED = 8.0;**
- **DETECTION = true;**
- **Vel\_Ang = false;**

**Veículo 2** ( veículo que será ultrapassado)

- **SPEED\_REQUIRED = 5.0;**
- **DETECTION = false;**
- **Vel\_Ang = false;**

Sendo esta análise efetuada numa parte reta da pista, o parâmetro `Vel_Ang` foi desativado de modo a permitir resultados mais coerentes com a manobra a realizar. Ao desativar o parâmetro `Vel_Ang`, o veículo não irá compensar a sua velocidade em relação com o ângulo da trajetória, permitindo deste modo que o veículo que executará a ultrapassagem não abrande ao mudar de via.

Numa primeira fase será analisada a distância mínima a todos os obstáculos durante a ultrapassagem. Para tal, no gráfico da figura 6.17 esta distância está representada a azul, estando também representado a verde e a laranja o número de pontos detetados pela zona de deteção frontal e traseira respetivamente. A vermelho está representado a ativação e desativação do parâmetro `LINES`, parâmetro este que ativa e desativa a linha central como obstáculo.

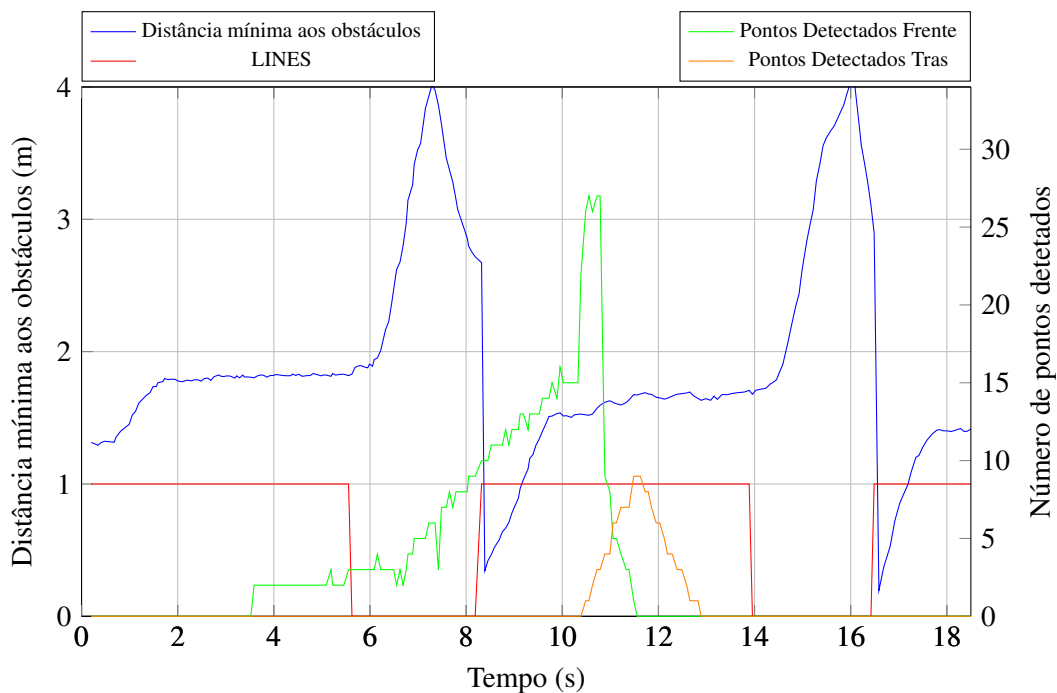


Figura 6.17: Distância mínima aos obstáculos durante uma ultrapassagem

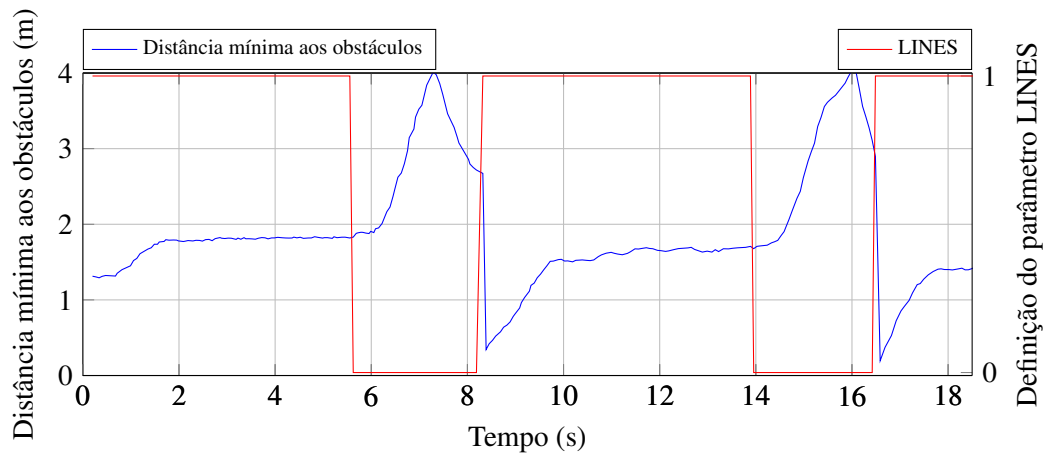


Figura 6.18: Relação entre a distância mínima aos obstáculos e a ativação do parâmetro LINES

Como é possível verificar no gráfico da figura 6.17 existe uma diminuição acentuada da distância mínima quando é ativado o parâmetro LINES (valor passa de 0 a 1), tendo em conta que este parâmetro é ativado quando o veículo se encontra a uma distância superior a 1 metro da linha central. Assim, no gráfico da figura 6.18 está exposto de uma forma mais simples a influência do parâmetro LINES, correspondendo aos valores esperados.

A ativação do parâmetro LINES está dependente da distância à linha central (maior ou igual a 1), sendo demonstrado o seu comportamento correto no gráfico da figura 6.19. Neste gráfico é demonstrado na linha vermelha a distância do veículo à linha central, utilizando para referência 3 linhas horizontais representando os valores 1, 0 e -1 a verde, amarelo e laranja respetivamente.

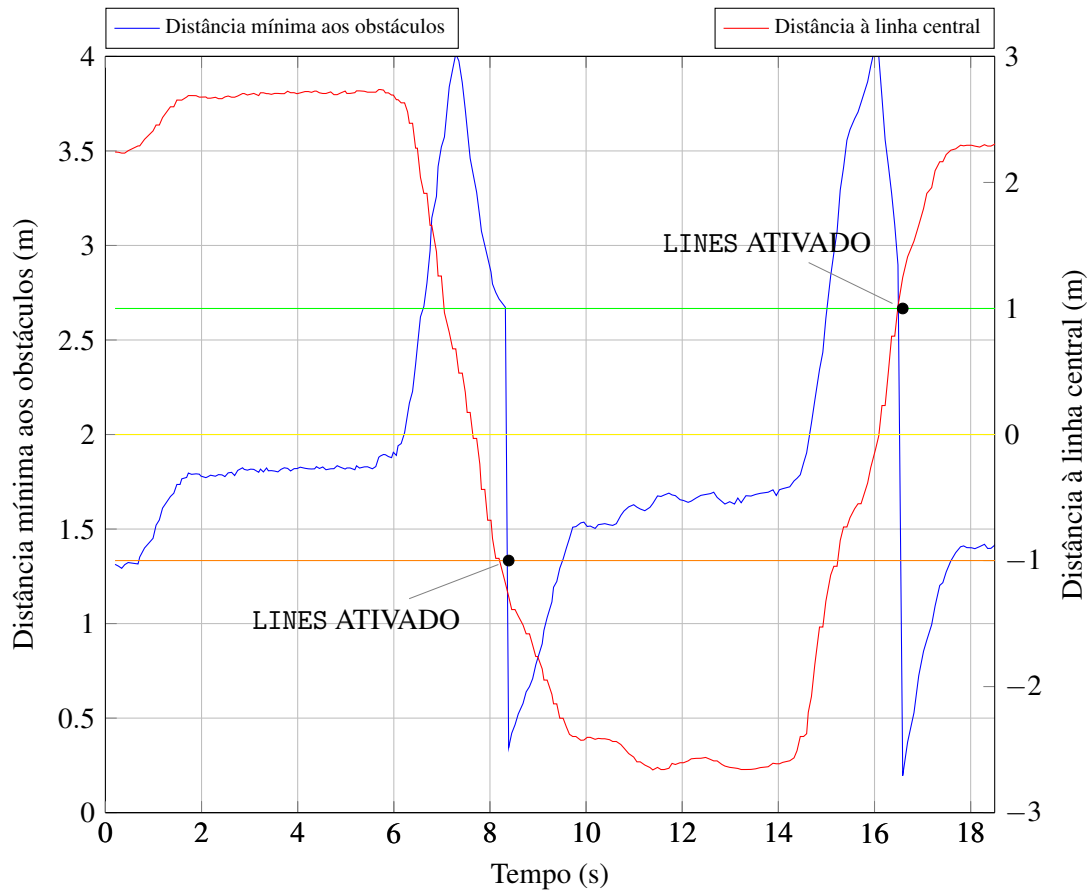


Figura 6.19: Ativação do parâmetro LINES de acordo com a distância à linha central

Após estas análises, e tendo em conta que o maior objetivo é obter a distância entre os dois veículos, no gráfico da figura 6.20 é demonstrado a distância em relação ao veículo 1 (veículo a ultrapassar) ao ponto mais próximo detetado do veículo 2 (veículo a ser ultrapassado).

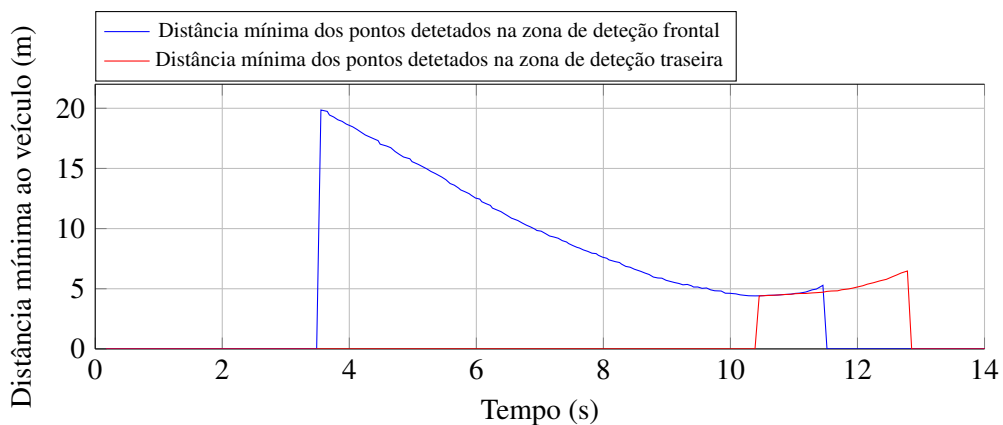


Figura 6.20: Distância mínima ao veículo a ser ultrapassado

Como é de esperar os resultados começam a uma distância de 20 metros, que foi definida

como limite máximo da zona de detecção através do parâmetro DetectDist. Com uma linha azul está representada a distância detetada pela zona de detecção frontal, enquanto que na linha vermelha está representado a distância detetada pela zona de detecção traseira. Durante esta ultrapassagem a distância mínima atingida é de 4,40 metros, correspondente à fase onde os veículos se encontram paralelos.

Com velocidades mais reduzidas os resultados tornam-se ainda mais estáveis, tal não acontecendo ao aumentar a velocidade do veículo a ultrapassar, comportando-se com movimentos oscilantes e deteções incorretas das zonas de detecção. Estes erros são resultado da velocidade da informação gerada pelos vários nós e fornecida ao veículo, consequência do limite do poder computacional disponível.

### 6.6.3 Comportamento do veículo ultrapassado

O veículo ultrapassado permaneceu com um comportamento estável não sendo influenciado pela ultrapassagem. No gráfico da figura 6.21 estão representados os valores principais deste veículo ao longo da ultrapassagem, mostrando a estabilidade do veículo ao longo do tempo.

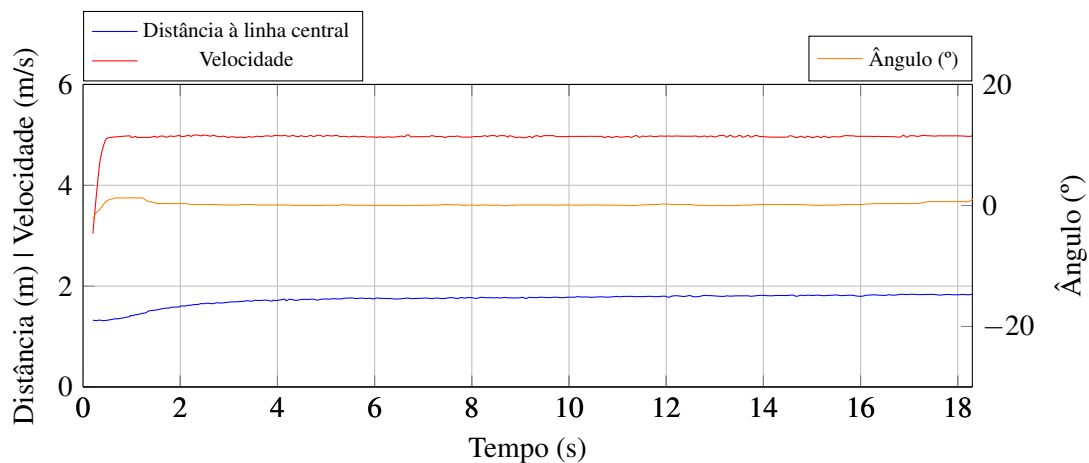


Figura 6.21: Comportamento do veículo ultrapassado

### 6.6.4 Ultrapassagem a múltiplos veículos

Com o objetivo de analisar o desempenho do método de detecção durante a ultrapassagem, mais precisamente, quando o veículo circula na via da esquerda, foi simulado uma ultrapassagem a três veículos iguais, a circular em à mesma velocidade de 3 metros por segundo, como é possível visualizar na figura 6.22. Aqui o parâmetro Ang\_Ve1 está desativado para o veículo que executará a ultrapassagem.

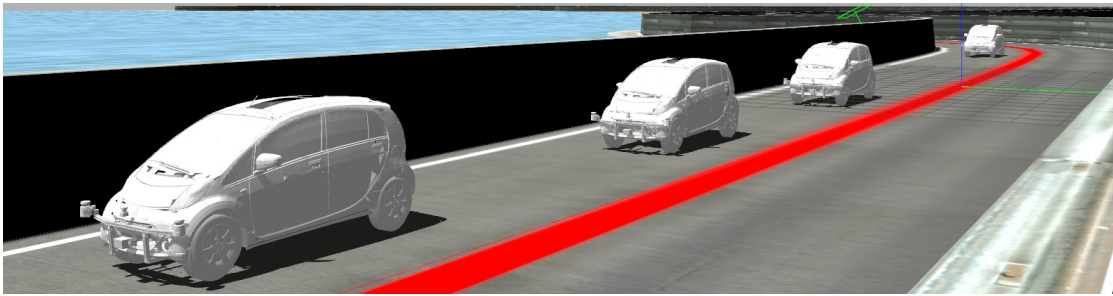
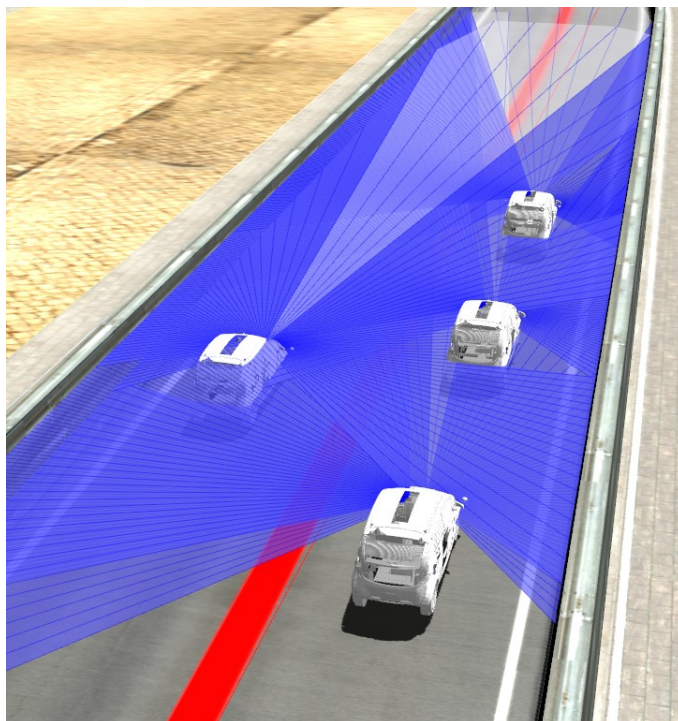
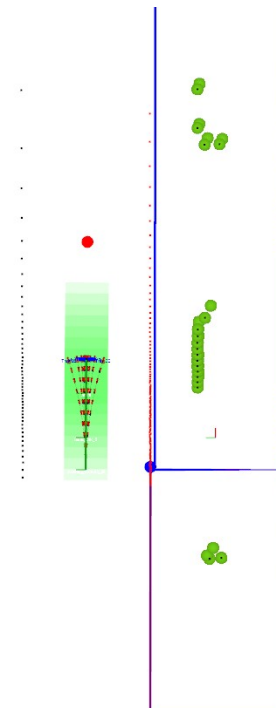


Figura 6.22: Ultrapassagem a múltiplos veículos

Na figura 6.23 é demonstrado a visualização no simulador e no Rviz da primeira ultrapassagem, onde o veículo em análise já deteta todos os três veículos a ultrapassar. Na figura 6.24 é possível visualizar a continuidade do movimento do veículo pela via da esquerda detetando os restantes dois veículos por ultrapassar.



(a) Visualização no Gazebo



(b) Visualização no Rviz da deteção dos três veículos

Figura 6.23: Ultrapassagem do primeiro veículo

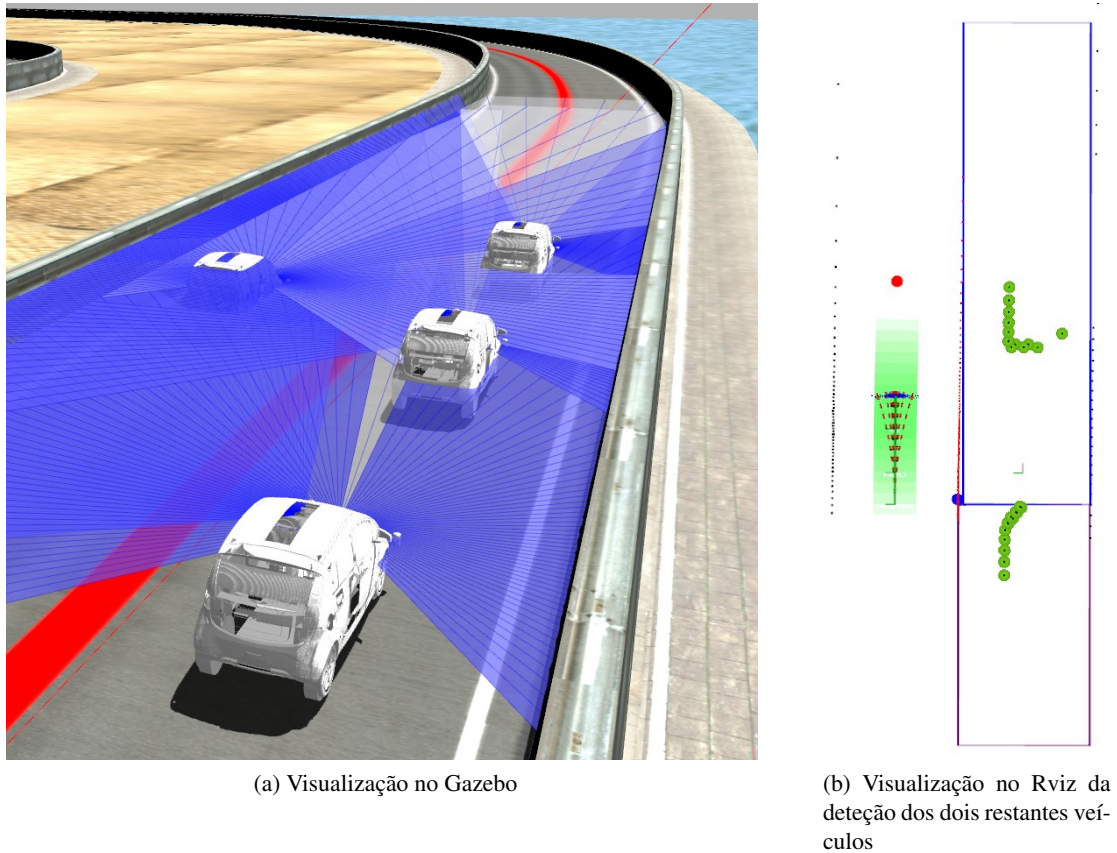


Figura 6.24: Ultrapassagem do segundo veículo

Através desta simulação, e como é possível comprovar no gráfico da figura 6.25, o veículo em análise deteta corretamente os veículos presentes na via da direita permanecendo o erro na ativação da linha central como obstáculo, com o verificado no gráfico da figura 6.26.

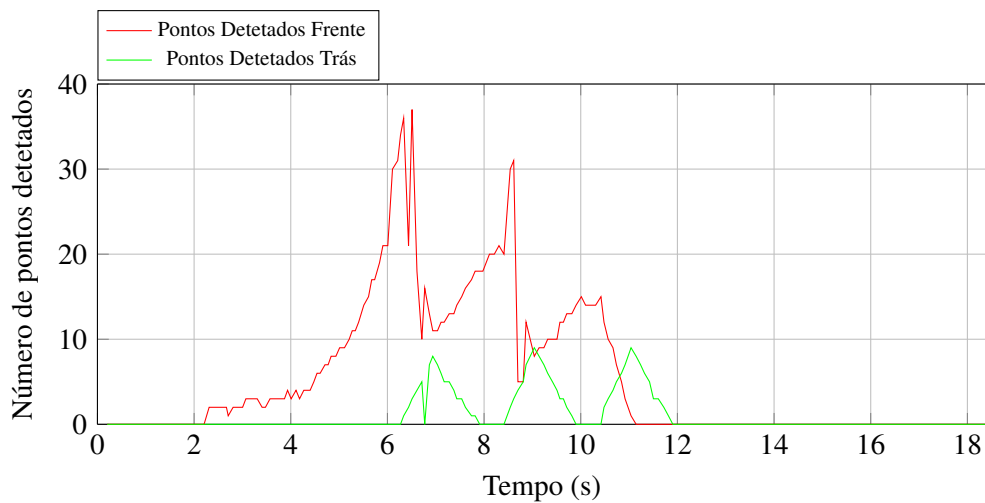


Figura 6.25: Distância mínima aos obstáculos durante a ultrapassagem de três veículos



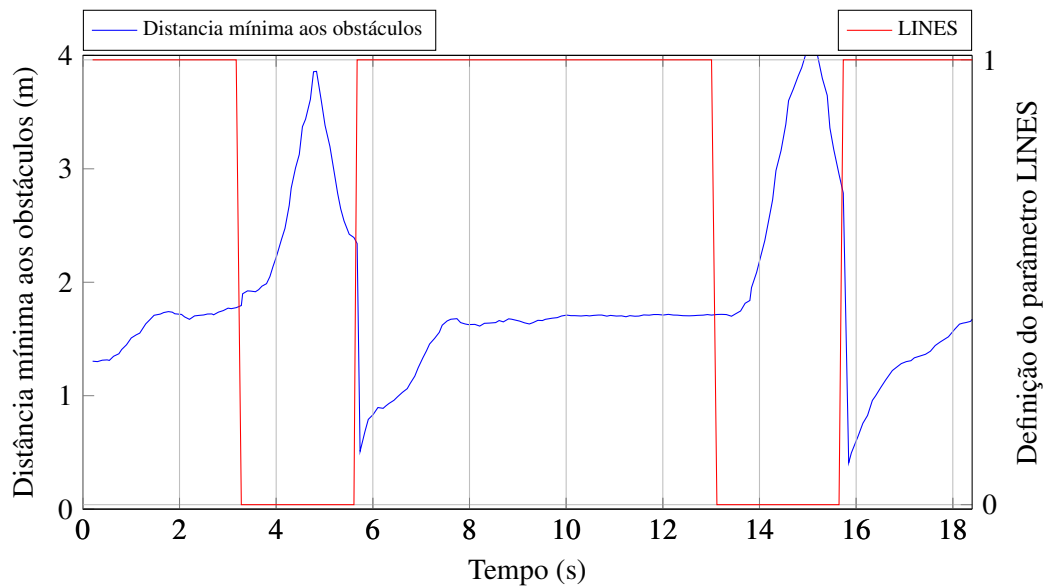


Figura 6.26: Relação entre a distância mínima aos obstáculos e a ativação do parâmetro LINES durante a ultrapassagem de três veículos

Através do gráfico da figura 6.27 é possível comprovar que o veículo apenas inicia a retoma para a via da direita após deixar de detetar obstáculos através das zonas de deteção, esperando o tempo pré determinado `WaitingTime` antes de realizar esta mudança de via.

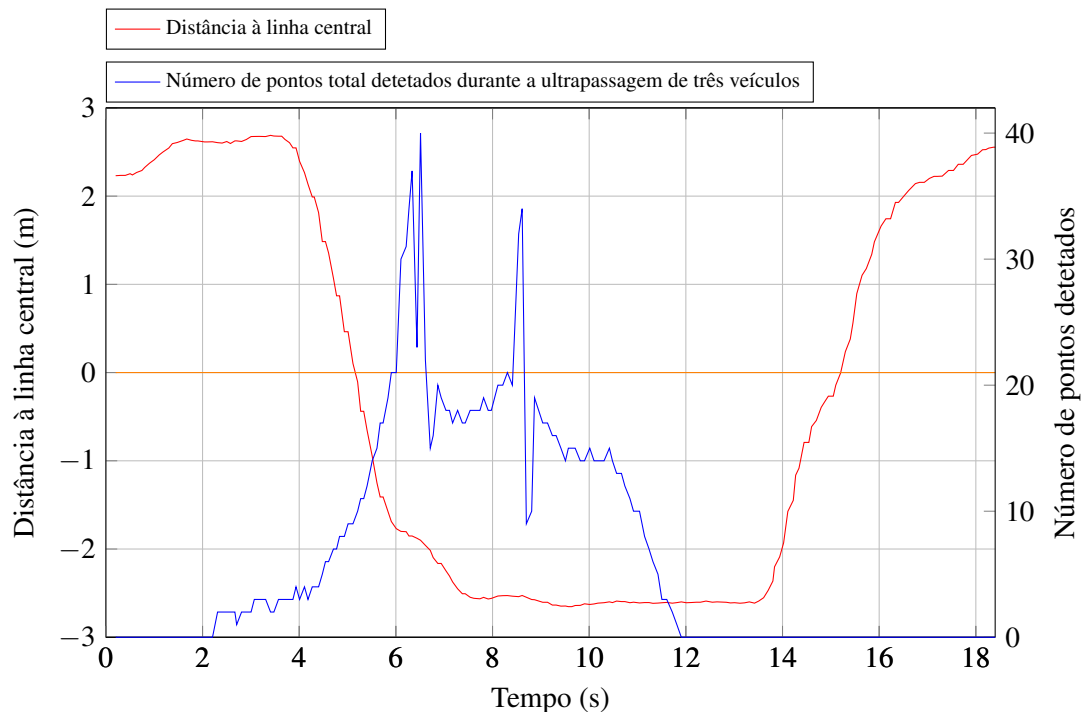


Figura 6.27: Distância à linha central consoante os pontos detetados

Para finalizar, no gráfico da figura 6.28 estão representadas as distâncias mínimas aos veículos ultrapassados comprovando os resultados obtidos na análise da ultrapassagem anterior.

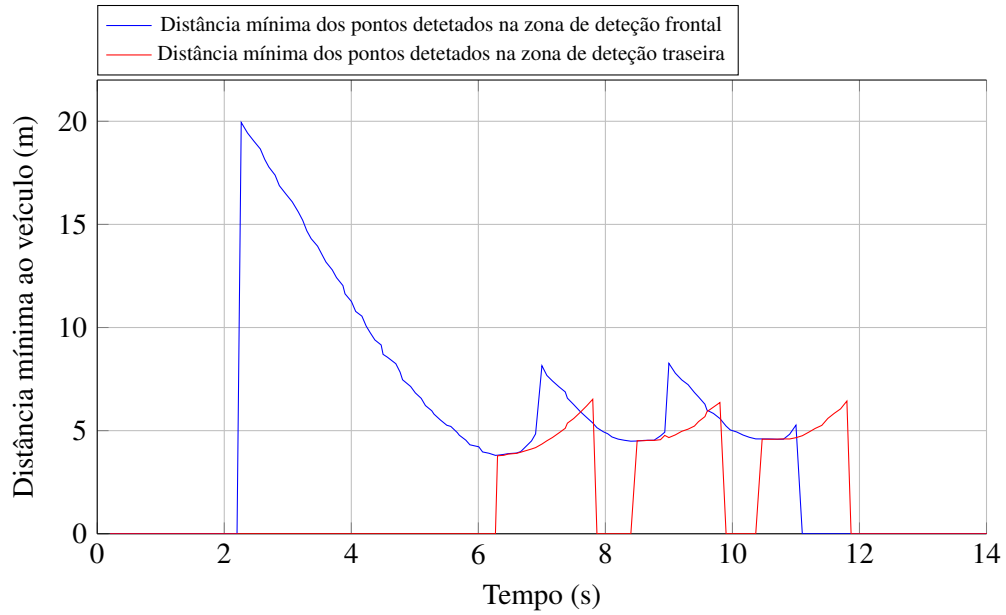


Figura 6.28: Distância mínima aos veículos a serem ultrapassados

## Capítulo 7

# Conclusões e trabalhos futuros

Neste capítulo será feita uma reflexão sobre os problemas e objetivos iniciais e o nível de sucesso na resolução dos mesmos. Será também resumida a metodologia utilizada e a sua adequação face aos resultados obtidos. Por fim será analisado o grau de cumprimento dos objetivos propostos tendo em foco as contribuições dadas com este projeto e as possibilidades de trabalhos futuros.

### 7.1 Problemas e objetivos iniciais

Esta dissertação comportava três objetivos principais que consistiam: na implementação da infraestrutura de simulação computacional em ambientes com múltiplos veículos; na definição, estudo e parametrização de trajetórias e métricas de avaliação e seleção em contexto de múltiplos veículos; e em testes de verificação, e se possível avaliação, das técnicas desenvolvidas em situação real a bordo do carro.

Sendo este um projeto focado na simulação de múltiplos veículos, o primeiro objetivo desta dissertação teve como foco o estudo da possibilidade da utilização de múltiplos agentes num único ambiente de simulação, utilizando para tal o simulador Gazebo, ou caso este não o permitisse, seria necessário encontrar um simulador que não tivesse esta restrição. Com este estudo realizado era então fundamental fazer as alterações necessárias para o sucesso deste objetivo.

O segundo objetivo desta dissertação previa a criação de parametrizações que permitissem ao veículo autónomo executar manobras pré definidas consoante os dados recolhidos pelos sensores, criando para tal situações-tipo que substituiriam os cálculos gerais das trajetórias dependendo da situação detetada.

Por último estas novas parametrizações seriam testadas a bordo do ATLASCAR2 comparando o comportamento do algoritmo com a condução do piloto, tendo em conta que este projeto ainda se encontra em fases iniciais não sendo viável o controlo do veículo pelo planeador de trajetórias desenvolvido.

### 7.2 Metodologia

Esta dissertação teve início com base num projeto passado [7] onde foi desenvolvido um planeador de trajetórias para o ATLASCAR2. Assim, o primeiro passo realizado foi recrear a simulação deste projeto, sendo para tal necessário a sua conversão para a versão mais recente do ROS e atualização do circuito utilizado na simulação. Com o ambiente de simulação a funcionar corretamente e com recurso a diversos tutoriais, foi reformulada a arquitetura dos ficheiros de

modo a permitir a utilização de múltiplos veículos no simulador Gazebo. Este objetivo criou grandes dificuldades iniciais tendo em conta a inexperiência do aluno e o número de ficheiros e detalhes necessários de serem alterados para a correta interligação de todos os sistemas (nós e tópicos).

Concluído o primeiro objetivo, foi necessário preparar o ambiente de simulação para que o desenvolvimento de situações pré definidas pudessem ser analisadas e desenvolvidas.

Para tal foram efetuadas diversas alterações no sistema existente, começando pela alteração do funcionamento do ponto atrator interativo existente, de modo a que este não necessitasse de uma primeira interação do programador. De seguida foi alterado o funcionamento inicial da simulação, obrigando os nós a "esperar" pelo início da simulação através do Gazebo, e por fim foi alterado o modelo do ATLASCAR2, ajustando quer o seu modelo visual, quer o seu modelo de colisões.

Complementando as ferramentas existentes durante a simulação, foi criado um programa que permite configurar dinamicamente os parâmetros que influenciam a simulação, isto é, estes parâmetros podem ser alterados durante a execução da simulação, de modo a analisar a sua influência no cálculo da trajetória.

Com o objetivo de simular as manobras necessárias foi desenvolvido uma solução para a navegação do veículo autónomo pela via da direita utilizando para tal a linha central, acoplando a este a reformulação do ponto atrator, permitindo melhorar substancialmente a parametrização destas manobras.

A conclusão deste segundo objetivo verificou-se com a parametrização de manobras-tipo, nomeadamente do cruzamento entre dois veículos e da ultrapassagem, ficando por testar estas manobras num ambiente real.

### 7.3 Avaliação do trabalho realizado e Contribuições

Avaliando o trabalho desenvolvido neste projeto, os resultados obtidos no primeiro objetivo foram muito satisfatórios, faltando apenas uma configuração mais automática na geração de veículos, não tendo sido desenvolvida devido à necessidade da constante alteração dos parâmetros da simulação. Em termos de organização dos ficheiros envolvidos foi procurado ter em conta a simplicidade da arquitetura e a qualidade da informação disponível de modo a permitir uma fácil análise ao algoritmo.

Quanto ao segundo objetivo, a reformulação do planeador de trajetórias teve grande impacto nos resultados, permitindo a criação de melhores trajetórias ao longo do percurso. As manobras-tipo desenvolvidas ficaram restritas às ferramentas disponíveis (não utilização de câmaras visuais para detetar os limites da estrada e a linha central), mesmo assim obtendo resultados satisfatórios.

Esta dissertação deixa uma contribuição importante no desenvolvimento do planeador de trajetórias de um veículo, sendo que a criação de uma arquitetura de múltiplos agentes pode ser analisada e replicada para diversos projetos, permitindo obter uma base fundamental no estudo e desenvolvimento de futuros trabalhos que necessitem desta vertente. Esta parte inicial da dissertação poderá ter um grande impacto tendo em conta a inexistência de soluções semelhantes, tal como verificado no capítulo "Revisão da Literatura" onde o projeto mais próximo de obter resultados positivos (Huskys [18]) não se encontrava concluído.

Para além desta arquitetura, esta dissertação reformulou o planeador de trajetórias, tornando-o mais dinâmico e permitindo um desenvolvimento futuro mais acessível, com foco na descrição

do código utilizado permitindo uma rápida adaptação ao mesmo.

Um marco importante neste projeto relaciona-se com o ponto atrator dinâmico, tendo-se provado uma ferramenta bastante promissora no planeamento de manobras podendo ainda ser integrado com os dados GPS recebidos pelo ATLASCAR2. Este ponto atrator não só proporcionou soluções a diversos problemas durante o desenvolvimento do planeador de trajetórias, como também corrigiu comportamentos inadequados existentes na utilização da sua versão anterior.

Por fim a criação das duas manobras-tipo permitiram tornar a simulação muito mais robusta, tendo a parametrização da manobra de ultrapassagem, apesar das suas imperfeições, fornecido um avanço importante ao algoritmo.

Resumindo, a dificuldade encontrada na resolução de uma arquitetura de múltiplos agentes no simulador Gazebo afetou o plano de trabalhos proposto no início desta dissertação, quase comprometendo assim os resultados a obter nas parametrizações de manobras-tipo. Ainda assim, e tendo em conta os desafios encontrados no desenvolvimento desta dissertação, os dois objetivos principais consideram-se cumpridos, sendo que a parametrização de manobras compreende um leque muito grande de possibilidades, deixando concluído as ferramentas necessárias para o desenvolvimento das mesmas.

## 7.4 Trabalhos futuros

Com o desenvolvimento do ambiente de simulação de múltiplos veículos, foi criado uma grande variedade de simulações possíveis de serem realizadas, permitindo aprofundar o planeamento de trajetórias em diferentes situações. Ainda assim, o simulador poderá ser aperfeiçoado com a alteração dos sensores LIDAR, alterando a utilização única do sensor na fronteira do veículo, para dois sensores colocados na posição real de acordo com a sua disposição no ATLASCAR2. Esta alteração não só permitiria uma melhor recolha de dados nas laterais do veículo, obtendo melhores informações sobre os obstáculos durante ultrapassagens, etc., como também permitiria a utilização de um modelo de colisão mais realista, sendo necessário ter em conta as necessidades computacionais desta alteração.

Outra alteração possível de ser realizada seria a colocação de um sensor LIDAR na retaguarda do veículo, permitindo obter informação e avaliar o comportamento dos veículos que se aproximam.

Tratando-se de um ambiente de múltiplos veículos, o projeto desenvolvido por Jorge Almeida (MTT [38]) poderá ter grande valor na aquisição de dados (posição e velocidade) sobre os veículos e pedestres que circulem na proximidade do veículo autónomo, auxiliando o processo de decisão e execução da manobra necessária.

Com a evolução do poder computacional poderão ser analisadas situações com um maior número de veículos, algo ainda difícil de realizar.



# Referências

- [1] Neels Christopher. The importance of incentives in the development of autonomous vehicles. <https://medium.com/predict/the-importance-of-incentives-in-the-development-of-autonomous-vehicles-967409458597>. Último acesso 21 Maio 2019.
- [2] Priya Dwivedi. Planning the path for a self-driving car on a highway. <https://towardsdatascience.com/planning-the-path-for-a-self-driving-car-on-a-highway-7134fddd8707>. Último acesso 21 Maio 2019.
- [3] Skott. How to overtake – safely. <http://www.qldroadsafety.com/index.php/2018/11/19/how-to-overtake-safely>. Último acesso 18 Maio 2019.
- [4] Robert Harwood. Simulation is the only way to test autonomous vehicles for millions of miles. <https://www.ansys.com/blog/simulation-can-test-autonomous-vehicles-for-millions-of-miles>. Último acesso 21 Maio 2019.
- [5] Michael Sarazen Alex Chen. Simulations pave the road for self-driving technologies. <https://medium.com/syncedreview/simulations-pave-the-road-for-self-driving-technologies-78b696227383>. Último acesso 21 Maio 2019.
- [6] Joel Filipe Pereira. Estacionamento autónomo usando percepção 3d. Master's thesis, Universidade de Aveiro, 2012.
- [7] Ricardo Luís Fernandes Silva. Navegação local do ATLASCAR2 para condução autónoma e assistência ao condutor. Master's thesis, Universidade de Aveiro, 2018.
- [8] Gazebo website. <https://gazebo.org/>. Último acesso 21 Maio 2019.
- [9] Carla. <http://carla.org/>. Último acesso 21 Maio 2019.
- [10] Tutorial de CARLA. [https://carla.readthedocs.io/en/latest/getting\\_started/](https://carla.readthedocs.io/en/latest/getting_started/). Último acesso 20 Maio 2019.
- [11] Lgsvl simulator github. <https://github.com/lgsvl/simulator>. Último acesso 21 Maio 2019.
- [12] Francisca Rosique, Pedro Navarro Lorente, Carlos Fernandez, and Antonio Padilla. A systematic review of perception system and simulators for autonomous vehicles research. *Sensors*, 19, 02 2019.

- [13] Deepdrive website. <https://deepdrive.io/>. Último acesso 21 Maio 2019.
- [14] Udacity github. <https://github.com/udacity/self-driving-car-sim>. Último acesso 21 Maio 2019.
- [15] Load multiple TurtleBot3s tutorial. <http://emanual.robotis.com/docs/en/platform/turtlebot3/applications/#load-multiple-turtlebot3s>. Último acesso 21 Maio 2019.
- [16] Turtlebot3 e-manual. <http://emanual.robotis.com/docs/en/platform/turtlebot3/overview/#turtlebot3>. Último acesso 21 Maio 2019.
- [17] Chris Bogdon. Simulating multiple husky ugvs in gazebo. <https://www.clearpathrobotics.com/blog/2016/03/simulating-multiple-husky-ugvs-in-gazebo>. Último acesso 21 Maio 2019.
- [18] Brian Bingham. nre\_simmultihusky wiki. [https://github.com/bsb808/nre\\_simmultihusky/wiki](https://github.com/bsb808/nre_simmultihusky/wiki). Último acesso 21 Maio 2019.
- [19] Michele Palmia. Understanding launch and namespaces. [https://github.com/micpalmia/youbot\\_ros\\_tools/wiki/Understanding-launch-and-namespaces](https://github.com/micpalmia/youbot_ros_tools/wiki/Understanding-launch-and-namespaces). Último acesso 21 Maio 2019.
- [20] Prashant Ganesh. rbcarsim github. <https://github.com/prasgane/rbcarsim>. Último acesso 21 Maio 2019.
- [21] Pagina inicial do LAR. <http://lars.mec.ua.pt/index.html>. Último acesso 21 Maio 2019.
- [22] Sick lms151. <https://www.sick.com/ag/en/detection-and-ranging-solutions/2d-lidar-sensors/lms1xx/lms151-10100/p/p141840>. Último acesso 21 Maio 2019.
- [23] Sick ld-mrs400001. <https://www.sick.com/ag/en/detection-and-ranging-solutions/3d-lidar-sensors/ld-mrs/ld-mrs400001/p/p112355>. Último acesso 21 Maio 2019.
- [24] Ros wiki. <http://wiki.ros.org/>. Último acesso 20 Maio 2019.
- [25] Ros kinetic kame wiki. <http://wiki.ros.org/kinetic>. Último acesso 21 Maio 2019.
- [26] Ros melodic morenia wiki. <http://wiki.ros.org/melodic>. Último acesso 21 Maio 2019.
- [27] Understanding ros nodes. <http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes>. Último acesso 21 Maio 2019.
- [28] Messages ros. <http://wiki.ros.org/Message>. Último acesso 21 Maio 2019.
- [29] Understanding ros topics. <http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics>. Último acesso 21 Maio 2019.
- [30] Namespaces in c++. [https://www.tutorialspoint.com/cplusplus/cpp\\_namespaces.htm](https://www.tutorialspoint.com/cplusplus/cpp_namespaces.htm). Último acesso 21 Maio 2019.



- [31] Guillermo Castillo (Wei Zhang). Ece 5463 introduction to robotics spring 2018 - ros tutorial 2. 2018.
- [32] Solidworks wikipedia page. <https://pt.wikipedia.org/wiki/SolidWorks>. Último acesso 21 Maio 2019.
- [33] Blender wikipedia page. <https://pt.wikipedia.org/wiki/Blender>. Último acesso 21 Maio 2019.
- [34] Webpage do solidworks. <https://www.solidworks.com/>. Último acesso 21 Maio 2019.
- [35] Webpage do blender. <https://www.blender.org/>. Último acesso 21 Maio 2019.
- [36] Alvaro Villamil. Mapa utilizado na simulação. <https://github.com/RoboticsURJC-students/2016-tfg-Alvaro-Villamil/tree/master/Monaco>. Último acesso 21 Maio 2019.
- [37] Alvaro Villamil. Prácticas docentes en jderobot: Circuito realista f1 y brazo manipulador. <http://jderobot.org/Avillamil-tfg>. Último acesso 21 Maio 2019.
- [38] Jorge Manuel Soares de Almeida. Multi hypothesis tracking, what is it? <http://lars.mec.ua.pt/lartk4/mtt/html/Mht.html#mhtpage>. Último acesso 21 Maio 2019.
- [39] Miguel C. Figueiredo, Rosaldo Rossetti, Rodrigo A. M. Braga, and Luís Reis. An approach to simulate autonomous vehicles in urban traffic scenarios. 11 2009.
- [40] Rene Alexander Diaz Martinez. A library on the robot operating system (ros) for model predictive control implementation. Master's thesis, KTH, School of Industrial Engineering and Management (ITM), Machine Design (Dept.), 2014.
- [41] Jorge Manuel Soares de Almeida. Seguimento ativo de agentes dinâmicos multivariados usando informação vectorial. Master's thesis, Universidade de Aveiro, 2016.
- [42] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.
- [43] Michele Palmia. youbot\_ros\_tools github. [https://github.com/micpalmia/youbot\\_ros\\_tools](https://github.com/micpalmia/youbot_ros_tools). Último acesso 21 Maio 2019.
- [44] Jonathan Hui. Lane keeping in autonomous driving with model predictive control & pid. [https://medium.com/@jonathan\\_hui/lane-keeping-in-autonomous-driving-with-model-predictive-control-50f06e989bc9](https://medium.com/@jonathan_hui/lane-keeping-in-autonomous-driving-with-model-predictive-control-50f06e989bc9). Último acesso 21 Maio 2019.
- [45] Vítor Santos. Github do professor vítor santos. <https://github.com/vitoruapt/lartkv5>. Último acesso 21 Maio 2019.
- [46] Ricardo Santos. Github de ricardo santos. <https://github.com/ricardofsilva>. Último acesso 21 Maio 2019.