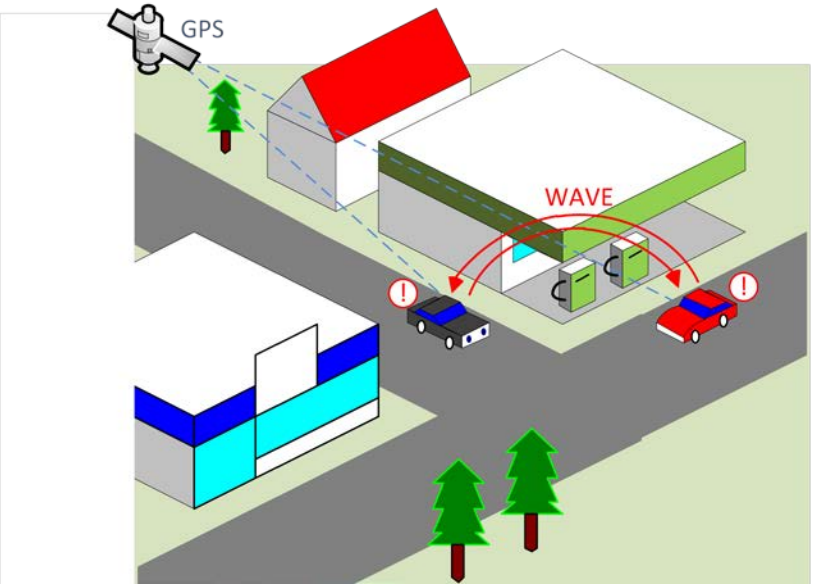
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA
SERVIÇO DE DOCUMENTAÇÃO E PUBLICAÇÕES

INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA
Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores



Aplicações e Algoritmos de Segurança Rodoviária com o uso de sistemas de GPS e de comunicações WAVE

Fábio Cardoso
Licenciado

Trabalho Final de Mestrado para Obtenção do Grau de Mestre em Engenharia de Electrónica e Telecomunicações

Orientador:
Professor António João Nunes Serrador

Júri:
Presidente: Professor João Miguel Duarte Ascenso
Vogais:
Professor Pedro Miguel Torres Mendes Jorge
Engenheiro Tomé Canas

Novembro 2013

Agradecimentos

A concretização desta tesa marca o final duma fase importante da minha vida que abre as portas para as próximas etapas. A dissertação que agora se apresenta resultou de um trajeto atribulado ao longo do qual fui recebendo o maior apoio e estímulo de muitos. Neste sentido, os méritos que ela possa ter, devem-se aos contributos das pessoas que durante, a sua elaboração, me proporcionaram testemunhos de vários géneros, expressando por isso a todos a minha mais profunda gratidão.

Destaco o papel desempenhado pelo orientador Engenheiro António Serrador, que manteve uma presença ativa ao longo da realização desta tese, mostrando a sua disponibilidade com a realização de reuniões semanais e ajudas por outros meios de comunicação, correio eletrónico e telemóvel. Foi uma das ajudas mais importantes, a sua reação tranquila perante os problemas fez com que eu progredisse-se de forma calma e saudável obtendo boas soluções. Também lhe estou grato pela proposta do tema desta dissertação que se mostrou bastante interessante, atual e útil para o futuro.

O meu agradecimento vai também para todo o grupo GIEST, com que trabalhei ao longo da realização desta dissertação e ao longo do cargo que tive como investigador bolseiro a desenvolver parte presente nesta tese. Os meus agradecimentos ao Ricardo Pereira, Filipe Palhinha, Duarte Carona, Ana Catarina e aos restantes investigadores presentes no Laboratório de Investigação e Desenvolvimento, que fizeram parte da minha companhia nos dias inteiros de trabalho, com quem passei bons momentos de humor, lazer e trabalho.

Exteriormente ao ISEL, destaco uma ajuda importante por parte da minha namorada Sónia Vaz, pois, além do seu grande apoio, disponibilizou-se para ajudar na realização dos testes práticos, que, sozinho, eram de difícil realização.

Agradeço aos meus pais, Clarinda Gomes e Alberto Cardoso, que sempre acreditaram nas minhas capacidades e mostraram o seu isentivo e a sua disponibilidade financeira para ser possível a conclusão desta etapa.

Por último, agradeço ao resto da minha grande família que mostrou o seu interesse e apoio nesta dissertação e no meu futuro.

Resumo

Os acidentes rodoviários são um problema de grande importância para toda a humanidade. O objetivo desta tese é desenvolver um sistema, através de comunicação WAVE, *vehicle-to-vehicle*, capaz de alertar os automobilistas de risco de colisões.

O sistema desenvolvido utiliza os dados recebidos por um sistema de GPS para o processamento dos algoritmos de segurança, estudados no estado da arte e desenvolvidos nesta tese. Para um cenário de aproximação súbita foram estudados os algoritmos da Mazda, Honda, PATH, NHTSA e IEEE, para um cenário de cruzamento/entroncamento os algoritmos da IET, Ford, Safety zones e Taiwan. Para o cenário de curva foi desenvolvido o algoritmo de Cobra.

Percebe-se a possibilidade de evitar um acidente com a obtenção de aviso dum cenário de risco de colisão, no interior duma localidade, cerca de 15 m antes do cruzamento, distância suficiente para as devidas precauções, em auto-estrada, circulando a 120 km/h, 60 m antes do veículo que circula a 40 km/h à frente, numa curva com 33 m de raio, recebe-se um aviso de excesso de velocidade aproximando-se excedendo os 31 km/h. Resultados plausíveis através da prática no terreno.

Palavras-chave

Algoritmos de segurança rodoviária, Aviso de colisão, Vehicle-to-vehicle, comunicação WAVE.

Abstract

Road accidents are problems of great importance to all humanity. The objective of this thesis is to develop a system, through communication WAVE, vehicle-to-vehicle, which act so as to alert the motorist of collisions risk.

The developed system uses the data received by a GPS system for the processing of security algorithms, studied in the state of art and developed in this thesis. For sudden approach scenario was studied the algorithms of Mazda, Honda, PATH, NHTSA and IEEE, for crossing/junction scenario the algorithm of IET, Ford, Safety Zones and Taiwan. For curve scenario was developed the Snake algorithm.

Perceives the possibility of avoiding an accident getting a warning, of a collision risk scenario at an intersection within a locality, with about 15 meters before crossing, enough time for precautions, in highway, at 120 km/h, getting a warning 60 m before the vehicle traveling at 40 km/h ahead, in a curve with radius 33 m, a exceeded speed warning was triggered at approaching this at more than 30 km/h. Plausible results through field practice.

Keywords

Road Safety Algorithms, Collision Warning, Vehicle-to-vehicle, WAVE communication.

Índice

Lista de Figuras

Lista de Tabelas

Lista de Siglas

BMP	Bitmap
CAN	Controller Area Network
CoG	Center of Gravity
CTRS	Sistema de Referência Terrestre Convencional
DOF	Degrees Of Freedom
DSRC	Dedicated Short Range Communications
ECEF	Earth-Centered Earth-Fixed
FPGA	Field-Programmable Gate Array
GIEST	Grupo de Investigação em Eletrónica e Sistemas de Telecomunicações
GPGGA	Global Positioning System Fix Data
GPS	Global Positioning System
IEEE	Institute of Electrical and Electronic Engineers
IET	Institution of Engineering and Technology
IMU	Inertial Measurement Unit
ISEL	Instituto Superior de Engenharia de Lisboa
IT	Instituto de Telecomunicações
ITS	Intelligent Transportation Systems
JPEG	Joint Photographic Experts Group
LLC	Logical Link Control
MAC	Media Access Control
NED	North-East-Down

NHTSA	National Highway Traffic Safety Administration
NMEA	National Marine Electronics Association
OBU	On Board Unit
PATH	Partners for Advanced Transit and Highways
PCB	Printed Circuit Board
PNG	Portable Network Graphics
RSU	Road Side Unit
UTM	Universal Transverse Mercator
WABCO	Westinghouse Air Brake Company
WAVE	Wireless Access in Vehicular Environments
WGS	World Geodetic System
WSMP	Wave Short Message Protocole

Lista de Símbolos

α'_x	Desaceleração máxima do veículo x
α'	Desaceleração máxima
α_x	Aceleração corrente do veículo *x*
α	Aceleração corrente
α_{rel}	Diferença de aceleração dos veículos, antecedente e procedente
τ_1	Tempo de atraso do sistema
τ_2	Tempo de atraso do condutor
τ	Tempo de atraso total
φ	Azimute
β	Ângulo da curva
ϕ_E, θ_E e ψ_E	Ângulos de Euler
δ	Ângulo de direção da roda dianteira
ψ	Variação da direção do veículo
Λ_x	Latitude do veículo *x*
λ_x	Longitude do veículo *x*
ϕ	Latitude
λ	Longitude
(a_c, b_c)	Coordenada cartesiana do centro da circunferência virtual
a	Raio Equatorial

b	Raio Polar
C_f	Rigidez em curva do pneu dianteiro
C_r	Rigidez em curva do pneu traseiro
d	Distância mais curta entre veículos
d_0	Distância de compensação
d_{br}	Distância crítica de travagem
d_p	Distância entre dois pontos cartesianos
d_w	Distância de aviso
e	Excentricidade
e_{F_y}	Perturbação lateral equivalente
e_{M_z}	Aceleração da guinada
F	Força centrífuga
f	Achatamento da Terra (*flattening*)
f^n	Aceleração na direção *n*
f_t	Coeficiente de atrito transversal
g^n	Força da gravidade na direção *n*
h	Altitude no formato GPGGA
I_Z	Momento de inércia da guinada
k	Coeficiente de atrito
m	Massa do veículo
R	Raio da curva

r	Taxa de guinada
R_α	Raio de curvatura para uma determinada azimute α
RA	Raio mínimo absoluto
$\overrightarrow{r_c} = (x_c, y_c)$	Coordenadas cartesianas do ponto de colisão
R_m	Raio de curvatura do Meridiano
R_{min}	Raio da circunferência de teste do veículo estacionado
R_N	Raio de curvatura do primeiro vertical
RN	Raio mínimo normal
$\overrightarrow{r_n} = (x, y)$	Coordenadas cartesianas da posição do veículo n
R_w	Raio da circunferência relativa à área de aviso do algoritmo Curva
Se	Sobrelevação do veículo, em percentagem
t	Variável temporal
T_{FS}	Tempo de paragem do veículo procedente
T_{LS}	Tempo de paragem do veículo antecedente
TR T_R	Tempo de reacção
TTA	Tempo para o aviso
TTC	Tempo até à colisão
TTX_n	Tempo até a interceção do veículo n
u	Velocidade do veículo
v	Velocidade
V_{max}	Velocidade máxima do veículo para uma determinada curva

v^n	Velocidade na direção *n*
v_{rel}	Diferença de velocidades dos veículos, antecedente e procedente
v_x	Velocidade do veículo x
W	Largura do veículo
w^n	Velocidade angular na direção *n*
L	Comprimento do veículo

1 Introdução

1.1 Enquadramento

No futuro as comunicações entre veículos e entre estes e as infraestruturas rodoviárias vão ser uma realidade. Para tal definem-se sistemas de comunicação como o IEEE 802.11p [1] que, juntamente com sensores ou outros sistemas como GPS e o CAN *bus* dos veículos, podem servir aplicações de grande interesse para a segurança rodoviária de forma preventiva.

Associado à evolução das telecomunicações, encontra-se o desenvolvimento de sistemas assentes na norma 802.11p, são os casos dos diversos projetos em desenvolvimento a nível global, havendo equipas de investigação e desenvolvimento das várias marcas de automóveis e em empresas com relevância em infraestruturas rodoviárias, como é o caso da Brisa em Portugal. Estando esta tese enquadrada na parceria que o ISEL tem com a Brisa Inovação e Tecnologia.

Os sistemas WAVE, *Wireless Access in Vehicular Environment*, também conhecidos como sistemas *vehicle-to-vehicle*, não assentam apenas sobre a norma do sistema de comunicação, 802.11p, mas também estão definidos nas camadas superiores. As normas 1609.3 [2] e 1609.4 [3] definem a camada de rede, têm vindo a sofrer alterações ao longo do tempo. A norma 1609.1 define a gestão de aplicações, que segundo a informação recebida dum dos criadores das normas, John Morin, irá ser substituída mais tarde pela norma 1609.6, daí a razão pela qual o cumprimento desta norma não é rígido ao longo desta tese.

1.2 Objetivo e Motivação

O objetivo desta dissertação é o desenvolvimento de algoritmos e aplicações de segurança rodoviária preventiva e a sua implementação num sistema WAVE, que tirem partido de um terminal GPS instalado a bordo dos veículos. Estas aplicações têm como finalidade avisar o condutor perante riscos de colisão, utilizando como referência os dados recebidos do sistema GPS, a localização, orientação e velocidade do veículo, que por sua vez são dados a conhecer aos restantes automobilistas enviando mensagens WAVE em *broadcast*.

O objetivo é garantir um sistema fiável, minimizando situações de falsos avisos. Contando com a análise e comparação entre os algoritmos, estudados e desenvolvidos, tanto a nível de simulação como no terreno.

São estudados três cenários principais: cruzamento ou entroncamento; aproximação súbita do veículo procedente e velocidade excessiva para uma determinada curva.

Perante um cenário de cruzamento é onde acontecem a maioria dos acidentes no interior das localidades, pretende-se, quando dois veículos se aproximam dum cruzamento e ambos não declaram uma desaceleração, que o sistema indique aos condutores do risco de colisão para que estes tomem as devidas precauções.

A aproximação súbita ao veículo que nos procede nem sempre é percetível, principalmente a velocidades elevadas, onde é necessária uma grande distância de travagem para parar com precaução e quando se apercebe que o veículo procedente está parado ou com uma velocidade reduzida, pode ser tarde demais. Sendo esta a situação mais frequente nos acidentes em auto-estradas e vias rápidas, sendo fundamental alertar os condutores dos perigos iminentes.

A maioria dos acidentes individuais acontecem em curvas, devido ao excesso de velocidade para um determinado raio de curvatura, por ação da elevada força centrífuga a que o veículo está sujeito leva ao seu despiste [4], solução para esta situação também desenvolvida nesta tese.

Os dois primeiros cenários, cruzamento ou entroncamento e aproximação súbita ao veículo procedente, foram desenvolvidos e implementados tendo como base algoritmos já presentes na literatura. Para o terceiro cenário, de acordo com os documentos estudados, ainda não existem algoritmos preventivos desenvolvidos para evitar o despiste do veículo em curvas, deste modo, é desenvolvido e proposto nesta tese um novo algoritmo para esse fim, sendo por isso uma contribuição inovadora.

Este é um fruto de maior importância para a humanidade, pois, os acidentes rodoviários foram causadores de 32 mil mortes nos Estados Unidos em 2010, segundo The New York Times [5], mais de 30 mil mortes por ano nas estradas Europeias [6] e, de acordo com o Diário de Noticias, morrem cerca de 1000 pessoas por ano em Portugal [7], espera-se reduzir esse valor com a introdução de sistemas como o desenvolvido nesta tese.

Os resultados desta tese são publicados na conferência CETC 2013, *Conference on Electronics, Telecommunications and Computers*.

1.3 Estrutura da dissertação

Esta dissertação encontra-se dividida por cinco capítulos principais. Cada capítulo descreve uma etapa diferente da realização do trabalho. Desta forma, a dissertação encontra-se organizada da seguinte forma:

- No capítulo introdutório é feita uma abordagem à importância do desenvolvimento de sistemas deste tipo, sendo destacados os principais objetivos da tese.
- O segundo capítulo apresenta algoritmos e testes a sistemas já existentes e publicados na literatura sendo apresentado o estado da arte relativo ao tema da dissertação.
- No terceiro capítulo é onde está presente o desenvolvimento e os passos de implementação, *software* e *hardware*, do sistema desenvolvido. Possuí uma primeira fase onde apresenta complementos aos algoritmos estudados no capítulo anterior e propõe novos conceitos e algoritmos.
- Os resultados constam no quarto capítulo, onde são apresentados e analisados, conta com a comparação entre os diversos algoritmos estudados no estado da arte e os resultados dos testes simulados e práticos, sendo apresentados também resultados do sistema implementado no terreno.
- Por último, o quinto capítulo, destaca as conclusões obtidas no desenvolvimento desta dissertação, incluindo falhas do sistema e possíveis melhoramentos em trabalho futuro.

2 Estado da arte

Os sistemas *vehicle-to-vehicle* são vistos como componentes essenciais no futuro da segurança rodoviária, tendo vindo a ser estudados e desenvolvidos por diversas empresas do ramo.

Estão disponíveis informações acerca dos algoritmos em estudo, parte delas estão destacadas ao longo deste capítulo, extraídas das várias referências mencionadas. Também existem empresas que já procederam à comercialização de *hardware* sobre este modelo de comunicação, e disponibilizam testes práticos que permite obter uma melhor perceção da mobilidade da tecnologia.

Na situação mais comum, os cálculos destes algoritmos utilizam dados em coordenadas cartesianas, que serão o resultado da conversão dos dados recebidos via GPS ou CAN *bus*, (duas fontes de dados utilizadas na segurança rodoviária [8] [9]), para uma maior precisão existem algoritmos desenvolvidos utilizando giroscópios e/ou acelerómetros [10].

2.1 Mobilidade

O sistema de comunicação é baseado na norma IEEE 802.11p, DSRC/WAVE [1], também conhecido pela sua frequência de trabalho: 5.9 GHz. O sistema WAVE é uma tecnologia rádio destinada a transmitir serviços de comunicações veiculares interoperáveis. O WAVE incluí serviços reconhecidos pela arquitetura *U.S. National Intelligent Transportation Systems* (ITS) e muitos outros contemplados pelas indústrias automotivas e de infraestruturas de transporte. Estes serviços incluem comunicações entre veículos e entre estes e as infraestruturas [2].

Os algoritmos de segurança rodoviária enquadram-se em diversos cenários, contidos em ambientes rurais ou urbanos. O ponto fundamental na interligação do sistema de comunicação e os algoritmos, é a distância a que a comunicação entre dois veículos se torna possível, de forma a disponibilizar tempo suficiente para o processamento do algoritmo, o envio e receção

de mensagens, a sua transmissão, a leitura do utilizador e o seu tempo de reação e de ação, caso se aplique.

A situação mais critica na segurança rodoviária consiste numa situação de cruzamento ou entroncamento em ambiente urbano, em que a visibilidade entre os dois veículos se encontra obstruída por edifícios.

Em função de testes práticos disponíveis, pode-se retirar exemplos de valores da receção do sinal, Figura 2.1, sendo neste caso em particular uma potência de -91 dBm, a uma distância de 116 m, em linha reta entre os veículos, resultados extraídos de [11] (outra fonte de resultados idênticos em [12]) o que indica a receção do sinal aproximadamente a 82 m antes da chegada do veículo ao cruzamento. Dada a velocidade máxima em localidades ser de 50 km/h, obtém-se aproximadamente 5,75 s até o cruzamento que, à partida, é tempo suficiente para todo o processo e a devida precaução. Noutra fonte, [10], também se concluiu que, para uma distância de 200 m em ambiente urbano, obtém-se uma latência de 654 ms e uma taxa de pacotes perdidos de 76 %. Estes valores dependeram do *hardware*, mas note-se que com estas referências é percetível a possibilidade de implementação, bem-sucedida, de algoritmos de segurança preventiva.

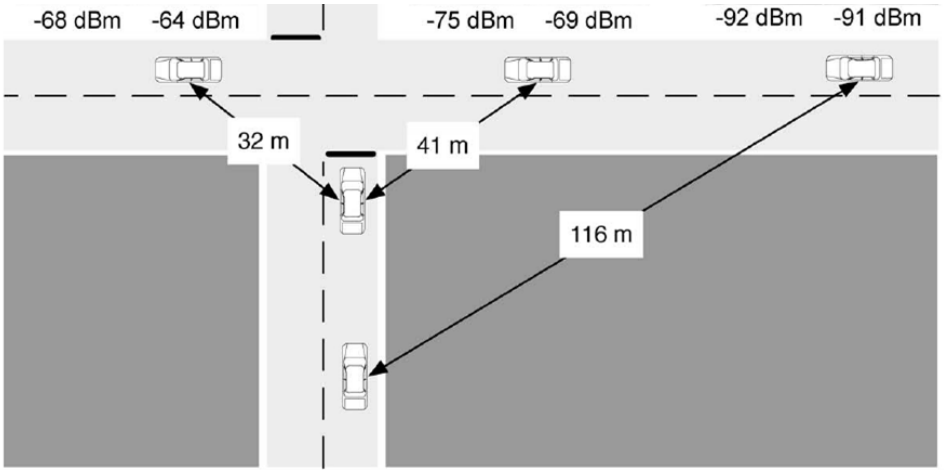


Figura 2.1 - Variação da potência do sinal recebido num sistema WAVE (extraído de [11]).

2.2 Aproximação súbita do veículo procedente

Alguns cenários a avaliar são: travagem repentina, aproximação súbita do veículo procedente. Sendo estes cenários estudados por vários autores em várias empresas. Alguns algoritmos contidos na bibliografia são mencionados adiante.

Neste cenário, os algoritmos são interpretados por uma só dimensão, tendo em conta ambas as velocidades dos veículos, v_1 e v_2, a distância d entre eles, e os restantes fatores que os rodeiam, o cálculo fundamental é saber a distância crítica entre ambos, d_W, que corresponde à distância em que os veículos se encontram em risco e a que deve ser acionado o aviso ao condutor, Figura 2.2.

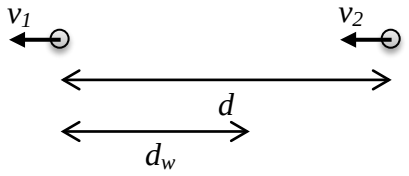


Figura 2.2 - Ilustração dos algoritmos de aproximação súbita ao veículo procedente.

2.2.1 Mazda

Baseado numa análise cinemática a Mazda [13], definiu a distância crítica como sendo dada por:

$$d_{w[\mathrm{m}]} = \frac{1}{2}\left(\frac{v^2}{\alpha'_1} - \frac{(v - v_{rel})^2}{\alpha'_2}\right) + v\tau_1 + v_{rel}\tau_2 + d_0\,, \qquad \textbf{(2.1)}$$

onde v é a velocidade do veículo antecedente, v_{rel} é a diferença de velocidades dos dois veículos, α'_1 e α'_2 são as desacelerações máximas dos veículos, τ_1 e τ_2 são os tempos de atraso do sistema e do condutor respetivamente e d_0 a distância de compensação.

Este algoritmo foi desenvolvido assumindo que ambos os veículos iniciam a travagem com uma desaceleração de α'_1 e α'_2 respetivamente , após os tempos de resposta do sistema e do condutor, τ_1 e τ_2.

2.2.2 Honda

A Honda desenvolveu o seu próprio algoritmo [13] baseado em resultados experimentais, onde a distância crítica é dada por:

$$d_{w[\mathrm{m}]} = 2{,}2v_{rel} + 6{,}2\ , \tag{2.2}$$

em que v_{rel} é a diferença de velocidades entre os dois veículos.

2.2.3 PATH

A parceria PATH, *Partners for Advanced Transit and Highways*, desenvolveu uma versão modificada do algoritmo da Mazda, de onde resulta

$$d_{w[\mathrm{m}]} = \frac{1}{2}\left(\frac{v^2}{\alpha'} - \frac{(v - v_{rel})^2}{\alpha'}\right) + v\tau + d_0\ , \tag{2.3}$$

onde v é a velocidade do veículo antecedente, v_{rel} é a diferença de velocidades dos dois veículos, α' é a desaceleração máxima dos veículos, τ é o tempo total de atraso do sistema e do condutor e d_0 a distância de compensação.

O algoritmo PATH, para além de conjugar os valores de desaceleração dos veículos e tempos de atraso, focou-se em complementar a distância crítica desenvolvida pela Mazda, adicionando um parâmetro de escala de aviso, dado por:

$$w = \frac{d - d_{br}}{d_w - d_{br}}, \tag{2.4}$$

onde *d* é a distância atual entre os dois veículos e d_{br} a distância crítica de travagem, dada por

$$d_{br[\mathrm{m}]} = v_{rel}\tau + \frac{1}{2}\alpha'\tau^2. \tag{2.5}$$

A distância crítica de travagem representa a distância onde já não é possível, seguindo a máxima desaceleração α e os tempos de atraso τ, evitar a colisão. Pode-se observar a representação das várias distâncias na Figura 2.3.

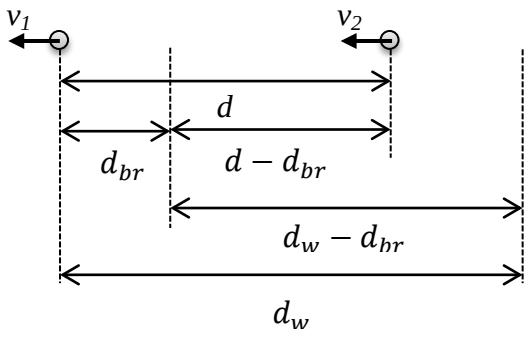


Figura 2.3 – Ilustração dos campos do algoritmo PATH.

Através do parâmetro *w* é possível apresentar ao condutor um aviso gradual, quando *w>1* indica fora de perigo, a partir que *w* atinge o valor 1 até ao valor 0 é apresentado ao condutor um aviso gradual, exemplificado na Figura 2.4, onde *a* corresponde a uma margem de segurança [13].

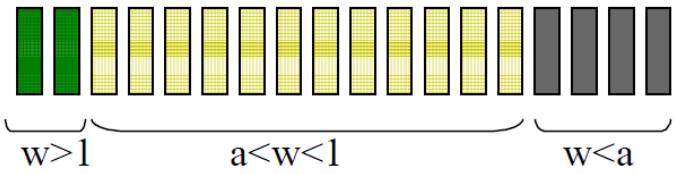


Figura 2.4 – Aviso luminoso gradual.

2.2.4 NHTSA

O algoritmo de NHTSA, *National Highway Traffic Safety Administration*, é também baseado numa análise cinemática, incluindo a informação da aceleração corrente dos veículos, onde a distância de aviso é dada por uma das equações:

$$d_{w1[\mathrm{m}]} = 0{,}1v + d_0 - \frac{1}{2}(a-\alpha')(T_R)^2 + \frac{1}{2}(a-\alpha_{rel})(T_{LS})^2 + (a-\alpha')T_R T_{FS} - v_{rel}T_{FS} - (a-\alpha_{rel})T_{FS}T_{LS} + \frac{1}{2}\alpha'(T_{FS})^2 \; ; \tag{2.6}$$

$$d_{w2[\mathrm{m}]} = 0{,}1v + d_0 - v_{rel}T_M - \frac{1}{2}(\alpha-\alpha_{rel}-\alpha')(T_M)^2 + (\alpha-\alpha')T_M T_R - \frac{1}{2}(\alpha-\alpha')(T_R)^2 \; , \tag{2.7}$$

onde d_0 é a distância de compensação, α é a aceleração actual do veículo antecedente, α' é a máxima desaceleração do veículo antecedente, α_{rel} é a diferença de aceleração entre os dois veículos, T_R é o tempo de reacção, T_{LS} e T_{FS} são os tempos de paragem do veículo procedente e antecedente respetivamente e T_M é o tempo para a diferença entre as velocidades dos dois veículos atingir o valor zero. T_{LS}, T_{FS} e T_M são cálculados pelas seguintes fórmulas:

$$T_{M[\mathrm{s}]} = \frac{(v_{rel} - \alpha_{rel}T_R)}{(\alpha' - \alpha + \alpha_{rel})} + T_R \; ; \tag{2.8}$$

$$T_{LS[\mathrm{s}]} = -\frac{(v - v_{rel})}{(\alpha - \alpha_{rel})} \; ; \tag{2.9}$$

$$T_{FS[\mathrm{s}]} = T_R - \frac{(v + \alpha T_R)}{\alpha'} \, . \tag{2.10}$$

O valor da distância de aviso é selecionado em função dos valores de T_{LS} e T_{FS}, caso T_{FS} seja superior a T_{LS} o valor válido é d_{w1}, caso contrário é utilizado d_{w2} [14].

2.2.5 IEEE

O algoritmo presente na literatura do IEEE, *Institute of Electrical and Electronic Engineers*, obtém a distância de aviso convertendo o tempo de reação do condutor e o tempo de travagem do veículo para distâncias em função da velocidade, de acordo com:

$$d_{w[\mathrm{m}]} = T_R v + 0{,}1v + 0{,}006v^2 \,, \tag{2.11}$$

onde v é a velocidade absoluta do veículo e T_R o tempo de reação. Não depende do veículo procedente, assume que para uma determinada velocidade deverá existir uma distância de segurança d_w. Onde a primeira parcela da equação é o resultado da distância de reação, e as restantes da distância de travagem [11].

2.3 Cruzamentos e entroncamentos

A análise de colisões unidimensional não é suficiente para evitar a maior parte dos acidentes rodoviários, porém foram desenvolvidos algoritmos, através de trigonometria e outros métodos, para situações de colisões relativas a cruzamentos e entroncamentos, na maioria dos casos, assumindo que os veículos seguem uma trajetória retilínea.

2.3.1 IET

A IET, *Institution of Engineering and Technology*, desenvolveu algoritmos, não só para a situação de aproximação súbita do veículo procedente, como também para colisões em cruzamentos ou entroncamento [15].

Analisando a situação num plano em duas dimensões. Cada veículo possui quatro componentes, posição *x* e *y*, velocidade v e azimute φ. As equações dinâmicas do veículo são dadas por:

$$\begin{cases} x(t_{i+1}) = x(t_i) + v \times (t_{i+1} - t_i) \cos\varphi \\ y(t_{i+1}) = y(t_i) + v \times (t_{i+1} - t_i) \sin\varphi \end{cases}. \tag{2.12}$$

Existem vários modelos possíveis para analisar a colisão entre dois veículos, considerando o veículo um ponto no espaço, uma circunferência ou um retângulo, como se encontra presente na Figura 2.5.

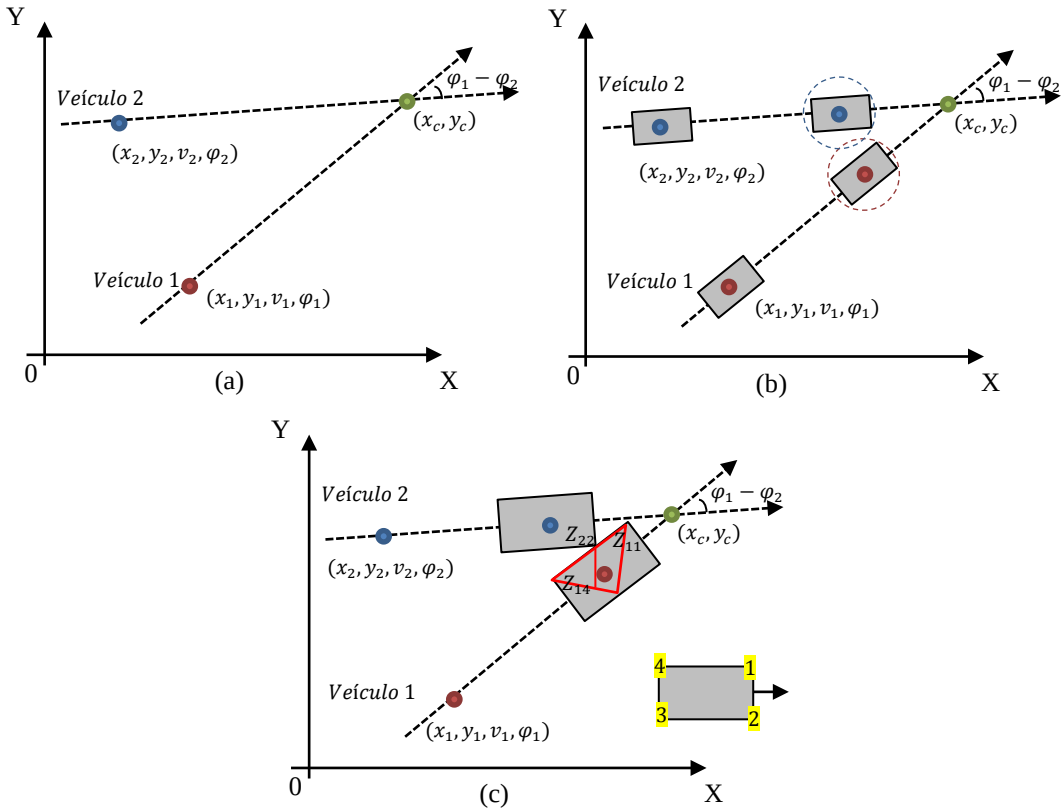


Figura 2.5 - (a) modelo partícula; (b) modelo círculo; (c) modelo retângulo.

Sendo a posição (x_n, y_n) da trajetória do veículo *n* com a direcção φ dada por:

$$x_n(t) = x_n(0) + \big(y_n(t) - y_n(0)\big)\cot\varphi \qquad \textbf{(2.13)}$$

ou

$$y_n(t) = y_n(0) + \big(x_n(t) - x_n(0)\big)\tan\varphi, \qquad \textbf{(2.14)}$$

a condição de colisão entre dois veículos é dada por

$$x_1(t) = x_2(t) = x_c\,, y_1(t) = y_2(t) = y_c\,, \qquad \textbf{(2.15)}$$

onde (x_c, y_c) representa o ponto de colisão, e é determinado pelas seguintes equações:

$$x_c = \begin{cases} C_1 \quad , \varphi_1 \vee \varphi_2 = 90° \vee 270° \\ \dfrac{y_2(0) - y_1(0) - (x_2(t_2)\tan\varphi_2 - x_1(t_1)\tan\varphi_1)}{\tan\varphi_1 - \tan\varphi_2} \end{cases} ; \qquad \textbf{(2.16)}$$

$$y_c = \begin{cases} C_2 \quad , \varphi_1 \vee \varphi_2 = 90° \vee 270° \\ \dfrac{x_2(0) - x_1(0) - (y_2(t_2)\cot\varphi_2 - y_1(t_1)\cot\varphi_1)}{\cot\varphi_1 - \cot\varphi_2} \end{cases} . \qquad \textbf{(2.17)}$$

Para existir colisão é necessário ambos os veículos chegarem ao mesmo tempo ao ponto de colisão, portanto, o tempo para a colisão *TTC* (*Time To Collision*) designa-se por

$$TTC = t_1 = t_2. \qquad \textbf{(2.18)}$$

A partir das equações **(2.17)** e **(2.18)** pode-se obter infinitas soluções, logo, para uma solução direta opta-se pela seguinte equação:

$$\sqrt{\left(x_1(t_1) - x_2(t_2)\right)^2 + \left(y_1(t_1) - y_2(t_2)\right)^2} = 0 . \qquad \textbf{(2.19)}$$

Assim, assumindo os veículos como sendo pontos no espaço, consegue-se obter o valor de TTC a partir das equações **(2.13)**, **(2.14)**, **(2.18)** e **(2.19)**. Caso as equações não apresentem soluções, significa que não existe risco colisão entre os veículos.

Para o modelo círculo, caso (b) da Figura 2.5, o carro é figurado por um círculo centrado em $(x(t), y(t))$ com raio *R*, assim, assume-se colisão quando se verifica a seguinte condição:

$$\sqrt{\left(x_1(t_1) - x_2(t_2)\right)^2 + \left(y_1(t_1) - y_2(t_2)\right)^2} = 2R , \qquad R = \frac{\sqrt{L^2 + W^2}}{2} , \qquad \textbf{(2.20)}$$

sendo *W* e *L* a largura e o comprimento do veículo, respectivamente.

Assumindo os veículos círculos de raio *R*, consegue-se obter o valor de *TTC* a partir das equações **(2.13)**, **(2.14)**, **(2.18)** e **(2.20)**.

Pode-se obter o ponto de colisão através de:

$$\begin{cases} x_c = \dfrac{x_1(TTC) + x_2(TTC)}{2} \\ y_c = \dfrac{y_1(TTC) + y_2(TTC)}{2} \end{cases} . \qquad \textbf{(2.21)}$$

Para a situação (c) da Figura 2.5, veículo figurado por um retângulo, a quantidade de cálculos é superior às outras abordagens, pois terá de ser feita a deteção de colisão de cada reta, relativa as arestas do retângulo, podendo ser preferível a utilização do modelo de circunferência poupando assim tempo de processamento.

Sendo Z_{nm} o vértice m do veículo n, como representado na Figura 2.5(c), obtém-se a seguinte equação para o determinar:

$$Z_{nm} = A_{nm}\frac{1}{2}\,K_m + (x_{n0}, y_{n0})\,, \tag{2.22}$$

onde,

$$A_n = \begin{bmatrix}\cos\varphi_n & -\sin\varphi_n \\ \sin\varphi_n & \cos\varphi_n\end{bmatrix},\ K_1 = [L - W]^T,$$

$$K_2 = [L\ W]^T\ ,\ K_3 = [-L\ W]^T,\ K_4 = [-L - W]^T,$$

sendo W e L a largura e o comprimento do veículo e (x_{n0}, y_{n0}) o seu centro.

Havendo colisão quando,

$$\frac{x_{2m} - x_{1\bar{m}}}{x_{1h} - x_{1\bar{m}}} = \frac{y_{2m} - y_{1\bar{m}}}{y_{1h} - y_{1\bar{m}}}, \tag{2.23}$$

que dá origem à condição,

$$\min(x_{1\bar{m}}, x_{1h}) \le x_{2m} \le \max(x_{1\bar{m}}, x_{1h})\,, \tag{2.24}$$

onde caso $\bar{m} = 1$ ou $\bar{m} = 3$, então $h = 2$ ou $h = 4$, e vice-versa.

Para o exemplo da Figura 2.5 a condição que deteta a colisão será: $\min(x_{11}, x_{14}) \le x_{22} \le \max(x_{11}, x_{14})$.

2.3.2 Ford

Já em 2002, nos laboratórios de investigação da Ford em parceria com a Universidade de Washington, os resultados eram idênticos aos da IET modelo partícula, da subsecção 2.3.1,

acrescentando um termo essencial para a implementação prática do algoritmo, o tempo para o aviso, TTA (*Time-To-Avoidance*) [16].

Este termo é essencial, pois o algoritmo necessita de saber quando deverá ser emitido um aviso ao condutor, e não apenas se existe colisão dadas as velocidades e posições instantâneas dos veículos, existem pelo menos dois motivos, mencionados no artigo da Ford em 2002, que levam a omitir o aviso mesmo existindo colisão. Um dos motivos é devido ao condutor já ter tomado as devidas precauções, como por exemplo, travar ou desacelerar, outro motivo é o excesso de tempo até à colisão, gerando avisos precoces, podendo ser falsos avisos, o que não é desejável.

As fórmulas relativas ao ponto de colisão são exatamente iguais às obtidas na subsecção 2.3.1, equações **(2.16)** e **(2.17)**. Complementando com o cálculo do tempo até a interceção de cada veículo, dado por:

$$TTX_n = \frac{|\overrightarrow{r_c} - \overrightarrow{r_n}|}{|\overrightarrow{v_n}|} \text{sign}\big((\overrightarrow{r_c} - \overrightarrow{r_n}) \times \overrightarrow{v_n}\big), \quad \textbf{(2.25)}$$

onde *n* é a identificação do veículo, $\overrightarrow{v_n}$ e $\overrightarrow{r_n}$ é a velocidade e posição (x, y) do veículo *n* e $\overrightarrow{r_c}$ é o ponto de colisão (x_c, y_c). Nota: a função *sign* determina o sinal de um número.

Neste caso o tempo até a colisão é dado por

$$TTC = TTX_1 = TTX_2. \quad \textbf{(2.26)}$$

Considerando a dimensão dos veículos, entre outros fatores, a expressão mais adequada é

$$|TTX_1 - TTX_2| < \gamma, \quad \textbf{(2.27)}$$

onde γ é o parâmetro de contenção, dado em função da dimensão dos veículos, velocidades e incertezas do sistema.

O tempo para o aviso é dado por

$$TTA = t_r + \frac{\beta v}{\mu g}, \quad \textbf{(2.28)}$$

sendo μ o coeficiente de fricção do pneu com a estrada, g a aceleração da gravidade, v a velocidade corrente e β o coeficiente de redução de velocidade. O coeficiente β varia no intervalo [0,1], sendo o valor 1 a necessidade de uma paragem por completo do veículo. Este

coeficiente é utilizado pois, na maioria dos casos, a colisão é evitada apenas com uma redução de velocidade, não sendo necessário parar o veículo. Note que, o coeficiente μ irá variar consoante as condições do piso, por exemplo, terra ou alcatrão, alcatrão molhado ou seco, melhorando a eficiência do algoritmo.

Tendo todos estes dados, o aviso é accionado assim que,

$$\mathrm{TTC} - \mathrm{TTA} < \tau, \tag{2.29}$$

onde τ é um parâmetro de ajuste do tempo de aviso eficaz.

Todos estes parâmetros, τ, μ e β, deverão ser ajustados tendo em conta o estudo dos fatores humanos.

2.3.3 Safety zones

O algoritmo *Safety Zones* consiste em criar uma zona retangular virtual de segurança em redor do veículo e detetar a sobreposição entre zonas de vários veículos, que indicará perigo de colisão [17].

O método de obtenção dos vértices do retângulo virtual e a deteção de perigo de colisão é idêntico ao algoritmo em 2.3.1, utilizando as equações **(2.22)** e **(2.24)**. A diferença está no comprimento do retângulo, alterando em **(2.22)** as ocorrências de L por $2F$, onde F, designado de distância de segurança idêntico a d_w em 2.2, tem em conta os vários parâmetros de segurança, velocidade do veículo, aceleração e tempos de reação, e obtém-se através das seguintes equações:

$$T_{stop} = \frac{v + \alpha T_R}{B_r}, \tag{2.30}$$

$$D_1 = vT_R + \frac{1}{2}\alpha T_R^2, \tag{2.31}$$

$$D_2 = V_1 T_{stop} + \frac{1}{2} B_r T_{stop}^2 , \qquad \mathrm{V}_1 = v + \alpha T_R, \tag{2.32}$$

$$F = D_1 + D_2 = vT_R + \frac{1}{2}\alpha T_R^2 + \frac{3}{2}\frac{(v + \alpha T_R)^2}{B_r}, \quad \textbf{(2.33)}$$

onde D_1 e D_2 são as distâncias representadas na Figura 2.6, T_{stop} é o tempo que o veículo leva a parar, B_r é o tempo de atraso dos travões, T_R é o tempo de reação do condutor, v e α é a velocidade e a aceleração correntes do veículo.

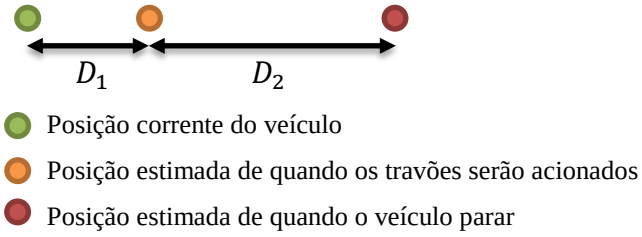


Figura 2.6 – Caraterização de D_1 e D_2.

O alerta é acionado quando um dos vértices do retângulo virtual se encontra no interior da zona virtual de outro veículo, exemplo presente na Figura 2.7 numa situação de três veículos, A, B e C. Utiliza-se um algoritmo padrão, e de conhecimento geral, para determinar se um sistema de coordenadas se encontra no interior de um polígono. Um ponto P (x, y), encontra-se no interior do polígono convexo se este se encontrar à esquerda de cada extremidade, percorrendo o polígono no sentido contrário aos ponteiros do relógio. Se A (x_1, y_1) e B (x_2, y_2) são os pontos da extremidade dum segmento de reta, o ponto P (x, y) encontra-se à sua esquerda se a seguinte expressão for verdadeira:

$$(x_2 - x_1)(\mathrm{y} - \mathrm{y}_1) - (y_2 - y_1)(x - x_1) > 0 \quad \textbf{(2.34)}$$

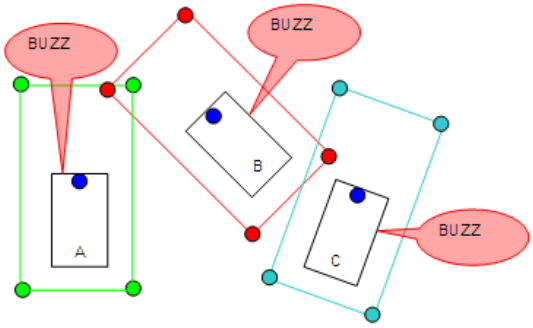


Figura 2.7 – Ilustração de exemplo do algoritmo *Safety Zones*.

2.3.4 Taiwan

Na ilha Formosa, na república da China, foi desenvolvido um algoritmo tendo em conta a terceira dimensão, altitude. A arquitetura está dividida em quatro componentes principais: sensores de IMU (*Inertial Measurement Unit*) com acelerómetros e giroscópios; recetor GPS e informação do veículo via CAN *bus* [10].

Na Figura 2.8 está presente o diagrama de blocos do método de obtenção das coordenadas cartesianas da posição do veículo, azimute e velocidade, ambas as componentes com três dimensões, através da transformação dos dados medidos. Note-se, os dados medidos através dos sensores de IMU tomam uma importância superior aos dados do GPS, pois garantem uma taxa de amostragem e um nível de precisão ainda não alcançável pelos sistemas de GPS comuns, utilizando este apenas para fornecer dados iniciais, de altitude e posição.

Uma conversão necessária é a transformação do modelo geométrico baseado numa elipse, sistema WGS-84, que trata-se dum Sistema de Referência Terrestre Convencional (CTRS), para um sistema de coordenadas cartesianas que facilite a observação do espaço e os cálculos, sistema North-East-Down (NED), utilizado também na aviação, especificada adiante [20].

A azimute do veículo possui três dimensões, através dos ângulos de Euler, ϕ_E, θ_E e ψ_E, obtidos tendo em conta o ponto inicial dado pelo GPS e os dados de IMU medidos ao logo do tempo, especificada adiante.

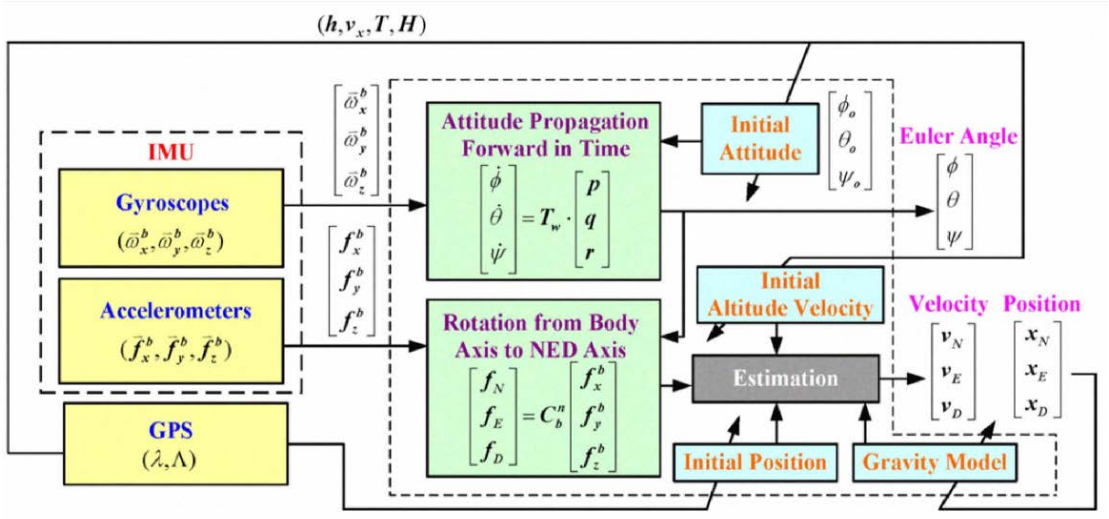


Figura 2.8 - Diagrama de blocos do algoritmo de Taiwan (extraído de [10]).

Dado um cenário de possível colisão, Figura 2.9, onde A e B simbolizam veículos posicionados em (Λ_1, λ_1) e (Λ_0, λ_0), no sistema WGS-84, com velocidades instantâneas de V_A e V_B, respetivamente, e considerando para ambos os pontos uma zona de possível erro em seu redor, *accuracy bubble* [18].

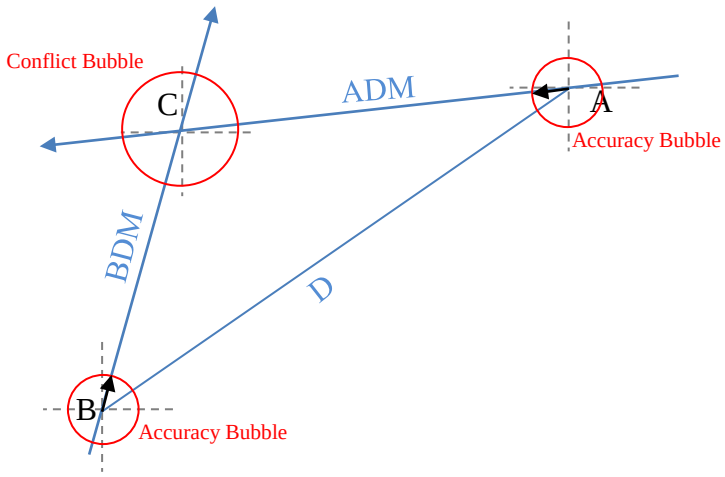


Figura 2.9 - Modelo geométrico de colisão no algoritmo de Taiwan.

O ponto C representa o ponto de colisão, que, num modo geral, acontece quando ambos os veículos possuem as mesmas componentes (Λ, λ, t) (latitude, longitude e tempo), deixando de ter relevância quando estes diferem.

A distância relativa entre os dois veículos é dada por:

$$D = \sqrt{[110946{,}2573 \times (\Lambda_1 - \Lambda_0)]^2 + [111319{,}4907 \times \cos(\lambda_0) \times (\lambda_1 - \lambda_0)]^2}. \quad \textbf{(2.35)}$$

O GPS fornece a última posição do veículo, por isso, é necessário estimar a posição corrente, através da equação matricial

$$\begin{bmatrix} \Lambda_k \\ \lambda_k \end{bmatrix} = \begin{bmatrix} \Lambda_{k-1} \\ \lambda_{k-1} \end{bmatrix} + \begin{bmatrix} \frac{1}{R_m + h} & 0 \\ 0 & \frac{1}{(R_N + h)\cos(\Lambda_{k-1})} \end{bmatrix} \begin{bmatrix} v_N \\ v_E \end{bmatrix} \Delta t\,, \quad \textbf{(2.36)}$$

onde R_m é o raio de curvatura do meridiano (*radius of curvature in Meridian)* e R_N o raio de curvatura do primeiro vertical (*radius of curvature in prime vertival)* [19].

A primeira transformação é do sistema WGS-84 para ECEF (*Earth-Centered Earth-Fixed*), e só depois para NED, utilizando as equações **(2.37)** e **(2.38)**. A altitude *h* é dada no formato

GPGGA pelo GPS, e os restantes parâmetros são a excentricidade *e* e o semieixo maior *a*. A equação **(2.37)** resulta da Terra não assumir uma forma esférica mas sim elipsoidal [20].

$$\begin{bmatrix} x^E \\ y^E \\ z^E \end{bmatrix} = \begin{bmatrix} (N+h)\cos\Lambda\cos\lambda \\ (N+h)\cos\Lambda\sin\lambda \\ [N(1-e^2)+h]\sin\Lambda \end{bmatrix}, \quad N = \frac{a}{\sqrt{1-e^2\sin^2\Lambda}} , \tag{2.37}$$

$$\begin{bmatrix} x^N \\ y^E \\ z^D \end{bmatrix} = \begin{bmatrix} -\cos(\lambda_0)\cdot\sin(\Lambda_0) & -\sin(\lambda_0)\cdot\sin(\Lambda_0) & \cos(\Lambda_0) \\ -\sin(\lambda_0) & \sin(\lambda_0) & 0 \\ -\cos(\lambda_0)\cdot\cos(\Lambda_0) & -\sin(\lambda_0)\cdot\sin(\Lambda_0) & -\sin(\Lambda_0) \end{bmatrix} \times \begin{bmatrix} x_1^E - x_0^E \\ y_1^E - y_0^E \\ z_1^E - z_0^E \end{bmatrix}. \tag{2.38}$$

Obtendo a posição dos veículos em coordenadas cartesianas prossegue-se ao cálculo das distâncias ADM e BDM (*Distance Margin*) e o tempo até a colisão, TTC, através de cálculos geométricos idênticos aos do tópico 2.3.1, tendo em conta a terceira dimensão.

Através das equações **(2.39)** e **(2.40)** obtém-se a posição corrente do veículo, longitude, latitude e altitude, $(\Lambda_t, \lambda_t, h_t)$, dependendo da posição anterior $(\Lambda_{t-1}, \lambda_{t-1}, h_{t-1})$ e os dados IMU medidos, aceleração dada pelos acelerómetros, f^n, e a velocidade angular dada pelos giroscópios, w^n. Onde $2w_{i/e}^n \times v_{t-1}^n$ corresponde à rotação da Terra, g^n à força da gravidade e $w_{e/n}^n \times v_{t-1}^n$ o efeito de Coriolis [21], sendo n a direção.

$$v_t^n = f^n - \left(w_{e/n}^n + 2w_{i/e}^n\right) \times v_{t-1}^n + g^n \quad , \tag{2.39}$$

$$\begin{bmatrix} \Lambda_t \\ \lambda_t \\ h_t \end{bmatrix} = \begin{bmatrix} \frac{1}{M+h_{t-1}} & 0 & 0 \\ 0 & \frac{1}{(N+h_{t-1})\cdot\cos\Lambda_{t-1}} & 0 \\ 0 & 0 & -1 \end{bmatrix} \times \begin{bmatrix} v^{north} \\ v^{east} \\ v^{down} \end{bmatrix} , \tag{2.40}$$

2.4 Trajetória

Existem aspetos importantes para além do estudo de trajetórias retilíneas, podendo resultar falsos avisos pois perdem a validade quando as rodas dos veículos se encontram voltadas para um dos lados, por esta razão, faz sentido a introdução aos algoritmos tendo em conta a previsão da trajetória do veículo.

2.4.1 Modelo bicicleta

Este modelo consiste na projeção da trajetória futura através da integração numérica dum modelo dinâmico. Utiliza-se um modelo simples 2-DOF (*Degrees of freedom)* chamado de "modelo bicicleta", como se pode observar na Figura 2.10, representando um corpo-fixo do veículo na coordenada (x, y) [22].

O modelo dinâmico descreve a variação da velocidade v de translação do veículo ao longo do eixo lateral do corpo-fixo, e a velocidade angular sobre um eixo apontar diretamente para baixo, i.e., a taxa de guinada r, assumindo que a massa suspensa permanece paralela ao plano da estrada. Denotando δ o ângulo de direção da roda dianteira (ângulo entre o plano da roda dianteira e eixo longitudinal do veículo), como mostra na Figura 2.10. Dados os valores de rigidez em curva dos pneus dianteiros e traseiros, C_f e C_r respetivamente, que são constantes de proporcionalidade entre o ângulo de deslizamento da roda (ângulo entre o plano da roda dianteira e do vetor de velocidade) e as forças laterais que atuam sobre os pneus a partir da superfície da estrada. Seja *m* a massa do veículo, *u* a velocidade do veículo (assumida constante ao longo do intervalo de previsão), I_Z o momento de inércia da guinada, *a* e *b* as distâncias desde o CoG até ambos os eixos, e por último e_{F_y} e e_{M_z}representam a perturbação lateral equivalente e aceleração da guinada, respetivamente. A soma das forças laterais e momentos de guinada produz a dinâmica do modelo da bicicleta,

$$\begin{bmatrix} \dot{v} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} \frac{1}{mu}\left(C_r + C_f\right) & \frac{1}{mu}\left(-bC_r + aC_f - mu^2\right) \\ \frac{1}{I_z u}\left(aC_f - bC_r\right) & \frac{1}{I_z u}\left(a^2 C_r + b^2 C_f\right) \end{bmatrix} \begin{bmatrix} v \\ r \end{bmatrix} + \begin{bmatrix} -\frac{C_f}{m} \\ -\frac{aC_f}{I_z} \end{bmatrix} \delta + \begin{bmatrix} e_{F_y} \\ e_{M_z} \end{bmatrix} \quad . \qquad \textbf{(2.41)}$$

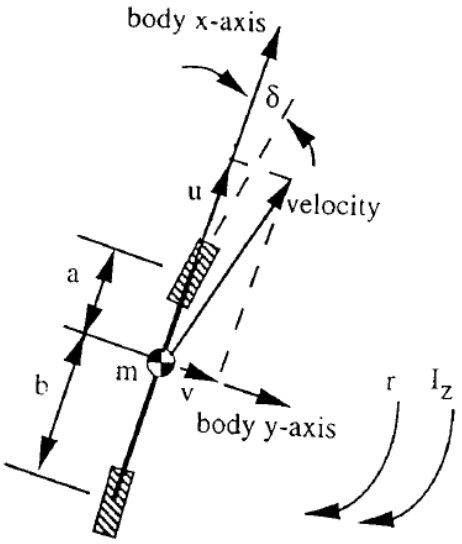


Figura 2.10 - Modelo bicicleta (extraído de [22]).

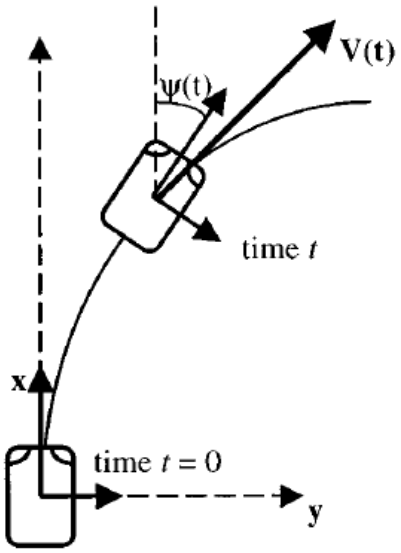


Figura 2.11 - Projeção da posição do veículo (extraído de [22]).

A posição e direção do veículo são calculadas aumentando este modelo com variáveis de deslocamento. Sendo *V(t)* a velocidade de translação do CoG no instante *t*, como está representado na Figura 2.11, denotando V_x e V_y as suas componentes, e $\psi(t)$ a variação da direção do veículo entre o intervalo de tempo 0 e *t*. Tem-se

$$\begin{aligned} V_x(t) &= u(t)\cos\psi(t) - v(t)\sin\psi(t) \\ V_y(t) &= u(t)\sin\psi(t) - v(t)\cos\psi(t) \end{aligned} \quad . \tag{2.42}$$

Assumindo que as variações da direção e da velocidade lateral absoluta são reduzidas, durante o intervalo de tempo de predição, tem-se

$$\begin{aligned} V_x(t) &= u \\ V_y(t) &= u \cdot \psi(t) + v(t) \end{aligned} \quad , \tag{2.43}$$

A velocidade *u* do veículo assume-se constante em todo o período da projeção da trajetória. Tendo as componentes de velocidade em *x* e y, a localização do CoG do veículo no instante de tempo *t*, é dada por:

$$\begin{aligned} x_v(t) &= \int_0^t V_x(t)dt = ut \\ y_v(t) &= \int_0^t V_y(t)dt \end{aligned} \quad , \tag{2.44}$$

Pela combinação das equações **(2.41)** e **(2.43)** e adicionando o ângulo relativo à direção do veículo ao modelo dinâmico, obtém-se

$$\begin{bmatrix} \dot{y}_v \\ \dot{v} \\ \dot{r} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 0 & \frac{C_r + C_f}{mu} & \frac{-bC_r + aC_f - mu^2}{mu} & 0 \\ 0 & \frac{aC_f - bC_r}{I_z u} & \frac{a^2 C_r + b^2 C_f}{I_z u} & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} y_v \\ v \\ r \\ \psi \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ -\frac{C_f}{m} & 1 & 0 \\ -\frac{aC_f}{I_z} & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \delta \\ e_{F_y} \\ e_{M_z} \end{bmatrix}, \qquad \textbf{(2.45)}$$

Estes são os primeiros resultados, e os mais simples, do "modelo bicicleta", porém, pode-se adicionar diversos distúrbios externos, alterando a trajetória para uma aproximação a uma parábola, Figura 2.12 [22].

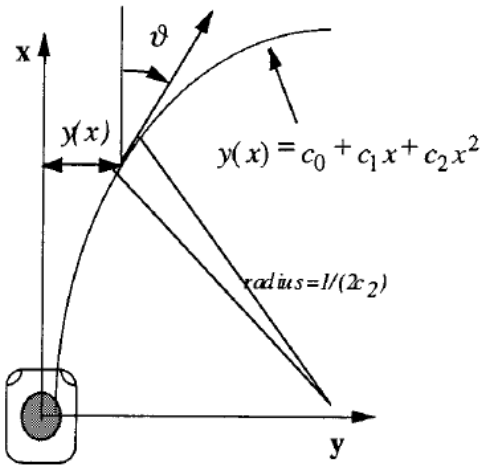


Figura 2.12 - Aproximação da trajetória a uma parábola (extraído de [22]).

Já foram estudados inúmeros algoritmos idênticos ao "modelo bicicleta", tendo em conta os quatro eixos do veículo, num modo geral, até cada roda poder tomar uma direção independente, com maior utilização na área da robótica [23][24]. Introduzindo um processamento mais complexo, o que, para a dimensão de erros, possíveis imprecisões do GPS, alteração repentina de trajetória, entre outros fatores, não será vantajoso face o modelo mais simples.

3 Desenvolvimento e Implementação

Com base nos objetivos e nos problemas não resolvidos pelos algoritmos estudados são propostas algumas soluções neste capitulo, destacando-se a adição de novos conceitos teóricos e a implementação de do sistema completo ao nível de rede e aplicacional assim como o *hardware* necessário.

3.1 Complemento aos algoritmos estudados

Apenas sobre o fundamento teórico não é possível uma implementação direta, pelo qual é necessário um desenvolvimento adicional aos algoritmos estudados, incluindo novas definições e algoritmos para o aperfeiçoamento do resultado final. Desta forma, as próximas secções descrevem os complementos aos algoritmos propostos e implementados nesta dissertação.

3.1.1 Desenvolvimento dos algoritmos estudados no estado da arte

Tomando partido dos modelos destacados no capítulo 2 relativos aos diversos algoritmos, não se obtém uma forma direta de aplicação num suporte digital, visto a maioria conter apenas o fundamento e a teoria, pelo que são necessárias algumas deduções e adaptações.

- *Algoritmos unidimensionais*

Após uma primeira fase de pré-filtros e decisão dos algoritmos a executar, é necessária a avaliação de certos fatores. A uma dimensão, é necessário saber se os dados recebidos fazem parte dum veículo procedente, antecedente ou se são irrelevantes, fazendo sentido apenas para veículos procedentes.

Uma forma rápida de deteção é fazendo uma deslocação espacial colocando o nosso veículo na origem, sabe-se assim que o veículo é procedente, se e só se, o ângulo que a posição do veículo faz com o eixo das abcissas for igual ao ângulo de deslocação, caso o veículo seja antecedente esse ângulo é igual ao da deslocação mais 180º, note-se, é necessário somar 180º

para pontos no segundo e terceiro quadrante, $x < 0$. Pode-se observar geometricamente este método na Figura 3.1. Obtém-se o ângulo θ_1 que a posição do veículo faz com o eixo das abcissas facilmente através da expressão:

$$\theta_1 = \operatorname{atan}\left(\frac{y_2 - y_1}{x_2 - x_1}\right) \quad , \tag{3.1}$$

onde (x_1, y_1) é a posição do nosso veículo e (x_2, y_2) do veículo procedente ou antecedente.

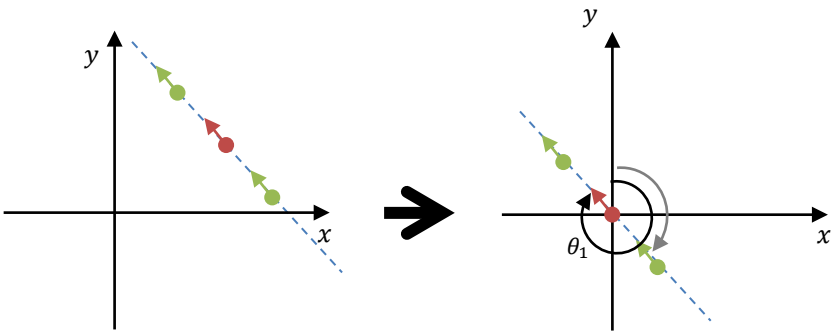


Figura 3.1 – Método de distinção de veículo procedente e antecedente.

Os algoritmos de uma dimensão apenas fornecem a distância ao veículo procedente conveniente para alertar o condutor, deve ser comparada com a distância corrente *d* entre ambos, sendo dada por:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \quad . \tag{3.2}$$

- *IET*

Completando o raciocínio do algoritmo IET, tópico 2.3.1, tomando partido das equações **(2.12)**, **(2.18)** e **(2.20)**, relativas à deslocação em *x* e *y* e a condição de colisão, deduz-se:

$$\begin{cases} x_n(t) = x_n(0) + v_n \times t \times \cos\varphi_n \\ y_n(t) = y_n(0) + v_n \times t \times \sin\varphi_n \\ \sqrt{\left(x_1(TTC) - x_2(TTC)\right)^2 + \left(y_1(TTC) - y_2(TTC)\right)^2} = V \end{cases} \Leftrightarrow$$

$$\Leftrightarrow \left(x_1(TTC) - x_2(TTC)\right)^2 + \left(y_1(TTC) - y_2(TTC)\right)^2 = V^2$$

$$\Leftrightarrow (x_1(0) + v_1 TTC\cos\varphi_1 - x_2(0) - v_2 TTC\cos\varphi_2)^2 + (y_1(0) + v_1 TTC\sin\varphi_1 - y_2(0) - v_2 TTC\sin\varphi_2)^2 = V^2$$

$$\Leftrightarrow \big(TTC(v_1\cos\varphi_1 - v_2\cos\varphi_2) + x_1(0) - x_2(0)\big)^2 + \big(TTC(v_1\sin\varphi_1 - v_2\sin\varphi_2) + y_1(0) - y_2(0)\big)^2 = V^2$$

$$\Leftrightarrow TTC^2(v_1\cos\varphi_1 - v_2\cos\varphi_2)^2 + 2TTC(v_1\cos\varphi_1 - v_2\cos\varphi_2)\big(x_1(0) - x_2(0)\big) + \big(x_1(0) - x_2(0)\big)^2 + TTC^2(v_1\sin\varphi_1 - v_2\sin\varphi_2)^2 + 2TTC(v_1\sin\varphi_1 - v_2\sin\varphi_2)\big(y_1(0) - y_2(0)\big) + \big(y_1(0) - y_2(0)\big)^2 = V^2$$

$$\Leftrightarrow TTC^2[(v_1\cos\varphi_1 - v_2\cos\varphi_2)^2 + (v_1\sin\varphi_1 - v_2\sin\varphi_2)^2] + TTC\big[2(v_1\cos\varphi_1 - v_2\cos\varphi_2)\big(x_1(0) - x_2(0)\big) + 2(v_1\sin\varphi_1 - v_2\sin\varphi_2)\big(y_1(0) - y_2(0)\big)\big] + \big(x_1(0) - x_2(0)\big)^2 + \big(y_1(0) - y_2(0)\big)^2 = V^2$$

$$\Leftrightarrow a \times TTC^2 + b \times TTC + c = 0$$

sendo

$$a = (v_1\cos\varphi_1 - v_2\cos\varphi_2)^2 + (v_1\sin\varphi_1 - v_2\sin\varphi_2)^2$$
$$= v_1^2\cos^2\varphi_1 - 2v_1\cos\varphi_1\, v_2\cos\varphi_2 + v_2^2\cos^2\varphi_2 + v_1^2\sin^2\varphi_1 - 2v_1\sin\varphi_1\, v_2\sin\varphi_2 + v_2^2\sin^2\varphi_2$$
$$= v_1^2(\cos^2\varphi_1 + \sin^2\varphi_1) + v_2^2(\cos^2\varphi_2 + \sin^2\varphi_2) - 2v_1\cos\varphi_1\, v_2\cos\varphi_2 - 2v_1\sin\varphi_1\, v_2\sin\varphi_2$$
$$= v_1^2 + v_2^2 - 2v_1\cos\varphi_1\, v_2\cos\varphi_2 - 2v_1\sin\varphi_1\, v_2\sin\varphi_2$$

$$b = 2(v_1\cos\varphi_1 - v_2\cos\varphi_2)\big(x_1(0) - x_2(0)\big) + 2(v_1\sin\varphi_1 - v_2\sin\varphi_2)\big(y_1(0) - y_2(0)\big)$$

$$c = \big(x_1(0) - x_2(0)\big)^2 + \big(y_1(0) - y_2(0)\big)^2 - V^2$$

Concluindo que, dois veículos estão em rota de colisão se

$$b^2 - 4ac > 0 \quad , \tag{3.3}$$

condição para que os resultados da fórmula resolvente sejam válidos, isto é, não sejam imaginários. Podendo extrair o tempo até a colisão, TTC, a partir da fórmula resolvente.

- *Safety Zones*

No algoritmo *Safety Zones*, da subsecção 2.3.3, não foi mencionada a deslocação do centro do retângulo virtual, pois este deixou de ser o centro do veículo como no algoritmo IET. É alterado de modo a que o veículo seja colocado junto à extremidade do retângulo virtual mais próxima da traseira, como está presente na Figura 3.2. O centro corresponde à coordenada (x_2, y_2), calculado através das seguintes expressões obtidas com base na Figura 3.2:

$$hip = \frac{L + d_w}{2} - \frac{L}{2} = \frac{d_w}{2} \quad , \quad \beta = 90 - \varphi \quad , \quad \Delta x = \sin(\beta)\, h \quad , \quad \Delta y = \cos(\beta)\, hip \quad ,$$

$$x_2 = x_1 + \Delta x \quad , \quad y_2 = y_1 + \Delta y$$

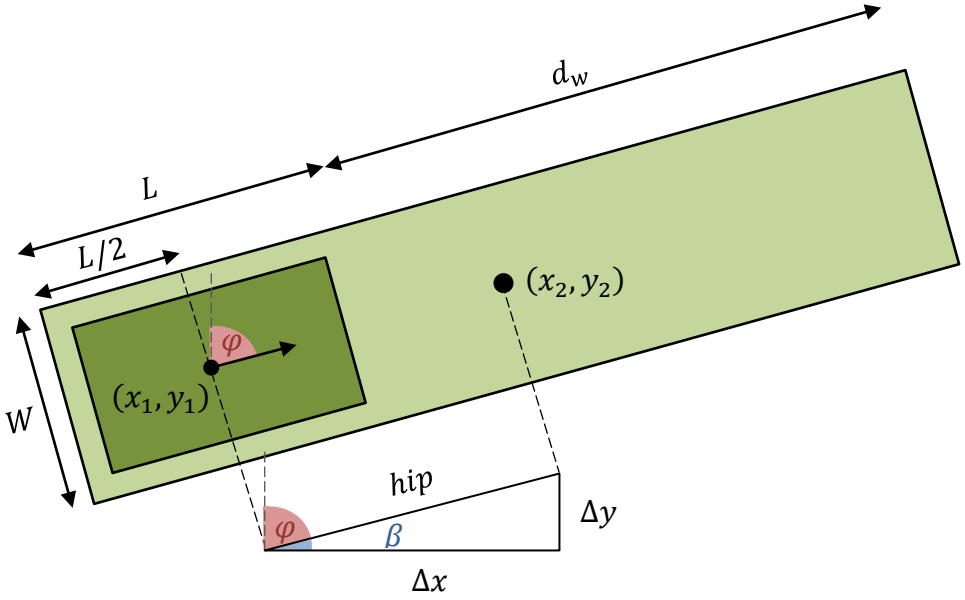


Figura 3.2 – Geometria da deslocação do centro do retângulo virtual.

3.1.2 Falsos avisos

Um aspeto a eliminar, para se garantir um sistema fiável, é o fato dos algoritmos poderem gerar falsos avisos, perdendo a confiança do condutor. Garantir um sistema fiável é assim um dos objetivos desta tese, por esta razão, a eliminação de falsos avisos é uma prioridade, com o cuidado de não omitir situações de perigo iminente.

Para o cenário de aproximação súbita do veículo procedente, na aplicação dos algoritmos a uma dimensão, são visíveis três cenários diferentes que geram o mesmo aviso na Figura 3.3. À esquerda encontra-se a situação correta de aviso, o veículo procedente, vermelho, encontra-se parado e o "nosso" veículo, azul, deverá ser alertado. Ao centro encontra-se a situação em que o veículo procedente se encontra parado no estacionamento e à direita o veículo procedente encontra-se em direção oposta, estes dois cenários geraram falsos avisos implementando apenas os algoritmos estudados, devido às imprecisões do sistema de GPS.

O último cenário é facilmente resolvido através do azimute dos veículos, caso apresentem uma diferença de aproximadamente 180º significa que o veículo se encontra em sentido inverso. Exclui-se assim a colisão frontal, por outro lado, os dados obtidos dum sistema de GPS comum não garantem precisão suficiente para diferenciar vias de trânsito da faixa de rodagem, devido aos erros poderem chegar até 10 m [25], o mesmo motivo pelo qual a situação ao centro da Figura 3.3 ser de difícil deteção.

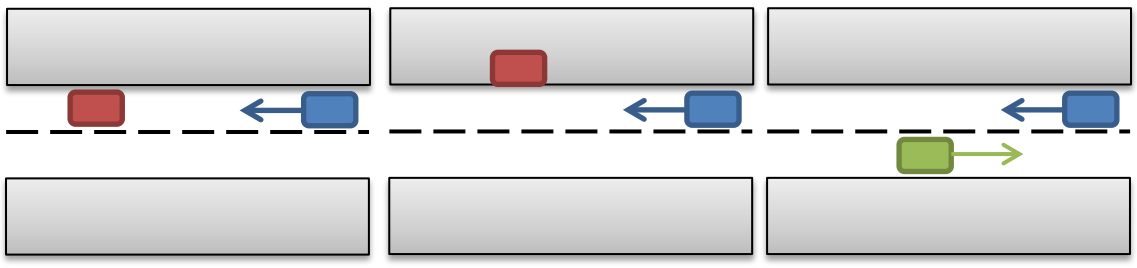


Figura 3.3 – Três cenários unidimensionais distintos geradores do mesmo aviso.

Nos algoritmos bidimensionais também se pode observar três cenários distintos que geram o mesmo aviso na Figura 3.7, a confusão entre entroncamento e curva, ou, a situação à direita, o caso de um dos veículos se encontrar sobre uma ponte.

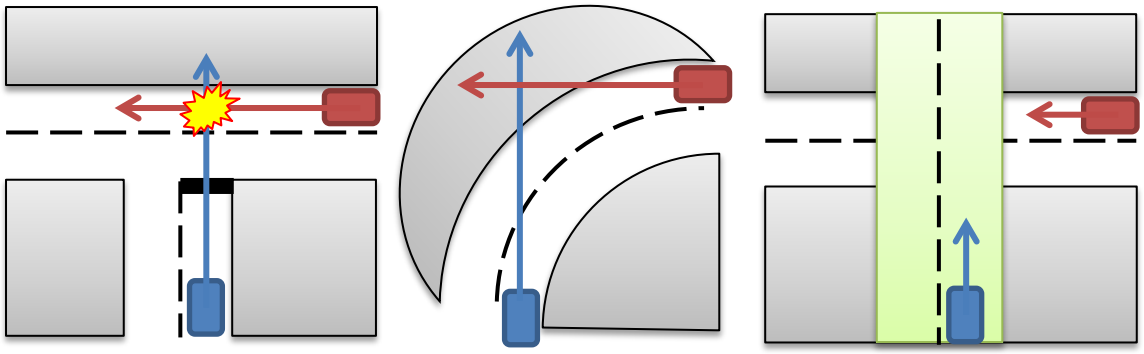


Figura 3.4 – Três cenários bidimensionais distintos geradores do mesmo aviso.

O complemento que se pretende adicionar para melhoria destas situações, é a gravação em memória das posições anteriores dos veículos na vizinhança, sendo este aspecto inovador e proposto nesta tese como algoritmo Cobra. Desta forma, o "modelo bicicleta", subsecção 2.4.1, usado para correção de falsos avisos, deixa de ser vantajoso, não só pelo facto de não se ter conhecimento direto do ângulo da posição das rodas dianteiras, mas também pela carga de processamento, pois tendo em memória a trajetória passada consegue-se prever, com menos processamento, se o veículo se encontra a efetuar uma curva ou uma reta, corrigindo alguns problemas destacados na Figura 3.3 e Figura 3.7, mais precisamente a ambiguidade entre curva e entroncamento e a situação do veículo no estacionamento.

A situação em que um dos veículos se encontra sobre uma ponte é resolvida pelo algoritmo em Taiwan estudado na subsecção 2.3.4, o que implica o uso do CAN *bus*, giroscópios e acelerómetros, o que não é implementável nesta tese.

Todos estes cenários de falsos avisos eram mitigados com a inclusão de mapas, o que também não é considerado nesta tese, mas que seria a melhor opção num projeto futuro, interligando os algoritmos com o sistema de GPS vulgar de planificação de trajetos.

Outro aspeto que poderá ser fruto de falsos avisos é o fato dos algoritmos utilizarem apenas um instante, não tendo conhecimento se o veículo se encontra numa aceleração ou desaceleração. Numa situação de cruzamento, em que um veículo circula a uma velocidade excessiva reduz com precaução para uma velocidade apropriada ao cruzamento, num instante desse procedimento o veículo tem fortes possibilidades de estar em rota de colisão, mas que, segundo os algoritmos estudados, seria gerado um falso aviso pois os algoritmos não têm

conhecimento da desaceleração do veículo. Este problema é facilmente resolvido com base no algoritmo proposto o Cobra, método explicado adiante.

3.1.3 Filtragem

O procedimento de filtragem consiste em todas as verificações executadas sobre os dados antes destes serem entregues aos algoritmos, descartando o mais rapidamente possível os dados inúteis, diminuindo assim o número de cálculos necessários e de possíveis falsos avisos. A maioria das verificações consta no algoritmo Cobra, descritas na subsecção seguinte.

A primeira filtragem é eliminar os veículos em sentido oposto, como já foi referido na subsecção anterior. Caso os dados recebidos contenham informação de azimute com aproximadamente mais 180 graus que o nosso veículo, são descartados, resultado da seguinte verificação:

$$|(\varphi - \varphi_1) - 180°| < E, \quad \textbf{(3.4)}$$

onde φ é o azimute do nosso veículo e φ_1 o azimute do outro veículo, dando uma margem definida por *E*.

Outro filtro necessário atua sobre os avisos precoces, detetados com base no valor do TTC, aplicável nos algoritmos bidimensionais. Caso o valor do TTC seja superior a um determinado valor, dependendo da velocidade corrente, os avisos não são relevantes pois muitas alterações na velocidade e direção do veículo podem acontecer e deverão ser omitidos.

3.1.4 Algoritmo Cobra

O algoritmo Cobra, idealizado e desenvolvido nesta tese, tem a finalidade de agir no terceiro cenário, isto é, velocidade excessiva para um determinado raio de curva. De acordo com os documentos estudados, ainda não se encontra em desenvolvimento por outras entidades, sendo uma novidade nesta área.

A definição do algoritmo Cobra consiste no conhecimento do rasto dos veículos na vizinhança, coordenadas geográficas, velocidades e direções, presentes e passadas, sendo por isso chamado de cobra.

Adotando uma solução em que se conheça a cobra dos veículos procedentes, ajuda na correção de falsos alarmes, por exemplo, sabendo que o veículo da frente efetuou uma curva, e não uma mudança de direção (distintas pelo raio de curvatura e velocidade média), sabe-se que não se procede um cruzamento, sendo os alarmes gerados pelos algoritmos em 2.3 incorretos. Podendo desta forma criar um outro algoritmo de segurança que indique que o condutor se encontra em excesso de velocidade para a curva que o procede, nomeando de algoritmo Curva, destacado na secção 3.1.5.

- *Aceleração*

Quando dois veículos se encontram em rota de colisão, com base no algoritmo IET, considera-se apenas um ponto de cada veículo, ou seja, um instante. Com base num instante não se consegue concluir se o veículo tenciona abrandar, situação de possível geração de falso aviso anteriormente mencionada. Com base na cobra consegue-se extrair toda essa informação, por exemplo, efetuando a média de aceleração do veículo dos últimos três pontos da cobra consegue-se saber o estado do veículo, e omitir o aviso caso este apresente uma desaceleração considerável.

- *Paragens*

Uma das informações obtidas através do sistema de GPS que, se não contiver um pré tratamento, é falsa, é o azimute quando não existe movimento, ou seja, situação de paragem num sinal de STOP, luminoso vermelho, numa passadeira, num estacionamento, etc. O algoritmo de cobra poderá corrigir este problema, assim que recebe um valor de velocidade inferior a, por exemplo, 2 km/h guarda os dados na sua estrutura normalmente à exceção do

azimute, que guardará igual ao da posição anterior da cobra, e só voltará a adicionar dados à cobra assim que estes apresentem movimento do veículo, eliminando assim pontos consecutivos da cobra com a mesma informação.

- *Estacionamento*

O estacionamento é uma situação destacada como falso aviso, confundido com um veículo parado na faixa de rodagem devido às imprecisões do sistema GPS. Sabendo a cobra de um veículo é facilmente detetado quando este se encontra num estacionamento, se todas as coordenadas dessa cobra estiverem contidas num raio mínimo. Uma forma de se fazer esta deteção é utilizando a expressão matemática da circunferência, ou seja, considera-se um veículo no estacionamento se todas as coordenadas da sua cobra cumprirem a expressão

$$(x - a_c)^2 + (y - b_c)^2 < R_{min}^2, \quad \textbf{(3.5)}$$

onde R_{min} é o raio mínimo, (x, y) a coordenada a verificar e (a_c, b_c) o centro da circunferência, que corresponde ao último ponto da cobra.

Esta situação pode não ser ignorada de todo, mas sim, emitir um aviso, não tão relevante como o aviso de risco de colisão, mas que indique aos restantes automobilistas que um veículo se encontra a sair do estacionamento.

3.1.5 Algoritmo Curva

Como o algoritmo Cobra é novidade, o algoritmo Curva, que é um derivado seu, também o é. Este algoritmo consiste em alertar o condutor que circula a uma velocidade excessiva para a curva que se aproxima, conhecendo os dados da curva enviados da "Cobra" do veículo procedente. É também ele que fará a distinção entre cruzamento/entroncamento e curva, um dos aspetos relatados como gerador de falsos avisos.

- *Obtenção do raio da curva*

A obtenção do raio da curva não é direta, mas sim através duma regressão quadrática, ou outra regressão idêntica, não sendo a melhor solução, pois a cobra do veículo procedente, para além de erros provenientes do GPS, pode não conter apenas as coordenadas relativas à curva, mas também uma reta antes e/ou depois, o que iria aumentar o erro da regressão. Deste modo, a forma mais simples é proceder a um algoritmo iterativo, percorrendo os pontos da cobra até detetar uma mudança do azimute do veículo, e registar esse ponto e o ponto até onde o azimute alterou, isto é, até ao ponto em que desde este o azimute mantenha aproximadamente o mesmo valor ou seguiu o sentido inverso.

Tendo os dois pontos, de início (x_1, y_1) e fim (x_2, y_2) da curva, pode-se coloca-los sobre uma circunferência e retirar o valor do seu raio, como está representado na Figura 3.5. Ao unir os dois pontos, d_p, e o centro da circunferência forma-se um triângulo isósceles, onde o ângulo β, no centro da circunferência, é dado pelo módulo da subtração dos dois valores do azimute extraídos da cobra φ_1 e φ_2, repartindo o triângulo em dois triângulos retângulos, onde a hipotenusa é o raio da curva *R*, deste modo consegue-se deduzir a seguinte expressão:

$$R = \frac{\frac{d}{2}}{sen\left(\frac{\beta}{2}\right)}\ ,\ \ \beta = |\varphi_2 - \varphi_1|\ ,\qquad d_p = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}, \tag{3.6}$$

Só se considera uma curva ou cruzamento se o ângulo β for superior a 40°, uma filtragem útil para descartar dados de curvas provenientes de erros do sistema de GPS ou de uma manobra irrelevante do veículo, por exemplo, evitar uma deformação no pavimento.

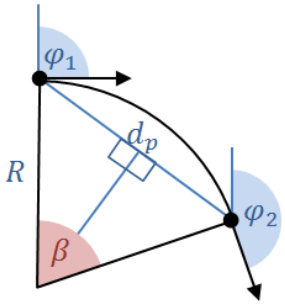


Figura 3.5 – Representação trigonométrica duma curva.

Este método não é ideal, pois os veículos não apresentam trajetórias circulares perfeitas em curvas, assumem formas maioritariamente ovais, o que gerará erros nos resultados. Considerando que o mundo real e não digital, ser constituído por imprecisões, os erros podem não ser relevantes, sendo isto confirmado pelos testes do algoritmo presentes no capítulo 4.

- *Obtenção da velocidade máxima*

Na área da engenharia Civil, já foram estudados modelos para a projeção do raio do arco circular duma curva em função da velocidade máxima destinada a essa faixa de rodagem.

A segurança do veículo e seus ocupantes numa curva está relacionada à força centrífuga a que este está sujeito, cuja intensidade é dada por:

$F = m \times u^2/R,$

em que m representa a massa do veículo, u a velocidade e R o raio da curva. A resistência a esta força centrífuga é assegurada pela força de atrito transversal mobilizada na interface pneu-pavimento e pela componente do peso do veículo, com sentido contrário ao da força centrífuga, gerada pela introdução da sobrelevação. Nesta situação, a condição de equilíbrio em relação ao deslizamento lateral, permite deduzir, segundo [26], a seguinte relação:

$$R = \frac{V^2}{127\ (f_t + Se)}, \tag{3.7}$$

onde Se é a sobrelevação em percentagem e f_t o coeficiente de atrito transversal.

Em vários países europeus, a sobrelevação é limitada atualmente a um máximo de 7%. Atribuindo valores máximos ao coeficiente de atrito transversal obtém-se os valores presentes na Tabela 3.1 [26].

Tabela 3.1 – Coeficientes de atrito transversal (valores extraídos de [26]).

Velocidade base (km/h)	f_t
40	0.16
50	0.16
60	0.15
70	0.14
80	0.14
90	0.13
100	0.12
110	0.10
120	0.09
130	0.08
140	0.06

Existem duas definições de raios de curvatura, raios mínimos absolutos (RA) e raios mínimos normais (RN). Os RA são dados pela expressão em **(3.7)** e são valores para acelerações centrífugas elevadas, próximos dos valores máximos admissíveis. Os RN devem assegurar uma circulação segura e cómoda, e são determinados de modo a que a aceleração centrífuga tenha um valor correspondente a 50% do valor máximo admissível. Por fim, obtém-se os valores presentes na Tabela 3.2.

Tabela 3.2 – Raios de curvatura mínimos (valores extraídos de [26]).

Velocidade base (km/h)	RA (m)	RN (m)
40	55	110
50	85	180
60	130	250
70	180	350
80	240	450
90	320	550
100	420	700
110	560	850
120	700	1000
130	900	1200

140	1200	1400

Através dos valores da Tabela 3.2 consegue-se gerar dois avisos ao condutor em função do raio da curva que o procede, por exemplo, para uma curva com um raio de 180 m, caso o veículo circule a uma velocidade acima dos 50 km/h, corresponde ao RN, deve ser gerado um aviso ligeiro, caso circule a uma velocidade superior a 70 km/h, corresponde ao RA, deve ser gerado um aviso mais consistente, pois já se encontra a uma velocidade base superior à definida para o raio mínimo absoluto.

A distinção entre curva e cruzamento é obtida facilmente através do raio obtido por este algoritmo, num cruzamento os veículos descrevem raios até 10 m, de modo a efetuarem a chamada perpendicular no código da estrada, como se poderá confirmar nos exemplos práticos do capítulo 4.

- *Área de aviso*

Após obtenção do raio da curva, é necessário saber quando avisar o condutor. A solução adotada consta na criação de uma área de aviso antes da curva, assim que a posição do veículo é assumida no interior da área, e apresenta um azimute na direção da curva, é verificada a sua velocidade e aciona-se um aviso ao condutor se a velocidade for excessiva.

Por motivos de simplificação de processamento considera-se uma área circular, facilmente se deteta um ponto *(x,y)* no interior de um círculo centrado em (a_c,b_c) de raio R_w através da seguinte verificação:

$$(x - a_c)^2 + (y - b_c)^2 < R_w^2.$$

O valor de R_w tem de ser de dimensão suficiente que abranja a largura da via da faixa de rodagem, um erro considerável de 10 m, que corresponde a ordem de grandeza dos erros possíveis em sistemas de GPS comuns [25], e garantir que pelo menos uma amostra de coordenadas esteja contida nessa área quando o veículo circular nessa faixa de rodagem, de modo a garantir a verificação. Assumindo a via da faixa de rodagem com 5 m de largura, uma velocidade máxima de 120 km/h, que corresponde a 33 m/s, resulta um raio fixo de 48 m, o que numa zona urbana pode abranger ruas paralelas e gerar falsos avisos.

Uma forma eficaz de evitar falsos avisos é considerar valores de raios dependentes da velocidade máxima para o raio da curva em questão, tendo o raio de curvatura da curva, com

base na Tabela 3.2, retira-se o valor da velocidade máxima V_{max}, calcula-se quantos metros percorre um veículo a essa velocidade num segundo e soma-se 15 m (5 m da largura da via da faixa de rodagem e 10 m de possíveis erros do sistema de GPS). Visto a tabela retirada da fonte não conter informação para velocidade inferiores a 40 km/h, atua-se diretamente na base que gerou a tabela, a equação **(3.7)**, assumindo o valor de coeficiente de atrito transversal, 0.16, e um valor de sobrelevação de 7%, obtém-se a seguinte expressão:

$$V_{max_{[\mathrm{km/h}]}} = \sqrt{R \times 127\,(f_t + Se)} = 5.4\sqrt{R}\,, \qquad \textbf{(3.8)}$$

que resulta $5.4\,\sqrt{R}/3.6 = 3\sqrt{R}/2$ m/s, tem-se então

$$R_{w_{[\mathrm{m}]}} = \frac{3\,\sqrt{R}}{2} + 15\,. \qquad \textbf{(3.9)}$$

O centro da área (a_c, b_c) é definido pelo ponto da cobra do veículo procedente relativo ao início da curva, ou em alternativa, utilizar não esse ponto mas sim pontos antecedentes, de modo a antecipar o aviso. Assume-se para cada curva um valor de azimute, φ_c, que corresponde ao do ponto da cobra considerado o centro da área de aviso, desta forma o veículo, na posição (x, y), com uma velocidade V e uma direção φ, é alertado quando:

- Se encontra na zona de aviso: $(x - a_c)^2 + (y - b_c)^2 < R_w^2$;
- Segue na direção da curva: $|\varphi - \varphi_c| < 20°$;
- Circula com velocidade excessiva para o raio da curva: $V > V_{max}$.

3.2 Implementação do *Hardware*

Pretende-se a implementação dum sistema completo, desde a camada superior, a interface com o utilizador e a implementação dos algoritmos estudados e desenvolvidos ao longo desta tese, até à camada física, a transmissão dos dados em RF, *Radio Frequency*, sobre as normas definidas.

3.2.1 Camada MAC e Física

As camadas inferiores, física e MAC, *Media Access Control*, são garantidas pelo sistema desenvolvido no decorrer do projeto HEADWAY, desenvolvido por membros de grupos de investigação e desenvolvimento do ISEL, GIEST (Grupo de Investigação em Eletrónica e Sistemas de Telecomunicações), e da Universidade de Aveiro, IT (Instituto de Telecomunicações), destinado para comunicações entre veículos sobre a norma 802.11p [1], na banda de frequência de 5.9 GHz. Pode-se observar o diagrama de blocos do *hardware* do sistema, implementado sobre uma placa PCB, *Printed Circuit Board*, na Figura 3.6 [27].

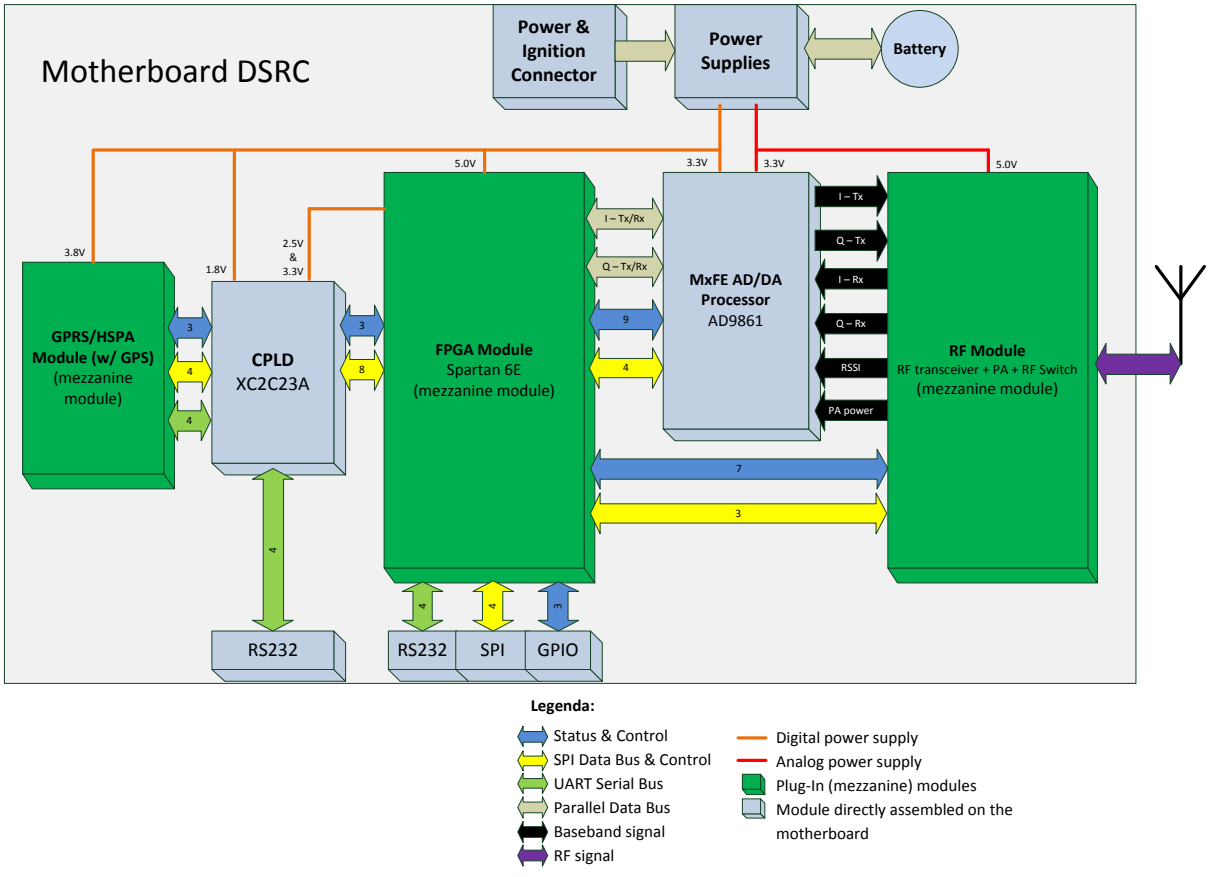


Figura 3.6 – Diagrama de blocos do *hardware* DSRC.

Tem como tensão de alimentação 12 V, correspondente à tensão disponível em veículos comuns. A placa mãe possui três conectores para módulos *mezzanines*, o módulo FPGA, onde estará implementado parte o fluxo da camada MAC e física de acordo com a norma 802.11p. O módulo RF, que está diretamente ligado à antena e fará a modulação e deslocação em frequência do sinal em banda base para RF, também este módulo foi desenvolvido pelos mesmos membros que a placa mãe, e o módulo GPRS, que tem como finalidade a atualização de *firmware* remota, não é usado nesta tese.

Anteriormente, a comunicação com as camadas superiores era feita através da porta RS232 disponibilizada na *motherboard*, e era utilizado o módulo FPGA Spartan 3E, mas, devido à falta de *slices* livres, o módulo foi substituído por o Spartan 6E, e para uma comunicação mais fiável e robusta foi substituída por ligação USB através da porta disponibilizada no módulo de FPGA.

Fora disponibilizado o código fonte em linguagem de programação C da biblioteca de comunicação com a camada MAC, que disponibiliza as seguintes primitivas:

- MA-UNITDATA (802.11) [1];
- MA-UNITDATAX (1609.3) [2];
- MA-UNITDATA-STATUS (802.11) [1].

3.2.2 Camada de Rede e Aplicação

Inicialmente, as camadas superiores estavam implementadas sobre o módulo TX27 [28], um sistema embebido com o sistema operativo Linux, *kernel* 2.6, processador Freescale i.MX27 400MHz, mas devido à falta de bibliotecas desenvolvidas para a sua arquitetura, nomeadamente a falta da biblioteca USB após a alteração da comunicação RS232 para USB, foi substituído por um módulo mais completo, Raspberry Pi Model B [29], um *single board computer*, com as seguintes características:

- ✓ CPU: 700 MHz Low Power ARM1176JZ-F [30];
- ✓ GPU: Dual Core VideoCore IV;
- ✓ Memória RAM: 512 MB SDRAM;
- ✓ Ethernet: onboard 10/100 RJ45 jack;

- ✓ USB: USB 2.0 Dual Connector;
- ✓ Vídeo: HDMI e Video composto RCA;
- ✓ Áudio: 3.5 mm jack e HDMI;
- ✓ Memória: SD, MMC, SDIO card slot;
- ✓ Dimensão: 8.6cm x 5.4cm x 1.7 cm.

Posteriormente foi-lhe instalado e configurado em função deste projeto o sistema operativo Arch Linux ARM. A alimentação é feita através dum conector micro-USB B, ou seja, 5 V, facilmente obtida num veículo particular comum utilizando um conversor USB.

Este sistema está capaz de assumir um dispositivo *vehicle-to-vehicle*, WAVE, assumindo um cargo de RSU, *Roadside Unit*, ou OBU, *Onboard Unit*, duas definições importantes nas normas 1609. RSU trata-se dum dispositivo WAVE que opera parado, disponibilizando serviços aos OBUs, está normalmente incorporado em infra-estruturas, tais como sinais de trânsito. OBU é um dispositivo WAVE que pode operar em movimento, fazendo trocas de dados com RSUs ou outros OBUs, estão normalmente incorporados em veículos ou podem ser portáteis, *hand-held*, destinado a utilizadores e tem como um dos principais objetivos a segurança ([31]). Nesta implementação tem-se os dois casos em cenário de testes, RSU e OBU, exemplificado adiante.

O fluxo da camada de rede disponibiliza dois serviços, mensagens WSMP, *Wave Short Messages Primitive*, para a comunicação com a camada superior, e LLC, *Logical Link Control*, na comunicação com a camada MAC. Pode-se observar o fluxo da camada de rede do envio e receção duma mensagem WSMP na Figura 3.7. O código, em linguagem de programação C, relativo à camada de rede, também foi desenvolvido no decorrer do projeto HEADWAY.

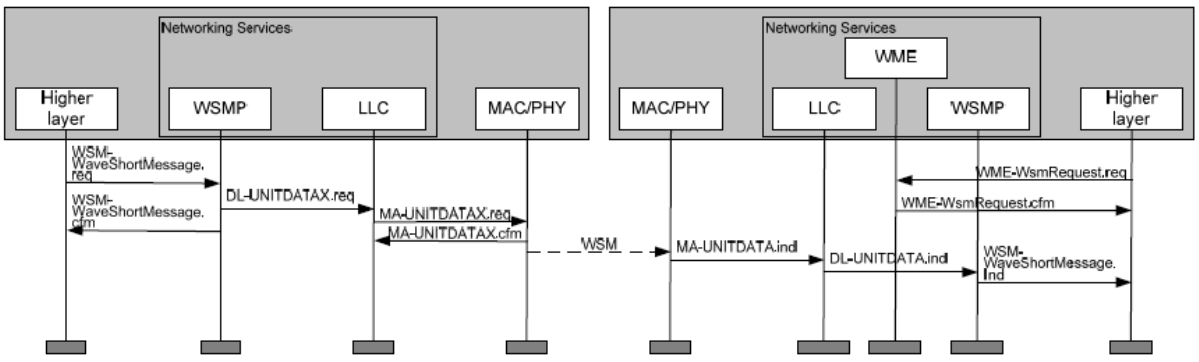


Figura 3.7 – Fluxo duma mensagem WSMP (retirado de [2]).

Como foi referido na introdução, a implementação das normas não são prioridade devido ao motivo já mencionado, no entanto, para manter um sistema compatível, ou de fácil portabilidade futura, o fluxo apresentado na Figura 3.7 é garantido construindo o pacote WSMP completo, como no exemplo da Figura 3.8, introduzindo valores neutros em campos não implementados. O fluxo foi desenvolvido de acordo com a norma 1609.3 [2].

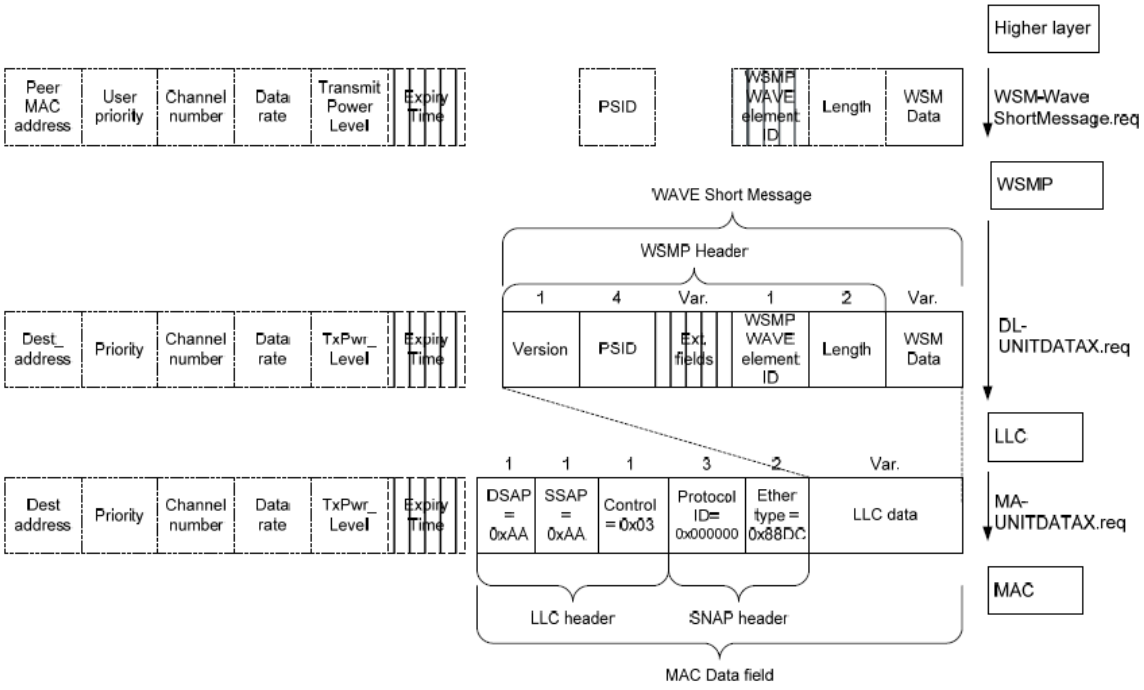


Figura 3.8 – Construção dum pacote WSMP. (retirado de [2]).

A camada de aplicação apenas disponibiliza um serviço, que implementa alguns dos algoritmos estudados e desenvolvidos. Para a comunicação com utilizador será utilizado um ecrã tátil, TFTCar CTF1020-5 10.2", ligado diretamente ao Raspberry Pi pela porta RCA de vídeo composto e USB de entrada tátil, alimentado com 12 VDC.

Outro componente importante para a realização deste trabalho é o sistema de GPS. Será utilizado o *kit* da Motorola M12 Evaluation Board, com o módulo GPS M12 Oncore, fornece dados WGS-84, comunica através de porta série, RS232, com um *baudrate* de 4800, sobre o protocolo NMEA 0183, *National Marine Electronics Association*.

3.2.3 Sistema completo

O OBU, implementado no veículo, deverá alertar o condutor de riscos de colisão. Numa primeira fase de testes, os dados dos restantes veículos são fornecidos a partir dum ficheiro criado com percursos anteriormente guardados com dados fornecidos pelo *kit* M12, utilizando para estes testes apenas três componentes, ecrã tátil, Raspberry Pi e *kit* M12.

Posteriormente pretende-se adicionar a transmissão dos dados via RF, daí a necessidade da utilização dum RSU, como foi referido, trata-se dum dispositivo que opera parado incorporado numa infraestrutura, ou seja, não necessita de ecrã nem GPS (para este caso), são maioritariamente controlados remotamente, neste caso será utilizada uma comunicação *ethernet.* O cargo do RSU será enviar dados de veículos simulados substituindo a presença dos mesmos, facilitando os testes e permitindo a presença de um número ilimitado de veículos, estará localizado no cruzamento de testes e é controlado remotamente através do *software* de simulação desenvolvido, especificado adiante. Pode-se observar uma ilustração do ambiente de testes na Figura 3.9.

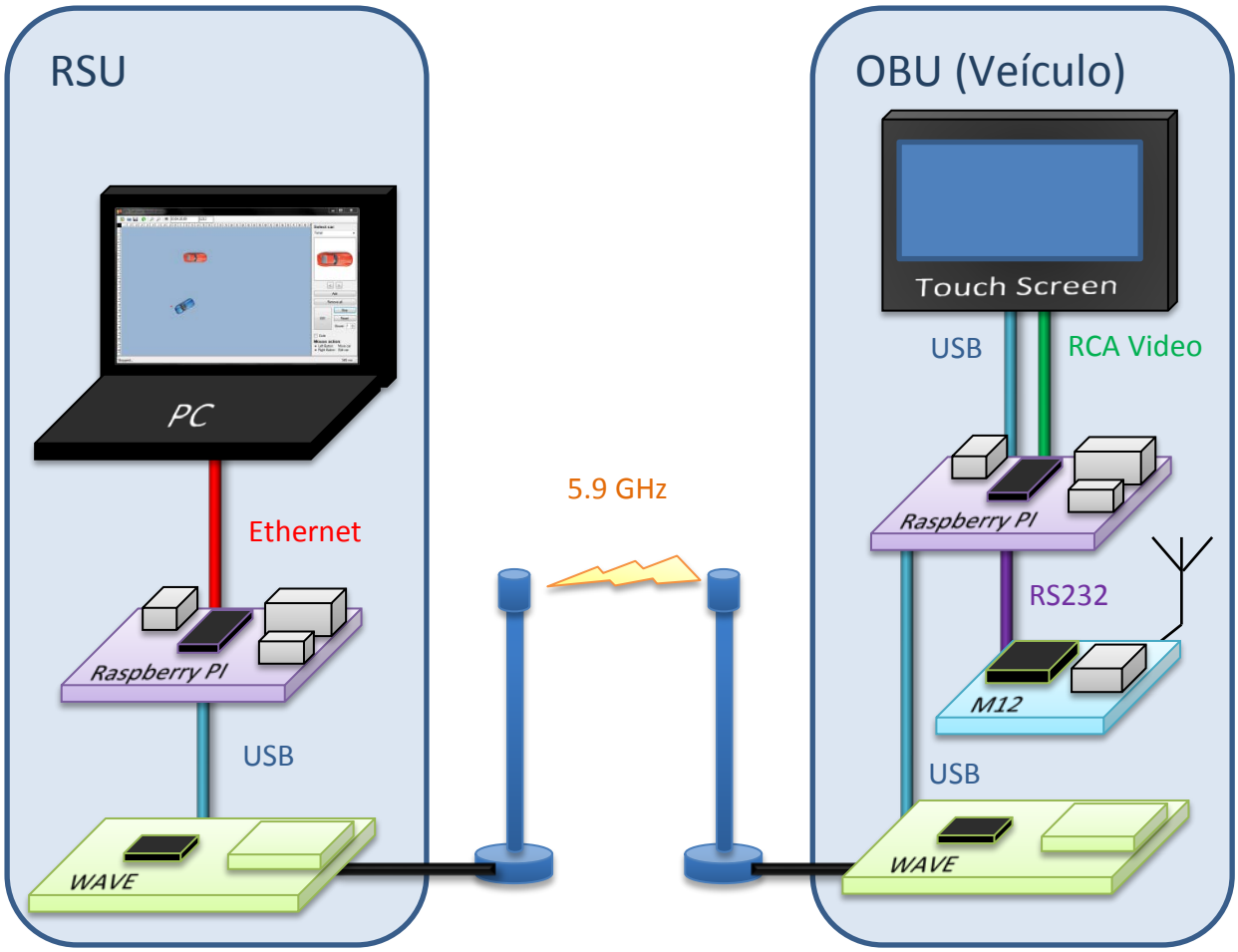


Figura 3.9 – Ilustração do ambiente de testes.

3.3 Implementação do *Software*

O *software* encarregue de todo o procedimento dos algoritmos estudados e desenvolvidos está escrito em linguagem de programação C. A aplicação principal é nomeada de RoadView, que permitirá ao condutor a visualização da localização, velocidade e azimute dos veículos na vizinhança, a chamada realidade aumentada, pois permite ao condutor "ver" para além da sua capacidade visual, e está encarregue de alertar o condutor de possíveis riscos de colisão. Conta com recursos fornecidos pelo sistema operativo Arch Linux, para criação de tarefas, utilização de controladores (da porta série e da entrada tátil) e a utilização direta da memória de vídeo, *framebuffer*. Pode-se observar um diagrama de blocos do *software* na Figura 3.10.

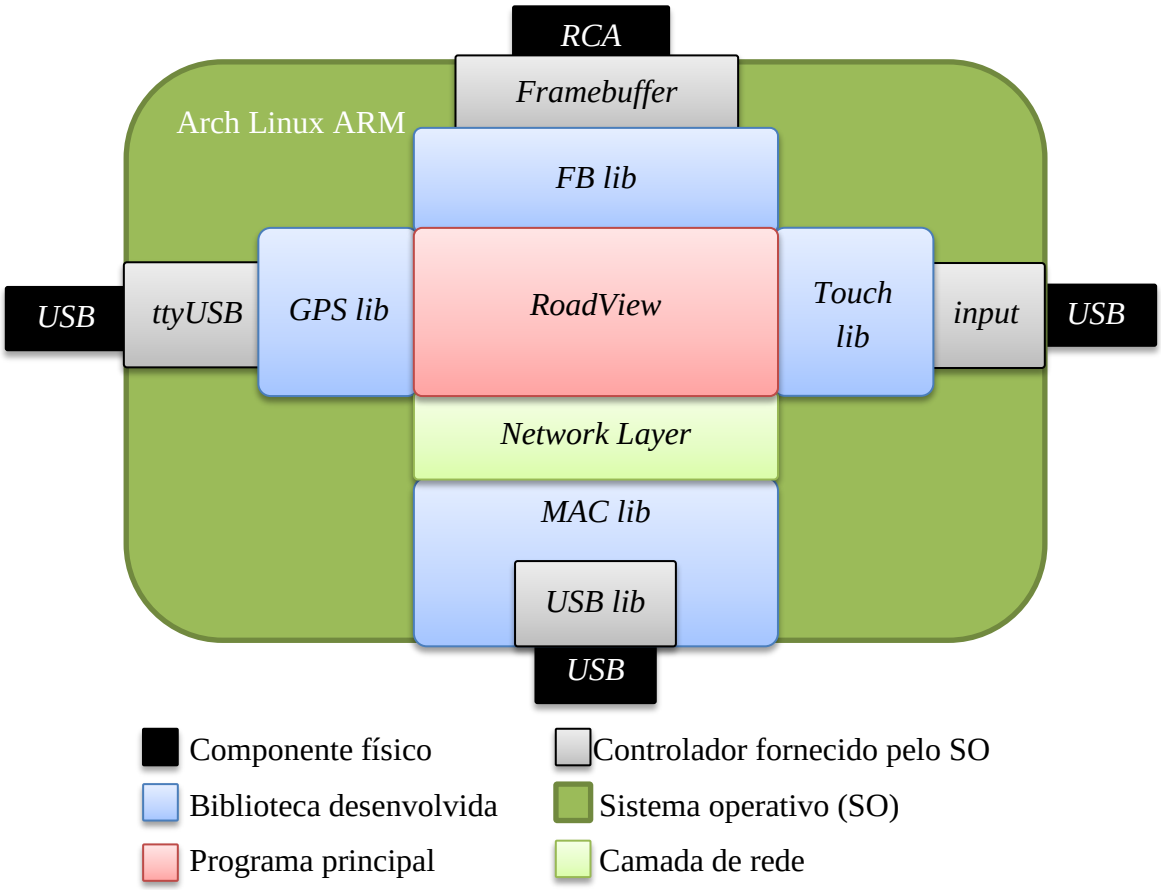


Figura 3.10 – Diagrama de blocos do *Software*.

3.3.1 *Software de simulação*

Para um desenvolvimento do projeto mais versátil e acelerado, foi desenvolvido um simulador que permite efetuar testes aos diversos algoritmos de segurança, duma forma

independente do *hardware*, através de trajetórias teóricas ou de coordenadas reais. Também permite a simulação de vários OBUs, utilizando a porta *ethernet* para enviar os dados simulados para o RSU que na realidade vão ser recebidos via RF, substituindo a presença física dos veículos.

A linguagem de programação utilizada é C#, pelo motivo de ser uma linguagem de alto nível onde facilmente se cria ambientes gráficos em Windows, e é portável para C, sendo esta a linguagem de programação em que os algoritmos de segurança estão implementados.

O *software* permite a inserção de simulação de veículos num painel de testes e atribuir-lhes valores dos diversos campos (posição, velocidade, azimute, trajetória (teóricas ou reais) e selecionar os algoritmos que irá implementar), e permite visualizar a execução das trajetórias dos veículos e a execução dos algoritmos.

A interface gráfica do programa tem o aspeto idêntico ao exemplo da Figura 3.11, com a seguinte legenda:

1. Carregar cenário real: permite importar dum ficheiro trajetórias de veículos com coordenadas geográficas sobre o sistema WGS84. O conteúdo do ficheiro pode conter as opções:

   i – indicar o nome da imagem que irá ser apresentada como fundo (opcional);

   z – escala, indicar o número de *pixels* correspondentes a um metro;

   o – centro, indicar a coordenada que irá corresponder a coordenada (0,0) do painel de testes;

   v*x* – veículo *x*, indicar o número e as várias coordenadas da trajetória do veículo, corresponderá a intervalos de um segundo entre cada coordenada.

   Exemplo de ficheiro:

```
i
track1.png
z
10,2
o
38o50'45.97"N     9o05'40.32"O      0o
v1
4
38o50'45.94"N     9o05'39.09"O      190o
```

```
38o50'45.75"N     9o05'39.18"O      200o
38o50'45.52"N     9o05'39.32"O      190o
38o50'45.29"N     9o05'39.43"O      190o
v2
4
38o50'44.32"N     9o05'37.84"O      280o
38o50'44.32"N     9o05'37.84"O      280o
38o50'44.32"N     9o05'37.84"O      280o
38o50'44.37"N     9o05'38.22"O      282º
```

2. Carregar cenário virtual: permite importar um ficheiro com o estado do guardado anteriormente;
3. Guardar cenário virtual: permite guardar o estado em ficheiro, incluí a posição dos veículos e todos os seus parâmetros;
4. Refrescar: redesenha a interface gráfica;
5. Zoom: aumenta/diminuí o zoom em 10%;
6. Iniciar ligação com o RSU: permite estabelecer uma ligação com o RSU, utilizando *sockets*, tendo em conta o endereço IP e o porto especificado, com o fim de enviar a cada segundo as posições dos veículos no painel de testes;
7. Endereço IP do RSU;
8. Número do porto de ligação ao RSU;
9. Nome do veículo: permite a seleção de um veículo num conjunto, para a inserção no painel de testes, diferindo entre eles apenas o aspeto;
10. Aspeto do veículo selecionado;
11. Alterar o azimute do veículo a inserir;
12. Botão de inserção do veículo: após a ação deste botão seleciona-se com o rato a posição do veículo pretendida no painel de testes;
13. Botão de remoção de todos os veículos presentes no painel de testes;
14. Botão de iniciar: cada veículo inicia a sua trajetória e o processamento dos algoritmos que implementam;
15. Botão de pausa: pausar o teste, podendo retomar premindo o botão de iniciar. Enquanto em pausa, ao colocar o rato sobre os veículos, é possível visualizar os seus parâmetros correntes, posição, velocidade e direção;
16. Botão de reiniciar: coloca todos os veículos na sua posição inicial;
17. Atrasar: permite a execução do teste lentamente, o tempo do teste irá ser multiplicado pelo valor presente neste campo;

18. Ciclo: permite ativar a funcionalidade de ciclo, no decorrer dum teste os veículos recomeçam a trajetória assim que a acabam;
19. Tempo: tempo real da execução do teste em milissegundos;
20. Painel de testes.

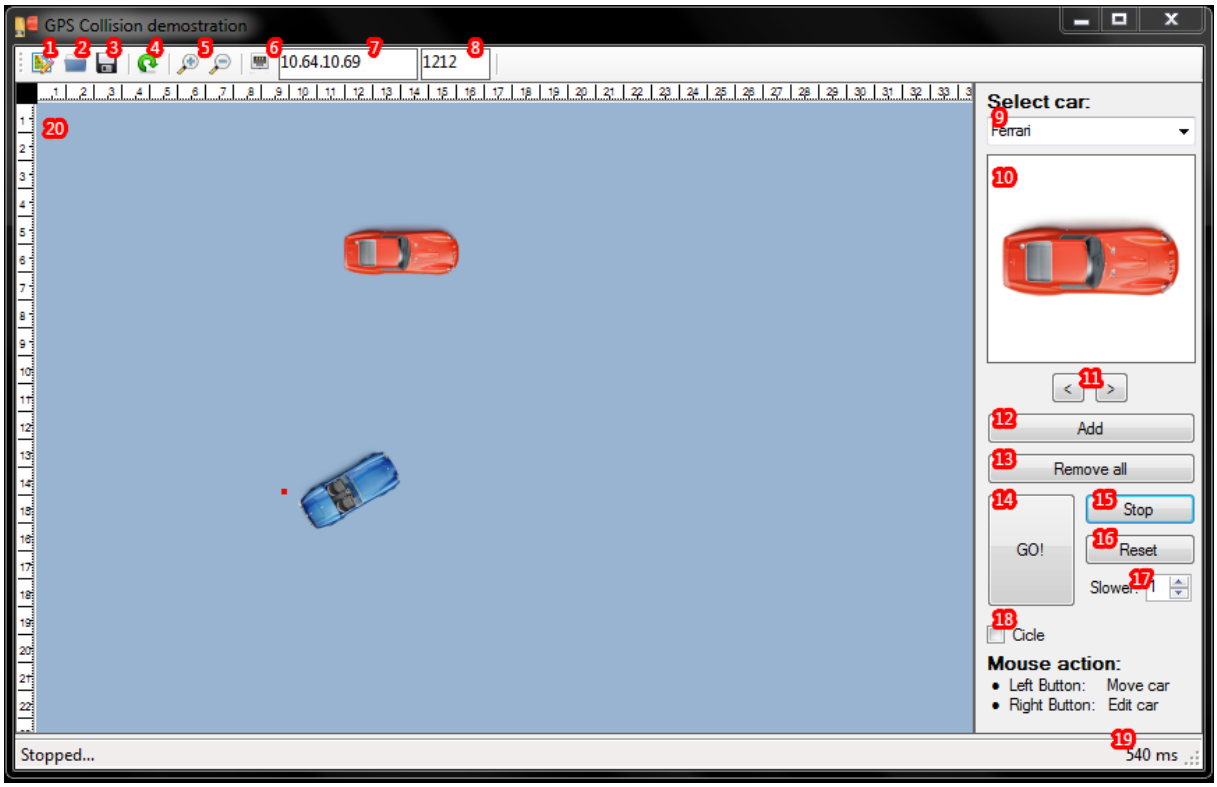


Figura 3.11 – Aspeto da interface gráfica do simulador.

Depois de inserido um veículo é possível alterar a sua posição clicando com o botão do lado esquerdo do rato sobre o próprio, ou alterar os vários parâmetros clicando sobre ele com o botão do lado direito do rato, que irá abrir a janela de propriedades.

A janela de propriedades dos veículos tem o aspeto idêntico ao exemplo da Figura 3.12. Nesta janela é possível a alteração dos seguintes parâmetros:

1. Velocidade, em km/h;
2. Posição, em centímetros;
3. Azimute, em graus;
4. Ativar a função "*It's me*", que corresponde a um valor booleano que será enviado para o RSU, para a situação de testes aos algoritmos sendo o próprio carro também ele simulado;

5. Trajetória, permite adicionar/remover movimentos, reta, curva à esquerda ou curva à direita;
6. Ativar e selecionar o algoritmo unidimensional que o veículo irá executar;
7. Ativar e selecionar o algoritmo bidimensional que o veículo irá executar;
8. Funcionalidade de travagem, existem dois modos de travagem possíveis:
    a. *Since warning*: o veículo diminui a velocidade, simbolizando uma travagem, até parar completamente, assim que receba o primeiro aviso dum dos algoritmos em execução;
    b. *When warning*: o veículo diminui a velocidade sempre que receba um aviso dum dos algoritmos em execução.

A travagem é visualizada no painel de testes por um quadrado encarnado junto ao veículo, como se pode verificar no exemplo do veículo azul da Figura 3.11.

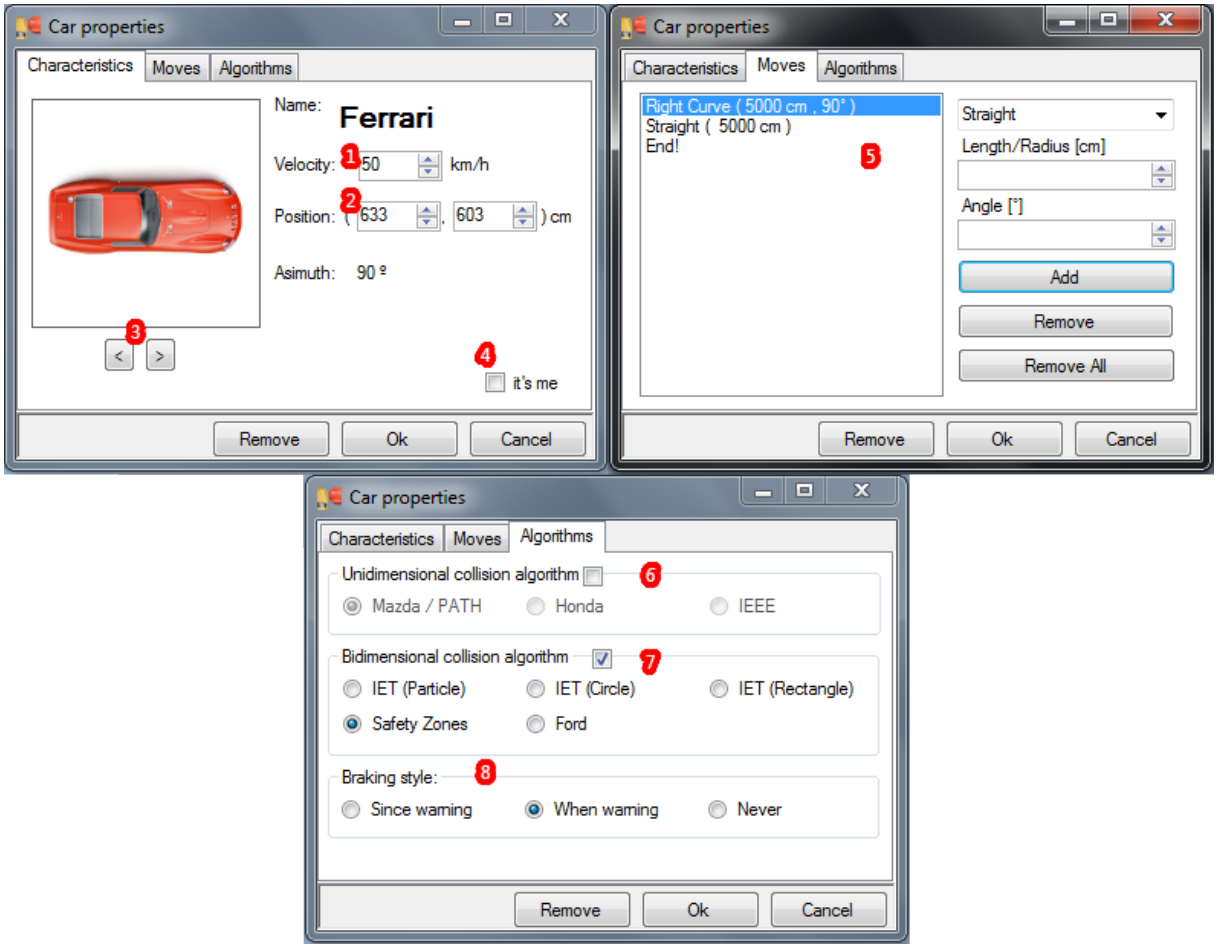


Figura 3.12 – Aspeto das interfaces gráficas das janelas de propriedades dos veículos.

3.3.2 ***Software* implementado**

Antes de proceder à realização da aplicação principal é necessária a criação de bibliotecas e *drivers*, nomeadamente, os controladores da porta série para a receção dos dados GPS, da porta USB para a receção dos dados de entrada do ecrã tátil e do *framebuffer* para a criação duma interface gráfica, destacados nos anexos de A a E.

Grande parte das bibliotecas criadas poderiam ser substituídas por recursos disponibilizados no sistema operativo, como o X11, mas a criação das mesmas foi feita anteriormente sobre o módulo TX27 que não possuía tais recursos, no entanto, após a passagem para o módulo Raspberry Pi decidiu-se a continuação das bibliotecas já desenvolvidas, aumentando assim a portabilidade.

- *GPS*

A interface de utilização do controlador GPS é constituída somente pela função de inicialização com a seguinte assinatura:

```
void GPS_init(char * serial_name, void (*func)(GPSCoor*));
```

onde `serial_name` é a *string* relativa ao nome do descritor da porta série e o segundo parâmetro é o ponteiro para a função que será executada sempre que forem recebidos dados GPS, que tem como parâmetro de saída o ponteiro para a estrutura onde deveram ser guardados os dados recebidos, com a seguinte definição:

```
typedef struct _coor{
        int lat_deg;
        double lat_min;
        char lat_dir;

        int lon_deg;
        double lon_min;
        char lon_dir;

        int hour;
        int min;
        double sec;

        double speed; // km/h
        double asimuth; //degree

        bool valid;
```

```
}GPSCoor;
```

O fluxo de inicialização começa por criar e abrir uma porta série através do seu controlador, anexo A, de seguida, é enviado o comando GPRMC, *Recommended Minimum Specific GPS/Transit Data*, sobre o protocolo NMEA 0183, necessário para o início de receção de dados com um intervalo de um segundo (mínimo), trata-se do envio da *string* `"$PMOTG,RMC,0001\r\n"`, por último é iniciada a tarefa encarregue de receber os dados.

O comando da receção dos dados tem a seguinte estrutura:

`$GPRMC,hhmmss.ss,a,ddmm.mmmm,n,dddmm.mmmm,w,z.z,y.y,ddmmyy,d.d,v\r\n"`,

tendo como dados principais os seguintes:

- `hhmmss.ss` – hora relativa às informações recebidas;
- `a` – validade, 'A' indica que os dados recebidos são válidos e 'V' inválidos;
- `ddmm.mmmm,n` – latitude, sendo `dd` os graus, de 0 a 90º, `mm.mmmm` os minutos, de 0 a 59.9999 e `n` a direção, podendo assumir as opções N ou S, norte ou sul;
- `dddmm.mmmm,w` – longitude, sendo `ddd` os graus, de 0 a 180º, `mm.mmmm` os minutos, de 0 a 59.9999 e `w` a direção, podendo assumir as opções E ou W, este ou oeste;
- `z.z` – velocidade em km/h;
- `y.y` – azimute, de 0 a 359.9º,

a tarefa de receção está encarregue de decifrar apenas este comando e executar a função externa referida na inicialização indicando o ponteiro para a estrutura com os dados decifrados.

Este controlador possui uma função *main* oculta, que permite um leque maior de manipulação do sistema de GPS, utilizada para *debug* e para a criação de ficheiros com trajetos.

- *RoadView*

A aplicação principal, como já foi referido na introdução deste capítulo, tem o nome de *RoadView* e está encarregue de alertar o condutor de riscos de colisão, também tem como objetivo a introdução da chamada realidade aumentada, isto é, permitir ao condutor visualizar a posição, velocidade e direção de veículos com que não tem contato visual. Anteriormente à sua execução é efetuada uma fase de seleção de recursos, definida no anexo F.

Na execução de *RoadView* é apresentado um painel correspondente a uma determinada área ao redor do veículo que irá conter vetores relativos aos veículos que o rodeiam, onde o ângulo do vetor corresponde à sua direção e a amplitude e cor em função da sua velocidade, apresentará ícones de aviso perante riscos de colisão. Foi desenvolvido de forma genérica, sendo indiferente a fonte dos dados do veículo e dos na vizinhança, fazendo sentido as combinações presentes na Tabela 3.3.

Tabela 3.3 – Principais combinações das fontes dos dados dos veículos.

Fonte dos dados		**Cenário**
Meu veículo	**Restantes veículos**	
Ficheiro	Ficheiro	Testar *software*.
Ethernet	Ethernet	Testar *software*.
-	Ethernet	RSU.
GPS	Ficheiro	OBU no 1º cenário de teste.
GPS	RF	OBU com dados reais no terreno

A API desenvolvida de forma a manter uma interface genérica está destacada no anexo G. O programa é iniciado e terminado através das funções **`RoadView_start`** e **`RoadView_stop`** respetivamente, os dados do veículo e dos restantes que o rodeiam, que constam numa lista global, são atualizados pelas funções **`RoadView_update_my`** e **`RoadView_update`**, ou em alternativa pelas funções **`RoadView_update_myCoor`** e **`RoadView_update_Coor`** onde `Coor` é o objeto definido no anexo H, elimina-se um veículo da lista e liberta-se a memória alocada para tal com a função **`RoadView_delete`**, através das funções **`RoadView_ZoomIn`** e **`RoadView_ZoomOut`** é possível alterar a distância visível em 10%, redesenha-se o painel com **`RoadView_redraw`**. **`RoadView_caution`** é a função encarregue de executar os algoritmos de segurança com base no veículo passado como parâmetro, que é chamada sempre que forem atualizados os dados dos veículos na vizinhança, com **`RoadView_update`** ou **`RoadView_updateCoor`**.

A inicialização do programa, **`RoadView_start`**, recebe indicação de calibração da entrada tátil no arranque, `input_cal`, e a *string* relativa ao seu descritor para executar a sua inicialização com **`input_init`**, anexo B, em função de `input_cal` é executado **`input_calibration`**, de seguida é efetuada a chamada a **`widget_init`**, anexo E, a

inicialização do vídeo é feita externamente a *RoadView* para permitir a sua omissão. De seguida, é desenhado o painel de visualização, que se trata apenas dum retângulo branco com a representação à escala do veículo no seu interior, e adiciona-se dois botões de zoom, utilizando **`w_button_add`** onde os seus eventos correspondem as funções **`RoadView_ZoomIn`** e **`RoadView_ZoomOut`**. São criadas desde já, e guardadas em variáveis globais, as estruturas `BITMAP` relativas aos ícones de aviso de modo a acelerar o processo quando for necessária a sua visualização. A interface gráfica tem o *layout* idêntico ao da Figura 3.13. Por último, é criada a tarefa e o seu relativo *mutex* que está encarregue de eliminar os veículos antigos, dos quais não foram recebidos dados durante um determinado tempo, este é contado por blocos, a verificação em toda a lista é efetuada a cada 30 ms, que corresponde exatamente à dimensão dum bloco, o veículo é eliminado ao fim de 150 blocos, ou seja, 4.5 s, contagem guardada no campo `count` da estrutura `Vehicle`.

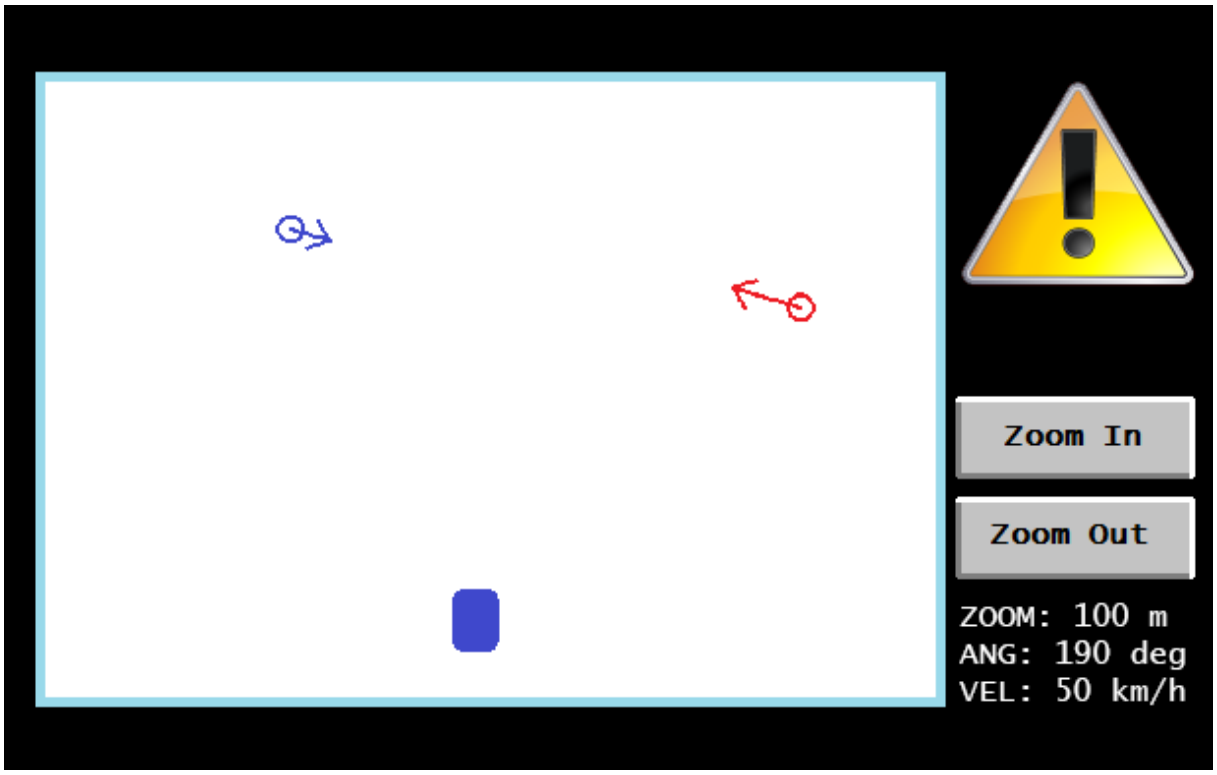


Figura 3.13 – Exemplo do *layout* de *RoadView*.

Ao atualizar-se a posição atual do veículo, com **`RoadView_update_my`** ou **`RoadView_update_myCoor`**, após a atualização dos seus dados que constam numa variável global do tipo `Position`, todos os restantes veículos presentes na lista são redesenhados, incluindo a representação do próprio veículo, pois por vezes é sobreposta com os vetores de outros veículos. Na atualização dum veículo, com **`RoadView_update`** ou

RoadView_update_Coor, é feita a manipulação da lista `road` e a lista da cobra desse veículo, de seguida é redesenhado no painel o seu vetor e efetuada a chamada a função **RoadView_caution**.

- *RoadView: algoritmos de segurança rodoviária*

A execução do processo de filtragem e dos algoritmos implementados é feita com a chamada à função **RoadView_caution**, passando como entrada o ponteiro para a estrutura do veículo em análise.

A primeira filtragem são os veículos em sentido oposto, como referido anteriormente.

De seguida é executado o algoritmo de curva, a sua implementação consta na API presente no anexo I. Começa pela chamada à função **getCurve**, que procura a existência de curvas na cobra dada pelo ponteiro `p` passado como parâmetro, relativo ao veículo em análise, fluxo especificado no próximo parágrafo, e guarda na estrutura da curva apontada por `c` os seus dados. Caso o algoritmo detete uma curva esta é adicionada a uma lista global com a execução da função **addCurve**. Na execução deste fluxo é construída uma lista global que guarda os dados de todas as curvas detetadas pelas cobras dos veículos na vizinhança. De seguida, deteta-se a presença do veículo no interior duma área de aviso com a função **checkCurves**, que retorna o ponteiro para a estrutura relativa à curva que se aproxima.

A execução da deteção da curva, presente na função **getCurve**, segue o fluxo presente na Figura 3.14, que corresponde ao especificado em 3.1.5, onde *pos* é o ponteiro para uma determinada posição da cobra `p`, que consiste numa instância da estrutura `Position`. O algoritmo começa por detetar uma variação nos valores de azimute da cobra, estado 1, de seguida deteta até que ponto ocorreu essa variação, estado 2, o valor comparado para uma variação ser considerável está definido na *macro* CURVE_ANG_DIF_DET, que, após várias experimentações práticas, possuí o valor de 10º. Por último, estado 3, efetua os cálculos do algoritmo curva, raio de curvatura, velocidade máxima e o raio relativo à área de aviso, correspondentes às equações **(3.6)**, **(3.8)** e **(3.9)** respetivamente, de seguida introduz os valores resultantes na estrutura relativa a essa curva, `c`. Todo este fluxo é interrompido caso *pos* atinja a última posição da cobra.

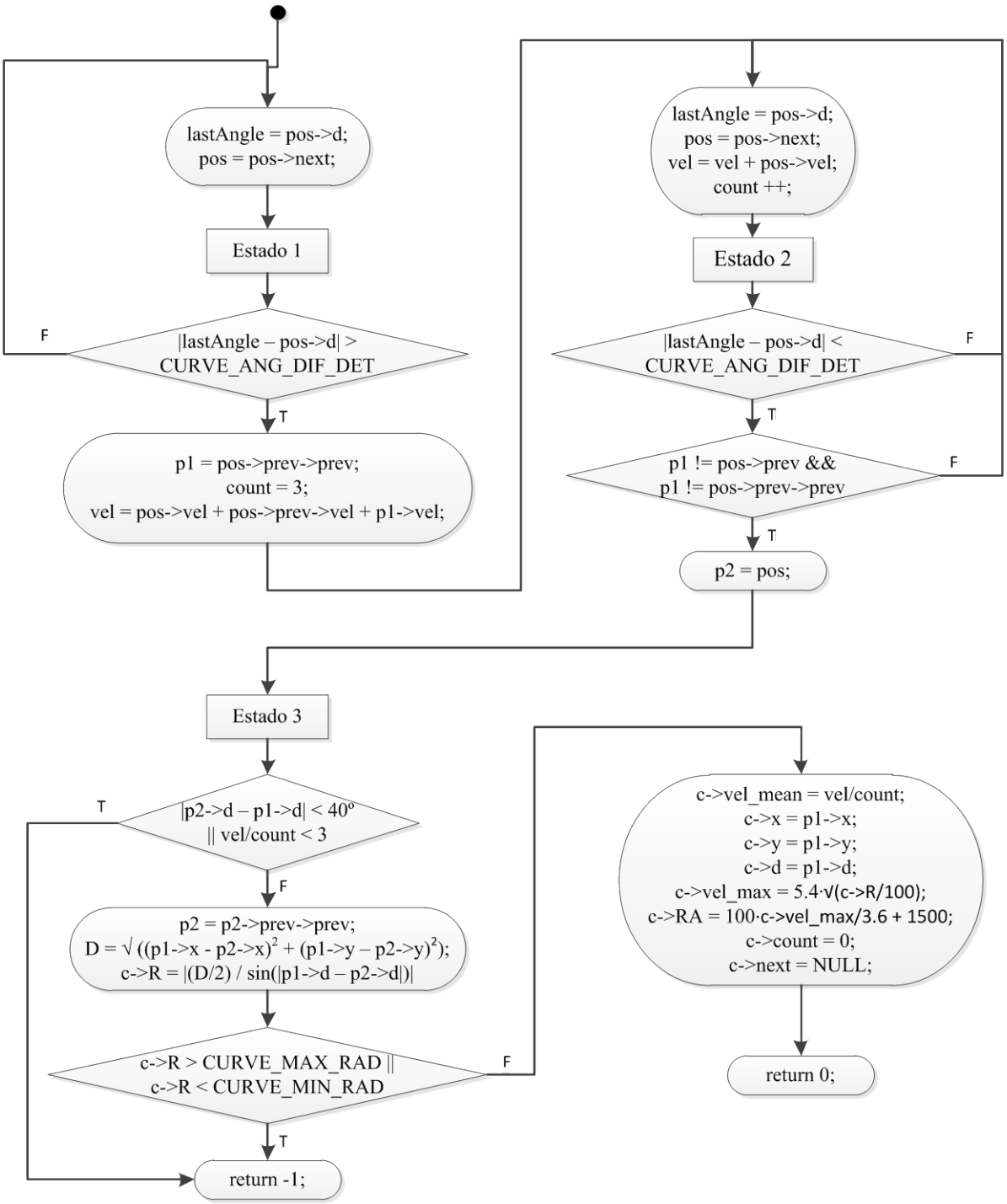


Figura 3.14 – Fluxograma da deteção de curva.

Após detetar-se a presença do veículo no interior duma área de aviso com a função `checkCurves`, aplica-se as definições do algoritmo curva:

- caso o raio de curvatura da curva seja superior ao valor definido na *macro* CURVE_MIN_RAD, que, após experimentações práticas e simuladas presentes no

tópico 4.1.3, assumiu-se um valor de 15 m, estamos perante uma curva e não um cruzamento, e se o veículo possuir uma velocidade instantânea superior à velocidade máxima calculada para essa curva o condutor é alertado com o aviso dessa velocidade;

- caso o raio de curvatura seja inferior a CURVE_MIN_RAD, ou caso o veículo não se encontre no interior duma área de aviso, assume-se estar perante um cruzamento e é efetuado o fluxo do algoritmo de previsão de colisão.

O algoritmo de previsão de colisão implementado é o *Safety Zones*, as razões de seleção juntificam-se nos comentários aos resultados simulados da comparação entre os algoritmos nos tópicos 4.1.1 e 4.1.2. Introduziu-se algumas alterações ao algoritmo, nomeadamente a distância crítica, que agora é dada pelo algoritmo da PATH, justificação também presente nos comentários aos resultados práticos, e a introdução duma distância de compensação à frequência de amostragem, isto é, adiciona-se à distância critica a previsão da distância até se obter uma nova coordenada, pois, com um período de amostragem de 1 segundo, o veículo pode percorrer alguns metros e algum desses instantes estar em risco de colisão e não ser alertado, desta forma elimina-se essa situação.

- *App: receção dos dados dos veículos através de comunicação ethernet*

A receção dos dados dos veículos através da comunicação *ethernet* é ativa pelo comando –p, indicando o número do porto de escuta. Caso o comando seja detetado a função *main* inicia uma tarefa que executa a função *startEthernetConnection*, esta começa por criar um *TCP socket server* no porto indicado, escreve na consola e no ecrã, caso conectado, o seu endereço IP e fica a escuta até receber uma ligação. Assim é recebida uma ligação válida executa a função *receiveData*, encarregue de receber os dados com a formatação presente na Tabela 3.4 e atualiza *RoadView* através das funções `RoadView_update` e `RoadView_update_my`, assim consecutivamente até ordem de terminar o programa.

Tabela 3.4 – Formatação da receção dos dados via *Ethernet*.

Nº de *bytes*	Nome	Descrição
2	*vehicle_id*	0 – Terminar programa, 1 – Meu veículo, >1 – ID do veículo.
4	*x_cm*	Inteiro que indica a coordenada X em centímetros.
4	*y_cm*	Inteiro que indica a coordenada Y em centímetros.

4	*vel*	Inteiro que indica a velocidade em km/h.
4	*angle*	Inteiro que indica a direção em graus.

- *App: receção dos dados dos veículos através de ficheiro*

A receção dos dados dos veículos através de ficheiro tem um procedimento idêntico à receção por *ethernet*, quando detetada a presença do comando –f, com a indicação do respetivo ficheiro, é criada uma tarefa que executa a função *startFileRead*, que efetua a leitura do ficheiro que possui o mesmo tipo de formatação que os ficheiros de coordenadas geográficas utilizados no *software* de simulação. Esta função preenche um *array* global de listas de estruturas `Coor`, cada posição do *array* corresponde a um veículo e a lista de coordenadas à sua trajetória, o campo *o* do ficheiro (origem, que no *software* de simulação corresponde à coordenada (0,0)) será a posição do nosso veículo caso a receção de localização por GPS não seja ativada com o comando –g.

De seguida é executada a função *moveCars*, que atualiza a posição de todos os veículos, presentes no *array* global de listas de coordenadas, com a chamada às funções **`RoadView_update`** e **`RoadView_update_my`** em intervalos de um segundo. A posição do nosso veículo pode ser alterada manualmente indicando um número com o teclado, que corresponderá ao número do ID do veículo presente no *array* que pretendemos que seja considerado o nosso veículo, utilizado para testes. É efetuado este procedimento de forma cíclica até ordem de terminação do programa.

- *App: receção dos dados do veículo através do módulo GPS*

A receção da localização do nosso veículo através do módulo GPS é executada com a indicação do comando –g, referindo a respetiva porta série. A função *main* quando deteta a presença deste comando executa a chamada à função de inicialização da biblioteca GPS, **`GPS_init`**, indicando o nome da porta série e a função de atualização da localização do nosso veículo, **`RoadView_update_myCoor`**.

- *App: receção/envio através de RF*

A receção e envio dos dados através de RF é ativada com o comando –t, indicando manualmente um número de identificação para o dispositivo em questão, utilizado como seu endereço MAC. A potência de transmissão pode ser alterada através do comando –P, indicando um número entre 0 e 63, correspondente a diferentes valores de potência de transmissão, no qual ainda não foram efetuados testes para a sua correspondência com valores de unidades de potência. A modulação utilizada é 64-QAM, atingindo uma velocidade 24 Mbps.

As camadas inferiores são inicializadas com a execução da função *ma_init*, disponibilizada na biblioteca fornecida, indicando uma estrutura do tipo *ma_dev*, e os ponteiros para as funções que serão executadas na receção de dados. As funções são:

- *indication_cb*: receção duma trama recebida por RF;
- *status_indication_cb*: indicação do estado do envio duma trama MA-UNITDATA;
- *xstatus_indication_cb*: indicação do estado do envio duma trama MA-UNITDATAX.

A definição do procedimento destas funções está descrita na norma IEEE 802.11p [1].

No interior da função *indication_cb* está presente o código que efetua a validação dos dados recebidos, estruturas do tipo `Coor`, e atualiza *RoadView* com a chamada à função e `RoadView_update_Coor`.

Após a inicialização da camada inferior é criada uma tarefa encarregue de enviar a posição corrente do veículo, no caso de OBU. Os dados são enviados, estruturas do tipo `Coor`, com a execução da função *ma_unitdatax_request*, disponibilizada na biblioteca fornecida.

Foi desenvolvida a função *RF_send*, que recebe como parâmetros de entrada um número de identificação e um ponteiro para uma estrutura do tipo `Coor`, e envia para a camada inferior o seu conteúdo indicado um endereço MAC de origem relativo ao número de identificação. Esta função é utilizada para a simulação do envio dos dados de vários veículos apenas com um dispositivo.

4 Resultados

4.1 Simulação

Através de simulação, utilizando o MATLAB, são comparados os diversos algoritmos estudados ao longo desta dissertação, baseando-se nos três cenários de referência: aproximação súbita ao veículo procedente; cruzamento ou entroncamento e curva.

4.1.1 Aproximação súbita ao veículo procedente

Para ser possível a comparação quantitativa entre os vários algoritmos unidimensionais, estudados no subtópico 2.2, é necessário definir alguns parâmetros, como a desaceleração máxima de um veículo e os tempos de atraso.

De acordo com testes da WABCO, *Westinghouse Air Brake Company*, a desaceleração máxima corresponde a aceleração da gravidade, $9{,}81\ m/s^2$, multiplicada por um coeficiente de atrito *k*, dado pelos valores da Tabela 4.1, e dependente da percentagem da força de travagem. Definindo para os testes que se seguem o valor de $6\ m/s^2$. [32]

Tabela 4.1 – Coeficiente de atrito *k* (valores extraídos de [32]) .

Superfície da estrada	Seco	Molhado
Granito	0.7	0.6
Asfalto	0.6	0.5
Paralelos	0.55	0.3
Neve	0.2	0.1
Gelo	0.1	0.01

Com base nos fatores humanos [33], obtém-se os valores de tempo de reação dum condutor presentes no gráfico da Figura 4.1, definindo para a realização dos testes um tempo de reação de 1.3 s.

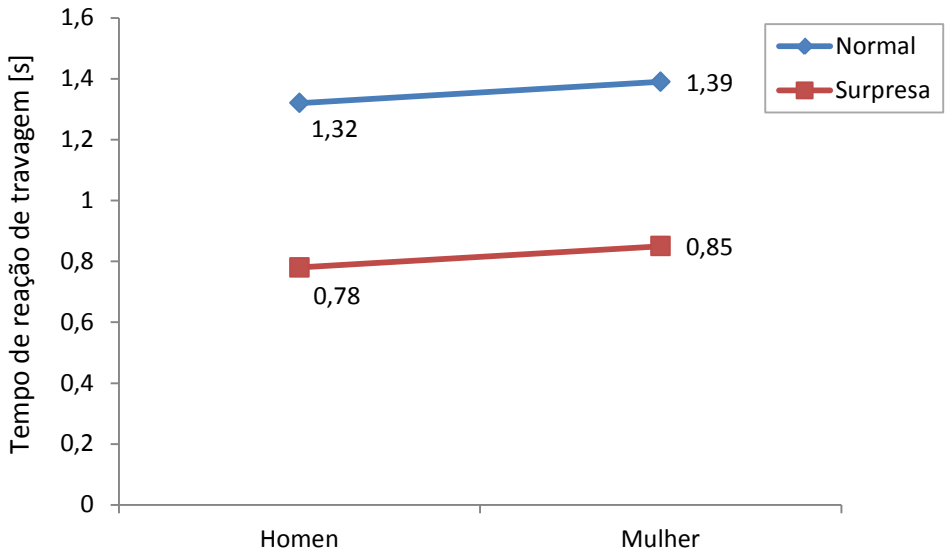


Figura 4.1 - Tempo de reação de travagem (valores extraídos de [33]).

Pode-se observar os resultados da comparação dos vários algoritmos unidimensionais estudados, a variação da distância de aviso d_w em metros em função da velocidade do veículo procedente, para valores de velocidades de 30, 50, 70 e 100 km/h na Figura 4.2, Figura 4.3, Figura 4.4 e Figura 4.5 respetivamente.

Como esperado, os resultados dos algoritmos da Mazda e da PATH são iguais, pois foram utilizadas desacelerações idênticas para ambos os veículos e um tempo de atraso do sistema nulo.

O resultado do algoritmo do IEEE é uma constante, pois não depende da velocidade do veículo procedente, o que será o algoritmo menos eficaz, pois caso as velocidades de ambos os veículos sejam iguais, não existe aproximação, seria acionado o aviso ao condutor à mesma distância que se a aproximação fosse repentina.

Analisando os resultados do algoritmo da Honda obtém-se uma reta, que, por um lado, apresenta uma boa solução, pois, quanto maior for a diferença de velocidades mais rápida é a aproximação, logo, maior deve ser a distância de aviso. Por outro lado, apresenta um aspeto negativo, o facto de resultar uma distância de segurança reduzida para velocidades idênticas, o mesmo para uma situação de diferença de velocidades reduzida ou quase nula, onde deveria ser acionado um aviso ao condutor que poderá não estar a aperceber-se da aproximação lenta.

Contudo, os algoritmos que apresentam a melhor solução são os da Mazda e da PATH, garantem uma distancia de aviso adequada para uma diferença de velocidades alta, e uma margem de segurança essencial para diferenças de velocidades reduzidas.

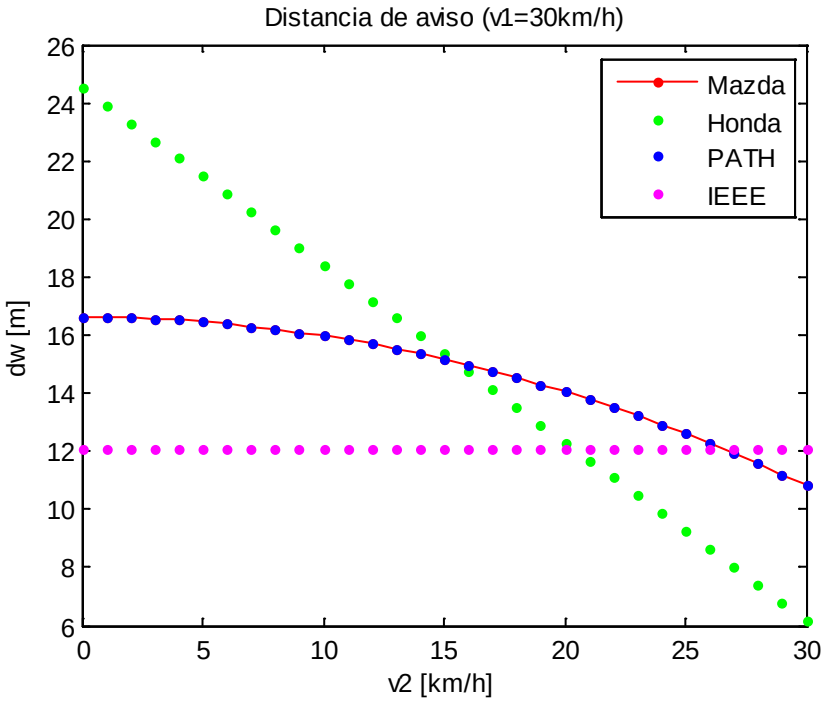


Figura 4.2 - Distância de aviso para v=30 km/h.

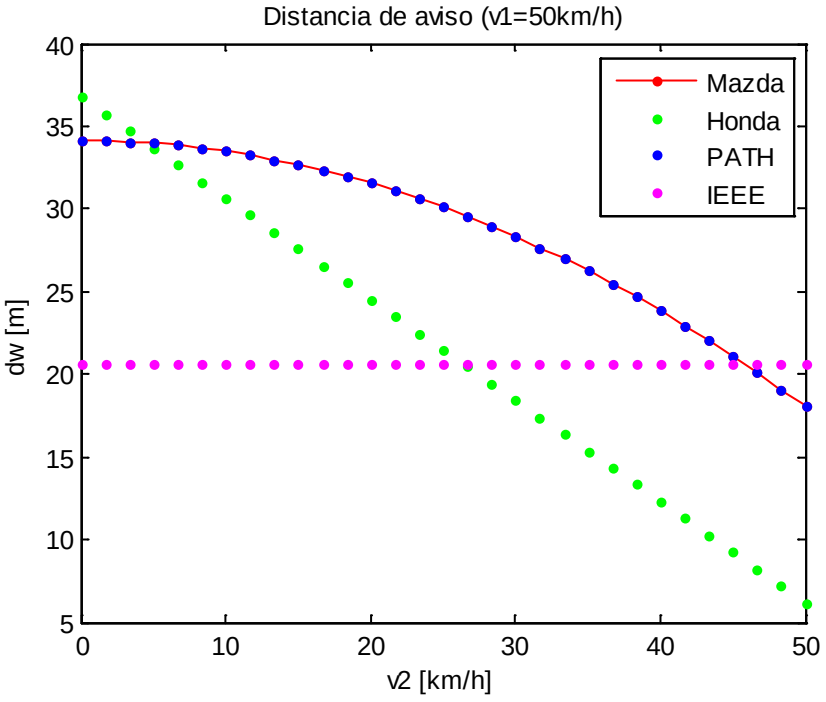


Figura 4.3 - Distância de aviso para v=50 km/h.

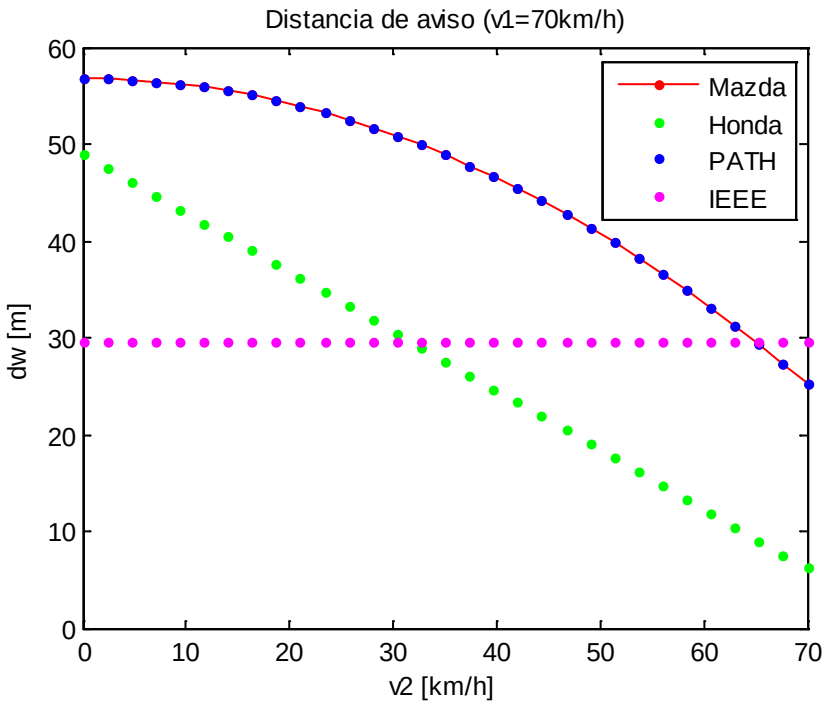


Figura 4.4 - Distância de aviso para v=70 km/h.

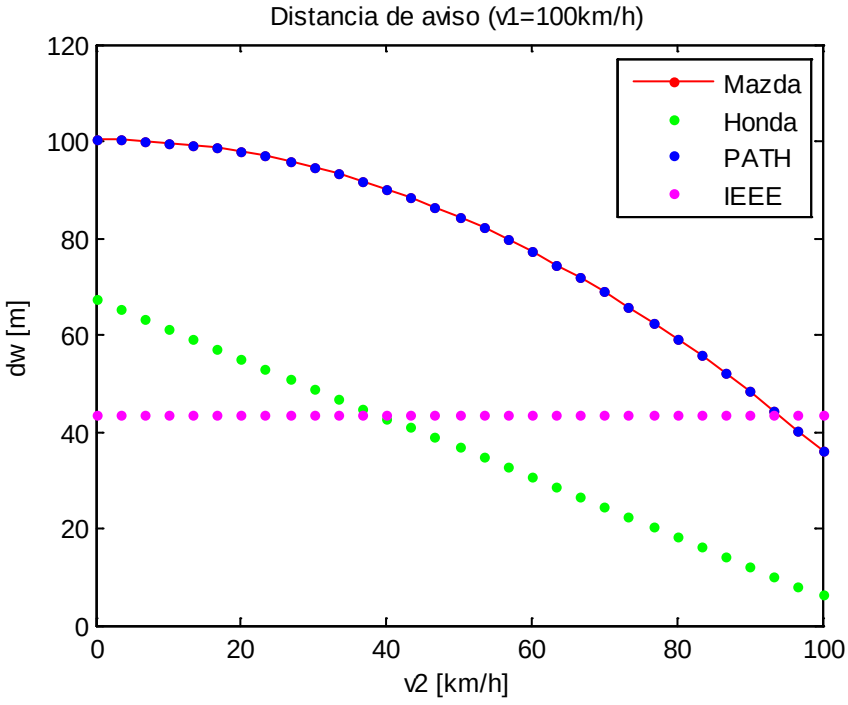


Figura 4.5 - Distância de aviso para v=100 km/h.

4.1.2 Cruzamento e entroncamento

Os algoritmos estudados para um cenário de cruzamento ou entroncamento constam no subtópico 2.3, onde alguns deles são testados e comentados neste subtópico.

Para um teste inicial ao algoritmo da IET, subtópico 2.3.1, utilizou-se como recurso matemático o MATLAB, para uma melhor perceção quantitativa, em dois cenários distintos, cruzamento/entroncamento e junção de vias de trânsito.

Na Figura 4.6 obtém-se o resultado do algoritmo IET modelo circunferência para uma situação de cruzamento, numa situação de dois veículos em circulação a uma velocidade de 50 km/h, resultando um valor de TTC, tempo até à colisão, de 3.397 s, assumindo o raio da circunferência de 4 m. Esta é uma das situações que poderia assentar sobre um caso de falso aviso, pois poder-se-ia estar perante uma curva e não um cruzamento.

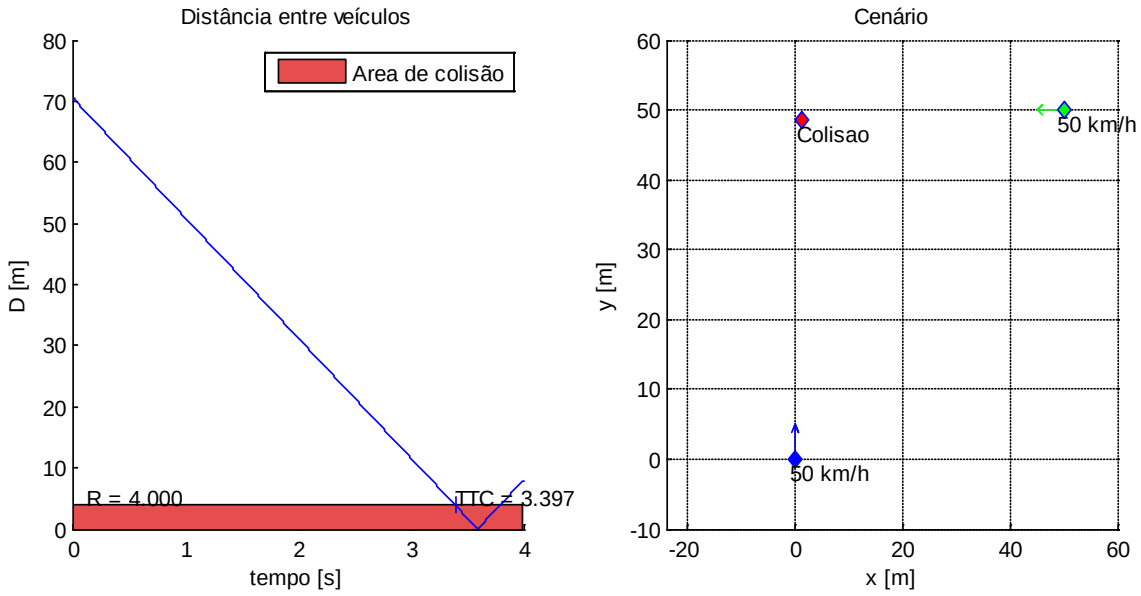


Figura 4.6 – Resultado do algoritmo IET num cenário de cruzamento.

Pode-se observar um exemplo idêntico ao anterior, alterando o ângulo de cruzamento, na Figura 4.7, onde se continua a confirmar o correto funcionamento do algoritmo. Podendo destacar, com base no resultado presente no gráfico da distância entre veículos, a desvantagem do modelo partícula, que só assume risco de colisão se a distância entre veículos atingir o valor nulo, que não acontece nesta situação o que resultaria num aviso essencial omitido.

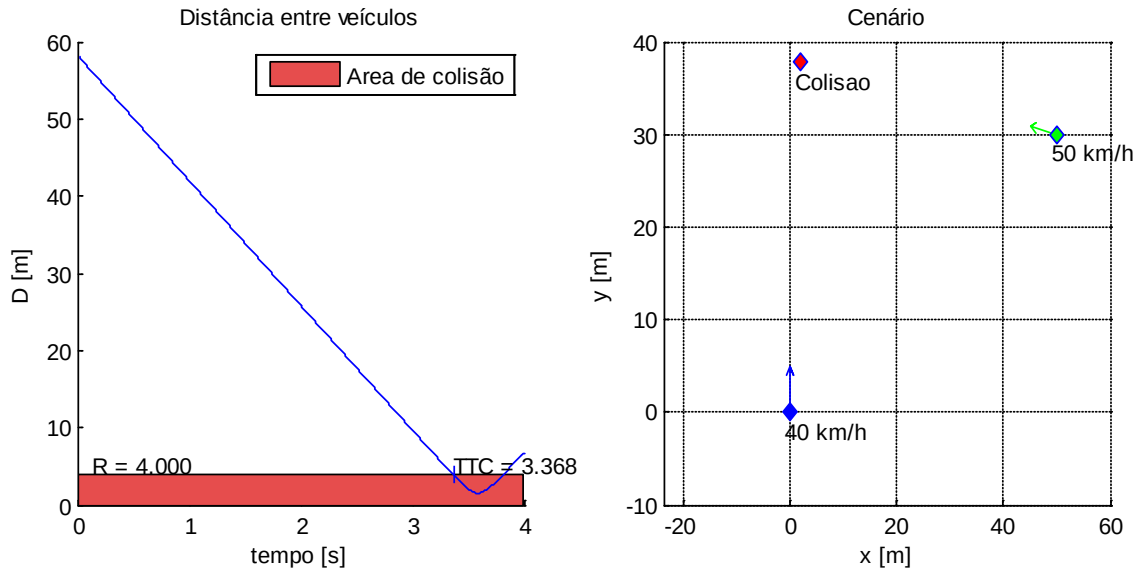


Figura 4.7 – Resultado do algoritmo IET num cenário de cruzamento com outro ângulo.

No exemplo da Figura 4.8 verifica-se a atuação precoce do algoritmo, obtendo aviso de colisão para uma distância ao cruzamento de 200 m, e um TTC de 14 s, motivo pelo qual é necessária uma filtragem nos resultados dos algoritmos.

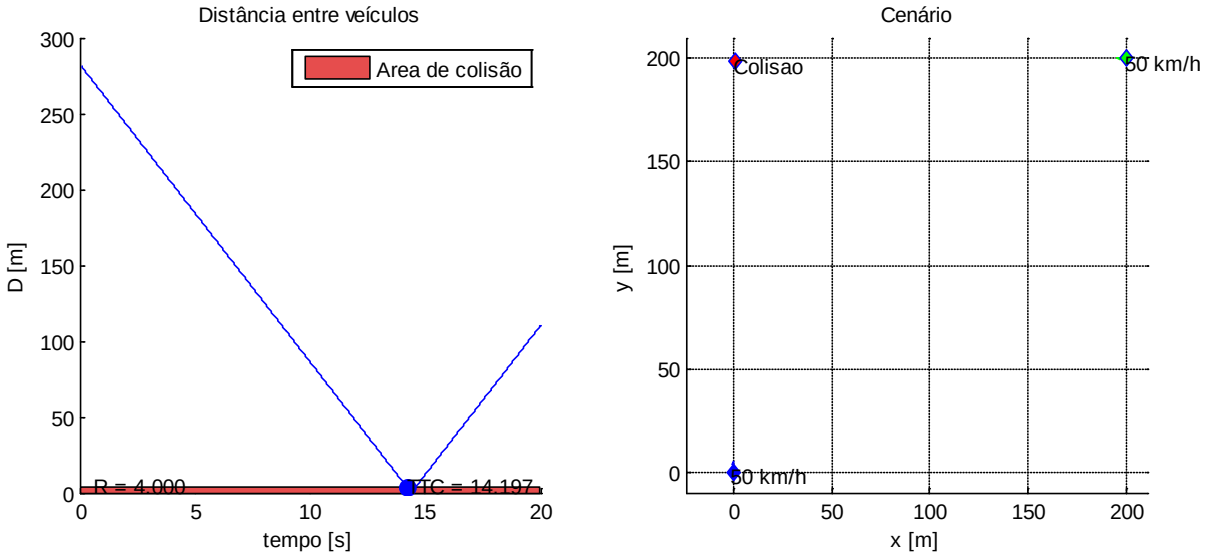


Figura 4.8 – Resultado do algoritmo IET num cenário de cruzamento a longa distância.

Na Figura 4.9 e Figura 4.10 estão presentes resultados do cenário de junção de vias de trânsito, por exemplo nas entradas das autoestradas. Confirma-se o correto funcionamento do algoritmo, sendo o segundo exemplo uma situação sem aviso de colisão, pois o algoritmo, como assume que os veículos seguem uma trajetória retilínea com a velocidade constante, assume que nesta situação não existe risco de colisão. Esta particularidade deste algoritmo

não é vantajosa sobre valores reais, pois um veículo não mantém a velocidade constante nem uma trajetória retilínea.

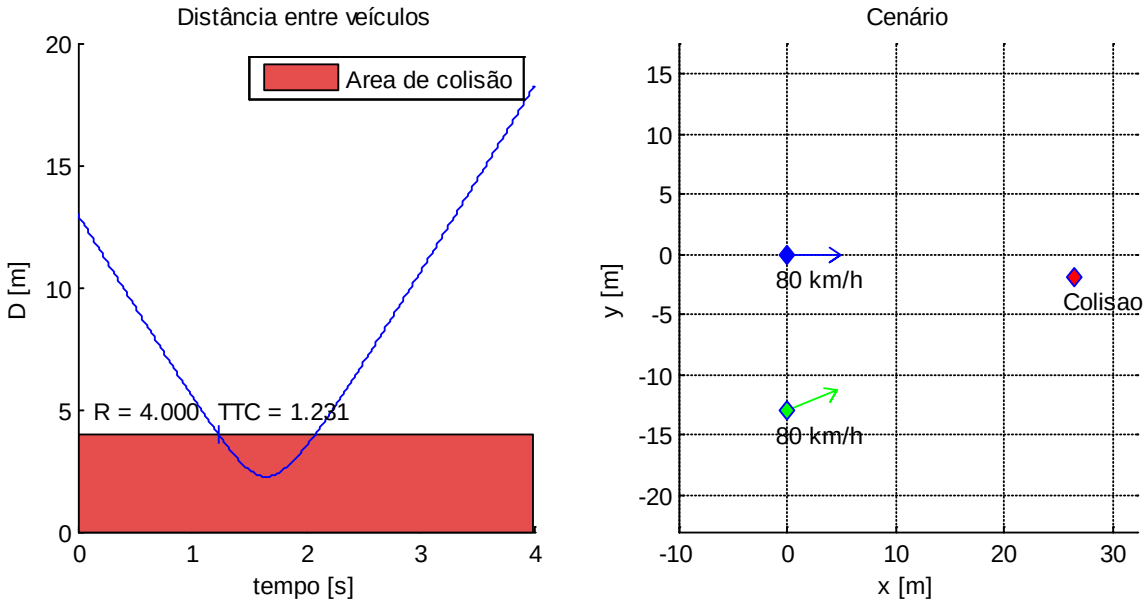


Figura 4.9 - Resultado do algoritmo IET num cenário de junção de vias.

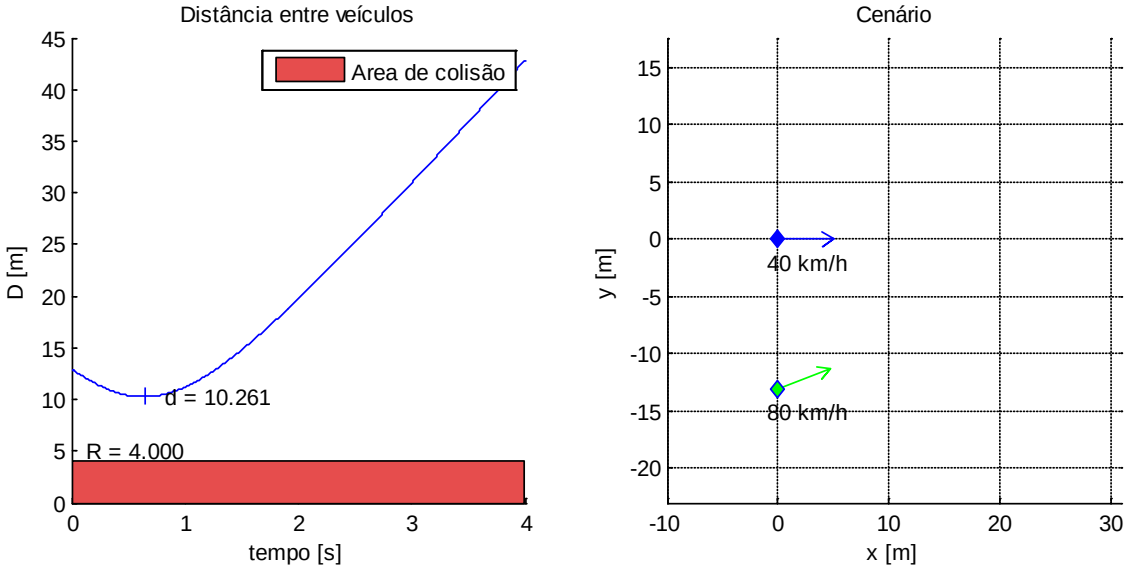


Figura 4.10 - Resultado do algoritmo IET num cenário de junção de vias sem aviso de colisão.

Outro algoritmo vantajoso e realizável neste projeto é o *Safety Zones*, subsecção 2.3.3, onde a geração de aviso de segurança consiste na sobreposição das zonas de segurança virtuais dos veículos.

O resultado do algoritmo num cenário de cruzamento está presente na Figura 4.11 e Figura 4.12, onde se observa o seu funcionamento bem-sucedido, com uma grande vantagem face o

algoritmo do IET, o facto de não gerar avisos precoces, como se pode confirmar na Figura 4.12.

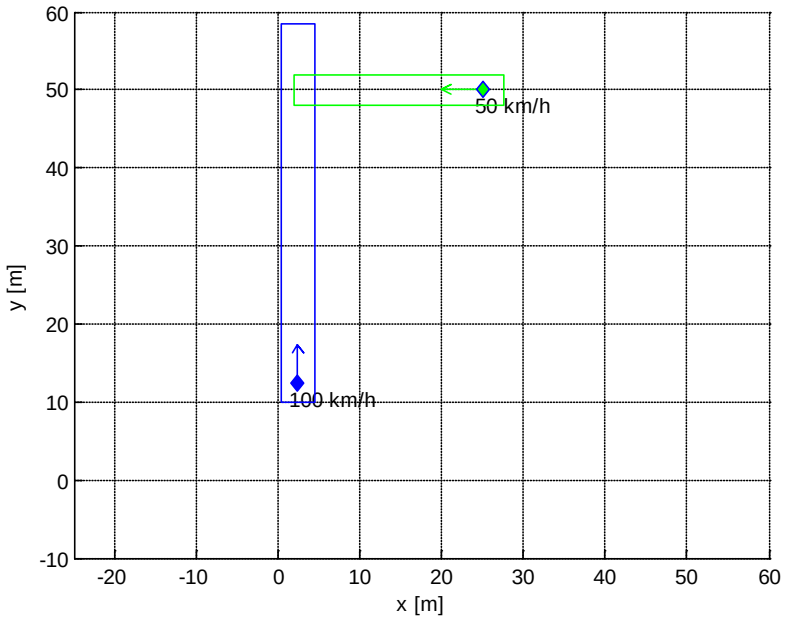


Figura 4.11 – Resultado do algoritmo *Safety Zones* num cenário de cruzamento.

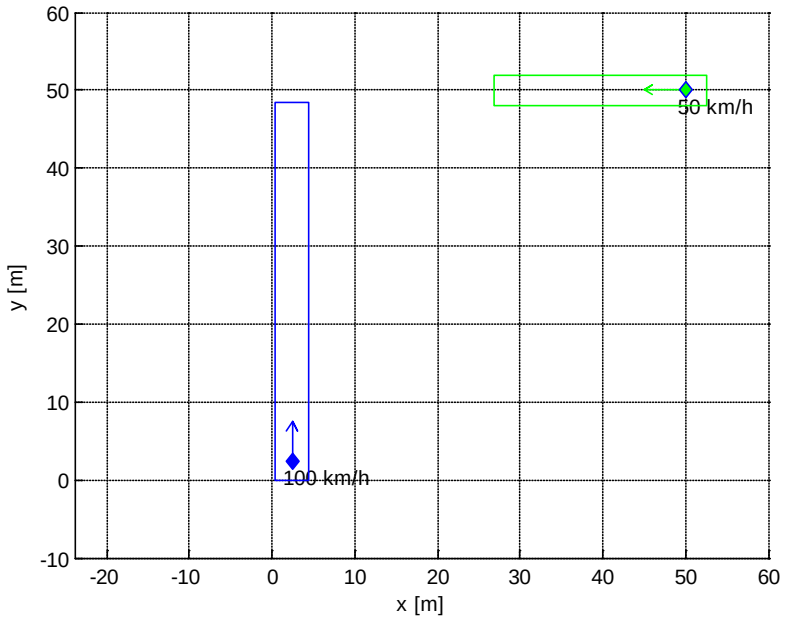


Figura 4.12– Resultado do algoritmo *Safety Zones* num cenário de cruzamento sem aviso de colisão.

O cenário de junção de vias de trânsito também se pode verificar no resultado presente na Figura 4.13.

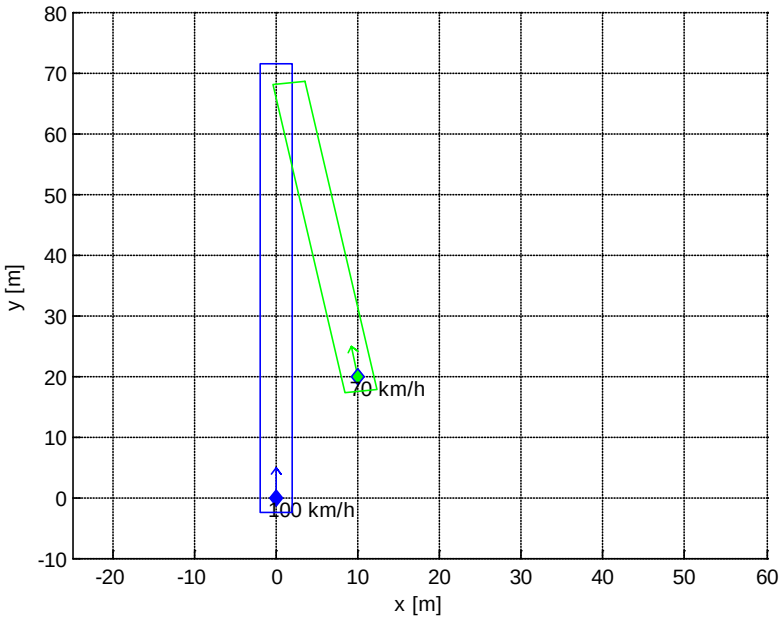


Figura 4.13 - Resultado do algoritmo *Safety Zones* num cenário de junção de vias.

Uma mais-valia deste algoritmo é o seu comportamento idêntico aos algoritmos unidimensionais em situações de aproximação súbita ao veículo procedente, como se pode verificar no resultado da Figura 4.14. Com esta característica do algoritmo, basta a sua implementação para ambos os cenários, poupando processamento.

A versatilidade deste algoritmo permite melhorar os seus resultados facilmente, por exemplo, utilizando o considerado melhor algoritmo unidimensional estudado nesta tese, algoritmo da PATH, para o cálculo da distância de segurança do retângulo virtual. Pode-se observar o seu resultado na Figura 4.15.

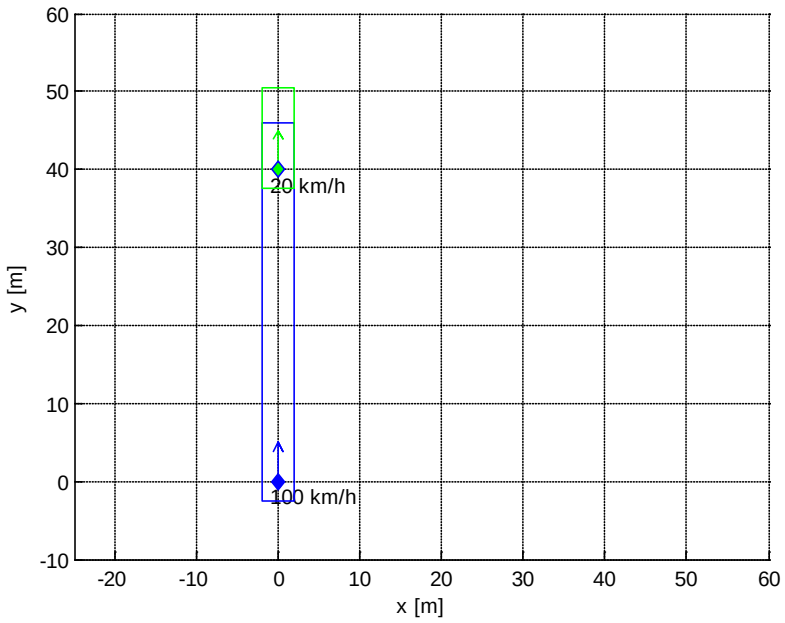


Figura 4.14 – Resultado do algoritmo *Safety Zones* num cenário de aproximação súbita ao veículo procedente.

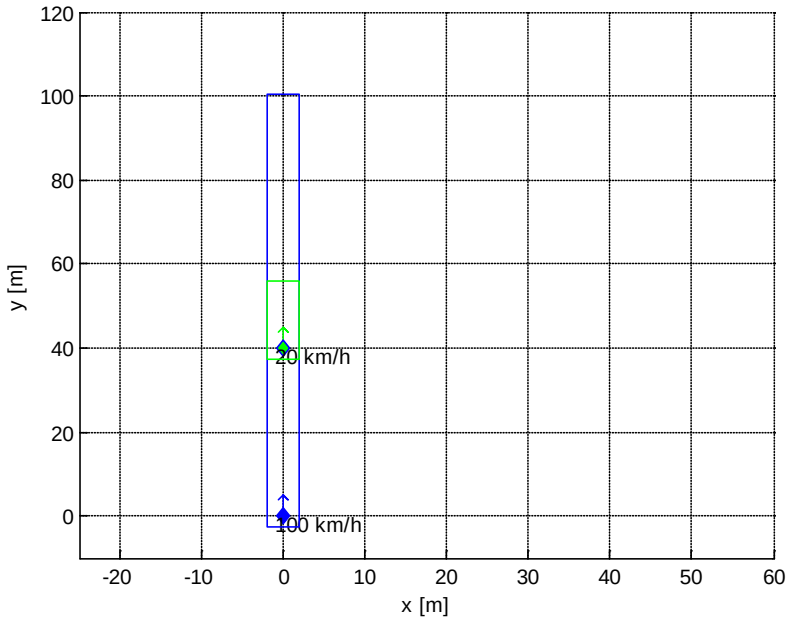


Figura 4.15 – Resultado do algoritmo *Safety Zones* num cenário de aproximação súbita ao veículo procedente, utilizando o algoritmo da PATH no cálculo do d_w.

Os restantes dois algoritmos bidimensionais que constam no estado da arte não são submetidos a testes através de simulação, pois o algoritmo da Ford é idêntico ao IET, acrescentando apenas um campo temporal, TTA, e o algoritmo Taiwan não é realizável no desenvolvimento desta tese, devido ao uso de acelerómetros e giroscópios.

4.1.3 Curva

Para teste ao algoritmo de curva são necessários dados reais obtidos através do sistema de GPS destinado para a realização deste trabalho. Foram guardados em ficheiro de texto os conjuntos de coordenadas relativos a três circuitos destintos, que incluem curvas e cruzamentos, numa urbanização de moradias, representação geográfica dos trajetos na Figura 4.16.

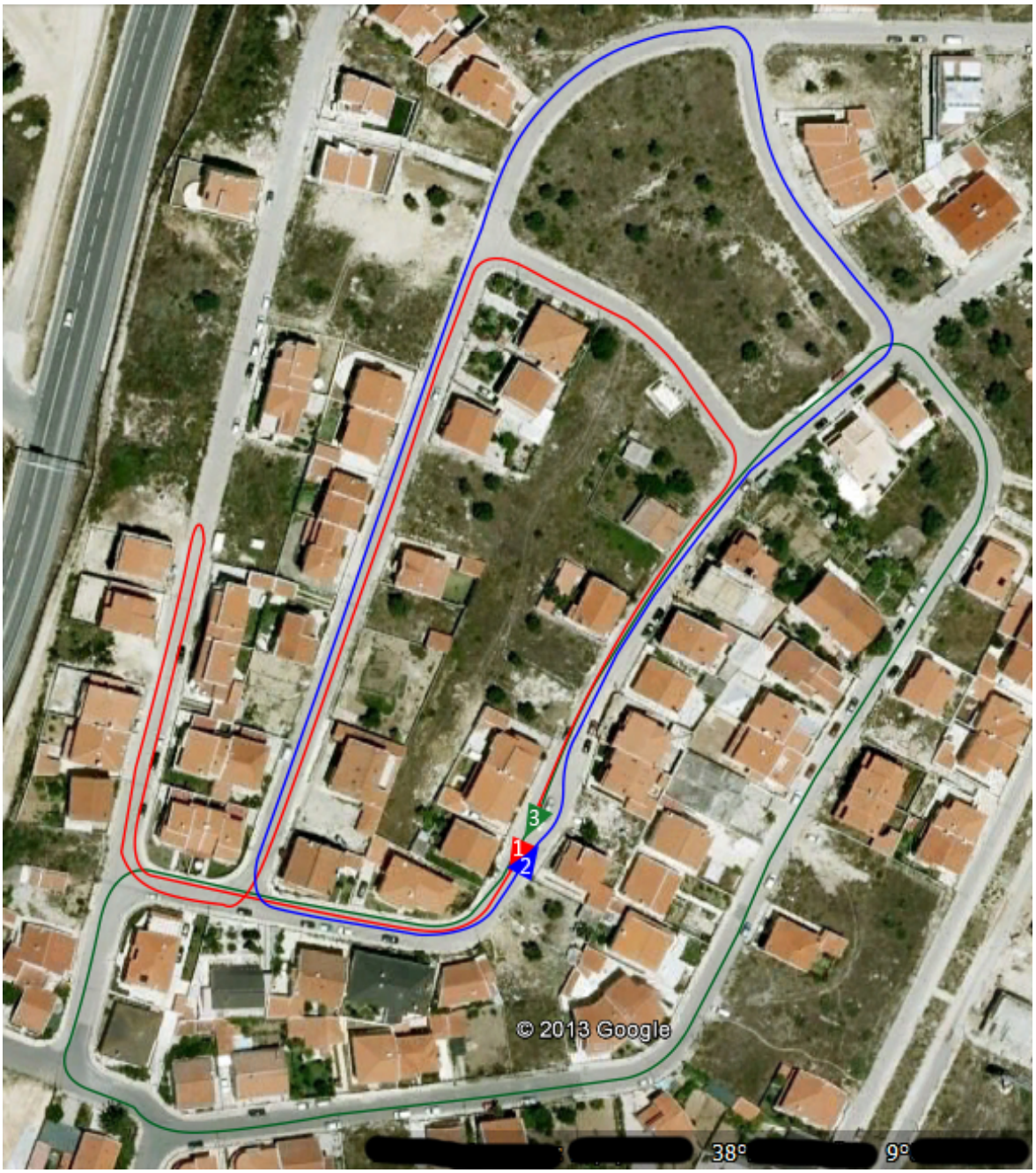


Figura 4.16 – Representação geográfica dos 3 circuitos gravados na urbanização de morarias (Google Earth).

Pode-se observar os resultados do algoritmo sobre os três circuitos na Figura 4.17, Figura 4.18 e Figura 4.19 respetivamente. A coordenada relativa à referência (0,0) dos gráficos apresentados foi selecionada ao acaso numa zona perto dos circuitos.

Separando as curvas dos cruzamentos/entroncamentos obtém-se os valores presentes na Tabela 4.2, onde as cores têm correspondência às dos gráficos dos circuitos. Pode-se concluir com estes testes que o raio de curvatura num cruzamento tem em média 9 m, menor que numa curva, isto deve-se à trajetória perpendicular que deve ser efetuada nos cruzamentos e entroncamentos segundo o código da estrada. Desta forma consegue-se fazer a distinção entre curva e cruzamento/entroncamento e atuar apenas o devido algoritmo, algoritmo bidimensional de risco de colisão ou algoritmo de curva.

Na Tabela 4.2, pode-se observar a velocidade média a que as curvas ou cruzamentos foram efetuados, note-se que algumas das curvas foram efetuadas a uma velocidade superior à suposta velocidade máxima para o seu raio de curvatura, segundo a equação **(3.8)** do tópico 3.1.5, nestas condições os condutores seriam alertados.

Tabela 4.2 – Valores dos raios do algoritmo de curva obtidos para os três circuitos reais.

Curvas				**Cruzamentos**		
Nº	**Raio [m]**	**Velocidade média [km/h]**	**Velocidade máxima [km/h]**	**Nº**	**Raio [m]**	**Velocidade média [km/h]**
1	14	17	20	4	9	33
4	33	36	31	5	9	20
2	12	21	18	6	8	27
5	28	24	28	7	11	24
6	18	42	22	2	8	7
7	15	34	21	3	8	12
				5	7	7
				3	8	17
				4	7	6
				8	5	10

Certos resultados não constam na Tabela 4.2, pois correspondem a erros. No circuito 1, na curva nº 2, está-se perante um erro do recetor de GPS, note-se que um veículo não conseguiria executar a trajetória relativa as várias coordenadas, velocidades e direções obtidas dessa curva. O erro de localização proveniente do sistema de GPS, como foi referido, pode chegar até aos 10 m, pode-se verificar na curva nº 2 e 4 do circuito 1, são relativas ao mesmo entroncamento mas diferem aproximadamente 20 m na sua localização, o que prova que os erros provenientes do sistema de GPS poderão ser superiores ao esperado, para o módulo GPS utilizado.

Ainda no circuito 1, curva nº 3, relativa a uma inversão do sentido de marcha, o algoritmo indicou uma curva com 11 m de raio, o que não é verídico, mas através da velocidade média dessa curva, 1 km/h, facilmente são filtradas estas situações.

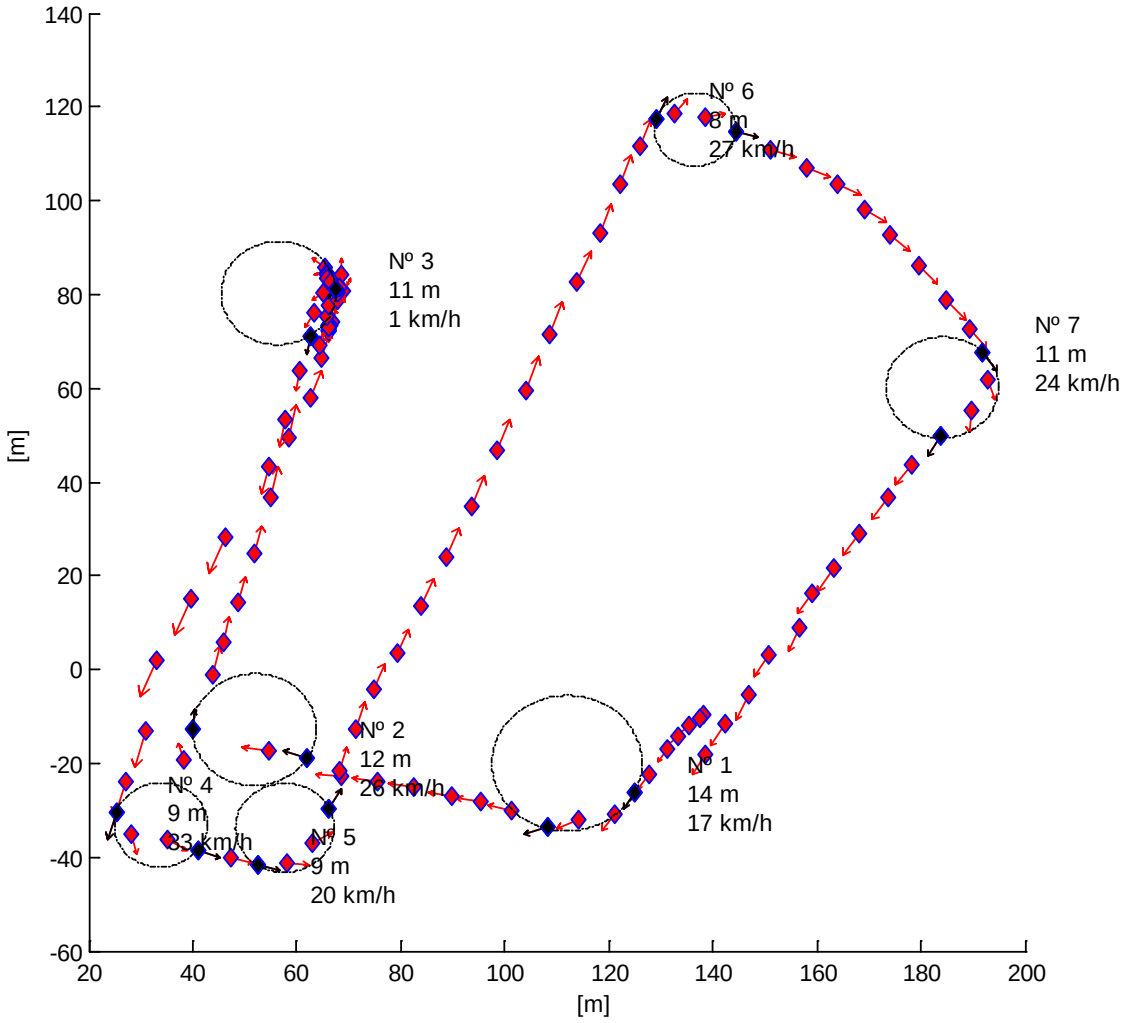


Figura 4.17 – Resultado do algoritmo curva na cobra do circuito 1.

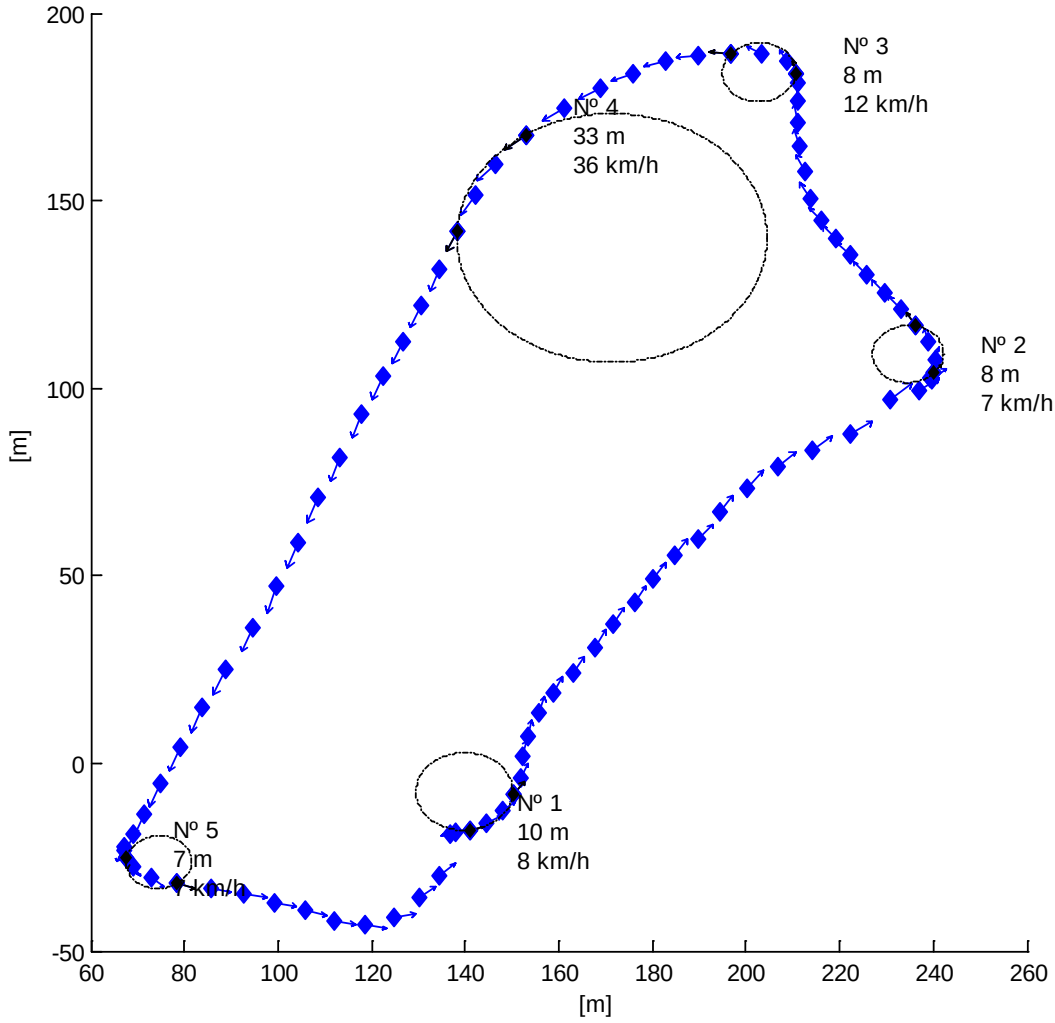


Figura 4.18 – Resultado do algoritmo curva na cobra do circuito 2.

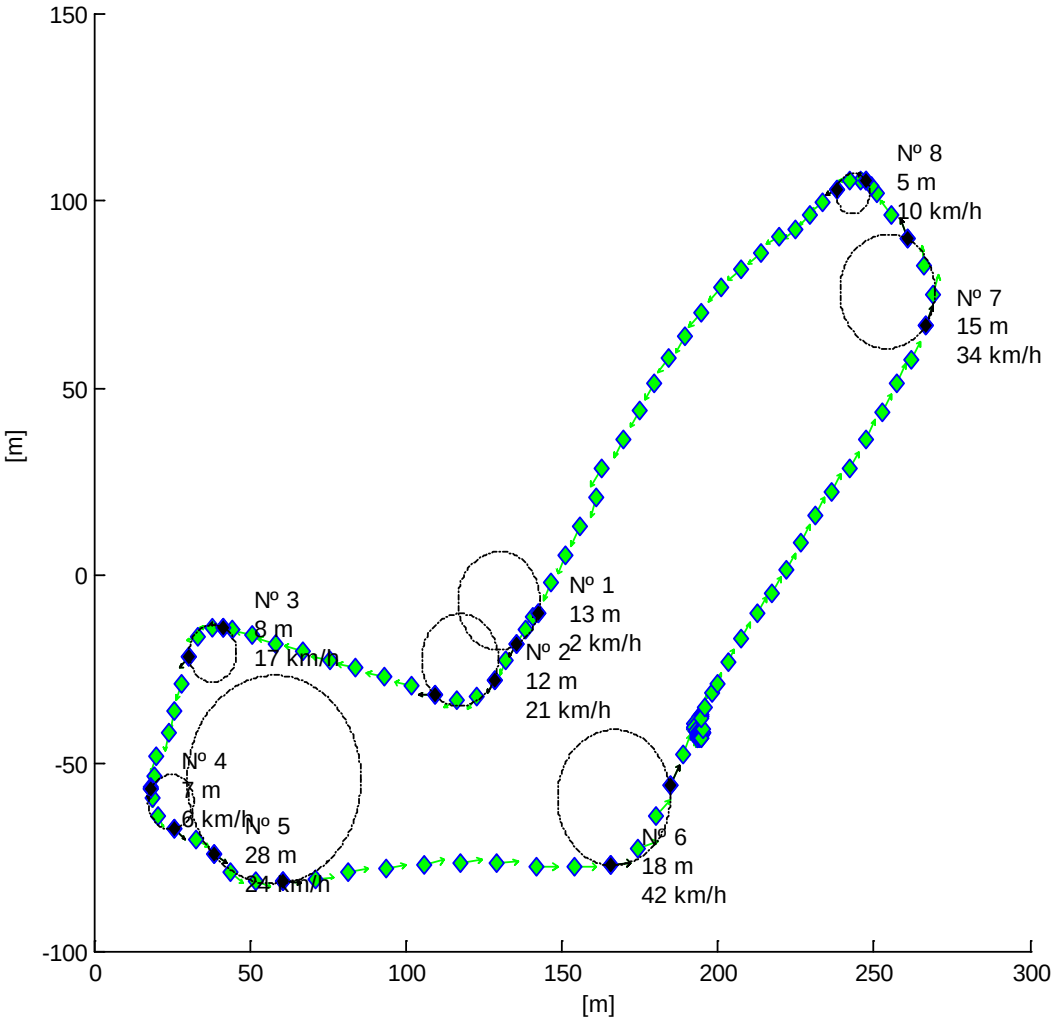


Figura 4.19 – Resultado do algoritmo curva na cobra do circuito 3.

4.2 Resultados práticos

Os resultados práticos do sistema total não são de fácil obtenção sobre uma representação gráfica. Deste modo, os resultados práticos relativos ao algoritmo implementado no sistema (*Safety Zones* com as alterações descritas em 3.3.2) podem ser retirados do *software* de simulação introduzindo percursos reais anteriormente gravados em ficheiro.

Pode-se observar na Figura 4.20 a simulação de dois percursos reais em rota de colisão num cenário de entroncamento, em que o veículo de cor amarela possui a implementação do algoritmo, e circulava a uma velocidade de 30 km/h quando foi alertado, a 15 m do cruzamento, de risco de colisão com outro veículo, de cor encarnada que circulava a uma velocidade de 40 km/h. Estes valores são idênticos aos obtidos através dos testes ao sistema no terreno.

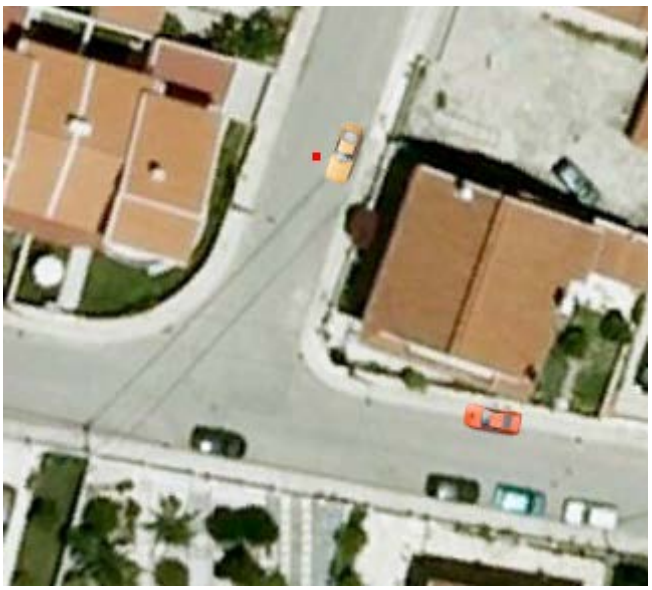


Figura 4.20 - Simulação de dois percursos reais em rota de colisão num entroncamento.

Também se pode observar resultados em autoestrada, cenário de aproximação súbita ao veículo procedente, na Figura 4.21, onde, o veículo de cor azul possui o OBU e circulava a 120 km/h quando foi alertado de risco de colisão com o veículo de cor encarnada, 60 m à sua frente, que circulava a 40 km/h.

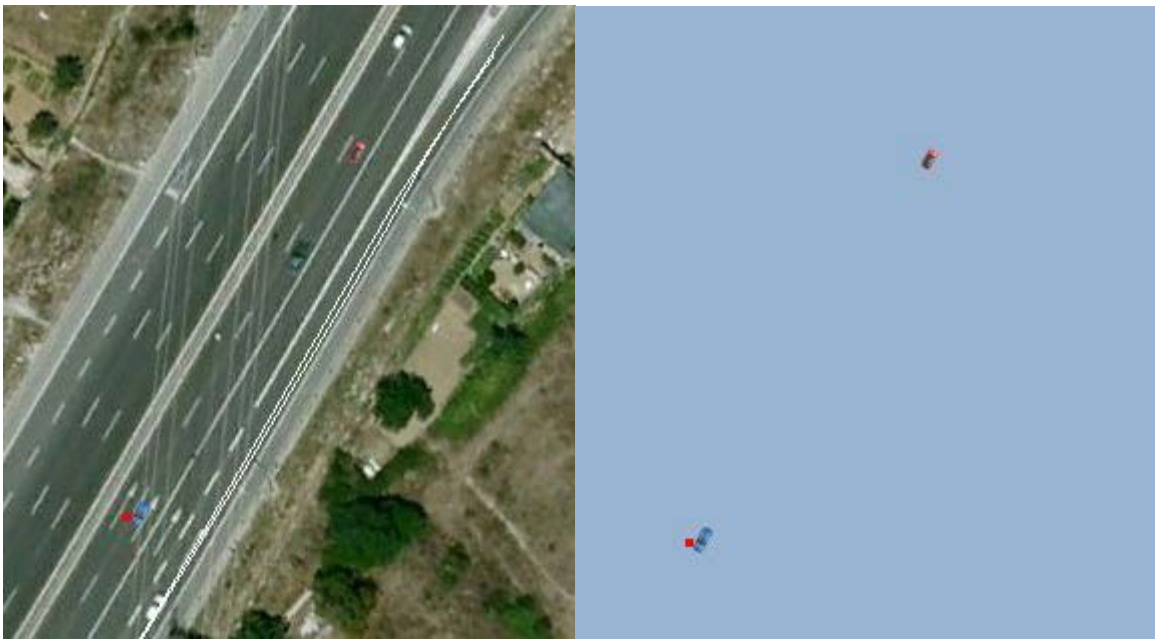


Figura 4.21 - Simulação de dois percursos reais em rota de colisão numa autoestrada.

Devido a problemas de assemblagem das placas PCB do sistema de RF não foi possível testar o sistema completo da forma prevista, no entanto, os resultados podem ser obtidos lendo dum ficheiro os dados que seriam recebidos via RF. Com base em fotografias à interface gráfica do sistema real, Figura 4.22 e Figura 4.23, pode-se observar exemplos do correto funcionamento dos algoritmos implementados, numa situação de risco de colisão com o veículo procedente e no caso de excesso de velocidade para a curva que procede, onde os dados da curva foram obtidos pelo trajeto efetuado pelo veículo procedente, vetor a verde.

Para uma melhor leitura às fotografias segue-se a seguinte descrição:

- Na da Figura 4.22, o veículo circulava a uma velocidade de 19 km/h com um azimute de 238º, a aplicação possuía uma zona de visibilidade de 34 m, e foi alertado dum veículo à sua frente com uma velocidade de 0 km/h;
- Relativamente à Figura 4.23, o veículo circulava a uma velocidade de 34 km/h com um azimute de 25º, a aplicação possuía uma zona de visibilidade de 61 m, e foi alertado do excesso de velocidade para a curva que procede, com um raio de curvatura de 33 m e uma velocidade máxima de 31 km/h. A circunferência encarnada marca a área de aviso.

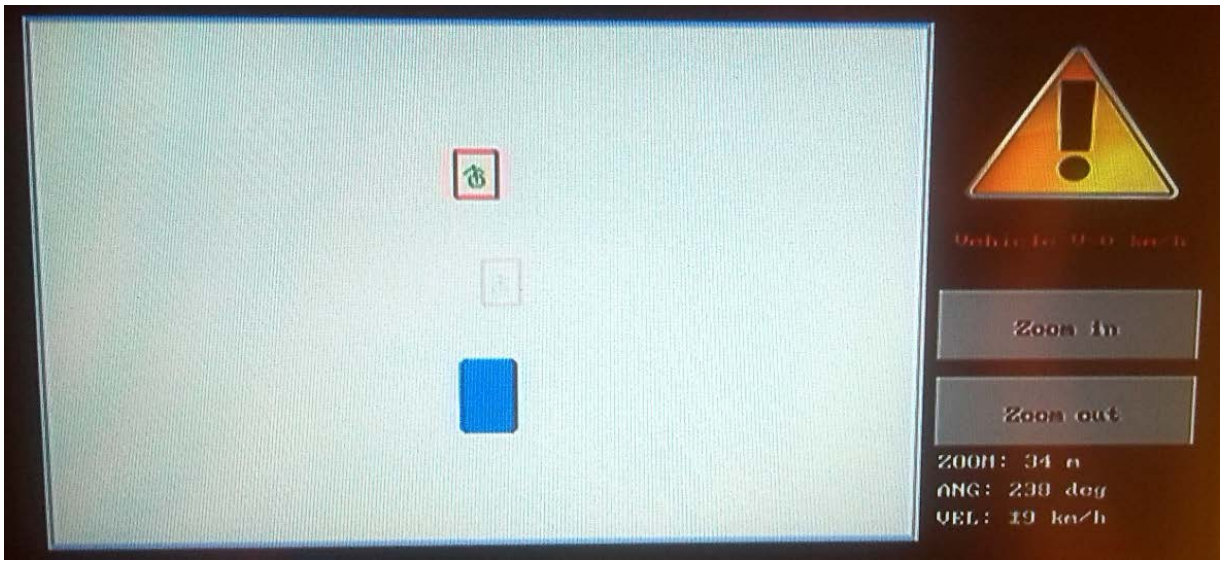


Figura 4.22 – Fotografia da interface gráfica do sistema numa situação real de aviso de colisão com um veículo parado.

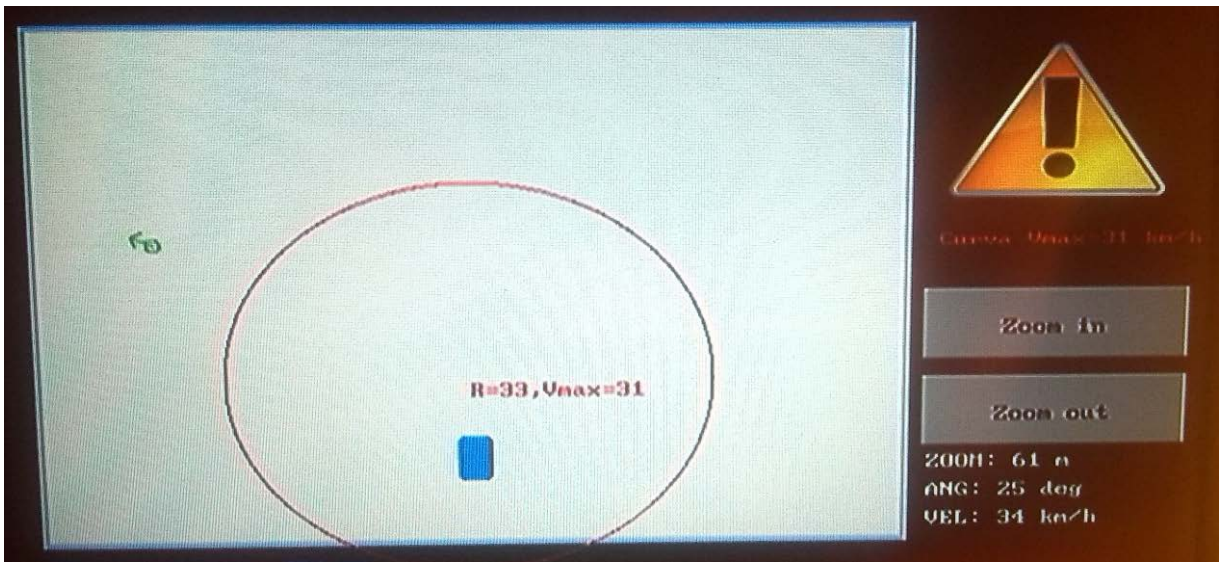


Figura 4.23 – Fotografia da interface gráfica do sistema numa situação real de aviso de excesso de velocidade para uma determinada curva.

5 Conclusões

5.1 Trabalho Realizado

Com base nos resultados obtidos nesta dissertação observa-se como é possível prever e até evitar um acidente rodoviário perante os três cenários destacados, aproximação súbita ao veículo procedente, risco de colisão perante um cruzamento/entroncamento ou despiste em curva.

A previsão do risco de colisão num cenário de aproximação súbita ao veículo procedente, que se trata do cenário mais estudado na literatura, foi concretizada com sucesso, obtendo um aviso prévio com eficiência. Elegendo os algoritmos da PATH e da Mazda como os mais adequados, que, perante uma situação em auto-estrada, circulando a 100 km/h avisa o condutor do veículo lento, a 20 km/h, a uma distância de 95 m, disponibilizando assim o tempo ideal para as devidas precauções. A eficiência deste algoritmo mostrou-se útil, de tal maneira a complementar os algoritmos bidimensionais, introduzindo este no cálculo do comprimento do retângulo virtual do algoritmo Safety Zones.

A geração de aviso perante a situação de cruzamento ou entroncamento também foi bem-sucedida utilizando o algoritmo Safety Zones. A atuação do algoritmo mostrou-se prática e fluida em testes no terreno, onde se obteve indicação de risco de colisão aproximadamente 15 m antes do cruzamento com outro veículo. Os algoritmos bidimensionais baseados apenas em fundamentos teóricos mostraram-se incapazes perante uma situação real, sendo o caso do algoritmo da IET, que através de simulação apresentou resultados bons, mas perante as imprecisões do sistema de GPS e a irregularidade na trajetória dos veículos não concretizou os objetivos pretendidos.

O novo conceito de Cobra provou ser versátil e de extrema utilidade, melhorando significativamente os resultados dos algoritmos estudados. O algoritmo Curva, para uma primeira abordagem ao conceito novo, apresentou bons resultados, o cálculo automático da velocidade máxima para um determinado raio de curvatura está bastante próximo do ideal, uma curva no interior duma localidade com 33 m de raio de curvatura obtém uma velocidade máxima de 31 km/h, que no terreno mostrou ser a velocidade ideal para a execução da curva

com segurança. Porém, o método de obtenção do raio de curvatura da curva depende da correta condução dos automobilistas, melhorou-se este aspeto introduzindo a gravação dos dados das curvas em memória e fazendo a média sempre que é obtida informação duma curva.

Com base nos resultados obtidos é visível a possibilidade de evitar acidentes rodoviários, porém, não foi possível testar o sistema completo utilizando a receção dos dados por RF, devido a erros na assemblagem das placas PCB do sistema de RF, o que não será problemático a um nível académico pois uma vez a comunicação RF estável o sistema total tenderá a obter a mesma resposta que foi obtida recebendo os dados dos veículos na vizinhança por outra fonte (ficheiro).

A implementação dos algoritmos no estado da arte mostrou-se insegura, para além da geração dos falsos avisos já mencionados, não detetam corretamente riscos de colisões sobre um cenário com coordenadas reais, erro que foi facilmente detetado efetuando testes no *software* de simulação, pois na realidade as trajetórias dos veículos não são retas nem circunferências ideais, daí a seleção pelo algoritmo de zonas de aviso, onde não se considera apenas uma reta mas sim uma área que pode ser alterada considerando a dimensão dos erros.

O sistema implementado tem vindo a ser desenvolvido no decorrer do projeto HEADWAY, esta dissertação complementou-o com a introdução dos algoritmos de segurança, onde ocorreram as maiores dificuldades, pois a complexidade e a quantidade de cálculos necessários para a implementação dos algoritmos é bastante suscetível a erros numéricos de difícil deteção.

Este sistema mostrou-se bastante versátil, podendo interligar os algoritmos com outros componentes disponíveis, por exemplo, a utilização do CAN *bus* ou mesmo de dados recebidos de RSUs, que futuramente poderá ser uma realidade. Como exemplo temos a demonstração, em colaboração com a Brisa e a UA, da receção da velocidade do veículo via WAVE, obtida através dum sistema de sensores magnéticos no pavimento, que noutro cenário poderia transmitir as condições do piso e atuar nos cálculos dos algoritmos, aumentando a distância crítica, de modo a prever situações de gelo no pavimento ou situações idênticas.

5.2 Trabalho Futuro

Como foi referido ao longo desta dissertação, uma versão melhorada do sistema seria a inclusão de mapas, interligando os algoritmos com um sistema de planificação de trajetos comum, corrigindo alguns problemas destacados.

De modo como o cenário de aproximação súbita é abordado ocorrerá problemas em faixas de rodagem com mais que uma via de trânsito, devido às imprecisões relativas ao sistema de GPS, gerando falsos avisos perante uma aproximação súbita ao veículo procedente presente na via de trânsito ao lado. Este problema necessita duma solução inovadora a pensar futuramente.

Num trabalho futuro existem certos aspetos que poderão ser melhorados neste sistema, nomeadamente: a implementação completa das normas em vigor; utilizar-se apenas módulos ao invés da utilização de *kits*, que é o caso do *kit* de GPS utilizado; melhorar a interface gráfica utilizando bibliotecas disponíveis como o X11 e resolver os problemas descritos acima nesta conclusão.

Anexos

A. Controlador da porta Série

O controlador da porta série permite abrir várias ligações em paralelo com controlo de multi tarefas, *thread safe*, definindo para cada a sua taxa de transmissão, dimensão da palavra, paridade, número de *stop bits* e o controlo de fluxo.

Uma ligação série é definida pela seguinte estrutura:

```
typedef struct _Serialport {
        int             mFileDescriptor;
        bool            isOpen;
        termios*        mOldPortSettings;
        char *          mInputBuffer;
        char *          mInputBuffer_get;
        char *          mInputBuffer_put;
        pthread_mutex_t*        mutex;
} Serialport;
```

onde `mFileDescriptor` é o nome do descritor relativo à porta série, na situação de utilização dum conversor RS232-USB, o nome do ficheiro será "/dev/ttyUSB0", `mOldPortSettings` é o ponteiro para as definições presentes na porta série antes desta ser aberta para permitir restaurar o estado da porta, `mInputBuffer`, `mInputBuffer_get` e `mInputBuffer_put` são ponteiros para o *ring buffer* de entrada, alocado na memória no momento de criação desta estrutura, e `mutex` é o ponteiro para o *mutex* que está encarregue de manter o controlo *thread safe*.

São criados enumeradores para as várias definições da ligação disponíveis: `BaudRate`; `StopBits`; `Parity` e `FlowControl`, utilizados na chamada às funções da seguinte API desenvolvida:

- `Serialport * Serialport_Create();`
- `void Serialport_Destroy(Serialport* serialport);`
- `void Serialport_Open(Serialport * serialport, const char * name, const BaudRate baudRate, const CharacterSize charSize, const Parity parityType, const StopBits stopBits, const FlowControl flowControl);`

- `void Serialport_Open_Default(Serialport * serialport, const char * name);`
- `bool Serialport_IsOpen(Serialport * serialport);`
- `void Serialport_Close(Serialport * serialport);`
- `void Serialport_SetBaudRate(Serialport * serialport, BaudRate baudRate );`
- `BaudRate Serialport_GetBaudRate(Serialport * serialport);`
- `void Serialport_SetCharSize( Serialport * serialport,CharacterSize charSize ) ;`
- `CharacterSize Serialport_GetCharSize(Serialport * serialport);`
- `void Serialport_SetParity( Serialport * serialport,Parity parityType );`
- `Parity Serialport_GetParity(Serialport * serialport);`
- `void Serialport_SetNumOfStopBits( Serialport * serialport,StopBits numOfStopBits );`
- `StopBits Serialport_GetNumOfStopBits(Serialport * serialport);`
- `void Serialport_SetFlowControl( Serialport * serialport, const FlowControl flowControl );`
- `FlowControl Serialport_GetFlowControl(Serialport * serialport) ;`
- `bool Serialport_IsDataAvailable(Serialport * serialport);`
- `unsigned char Serialport_ReadByte( Serialport * serialport, const unsigned int msTimeout);`
- `int Serialport_Read( Serialport * serialport, char* dataBuffer, const unsigned int numOfBytes, const unsigned int msTimeout);`
- `void Serialport_WriteByte(Serialport * serialport,const unsigned char dataByte);`
- `void Serialport_Write(Serialport * serialport,char* dataBuffer, const unsigned int numOfBytes);`

Para permitir a receção, no momento de criação da estrutura da porta série é associado, através da chamada à função *signal*, ao evento SIGIO a execução da função auxiliar *Serialport_Signal*, que guarda os dados recebidos no *ring buffer* relativo a essa porta assim que esta indica ao sistema operativo dados disponíveis. As funções **`Serialport_Read`**, **`Serialport_ReadByte`** e **`Serialport_IsDataAvailable`** atuam somente sobre o *ring buffer* numa espera passiva bloqueante, disponibilizando a opção de tempo limite de espera.

O envio dos dados é concretizado com a execução das funções **`Serialport_WriteByte`** e **`Serialport_Write`** escrevendo diretamente no descritor da porta série.

B. Controlador da entrada tátil

A entrada tátil será ligada pela porta USB, onde o sistema operativo reconhece como um dispositivo de entrada (idêntico a um rato), que pode ser utilizado diretamente através do descritor "/dev/input/eventX", onde X é um número que pode variar.

O controlador disponibiliza a seguinte API:

- `void input_init(char * input_dev);`
- `void input_stop();`
- `void input_calibration();`
- `void input_getClickLock(Point * p);`
- `int input_getClick(Point * p); //retorna o id do input`
- `void input_flush();`

A inicialização faz a abertura do descritor presente na *string* `input_dev`, inicia um *mutex* que possui como variável global, para garantir um sistema *thread safe*, e inicia a tarefa encarregue de fazer a leitura do descritor.

A tarefa criada está constantemente a ler do descritor estruturas do tipo `input_dev`, que possui três campos: tipo; código e valor, definida em `<linux/input.h>`, e apenas processa o evento do tipo EV_ABS, e regista os valores cujo os códigos são ABS_Y e ABS_X, relativos à posição espacial (*x*,*y*) do toque. Esses valores são registados num *array* circular global de dimensão fixa, e é incrementada a variável global *input_id* que indica o número de identificação do último toque. As funções **`input_getClickLock`** e **`input_getClick`** apenas retornam o valor mais recente desse *array* circular, diferindo entre si a espera, bloqueante ou não bloqueante respetivamente.

A calibração do toque é efetuada pela chamada à função **`input_calibration`**, que presume uma prévia inicialização do vídeo, desenha no ecrã dois pontos conhecidos, um de cada vez, e dá a indicação para o utilizador clicar sobre eles, registando os valores obtidos nesses dois pontos, são criadas duas funções de reta, $mx+b$ e $my+b$, de modo a que o valor de entrada corresponda ao ponto desenhado, resultando as variáveis globais *cal_mx*, *cal_bx*, *cal_my* e *cal_by*. Assim nas funções **`input_getClickLock`** e **`input_getClick`** antes do retorno é efetuado o cálculo das equações de reta de calibração.

C. Controlador de vídeo

Para a manipulação de vídeo a forma mais direta é utilizando o *framebuffer*, uma das funcionalidades disponibilizadas no sistema operativo Arch Linux, onde se pode alterar a imagem a apresentar escrevendo diretamente o descritor /dev/fb0.

De modo a acelerar o processo de implementação da tese foi utilizado como base o código fonte criado por Daniele Venzano em 2000, que disponibiliza a seguinte API:

- `int FB_initlib(char *dev);`
- `int FB_exit();`
- `inline FB_pixel FB_makecol(u_char r,u_char g, u_char b, u_char t);`
- `void FB_clear_screen(FB_pixel color);`
- `inline void FB_putpixel(int x, int y, FB_pixel color);`
- `FB_pixel FB_getpixel(int x, int y);`
- `void FB_line(int sx, int sy, int ex, int ey, FB_pixel color);`
- `void FB_vline(int x, int sy, int ey, FB_pixel color);`
- `void FB_hline(int sx, int ex, int y, FB_pixel color);`
- `void FB_rect(int sx, int sy, int ex, int ey, FB_pixel color);`
- `void FB_rectfill(int sx, int sy, int ex, int ey, FB_pixel color);`
- `void FB_triangle(int x1,int y1,int x2,int y2,int x3,int y3,FB_pixel col);`
- `void FB_circle(int cx, int cy, int radius, FB_pixel color);`
- `int FB_getVisual();`
- `void FB_getres(int *x, int *y);`
- `int FB_getXVres();`
- `int FB_getYVres();`
- `int FB_getbpp();`
- `int FB_setbpp(int bpp);`
- `int FB_change_font(char *psf_file);`
- `int FB_putc(int c, int x, int y, FB_pixel col);`
- `int FB_printf(int x, int y, FB_pixel col, char *format, ...);`

Para o desenho de: uma linha; linha vertical; linha horizontal; retângulo; retângulo preenchido; triângulo e círculo, tem-se as funções: **`FB_line`**; **`FB_vline`**; **`FB_hline`**; **`FB_rect`**; **`FB_rectfill`**; **`FB_triangle`** e **`FB_circle`** respetivamente. Preencher o ecrã com uma cor é

feito com a chamada à função **FB_clear_screen** e o desenho de uma sequência de caracteres com **FB_printf**, idêntica à função do C *printf*. A informação binária do desenho dos carateres é retirada de ficheiros de fontes do tipo PSF, que é definido numa variável de ambiente no ficheiro Makefile da biblioteca, que poderá ser alterado em tempo de execução com a função **FB_change_font**.

Uma das funções mais importantes é a **FB_makecol** que indicando os *bytes* de cor encarnada, verde, azul e transparência é retornado uma variável do tipo `FB_pixel`, que consiste numa variável sem sinal de 32 bits com a palavra binária correspondente à cor indicada pronta a escrever no *framebuffer*, que variará dependente das suas características, *bpp*, *bits per pixel* e disposição das cores.

Iniciar e terminar a utilização do *framebuffer* é concretizado pelas funções **FB_initlib** e **FB_exit**.

Devido à necessidade de desenho de imagens foram desenvolvidas funções complementares a esta biblioteca, que atuam sobre a seguinte estrutura relativa a uma imagem:

```
typedef struct tagBITMAP   /* the structure for a bitmap. */
{
      int width;
      int height;
      FB_pixel * data;
} BITMAP;
```

Foi criada a seguinte API:

- `BITMAP * FB_bitmapLoad(char * image_name);`
- `void FB_bitmapDestroy(BITMAP* bmp);`
- `void FB_bitmapDraw_T(BITMAP *bmp,int x,int y, FB_pixel transp_color);`
- `void FB_bitmapDraw(BITMAP *bmp,int x,int y);`
- `void FB_imageDraw_T(char * image_name,int x,int y, FB_pixel transp_color);`
- `void FB_imageDraw(char * image_name,int x,int y);`

Com a função **FB_bitmapLoad** é criada a estrutura `BITMAP` a partir dum ficheiro BMP, JPEG ou PNG (*Bitmap*, *Joint Photographic Experts Group* ou *Portable Network Graphics*). A conversão dum ficheiro BMP é quase direta, dependendo dos *bits* por pixel retira-se diretamente os valores RGB e converte-se para `FB_pixel`, os restantes ficheiros, JPEG e

PNG, são descomprimidos com o auxílio das bibliotecas *png* e *jpeg* contidas no sistema operativo, seguindo os exemplos de uso fornecidos. A função **FB_bitmapDestroy** faz a libertação de memória alocada para a estrutura criada.

As funções **FB_bitmapDraw** e **FB_bitmapDraw_T** desenham, pixel a pixel, a imagem contida na estrutura *bmp* na posição (*x*,*y*), deferindo entre si a possibilidade de indicar a cor transparente, ou seja, a cor desprezada, que não será desenhada, podendo passar o valor -1 que indicará que a cor transparente é o primeiro pixel da imagem. As funções **FB_imageDraw** e **FB_imageDraw_T** têm o mesmo efeito que as anteriores, são utilizadas para omitir a criação da estrutura, internamente é criada, desenhada e destruída.

D. Biblioteca de eventos

Foi desenvolvida uma biblioteca de eventos, de modo a facilitar a geração de movimentos gráficos em função da entrada, possibilitando inúmeras mais funções pois foi desenvolvida de forma genérica.

Um evento é caraterizado pela estrutura:

```
typedef struct event_{
        bool act;
        void * parameter;
        void (*func)(void*);
}Event;
```

A API disponibilizada é a seguinte:

- `void event_init();`
- `void event_stop();`
- `Event * event_add(void (*func)(void*),void * parameter);`
- `Event* event_once_add(void (*func)(void*),void * parameter, int ms);`
- `void event_remove(Event * ev);`

A função de inicialização, **`event_init`**, inicia a tarefa encarregue de gerar os eventos. Esta percorre constantemente a lista global de eventos e caso `act` seja *true* executa a função `func` com o parâmetro `parameter`. Adiciona-se à lista global de eventos através da função **`event_add`**, e remove-se utilizando a função **`event_remove`**.

Foi também desenvolvido o evento com tempo de atraso, acionado através da função **`event_once_add`**, este é gerado passados os milissegundos indicados e é eliminado de seguida. Internamente é caraterizado pela estrutura:

```
typedef struct event_timed_{
        int start_ms;
        int goal_ms;
        void * parameter;
        void (*func)(void*);
        Event * node;
}EventTimed;
```

onde `start_ms` é o valor em milissegundos presente no relógio da máquina na altura da criação, `goal_ms` é o valor que define quanto tempo deve demorar até o evento ser gerado, `func` e `parameter` a função e o respetivo parâmetro que deve ser executada na geração do

evento. Este evento é gerado tomando partido dum evento normal, na sequencia da sua criação é adicionado à lista global de eventos uma estrutura `Event` em que o `parameter` é o ponteiro para a estrutura `EventTimed` criada e `func` é uma função fixa (*timedEvent*) que está encarregue de controlar o momento em que o valor no relógio da máquina subtraído a `start_ms` é superior ou igual a `goal_ms` com o propósito de gerar o evento. Assim que é gerado será eliminado, daí a adição do campo `node` na estrutura `EventTimed`, que apontará para a estrutura `Event` adicionado à lista de eventos com a finalidade de a eliminar.

Para uma melhor visualização da lista global de eventos pode-se observar a esquematização dum exemplo na Figura D.1.

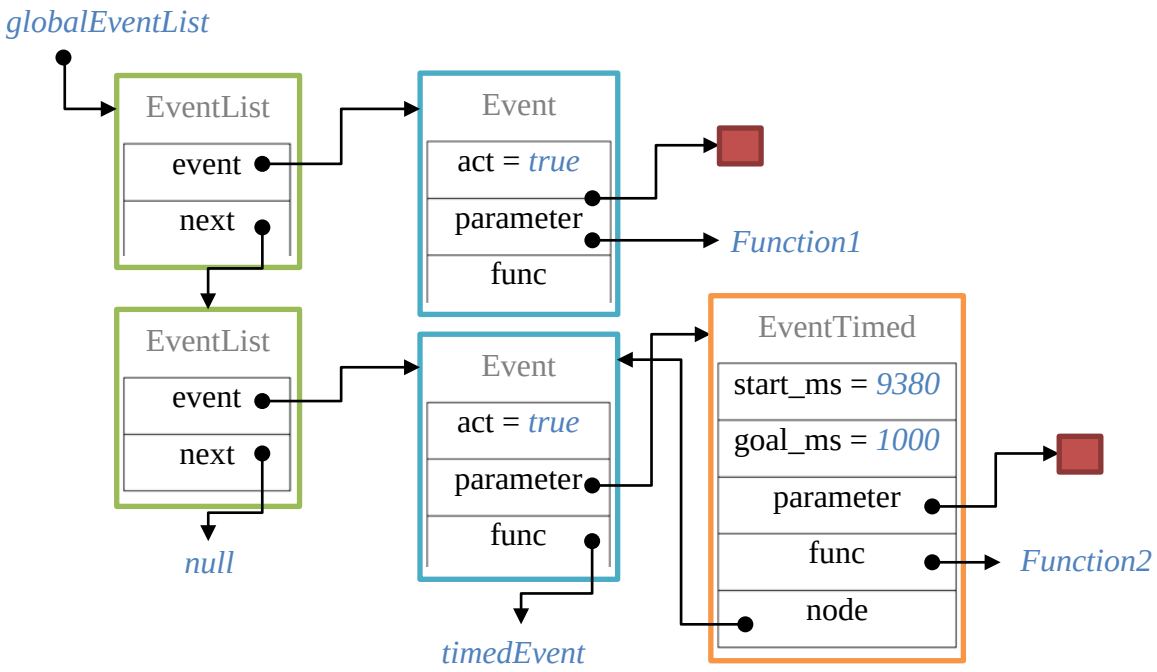


Figura D.1 – Esquema de exemplo da estrutura da lista de eventos.

E. Biblioteca de *Widgets*

A criação duma biblioteca de *Widgets* é útil principalmente para a inserção de botões, dando uso as bibliotecas de vídeo e de eventos.

A biblioteca desenvolvida, que consiste na manipulação duma lista de componentes composta por `W_Button`, `W_ImageButton` e `W_Label`, disponibiliza a seguinte API:

- `void widget_init();`
- `void widget_exit();`
- `inline void w_redraw(W_List * node);`
- `void w_redraw_all();`
- `void widget_free_all();`
- `void w_clean_screen();`
- `W_List* w_button_add(char*text , int font_size , int x, int y, int x1, int y1, void (*action)(void*), void * param);`
- `W_List* w_label_add(char*text , int font_size , int x, int y,int color);`
- `void w_label_text(W_Label * l, char * text);`
- `W_List* w_imagebutton_add(char * image_name , int x, int y, int trans_color,void (*action)(void*), void * param);`

A inicialização, **`widget_init`**, apenas inicia a lista de componentes a *null* e **`widget_exit`** está encarregue de libertar a memória alocada para essa lista.

Os botões podem ser adicionados e desenhados a partir das funções **`w_button_add`** e **`w_imagebutton_add`**, indicando a função `action`, com o respetivo parâmetro `param`, que deverá ser executada ao premir o botão. A função **`w_label_add`** adiciona uma *Label*, que permite a alteração do seu texto em tempo de execução pela função **`w_label_text`**.

A função **`w_redraw`** permite redesenhar um componente e **`w_redraw_all`** permite redesenhar todos os componentes na lista. Para se eliminar todos os componentes é utilizada a função **`widget_free_all`**, caso se pretenda eliminar todos e limpar o ecrã utiliza-se **`w_clean_screen`**.

A estrutura que carateriza a lista tem o seguinte aspeto:

```
typedef struct _W_list{
```

```
        W_Type type;
        void * component;
        void (*destroy)(void*);
        void (*redraw)(void*);
        void (*move)(void*, int x, int y);
        struct _W_list * next;
}W_List;
```

O tipo está relativo ao enumerador:

```
typedef enum _W_Type { W_BUTTON , W_IMAGEBUTTON, W_LABEL} W_Type;
```

E para o exemplo do `W_BUTTON` o componente está definido por:

```
typedef struct _W_Button{
        Point location;
        Point end;
        int text_len;
        char* text;
        int font_size;
        char pressed;
        Event * event_press;
        Event * event_release;
        int last_time; //miliseconds
        int last_input;
        void * parameter;
        void (*action)(void*);
}W_Button;
```

Onde a sua localização é o ponto `location`, o ponto `end` é útil para definir, junto com `location`, a área de toque, `pressed` indica se o botão se encontra premido, `event_press` e `event_release` são os ponteiros para os eventos criados com métodos fixos que desenha e aciona a ação quando é premido e desenha quando deixar de estar premido ao fim dum tempo definido, controlado por `last_time`, o campo `last_input` guarda o ID da última entrada do toque para garantir que não assume que o botão é premido mais que uma vez com o mesmo toque, `parameter` e `action` é função de ação junto com o respetivo parâmetro que será executada com o evento `event_press`.

Todos os componentes criados para esta biblioteca deverão de conter as funções de redesenhar, destruir e mover para serem adicionados à lista global de eventos. Futuramente, facilmente se adiciona novos componentes garantindo esta interface.

F. Aplicação *main*

Previamente, antes do processamento do programa *RoadView*, existe uma fase de seleção de recursos presente na função principal, *main*, do programa *App*. A utilização do programa tem os seguintes argumentos:

```
App [-p <port>] [-f <filename>] [-g /dev/<serialport>] [-i /dev/input/<inputdev>] [-r] [-t]
[-c] [-h]
        -h : this help
        -p : receive coordinates of the other vehicles by socket in port indicated
        -f : extract the coordinates of the other vehicle from file indicated
        -r : receive the coordinates of the other vehicle by radio
        -g : turn on the GPS, in serial port indicated
        -t : turn on transmition the coordinates of the other vehicles by radio
        -n : no graphics
        -i : select touch input device (default: /dev/input/event1)
        -c : calibrate touch input on start
Examples:
        sudo ./App -f /tracks/bairro.tck
        sudo ./App -i /dev/input/event0 -c -g /dev/ttyUSB0 -f /tracks/bairro.tck
        sudo ./App -p 1234
        sudo ./App -t -p 1234
```

Tomando partido das opções disponibilizadas é possível realizar os diversos cenários de testes descritos em 3.2.3, para o exemplo do RSU o comando deverá ser `sudo ./App –t -p 1234`, que indica que irá receber os dados dos veículos através da comunicação por *sockets* no porto 1234, vindos do *software* de simulação e, com a indicação da opção `-t`, os dados são transmitidos via RF simulando a presença física dos veículos.

O primeiro teste do OBU, sem receção via RF, deverá ser feito através do comando `sudo ./App -i /dev/input/event0 -c -g /dev/ttyUSB0 -f /tracks/bairro.tck`, onde é indicado o dispositivo de entrada tátil, `/dev/input/event0`, com a opção `-c` efetua a sua calibração no arranque do programa, receberá os dados do veículo do módulo de GPS ligado à porta série `/dev/ttyUSB0` e os dados dos restantes veículos do ficheiro `/tracks/bairro.tck`, que possui uma formatação idêntica aos ficheiros utilizados no programa de simulação. Para executar o OBU sobre o sistema completo o comando será `sudo ./App -i /dev/input/event0 -c -g /dev/ttyUSB0 –r`, onde o comando `-f` foi alterado pelo `–r`.

G. Especificações da aplicação RoadView

Para a criação da lista global que guarda os dados dos veículos, incluindo a sua cobra, foram criadas as seguintes estruturas:

```
typedef struct _position{
        int x; //cm
        int y; //cm
        int vel; //km/h
        double d; // rad

        struct _position * next;
        struct _position * prev;
}Position;

typedef struct _vehicle{
        int id;

        int snake_count;
        Position * pos;

        int count; // para eliminar veiculo por timeout

        int last_x; // pixels
        int last_y; // pixels
        int last_vel; // km/h
        double last_angle; // rad
}Vehicle;

typedef struct _road{
        Vehicle * v;
        struct _road * next;
}Road;
```

A estrutura `Position` possui os dados dum veículo num determinado instante, posição, velocidade e direção, e constituí uma lista duplamente ligada através dos ponteiros `next` e `prev`, utilizada para a criação da cobra.

`Vehicle` é a estrutura relativa a um veículo, possui o seu ID, a cobra `pos` e a sua dimensão `snake_count`, a variável `count` é utilizada para eliminar o veículo após a falta de receção de dados durante um determinado tempo, as restantes variáveis com o prefixo *last* são utilizadas para eliminar do ecrã o vetor relativo a posição anterior do veículo, poupando processamento pois estas variáveis já possuem os valores em pixéis, evitando erros caso, por exemplo, o zoom seja alterado.

`Road` é a estrutura que compõe a lista global de veículos nomeada de *road*.

De forma a manter o programa genérico foi desenvolvida a seguinte API:

- `void RoadView_start(bool input_cal, char * input_dev);`
- `void RoadView_stop();`
- `void RoadView_delete(int vehicle_id);`
- `void RoadView_updateCoor(int vehicle_id,Coor * c);`
- `void RoadView_update_myCoor(Coor * c);`
- `void RoadView_update(int vehicle_id, int x_cm, int y_cm, int vel, unsigned int angle);`
- `void RoadView_update_my(int x_cm, int y_cm, int vel, unsigned int angle);`
- `bool RoadView_caution(Vehicle * v);`
- `int RoadView_ZoomIn(); // return meters visible in board area`
- `int RoadView_ZoomOut(); // return meters visible in board area`
- `void RoadView_redraw();`

Para o desenho dum vetor, relativo a um veículo, foi criada a função auxiliar *drawCar*, que passa por alguns passos de conversão:

1º Definir a origem, subtraindo à posição do veículo a desenhar a posição do veículo corrente;

2º Rodar o plano, de modo ao eixo das ordenadas assentar sobre a direção α do veículo corrente, utilizando as seguintes equações de rotação:

   a. $x = x\cos\alpha - y\sin\alpha$;

   b. $y = x\sin\alpha - y\cos\alpha$;

3º Converter a posição para pixéis, utilizando a variável global *meter_per_pixel* que é alterada consoante o zoom pretendido, que consiste em:

   a. $x = x/100/meter\_per\_pixel$;

   b. $y = y/100/meter\_per\_pixel$;

4º Posicionar o veículo no painel de visualização, tendo em consideração a posição e dimensão do painel definidos pelas variáveis globais *table_location* e *table_length* e a definição de *CAR_BACK_CM_VIEW* que indica quantos centímetros devem ser visíveis para a parte de trás do veículo, resultando as seguintes operações:

   a. $x = table\_location.x + table\_length.x/2 + x$;

   b. $y = table\_location.y - y + CAR\_BACK\_CM\_VIEW/100/meter\_per\_pixel$;

5º Verificar se este se encontra no interior do painel;

6º Definir a cor a desenhar:

a. Velocidade < 30: Verde;
b. Velocidade < 90: Azul;
c. Velocidade > 90: Vermelho;

7º Desenhar o vetor.

H. API do objeto Coor

Como auxilio fora criado um objeto, mas especificamente, uma estrutura com respetivas funções de manipulação, que contém as coordenadas, geográficas ou/e cartesianas, dos dados recebidos por *ethernet*, ficheiro ou pelo módulo de GPS, e é composto pela seguinte estrutura:

```
typedef enum _Coor_calc {NONE = 0x0, UTM = 0x1, GEO = 0x2 , BOTH =
0x3} Coor_calc;
typedef struct _Coor{
        double lon;
        double lat;
        double asimuth;
        double vel;

        int x; //cm em funcao de origin
        int y; //cm

        Coor_calc calc;
        struct _Coor * origin;
}Coor;
```

E possui a seguinte API:

- `Coor * Coor_new0();`
- `Coor * Coor_new1(double lon, double lat, double asimuth, double vel);`
- `Coor * Coor_new2(int lon_degree, double lon_min, double lon_sec, int lat_degree, double lat_min, double lat_sec, double asimuth, double vel);`
- `void Coor_destroy(Coor * c);`
- `void Coor_utmCalc(Coor * c);`
- `Coor * Coor_parce_w0(char* s, Coor* dest, Coor * origin);`
- `Coor * Coor_parce_str(char* s, Coor* dest);`

A estrutura pode ser criada através das funções **`Coor_new0`**, **`Coor_new1`** e **`Coor_new2`**, diferindo entre si os dados definidos nos parâmetros, é destruída através da função **`Coor_destroy`**. Neste objeto é onde é feita a conversão dos dados WSG84 para UTM com a chamada a função **`Coor_utmCalc`** sobre o método descrito em J, apenas efetua os cálculos uma única vez, sabe-se através do enumerador `Coor_calc`, o campo `origin` possui a informação da origem. As funções **`Coor_parce_w0`** e **`Coor_parce_str`** são utilizadas na leitura dos ficheiros que contêm a informação das coordenadas, estão capazes de converter uma linha do ficheiro, passada pelo ponteiro `s`, para a estrutura `dest`.

I. API do algoritmo de Curva

O algoritmo de curva consiste na execução das seguintes funções:

- `int getCurve(Position * p, Curve * c);`
- `void addCurve(Curve * c);`
- `Curve * checkCurves().`

Os dados relativos a uma curva constam em instâncias da seguinte estrutura:

```
typedef struct _curve{
        int R; // raio da curva, cm
        int RA; // raio da area de aviso, cm
        int vel_mean; // velocidade media, km/h
        int vel_max; // velocidade max para o raio da curva, km/h
        int x; // coordenada x do centro da área de aviso, cm
        int y; // coordenada y do centro da área de aviso, cm
        double d; // azimute de entrada na área de aviso, degrees
        int count; // count seconds to delete by time
        int entries; // nr of reg in same curve
        struct _curve * next;
}Curve;
```

J. Conversão de coordenadas

Os dados reais utilizados na implementação dos algoritmos são provenientes dum sistema de GPS real, maioritariamente sobre o sistema WSG-84, mas grande parte dos algoritmos espera um sistema coordenadas cartesianas bidimensionais, ou seja, terá de existir uma conversão dos dados do sistema de GPS num sistema de coordenadas geográficas chamado de UTM, *Universal Transverse Mercator*, ou outro sistema idêntico. Tendo como base na conversão a referência [34] obtém-se os passos que se seguem.

O sistema UTM divide o mapa do Mundo em 60 zonas, dependendo da longitude, definindo para cada zona um meridiano central, pode-se observar alguns exemplos na Tabela J.1. É essa linha vertical virtual que corresponderá a referência da coordenada horizontal do sistema UTM, *x* ou também conhecido pelo símbolo *E*, e a referência da coordenada vertical, *y* ou também *N*, é a linha do equador. Pode-se observar um exemplo duma zona do sistema UTM na Figura J.2.

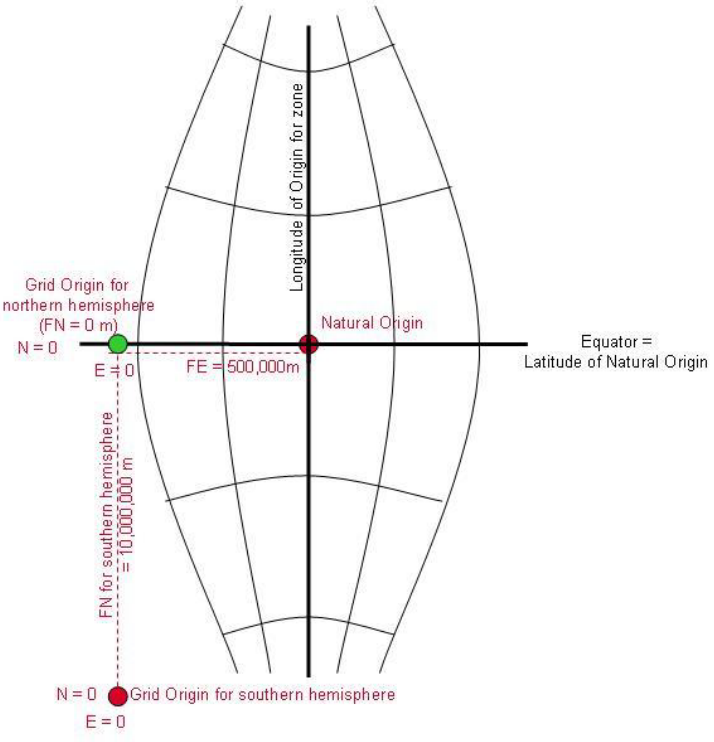


Figura J.2 – Exemplo duma zona do sistema UTM (retirado de [34]).

Tabela J.1 – Meridiano central de algumas zonas do sistema UTM.

Zona	Longitude [º]	CM (λ_0) [º]
31	[0 , 6 [	3
32	[6 , 12 [	9
33	[12 , 18 [	15
34	[18 , 24 [	21

Existem dois valores que devem ser definidos antes de proceder à conversão que dependem do tipo de coordenadas geográficas de origem, trata-se do valor do Raio Equatorial *a* e o Raio Polar *b*, 6.378.137 e 6.356.752,3142 m respetivamente para WGS-84, de onde se pode retirar o valor do achatamento da Terra (*flattening)*, através da expressão

$$f = \frac{a-b}{a} = \frac{1}{298{,}257223563}. \quad \textbf{(J.1)}$$

Para limitar ainda mais a distorção de escala dentro da cobertura da zona, algumas projeções transversais introduzem um fator de escala na origem, k_0, no caso da projeção UTM tem o valor 0,9996.

De seguida, dada uma determinada latitude ϕ e longitude λ, em radianos, pode-se obter o valor do raio de curvatura no plano do meridiano através de

$$R_m = a\frac{1-e^2}{(1-e^2\sin^2\phi)^{3/2}}, \quad \textbf{(J.2)}$$

onde

$$e^2 = 2f - f^2 = 0{,}00676865, \quad \textbf{(J.3)}$$

obtém-se o valor do raio de curvatura do primeiro vertical (*prime vertical)* pela expressão

$$R_N = \frac{a}{\sqrt{a-e^2\sin^2\phi}}, \quad \textbf{(J.4)}$$

e o raio de curvatura para uma determinada azimute α a partir de

$$R_\alpha = \frac{R_m R_N}{R_m \sin^2 \alpha + R_N \cos^2 \alpha}, \quad \textbf{(J.5)}$$

pode-se observar uma simples representação das três componentes na Figura J.3.

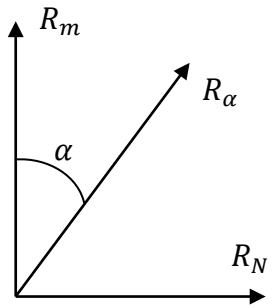


Figura J.3 – Simples representação de R_m, R_α e R_N.

Para facilitar a conversão para o sistema UTM necessita-se dos seguintes cálculos intermédios:

$$T = \tan^2 \phi, \quad \textbf{(J.6)}$$

$$C = e'^2 \cos^2 \phi, \quad \textbf{(J.7)}$$

$$e'^2 = \frac{e^2}{1 - e^2} = 0{,}00681478, \quad \textbf{(J.8)}$$

$$A = (\lambda - \lambda_0) \cos \phi, \quad \textbf{(J.9)}$$

$$M' = a\left[\begin{array}{c}\left(1 - \frac{e^2}{4} - \frac{3e^4}{64} - \frac{5e^6}{256} - \cdots\right)\phi - \left(\frac{3e^2}{8} + \frac{3e^4}{32} + \frac{45e^6}{1024} + \cdots\right)\sin 2\phi + \\ \left(\frac{15e^4}{256} + \frac{45e^6}{1024} + \cdots\right)\sin 4\phi - \left(\frac{35e^6}{3072} + \cdots\right)\sin 6\phi + \cdots\end{array}\right]. \quad \textbf{(J.10)}$$

Obtém-se finalmente os valores das coordenadas em UTM pelas seguintes expressões:

$$x = k_0 R_N \left[A + (1 - T + C)\frac{A^3}{6} + (5 - 18T + T^2 + 72C - 58e'^2)\frac{A^5}{120} \right] + 500000\,, \tag{J.11}$$

$$y = k_0 \left\{ M' + R_N \tan\phi \left[\frac{A^2}{2} + (5 - T + 9C + 4C^2)\frac{A^4}{24} + (61 - 58T + T^2 + 600C - 330e'^2)\frac{A^6}{720} \right] \right\}. \tag{J.12}$$

Pode-se observar uma exemplificação de coordenadas UTM na Figura J.4.

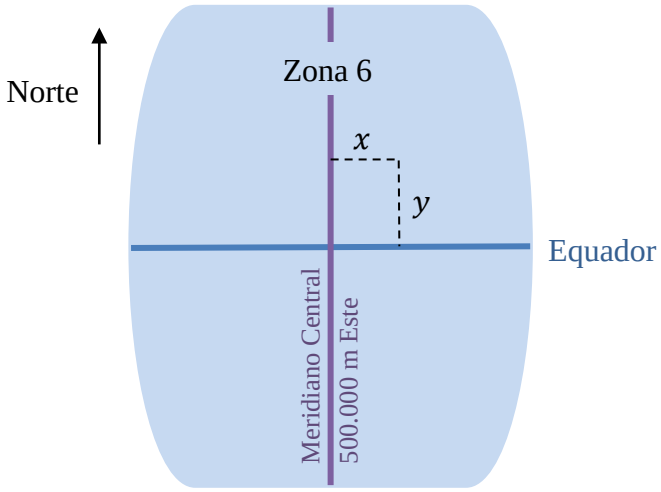


Figura J.4 – Exemplificação de coordenadas UTM.

Bibliografia

[1] **IEEE Computer Society,** *IEEE Standard for Information Technology - Telecommunications and Information Exchange Between Systems - Local and Metropolitan Area Networks - Specific Requirements, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY)*, IEEE Std 802.11p, US-NY, July, 2010.

[2] **IEEE Vehicular Technology Society,** *IEEE Standard for Wireless Access in Vehicular Environments (WAVE) - Networking Services*, IEEE Std 1609.3, US-NY, Dec., 2010.

[3] **IEEE Vehicular Technology Society,** *IEEE Standard for Wireless Access in Vehicular Environments (WAVE) - Multi-channel Operation*, IEEE Std 1609.4, US-NY, 2010.

[4] **Peter D. Cenek, Colin A. Brodie, Robert B. Davies, Fergus N. Tate,** *A Prioritisation Scheme for the safety management of curves*, 3rd International Surface Friction Conference, Safer Road Surfaces – Saving Lives, Gold Coast, Australia, 2011.

[5] **The New York Times,** *Traffic Deaths Fall to Lowest Level Since 1949*. Disponível em: http://wheels.blogs.nytimes.com/2011/12/09/traffic-deaths-fall-to-lowest-level-since-1949/ [Acedido em 25-09-2013] .

[6] **European Commission,** *Mobility And Transport, Road Safety, Statistics – accidents data*. Disponível em: http://ec.europa.eu/transport/road_safety/specialist/statistics/ [Acedido em 25-09-2013]

[7] **DN Portugal,** *937 mortes em acidentes rodoviários no ano passado*. Disponível em: http://www.dn.pt/inicio/portugal/interior.aspx?content_id=1929570 [Acedido em 16-09-2013].

[8] **Sehun Kim, Sunghyun Lee, Inchan Yoon, Mija Yoon and Do-Hyeun Kim,** *The Vehicle Collision Warning System based on GPS*, Jeju National University, First ACIS/JNU International Conference on Computers, Networks, Systems, and Industrial Engineering, Jeju, Korea, 2011.

[9] **S Mangan, J Wang, Q Wu,** *Longitudinal Road Gradient Estimation Using Vehicle CAN Bus Data*, Department of Electrical Engineering and Electronics, University of Liverpool, Liverpool, UK, 2003.

[10] **Chun-Liang Chen, C. W. Hsu and Chi-Ming Wu,** *A DSRC-Based Collision Warning and Autonomous System for Inter-Vehicle Communication*, Cross Strait Quad-Regional Radio Science and Wireless Technology Conference, Nantou, Taiwan, 2011.

[11] **Paul Alexander, David Haley, and Alex Grant,** *Cooperative Intelligent Transport Systems: 5.9-GHz Field Trials*, Proceedings of the IEEE, vol. 99, no. 7, Australia, July, 2011.

[12] **Lin Cheng, Benjamin E. Henty, Daniel D. Stancil, Fan Bai, and Priyantha Mudalige,** *Mobile Vehicle-to-Vehicle Narrow-Band Channel Measurement and Characterization of the 5.9 GHz Dedicated Short Range Communication (DSRC) Frequency Band*, IEEE Journal on Selected Areas in Communications, vol. 25, no. 8, US-PA, Oct., 2007.

[13] **Öncü Ararat, Emre Kural and Bilin Aksun Guiveç,** *Development of a Collision Warning System for Adaptive Cruise Control Vehicles Using a Comparison Analysis of Recent Algorithms*, Intelligent Vehicles Symposium,

Tokyo, Japan, June,2006.

[14] **NHTSA**, *Development of an FCW Algorithm Evaluation Methodology With Evaluation of Three Alert Algorithms Final Report*, DOT HS 811 145, U.S. Department of Transportation, US-VA, June, 2009.

[15] **Yunpeng Wang, Wenjuan E, Daxin Tian, Guangquan Lu, Guizhen Yu, Yifan Wang,** *Vehicle Collision Warning System and Collision Detection Algorithm Based on Vehicle Infrastructure Integration*, IET Colletive Inspiration, Jilin University, Jilin, China, 2011.

[16] **Ronald Miller, Qingfeng Huang,** *An Adaptive Peer-to-Peer Collision Warning System*, Ford Research Lab and Washington University, US-WA, 2002.

[17] **Anurag D, Srideep Ghosh and Somprakash Bandyopadhyay,** *GPS based Vehicular Collision Warning System using IEEE 802.15.4 MAC/PHY Standard*, Indian Institute of Management Calcutta and University of Calcutta, Kolkata, India, 2008.

[18] **Ivan Stojmenovic,** *Geocasting with Guaranteed Delivery in Sensor Networks*, University of Ottawa, IEEE Wireless Communications, Ottawa, Canada, Dec., 2004.

[19] **James R. Clynch,** *Radius of the Earth - Radii Used in Geodesy*, Feb., 2006.

[20] **G. Cai,** *Chapter 2 - Coordinate Systems and Transformations*, Unmanned Rotorcraft Systems, Springer, London, UK, 2011.

[21] **Anders O. Persson,** *The Coriolis Effect: Four centuries of conflict between common sense and mathematics, Part I: A history to 1885*, History of Meteorology 2, Department for research and development Swedish

Meteorological and Hydrological Institute, Norrköping, Sweden, 2005.

[22] **Chiu-Feng Lin, A. Galip Ulsoy and David J. LeBlanc,** *Vehicle Dynamics and External Disturbance Estimation for Vehicle Path Prediction*, IEEE Transactions on control Systems Technology, vol.8, no. 3, May, 2000.

[23] **Carlo Ferraresi, Giuseppe Quaglia and Giuseppe Gherlone,** *Real Time Trajectory Generation for Vehicle Position Control*, Department of Mechanics - Politecnico di Torino, IEEE ICIT02, Bangkok, Thailand, 2002.

[24] **Danwei Wang and Feng Qi,** *Trajectory Planning for a Four-Wheel-Steering Vehicle, Part I: A history to 1885*, School of Electrical and Electronic Engineering, Nanyang Technological University, International Conference on Robotics & Automation, Seoul, Korea, May, 2001.

[25] **Bruno Moreno Dtuck e Hélio Koiti Kuga,** *Medindo distâncias através de um único recetor GPS*, INPE-12331NTC/321, Instituto Nacional de Pesquisas Espaciais, Ministério da Ciência e Tecnologia, São Paulo, Brasil, 2005.

[26] **ENGIVIA e LNEC,** *Norma de Traçado Revisão*, Disposições Normativas, Instituto de Infra-Estruturas Rodoviárias IP, Lisboa, Portugal, Nov., 2010.

[27] **Nelson Cardoso,** *HEADWAY Platform User Manual*, Instituto de Telecomunicações, ver. 1.1, Aveiro, Portugal, Sep., 2011.

[28] **Ka-Ro electronics GmbH,** *TX27 datasheet*, Aachen, Germany.

[29] **element14,** *Quick Start Guide, The Raspbery Pi – Single Board Computer*, UK.

[30] **Broadcom Europe Ltd,** *BCM2835 ARM Peripherals*, Boardcom Corporation, Cambridge, UK, Feb., 2012.

[31] **Intelligent Transportation Systems Committee,** *IEEE Trial – Use Standard for Wireless Access in Vehicular Environments (WAVE) – Security Services for Application and Management Messages*, IEEE Std 1609.2, US-NY, July, 2013.

[32] **WABCO,** *Braking Deceleration and Projection, Basic Trainning 23*, 2012. Disponível em http://inform.wabco-auto.com/intl/pdf/815/00/57/8150100573-23.pdf [Acedido em 15-01-2013]

[33] **Atif Mehmood and Said M. Easa,** *Modeling Reaction Time in Car-Following Behaviour Based on Human Factors*, Ryerson University, International Journal of Engineering and Applied Sciences 5, Toronto, Canada, Feb.,2009.

[34] **OGP Publication,** *Coordinate Conversion and Transformations including Formulas*, OGP Publication 373-7-2 – Geomatics Guidance Note, no 7, part 2, July, 2012.