Escola Politècnica Superior d'Enginyeria de Manresa

Universitat Politècnica de Catalunya

Planificació de trajectòries per esquivar obstacles

Treball Fi de Grau

Grau en Enginyeria Electrònica Industrial i Automàtica

Autor: Roger Vilaró Pacheco

Tutora: Teresa Escobet Canal

Convocatòria: Octubre 2016

Resum

Aquest treball s'emmarca dins el projecte Elektra, que tracta sobre el desenvolupament d'un vehicle elèctric autònom completament automatitzat. Aquest vehicle està ubicat al Centre de Visió per Computador (CVC) i hi col·labora el centre de recerca CS2AC-UPC. El treball realitzat s'ha centrat en un estudi teòric i simulat de les accions que ha de fer el vehicle per evitar col·lisions.

El comandament d'aquests vehicles es realitza mitjançant diferents llaços de control. El control més bàsic (baix nivell) és el seguiment d'una trajectòria definida mitjançant l'angle d'orientació rodes del vehicle i la velocitat de circulació. En un nivell superior, es troba el planificador de trajectòries, aquest consta de dos subnivells: el nivell que anomenem global i el local. El nivell global consisteix en definir una sèrie de punts ('*waypoints*') o coordenades que marquen el trajecte a realitzar. El nivell local té per objectiu definir una trajectòria realitzable entre els punts del nivell global. Aqueta trajectòria s'haurà de modificar en el cas de que un obstacle impedeixi el pas del vehicle.

Aquest treball final de grau s'ubica en aquest últim cas, es tracta de cercar algoritmes, que a partir de la informació del vehicle i de l'entorn, com pot ser la distància i la mida d'un obstacle, es calculi una trajectòria alternativa per evitar la col·lisió o superar l'obstacle, i, un cop superat, tornar a la trajectòria global.

Per poder validar l'algoritme desenvolupat es disposa de dos entorns de simulació. Un és el Matlab/Simulink que s'ha utilitzat per desenvolupar el codi, i l'altre és un entorn de simulació en 3D anomenat *Unity,* que permet, a partir d'una programació preestablerta, afegir el codi i els algoritmes necessaris per simular el comportament de vehicle en presència d'obstacles. Un cop simulat i comprovat el correcte funcionament, un equip del Centre de Visió per Computador adaptarà el codi per poder-lo introduir al vehicle i veure com reacciona al espai real.

Paraules clau: vehicle autònom, planificació, control, simulació, evitar obstacles, Unity, Matlab

Abstract

This project is included in the Elektra's project, which consists in the development of an automated electric vehicle. This car is located at the *Centre de Visió per Comportador* (CVC) and it is developed with the collaboration of the research centre CS2AC-UPC. This work is focused on a theoretical and simulated study of the actions to avoid collisions.

The vehicle's regulation will be created by different control loops. The most basic control (low level) is the keeping of a trajectory defining the wheels orientation angle and the vehicle speed. In the higher level, there is the path planning, which consists of two sub-levels: the global level and the local level. The global level defines a group of waypoints or coordinates which determine a specific route. The other one, the local level, defines the trajectory between two waypoints. This trajectory will be modified if an obstacle blocks the predefined route.

This final project is based on the last case; the aim is to find algorithms which can use the vehicle and environment information, such as distance or obstacle size, in order to calculate an alternative path and avoid the collision or an obstacle and, once they are avoided, getting back to the global trajectory.

In order to validate the developed algorithm, two kinds of simulators were used. The first one was Matlab, which was used to develop the code, and the other one was a 3D environment program, called Unity, which from a pre-established programming, adding the necessary code and the algorithms we can simulate the behaviour of the car with the obstacle presence. Once the proper functioning has been simulated and checked, one team from the *Centre de Visió per Computador* will adapt the code in order to introduce it to the vehicle and check how it reacts in the real environment.

Key words: autonomous vehicle, planning, control, simulation, avoid obstacles, Unity, Matlab.

Índex apartats

Índex Figures

Índex Equacions

Índex Taules

1. Introducció

El treball fi de grau realitzat s'emmarca en un projecte de col·laboració coordinat pel grup de recerca ADAS ('*Advanced Driver Assistance Systems*') del Centre de Visió per Computador (CVC). Aquest centre està ubicat al campus de la Universitat Autònoma de Barcelona, en el que participa el centre de recerca CS2AC (Centre de Recerca en Supervisió, seguretat i Control Automàtic) amb el suport d'alumnes de diferents titulacions de grau i màster de la Universitat Politècnica de Catalunya.

1.1 Context del projecte

El projecte té per objectiu realitzar el control autònom d'un vehicle de tal manera que, sense la intervenció humana, pugui desplaçar-se per un entorn urbà de forma segura. L'objectiu final seria el de desenvolupar sistemes avançats d'ajuda a la conducció que permetin automatitzar i adaptar vehicles perquè siguin més segurs i pugin oferir una conducció més segura.

Per la realització d'aquest projecte es disposa d'un vehicle equipat amb una sèrie de càmeres que permeten interpretar el medi on es troba el vehicle; un IMU (*Inertial Measurement Unit*), el qual consta d'un acceleròmetre i un giròscop, el primer ens dóna informació de les acceleracions linears i angulars del vehicle, i, el segon, serà emprat per mantenir un punt angular absolut de referència. També disposa d'un GPS i una antena GPS, utilitats per ubicar la posició del vehicle; i encòders a cada una de les rodes, emprats per calcular velocitats.

Per a l'assoliment de l'objectiu global del projecte cal realitzar una sèrie de tasques, que a nivell de desenvolupament s'han agrupat en el conjunt mòduls mostrats en la Figura 1.

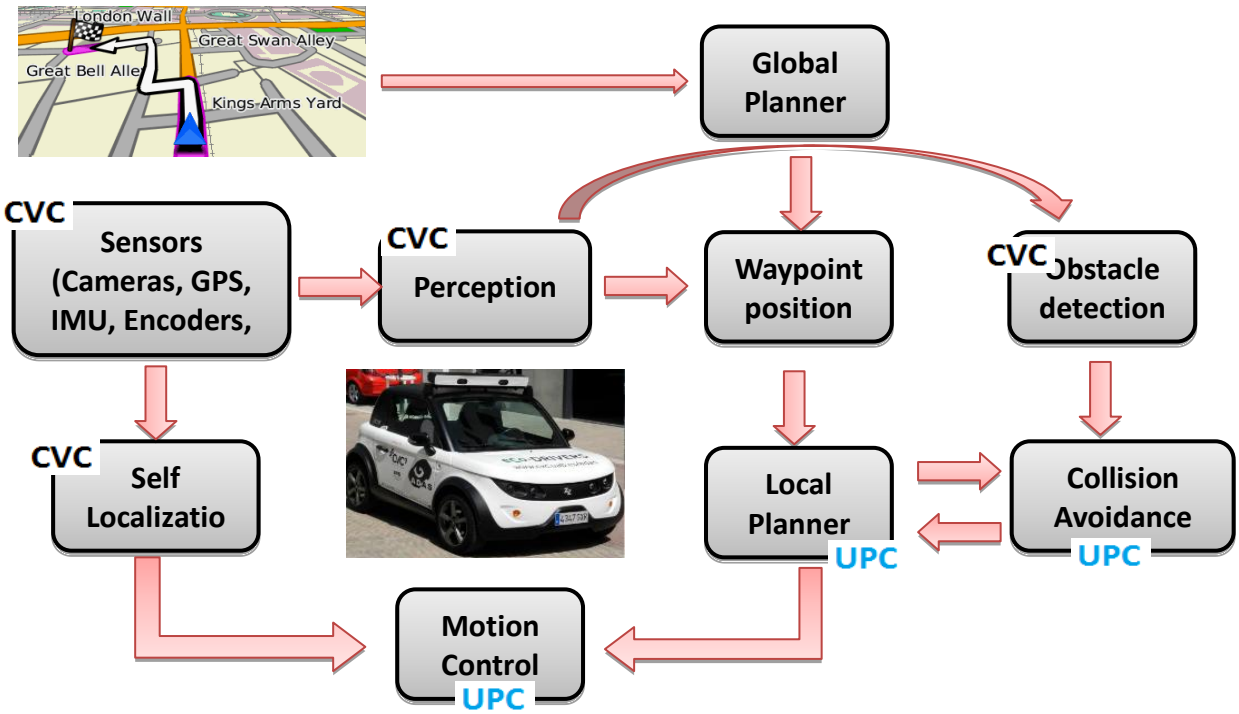


Figura 1. Tasques a realitzar per a l'assoliment del projecte

L'objectiu de cada mòdul seria:

1. Punt d'origen i destí + *Global Planner*: té per objectiu definir el punt d'inici i final del trajecte, i determinar la ruta a seguir.
2. *Waypoint position decision*: aquest ha de decidir la col·locació de diferents punts que ens ajudin a definir la trajectòria del vehicle. Els '*waypoints*' normalment es col·locaran allà on hi ha canvis d'orientació o en trossos llargs on la orientació no varia per observar si el vehicle continua per el lloc adequat.
3. *Local Planner* : l'objectiu d'aquest és generar subpunts entre els *waypoints* definits en el punt 2, cada punt tindrà associat una referència d'orientació, velocitat i acceleració, explicat a [1] i [15].
4. *Motion Control*: a partir de les referències de velocitat, acceleració, orientació i coneixent la posició, aquest mòdul tindrà per objectiu comandar el vehicle per assolir el següent *waypoint*. S'ha treballat amb els models [1] i [12].
5. *Self localitzation*: gràcies al GPS i l'antena del vehicle, es posicionarà el vehicle al punt exacte de la trajectòria que volem seguir.
6. *Sensors + Perception*: aquest mòdul ha d'extreure i processar la informació obtinguda dels sensors del vehicle: velocitats, acceleracions i ubicació el vehicle, també hauria d'obtenir informació de seu entorn, on *Perception* té per objectiu diferenciar entre dues possibles situacions:

   6.1 El vehicle pot dirigir-se sense problemes cap al següent punt local, aquest cas es dóna quan no es detecten obstacles en la trajectòria de vehicle. Donat aquest cas saltem al punt 4.

   6.2 El vehicle no es pot dirigir al següent punt local, ja que s'ha detectat un obstacle en la seva trajectòria. En aquest cas saltem al mòdul 7.
7. *Obstacle detection + Collision Avoidance*: aquest mòdul sols s'activa quan es detecta un obstacle. En aquestes circumstàncies cal fer un estudi del tipus d'obstacle amb que ens trobem i decidir com actuar per tal d'evitar la col·lisió amb aquest. La informació extreta de l'estudi es transmet al *Local Planner* (punt 3) per poder planificar de nou l'acció a realitzar, ja sigui frenar, girar bruscament, girar suaument, continuar el trajecte a velocitat inferior, ... En aquest punt caldrà decidir si es pot continuar seguint la trajectòria pre-definida per el *global planner* o modificar-la. Idees mostrades a [4], [7] i [13]

Per dur a terme el mòdul 6, es disposa d'un algoritme d'anàlisi de l'entorn que cada 100 ms és capaç de processar les imatges captades amb les càmeres i la informació provinent del IMU del vehicle. Aquest algoritme avalua les opcions comentades en els punts 6.1 i 6.2, donant informació sobre si l'espai navegable està lliure d'obstacles, i sinó ho està, proporciona informació sobre l'obstacle detectat.

1.2 Objectius

El treball fi de grau desenvolupat es centra en el mòdul 7 explicat anteriorment. Concretament, utilitzant la informació procedent de l'algoritme d'anàlisi de l'entorn, caldrà processar-la i planificar localment les accions que el vehicle haurà de realitzar davant de la presència d'obstacles. En el treball s'estudia quines accions realitzar en la presència d'obstacles estàtics i dinàmics.

La primera part del treball realitzat es centre en el cas d'obstacles estàtics o objectes sense moviment. Per l'estudi es consideren dues variables: la posició del obstacle (lluny o a prop) i la velocitat de desplaçament del vehicle, això permet calcular el temps de col·lisió, planificant dues accions diferents:

- Si estem a una distància llunyana es planifica una frenada suau
- Si estem a prop, caldrà planificar una frenada abrupta.

En el cas de que l'objecte estigui a una distància llunyana, caldrà avaluar si és factible planificar una nova trajectòria vorejant l'obstacle o senzillament aturar el vehicle. Aquesta seria la segona part del treball.

1.3 Motivació

La societat en que vivim està evolucionant cap a un món on les accions quotidianes seran més fàcils gràcies a la tecnologia.

Als darrers anys, la industria automobilística està realitzant grans avenços per garantir la seguretat i confort de les persones. Un dels futurs escenaris en aquest món serà la normalització i regulació dels vehicles autònoms a les nostres carreteres, on no serà necessària l'acció humana per poder desplaçar-te.

Resulta interessant formar part d'un grup de recerca on s'està investigant i desenvolupant un vehicle autònom. Per una part, tenim que és un projecte innovador i pioner a l'estat espanyol, en plena fase de desenvolupament. D'altra banda, formar part d'un grup de recerca i investigació t'ajuda a comprendre les diferents fases en que passa un element innovador abans no s'ha fet real per a tothom.

Un dels coneixements que espero ampliar i millorar amb la realització d'aquest treball final de grau és el de control i planificació. Aquests temes sempre els he trobat interessants, i, de cara a un futur, em poden ser útils per la realització d'un màster o en àmbit laboral.

1.4 Descripció parts del TFG

Aquest treball final de grau es divideix en les parts següents:

- Planificador de trajectòries: es realitza una explicació del que és un planificador i control de trajectòries i s'explica el funcionament general dels programes de simulació utilitzats
- El vehicle i la seva simulació: s'expliquen les característiques del vehicle elèctric i es comenten breument els codis utilitzats per la simulació en els dos programes, *Unity* i *Matlab*
- Aturada del vehicle davant d'un obstacle: en aquest apartat es desenvolupen una sèrie d'equacions que ens permetran realitzar una aturada del vehicle abans d'impactar contra un obstacle que es trobi a la seva trajectòria
- Aturada i maniobra per esquivar l'obstacle: plantejament teòric per realitzar maniobres per evitar la col·lisió i seguir amb la ruta planificada
- Futurs treballs: llista de futurs projectes relacionats amb el desenvolupament d'aquest treball que es poden dur a terme per continuar amb la recerca relacionada amb aquest tema
- Impacte social i mediambiental: comentaris del impacte que pot tenir el desenvolupament de vehicles elèctrics autònoms a la nostra societat i mediambientalment.
- Conclusions: recull de conclusions i impressions de la realització d'aquest treball final de grau

2. Planificador de trajectòries

Un planificador de trajectòries és un algoritme que permet convertir la descripció d'alt nivell, com pot ser desplaçar-se d'un punt a un altre, en una seqüència detallada de punts, anomenats *waypoints,* que permetin executar-la. En aquest capítol exposarem els trets característics de la planificació de trajectòries així com l'eina de simulació sobre la qual treballarem per testejar els algoritmes desenvolupats.

2.1 Característiques bàsiques d'un planificador de trajectòries

Per realitzar una bona planificació de la trajectòria, explicada al articles [1] [15], es requereixen complir tres tasques: la planificació de camins lliures d'obstacles, la detecció de possibles col·lisions, i, les accions a realitzar per evitar les possibles col·lisions. Per dur a terme les tres tasques necessitem resoldre una sèrie de problemes:

- Ubicació del vehicle respecte el seu entorn
- Coneixement de l'estructura de l'entorn (mapes)
- Capacitat d'actualitzar dades de l'entorn (sensors)
- Generació de trajectòries realitzables a partir de la informació de l'entorn
- Execució de les trajectòries programades evitats col·lisions eventuals

La primera tasca, la **planificació de camins**, es refereix al conjunt d'algoritmes dissenyats per obtenir trajectòries lliures d'obstacles, que pot efectuar el vehicle. Generalment es tracta de resoldre un problema d'optimització minimitzant un criteri relacionat amb la velocitat del vehicle o acceleracions, adaptant-lo a les limitacions i necessitats del vehicle, tenint en compte que la trajectòria ideada pot veure's alterada per elements que facin impossible la realització del camí dissenyat .

La **detecció de possibles col·lisions** pretén detectar els obstacles que fan modificar la trajectòria. S'estudia la possibilitat de col·lisió amb algun element, en funció de la trajectòria, la velocitat i els objectes del entorn. Es realitza la detecció a partir de les dades que ens proporcionen els sensors del vehicle i les dades del entorn.

La última tasca, **evitar possibles col·lisions**, consisteix en un conjunt d'operacions encaminades a la generació d'accions per evitar col·lisions quan detectem un obstacle o element que dificulta la trajectòria. Dins d'aquesta tasca és on es desenvoluparà gran part d'aquest treball.

En resum, un planificador genera una trajectòria on tenim una seqüència d'espais lliures que uneixen els punts d'origen i destí sense que s'envaeixi la zona de seguretat dels obstacles.

Per fer-nos una idea d'una planificació entre dos punts podem observar la Figura 2.1, on veiem els waypoints en forma de esferes de color blau cel i un puntejat de blau fosc, que indica la trajectòria ideonea pel vehicle fins el següent waypoint. A la Figura 2.2 es mostra el proces en que, quan el vehicle ha detectat un obstacle, es realitza una maniobra evasiva per evitar la col·lisió i seguir amb el recorregut plestablert.

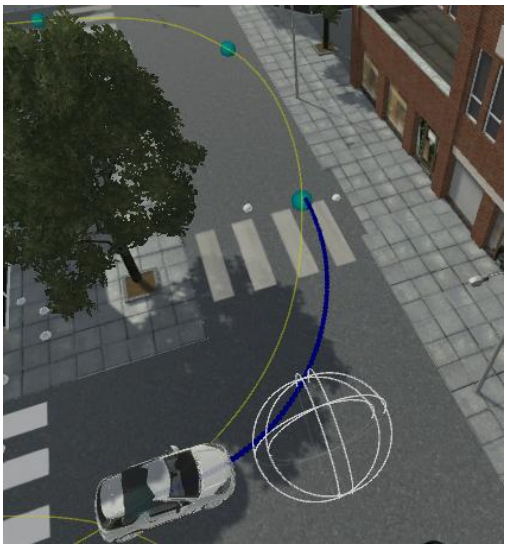


Figura 2.1 Trajectòria planificada (blau fosc) entre dos punts (blau cel)

Figura 2.2 Exemple d'evitar una col·lisió

Fins ara ens hem referit a l'entorn com l'espai per on ens movem, podríem dir que l'entorn en que treballarem es divideix en dos subgrups:

- **Zones lliures**: aquests espais no estan ocupats, i, per tant, el vehicle pot circular i ocupar aquestes zones per planificar trajectòries.
- **Zones ocupades**: aquestes parts són llocs ja ocupats on no es pot circular ni realitzar cap maniobra.

El nostre vehicle es mourà sempre per les zones lliures, aquestes zones les definirem a partir de les càmeres i sensors del que disposa el vehicle. D'aquesta manera podem identificar l'entorn, i a partir d'uns algoritmes de l'anàlisi de l'entorn realitzats pel CVC, determinar les zones per on es podrà circular.

2.2 Entorn de simulació

Per fer una part del treball s'han utilitzat entorns de simulació que ens han permès implementar i comprovar si els algoritmes desenvolupats assolien les expectatives de cada cas, en funció del que es demana.

Els dos entorns de programació han sigut:

- ***Matlab*** versió **R2015a,** enllaç web [11]
- ***Unity***: al inici del treball final de grau amb la versió 5.0.1, i , al final, versió **5.1.3**, enllaç web [16]

La principal diferència entre aquests dos programes és l'entorn de simulació que generen. Per la simulació amb el *Matlab* utilitzem un espai de dues dimensions (x,y), en canvi, amb el *Unity* podem generar espais tridimensionals (x, y, z) tot i que el moviment del vehicle es realitza en un pla de dos dimensions.

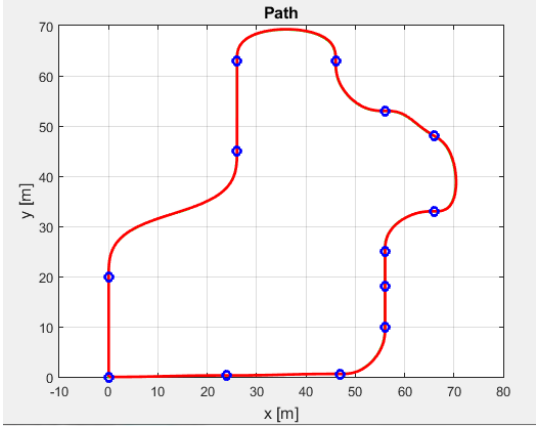


Figura 2.3 Exemple ruta amb Matlab

Figura 2.4 Zona lliure del Unity

Tal com es pot observar a les Figures 2.3 i 2.4, el *Unity* resulta més atractiu ja que ens dóna una visió molt més real respecte el *Matlab*. A més a més, el *Unity* ens permet l'opció de incloure més variables a l'espai per acostar-nos més a casos hipotètics reals; poden incorporar edificis, vianants, vehicles...

La base de funcionament dels dos programes és molt similar, per resumir-ho, a grans trets, podríem dir que consta dels següents passos, descrits de forma esquemàtica a la Figura 2.5:

1) Definim la ruta que volem que realitzi el vehicle a partir de *waypoints*. Aquests *waypoints* consten de una coordenada (x,y) i una theta que defineix la direcció que ha de seguir el vehicle en aquell punt.
2) El planificador ens genera els diferents punts per on passarà el vehicle entre dos *waypoints* consecutius, la trajectòria.
3) Detecció d'un obstacle en el transcurs de la trajectòria.
   - Si detectem un obstacle, es realitzarà una maniobra per evitar la col·lisió amb aquest
4) Si no detectem obstacle, quan assolim el *waypoint* desitjat, identifiquem quin és el següent *waypoint* i planifiquem la trajectòria que s'ha de seguir. I així successivament fins que arribem al destí de la ruta desitjada.

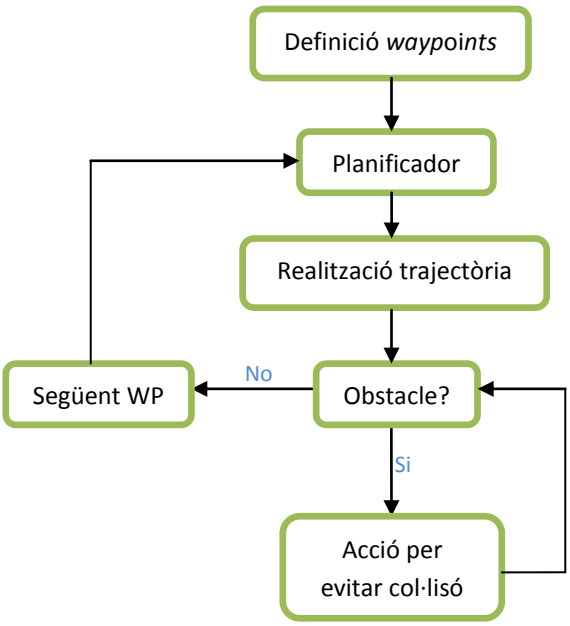


Figura 2.5 Passos guia del funcionament dels simuladors

A continuació s'explicaran els dos programes de simulació esmentats: el *Matlab* de forma més breu, ja que durant la carrera l'hem vist en més d'una assignatura, i, el *Unity*, més àmpliament per poder comprendre millor el seu funcionament general.

2.1.1 *MATLAB*

L'entorn de treball *MATLAB [11],* de l'empresa *MathWorks*, és un producte molt utilitzat en àmbit educatiu i en investigació. Durant la carrera s'ha utilitzat en diverses assignatures però amb diferents finalitats; sempre per qüestions de càlculs i simulacions, però aplicats a diferents àmbits de l'enginyeria.

La base de funcionament d'aquest programa és a través d'operacions amb vectors i matrius, d'aquí és on prové el seu nom (*MATrix LABoratory*). Disposa d'un ampli manual, que sempre és de gran ajuda, i una amplia comunitat d'usuaris i fòrums on es pot trobar molta informació útil per la realització de qualsevol projecte. També incorpora una gran quantitat de funcions matemàtiques i rutines, diversos programes i eines interactives de diferents àrees com poden ser les senyals, comunicacions, finances, filtres...entre d'altres. A més a més, té softwares paral·lels, com per exemple el *Simulink*, que permeten realitzar simulacions i anàlisis de programes més gràficament adaptant-lo a les teves necessitats.

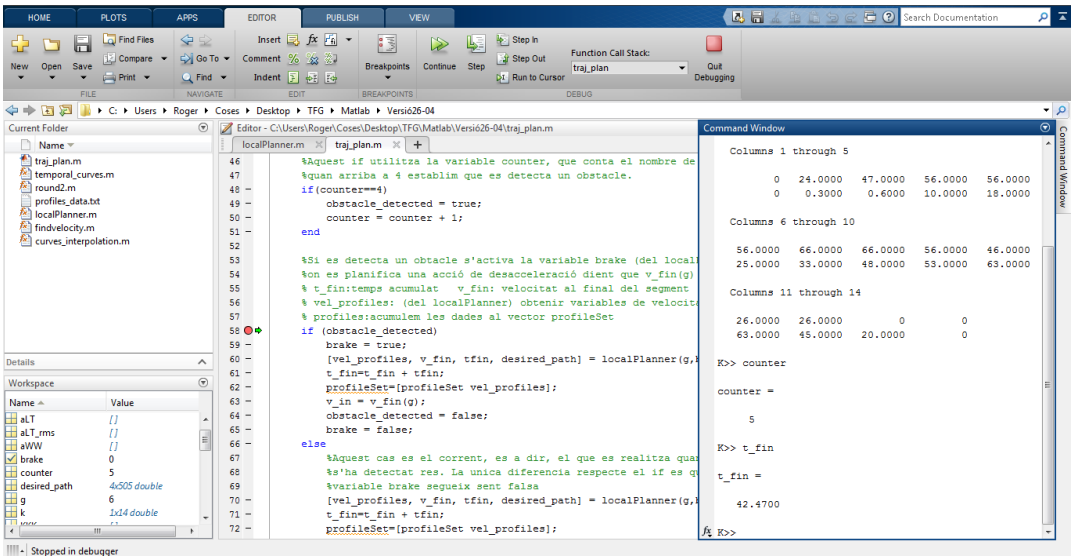


Figura 2.6 Exemple interfície Matlab

Quan obrim el programa *MATLAB* se'ns obre una finestra de línees de comandes (finestra de la dreta en la Figura 2.6), on podem introduir funcions i operacions, sempre que carreguem les llibreries necessàries per fer-ho. Tot aquest llenguatge que introduïm serà del tipus interpretat, és a dir, no s'ha de compilar i seguidament executar; l'avantatge que té aquest fet és que es pot modificar una rutina o funció sense la necessitat d'interrompre l'execució del programa. També ens permet obrir i executar programes escrits en altres llenguatges, com pot ser en *C*, sense haver-ho de modificar.

MATLAB és un programa molt visual, en el sentit que podem veure contínuament l'estat i valor de les variables del programa mitjançant l'àrea *workspace* (finestra inferior esquerra de la Figura 2.6) del entorn de treball, a més a més quan depurem un codi es poden marcar diversos *breakpoints* (punt vermell de la línia 58 de la Figura 2.6) per la millor supervisió del funcionament.

2.1.2 *Unity*

El *Unity* [16], anteriorment anomenat Unity3D, és una eina per la creació de videojocs en 2D i 3D utilitzada arreu del món. Als darrers anys la seva utilització ha crescut exponencialment ja que és possible desenvolupar projectes en 17 plataformes diferents; com poden ser Windows, Mac, Android i iOS, entre d'altres. Actualment conta amb un total de 2,9 milions de desenvolupadors registrats, això és degut a que la seva metodologia és molt flexible i té una gran versatilitat. A més a més, aquest programa ofereix una versió gratuïta on hi trobem casi totes les funcionalitats completes del programa, menys algunes específiques com la personalització de la pantalla d'inici, *Unity Analytics* per realitzar estadístiques del comportament dels jugadors al videojoc, ..., que no seran necessàries per el desenvolupament del projecte.

A la Figura 2.7 es mostra la interfície del programa, a la columna de l'esquerra veiem els diferents objectes que formen part de l'escena que es visualitza (part central), i a la columna de la dreta apareixen els components que configuren l'objecte seleccionat de la columna esquerra.

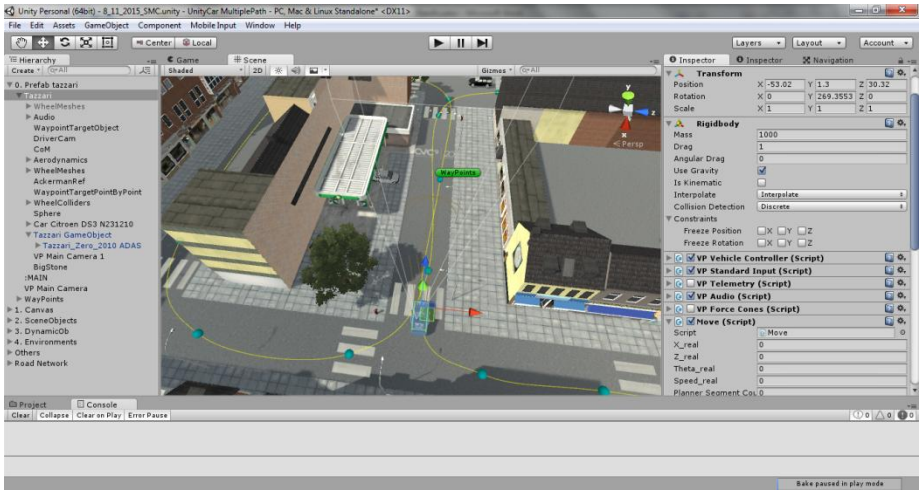


Figura 2.7 Interfície del Unity

A continuació s'explicarà més detalladament el funcionament i la programació per realitzar programes amb el *Unity*.

a) Llenguatge

Aquest programa disposa d'un editor gràfic amb una interfície molt intuïtiva que permet crear programes sense ser un expert en programació, tot i que el resultat serà limitat ja que els components que afegeixen una funcionalitat als objectes creats són molt bàsics. Per poder programar aquests objectes s'ha de generar un *script*; *Unity* ens ofereix la possibilitat de programar-los en llenguatge *Boo*, *Unityscript*(semblant al *Javascript*) i en *C#*.

- ***Boo***: llenguatge molt similar al *Python*. Es caracteritza per ser senzill d'utilitzar i poc complexa. Actualment s'està quedant obsolet ja que és més difícil crear codis d'alta complexitat amb la facilitat amb que es poden fer amb *C#* o *Unityscript*.

- ***Unityscript*:** llenguatge propi del programa *Unity*, és bastant simple però té l'avantatge que a la web de *Unity* hi ha molts tutorials i documentació útil que faciliten la programació.
- ***C#*:** ens permet crear llenguatges orientats a objectes, fet que facilita molt la creació de videojocs. Actualment és el més utilitzat per els desenvolupadors de *Unity*, i al ser un llenguatge extern al motor, et permet utilitzar-lo en altres motors.

Aquest últim llenguatge esmentat és amb el que es realitza la programació del programa simulat per el funcionament del vehicle autònom.

b) Entorn de desenvolupament

El *Unity* té integrat com a entorn de desenvolupament, *IDE* (*integrated development enviroment*), el *MonoDevelop* per poder obrir i modificar el conjunt d'instruccions o "*scripts*". Té una interfície molt intuïtiva, i disposa de totes les funcions relacionades amb el *Unity* necessàries, ja que està integrada amb el programa. Per executar aquest software i obrir els *script*s, únicament s'ha de fer un dobleclic a la interfície del *Unity* on apareixen les funcions, i ja s'obrirà el *MonoDevelop* on podrem depurar els *scripts* pas a pas mitjançant *breakpoints* o modificar el codi. Una vegada realitzat el programa, es guarda i es pot carregar automàticament al *Unity* per comprovar seguidament el funcionament.

c) Objectes de joc

Unity treballa principalment amb projectes i escenes. Un projecte és un videojoc o aplicació formats per una o més escenes. En un videojoc una escena és el nivell de joc, i per cada escena podem crear objectes o importar-los d'altres programes. Aquests objectes els anomenem "*Gameobjects*", i són tots els elements que formen la escena, més les llums i càmeres que s'utilitzen durant l'escena.

Cada *Gameobject* consta d'una sèrie de components amb els quals podem caracteritzar l'objecte atorgant-los la funcionalitat que han de tenir. Aquests components es poden modificar a traves de la interfície del *Unity* o a través del codi que tenen associat. Alguns dels components més importants i utilitzats per el desenvolupament del treball són els següents:

- ***Transform***: l'objecte és defineix amb una posició, orientació i mida dins l'escena
- ***MeshRenderer***: aquest component s'encarrega de renderitzar l'objecte en temps d'execució. Tots els elements que es veuen durant l'escena conten amb aquest component.
- ***Colliders***: són àrees que emboliquen al *GameObject*, determinen la mida del objecte dins l'escena. S'utilitzen per comprovar les col·lisions que es poden generar en una escena entre objectes. Hi ha diferents tipus de àrees: *BoxCollider* (àrea en forma de cub), *SphereCollider* (àrea en forma d'esfera), *CapusuleCollider* (àrea en càpsula) i *MeshCollider*

(col·lisionador de malla que s'adapta a la forma exacte del objecte, és molt precís però consumeix molts recursos).

- ***RigidBody***: permet assignar característiques que tindria el cos a la realitat; massa, gravetat, friccions, etc. S'utilitza principalment per els objectes que interaccionen amb altres objectes.
- ***Script***: ja esmentat anteriorment, és la part de codi afegit a l'objecte per donar-li ordres i configurar-lo, aquest codi no es pot escriure amb les eines bàsiques del *Unity*.

A la Figura 2.8 es mostra el *GameObject Tazzari*, és a dir, el vehicle utilitzat al programa de simulació. Veiem que per configurar el vehicle s'han utilitzat els components explicats; el *Transform* per situar-lo dins el mapa, el *MeshRenderer* per dotar l'objecte d'una carcassa de cotxe (un *Citroën DS3* en aquest cas), el *Collider* en forma de cub (línies de color verd que formen un cub), el *RigidBody* per proporcionar-li propietats físiques, i altres components, introduïts mitjançant diferents *scripts*.

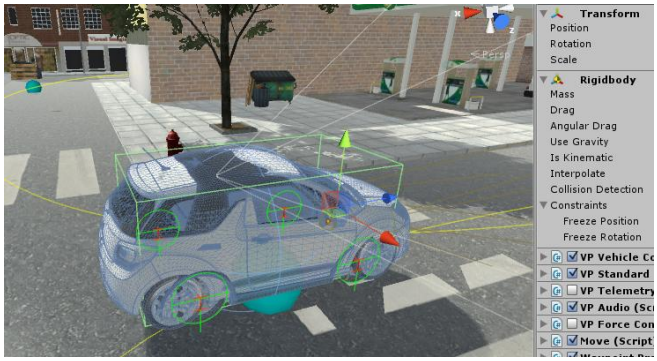


Figura 2.8 Imatge del vehicle al Unity

2.3 Control i planificador

Tal com s'ha comentat anteriorment, en aquest treball s'utilitzen tres bases: el control del vehicle, la planificació de les trajectòries i accions per evitar la col·lisió amb obstacles. Amb la coordinació del control i planificació, podem evitar col·lisions, i en aquest punt és on es centra el treball final de grau realitzat.

- **Control del vehicle [6, 15]:** aquesta part s'encarrega de controlar la velocitat i l'angle de gir del vehicle en funció de la ruta planificada. Amb ell es vol aconseguir que el vehicle segueixi la ruta planificada amb el mínim error i que tingui un comportament estable. La programació d'aquest control es basa amb el mètode de *Lyapunov* i *Sliding Mode Control* que ha realitzat l'Eugenio Alcalá a la seva tesis del màster [1], complementada amb informació dels articles [6] i [15].

- **Planificació de trajectòries:** aquesta tasca té per objectiu el de, donat uns punts, *waypoints*, o coordenades en l'espai, planificar la trajectòria òptima per desplaçar-se d'un punt a l'altre. El planificador determina la velocitat de desplaçament, les acceleracions i els girs del vehicle, que actuaran de senyals de referència al sistema de control del vehicle.

En un espai ideal, n'hi hauria prou amb aquests dos conceptes per moure'ns amb seguretat. Però aquest projecte està encarat cap al següent pas, l'espai on ens mourem mai serà ideal, i, hi ha la possibilitat que en la nostra trajectòria hi a pareixin obstacles que caldrà evitar. Per tant, caldrà modificar la trajectòria planificada per evitar la col·lisió i aconseguir que el vehicle autònom sigui segur.

Hi ha moltes situacions en que ens pot aparèixer un obstacle que obstaculitzi el pas del vehicle. Per exemple, a les ciutats i pobles caldrà evitar vehicles, vianants, elements urbanístics, ..., i a les carreteres convencionals ens podem trobar animals, elements naturals, obres, ..., a tots ells els considerarem obstacles, sense fer diferencies.

La idea principal és que ens podem trobar un obstacle que apareix de forma aleatòria en la trajectòria planificada, i caldrà fer un seguit d'accions per evitar l'impacte amb aquest element. En aquest treball es diferencia entre dos possibles casos:

- Evitar la col·lisió mitjançant la frenada
- Evitar la col·lisió mitjançant una frenada i una maniobra

En el primer cas, primer de tot s'ha d'identificar l'obstacle. Un cop s'ha detectat, procedim a fer l'estudi de la situació: definim a quina distància es troba el vehicle de i en funció de la velocitat actual es decidirà com es frena el vehicle per evitar l'impacte. Hi ha dues formes de frenar de forma abrupta (amb tota la potència del vehicle) o de forma suau.

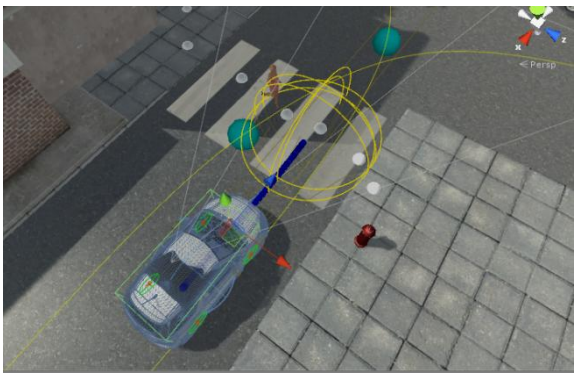


Figura 2.9 Exemple de detecció i definició del recorregut que tenim per aturar el vehicle

Cal senyalar que amb el vehicle real, la detecció d'obstacles es realitza mitjançant la manipulació adequada de la informació captada per les càmeres que el vehicle porta incorporades. Aquest programes seran els encarregats de proporcionar en tot moment la distància i la mida de l'obstacle detectat. La Figura 2.9 mostra un exemple del camp de visió de les càmeres del vehicle.

En el segon cas, es realitzen els mateixos passos que per el primer cas. La diferència es troba en que un cop s'ha frenat, es vol realitzar una maniobra per rodejar l'element i així poder continuar amb la ruta prefixada. Tal com podem veure a la Figura 2.10, que un cop detectat l'obstacle i s'ha frenat es defineix una nova trajectòria per poder-lo esquivar.

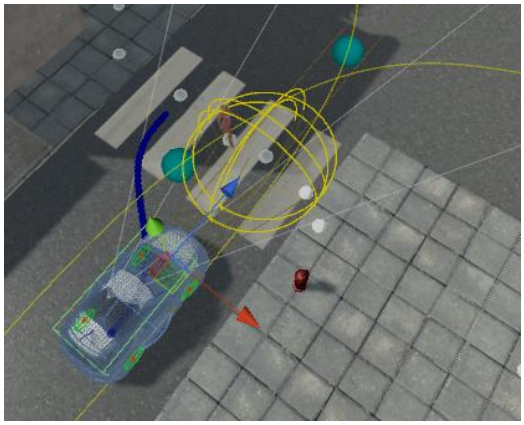


Figura 2.10 Detecció de l'obstacle i generació de trajectòria per esquivar-ho

3. El vehicle i la seva simulació

3.1 Característiques del vehicle real

El vehicle que s'utilitza per el projecte del grup de recerca ADAS i el CS2AC està ubicat a la Universitat Autònoma de Barcelona, a l'edifici del Centre de Visió per Computador. El model de vehicle és el Zero EM1, de la firma italiana *Tazzari*, que es pot veure a la Figura 3.1.

Figura 3.1 Vehicle Tazzari Zero EM1

Les característiques tècniques principals d'aquest vehicle són les següents:

Taula 3.1 Característiques del Tazzari Zero

Alimentació	Elèctrica
Bateries	Li-ion Fe de 13 kW de potència
Autonomia	150 km
Tipus de carrega	Carregador de 220V arribant al 100% en 9h
Motor	Motor trifàsic
Tracció	A les rodes de darrera
Rodes	175/55 R15
Velocitat màxima	100 km/h
Longitud x Amplada	2.88 x1.56 m
Altura	1.425 m

Per la realització d'aquest treball i projecte es posen unes limitacions al vehicle per poder realitzar els simulacres i simulacions amb una certa seguretat. Per tant, per realitzar els càlculs ens basem amb les dades que ens han proporcionat els grups ADAS i CS2AC.

La informació que hem de tenir en compte per poder realitzar els càlculs teòrics és la següent:

Taula 3.2 Paràmetres utilitzats per la simulació

Velocitat màxima	4.2 m/s
Distància màxima de detecció d'obstacles	12 m
Acceleració màxima	0.7 m/s^2

Es determina aquesta velocitat màxima ja que es considera segura i que no pot ocasionar danys si per algun motiu el cotxe es descontroles. El valor de la desacceleració per frenar també és de 0.7 m/s^2 ja que és la que ens proporciona el fre motor.

A partir d'aquesta informació i paràmetres, desenvoluparem els casos pràctics i teòrics que veurem als apartats 4 i 5.

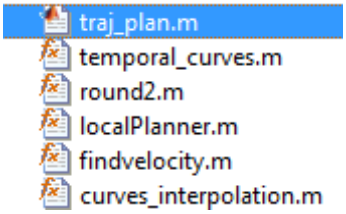
3.2 Simulació amb el *MATLAB*

Per poder fer la simulació amb el programa Matlab s'ha partit de la base dels scripts generats per membres del CS2AC i ADAS, que han estat complementats amb la generació d'un nou script que permet modificar la trajectòria un cop s'ha detectat un obstacle.

S'ha treballat amb els codis de *Matlab* mostrats a la Figura 3.2, els quals s'han classificat en tres blocs:

- Codis generats per el control i comportament del vehicle (*temporal_curves.m, round2.m, findvelocity.m, curves_interpolation.m)*
- Codis per la generació de trajectòries i velocitats entre waypoints (*localPlanner.m*)
- Codis per la realització de la trajectòria en funció de la posició i orientació dels waypoints (*traj_plan.m*)

Figura 3.2 Codis del Matlab



Per veure l'estructura d'aquests codis es pot consultar l'annex adjunt al final del treball. Per fer-nos una idea de com funciona el sistema; cada segment de durada de temps t, es divideix en cinc parts. Un cop s'ha dividit, es fan els càlculs d'acceleració, longitud, velocitat i empenta en funció del temps de cada part. A continuació, es fa una breu explicació de la funció de cada codi:

- ***temporal_curves***: a partir de les cinc parts crea els vectors de temps, velocitat, acceleració, longitud i empenta on s'emmagatzemen les dades de cada part.
- ***round2***: arrodoneix a dos decimals els valors que s'hi introdueixen per poder-los tractar correctament als altres codis.
- ***findvelocity***: a partir de les dades vectorials de velocitat, acceleració, temps, longitud i empenta de les cinc parts del segment, es calcula la velocitat amb que es desplaça el vehicle a partir de la longitud

- ***curves_interpolation***: realitza la partició del segment en cinc parts, dient el valor del temps per cada part
- ***localPlanner:*** en funció de la direcció i longitud de cada segment es calculen els subpunts (x,y) que van de *waypoint* a *waypoint* per on ha de passar el vehicle. En funció del traçat i girs, es determinen les velocitats i acceleracions linears i angulars amb les quals cal assolir els diferents punts tenint en compte les restriccions del vehicle
- ***traj_plan:*** aquest fitxer es genera per determinar els *waypoints* o posicions de l'espai que ha de seguir el vehicle durant el trajecte.

En general, el programa que es desenvolupa amb el *Matlab* funciona de la següent manera: primer es planteja una sèrie de *waypoints*, seguidament definim on hi ha l'obstacle, quan es detecta aquest obstacle el cotxe frena, i, quan deixem de detectar l'obstacle, el cotxe reprèn la marxa allà on s'havia quedat fins que arriba al final de la seva ruta, determinada per l'últim *waypoint*.

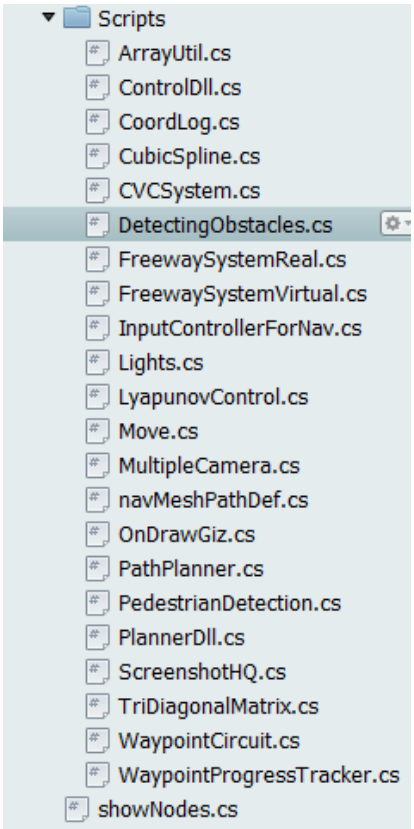
3.3 Simulació amb el *UNITY*

La simulació amb el *Unity* resulta ser molt més visual ja que veiem les escenes on es desenvolupa el vehicle. Tot el projecte desenvolupat amb aquest programa és molt més complex que amb el *Matlab* i, l'avantatge de treballar amb el *Unity* és que et permet treballar en un espai virtual molt semblant a la realitat.

Els codis desenvolupats al *Unity* són una evolució dels que hi ha al *Matlab.* Per treballar els *scripts* amb aquest programa es necessita un entorn de desenvolupament com és el *MonoDevelop*, tal com s'ha explicat abans. Cada element que es pot veure a l'escena té *scripts* associats, per tant, és un programa format per molts elements. Per la realització d'aquest treball només ens hem centrat amb els codis que formen les diferents parts del vehicle i l'entorn relacionat més pròxim.

Com que la quantitat de codis que intervenen és molt gran, només comentaré aquells que crec que ajudaran a comprendre millor el funcionament d'aturada davant d'un obstacle:

Figura 3.3 Codis del Unity



- ***ControlDll*** i ***LyaounovControl***: codis que realitzen el control del vehicle, semblant als *scripts* de control comentats amb el *Matlab*, però més detallats per un funcionament més òptim.

- ***WaypointCircuit*** i ***WaypointProgressTracker***: definició dels diferents *waypoints* que formen la ruta, i determinació del punt exacte on es troba el vehicle dins el recorregut.

- ***DetectingObstacles***: aquest *script* ens permet fer la detecció de l'obstacle mitjançant la simulació de la visualització de les càmeres que té el vehicle. Es generen dos esferes davant al vehicle i quan un objecte hi entra ens dóna la senyal d'obstacle detectat.

- ***PathPlanner*** i ***PlannerDll:*** semblant al *localPlanner* del *Matlab,* el *PathPlanner* es defineix la trajectòria entre *waypoints* mitjançant subpunts, tot definint les velocitats i acceleracions límits per poder realitzar els moviments. El *PlannerDll* sols s'utilitza per inicialitzar i definir variables utilitzades al *PathPlanner* i al *Move*.

- ***Move:*** aquest codi seria l'equivalent al *traj_plan* del *Matlab*. Es recorren els diferents punts del recorregut, comprovant contínuament si hi ha algun obstacle, i si ens apareix un obstacle realitzarem l'acció per evitar la col·lisió.

4. Aturada del vehicle davant d'un obstacle

4.1 Marc teòric

Seguint el model proposat per Lee i Kim [9], en aquest primer cas el que volem analitzar és la forma més bàsica per evitar la col·lisió amb un obstacle fix. Hi hauran dos maneres d'actuar:

- **Frenada abrupta:** si l'obstacle es troba a una distància molt propera, o si la velocitat del vehicle implica que en un curt espai de temps el vehicle pot col·lisionar.
- **Frenada progressiva:** si l'obstacle es troba a una distància suficient com per frenar de manera suau i la velocitat del vehicle permet una aturada a temps.

Les variables principals a tenir en compte en ambdós casos són les següents:

- Velocitat del vehicle (v_{vh})
- Posició de l'objecte o obstacle (x_o)
- Posició del vehicle (x_{vh})

A partir de les variables esmentades procedim a realitzar els següents càlculs, basant-nos amb els articles [2], [3] i [14]:

1. **Temps de col·lisió** $(\boldsymbol{t_c})$ contra l'obstacle, aquest es calcular a partir de les posicions i la velocitat del vehicle

$$x_i = x_o - x_{vh}\ (m) \tag{4.1}$$

$$\boldsymbol{t_c} = \frac{x_i}{v_{vh}}\ (s) \tag{4.2}$$

   a on x_i és la distància entre el vehicle i l'objecte.

2. **Temps de reacció del sistema** $(\boldsymbol{t_r})$, aquest temps té per objectiu predir el temps que passa des de que identifiquem l'obstacle fins que el nostre vehicle reacciona. Aquest temps depèn de la capacitat del computador per calcular les accions a realitzar. Per falta d'informació, assumirem que el temps de reacció del sistema davant la detecció d'un obstacle és de 100 ms.

3. **Temps de maniobra** $(\boldsymbol{t_m})$, aquest temps serà l'espai de temps que tindrem entre que el nostre cotxe reacciona i el temps de col·lisió calculat anteriorment. Per el càlcul d'aquests temps s'ha seguit l'estratègia documentada a [3], en el qual es proposa calcular el temps de maniobra en funció de les capacitats dinàmiques del vehicle per tant tenim dues formes de calcular el temps de maniobra, una (4.3) té en compte el temps de col·lisió calculat a (4.2) i l'altre té en compte l'acceleració màxima de frenada del vehicle, $a_{I,decc}$,

$$\boldsymbol{t_m} = t_c - t_r\ (s) \tag{4.3}$$

$$\boldsymbol{t_m} = \frac{v_{vh}}{a_{I,decc}}\ (s) \tag{4.4}$$

4. En funció de la velocitat del vehicle i el temps de maniobra calculat determinarem la millor opció:

   4.1 Si el t_m és molt petit -> frenada sobtada (indiferentment de la velocitat)

   4.2 Si el t_m és relativament gran, però la v_{vh} massa elevada -> frenada sobtada

   4.3 Si el t_m és gran i la v_{vh} dintre dels paràmetres estables -> frenada progressiva

A partir de la informació comentada al capítol 3 d'aquest treball s'han definit les característiques crítiques (Taula 4.1) que ens han de permetre calcular les diferents variables definides anteriorment.

Taula 4.1. Característiques crítiques del vehicle

Velocitat màxima	4.2 m/s
Distància màxima de detecció d'obstacles	12 m
Acceleració màxima	0.7 m/s^2

4.2 Estudi de casos

4.2.1 Desplaçament del vehicle a velocitat màxima

Analitzem una de les possibles situacions en les que ens podem trobar. Seria el cas en que el vehicle es desplaça a seva velocitat màxima i es detecta un obstacle a 12 m de distància..

- Velocitat del vehicle (v_{vh}) = 4,2 m/s
- Posició del vehicle (x_{vh}) = x
- Posició de l'objecte o obstacle (x_o) = x + 12
-

1) Emprant les equacions (4.1) i (4.2) Calculem temps de col·lisió

$$x_i = x_o - x_{vh} = (x + 12) - x = 12$$

$$\boldsymbol{t_c} = \frac{x_i}{v_{vh}} = \frac{12}{4{,}2} = 2.857s$$

2) Considerant un temps de reacció de $\boldsymbol{t_r} = 0.1s$, el temps de maniobra i la velocitat de desacceleració calculats amb les equacions (4.3) i (4.4) respectivament serà de:

$$\boldsymbol{t_m} = 2.857 - 0.1 = 2.757\ s$$

$$a_{I,decc} = \frac{4.2}{2.757} = 1.52\ {}^{m}/_{s^2}$$

Emprant l'equació del moviment rectilini d'un cos uniformement desaccelerat (4.5) podem calcular la distància necessària per realitzar la frenada:

$$x = x_0 + v_{vh} * t_m + \frac{1}{2} * a * {t_m}^2 \quad (s) \tag{4.5}$$

$$d_r = 0 + 4{,}2 * 2.757 - 0.5 * 1.52 * 2.757^2 = 5.8\, m$$

Segons la configuració del vehicle s'ha preestablert que tant la acceleració de frenada com l'acceleració d'avançament del vehicle seria de 0,7 m/s^2, per poder frenar suaument. Per adequar aquests càlculs al cas d'estudi, reformulem els dos últims càlculs per ajustar-nos al model real:

$$\boldsymbol{t_m} = \frac{v_{vh}}{a_{I,decc}} = \frac{4.2}{0.7} = 6s$$

$$d_r = 0 + 4{,}2 * 6 - 0.5 * 0.7 * 6^2 = 12.6\, m$$

D'aquest resultat es dedueix que si el vehicle es desplaça a la seva màxima velocitat configurada no tindrà prou espai per frenar suaument i acabarà impactant contra l'objecte. Per seguretat, en els següents casos d'estudi, considerarem que el vehicle s'ha d'aturar 1 metre abans d'arribar a l'obstacle.

4.2.2 Càlcul per realitzar frenada suau

En el primer cas d'estudi s'ha vist veiem que es produeix una col·lisió, en aquest cas d'estudi deixarem la variable v_{vh} com a incògnita i determinarem quina és la velocitat màxima a la que pot desplaçar-se el vehicle quan a 12m de distància es detecta un obstacle per poder frenar suaument:

- Sabem que l'acceleració de frenada és $a_{I,decc} = 0{,}7\ {}^{m}/_{s^2}$
- La distància màxima x a recorre és de $11\ m$ (tenint en compte el marge de seguretat de 1m)

$$\boldsymbol{t_m} = \frac{v_{vh}}{a_{I,decc}} = \frac{v_{vh}}{0.7}$$

Substituïm $\boldsymbol{t_m}$ a l'equació (4.5), obtenint-ne

$$d_r = 0 + v_{vh} * \frac{v_{vh}}{0.7} - \frac{1}{2} * 0.7 * \left(\frac{v_{vh}}{0.7}\right)^2 = \frac{v_{vh}^2}{0.7} - 0.5 * \frac{v_{vh}^2}{0.7}$$

$$d_r = 0.5 * \frac{v_{vh}^2}{0.7}\ (m)$$

Ja que $d_r = 11\ m$, deduïm que la velocitat del vehicle ha de ser de:

$$v_{vh} = \sqrt{\frac{d_r}{\frac{0.5}{0.7}}} = \sqrt{\frac{11}{\frac{0.5}{0.7}}} = 3.924\ {}^{m}/_{s}$$

Un cop s'ha realitzat aquest càlcul veiem que la velocitat màxima en que podem arribar a la detecció d'un obstacle quan està a 12 m és de $3.924\ {}^{m}/_{s}$.

A partir dels càlculs anteriors, realitzem el mateix càlcul per diferents distàncies entre el vehicle i l'obstacle per definir els punts de velocitats màxima. A la Taula 4.2, podem observar la relació entre les velocitats màximes que pot assolir el vehicle en funció de la distancia màxima que pot realitzar:

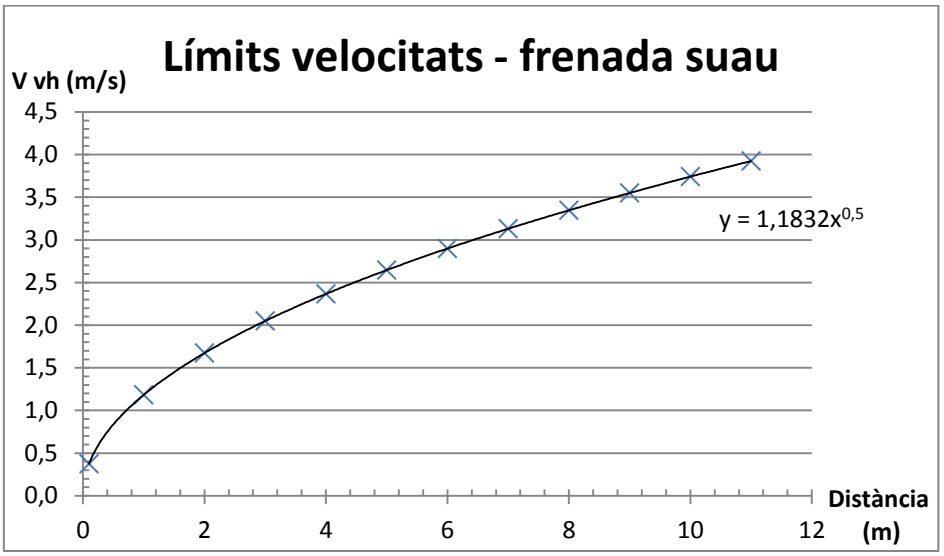
Taula 4.2 Distància de frenada - velocitat màxima per frenar suaument

Distància recorreguda (m)	Velocitat vehicle (m/s)
11	3,924
10	3,742
9	3,550
8	3,347
7	3,130
6	2,898

Distància recorreguda (m)	Velocitat vehicle (m/s)
5	2,646
4	2,366
3	2,049
2	1,673
1	1,183
0,1	0,374

Amb les dades obtingudes realitzem el gràfic que es mostra a la Figura 4.1. les X marquen els punts calculats. Amb totes aquestes i amb l'ajut d'un programa de regressió lineal cerquem la funció que millor s'ajusti a punts calculats.. De les diferents proves realitzades escollim aquella que presenti menys error i per tant se'ns aproximi més als punts, així l'equació generada tindrà el menor error possible respecte el càlcul de l'equació expressada anteriorment.

Figura 4.1 Gràfic distància-velocitat del vehicle en aturada suau



La funció que permet representar millor la tendència que s'aproximava més als punts ha estat la potencial, obtenint així la següent equació:

$$y = 1.1832 * x^{0.5}$$

a la y representa la velocitat del vehicle (v_{vh}) i la x la distància que recorre el vehicle per frenar de forma suau (d_{r-s}):

$$v_{vh} = 1.1832 * {d_{r-s}}^{0.5}$$

Per valorar si els resultats obtinguts mitjançant la tendència són correctes, cal comparar-los amb el valor real, $v_{vh} = \sqrt{\frac{d_{r-s}}{\frac{0.5}{0.7}}}$, calculant l'error absolut i l'error relatiu que hi ha entre les dos equacions.

Error absolut (Ea) = valor real – valor tendència

*Error relatiu (Er) = (error absolut / valor real) * 100*

Taula 4.3 Errors de la velocitat del vehicle en frenada suau

Distància recorreguda (m)	Velocitat vehicle real (m/s)	Velocitat vehicle tendència (m/s)	Error absolut (m)	Error relatiu (%)
11	3,924283	3,924230	0,000053	0,0013486
10	3,741657	3,741607	0,000050	0,0013486
9	3,549648	3,549600	0,000048	0,0013486
8	3,346640	3,346595	0,000045	0,0013486
7	3,130495	3,130453	0,000042	0,0013486
6	2,898275	2,898236	0,000039	0,0013486
5	2,645751	2,645716	0,000036	0,0013486
4	2,366432	2,366400	0,000032	0,0013486
3	2,049390	2,049363	0,000028	0,0013486
2	1,673320	1,673297	0,000023	0,0013486
1	1,183216	1,183200	0,000016	0,0013486
0,1	0,374166	0,374161	0,000005	0,0013486

Si estudiem la Taula 4.3 veurem que l'equació de la tendència potencial ens genera un error de centèsimes de mil·límetre per segon. Per tant, podem considerar l'error com a 0 ja que en l'entorn i a l'espai on es mou el vehicle ens suposa una variació de tant sols 0,3 mm en el pitjor dels casos, quan es va a màxima velocitat, tal com es demostra a continuació:

Cas de velocitat del vehicle real

$$\boldsymbol{t_m} = \frac{3.924283}{0.7} = 5.60612s$$

$$d_r = 11\ m$$

Cas de velocitat del vehicle a partir de la tendència

$$\boldsymbol{t_m} = \frac{3.92423}{0.7} = 5.60604s$$

$$d_r = 10{,}9997\ m$$

Finalment, després de comprovar que l'error que es genera és casi imperceptible, deixem l'equació en funció de d_{r-s} ja que ens serà més útil per la definició del codi:

$$d_{r-s} = \sqrt[0.5]{\frac{v_{vh}}{1.1832}} \tag{4.6}$$

Aquesta equació ens serveix per determinar el punt on el vehicle ha de realitzar una frenada suau.

4.2.3 Càlcul per realitzar frenada abrupta

A continuació,basant-nos amb l'idea proposada a l'article [8] sobre frenades d'emergència, analitzarem el cas que es requereix una frenada abrupta del vehicle. Per falta de dades teòriques, es van realitzar unes proves sobre el terreny i es va determinar que a màxima velocitat de 4,2 m/s el vehicle recorre 1,5 m. Partint d'aquesta informació calcularem l'acceleració de frenada $a_{I,decc-màx}$ que es genera:

$$\boldsymbol{t_m} = \frac{4{,}2}{a_{I,decc}}$$

Igual que en el cas anterior, substituïm en l'equació (4.5) el temps $\boldsymbol{t_m}$

$$x = 0 + 4.2 * \frac{4{,}2}{a_{I,decc}} - \frac{1}{2} * a_{I,decc} * \left(\frac{4{,}2}{a_{I,decc}}\right)^2$$

$$\mathrm{x} = \frac{4.2^2}{a_{I,decc}} - \frac{1}{2} * a_{I,decc} * \frac{4{,}2^2}{a_{I,decc}^2} = \frac{4.2^2}{a_{I,decc}} - \frac{1}{2} * \frac{4{,}2^2}{a_{I,decc}}$$

$$x = 0.5 * \frac{4{,}2^2}{a_{I,decc}}$$

Ja que $x = 1.5\ m$, l'acceleració de frenada ha de ser:

$$a_{I,decc-màx} = \frac{0{,}5 * 4{,}2^2}{1{,}5} = 5{,}88 \frac{m}{s^2}$$

Amb aquest càlcul definim que l'acceleració màxima de frenada del vehicle quan frena de forma abrupta és de **5,88 m/s²**. Igual que en el càlcul per realitzar la frenada suau, realitzem una taula, Taula 4.4, determinant les velocitats màximes que por assolir el vehicle en funció de la distancia màxima que recorre quan frenem abruptament:

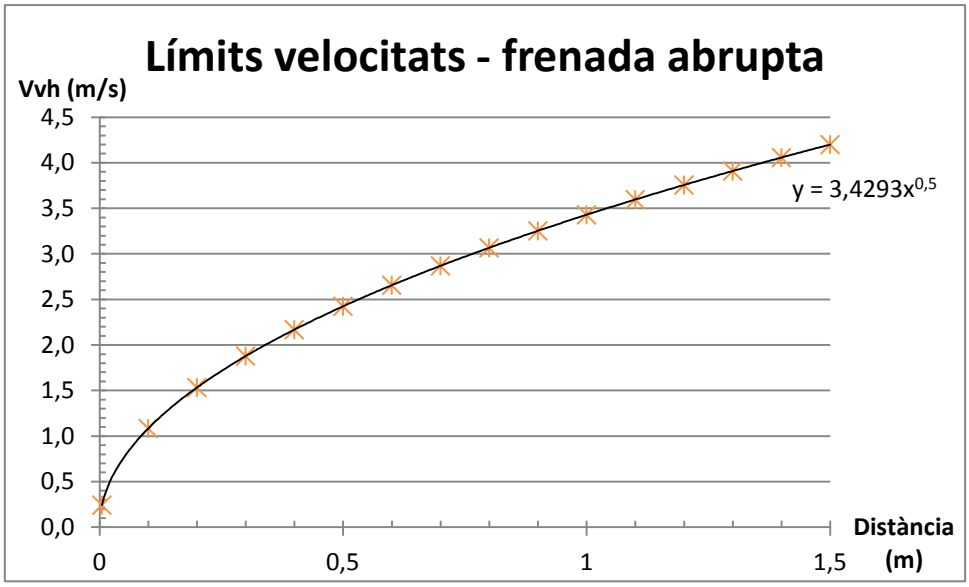
Taula 4.4 Distància de frenada - velocitat màxima per frenar abruptament

Distancia recorreguda (m)	Velocitat vehicle (m/s)
1,5	4,200
1,4	4,058
1,3	3,910
1,2	3,757
1,1	3,597
1,0	3,429
0,9	3,253
0,8	3,067

Distancia recorreguda (m)	Velocitat vehicle (m/s)
0,7	2,869
0,6	2,656
0,5	2,425
0,4	2,169
0,3	1,878
0,2	1,534
0,1	1,084
0,005	0,242

Amb les dades anteriors realitzem el gràfic mostrat en la Figura 4.2 amb l'objectiu de poder calcular la relació entre la velocitat del vehicle i la distància que recorre en funció de l'acceleració quan realitzem una frenada abrupta:

Figura 4.2 Gràfic distància - velocitat del vehicle en aturada abrupta



La funció que ha donat una millor aproximació és la funció exponencial amb:

$$y = 3{,}493 * x^{0.5}$$

Sabent que la y és la velocitat del vehicle (v_{vh}) i, la x, la distància de frenada de forma abrupta (d_{r-a}) que s'ha de fer, l'equació ens queda de la següent manera:

$$v_{vh} = 3.493 * {d_{r-a}}^{0.5}$$

Per valorar si els resultats obtinguts mitjançant la tendència són correctes, cal comparar-los amb el valor real, $v_{vh} = \sqrt{\frac{d_{r-a}}{\frac{0.5}{5{,}88}}}$, calculant l'error absolut i l'error relatiu que hi ha entre les dos equacions.

Error absolut (Ea) = valor real – valor tendència

*Error relatiu (Er) = (error absolut / valor real) * 100*

Taula 4.5 Errors de la velocitat del vehicle en frenada abruptament

Distancia recorreguda (m)	Velocitat vehicle (m/s)	Velocitat del vehicle tendència (m/s)	Error absolut (m)	Error relatiu (%)
1,5	4,200000	4,200018	-0,000018	-0,0004187
1,4	4,057585	4,057602	-0,000017	-0,0004187
1,3	3,909987	3,910004	-0,000016	-0,0004187
1,2	3,756594	3,756610	-0,000016	-0,0004187
1,1	3,596665	3,596680	-0,000015	-0,0004187
1,0	3,429286	3,429300	-0,000014	-0,0004187
0,9	3,253306	3,253320	-0,000014	-0,0004187
0,8	3,067246	3,067259	-0,000013	-0,0004187
0,7	2,869146	2,869158	-0,000012	-0,0004187
0,6	2,656313	2,656324	-0,000011	-0,0004187
0,5	2,424871	2,424881	-0,000010	-0,0004187
0,4	2,168871	2,168880	-0,000009	-0,0004187
0,3	1,878297	1,878305	-0,000008	-0,0004187
0,2	1,533623	1,533630	-0,000006	-0,0004187
0,1	1,084435	1,084440	-0,000005	-0,0004187
0,005	0,242487	0,242488	-0,000001	-0,0004187

Igual que en el cas anterior de frenada suau, a partir de la Taula 4.5, realitzem l'estudi de la variació de distàncies per veure si ho hem de tenir en compte per definir la funció de la distància recorreguda en frenada abrupta:

Cas de velocitat del vehicle real

$$\boldsymbol{t_m} = \frac{4{,}2}{5.88} = 0.714285\ s$$

$$d_r = 1{,}5\ m$$

Cas de velocitat del vehicle a partir de la tendència

$$\boldsymbol{t_m} = \frac{4{,}200018}{5.88} = 0{,}714289\ s$$

$$d_r = 1{,}500013\ m$$

Tal com es pot veure la variació és mínima, per tant no el tenim en compte. Finalment, definim l'equació de la distància que es recorre frenant abruptament que ens permet definir el punt d'inflexió:

$$d_{r-a} = \sqrt[0.5]{\frac{v_{vh}}{3.493}}(m) \tag{4.7}$$

Un cop obtingudes les dades i equacions, podrem generar el codi en funció d'aquests paràmetres per determinar en quins moments el cotxe ha de frenar de forma abrupta, de forma suau o, en el pitjor cas, no es pot evitar la col·lisió.

4.3 Algoritme de frenada suau o abrupta

El marc teòric explicat a la secció 4.1 s'ha concretat amb el següent codi:

```
%% Codi frenada suau o abrupta

if obstacle_detectat
    Vvh = Calcul Vvh;
    xi = Calcul distancia entre objectes;
    dr-s = Calcul dr-s;
    dr-a = Calcul dr-a;

    if (Vvh > 3.924)  &  ( xi>2.5)
        Frenada abrupta;

    elseif (Vvh < 3.924)
        if dr-s >= (xi-1)  &  dr-a <(xi-1)
            Frenada abrupta;

        elseif dr-s < (xi-1)
            Frenada suau;

        else
            IMPACTE;
        end

    else
        IMPACTE;
    end
end
```

Primer de tot, aquest codi s'inicialitza quan detectem un obstacle. Un cop detectat, calculem la velocitat del vehicle (v_{vh}), la distància que hi ha entre el vehicle i l'obstacle(x_i), i les distàncies que recorrerà el vehicle, emprant les equacions definides als apartat anteriors, en funció de si realitza una frenada suau (d_{r-s}) o abrupta (d_{r-a}).

Quan tenim la informació, plantegem els tres possibles casos que podrien succeir:

- Si la v_{vh} és superior a $3{,}924\ {}^{m}/_{s}$ i la x_i més gran que $2{,}5\ m$
    - Frenem de forma abrupta, ja que tal com s'ha demostrat anteriorment, frenant de forma suau no seriem capaços de parar el vehicle 1m abans de l'obstacle.

- Si la v_{vh} és inferior als $3{,}924\ {}^{m}/_{s}$
    - Si la d_{r-s} és més gran o igual que la $x_i - 1$ i la d_{r-a} és més petita que la $x_i - 1$
        - Frenada abrupta, ja que tenim l'espai suficient per frenar abruptament però no suaument.
    - Si la d_{r-s} és inferior a la distància a la $x_i - 1$
        - Frenada suau, ja que tenim prou espai.

- Si no es compleix cap de les dues condicions anteriors
  - Es realitza un frenada abrupta però la posició final del vehicle no respectarà el metre de marge o senzillament acabarà impactant amb l'obstacle, aquest cas l'anomenem $IMPACTE$.

- Si la v_{vh} és superior $3{,}924\ {}^{m}/_{S}$ i la x_i inferior a $2{,}5\ m$
  - Es frena de forma abrupta, però la posició final serà inferior a un metre o hi haurà una col·lisió, $IMPACTE$

4.4 Resultats

El codi desenvolupat s'ha testejat en dos entorns de simulació diferents, en *Matlab* i en *Unity*. Els resultats obtinguts es mostren en els següents apartats.

4.4.1 *MATLAB*

Tal com s'ha comentat anteriorment, el codi generat amb el *Matlab* està condicionat per els codis ja creats pels membres del CS2AC i ADAS.

Els que s'han modificat i generat per poder realitzar la simulació han estat els següents:

- ***localPlanner:*** aquest *script* planifica el recorregut que hi ha entre dos *waypoints* generant subpunts. A més a més, en funció de la posició i orientació dels punts determina les velocitats i acceleracions linears i angulars amb les quals cal assolir diferents punts tenint en compte les restriccions del vehicle, com són les velocitats i les acceleracions màximes. Aquest *script*, desenvolupat a la tesis de l'Eugenio, serà utilitzat per modificar la trajectòria del vehicle en funció de les velocitats i les acceleracions màximes d'arrencada i frenada que assoleix el vehicle.
- ***traj_plan:*** a partir dels *waypoints* definits es crea un bucle per recorres els diferents punts tot analitzant si hi ha obstacles o no durant la trajectòria.

Per fer la simulació s'ha desenvolupat el següent codi, anomenat *traj_plan.m,*desglossat per parts per una millor explicació:

- Primer de tot s'inicialitzen i es defineixen les variables que s'utilitzen, Figura 4.3. La majoria de variables són utilitzades pel *script localPlanner* que és el que s'encarrega de fer la planificació per assolir els diferents *waypoints* de la trajectòria.

Figura 4.3 Inicialització i definició variables Matlab

```
%%% Inicialització i definició de variables
Points = [xxx;yyy];
kkkk = [zeros(2,length(Points))];
v_fin = [zeros(1,length(Points))];
k = zeros(1,length(xxx));
kx=1.4;
ky=1.4;
v_in=0;
L=0;
ttt=[];
LL=[];
all=[];
aWW=[];
aLT=[];
aLT_rms=[];
KKK=[];
point=[];
XY=[];
desired_path=[];
t_fin=0;
Theta(1)=0;
profileSet=[];
obstacle_detected = false;
brake = false;
counter = 0;
obstacle=[];
tram fi = false;
```

- Determinem els *waypoints* i direccions que formen el recorregut, Figura 4.4:

Figura 4.4 Definició waypoints

```
%%% Exemple de trajectòria
xxx  = [0, 30, 65,     83,   65,   30,  0];
yyy  = [0,  0,  0,    -18,  -36,  -36, -36];
teta = [0,  0,  0,  -pi/2,   pi,   pi,  pi];

```

- Seguidament desenvolupem el codi per assolir els diferents *waypoints*. A la Figura 4.5 podem veure el codi desenvolupat.

  La primera sentència condicional *if* s'utilitza per determinar quan es detecta un obstacle o s'ha arribat al final de la trajectòria. En el cas de detecció d'obstacle, incrementem la variable *counter*, que ens determina el segment on es troba el vehicle.

  El segon condicional *if,* dintre del *for,* és el que s'utilitza per planificar la trajectòria entre els diferents punts:

  - Si es detecta un obstacle o s'ha arribat al final de la trajectòria activem la variable *brake* del *localPlanner*, Figura 4.6. En aquesta variable diem que la velocitat final del segment on es trobi, determinat per la variable *counter*, ha de ser $0\ ^{m}/_{s}$.
    A més a més, a partir de les dades del *localPlanner,* determinem el temps i velocitat on succeeix l'acció i emmagatzemem la velocitat per la posterior representació. Finalment, diem que ja no es detecta cap obstacle i s'ha de deixar de frenar pel pròxim cicle.
  - En el cas que no es detecta obstacle, s'ha de fer el mateix que en el cas anterior però sense les acció del *brake* ni la detecció d'obstacle. Afegim el *counter* per incrementar-lo ja que haurem superat un segment.

Figura 4.5 Codi desenvolupat en Matlab

```
for g=1:length(Points)-1

    if(counter ==2) | (counter ==5)
        if (counter ==2)
        obstacle_detected = true;
        counter = counter + 1;
        else
            tram_fi = true;
        end
    end

    if (obstacle_detected) | (tram_fi)
        brake = true;
        [vel_profiles, v_fin, tfin, desired_path] = localPlanner(g,brake,v_in, desired_path);
        t_fin=t_fin + tfin;
        profileSet=[profileSet vel_profiles];
        v_in = v_fin(g);
        obstacle_detected = false;
        brake = false;
    else
        [vel_profiles, v_fin, tfin, desired_path] = localPlanner(g,brake,v_in, desired_path);
        t_fin=t_fin + tfin;
        profileSet=[profileSet vel_profiles];
        v_in = v_fin(g);
        counter = counter + 1;
    end
end % (FOR)
```

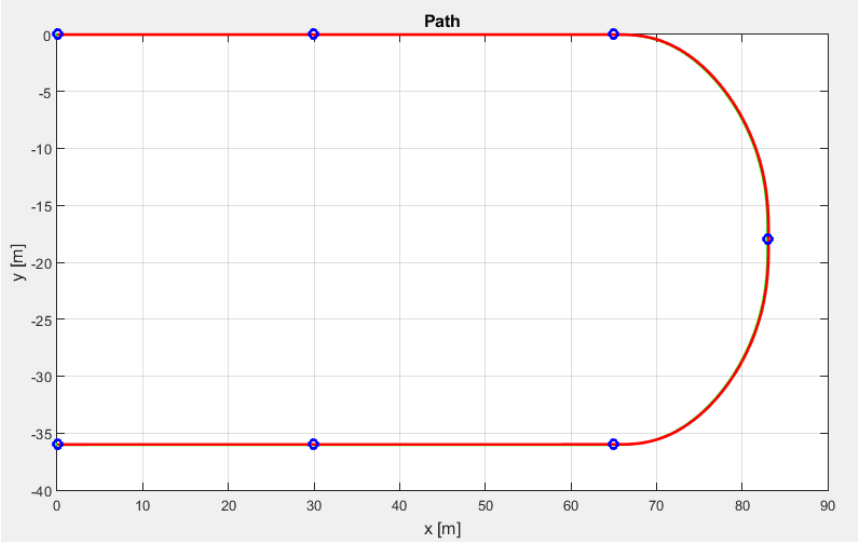
Figura 4.6 Variable brake del localPlanner

```
if (brake == 0)
    v_fin(g) = LLL/(tfin);
else
    v_fin(g) = 0;
end
```

A partir del codi anterior, desenvolupem dos gràfics per veure el comportament del vehicle a la simulació.

- A la Figura 4.7 veiem la trajectòria que segueix el vehicle en funció dels *waypoints* i direccions definits a la Figura 4.4.

Figura 4.7 Trajectòria testejada



- A la Figura 4.8 veiem el comportament del vehicle relacionant les velocitats que assoleix respecte el temps transcorregut. Les velocitats màximes i acceleracions del vehicle les determinem al *script* del *localPlanner*; així limitem i adaptem el vehicle en funció dels càlculs realitzats al marc teòric. Aquestes dades seran utilitzades pel control del vehicle. Tal com es pot veure a la figura, realitzem una frenada suau al final del tercer segment (*counter =2*) i quan s'arriba al final de la trajectòria. A continuació fem el càlcul per veure si la simulació d'aturada suau resulta igual a la teòrica calculada:

Simulació

$$a_{I,decc} = \frac{v_{vh}}{t_m} = \frac{2.791}{(30.27 - 26) - 0{,}1} = 0{,}67\ {}^{m}/_{s^2}$$

$$d_r = 2{,}791 * 4.17 - (0{,}5 * 0{,}67 * 4{,}17^2) = 5{,}81m$$

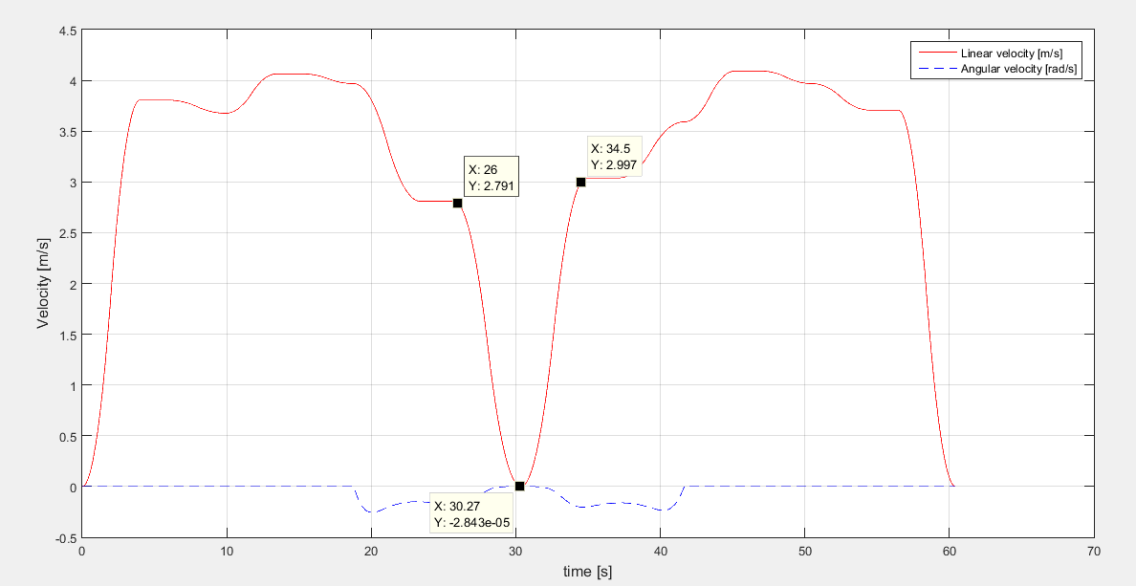
Teòrica

$$a_{I,decc} = 0{,}7\ {}^{m}/_{s^2}$$

$$d_r = 2{,}791 * 4.17 - (0{,}5 * 0{,}7 * 4{,}17^2) = 5{,}56m$$

Veiem que tenim un error de 0,25 m a la simulació, però estaria dins el rang d'error ja que tenim un marge de 1 metre abans d'impactar amb el vehicle.

Figura 4.8 Gràfic velocitat - temps en Matlab



Com s'ha comentat al principi, s'ha treballat a partir dels *scripts* desenvolupats per altres programadors. Aquest fet resulta un handicap ja que no es pot plasmar la idea teòrica desenvolupada. Per exemple, quan al sistema controlador i planificador introduïm que l'acceleració màxima és de $5.88\ {}^{m}/_{s^2}$ (per simular una frenada abrupta), es descompensa i el control i comportament del vehicle no segueix les especificacions de les lleis de control amb què funciona.

A més a més, no he estat capaç de simular la detecció d'un punt aleatori, ja que el *localPlanner* genera un centenar de punts desconeguts entre *waypoint* i *waypoint.* Degut al fet anterior i aquest, l'única simulació que he pogut fer ha estat la de la frenada suau, tal com s'acaba de demostrar.

Un altre aspecte a tenir en compte és el procés de control del vehicle, que segueix les lleis de *Lyapunov* i el *Slide Mode Control* i ens generen errors, tal com és mostra a la tesi [1] de la bibliografia.

4.4.2 *UNITY*

A l'apartat 3.3 d'aquest treball s'ha fet una breu explicació de com s'utilitza el *Unity* i els diferents codis que hi intervenen per poder fer la simulació.

En aquest cas, el CVC ha configurat les velocitats i acceleracions a partir de la informació teòrica utilitzada per simular, i no les he modificat. Amb el *Unity,* s'ha desenvolupat un codi pel planificador més avançat que el del *Matlab*, *plannerDll*. Aquest codi calcula una frenada progressiva fins a un punt determinat, en funció de si es troba un objecte o és el final del recorregut. És a dir, en funció de la distància entre dos objectes, es planifica una frenada progressiva, de la potència necessària per quedar aturat abans d'impactar.

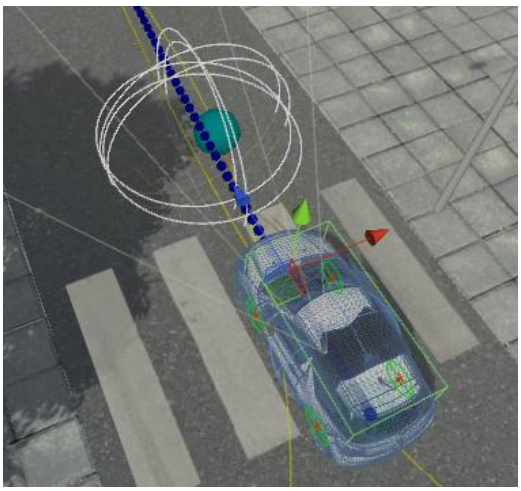
Aquest codi aplicat al *script Move* es mostra de la següent manera:

$$plannerDll.Compute(thetaA, zA, xA, thetaB, zB, xB, 1);$$

A partir de dos punts, el punt A, que és la posició inicial, i el punt B, la posició allà on es vol anar, es defineix la velocitat amb que s'ha d'arribar al punt B. Si el seu valor és 0, assolirà la velocitat estàndard en funció de les lleis de control, si el seu valor és 1, significa que s'ha d'arribar abans del punt B amb una velocitat de $0\ ^{m}/_{s}$.

La programació per la detecció d'obstacles s'ha fet treballant bàsicament amb dos *scripts*:

Figura 4.9 Sensor de detecció d'obstacles al Unity



- ***DetectingObstacles.cs:*** aquest *script* genera dos esferes davant del vehicle, que representen el sensor per detectar obstacles, Figura 3.3. Les dos esferes són de la mateixa mida, però no comparteixen el centre, una està més avançada que l'altre per poder facilitar la detecció, aquestes esferes tenen un radi de 4m ja que si es feien de 12m eren massa grans i detectaven elements que no afectaven a la trajectòria del vehicle. El sistema detecta un obstacle quan un element, *Gameobject*, es troba entre o al interior de les dos esferes. Quan es dona aquest cas, el sistema retorna que s'ha detectat un obstacle i es salta al *script Move.cs* per executar les ordres necessàries.

Figura 4.10 Codi per detectar obstacles

```
30          // If between the first sphere and second sphere is any obstacle that belong
31          if (Physics.CheckCapsule(sphere1, sphere2, radio, ignoredLayer))
32              obstacleDetected = true;
33          else
34              obstacleDetected = false;
```

- ***Move.cs:*** té una sèrie d'instruccions per poder realitzar la parada del vehicle abans de la col·lisió, frenant de forma progressiva gràcies al planificador *plannerDll*, tal com s'ha comentat abans.

El codi que s'ha desenvolupat al *script Move.cs* per actuar davant la detecció d'un obstacle per evitar la col·lisió és el següent:

Figura 4.11 Codi detecció d'obstacles al Unity

```
215
216             if(detectingObstacles.obstacleDetected)
217             {
218                 double l0 = plannerDll.accumulatedLenght;
219                 plannerDll.Compute(thetaA, zA, xA, thetaB, zB, xB,1);       // In this case the final velocity is zero.
220                 double l1 = plannerDll.accumulatedLenght;
221                 double l = l1 - l0;
222                 int nPoints = plannerDll.getSize();
223                 int index = (int) (4*nPoints/l);
224                 xB                  = plannerDll.getX (index);
225                 zB                  = plannerDll.getZ (index);
226                 thetaB              = -plannerDll.getTheta (index);
227                 //plannerDll.Compute(thetaA, zA, xA, thetaB, zB, xB,1);
228             }
229             plannerSegmentCounter--;
230             plannerDll.Compute(thetaA, zA, xA, thetaB, zB, xB,1);       // In this case the final velocity is zero.
231             braking = false;
232             lastSegment = true;
233             GetWayPoints(initialCarPosTransform, plannerSegmentCounter, out thetaA, out zA, out xA, out thetaB, out zB, out xB);
234             waitCounter++;
235         }
236     }
```

Quan es detecta un obstacle entrem en aquesta rutina, Figura 4.11.

- El primer que es fa es calcular la distància que hi ha fins l'objecte detectat. A partir de la instrucció $plannerDll.accumulatedLenght$, que ens dóna la longitud que ha transcorregut des de que s'ha iniciat la marxa fins el punt on es troba el vehicle, podem esbrinar la distància on es troba l'objecte.
    - Quan es detecta un objecte, determinem que el cotxe es troba al punt A. Calculem la distància que ha recorregut el vehicle des del inici de la ruta.
    - Seguidament, fem la planificació fins al punt B (que es on es troba l'objecte) determinant que la velocitat en aquell punt ha de ser de 0m/s.
    - A continuació, calculem la distància que hi ha entre el punt B i el inici de la ruta.
    - Realitzem la resta entre les dues distàncies, i així es pot saber la distancia entre el vehicle i l'objecte. El problema que tenim es que aquest valor es *double* i necessitem un *integer* per poder introduir els punts al planificador.
    - Mitjançant la comanda $plannerDll.getSize$ obtenim la informació relacionada de la trajectòria entre el punt A i el punt B en forma de *integer*, aquesta informació la guardarem com a *nPoints*.

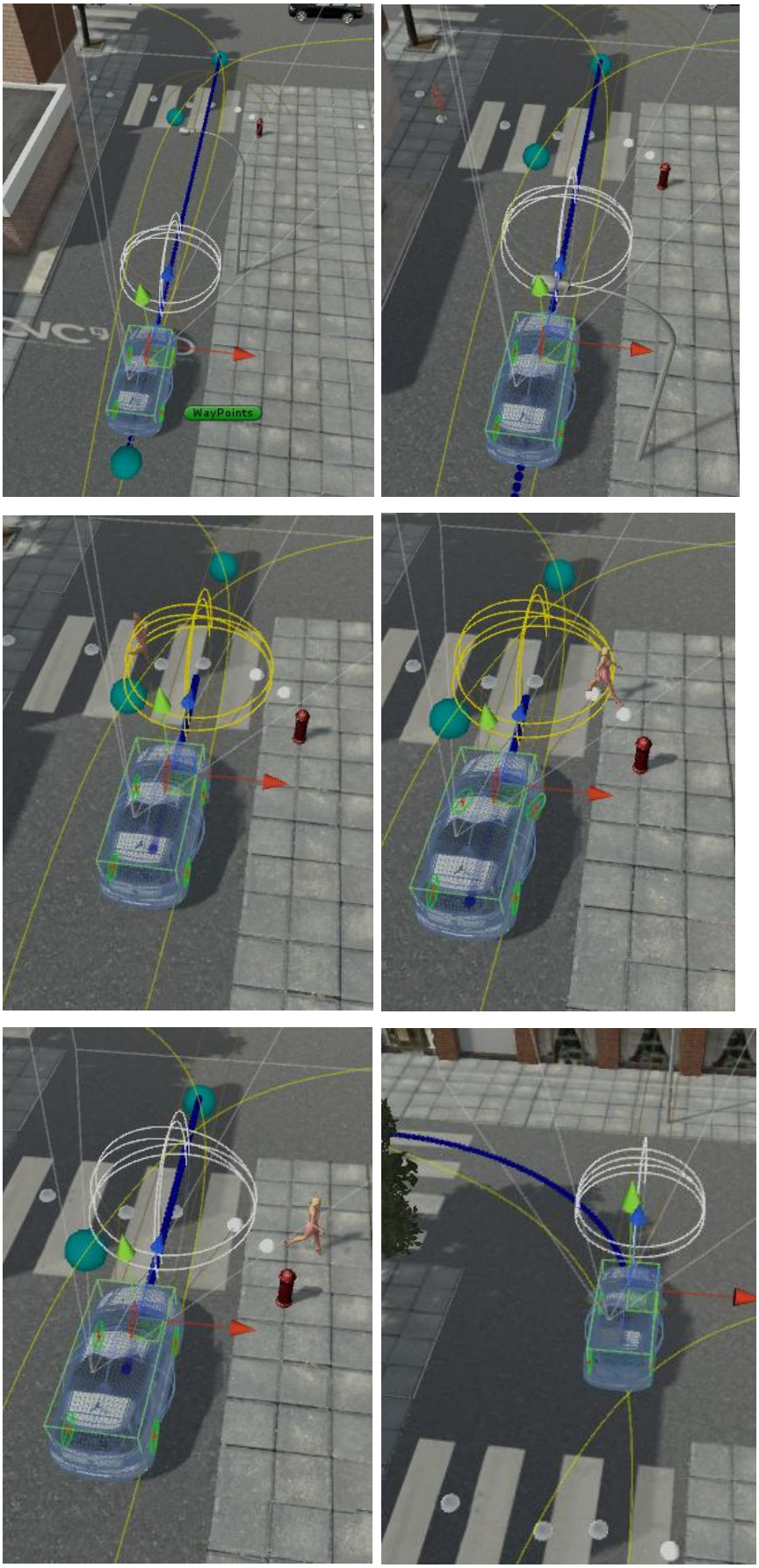
  - Llavors, es crea el valor *integer 'index'*, que serà el resultat de multiplicar el valor de *nPoints* per 4 (aquest 4 ve determinat pel radi de les esferes que s'utilitzen com a detector d'obstacles) i el dividim per la distància *l* calculada. I així obtenim la distància aproximada fins a l'objecte.
  - Finalment, extraiem la informació de *xB*, *zB* i *thetaB* del valor *index* per poder tenir les coordenades i direcció del punt B on s'ha estimat que hi havia l'obstacle

- Es disminueix el $plannerSegmentCounter$, que es el valor que ens diu a quin segment de la ruta ens trobem, perquè la rutina on s'inclou aquest codi sempre incrementa aquesta variable quan es realitza una planificació entre un punt A i B. I representa que l'aturada es realitza en un punt entre dos *waypoints*.
- El $plannerDll.Compute(thetaA, zA, xA, thetaB, zB, xB, 1)$ planifica la trajectòria entre el punt on es troba el vehicle i on hi ha l'objecte. Determinant que la velocitat abans d'arribar al punt B ha de ser 0 m/s.
- Diem que la variable $braking = false$ ja que només s'activarà quan s'hagi de fer una frenada d'emergència
- La variable $lastSegment = true$ perquè així es realitzarà una frenada progressiva suau, ja que es realitza un control que frena progressivament com quan el vehicle ha arribat al final de la seva ruta.
- Al $GetWayPoints$ definim la posició del vehicle quan s'ha aturat, punt A, i anomenem punt B al següent *waypoint* de la ruta.
- Finalment, s'incrementa el $waitCounter$, que és un comptador que va de 0 a 1 en funció de si el vehicle ha realitzat un procés de frenada.

A la Figura 4.12 podem observar la seqüència d'una aturada amb la simulació realitzada pel Unity:

- Escena 1: el vehicle segueix la trajectòria (blau fort) que l'uneix amb el següent *waypoint*
- Escena 2: un vianant apareix a l'escena (part superior esquerra) seguint la seva trajectòria marcada per punts blancs
- Escena 3: el vianant entre dins l'angle de visió del vehicle (esferes) i, per tant, el sistema detecta un obstacle per seguir la seva trajectòria i planifica una frenada
- Escena 4: el vehicle ha realitzat la maniobra de frenada per no col·lisionar, però segueix detectant un obstacle
- Escena 5: es deixa de detectar l'obstacle i es replanifica la trajectòria per continuar el recorregut preestablert
- Escena 6: un cop assolit el *waypoint*, es planifica la trajectòria per arribar al següent, i així successivament fins que s'arriba al punt de destí.

Figura 4.12 Seqüència de la frenada amb el Unity



5. Aturada i maniobra per esquivar l'obstacle

5.1 Marc teòric

En aquest segon cas, ens basarem en els articles [4], [5] i [7] per realitzar el plantejament. El que volem analitzar és la forma de poder parar, i, posteriorment, vorejar l'obstacle per evitar la col·lisió i seguir amb la ruta preestablerta. Només s'ha realitzat el plantejament ja que les simulacions no han estat satisfactòries.

Hi hauran dos passos d'actuació aquest cas:

- Realitzar frenada abrupta o frenada suau
- Generar la trajectòria per poder esquivar l'obstacle

Les variables principals a tenir en compte per el plantejament dels dos passos són les següents:

- Velocitat del vehicle (v_{vh})
- Posició de l'objecte o obstacle (x_o)
- Dimensions de l'objecte en el pla (x,y) respecte les càmeres del vehicle (0,0) a partir d'una vista superior ($x_{dreta} \rightarrow x_d$, $x_{esquerra} \rightarrow x_e$, y_{inici} , y_{fi}), ja que l'alçada del objecte no afecta la trajectòria
- Posició del vehicle (x_{vh})

Per realitzar els càlculs del primer pas, utilitzarem el procés que s'ha explicat pel cas anterior. Per tant, es realitzaran els següents càlculs:

1. **Temps de col·lisió (t_c)**

$$x_i = x_o - x_{vh}\ (m)$$

$$\boldsymbol{t_c} = \frac{x_i}{v_{vh}}\ (s)$$

   a on x_i és la distància entre el vehicle i l'objecte.

2. **Temps de reacció del sistema** $(\boldsymbol{t_r})$ $t_r = 100ms$

3. **Temps de maniobra (t_m)**

$$\boldsymbol{t_m} = t_c - t_r\ (s)$$

$$\boldsymbol{t_m} = \frac{v_{vh}}{a_{I,decc}}\ (s)$$

4. En funció de la velocitat del vehicle i el temps de maniobra calculat determinarem la millor opció:

   4.1 Si el t_m és molt petit -> frenada sobtada (indiferentment de la velocitat)

   4.2 Si el t_m és relativament gran, però la v_{vh} massa elevada -> frenada sobtada

   4.3 Si el t_m és gran i la v_{vh} dintre dels paràmetres estables -> frenada progressiva

En funció del cas que es dóna al quart pas, decidim si podem realitzar la maniobra per superar l'obstacle. En aquest cas, la distància que hi ha d'haver entre el vehicle i l'objecte ha de ser de 3 m per poder fer la maniobra d'esquivar, considerant que l'obstacle amb que ens trobem no és més ample que un vehicle de gran tonatge. Si es compleixen aquests requisits continuarem de la següent manera:

5. **Dimensions de l'obstacle**: considerant les càmeres del vehicle el punt (0,0), determinar l'amplada del obstacle. Si té una amplada superior a 2,8 m no realitzarem cap acció.

$$x_d - x_e = amplada \tag{5.1}$$

$$Limit\ objecte\ per\ la\ dreta = x_d$$

$$Límit\ objecte\ per\ l'esquerra = x_e$$

6. **Generació *waypoint* d'inici de maniobra** (wp_1) : a partir dels límits de l'objecte, l'entorn que el rodeja, i la posició i direcció del vehicle, es decidirà per quina banda es generarà el nou *waypoint* per determinar la primera maniobra. Si es detecta un altre obstacle o no es pot realitzar cap maniobra per l'esquerra anirem per la dreta, Figura 5.1, sempre i quan estigui lliure, i viceversa, Figura 5.2. En cas de obstacles pels dos costats, el vehicle es quedarà aturat al lloc on ha frenat.

<u>Desviació dreta</u>

$$x_i = x_o - x_{vh} = d_1$$

$$d_d = wp_1 - x_d \tag{5.2}$$

$$\beta = tan^{-1}\frac{d_d}{d_1} \tag{5.3}$$

<u>Desviació esquerra</u>

$$x_i = x_o - x_{vh} = d_1$$

$$d_e = |wp_1 - x_e| \tag{5.4}$$

$$\beta = tan^{-1}\frac{d_e}{d_1} \tag{5.5}$$

Figura 5.1 Desviació dreta

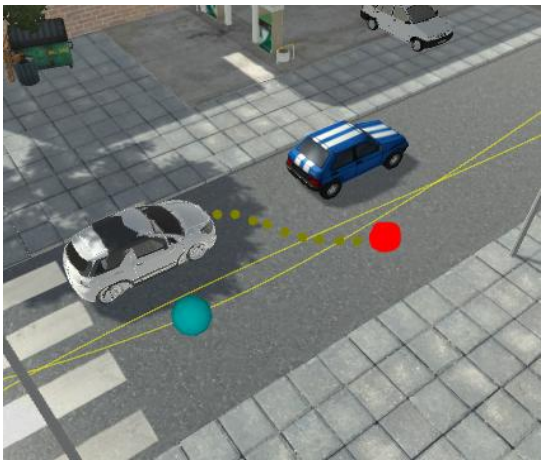
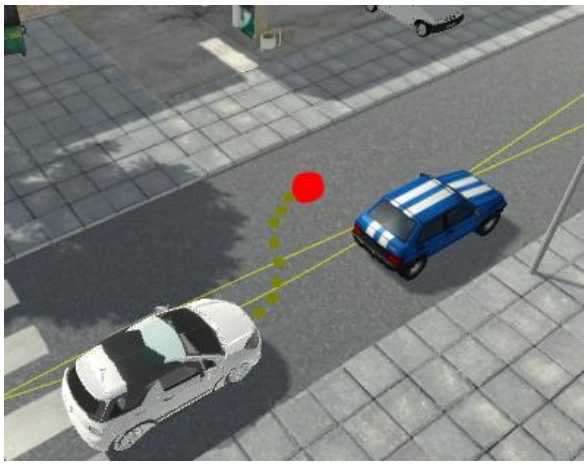


Figura 5.2 Desviació esquerra



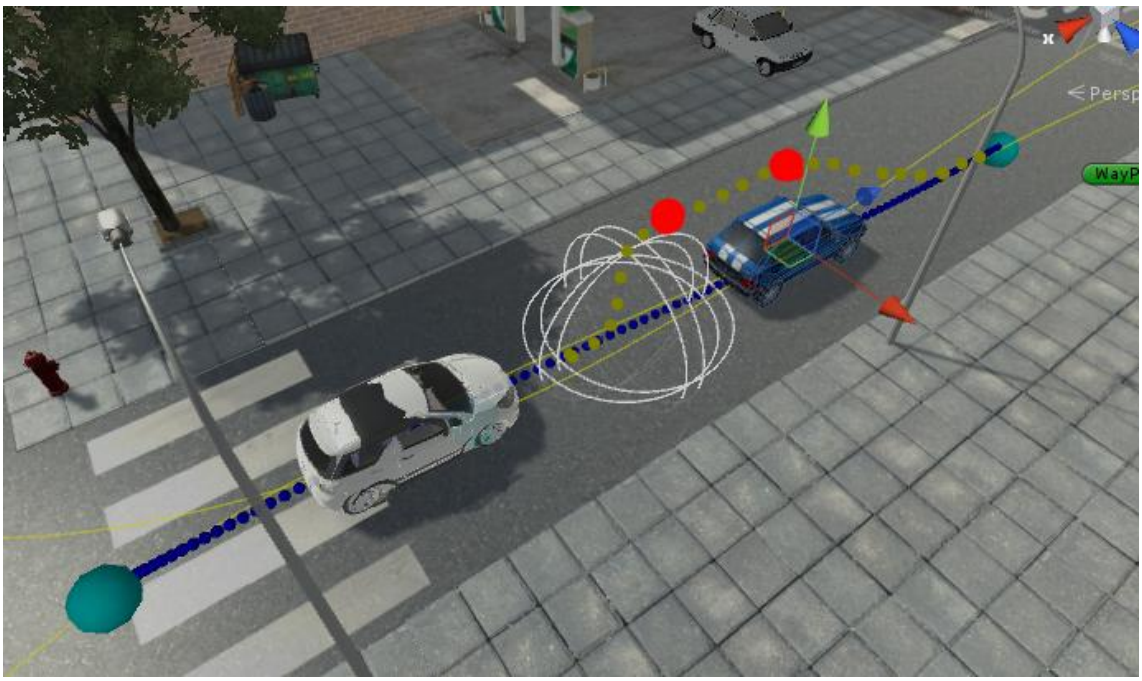
7. **Generar waypoint final de maniobra** (wp_{fi}): un cop assolit el *waypoint* d'inici de maniobra, estudiar la longitud de l'objecte per poder determinar el *waypoint* final de la maniobra. Es generarà una trajectòria paral·lela a la longitud del obstacle, i un cop assolit aquest punt, farem la planificació al *waypoint* següent de la ruta.

$$l_0 = wp_1 = (x_1, y_1)$$
$$l_{fi} = wp_{fi} = (x_{fi}, y_{fi})$$
$$l_{obstacle} = l_{fi} - l_0 \quad (5.6)$$

Per realitzar les proves, definim que el wp_1 es col·loca a 2 m del límit corresponent de l'objecte que s'esquiva. La raó d'aquesta definició es deguda a que les càmeres es troben al centre del vehicle, per tant, sabent que l'amplada del vehicle és de 1,56m i que el centre del vehicle passa per sobre del *waypoint*, tenim 0,78m ocupats per l'amplada del vehicle. Resten 1,22m,que seran utilitzats per deixar una distància de seguretat entre vehicle i obstacle per si la planificació requereix més espai per realitzar el gir.

També definim que la distància que hi ha d'haver mínima entre l'obstacle i el vehicle quan s'ha realitzat la frenada ha de ser de 3 m. Així, entre els 3m de distància i la col·locació del wp_1 a 2 m del lateral, tenim prou espai per poder planificar una maniobra per evitar el contacte amb l'objecte.

Figura 5.3 Maniobra per esquivar l'obstacle



A la Figura 5.3 es mostra un exemple de maniobra que s'ha descrit anteriorment. Quan es detecta un obstacle, realitzarem una frenada suau o abrupta per quedar situats a 3 m del obstacle. Seguidament, farem l'estudi de la situació, en aquest cas, la zona lliure és l'esquerra, per tant, es generarà el wp_1 2m a l'esquerra del x_e del obstacle. Es realitza la planificació i control fins al wp_1 i allà, a través de la informació que ens proporcionen les càmeres, fem un càlcul de la longitud del vehicle i generem el wp_{fi}. Un cop generat, es planifica i es realitza la trajectòria per assolir aquest *waypoint,* i, finalment, es planifica la trajectòria per assolir el *waypoint* de la ruta que correspon al tram que s'estava realitzant abans de la detecció de l'obstacle.

Com que el càlcul teòric matemàtic d'aquest apartat és molt semblant al del apartat 4, es va decidir realitzar la simulació directament amb el *Unity*. Després de realitzar varis intents, no es va aconseguir realitzar una simulació amb els resultats desitjats, i és per aquest motiu que en aquest apartat només es realitza el plantejament teòric explicat anteriorment.

6. Futurs treballs

Aquest treball final de grau és només la base d'un tema que es pot desenvolupar i encarar cap a diferents treballs. La idea estudiada es pot considerar bàsica, una solució pràctica i útil però que es pot millorar. Algunes idees per ampliar i complementar aquest treball són les següents:

- Desenvolupar nous algoritmes de control per aconseguir una frenada combinada entre el fre motor i el sistema de frenada del vehicle, per poder aconseguir una frenada aturant-nos allà on desitgem.
- Desenvolupar un codi semblant al realitzat en funció de les condicions de la via on ens movem
- Generar un sistema perquè l'acció d'esquivar un obstacle és pugi fer amb una sola ordre, sense necessitar l'aturada, calculant els punts crítics i angles de gir adequats
- Aplicar exactament el codi desenvolupat en aquest treball al vehicle i validar el funcionament
- Desenvolupar i generar un codi a partir de la mateixa idea, però estudiant-ho amb velocitats més elevades i testejar-ho al vehicle
- Ampliar el sistema de detecció d'obstacles i generació de trajectòries alternatives per vorejar-lo quan hi ha més d'un obstacle
- Desenvolupament del codi amb obstacles dinàmics, calcular distàncies de frenada adequades en funció de la trajectòria de l'objecte

7. Impactes socials i mediambientals

"Nunca andes por el camino trazado pues te conducirá a donde otros ya fueron"

Alexander Graham Bell

M'agradaria obrir aquest apartat amb aquesta reflexió perquè la idea d'aquest treball és la de col·laborar amb un projecte que és innovador, i que a la llarga, ens conduirà a un futur més sostenible i segur per a tots nosaltres.

Per una banda, tenim la repercussió que tindrà a la nostra societat actual a curt i a llarg termini. Sóc conscient que no puc preveure que pot passar en un futur, però a continuació comentaré els possibles efectes que podria tenir a la societat:

- **Seguretat:** el desenvolupament i aplicació de codis d'aquest caire als vehicle augmentarà la seguretat al volant. Deixarem de dependre del factor humà per dependre del criteri de les màquines, evitant així errors humans quotidians que fem al volant, com consultar mòbils, distraccions,..., que generen accidents o una mala fluïdesa del tràfic.
- **Autonomia:** la programació de vehicles autònoms ens ajudarà a moure'ns sense dependre de ningú. Persones invalides o que necessiten l'ajuda d'una segona persona per desplaçar-se no necessitaran a ningú, ja que tant sols haurem de dir al vehicle on es vol anar i ell ens portarà.
- **Menys temps i més fluïdesa pels trajectes**: a causa de tenir una conducció més eficient i normalitzada per tots els vehicles, implicarà menys retencions i problemes ja que no hi haurà pertorbacions a la mobilitat, partint de la idea que en un futur tothom vagi amb vehicles autònoms.
- **Recerca i innovació**: desenvolupar temes innovadors i enfocats a l'industria automobilística resulta molt atractiu per les grans companyies del sector, molt potents econòmicament parlant. Aquest fet pot suposar una implicació d'aquestes industries en subvencionant més projectes d'aquest estil o duent les idees desenvolupades a les industries del nostre territori, per testejar i provar noves idees.

D'altra banda, el desenvolupament de vehicles elèctrics autònoms suposa un gran impacte mediambiental que millorarà la qualitat de vida. Els factors més importants a destacar són els següents:

- **Reducció de gasos contaminants a l'atmosfera**: al utilitzar vehicles elèctrics, els gasos nocius que malmeten el medi ambient i les nostres vides es veuran reduïts dràsticament. A més a més, tal com s'ha comentat anteriorment, com que tindrem una mobilitat més fluida i els temps de trajectes disminuiran, significa que estarem menys temps expulsant gasos perjudicials per tots nosaltres
- **Reducció de la contaminació acústica**: si la utilització de vehicles elèctrics augmenta els sorolls generats en llocs d'alta concentració automobilística es veurà altament reduït. Al mateix temps això pot ser un inconvenient, ja que la societat no està acostumada a vehicles silenciosos, i pot suposar més d'un ensurt per alguns vianants. Tot i que, amb el desenvolupament d'algoritmes i codis com els que s'han generat en aquest treball el que es vol evitar són aquestes col·lisions inesperades.
- **Bateries dels vehicles elèctrics**: aquest punt és un dels més negatius. Les bateries utilitzades estan fabricades amb elements nocius que costen molt de reciclar o destruir, i com ja sabem, aquestes bateries tenen una vida limitada. Actualment, hi ha moltes investigacions per trobar noves alimentacions per les bateries, menys contaminants per reduir el impacte que tenen amb la natura.

La majoria de punts exposats als dos apartats són avantatges i raons de pes per pensar que s'ha d'evolucionar cap a aquesta direcció. Tot i així, sempre hi haurà la gent escèptica a les noves tecnologies, o amants de la gasolina, que no rebran amb gaire simpatia la implementació d'aquestes idees. Però la convivència entre aquests dos bàndols és possible, i mica en mica, aquestes idees acabaran incidint a la societat, ja que és un bé comú lògic per a la majoria de persones.

8. Conclusions

En aquest treball s'ha tractat el tema de la planificació de trajectòries per evitar la col·lisió contra obstacles. Com ja s'ha comentat anteriorment, s'inclou dins d'un projecte relacionat amb el desenvolupament d'un vehicle autònom elèctric ubicat a la UAB.

El projecte

Al formar part d'un gran projecte, el treball que es realitza ve condicionat per els estudis que han realitzat altres membres del projecte. A partir del que han desenvolupat o ideat, has d'adaptar el treball a les seves especificacions, per poder-lo desenvolupar correctament.

Aquest fet resulta ser el primer handicap, ja que vaig entrar a formar part del projecte de planificació i control, on s'ubica el meu treball, en una fase embrionària, on encara no estaven del tot definits i en funcionament els codis que s'utilitzarien. Per poder desenvolupar les idees depenia dels codis de control i planificació vàlids realitzats en aquesta part del projecte. Abans no hi van haver els codis amb les idees bàsiques de control i planificació funcionant correctament, va passar un cert temps on desenvolupava idees però no les podia acabar aplicant ja que el codi variava i el plantejament ja no estava ben encarat.

Un altre aspecte amb el que em vaig trobar és que la transmissió d'informació entre membres del projecte era complicada; hi ha molts integrants, i per aconseguir informació relacionada amb el significat d'algunes parts dels codis o, funcionament del vehicle i els seus accessoris, et portava molt de temps. Ja fos perquè era una persona externa, o senzillament perquè no hi havia manera de poder-m'hi comunicar. Aquest fet ha generat que algunes parts no les hagi acabat d'assimilar tant bé com ho hauria fet amb les explicacions dels autors.

Tot i això, ha estat una experiència enriquidora ja que he observat com treballa un grup de gent amb un mateix objectiu. La dedicació d'aquestes persones és total, ja que és la seva feina, la majoria d'ells estan becats o són desenvolupadors i es passen més de vuit hores diàries allà, convivint i treballant per un mateix objectiu.

El plantejament

Tal com acabo de comentar, abans no es va poder realitzar un plantejament vàlid es van desenvolupar varies idees. Al voler fer tots els plantejaments i codis en funció del que li passa al vehicle a la realitat (velocitats que assoleix, acceleracions, càmeres de detecció, ubicació GPS) sempre depens d'aquests elements. Durant el període que ha durat aquest TFG, algun d'aquests elements es van canviar o se'n va modificar la configuració perquè es desenvolupaven sistemes més òptims dels que hi havia. Com per exemple, la ubicació GPS del vehicle o les dades obtingudes a través de les càmeres.

Finalment, el que es va fer, es treballar amb la informació amb que es treballava al Maig, ja que era suficient per poder fer un plantejament i introduir la idea de parar el vehicle abans de la col·lisió. Jo crec que va ser un error esperar tant, i que hauria d'haver fet un model determinant jo mateix els valors necessaris a partir de les diverses simulacions que s'havien fet i llavors intentar-ho adaptar a les modificacions que es feien. Però tampoc hagués tingut els codis que funcionessin correctament de control i planificació.

Els resultats

Un cop es va fer el plantejament, determinant les dades convenients, es va poder desenvolupar una mica el codi i la idea principal de funcionament. És una idea bàsica, però al mateix temps és útil ja que funciona, tal com s'ha pogut veure amb les simulacions.

Per desenvolupar el treball es troba informació relacionada amb la planificació i control de vehicles, però no gaire d'algoritmes per evitar obstacles. La informació que es troba, va en funció de la planificació i control que es fa per cada estudi o cas, i no he estat capaç de trobar informació per evitar obstacles relacionada amb el control i planificació que va realitzar l'Eugenio de *Lyapunov* i *Slide Mode Control*.

Un altre inconvenient dels resultats és que són molt ideals, és a dir, no es tenen en compte tots els factors que poden intervenir a l'acció de frenar un vehicle; variables com l'estat de la via (fricció entre la via i els pneumàtics, ja que depenent de la frenada el cotxe podria lliscar i allargar així la seva distància de frenada) o condicions meteorològiques (que poden afectar el procés de captació d'imatges o el rendiment del vehicle).

També haig de dir que ens hem basat en objectes estàtics, i no sempre ens trobarem amb aquest tipus d'obstacles. Un dels següents passos seria el desenvolupament però per objectes dinàmics; a partir de x cicles de procés, determinar la velocitat i direcció de l'objecte per evitar la col·lisió.

La simulació

Per la simulació, tal com s'ha comentat anteriorment, s'han utilitzat codis de planificació i control ja existents. Un problema que he tingut és que no he estat capaç d'entendre tots els codis amb facilitat ja que eren complexos, i, tot i demanar ajudar a diversos autors o ajudants, no em resolien del tot els dubtes. El que m'ha faltat, per poder desenvolupar més a fons les idees, ha estat la manera de poder introduir *waypoints* aleatoris (definits en funció de les posicions dels obstacles o frenades inesperades) a la trajectòria predefinida desde un principi. Aquest fet m'hagués ajudat a poder simular més bé les idees plantejades.

Pel que fa al *Matlab*, és un codi que ens permet veure el comportament del vehicle amb facilitat però no podem plasmar el codi teòric desenvolupat. És menys complexa que el *Unity*, i al haver-hi treballat durant el grau, ha estat més fàcil la generació del codi per la posterior simulació.

El *Unity* és un programa que no havia vist mai abans, que s'ha programat mitjançant *C#,* que tampoc l'havia tocat abans, tot i que la seva similitud amb el *C* ho ha facilitat. Es peculiar perquè a vegades el que tens programat deixa de funcionar entre simulació i simulació, sense tocar ni modificar el codi. En aquests casos el millor és reiniciar l'ordinador i tornar-ho a executar perquè torni a funcionar. Al poder modificar dades en dos llocs diferents, a la interfície del *Unity* i als *scripts* relacionats amb cada *GameObject*, a vegades es generen conflictes i sorgeixen errors, ja que quan modifiques dades a un lloc no s'actualitzen a l'altre.. Per la programació del *Unity* em va ajudar l'Eugenio ja que hi havia comandes i parts que no entenia del tot. A partir del plantejament que li vaig exposar, vam desenvolupar el codi per poder-ho simular i veure'n el funcionament.

Conclusió final

En resum, aquest treball final de grau m'ha servit per poder entendre i observar el funcionament d'un vehicle autònom. La configuració del vehicle passa per moltes fases; les idees es desenvolupen a través de molts nivells, i abans no es poden implementar al cotxe per veure el funcionament a la realitat, passa per molts filtres. Aquest procés és un treball molt laboriós que no s'aprecia desde fora.

Amb aquest treball també he pogut observar que és complicat plasmar les idees teòriques a la pràctica. Molts cops estem limitats per agents externs al nostre treball, com han estat els codis creats pel control i planificació del vehicle, que sense ells no es podria desenvolupar el projecte amb que es treballa. Tot i així, a partir de la simulació podem desenvolupar, ajustar i testejar la teoria per aproximar-ho més a la realitat. I finalment, implementar-ho al món real tot esperant que els resultats s'aproximin a la teoria realitzada.

Gràcies a aquest treball he pogut treballar amb un programa de simulació virtual molt interessant, que m'ha obert les portes d'un món amb que no m'havia fixat fins ara, que em resulta atractiu.

9. Bibliografia

[1] Alcalá, E. Modelling, Planning and Nonlinear Control Techniques for Autonomous Vehicles. ETSEIB-UPC, Barcelona, 2016.

[2] Berthelot, A; Tamke, A; Dang, T; Beurel, G. A novel approach for the probabilistic computation of Time-To-Collision. IEEE Intelligent Vehicles Symposium, Alcala de Henares, 2012.

[3] Berthelot, A; Tamke, A; Dang, T; Beurel, G. Handling Uncertainties in Criticality Assessment. IEEE Intelligent Vehicles Symposium, Baden-, 2012.

[4] Brännström, M; Coelingh, E. Decision-making on when to brake and when to sterr to avoid a collision. IEEE International Journal of Vehicle Safety, Göteborg, Vol 7, p 86-106, 2014.

[5] Gray, A; Ali, M; Gao, Y; Hendrick, J; Borrelli, F. Semi-Autonomous Vehicle Control for Road Departure and Obstacle Avoidance. University of California and Volvo Car Corporation, Göteborg, 2013.

[6] Gonzalez, D; Pérez, J; Lattarulo, R; Milanés, V; Nashashibi, F. Continuous curvature planning with obstacle avoidance capabilities in urban scenarios. IEEE 17th International Conference on Intelligent Transportation Systems, Quingdao, 2014.

[7] Gu, T; Snider, J; Dolan, J; Lee, J. Focused Trajectory Planning for Autonomous On-Road Driving. IEEE, Korea, 2013.

[8] Han, I; Luan, B; Hsieh, F. Development of Autonomous Emergency Braking Control System Based on Road Friction. IEEE International Conference on Automation Science and Engineering, Taipei, 2014.

[9] Lee , D. H; Kim, S. K; Kim, C. S; Huh, K.S. Development of an autonomous braking system using the predicted stopping distance. International Journal of Automotive Technology, Korea, Vol 15, p 341-346, 2014.

[10] Macek, K; Vasquez, A; Fraichard, T: Siegwart, R. Safe Vehicle Navigation in Dynamic Urban Environments: A Hierarchical Approach. IROS Workshop on Planning, Perception and Navigation for Intelligent Vehicles, Nice, 2008.

[11] MathWorks. Matlab Manual [en línia]. Versió 2015. Natick: Matlab, 2015. Disponible a: < https://www.mathworks.com/help/pdf_doc/matlab/getstart.pdf>

[12] Sellart, L; Alcalá, E; Puig, V; Quevedo, J; Saludes, J; Vàzquez, D; López, A. Comparison of two non-linear model-based control strategies for autonomous vehicles. Universitat Politècnica de Catalunya, Barcelona, 2015.

[13] Sosa, R; Velazquez, G. Obstacles Detection and Collision Avoidance System Developed with Virtual Models. ITESM Campus Toluca, Toluca, 2008.

[14] Tamke, A; Dang, T; Breuel, G. A Flexible Method for Critically Assessment in Driver Assistance Systems, IEEE Intelligent Vehicles Symposium, Vol 4, Baden-Baden, 2011.

[15] Tian, J. Trajectory Planning for Autonomous Systems. University of Manchester, p 10-30, Manchester, 2013.

[16] Unity Technologies. Unity Manual [en línia]. Versió 5.4. Unity, 2016. Disponible a: < https://docs.unity3d.com/Manual/index.html >

10. Annex

- Dades del vehicle

TAZZARI EV
ZERO EMISSION MOBILITY

13/09/2016

TAZZARI ZERO EM1

Roof color	Metal Lunar Grey Roof
Exterior color	Metal Lunar Grey
Wheels color	Grey wheels
Interiors color	Prestige Black

TECHNICAL SPECIFICATIONS

Category: Car - M1
Body type: 2 doors
Length: 2880 mm / 113.4"
Width: 1560 mm / 61.4"
Height: 1425 mm / 56.1"
Maximum speed: 100 km/h / 62 mph
Estimated range / Economy drive: 150 km (93 miles)
Luggage capacity (litres): 2 luggage areas with 180 litre capacity
Tyres and wheels: 175/55 R15
Motor: Rear-wheel drive 3-phase
Maximum torque peak: 150 Nm
Batteries type: Li-ion Fe 13kW/150 km / 93 miles
Charging time - Standard 220V batteries charger: 100% in 9 hrs

STANDARD EQUIPEMENT

Electric heating system
Two years' full Guarantee (on Battery and Vehicle)
3-speed 220V Multifast battery charger (5 hours)
Lithium battery equalizer
Alcantara material upholstery
Econometer LED display unit
LED Daytime Running Lights
LED Tail lights and third brake light
Reversing light
Rear fog light
Lunar Grey painted mirrors with integrated LED turn signals
180 liters' capacity hood and trunk equipped with LED courtesy lights
Electrically adjustable side-view mirrors
Electric windows
Central door locking
Alarm (automatic window-raiser device and siren)
Immobilizer
Racing-inspired control pedals
Aluminum designer interior handles
faux leather seats
Phone, tablet and PC charging port (max 90W)
Electric regenerative braking system
10-segment charge indicator
4 working modules with different driving settings
"Comfort PLUS" shock absorbers
Interior with storage compartments and drink holders
Touch Screen Instrument Panel
New high-brightness low beam lights electrically adjustable from inside
Vehicle interior LED courtesy light
Parking sensors
4 disc brake braking system with power brake and ABS
Electrically heated windshield
Heated rear window
Air conditioning system

AUDIO SYSTEM 2015

Radio with Removable Front panel
LED Widescreen Display
CD, DVD, DivX, Mp3, Mp4 player
with SD Card and USB input (up to 32GB)
AUX input
Bluetooth Hands-free phone set
Wireless audio Streaming via Bluetooth (A2DP)
Full Range 2 Way Speakers
Rear view camera
Automatic actuation rear view camera
Automatic night-time mode
CMOS HD Sensor
170Â° viewing angle

OPTIONALS

LITHIUM batteries 150K	€ 3900

- **Codis *Matlab***

curves_interpolation.m

```
function
v=curves_interpolation(xi,yi,xf,yf,L,t_fin,vi,vf,ai,af,tetaA,l,KK,x_in
it,y_init)

%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%   initial conditions  %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%
a0=ai;
v0=vi;
a5=af;
v5=vf;
%%%%%%%%%%%%%%%%%%%%%%%%%%%

t_gama=round2(t_fin/5);

if ai < 0 & vi > 0
    t_beta=2*vi/ai;
elseif af > 0 & vf > 0
    t_beta=2*vf/af;
else
    t_beta=t_gama;
end

    t1=t_gama;
    t2=t1;
    t4=t2;
    t5=t4;
    t3=t_fin-t1-t2-t4-t5;
    t=round2([t1 t2 t3 t4 t5]);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% FIND a starting solution  %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    v = findvelocity(L,vi,vf,ai,af,t);
    v2=v(1);
    v3=v(2);
    a2=v(3);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
aa=temporal_curves(vi,ai,vf,af,v2,v3,a2,t,L);
t_fin=t(1)+t(2)+t(3)+t(4)+t(5);

%_______________________________________________

l_init=interp1(0:0.01:1, l, 0:0.01/t_fin:1, 'spline');
KK_init=interp1(0:0.01:1, KK, 0:0.01/t_fin:1, 'spline');

tt2=interp1(aa(4,:),aa(1,:),l_init,'spline');
speed=interp1(aa(1,:),aa(2,:),tt2,'spline');
omega3=KK_init.*speed;
omega4=interp1(tt2,omega3,0:0.01:t_fin,'spline');

%_______________________________________________

v=[aa;omega4;KK_init];
```

temporal_curves.m

```
function g=temporal_curves(vi,ai,vf,af,v2,v3,a2,tt,L)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%   initial conditions   %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
a0=ai;
v0=vi;
a5=af;
v5=vf;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    t1=tt(1);
    t2=tt(2);
    t3=tt(3);
    t4=tt(4);
    t5=tt(5);

    tt=([t1 t2 t3 t4 t5]);

    a(1,1)=vi;
    a(1,2)=(2*v2*t1+2*vi*t2+(ai-a2)*t1*t2)/(2*(t1+t2));
    a(1,3)=v2;
    a(1,4)=v3;
    a(1,5)=(2*(vf*t4+v3*t5)*t3+(2*v3-2*v2-a2*t3-
af*t3)*t4*t5)/(2*t3*(t4+t5));

    a(2,1)=ai/2;
    a(2,2)=(-ai*t1-a2*t2-2*vi+2*v2)/(2*(t1+t2));
    a(2,3)=a2/2;
    a(2,4)=(-a2*t3-2*v2+2*v3)/(2*t3);
    a(2,5)=(2*(v2-v3)*t4-(2*v3-2*vf-a2*t4+af*t5)*t3)/(2*t3*(t4+t5));

    a(3,1)=(-2*ai*t1-(ai+a2)*t2-2*vi+2*v2)/(6*t1*(t1+t2));
    a(3,2)=((ai+a2)*t1+2*a2*t2+2*vi-2*v2)/(6*t2*(t1+t2));
    a(3,3)=(-2*a2*t3+2*v3-2*v2)/(6*t3*t3);
    a(3,4)=(2*(v2-v3)*(t5+2*t4)+(2*a2*t4-2*(v3-vf)+(a2-
af)*t5)*t3)/(6*t3*t4*(t4+t5));
    a(3,5)=((2*v3-2*vf-a2*t4+af*t4+2*af*t5)*t3-2*(v2-
v3)*t4)/(6*t3*t5*(t4+t5));

%%%%%%%%%%
%%  t1  %%
%%%%%%%%%%
m=1;
    for l=0:0.01:t1
        aa1(1,m)=l;                                  % the time vector
        aa1(2,m)=a(1,1)+2*a(2,1)*l+3*a(3,1)*l^2;     % the velocity
vector
        aa1(3,m)=2*a(2,1)+6*a(3,1)*l;                % the acceleration
vector
        aa1(4,m)=a(1,1)*l+a(2,1)*l^2+a(3,1)*l^3;     % the length
segments vector
        aa1(5,m)=abs(6*a(3,1));                      % the jerk vector
        m=m+1;
    end

%%%%%%%%%%
%%  t2  %%
```

```
%%%%%%%%%%
m=1;
    for l=0.01:0.01:t2
        aa2(1,m)=aa1(1,length(aa1))+l;
        aa2(2,m)=a(1,2)+2*a(2,2)*l+3*a(3,2)*l^2;
        aa2(3,m)=2*a(2,2)+6*a(3,2)*l;
        aa2(4,m)=(aa1(4,length(aa1))+a(1,2)*l +a(2,2)*l^2+a(3,2)*l^3);
        aa2(5,m)=abs(6*a(3,2));
        m=m+1;
    end

%%%%%%%%%%
%%  t3  %%
%%%%%%%%%%
m=1;
    for l=0.01:0.01:t3
        aa3(1,m)=aa2(1,length(aa2))+l;
        aa3(2,m)=a(1,3)+2*a(2,3)*l+3*a(3,3)*l^2;
        aa3(3,m)=2*a(2,3)+6*a(3,3)*l;
        aa3(4,m)=(aa2(4,length(aa2))+a(1,3)*l+a(2,3)*l^2+a(3,3)*l^3);
        aa3(5,m)=abs(6*a(3,3));
        m=m+1;
    end

%%%%%%%%%%
%%  t4  %%
%%%%%%%%%%
m=1;
    for l=0.01:0.01:t4
        aa4(1,m)=aa3(1,length(aa3))+l;
        aa4(2,m)=a(1,4)+2*a(2,4)*l+3*a(3,4)*l^2;
        aa4(3,m)=2*a(2,4)+6*a(3,4)*l;
        aa4(4,m)=(aa3(4,length(aa3))+a(1,4)*l+a(2,4)*l^2+a(3,4)*l^3);
        aa4(5,m)=abs(6*a(3,4));
        m=m+1;
    end

%%%%%%%%%%
%%  t5  %%
%%%%%%%%%%
m=1;
    for l=0.01:0.01:t5
        aa5(1,m)=aa4(1,length(aa4))+l;
        aa5(2,m)=a(1,5)+2*a(2,5)*l+3*a(3,5)*l^2;
        aa5(3,m)=2*a(2,5)+6*a(3,5)*l;
        aa5(4,m)=aa4(4,length(aa4))+a(1,5)*l+a(2,5)*l^2+a(3,5)*l^3;
        aa5(5,m)=abs(6*a(3,5));
        m=m+1;
    end

aa5(4,length(aa5))=round2(aa5(4,length(aa5)));

aa=[aa1 aa2 aa3 aa4 aa5];
S=aa(4,length(aa));
g=aa;

yh = 0:0.01:(t1+t2+t3+t4+t5);
```

findvelocity.m

```
function v=findvelocity(sf,vi,vf,ai,af,t)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% FIND a starting solution  %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% sf es L
% t es el vector que te el temps dels 5 trams

v0=vi;
v5=vf;
a0=ai;
a5=af;

v2=0;
v3=v2;
a2=0;

t1=t(1);
t2=t(2);
t3=t(3);
t4=t(4);
t5=t(5);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 S=0;

    while S ~= sf % desigual

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% THE SIMPLIFIED CURVE PARAMETRIZATION %%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    clear aa1 aa2 aa3 aa4 aa5 aa

        a(1,1)=v0;
        a(1,2)=(2*(v2*t1+v0*t2)+a0*t1*t2)/(2*(t1+t2));
        a(1,3)=v2;
        a(1,4)=v2;
        a(1,5)=(2*(v2*t5+v5*t4)-a5*t4*t5)/(2*(t4+t5));

        a(2,1)=a0/2;
        a(2,2)=(2*(v2-v0)-a0*t1)/(2*(t1+t2));
        a(2,3)=0;
        a(2,4)=0;
        a(2,5)=(2*(v5-v2)-a5*t5)/(2*(t4+t5));

        a(3,1)=(2*(v2-v0-a0*t1)-a0*t2)/(6*t1*(t1+t2));
        a(3,2)=-(2*(v2-v0)-a0*t1)/(6*t2*(t1+t2));
        a(3,3)=0;
        a(3,4)=(-a5*t5-2*v2+2*v5)/(6*t4*(t4+t5));
        a(3,5)=(2*(v2-v5+a5*t5)+a5*t4)/(6*t5*(t4+t5));

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        %%%%%%%%%%
        %%  t1  %%
```

```
        %%%%%%%%%%

        for m=1:5
            va(m)=a(1,m)+2*a(2,m)*t(m)+3*a(3,m)*(t(m))^2;            %
the velocity vector
            aa(m)=2*a(2,m)+6*a(3,m)*t(m);                            %
the acceleration vector
            sa(m)=a(1,m)*t(m)+a(2,m)*(t(m))^2+a(3,m)*(t(m))^3;       %
the length segments vector
            ja(m)=abs(6*a(3,m));                                     %
the jerk vector
        end

        v1=va(1);
        v2=va(2);
        v3=va(3);
        v4=va(4);
        v5=va(5);
        a2=aa(2);

        s1=sa(1);
        s2=sa(2);
        s3=sa(3);
        s4=sa(4);
        s5=sa(5);

        S=round2(s1+s2+s3+s4+s5);

        if abs(sf-S) > 1
            if S < sf
                v2=v2+0.05;
            elseif S ~= sf
                v2=v2-0.005;
            end
        else

            if S < sf
                v2=v2+0.0001;
            elseif S ~= sf
                v2=v2-0.00001;
            end
        end
    end

S_vitee=S;
v=[va(2) va(3) aa(2)];
```

round2.m

```
function y = round2(x)
y = 0.01 * round(100*x);
```

localPlanner.m

```
function [vel_profiles, v_fin, tfin, desired_path] =
localPlanner(g,brake,v_in,desired_path)

global kkkk teta Points k
tetaA=teta(g);
tetaB=teta(g+1);
xA=Points(1,g);
yA=Points(2,g);
xB=Points(1,g+1);
yB=Points(2,g+1);

kA=k(g);
kB=k(g+1);
g1=norm(Points(:,g)-Points(:,g+1));
g2=g1;
g3=0;
g4=-g3;

kx=1.4;
ky=1.4;

x0 = xA;
x1 = g1*cos(tetaA);
x2 = 1/2*(g3*cos(tetaA)-(g1)^2*kA*sin(tetaA));
x3 = 10*(xB-xA)-(6*g1 + 3/2*g3)*cos(tetaA) - (4*g2 - (1/2)*g4
)*cos(tetaB) + (3/2)*(g1)^2*kA*sin(tetaA) -
(1/2)*(g2)^2*kB*sin(tetaB);
x4 = -15*(xB - xA) + (8*g1 + (3/2)*g3)*cos(tetaA) + (7*g2 -
g4)*cos(tetaB) - (3/2)*(g1)^2*kA*sin(tetaA) + (g2)^2*kB*sin(tetaB);
x5 = 6*(xB - xA) - (3*g1 + (1/2)*g3)*cos(tetaA) - (3*g2 -
(1/2)*g4)*cos(tetaB) + (1/2)*(g1)^2*kA*sin(tetaA)-
(1/2)*(g2)^2*kB*sin(tetaB);

y0 = yA;
y1 = g1*sin(tetaA);
y2 = 1/2*(g3*sin(tetaA)+(g1)^2*kA*cos(tetaA));
y3 = 10*(yB-yA)-(6*g1 + (3/2)*g3)*sin(tetaA) - (4*g2 - (1/2)*g4
)*sin(tetaB) - (3/2)*(g1)^2*kA*cos(tetaA) +
(1/2)*(g2)^2*kB*cos(tetaB);
y4 = -15*(yB - yA) + (8*g1 + (3/2)*g3)*sin(tetaA) + (7*g2 -
g4)*sin(tetaB) + (3/2)*(g1)^2*kA*cos(tetaA) - (g2)^2*kB*cos(tetaB);
y5 = 6*(yB - yA) - (3*g1 + (1/2)*g3)*sin(tetaA) - (3*g2 -
(1/2)*g4)*sin(tetaB) -
(1/2)*(g1)^2*kA*cos(tetaA)+(1/2)*(g2)^2*kB*cos(tetaB);

X=[x0 x1 x2 x3 x4 x5];
Y=[y0 y1 y2 y3 y4 y5];

clear l
n=1;
l(1)=0;

for u=0:0.01:1
    alfa(n)=x0+x1*u+x2*u^2+x3*u^3+x4*u^4+x5*u^5;
    beta(n)=y0+y1*u+y2*u^2+y3*u^3+y4*u^4+y5*u^5;

    if n > 1 %Longitud del segmento
```

```
        l(n) = l(n-1) + sqrt((alfa(n) - alfa(n-1))^2 + (beta(n) -
beta(n-1))^2);
    end

    x_d(n) = x1 + 2*x2*u + 3*x3*u^2 + 4*x4*u^3 + 5*x5*u^4;
    y_d(n) = y1 + 2*y2*u + 3*y3*u^2 + 4*y4*u^3 + 5*y5*u^4;

    x_d_d(n) = 2*x2 + 6*x3*u + 12*x4*u^2 + 20*x5*u^3;
    y_d_d(n) = 2*y2 + 6*y3*u + 12*y4*u^2 + 20*y5*u^3;

    KK(n)=(x_d(n)*y_d_d(n) - x_d_d(n)*y_d(n))/((sqrt((x_d(n))^2 +
(y_d(n))^2))^3);
    if u<1
        alfa_1(n)=alfa(n);
        beta_1(n)=beta(n);
    end
    n = n + 1;
end

LLL=round2(l(length(l)));
l(length(l))=LLL;

desired_path = [desired_path [alfa;beta;KK;l]];

acc_max = 0.9;
tfin=round2(sqrt(2*LLL/acc_max));


if g==kkkk(1,g)
  tfin=kkkk(2,g)+kkkk(2,g)*10/100; % 10%
end

if (brake == 0)
    v_fin(g) = LLL/(tfin);
else
    v_fin(g) = 0;
end

aW = 4;
while aW > 1

V=curves_interpolation(xA,yA,xB,yB,LLL,tfin,v_in,v_fin(g),0,0,tetaA,l,
KK,alfa,beta);

    aT=V(3,:);
    aL=V(6,:).*(V(2,:));
    aL_rms = sqrt((sum(aL.^2))/length(aL));
    aT_rms = sqrt((sum(aT.^2))/length(aT));
    aW=sqrt(kx*kx*aT_rms^2 + ky*ky*aL_rms^2);

    if aW > 1
        tfin=tfin+tfin*2.5/100;
        if min(V(2,:))<-0.005
            if g==1
                v_fin(g)=v_fin(g)-0.05;
                 return;
            elseif v_fin(g-1)>0.05 && v_fin(g-1) > v_fin(g)
                v_fin(g-1)=v_fin(g-1)-0.4;
```

```
                v_fin(g)=0;
                disp('dentro de elseif')

            else v_fin(g)<0.05
                v_fin(g)=v_fin(g)-0.05;
                disp('dentro de else')
                return;
            end
        end
    end

end %(WHILE)

vel_profiles = V;
```

traj_plan.m

```
clear all
clear
clc
global kkkk xxx yyy teta Points k
global kx ky;

%%% Inicialització i definició de variables
Points = [xxx;yyy];
kkkk = [zeros(2,length(Points))];
v_fin = [zeros(1,length(Points))];
k = zeros(1,length(xxx));
kx=1.4;
ky=1.4;
v_in=0;
L=0;
ttt=[];
LL=[];
all=[];
aWW=[];
aLT=[];
aLT_rms=[];
KKK=[];
point=[];
XY=[];
desired_path=[];
t_fin=0;
Theta(1)=0;
profileSet=[];
obstacle_detected = false;
brake = false;
counter = 0;
obstacle=[];
tram_fi = false;

%%% Exemple de trajectòria
xxx  = [0, 30, 65,     83,   65,  37, 33, 28, 24,  0];
yyy  = [0,  0,  0,    -18,  -36, -36,-39,-39,-36, -36];
teta = [0,  0,  0,  -pi/2,   pi,  pi, pi, pi, pi, pi];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%
```

```
for g=1:length(Points)-1
    %Aquest if utilitza la variable counter, que conta el nombre de
    %segments, l'utilitzarem per definir quan hi ha un obstacle o s'ha
    %acabat la trajectòria
    if(counter ==5) | (counter ==8)
        if (counter ==5)
        obstacle_detected = true;
        counter = counter + 1;
        else
            tram_fi = true;
        end
    end

    if (obstacle_detected) | (tram_fi)
%Si es detecta un obtacle s'activa la variable brake (localPlanner)
%on es planifica una acció de desacceleració dient que v_fin(g) = 0.
%t_fin:temps acumulat // v_fin: velocitat al final del segment
%vel_profiles: (del localPlanner) obtenir variables de velocitat,
acceleracio, logitud, posicio)
%profiles:acumulem dades al vector profileSet per la posterior
representació
        brake = true;
        [vel_profiles, v_fin, tfin, desired_path] =
localPlanner(g,brake,v_in, desired_path);
        t_fin=t_fin + tfin;
        profileSet=[profileSet vel_profiles];
        v_in = v_fin(g);
        obstacle_detected = false;
        brake = false;
    else
      %Aquest cas es el corrent, es a dir, el que es realitza quan no
      %s'ha detectat res. La unica diferencia respecte el if es que la
      %variable brake segueix sent falsa
        [vel_profiles, v_fin, tfin, desired_path] =
localPlanner(g,brake,v_in, desired_path);
        t_fin=t_fin + tfin;
        profileSet=[profileSet vel_profiles];
        v_in = v_fin(g);
        counter = counter + 1;
    end
end % (FOR)

%%% D'aquí cap avall només representació

% Posicio i orientacio inicial
x_rrr(1) = Points(1,1);
y_rrr(1) = Points(2,1);
teta_rrr(1) = teta(1);

pass=0.01;
mm=1;
for m=0:pass:t_fin
    teta_rrr(mm+1)=pass*(profileSet(6,mm))+teta_rrr(mm);

     if (teta_rrr(mm+1)<-pi)
          teta_rrr(mm+1)=teta_rrr(mm+1)+2*pi;
     elseif (teta_rrr(mm+1)>pi)
          teta_rrr(mm+1)=teta_rrr(mm+1)-2*pi;
     end
```

```
    x_rrr(mm+1)=pass*profileSet(2,mm)*cos(teta_rrr(mm))+x_rrr(mm);
    y_rrr(mm+1)=pass*profileSet(2,mm)*sin(teta_rrr(mm))+y_rrr(mm);
    mm=mm+1;
end

XYTheta = [x_rrr; y_rrr; teta_rrr];
time_total=0:0.01:length(profileSet)*0.01-0.01;

nr_points=round(t_fin/10);
x_inter=interp1(0:0.01:t_fin+0.01,x_rrr,[5 10 15 20],'spline');
y_inter=interp1(0:0.01:t_fin+0.01,y_rrr,[5 10 15 20],'spline');

figure(1)
plot(desired_path(1,:),desired_path(2,:),'g',x_rrr,y_rrr,'r',Points(1,
:),Points(2,:),'ob','LineWidth',2)
title('Path','fontsize',12)
xlabel('x [m]','fontsize',12)
ylabel('y [m]','fontsize',12)
grid on

figure(2)
plot(time_total,profileSet(2,:),'r', time_total,profileSet(6,:),'--b')
%,point(2,:),point(1,:),'b*')
xlabel('time [s]','fontsize',12)
ylabel('Velocity [m/s]','fontsize',12)
legend('Linear velocity [m/s]','Angular velocity [rad/s]')
grid on

timee=0:0.01:length(profileSet(1,:))*0.01-0.01;
velocity=[profileSet];

speed = velocity(2,1:10:length(velocity))';
acceler = velocity(3,1:10:length(velocity))';
angular_velo = velocity(6,1:10:length(velocity))';
acc_angular = (angular_velo.*speed);

date = [speed'; acceler'; angular_velo'; acc_angular'];
dlmwrite('profiles_data.txt',date');

return;
```