Università Politecnica delle Marche

Laurea Magistrale in Ingegneria Informatica e dell'Automazione

Design and Simulation of a Collision Warning System for Vulnerable Road Users (VRUs) with a priori Map Knowledge

Design e simulazione di un sistema per predire collisioni tra utenti vulnerabili della strada (VRU) e veicoli

Tesi di Laurea di:
Chiara Leonori

Relatore: Chiar.mo
Prof. David Scaradozzi

Correlatori:
Markus Wendl
Torsten Hafer

A.A. 2020/2021

To you, babbetto,
my father, my rock, my angel.
Though I cannot hug you,
I know you are always with me.

Tutto é per te e grazie a te.

Abstract

In Stati come la Germania, un numero sempre maggiore di persone rinunciano all'utilizzo di un'auto, prediligendo spostamenti tramite bicicletta o a piedi, anche al fine di proteggere l'ambiente. Per questo motivo, é stata introdotta nel Codice della Strada nel 2021 una nuova categoria denominata VRU, Vulnerable Road Users o Utenti Vulnerabili della Strada, nei cui confronti sono previste misure per agevolare gli spostamenti e ridurre i disagi. In particolare, tale termine indica tutti i soggetti non motorizzati e sprovvisti di protezioni esterne, quindi piú a rischio, come ad esempio pedoni e ciclisti. Tali provvedimenti, seppur riducendo notevolmente il numero di incidenti in cui i VRU sono coinvolti, non risolvono il problema alla radice. Vari sistemi sono stati sviluppati negli ultimi anni per essere installati nei veicoli o nelle infrastrutture stradali, per aumentare la sicurezza dei VRU e diminuire incidenti in situazioni di NLoS (Non-line-of-Sight): sensori, come ad esempio LiDAR e camere, e tecnologie V2P (Vehicle2Pedestrian) rappresentano solamente un esempio.
In tale cornice si colloca il progetto *People Mover*, promosso e finanziato dalla cittá di Ratisbona (Germania), volto a diminuire gli incidenti che vedono coinvolti VRU. Tale progetto, in fase di decollo, mira all'installazione di una infrastruttura stradale nella zona industriale, e precisamente in uno dei crocevia più trafficati.
Lo studio condotto in questa tesi propone un sistema preposto alla predizioni di rischi per i VRU, basato sulla conoscenza a priori della mappa stradale circostante e su un rilevamento precoce delle intenzioni dei singoli veicoli per una accurata predizione delle traiettorie.

Stato dell'Arte Nel rilevare i VRU a rischio, risulta di fondamentale importanza una precisa predizione delle traiettorie dei vari veicoli che popolano tale incrocio, in modo tale da diminuire i vari false warning (falsi positivi).

A tal fine sono stati condotti diversi studi nella letteratura, che si dividono principalmente in tre modelli:

- modelli basati sulla fisica, dove il movimento é dettato dalle leggi della fisica;
- modelli basati sulle manovre, dove grande influenza hanno le attuali (e future) manovre del guidatore;
- modelli che tengono in considerazione le varie interazioni tra i veicoli.

I diversi modelli di moto presentati nel corso della letteratura, come CV (Constant Velocity), CA (Constant Acceleration), CTRA (Constant Turn Rate and Acceleration), sono stati analizzati da Schubert et al., che ha riscontrato nel CTRA una maggiore accuratezza per predizioni a breve termine. Al contrario, in orizzonti di tempo maggiori, le varie predizioni sono affette da errori elevati, a causa di incertezze e limiti dell'evoluzione dei diversi modelli. Nel corso della letteratura sono quindi state proposte diverse soluzioni che permettono di passare da un modello ad un altro, in modo da descrivere la sua evoluzione nel modo migliore. IMM (Interacting Multiple Model) e SKF (Switching Kalman Filter) combinano le ipotesi di stato da diversi modelli di filtro, per ottenere una stima migliore nel caso di dinamiche mutevoli. Una seconda metodologia consiste nel combinare soluzioni provenienti da traiettorie calcolate mediante modelli che descrivono la dinamica del veicoli e traiettorie basate sulla esatta geometria stradale. Nonostante l'efficienza in real-time, i risultati prodotti da questi metodi possono non essere sufficienti, in quanto le varie manovre possibili non vengono prese in considerazione.
Keep lane (mantenere la corsia), *Change lane* (cambio corsia), *U-turn* (inversione a U) e *overtaking* (sorpasso) sono solamente un set delle possibili manovre che un veicolo esegue. Diversi sistemi MRM (Maneuver Recognition Model), HMM (Hidden Markov Models) o DNN (Deep Neural Networks) sono stati quindi studiati e sviluppati, per integrare tali comportamenti nella predizione delle traiettorie. Nel primo caso, il calcolo si basa su una funzione pesata che combina l'output del CTRA e quello del MRM, basato ad esempio su DBA (Dynamic Bayesian Network). Un filtro Bayesiano (classificatore HMM) puó essere altrimenti utilizzato per rilevare le possibili manovre: una volta calcolati i parametri mancanti, come lo yaw rate, tramite il filtro di Kalman, le osservazioni integrate con le informazioni derivanti dalla mappa possono essere passate al classificatore HMM, in modo da calcolare

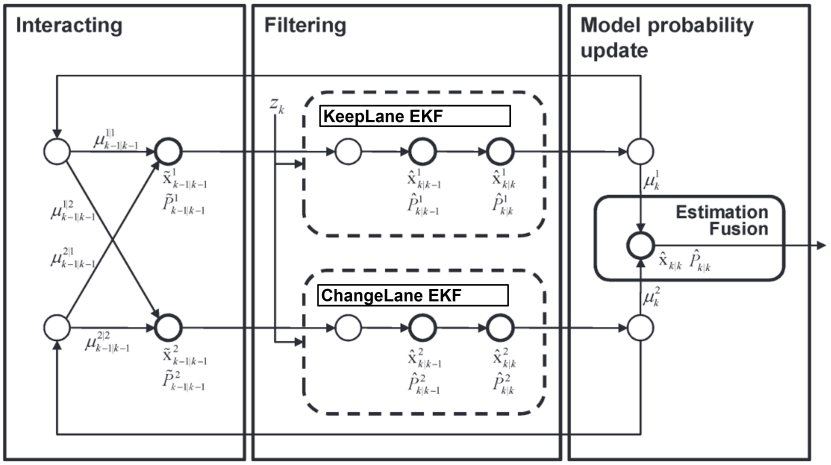
una traiettoria regolare. Negli ultimi anni anche le reti neurali hanno trovato impiego in questo campo: l'integrazione di caratteristiche cinematiche, come la posizione e la velocità, e il contesto ha permesso il riconoscimento delle intenzioni di manovra, senza ricorrere al modelli fisici.
Le metodologie appena presentate risultano però incomplete in uno scenario a traffico elevato, assumendo il veicolo indipendente dall'ambiente e dai soggetti circostanti. Sebbene tali scenari sono pressoché impossibili da trasformare in modelli matematici, studi recenti hanno visto le reti neurali LSTM (Long Short-Term Memory) e CNN (Convolutional Neural Neutworks) giocare un ruolo fondamentale. Tramite l'analisi del comportamento temporale e la sequenza delle coordinate dei veicoli circostanti, la traiettoria dell'ego-vehicle viene calcolata tramite informazioni probabilistiche circa la posizione futura dei soggetti circostanti. Le reti SCALE-Net, seppur non risolvendo completamente la problematica, si propongono di ridurre l'elevato carico computazionale dei sistemi basati sulle LSTM.
La predizione delle traiettorie, come già accennato, risulta essere lo step iniziale per identificare una futura collisione. Il metodo TTC (Time-To-Collision), definito come il tempo necessario affinché due veicoli collidano, supponendo che continuino la loro traiettoria alla medesima velocità, risulta essere assai diffuso. Parallelamente ai metodi binari, che cercano l'intersezione di due traiettorie, sono stati sviluppati metodi probabilistici, che si basano su HMM per calcolare la probabilità di collisione su una porzione di spazio discretizzato, ovvero di condivisione della stessa cella di spazio da parte di due veicoli.

Concetti principali Questa tesi si propone di illustrare un nuovo metodo di predizione delle collisioni basato sulla conoscenza a priori della mappa stradale in cui tale sistema verrà messo in funzione. Il contributo principale di questa tesi risulta essere proprio il ricorso a metodologie computazionalmente non pesanti e di semplici funzioni, con conseguente riduzione dei tempi di esecuzione, garantendo buone prestazione a real-time. Tali vantaggi sono dovuti alla presenza di conoscenze pregresse circa l'ambiente in cui tale sistema viene messo in funzionamento: le cosidette *predefined routes*. Una route predefinita é una traiettoria percorribile da veicoli (non pedoni), nel rispetto delle regole stradali. In prossimità di un incrocio, ad esempio, un veicolo che si trova nella corsia più a destra potrà solamente effettuare una svolta a destra, mentre quello nella corsia centrale procedere diritto oppure svoltare

a sinistra. L'impiego di tali routes semplifica la generazione delle traiettorie di ogni singolo soggetto presente nello scenario.
Ogni soggetto viene infatti descritto tramite uno stato $x_A = [x, y, \theta, v, w, a]$, dove (x, y) indicano la posizione nello spazio bidimensionale attraverso la quale una route viene assegnata, θ l'heading angle, v e a rispettivamente la velocità e l'accelerazione lineare, w la velocità angolare.
Segue la definizione di traiettoria come insieme di punti rappresentanti le future posizioni all'interno di una ben definita finestra temporale. Tale insieme viene generato a partire dalla conoscenza di due informazioni principali: tipo di soggetto (pedone o non pedone) e attuale stato (per i pedoni *Moving* o *Waiting*, altrimenti *KeepLane*, *Curving* o *ChangeLane*). Mentre il percorso discretizzato generato per i pedoni non é sottoposto a successivi vincoli, quello dei non pedoni dipende strettamente dalla manovra riconosciuta. Qualora il veicolo sia in *KeepLane* o *Curving*, l'insieme dei punti é un sottoinsieme dei punti appartenenti alle predefined routes, altrimenti risulta essere un sottoinsieme dei punti definiti da una Curva di Bézier, che descrive il moto necessario per passare dalla route di origine alla route obiettivo.
A questo punto le traiettorie dei singoli veicoli vengono intersecate in uno spazio bidimensionale sulla mappa, per identificare future collisioni, basandosi sul concetto di *area di conflitto* o *conflict area*. Un'*area di conflitto* viene definita come una porzione di superficie, contenente almeno un punto dello spazio, occupata da almeno due soggetti nello stesso instante temporale.
Le traiettorie rappresentabili fino a questo momento tramite segmenti curvilinei, vengono espanse integrando le reali dimensioni del singolo soggetto (larghezza e lunghezza). Infine, se le aree occupate da due soggetti nello stesso istante di tempo hanno almeno un punto in comune, si avrà una collisione.

Riconoscimento Manovra - IMM Il modulo di riconoscimento manovra si basa sull'utilizzo del filtro IMM (Interacting Multiple Model), filtro ibrido sub-ottimo, proposto inizialmente da Blom, che manda in esecuzione diversi filtri in parallelo, mescolando lo stato e la covarianza dei vari filtri, tramite pesi assegnati ad ogni filtro, per ottenere una stima composita dello stato e della covarianza. Il filtro IMM viene implementato tramite un algoritmo ricorsivo, che si basa su precisi parametri in input:

- il vettore delle probabilità del modello, che definisce con quale probabilità ogni singolo modello descriva l'attuale stato,



Schema generale del funzionamento generale del filtro IMM, che illustra i tre step principali del filtro: interazione, filtraggio e update delle probabilità.

- la matrice delle probabilità di transizione, che indica con quale probabilità si passa da un modello i ad un modello j (i termini per cui $i = j$ indicano la probabilità di rimanere in tale modello).
- la definizione dei singoli filtri M e delle loro stime.

Nella figura sottostante viene illustrato il funzionamento generale di un filtro IMM, applicato al caso in esame.

In questo studio, i filtri implementati sono due, uno per il modello $KeepLane$ e il secondo per il modello $ChangeLane$, che integra la velocità angolare. L'output del filtro, che definisce le probabilità di modello, viene quindi integrato con le informazioni provenienti dalla mappa e dall'insieme di predefined routes. In tal modo è possibile l'utilizzo di soli due modelli che descrivono in maniera generale tali manovre, senza dover ricorrere alla reale geometria della strada per riconoscere la terza manovra del set, ovvero $Curving$.

Predizione della traiettoria - Veicoli Il modulo di predizione delle traiettorie genera le future posizioni del veicolo (in questa sezione, si usa il termine veicolo per indicare un soggetto motorizzato o non, utenti della strada, non pedoni), partendo dalla conoscenza della futura manovra.
Qualora si sia riscontrata una $ChangeLane$, il percorso discretizzato nei diversi istanti di tempo viene generato lungo due segmenti di Curve quadratiche

di Bèzier simmetriche. Tale curva viene calcolata durante la prima identificazione del cambiamento di stato, per mezzo dei seguenti control points (punti di controllo):

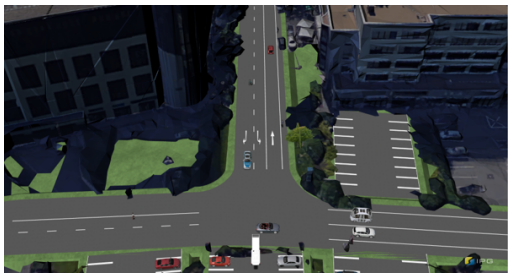
- $P_0(0,0); P_1(c,0); P_2(a,b);$
- $P_2(a,b); Q_1(2a-c,2b); Q_2(2a,2b);$

Dove a dipende dalla velocità del veicolo, b rappresenta la metà della distanza tra la corsia attuale e la corsia finale, mentre c è la sezione aurea di a. In caso contrario, la traiettoria viene generata lungo la route predefinita associata allo stato corrente al veicolo.
Durante la generazione di questi punti viene considerato per ogni istante di tempo un'incertezza sulla velocità (sottoforma di $\pm\Delta v$), in quanto inverosimile mantenere la stessa velocità per la durata di percorrenza di un incrocio, dovuto a manovre come svolta a destra (o sinistra) e alla presenza di altri veicoli.

Predizione della traiettoria - Pedoni Il modello di moto preso in esame per quanto riguarda i pedoni è un semplice CA (Constant Acceleration), applicato allo stato corrente, qualora il pedone fosse già in moto.
Altrimenti, il moto del pedone viene inizializzato e la traiettoria viene calcolata supponendo che esso attraversi in maniera ortogonale la predefined route più vicina.

Predizione della collisione Una volta definito l'insieme delle traiettorie, il modulo di identificazione di possibili rischi entra in esecuzione. Facendo uso di una metodologia binaria, il carico computazione necessita di essere ridotto. A questo proposito, un algoritmo di preselezione analizza quali coppie di veicoli e VRU sono "vicini" abbastanza, da poter rappresentare un pericolo (coppie di pedoni vengono ignorate). Di queste coppie rimanenti, per ogni istante di tempo vengono analizzate le varie traiettorie, rappresentabili come un polinomio interpolato tra i punto della route raggiungibili con una velocità pari a $v - \Delta v$ e $v + \Delta v$. Se l'insieme risultato dall'intersezione dei due polinomio é non vuoto, una collisione viene identificata. Nel caso contrario si procede integrando le reali dimensioni del soggetto (larghezza e lunghezza), trasformando i due polinomi in un porzioni di superficie: se le aree hanno punti in comune, gli oggetti devono essere informati del rischio.

Vista dell'incrocio (GoogleMaps)



Incrocio modellato in CarMaker

Simulazione Con lo scopo di raccogliere ed esaminare i dati per una validazione dello studio effettuato, é stato impiegato il software di simulazione CarMaker, prodotto da IPG Automotive. In questo contesto sono stati ricreati scenari realistici, in cui l'incrocio preso in esame é stato completamente modellato.

A tal fine, una mappa 3D é stata creata a partire da GoogleMaps e dal software open-source Blender, in modo tale da avere a disposizione una base su cui modellare in maniera realistica le varie strade. Ciò risulta di cruciale importanza nella generazione del set di route predefinite: un oggetto campione, dotato di un sensore stradale e configurato per percorrere determinate routes, raccoglie tali dati che saranno poi elaborati da uno script Matlab per la creazione del DB, che include tutte le informazioni impiegate online durante la predizione delle traiettorie.
Ogni scenario, avente come base tale mappa, viene popolato non solo dall'ego-vehicle (piattaforma elevatrice bianca), che coincide con l'origine degli assi del sistema di riferimento preso in considerazione, ma anche da Diversi veicoli, biciclette e pedoni, per simulare situazioni di traffico quotidiano.
Tramite l'API di CarMaker, CarMaker for Simulink (cm4sl), viene effettuata la simulazione in Simulink, dove un subsystem implementa il modulo di generazione delle traiettorie e il modulo di predizione delle collisioni. I risultati vengono quindi mostrati tramite diversi plot.
Una GUI é stata altresì implementata, per agevolare l'user-experience.

Risultati I risultati ottenuti in questa tesi possono essere riassunti nei punti seguenti.

Riconoscimento di manovra La simulazione dei diversi scenari ha permesso di raccogliere dati riguardanti il ritardo tra l'inizio della manovra e l'identificazione di essa. Per manovre con una durata più lunga (circa 5s) e ad una velocità che rientra nei limiti (30 km/h), si ha un ritardo di circa $1.5s$, instante in cui il veicolo non ha ancora attraversato la linea di separazione delle corsie. Nel caso di VRU o biciclette o di manovre in un intervallo di tempo limitato, il ritardo é contenuto nel range $]0.08s; 0.25s[$.

Predizione della traiettoria I risultati ottenuti per quanto riguarda la predizione delle traiettorie possono essere riassunti nei seguenti punti:

- la predizione delle traiettorie per pedoni, in assenza di movimenti improvvisi (difficili da prevedere), é accettabile per predizioni all'interno di una finestra temporale fino a $t = 3s$. Per $t = 4s$, l'errore è pressoché inaccettabile, raggiungendo anche i $e = 12/15m$.

- la predizione delle traiettorie per non-pedoni, in assenza di accelerazioni o decelerazioni sono in generale accurate e affetto da errore accettabile ($e = 3 - 5m$), sia per manovre di KeepLane che di ChangeLane.

- in caso di accelerazioni o decelerazioni improvvise (difficili da prevedere), il calcolo della traiettorie é soggetta ad errori elevati, che raggiungono anche $15/20m$ in un orizzonte temporale $t = 4s$.

Predizione delle collisioni Sebbene il sistema abbia riconosciuto tutte le situazioni di pericolo implementate nei vari scenari, sono stati rilevati numerosi false warning (falsi positivi), causati dalla distanza di sicurezza. Un buon compromesso potrebbe essere in questo caso quello di limitare l'orizzonte temporale a $t = [0.5s; 3s]$.

Conclusioni Come accennato, il contributo principale di questa tesi é lo studio e l'implementazione di un sistema di riconoscimento dei rischi volto principalmente alla salvaguardia dei VRU, tramite l'impiego della conoscenza della mappa stradale (o set di route predefinite, secondo le leggi del traffico stradale). Il modulo di riconoscimento manovra, che integra un filtro IMM e le informazioni provenienti dalla mappa, é stato dimostrato essere capace dell'identificazione di cambiamenti di manovra con buon anticipo, andando a risparmiare tempo in un avvertimento circa un eventuale rischio. Le Curve

di Bezier, inoltre, oltre ad essere computazionalmente non pesanti, emulano il cambiamento di corsia con una buona approssimazione. Il numero di traiettorie generate per ogni veicolo é esattamente una, riducendo drasticamente il numero di traiettorie considerate in altri studi.
Dal momento che questo studio é stato effettuato in un mondo ideale e facendo uso di mappe generate da un software di simulazione, futuri studi si focalizzeranno sulla raccolta di tali informazioni partendo da dati reali. Ulteriori ampliamenti nella parte implementativa saranno poi necessari per includere restanti manovre (come sorpassi o inversioni a U). Incertezze riguardanti la posizione laterale dovranno essere integrate per simulare i diversi stili di guida. L'attuale studio non prende inoltre in esame casi in cui una route si biforca: una possibile soluzione potrebbe essere quella di considerare tutte le possibili traiettorie.
Il lavoro fino ad ora effettuato solamente in un *mondo ideale*, verrà entro l'anno 2021-2022 testato nella realtà, nel contesto del progetto *People Mover*.

Acknowledgments

First and foremost, I would like to thank my supervisor at the University, Prof. David Scaradozzi, whose enthusiasm in teaching and helping students has always impressed me. Thank you for all your support not only during this work but also throughout all these years.
This thesis would not have been possible without my advisors, Markus Wendl and Torsten Hafer. Your inputs and suggestions have guided me over these months. I will never forget the help and the soothing words, during my ups and downs. I would also thank all the Team of Autonomous Reply, headed by Peter Schieckofer, whose passion and energy in these topics have motivated me each day more and more. Hope we can soon meet in Munich for a pizza!
And thank you, mamma and Giacomo. We have been through so many difficulties these years, that left an heavy mark in us, but nevertheless we have been always together (though the 800km distance between us), like today.
Last but not least, I would thank you my boyfriend, Andreas. The man, who could calm me down during my time of stress when I thought I couldn't do it. The man, who supported and supports me each day. The man, who told me "if you start with a PhD, I will disappear throughout all that period".

Contents

Chapter 1

Introduction

In 2020, about 9.8% of the German population rode their bike daily, and almost 25% did at least once per week. Due to lack of an adequate protection such as an external shield and the inability to respond in vehicular collisions, cyclists, as well as pedestrians and motorcyclists, are called Vulnerable Road Users (VRU). In the ITS (Intelligent Transport Systems) they are defined as "non-motorized road users, such as pedestrians and cyclists as well as motorcyclists and persons with disabilities or reduced mobility and orientation".

As not only the number of VRU, but also the number of their deaths continues to climb, reaching 350.000 in 2016[24], many approaches have been investigated to improve their safety. In the past decade, governments made many efforts to implement new regulations not only to reduce drivers' and VRUs' errors but also to increase the awareness of VRU safety problem. Although such initiatives have prevented a relevant number of road crashes, they cannot resolve the root problem, and the number of accidents that involve pedestrian or cyclists, remain high. Hence, in the meanwhile, different measures to improve traffic safety using ITS have been developed, by making the vehicle or the road infrastructure more intelligent. In-vehicle systems use sensors as radar, infrared, camera or LiDAR, to detect a VRU in the environment, calculating a collision risk between the ego-vehicle and the VRU. This features are called V2P (Vehicle to Pedestrian) technologies: side-underrun facilities, blind spot warning devices, pedestrian friendly front-end design air bags, infrared pedestrian monitoring and detection (with Autonomous Emergency Brake system), pedestrian in crossing detection system (based on V2P communication). However, these products cannot handle situation without line-of-sight (LOS), i.e. when parked cars obscure the LOS.

Solutions to increase safety at intersections and crossing have been developed, installing and integrating sensors, such cameras or LiDAR, in the road infrastructure. Information and data gathered by sensors, are then sent to a control unit, that estimates the trajectories of the different road users, to calculate a risk assessment of potential collision. In case of crucial situation, not only the vehicle but also the VRU will be warned.

In this context, furthermore, trajectory prediction plays an important role, to foresee vehicles' maneuvers and lane changing before approaching an intersection, to buy time in VRUs warning in case of possible a potential critical situations.

1.1 Motivation behind the Project

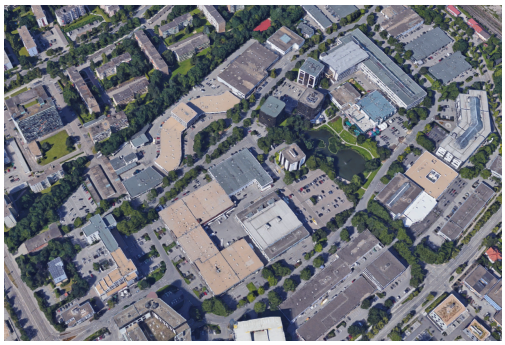
The "People Mover" Project promoted from the city of Regensburg (Germany), targets, among others, these issues. Within this Project, currently (Q2 2021) at its initialization phase, Autonomous Reply GmbH, specialized in innovative Autonomous Things (AuT) solutions, is charged with the task of developing a collision warning system, aiming to inform VRU about critical situations.

The whole infrastructure, comprising sensors, such as LiDARs and cameras, to collect data of traffic participants, and edge devices that communicate with the Cloud, is planned to be firstly installed in a well-defined city quarter, the Business Park (see fig. 1.1), and afterwards to be expanded to further urban areas. The Business Park represents in Regensburg the main and most important location of offices, trade and commerce, with its 155 000 qm rental space. During the 2021, two electric autonomous shuttles are planned to be put into operation in the Business Park.
Autonomous Reply (hence, this work) focuses on the development of the aforementioned collision warning system for VRU for a specific portion of the Business Park, the T-Crossroad in fig. 1.2.

The concept elaborated by Autonomous Reply relies on three sets of data collector systems (sensors-edge devices), located roughly as shown in fig. 1.3, completely independent of each other. This configuration shall avoid the so-called NLoS (Non-line-of-sight) situations.

LiDARs, cameras and GNSS (Global Navigation Satellite System) receivers are connected to the edge device. The first two sensors allow to collect information about each traffic participant, to perform 3D-Object Recognition

(a) Schema of the BusinessPark



(b) Google Maps Screenshot

Figure 1.1: Business Park in Regensburg

Figure 1.2: Google Maps screenshot of the T-Crossroad in the Regensburg Business Park, this work is focused on

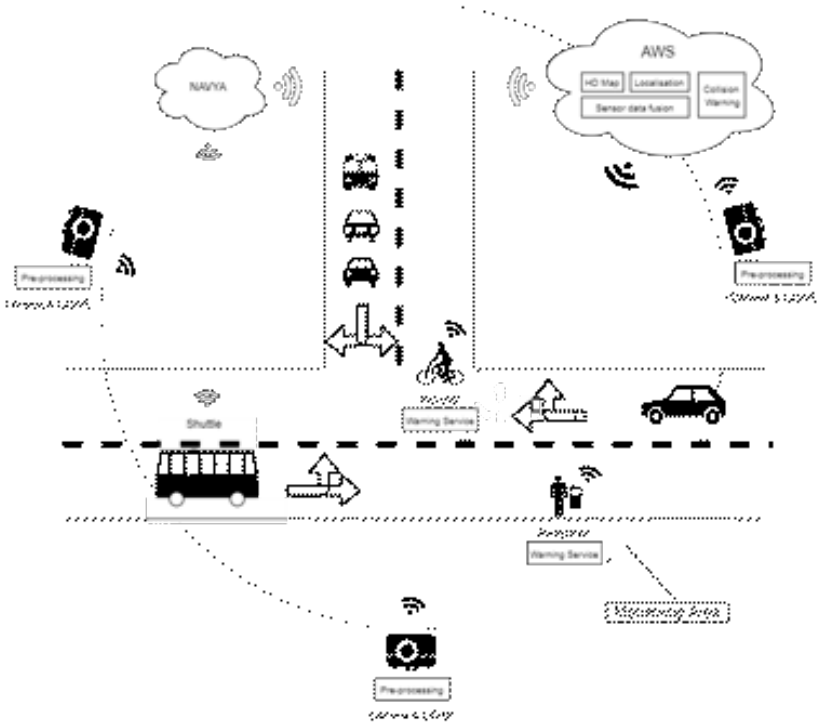


Figure 1.3: Schema of the collision warning service, provided by Autonomous Reply.

(position, orientation, dimension, type) through neural networks. This list of object data is then finalized with the time stamp provided by the GNSS sensor: it defines the detection time for a later comparison with the data coming from the shuttle.

After each device has sent its own object list to the Cloud, the data fusion step is performed. The module responsible for the collision prediction is then triggered, after predicting each object future trajectory, on the basis of the object type (pedestrian, vehicle, bicycle, etc). Is a collision involving a VRU foreseen, the VRU shall be warned through an acoustic and/or optical signal: each VRU may be required to be provided by a smartphone to receive the message.

1.2 Problem Statement

The trajectory prediction problem could fall within the not completely deterministic problem, since an observer could lack of important information such as the driver's intentions or driving style. Many prediction approaches estimate the future velocity and acceleration, as well as heading angle, between certain bounds, through hypothesis about the driver comfort and physics limits (a vehicle does not accelerate or decelerate faster than the engine or breaks allow). The number of possible trajectories can be though high.

This study, conducted within the trajectory prediction and collision warning module of the "People Mover" Project, aims to integrate physical map information, such as the pre-defined routes, generated in accordance to the traffic rules. A route is defined in this context as the pathway traveled by a vehicle, following always the same lane, without any overtaking or changing lane maneuver. Assume a vehicle is entering the crossroad from the top. According to the road traffic regulations, it is foreseen to turn right (its right), when driving in the most right lane. On the contrary, if in the middle lane, it will turn left (as described by the road markings). A vehicle maneuver detection results therefore in the key instrument to early updated the route, hence the vehicle future trajectory.

Given the state (position, orientation and velocities), dimensions and type of each traffic object, the future trajectory in the time horizon shall be generated, in accordance to the pre-defined routes and the maneuver the vehicle is currently performing. The traffic participants predicted paths shall be then overlapped, to identify possible collisions or critical situations, by

detecting possible conflicting areas occupied by more than one object in the same time window in the future.

This study is completely carried out in simulation, making use of Matlab/Simulink and the IPG CarMaker software. As the actual position of the three sets sensors/edge-device is (at the present time) not yet known, no sensor is employed: data collected and elaborated during the different tests are provided by CarMaker, and are assumed to be the ones that are passed in the Cloud to the Trajectory Prediction module after the sensor fusion step. In this context, hence, issues like tracking and latency are not taken into account. The methodology developed is intended to be integrated into the whole system, being fed with the real fused and process data, collected by the sensors and passed through the object recognition module.

To summarize, the key contribution of this thesis is a method to raise collision warnings as a result of the traffic participants trajectory predictions, computed basing on the *a priori* generated routes, in conformity with the map geometry and the traffic rules.

1.3 Thesis Structure

The structure of this thesis is presented here:

- Chapter 2 *State of the Art*: in this chapter a review of the literature of trajectory prediction and collision detection approaches is set out.
- Chapter 3 *Proposal and Approach*: this chapter presents the adopted solution, not only under a more abstract point of view, describing its core concepts, but also in a software architecture perspective, introducing the CarMaker software, to simulate and test the system developed.
- Chapter 4 *Methodology*: a detailed overview about the different techniques used for the trajectory prediction, basing on the traffic object type is discussed in this chapter. The collision prediction is then exhaustively outlined.
- Chapter 5 *Simulation*: this chapter jumps into the implementation, focusing on the software key components.
- Chapter 6 *Results*: after reviewing the different test driving scenario, the experimental results are discussed and evaluated.

- Chapter 7 *Conclusions*: this provides a short summary of the thesis, illustrating the limitations of the methodology and providing suggestions for future works.

Chapter 2

State of the Art

This chapter aims to provide a clear overview of the methods present in literature. The structure of this chapter derives from the one presented in a survey conducted by Lefevre et al. [20], where the different motion modeling and prediction approaches have been organised into three main categories:

- Physics-based model: the motion depends only on physics laws (Section 2.1);
- Maneuver-based model: the motion is also influenced by the driver current maneuver (Section 2.2);
- Interaction-aware model: the surrounding vehicles' maneuvers affects the single vehicle path prediction (Section 2.3).

Finally, an outline about the methodology for the collision prediction problem is presented in 2.4

2.1 Physics-based Model for Trajectory Prediction

The evolution of the state of a vehicle model depends on dynamic and kinematic models. The former refers to Lagrange's equations, that take into account the external forces, to which it is subjected, resulting in complex and large models. Moreover, these forces are measurable only by exteroceptive sensors. ITS systems, therefore, make wider use of the kinematic models,

that rely only on the motion model state: position, steering angle, speed and acceleration. Integrating these parameters with some vehicle properties (such as the weight and its dimensions) as well as external conditions (i.e. the road friction coefficient) the future state can be computed.

Schubert et al. [30], analyzing the several motion models (whose detailed overview is given in section 4.1.1), derived the higher performance of the *Constant Turn Rate and Acceleration* (CTRA) motion model, in comparison with the CV (*Constant Velocity*), CA (*Constant Acceleration*). Though the accuracy of their results is remarkably for short term prediction ($< 1s$) [23], they are not reliable for long-term predictions, since uncertainties and shortcomings of the evolution model are not taken into account.

The issues deriving from the great range of maneuvers, and, hence, of models describing the vehicle motions are solved by Switching Kalman Filter (SKF) and Interacting Multiple Model (IMM) filters [15, 22, 32]. These filters run in parallel a set of Kalman filter, each one implementing a distinct state transition model, that represents a possible evolution vehicle model, and switch between them, basing on a probability and weights system, that selects which model is actually in effect [15, 22, 32].

Polychronopoulos [27] presented, on the other hand, a solution to switch between the different models using heuristics: a path is estimated based on vehicle dynamics and adaptive multiple models, while an alternative path is estimated based on the road geometry; the paths are hence combined based on *a priori* knowledge (see fig. 2.1).

Although physics-based motion models have good real-time efficiency, they are not sufficient to describe changes in vehicle motion due to abruptly maneuvering behaviors.

2.2 Maneuver-based Model for Trajectory Prediction

The *Maneuver-base models* are based on a early recognition of a driver maneuvers (executing lane change, turning left, performing a u-turn, etc.). The estimation of the driver intention, hence, leads to the generation of the future path corresponding to the observed maneuver. Maneuver Recognition Models (MRM) are frequent in the literature and they usually take advantage of a fully-defined road mathematical representation [32, 12, 34]. Besides, a

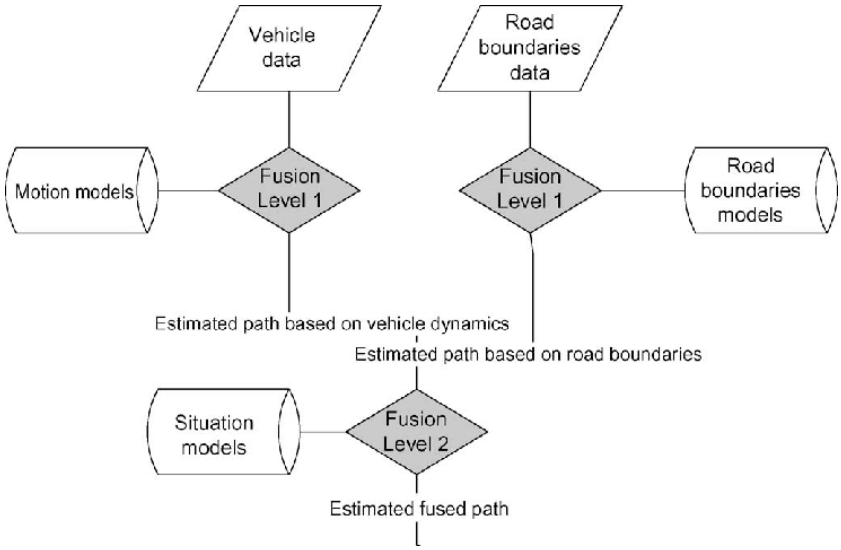


Figure 2.1: Structure of the fusion process for the estimation of the path [27].

classification of a wide range of maneuver can be performed using discriminative learning algorithms, such as MLP (Multilayer Perceptron), or SVM (Support Learning Machines). A further common approach focuses on HMM (Hidden Markov Model), that breaks down each maneuver into a chain of consecutive events, whose transition probabilities are learned from data.

2.2.1 Maneuver Recognition Model

Toledo-Moreo [32] introduced an IMM-based lane change prediction method to predict lane changes in straight and curved roads with short latency times in highways. As the set of maneuver is limited to a two-element list, *ChangeLane* and *KeepLane*, the IMM is configured with two EKFs. Different state transition functions are though fed to the EKF for straight and curved road stretches: in the second case, the road is defined through a $y = c_1x + c_0$ equation and the knowledge of the road parameters is required. This drawback is significant in scenarios where the road has many curvatures. This IMM-based algorithm could foresee a change lane between 1s and 1.5s before the vehicle fully crossed the lane line.

Houenou et al. [12] presented a Maneuver Recognition Model (MRM), to detect the current maneuver (keep lane, change lane, turn), based on an

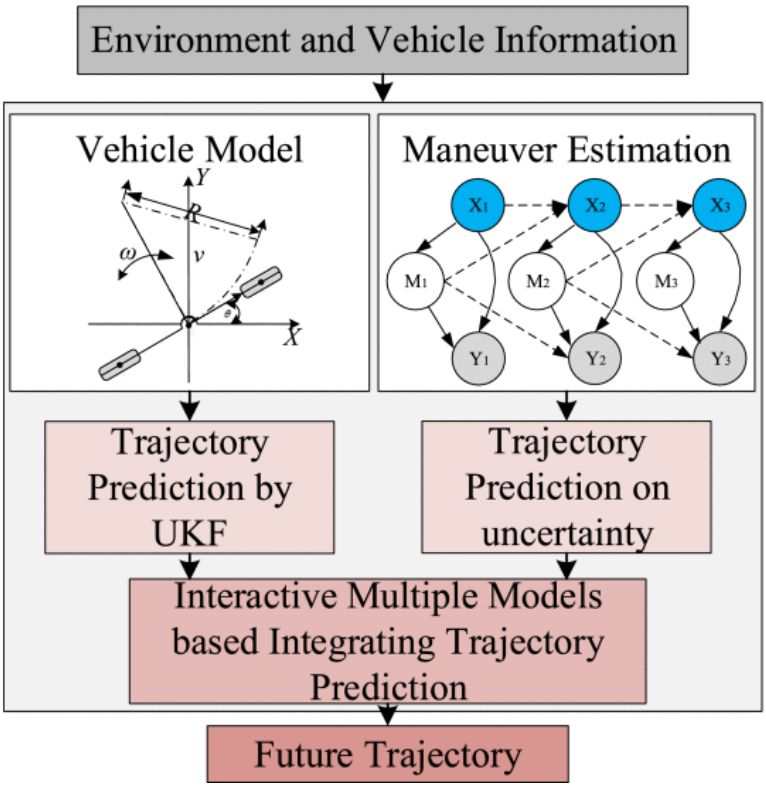


Figure 2.2: Integration of physics- and maneuver-based trajectory prediction models [34].

early detection of the lane where the driver is intending to. A weight function combined, then, the trajectory computed by a CTRA motion model and the one derived by the recognized maneuver, whose complexity grew significantly in case of non constant road curvature.

A combination of a physics-based prediction (CTRA) with a simple MRM based on a Dynamic Bayesian Network (DBA), whose parameter are learned from realistic driving data [34]. The IMMTP (Interacting Multiple Model Trajectory Prediction) exploits the first method for short-term prediction, while referring to the second one for longer time horizon. A schematic representation is shown in fig. 2.2.

2.2.2 Hidden Markov Models

Drakoulis et al. [29] predicted intersection crossing trajectories by means of the vehicle kinematics, the road topology and the short-term maneuvering intention, limiting the vast space of feasible future trajectories. Once the Kalman filter has calculated missing parameters, such as the yaw rate, the observations, integrated with the localization and map information, are passed through a Bayes filter (HMM classifier), to detect possible maneuvers and a smooth long-term trajectory is derived. The final predicted trajectory is derived through the combination of the the short-term prediction (CTRA motion model) and the intention-aware qualitative prediction. This approach allows to predict driver intention of turning left or right on circa 3.5s, before entering the turn.

2.2.3 LSTM and RNN

Rule-based approaches, that simulate lane-changing behavior according to a series of traffic rules, are widely used, but the model performance is affected by the difficulty to describe the pattern of a driver potential decision. In the last years, much progress has been made in this field through deep neural networks: RNN (Recurrent Neural Networks) and LSTM (Long Short-Term Memory) are the most common [1, 8, 25, 26, 35]. LSTM is composed of three gates and this internal gate mechanism can regulate the flow of information by enabling timeseries analysis: the gates can learn which data in a sequence is important to keep or throw away. [35] presented a method to predict the driver intention based on a LSTM, analysing a combination of kinematic features (position, speed, heading) and location context, without using a physical based model for prediction. The characteristics of each maneuver contributed to the maneuver classification: for instance, a driver turning across traffic will slow down and move to the center of intersection and a driver turning left will generally move to the outside of the road and slow down in preparation for the turn. RNN and graphical models have been used as well to predict the future categorical driving intent, for lane changes in highways, up to three seconds into the future given a 1 second time-window of past position and speed measurements [26].

In 2019, LSTM, GRU (Gated Recurrent Units, similar to LSTM without handling hidden states) and SAE (Stacked Denoising Autoencoder, composed of networks trained as autoencoders, learning efficient coding of unlabeled

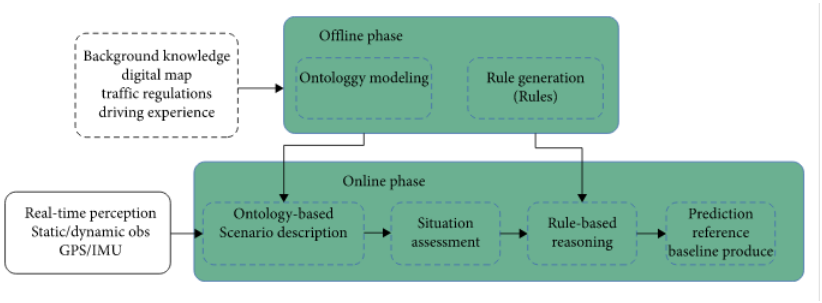


Figure 2.3: The architecture of the proposed prediction reference baseline determination method [33].

data and validate the encoding by attempting to regenerate the input) have been compared in [21]. LSTM and GRU have demonstrated to have higher performances (recall, accuracy and precision) than SAE. Furthermore, as the sequence length grows, the performances of LSTM an GRU improve: the two result in higher accuracy respectively for long-term and short-term sequences.

A Lane-Attention [25], based on LSTM, has been recently introduce. It treats lanes as a graph and uses attention mechanism to aggregate the static environmental information. The model is trained to learn drivers' intention, manifested as the different levels of attention scores, that could be passed to a subsequent trajectory prediction.

A complete trajectory prediction approach based on Knowledge-Driven LSTM in urban roads was presented in [33]. A first offline phase establishes the ontology model "human-vehicle-road", extracting the main characteristics of the road network and traffic participants, and the behavior prediction rule, using the background knowledge about the traffic regulations and the driving experience. A Prolog reasoning system matches these rules with the actual knowledge (ontology model obtained by the factual scenario), inferring the driver intent. Finally, the trajectory prediction phase uses the LSTM to learn the continuous features of the historical trajectory on the basis of the current intent and then generates the future path (fig. 2.3).

2.3 Interaction-aware Model for Trajectory Prediction

The previous categories assume that the vehicles move independently from each other. This assumption falls in case of *Interaction-aware Model*, where inter-vehicle dependencies can be modeled with Coupled HMMS (CMMHS) [5], that analyse pairwise dependencies between entities. The number of possible pairwise grows quadratically with the number of traffic participants, hence the complexity is not manageable. LSTM are also widely used in this context: a small outline is given hereafter.

2.3.1 LSTM and CNN

The first study aiming to solve this issue using LSTM predicted single vehicle trajectory at each iteration, leading to a significant computational load. [16] analyses the temporal behavior and predicts the future coordinates of the vehicle, feeding the sequences of vehicle coordinates obtained from sensor to the LSTM and producing probabilistic information on the future location of the vehicles. Such an approach, exploiting the vehicle states, can efficiently capture the single vehicle maneuver, but the computation time and the prediction error increase with the number of vehicle. [13, 19], on the other hand, take advantages from the occupancy grid map in a traffic scene, making use of CNN (Convolution Neural Networks) and LSTM, to predict the future occupancy of each cell. By using integrated driving scene input, the model is powerful to capture scene-level interaction context, even in dense traffic situations, but they are still computational heavy for a single iteration.

2.3.2 SCALE-Net

With the aim to guarantee the consistency in terms of accuracy and computational load, a fully scalable trajectory prediction network that models inter-vehicle interactions, SCALE-Net is proposed in [14], based on edge-enhanced graph convolutional neural network (EGCN) [9] and LSTMs. Historical states of the vehicles are used as input of the overall system. Though this model can be applied to any level of traffic complexity with low computational resources, SCALE-Net cannot consider the road structures, one of the most prominent features of traffic scenes. An overview of the architecture

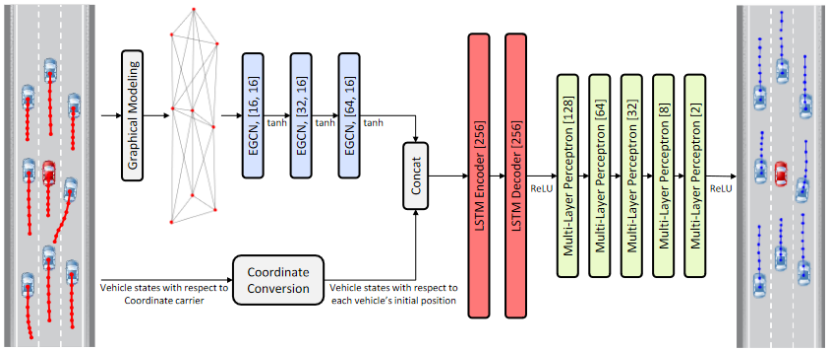


Figure 2.4: Overall architecture of SCALE-Net for the trajectory prediction algorithm [9].

employed in fig. 2.4 is only for information purposes.

2.4 Collision Prediction or Risk Assessment

One of the major challenges after predicting trajectories is to detect possible critical situation, in order to react accordingly to mitigate or to fully avoid accidents.

2.4.1 TTC

The well-known TTC (Time to Collision) [11], defined as "the time equired for two vehicles to collide if they continue at their present speed and on the same path", finds many application in the literature as risk indicator: the lower the TTC, the higher the probability of collision. Kim et al [17] computed the TTC comparing the entrance and exit time of two vehicles into and out of the conflict area (region of the space where the two areas overlap) in curvilinear coordinates (fig. 2.5), after calculating the future trajectory using quintic polynomial. This work showed acceptable results only for non long-term prediction, as uncertainties on the future are not taken into account.

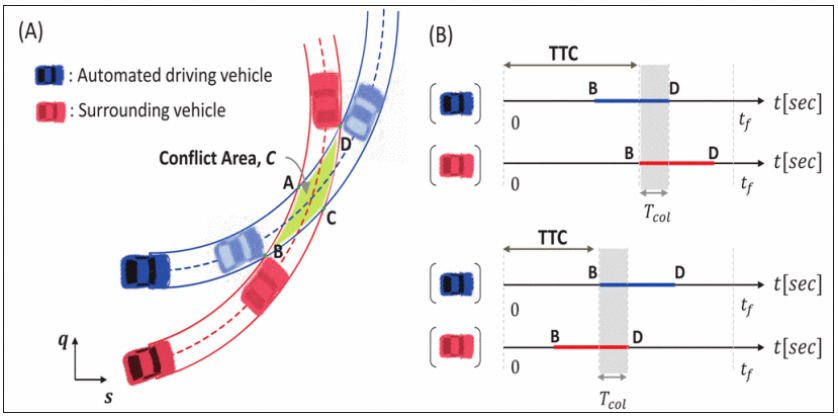


Figure 2.5: 5th degree polynomial trajectory based TTC computation method: entrance time and exit time in the conflict area are compared [9].

2.4.2 Binary and Probabilistic Methods

The simplest, but computational complex method to evaluate a potential risk is to find the intersection between two trajectories, once the linear differential equations of the motion models (*Physics-based model*) have been solved. Another solution is to approximate the trajectories with linear piecewise segments or to discretize the trajectories, checking iteratively for a collision at each discrete timestep [3]. Vehicle dimensions are fundamental in a risk assessment, and usually it is represented as a polygon [6], or ellipse if the uncertainties on the measurements is known [2].

These methods provide only basic information such as whether, where and when a collision will occur, while new approaches based on HMM supply details such as the probability and the severity of the risk. These studies compute the collision probability on a discretized position space, exploring the probability that two vehicles share the same cell. In [10] a two layer reasoning computation architecture measures the collision probability as the percentage of overlap between the future trajectories represented as geometric shapes. This early warning system has been demonstrated to work in real-time even in complex-scenarios. [8] applied a monitoring algorithm for collision prediction based on VANET (Vehicle ad-hoc networks, whose nodes are vehicles, created "*spontaneously*" for data exchange between vehicles). After computing the vehicles' uncertain trajectories using LSTM, the uncertainty regions occupied by two vehicles at time t are compared: the dimension

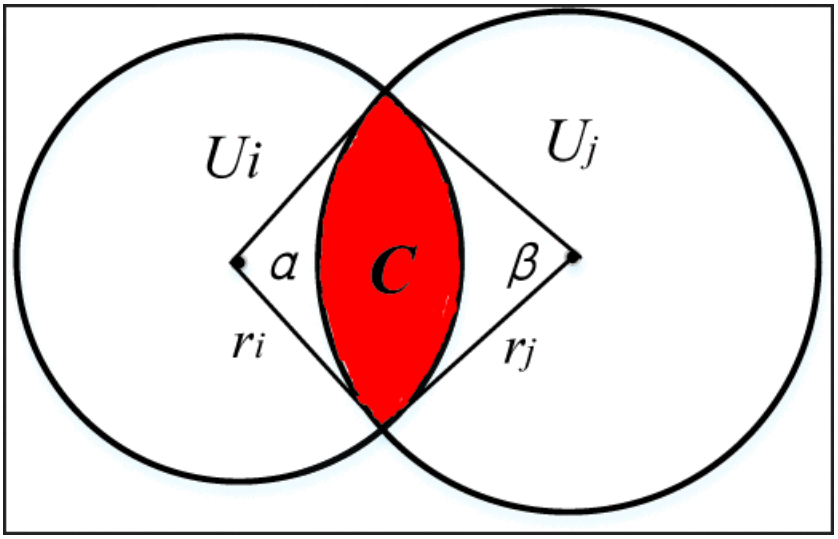


Figure 2.6: Example of two uncertainty regions intersecting: a collision is therefore foreseen [8].

of the intersection area defines the amount of risk (fig. 2.6).

Chapter 3

Proposal and approach

This chapter presents an overview of the adopted approach: after predicting potential future trajectories, through a method that falls within a mixture of physical-based and maneuver-based motion prediction, for all the moving entities in the scene, the risk is estimated by analysing pair of traffic participants and their trajectories. This requires predicting traffic participants trajectories in reference to the exact geometry of the road and the current maneuver that a vehicle intends to perform. After discussing (Section 3.1) the solution and its reasons behind in a more mathematical and abstract manner, the software implemented is described under an architectural point-of-view (Section 3.2). Section 3.3 introduces, finally, the IPG CarMaker software, that allows to validate the implementation and the methodology, simulating different driving use cases.

3.1 Collision warning with map-based trajectory predictions

This section introduces the core concepts of this work under a mathematical point of view. Afterwards, the prediction strategies are outlined more in details, jumping into the proper workflow.
Advantages have been taken of the knowledge of the road geometry, allowing the generation of trajectory predictions along the predefined routes, expressed as a set of points occupied by an actor over a time horizon. The current maneuver of a traffic object (in this specific case, pedestrians are excluded) plays an important role in the trajectory generation. If the space regions,

occupied at a certain time step by a pair of actors, have at least one point in common (*conflict area*), a collision is foreseen.

3.1.1 The core concepts

Assume that the crossroad is populated by dynamic actors, such as VRUs (pedestrians, cyclists or skaters) and vehicles (cars, trucks, buses, etc.) and that the system has no kind of control over the state of each actor. Assume that the world is described by a two-dimensional space modeled by a top-bottom view of the real world.
The state of an actor A is a vector $x_A(t) \in \mathbf{R}^n$ as a function of time t that encodes the properties of actor A at time t. The n-dimensional vector is in the form of

$$x_A = [x, y, \theta, v, w, a] \tag{3.1}$$

holding the position of the actor (x, y), the direction (or heading angle) θ, the scalar velocity and angular velocity, as well as the acceleration.
The set Ψ, called predefined route, is a well-known set of points

$$\Psi \in \mathbf{R}^2 \tag{3.2}$$

The one-dimensional trajectory of an actor over time

$$P_A(x_A) = \{(x, y, t) : (x, y) = P_A(x_A, t), t \in T\} \subseteq \mathbf{R}^2 \times T \tag{3.3}$$

is a set of poses of the actor A within a future time interval T, as a function of the current state x_A.
Future trajectory poses of 2-4 wheeler actors are further restricted to relies inside the predefined routes set of points:

$$P_A(x_A) = \{(x, y, t) \in \Psi, t \in T\} \tag{3.4}$$

It is essential to consider also the region of the two-dimensional space

$$\Omega(P_A(x_A)) \subseteq \mathbf{R}^n \times T \tag{3.5}$$

physically occupied by an actor over time, as a function of its trajectory P_A.
The pair of actors $\Upsilon_{A|B} = \Omega(P_A(x_A)), \Omega(P_B(x_B))$ is the collection of the 2D trajectories of two actors.
A collision of between $\Upsilon_{A|B}$ is detected if the set of points defining the conflict area $\Phi_{\Upsilon_{A|B}}$ is not empty.

$$\Phi_{\Upsilon_{A|B}} = \{\Omega(P_A(x_A, t)) \cap \Omega(P_B(x_B, t)), t \in T\} \neq \varnothing \tag{3.6}$$

3.1.2 The applied solution

When predicting vehicle trajectories, it is reasonable to assume that all observed vehicles will behave according to traffic rules. Their movements are confined to the boundaries of the road and their trajectories can be segmented into smaller path, characterized by a specific starting and end point. Consider a group of vehicle driving in the crossroad of this study. The driven trajectories, ignoring in this case overtaking or lane change maneuvers, would create a graph G = (E, N), where N are nodes (or vertices) and E edges (or segments) of the graph G, and delineate the predefined routes. For this purpose, the crossroad is modeled in CarMaker (sec. 3.3), exploiting its capabilities of simulating real scenarios: the generation of the business park 3D map and the accurate replication of the road geometry made the process of recreating the real map simple and detailed.

Firstly, it is necessary to emphasize the distinct approaches regarding the trajectory prediction methodology applied to pedestrians and to 2-4 wheelers, due to their complete incomparable dynamic motion models and movements. A clarification about the workflows is presented in fig. 3.1. Actors equipped with wheels (vehicles, bicycles, motorcycles) move along the map routes, that are always characterized by a polynomial, calculated during the predefined route generation step. The future pose at a specific time step $t \in T$, that is going to define the trajectory $P_A(x_A)$, is calculated as a function of x_A. In other words, given initial position, the current velocity and the acceleration, the final position on the route after t is known.

Realistically seen, a vehicle performs not only the so-called keep lane maneuver, but often changes its route, path (change lane maneuver), impacting the trajectory prediction. During this operation, it deviates from the course of the predefined road map, requiring a different methodology for the trajectory generation. Many trajectory planning techniques for vehicles lane change had been discussed in the literature. Among them, B-spline curves and clothoid curves are widely used, but these methods could cause a complex calculation and the maximum lateral acceleration was hard to control. A further approach based on Bezier Curve (see fig. 3.2) was presented in 2013 by Chen et al. [7] : they demonstrated how this kind of curve could be solved quickly and efficiently, satisfying real-time requirements. In this study, this methodology is adopted, resulting in generated trajectories following Bézier Curves in case of lane change operations (fig. 3.3).

The lane change maneuver detection plays therefore an important role:

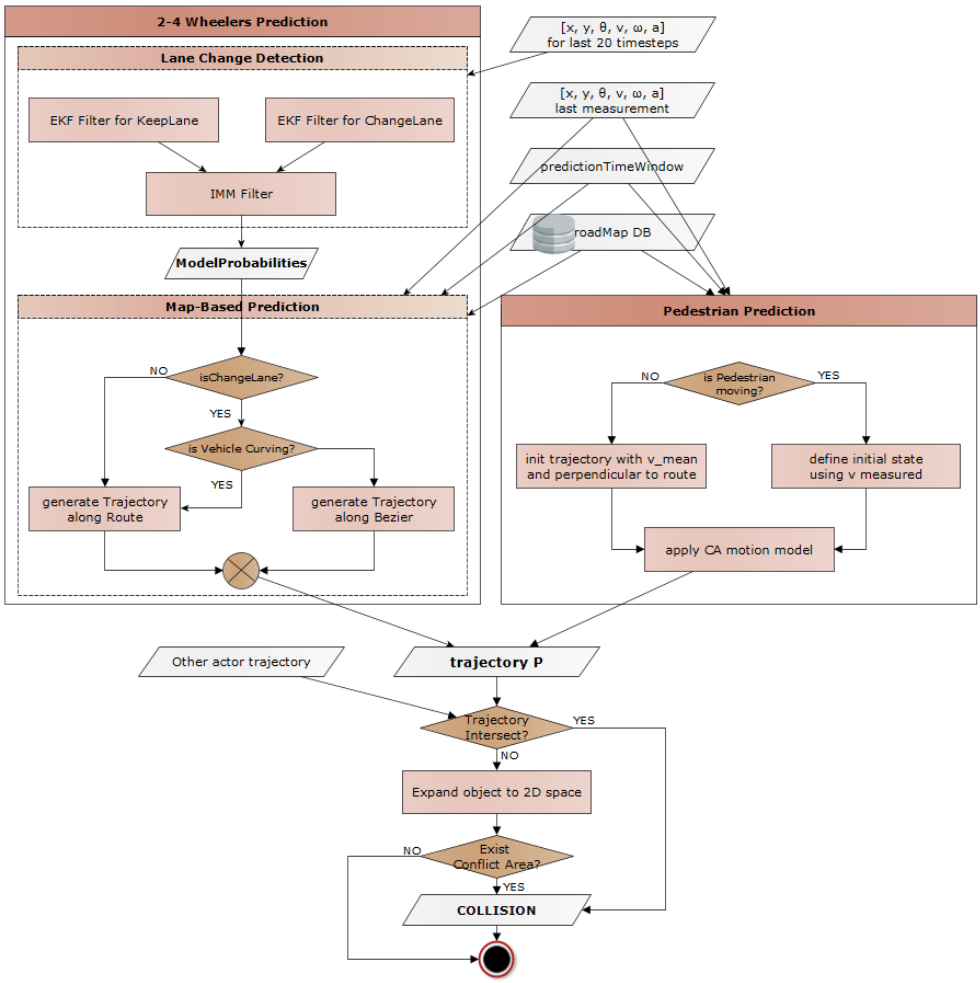
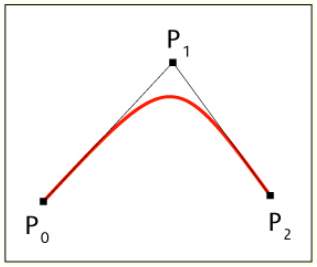
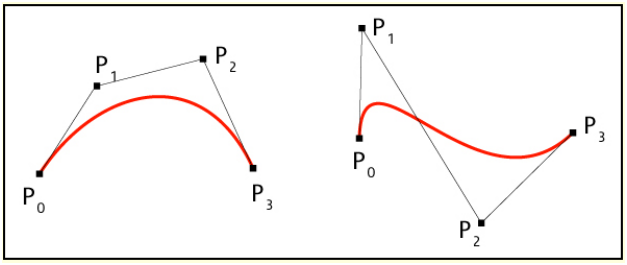


Figure 3.1: Flowchart describing the work-flow adopted for the pedestrian and vehicles/bicycles predictions, followed by the main steps of the collision detection process.



(a) Quadratic Bézier Curve



(b) Cubic Bézier Curve

Figure 3.2: Example of Bézier Curves. On the left (a) a quadratic Bézier Curve, defined by its 3 control points. On the right (b) a couple of cubic Bézier Curve and their control points

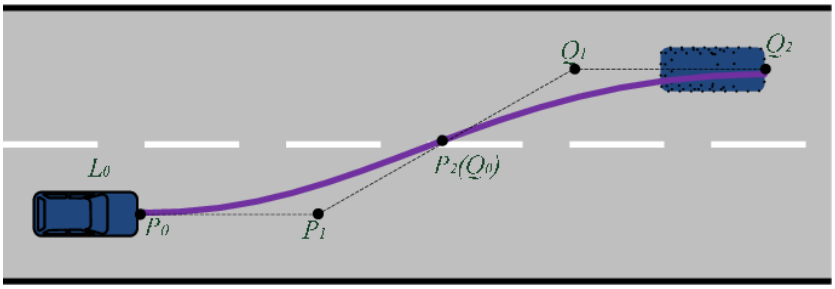


Figure 3.3: Example of a Bézier Curve used in path planning for a lane change maneuver. Source: *Lane change path planning based on piecewise Bézier Curve for autonomous vehicle.*

the sooner it is observed, the sooner the trajectory prediction can be updated. This task is carried out by an Interacting Multiple Model (IMM) filter that computes the probabilities that the motion model describing the lane change maneuver is in effect at current time step, against the lane keeping model. The filter solves the problem of using a single motion model the wide range of vehicle maneuvers, running a set of filter in parallel, combining their states and covariances and calculating the model probabilities. In this study, a pair of EKF (Extended Kalman Filter) are configured with respectively a state measurement motion model for the keep lane maneuver and for the change lane maneuver. The model probabilities indicates which curve (Bézier Curve or the road map) the actor is going to follow.

The generation of a pedestrian trajectory closely depends on the his current movement. When standing alongside the road, waiting for crossing, his velocity may tend to zero. The generated path, basing on its velocity, would result in a cloud of point near the current pose. Thus, his velocity is assigned an average velocity and the pedestrian is assumed to move perpendicularly to the road (crossing the street). If the state x_i already defines a velocity, that does not tend to null, and an heading angle, the future positions are predicted basing on his state x_i. The CA motion model applied is described in the next chapter (sec. 4.1.1).

After collecting all the trajectories of the actor populating the crossroad, the generated paths for each pair of actors $\Upsilon_{i|j}$ are intersected, searching for a common point in the set of trajectories. This shall be reached by both the traffic object at the same time step t. This method is anyway not enough, as it considers only the center of the mass of the vehicle. Each 1D trajectory (describing the path of the center of mass of the actor) is then transformed into a 2D area Ω_i, by means of the actor's real dimensions. The space sections $\Omega_{i|t}$ and $\Omega_{j,t}$ at each time step $t \in T$ are overlapped, to find a conflict area. If for any t the conflict area contains at least one point, a collision is detected.

3.1.3 Motivation

The specific real world scenario and the knowledge of the road map played an important role in elaborating the aforementioned approach: the awareness of predefined routes have considerably simplified the trajectory prediction step, without the need of over engineered calculation with heavy computational cost. Therefore, although interesting and maybe applicable in future studies and works following this thesis, methodology based on neural net-

works, and especially Long-Short-Term-Memory neural networks, have not been involved.

It's worth mentioning that many solutions in the literature have been applied to create such a map graph. Quehl et al. [28] proposed to generate this graph using the accumulated trajectory data coming from a set of vehicle equipped with sensors for accurate tracking. Hence, the possible paths were infered: the recorded trajectories were first converted to a top-view gray-scale image and, after removing short branches (noise), reduced to simple lines, through image processing and agglomeration algorithms. A further approach involves online maps, that allows the extraction of information about the surrounding road structure, such as OpenStreetMap. However, these maps could be affected by a non precise geo-reference, such that accurate lane depiction could be missing, as well as the number of lanes.

The always varying geometry model of the map, along the routes, would impose a great amount of motion model configured in the IMM filter (almost one for each curved road segment). Thus, the integration of the map definition, that enhance and improve the differentiation between a lane change maneuver and a turn maneuver, that would generate completely distinct path.

The choice of employing CarMaker relies on two main reasons. The first one is based on the intent of running the overall system in simulation, without physical vehicles. The second one is originated from the drawbacks of online street maps.

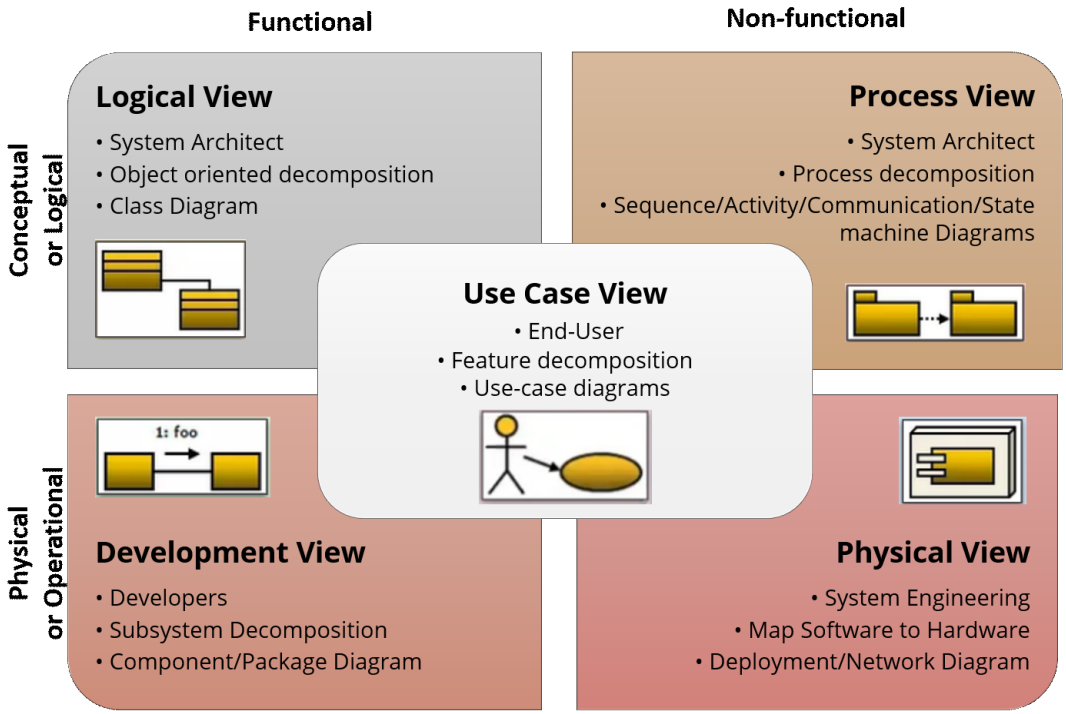
3.2 Software architecture: 4+1 Architectural View Model

The "4+1" View Model for describing the Software Architecture was introduced by Philippe Kruchten in 1995 [18]. The purpose of this view is to provide five concurrent views, to capture and illustrate the different aspects of the system design. The "4+1" View Model gets its name from the 4 primary views and 1 supporting view that are used:

- Logical view: to give an overview of the functionality.
- Development view: to statically describe the system on a developers' perspective.

- Process view: to illustrate the communication between the different processes.
- Physical view: to outline the services' deployment.
- Use Case view: the supporting view, aimed at describing the functionality of the system from a user perspective.

These multiple views address separately not only the concerns of the various "stakeholders" of the architecture (end-user, developers, etc.), but also the functional (logical and development views) and non-functional requirements (process and deployment views).



3.2.1 The Logical view

The Logical view captures the functionality of the system in terms of its static structure and dynamic behavior: through the decomposition of structural elements, functional requirements are described. Since a UML Component can effectively depict component-based systems, the package diagram in Fig. 3.4 was deemed appropriate to show the arrangement of the model elements.

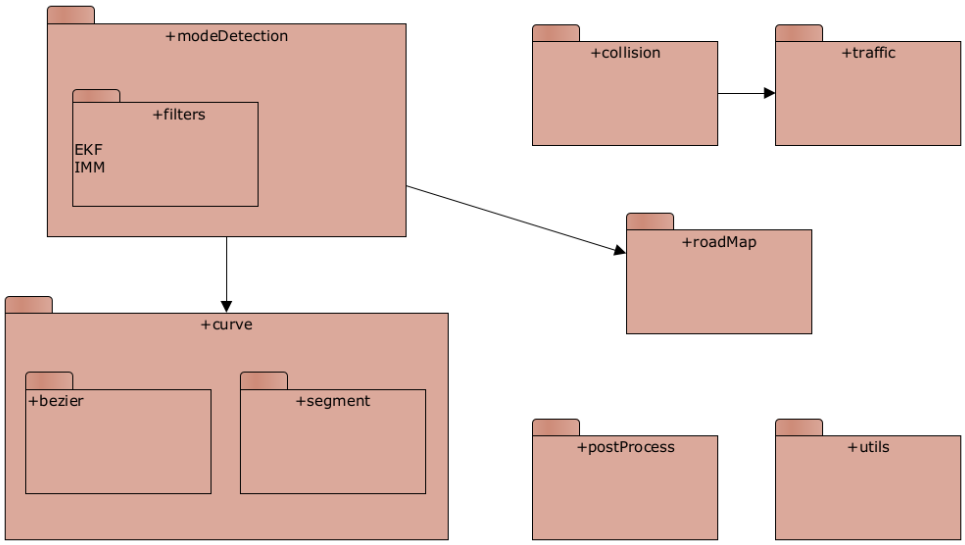


Figure 3.4: Package Diagram that show the organization of the packages

At a first glance, it appears clear that the organization of the software is kept simple. Each package is as much as possible independent from each other and develop a specific functionality of the software.

3.2.2 The Development view

The Development view describes the system from a programmer's perspective. It is concerned with the organization of physical code, its main modules and their dependencies. Class diagrams are the most suitable diagrams for this view, as they provide a good overview of the code structure, modeling classes, their component parts and the relations between one another.
The different types of objects implemented within this system are shown in figure 3.5.

All the classes, excepting the `IMM` and `EKF`, are Simulink Busses, analogous to a structure definition in C: they defines the number, the properties, such data type, and the configuration of each bus element. Easing and automatically validating the data flow during Simulink models simulations, especially where Matlab functions are used, `Simulink.Bus` data type has been choosen for all the other objects.

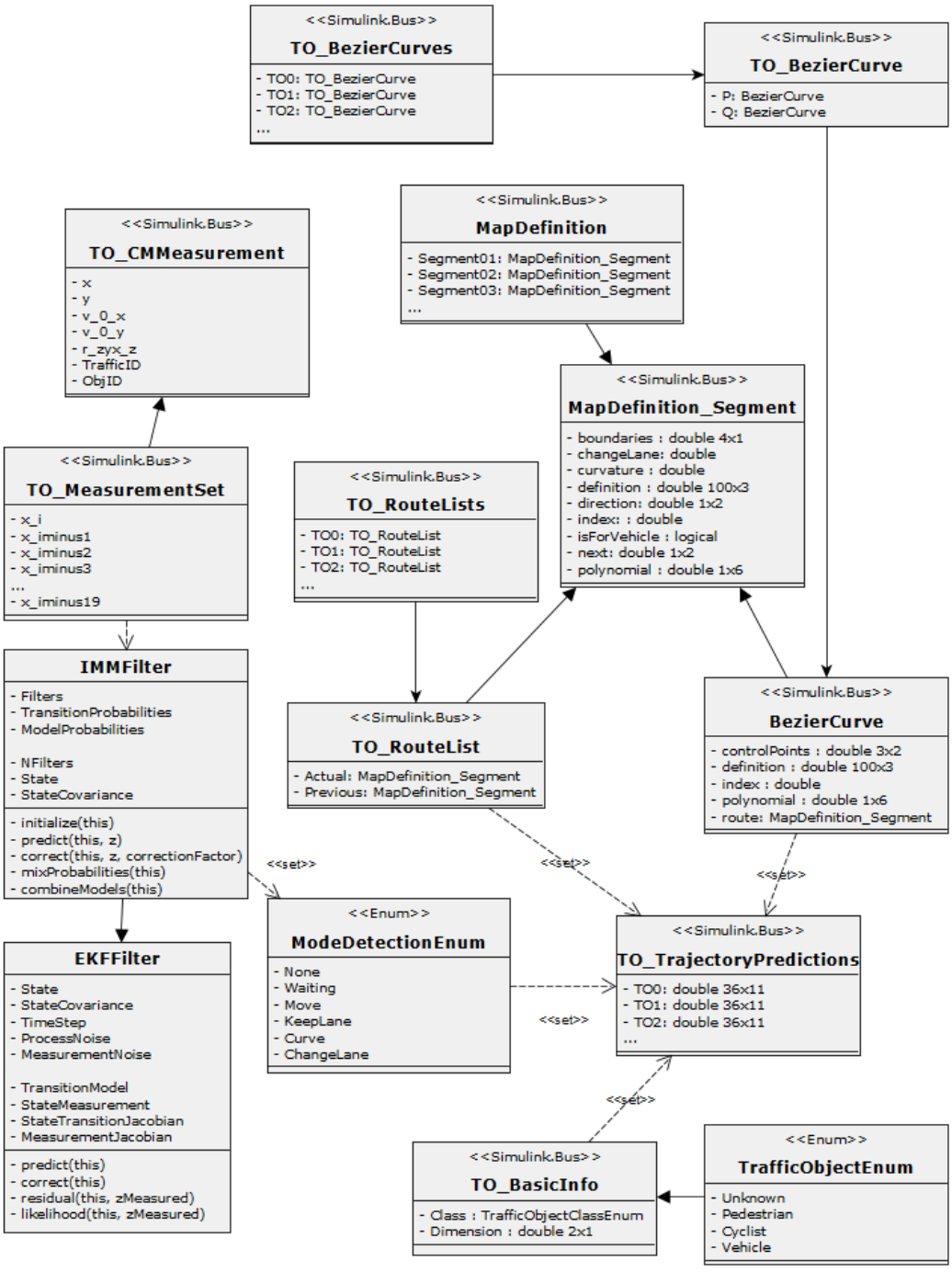


Figure 3.5: Class Diagram for the development view, providing a code structure description

3.2.3 The Process view

The Process view focuses on the run-time behavior of the system and the elements of the system that relate to process performance, as well as the flow of information. While the general workflow is usually illustrated through states and transitions diagrams (such sequence diagrams or activity diagrams), non-functional aspects (like scalability, throughput, and process response times) can be easier put in words than in diagrams.

Such diagrams present the system for a dynamic point of view, unlike class diagrams, that focuses on the static definition and description of the different component parts.

The activity diagram presented in figure 3.6 documents the implementation of the collision warning system process. The process can be roughly divided into two parts: the first one for the trajectory prediction and the second one that basically predicts the collision. The former step is comprises two flows of activities, depending on the class (type) of the detected object (is it a person or a vehicle?): in case of pedestrians, a simple CA motion model (see section 4.1.1) is applied. For objects with wheels (vehicles, bicycles, roller, etc) the prediction starts with the detection of the driving mode (*keepLane* or *changeLane*) using two EKFs (one for each motion model) that runs parallel within an IMM Filter. This defines the probability that a certain model (keepLane or changeLane) is in effect at a specific time step. Given the probabilities and the road geometry, the trajectory can be calculated for many time windows in the future. After defining all the trajectory points for each time step, the second (and final) step can begin: the trajectories calculated until now, are those of the center of mass of the object, not of a 2D object. For this reason, the width and height of each object are integrated to calculate if a conflict area exists at each time step.

3.2.4 The Physical view

The Physical view shows the system from a system engineer's point-of-view, representing the deployment layout or infrastructure of an application. The software components are mapped across the hardware including computers and devices. Standard options include UML deployment diagram or Network diagram.

In this case, the deployment diagram (3.7) is really simple, as the system runs only in simulation mode on the laptop: the Simulink model is respon-

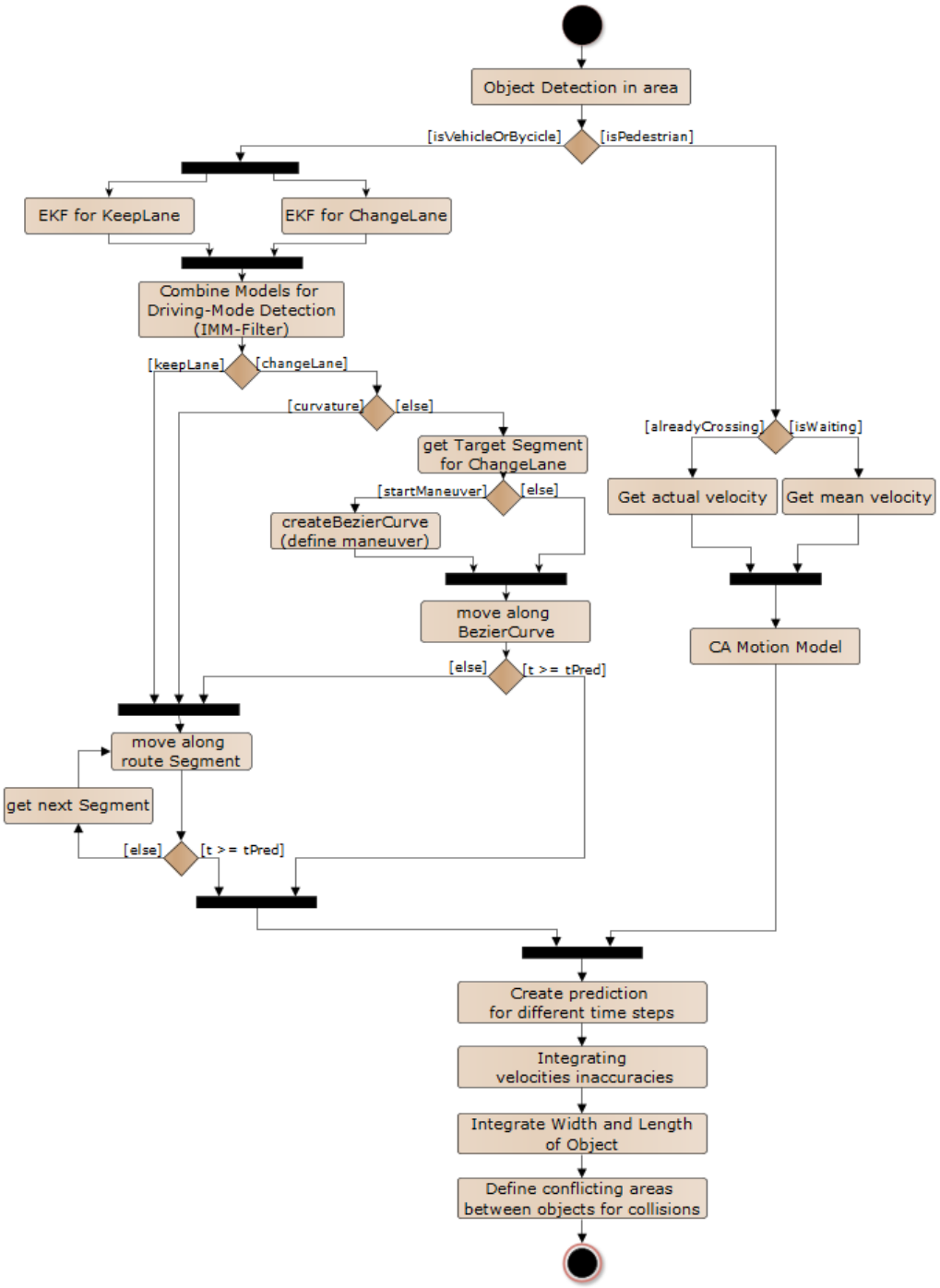


Figure 3.6: Activity diagram that illustrates the system dynamic behavior

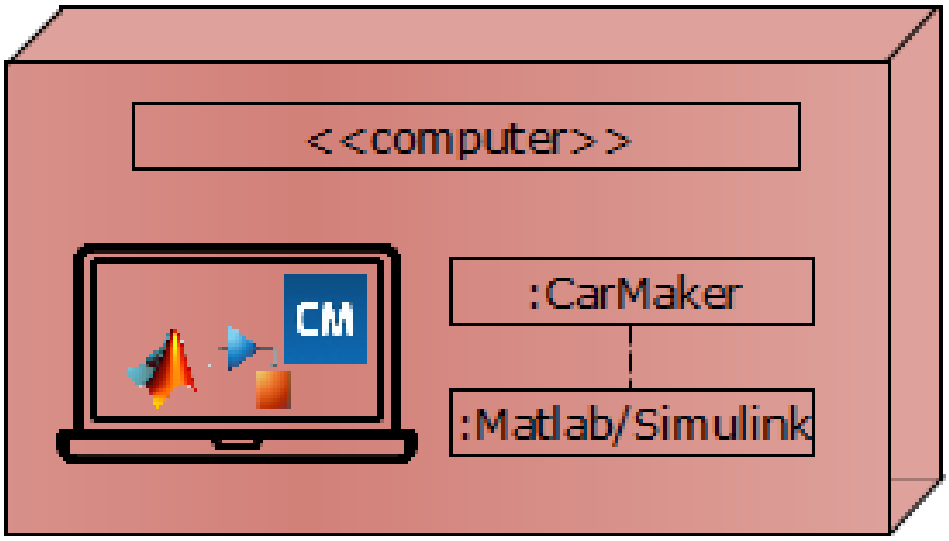


Figure 3.7: Deployment diagram with the CarMaker and Matlab/Simulink

sible for the trajectory prediction and collision warning computation, that is executed real-time during the simulation in CarMaker, using the CarMaker for Simulink integration.

3.2.5 The Use Case view

A Use Case diagram, used for high level of design, specifies the events of a system and their flows, without referring to the implementation. The representation of the externally visible system behavior leads to a great effectiveness in communicating the system behavior in the users' perspective, main purpose of the Use Case view.
The use case diagram in figure 3.8 depicts the general collision warning system behavior.

Two types of actors (on the left side) are supposed to interact with the system: pedestrians and vehicles/bicycles: a single use case can be populated by zero or more of each kind of actors. The third one (on the right) is the external system, tasked with the real warning to the VRU. It is directly connected to the end goal ("find conflicting areas"), that is reached through the different actors' trajectory predictions.

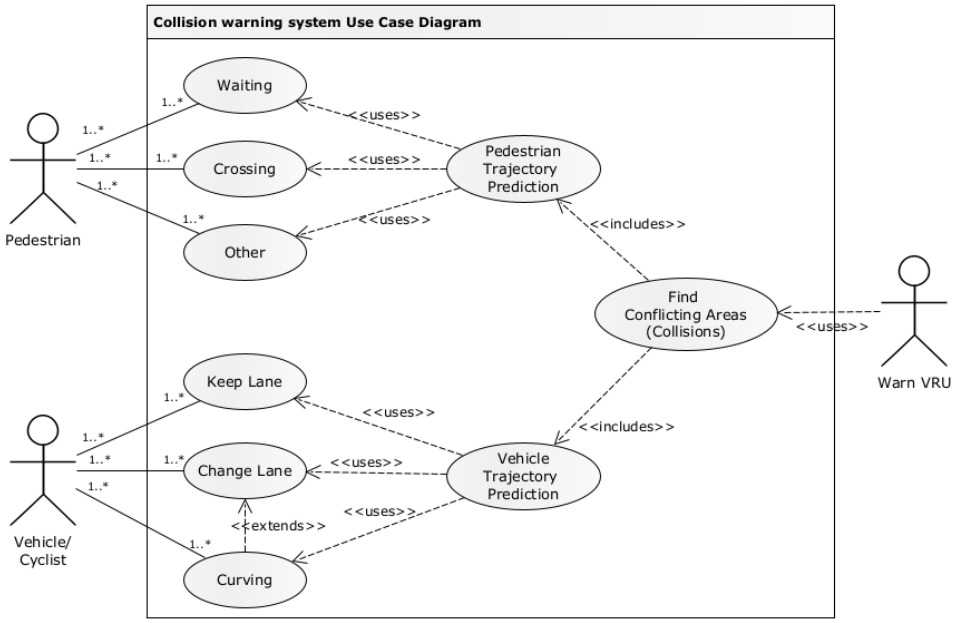


Figure 3.8: Use Case Diagram that shows the system behavior from a user perspective

3.3 Introduction to CarMaker

The company IPG Automotive is widely known in the vehicle dynamics simulation field, providing systems for the whole virtual vehicle development. Besides TruckMaker and MotorcycleMaker, which are adopted to test respectively heavy-duty vehicles and motorized two-wheelers, IPG Automotive provides the Simulator CarMaker, a simulation tool used for testing light-duty vehicles in virtual realistic environments during each phase of the in-the-loop development process.
CarMaker is a test platform which allows to recreate real-world driving scenarios in a virtual environment, simulating every type of road and traffic, performing realistic execution through the event and maneuvers-based testing method. The parametrization of the test scenario is stored in the *TestRun*, which collects the information about the overall environment:

- Vehicle: the definition of the ego-vehicle data set used.
- Road: the definition of the road, such as its geometry, its characteristics and the surrounding buildings, and the driving routes.
- Maneuver: the definition of the different maneuvers that the ego-vehicle performs during the simulation. This list of instructions do not set directly the vehicle motion, but the control actions that are meant to lead to certain vehicle motion: in other words, the vehicle follows the desired vehicle motion, if not beyond the vehicle limits.
- Driver: the definition of the driver behavior (defensive, normal, aggressive).
- Traffic: further static or moving traffic objects in the simulation (as pedestrians, cyclists, further vehicles, etc).
- Environment: the description of environmental conditions like the temperature, the time of day or the wind velocity, that influences the simulation results.

The ego-vehicle can be configured in every aspects, some of them are summarized in the following list: modules:

- Sensor Cluster: the parametrization of the sensor mounted on the ego-vehicle (radar, camera, etc.).

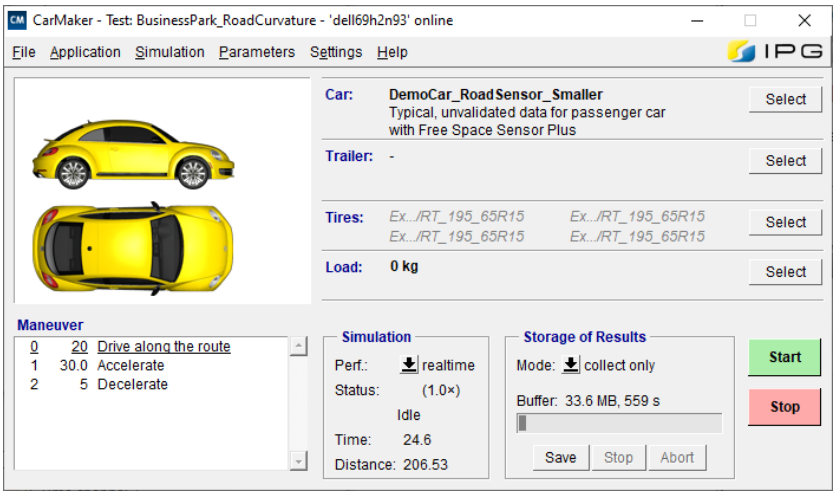


Figure 3.9: IPG CarMaker GUI

- Tire: the configuration of the tire data set.
- Trailer: the definition of a potential trailer attached to the ego-vehicle.

A preview of the CarMaker GUI is showed in 3.9. The upper section is completely dedicated to the vehicle (Car, Trailer, Tires and Load), while the maneuver section, on the left-down corner, displays the set of instructions for the test driver. The simulation box allows to completely control the simulation status, and to start and stop the simulation: if the simulation does not run in a HIL environment, another performance than real-time (run slower or faster) are possible. The Status entry display the status of the simulation:

- Starting Application: during the connection of the GUI to the application, when the simulation is started for the first time.
- Idle: default state when the simulation is not running.
- Preparation: vehicle parameters are set.
- Running: the simulation is running.
- Pause: the simulation is paused, to be resumed at a later point.

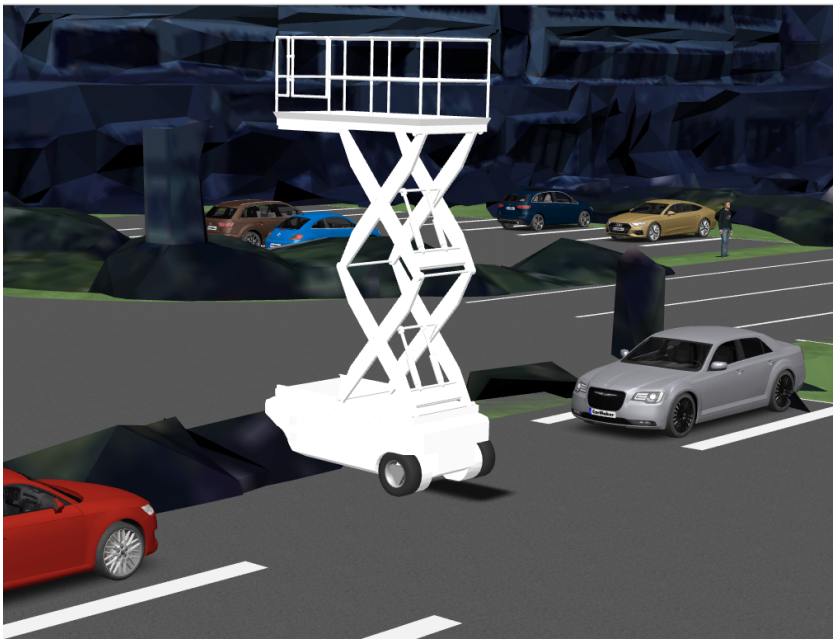


Figure 3.10: ScissorLift object used as test vehicle (1.5m x 1m x 2.5m)

Finally, the Storage of Results area allows the user whether to save (save all option is selected) or not (collect only) the results to the disk. This last option saves the results temporarily in a ring buffer, without being continuously and automatically saved to the disk. Which output quantities shall be saved, can be configured as well.

3.3.1 Actors: Test vehicle and traffic objects

One or more actors (at least the ego vehicle) populate each test scenario.

Ego-Vehicle

Two different types of Ego-Vehicle are used in these tests. The first one is, as showed in fig. 3.9 inside the Car section, the DemoCar, a Volkswagen Beetle adopted by IPG for EuroNCAP tests, with a RoadSensor. The second vehicle is a ScissorLift 3.10, a static object, whose car dynamic is inherited from the DemoCar, but the vehicle 3D model and the wheel frames have been adapted in the Vehicle Data Set (see fig. 3.11 for the wheel frames configuration).

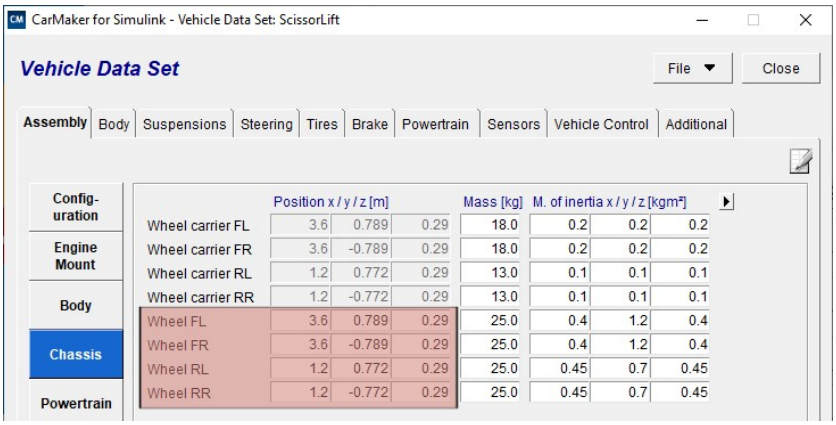


Figure 3.11: CarMaker Vehicle Data Set window, where the Wheel position can be configured

While the former is only used for the road map acquisition and creation, the second one appears in all of the scenarios used for the collision warning system tests.

Traffic

Additional road users can be added into the driving scenario. A traffic object can be whether movable, a dynamic part of the test run, such as parking or driving vehicles, bicycle riders, pedestrians, animals, or a stationary object (buildings, gas stations, etc), that are mainly used to create an ambience, which lead to a more realistic visual representation. Each object name, dimensions and orientation can be customized, as well as the start route and the start position offset. CarMaker allows the use of a three-dimensional model, to enhance the visualization of the traffic object, similar to the ego-vehicle (fig. 3.12).

The traffic object can be configured to use different motion model provided by CarMaker. The single track model represents the lateral and yaw motions of a traffic object: it can be 4Wheels in case of vehicles (see fig. 3.13) or 2Wheels for bicycles. The difference between the two resides in the roll and pitch model. While the pedestrian motion model is specific for pedestrian and animal objects, a further motion model, called ball, simulates the motion of a ball bouncing and rolling on the road.

Analogous to the test vehicle, driving maneuver independent to the ego-

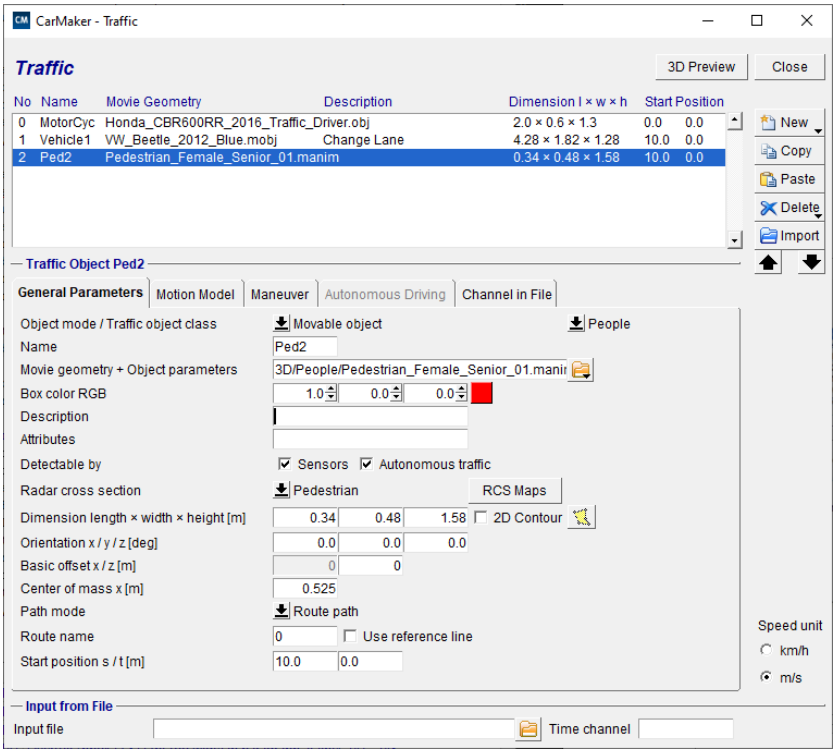


Figure 3.12: CarMaker Traffic GUI, with the list and general configuration of traffic objects

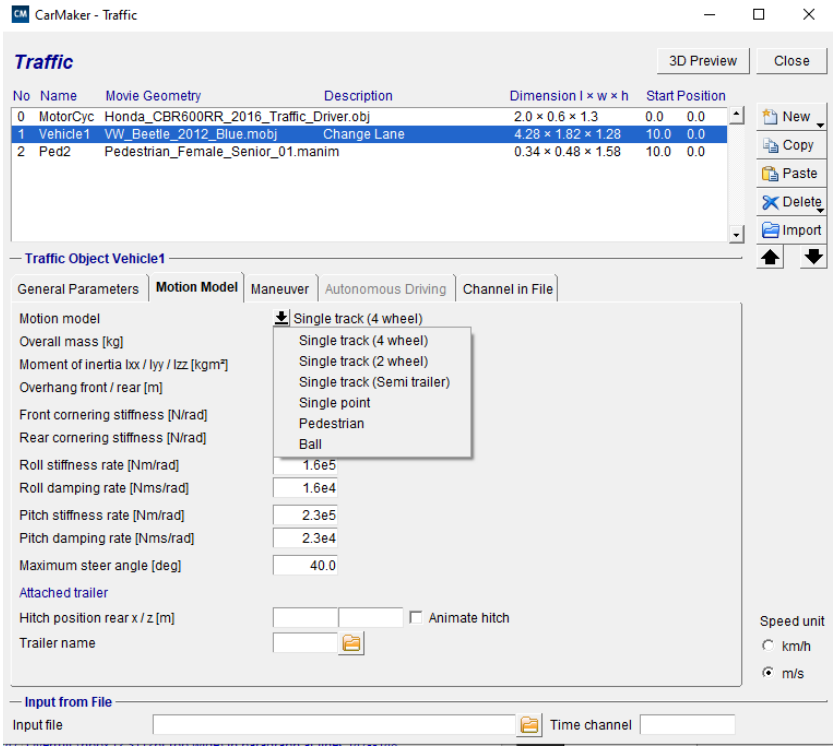


Figure 3.13: CarMaker Traffic Motion Model section. For vehicles the 4 Wheels Single Track motion model is automatically set.

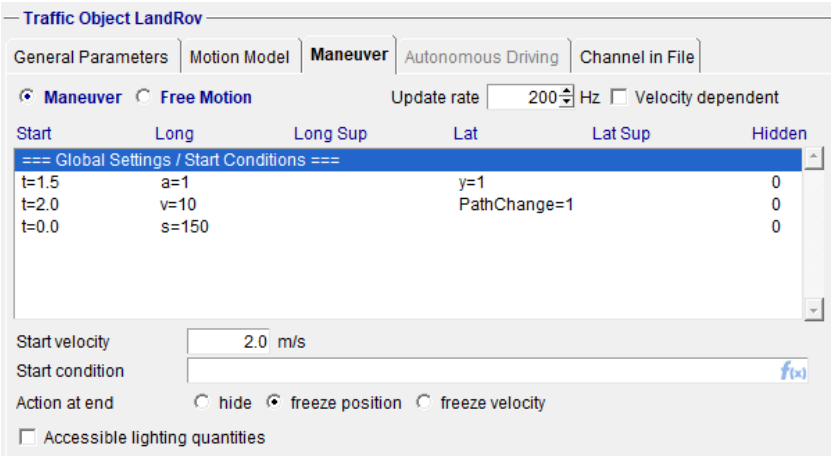


Figure 3.14: CarMaker Traffic Maneveur, where the minimaneuver of each traffic objects can be defined

vehicle can be defined for each traffic object, not only for single track motion model objects, but also for pedestrians: several options that define the longitudinal and lateral movements (such as change lane) are available (fig. 3.14).

3.3.2 Scenario Road and Routes

The Scenario Editor is the GUI that not only enables the creation of road networks for vehicle and driving simulation but also the definition of routes for the test vehicle and the traffic objects.

The road networks is defined by links and by junctions, that joins multiple links with each other. Each link is then divided into lanes, a maximum of 10 lanes can be defined: parameters as their section, type (driving lane, bicycle lane, bus lane, parking area etc.) can be configured as desired (fig. 3.15).
Furthermore, some editor options facilitates the use of road accessories, such as road markings, traffic signs and traffic lights, traffic barriers, and the beautification of the simulation environment. Elements belonging to this latter feature are bridges, tunnels and geometry objects.
A path is defined as a trajectory section on a link that is used for creating routes. Hence a route is a set of consecutive paths. Various routes can be defined in order to create many different driving possibility on the road.

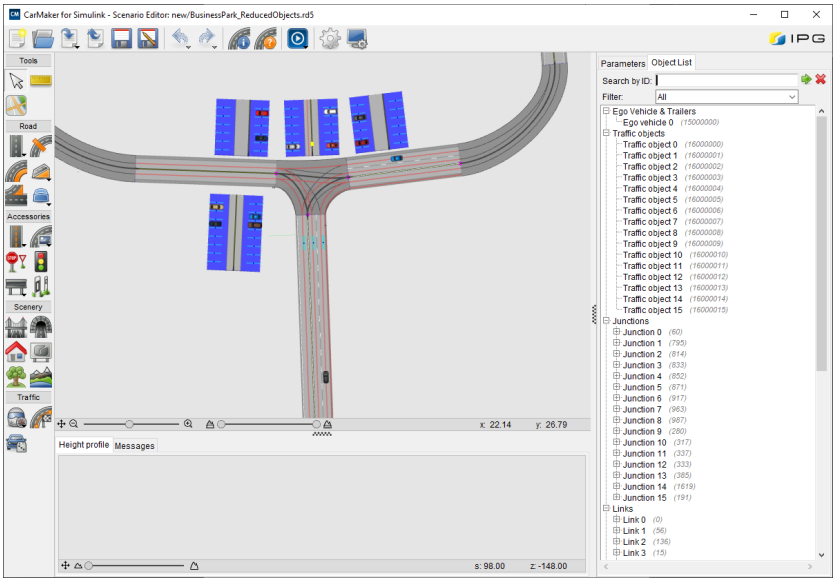


Figure 3.15: CarMaker Scenario Editor window

3D-Map

In order to recreate a realistic road scenario a 3D map has been attached to the road as a geometry object. The capture of the 3D-Map from GoogleMaps has been performed using RenderDoc, a free MIT licensed stand-alone graphical debugging tool for single-frame captures. A second software, Blender, a free and open source creation suite, allowed to improve the acquired 3D Map and to remove already parked cars (3.16).
After importing the map into the scenario geometry object (by means of Blender a .dae Collada file format could be generated), parked cars have been added again in the road scenario using traffic objects: this choice has improved the visualization, keeping it as realistic as possible. The final result of the driving scenario, after adding the roads, road markings, traffic objects, is showed in fig. 3.17.

3.3.3 CarMaker Frames

The CarMaker inertial axis system 3.18, known as global system, is a earth fixed origin with the following properties:

- O is the origin, while X, Y, Z are the 3 axis.

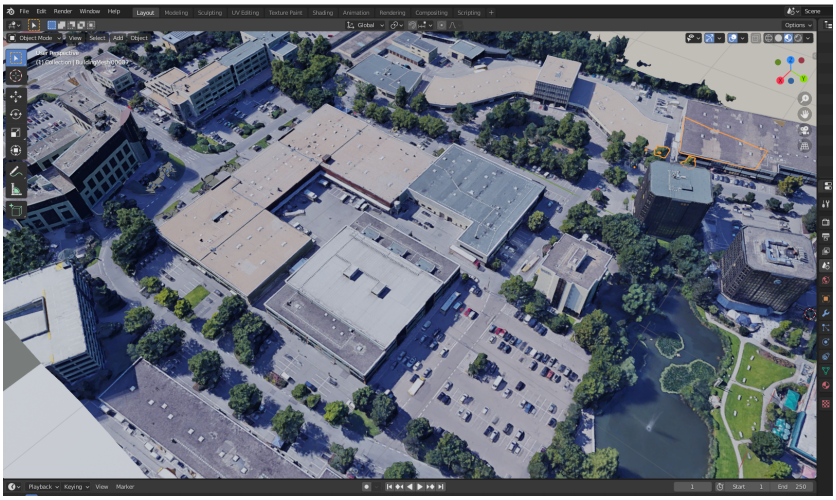


Figure 3.16: Blender, import of 3D-Map, after removing some parked cars and some view-obstructing trees

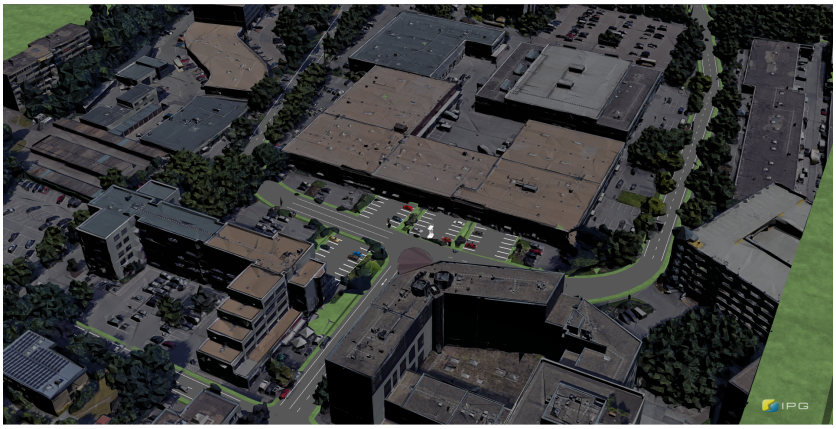


Figure 3.17: Driving scenario in CarMaker, with roads, road markings and traffic objects

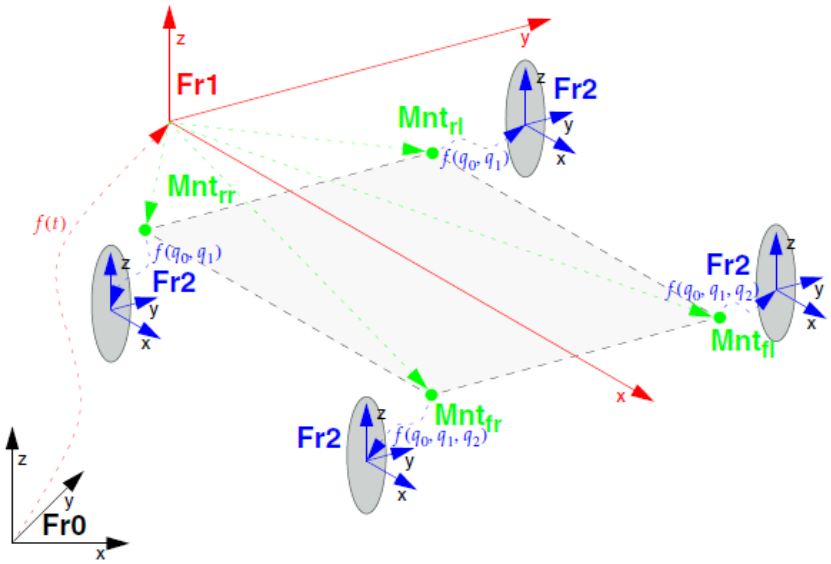


Figure 3.18: Visual description provided by CarMaker of the global and local system, with the wheel frames

- O, X, Y is the horizontal driving plane.
- Z is directed upwards, as product of XxY.

Another important axis system is given by the so called local system: a frame attached to each moving object, where X points to the moving direction, Y to the left and Z upwards. Many points, like car loads or sensor mountings, are parametrized in referece to this frame. The wheel position, as well, as shown in fig. 3.11, has been defined referring to the local frame. Within the local frame a mount-point is defined, that corresponds to the origin O, center of each wheel. This axis system, attached to the mount-point has its X axis pointing in forward driving direction, while Y is the wheel spin axis, therefore (O, X, Z) is the wheel plane.

3.3.4 CarMaker for Simulink

CarMaker for Simulink is a complete integration of CarMaker into the Matlab and Simulink modeling and simulation environment. CarMaker features and functionalities are integrated into the Simulink environment using S-Functions and API functions provided by Matlab/Simulink. A custom Car-

Figure 3.19: The CarMaker Simulink model structure

Maker blockset is provided, in order to directly connect a Simulink model with CarMaker. These CarMaker for Simulink blocks are direct feed through and run with the fixed step size of the CarMaker application (1000 Hz). The logic of the collision warning system is developed in Simulink thanks to the CarMaker extension. By opening the Simulink model, it is possible to see the main structure (3.19):

- By clicking on the "Edit Model Configuration", few additional settings for advanced application can be set (environment variables, or server application name).
- "CarMaker" block is the Simulink representation of CarMaker models, as shown in figure 3.20, with several subsystems of the CarMaker environment, where the exchanged signal are accessible;
- "Open GUI" block connects the Simulink model to a running CarMaker GUI; if no running GUI is detected a new CarMaker process is started and a TestRun shall be selected.

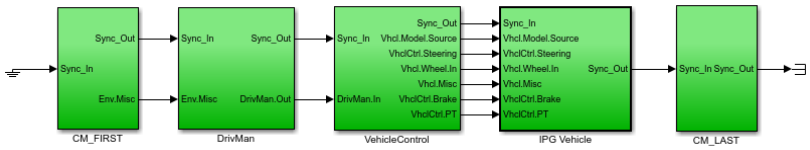


Figure 3.20: The CarMaker subsystem, representing the general structure of CarMaker in Simulink

Chapter 4

Methodology

This chapter focuses on explaining in details the solution adopted for the collision prediction. Firstly the methodology applied for the trajectory prediction of vehicles and bikes are presented: after going into detail for what the mode detection concerns (Section 4.1), the proper trajectory prediction based on pre-defined route is discussed (Section 4.2). Thereupon Section 4.3 presents the method adopted instead for pedestrians. Finally, when all the future trajectories are defined, the approach used for the collision detection is explained (Section 4.5).

4.1 2-4 Wheelers Lane Change detection

This chapter gives an overview about the approach used for the lane change intent detection. Firstly, section 4.1.1 provides a general overview of the motion models, that describe a vehicle dynamic behavior. Despite the fact that an Extended Kalman Filter (EKF) is applied to non-linear system 4.1.2, there is not a unique vehicle model (hence, not a unique filter) suitable for all the situation in which the vehicle can be involved. Therefore, the diversity of possible road maneuvers leads to the application of an Interacting Multiple Model (IMM) 4.1.3, that is able to run many models at the same time, selecting the one which better represent the system behavior. In this study, two EKFs have been chosen, one with a constant acceleration-like (CA) state update model and one implementing a constant turn rate and acceleration (CTRA) state update model. The former describes the kinematic model while keeping the lane, while the latter fits the change lane use case. Applying

the IMM filter on a set of measurement, the model probabilities computed by the IMM filter, allows to describe The Motion Mode Probabilities estimated by the IMM allows to detect a lane change maneuver, defining how probable is that a model is in effect at the current time step.

4.1.1 Motion Models

Motion models are mathematical frameworks, describing a vehicle dynamic behavior. They can be used to perform short-term prediction of a vehicle future state using the current state. Due to the great number of motion models used in the literature, it is not possible to give a precise overview, but a first systematization can be achieved by distinct levels of complexity. Roughly, at the lower end of such a scale, there are models that do not consider rotation (linear motion models): *constant velocity* (CV) and *constant acceleration* (CA) are two examples. A second level of complexity is defined by curvilinear motion models, as *Constant Turn Rate and Velocity* (CTRV) and *Constant Turn Rate and Acceleration* (CTRA). Schubert et al.[30] demonstrated that CTRV and CTRA provide better performances than CV: the use of the yaw rate reduce position errors. Furthermore, the incorporation of the acceleration additionally improves the overall tracking performance.

Constant Velocity Model The constant velocity (CV) model is a linear and rectilinear motion model, with consider a constant velocity. The state vector is given as $X = [x\, v_x\, y\, v_y]$. The state update equation is:

$$\overrightarrow{x}(t+dt) = \begin{bmatrix} x(t) + v_x dt \\ v_x \\ y(t) + v_y dt \\ v_y \end{bmatrix} \tag{4.1}$$

Constant Acceleration Model As the CV model, the Constant Acceleration Model (CA) is a linear and rectilinear motion model, that consider a constant acceleration. The state vector is given as $X = [x\, v_x\, a_x\, y\, v_y\, a_y]$ The state update equation is:

$$\overrightarrow{x}(t+dt) = \begin{bmatrix} x(t) + v_x dt + \frac{1}{2} a_x dt^2 \\ v_x + a_x dt \\ a_x \\ y(t) + v_y dt + \frac{1}{2} a_y dt^2 \\ v_y + a_y dt \\ a_y \end{bmatrix} \tag{4.2}$$

Constant Turn Rate and Velocity model The Constant Turn Rate and Velocity (CTRV) model is a non-linear and curvilinear motion model. The state vector is $X = [x\, y\, \theta\, v\, \omega]$ and the state update equation is:

$$\overrightarrow{x}(t+dt) = \begin{bmatrix} x(t) + \frac{v}{\omega} sin(\omega dt + \theta) - \frac{v}{\omega} sin(\omega) \\ y(t) - \frac{v}{\omega} cos(\omega dt + \theta) + \frac{v}{\omega} sin(\omega) \\ \omega dt + \theta \\ v \\ \omega \end{bmatrix} \tag{4.3}$$

Constant Turn Rate and Acceleration model The Constant Turn Rate and Acceleration (CTRA) model is a nonlinear and curvilinear motion model. The state vector is $X = [x\, y\, \theta\, v\, a\, \omega]$ and the state update equation is:

$$\overrightarrow{x}(t+dt) = \begin{bmatrix} x(t) + \Delta x \\ y(t) + \Delta y \\ \omega dt \\ a dt \\ 0 \\ 0 \end{bmatrix} \tag{4.4}$$

with

$$\Delta x = \frac{1}{\omega^2}[(v(t)\omega + a\omega dt) sin(\theta(t) + \omega dt) + a cos(\theta(t) + \omega dt) - v(t)\omega sin(\theta(t) - a cos(\theta(t))] \tag{4.5}$$

and

$$\Delta y = \frac{1}{\omega^2}[(-v(t)\omega - a\omega dt) cos(\theta(t) + \omega dt) + a cos(\theta(t) + \omega dt) + v(t)\omega cos(\theta(t) - a sin(\theta(t))] \tag{4.6}$$

4.1.2 Extended Kalman Filter

The Kalman Filter (KF) is a well-known recursive algorithm for estimation and prediction of dynamic systems. KF is used for linear transition functions whereas under non-linear transition, Extended Kalman Filter (EKF) is used. The main difference between the two filters lies in the linearization of non-linear functions performed by the EKF, employing a Jacobian matrix.

Kalman Filter

In 1960, R.E. Kalman published his famous paper describing a recursive solution to the discrete-data linear filtering problem. The Kalman Filter is a recursive algorithm, used to estimate states based on linear dynamical system in state space format. The main principle of Kalman Filter consists in finding the probability of the predicted state hypothesis, given by prior state hypothesis, and then using the data from measurement sensor to correct the hypothesis to get the best estimation for each time. Basically, applying KF has to concern about model creation including state and measurement model. In other words, it estimates the state $x \in \mathbf{R}^n$ of a discrete-time controlled process that is governed by the linear stochastic difference equations:

$$x_k = Ax_{k-1} + Bu_{k-1} + w_{k-1} \tag{4.7}$$

$$z_k = Hx_k + v_k \tag{4.8}$$

The first equation, linear stochastic equation, means that each x_k is evaluated. The second equation explains that any measured value, unsure of its accuracy, is a linear combination of the measurement noise and the signal value (both considered to be Gaussian). Both process and measured noise are assumed to be independent, white and with normal probability distribution.

$$p(w) \sim N(0, Q) \tag{4.9}$$

$$p(v) \sim N(0, R) \tag{4.10}$$

where Q and R are respectively the process noise covariance and the measurement noise covariance, that might change with each time step or measurement.

The Kalman filter estimates a process by using a form of feedback control: the filter estimates the process state at some time and then obtains

feedback in the form of noisy measurements. Hence, the Kalman Filter algorithm can be divided into two steps: prediction, described by time update equations(4.11), and correction, whose equations are known as measurement update equations(4.12).

$$\begin{aligned} \hat{x}_k^- &= A\hat{x}_{k-1} + Bu_{k-1} \\ \hat{P}_k^- &= A\hat{P}_{k-1}A^T + Q \end{aligned} \tag{4.11}$$

$$\begin{aligned} y_k &= z_k - H\hat{x}_k^- \\ K_k &= P_k^- H^T (HP_k^- H^T + T)^{-1} \\ \hat{x}_k &= \hat{x}_k^- + K_k y_k \\ P_k &= (I - K_k H)\hat{P}_k^- \end{aligned} \tag{4.12}$$

The time update equations are responsible for predicting forward (in time) the current state $\hat{x}_k^-$ and the error covariance estimates $\hat{P}_k^-$ to obtain the *a priori* estimates for the next time step. The state and covariance estimates are projected forward from time step $k-1$ to step k. The measurement update equations are responsible for the feedback: after computing the measurement residual y_k, also known as innovation, and the Kalman gain K_k, the *a posteriori* state covariance P_k and state estimate $\hat{x}_k$, are obtained, by incorporating the process measurement z_k. In details, the residual is the difference between the true measurement z_k and the estimated measurement $H\hat{x}_k^-$, where H is the measurement matrix. Hence, the update state estimate $\hat{x}_k$ can be calculated as the difference between the previous state estimate and the correction term $K_k y_k$. The final step is the computation of the updated state error covariance P_k, that together with the updated state estimate $\hat{x}_k$, will be used in the next iteration, or time step. The updated error covariance is smaller than the predicted error covariance, which means the filter is more certain of the state estimate after the measurement is utilized in the update stage.

The Extended Kalman Filter

When either the system state dynamics or the measurement model is nonlinear, the Extended Kalman Filter can deal with these systems, linearizing about the current mean and covariance, using Jacobian matrices. The state model and the measurement model are respectively described by (4.13) and (4.14).

$$x_k = f(x_{k-1}, u_{k-1} + w_{k-1}) \tag{4.13}$$

$$z_k = h(x_k + v_k) \tag{4.14}$$

where f is the non-linear transition function of the previous state x_{k-1} and the control input u_{k-1}, that provides the current state x_k, h is the measurement function that relates the current state x_k to the measurement z_k. w_k and v_k are Gaussian white noises for the process model and the measurement model with covariance Q and R.

By defining the Jacobian matrix J of partial derivatives of f with respect to x (4.15) and the Jacobian matrix H of partial derivatives of h with respect to x (4.16), the complete set of equation can be derived: as in the linear discrete Kalman Filter, the process of the EKF recursive algorithm is divided into the prediction stage (4.17) and the correction stage (4.18).

$$J_{k-1} = \left.\frac{\delta f}{\delta x}\right|_{\hat{x}_{k-1}, u_{k-1}} \tag{4.15}$$

$$H_k = \left.\frac{\delta h}{\delta x}\right|_{x_k^-} \tag{4.16}$$

After obtaining the predicted state estimate $\hat{x}_k^-$, by the nonlinear functions $f(x_{k-1}, u_{k-1})$, and the predicted state error covariance $\hat{P}_k^-$, the correction part can start. The Kalman gain K represents the trustable value of state model and measurement variable: when R approaches to 0, it means the measurement variable is more trustable than state model but if P_k^- approaches to 0, then it is viceversa. An important feature of the EKF is that the Jacobian H_k in the equation of K, serves to correctly propagate only the relevant component of the measurement information. After updating the state by the use of the Kalman gain K and the innovation y_k, and the state error covariance, all parameters are updated. Hence, in each iteration predicted and estimated data continue to become more accurate.

$$\begin{aligned} \hat{x}_k^- &= f(\hat{x}_{k-1}, u_{k-1} + 0) \\ \hat{P}_k^- &= J_k \hat{P}_{k-1} J_k^T + Q_{k-1} \end{aligned} \tag{4.17}$$

$$
\begin{aligned}
y_k &= z_k - h(\hat{x}_k^-) \\
K_k &= P_k^- H_k^T (R + H_k P_k^- H_k^T)^{-1} \\
\hat{x}_k &= \hat{x}_k^- + K_k y_k \\
P_k &= (I - K_k H_k) \hat{P}_k^-
\end{aligned}
\tag{4.18}
$$

Advantages and disadvantages of EKF

The advantage of the EKF over other non-linear filtering methods is its relative simplicity compared to its performance. Since it is computationally cheaper than other non linear filtering methods such as point-mass filters and particle filters, the Extended Kalman Filter has been used in various real-time application like navigation systems.
On the other hand, the Extended Kalman Filter is based on a local linear approximation of the state and measurement, in order to apply the Kalman Filter equations to the resulting linear estimation problem. Hence, for problems that contain considerable non-linearities, further filters as Particle Filter or Uscented Kalman Filter (UKF) should be used, with increased computational cost.
For the purpose of this study the EKF was choosen for two reasons: approximation errors ignored during the prediction/update state are negligible and the computational cost play an important role. The results demonstrate that this choice is reasonable.

4.1.3 IMM - Interacting Multiple Model

The Interacting Multiple Model (IMM) estimator is a sub-optimal hybrid filter that has been shown to be one of the most cost-effective hybrid state estimation schemes [22]. The model of hybrid system and the IMM algorithm, initially proposed by Blom [4], may serve as a basis for tracking maneuvers[31] and lane change prediction[32]. Hybrid systems are characterized by:

- *State*, that evolves according to a stochastic differential equation model;
- *Model*, that is governed by a stochastic process: it is one of a finite number of possible models (each corresponds to a behavior mode), that undergo switches from one model to another according to a set of transition probabilities.

In practice, the IMM can operate different Kalman Filters in parallel, and carefully blends state and covariance from each filter to make a composite state estimate and covariance.

Problem Formulation

In general, an hybrid system with additive noise can be described as follow:

$$x(k+1) = f[k, x(k), m(k+1)] + g[k, x(k), m(k+1), v[k, m(k+1)]] \tag{4.19a}$$

$$z(k) = h[k, x(k), m(k)] + w[k, m(k)]] \tag{4.19b}$$

where x is the base state, z is the mode-dependent noisy measurement and $m(k)$ s the modal state at time k. The transition probability of the system mode is

$$P\{m_j(k+q)|m_i(k)\} = \phi[k, x(k), m_i, m_j] \quad \forall m_i, m_j \epsilon M_s \tag{4.20}$$

where $m_j(k)$ is the event that mode j is in effect at time k, M_s is the set of all modal states at all times, v and w are the mode-dependent process and measurement noise sequences with means $\overline{v_j}$ and $\overline{w_j}$ and covariances Q_j and R_j, respectively.

For the purpose of this thesis, a fixed-structure hybrid system is analyzed. In such hybrid systems, a set of modes are selected in advance. Hence, the system 4.21 can be rewritten as

$$x(k+1) = f_j[k, x(k)] + g_j[k, x(k), v_j(k) \quad \forall j \epsilon M_f \tag{4.21a}$$

$$z(k) = h_j[k, x(k)] + w_j(k) \quad \forall j \epsilon M_f \tag{4.21b}$$

with M_f fixed set of N_f modes.

The problem of hybrid state estimation is to estimate the base state and the model state based on the measurement sequence.

Design Parameters

The IMM design parameters are as follows:

- The set of models for the various regimes and their structures (M_s);

- The process noise intensities (v)
- The jump structure and the transition probabilities matrix (TPM) between the different n models.

Algorithm

The IMM algorithm has three important properties: it is recursive, modular, and has fixed computational requirement It repeatedly executes three steps per iteration:

1. Interaction between filters (interaction);
2. Individual filter update (filtering);
3. Combine filter information (combination).

A more detailed overview about each step is given by following the notation and the explanation provided by Bar-Shalom et al.[22]

Interaction For this step, the IMM algorithm requires three set of input: the vector of model probabilities at current time step, the transition probability matrix and the M individual filters' estimates at time k.
First of all the mixing probability is computed (4.22)

$$\mu_{i|j}(k-1|k-1) = \frac{1}{\overline{c_j}} p_{ij}\mu_i(k-1), \forall i,j \in M_f \tag{4.22}$$

where $\overline{c_j}$ is a normalization factor

$$\overline{c_j} = \sum_{m_i \in M_f} p_{ij}\mu_i(k-1) \tag{4.23}$$

Now, the mixed initial state $x_{0j}(k-1|k-1)$ and covariance $P_{0j}(k-1|k-1)$ of the current time step can be derived by blending together the state estimates and the covariance from all the filters at the previous time step.

$$\hat{x}_{0j}(k-1|k-1) = \sum_i \hat{x}_i(k-1|k-1)\mu_{i|j}(k-1|k-1) \tag{4.24}$$

$$P_{0j}(k-1|k-1) = \sum_i \mu_{i|j}(k-1|k-1)\{P_i(k-1|k-1) + [\hat{x}_i(k-1|k-1) - \hat{x}_{0j}(k-1|k-1)] \times [\hat{x}_i(k-1|k-1) - \hat{x}_{0j}(k-1|k-1)]^T\} \quad (4.25)$$

Filtering The estimate (4.24) and covariance (4.25) are used as input to the Kalman filter matched to $m_j(k)$ to obtain the state estimate $\hat{x}_j(k|k)$ and covariance $P_j(k|k)$ at time k, as well as the mode likelihood $\Lambda_j(k)$

$$\Lambda_j(k) = |S_j(k)|^{-\frac{1}{2}} exp\{-\frac{1}{2} y_j^T(k) S_j^{-1}(k) y_j(k)\} \quad (4.26)$$

where $y_j(k)$ and $S_j(k)$ are innovation and its covariance (4.27) of the jth conditional filter.

$$S_j(k) = H_j(k) P_j(k|k-1) H_j(k)^T + R_j(k) \quad (4.27)$$

The mode probabilities are then updated as well

$$\mu_k(k) = \frac{1}{c}\Lambda_j(k) \sum_i p_{ij}\mu_i(k-1) = \frac{1}{c}\Lambda_j(k)\bar{c}_j \quad (4.28)$$

where $c = \sum_{j=1}^{s} \Lambda_j(k)\bar{c}_j$ is the normalization factor, to ensure that $\mu_j k$ sums to one.

Combination The combination of the updated mode conditioned estimates and covariances produces the output estimates:

$$\begin{aligned} \hat{x}(k|k) &= \sum_j \hat{x}_j(k|k)\mu_j(k) \\ P(k|k) &= \sum \{P_j(k|k) + [\hat{x}_j(k|k) - \hat{x}(k|k)][\hat{x}_j(k|k) - \hat{x}(k|k)]^T\}\mu_j(k) \end{aligned} \quad (4.29)$$

4.2 2-4 Wheelers Map-Based Trajectory Prediction

4.2.1 Predefined Routes

The generation of the trajectory of a vehicle or a bike is strictly linked to the geometry of the road map. As already mentioned in Section 3.1, predefined

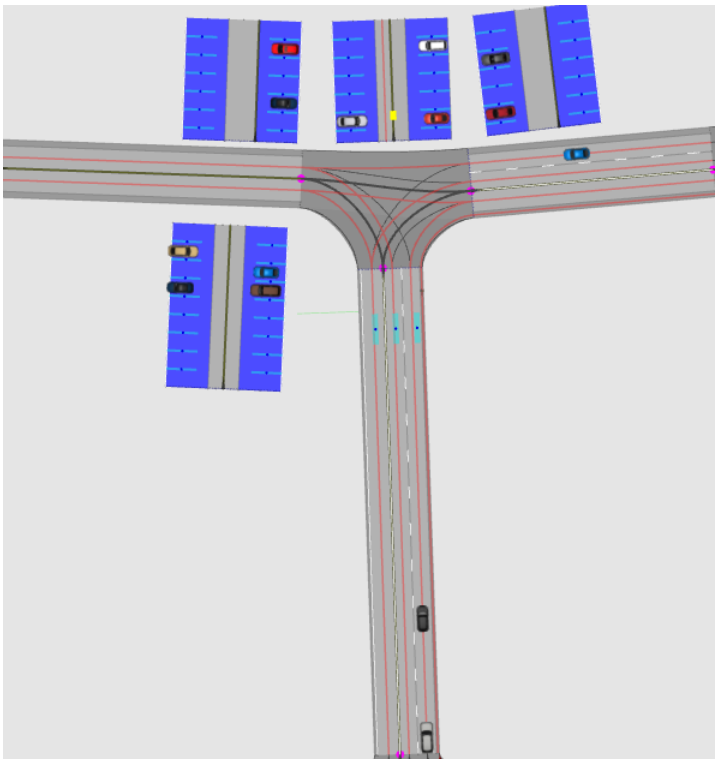


Figure 4.1: Car Maker road scenario with all the different possible routes (in red)

routes, that vehicles are driving along, shall be defined. The general approach adopted to generate them is now presented.

Collect the route data

CarMaker, as discussed in 3.3.2, allows to add different routes in the road scenario, for the ego-vehicle and the different traffic objects (autos, bicycles, motorcycles, etc.).

Starting from the *generic-model.slx* made available by CarMaker, a simple Simulink model has been developed (fig. 4.2), to simulate a vehicle (velocity and vehicle specifications have no relevance in this case) driving along each route, from the start until the end, without further maneuvers. Crucial importance has the *RoadSensor* mounted on the ego-vehicle: it collects information about the road, like the road bend and deviation, road marker

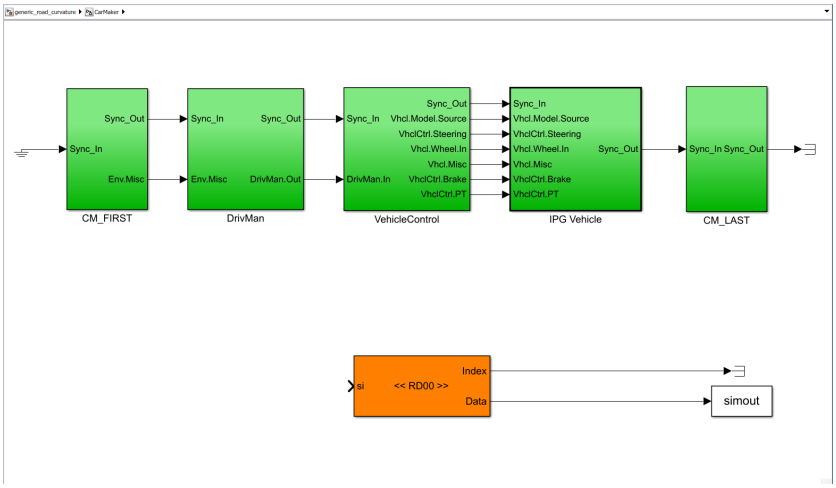


Figure 4.2: Simulink first level subsystem, where data from the RoadSensor are written to a timeseries in the Matlab Workspace, to be then stored in a .mat file

attributes or longitudinal and lateral slope, and about the relative position. The *RoadSensor* block (available in the CM4SL library) requests information from the mounted road sensor and outputs them to a Simulink.Bus element, in order to create the map graph offline. Using the ToWorkspace block, this bus will result as a timeseries in the Matlab workspace after the simulation, ready to be stored in a .mat file.

After collecting each route data, in other words, after having run all the simulations, the actual generation of the graph, can be simply triggered by a standalone Matlab script, whose single input is the set of .mat files.

Define a Route Segment

In the first instance, each route is splitted into shorter segments (linear or curvilinear), basing on the road curvature: as soon as the road switches from a linear path (curvature = 0) to a positive (or negative) not null value, or viceversa, a new segment starts. In the example in fig. 4.3, the route consists of two linear segments (the first one in rosa and the last one in blue), where the curvature is 0, and of a curvilinear segment (the second one in yellow). Both type of segments are described by the following entries:

- *index*: segment number, used as unique id to identify it within the

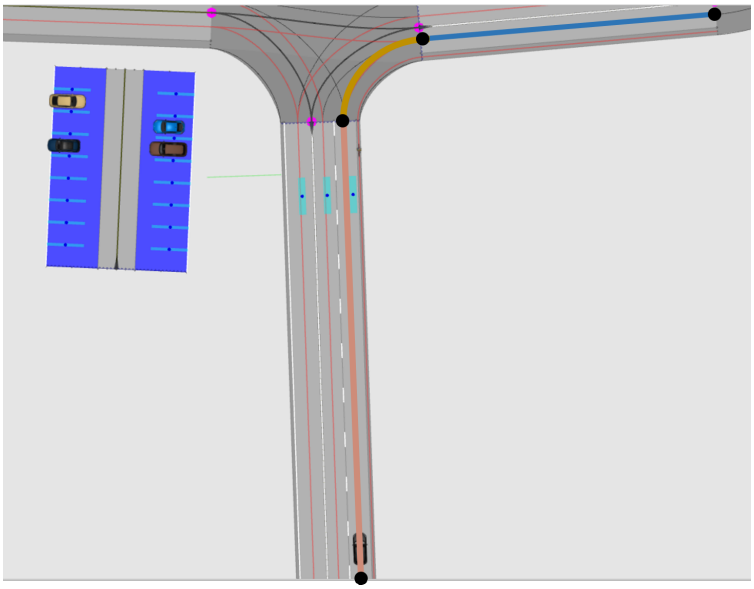


Figure 4.3: Example of road splitted into shorter segments, basing on the road curvature

entire map definition.

- *boundaries*: the (x, y) coordinates of the starting and end point (black filled scattered in the previous example).
- *curvature*: the curvature of the path at xy plane. 0 for linear segments, and $abs(curvature) > 0$ for curvilinear segments.
- *polynomial*: each segment is described by a polynomial. Whereas for linear segments a first order polynomial, such $y = mx + q$, fits perfectly the segment, for a curvilinear segments a 5th degree polynomial is necessary.
- *direction*: two-elements vector, where the two values (in radians) define the vehicle heading angle at the beginning and the end of the segment. Please note, in case of linear segments, the two angles are identical. Curvilinear segment direction can be affected by inaccuracies originated by the not 100% fitting polynomial. These values are calculated deriving the angle of the tangent of the curve at the aforementioned points.

- *next*: two-elements vector containing the indexes of the segments, by which the segment is followed. This attribute is initialized with a vector $[-1; -1]$: when a sequent adjacent is found, the first -1 is replace with the respective index. Please note, that the size of this vector remains fixed (no more than two items can follow this path), in order to avoid Bus definition issues during simulation: this assumption is reasonable in this case of study, as each segment has maximum 2 following paths. This value can be though always be changed, with a redefinition of the size of the `next` attribute of the `MapDefinition_Segment`.

- *changeLane*: segment index (if exists) of the possible target lane in case of a lane change maneuver starting from the segment in object. The value is initialized with -1. The size if fixed and the consideration made for the attribute above apply here as well. A change lane segment is defined if and only if the distance between the two route is less than the maximum of a lane width and the direction are identical, to avoid wrong way paths.

- *isForVehicle*: boolean value that distinguish driving lanes from bicycle lanes. It plays an important role while defining the entry above: a vehicle target lane is supposed not be a bicycle lane.

- *definition*: 101x3 vector used during the trajectory prediction, to move a traffic object along the segment (see section 4.2.3). The curve (whether linear or curvilinear) is divided into N units, whose length can be approximated by Eq. 4.30. Hence, each row of the vector contains the x and y coordinates, as well as the length of the section defined by the start of the curve and the (x, y) coordinates.

$$\begin{aligned} dx &= x_i - x_{i-1} \\ dy &= y_i - y_{i-1} \\ arcLength &= arcLength_{i-1} + \sqrt{dx^2 + dy^2} \end{aligned} \tag{4.30}$$

Generated Map

All the segments are then contained in a DB, in form of a .mat file. In the fig. 4.4, the generate map used in this study is illustrated. Boundaries of the each segment are scattered. Cyclists routes are plotted with dashed lines, indexes are in italic.

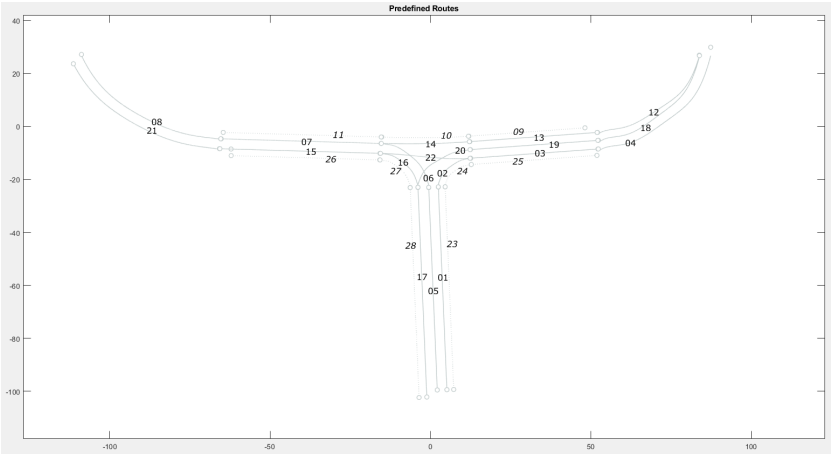


Figure 4.4: Plot of the generated Map: vehicles routes are defined by continuous lines, bicycle routes with dashed lines. Segment boundaries are scattered.

4.2.2 Lane Change with Bézier Curve

Bézier Curves were invented in 1962 by the French engineer Pierre Bézier for designing automobile bodies. A Bézier Curve of degree n can be represented as

$$P_{[t_0,t_1]}(t) = \sum_{i=0}^{n} B_i^n(t) P_i \tag{4.31}$$

where P_i are the control points such that $P_(t_0) = P_0$ and $P(t_1) = P_n$, $B_i^n(t)$ is a Bernstein polynomial given by

$$B_i^n(t) = \binom{n}{i} (\frac{t_1 - t}{t_1 - t_0})^{n-1} (\frac{t - t_0}{t_1 - t_0})^i, i \in \{0, 1, ..., n\} \tag{4.32}$$

Bézier curves have three main distinct properties:

- They always passes through P_0 and P_n
- They are always tangent to the lines connecting P_0 to P_1 and P_n to P_{n-1} at P_0 and P_n respectively.
- They always lie within the convex hull consisting of their control points $(P(t) \in convexhull P_0, P_1, ...P_n)$.

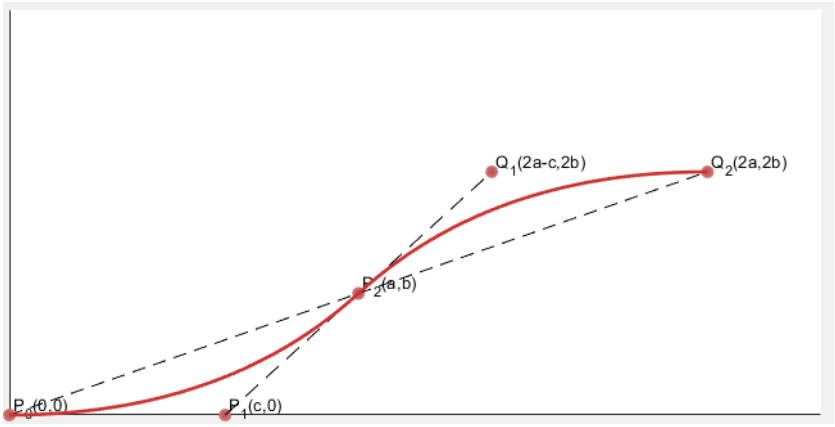


Figure 4.5: Geometric description of Bézier control points

Compute the Bézier Curve

The approach presented in [7] uses piecewise quadratic Bézier curves in order to avoid problem of heavier calculation required by higher degree Bézier Curves, whose fitting precision is sometimes still not enough. In this study, the results presented using the quadratic Bézier curves are plausible.
The lane change path has been generated by two quadratic Bézier curves, P and Q, whose control points are, as described in 4.5

- $P_0(0,0); P_1(c,0); P_2(a,b);$
- $P_2(a,b); Q_1(2a-c,2b); Q_2(2a,2b);$

where $c \in (0,a)$, in particular $c = 0.618a$, which means that c is the golden section point of P_0P_2', where P_2' is the projection of P_2 on the x-axis. Parameter b is half the width of the lane. Finally a depends on the speed of the vehicle $a = 1.393 * v$, where v is expressed in m/s.

Rotate the Bézier Curve

After defining the maneuver, the two quadratic Bézier Curves shall be integrated into the road map, by means of a rotation about the origin P_0 (center or rotation) of an angle α, that is the angle in the xy plane counterclockwise from the positive x-axis, formed by the road segment and the y-axis. The rotation matrix is shown in 4.33:

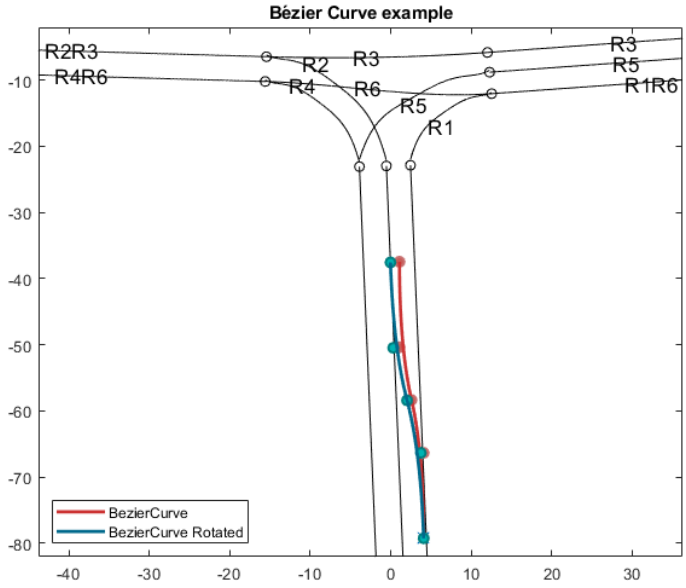


Figure 4.6: Before (red) and after (blue) curve rotation.

$$\begin{bmatrix} cos(\alpha) & -sin(\alpha) \\ sin(\alpha) & cos(\alpha) \end{bmatrix} \tag{4.33}$$

4.2.3 Drive along a Route

In order to let a traffic object "move along" a segment or a Bézier curve, the curve definition (as described in 4.30) plays an important role. The distance s that a vehicle can travel, given the velocity and the acceleration in a known time window, corresponds to the arc length between the starting position and the position that the vehicle will reach.

Given v and a, the traveled distance s can be easily defined using the equation of motion:

$$\begin{aligned} s &= s_0 + v_0 t + \frac{1}{2} a t^2 \\ v &= v_0 + at \end{aligned} \tag{4.34}$$

If s exceeds the curve arc length, the maneuver is finalized before dt, that means in the remaining time $\Delta t = t - t_{maneuver}$ the vehicle will drive along the next segment.

4.3 Pedestrian Trajectory Prediction

For pedestrian, a CA motion model is applied, resulting in an extreme easier workflow than the one adopted for 2-4Wheelers. Even in this case there is more than one status possible: `TrafficObjectStatusEnum.WAITING` or `TrafficObjectStatusEnum.MOVING`. While the former status denotes a pedestrian standing and (maybe) waiting to cross the nearest road, the latter may describe a VRU moving alongside the route or already crossing the road. There is no real difference between these two type of moves within the trajectory prediction: if a pedestrian is already moving, he is supposed to follow its path, whether perpendicular or parallel to the road.
The key difference between `WAITING` or `MOVING` lies in the velocity and heading angle fed into the CA motion model. When a pedestrian is moving, the velocity and the heading is already given by his current path. In case of a standing pedestrian, the velocity tends to 0, therefore a mean velocity is required for the trajectory initialization. The heading angle could not match the one used in the event the person starts crossing the road: he could just being look around, controlling if vehicles are approaching. Hence, the initial heading angle is perpendicular to the closest road segment.

4.4 Trajectory Generator

Having outlined the general approach for VRU and not VRU trajectory prediction, the path generation process, that applies to both the categories, is illustrated here below.

The until now unanswered question is "*Which is the time horizon? And how is a trajectory computed?*". The prediction of the future position is computed for each instant in the time window, from 0.5s until 4s, with a time interval δt of 0.1s $(0.5, 0.6, 0.7, ..., 4)$. This configurations have been assigned on the basis of experiments, resulting an optimal compromise between computational cost and precision.
The prediction calculated on the basis of the current velocity and accelera-

tion can result in an intrinsic inaccuracy, as highly unlikely that a vehicle, for example, keeps its velocity in a crossroad over the considered time window. Hence, a variation in the speed ($\pm\Delta V$), or in other words an acceleration/deceleration is integrated. However, it is although improbable, that the velocity in the next 0.5s is affected by an error as high as after 4s. Therefore, it deems it appropriate to variate the ΔV progressively, increasing it the further in the future the prediction is computed.
The trajectory is defined by a set of positions, two for each time step: one derived using $-\Delta V$, that defines the nearest reachable position, and the other by $-\Delta V$ (the most distant position the object can get to). These two points are the boundaries, extremes, of the polynomial that covers the possible occupied area by the center of the mass of the object (at this step, the real dimension are still not considered) on the route.

4.5 Collision Prediction

Finally all the traffic object trajectories are processed to detect eventual collision. As defined in Section 3.1, an accident happens when the areas occupied by a pair of objects, at the same time, have at least one point in common. A first (faster) approach relies on intersecting the center of mass trajectories, expressed in form of a polynomials. For this purpose the problem of finding roots of $P_z(x) = P_1(x) - P_2(x) = 0$ is solved: if at least a real solution within the boundaries defined in the previous step exists, then a collision will take place.
On the contrary, moving from a 1D to a 2D space, by integrating the width and height of the traffic object is from here on strictly required: the object is no more represented as a point, but as a rectangle. This means, the polynomial must be shifted to the left and to the right (or to the bottom and to the top) to take the vehicle width into account, delineating an area on the map.
The trajectory is divided in N sample segments, defined by N+1 equidistant points, as shown in fig. 4.7. Around these points a rectangle is created (grey dashed lines), by means of the real dimensions, and then rotated to match the heading angle of the route (red dashed line). The tangent to the segment (in grey) in this point splits the object in two sections: the points on the left are listed in the set of coordinates the first polynomial (yellow continuous line) shall fit. The first and the last element in the list represent the boundaries

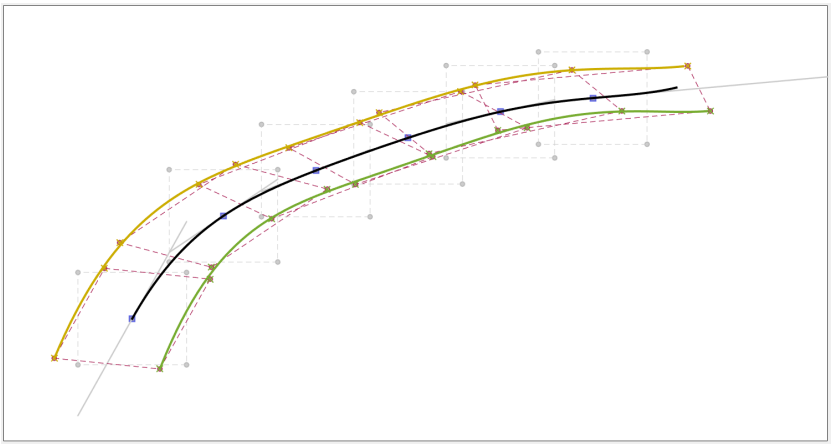


Figure 4.7: Example of expanding a center of mass trajectory to a 2D trajectory, of an object with a specific width and length. The initial trajectory in black is divided into smaller segment defined by the points in blue. After the rectangle is created (dashed grey) and rotated (dashed red), the angles are divided into two sets, the one on the left that defines the first polynomial (yellow) and the one on the right for the second polynomial (green).

of the polynomial. Similarly the polynomial on the right (green continuous line), as well as its boundaries are derived.

The definition of the polynomial allows to define a curvilinear area that the traffic object occupies, at a specified time step. The next step is to check whether a conflict area exists. The region is approximated by breaking the area down into rectangles or trapezoids (fig. 4.8).

If at least one of the extremes of the second area is located inside or on the edge of one polygon, a conflict area is found and, thus, a collision predicted.

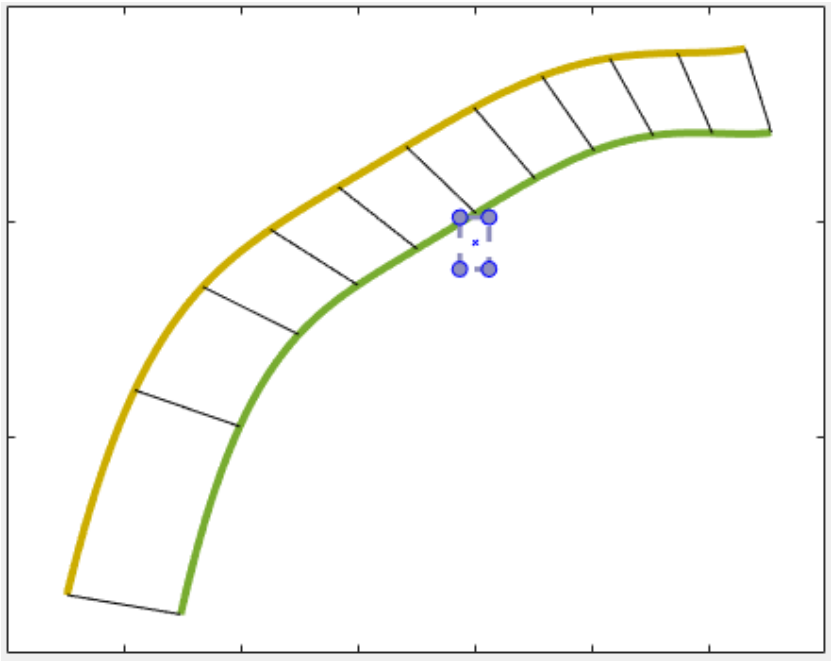


Figure 4.8: The area between the two polynomials is divided into trapezoids. One of the extremes (in blue) of the second ares is inside the first region: a collision warning shall be triggered.

Chapter 5

Simulation

A general overview of the Simulink Model is firstly presented (Section 5.1), describing the implementation of the main blocks such as the VehiclePrediction, the PedestrianPrediction and the Collision blocks. A GUI (Section 5.2) has been developed as well in order to ease the use and the results display.

5.1 Simulink Model overview

As presented in 3.3.4, a `generic.mdl`, that implements the plain CarMaker vehicle model, is present in each CM4SL new project. Instead of creating a complete new model from scratch, the `generic.mdl` model has been extended by the trajectory prediction and collision prediction functionality (fig. 5.1).

The `initialize` system, that is triggered by the model initialize event, set the different state writers (`RouteStruct`, `BezierCurve` and `Trajectory Prediction`), used as initial condition for the three UnitDelays (respectively `UD_RouteStruct`, `UD_BezierCurve` and `UD_TrajectoryPredictions`), in the trajectory prediction step. These UnitDelays are responsible of sampling and holding for one sample period delay the values calculated for the `RouteList`, `BezierCurve` and `TrajectoryPredictions` at each time step (a detailed overview is provided in 5.1.2.
The main logic of the collision prediction is contained in the `CollisionWarning` block (fig. 5.2). The number of TrafficObject present in the Scenario, as well as the test vehicle position are calculated and fed into the `TrajectoryPrediction` block. The Scissor List position is red using the Inertial Sensor mounted on the vehicle. The Inertial Sensor block 5.3 allows to read all the data recorder

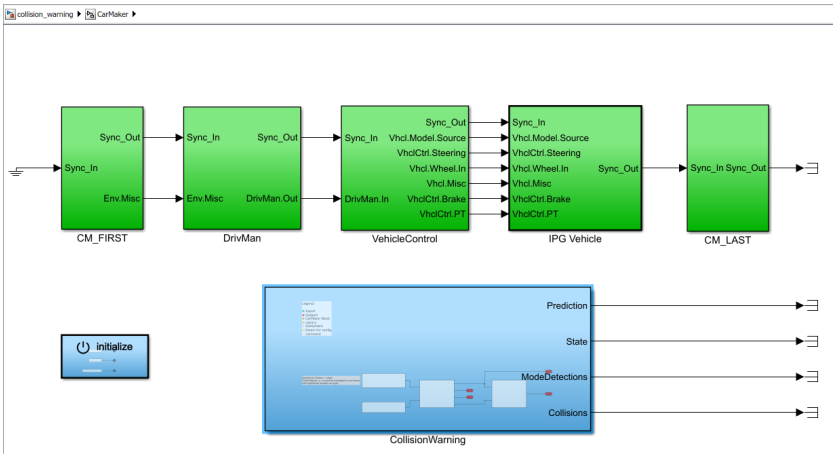


Figure 5.1: Simulink root model with the initialization and the Collision Warning subsystem

by this sensor during the simulation, like position, transitional and rotational velocity, transitional and rotational acceleration in the global frame (see Section 3.3.3). In this case, the position (x, y) of the object in the global frame is stored in the Bus Creator, that will be propagated throughout the model, to shift the traffic object coordinates to the frame having its center in the local frame of the Scissor Lift, FrO.

The number of traffic objects in the scene is read by a simple CarMaker block, `Read CM Parameter` (fig. 5.4, that let Simulink access the numerical paramter contained in the CarMaker infofiles, plain ASCII file where all the TestRun parameter are stored in form of key-value pair.

5.1.1 Setup process

The `setup()` function, configured as *PreLoadFcn* Callback in the root of the Simulink model, is responsible of the environment setup. In particular:

- Initialization of CarMaker: the script `cmenv` script, provided by CarMaker, extends the Matlab search path to search for the CarMaker for Simulink's installation directory. This script must be always kept in the same directory as the Simulink model;

- Load the CarMaker selected scenario, by `cmguicmd`. During this step, an error can occur (*connection failed*), if the CarMaker GUI is not yet

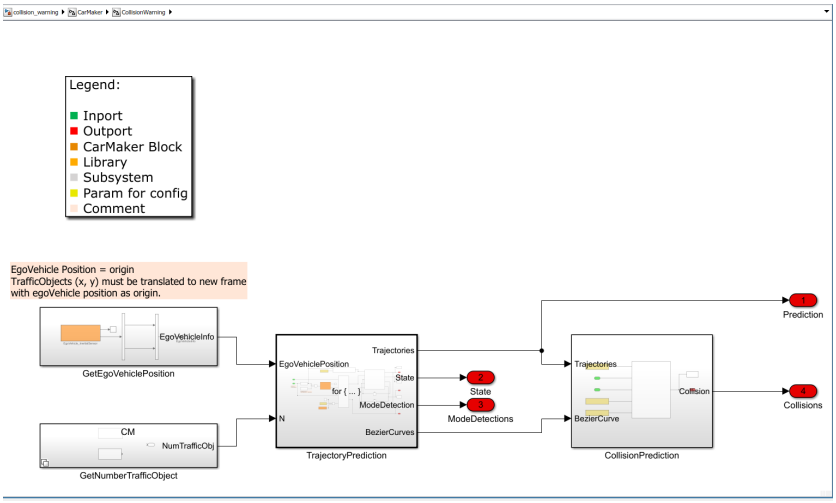


Figure 5.2: Collision warning system with the two main blocks *TrajectoryPrediction* and *CollisionPrediction*

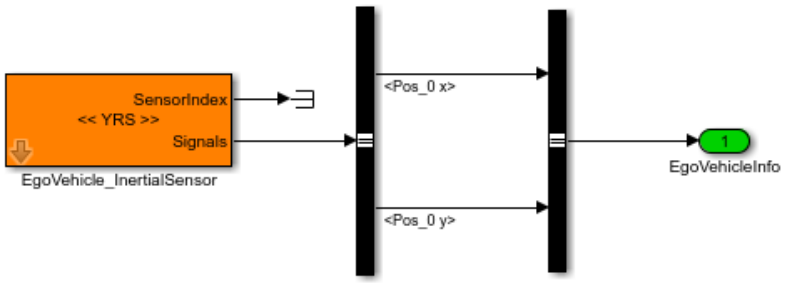


Figure 5.3: The inertial sensor block responsible for reading the ego vehicle position

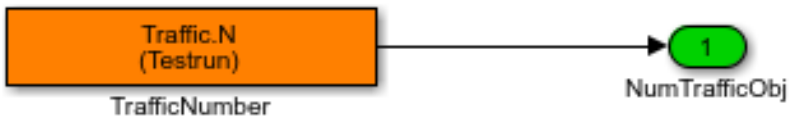


Figure 5.4: The Read CM infofile parameter block, used to access the entry *Traffic.N* in the TestRun file

open: the only possible solution is to open manually the CM GUI and run again the setup function.

- Parse of TestRun to store some actor configurations, otherwise not accessible at runtime: name, object type (Vehicle, People, Cyclist, ...) and dimension (length and width).
- Load of mess struct: this variable contain all the filter's parameterization, such as the measurement noise and process noise.
- Add project paths to Matlab path: not only the CarMaker path shall be added to the search path, but also the project specific paths, where the packages are to be found.
- Load bus elements: the file *busObjects.mat* contains all the bus definition, in order not to have to redefine them by each initialization. The list of the bus elements loaded into the workspace corresponds to the class defined in fig. 3.5.
- Load the road map into the Workspace: the file *mapDB.mat*, load the map definition into the workspace. In case this file is deleted, for any reason, or if is extended with new route, it can be recreated calling the `roadmap.createMapDB()` function.

5.1.2 Trajectory Prediction

The need of getting the amount of objects involved in the simulation is soon explained, by having a look inside the `TrajectoryPrediction` subsystem (fig. 5.5): this quantity sets the number of for-loop iterations, meaning the subsystem repeats its execution n times at each time step. Through the computation of the `GlobalObjectID`, unique object ID, that characterizes an object in the list of all the items in the scenario, is possible to collect all the data inherent to the specific traffic object using the `TrafficObject` block. It delivers the information about its absolute position (position in the global frame), translation velocity, rotation and acceleration. This data, in form of a bus element, are fed into the `GetTrafficObjectMeasurements` subsystem (fig. 5.6), responsible to collect the previous 20 state measurements, that shall be delivered to the single trajectory prediction process throughout the `TrafficObjectMeasurements` bus.

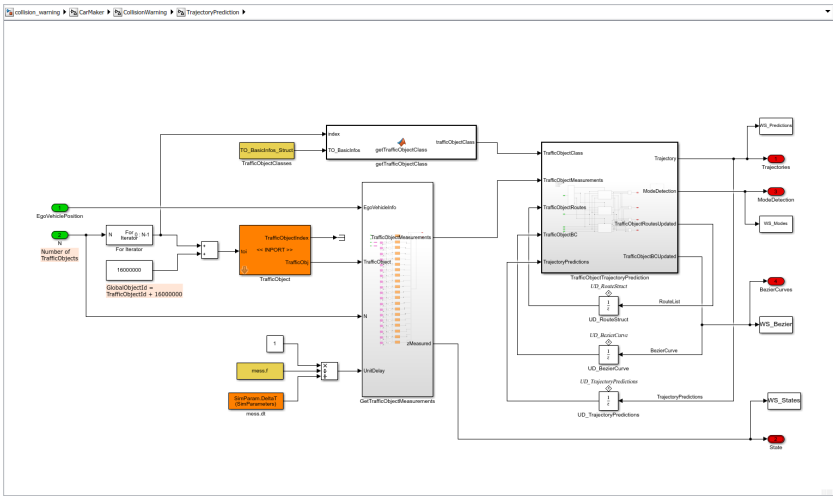


Figure 5.5: Subsystem dedicated to the calculation of the trajectory prediction of all the object in the simulation

The delay blocks configured with an external delay length allow to obtain a previous measurement at a certain time step in the past. It is worth noting, the difference between the simulation frequency ($f_{sim} = 100Hz$) and the sampling frequency ($f_{real} = 20Hz$), that is supposed to be used when working with a real use case. This means, at k time step, the previous measurement is not $k-1$, but $k - f_{sim}/f_{real} = k-5$. Furthermore, for each time step, n iterations are repeated, and this leads to a total delay length of $k - n \cdot f_{sim}/f_{real} = k - 5n$. The second last measurement is computed through a delay length of $k - 2 \cdot n \cdot f_{sim}/f_{real} = k - 10n$, the third last with $k - 3n \cdot f_{sim}/f_{real} = k - 15n$. Therefore, the delay length can be expressed by $k - i \cdot n \cdot f_{sim}/f_{real}$, where i is the i-th past sample. Please note, that the simulation sample time is read using a `Read CM Parameter`, that accesses the *SimParameter* infofile, where the `SimParam.Delta` parameter can be defined: the choice of 100Hz as frequency configuration, is due to an error raising with a greater sample time (0.02 would raise the following error *IPGDriver: Time stepsize too big at time = 0.080000: dt=0.020000 (> 0.02s) (id=300)*)
The traffic object coordinates shall be now translated into the FrO frame from the global frame, task performed by the `GetStateInFrO` block library, that create `TrafficObject` bus, whose signals are the position, the velocity (on the x and y axes), the heading angle, the traffic and global object ID.

Before proceeding with the real trajectory prediction computation, the

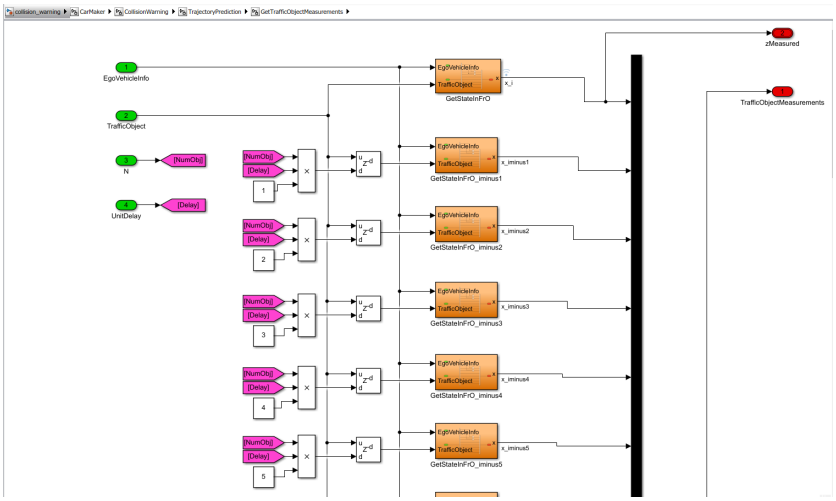


Figure 5.6: Subsystem that packs the last 20 measurement in the *TrafficObjectMeasurements*

type of the detected traffic object shall be defined: given the traffic object id, the `TO_BasicInfos` bus is accessed, to return its object class. For this purpose, the Matlab Function shown in fig. 5.7 has been implemented: the `switch-case` solution is necessary, due to the impossibility of a dynamic access of a field or property of a non scalar struct (as in this case) or object is not supported for code generation. The returned enumeration class is passed as well to the `TrafficObjectTrajectoryPrediction` subsystem. Other input of this subsystem are the 20 last traffic object measurments (`TrafficObjectMeasurements` bus), the previous `TrafficObjectRoutes`, `TrafficObjectBezierCurve` and `TrajectoryPrediction`, whose actual value is propagated outside the system, together with the `ModeDetection`.

5.1.3 2-4 Wheelers Trajectory Prediction

The case of a Cyclist is analogous to the one of a Vehicle, therefore the two will be discussed together in this paragraph, referring to the traffic object as a vehicle. The structure of this subsystem aims to split each step of the process, in order to be implemented as Matlab Function, separately one from another, keeping them as simple as possible (fig. 5.8).

```
function trafficObjectClass = getTrafficObjectClass(index, TO_BasicInfos)

trafficObjectClass = TrafficObjectClassEnum.Unknown;

switch index
    case 0
        trafficObjectClass = TO_BasicInfos.TO0.Class;
    case 1
        trafficObjectClass = TO_BasicInfos.TO1.Class;
    case 2
        trafficObjectClass = TO_BasicInfos.TO2.Class;
    case 3
        trafficObjectClass = TO_BasicInfos.TO3.Class;
    case 4
        trafficObjectClass = TO_BasicInfos.TO4.Class;
    case 5
        trafficObjectClass = TO_BasicInfos.TO5.Class;
    case 6
        trafficObjectClass = TO_BasicInfos.TO6.Class;
    case 7
        trafficObjectClass = TO_BasicInfos.TO7.Class;
end

end
```

Figure 5.7: Matlab Function for retrieving the object class by accessing the *TO_BasicInfos* bus

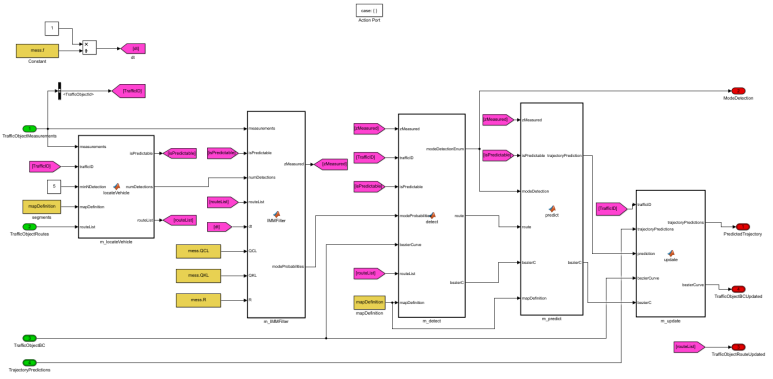


Figure 5.8: Vehicle and Cyclist TrajectoryPrediction system

Locate the Object on the Map

First of all, the `m_locateVehicle` Matlab Function checks the fundamental pre-conditions, detectability and predictability, for a lane change detection. The first defines whether the traffic object is in the area delineated by a circle with origin in the center of the mass of the scissor lift and radius equal to 100m (please note, this value can be set in the setup configuration by the user). The latter one determines if the number of object detections exceeds a threshold. The analysis carried out during the development of the system identifies a threshold equal to 5 reasonable, the IMM response working with 5 measurement seems to be acceptable. This means that the first prediction is computed, earliest, after 0.2s (available state measurement at 0s, 0.05s, 0.1s, 0.15s, 0.2s).
The 2-4 wheeler objects, proven to be detectable and predictable, are then located on the map, computing the route where they are driving along. The `roadmap.getRoute()` function is of the utmost importance in this context: it searches not only the closest segment, proving firstly that the object is inside the segment boundaries, but also that the segment direction matches with the object heading angle. This pre-selection helps to reject immediately too distant or wrong-way road segments. Summing up, if the vehicle is inside the road boundaries (and the distance between the route and the traffic object beneath a threshold) and the route direction matches the object heading angle, than the segment can be selected. Due to inaccuracies coming from the estimation of the best-fitting 5th degree polynomial during the mapDefinition database creation, a fallback mechanism has been implemented, in order to handle wrong initial and final direction angles of the segment: this method saves all the *possibleRoutes*, whose direction seems not to meet the pre-conditions, giving them "a second chance" if the no exact route has been found. In this event, the closest *possibleRoutes* route (prerequisite is a distance below the threshold) is returned.

Lane-Change Detection

The `m_IMMFilter` computes the mode probabilities related to the KeepLane and ChangeLane motion models in act at the time step, by means of the IMM Filter (for further details, please refer to section 4.1.3). Firstly the IMM Filter is initialized, hence, the two required EKFs, shall be configured. Eq. 5.1 and Eq. 5.2 show respectively the measurement noise and process

noise matrix: while the latter is the same for both of the EKF, the former differs owing to the heading angle.

$$Q_{CL} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.15 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2.0. \end{bmatrix} Q_{KL} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.0205 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2.0. \end{bmatrix} \tag{5.1}$$

$$R = \begin{bmatrix} 0.2500 & 0 & 0 & 0 & 0 \\ 0 & 0.2500 & 0 & 0 & 0 \\ 0 & 0 & 0.0004 & 0 & 0 \\ 0 & 0 & 0 & 0.0001 & 0 \\ 0 & 0 & 0 & 0 & 0.099. \end{bmatrix} \tag{5.2}$$

The probabilities of filter model transitions are set to:

$$R = \begin{bmatrix} 0.989 & 0.011 \\ 0.019 & 0.981 \end{bmatrix} \tag{5.3}$$

For a complete initialization of the IMM Filter, the initial state, as well as the initial state error covariance matrix, that is initialized with an identity matrix.
The initial state measurement received from the previous block, shall be processed, in order to transform it from the form (5.4) to the one requested by the motion models (5.5).

$$\begin{bmatrix} x & y & v_x & v_y & \theta \end{bmatrix}^T \tag{5.4}$$

$$\begin{bmatrix} x & y & \theta & v & w & a \end{bmatrix}^T \tag{5.5}$$

As the scissor lift is rotated of $\frac{\pi}{2}$, the heading angle delivered by CarMaker, must be compensate and normalized, adding $\frac{\pi}{2}$. The velocity, on the other side, is derived by means of v_x and v_y. Acceleration and the yaw rate are initialize to 0.
As soon as the initialization is completed, the filter runs on the set of measurements (min. 5 measurements, max. 20 measurements). For each step,

the predict function is immediately followed by the correction function, fed up with the real measurement: the yaw rate and the acceleration result respectively from Eq. 5.6 and Eq. 5.7.

$$w = \frac{\theta_i - \theta_{i-1}}{dt} \tag{5.6}$$

$$a = \frac{\sqrt{v_{x|i}^2 + v_{y|i}^2} - \sqrt{v_{x|i-1}^2 + v_{y|i-1}^2}}{dt} \tag{5.7}$$

The output of the filter, `modeProbabilities`, is propagated through out the model as a two-elements vector: the first element represents the probability that the object is in a `KEEP_LANE` mode, while the second one that the object is going to `CHANGE_LANE`.

Map-Based Prediction

The filter cannot distinguish between a change lane and a vehicle/cyclist driving along a curve, although they represents two completely different maneuvers: the first one would follow a Bézier Curve, while the second only the course of the road. For this reason, the `m_modeDetection` block is considered apart, despite the strict dependency from the mode probabilities calculated in the previous step by the IMM filter.
This function incorporates the computed probabilities with the map definition, to identify the `CURVING` mode. Below, an example of the algorithm adopted:

Algorithm 1: Get the driving mode status

```
Result: TrafficObjectStatusEnum
if modeProabilities(1) > modeProbabilities(2) then
    if isVehicleCurving then
        return TrafficObjectStatusEnum.CURVING;
    else
        if BezierCurve not yet defined then
            defineBezierCurve();
        end
        return TrafficObjectStatusEnum.CHANGE_LANE;
    end
else
    return TrafficObjectStatusEnum.KEEP_LANE;
end
```

As briefly illustrated in the algorithm, the Bézier Curve for a change lane maneuver is defined if not already initialized: this happens at the first change lane detection.
Hence, all the information, such as the actual route on the map of the traffic object, as well as the driving mode and the actual measurement state, are available and the proper trajectory prediction can start.
Crucially important is that the prediction is not computed for a specific instant in the future, but for a set of intervals, as described in section 4.4: from 0.5s until 4s with unit interval of 0.1s $(0.5, 0.6, 0.7, ..., 4)$. Thus, the 36 predictions are computed for each simulation time step, and results are stored in a 2D array, as:

$$\begin{bmatrix} 0.5 & x_{1|min} & x_{1|max} & y_{1|min} & y_{1|max} & v_{1|min} & v_{1|max} & R_{1|min} & R_{1|max} \\ 0.6 & x_{2|min} & x_{2|max} & y_{2|min} & y_{2|max} & v_{2|min} & v_{2|max} & R_{2|min} & R_{2|max} \\ ... & ... & ... & ... & ... & ... & ... & ... & ... \\ 4 & x_{36|min} & x_{36|max} & y_{36|min} & y_{36|max} & v_{36|min} & v_{36|max} & R_{36|min} & R_{36|max}. \end{bmatrix} \tag{5.8}$$

The *min* and *max* subscripts indicated respectively the minimum and maximum position that can be reached from the traffic object when integrating an error on the velocity, $\pm\Delta V = 10\%$. It shall not be ruled out, that this couple of positions on the map, can be located on different routes: hence, the $R_{i|min}$ and $R_{i|max}$ are stored as well, to calculate, if necessary, the area occupied by the object during the collision prediction algorithm (Section

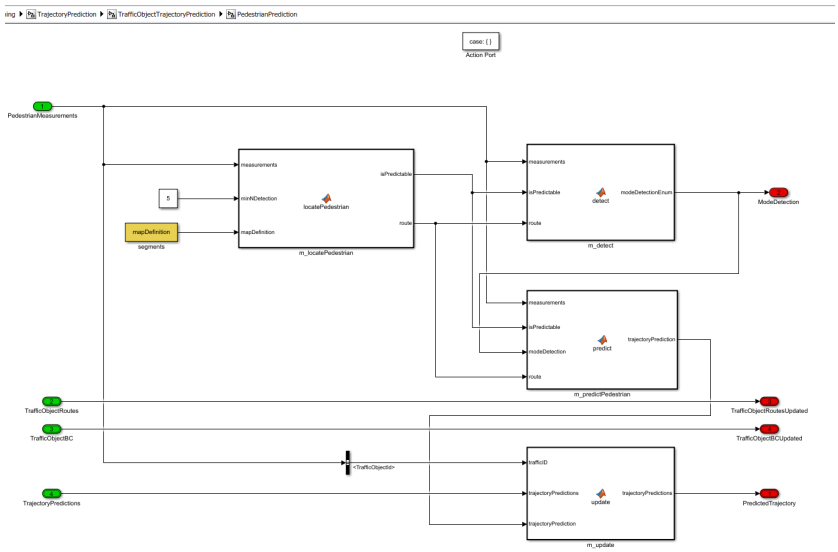


Figure 5.9: Pedestrian Trajectory Prediction system

5.1.5).

5.1.4 Pedestrian Trajectory Prediction

The workflow for a pedestrian prediction, though an analogous main process (fig.5.9), mostly diverges from the 2-4Wheelers', as already discussed in Section 4.3.
After checking the pre-conditions (detectability and predictability), as explained in 5.1.3, they are extended with a further requirements: the pedestrian shall be close enough to at least one route segment, meaning his potential interaction with vehicles or bikes on the road. This approach ensures to ignore pedestrians in parking slots just waiting to get in the car, for example: it leads to fewer calculations and, thus, a smaller computational load.

Are the preconditions satisfied, the `m_modeDetection` returns whether the pedestrian is in a `TrafficObjectStatusEnum.MOVING` or `TrafficObject StatusEnum.WAITING` status, that is trivially set when the velocity of the pedestrian is negligible (less than $10^{-2} m/s$).
Thus, the prediction computation starts: the output is here as well a 2D array, as shown in eq.5.8. The same considerations made about the time window (from 0.5s to 4s) and about the ΔV apply also in this case.

The trajectory prediction for both mode detections is equivalent (Section 4.3, but varies in the initial state definition.
When the pedestrian is waiting for crossing, his velocity tends to 0. The initial state is hence initialized with a minimum velocity and a maximum velocity (for each prediction, a *min* position and a *max* position is required). While the lower velocity, $1m/s$, has been defined considering old people (or Tyrannosaurus Rex, whose walking speed has been derived to be $4.6km/h$ by a recent study based on a biomechanical model), the upper threshold walking speed is $3m/s$, without taking into account that people would cross the road running, in order not to produce many false warning.

As shown in 5.4, the measurement vector delivered by CarMaker needs to be transformed, to be fed into the CA motion model (see Section 4.1.1), as a six-element vector (Eq. 5.9)

$$\begin{bmatrix} x & v_x a_x & y & v_y & a_y \end{bmatrix}^T \tag{5.9}$$

The initial state vector is, hence, derived as follows:

$$\begin{bmatrix} x \\ v_x \\ a_x \\ y \\ v_y \\ a_y \end{bmatrix} = \begin{bmatrix} x \\ v_{mean} cos(\theta_\perp) \\ 0 \\ y \\ v_{mean} sin(\theta_\perp) \\ 0 \end{bmatrix} \tag{5.10}$$

where $\theta_\perp$ is the angle defined by the perpendicular to the route direction, that the pedestrian is supposed to cross.

On the other hand, if the pedestrian is detected to be moving (crossing or not), the velocity is already defined by the measurement vector and the error ($\pm\Delta v$). The current state is therefore obtained by Eq. 5.11:

$$\begin{bmatrix} x \\ v_x \\ a_x \\ y \\ v_y \\ a_y \end{bmatrix} = \begin{bmatrix} x \\ v_x + v_x \Delta V cos(\theta) \\ \frac{v_{x|i} - v_{x|i-1}}{dt} \\ y \\ v_y + v_y \Delta V sin(\theta) \\ \frac{v_{y|i} - v_{y|i-1}}{dt} \end{bmatrix} \tag{5.11}$$

where θ is simply the traffic object heading angle.

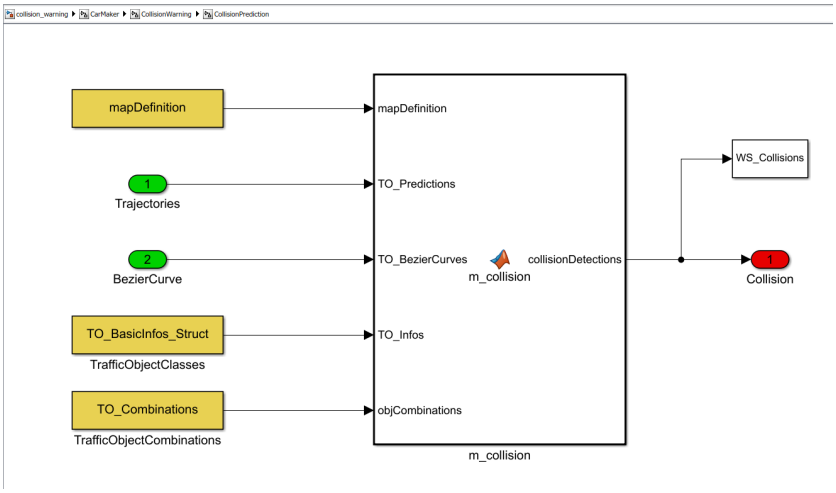


Figure 5.10: Collision prediction system

The CA motion model, finally, calculates the state in the future based on the current state (initial state) and the time interval.

5.1.5 Collision Prediction

After calculating the required input, such as the `TrajectoryPredictions` and the `TO_BezierCurves`, the process, that evaluates a possible collision between pair of objects (excluding pairs $(Pedestrian, Pedestrian)$), starts (fig. 5.10).

A code snippet of the Matlab Function is shown in fig. 5.11 The output matrices (one for each object), defining the trajectories, are iterated through and at each time step prediction a possible collision is evaluated: a collision is detected if two areas occupied by two objects have at least one point overlapping (*conflict area*, see Section 4.5) at the same time step. Hence, a pre-selection is performed in order to reduce the computational time of this process, that drastically use the number of iteration through the matrices. The function `collision.canTrajectoryIntersect()` checks if the two objects' trajectory could intersect, by comparing the coordinates of the *min* and *max* positions: these are considered as boundaries of two segments that can intersect if:

- $x_{i|min}$ or $x_{i|max} \in [x_{j|min}, x_{j|min}]$ where $i = 1, 2; j = 1, 2; i \neq j$

```
function collisionDetections = m_collision(mapDefinition, TO_Predictions, TO_BezierCurves, TO_Infos, objCombinations)
coder.extrinsic('combvec');

collisionDetections = -ones(8, 3);

xColumns = [2, 3];
yColumns = [4, 5];

for iComb=coder.unroll(1:size(objCombinations, 1))
    objPair = objCombinations(iComb, :);

    % get single information about obj n1
    [objData1, objBezier1, objClass1, objDimensions1] = getTOInfo(TO_Predictions, TO_BezierCurves, TO_Infos, objPair(1));
    % get single information about obj n2
    [objData2, objBezier2, objClass2, objDimensions2] = getTOInfo(TO_Predictions, TO_BezierCurves, TO_Infos, objPair(2));

    if all(objData1(:, 2:end) == 0) || all(objData2(:, 2:end) == 0)
        continue;
    end

    if objClass1 == TrafficObjectClassEnum.Pedestrian && ...
            objClass2 == TrafficObjectClassEnum.Pedestrian
        continue;
    end

    for t = 1:size(objData1, 1)
        if ~collision.canTrajectoriesIntersect(objData1(t, xColumns), objData1(t, yColumns), ...
                objData2(t, xColumns), objData2(t, yColumns))
            continue;
        end
        x = collision.getCollisionBetweenTwoTrafficObjects(objData1(t, :), objClass1, objBezier1, objDimensions1, ...
            objData2(t, :), objClass2, objBezier2, objDimensions2, ...
            mapDefinition);
        if ~isempty(x)
            collisionDetections(objPair(1)+1, :) = [1, t, objPair(2)];
            collisionDetections(objPair(2)+1, :) = [1, t, objPair(1)];
            break;
        end
    end
end
end
```

Figure 5.11: Collision prediction Matlab Function

- $y_{i|min}$ or $y_{i|max} \in [y_{j|min}, y_{j|min}]$ where $i = 1, 2; j = 1, 2; i \neq j$

As the trajectories still describe the path of the center of mass of the object (width and height are not yet integrated), the function relies on the use of a tolerance (2). A square, whose sides represents twice the tolerance set) is drawn around the point, to simulate the maximum width and length an object can have. Since this specific function is only called during the pre-selection, it is acceptable to consider such a tolerance for a pedestrian as well.

The boolean flag returned allows the process of collision evaluation to go on in the detection through the function `collision.getCollision BetweenTwoTrafficObjects()`, otherwise the next time step is stepped over. Firstly the center of mass trajectories are intersected: if they cross each other, there is no need to *expand* the object. The predicted path is described by a polynomial (the route segment polynomial or the Bezier Curve polynomial) stored in the prediction matrix, with boundaries defined at *min* position and *max* position. If a common real root within the boundaries is found, then a collision is predicted. By contrast, the traffic object center of mass is *expanded*, by defining two polynomial, that describe the trajectory of the two sides (the first at $-width/2$

and $+width/2$) of the traffic object (a more detailed explanation is provided in Section 4.5). It's worth mentioning, that in order to define these best-fitting polynomials, the centering and scaling values μ are calculated as well through the Matlab `polyfit` function. μ is a two-elements vector with centering and scaling values, that `polyfit` uses to centers x at zero and scales it to have unit standard deviation: $\mu(1)$ is the mean value of x, while $\mu(2)$ is the standard deviation of x. This centering and scaling transformation improves the numerical properties of both the polynomial and the fitting algorithm. These polynomials identify the region predicted to be occupied by a traffic object at a specific time step. A collision is predicted, if at least one extreme of the first area resides inside the second region.

The output is given as 2D-Vector, where each row defines a traffic object, the first columns defines whether a collision will happen (1) or not, while the second and third position store, respectively, the time of collision and the index of other traffic object involved.

5.1.6 Simulation Outputs

A *StopFcn* Callback is configured, to execute a `postProcess.plotStaticResults()`, after the simulation stops. This function displays the simulation results, accessing the following timeseries in the Workspace:

- `WS_Collisions`: 2D-Vector containing information about predicted collision at each time step as explained at the end of section 5.1.5;
- `WS_Predictions`: `TO_TrajectoryPredictions` Simulink.Bus;
- `WS_Modes`: mode detection as `Enum: TO_StatusEnum`, for each traffic object at each simulation time step;
- `WS_States`: CM measurement as `TO_CMMeasurement`, Simulink.Bus for each traffic object at each simulation time step;

An example of the figure resulting from the plot, is shown in fig. 5.12. A detailed description follows in Section 6.1.1.

5.2 Graphical User Interface

It is worth mentioning the developed user-friendly GUI, to ease the simulation of some different pre-defined CarMaker scenarios and the plot, hence, the

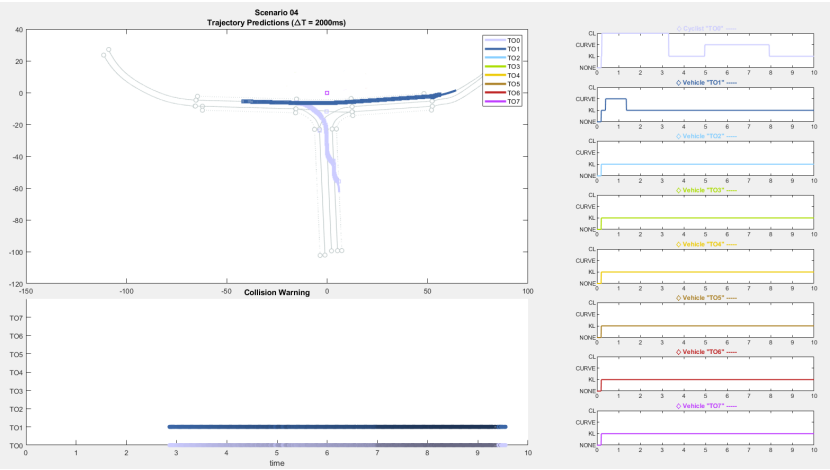


Figure 5.12: Example of plot create at the end of simulation, for a first evaluation of the results

evaluation of the results.

The two panels on top in fig. 5.13 allows to choose two different source where the results shall be loaded from. Throughout the *Simulate TestRun* dashboard, a pre-defined scenario can be simulated and, if the *Generate Plot* is enabled, results are plotted and displayed. The selection of a specific scenario loads the scenario recorded video, if available. Please note, the video is not recorded real-time during the simulation from CarMaker. However, a further different video, can be always selected in the *CM Recorded Video* search field. This is pretty useful when loading results from an existing .mat file using the *Load result* functionality. To enable the *Load Results* button a validation of the .mat file shall be performed 5.14: this checks that all the required variables (such as `WS_Collisions` or `WS_TrajectoryPredictions` for an eventual plot are contained in the file. If the validation files a small error message is displayed in the Status bar.

Is a video loaded, the play button is enabled, meaning the video can be started thereby. The time slider, that visualizes the temporal progress of the video, allows the user move the animation forwards and backwards.

After the simulation is terminated or after loading already existing results in the workspace, the *Collision Warning* and the *Custom Plot* panels are enabled. In the first one, during playback, the lamp indicators become yellow as soon as a collision for the corresponding object is predicted 5.15. If the

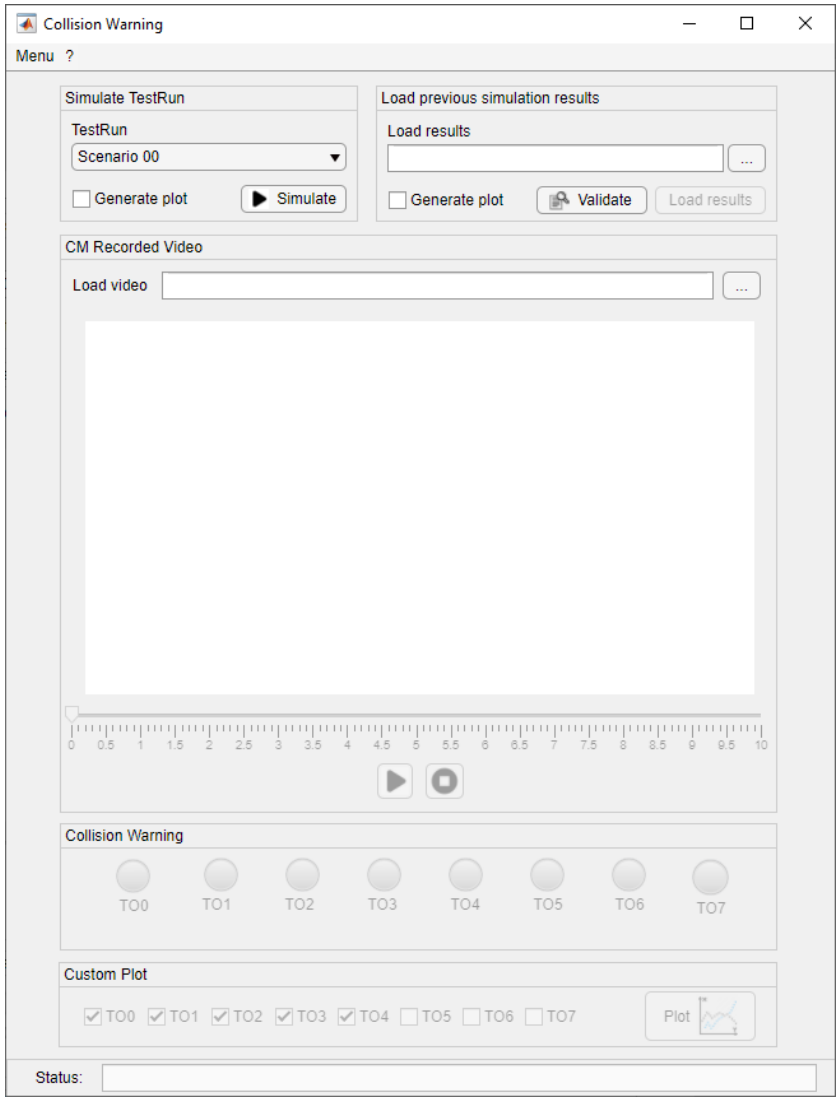


Figure 5.13: Screenshot of the GUI as soon as it opens

Figure 5.14: The load results button is now enabled, after the validation of the file. A custom video is selected, because of a different point of view.

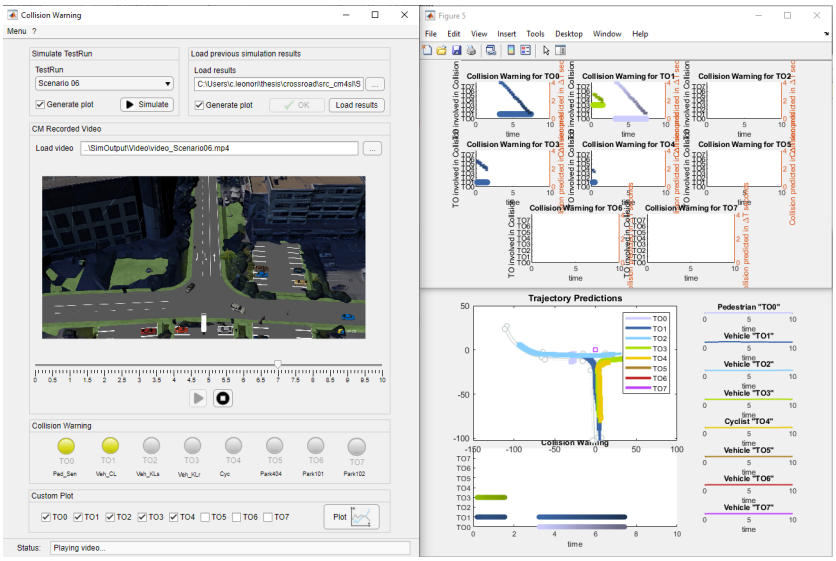


Figure 5.15: The two yellow lamps indicate a collision prediction between the two objects. On the right side the generated plots.

traffic objects names (configured in the CarMaker TestRun) are available in the Matlab workspace, they are display under each lamp, in order to ease the mapping between the traffic object number and the traffic object in the video.

In a more crowded scenario, the *Custom Plot* can help to plot only a specific set of traffic objects, g. This results in the plots shown in fig. 5.16: these will be deeply presented in the following Chapter.

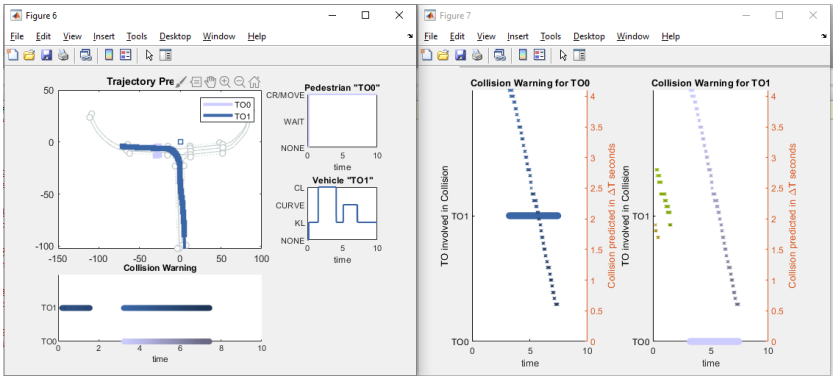


Figure 5.16: Resulting plot after selecting only the first two object in the GUI in the *Custom Plot*.

Chapter 6

Experimental results and evaluation

This chapter aims to present and discuss the results obtained during this work. In Section **??**, the CarMaker driving scenarios, designed to test and validate this work, are outlined. After giving separately insights into the observed data for the Lane Change module, the trajectory prediction component (for pedestrian and 2-4Wheelers) and, lastly, for the collision prediction (Section 6.1), a couple of more complicated scenarios are presented, to come to the final discussion and conclusion in section 6.2.

6.1 Test Driving Scenarios and Results

In order to validate and test the overall collision warning system, a scenario-based approach was chosen. By definition, a scenario is a ordered set of interactions between the system and external actors. Each scenario not only documents the system requirements, specifying the system behavior, but also validates the system while development. In total, 9 scenarios have been established and set up in CarMaker: some of them consider VRUs involved in accidents, whereas others demonstrate, how critical situations can be detected for pedestrians willing to cross the road and waiting on the road side.

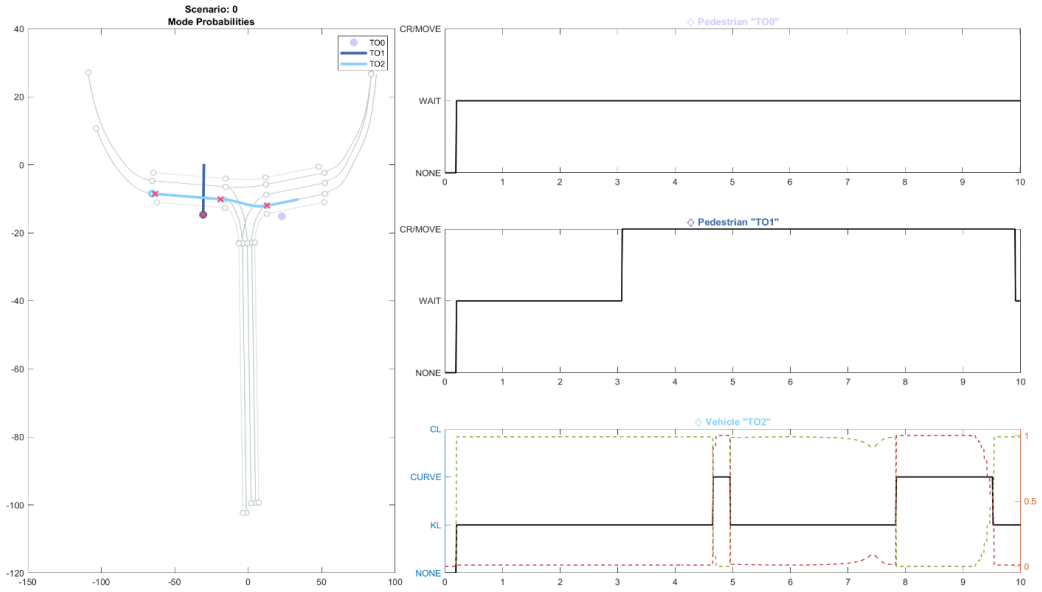


Figure 6.1: Mode Probabilities Plot for Lane Change of scenario 00.

6.1.1 Evaluation guidelines

To provide an overview of the employed evaluation methods, the output generated by the scenario 00 (sec. 6.1.2 is shown and described.

Mode Probabilities and Maneuver Detection The mode probabilities plot in fig. 6.1 aims to give an overview of results produced by the maneuver recognition module. The graph on the left shows the real traveled path of each traffic object. The red crosses illustrate at with position a change in the mode detection is detected. The circles indicate the start position. The graphs on the right illustrate in detail, not only the IMM filter output (green dashed line for the ChangeLane and red for the KeepLane, y axis on the right), but in general, also the traffic object status obtained integrating the map definition information (y axis on the left). Whereas pedestrians are characterized by a set of status `CR\MOVE` = crossing or moving, `WAIT` = standing, `NONE` = unknown, for other traffic objects the possible motion modes are `CL` = change lane, `CURVE` = turning left/right, `KL` = keep lane, `NONE` = unknown.

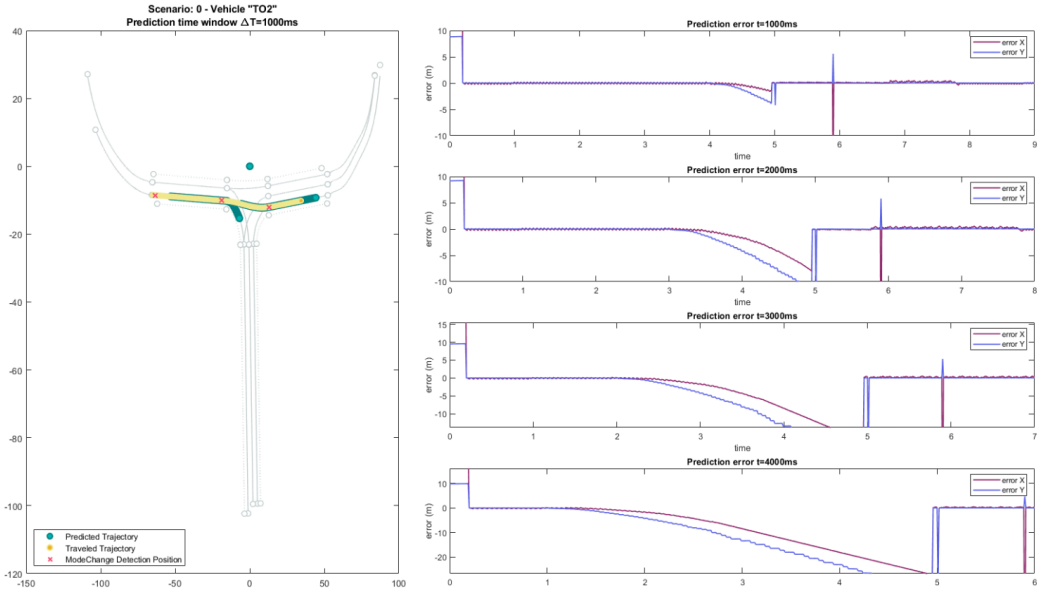


Figure 6.2: Trajectory Prediction Plot for vehicle *TO2* in Scenario00.

Trajectory Prediction Each object trajectory predictions are described in a plot analogous to fig. 6.2. On the left, the prediction computed for the time horizon 1s are displayed. The subplots on the right side aim to graph the error in the predictions of the x position and y position for $\Delta T = 1s, 2s, 3s, 4s$.

Collision Prediction Two plots are provided to detail the collision prediction between traffic objects. The first one (fig. 6.3), gives an overview of all the predicted trajectories, reporting, below, a schema of the potential critical situations for each traffic object. The shades on the plot aim to describe how distant the collision is/would be in terms of time: the soon it can happen, the dark it is plotted. The second graph (fig.6.4) aims to details the results, showing which other traffic object is involved in the predicted impact and the specific time horizon. This graph is fundamental for crowded scenarios, as the Scenario 07 (sec. 6.1.9)

The second graph explains more in details the collision prediction for each traffic object, in order to illustrate the second object involved in the potential accident (y axes on the left) and the time horizon in which it is predicted (y axes on the right).

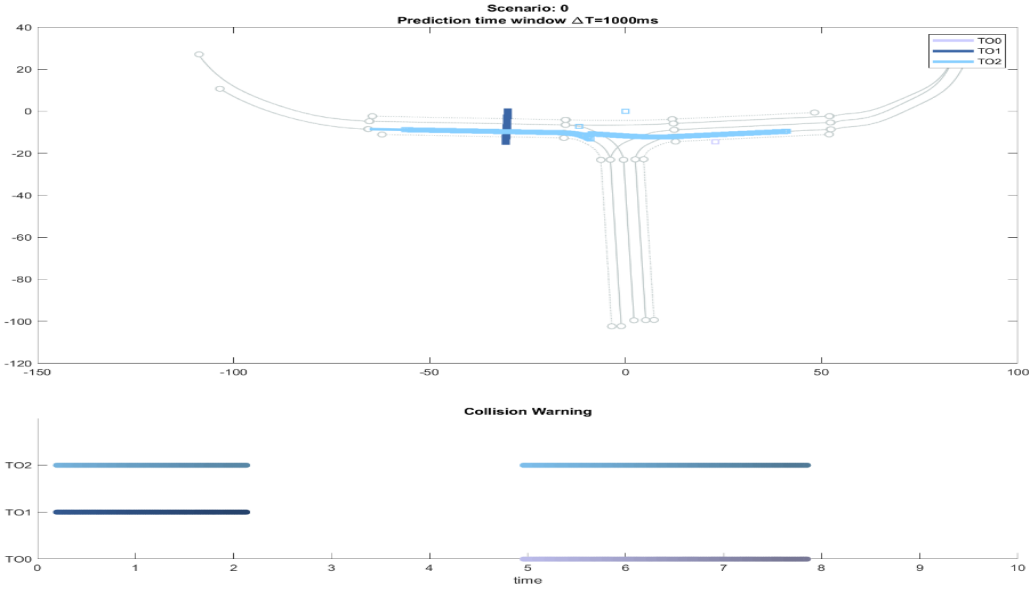


Figure 6.3: Simple collision prediction plot in Scenario00

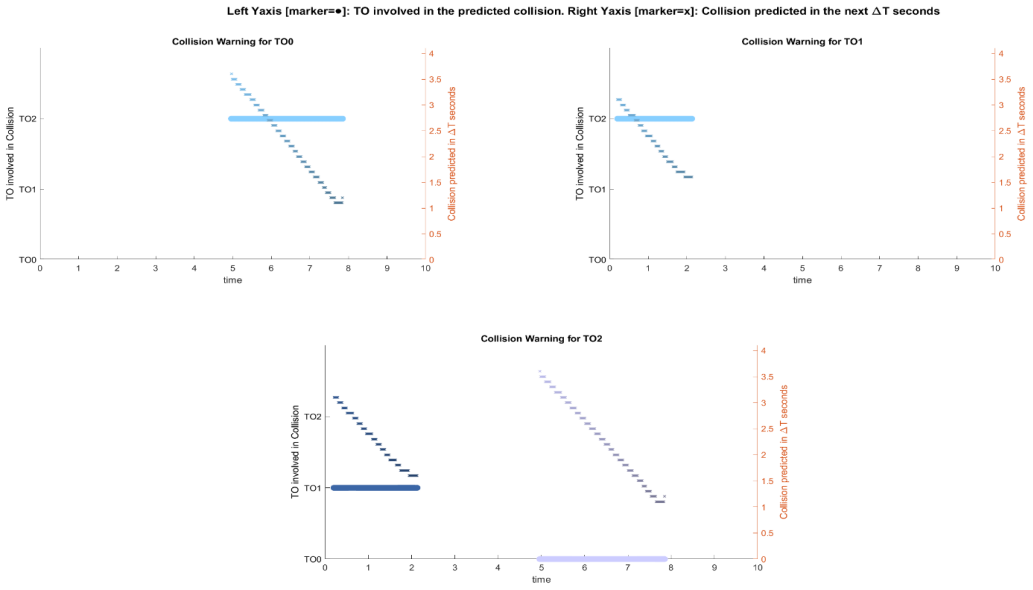


Figure 6.4: Detailed collision prediction plot in Scenario00

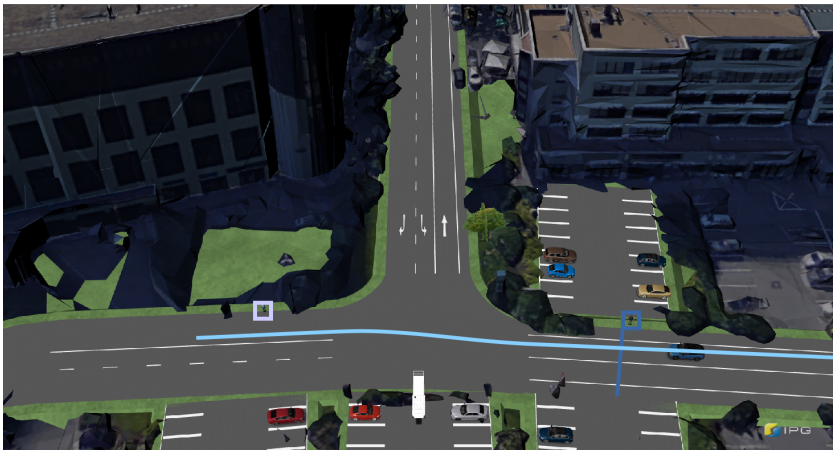


Figure 6.5: **Scenario00**: vehicle is driving straight on (sky blue). The two pedestrians are waiting for crossing (blue and white boundaries). At 3s, when the vehicle is passed by, the pedestrian on the right starts crossing the road (blue trajectory).

6.1.2 Scenario 00

The first scenario is the main baseline use case. A vehicle is driving straight on in the crossroad, from the left to the right, at a velocity of $v = 10m/s$ (light blue trajectory in fig. 6.5). Two pedestrians are standing on the side of the road. The pedestrian on the right (in blue) side waits for crossing, and as soon as the auto is passed by (at 3s), starts to cross the road at a velocity of 3m/s: the movement is only longitudinally, towards the opposite parking place. The pedestrian on the left (in white), instead, is just standing on the road side. There are no collision in this scenario, but warning are supposed to be raised, to signal the VRUs that a vehicle is approaching and a crossing the road would represent a critical situation.

Mode Detection The vehicle is configured to keep its route. The mode detection switches between `KL` and `CURVE`, in correspondence of a non linear path segment. Whereas the pedestrian `TO0` is only waiting, `TO1` is detected to start crossing the road at 3.06s (mini maneuver configured in CarMaker to start at 3s).

Trajectory Prediction `TO0` is displayed in Fig 6.6: the pedestrian does not move from its position, hence, the prediction for $\Delta T = 1s$ does not undergo changes: the error for any time horizon remain stable, as he/she does not reach the predicted position. Please note, the prediction error time line is shorter as the time window increase: the simulation span is 10s, thus, having a time horizon of $\Delta T = 4s$, the maximum time for which the prediction error value can be calculated is trivially $10s - 4s = 6s$.

Unlike `TO0`, as `TO1` keeps proceeding on its path, the generated path is updated as well. During the crossing, the error increases with the time horizon, while for prediction of $\Delta T = 1s$ is almost negligible (max. 0.32m on the y axes), it amounts to ca. 2.2m for $\Delta T = 2s$, reaching 6m for $\Delta T = 3s$, approaching 13.5m for $\Delta T = 4s$. Such a great error in the prediction is due to the acceleration, as the pedestrian motion model is not a constant velocity model: at the beginning of the action ($t = 3s$), has a velocity of $v = 0m/s$, after 0.5s it reaches $v = 1m/s$ and at 4s is already $v = 2m/s$. After some meters its acceleration starts to decrease until he stops. The motion model used in this work is a CA and together with the uncertainties on the velocity has delivered acceptable results, as described later in the collision prediction section.

The trajectory of vehicle `TO2` in fig. 6.2 is affected as well by errors: in this case, because of a not unique next segment for the traveled route (the vehicle could not only drive straight on but also, as wrongly predicted, exit the crossroad). Until 5s the path is generated along the route that exits the crossroad, but as soon as the vehicle is detected to be on the other (the correct one) route, the prediction is updated, and the errors significant decrease.

Collision Prediction Right from the start until $t_D = 2.13s$, a critical situation is predicted for the VRU `TO1`: a collision would occurs, if he starts cross the road. Afterwards, at $t_D = 4.96s$, as soon as the vehicle `TO1` is predicted to follow the correct route (and not the one exiting the crossroad), the VRU `TO1` shall be warned (until the vehicle is passed by at 8s more or less).

6.1.3 Scenario 01

Unlike the previous scenario, here (fig. 6.8) a collision is foreseen between the vehicle and the pedestrian (the only two traffic objects populating the crossroad). The vehicle coming from the left, exits the crossroad with a turn

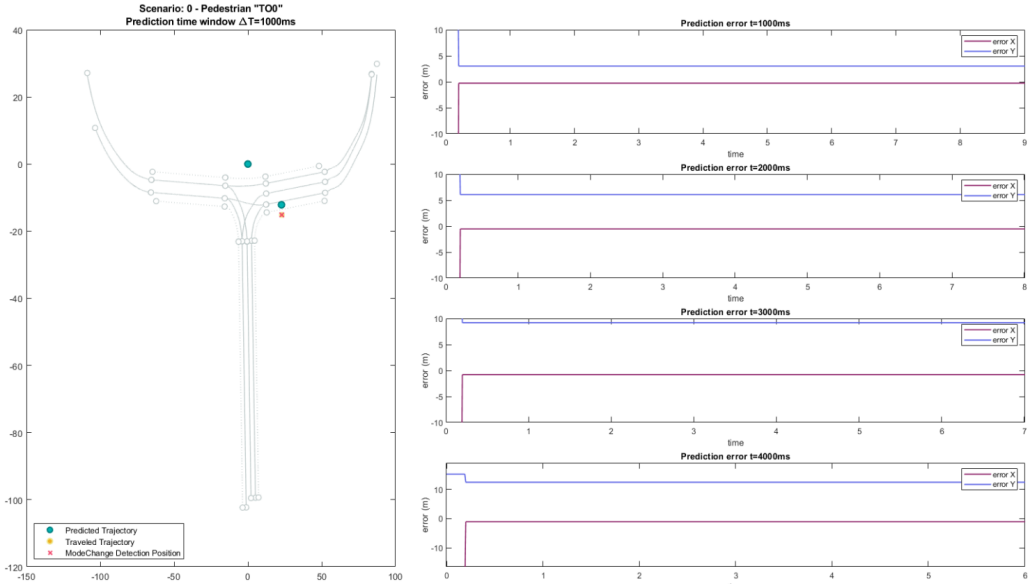


Figure 6.6: Trajectory Prediction Plot for pedestrian *TO0* in Scenario00.

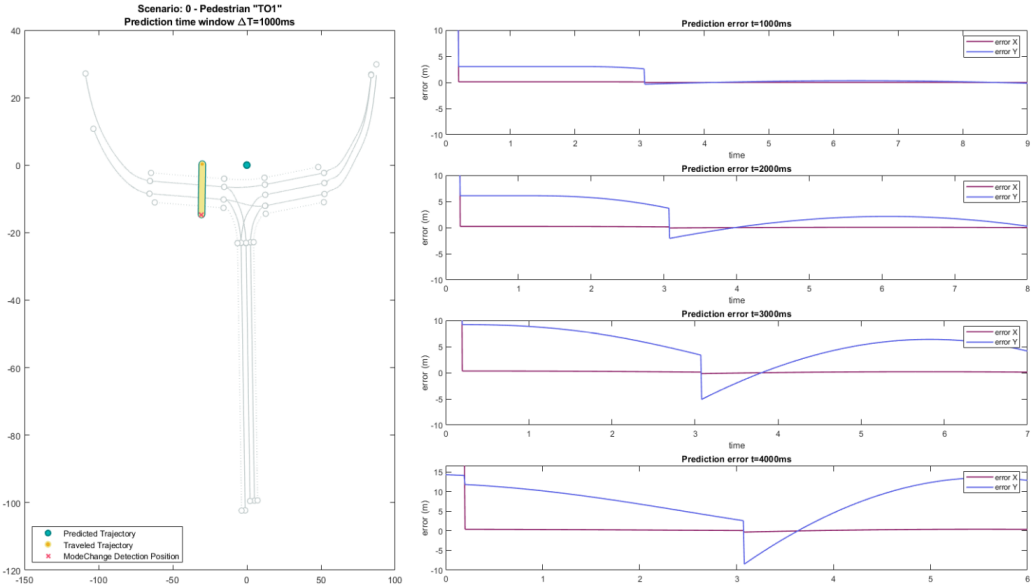


Figure 6.7: Trajectory Prediction Plot for pedestrian *TO1* in Scenario00.

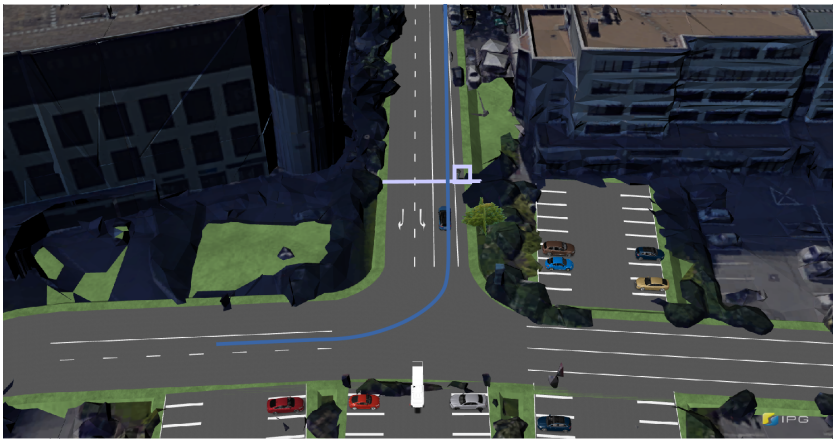


Figure 6.8: **Scenario01**: vehicle is exiting the crossroad (white) and pedestrian is crossing the road (blue boundaries).

left. Its velocity is set to $10m/s$. Suppose a rainy day, a pedestrian equipped with an umbrella has a limited visibility and overlooks the approaching auto. Thus, he/she starts to cross the road after 3s, with a maximum velocity of $v_{max} = 2.5m/s$. Unfortunately, a collision at $t_C = 5.3s$ is inevitably.

Collision Prediction At $t_{PC} = 1.52s$, a possible collision for the pedestrian is predicted, with a time horizon of $\Delta T = 4s$. The warning signal is disabled at $t = 3.04$. It restarts, though, at $t_{PC} = 3.28s$ (time horizon $\Delta T = 3s$). The warn remains enabled until the pedestrian is overrun ($t_C = 5.3s$). Thus, the collision is initially foreseen $\delta t = 3.78s$ before is happens.

The small time span, when no risk is foreseen, is due to the pedestrian's acceleration required to pass from a standing position to a velocity of $v = 2.5m/s$. For a more detailed explanation, please refer to the Scenario00, where an analogous use case for the `TO1` is explained. Fig. 6.11 shows the same pattern for the prediction error seen in the previous scenario.

The error to which the vehicle `TO1` is kept small, for all the time horizons (fig. 6.12).

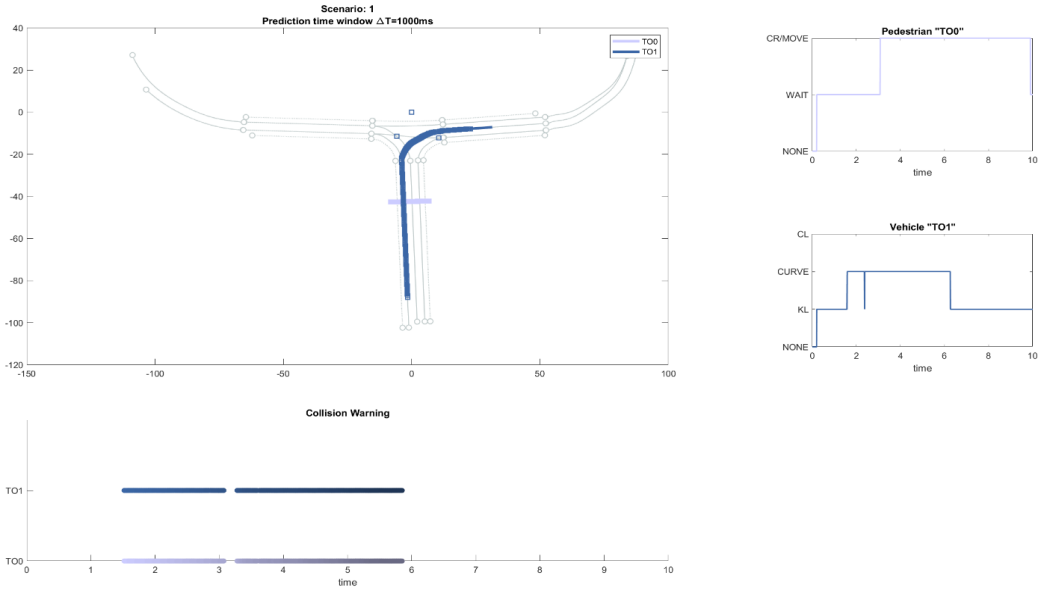


Figure 6.9: **Scenario01**: Collision prediction graph.

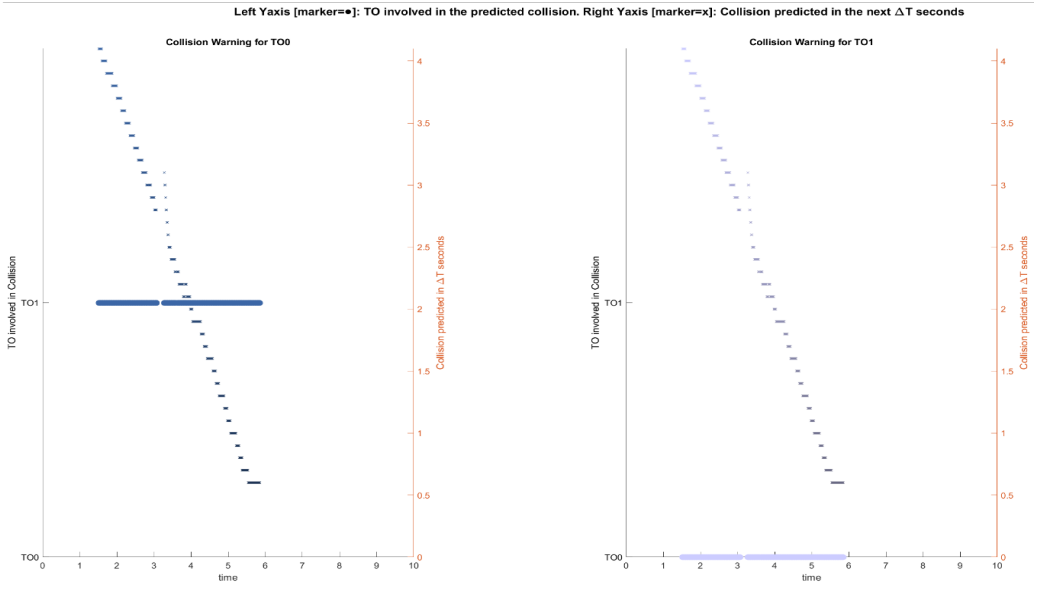


Figure 6.10: **Scenario01**: A more detailed collision graph to document the predictions between *TO0* and *TO1*.

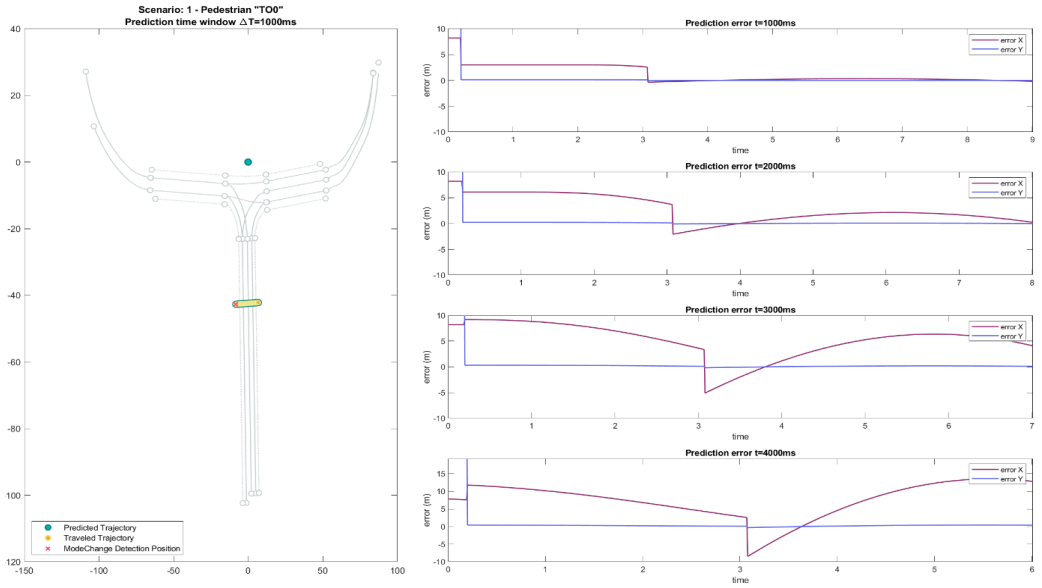


Figure 6.11: **Scenario01, TO0**: Prediction error graph. The pattern is analogous to the one of *TO1* in Scenario01.

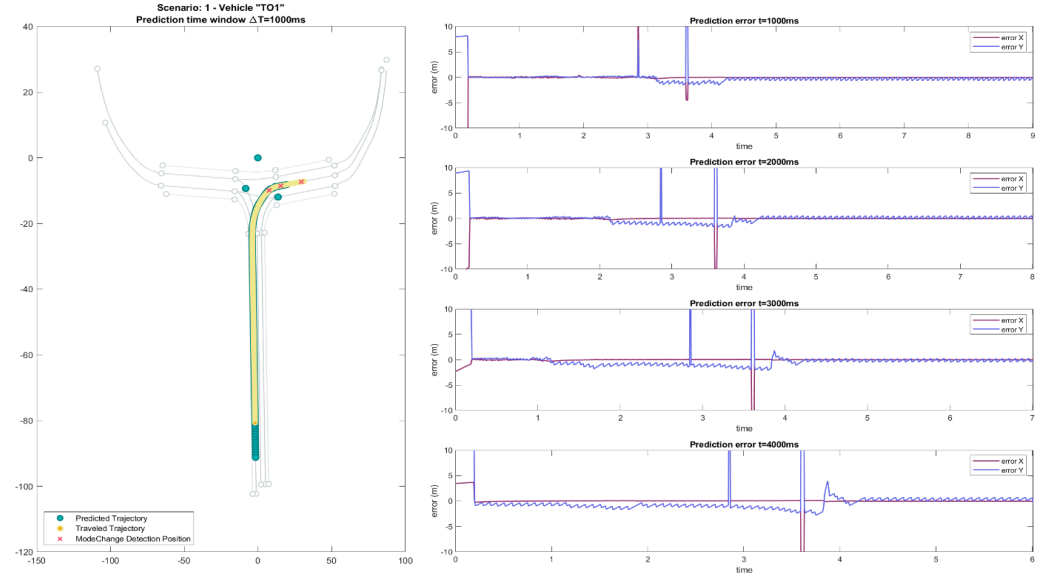


Figure 6.12: **Scenario01, TO1**: Prediction error graph that show the error in both axes for the different time horizons.

Figure 6.13: Scenario 2: vehicle(white) is entering the crossroad, and after a lane change maneuver turns left. The pedestrian is hit at 7.2s (blue boundaries).

6.1.4 Scenario 02

In this case (fig. 6.13) a pedestrian (blue) and a vehicle are involved in the impact at time $t_C = 7.2s$. The vehicle enters the scene and performs a change lane maneuver in the first 5s, to turn left at the intersection. Its velocity is constant. The pedestrian, an old woman, starts slowly to cross the road with a longitudinal velocity of $v_x = 1m/s$ and a latitudinal of almost $v_y \approx 1m/s$.

Mode Detection This scenario has been mainly used to validate the Bézier Curve computation with path prediction for a lane change maneuver. As documented in fig. 6.14, the change lane mode detection is over the time span $t = [1.51s, 4.5s[$. At the first detection, the vehicle has not yet cross the lane line (fig. 6.15). Then, the keep lane mode prevails, until the vehicle starts turning left ($t = 5.15s$).

Trajectory Prediction Fig. 6.16 shows the slight error on the predicted Bézier Curve, that has a peak of only $0.8m$, at $t = 3.36s$, during the maneuver time window $t = [1.51s, 4.5s[$. Afterwards, a relative greater error is due to the adjustment of the prediction to the updated driving mode. It's worth mentioning, that the error on the pedestrian generated path is acceptable (fig. 6.17). The two peaks present (for each ΔT) are generated by the abruptly change of direction of the old woman.

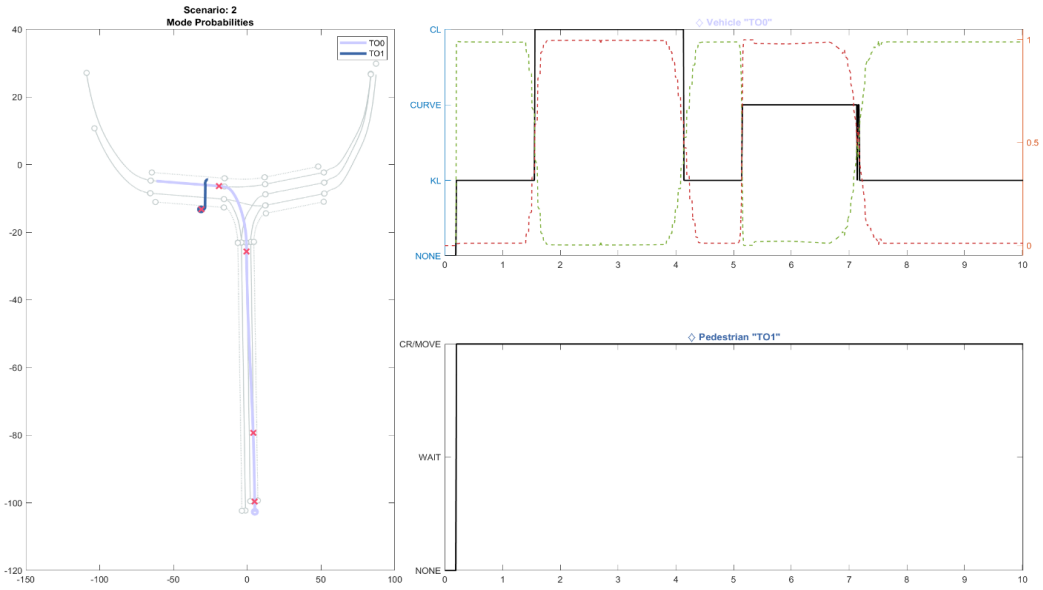


Figure 6.14: **Scenario02**: Mode detection graph. The vehicle *TO0* is detected to change lane after 1.51s the beginning of the maneuver.

Figure 6.15: Position of the vehicle *TO0* during the first lane change detection: it has not yet crossed the lane line.

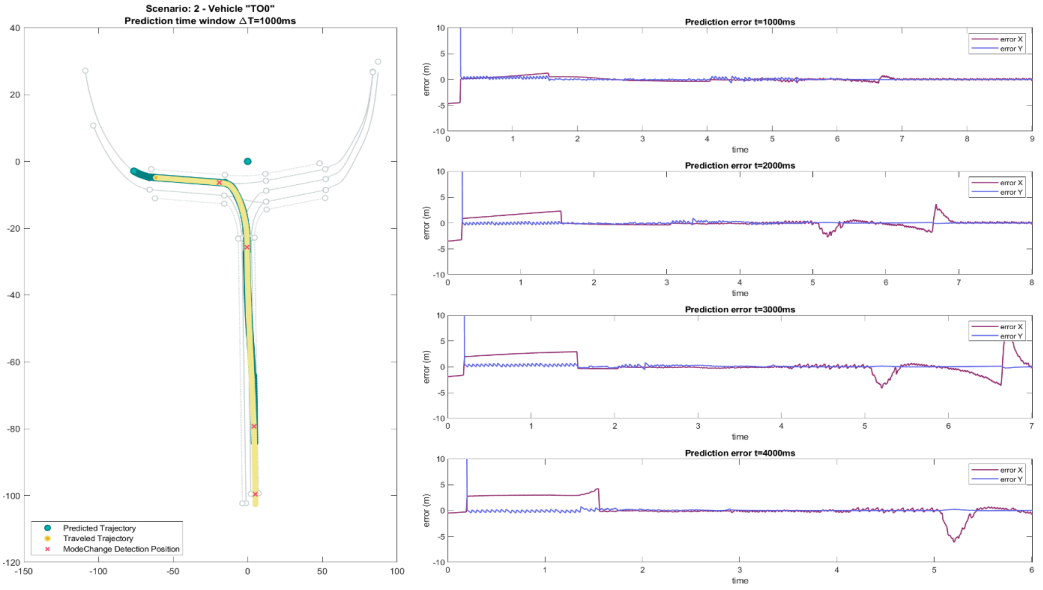


Figure 6.16: **Scenario02, TO0**: Prediction error graph. The Bézier Curve computed during for the lane change maneuver is subjected to negligible errors.

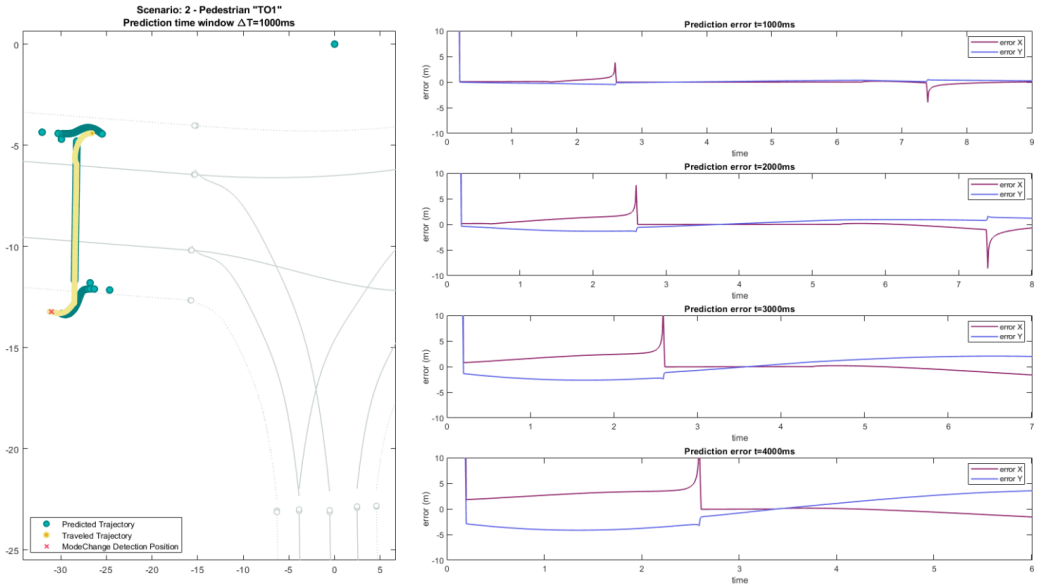


Figure 6.17: **Scenario02, TO1**: Prediction error graph that show the error in both axes for the different time horizons.

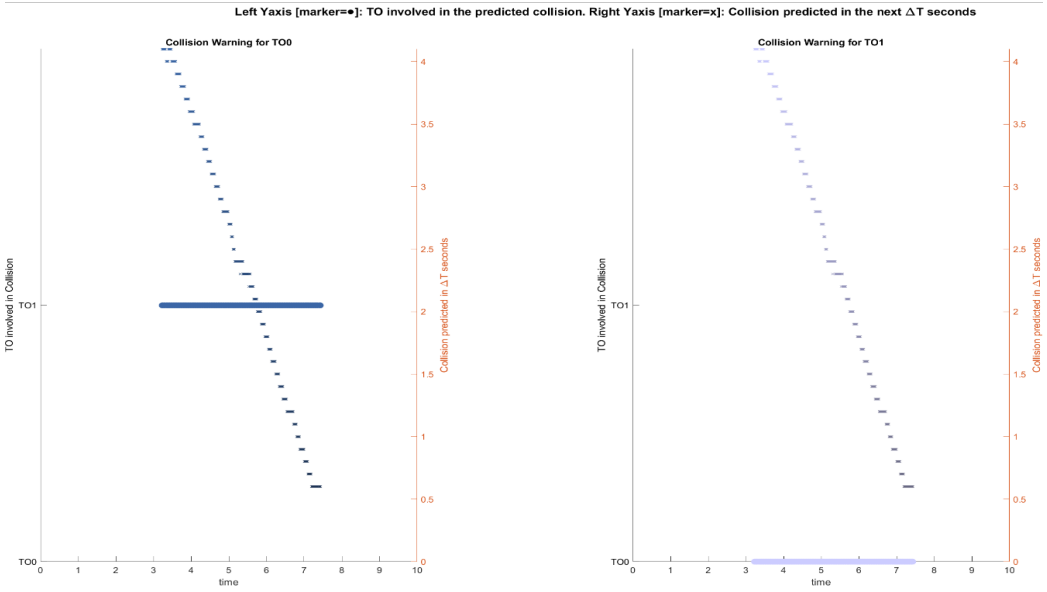


Figure 6.18: **Scenario02**:Collision prediction graph. The first detection is 4s before the impact ($t = 3.23s$).

Collision Prediction Another important result in this scenario is given by the collision prediction, that is firstly foreseen at $t_{PC} = 3.23s$, 4s before it happens.

6.1.5 Scenario 03

A further scenario (fig. 6.19) used to validate the pedestrian motion and the collision prediction is the fourth one. Two children are in the parking place next to the scissor lift. Whereas the female (sky blue boundaries) stands still, the male (white boundaries) moves initially longitudinally ($v_x = 1m/s$) and then start crossing the road while looking at the smartphone and without paying attention to the approaching vehicle. This, after an initial speed of ($v = 10m/s$), accelerates as the intersection seems to be free, reaching after 5s a speed of $v = 15m/s$ and hits the pedestrian at $t = 5.7s$.

Trajectory prediction Fig. 6.20 shows the prediction error for the trajectory generation of `TO0`. During the first 1s time window, the high error is due to the abrupt movement of the child and the varying acceleration. As

Figure 6.19: Scenario 3: vehicle(blue) is entering the crossroad and turns left. The pedestrian is hit at 5.7s (white boundaries).

soon as he begin the cross road maneuver, the error decreases, stabilizing at lower values ($max_{\Delta T=1s} = -0.01m, max_{\Delta T=2s} = -0.08m, max_{\Delta T=3s} = -2.5m, max_{\Delta T=4s} = -5.5m$).

Collision prediction A dangerous situation is foreseen for the female pedestrian `TO2` at $t = [1.03s, 2.63s[$, in case she would move towards the opposite parking place. The real collision between the vehicle `TO1` and the child `TO0`, at $t_C = 5.7s$, is firstly detected at $t_{CP} = 1.78s$, with a time horizon of almost $\Delta T = 3.9s$.

6.1.6 Scenario 04

An other type of VRU appears in the Scenario04 (fig. 6.22), a cyclist. Starting on the bicycle lane route, he changes its lane twice straight, to be able to turn left at the intersection. After 1.5s he has completed the first lane change and at 3s the second one. Then he proceeds with a velocity $v = 9m/s$. A vehicle coming from the left side seems not to see the cyclist and hit him at $t = 7s$, almost at the end of his turn maneuver.

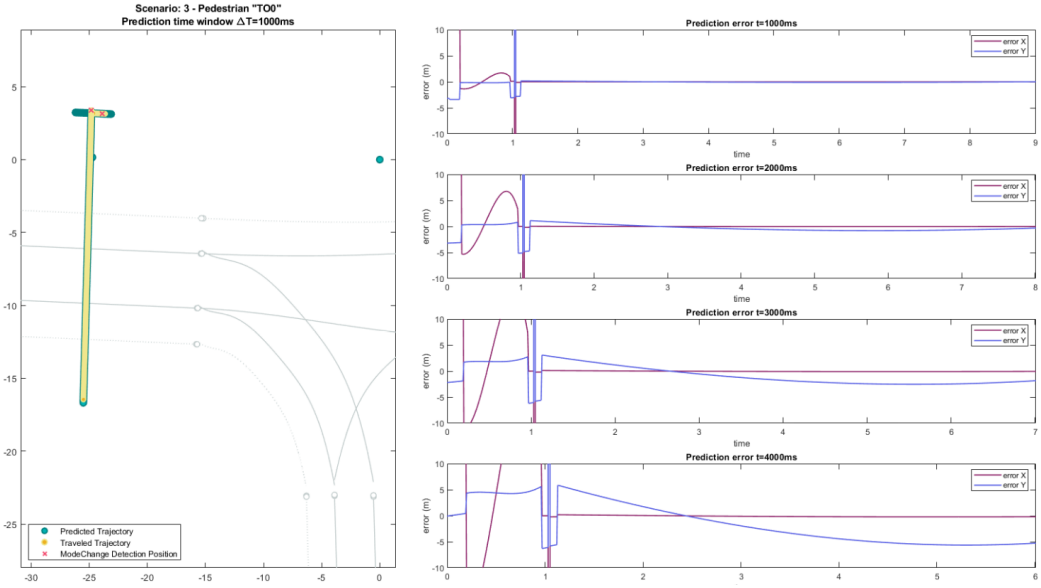


Figure 6.20: **Scenario03, TO0**: Prediction error graph. The error decrease significantly as soon as the pedestrian stabilizes his movement.

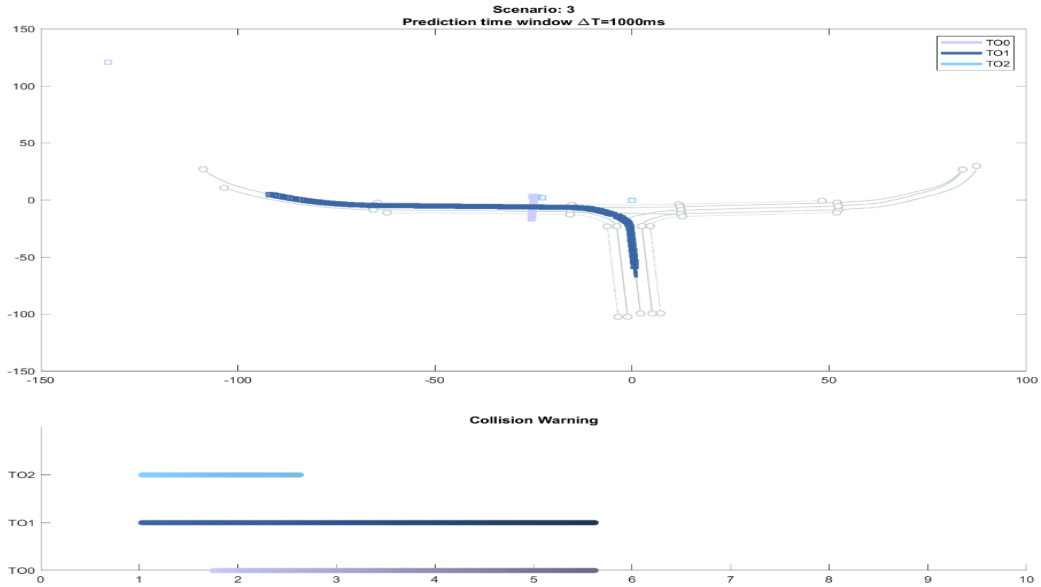


Figure 6.21: **Scenario03**: Collision prediction graph. The pedestrian *TO2* should be warned between 1.03s and 2.63s because of the approaching vehicle, while the impact between *TO0* and *TO1* is detected 3.9s ahead.

Figure 6.22: Scenario 4: vehicle(blue) hits the cyclist (white boundaries), during the turn left maneuver.

Mode Detection The Scenario04 helps to evaluate the mode probabilities and trajectory prediction for 2Wheelers VRU, such as skaters or cyclist. In this case, a cyclists performs two straight lane changes: the mode probabilities graph in fig. 6.23 shows that the cyclist is detected to change the lane 0.23s after the start of the simulation. This maneuver lasts 3s, instant when the IMM filter starts to decrease the `CL` model probability. At $t = 4.92s$ the curving mode replace the keep lane, until the end of the turn ($t = 7.93s$). The vehicle model probabilities switches between `KL` and `CURVE`, depending on the curvature of the road.

Trajectory Prediction The error on the prediction is graphed in fig. 6.24. With exception of the peak at $t = 1.81s$ due to the beginning of a new change lane, the error computed for $\Delta T = [1s, 2s, 3s]$ are almost negligible. For longer-term predictions, it reaches $7.7m$ before the second lane change and $4.4m$ at $t = 5s$. The generated path for `TO1` is more accurate, and does not exceed $0.65m$ during the simulation (fig. 6.25).

Collision Prediction The collision is foreseen starting at $t_{CP} = 2.86s$, $\delta t = 4.2s$ ahead, though the time horizon is set to 4s, due to the uncertainties on the position taken into account during the path generation of the cyclist and the vehicle (fig. 6.26).

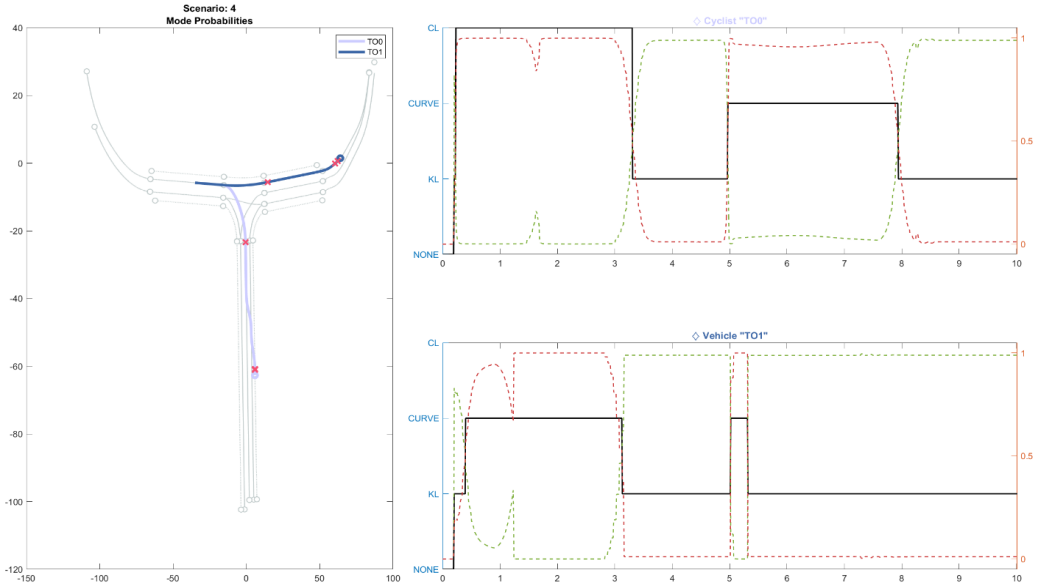


Figure 6.23: **Scenario04**: Mode probability graph. Cyclist is detected to change the lane after 0.23 from simulation start.

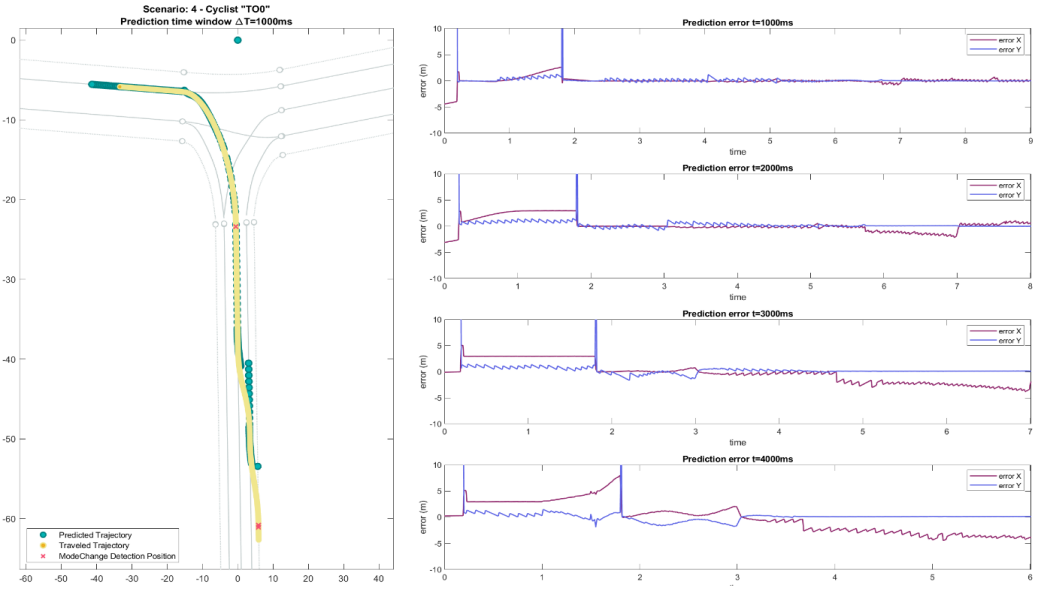


Figure 6.24: **Scenario04, TO0**: Prediction error graph. The error decrease significantly as soon as the pedestrian stabilizes his movement.

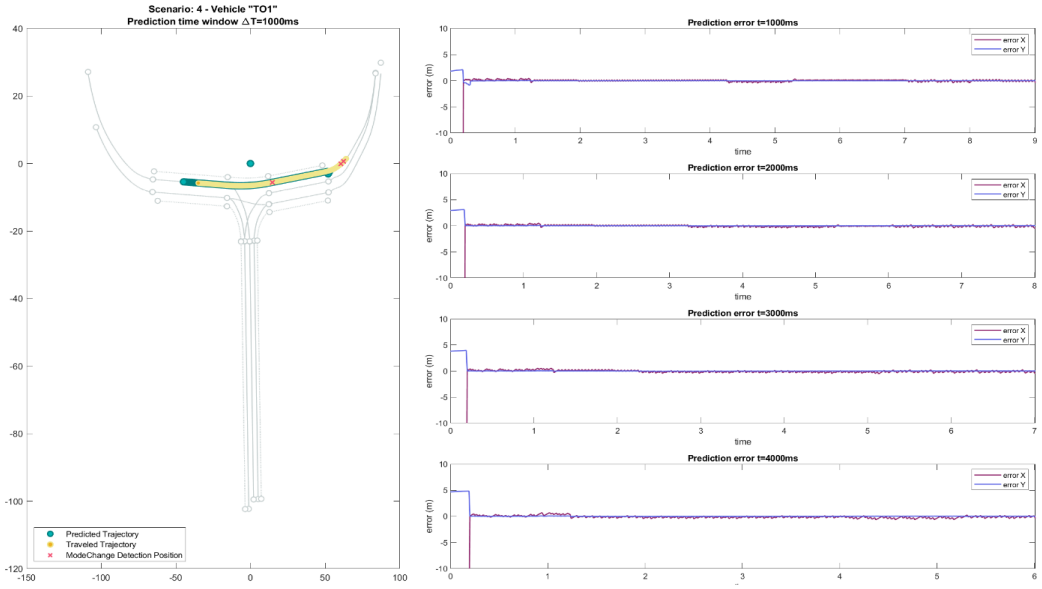


Figure 6.25: **Scenario04, TO1**: Prediction error graph. The error does not exceed 0.65m during the complete simulation.

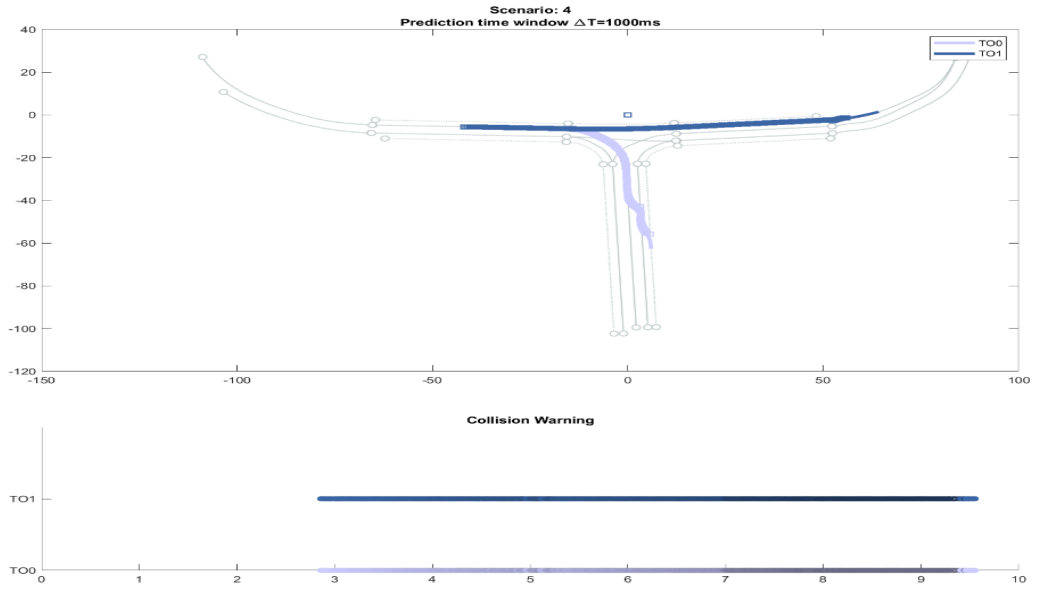


Figure 6.26: **Scenario04**: Collision prediction graph. The impact is predicted at $t = 2.86s$, 4.2s ahead.

Figure 6.27: Scenario 5: vehicle(blue) hits the airhead skater (white). The pedestrian is in red boundaries.

6.1.7 Scenario 05

A really airhead skater, ($v_{init} = 8m/s$) is hit in Scenario 05 (fig. 6.27). He decelerates at the curve, with a rate of $-0.3m/s^2$ for 6.5s. Once he has invaded the vehicle lane at 6.5s at a $v = 5m/s$, he continues on his path until the collision ($t = 8.8s$), due to the vehicle approaching from behind. The vehicle ($v_{init} = 9.75m/s$) accelerates for 3s reaching a speed $v = 12.75m/s$, and then tries to break (in vain), as soon as he realized that the skater was on its path. A male worker is on the roadside.

Mode Detection As in the previous scenario, a 2Wheeler VRU (a skater, in this case) populates the scene. A scheme of the driven path and the detected driving mode of the traffic participants is displayed in fig. 6.28. At the beginning ($t_{Curve|start} = 0.55s$) the turn maneuver is recognized, until $t_{Curve|end} = 3.85s$. The IMM filter detects a lane change at $t_{CL|start} = 6.58s$, 0.58s after the start of the maneuver, that lasts 1.5s. In this case the IMM filter keeps the `CL` probability higher, but the integration of the route geometry information, let the algorithm return a keep lane.

Trajectory Prediction The maximum error on the prediction (fig. 6.29) is for the skater $-2.4m$ for $\Delta T = 2s, 3s, 4s$, while for $\Delta T = 1s$ it does not exceeds $1.6m$ on the y axis. Such an inaccuracy in the prediction is

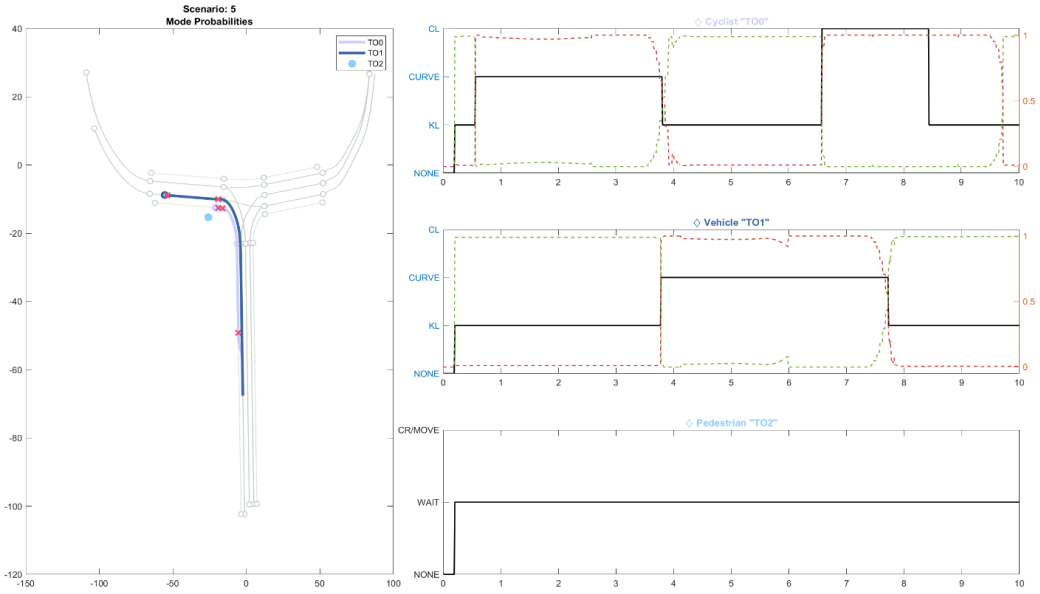


Figure 6.28: **Scenario05**: Mode probability graph. The lane change is firstly predicted at $t = 6.58s$, 0.58s after the beginning.

encountered during the change of the skater motion (at 6s he starts the maneuver).

Collision Prediction These errors, though, seem to be acceptable in reference to the collision prediction (fig. 6.30): as soon as the `CL` mode is detected ($t_{CL|start} = 6.58s$), the collision is predicted with a time horizon of $\Delta T = 2.3s$ (hence, $\delta t = 2.2s$ ahead). The male worker in the scene would be warned until $t = 1.86s$ because of the vehicle TO1.

6.1.8 Scenario 06

In Scenario 06 (fig. 6.31) 3 vehicles and two VRUs (a pedestrian and a cyclist) are defined:

- The pedestrian, TO0, an old woman (white boundaries), who survives to three impacts: the first one with the cyclist (at $t = 3.7s$), the second one with the grey vehicle (blue trajectory) at $t = 7.5s$, and finally at $t = 9.15s$ with the sky blue vehicle.

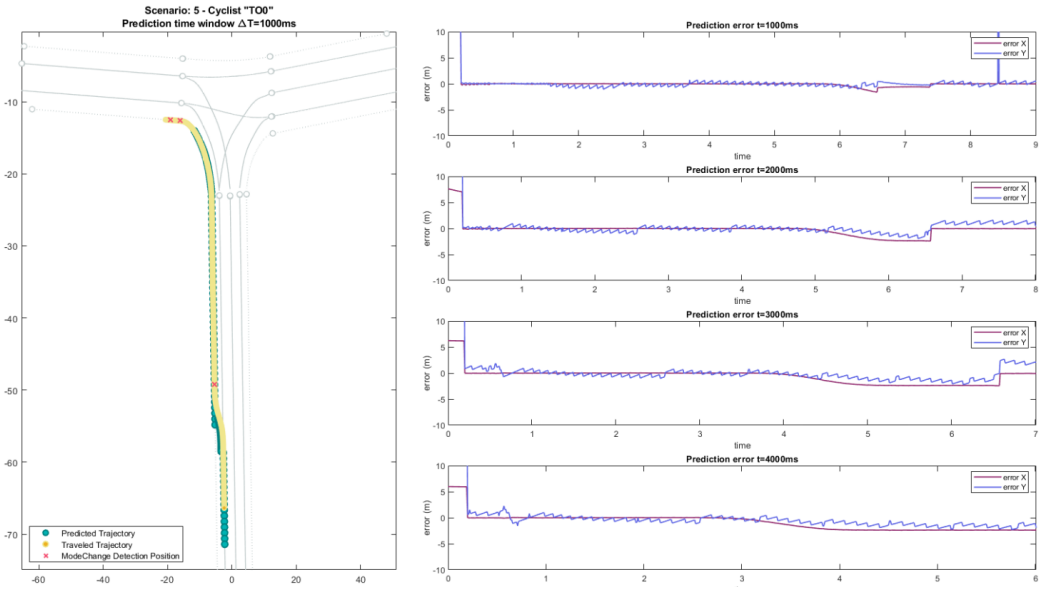


Figure 6.29: **Scenario05, TO0**: Prediction error graph. The error does not exceed 1.6m during the complete simulation.

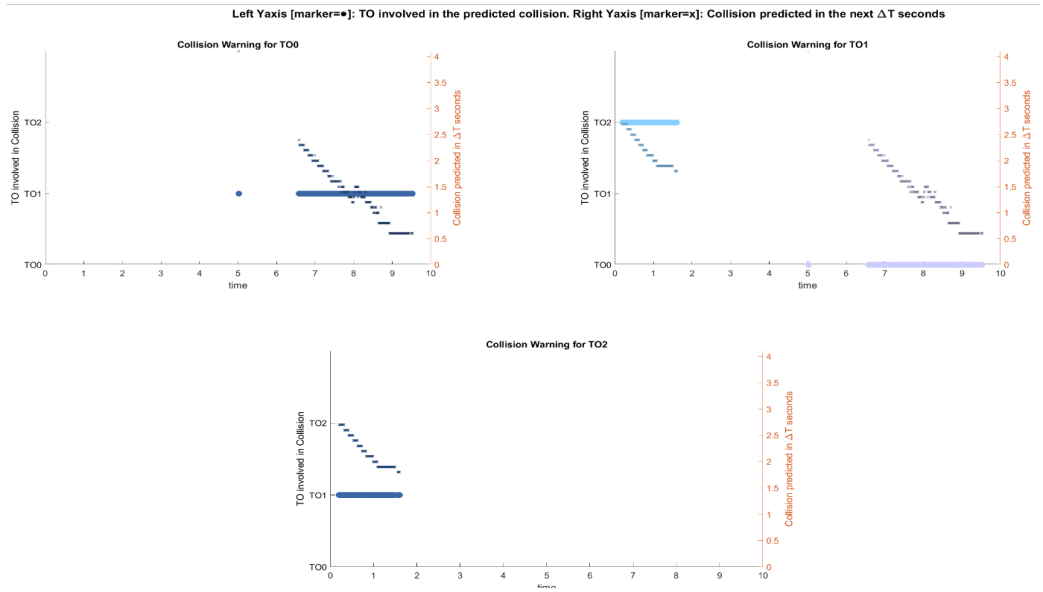


Figure 6.30: **Scenario05**: Collision prediction graph. The impact is predicted at $t = 6.58s$ as soon as the change lane is detected.

Figure 6.31: Scenario 6: the LandRover *TO1* (blue), after overtaking the black vehicle (green), turns right and hit the pedestrian (red). The last vehicle and the cyclist keep their lanes and hit the pedestrian respectively at $t = 3.7s$ and $t = 9.15s$.

- A grey vehicle, `TO1` (blue trajectory). It enters the intersection from the top, and approaching a second vehicle (TO3), changes its lane to turn left at the intersection ($v = 15m/s$).
- A blue vehicle, `TO2` (sky blue trajectory). keeps its route and speed of 10 m/s for all the simulation.
- A black vehicle, `TO3` (green trajectory). It enters the crossroad ant turns right at the intersection, keeping its velocity constant ($v = 10m/s$).
- The cyclist, `TO4` (yellow trajectory). His starting speed is $v_{start} = 8m/s$ and reaches during the simulation a speed of $v_{end} = 8m/s$.

Mode Detection A summary of the development of the scene is given in 6.32. In this graph it is interesting to notice, that the lane change of `TO1` is firstly identified at $t_{CL|start} = 1.52s$, when the vehicle has not yet cross the lane line.

Trajectory Prediction In the Appendix A A, all the prediction error graphs are provided.

Collision Prediction Through the analysis of the collision detection graph (fig. 6.33), the three different accidents, where the pedestrian is involved can be easily analysed:

- The first collision between the pedestrian `TO0` and the cyclist `TO4`, is firstly detected at $t_{CP} = 0.6s$ with a time horizon of $\Delta T = 3.7s$. Hence, it is predicted to occur at $t_P = 4.3s$, though it occurs at $t_C = 3.7s$: (inaccuracy due to the acceleration of the cyclist over the time).
- From $t_{CP} = 3.53s$ ($\Delta T = 4s$), another critical situation is detected for the pedestrian, because of `TO1`. $t_{CP} = 7.53s$ against the real time of collision $t_C = 7.5s$.
- At $t_{CP} = 5.47s$ `TO2` is predicted to collide with the pedestrian ($\Delta T = 3.7s$). $t_P = 9.17s$ against $t_C = 9.15s$.

A further critical situation is detected between `TO1` (vehicle approaching from behind, in blue) and `TO3` (green trajectory). Would the vehicle `TO1` keep the same lane and the same velocity, would impact `TO3` at ca. $t = 3s$. 0.03s after the lane change detection, this warning is disabled, as the trajectory is updated.

6.1.9 Scenario 07

The Scenario 07 (fig. 6.34) is more crowded than the previous, but no real collision occurs, though, vehicle should be warned because of too small safety distance. Here below a list of the traffic objects present in the scene:

- A male worker (white boundary): he just stands still at the roadside on the left.
- A mum with her baby (sky blue boundary): she does not move and waits to cross the road.
- A girl (blue boundaries): she moves along the roadside ($v_x = 2m/s$) and then stops for 6s. At $t = 8s$ she starts move perpendicular to the route.
- A grey Audi Q8 (green trajectory): after accelerating for the first 3s with a rate of $a = 2m/s^2$, it exits the crossroad at a velocity $v = 9m/s$.

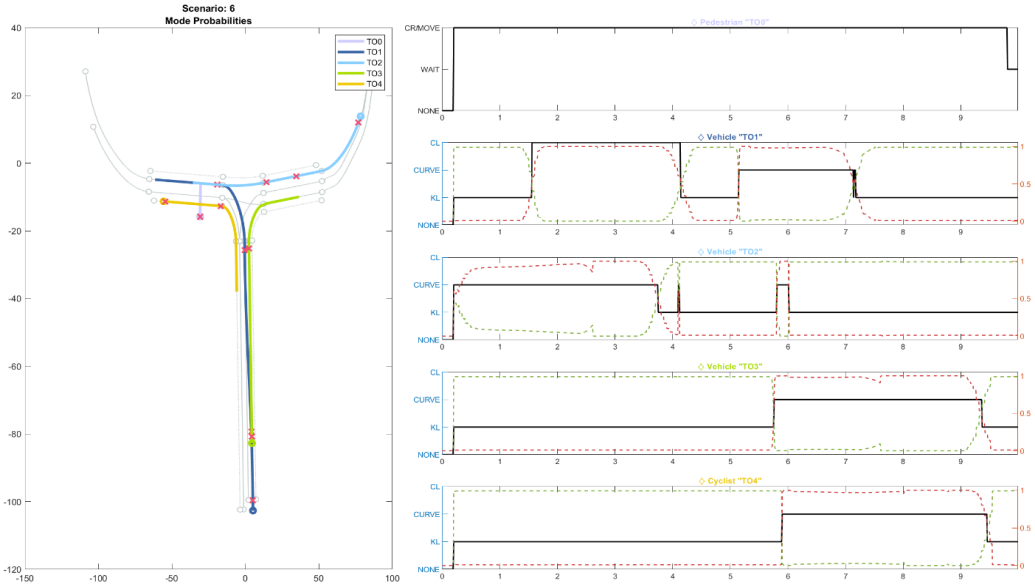


Figure 6.32: **Scenario06**: Mode probability graph. Summary of the scene development and the mode detected for all the traffic participants.

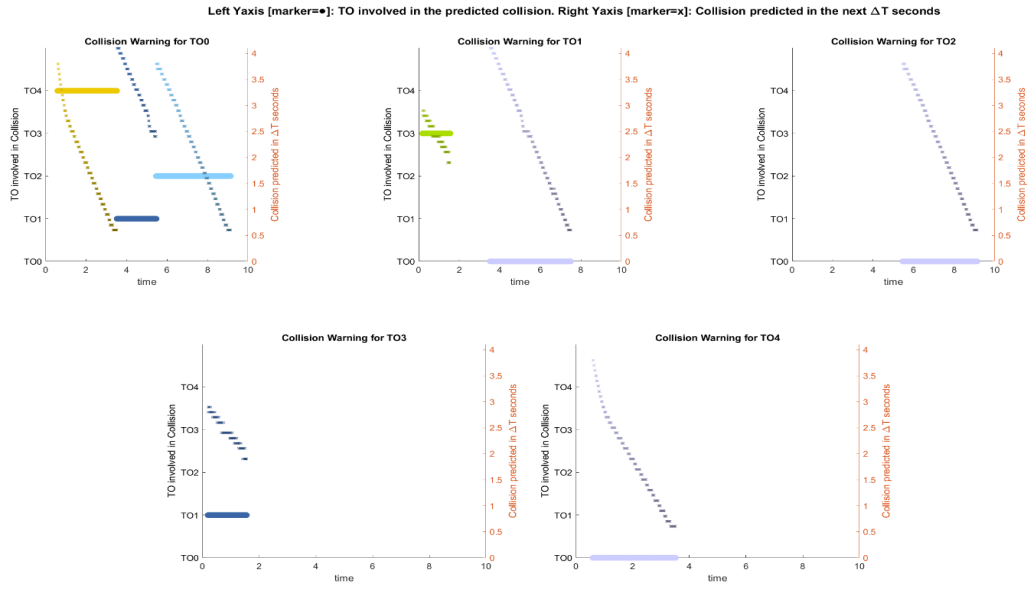


Figure 6.33: **Scenario06**: Collision prediction graph. A more detailed overview about the collision prediction between the different traffic objects.

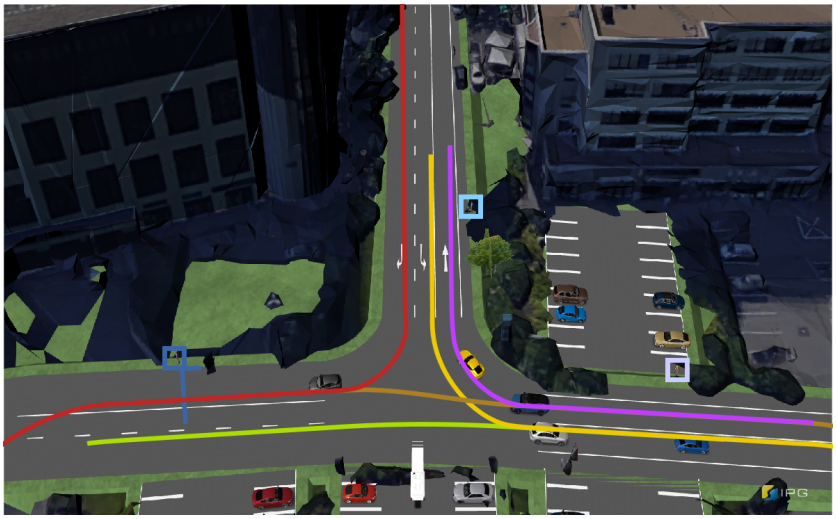


Figure 6.34: Scenario 7: scenario with no collisions, populated by vehicles and pedestrians

- A blue Audi S3 (yellow trajectory): it enters the business park and turns left, with an acceleration of $a = 2m/s^2$ between $[0s, 2s]$ and $[3.5, 5.5s]$. Afterwards it keeps its velocity.
- A blue NIO ES8 (brown trajectory): it simply proceeds with a constant velocity $v = 10m/s$ straight on.
- A black Renault Megane (red trajectory): it enters the crossroad, turning right, accelerating during the overall simulation reaching $v = 19m/s$.
- A yellow IPG CompanyCar (purple trajectory): it exits the business park, at a constant velocity of $v = 10m/s$.

Collision Prediction A summary of the development of the scene is given in 6.35. In this scenario, the focus in on the collision prediction, thus, only the graph in fig. 6.36 will be discussed, going through each traffic object.

- Pedestrian `TO0` (white/lila): the pedestrian is firstly warned because of `TO7`, until $t = 1.45s$: `TO7` is not yet passed by, but it does not represent a risk anymore, because, if the pedestrian would start to cross the road

at that instant, the trajectories of the two objects would not intersect at the same time. Soon after $t = 1.46s$ a second potential risk is identified because of TO5: in this case, too, the warning is shut down before the car is passed by ($t = 2.44s$). Almost at the same time, a potential threat is represented by TO4, but only with an high time horizon, due to the position uncertainties considered. This could be, hence, considered as false warning. As well as the signal raised for 0.11s due to the vehicle TO3.

- Pedestrian TO1 (blue): once she stops walking along the road, she is warned over a 3s time window ($t = [3s, 6s]$), that the vehicle TO6 could overrun her. The same for the vehicle TO5 for ($t = [6.24s, 8s]$). Hence, she starts moving towards the opposite parking place, and the warning is again enabled at $t = 9.3s$, though no collision occur.

- Pedestrian TO2 (sky blue): a potential critical situation is detected for the mum with her baby over the time window $t = [4.13s, 7.75s]$. The single scatter warning of TO5, is due to the not unique next segment of the route, where the vehicle is driving at the beginning: it could turn left or go straight on(please, refer to sec. 6.1.2 for a more detailed explanation).

- Vehicle TO3 (green): a collision is predicted with TO4 because of too short safety distance between the vehicles. The uncertainties in the position and the constant acceleration of TO3 for the first 3s bring to a collision prediction over the time span $t = [1.63s, 3s]$. The prediction with the pedestrian has already been discussed.

- Vehicle TO4 (yellow): the predictions referring to TO0 and TO3 have already been discussed.

- Vehicle TO5 (brown): the possibility of a collision with TO7 is due to the uncertainties and loss of long enough safety distance. They are, in fact, predicted for long time horizons and until $t = 5.91s$, when the correct route for the vehicle is selected.

- Vehicle TO6 (red): please, refer to the pedestrian TO1.

- Vehicle TO7 (purple): the potential impacts with TO0, TO5 and TO2 have been already detailed in the respective points.

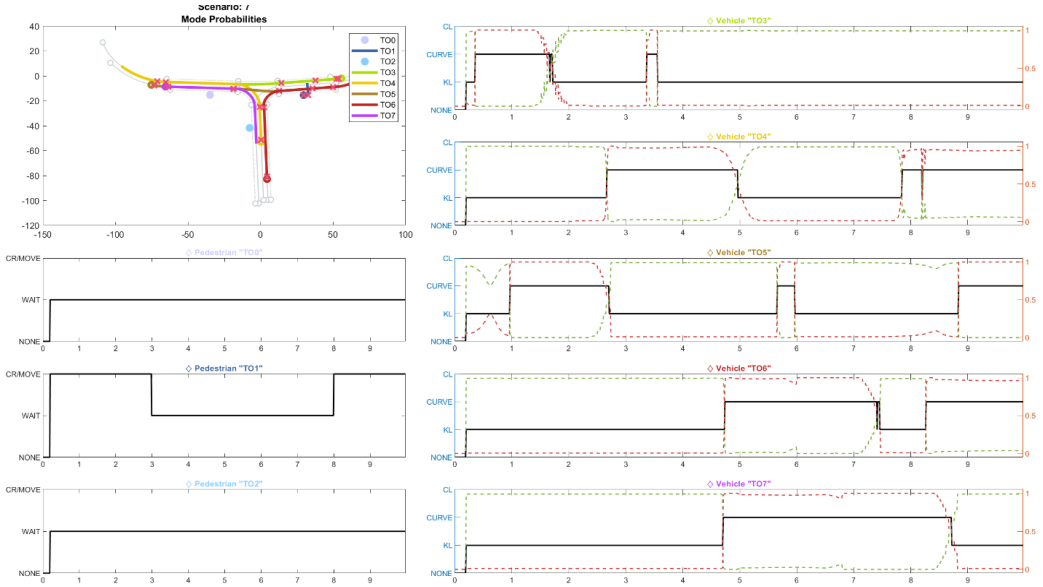


Figure 6.35: **Scenario07**: Mode probability graph. Summary of the scene development and the mode detected for all the traffic participants.

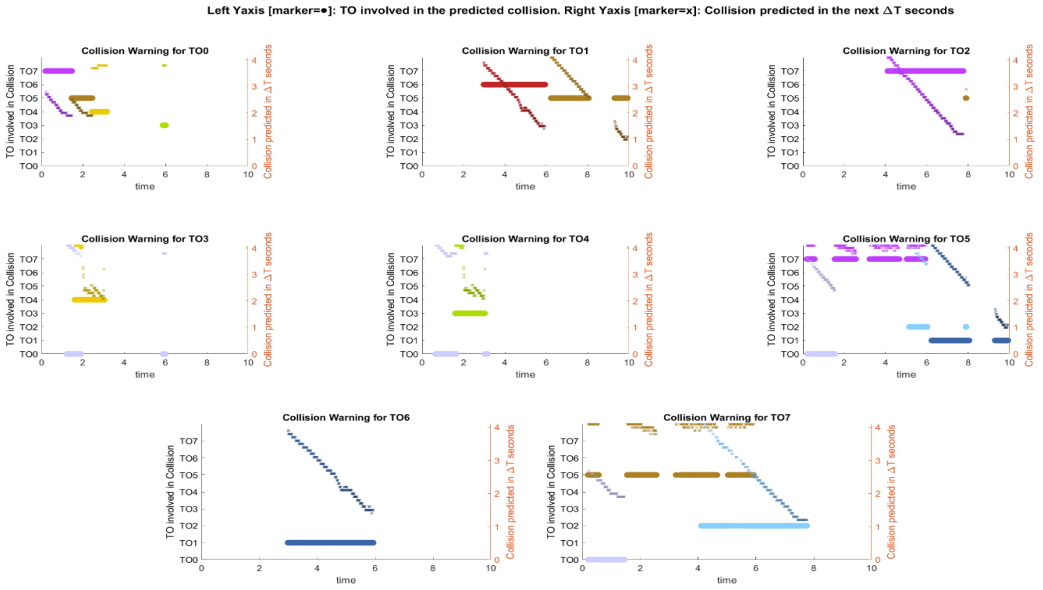


Figure 6.36: **Scenario07**: Collision prediction graph. A more detailed overview about the collision prediction between the different traffic objects.

6.1.10 Scenario 08

The last Scenario07 is intended to model some other particular cases, aiming to validate whole collision prediction system.

- A pedestrian (white/lila) stands for the first 3 second, and then crosses the road with a velocity $v = 2m/s$ for 8s. Hence, she walks along the roadside at $v = 1.2m/s$
- A Wheelchair pedestrian (blue) gets to the opposite site at $v = 4m/s$ between $t = 5.5s$ and $t = 9.5s$. Afterwards he proceeds slowly with a $v = 2m/s$.
- An E-Scooter rider (sky blue) exits the intersection, maintaining an acceleration rate for the first 5s $a = 0.5m/s^2$, and soon decelerating the next 5s at $a = -0.25m/s^2$.
- An grey Audi (green) changes its path the first 2.5s at a velocity of $v = 10m/s$. Then, he proceeds on its lane until it decides to get back to the previous route abruptly between $t = [3.5s, 5s]$, with an acceleration of $a = 2m/s^2$. Once it accelerates again for 1s, it keeps its constant velocity.
- A police auto (yellow) populates this scene: once it changes its lane between $t = 0.5$ and $3s$ entering the business park, it turns right at the intersection at $v = 9m/s$ and then decelerates at $t = [5.5s, 8s]$.
- A Mercedes Benz (brown) starts at a velocity of $v_{init} = 15m/s$. Due to the grey Audi, it shall decelerate ($a = -3.8m/s^2$) for 2.5s in order to avoid an accident. After it keeps its velocity for the next 2s, it accelerates again consistently ($a = 4m/s^2$) for 2s.
- A red Peugeot (red) exits the business park, decelerating at the curve ($a = 1m/s^2$) for 1s, and then accelerating again for the next 2s ($a = 2m/s^2$).
- A NAVYA shuttle bus (purple) restart its path after stopping at a bus station. It proceeds with a velocity of $v = 8.3m/s$ over the simulation.

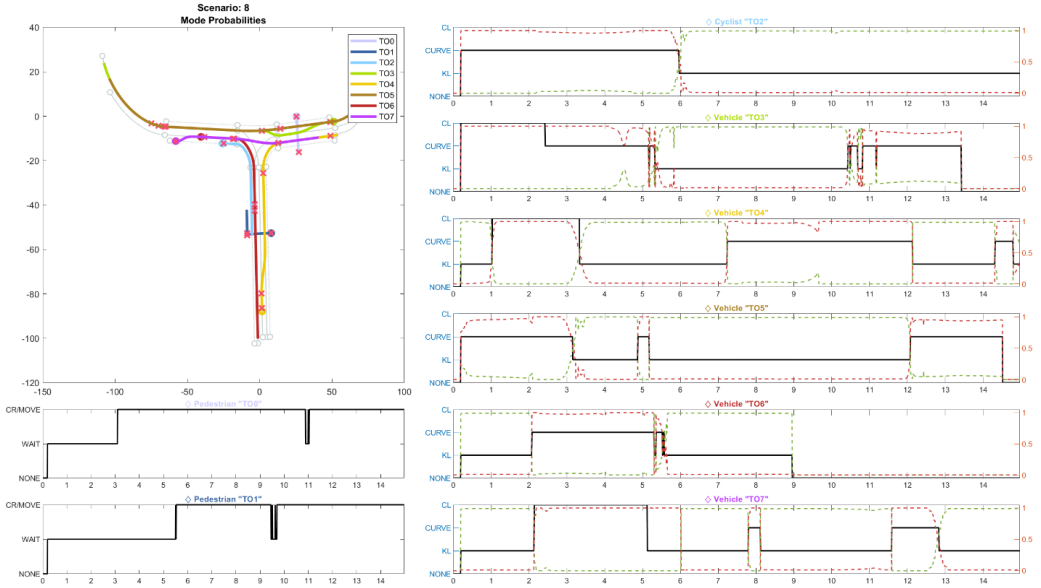


Figure 6.37: **Scenario08**: Mode probability graph. Summary of the scene development and the mode detected for all the traffic participants.

Mode Probabilities As in the previous scenario, the focus is on the collision prediction analysis. Though, it is necessary to have firstly an insight to the detected mode probabilities (fig. Fig. 6.37) for some remarks, especially related to the NAVYA shuttle (`TO7`) and the vehicle `TO3`, that abruptly changes its route. `TO3` maneuver is soon detected, until $t = 2.47s$ (end of the maneuver is set to $t = 2.5s$). During the transition time span $t = [3.5s, 5s]$, the vehicle is labeled as `CURVING`, as a real lane change in the road intersection would be not allowed. Though it could be seen as a system limitation, it must be kept in mind, that this pre-defined route mechanism relies on traffic rules. The prediction is hence updated as soon as the vehicle is *close enough* to the new route. The shuttle motion, that starts at 2s, is recognised as a change lane maneuver, that lasts according the the maneuver recognition module until $t = 5.13s$, though it is completed at $t = 4s$: in this time window, the error is huge, as a proper lane change route cannot be defined.

Collision Probabilities As for Scenario07, all detected potential critical situation will be now in details explained (fig. 6.39 and 6.40):

- Pedestrian `TO0` (white/lila): he is firstly warned because of `TO5`, until

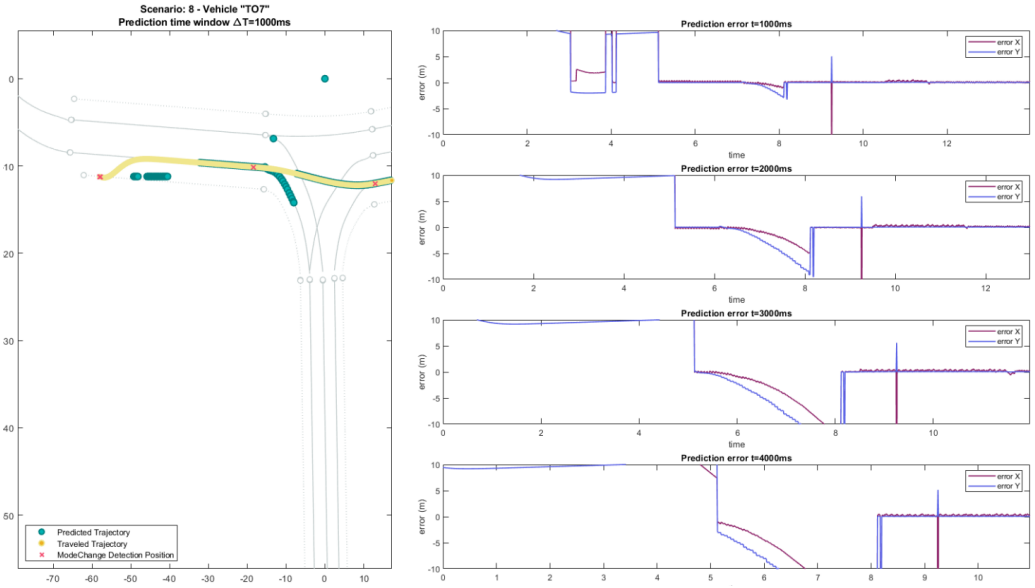


Figure 6.38: **Scenario08, TO7**: Trajectory prediction error graph for the shuttle. The error is huge as no proper lane change route is found.

it is enough far away, such that the trajectories of the objects would not intersect anymore at the same time if the pedestrian would start at the instant to cross the road ($t = 2.69s$). At the same time, a warning is shortly raised due to `TO3`. Then, as soon as the route of the shuttle is correctly updated, a collision warning is registered between `TO7` and `TO0` ($t = 8.71$).

- Wheelchair VRU `TO1` (blue): the collision warning (`TO4`) terminates at $t = 1.99s$. The E-Scooter represents no continuous real threat.
- The E-Scooter `TO2` (sky blue): the collision with `TO1` is predicted at $t = 5.62s$ because of the integrated uncertainties. A longer warning is raised between $t = 8.38s$ and $t = 8.83s$, when the E-Scooter is decelerating.
- The Vehicle `TO3` (green) is subjected to a warning for a potential crush with the red vehicle `TO6`, until it does not change its path at $t = 2.5s$. Afterwards a not long enough safety distance is the reason behind the predicted collision with `TO5`.

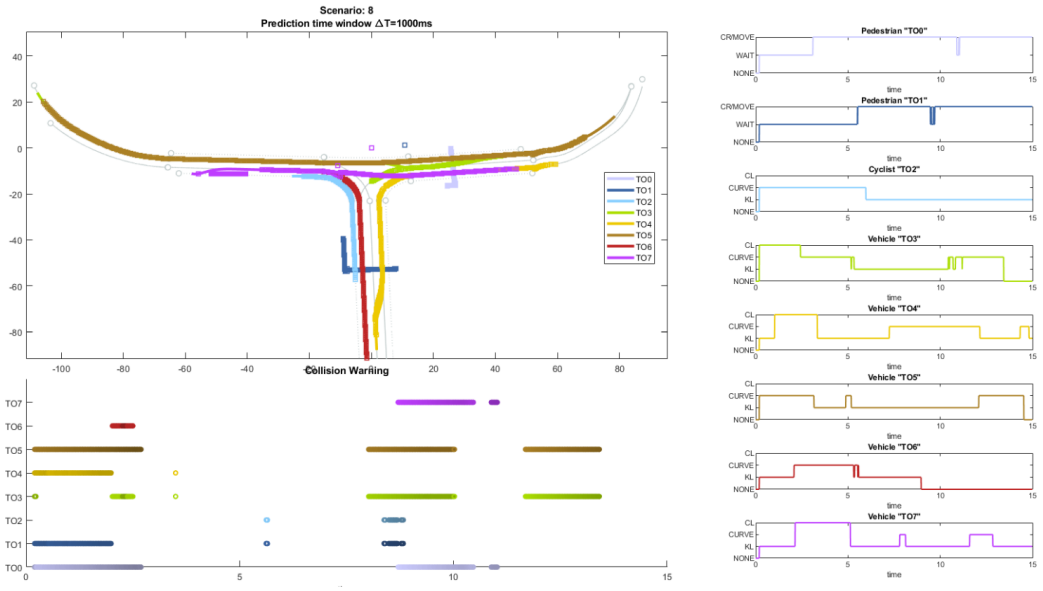


Figure 6.39: **Scenario08**: Collision prediction graph.

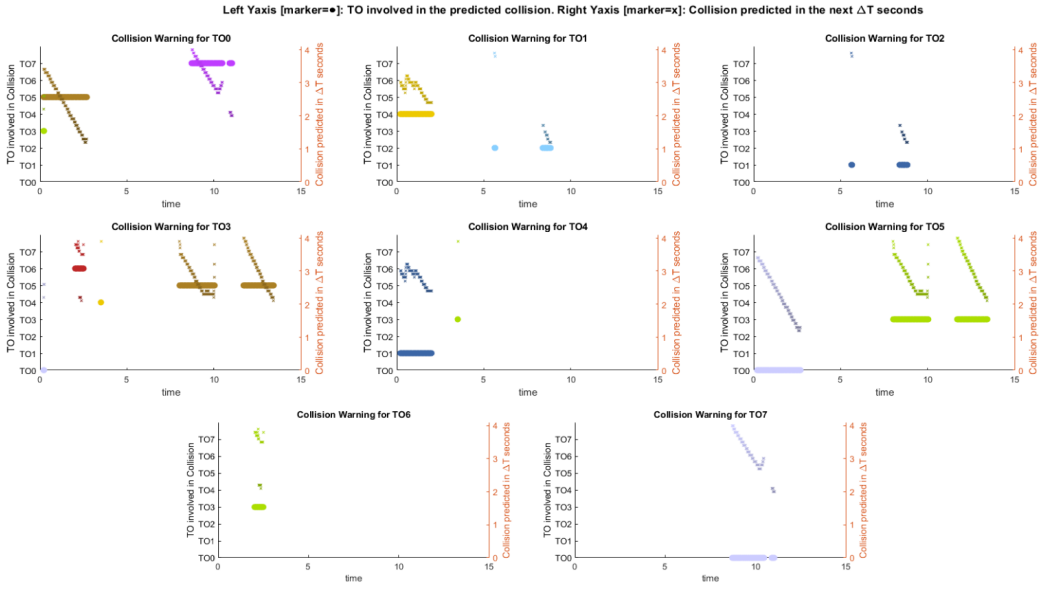


Figure 6.40: **Scenario08**: Collision prediction graph. A more detailed overview about the collision prediction between the different traffic objects.

scenario	TO	$v(m/s)$	t_{start}	$t_{D\|start}$	t_{end}	$t_{D\|end}$
02	01	10	0.00s	1.51s	5.00s	4.50s
04	00	8	0.00s	0.23s	5.00s	4.92s
05	00	5	6.50s	6.58s	8.00s	8.44s
06	01	10	0.00s	1.51s	5.00s	4.50s
08	04	10	0.00s	0.23s	2.50s	2.42s
08	07	8	2.00s	2.12s	5.00s	5.12s

Table 6.1: Summary table for a more clear overview about the delay in the lane change detection

6.2 Discussion

In this section, the results obtained in this work are presented in a summary form, as a means of inferring conclusions.

6.2.1 Lane Change Detection

The results achieved by the maneuver motion module regarding the lane change maneuvers is shown in table 6.1. The greatest delay in the recognition is given by the vehicle maneuver over 5s: as already illustrated in fig. 6.15, the vehicle has though not yet crossed the lane line. Results achieved for vehicles and VRU are, otherwise, almost the same, and the delay in the detection is in the range of 0.08s - 0.23s.

6.2.2 Trajectory Prediction

The results inherent to the future path generation can be collected and summed up in the following points:

- The trajectory prediction for pedestrian, who are not moving abruptly, is acceptable for the time horizons $\Delta T = 1s, 2s, 3s$. The more the ΔT increases, the more consistent is the error. For $\Delta T = 4s$ the prediction is subjected to huge inaccuracies (reaching 12/15m).
- The adopted quadratic piece-wise Bezier Curve have demonstrated to generate accurate path.

scenario	TO	TO	t_P	$t_C P$	ΔT
02	Ped 00	Veh 01	7.2s	3.23s	4.0s
03	Ped 00	Veh 01	5.7s	1.78s	3.9s
04	Cyc 00	Veh 01	7.0s	2.86s	4.2s
05	Ska 00	Veh 01	8.8s	6.58s	2.2s
06	Ped 00	Cyc 01	3.7s	0.60s	3.1s
06	Ped 00	Veh 01	7.5s	3.53s	3.9s
06	Ped 00	Veh 02	9.15s	5.47s	3.7s

Table 6.2: Summary of the collision time and collision prediction time for all the scenario, where impacts are configured.

- The trajectory prediction for non-pedestrian driving without suddenly acceleration and deceleration deliver accurate good results for all the time horizons, both for KeepLane and ChangeLane. In case of accelerations/decelerations, the prediction is affected by important errors (15/20), especially for time horizons close to $\Delta T = 4s$.
- The trajectory prediction in case of a route forks, can be generated along the wrong following road segment (Scenario00, i.e.).
- The shuttle `TO7` in the Scenario 07 opens the door to future developments, to allow correct trajectory predictions for traffic objects that stops at bus stops, or just overtake a preceding vehicle.

6.2.3 Collision Prediction

A concise overview for the collision prediction in case of impacts is provided in table 6.2.

The Scenario04 and Scenario05 describe the fundamental role, that the lane change detection module can have in savings time for a collision prediction and notification to the VRU: the collisions are detected before the traffic object is on the target route.

During the collision prediction analysis of more complicated scenarios such as Scenario06 and Scenario07, many warnings have been raised, because of the safety distance. A good compromise, in this case, would be to avoid prediction for long time horizon (i.e. $t = [3.5s, 4s]$), or to consider more appropriate uncertainties in the traffic object future poses.

Chapter 7

Conclusion and future work

The main contribution of this work is to introduce a real-time method to compute the collision risk for VRU (and vehicles), taking advantage of an *a priori* knowledge of the road geometry and pre-defined routes. The maneuver recognition module (IMM filter integrated with map information) has been demonstrated to deliver consistent and early lane change detection, that save time in the collision prediction and, thus, in VRU risk notification. The adopted piecewise quadratic Bezier Curve, whose computational load had proved in the past to be light, can accurate approximate a lane change maneuver path. The number of generated trajectories for each vehicle is exactly one, reducing drastically the number of trajectory in other studies.

As this study aims to introduce a new collision system based on trajectory generated using pre-defined routes, future works are foreseen to refine and improve the whole module. First of all, maneuvers like overtaken or restart from the roadside are not foreseen and the predictions are currently not reliable in these cases. Position uncertainties are integrated but only on the longitudinal direction of the motion: no lateral offsets are considered, though in real-time scenarios is high likely that some vehicles take a curve slighter than others, for example. A further improvement shall be made within the map definition and trajectory generation context, for forking roads: a generation of more than one possible trajectory in this case seems to be the most reliable solution. An alternative approach could be the definition of probability based on real collected data, that describes how likely a specific segment will be chosen over the other.

In the future, this work will be tested on real scenarios within the "People Mover" project.

Appendix A

Scenario06: Prediction Errors

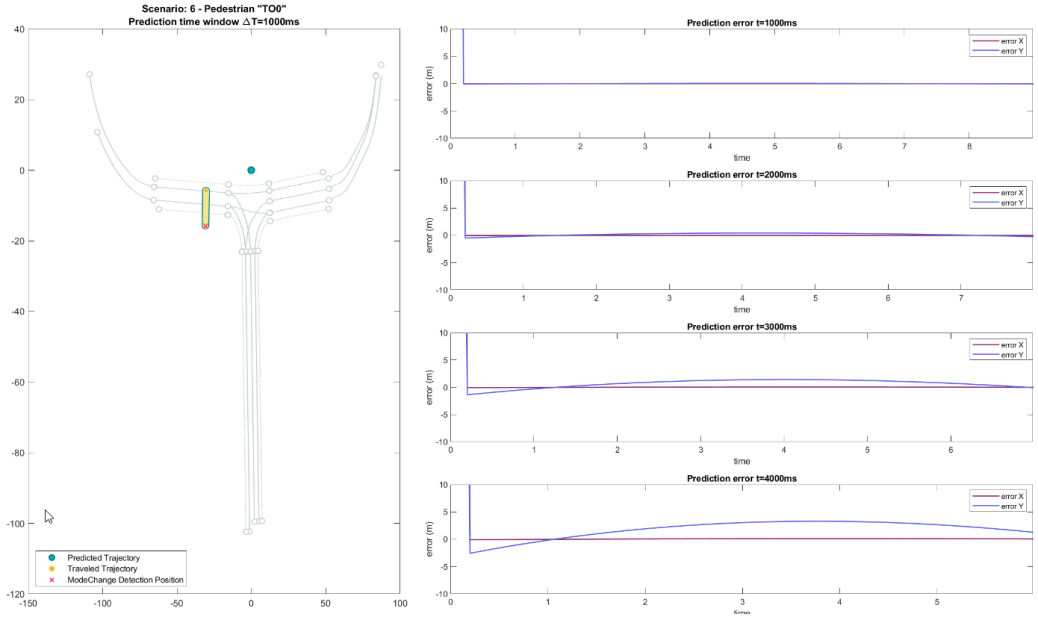


Figure A.1: **Scenario00, TO0**: Trajectory prediction error graph.

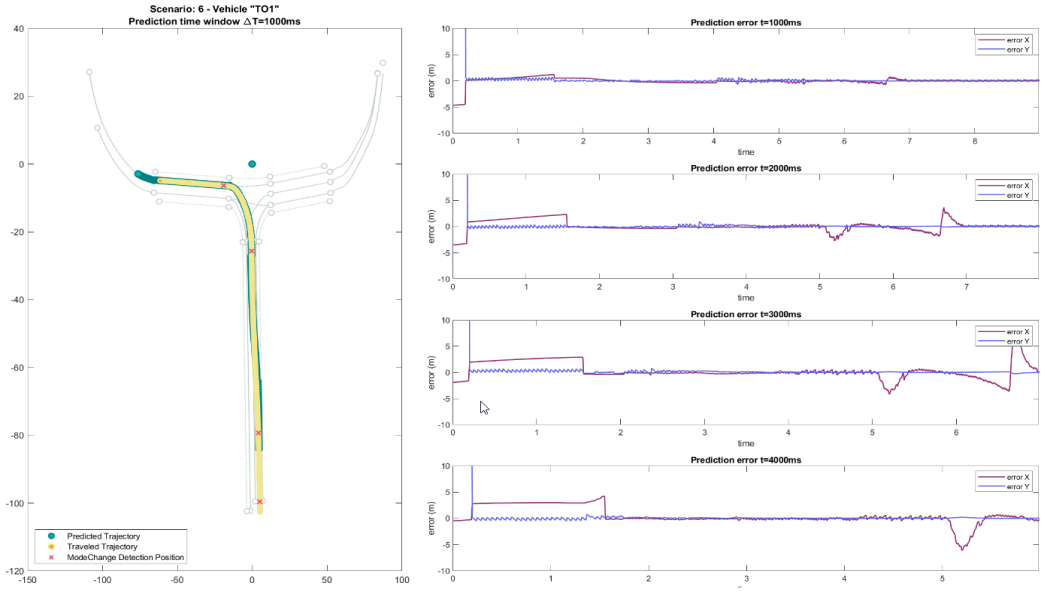


Figure A.2: **Scenario00, TO1**: Trajectory prediction error graph.

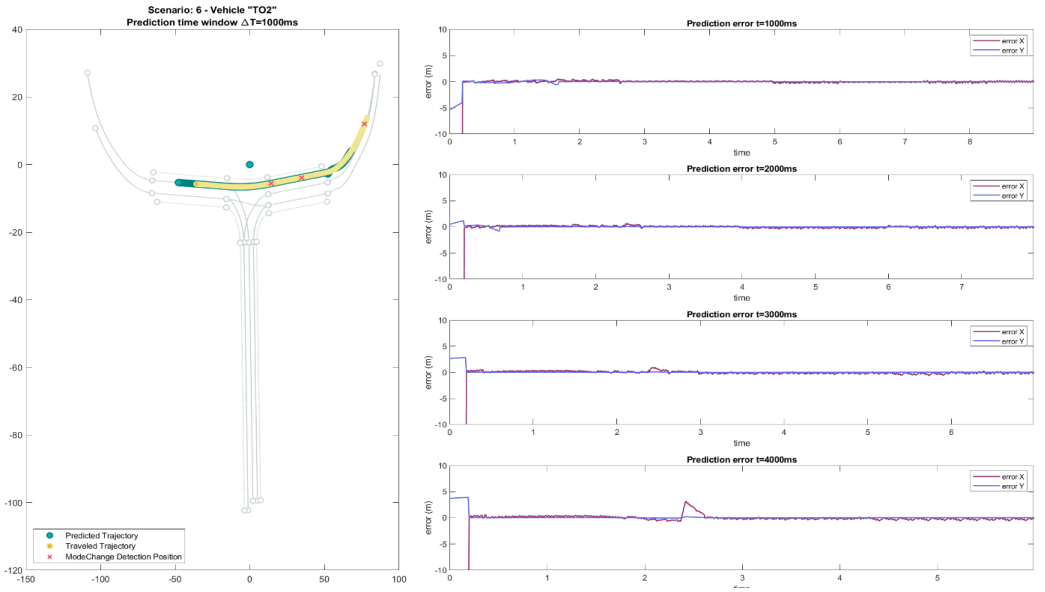


Figure A.3: **Scenario00, TO2**: Trajectory prediction error graph.

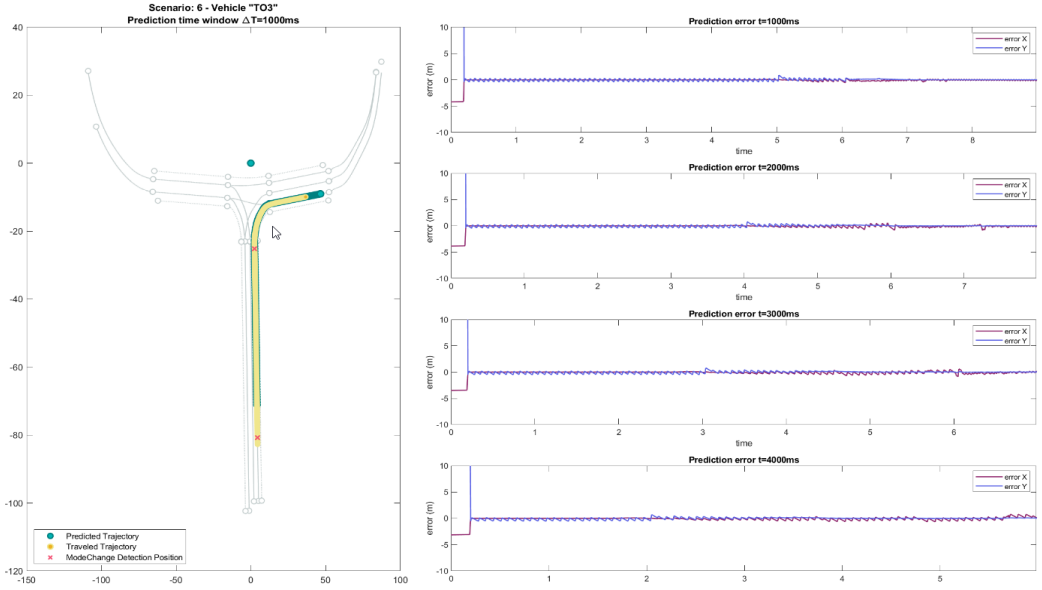


Figure A.4: **Scenario00, TO3**: Trajectory prediction error graph.

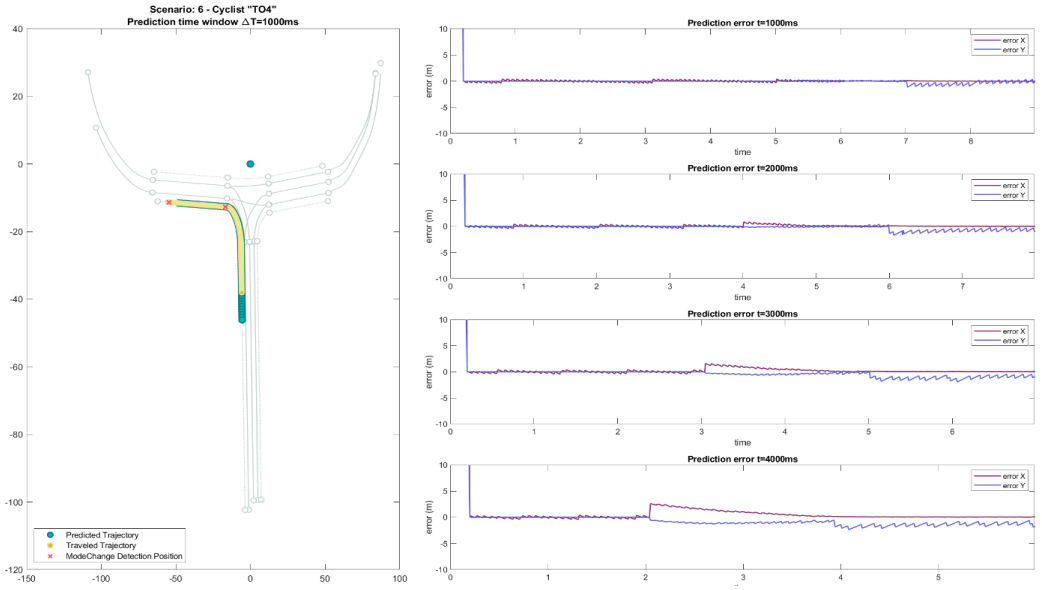


Figure A.5: **Scenario00, TO4**: Trajectory prediction error graph.

Bibliography

[1] Florent Altché and Arnaud de La Fortelle. An lstm network for highway trajectory prediction. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 353–359, 2017.

[2] Samer Ammoun and Fawzi Nashashibi. Real time trajectory prediction for collision risk estimation between vehicles. In *2009 IEEE 5th International Conference on Intelligent Computer Communication and Processing*, pages 417–422, 2009.

[3] Georges S. Aoude, Brandon D. Luders, Kenneth K. H. Lee, Daniel S. Levine, and Jonathan P. How. Threat assessment design for driver assistance system at intersections. In *13th International IEEE Conference on Intelligent Transportation Systems*, pages 1855–1862, 2010.

[4] H. Blom. An efficient decision-making-free filter for processes with abrupt changes. *IFAC Proceedings Volumes*, 18:631–636, 1985.

[5] M. Brand, N. Oliver, and A. Pentland. Coupled hidden markov models for complex action recognition. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 994–999, 1997.

[6] Mattias Brännström, Erik Coelingh, and Jonas Sjöberg. Model-based threat assessment for avoiding arbitrary vehicle collisions. *IEEE Transactions on Intelligent Transportation Systems*, 11(3):658–669, 2010.

[7] Jiajia Chen, Pan Zhao, Tao Mei, and Huawei Liang. Lane change path planning based on piecewise bezier curve for autonomous vehicle. In *Proceedings of 2013 IEEE International Conference on Vehicular Electronics and Safety*, pages 17–22, 2013.

[8] Hongwei Ding, Hao Wu, Lan Dong, and Zejun Li. Vehicle intersection collision monitoring algorithm based on vanets and uncertain trajectories. In *2018 16th International Conference on Intelligent Transportation Systems Telecommunications (ITST)*, pages 1–7, 2018.

[9] Liyu Gong and Qiang Cheng. Exploiting edge features in graph neural networks, 2019.

[10] Daniel Greene, Juan Liu, Jim Reich, Yukio Hirokawa, Akio Shinagawa, Hayuru Ito, and Tatsuo Mikami. An efficient computational architecture for a collision early-warning system for vehicles, pedestrians, and bicyclists. *IEEE Transactions on Intelligent Transportation Systems*, 12(4):942–953, 2011.

[11] Jia Hou, George F. List, and Xiucheng Guo. New algorithms for computing the time-to-collision in freeway traffic simulation models. *Computational Intelligence and Neuroscience*, 2014:761047, Dec 2014.

[12] Adam Houenou, Philippe Bonnifait, Véronique Cherfaoui, and Wen Yao. Vehicle trajectory prediction based on motion model and maneuver recognition. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4363–4369, 2013.

[13] Hyeong-Seok Jeon, Dong-Suk Kum, and Woo-Yeol Jeong. Traffic scene prediction via deep learning: Introduction of multi-channel occupancy grid map as a scene representation. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1496–1501, 2018.

[14] Hyeongseok Jeon, Junwon Choi, and Dongsuk Kum. Scale-net: Scalable vehicle trajectory prediction network under random number of interacting vehicles via edge-enhanced graph convolutional neural network. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2095–2102, 2020.

[15] N. Kaempchen, K. Weiss, M. Schaefer, and K.C.J. Dietmayer. Imm object tracking for high dynamic driving maneuvers. pages 825–830, 2004.

[16] ByeoungDo Kim, Chang Mook Kang, Jaekyum Kim, Seung Hi Lee, Chung Choo Chung, and Jun Won Choi. Probabilistic vehicle trajectory

prediction over occupancy grid map via recurrent neural network. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 399–404, 2017.

[17] Jae-Hwan Kim and Dong-Suk Kum. Threat prediction algorithm based on local path candidates and surrounding vehicle trajectory predictions for automated driving vehicles. In *2015 IEEE Intelligent Vehicles Symposium (IV)*, pages 1220–1225, 2015.

[18] P.B. Kruchten. The 4+1 view model of architecture. *IEEE Software*, 12(6):42–50, 1995.

[19] Donghan Lee, Youngwook Paul Kwon, Sara McMains, and J. Karl Hedrick. Convolution neural network-based lane change intention prediction of surrounding vehicles for acc. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–6, 2017.

[20] Stephanie Lefevre, Dizan Vasquez, and Christian Laugier. A survey on motion prediction and risk assessment for intelligent vehicles. *Robomech Journal*, 1, 07 2014.

[21] Jie Li, Tao Li, Yujie Zhang, Junping Xiang, and Dalin Xu. A comparative study on lane-changing decision model using deep learning methods. In *2019 Chinese Automation Congress (CAC)*, pages 1519–1523, 2019.

[22] E. Mazor, A. Averbuch, Y. Bar-Shalom, and J. Dayan. Interacting multiple model methods in target tracking: a survey. *IEEE Transactions on Aerospace and Electronic Systems*, 34(1):103–123, 1998.

[23] R. Miller and Qingfeng Huang. An adaptive peer-to-peer collision warning system. In *Vehicular Technology Conference. IEEE 55th Vehicular Technology Conference. VTC Spring 2002 (Cat. No.02CH37367)*, volume 1, pages 317–321 vol.1, 2002.

[24] World Health Organization. Global status report on road safety 2018. *Global status report on road safety*, 2018.

[25] Jiacheng Pan, Hongyi Sun, Kecheng Xu, Yifei Jiang, Xiangquan Xiao, Jiangtao Hu, and Jinghao Miao. Lane-attention: Predicting vehicles' moving trajectories by learning their attention over lanes. In *2020*

IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 7949–7956, 2020.

[26] Sajan Patel, Brent Griffin, Kristofer Kusano, and Jason Corso. Predicting future lane changes of other highway vehicles using rnn-based deep models. 01 2018.

[27] Aris Polychronopoulos, Manolis Tsogas, Angelos J. Amditis, and Luisa Andreone. Sensor fusion for predicting vehicles' path for collision avoidance systems. *IEEE Transactions on Intelligent Transportation Systems*, 8(3):549–562, 2007.

[28] Jannik Quehl, Haohao Hu, Sascha Wirges, and Martin Lauer. An approach to vehicle trajectory prediction using automatically generated traffic maps. 2018.

[29] Drakoulis Richardos, Bolovinou Anastasia, Drainakis Georgios, and Amditis Angelos. Vehicle maneuver-based long-term trajectory prediction at intersection crossings. In *2020 IEEE 3rd Connected and Automated Vehicles Symposium (CAVS)*, pages 1–6, 2020.

[30] Robin Schubert, Eric Richter, and Gerd Wanielik. Comparison and evaluation of advanced motion models for vehicle tracking. In *2008 11th International Conference on Information Fusion*, pages 1–6, 2008.

[31] Iliyana Simeonova and Tzvetan Semerdjiev. Specific features of imm tracking filter design. *An International Journal of Information and Security*, 9, 01 2002.

[32] Rafael Toledo-Moreo and Miguel A. Zamora-Izquierdo. Imm-based lane-change prediction in highways with low-cost gps/ins. *IEEE Transactions on Intelligent Transportation Systems*, 10(1):180–185, 2009.

[33] Shaobo Wang, Pan Zhao, Biao Yu, Weixin Huang, and Huawei Liang. Vehicle trajectory prediction by knowledge-driven lstm network in urban environments. *Journal of Advanced Transportation*, 2020:8894060, Nov 2020.

[34] Guotao Xie, Hongbo Gao, Lijun Qian, Bin Huang, Keqiang Li, and Jianqiang Wang. Vehicle trajectory prediction by integrating physics- and

maneuver-based approaches using interactive multiple models. *IEEE Transactions on Industrial Electronics*, 65(7):5999–6008, 2018.

[35] Alex Zyner, Stewart Worrall, James Ward, and Eduardo Nebot. Long short term memory for driver intent prediction. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 1484–1489, 2017.