Universidad de Alcalá

PhD. Program in Electronics: Advanced Electronic Systems. Intelligent Systems

Predictive Techniques for Scene Understanding by using Deep Learning in Autonomous Driving

PhD. Thesis Presented by

Carlos Gómez Huélamo

2023

Universidad de Alcalá

PhD. Program in Electronics: Advanced Electronic Systems. Intelligent Systems

Predictive Techniques for Scene Understanding by using Deep Learning in Autonomous Driving

PhD. Thesis Presented by

Carlos Gómez Huélamo

Advisors

Dr. Luis Miguel Bergasa Pascual
Dr. Rafael Barea Navarro

Alcalá de Henares, TBD

A mi Madre, allá donde esté ......

"En este vasto mundo
navegáis en pos de un sueño,
surcando el ancho mar
que se extiende frente a vosotros.
El puerto de destino es el mañana
cada día más incierto.
Encontrad el camino,
cumplid vuestros sueños,
estáis todos en el mismo barco
y vuestra bandera es la libertad"
Opening 3 de One Piece
Autor: The Babystars

Acknowledgements

Esta Tesis Doctoral supone el culmen a cuatro años (Abril 2019 - Abril 2023) realmente duros, cargado de emociones, triunfos, pandemias, estafas y tropiezos, todo a partes iguales. Este es probablemente (aunque como diría Sean Connery interpretando a James Bond en 1983, *Never Say Never Again*) mi último gran documento individual, académicamente hablando.

Durante mi etapa universitaria (2013 hasta el momento, 2023) he tenido ciertos momentos puntuales en los que he sentido un salto cualitativo como profesional: El primero fue en el segundo cuatrimestre de segundo de carrera, cuando las cosas se pusieron tensas con Control II e Informática Industrial. Vaya sudores. El segundo probablemente fue con el fallecimiento de mi madre durante mi ERASMUS+ en Irlanda. Duros y oscuros momentos, alejado de mis seres queridos. El tercer momento llega en segundo de máster, durante mi querido ERASMUS+ en Finlandia, donde compagino una estancia preciosa en Tampere con el máster y un pre-inicio de doctorado. Me equivoqué al empezar tan pronto con la beca, "queriendo cobrar" cuanto antes, en vez de terminar tranquilamente el TFM y plantear la tesis, pero eso no lo sabría hasta tiempo después. Pero no es hasta la tesis donde empezaron los quebraderos de cabeza reales. Continuamente altibajos, mala planificación por mi parte, momentos puntuales donde me equivoqué rotundamente al empecinarme en soldar una estructura compleja para nuestro vehículo sin ayuda, no estudiar PyTorch tras el congreso WAF 2018 tras la sugerencia de mi tutor, no enfocarme en técnica individual hasta bien entrado el doctorado, no querer hacer nada hasta que no tuviese la teoría perfectamente asimilada, tener demasiado respeto a la Inteligencia Artificial y escurrir el bulto de mi tesis en un compañero mientras yo me dedicaba a integrar y corregir los bugs del grupo que para mí *era lo fácil*. Mal. Todo mal. Pero todo cambió tras mi segunda estancia, en Estados Unidos, cuando tras llorar por no entender el camino a seguir, nadie que me ayudara, decidí crear mi propio camino, con paciencia y fé, práctica y error compaginado con lectura de artículos, para mejorar mi confianza y autoestima, y finalmente logré empezar a entender lo que era el Deep Learning. Gracias a todos mis errores, desventuras y discusiones, a día de hoy, excepto momentos inevitables, me encuentro con muchísima capacidad para atacar y gestionar prácticamente cualquier problema, consultar documentación y organizarme, aunque esta sigue siendo mi tarea pendiente.

Cada año, desde hace ya varios, mi primera publicación en Instagram viene seguida de la frase "Trabaja duro en silencio y deja que tu éxito haga todo el ruido". Filosofía Kaizen, de mejora y aprendizaje continuo, para así cada día entender el mundo un poquito mejor. Si toda la dedicación y estudio que he depositado en este trabajo sirven para algo en mi futuro, sé que todo el esfuerzo habrá merecido la pena.

Después de este particular monólogo, a lo cual soy muy propenso y de lo cual mis amigos y compañeros no cesan en su empeño de recordádmelo, debo, como no puede ser de otra manera, dar paso a los agradecimientos.

En primer lugar, me gustaría agradecer a mis profesores del grupo RobeSafe, especialmente a mis tutores Luis Miguel Bergasa Pascual y Rafael Barea Navarro, por ofrecerme estar en el grupo (así como aguantarme) durante todos estos años e intentar que tuviésemos el mejor *experimental setup* y *roadmap* en el laboratorio, aunque no fuese siempre sencillo. Sin dudas considero realmente interesante la temática propuesta en esta tesis doctoral, predicción de agentes en el contexto de vehículo inteligente, ya que entra en el plano filosófico sobre cómo razonar el futuro de los objetos y cómo podría afectar a la capacidad ejecutiva del agente que deba tomar una decisión. Mi mente hace tiempo que cambió y me fijo siempre que conduzco de todo lo que intento reproducir con mis estudios. Habrá que seguir esta tendencia muy de cerca en los próximos años, porque personalmente considero que sus aplicaciones son fantásticas.

A mis tutores en las estancias de doctorado, Christoph Stiller y Eduardo Molinos en el Karslruhe Institute of Technology (KIT, Alemania) y Wei Zhan y Masayoshi Tomizuka en la University of California, Berkeley (UCB, Estados Unidos). Se suele decir que unas veces se gana y otras se aprende, y yo en estas estancias quizás aprendí demasiado ... No obstante, me guardo grandísimos momentos (admirar las secuoyas gigantes o hacer mi primera escalada en roca entre ellos) y amigos, como Su Shaoshu o Frank Bieder, con los que aún guardo un cierto contacto.

A mis compañeros, mejor dicho, amigos, de laboratorio: Javier Araluce, Rodrigo, Felipe (chavalín), Santiago, Miguel Antunes, Miguel Eduardo, antiguos compañeros como Javier del Egido, Óscar, Alejandro, Eduardo, Roberto y Pablo Alcantarilla, y a nuevos becarios como Fabio, Navil y Pablo. Gracias de corazón por estar ahí, en nuestras charlas sobre tecnología, empleos, el camino correcto a seguir y la vida en general.

Especial mención vuelvo a hacer a Miguel Eduardo y mi compañero Marcos Conde, cuya apoyo intenso ayudó a centrar mi camino en técnica y escritura. Muchísimas gracias por todo lo que aprendí a vuestro lado. Especial mención a mi querido profesor Ángel Álvarez, por sus valiosos consejos para afrontar mi carrera profesional y mi vida en general de la mejor forma posible. Eres muy grande.

A mis amigos de la universidad, especialmente a Rocío, Juan Carlos, Esther, Sergio, Pablo, Rubén y Adrián Rocandio. Aún me acuerdo de cuando empezamos con la carrera y como el paso inexorable del tiempo nos moldea a conveniencia. Os deseo lo mejor en vuestro futuro.

A mis buenos amigos Samuel y Adrián, con quien gran parte de mi vida he compartido. Con especial cariño guardo las interminables charlas sobre la vida y el futuro después de Karate, de comer, de cenar, en el coche, siempre quejándonos de la hora que marcaba el reloj al final de tan interminables conversaciones.

A mi familia, uno de los pilares de mi vida. A mi padre Juan Antonio y a mi madre Petra, que en paz descanse, les debo todo lo que soy y es por ello por lo que les estaré siempre agradecido. Querido padre, gracias por ser tan Genaro, arisco y pesado. Siempre has sido mi ejemplo a seguir, aunque cuando tenga 42 años me sigas regañando por subir con las zapatillas puestas. Querida madre, no se muere quien se va, sólo se muere el que se olvida, y tú nunca caerás en el olvido. A mi querida hermana-calili-chessmaster Silvia, con quien tantas regañinas he tenido, pero el cariño que nos tenemos las supera a todas. A mi perrita Nuka (a.k.a. Dragón, ChupaChups cuando le cortamos el pelo o Nuki-Nuki), cuyos paseos matutinos son probablemente el ingrediente secreto para la elaboración de esta tesis, dando rienda suelta a mi cabeza para imaginar nuevas propuestas mientras miraba el cielo azul. A mi querido *experimental setup* que me ha acompañado durante toda mi vida académica: silla de madera de la cocina, ratón de Hello Kitty tomado prestado de mi hermana y mi querido portátil táctil. Sois la base de todo mi trabajo.

Al resto de la familia, amigos, compañeros, entrenadores y profesores, gracias por todo.

Y, por último, la persona más importante de mi vida ahora mismo. Mi querida Marta, la persona más maravillosa y buena que conozco. Hemos compartido risas, lloros, besos y abrazos. Nunca me cansaré de repetirte lo suave que tienes la piel tras darte un beso en la mejilla y después hacerte de rabiar. Espero que esta situación esté dentro de un while cuya condición sea *True*.

"Te quiero más que ayer, pero menos que mañana. Hoy, y siempre"

Mi querido lector, disculpa mi monólogo de agradecimientos, es mi forma de ser y la cual tengo por bandera, aunque creo que ha quedado bonito. Podría decir mil anécdotas más de mi doctorado, pero como diría Aragorn, legítimo Rey de Gondor, enfrente de la mismísima Puerta Negra: *Hoy no es ese día.*

Vamos a la lectura importante, que empiece el *Rock and Roll* !!

Resumen

La conducción autónoma es considerada como una de los más grandes retos tecnológicos actuales. Cuando los coches autónomos conquisten nuestras carreteras, los accidentes se reducirán notablemente, hasta casi desaparecer, ya que la tecnología estará testada y no incumplirá las normas de conducción, entre otros beneficios sociales y económicos.

Uno de los aspectos más críticos a la hora de desarrollar un vehículo autónomo es percibir y entender la escena que le rodea. Esta tarea debe ser tan precisa y eficiente como sea posible para posteriormente predecir el futuro de esta misma y ayudar a la toma de decisiones. De esta forma, las acciones tomadas por el vehículo garantizarán tanto la seguridad del vehículo en sí mismo y sus ocupantes, como la de los obstáculos circundantes, tales como viandantes, otros vehículos o infraestructura de la carretera.

En ese sentido, esta tesis doctoral se centra en el estudio y desarrollo de distintas técnicas predictivas para el entendimiento de la escena en el contexto de la conducción autónoma. Durante la tesis, se observa una incorporación progresiva de técnicas de aprendizaje profundo en los distintos algoritmos propuestos para mejorar el razonamiento sobre qué está ocurriendo en el escenario de tráfico así como modelar las complejas interacciones entre la información social (distintos participantes o agentes del escenario, tales como vehículos, ciclistas o peatones) y física (es decir, la información geométrica, semántica y topológica del mapa de alta definición) presente en la escena.

La capa de percepción de un vehículo autónomo se divide modularmente en tres etapas: Detección, Monitorización y Predicción. Para iniciar el estudio de las etapas de monitorización y predicción, se propone un algoritmo de *Multi-Object Tracking* basado en técnicas clásicas de estimación de movimiento y asociación validado en el dataset KITTI, el cual tiene métricas del estado del arte. Por otra parte, se propone el uso de un filtro inteligente basado en información contextual de mapa, cuyo objetivo es monitorizar los agentes más relevantes de la escena en el tiempo, representando estos agentes filtrados la entrada preliminar para realizar predicciones unimodales basadas en un modelo cinemático. Para validar esta propuesta de filtro inteligente se usa CARLA, uno de los simuladores hiperrealistas para conducción autónoma más prometedores en la actualidad, comprobando cómo al usar información contextual de mapa se puede reducir notablemente el tiempo de inferencia de *tracking* y predicción prestando atención a los agentes realmente relevantes del escenario de tráfico.

Tras observar las limitaciones de un modelo de predicción basado en cinemática para la predicción a largo plazo de un agente, los distintos algoritmos de la tesis se centran en el módulo de predicción, usando los datasets Argoverse 1 y Argoverse 2, donde se asume que los agentes proporcionados en cada escenario de tráfico ya están monitorizados durante un cierto número de observaciones.

En primer lugar, se introduce un modelo basado en redes neuronales recurrentes (particularmente redes LSTM, *Long-Short Term Memory*) y mecanismo de atención para codificar las trayectorias pasadas de los agentes, y una representación simplificada del mapa en forma de posiciones finales potenciales en la carretera para calcular las trayectorias futuras unimodales, todo envuelto en un marco GAN (*Generative Adversarial Network*), obteniendo métricas similares al estado del arte en el caso unimodal.

Una vez validado el modelo anterior en Argoverse 1, se proponen distintos modelos (sólo social, incorporando mapa, y una mejora final basada en *Transformer encoder* y redes convolucionales 1D) base precisos y eficientes basados en el modelo de predicción anterior, introduciendo el uso de las redes gráficas (particularmente GCN, *Graph Convolutional Network*) para codificar de una forma potente las interacciones de los agentes y el preprocesamiento de trayectorias preliminares a partir de mapa con un método heurístico, para posteriormente predecir distintas trayectorias futuras en este caso multimodales, es decir, cubriendo distintos posibles futuros para el agente de interés. Los modelos base propuestos obtienen métricas de regresión del estado del arte tanto en el caso multimodal como unimodal manteniendo un claro compromiso de eficiencia con respecto a otras propuestas.

El modelo final de la tesis, inspirado en los modelos anteriores y validado en el más reciente dataset para algoritmos de predicción en conducción autónoma (Argoverse 2), introduciendo la información topológica y semántica de los carriles futuros preliminares con el método heurístico antes mencionado, codificación de mapa basada en aprendizaje profundo con redes GCN, ciclo de fusión de características físicas y sociales, estimación de posiciones finales en la carretera con aprendizaje profundo y agregación de su entorno circundante, y finalmente módulo de refinado para mejorar la calidad de las predicciones multimodales finales de un modo elegante y eficiente. Comparado con el estado del arte, nuestro método logra métricas de predicción a la par con los métodos mejor posicionados en el Leaderboard de Argoverse 2, reduciendo de forma notable el número de parámetros y operaciones de coma flotante por segundo.

Finalmente, este modelo final es validado en simulación en distintas aplicaciones de conducción autónoma, como la toma de decisiones en el simulador SMARTS o el estudio de adaptación de dominio en el simulador hiperrealista CARLA con otras capas del vehículo como paso preliminar a su implementación en un vehículo autónomo real.

Palabras clave: Conducción Autónoma, Predicción de Movimiento, Aprendizaje Profundo, Entendimiento de la Escena, Eficiencia, Simulación.

Abstract

Autonomous driving is considered one of the greatest technological challenges today. When autonomous cars take over our roads, accidents will be significantly reduced, almost disappearing, as the technology will be tested and will not violate driving regulations, among other social and economic benefits.

One of the most critical aspects in developing an autonomous vehicle is perceiving and understanding the surrounding scene. This task must be as accurate and efficient as possible to predict the future of the scene and assist in decision-making. In this way, the actions taken by the vehicle will ensure the safety of the vehicle itself and its occupants, as well as that of surrounding obstacles such as pedestrians, other vehicles, or road infrastructure.

In this context, this doctoral thesis focuses on the study and development of different predictive techniques for scene understanding in the context of autonomous driving. Throughout the thesis, there is a progressive incorporation of Deep Learning techniques into the proposed algorithms to improve reasoning about what is happening in the traffic scenario and model the complex interactions between social information (different participants or agents in the scene, such as vehicles, cyclists, or pedestrians) and physical information (*i.e.* geometric, semantic, and topological information from the high-definition map) present in the scene.

The perception layer of an autonomous vehicle is modularly divided into three stages: Detection, Monitoring, and Prediction. To start studying the monitoring and prediction stages, a Multi-Object Tracking algorithm based on classical motion estimation and association techniques is proposed and validated on the KITTI dataset, which has state-of-the-art metrics. Furthermore, the use of an intelligent filter based on contextual map information is proposed, aiming to monitor the most relevant agents in the scene over time. These filtered agents serve as the preliminary input for making uni-modal predictions based on a kinematic model. CARLA, one of the most promising hyper-realistic simulators for autonomous driving, is used to validate this intelligent filter proposal. It demonstrates how using contextual map information can significantly reduce the inference time for tracking and prediction by focusing on the truly relevant agents in the traffic scenario.

After observing the limitations of a kinematics-based prediction model for long-term agent prediction, the different algorithms in the thesis focus on the prediction module using the Argoverse 1 and Argoverse 2 datasets. These datasets assume that the agents provided in each traffic scenario have already been monitored for a certain number of observations.

Firstly, a model based on recurrent neural networks (particularly Long-Short Term Memory networks) and attention mechanism is introduced to encode the past trajectories of agents. It also uses a simplified representation of the map in the form of potential final positions on the road to calculate uni-modal future trajectories. These components are wrapped in a Generative Adversarial Network (GAN) framework, achieving metrics similar to the state-of-the-art in the uni-modal case.

Once the previous model is validated on Argoverse 1, different precise and efficient base models are proposed for the prediction module. These models incorporate social information, map information, and a final improvement based on a Transformer encoder and 1D-Convolutional Neural Networks. They introduce the use of graph networks (particularly Graph Convolutional Networks, GCNs) to encode the interactions between agents and preprocess preliminary trajectories from the map using a heuristic method. These models predict different multimodal future trajectories, covering various possible futures for the agent of interest. The proposed base models achieve state-of-the-art regression metrics in both multimodal and unimodal cases while maintaining a clear efficiency compromise compared to other proposals.

The final model of the thesis, inspired by the previous models and validated on the most recent dataset for autonomous driving prediction algorithms (Argoverse 2), incorporates topological and semantic information of the preliminary future lanes using the aforementioned heuristic method. It uses map encoding based on Deep Learning with Graph Convolutional Networks, a fusion cycle of physical and social features, Deep Learning aggregation of surrounding environment, and a refinement module to enhance the quality of the final multimodal predictions in an elegant and efficient manner. Compared to the state-of-the-art, our method achieves prediction metrics on par with the top-ranked methods in the Argoverse 2 Leaderboard while significantly reducing the number of parameters and floating-point operations per second.

Finally, this final model is validated in simulation in various autonomous driving applications, such as decision-making in the SMARTS simulator or domain adaptation study in the hyper-realistic CARLA simulator, involving other vehicle layers as a preliminary step towards its implementation in a real autonomous vehicle.

Keywords: Autonomous Driving, Motion Prediction, Deep Learning, Scene Understanding, Efficiency, Simulation.

Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

ACC	Adaptive Cruise Control.
AD	Autonomous Driving.
AD-PerDevKit	Autonomous Driving Perception Development Kit.
ADAM	ADaptive Moment Estimation.
ADAS	Advanced Driver Assistance System.
ADE	Average Displacement Error.
ADS	Autonomous Driving Stack.
AI	Artificial Intelligence.
BEV	Bird's Eye View.
CA	Constant Acceleration.
CARLA	CAr Learning to Act simulator.
CCA	Constant Curvature and Acceleration.
cGAN	conditional Generative Adversarial Network.
CMP	Conditional Motion Prediction.
CNN	Convolutional Neural Network.
CSAA	Constant Steering Angle and Acceleration.
CSAV	Constant Steering Angle and Velocity.
CTRA	Constant Turn Rate and Acceleration.
CTRV	Constant Turn Rate and Velocity.
CV	Constant Velocity.
DAMOT	Detection and Multiple-Object Tracking.
DL	Deep Learning.
DM	Decision-Making.
FDE	Final Displacement Error.
GAN	Generative Adversarial Network.

GCN	Graph Convolutional Network.
GNN	Graph Neural Network.
GRU	Gated Recurrent Unit.
GT	Ground-Truth.
HA	Hungarian Algorithm.
ITS	Intelligent Transportation Systems.
KF	Kalman Filter.
LiDAR	Light Detection And Range.
LSTM	Long Short-Term Memory.
MHSA	Multi-Head Self-Attention.
minADE	minimum Average Displacement Error.
minFDE	minimum Final Displacement Error.
ML	Machine Learning.
MLP	Multilayer Perceptron.
MOT	Multi-Object Tracking.
MP	Motion Prediction.
MSE	Mean Squared Error.
NHTSA	National Highway Traffic Safety Administration.
NLL	Negative Log Likelihood.
PMP	Passive Motion Prediction.
RL	Reinforcement Learning.
RNN	Recurrent Neural Network.
ROS	Robot Operating System.
RVIZ	ROS VIsualiZator.
SOTA	State-of-the-Art.
VRU	Vulnerable Road User.
WTA	Winner-Takes-All.

Chapter 1

Introduction

Aaay, el oro, la fama, el poder.
Todo lo tuvo el hombre que en su día se autoproclamó
el rey de los piratas, ¡GOLD ROGER!
Mas sus últimas palabras no fueron muy afortunadas:
"¿¡MI TESORO!? Lo dejé todo allí, buscadlo si queréis,
ojalá se le atragante al rufián que lo encuentre.

Opening 1 de One Piece: "We are"
Autor original: Hiroshi Kitadani

1.1. Motivation

Autonomous Driving (AD) have held the attention of technology enthusiasts and futurists for some time as is evidenced by the continuous research and development of this topic in Intelligent Transportation Systems (ITS) over the past decades, being one of the emerging technologies of the *Fourth Industrial Revolution*, and particularly of the Industry 4.0.

The concept *Fourth Industrial Revolution* or Industry 4.0 was first introduced by Klaus Schwab, CEO (Chief Executive Officer) of the World Economic Forum, in a 2015 article in Foreign Affairs (American magazine of international relations and United States foreign policy). A technological revolution can be defined as a period in which one or more technologies are replaced by other kinds of technologies in a short amount of time. Hence, it is an era of accelerated technological progress featured by Researching, Development and Innovation whose rapid application and diffusion cause an abrupt change in society. In particular, the *Fourth Industrial Revolution* conceptualizes rapid change to industries, technology, processes and societal patterns in the 21st century due to increasing interconnectivity and smart automation. This industrial revolution focuses on operational efficiency, being the following four themes which summarize it:

- **Decentralized decisions**: Ability of cyber physical systems to make decisions on their own and to perform their tasks as autonomously as possible.

- **Information transparency**: Provide operators with comprehensive information to make decisions. Inter-connectivity allows operators to gather large amounts of information and data from all points in the manufacturing process in order to identify key areas or aspects that can benefit from improvement to enhance functionality.
- **Technical assistance**: Ability to assist humans with unsafe or difficult tasks and technological facility of systems to help humans in problem-solving and Decision-Making (DM).
- **Interconnection**: Ability of machines, sensors, devices and people to communicate and conect with each other via the Internet of Things (IoT) or the Internet of People (IoP).

Based on the aforementioned principles, this revolution is expected to be marked by breakthroughs in emerging technologies in fields such as nanotechnology, quantum computing, 3D printing, Internet of Things (IoT), fifth-generation wireless technologies (5G), Robotics, Computer Vision (CV), Artificial Intelligence (AI) or the scope of this PhD thesis, Autonomous Driving Stacks (ADSs). The sum of all these advances are resulting in machines that can potentially see, hear and what is more important, think, moving more deftly than humans.

An ADS, driverless car or autonomous car, is a vehicle than can sense its surrounding and moving safely with little or even no human intervention. These ADSs must combine a variety of sensors to understand the traffic scenario, like RADAR (RAdio Detection A Ranging), Light Detection And Range (LiDAR), cameras, Inertial Measurement Unit (IMU), wheel odometry, GNSS (Global Navigation Satellite System) or ultrasonic sensors, in order to detect, track and predict (which is the main purpose of this thesis) the most relevant obstacles around the ego-vehicle. Then, advanced control and planning systems process this sensory information in combination with a predefined global route to calculate the corresponding control commands to drive the vehicle throughout the environment, ensuring a safe driving.

The dream of seeing fleets of ADSs efficiently delivering goods and people to their destination has fueled billions of dollars and captured consumer's imaginations in investment in recent years. Nevertheless, according to the "Autonomous driving's future: Convenient and connected" report, published by the global management consulting firm McKinsey & Company in January 2023, even after some setbacks have pushed out timelines for AD launches and delayed customer adoption, the transportation community still broadly agrees that AD has the potential to transform consumer behaviour, transportation and society at large. AD is considered as one of the solutions to the aforementioned problems and one of the greatest challenges of the automotive industry today.

The World Health Organization (WHO) has indicated that the global population is increasingly concentrated in cities, which has significant implications for public health.

The trend of urbanization has been observed for several decades, with people migrating from rural to urban centers in search of better opportunities and improved living conditions. As a result, cities are becoming more crowded, and the WHO predicts that by the year 2050, nearly 70 % of the world population will reside in urban areas.

This concentration of people in cities poses both challenges and opportunities for public health. On the positive side, cities offer access to better healthcare facilities, educational institutions, and employment opportunities. Urban areas also tend to have improved sanitation systems and infrastructure, which can contribute to better overall health outcomes. However, the rapid growth of cities can strain existing resources and lead to overcrowding, inadequate housing, and increased pollution levels, all of which can have adverse effects on public health.

Aware of this problem, the European Commision (EU) has set several objectives and statistics to improve transport mobility in the future. These objectives and statistics are part of the EU broader vision for a sustainable, efficient, and interconnected transport system to create a seamless, sustainable, and integrated transport system across member states:

- **Sustainable Transport**: The European Commission aims to promote sustainable modes of transport, such as railways, public transport, cycling, and walking, to reduce greenhouse gas emissions, congestion, and air pollution. The target is to achieve a 60 % reduction in transport emissions by 2050 compared to 1990 levels.
- **Infrastructure Investment**: The EU has committed to investing in modern and efficient transport infrastructure, including the Trans-European Transport Network (TEN-T) and the Connecting Europe Facility (CEF). These investments aim to improve connectivity, remove bottlenecks, and enhance multi-modal transport options across the EU.
- **Road Safety**: The European Commission has prioritized improving road safety to reduce the number of accidents, injuries, and fatalities. The objective was to reduce road fatalities by 50 % by 2030 compared to 2020 levels. This includes implementing measures such as stricter vehicle safety standards, promoting safe driving behaviors, and improving road infrastructure.
- **Digitalization and Innovation**: The EU aims to harness digital technologies and innovation to improve transport efficiency, reliability, and user experience. This includes initiatives such as ITSs, the use of big data for planning and optimization, and the development of Connected and Autonomous Vehicles (CAVs) to enhance mobility.
- **Freight Transport Efficiency**: The European Commission aims to improve the efficiency and sustainability of freight transport through measures such as promoting

inter-modal transport, increasing the use of clean and energy-efficient vehicles, and implementing logistics optimization strategies. The objective is to reduce external costs, including congestion, noise, and emissions, associated with freight transport.

In that sense, the existence of reliable and economically affordable ADSs are expected to create a huge impact on society affecting social, demographic, environmental and economic aspects. It can produce substantial value for the automotive industry, drivers and society, making driving safer, more convenient and more enjoyable. In other words, the hours on the road previously spent by manual driving could be used to work, watch a funny movie or even to video call a friend. For employees with long commutes, AD might shorten the workday, increasing worker productivity. Since workers, specially those related to digital jobs or related fields, may perform their jobs from an ADS, they could more easily move further away from the office, which, in turn, could attract more people to suburbs and rural areas. Besides this, it is estimated to cause a reduction in road deaths, reduce fuel consumption and harmful emission associated and improve traffic flow, as well as an improvement in the overall driver comfort and mobility in groups with impaired faculties, such as disable or elderly people, providing them with mobility options that go beyond car-sharing services or public transportation. Other industrial applications of autonomous vehicles are agriculture, retail, manufacturing, commercial and freight transport or mining.

1.2. Historical Context

ADSs have become a challenge for automation and technology companies, which has derived in an intense competition. Though today companies such as Mercedes, Ford or Tesla are racing to build ADSs for a radically changing consumer world, the research and development of autonomous robots is not new.

In 1500, centuries before the invention of the automobile, Leonardo da Vinci designed a cart that could move without being pulled or pushed. In 1868, Robert Whitehead invented a torpedo that could propel itself underwater in order to be a game-changer for naval fleets all over the world. In terms of robotic solutions for intelligent mobility, the study was started in the 1920s, being the concept of Autonomous Car defined in Futurama, an exhibit at the 1939 New York Wolrd's Fair. General Motors created the exhibit to display its vision of what the world would look like in 20 years, including an automated highway system that would guide ADS. By 1958, General Motors made this concept a reality (at least as a proof of concept) being the car's front end embedded with sensors to detect the current flowing through a wire embedded in the road. The first semi-automated car was developed in 1977 by Japan's Tsukuba Mechanical Engineering Laboratory. The vehicle reached speeds up to 30 km/h with the support of an elevated rail.

Figure 1.1: Stanley, 2005 DARPA Grand Challenge winner
Source: *Stanford university*

Nevertheless, the first truly autonomous cars appeared in the 1980s with Carnegie Mellon University's Navlab and ALV projects funded by the USA company DARPA (Defense Advanced Research Projects Agency) in 1984 and EUREKA Prometheus project (1987) developed by Mercedes-Benz and Bundeswehr University Munich's. By 1985, the ALV project had shown self-driving speeds on two-lane roads of 31 km/h with obstacle avoidance added in 1986 and off-road driving in day and night conditions by 1987. Furthermore, from the 1960s through the second DARPA Grand Challenge in 2005 (212 km off-road course near the California-Nevada state line, surpassed by all but one of the 23 finalists), automated vehicle research in the United States was primarily funded by DARPA, the US Army and US Navy, yielding rapid advances in terms of speed, car control, sensor systems and driving competence in more complex conditions. This caused a boost in the development of autonomous prototypes by companies and research organizations, most of them from the United States. Figure 1.1 shows Stanley, the 2005 DARPA Gran Challenge winner, from Stanford university.

Even though self-driving cars have not yet displaced conventional cars, there can be found several examples of how it has become a hot topic for powerful companies such as Delphi Automotive Systems, Audi, BMW, Tesla, Mercedes-Benz or Waymo. In 2005 Delphi broke the Navlab's record achievement (driving 4,584 km while remaining 98 % of the time autonomously) by piloting an Audi, improved with Delphi technology, over 5,472 km through 15 states while remaining in self-driving mode 99 % of the time. Moreover, in 2005 the USA states of Michigan, Virginia, California, Florida, Nevada and the capital, Washington D.C., allowed the testing of automated cars on public roads.

In 2017, Audi stated that its A8 car prototype would be automated at speeds up to 60 km/h by using its perception system named "Audi AI". Also, in 2017 Waymo (self-driving technology development company subsidiary of Alphabet Inc) started a limited trial of a self-driving taxi service in Phoenix, Arizona.

Figure 1.2 shows the total number of autonomous test miles and miles per disengagement in California (Dec 2019 - Nov 2020) by some of the most important AD technology development companies around the world. The concept disengagement is quite useful to assess the quality of an ADS, defined as the deactivation of the autonomous mode when a failure of the autonomous technology is detected or when a safe operation requires that the autonomous vehicle test driver disengages the autonomous mode, resulting in control being seized by the human driver. As observed, Waymo and Cruise were, by far, the companies with the highest number of miles per disengagement, indicating the superiority and stability of their ADS. Furthermore, we can conclude from Figure 1.2 that most AD companies are from the United States of America (specially from the state of California) and China, being United States clearly the most developed country in this field.

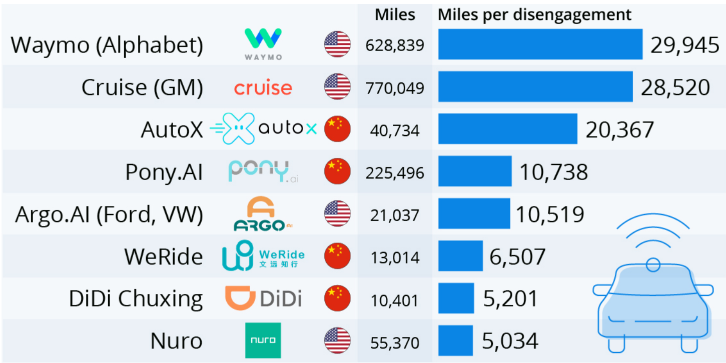


Figure 1.2: Number of autonomous test miles and miles per disengagement (Dec 2019 - Nov 2020)
Source: *DMV California, via The Last Driver License Holder*

At the moment of writing this thesis (2023), most vehicles on the road are considered to be semi-autonomous due to presence of Advanced Driver Assistance Systems (ADASs) that include assisted parking, lane departure warning, driver monitoring, emergency break or Adaptive Cruise Control (ACC), among others. Regarding this, the Society of Automotive Engineers (SAE) published the concept of autonomy levels in 2014, as part of its "Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems" [1] report. Figure 1.3 illustrates the six levels of autonomy (the higher the level, the more autonomous the car is), where it can be appreciated that Level Zero means "No Automation", being the acceleration, braking and steering controlled by a human driver at all times, and Level Five represents Full Automation, where there is a full-time automation of all driving tasks on any road, under any conditions, whether there is a human on board or not.

According to a 2021 McKinsey consumer survey, growing demand for AD systems could create billions of dollars in revenue. Based on a consumer interest in AD features and commercial solutions available on the market today, ADAS and AD could generate between $300 and $400 billions in the passenger car market by 2035. Figure 1.4 illustrates

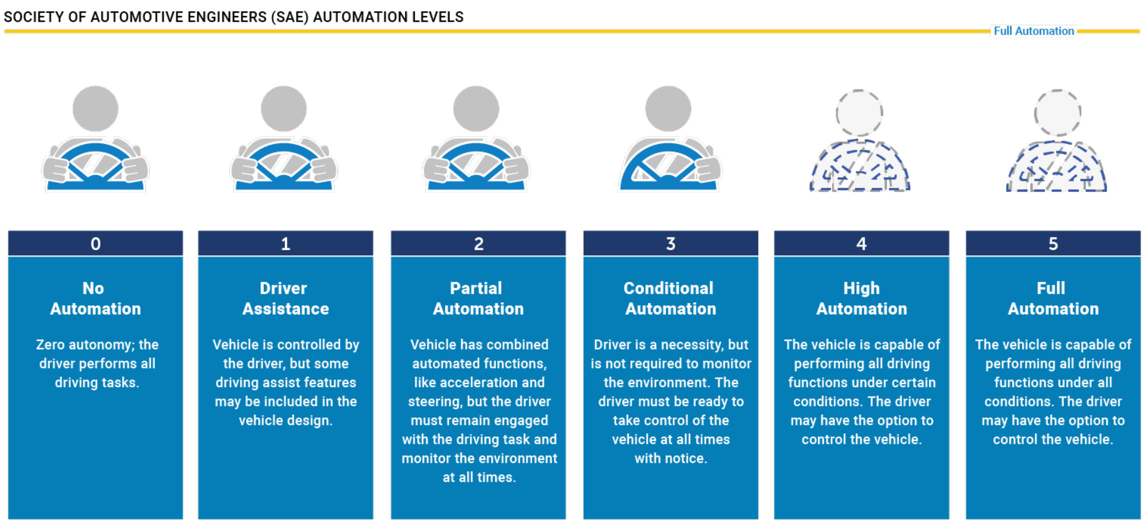


Figure 1.3: Society of Automotive Engineers (SAE) automation levels
Source: *NHTSA (National Highway Traffic Safety Administration)*

an interesting study reporting the revenues of ADAS and AD from Level 1 (Driver Assistance) to Level 4 (High Automation). As expected, Level 5 is excluded from this study due to the huge difficulties the automotive companies would have to face to adapt their systems under totally different environmental conditions.

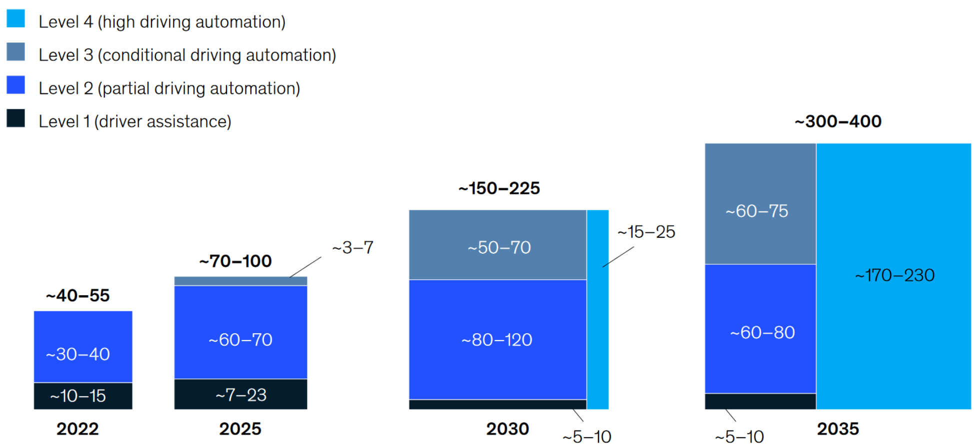


Figure 1.4: Advanced Driver Assistance systems (ADAS) and Autonomous Driving (AD) revenues in \$ billion
Source: *McKinsey Center for Future Mobility*

So far this Chapter has focused on a commercial analysis of the ITS field. The remaining content of this Chapter focuses on the technical study of an ADS, problem statement and objectives of this thesis.

1.3. Autonomous Driving Stack

To sum up what commented above, increasing the level of autonomous navigation in mobile robots (from agriculture to public and private transport) are expected to create tangible business benefits to those users and companies employing them. However, designing an ADS does not seem to be an easy task. In the State-of-the-Art (SOTA) we can distinguish two main kind of software architectures: End-to-End and modular. Note that in this thesis we focus on the software components of the ADS, not on the hardware tasks of the vehicle.

Figure 1.5 illustrates the entire AD architecture starting from sensing to longitudinal (throttle/brake) and lateral (steering angle) control of the vehicle, which are the commanded signals that feed the low-level electronic system that moves the vehicle and that is known as the Drive-By-Wire system [2]. End-to-End are considered black-box models, where a single neural network performs the whole driving task (throttle/steering/brake) from raw sensor data, in such a way the error be may vanished since intermediate representations are jointly optimized, but these are not very interpretable. On the other hand, modular architectures (considered as glass models as counterpart to End-to-End approaches) separate the driving task into individually programmed or trained modules. This solution is more interpretable, since the know-how of a research group or company is easily transferred, they allow parallel development, being the standard solution in industrial research, but the error is propagated, where intermediate representations can led to sub-optimal performance. For example, incorrect object detection can lead to low-quality tracking and Motion Prediction (MP).

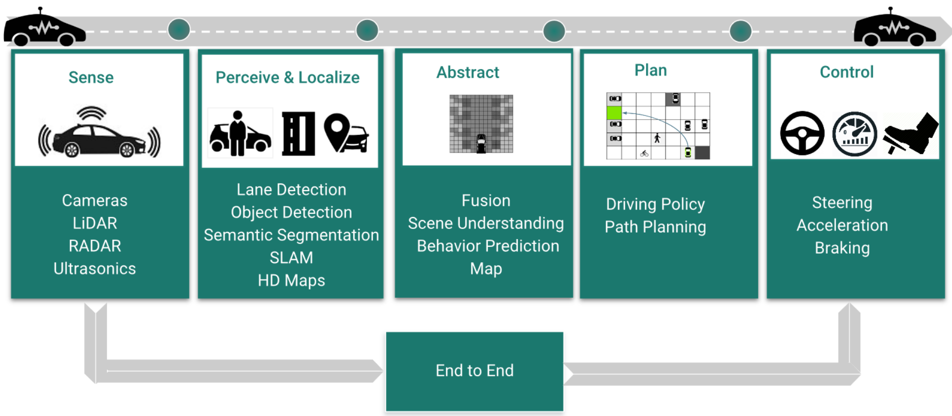


Figure 1.5: Autonomous Driving Stack (ADS) modular vs end-to-end pipeline
Source: *Vrunet: Multi-task learning model for intent prediction of vulnerable road users* [3]

Considering the RobeSafe (Robotics and eSafety) research group and the main projects (Techs4AgeCar, AIVATAR) where this thesis has been developed, we integrate our algorithms in a software modular approach. An example of this modular approach is shown in Figure 1.6. Despite the fact in literature some authors disagree on the specific software architecture of an ADS, specially the motion prediction module, which is usually classified as a perception algorithm but sometimes is included as part of the planning or DM layers, we can hierarchically break down (from raw data to the driving task) a standard AD architecture into the following software layers:

- **Localization layer**: Positions the vehicle on a map with real-time and centimetric accuracy approach. The main source of information is a robust differential-GNSS, though IMUs, wheel odometry and even cameras.

- **Perception layer**: Understands the environment around the ego-vehicle thanks to the information collected by the sensors. If defined as multi-stage, the perception layer first detects the most relevant obstacles, then tracks them over time and predicts their trajectories. In that sense, the perception layer represents one of the most important modules of an ADS, responsible of analyzing the online information, also referred as the traffic situation, through the use of a global perception system which involves different on-board sensors as: LiDAR, Inertial Measurement Unit (IMU), RAdio Detection And Ranging (RADAR), Differential-Global Navigation Satellite System (D-GNSS), Wheel odometers or Cameras. Additionally, HD map information is frequently used in the MP tasks by most SOTA algorithms.

- **Mapping layer**: Responsible for creating a topological, semantic and geographical modeling of the environment through which the vehicle drives, being the HD Map graph the most common source of information.

- **Planning layer**: This layer is comprised of three components: route, behaviour and trajectory planner. The route planner computes the most optimal (in terms of distance, time and so forth and so on) global route from some predefined start and goal. It uses the localization and mapping output. On the other hand, the behaviour planner, also referred as DM layer by some authors, performs high-level DM of driving behaviours such as lane changes or progress through intersections, mostly focused on the previously computed global route and current localization. It can be seen as an atomization of the global route in different behaviors to reach the goal. Finally, the trajectory planner, also known as local planner, generates a time schedule for how to follow a path given constraints such as position, velocity and acceleration in order to meet the previously decided behaviour and taking into account the prediction from the perception layer, avoiding obstacles in optimal direction and speed conditions.

- **Control layer**: Once the local plan is calculated, the control layer is responsible for generating the commands that are sent to the actuators. It receives as input some waypoints from the trajectory planner and most authors perform spline interpolations and a velocity profile that ensures a smooth and continous trajectory.

1.4. Problem statement

As commented in previous sections, in order to operate efficiently and safely in highly dynamic, complex and interactive driving scenarios, ADS need to smartly reason like human beings via predicting future motions of surrounding traffic participants during

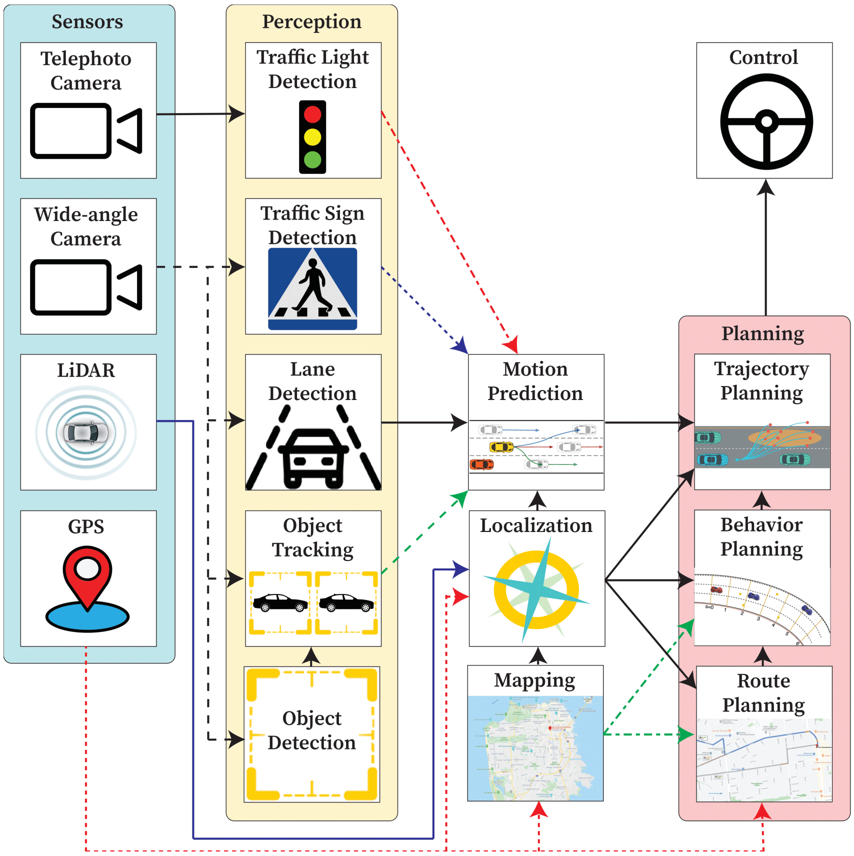


Figure 1.6: Autonomous Driving Stack (ADS) modular pipeline
Source: *Pylot: A modular platform for exploring latency-accuracy trade-offs in autonomous vehicles* [4]

navigation. Nevertheless, achieving accurate and robust MP is one of the most difficult challenges to achieve full-autonomy, since it is equivalent to a bridge between the former stages of the perception layer, where the scene is understood detecting and tracking static and dynamic objects of the environment, and the planning and control layer, where the future trajectory of the ego-vehicle is computed and the driving commands are sent to the physical layer (e.g. Drive-by-Wire [2]). Here are some of the most important challenges:

- **Heterogeneity of traffic participants**: Traffic participants (specially those which are dynamic) can be roughly classified as cyclists, pedestrians or other vehicles. The prediction model should be capable of differentiating the motion patterns of heterogeneous traffic participants, in such a way fine-grained classification (detection module) is quite beneficial to include additional metadata along with the past observations.

- **Complexity of road structure**: Road structures are highly diverse and complex, specially in highways and urban areas, which noticeably affect the motion behaviours of traffic participants.

- **Variable number of interactive agents**: The prediction model must deal with a number of associated traffic participants within a certain area that can vary from time to time, such as intersections or roundabouts. Then, while driving, a comprehensive representation of the scene must be able to accommodate an arbitrary number of involved traffic participants.

- **Multi-modality of driving behaviours**: In real-world, despite we know the behaviour our vehicle will carry out, the motion patterns of other traffic participants can be considered inherently multi-modal since there is usually more than one reasonable option for a driver to choose, specially in intersections, when the number of lanes increases or even in the same lane with different velocity profiles (constant velocity, sudden break, sudden acceleration). In that sense, a robust and reliable MP model is expected to be human-like and capture different plausible motion modalities where an agent can travel in the prediction horizon.

- **Complex interpendencies among traffic participants and road infrastructure**: Agent-Agent, Agent-Road and Road-Road interpendencies are of great importance for MP and interaction modeling, even more taking into account the complexity of road structures and heterogeneity of traffic participants aforementioned. As expected, an agent future trajectory will be affected not only by its own past trajectory and driving objectives (given by the behaviour planner) but also by other surrounding agents past trajectories, traffic rules and physical constraints.

1.5. Contributions of this Thesis

The main scope of this thesis is to develop novel and efficient interaction-aware Deep Learning (DL) based MP models in the field of AD, focusing on long-term (from 3 to 6 s) prediction horizon and AD, where traffic participants can range from trucks to pedestrians. The main inputs will be the physical (map) information and historical states (that may include agent position, velocity, orientation, object type and category) of traffic participants in Bird's Eye View (BEV), assuming these objects have been previously tracked by our ego-vehicle (also referred as the autonomous car).

The validation of these methods will be done using a single target agent (either the model has considered multiple or a single agent in the loss function), as proposed by some of the most important prediction datasets in the literature, like Argoverse 1 [5] and Argoverse 2 [6], at the moment of writing this thesis. In this work, the solutions to the aforementioned challenges will be discussed and investigated progressively.

We summarise the contributions of this doctoral thesis in the following points:

- **Contribution 1**: SmartMOT, a simple-yet-accurate combination of traditional techniques is proposed for state estimation and data association respectively, in order to solve the tracking-by-detection paradigm. Moreover, we incorporate HD map semantic, geometric and topological information, in addition to the ego-vehicle status, to enhance the efficiency and reliability of the Multi-Object Tracking (MOT) system and subsequent uni-modal predictions (Chapter 4).

- **Contribution 2**: An Attention-based Generative Adversarial Network (GAN) prediction model that can successfully predict plausible uni-modal future trajectories in the context of vehicle prediction (Argoverse 1 dataset), taking into account not only the past trajectory of the agents (encoded by Long Short-Term Memory (LSTM) networks and attention mechanisms) but also the HD map information to compute a set of acceptable target points representing the physical constraints for our problem (Chapter 5).
- **Contribution 3**: Three efficient (social, map and augmented) baselines for multi-modal MP in the Argoverse 1 dataset built upon the previous model proposed in Chapter 5 removing the adversarial framework. In this case, we integrate Graph Neural Networks (GNNs) to compute complex interactions among the different agents and a heuristic-based method to calculate some preliminary future centerlines for the vehicles as an enhanced interpretable representation of the map information with respect to the acceptable target points aforementioned (Chapter 6.2.1, 6.2.2 and 6.2.3).
- **Contribution 4**: A multi-modal multi-agent MP model in the Argoverse 2 dataset, built upon the augmented efficient baseline, which incorporates topological and semantic information of preliminary future lanes using the aforementioned heuristic method, map encoding based on DL with GNN, a cycle of physical and social feature fusion, DL-based estimation of final positions on the road, aggregation of the surrounding environment, and finally, a refinement module to enhance the quality of the final multi-modal predictions in an elegant and efficient manner. Compared to the state of the art, our method achieves prediction metrics up-to-pair with to the top-performing methods on the Argoverse 2 Leaderboard while significantly reducing the number of parameters and floating-point operations per second (Chapter 7).
- **Contribution 5**: The final model of the thesis, only considering social information, is validated in two interesting applications, such as DM in the SMARTS simulator or domain adaptation studies in the hyper-realistic CARLA simulator, involving other vehicle layers as a preliminary step towards implementation in a real autonomous vehicle (Chapter 8.2 and 8.3).
- **Contribution 6**:

In order to achieve the main scope, the following objectives will be met:

1. Review of SOTA MP, focused on DL and the AD paradigm.
2. Propose of several efficient MP architectures, studying the progressive incorporation of DL mechanisms and different sources of information and metadata, achieving

SOTA accuracy while reducing in millions of parameters previous models as well as inference time.

3. Validate the proposed models in downstream applications, such as DM or behaviour planning, taking into account former stages of the perception layer (detection and tracking) instead of static files (benchmarks) in hyper-realistic simulation, as a preliminary stage before implementing it in a real-world vehicle.

1.6. Structure of this Thesis

The organization of this document has been done as follows:

- **Chapter 2** reviews the contextual factors, a MP methods classification according to the context encoding or representation approaches and SOTA databases and simulators to validate the algorithms.
- **Chapter 3** presents a technical background, mostly focused on physics-based methods and DL mechanisms to deal with temporal sequences and interactions, to deeply understand the proposed methods.
- **Chapter 4** addresses our integration between single-yet-powerful MOT and HD map information as a preliminary stage before computing uni-modal predictions.
- **Chapter 5** illustrates our GAN-based proposal, the first DL-based vehicle MP method of this thesis, considering both physical context, computing acceptable target points from the driveable area around the target agent, and social context, computing the past trajectory as a temporal sequence via recurrent networks and social interactions with attention mechanism.
- **Chapter 6** presents our efficient baselines, where high-level and well-structured physical context is structured in the form of centerlines, Graph Convolutional Network (GCN)-based approaches are studied to model more complex agent-agent interactions and transformer encoders are employed at the end of the chapter for powerful-yet-efficient context encoding and multi-modal decoding.
- **Chapter 7** illustrates the final model of the thesis which takes into account agent-agent, agent-map, map-agent and map-map interactions, using a novel scene representation with heuristic proposals, graph-based encoding, DL-based goal areas proposals and motion refinement.
- **Chapter 8** addresses the integration and validation of the final model of the thesis in a hyper-realistic simulator with upstream and downstream modules to contribute the entire pipeline and closed-loop for AD.

- **Chapter 9** summarizes the thesis and provides some promising directions for future work in the areas of MP and validation.

Chapter 2

Related Works

Llegaré a ser el mejor, El mejor que habrá jamás
Mi causa es ser su entrenador, Tras poderlos capturar.
Viajaré a cualquier lugar, Llegaré a cualquier rincón
Y al fin podré desentrañar, El poder de su interior.
¡Pokémon! Hazte con todos (solos tú y yo),
Es mi destino, mi misión
¡Pokémon! Tú eres mi amigo fiel,
Nos debemos defender.

Opening 1 de Pokémon: "Gotta catch 'em all!"
Autor original: Jason Paige

2.1. Introduction

One of the crucial tasks that ADSs must face during navigation, specially in arbitrarily complex urban scenarios, is to predict the behaviour of dynamic obstacles [5], [7]. In a similar way to humans that pay more attention to nearby obstacles and upcoming turns than considering the obstacles far away, the perception layer of an ADS must focus more on the salient regions of the scene, particularly on the more relevant dynamic agents to predict their future behaviour before conducting a maneuver, such as lane changing or accelerating.

Before proceeding with the study of the different methods of the SOTA of MP in the field of AD, one important thing to note is that this thesis is focused on non-conditional motion prediction, also referred as Passive Motion Prediction (PMP), where the prediction of surrounding agents is not influenced by the future decisions of the ego-vehicle or even other agents, referred as Conditional Motion Prediction (CMP) in the literature. Most existing works [8]–[13] focus on a passive prediction scheme, where the future states of a particular agent are predicted given its past information, other surrounding agents information and interactions as well as the physical context. Then, downstream planning modules, specially the behaviour planning module (also referred as DM layer, as stated in Section 1.3) and the ego-vehicle (our vehicle) future actions are computed according to the predicted trajectories in a passive manner, that is, without modifying the output of the prediction model, and the global route previously calculated.

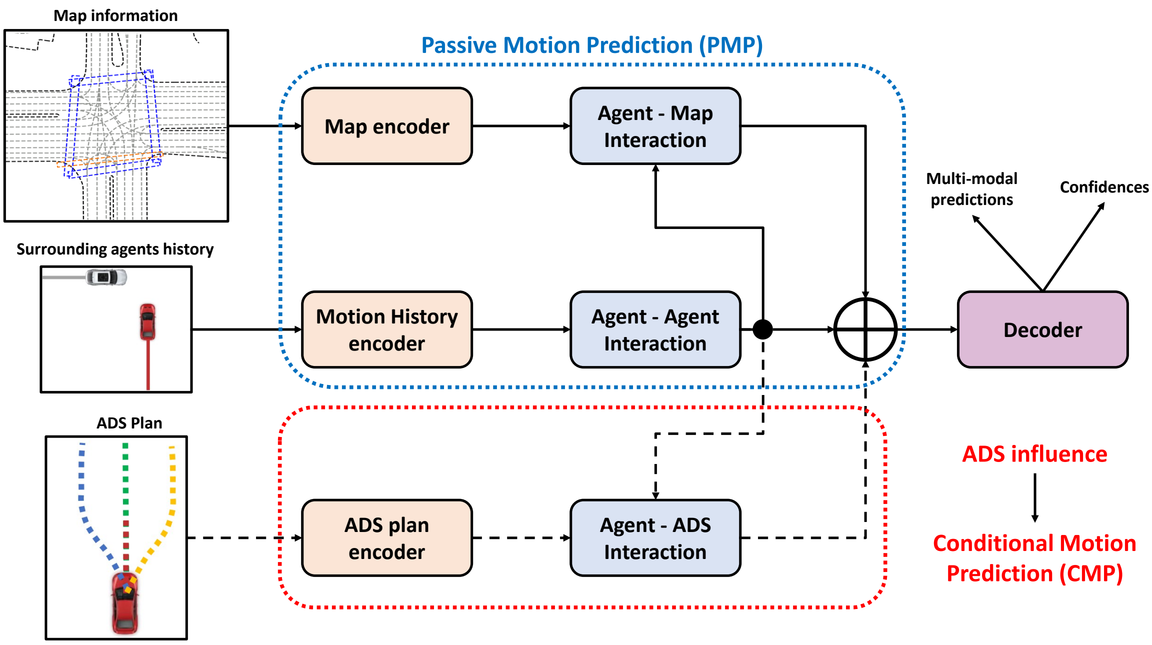


Figure 2.1: Example of a Conditional Motion Prediction (CMP) pipeline. We **highlight** the influence of the ADS in the prediction of surrounding agents.

Nevertheless, to ensure safety under various predicted trajectories of the surrounding agents, our ADS must overly conservative with inefficient maneuvers, specifically in arbitrarily complex traffic scenarios, because PMP models ignore the fact that the future states of an agent can influence the future actions of other agents, what is the most realistic situation. Figure 2.1 shows the differences between these two approaches. To this end, researchers recently started to explore a more coherent interactive prediction and planning framework which relies on predicting the surrounding agents future trajectories conditioned on the ego-vehicle future actions [14] [15] [16], as a preliminary state to implement a fully-interaction graph where the future states of all agents (either autonomous prototypes or human-driven) influence in the decision of all agents.

Once the differences between CMP and PMP have been illustrated, we proceed with the problem formulation, main contextual factors and classification of prediction methods.

2.2. Problem Formulation of Motion Prediction

Given a sequence of past trajectories $a_P = [a_{-obs'_{len}+1}, a_{-obs'_{len}+2}, ..., a_0]$ for an agent, we aim to predict its future steps $a_F = [a_1, a_2, ..., a_{pred_{len}}]$ up to a fixed time step $pred_{len}$. Running in a specific traffic scenario, each agent will interact with static HD maps m and the other dynamic actors, meeting the corresponding traffic and social rules. Therefore, the probabilistic distribution that we want to capture is $p(a_F|m, a_P, a_P^O)$, where a_P^O denotes the other agents observed states.

The output of most existing methods is a weighted set of trajectories $A_F = \{a_F^k\}_{k\in[0,K-1]} = \{(a_1^k, a_2^k, ..., a_{pred_{len}}^k)\}_{k\in[0,K-1]}$ for each agent, where K represents the number of modes or plausible future directions, due to the inherent uncertainty associated to the prediction problem. Note that the set is weighted since each mode will have an associated confidence or probability of occurrence, where the sum of all mode probabilities must be equal to 1. This weighted set of trajectories for each agent will be used by downstream decision modules. On top of that, TNT (Target-driven trajectory prediction) [17] is one of the first methods that introduces specific preliminary future positions in the problem formulation, also referred as goals, being TNT [17]-like methods distribution approximated as:

$$\sum_{\tau\in T(m,a_P,a_P^O)} p(\tau|m, a_P, a_P^O)p(a_F|\tau, m, a_P, a_P^O) \tag{2.1}$$

where $T(m, a_P, a_P^O)$ is the space of candidate goals depending on the driving context and τ a specific goal.

However, the map space m is large, and the goal space $T(m, a_P, a_P^O)$ requires careful design. In that sense, some methods expect to accurately predict the actor motion by extracting good features. For example, LaneGCN [13] tries to approximate $p(a_F|m, a_P, a_P^O)$ by modeling $p(a_F|M_{a_0}, a_P, a_P^O)$, where M_{a_0} is a "local" map features that is related to the actor's state a_0 at final observed step $t = 0$.

To extract M_{a_0}, they use a_0 as an anchor to retrieve its surrounding map elements and aggregate their features. We found that not only the "local" map information is important, but also the goal area maps information is of great importance for accurate trajectory prediction. So, the probability can be reconstructed as:

$$\sum_{\tau} p(\tau|M_{a_0}, a_P, a_P^O)p(M_\tau|m, \tau)p(a_F|M_\tau, M_{a_0}, a_P, a_P^O) \tag{2.2}$$

Then, in this work we aim to include preliminary heuristic information, as well as predict possible goals τ based on agents motion histories and driving context to retrieve the map elements in goal areas explicitly and aggregate their map features as M_τ. to meet the requirements of the problem formulation.

2.3. Contextual Factors and Classification of Motion Prediction methods

This section studies the contextual factors (inputs) and classification of MP in the field of AD according to its encoding method of the different inputs and output types. Figure 2.2 summarizes the main inputs and outputs of these methods.

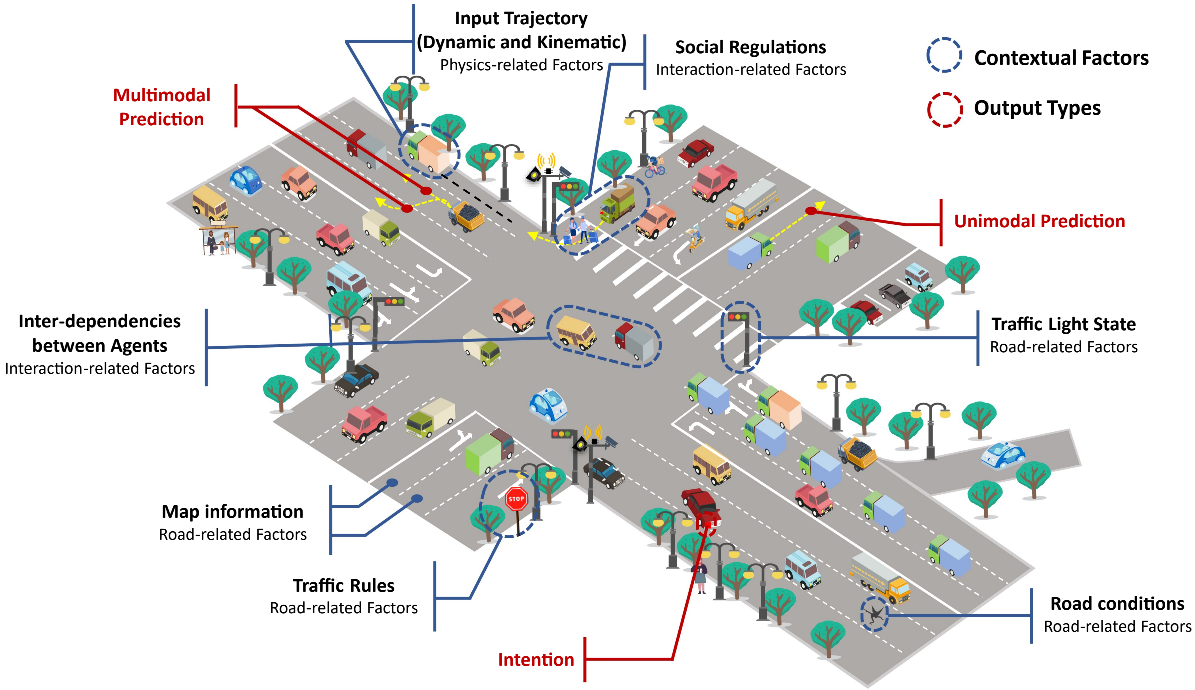


Figure 2.2: Contextual factors and output types in Vehicle Motion Prediction
Source: *A survey on trajectory-prediction methods for autonomous driving* [18]

In order to model the future states of surrounding agents, MP models must pay attention to the current environment, where some contextual factors can be clearly identified:

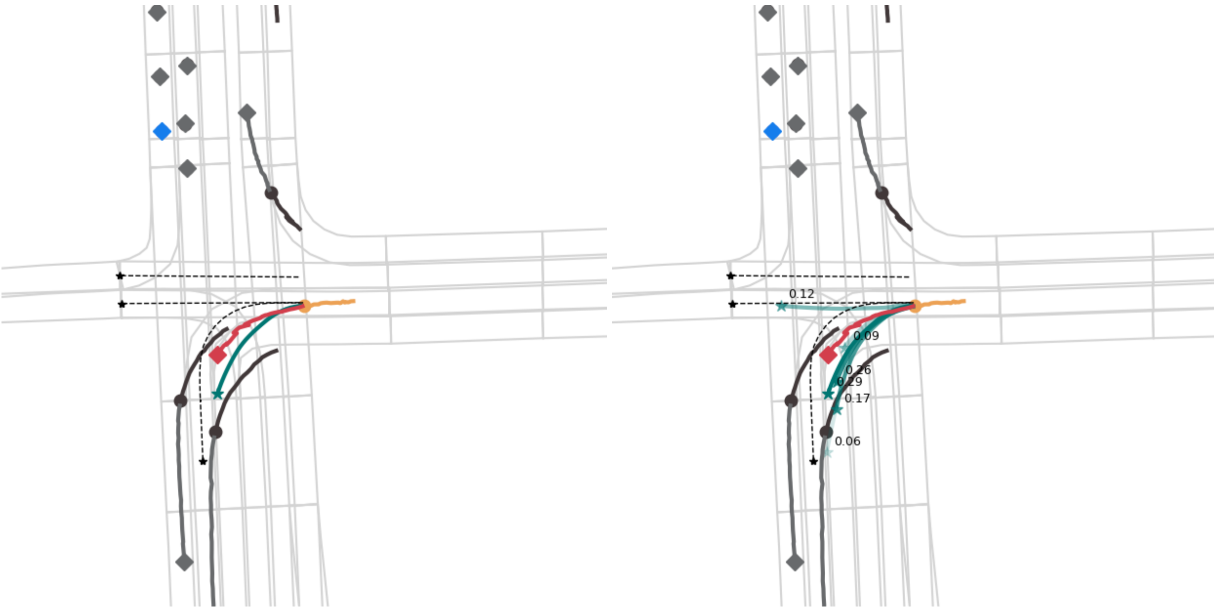
- **Physics-related factors** refer to the kinematic and dynamic variables of the agents, specifically to their spatio-temporal variables (such as position, velocity, acceleration, object type or mass) as well as past behaviours.
- **Interaction-related factors** include the inter-dependencies and social regulations between agents maneuvers. It is important to consider that traffic agents can be either non-relevant pedestrians on the sidewalk as well as an extremely relevant truck in front of the ego-vehicle.
- **Road-related factors** include the corresponding traffic rules (lane type, traffic signals, stops, etc.) as well as the modeling of the map (usually HD-map), including its topological, semantic and geometrical information.

On the other hand, regarding the output types, MP methods need to provide the future trajectories of traffic participants. Nevertheless, these methods can provide these future trajectories in different ways, even though these outputs can be unified as a single output, depending on the application:

- **Uni-modal prediction**: In the uni-modal case, the prediction method only returns a single future trajectory, without taking into account other possible behaviours.
- **Multi-modal prediction**: Models that generate a multi-modal prediction compute multiple K future trajectories (also referred as modes in the literature) with the probability of each future trajectory. The higher the mode probability (also referred as score), the more probable a particular future trajectory should be. Instead of providing a single trajectory, these models would generate a range of probable trajectories, considering different driving styles, intentions, and uncertainties. Multi-modal prediction can enable ADS to anticipate and respond to a wider range of possible scenarios, enhancing safety and adaptability. This output type is specially useful for fast-changing and highly interactive situations where multiple options are available. One important thing to note is that multi-modal methods must be designed in order to not only reason in terms of different maneuvers (keep straight, turn right, lane change, etc.) but also different velocity profiles (constant velocity, acceleration, sudden break, etc.) regarding the same maneuver. This type of prediction will be the final objective throughout this thesis.
- **Intention**: Also referred as maneuver in the literature, the intention can be part of the final output or just be an intermediate step in the method. MP methods usually produce maneuver intention (*i.e.* a discrete space of actions, such as turn left, turn right, sudden acceleration, emergency break, and so forth and so on) to assist in the subsequent prediction. In the literature, the maneuver or behaviour is usually returned by a behavioural planner (DM) module, while the specific future trajectory is usually returned by a MP algorithm to compute the specific future steps of the agents.

As we will study throughout this section, most prediction methods focus on the multi-modal output with an associated probability for each mode, since this is the most realistic way to imitate the human brain during navigation. First of all, there are plenty of problems during navigation, since there is a high uncertainty of traffic behavior and a large number of different situations. That means that one cannot use a discrete number of situations and a discrete number of car movements. Second, the main tasks of ADS, like ensuring safe and efficient operations and anticipating a multitude of possible behaviors of traffic actors in its surroundings, provide a large need in knowing the position of all vehicles beforehand. Multi-modal means that we have multiple predictions for each timeframe.

Figure 2.3 illustrates an interesting traffic scenario processed by one of our algorithms. The target agent is getting close to an intersection, where several future maneuvers are plausible. In both cases (uni-modal and multi-modal, which are the most common



(a) Uni-modal prediction

(b) Multi-modal prediction

Figure 2.3: Modeling uni-modality vs multi-modality of future 3-second agent trajectories with one of our algorithms; We represent: our vehicle (**ego**), the **target agent**, and **other agents**. We can also see the **ground-truth** trajectory of the target agent, our **multi-modal predictions** (with the corresponding confidences) for the target agent and **plausible centerlines**, also for the target agent in this algorithm. Circles represent last observations and diamonds last future positions.

ones), the model must reason the future trajectory of the target agent based on its past observations, interactions with other agents and physical context. On the left (2.3a) illustrates the uni-modal case, where a single trajectory is predicted. On the right (2.3b), multiple trajectories (or modes) with associated probabilities are predicted, with most modes turning left and one mode keeping straight since in similar situations the agent could also perform this behaviour with a similar context. As stated above, the main point of having this multi-modality is to compute different plausible options with an associated confidence. Note that lots of SOTA algorithms study the problem of multi-modal predictions since similar past trajectories (*e.g.* a vehicle stopped in front of an intersection) can produce totally different future trajectories (*e.g.* turn left, turn right or keep straight).

After defining the contextual factors and output types, a classification of the prediction methods according to different modeling approaches is illustrated. Over the last two decades, MP can be divided into four parts in chronological order: Physics-based, classic Machine Learning (ML)-based, Reinforcement Learning (RL)-based and DL-based, as shown in Figure 2.4. The remaining content of this section, based on the study performed by [18], illustrates the main algorithms used in this thesis, especially focusing on DL since in this work we focus on predictive techniques for scene understanding based on deep models.

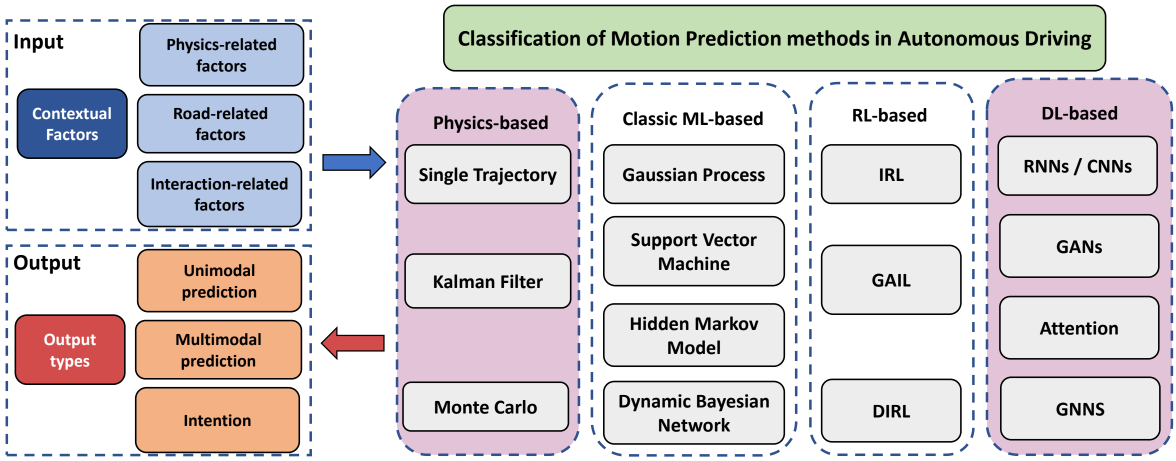


Figure 2.4: Contextual factors and output types in Vehicle Motion Prediction. We highlight the main algorithms used in this thesis: Physics-based and Deep Learning based.

2.3.1. Physics-based Motion Prediction

Physics-based methods are the first and simplest methods used by researchers. Although the accuracy of these methods is relatively low, more and more models use the idea of physics-based models to improve the accuracy. Physics-based methods have more accurate results when the movement of vehicles can be accurately described by kinematics or dynamics models, but the physical model of the traffic participants is constantly changing, such that most of these methods are only suitable for short-term prediction (no more than 1s). Dynamics models can be quite complex, including many inherent parameters and introducing an extra computation burden, in such a way that researchers prefer simple dynamic methods for motion prediction. In terms of vehicle MP, the bicycle model is usually employed to model the vehicle physics, driven by the front wheels [19], [20]. In the literature three main types of physics-based models are distinguished, where the main different is the way in which the uncertainty is handled: Single Trajectory, Kalman Filter (KF)-based and Monte Carlo-based.

2.3.1.1. Single-Trajectory

One of the most straightforward methods to predict an agent trajectory is to directly apply the agent current state to the physic model. In order to increase the accuracy and stability of the estimation, the vehicles are mostly assumed to comply with motion models that describe their dynamic behavior. In the past, numerous single-trajectory motion models have been proposed for this task [20]–[22]. A first systematization can be achieved by defining different levels of complexity. At the lower end of such a scale, linear motion models are situated. These models assume a Constant Velocity (CV) or a

Constant Acceleration (CA). Their major advantage is the linearity of the state transition equation which allows an optimal propagation of the state probability distribution. On the other hand, these models assume straight motions and are thus not able to take rotations (especially the yaw rate) into account.

A second level of complexity can be defined by taking rotations around the z-axis into account. The resulting models are sometimes referred to as curvilinear models. They can be further divided by the state variables which are assumed to be constant. The most simple model of this level is the Constant Turn Rate and Velocity (CTRV) model. By defining the derivative of the velocity as the constant variable, the Constant Turn Rate and Acceleration (CTRA) model can be derived. Both CTRV and CTRA assume that there is no correlation between the velocity v and the yaw rate ω. As a consequence, disturbed yaw rate measurements can change the yaw angle of the vehicle even if it is not moving. In order to avoid this problem, the correlation between v and ω can be modeled by using the steering angle Φ (angle between the axis of motion and the direction of the front wheels) as constant variable and derive the yaw rate from v and Φ. The resulting model is called Constant Steering Angle and Velocity (CSAV). Again, the velocity can be assumed to change linearly, which leads to the Constant Curvature and Acceleration (CCA) model.

From a geometrical point of view, nearly all curvilinear models are assuming that the vehicle is moving on a circular trajectory (either with a constant velocity or acceleration). The only exception is the CTRA model, which models a linear variation of the curvature and thus assumes that the vehicle is following a clothoid.

On the other hand, single trajectory methods are not able to consider the road-related factors and the uncertainty of the current state is unreliable for long-term prediction in such a way these single trajectory models should only be used for estimating uni-modal trajectories of the surrounding agents in the short-term. We further study the state transition equations of physics-based models in Chapter 3 since they will be used in the algorithms proposed in this thesis as preliminary proposals for the DL models.

2.3.1.2. Kalman Filter

Kalman Filter (KF)-based methods aim to solve one of drawbacks of physics-based models: In real-world, the states of agents are not perfectly known since they present an associated noise. These methods model the uncertainty of the current agent state and its physic model by means of a Normal (Gaussian) distribution. Compared to the single trajectory methods, the main advantage is that KF methods consider the uncertainty of the predicted trajectory, specially when using its Extended (EKF) or Unscented (UKF) versions where non-linearities are modeled. As proposed by the original algorithm [23], the prediction and update steps are combined into a loop where the mean value and covariance matrix of the agent state is computed for each future step, calculated as an average trajectory with related uncertainty.

Nevertheless, these KF-based methods use uni-modal Gaussian distributions, which are not enough to represent agents interactions. In that sense, [24] propose an Interactive Multiple Model (IMM) to compute a multi-modal prediction. Moreover, [25] model a set of KFs used to describe physical models of the vehicles and switch between them, defined as Switched Kalman Filter (SKF). [26] propose IMM-KF, a novel Interacting Multiple Model Kalman Filter which takes interaction-related factors (social regulation, inter-dependencies) into consideration, as shown in Figure 2.2.

2.3.1.3. Monte Carlo

In the same way KF methods aimed to solve the associated noise to the physics state of the agent, Monte Carlo method aims to simulate the state distribution approximately since an analytical expression for the predicted state distribution is usually unknown without any assumptions of the linearity or the model's Gaussian nature. This method randomly samples the input variables and applies the physics model to compute potential future trajectories. In order to ensure the plausibility of the future behaviour in the context of AD, the generated future states are usually filtered with a lateral acceleration lower than the actual allowable lateral acceleration [27], though other vehicle physical limitation can also be used such that the input of the model will be more realistic. [28] present a model that identifies a preliminary maneuver and then applies the Monte Carlo method to compute future trajectories by the identified maneuver. Furthermore, [29] first use the Monte Carlo algorithm to predict future trajectories and then utilize MPC (Model Predictive Control) algorithm to refine these preliminary future trajectories.

2.3.2. Deep Learning based Motion Prediction

DL-based methods are by far the most used at this moment in the field of MP in AD to predict the future trajectory of traffic participants, being this thesis focused in these particular methods. Most traditional predictions methods [18], which usually only consider physics-related factors (like the velocity and acceleration of the target vehicle that is going to be predicted) and road-related factors (prediction as close as possible to the road centerline), are only suitable for short-time prediction tasks [18] and simple traffic scenarios.

Recently, MP methods based on DL have become increasingly popular since they are able not only to take into account these above-mentioned factors but also consider interaction-related factors (like agent-agent [30], agent-map [31] and map-map [13]) in such a way the algorithm can adapt to more complex traffic scenarios (intersections, sudden breaks and accelerations, etc.). It must be consider that multi-modal, specially in the field of vehicle motion prediction, does not refer necessarily to different directions (*e.g.* turn to the left, turn to the right, continue forward in an intersection), but it may refer to different predictions in the same direction that model a sudden positive or negative acceleration, so as to imitate a realistic human behaviour in complex situations.

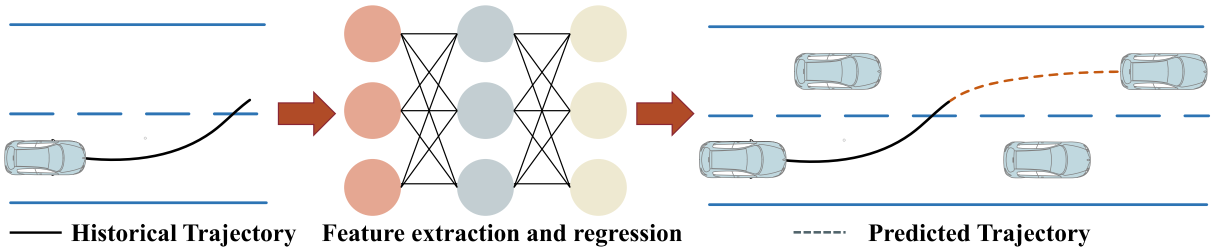


Figure 2.5: Deep Learning methods applied in Motion Prediction
Source: *A survey on trajectory-prediction methods for autonomous driving* [18]

The main DL-based approaches are: Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), GANs, Attention mechanisms (where the Transformer architecture is included) and GNNs. Figure 2.5 illustrates the main idea of using DL to predict the future trajectories of the agents. Since this thesis is focused on developing efficient and accurate DL-based MP models in the field of AD, the theoretical explanation of the different DL mechanisms used in our pipelines will be further detailed in Chapter 3 along with some physics-based models theory to fully-understand the proposed models. The input is represented by, at least, the physics-factors of the corresponding, though road-factors and interaction-factors are present in most DL algorithms. Then, a neural network extracts the most important features and outputs a future trajectory according to the training process in a supervised way.

In order to classify DL based MP methods, we can identify different factors (physics, interaction and road) and determine which type of neural network is employed to extract features of the corresponding input. In the literature we mainly distinguish the following inputs and outputs: Motion history (physics-based factors), Social information (interaction-based factors), Map information (road-relared factors), how the model returns the output trajectory and its corresponding distribution. Table 2.1 summarizes main SOTA methods.

- **Motion history**: Most methods encode the sequence of past observed states using 1D-CNN [13], [33], able to model spatial information, or via a recurrent net [32] (LSTM, Gated Recurrent Unit (GRU)), which are more useful to handle temporal information. Other methods that use a raster version of the whole scenario represent the agent states rendered as a stack of binary mask images depicting agent oriented bounding boxes [8]. On the other hand, other approaches encode the past history of the agents in a similar way to the road components of the scene given a set of vectors or polylines [17], [34] that can model the high-order interactions among all

Table 2.1: Main SOTA DL methods for MP. Main categories are Encoder, Decoder, Output representation and Distribution over future trajectories

Method	Encoder			Decoder	Output	Trajectory Distribution
	Motion history	Social info	Map info			
SocialLSTM [32]	LSTM	spatial pooling	–	LSTM	states	samples
SocialGan [30]	LSTM	maxpool	–	LSTM	states	samples
Jean [33]	LSTM	attention	–	LSTM	states	GMM
TNT [17]	polyline	maxpool, attention	polyline	MLP	states	weighted set
LaneGCN [13]	1D-conv	GNN	GNN	MLP	states	weighted set
WIMP [16]	LSTM	GNN+attention	polyline	LSTM	states	GMM
VectorNet [34]	polyline	maxpool, attention	polyline	MLP	states	uni-modal
SceneTransformer [35]	attention	attention	polyline	attention	states	weighted set
HOME [8]	raster	attention	raster	conv	states	heatmap
GOHOME [9]	1D-conv+GRU	GNN	GNN	MLP	states	heatmap
MP3 [36]	raster	conv	raster	conv	cost function	weighted samples
CoverNet [37]	raster	conv	raster	lookup	states	GMM w/ dyn. anch.
DESIRE [38]	GRU	spatial pooling	raster	GRU	states	samples
MFP [14]	GRU	RNNs+attention	raster	GRU	states	samples
MANTRA [39]	GRU	–	raster	GRU	states	samples
PRANK [40]	raster	conv	raster	lookup	states	weighted set
IntentNet [31]	raster	conv	raster	conv	states	uni-modal
Multi-modal [41]	raster	conv	raster	conv	states	weighted set
MultiPath [42]	raster	conv	raster	MLP	states	GMM w/ static anchors
MultiPath++ [10]	LSTM	RNNs+maxpool	polyline	MLP	control poly	GMM
PLOP [43]	LSTM	conv	raster	MLP	state poly	GMM
Trajectron++ [7]	LSTM	RNNs+attention	raster	GRU	controls	GMM
CRAT-PRED [12]	LSTM	GNN+attention	–	MLP	states	weighted set
R2P2 [44]	GRU	–	polyline	GRU	motion	samples
DKM [45]	raster	conv	raster	conv	controls	weighted set
GANet [11]	1D-conv+GRU	GNN	GNN	MLP	states	weighted set

components, or even employing attention to combine features across road elements and agent interactions [35]. In our approaches, we will mainly use LSTM, 1D-CNN and Attention mechanism given the relative displacements of the agents.

- **Social information**: In complex scenarios, motion history encoding of a particular target agent is not sufficient to represent the latent space of the traffic situation, but the algorithm must deal with a dynamic set of neighbouring agents around the target agent. Common techniques are aggregating neighbour motion history with a permutation-invariant set operator: soft attention [35], a combination of soft attention and RNN [10] / GNN [12] or social pooling [30], [32]. Raster based approaches rely on 2D convolutions [42] [36] over the spatial grid to implicitly capture agent interactions in such a way long-term interactions are dependent on the neural network receptive fields. We will make use of Attention-based and GNN-based algorithms in our approaches.

- **Map information**: High-fidelity maps [46] have been widely adopted to provide offline information (also known as physical context) to complement the online information provided by the sensor suite of the vehicle and its corresponding algorithms. Recent learning-based approaches [31], [47], [48], which present the benefit of having probabilistic interpretations of different behaviour hypotheses, require to build a representation to encode the trajectory and map information. Map information is probably the feature with the clearest dichotomy: raster vs vector treatment. The raster approach encodes the world around the particular target agent as a stack of images (generally from a top-down orthographic view, also known as Bird's Eye View). This world encoding may include from agent state history, agent interactions and usually the road configuration, integrated all this different-sources information as a multi-channel image [8], in such a way the user can use an off-the-shelf CNN-based pipeline in order to leverage this powerful information. Nevertheless, this representation has several downsides: constrained field of view, difficulty in modeling long-range interactions and even difficulty in representing continuous physical states due to the inherent world to image (pixel) discretization.

On the other hand, the polyline approach may describe curves, such as lanes, boundaries, intersections and crosswalks, as piecewise linear segments, which usually represents a more compact and efficient representation than using CNNs due to the sparse nature of road networks. Some state-of-the-art algorithms not only describe the world around a particular agent as a set-of-polylines [16] [17] in an agent-centric coordinate system, but they also leverage the road network connectivity structure [13] [49] treating road lanes as a set of nodes (waypoints) and edges (connections between waypoints) in a graph neural network so as to include the topological and semantic information of the map. In our case, we will use discrete map information in the form of target goals and high-level well-structured centerlines, as well as road network connectivity structure using GNN-based operations to compute deep physical features.

- **Decoder**: Pioneering works of DL based MP usually adopt the autoencoder architecture, where the decoder is often represented by a recurrent network (GRU, LSTM, etc., specially designed to handle temporal information) to generate future trajectories in an autoregressive way, or by CNNs [8] [9] / MLP [13] [12] using the non-autoregressive strategy. The method may use an autoregressive strategy where the pipeline generates tokens (in this case, positions or relative displacements) in a sequential manner, in such a way the new output is dependent on the previously generated output, whilst Multilayer Perceptron (MLP) [12], CNN [8] or transformer [35] based strategies usually follow a non-auto-regressive approach, where from a latent space the whole future trajectory is predicted. Throughout this work, we will make use of the auto-regressive strategy using LSTM networks as well as the decoding from the latent space in the latest proposed algorithm, where we will appreciate that given an enriched encoded representation, it is possible to directly decode from the deep features instead of carrying out the decoding process iteratively.

- **Output**: The most popular model output representation is a sequence of states (absolute positions) or state differences (relative displacements for any dimension considered). The spacetime trajectory may be intrinsically represented as a continous polynomial representation or a sequence of sample points. Other works [8] [9] first predict a heatmap and then decode the corresponding output trajectories after sampling points from the heatmap, whilst [36] [50] learn a cost function evaluator of trajectories that are enumerated heuristically instead of being generated by a learned model. As proposed by most methods, we will represent our output as a sequence of states, being absolute positions when decoding from the encoded latent space or relative displacements when decoding iteratively by means of the auto-regressive approach as aforementioned.

- **Trajectory Distribution**: The choice of output trajectory distributions has several approaches on downstream applications. Regardless the agent to be predicted is described as a (non-)holonomic [51] platform, an intrinsic property of the MP problem is that the agent may follow one of a diverse set of possible future trajectories, instead of a single future trajectory (uni-modal) [34], which might not be optimal. To address the multi-modal issue, a popular choice to represent a multi-modal prediction are Gaussian Mixture Models (GMMs) due to their compact parameterized form, where mode collapse (associated frequently to GMMs) is addressed through the use of trajectory anchors [42] or training tricks [41]. Other approaches model a discrete distribution via a collection of trajectory samples extracted from a latent space and decoded by the model [44] or over a set of trajectories (fixed a priori or learned). In that sense, at the moment of writing this work, the most used trajectory distribution is a weighted set of trajectories [12], [13], [17], where each set is a trajectory with a discrete number of future steps and an associated confidence indicating the probability of occurrence of the corresponding behaviour. In this work, for those approaches which make use of the multi-modal approach, a weighted set will be used to model the trajectory distribution, where each trajectory is a sequence of discrete states, and each sequence has a certain confidence illustrating the probability of occurrence of that particular behaviour.

2.4. Motion Prediction Datasets

As stated in previous sections, MP (also referred in the literature as Motion Forecasting) addresses the problem of predicting future states (or occupancy maps) for dynamic actors within a local environment. Some examples of relevant actors for autonomous driving include: vehicles (both parked and moving), pedestrians, cyclists, scooters, and pets. Predicted futures generated by a forecasting system are consumed as the primary inputs in motion planning, which conditions trajectory selection on such forecasts. Generating these forecasts presents a complex, multi-modal problem involving many diverse, partially-observed, and socially interacting agents. Considering these requirements, there are several datasets in the literature to train and validate multiple proposals that vary in terms of data size, geographic coverage, sensor modalities, annotations, and limitations. Choosing the most appropriate dataset depends on the specific research goals, target environment, and available resources. Researchers should consider these factors when selecting a dataset for motion forecasting tasks in autonomous driving. Table 2.2 shows an interesting comparison between different SOTA datasets in the field of vehicle MP.

Table 2.2: Comparison between different *state-of-the-art* vehicle Motion Prediction datasets. Hyphens "-" indicate that attributes are either not applicable, or not available. † Public leaderboard counts as retrieved on Aug. 27, 2021.
Source: *Argoverse 2: Next generation datasets for self-driving perception and forecasting* [6]

	Argoverse 1 [5]	Interaction [52]	Lyft [53]	Waymo [54]	NuScenes [55]	Yandex [56]	Argoverse 2 [6]
Scenarios	324k	-	170k	104k	41k	600k	250k
Unique Tracks	11.7M	40k	53.4M	7.6M	-	17.4M	13.9M
Average Track Length	2.48 s	19.8 s	1.8 s	7.04 s	-	-	5.16 s
Total Time	320 h	16.5 h	1118 h	574 h	5.5 h	1667 h	763 h
Scenario Duration	5 s	-	25 s	9.1 s	8 s	10 s	11 s
Test Forecast Horizon	3 s	3 s	5 s	8 s	6 s	5 s	6 s
Sampling Rate	10 Hz	10 Hz	10 Hz	10 Hz	2 Hz	5 Hz	10 Hz
Cities	2	6	1	6	2	6	6
Unique Roadways	290 km	2 km	10 km	1750 km	-	-	2220 km
Avg. tracks per scenario	50	-	79	-	75	29	73
Evaluated object categories	1	1	3	3	1	2	5
Multi-agent evaluation	×	✓	✓	✓	×	✓	✓
Mined for Interestingness	✓	×	-	✓	×	×	✓
Vector Map	✓	×	×	✓	✓	×	✓
Download Size	4.8 GB	-	22 GB	1.4 TB	48 GB	120 GB	58 GB
Public Leaderboard Entries†	194	-	935	23	18	3	-

As observed, Yandex is the dataset with the highest number of scenarios, total recorded time and amount of data, but there are very few entries and all of them dated from 2021, so, at the moment of writing this thesis, the research community is not focused in this dataset anymore. On the other hand, Lyft has the highest number of entries, but the number of unique roadways is limited to 10 km and it does not provide a vector map. Interaction is limited to some interesting scenarios, with very few unique tracks and roadways compared to the other datasets. Overall, these datasets have common goals of enabling motion prediction and improving the safety and efficiency of ADSs. However, their coverage, data size, features, and limitations vary, which makes it important to consider the specific requirements and use cases when choosing a dataset for research or development purposes.

Regarding this, we conclude the best vehicle MP datasets in the literature are NuScenes, Waymo, Argoverse 1 and Argoverse 2. In that sense, since the DL part of the thesis was started (around 2021) approximately when Argoverse 1 had a lot of interest from the research community, and, on top of that, they recently released Argoverse 2 ()the largest MP dataset to date) we finally decided to build our algorithms upon the Argoverse 1 and Argoverse 2 datasets.

2.4.1. Argoverse 1 Motion Forecasting

The Argoverse 1 Motion Forecasting dataset is a widely used dataset in the field of AD for studying and developing algorithms related to MP. It includes HD maps and a detailed map API to get the corresponding rasterized or vector information of the map.

This dataset is a curated collection of 324,557 scenarios (particularly, 205942 training samples, 39472 validation samples and 78143 test samples). Data was sampled at 10 Hz, where each sample contains the BEV position (x,y) of all agents in the scene in the past 2s (20 observed points), the local map, and the labels are the 3s (30 predicted points) future positions of one target agent in the scene. For training and validation, full 5-second trajectories are provided, while for testing, only the first 2 seconds trajectories are given.

Table 2.3: Distribution of the Target Agent Maneuver in the Argoverse 1 Motion Forecasting Dataset
Source: *Improving diversity of multiple trajectory prediction based on map-adaptive lane loss* [57]

Maneuver	Training	Validation
Going straight	191024 (92.75%)	34958 (90.70%)
Left turn	7860 (3.82%)	1880 (4.88%)
Right turn	4757 (2.31%)	1238 (3.21%)
Left lane change	1084 (0.53%)	284 (0.74%)
Right lane change	1217 (0.59%)	184 (0.48%)

Table 2.3 shows an estimated distribution of the target agent maneuver in the Argoverse 1 dataset, regarding the most common use cases in urban scenarios, such as going straight, turn and lane change. We can observe how most sequences are focused on keeping the same lane. Nevertheless, most of these *going straight* use cases do not conduct a trivial constant velocity trajectory, but there are sudden accelerations or breaks, which are quite interesting and challenging.

We have used this dataset to train and validate our algorithms in Chapters 5, 6.

2.4.2. Argoverse 2 Motion Forecasting

We have used the Argoverse 2 [6] Motion Forecasting dataset to implement and validate our final MP included in Chapter 7. In the same way than Argoverse 1, Argoverse 2 is a high-quality MP dataset where the real driving scenario are paired with the corresponding local HD map. Nevertheless, while Argoverse 1 provides a substantial amount of labeled data, it may still have limitations in capturing the full diversity of real-world driving scenarios. Built upon the success of Argoverse 1, the Argoverse 2 Motion Forecasting dataset provides an updated set of prediction scenarios collected from a self-driving fleet, improving its previous version by spanning +2000 km over six different cities and the traffic scenarios approximately twice longer and more diverse.

The design decisions [6] to create the Argoverse 2 dataset as a noticeable enhanced version of Argoverse 1 were:

- **Motion forecasting is a safety critical system in a long-tailed domain**: A good dataset must be biased towards diverse, interesting and challenging scenarios containing different types of focal agents. The Argoverse 2 goal is to encourage the development of methods that ensure safety during tail events, rather than to optimize the expected performance on easy miles where even simple models are able to solve the problem.

- **Goldilocks zone of task difficulty**: The number and diversity of methods performing at or near the SOTA continues growing since early 2020 when Argoverse 1 was released, but the performance on the test set (leaderboard) has begun to plateau, also referred in the literature as goldilocks zone.

  In that sense, Argoverse 2 is designed to increase prediction difficulty incrementally, spurring productive focused research for the next few years. These changes are intended to encourage methods that perform well on extended forecast horizons (3 s $\rightarrow$ 6 s), handle multiple types of dynamic objects (1 $\rightarrow$ 5), and ensure safety in scenarios from the long tail. Future Argoverse releases could continue to increase the problem difficulty by reducing observation windows and increasing forecasting horizons.

- **Usability matters**: Argoverse 1 benefited from a large and active research community in large part due to the simplicity of setup and usage. Existing Argoverse models can be easily ported to run on Argoverse 2. In particular, the Argoverse 2 have prioritized intuitive access to map elements, encouraging methods which use the lane graph as a strong prior. To improve training and generalization, all poses have also been interpolated and resampled at exactly 10 Hz (Argoverse 1 was approximate). In that sense, Argoverse 2 includes fewer, but longer and more complex scenarios. This ensures that total dataset size remains large enough to train complex models but small enough to be easily downloadable and manageable.

2.4.3. Evaluation metrics

Most MP datasets (either in the field of AD or others focused on pedestrian motion prediction) use the same metrics to evaluate the performance of the different proposed algorithms. In this work we focus on the most important ones: minimum Average Displacement Error (minADE) and minimum Final Displacement Error (minFDE) (both in the uni-modal and multi-modal scenario) to evaluate our models with respect to the SOTA both in terms of validation and tests sets.

1. The minimum Average Displacement Error (minADE, also referred as $ADE_{K=N}$) measures the average $\mathcal{L}_2$ distance between the best predicted trajectory and the ground-truth trajectory over all time steps. The best here refers to the trajectory (or mode) that has the minimum average error. It is defined as:

$$minADE = \min_{i=1}^{K=N} \left(\frac{1}{T} \sum_{t=1}^{T} \sqrt{\left(x_{i,t}^{pred} - x_{i,t}^{gt}\right)^2 + \left(y_{i,t}^{pred} - y_{i,t}^{gt}\right)^2} \right)$$

where $K = N$ is the total number of predictions or modes, T is the number of time steps, $(x_{i,t}^{pred}, y_{i,t}^{pred})$ are the predicted coordinates of vehicle i at time step t, and $(x_{i,t}^{gt}, y_{i,t}^{gt})$ are the ground-truth coordinates of vehicle i at time step t. The minADE metric penalizes the algorithm for the worst average displacement error among all the predictions.

2. The minimum Final Displacement Error (minFDE, also referred as $FDE_{K=N}$) measures the minimum $\mathcal{L}_2$ distance between the final predicted position and the corresponding ground-truth position. The best here refers to the trajectory (or mode) that has the minimum average error. It is defined as:

$$minFDE = \min_{i=1}^{K=N} \sqrt{\left(x_{i,T}^{pred} - x_{i,T}^{gt}\right)^2 + \left(y_{i,T}^{pred} - y_{i,T}^{gt}\right)^2}$$

where $(x_{i,T}^{pred}, y_{i,T}^{pred})$ are the predicted coordinates of vehicle i at the last time step T, and $(x_{i,T}^{gt}, y_{i,T}^{gt})$ are the ground truth coordinates of vehicle i at the last time step T. The minFDE metric penalizes the algorithm for the worst final displacement error among all the predictions.

In this work, except for the GAN-based model (Chapter 5) which is focused on uni-modal prediction, we report results for $K = 1$ (uni-modal case, only the mode with the best confidence is considered) and $K = 6$ as this is the standard multi-modal in the Argoverse 1 and 2 Motion Forecasting datasets in order to compare with other models.

2.5. Comparison of *state-of-the-art* simulators in Autonomous Driving

The last section of this Chapter focuses on justifying the necessity of why a hyper-realistic simulator is required to validate the algorithms instead of only using the graphics and metrics calculated on the corresponding datasets. We aim to integrate the best proposal of the thesis with other upstream and downstream modules developed in our research group (specially the MOT and DM modules) to validate the influence of the prediction step in a holistic way.

In order to validate a whole ADS the system must be tested in countless environments and scenarios, which would escalate the cost and development time exponentially with the physical approach. Considering this, the use of photo-realistic simulation (virtual development and validation testing) and an appropriate design of the driving scenarios are the current keys to build safe and robust AD technology. These simulators have evolved from merely simulating vehicle dynamics to also simulating more complex functionalities. Simulators intended to be used for testing AD technology must have requirements that extend from simulating physical car models to several sensor models, path planning, control and so forth and so on. Some SOTA simulators [58] are as following:

- **CarSim** [59] is a vehicle simulator commonly used by academia and industry. Its newest version supports moving objects and sensors that benefit simulations involving self-driving tecnology and ADAS. These moving objects may be linked to 3D objects with their own embedded animations, such as vehicles, cyclists or pedestrians.

- **PreScan** [60] provides a simulator framework to design self-driving cars and ADAS. It presents PreScan's automatic traffic generator which enables manufacturers to validate their autonomous navigation architectures providing a variety of realistic environments and traffic conditions. This simulator also supports HIL simulation, quite common for evaluating Electronic Control Units (ECUs) used in real-world applications.

- **CAr Learning to Act simulator (CARLA)** [61] an open-source autonomous driving simulator implemented as a layer over Unreal Engine 4 (UE4) [62]. This simulation engine provides to CARLA an ecosystem of interoperable plugins, a realistic physics and a state-of-the-art image quality. CARLA is designed as a server-client system so as to support this functionality provided by UE4, where the simulation is rendered and run by the server. The environment is composed of 3D models of static objects, such as buildings, infrastructure or vegetation, as well as dynamic objects like pedestrians, cyclists or vehicles. These objects are designed using low-weight geometric textures and models though maintaining visual realism by making use of variable level of detail and carefully crafting the materials. Moreover, one of the main advantages when using CARLA is the possibility to modify in an easy way the vehicle on-board sensors and their features in order to obtain accurate data, the weather and even the possibility to create realistic traffic scenarios.

- **Gazebo** [63] is an scalable, open-source, flexible and multi-robot 3D simulator. It supports the recreation of both outdoor and indoor environments in which there are two core elements that define the 3D scene, also known as world and model. The world is used to represent the 3D scene, defined in a Simulation Description File (SDF) and a model is basically any 3D object. Gazebo uses Open Dynamic Engine (ODE) as its default physic engine.

- **LGSVL (LG Electronics America R&D Center)** [64] is the most recent simulator for testing autonomous driving technology, focused on multi-robot simulation. It is based on the Unity game engine [65], providing different bridges for message passing between the simulator backbone and the autonomous driving stack. LGSVL provides a PythonAPI to control different environment entitities, such as weather conditions, the position of the adversaries, etc. in a similar way to the CARLA simulator. It also provides Functional Mockup Interface (FMI) so as to integrate vehicle dynamics platform to the external third party dynamics models.

In order to choose the right simulator, there is a set of criteria [58] that may serve as a metric to identify which simulators are most suitable for our purposes. In our case we have selected such as perception (sensors and weather conditions), multi-view geometry, traffic infrastructure, vehicle control, traffic scenario simulation, 3D virtual environment, 2D/3D groundtruth, scalability via a server multi-client architecture and last but not the least, if the simulator is open-source. In Table 2.4, we provide a comparison summary where all five simulators aforementioned are further compared. For further details about this AD comparison, we refer the reader to [58].

Table 2.4: Comparison of some *state-of-the-art* simulators for AD. GT stands for Ground-Truth. ✓ and ✗ indicate that the corresponding requirement is supported or not respectively. U = Unknown, TL = Traffic Light, SS = Stop Signal, INT = Intersections, IN = Indoor, OUT = Outdoor, ROS = Robot Operating System. MCA = Multi-Client Architecture. Hyphens "-" indicate that attributes are either not applicable, or not available.

Requirements	**CarSim**	**PreScan**	**CARLA**	**Gazebo**	**LGSVL**
Sensor models supported	✓	✓	✓	✓	✓
Different weather conditions	✗	✓	✓	✗	✓
Camera Calibration	✗	✓	✓	✗	✗
Path Planning	✓	✓	✓	✓	✓
Proper vehicle control dynamics	✓	✓	✓	✓	✓
3D Virtual Environment	✓	✓	✓, OUT (Urban)	✓, IN & OUT	✓, OUT (Urban)
Traffic Infrastructure	✓	✓	✓ (including TLs, INTs, SSs, lanes)	✓ (including TLs, INTs, SSs, lanes)	✓
Simulate different dynamic objects:	✓	✓	✓	✗	✓
2D/3D ground-truth	✗	✗	✓	-	✓
Interfaces to other software	✓, with MATLAB	✓, with MATLAB	✓, with ROS, Autoware	✓, with ROS	✓, with Autoware, Apollo, ROS
Scalability via a server MCA	-	-	✓	✓	✓
Open Source	✗	✗	✓	✓	✓
Stability	✓	✓	✓	✓	✓
Portability	✓	✓	✓, Windows & Linux	✓, Windows & Linux	✓, Windows & Linux
Flexible API	✓	-	✓	✓	✓

In that sense, we identify that CarSim is usually connected to MATLAB/Simulink [66] to simulate simple scenarios, with efficient plot functions and computation, where the user can control the vehicle models from CarSim and build their upper control algorithms in MATLAB/Simulink to do a co-simulation project, but the realism, the quality of the sensors and the complexity is limited. PreScan presents better capabilities to build realistic environments and simulate different weather conditions, unlike MATLAB and CarSim. Gazebo is quite popular as a robotic simulator, but the effort and time needed to create complex and dynamic scenes does not make it the first choice for testing self-driving technology.

To this end, we have two simulators as our final options: LGSVL and CARLA. At the moment of writing this thesis, they are the most suited simulators for end-to-end testing of unique functionalities offered by autonomous vehicles, such as perception, mapping, vehicle control or localization. Most of their features, summarized in [58] are identical (open-source, traffic generation simulation, portability, 2D/3D groundtruth, flexible API and so forth and so on), with the only difference that LGSVL does not present camera calibration to perform multi-view geometry or Simultaneous Localization and Mapping

Figure 2.6: CARLA simulator overview

(SLAM). Regarding this, we decided to use the CARLA simulator since the performance is very similar to LGSVL and the group had previous experience in the use of this simulator.

2.6. Summary

Table 2.5: Summary of Motion Prediction methods features. Short-term and long-term characterize prediction horizons of no more than 1-s and no less than 3-s, respectively.

Methods	Accuracy	Prediction Horizon	Computation Cost	Applications
Physics-based	High in short-term prediction, low in other prediction horizon	Short	Small	Colision risk analysis
Classic Machine Learning-based	Good at recognizing maneuvers but generalization ability is poor	Medium	Medium	Maneuver recognition
Reinforcement Learning-based	Relatively high, prediction methods are relatively few	Long	High	More applied in planning
Deep Learning-based	High in considering some factors	Long	Relatively high	More and more applied in real-world

In order to finish this Chapter, we perform a brief comparison between the different methods (Physics-based, Classic ML, RL and DL) in terms of accuracy, prediction horizon, computation cost and applications in the AD field.

- Physics-Based Methods are suitable for the movement of vehicles, which can be accurately described by kinematics or dynamics models. Given a suitable physics model, these methods can be applied to a variety of scenarios at small computational cost and in a short time but without training. However, the prediction results based on such models heavily depends on the inputs and the model selection. The inputs are closely related to human or machine drivers, influenced by the driving environment or the interactions with other participants. Therefore, without the capability to describe such factors, physics-based models are limited to short-term prediction and in static scenes. Because of its simplicity and fast response, these methods can be easily used in real applications for ADSs, such as collision risk analysis.

- Classic ML-Based Methods, compared with physics-based methods, are able to consider more factors and its accuracy is relatively high with a longer prediction length at a higher computing cost. Most of these methods are maneuver-based methods, which predicts the trajectory with the maneuver known as a prior. However, vehicle maneuvers of human drivers are usually diverse and vary greatly in different scenarios such that the generalization ability of this approach is poor. In real applications for ADSs, such methods are used in scenarios such as lane change studies, leveraging their advantages in maneuver recognition.
- RL-Based Methods imitate the human DM process and obtain the reward function through learning the expert demonstration to generate the corresponding optimal driving policy. They can continuously evolve through learning and adapt to complex environments and long prediction horizons. Such methods probably generate higher accuracy trajectories than DL methods in a longer time domain. However, most of these methods are typically computationally expensive in their recovery of an expert cost function, require long training times and its training is based on actions and episodes, in such a way its use in real-world MP applications is not suitable, being RL more applied to trajectory planning, taking its advantages in the DM process.
- DL-Based Methods can perform accurate predictions in a longer time horizon with respect to traditional methods that are only suitable for simple scenes and short-term prediction. By means of powerful neural networks, such as RNNs, CNNs, GANs, Attention mechanisms or GNNs for feature extraction, physics-related, interaction-related and road-related factors are processed as inputs to the model. Furthermore, they can adapt to more complex environments and a longer prediction horizon. DL-based methods require to use a large amount of data for training.

  Besides, with the increase of consideration factors and the increase of the number of network layers, the computing costs and time increases sharply. Such methods can naturally generate multi-modal trajectories, which is consistent with the diversity of vehicles maneuvers. In real applications for ADS, it is necessary to reach a balance between calculation time and model complexity to ensure the real-time performance and safety of ADS. At present, more and more real-world trials use these methods to predict the future trajectory of traffic participants.

As observed in Table 2.5 and discussed in this section summarizing the different MP algorithms, we focus this thesis on DL methods since they are the most suitable methods for long-term prediction with a lower computation cost than RL-based approaches, specially focusing on the ability of extracting and combining the latent spaces of the different inputs by means of SOTA algorithms. Then, they are more suitable to model complex real-world applications, which is the final objective of this thesis.

Furthermore, the validation framework (both in terms of datasets and AD simulator) is defined. In this thesis we choose Argoverse 1 and Argoverse 2 as databases to build our

DL-based MP algorithms, since both are extensively used by the research community to build this kind of algorithms, and CARLA to validate the prediction methods in a holistic way (that is, integrated with other layers such as DM or control) in a hyper-realistic AD simulator as a preliminary stage before implementing it in a real-world platform.

Chapter 3

Theoretical Background

Desde que el mundo cambió,
estamos mucho más unidos con los Digimon,
luchamos juntos contra el mal.
Algo extraño pasaba,
Digievolucionaban, en tamaño y color,
Ellos son los Digimon.

Opening 1 de Digimon: "Butterfly"
Autor original: Kōji Wada

3.1. Introduction

As commented in previous Chapters, the MP algorithms covered by this thesis range from tracking multiple objects and subsequent prediction with physics-based methods to the most recent SOTA techniques to compute the deep traffic context and then decode multi-modal predictions with associated confidences, assuming the physical information is given and surrounding participants have been multi-tracked beforehand. Throughout this Chapter, an in-depth theoretical study will be made of those algorithms, neural networks or heuristics that form the foundations of this work in order to address the proposed methods in future chapters.

First of all, we will start with the mathematical formulation of the methods to perform physics-based Multi-Object Tracking, such as the well-known Kalman Filter (KF) algorithm [23] for agent state estimation and the Hungarian Algorithm (HA) [67] for the association of detections and trackers, which represents the preliminary stage before carrying out the subsequent physics-based uni-modal prediction. On top of that, since several single-trajectory models (CTRV, CTRA) are used to compute the most plausible centerlines in the Argoverse 1 [5] and Argoverse 2 [6] datasets, we will review the state transition equations to properly understand the constraints for each model. Furthermore, the principal DL techniques (*e.g.* 1D-CNN, LSTM, GAN, Attention mechanisms or GCN) and training losses used in this work to encode and decode the aimed multi-modal predictions will be stated, first a general mathematical formulation and applications, and then, how the corresponding technique is used in the MP field.

3.2. Physics-based algorithms

As stated in Chapter 2, in terms of MP, initially researchers rely on physics-based methods which are basic and straightforward. These methods may not offer high accuracy, but many models use the underlying idea of physics-based models to improve their accuracy. Physics-based approaches yield better results when the movement of vehicles is described by kinematics or dynamics models accurately. Nevertheless, the physical model of traffic participants is constantly evolving, so most physics-based models are only applicable for short-term predictions of no more than one second. In this Section we will study the mathematical formulation of the physics-based prediction algorithms, as well as the data association problem regarding the Multi-Object Tracking stage, that are directly related to our algorithm proposed in Chapter 4.

3.2.1. Kalman Filter under the hood

The Kalman Filter (KF) [23] is a recursive algorithm used for estimating the state of a dynamic system in the presence of noise. It is widely used in various fields such as engineering, control systems, and robotics. The algorithm works by combining a prediction of the system state based on a mathematical model with measurements (updates) from sensors to improve the accuracy of the state estimation, as shown in Figure 3.1. The Kalman Filter is a powerful algorithm for state estimation, and it has many variations and extensions that can handle various types of systems and measurements. The filter can be applied to non-linear systems using the Extended Kalman Filter (EKF) or the Unscented Kalman Filter (UKF), and it can handle multiple models using the multiple model Kalman Filter. The filter can also be used for smoothing, which involves estimating the state of the system based on past and future measurements.

In its most simple version, the KF assumes that the system can be described using a set of linear equations, where the state of the system can be represented by a vector $\mathbf{x}_k$, and the measurements can be represented by a vector $\mathbf{z}_k$. The KF also assumes that the noise in the system follows a Gaussian distribution.

The state estimation process is performed in two steps: the prediction step and the update step. In the prediction step, the current state of the system is predicted based on the previous state and a mathematical model of the system. In the update step, the predicted state is corrected based on a measurement from the sensors.

The prediction step can be represented using the following equations:

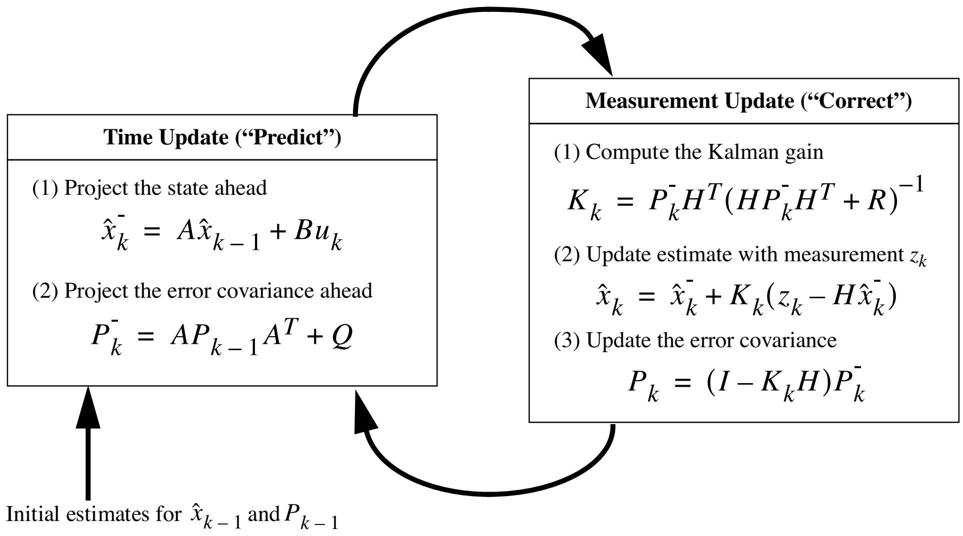


Figure 3.1: Overview of the Kalman Filter Predict-Update cycle. The Kalman Filter keeps track of the estimated state of the system and the variance or uncertainty of the estimate. The estimate is updated using a state transition model and measurements.
Source: *An introduction to the kalman filter* [68]

$$\begin{aligned} \hat{x}_k^- &= A\hat{x}_{k-1} + Bu_k \\ P_k^- &= AP_{k-1}A^T + Q \end{aligned} \tag{3.1}$$

where $\hat{x}_k^-$ is the predicted state of the system at time k, $\hat{x}_{k-1}$ is the estimated state of the system at time $k-1$, A is the state transition matrix, B is the control-input matrix, u_k is the control input at time k, P_k^- is the predicted error covariance matrix, and Q is the process noise covariance matrix. Note that the state transition, control-input and process noise covariance matrix values do not depend on the time-step time k.

On the other hand, the update step can be represented using the following equations:

$$\begin{aligned} K_k &= P_k^- H_k^T (HP_k^- H^T + R)^{-1} \\ \hat{x}_k &= \hat{x}_k^- + K_k(z_k - H_k\hat{x}_k^-) \\ P_k &= (I - K_k H)P_k^- \end{aligned} \tag{3.2}$$

where K_k is the KF gain, H is the measurement matrix, R_k is the measurement noise covariance matrix and I the identity matrix of the corresponding dimension. The KF gain determines the relative weight given to the predicted state and the measurement, and is adjusted based on the measurement noise covariance matrix. The measurement matrix relates the measurements to the state variables, and is used to convert the measurements into the same units as the state variables.

3.2.2. Hungarian algorithm formulation

The Hungarian Algorithm (HA) [67], also known as the Kuhn-Munkres algorithm, is an efficient algorithm for solving the assignment problem in combinatorial optimization. An example to illustrate the HA solution for an assignment problem involves finding the optimal assignment of n workers to n jobs, given the cost of assigning each worker to each job.

The HA works by iteratively finding a set of independent zero-cost assignments, which correspond to a perfect matching in a bipartite graph. These assignments are then used to reduce the problem size and find a new set of independent zero-cost assignments, until all workers are assigned to jobs.

This algorithm has a time complexity of $O(n^3)$, which makes it one of the most efficient algorithms for solving the assignment problem. In this algorithm, the cost matrix C is assumed to be an $n \times n$ matrix, where each element $C_{i,j}$ represents the cost of assigning worker i to job j. The matrix M represents the matching, where each element $M_{i,j}$ is 1 if worker i is assigned to job j, and 0 otherwise.

Algorithm 1: The Hungarian algorithm for solving the minimum cost perfect matching problem

```
Input  : A cost matrix C of size n × n
Output: A minimum cost perfect matching
Initialize the label vectors u_i = min_{j∈1,...,n} C_{i,j} and v_j = 0 for all i, j; Initialize the empty
 matching M; while |M| < n do
    Choose an unmatched row i; Initialize the set T = i and the predecessor vector P = ∅; while
     true do
        Let S be the set of columns j such that i ∈ T and C_{i,j} = u_i + v_j; If |S| > 0, choose any
         column j in S; else
            Choose a column j such that v_j = min_{k∈1,...,n} v_k and let S be the set of rows i such
             that j ∈ M or C_{i,j} = u_i + v_j; Increment each u_i for i ∈ T by
             δ = min_{i∈T,j∈S}(u_i + v_j − C_{i,j}); Decrement each v_j for j ∈ S by δ; for each row i ∈ T
             and each column j ∈ S do
                if C_{i,j} = u_i + v_j then
                    Add the edge (i, j) to the alternating tree represented by M and P; if j is
                     unmatched then
                        Augment M along the alternating tree to create a larger matching; return
                         the updated matching M;
                    end
                    else
                        Add j to T and continue the while loop;
                    end
                end
            end
        end
    end
end
```

As observed in Algorithm 1, the HA iteratively finds an uncovered zero in the cost matrix C and adds it to the matching M, while also adjusting the cost matrix to ensure that all rows and columns are covered by the matching. This is done by finding the minimum uncovered cost in each row and subtracting it from all uncovered costs in the row, and finding the minimum cost in each uncovered column and adding it to all uncovered costs in the column. Once all rows and columns are covered, the algorithm returns the final matching M.

In conclusion, the HA is a powerful optimization algorithm that solves the assignment problem in polynomial time. The algorithm has a wide range of applications and can be easily adapted to handle various constraints and objectives. The algorithm is also guaranteed to find the optimal solution to the assignment problem, which makes it a valuable tool in many practical settings.

3.2.3. State Transition Equations of Single-Trajectory Models

As illustrated in Chapter 2, Single-Trajectory prediction methods are used in the field of motion estimation and control, to predict the future state of an object based on its current state and motion. In these methods, the agents are mostly assumed to comply with motion models that describe their dynamic behavior in such a way these are not able to consider the road-related factors and the uncertainty of the current state is unreliable for long-term prediction. Then, these models should only be used for estimating uni-modal trajectories of the surrounding agents in the short-term (no more than 1-s).

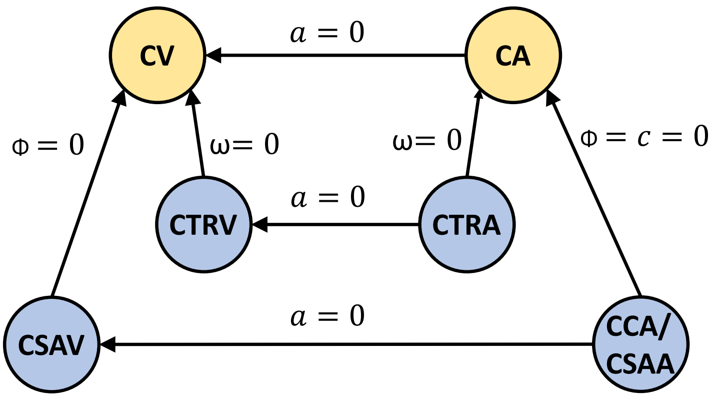


Figure 3.2: Overview of Single Trajectory prediction methods. Every sophisticated model can be transformed into a simpler one by setting one state veriable to zero. a, ϕ, c, ω represent the agent acceleration, steering angle, curvature and yaw rate (angular velocity) respectively
Source: *Comparison and evaluation of advanced motion models for vehicle tracking* [69]

Several single-trajectory prediction models have been proposed in the literature, as illustrated in Figure 3.2:

- Constant Velocity (CV) model assumes that the object moves at a constant velocity and its future position can be predicted by simply extrapolating its current velocity vector. This model is simple and computationally efficient, but it may not accurately capture the object motion if it changes its velocity.
- Constant Acceleration (CA) model is an extension of the CV model, assuming that the object can also change its acceleration in a linear manner. This model can provide

more accurate predictions of the object future position and velocity, but it may not be suitable for objects with more complex motion patterns.

- Constant Turn Rate and Velocity (CTRV) model assumes that the object moves in a circular trajectory with a constant turn rate and velocity magnitude. This model is useful for predicting the motion of objects such as drones or missiles, but it may not accurately represent the motion of objects with more complex trajectories.
- Constant Turn Rate and Acceleration (CTRA) model is an extension of the CTRV model, allowing the object to change its acceleration while maintaining a constant turn rate. This model is often used in applications such as autonomous driving or aircraft navigation, where objects can accelerate or decelerate.
- Constant Steering Angle and Velocity (CSAV) model assumes that the object moves along a straight line with a constant speed and acceleration vector. This model can be useful for predicting the motion of objects such as trains or boats.
- Constant Curvature and Acceleration (CCA) model assumes that the object moves along a constant heading angle and can change its acceleration in a linear manner, also referred in the literature as Constant Steering Angle and Acceleration (CSAA) if the steering angle is used as a state variable instead of the curvature. From an algorithmic point of view, however, both names refer to the same model, which can be useful for predicting the motion of objects such as bicycles or motorcycles.

It can be observed how a sophisticated model can be transformed into a simpler one by setting one state variable to zero. For example, CTRA is transformed to CTRV if the agent acceleration a is set to zero. Similarly, CTRA is transformed to CA if the angular velocity ω is set to 0. In this Section we particularly focus on the prediction equations and differences of the most commonly used single-trajectory prediction methods in the field of AD: CTRV and CTRA.

3.2.3.1. Constant Turn Rate Velocity (CTRV)

The Constant Turn Rate and Velocity (CTRV) model is a mathematical model used in the field of autonomous navigation to predict the future motion of a moving object. The model assumes that the agent moves along a smooth, continuous path, *i.e.* the velocity and turn rate remain constant throughout the prediction interval.

With this model we are able to describe and predict the motion of the object using a set of differential equations that relate the rate of change of the agent state to its current state and any external inputs. It uses as main variables the current position, velocity, heading angle, and turn rate, and propagating them forward in time.

The prediction equations for the CTRV model are described as follows:

$$
\begin{aligned}
x_{k+1} &= x_k + \frac{v_k}{\omega_k}\left[\sin(\psi_k + \omega_k \Delta t) - \sin(\psi_k)\right] \\
y_{k+1} &= y_k + \frac{v_k}{\omega_k}\left[\cos(\psi_k) - \cos(\psi_k + \omega_k \Delta t)\right] \\
v_{k+1} &= v_k \\
\psi_{k+1} &= \psi_k + \omega_k \Delta t \\
\omega_{k+1} &= \omega_k
\end{aligned}
\tag{3.3}
$$

where x_k and y_k are the coordinates of the agent position, v_k is the velocity magnitude, ψ_k is the heading angle (in radians), ω_k is the turn rate (in radians per second) and Δt is the time step size at timestep k respectively.

The first two equations predict the agent position at the next time step, based on its current position, velocity, heading angle, and turn rate. The third and fifth equations predict that the velocity and turn rate will remain constant. The fourth equation predicts the object heading angle at the next time step, based on its current heading angle and turn rate.

3.2.3.2. Constant Turn Rate Acceleration (CTRA)

The Constant Turn Rate and Acceleration (CTRA) model is an extension of the Constant Turn Rate and Velocity (CTRV) model, allowing the agent to change its acceleration while maintaining a constant turn rate.

The motion of the CTRA agent is modeled using the following set of equations:

$$
\begin{aligned}
x_{k+1} &= x_k + \frac{v_k}{\omega_k}\left[\sin(\theta_k + \omega_k \Delta t) - \sin(\theta_k)\right] \\
y_{k+1} &= y_k - \frac{v_k}{\omega_k}\left[\cos(\theta_k + \omega_k \Delta t) - \cos(\theta_k)\right] \\
\theta_{k+1} &= \theta_k + \omega_k \Delta t \\
v_{k+1} &= v_k + a_k \Delta t \\
\omega_{k+1} &= \omega_k
\end{aligned}
\tag{3.4}
$$

where the variables are the same than the CTRV but at this point including the heading angle θ_k and agent acceleration a_k at timestep k.

The first two equations describe the agent position updates based on its current position, velocity, heading angle, turn rate, and the time step. The third equation describes the agent heading angle update based on its current heading angle and turn rate. The fourth equation describes the agent velocity update based on its current velocity and acceleration. The last equation assumes that the agent turn rate remains constant over time.

As observed, the prediction equations for the CTRV model are based on the kinematic equations for circular motion, where the object position, velocity, and heading angle change in a continuous and smooth manner. In contrast, the prediction equations for the CTRA model include an additional term for the object acceleration, which allows for more accurate predictions of the object future motion when it can accelerate or decelerate.

In terms of implementation, the CTRA model requires more computational resources due to the additional term for acceleration, which increases the complexity of the equations. This can make the CTRV model more computationally efficient and faster to calculate in real-time applications.

3.3. Deep Learning algorithms

In this section the mathematical formulation of the DL layers employed in this thesis is covered. In particular, these are the 1D-CNN, LSTM, conditional Generative Adversarial Network (cGAN), Attention mechanism and GCN layers. We briefly introduce each layer, as well as the corresponding mathematical formulation to enhance the comprehension of the learning-based methods proposed in future Chapters.

3.3.1. Convolutional Neural Networks (CNNs)

CNNs are commonly used for image and signal processing tasks. The type of CNN architecture used depends on the nature of the input data. Here are the differences among the three main types of CNNs:

- **1D Convolutional Neural Networks (1D-CNNs)**: 1D-CNNs are used for processing sequential data such as time series data, speech signals or text data. The input is a one-dimensional sequence of data, such as a time series of sensor readings. 1D-CNNs typically have fewer parameters and are computationally efficient compared to 2D- and 3D-CNNs.
- **2D Convolutional Neural Networks (2D-CNNs)**: 2D-CNNs are used for processing 2D images. The input is a two-dimensional image represented as a matrix of pixels. The convolutional layers in a 2D-CNN apply filters that slide over the image to extract local features. The output of each convolutional layer is a set of 2D feature maps. 2D-CNNs are widely used for image classification, object detection, and image segmentation.
- **3D Convolutional Neural Networks (3D-CNNs)**: 3D-CNNs are used for processing 3D volumetric data such as CT scans, MRI images, and video data. The input

is a three-dimensional volume represented as a sequence of 2D images. The convolutional layers in a 3D-CNN apply filters that slide over the 3D volume to extract local features. The output of each convolutional layer is a set of 3D feature maps. 3D-CNNs are used for tasks such as medical image analysis, video classification, 3D object detection and action recognition.

As we will see in further Chapters, in this work we focus on 1D-CNNs. The basic building blocks of a 1D CNN are convolutional layers, which learn local patterns in the input data by applying a set of filters to it, as illustrated in Figure 3.3. Each filter slides over the input sequence and performs a dot product operation at each position to generate a feature map. These feature maps capture the presence of certain patterns at different positions in the input sequence.

The architecture of a 1D-CNN typically consists of several convolutional layers, followed by one or more fully connected layers for classification or regression. The output of each convolutional layer is fed into the next layer, with optional pooling layers in between to reduce the spatial dimension of the feature maps, which are finally flattened before the final layer. The output of the network is obtained by passing the output of the last fully connected layer through a suitable activation function, such as softmax for classification or linear for regression.

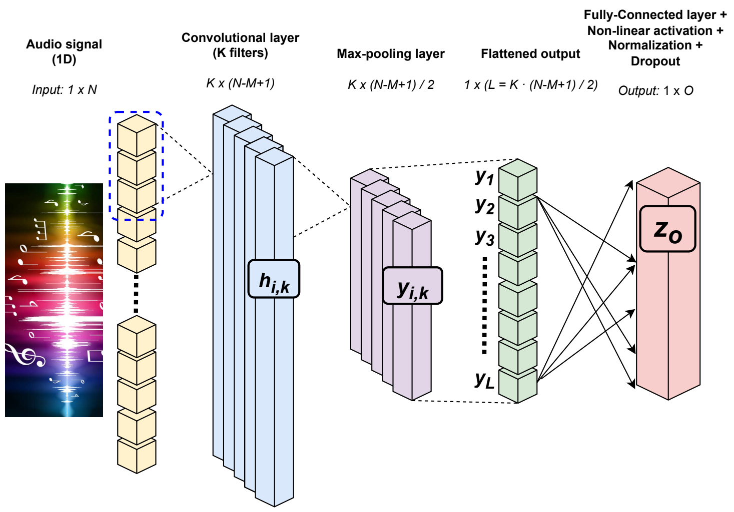


Figure 3.3: Example of a Convolutional Neural Network (CNN) architecture to process 1D-input signals

Let x be a 1-dimensional input sequence of length N, represented as a vector $x = [x_1, x_2, ..., x_N]$. Let K be a set of filters of length M, represented as a vector $k = [k_1, k_2, ..., k_M]$. We can apply the filter k to the input sequence x by computing a convolution operation:

$$(k * x)_i = \sum_{j=1}^{M} k_j x_{i+j-M} \tag{3.5}$$

where $*$ denotes the convolution operation, and i ranges from M to N. This operation produces a new sequence of length $N - M + 1$, representing the local features extracted from the input sequence x by the filter k.

To extract different types of features, we apply K filters in each convolutional layers in the network. Then, each layer applies a set of filters, and produces a set of feature maps. The feature maps can be computed as:

$$h_{i,k} = f(\sum_{k=1}^{M} W_{j,k} x_{i+k-M} + b_j) \tag{3.6}$$

where h is the feature map at position i and filter k, W is the weight matrix of the k-th filter, b is the bias term for the k-th filter, and f is an activation function (such as ReLU or sigmoid). Note that in Figure 3.3 a bench of $K = 4$ filters is applied in the convolutional layer.

After each convolutional layer, we typically apply a pooling layer to reduce the dimensionality of the features and increase the translation invariance of the network. The most common pooling operation is max pooling, which selects the maximum value within a window of size p:

$$y_{i,k} = \max_{j=0}^{p-1} h_{i+k,j} \tag{3.7}$$

where y is the output of the pooling layer at position i and filter k. After this operation, feature dimensionality is normally halved ($L = (N - M + 1)/2$), as observed in Figure 3.3. Once we have the corresponding features maps after the max pooling operation, we typically take the output of the pooling layer and flatten into a vector of L, and fed into a set of fully connected layers, with optional dropout regularization and batch normalization

Finally, we can use one or more fully connected layers to perform the final classification or regression task. The output of the pooling layer is flattened into a vector of length O, and fed into a set of fully connected layers, with optional dropout regularization and batch normalization:

$$z_o = f(\sum_{o=1}^{O} W_{l,o} y_l + b_o) \tag{3.8}$$

where z_o is the o-th element of the output vector with length O, W is the weight matrix, y is the flattened output of the pooling layer with length L, b is the bias term, and f is an activation function.

We make use of the 1D-CNN mechanism in Chapters 6 and 7, studying its effect in our algorithms, leveraging its wider receptive field compared with a MLP to reduce the influence of noisy input data.

3.3.2. Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs) are a type of neural network that are designed to process sequential data, where the order of the input matters, usually assuming the presence of discrete timestep. They are widely used in a variety of applications such as natural language processing, speech recognition, image captioning, and time series prediction.

The general theory of RNNs is that they have a feedback loop in their architecture that allows information to be passed from one time step to the next, thereby enabling the network to capture dependencies between the inputs at different time steps. The input at each time step is fed into the network along with the output from the previous time step, and the network uses this information to generate a new output.

There are several types of RNNs, including the basic RNN, the Gated Recurrent Unit (GRU), and the Long Short-Term Memory (LSTM) [70]. The basic RNN is the simplest form of RNN, where the output at each time step is a function of the input at that time step and the output from the previous time step. However, it suffers from the vanishing gradient problem, where the gradients become exponentially small as they propagate through time, making it difficult to learn long-term dependencies.

The GRU is a variation of the basic RNN that uses gating mechanisms to control the flow of information through the network. It has fewer parameters than the LSTM and can be faster to train, but it may not perform as well as the LSTM on tasks that require more complex temporal dependencies.

3.3.2.1. Long Short-Term Memory

LSTM networks were introduced to address the vanishing gradient problem in standard RNNs. They achieve this by using a memory cell to store information over long periods of time, and three gating mechanisms to control the flow of information through the network. Figure 3.4 provides a detailed overview of the LSTM cell structure, illustrating the workflow from the previous cell output and input at time $t-1$ in order to compute the current cell state and hidden state at time t.

The three gates are called the input gate, forget gate, and output gate, and they are responsible for deciding which information to store in the memory cell, which information to forget, and which information to output, respectively.

The equations for the input, forget, and output gates are as follows:

- **Input gate**: $i_k = \sigma(W_i R_i \cdot [h_{k-1}, x_k] + b_i)$
- **Forget gate**: $f_k = \sigma(W_f R_f * [h_{k-1}, x_k] + b_f)$
- **Output gate**: $o_k = \sigma(W_o R_o * [h_{k-1}, x_k] + b_o)$

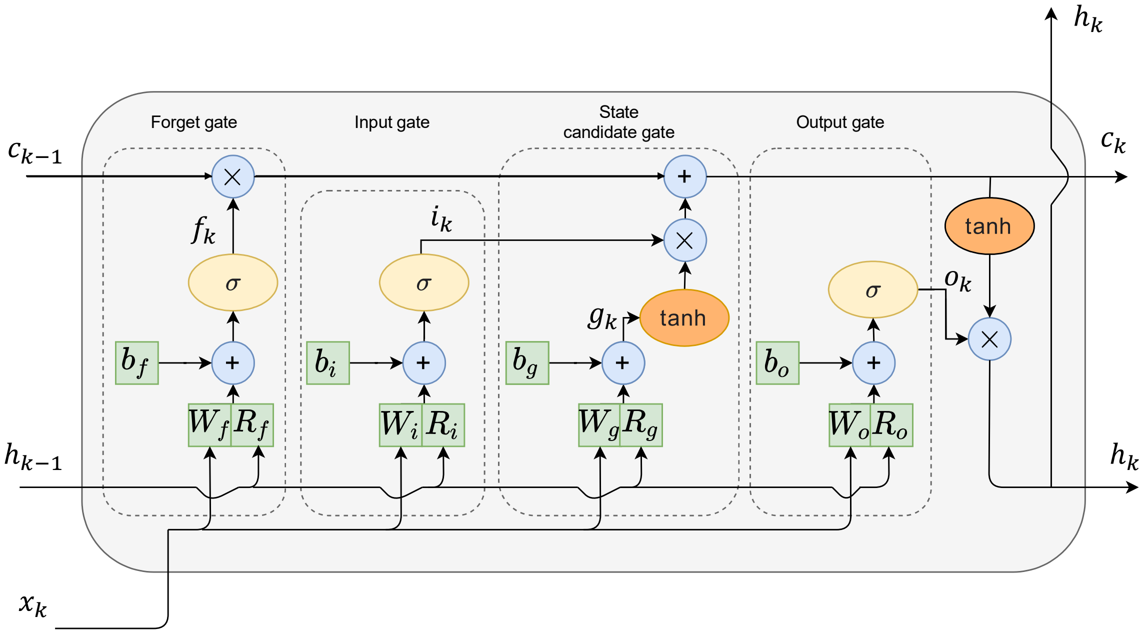


Figure 3.4: Overview of the LSTM cell structure

where x_k is the input at the current timestep k, h_{k-1} is the hidden state from the previous timestep, i_k is the input gate activation, f_k is the forget gate activation, and o_k is the output gate activation for the current timestep k respectively. On the other hand, $W_i R_i$, $W_f R_f$, and $W_o R_o$ are weight matrices (where the component W denotes the weights associated with the input signals x and R denotes the so-called recursive weights, associated with the hidden state of the cell from the previous timestep), and b_i, b_f, and b_o are bias vectors.

Moreover, the memory cell is updated based on the input, forget, and output gates, as well as a new candidate value that is computed based on the current input and hidden state. The equations for the memory cell and candidate value are as follows:

$$\begin{aligned} C_k &= f_k \cdot C_{k-1} + i_k \cdot \tilde{C}_k \\ \tilde{C}_k &= \tanh(W_g R_g \cdot [h_{k-1}, x_k] + b_g) \end{aligned} \tag{3.9}$$

where $\tilde{C}_k$ is the candidate value, C_t is the new memory cell value, and C_{k-1} is the previous memory cell value.

Finally, the hidden state at time t is computed based on the output gate and the new memory cell value, using the following equation:

$$h_k = o_k \cdot \tanh(C_k) \tag{3.10}$$

This equation scales the memory cell value by the output gate activation, then applies a hyperbolic tangent function to obtain the new hidden state.

In summary, LSTMs use a memory cell and three gating mechanisms to learn long-term dependencies in sequential data. The input, forget, and output gates control the flow of information through the network, while the memory cell stores information over time. The equations for LSTMs involve computing the input, forget, and output gate activations, the candidate value, the new memory cell value, and the new hidden state.

We make use of the LSTM network in Chapters 5 and 6 due to its powerful ability to represent as a high-dimensional space the past/future trajectories of the agents, though, as it will be seen in Section 3.3.4 of this Chapter, for the final proposal of the thesis we employ transformer-based modules since they are more powerful than LSTM and easier to train.

3.3.3. Generative Adversarial Networks (GANs)

A discriminative model is a type of machine learning model that learns the relationship between the input features and the target output directly. The goal of a discriminative model is to learn a decision boundary that separates different classes in the input data. In other words, discriminative models focus on learning the conditional probability distribution of the target variable given the input features.

Discriminative models are often used in supervised learning tasks, such as classification and regression, where the goal is to predict a target variable based on a set of input features. Common examples of discriminative models include logistic regression, support vector machines, and neural networks.

Unlike generative models, which learn the joint probability distribution of the input and target variables, discriminative models do not model the probability distribution of the input data explicitly. Instead, they focus on learning a mapping function that directly maps the input features to the target output.

In 2014, a breakthrough paper introduced Generative Adversarial Networks (GANs) [71], a clever new way to leverage the power of discriminative models to get good generative models. At their heart, GANs rely on the idea that a data generator is good if we cannot tell fake data apart from real data. In statistics, this is called a two-sample test, a test to answer the question whether datasets $X = \{x_1, \dots, x_n\}$ and $X' = \{x'_1, \dots, x'_n\}$ were drawn from the same distribution. This allows to improve the data generator until it generates something that resembles the real data. At the very least, it needs to fool the classifier even if our classifier is a state of the art deep neural network. GANs are a powerful technique for generating realistic samples that are similar to some training data. As observed in Figure 3.5, a GAN consists of two neural networks, a generator network and a discriminator network. The generator and discriminator networks are trained in a two-

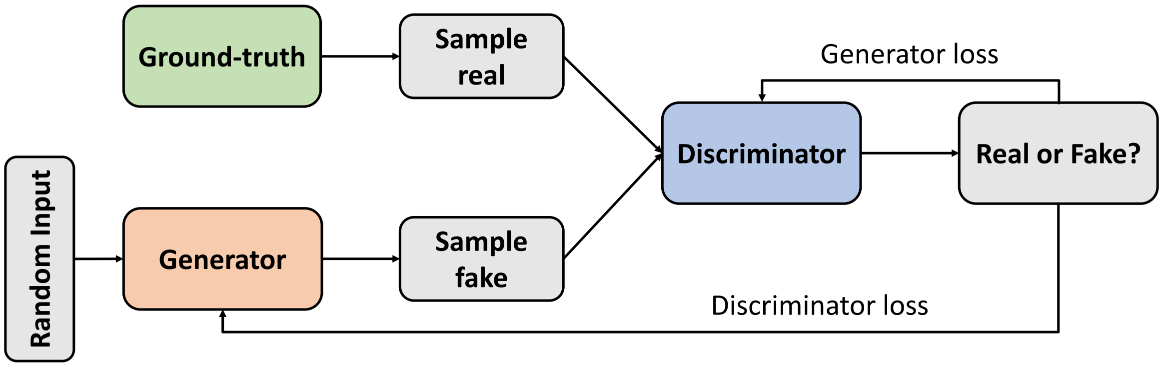


Figure 3.5: Overview of a Generative Adversarial Network (GAN) structure

player game, where the generator tries to produce samples that fool the discriminator, and the discriminator tries to correctly classify between real and generated data. The objective function for a GAN is a min-max game between the generator and discriminator, and can be optimized using gradient descent or some other optimization algorithm. Some common applications where GANs are widely used are generating realistic images, videos, audio, data augmentation or style transfer.

The generator network takes as input a random noise vector, and generates a sample that is meant to mimic real data. The discriminator network takes as input a sample, and tries to distinguish between real data and generated data. The two networks are trained in a two-player game, where the generator tries to produce samples that fool the discriminator, and the discriminator tries to correctly classify between real and generated data.

The objective function for a GAN can be formulated as a min-max game between the generator and discriminator. The generator tries to minimize the probability that the discriminator correctly classifies the generated samples as fake, while the discriminator tries to maximize the probability that it correctly classifies the real and generated samples.

The objective function for a GAN can be written as:

$$\begin{aligned}\min_{\theta_G}\max_{\theta_D} V(D,G) &= \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z)))] \\ &= \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z;\theta_G)))]\end{aligned} \tag{3.11}$$

where θ_G and θ_D are the parameters of the generator and discriminator networks, respectively, x is a real sample from the training data distribution p_{data}, z is a noise vector sampled from a prior distribution p_z, and $G(z;\theta_G)$ is the generated sample from the generator network.

The first term of the objective function represents the expected log-probability of the discriminator correctly classifying a real sample as real. The second term represents the expected log-probability of the discriminator correctly classifying a generated sample as fake.

During training, the generator network is updated to minimize the objective function with respect to θ_G, while the discriminator network is updated to maximize the objective function with respect to θ_D. This can be done using gradient descent or some other optimization algorithm.

There are some well-known types of GANs, such as:

- **Deep Convolutional GANs (DCGANs)**: These are GANs that use CNNs in the generator and discriminator networks. DCGANs are commonly used for image generation and have been shown to produce high-quality, realistic images.
- **Wasserstein GANs (WGANs)**: These are GANs that use the Wasserstein distance instead of the traditional Kullback-Leibler (KL) divergence or Jensen-Shannon (JS) divergence for measuring the difference between the real and generated distributions. WGANs have been shown to produce more stable training and generate higher-quality samples.
- **CycleGANs**: These are GANs that are designed for image-to-image translation tasks, where the goal is to transform an image from one domain to another. CycleGANs use two generators and two discriminators to learn the mapping between the two domains.

Nevertheless, in this work we focus on a particular type of GAN, known as conditional cGAN, where the generator is conditioned on some additional information, allowing for more specific generation of data.

3.3.3.1. GAN vs cGAN

The main difference between the original GAN and cGAN is that the original GAN generates samples without any control over the generated output, whereas the cGAN generates samples conditioned on some additional information.

In the original GAN, the generator network takes as input a random noise vector, and generates a sample that is meant to mimic real data. The discriminator network takes as input a sample, and tries to distinguish between real data and generated data. The goal of the original GAN is to train the generator network to generate samples that are indistinguishable from real data.

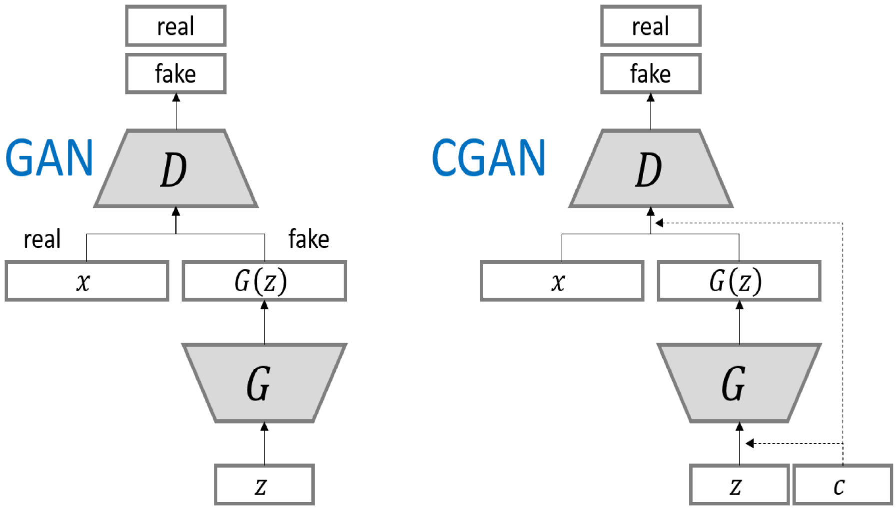


Figure 3.6: Comparison between the original GAN and conditional GAN (cGAN)
Source: *Logan: Generating logos with a generative adversarial neural network conditioned on color* [72]

In contrast, a cGAN is a GAN that is conditioned on some additional information, such as class labels or image annotations. The generator network in a cGAN takes as input both a random noise vector and a condition vector, and generates a sample that is conditioned on the additional information. The discriminator network also takes as input both the sample and the condition vector, and tries to distinguish between real data and generated data based on the condition. Figure 3.6 illustrates the differences between both architectures.

The cGAN can be used for a variety of applications, such as image-to-image translation, where the additional information is an input image that is translated to a different output image. For example, a cGAN can be trained to convert grayscale images to color images, or to transform images of one type of object to another type of object.

The training process for a GAN is similar to that of the original GAN, but the loss function for the generator and discriminator networks are modified to include the condition vector. Specifically, the loss function for the generator network includes a term that measures how well the generated samples match the condition vector, in addition to the term that measures how well the generated samples fool the discriminator. Similarly, the loss function for the discriminator network includes a term that measures how well the discriminator can identify the condition vector, in addition to the term that measures how well it can distinguish between real and generated data.

We make use a cGAN-based approach in Chapter 5 to compute plausible uni-modal predictions, where the input to the generative model (as we will see, represented by a LSTM network) is a noise vector z sampled from a multi-variate normal distribution and the physical and social represent the conditions to the model.

3.3.4. Attention Mechanism

The attention mechanism [73] is a computational method used in DL models to help the model focus on the most important parts of the input data. It is commonly used in Natural Language Processing (NLP), speech recognition, and computer vision. In terms of MP, the attention mechanism is usually employed to model interaction among entities (either social or physical) to compute the most important features once the corresponding encoders have previously computed a deep description of the scene, *i.e.* attention is commonly employed to extract relevant information from high-dimensional vectors (*e.g.* 64 or 128).

The basic idea of attention is to compute a set of weights that indicate the relative importance of different parts of the input. These weights are then used to compute a weighted sum of the input, which is used as the output of the attention mechanism. The attention mechanism can be formulated mathematically using the following equations:

- First, we compute a set of *keys*, *values*, and a *query* vector, which are used to compute the attention weights:

$$\begin{aligned} K &= k_1, k_2, ..., k_n \\ V &= v_1, v_2, ..., v_n \\ Q &= q_1, q_2, ..., q_d \end{aligned} \tag{3.12}$$

  where n is the number of elements in the input, and d is the dimension of the query vector.

- Next, we compute the attention weights using a function that compares the query vector to each of the keys:

$$a_i = \text{softmax}(q^T k_i) \tag{3.13}$$

  where a_i is the attention weight for the i-th element of the input.

- Finally, we compute the output of the attention mechanism as a weighted sum of the values:

$$o = \sum_{i=1}^{n} a_i v_i \tag{3.14}$$

3.3.4.1. Positional Encoding

In the attention mechanism, positional encoding is used to incorporate the order of the input sequence into the model. Positional encoding involves adding a fixed-length vector to the input embeddings that encodes the position of each element in the sequence. It has shown to be an effective way of incorporating sequential information into transformer-based in applications such machine translation, language modeling, sentiment analysis or the present case, MP in the field of AD, where the order the sequence plays a fundamental role.

The mathematical formulation of positional encoding is as follows:

$$\mathrm{PE}_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d}}\right) \tag{3.15}$$

$$\mathrm{PE}_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d}}\right) \tag{3.16}$$

where pos is the position of the element in the sequence, i is the index of the dimension, and d is the dimension of the input embeddings. The positional encoding vectors are added to the input embeddings before they are passed through the self-attention mechanism.

The choice of the hyperparameters used in the positional encoding formulation, such as the use of the sine and cosine functions and the value of 10000, is arbitrary but has been shown to work well in practice. The positional encoding vectors add a sinusoidal pattern to the input embeddings that allows the model to distinguish between elements at different positions in the sequence.

3.3.4.2. Multi-Head Attention

Multi-head attention is a type of attention mechanism used in DL models, particularly in transformer-based architectures. This Multi-Head concept allows the model where it is integrated to attend to multiple parts of the input at once, which is really useful for capturing complex relationships between different parts of the input. It involves splitting the input into multiple heads and computing separate attention scores for each head, which are then combined to produce the final output. It can be formulated mathematically using the following equations:

- First, we split the keys, values, and query vectors into multiple *heads*, each of which has its own set of learned weight matrices:

$$\begin{aligned} K_i &= k_{i,1}, k_{i,2}, ..., k_{i,n} \\ V_i &= v_{i,1}, v_{i,2}, ..., v_{i,n} \\ Q_i &= q_{i,1}, q_{i,2}, ..., q_{i,d} \end{aligned} \tag{3.17}$$

  where i is the index of the head.

- Next, we compute the attention weights and outputs for each head separately, using the same equations as in the basic attention mechanism:

$$a_{i,j} = \text{softmax}(Q_i^T K_{i,j}) \tag{3.18}$$

$$o_i = \sum_{j=1}^{n} a_{i,j} V_{i,j} \tag{3.19}$$

  where in this case i represents the head index and j the corresponding element of the input, being o_i the whole output of the i-th head.

- Finally, we concatenate the outputs from all the heads and multiply them by a learned weight matrix W_O to produce the final output:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(o_1, \ldots, o_H) W_O \tag{3.20}$$

  where H is the number of attention heads.

3.3.4.3. Self-Attention vs Cross-Attention

Both self-attention and cross-attention are usually employed by means of several heads, that is, Multi-Head Self-Attention (MHSA) and Multi-Head Cross-Attention (MHCA). Self-attention operates on a single sequence, while cross-attention operates on two sequences.

Self-attention is a type of attention mechanism used in deep learning models, particularly in transformer-based architectures, that allows the model to attend to different parts of the input sequence to compute a representation of each input element.

In self-attention, the input sequence (same source of information) is transformed into three different vectors: the query vector, the key vector, and the value vector. These vectors are then used to compute the attention score for each element of the input sequence with respect to every other element in the sequence. The attention scores are used to weight the value vectors, which are then combined to produce the final output.

The mathematical formulation of self-attention is as follows:

$$\text{Self-Attention}(X_1) = \text{softmax}\left(\frac{Q_1 K_1^T}{\sqrt{d_k}}\right) V_1 \tag{3.21}$$

where X_1 is the input sequence, Q, K, and V are the query, key, and value vectors, respectively, and d_k is the dimension of the key vectors. Subindex *1* indicates that the query, key and value come from the same source of information. The softmax function is applied row-wise to the matrix $\frac{QK^T}{\sqrt{d_k}}$, resulting in an attention matrix that has the same dimensions as X. The attention matrix is then used to weight the value vectors V, which are combined to produce the final output.

The query, key, and value vectors are obtained through linear transformations of the input embeddings. The exact way in which these transformations are carried out can vary depending on the specific architecture, but in general they involve matrix multiplications with learnable weight matrices.

Self-attention has been shown to be a powerful mechanism for capturing long-range dependencies in sequential data, and it has been used successfully in a variety of natural language processing tasks, including machine translation, language modeling, and sentiment analysis.

On the other hand, in cross-attention, there are two sources of information, where one of the input sequences serves as the query sequence, while the other sequence serves as the key-value sequence. The query sequence is transformed into a query vector, while the key-value sequence is transformed into key and value vectors. The query vector is then used to compute the attention score for each element in the query sequence with respect to every element in the key-value sequence. The attention scores are used to weight the value vectors, which are then combined to produce the final output. Cross-attention is commonly used in machine translation models, where one of the input sequences is the source language and the other sequence is the target language.

The mathematical formulation of cross-attention is as follows:

$$\text{Cross-Attention}(X_1, X_2) = \text{softmax}\left(\frac{Q_1 K_2^T}{\sqrt{d_k}}\right) V_2 \tag{3.22}$$

where X_1 and X_2 are the input sequences which must have the same dimensionality, Q_1 is the query vector that comes from the first source of information, and K_2 and V_2 are key and value vectors that come from the second source of information respectively. As stated before, d_k is the dimension of the key vectors. The softmax function is applied row-wise to the matrix $\frac{QK^T}{\sqrt{d_k}}$, resulting in an attention matrix that has the same dimensions as the query sequence. The attention matrix is then used to weight the value vectors, which are combined to produce the final output.

To summarize the differences between both types of attention, in self-attention, the query, key, and value vectors are all derived from the same input sequence, while in cross-attention, the key-value sequence provides the key and value vectors. Self-attention is used to capture relationships between elements within a sequence, while cross-attention is used to capture relationships between elements in different sequences. Both types of attention can be appreciated as part of the Transformer architecture (Figure 3.7) proposed in [73].

3.3.4.4. Transformer

The Transformer model wraps-up previous attention mechanisms into a single and elegant architecture which allows the model to weigh the importance of different parts of the input sequence when computing a representation of each element in the sequence. Introduced in 2017 [73], it has become the absolute standard for Natural language Processing (NLP) tasks such as language modeling, question answering or machine translation. Figure 3.7 depicts the Transformer architecture proposed in the original paper.

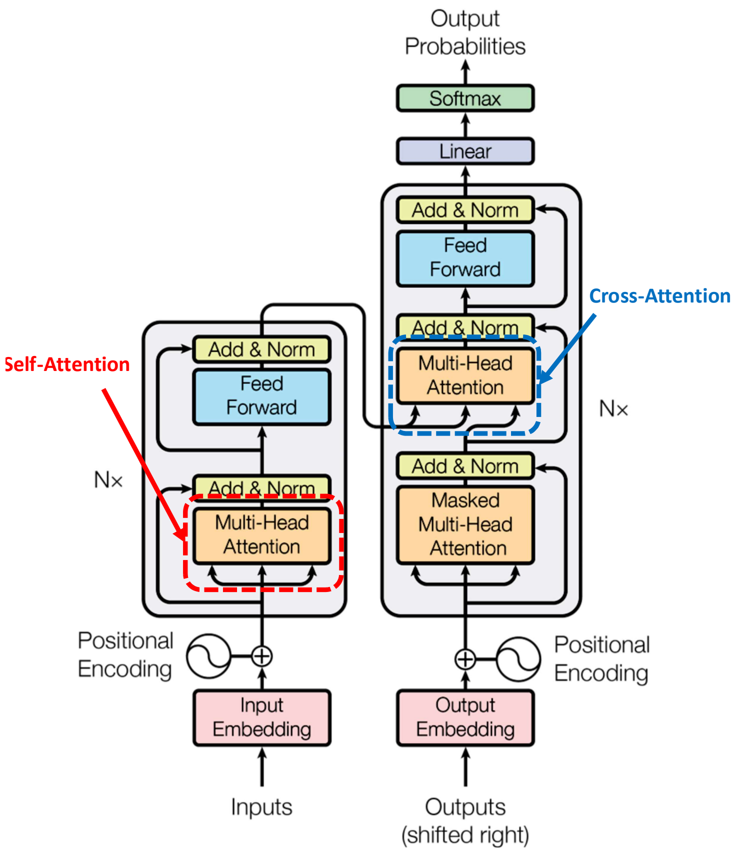


Figure 3.7: Overview of the Transformer architecture
Source: *Attention Is All You Need* [73]

The Transformer architecture consists of an encoder and a decoder. The encoder takes as input a sequence of embeddings, such as word embeddings or character embeddings, and produces a sequence of hidden states that capture the information in the input sequence. The decoder takes as input the encoder hidden states and produces a sequence of output embeddings that represent the output sequence.

The encoder consists of multiple layers, each of which contains two sub-layers: a self-attention sub-layer and a feedforward sub-layer. In the self-attention sub-layer, the model computes a representation of each element in the input sequence by attending to all the other elements in the sequence. The feedforward sub-layer applies a pointwise fully connected layer to each position in the sequence independently and identically.

The decoder also consists of multiple layers, each of which contains three sub-layers: a self-attention sub-layer, a cross-attention sub-layer, and a feedforward sub-layer. In the self-attention sub-layer, the model attends to the previously generated output embeddings to compute the next output embedding. In the cross-attention sub-layer, the model attends to the encoder hidden states to incorporate information from the input sequence. The feedforward sub-layer applies a pointwise fully connected layer to each position in the sequence independently and identically.

The Transformer architecture uses residual connections and layer normalization to facilitate training of deep neural networks. Residual connections allow information to flow directly from one layer to another, bypassing the intermediate layers, which can help prevent the vanishing gradient problem. Layer normalization normalizes the input to each sub-layer to have zero mean and unit variance, which can help stabilize the training process. Finally, the original architecture proposes a fully-connected layer and softmax operation to compute the probability distribution over the target vocabulary that determines the words with the highest probability.

Overall, the Transformer architecture has achieved SOTA results on a wide range of natural language processing tasks, and its success has led to the development of a variety of transformer-based architectures. As we will see in future sections, Transformers have several advantages over recurrent networks (such as LSTM) to encode and decode information with temporal dependencies. Here are some key advantages of transformers:

- **Attention mechanism**: Transformers employ the attention mechanism (both self and cross, if the Transformer decoder is used) that allows them to capture relationships between elements in a sequence more effectively. This mechanism enables the model to focus on relevant elements (or agents, in the field of MP) and assign different weights to different words during the encoding and decoding processes. LSTM networks, on the other hand, rely on recurrent connections that propagate information sequentially, which may not capture long-range dependencies as effectively as self-attention.

- **Parallel Computation**: Transformers are highly parallelizable, meaning that they can process inputs in parallel, which accelerates training and inference. In contrast, LSTM networks are inherently sequential in nature, as the recurrent connections require information from previous time steps to be processed before moving on to the next step. This sequential nature limits the parallelizability of LSTM networks and can result in slower training and inference times.

- **Long-term Dependency**: Transformers are specifically designed to capture long-term dependencies in sequences. The self-attention mechanism allows the model to consider all positions in the input sequence when making predictions, enabling it to capture dependencies over long distances. LSTM networks, although capable of capturing short-term dependencies, can struggle with capturing long-term dependencies due to the vanishing or exploding gradient problem.

- **Transfer Learning**: Transformers have shown to be highly effective for transfer learning tasks. Pretrained transformer models, such as BERT (Bidirectional Encoder Representations from Transformers), have been trained on large-scale datasets and can be fine-tuned for specific downstream tasks with relatively small amounts of task-specific data. LSTM networks typically require more task-specific data to achieve good performance, as they don't benefit from the same level of transfer learning capabilities as transformers.

- **Handling Variable-length Inputs**: Transformers can handle variable-length inputs more easily than LSTM networks. Since transformers process the entire input sequence in parallel, they do not rely on fixed-length input vectors like LSTM networks. This flexibility is particularly advantageous for tasks with variable-length inputs, such as the present work, where a traffic scenario may have an undetermined number of agents around the ego-vehicle.

While transformers have these advantages, it is important to note that LSTM networks are still extensively used by the research community. They have been widely used and well-studied in various applications, especially in tasks where sequential information is critical. The choice between transformers and LSTM networks depends on the specific requirements of the task at hand. The attention mechanisms (including self-attention, cross-attention and transformer encoder) will be a key component of this thesis in the proposed learning-based models (Chapters 5, 6 and specially 7).

3.3.5. Graphs

A graph is a type of data structure that contains nodes and edges. A node can be a person, place, or thing, and the edges define the relationship between nodes. The edges can be directed and undirected based on directional dependencies.

Graphs are excellent in dealing with complex problems with relationships and interactions. They are used in pattern recognition, social networks analysis, recommendation systems, and semantic analysis. Creating graph-based solutions is a whole new field that offers rich insights into complex and interlinked datasets.

Nevertheless, Graph-based data structures have drawbacks, and researchers must understand them before developing graph-based solutions.

- **A graph exists in non-euclidean space**. It does not exist in 2D or 3D space, which makes it harder to interpret the data. To visualize the structure in 2D space, various dimensionality reduction tools must be used.

- **Graphs are dynamic**, *i.e.* they do not have a fixed form. There can be two visually different graphs, but they might have similar adjacency matrix representations. It makes it difficult for us to analyze data using traditional statistical tools.

- **Large size and dimensionality** will increase the graph complexity for human interpretations. The dense structure with multiple nodes and thousands of edges is harder to understand and extract insights.

3.3.5.1. Graph Neural Networks (GNNs)

Graph Neural Networks (GNNs) are a class of neural networks that operate on graphs, which are collections of nodes and edges. A typical graph is represented by a set of node features, represented by a matrix $X \in \mathbb{R}^{N \times D}$, where N is the number of nodes, and D is the number of features associated with each node. Each node is also connected to other nodes via edges, which are represented by an adjacency matrix $A \in 0, 1^{N \times N}$, where $A_{ij} = 1$ if there is an edge between nodes i and j, and 0 otherwise, as observed in Figure 3.8.

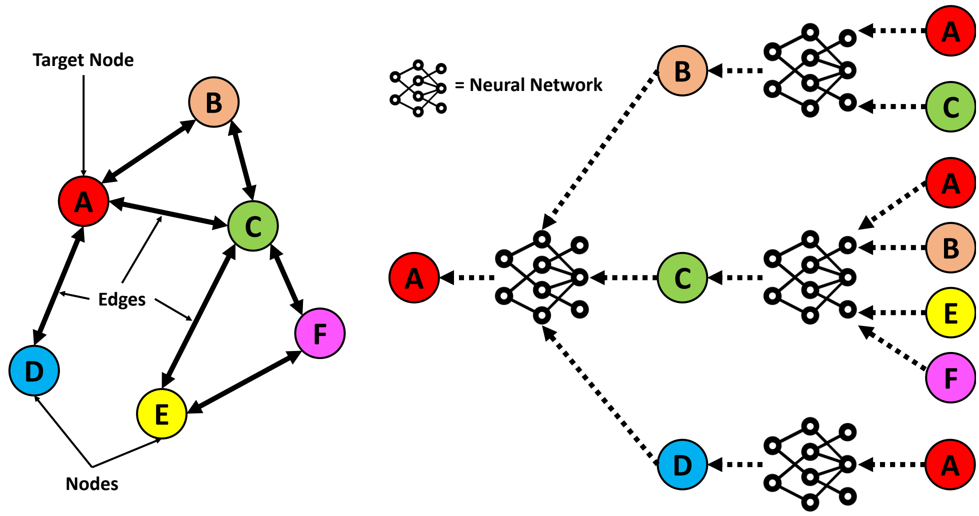


Figure 3.8: Overview of a Graph Neural Network. On the left, it can be observed the input graph of N agents (in this particular case six agents for simplicity) where **A** represents the target node over which the influence of neighbouring nodes is going to be calculated. On the right, the latent space of the neighbours is measured as a preliminary stage before aggregating their information to the target node.

The basic idea behind GNNs is to iteratively update the node representations using information from the graph neighborhood. Specifically, at each iteration, each node aggregates information from its neighbors, and the resulting aggregated representation is used to update the node own representation. This process is typically repeated for multiple iterations until convergence.

One common way to implement this idea is to use the following update rule:

$$h_i^{(l+1)} = f(h_i^{(l)}, \sum_j A_{ij} h_j^{(l)}) \tag{3.23}$$

where $h_i^{(l)}$ is the representation of node i at iteration l, f is a non-linear activation function, and $\sum_j A_{ij} h_j^{(l)}$ is the sum of the representations of the neighbours of node i at iteration l (as stated above, j represents the neighbour node).

By stacking multiple layers of such updates, a GNN can learn increasingly complex representations of the graph.

There are several types of neural networks, such as:

- **Graph Auto-Encoder Networks**, which learn graph representation using an encoder and attempt to reconstruct input graphs using a decoder. They are commonly used in link prediction as Auto-Encoders and are good at dealing with class balance.
- **Recurrent Graph Neural Networks(RGNNs)** are able to learn the best diffusion pattern, and they can handle multi-relational graphs where a single node has multiple relations. They are commonly used in generating text, machine translation, speech recognition or generating image descriptions.
- **Gated Graph Neural Networks (GGNNs)** improve Recurrent Graph Neural Networks by adding a node, edge, and time gates on long-term dependencies, being the common uses similar to RGNNs.

Nevertheless, in this work we focus on a particular type of GNN, that is, Graph Convolutional Networks (GCNs), which are the most common type of GNN used in the field of MP for AD.

3.3.5.2. Graph Convolutional Networks (GCNs)

The majority of GNNs are Graph Convolutional Networks (GCNs), which are a specific type of GNN that use convolutional operations to aggregate information from the graph neighborhood. GCNs were introduced when CNN failed to achieve optimal results due to the arbitrary size of the graph and complex structure. The basic idea behind GCNs is to treat the graph as a signal and use convolutional operations to extract features by inspecting neighboring nodes. GCNs aggregate node vectors, pass the result to the dense layer, and apply non-linearity using the activation function.

The major difference between GCN and CNN is that it is developed to work on non-euclidean data structures where the order of nodes and edges can vary, as it can be appreciated in Figure 3.9. In the 2D convolution operation, each pixel in an image is taken as a node where neighbours are determined by the filter size. The 2D convolution takes the weighted average of pixel values of the red node along with its neighbors. The

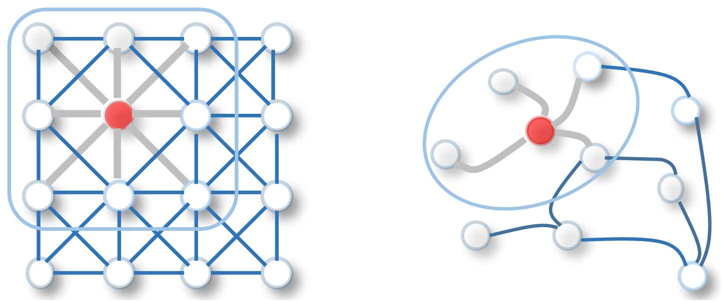


Figure 3.9: Overview of the sliding kernel of a 2D-CNN vs GCN
Source: *A comprehensive survey on graph neural networks* [74]

neighbours of a node are ordered and have a fixed size. On the other hand, to get a hidden representation of a given node red, one simple solution of the graph convolutional operation is to take the average value of the node features of the red node along with its neighbors. Different from image data, the neighbors of a node are unordered and variable in size.

The convolution in GCN is the same as a convolution in CNNs. It multiplies neurons with weights (filters) to learn from data features. A convolutional operation on a graph is defined as:

$$H^{(l+1)} = \sigma(D^{-\frac{1}{2}} A D^{-\frac{1}{2}} H^{(l)} W^{(l)}) \tag{3.24}$$

where $H^{(l)}$ is the matrix of node representations at iteration l, $W^{(l)}$ is the weight matrix of the l-th layer, A is the adjacency matrix of the graph, D is the degree matrix of the graph, and σ is a non-linear activation function. Note that in the mathematical field of algebraic graph theory, the degree matrix of an undirected graph is a diagonal matrix which contains information about the degree of each vertex, that is, the number of edges attached to each vertex

The operation $D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ is a normalization step that scales the adjacency matrix by the inverse square root of the degree matrix. This normalization ensures that nodes with different degrees have similar influence in the convolutional operation. By stacking multiple layers of such convolutions, a GCN can learn increasingly complex features of the graph.

3.3.6. Training

Training is the process of iterating through a dataset to make the network learn the optimal mapping (combination of weights) from input to desired output in the data samples. In each iteration, a forward pass through the network is performed, computing the output of each layer until the end. This produces the output response of the network, which is then compared to a desired output through a defined Loss function ($\mathcal{L}$). This function estimates the output error, which is back-propagated through the network to update its weights with the aim of minimizing the error.

Regarding the supervised training paradigm, Backpropagation [75] is an algorithm used for training artificial neural networks (ANNs) to update the weights and biases of the neurons in the multi-layer network based on the error between the predicted output and the actual output.

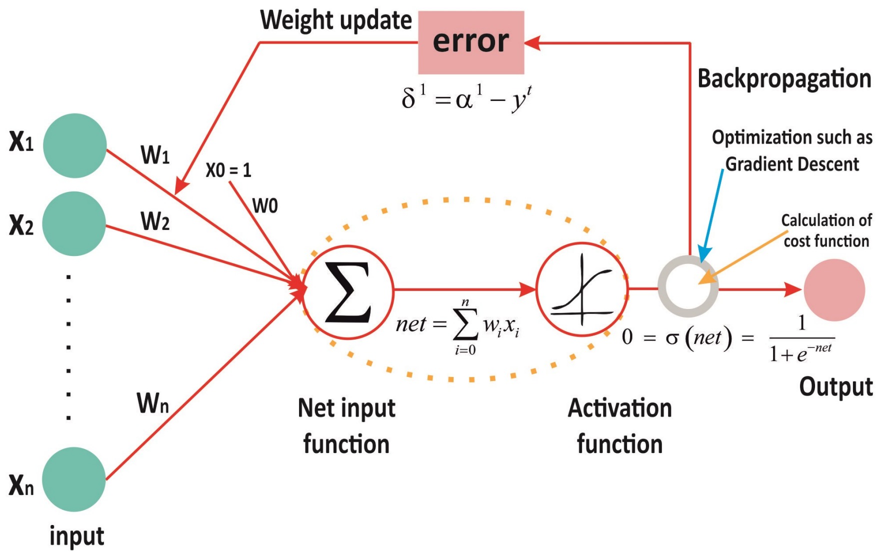


Figure 3.10: Gradient Descent and Backpropagation
Source: *Backpropagation: The basic theory* [75]

The backpropagation (Figure 3.10) algorithm consists of two phases: the forward pass and the backward pass. During the forward pass, the input is fed into the network and propagated through the layers to obtain the predicted output. During the backward pass, the error between the predicted output and the actual output is computed and used to update the weights and biases of the neurons in the network.

The backpropagation algorithm is based on the chain rule of calculus, which allows the gradient of the loss function with respect to each weight and bias to be computed recursively from the output layer to the input layer of the network. The gradient descent algorithm is then used to update the weights and biases in the direction of the negative gradient of the loss function. In that sense, the main losses used in this work are described throughout the remaining content of this Chapter.

3.3.6.1. Optimizer and learning rate

There are several optimizers commonly used for training deep neural networks in the literature, such as Stochastic Gradient Descent (SGD) [76], Adagrad [77], Root Mean Square Propagation (RMSprop) [78] or Adadelta [79]. In this work we use one of the most extended, the ADaptive Moment Estimation (ADAM) [80] optimizer. It is a popular optimization algorithm used in deep learning, particularly for training neural networks. It is an adaptive learning rate optimization algorithm that is well suited for large datasets and high-dimensional parameter spaces.

The basic idea behind ADAM is to compute adaptive learning rates for each parameter based on estimates of the first and second moments of the gradients. The algorithm computes a moving average of the gradients and their squared values, and uses these estimates to update the parameters with a learning rate that adapts to the local curvature of the loss function. The algorithm also includes bias-correction terms to ensure that the estimates are unbiased, especially in the early stages of training when the estimates are highly uncertain.

The update rule for ADAM can be expressed mathematically as follows:

$$
\begin{aligned}
m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\
v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \\
\hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\
\hat{v}t &= \frac{v_t}{1-\beta_2^t} \\
\theta t + 1 &= \theta_t - \frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t
\end{aligned}
\tag{3.25}
$$

where θ_t is the parameter vector at time step t, g_t is the gradient vector, m_t and v_t are the first and second moment estimates at time step t, $\hat{m}_t$ and $\hat{v}_t$ are the bias-corrected moment estimates, α is the learning rate, β_1 and β_2 are the decay rates for the moment estimates, and ϵ is a small constant to prevent division by zero.

The algorithm starts with initializing m_0 and v_0 as zero vectors, and θ_0 as the initial parameter vector. At each iteration, the gradient vector g_t is computed using a batch of training data, and the moment estimates m_t and v_t are updated according to the first two lines of the update rule. The bias-corrected moment estimates $\hat{m}_t$ and $\hat{v}_t$ are then computed using the next two lines of the update rule. Finally, the parameter vector θ_t is updated using the last line of the update rule.

The hyperparameters α, β_1, β_2, and ϵ can be tuned to optimize performance on a particular dataset and neural network architecture.

On the other hand, the learning rate is a key hyperparameter in the optimization process of machine learning algorithms, including optimizer updates like the ADAM optimizer. The learning rate controls the step size taken in the direction of the negative gradient during each iteration of the optimization process.

If the learning rate is too small, the optimization process will be slow and may get stuck in local minima. On the other hand, if the learning rate is too large, the optimization process may overshoot the minimum and oscillate back and forth, or even diverge.

In the context of the ADAM optimizer, the learning rate is used to adjust the size of the update step taken in the direction of the estimated gradient. The update step is multiplied by the learning rate, which determines the size of the step. A larger learning rate will result in larger update steps, and a smaller learning rate will result in smaller update steps.

In practice, the learning rate is usually set through a process called hyperparameter tuning, where different values of the learning rate are tried on a validation set to find the optimal value that results in the best performance of the model on the test set.

One common technique to adjust the learning rate during training is called learning rate scheduling. This involves decreasing the learning rate over time, often according to a predetermined schedule or based on the performance of the model on a validation set. This technique can help improve the convergence and stability of the optimization process, particularly in the later stages of training when the model is close to the optimal solution.

Overall, the learning rate plays a crucial role in the optimization process of machine learning algorithms, including optimizer updates like the ADAM optimizer. Selecting an appropriate learning rate is important for achieving fast and stable convergence to the optimal solution.

In this thesis, we will use the ADAM optimizer and learning rate scheduler in Chapters 5, 6 and 7. Particularly, in terms of the learning rate scheduler, we will make use of the well-established *ReduceLROnPlateau*, which reduces the learning rate when a certain metric of interest (in our case, minADE) has stopped improving for a pre-defined "patience" (*i.e.* number of epochs over the dataset).

3.3.6.2. Losses

3.3.6.2.1. Regression losses A regression loss is a type of loss function used in regression problems to measure the difference between the predicted and true values of a continuous variable. In other words, it is a way to quantify how well a machine learning model is able to predict numerical values based on input data.

In regression problems, the goal is to learn a function that maps input features to output values. A regression loss is used to train the model by penalizing the difference between the predicted and true output values. The loss function is typically minimized during training, so that the model learns to make more accurate predictions. There are several types of regression loss functions, such as the Quantile loss, Log-Cosh or Mean

Absolute Error (MAE). In this work we mainly focus on the Mean Squared Error (MSE) and SmoothL1 losses, also known as the Huber loss.

Mean Square Error (MSE) loss The Mean Squared Error (MSE) loss is a commonly used loss function in regression problems. It measures the average squared difference between the predicted and true values of a continuous variable. The MSE loss is given by:

$$MSE(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{3.26}$$

where y_i is the true value of the i-th data point, $\hat{y}_i$ is the predicted value, and n is the number of data points.

The MSE loss penalizes large errors more strongly than small errors, since it uses the square of the difference between the predicted and true values. This makes it sensitive to outliers and can lead to overfitting if the data contains extreme values.

The MSE loss is used in many regression problems, such as linear regression or polynomial regression. It is often used as a performance metric to evaluate the performance of the model predictions.

SmoothL1 loss The SmoothL1 loss, also known as Huber loss, is a loss function used in regression problems to measure the difference between the predicted and true values of a continuous variable. It is a variant of the L1 loss that is less sensitive to outliers and has a smooth gradient near zero.

The SmoothL1 loss function can be defined as follows:

$$L(y, \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y})^2 & \text{if } |y - \hat{y}| < 1 \\ |y - \hat{y}| - \frac{1}{2} & \text{otherwise} \end{cases} \tag{3.27}$$

where y is the true value, $\hat{y}$ is the predicted value, $|y - \hat{y}|$ corresponds to the L_1 loss (absolute error or distance) and $\frac{1}{2}(y - \hat{y})^2$ corresponds to the L_2 loss (Euclidean error or distance).

The SmoothL1 loss function behaves like the L1 loss for small errors and like the $\mathcal{L}_2$ loss for large errors. Specifically, for errors smaller than 1, it uses the squared difference between the predicted and true values ($\mathcal{L}_2$), which has a smooth gradient. For larger errors, it uses the absolute difference ($\mathcal{L}_1$), which is less sensitive to outliers than the squared difference.

The SmoothL1 loss is used in regression problems when the data contains outliers or when the model needs to be less sensitive to large errors. It is commonly used in object detection and localization tasks, where the predicted bounding boxes can be highly sensitive to small changes in the input data.

3.3.6.2.2. Softmax loss The softmax loss, also known as the cross-entropy loss, is a commonly used loss function in classification problems. It measures the difference between the predicted probability distribution and the true probability distribution of a categorical variable. The softmax loss is given by:

$$CE(y, \hat{y}) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{k} y_{i,j}\log(\hat{y_{i,j}}) \tag{3.28}$$

where $y_{i,j}$ is the true probability of the i-th data point belonging to class j, $\hat{y_{i,j}}$ is the predicted probability, n is the number of data points, and k is the number of classes.

The softmax function is applied to the output of the model to obtain a probability distribution over the classes. The predicted probability of class j is given by:

$$\hat{y_{i,j}} = \frac{e^{z_{i,j}}}{\sum_{l=1}^{k} e^{z_{i,l}}} \tag{3.29}$$

where $z_{i,j}$ is the unnormalized score or logit for class j of the i-th data point.

The softmax loss penalizes the model more heavily for predictions that are far from the true probabilities. It encourages the model to assign higher probabilities to the correct classes and lower probabilities to the incorrect classes.

The softmax loss is used in many classification problems, such as image classification, natural language processing, and speech recognition. It is often used as a performance metric to evaluate the quality of the model predictions.

3.3.6.2.3. Negative Log-Likelihood (NLL) loss The Negative Log Likelihood (NLL) loss is a loss function commonly used in classification problems, particularly in deep learning models that output probabilities for each class. It is a type of Maximum Likelihood Estimation (MLE) loss, which means it attempts to maximize the similarity between the predicted probability distribution and the true probability distribution of the target classes.

The NLL loss function can be defined as follows:

$$L(y_i, \hat{y}_i) = -\log(\hat{y_{i,y_i}}) \tag{3.30}$$

where y_i is the true label of the i-th data point, $\hat{y}_i$ is the predicted probability distribution for that point, and $\hat{y_{i,y_i}}$ is the predicted probability for the true label.

This penalizes the model for assigning low probabilities to the true label, while rewarding it for assigning high probabilities. It is a logarithmic loss function, which means that the penalty for low probabilities increases exponentially as the predicted probability approaches zero.

The Negative Log Likelihood (NLL) loss is used in classification problems because it provides a gradient that can be used to update the model parameters during training, in order to improve the accuracy of the predicted probabilities. It is commonly used in conjunction with softmax activation function, which ensures that the predicted probabilities sum to one across all classes.

Nevertheless, after revisiting the literature when developing the multi-modal prediction paradigm, we realized that most works employed the Winner-Takes-All (WTA) and max-margin (*a.k.a.* Hinge) losses to maximize the similarity between the future trajectories and the ground-truth. A comparison between employing the NLL loss and Winner-Takes-All (WTA)+Hinge will be further detailed in Chapter 6.

3.3.6.2.4. Winner-Takes-All loss Winner-Takes-All (WTA) loss is a loss function used in clustering problems, particularly in competitive learning models. It is a type of unsupervised learning, which means that it does not require labeled data for training.

The WTA loss function can be defined as follows:

$$L(y_i, f(x_i)) = \begin{cases} 0 & \text{if } y_i = \arg\max_j f_j(x_i) \\ 1 & \text{otherwise} \end{cases} \tag{3.31}$$

where y_i is the true cluster of the i-th data point, $f_j(x_i)$ is the activation of the j-th neuron in the output layer for that point, and $\arg\max_j f_j(x_i)$ is the index of the neuron with the highest activation.

This loss function penalizes the model for assigning a data point to the wrong cluster. It works by forcing each neuron in the output layer to specialize in a particular cluster, such that the neuron with the highest activation for a given point corresponds to the true cluster of that point.

The WTA loss is used in clustering problems because it encourages the model to learn a set of representative clusters that capture the structure of the data, without requiring any prior knowledge of the true labels. It is commonly used in conjunction with competitive learning algorithms, such as self-organizing maps (SOMs), that use local competition between neurons to learn a topology-preserving mapping from the input space to the output space.

3.3.6.2.5. Hinge loss Hinge loss is a loss function used in classification problems, particularly in Support Vector Machines (SVMs). It is a type of max-margin loss, which means it attempts to maximize the margin between the decision boundary and the data points.

The hinge loss function can be defined as follows:

$$L(y_i, f(x_i)) = \max(0, 1 - y_i f(x_i)) \tag{3.32}$$

where y_i is the true label of the i-th data point, $f(x_i)$ is the predicted score for that point, and 1 is a margin hyperparameter that determines the width of the margin.

If $y_i f(x_i) \geq 1$, then the point is correctly classified and the loss is zero. If $y_i f(x_i) < 1$, then the point is misclassified and the loss is proportional to the distance between the predicted score and the correct score. The loss function penalizes misclassifications linearly, with a slope of -1 for negative misclassifications and 0 for positive misclassifications.

The hinge loss is used in SVMs because it encourages the model to find a decision boundary that maximizes the margin between the classes, while still correctly classifying the data points. This results in a more robust and generalizable model.

3.3.6.3. Regularization techniques

Regularization techniques in deep learning are used to prevent overfitting and improve the generalization performance of a model. Overfitting occurs when a model fits the training data too closely and captures noise or irrelevant patterns, resulting in poor performance on new, unseen data. Here are some commonly used regularization techniques in deep learning, which can be combined and tuned to improve the performance of a model on a specific task or dataset:

- **$\mathcal{L}_1$ and $\mathcal{L}_2$** regularization are two popular regularization techniques that add a penalty term to the loss function during training. L1 regularization adds the sum of the absolute values of the weights to the loss function, while $\mathcal{L}_2$ regularization adds the sum of the squares of the weights. This encourages the model to learn simpler and more generalizable representations by shrinking the weights towards zero.

- **Dropout** is a regularization technique that randomly drops out some units (neurons) in a layer during training. This forces the remaining units to learn more robust and diverse representations that generalize better to new data. Dropout has been shown to be effective in reducing overfitting, particularly in deep neural networks with many layers.

- **Data augmentation** is a technique that artificially increases the size of the training set by generating new examples from existing ones. This can be done by applying transformations such as rotations, translations, flips, or adding noise to the input data. Data augmentation can help reduce overfitting by increasing the diversity of the training data and improving the generalization performance of the model.

- **Early stopping** is a technique that monitors the performance of the model on a validation set during training and stops the training process when the performance starts to degrade. This prevents the model from overfitting to the training data and allows it to generalize better to new data.
- **Batch normalization** is a technique that normalizes the activations of a layer by subtracting the batch mean and dividing by the batch standard deviation. This helps stabilize the distribution of the activations and reduces the internal covariate shift, which can improve the training process and reduce overfitting.
- **Weight decay** is a technique that adds a penalty term to the loss function during training, similar to L2 regularization. The penalty term is proportional to the square of the weights, and it encourages the model to learn smaller and simpler weights, which can reduce overfitting.

Moreover, another interesting technique to help the model improve during training is hard-mining. Hard-mining is a technique in deep learning used to improve the training of a model by focusing on the samples that are most difficult to classify correctly.

During training, the model makes predictions on a batch of training data, and the loss function is computed based on the difference between the predicted values and the true values. In hard-mining, the training samples that contribute the most to the loss function are identified, and these samples are given more importance during training.

The idea behind hard-mining is that by focusing on the difficult samples, the model is forced to learn more discriminative features that can better distinguish between the different classes. This can lead to better generalization performance and improved accuracy on the test data.

There are several ways to implement hard-mining in deep learning, including:

- **Hard negative mining** involves selecting the training samples that are misclassified with the highest confidence by the model and including them in the next batch of training data. By focusing on the difficult negative samples, the model can learn to better distinguish between the different classes and reduce the number of false negatives.
- **Hard positive mining** involves selecting the training samples that are misclassified with the lowest confidence by the model and including them in the next batch of training data. By focusing on the difficult positive samples, the model can learn to better distinguish between the different classes and reduce the number of false positives.
- **Curriculum learning** involves gradually increasing the difficulty of the training data over time. The model is first trained on easy samples and then gradually

exposed to more difficult samples. This can help the model learn more robust and generalizable features by gradually increasing the complexity of the training data.

Overall, hard-mining is a useful technique in deep learning for improving the training of a model by focusing on the difficult samples. By incorporating hard-mining into the training process, the model can learn more discriminative features and improve its generalization performance on new, unseen data.

3.4. Summary

In this Chapter, the mathematical background of various physics-based and DL-based techniques used for MP models is thoroughly examined. The Chapter aims to provide a comprehensive understanding of these methods and their applications in the field of motion prediction.

The Chapter begins by introducing the physics-based techniques, starting with the KF. The Kalman Filter is a recursive algorithm used to estimate the state of a dynamic system by incorporating noisy measurements. Its mathematical foundation, including the state-space model and the prediction and update steps, is discussed in detail. Next, the HA is presented, which is a combinatorial optimization algorithm used for data association in multiple object tracking. The Chapter delves into the mathematical formulation of the Hungarian Algorithm and how it can be applied to motion prediction tasks.

Moving on to the physics-based models, the Chapter explores the CTRV and CTRA models. These models are widely used for trajectory prediction and capture the motion dynamics of objects by considering their position, velocity, and acceleration. The mathematical equations for these models are examined, highlighting how they can be utilized to predict future motion paths.

The Chapter then transitions to DL-based techniques, which have gained significant attention in recent years due to their ability to capture complex patterns and dependencies in data. Several models are discussed in detail, starting with the 1D-CNN. The mathematical architecture of the 1D-CNN is explained, emphasizing its ability to extract spatial and temporal features from sequential data for motion prediction.

Next, the LSTM model is introduced, which is a type of RNN specifically designed to handle sequence data. The chapter provides an overview of the LSTM architecture, including its memory cell and gate mechanisms, which enable it to capture long-term dependencies and predict future motion accurately.

The chapter then explores GANs and their application in motion prediction. GANs consist of a generator and discriminator network that compete against each other, resulting in the generation of realistic and coherent motion trajectories. The mathematical formulation of GANs and their training process are examined in detail.

Furthermore, the Attention mechanism is examined, which is a component commonly integrated into deep learning models to focus on relevant parts of the input data. The chapter explores the mathematical formulation of the Attention mechanism and its application in motion prediction models, highlighting its ability to assign different weights to different input features based on their relevance.

Then, the GCN is discussed, which is a deep learning model capable of operating on graph-structured data. The chapter explains how GCNs can be used to represent and predict motion patterns in scenarios where objects interact with each other.

Finally, an overview of the training process, including the optimization and back-propagation strategy, the losses covered in this thesis and regularization techniques, are explored.

Chapter 4

SmartMOT: Exploiting the fusion of HD maps and Multi-Object Tracking for Real-Time scene understanding

Avanzad, sin temor a la oscuridad.
Luchad jinetes de Theoden.
Caerán las lanzas, se quebrarán los escudos.
Aún restará la espada.
Rojo será el día, hasta el nacer del sol.
Cabalgad, cabalgad, cabalgad hacia la desolación
y el fin del mundo. Muerte, muerte, muerte.

Discurso de Theoden, Rey de Rohan
El Señor de los Anillos: El Retorno del Rey

4.1. Introduction

In order to achieve a reliable navigation, Autonomous Driving Stacks (ADSs) must perform safe driving behaviours following conventional traffic rules. One of the key concepts when developing safe ADSs is the perception of the environment. Furthermore, the reliability of the DM and local planning modules lies on the performance of the environment detector and its ability to predict future situations. In that sense, a real-time MOT system, which goal is to associate detections (usually in the 3D or BEV space) in a sequence, is essential for AD applications, representing in most cases the preliminary stage before predicting the subsequent future trajectories of these obstacles in the scene, giving the car a valuable reaction time to avoid critical situations or to anticipate its behaviour for the corresponding traffic scenario. The improvements in object detection in the last years have allowed the research community, specially those groups related to AD, to focus on MOT techniques as a preliminary stage before implementing MP, yielding higher accuracy at the cost of computational complexity, making its use prohibitive in real-time systems.

MOT systems aim to estimate the orientation, location and scale of all the objects in the environment over time. While object detection only captures the information of the environment in a single frame, a tracking system must take temporal information into account, filtering outliers (*a.k.a.* false positives) in consecutive detections and being robust to partial or full occlusions. When travelling throughout a route programmed by the path-planner, the vehicle may detect an undetermined number of unforeseen objects over which the MOT module should consider only the most relevant from a safety point of view (such as pedestrians, cyclists or cars) to predict and monitor their trajectories. Then, the vehicle can use the evolution of the scene over time to infer driving behaviour and motion patters for improved MP.

Most MOT approaches [81], [82] model the state of each obstacle with its 3D position, scale, orientation and their corresponding linear and angular velocity. These approaches introduce an unnecessary complexity and computational cost to the system, since most traffic scenes can be described in terms of 2D position, angular and linear velocity, apart from the orientation and scale of the resulting bounding box, that is, a BEV perspective, as depicted in Figure 4.1.

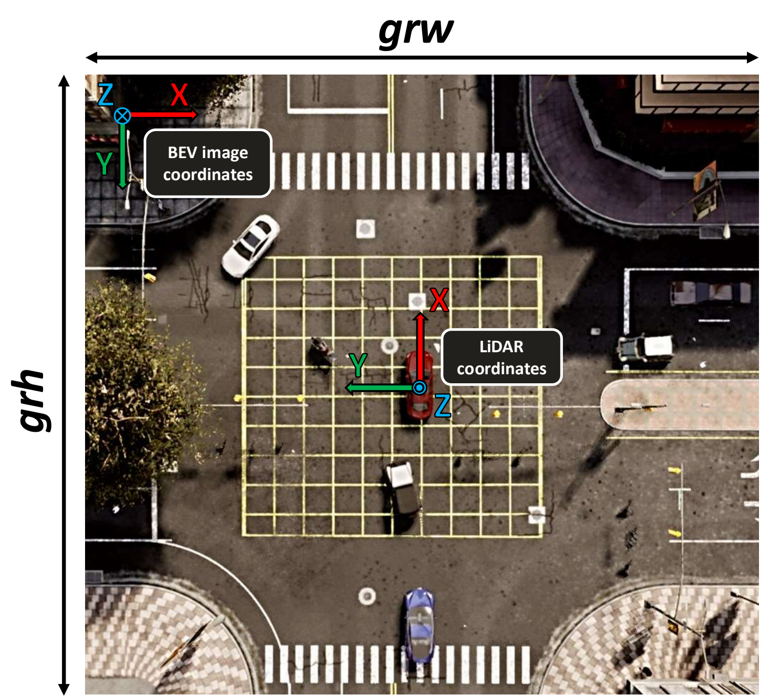


Figure 4.1: LiDAR to BEV coordinates transformation illustrated in the CARLA simulator. *grw* and *grh* stands for the height and width of our real-world grid respectively.

In this thesis, we propose an efficient MOT module over a BEV perspective called SmartMOT. This Chapter summarizes the SmartMOT pipeline and the experimetal results obtained with our proposal. The work in this Chapter was partially published in the following conference paper [83]: "ASmartmot: exploiting the fusion of hdmaps and multi-object tracking for real-time scene understanding in intelligent vehicles applications", 2021 IEEE Intelligent Vehicles Symposium (IV), p. 710-715.

4.2. SmartMOT

In order to solve the problem of monitoring the relevant objects in an efficient way, we propose SmartMOT [83], a simple-yet-accurate combination of traditional techniques such as the Kalman Filter [23] and Hungarian algorithm [67] for state estimation and data association respectively, in order to solve the tracking-by-detection paradigm. Moreover, as mentioned in Section 4.1, the core interest of SmartMOT is the incorporation of HD map semantic, geometric and topological information, in addition to the ego-vehicle status, so as to enhance the efficiency and reliability of the tracking system and subsequent predictions, as observed in Figure 4.2.

The remaining content of this chapter summarizes the SmartMOT pipeline, being made up by: (1) 3D object detection module that returns the bounding boxes; (2) Monitored Area computation to filter non-relevant objects, *e.g.* the VRUs that are inside the sidewalk far away the road or the vehicles that are located in a lane in which lane change is not allowed, (3) BEV Kalman Filter that predicts the object state from the current frame and updates the object state based on the detected bounding boxes at current frame, (4) Hungarian algorithm, which associates the current trackers with new detections, (5) Birth and Death memory that deals with the disappeared trajectories (unmatched trajectories exceeding age_{max} frames) and the newly appeared trajectories (matched trajectories exceeding f_{min} frames) and finally (6) CTRV model which performs short-term physics-based motion prediction of the updated trackers information using both the linear and angular velocity information. As observed, except for the pre-trained object detector module (or ground-truth with the corresponding noise, if used), our MOT system does not need any training and can be directly used for inference.

4.2.1. 3D Object Detection

The first step our MOT algorithm must carry out is to detect the obstacles in the environment around the ego-vehicle. As we will see in Chapter **??**, in order to avoid perspective distortion, we make use of some well known 3D and 2D object detection approaches [84], [85] to perform sensor fusion and retrieve the bounding boxes in the 3D space. Nevertheless, since this thesis do not focus on the object detection stage of the perception layer, some experiments are conducted assuming ground-truth detection including Gaussian noise in the x,y,z to simulate real-world detections. Then, at a given frame t, the detections provided by the object detection module are given in the following form:

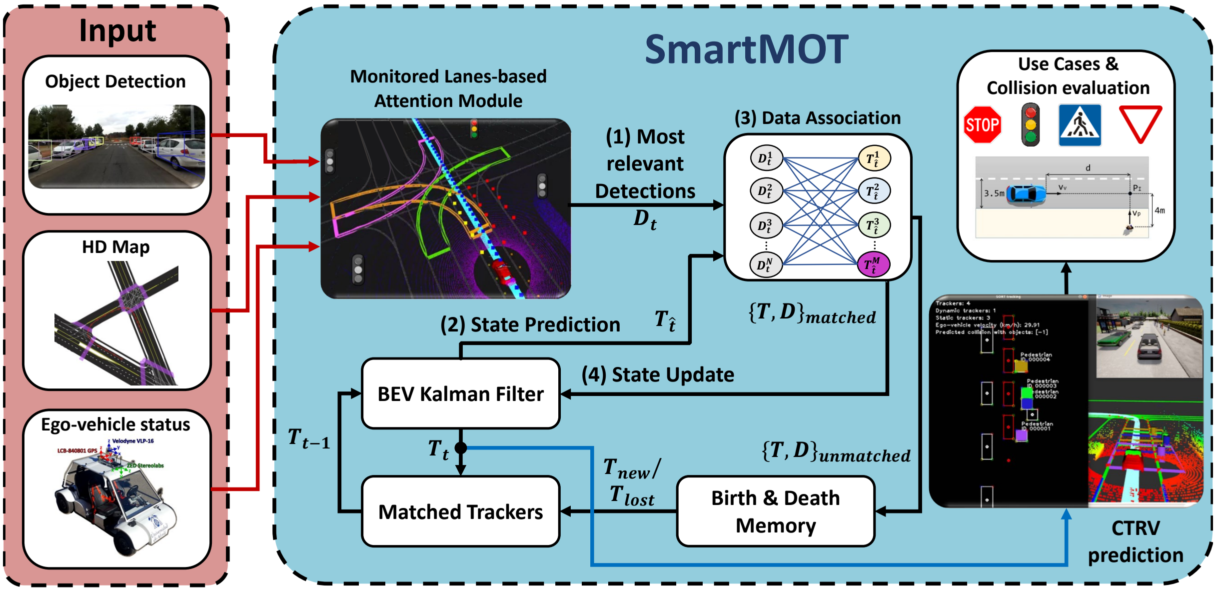


Figure 4.2: **SmartMOT pipeline**: **(1)** The object detection module, mapping layer and localization layer provide the 3D bounding boxes at frame *t*, monitored lanes and ego-vehicle status data respectively; **(2)** A Monitored Lanes-based Attention Module filters the non-relevant traffic participants and transforms the remaining into the BEV image plane; **(3)** A BEV Kalman Filter predicts the state of trajectories in frame *t-1* to current frame $\hat{t}$ throughout the prediction step; **(4)** detections at frame t and predicted trajectories at $\hat{t}$ are matched using the Khun-Munkres (*a.k.a.* Hungarian) algorithm; **(5)** matched trajectories are updated based on their corresponding matched detections and every tracker is evaluated again based on its particular monitorized area, to obtain updated trajectories at frame t; **(6)** Unmatched trajectories and detections are used to delete disappeared trajectories or create new ones respectively; **(7)** Updated trackers at frame t are predicted using a CTRV model and then evaluated using the monitors module.

$$\mathbf{D}_t = [\mathbf{D}_t^1, \mathbf{D}_t^2, ..., \mathbf{D}_t^N] \tag{4.1}$$

Where N is the number of detected 3D bounding boxes at a given frame and threshold. At this point, instead of using all the 3D information of the object [81], [82], we take its projection on the floor plane (BEV information), to reduce the complexity and computational cost of the tracking stage, specially in those urban scenarios full of vehicles, based on the assumption that the height (z) dimension is not as important as other coordinates (x-axis, y-axis) in a context of self-driving navigation. Detected 3D bounding boxes are referred to the LiDAR coordinate system. A grid is applied to establish a relation between real-world and image dimensions to discretize the possible positions of the detected bounding boxes and decrease the complexity and computational cost of the tracking module. This grid is featured by a rectangle, whose center is located at the LiDAR position on the vehicle, where grw and grh represent its width and height in LiDAR coordinates m (meters) respectively. Then, each detection in Equation 4.2 is represented as the tuple:

$$\mathbf{D}_t^i = [x_m, y_m, w_m, l_m, \theta, type, score] \tag{4.2}$$

Where x_m, y_m correspond to the object centroid in LiDAR coordinates (m), w_m and l_m correspond to the width and length of the object respectively (m), θ its orientation angle around the LiDAR Z-axis, object type and detection confidence. Figure 4.1 illustrates the transformation from the source coordinate system (LiDAR), measured in m and placed at the ego-vehicle, to the target coordinate system (BEV), measured in px (pixels) and placed on the top-left corner of the grid, which is the most common way to work with images in computer vision. In other words, we deal with the MOT problem from the BEV image perspective, in order to adapt MOT algorithms originally designed for computer vision purposes.

Equations 4.3 and 4.4 show the transformation matrix between both coordinate systems, including both the rotation and the translation ($\frac{grw}{2}$ and $\frac{grh}{2}$), where a $\mathbf{LiDAR}_{point} = [x_m, y_m, z_m, 1]^T$ is given as the column vector in homogeneous coordinates.

$$\mathbf{T} = \begin{bmatrix} 0 & -1 & 0 & \frac{grw}{2} \\ -1 & 0 & 0 & \frac{grh}{2} \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.3}$$

$$\mathbf{BEV}_{point} = \mathbf{T} \cdot \mathbf{LiDAR}_{point} \tag{4.4}$$

At this point, each detection is represented by the tuple shown in Equation 4.2, but now x_m, y_m represent the obstacle centroid in BEV image perspective. Furthermore, the resolution of the BEV image can be modified, in such a way the image width in pixels is given to the algorithm and the image height is calculated according to the aspect ratio of the real world with respect to the width of the image in pixels. Finally, to convert a point from real-word units (m) to camera units (px), we apply the corresponding scale factor to each coordinate:

$$\begin{bmatrix} x_{px} \\ y_{pc} \end{bmatrix} = \begin{bmatrix} \frac{gpw}{grw} & 0 \\ 0 & \frac{gph}{grh} \end{bmatrix} \begin{bmatrix} x_m \\ y_m \end{bmatrix} \tag{4.5}$$

However, it is very common to have different scales for x and y-axis since it is more interesting to have a further view in the x LiDAR axis rather than a large side sweep in terms of y LiDAR axis. Considering this hypothesis, the right way to obtain the width and length of the BEV LiDAR bounding box in *pixels* is to obtain the corners of the rotated bounding box in pixels and then compute the L_2 (*a.k.a.* Euclidean distance) among the corresponding corners to obtain the width and length in *pixels*. Nevertheless, the object detector provides the rotation angle of the obstacle (featured as θ) according to its own coordinate system and not around the ego-vehicle coordinate system. Regarding this constraint, to calculate the dimensions of the bounding box in pixels, three steps must be followed:

$$
\begin{aligned}
c1_m &= (x_m - \frac{l_m}{2}, y_m - \frac{w_m}{2}) \\
c2_m &= (x_m - \frac{l_m}{2}, y_m + \frac{w_m}{2}) \\
c3_m &= (x_m + \frac{l_m}{2}, y_m - \frac{w_m}{2}) \\
c4_m &= (x_m + \frac{l_m}{2}, y_m + \frac{w_m}{2})
\end{aligned}
\tag{4.6}
$$

First, we assume a horizontal bounding box ($\theta = 0$) at the BEV image coordinate system origin, where *c1* corresponds to the top-left corner (*c2*, *c3* and *c4* are placed clockwise). Then, using the above equations for each corner, the Euclidean distance is applied between *c1* and *c2* to obtain the width in pixels, in the same way that the Euclidean distance is applied between *c1* and *c4* to obtain the length in pixels.

$$w_{px} = \sqrt{(c1_{px,x} - c2_{px,x})^2 + (c1_{px,y} - c2_{px,y})^2} \tag{4.7}$$

$$l_{px} = \sqrt{(c4_{px,x} - c2_{px,x})^2 + (c4_{px,y} - c2_{px,y})^2} \tag{4.8}$$

Finally, the first four variables of the detection tuple shown in Equation 4.2 are converted into pixels, in such a way the tracking algorithm will monitor these bounding boxes in the BEV image by using the following tuple:

$$\mathbf{D}_{t,i} = [x_{px}, y_{px}, w_{px}, l_{px}, \theta, type, score] \tag{4.9}$$

4.2.2. Monitored Lanes-based Attention Module

ADSs need to locate itself in the environment to know what is happening around in order to make decisions and execute a correct navigation like a human driver would. When we talk about localization, the first thing we need is a map where to be located and, particularly for AD, a HD map that contains not only a general geometric description of the scene, but also the topological information of the lanes (lanes type, boundaries constraints, etc.) as well as the semantic information of the road.

A HD map is usually a text file describing the real-world features related to the road map and its location within a 2D/3D space, and can do things that other sensors cannot [86]: First, they have an *infinite range* and, therefore, can *see* even into occluded areas.

Second, HD maps will never fail due to environmental conditions. Lastly, HD maps contain highly refined data. This information can be used by different modules of an AV, (including localization, vehicle control, path planning, perception and system management) drastically reducing the computational load and complexity in comparison to other more complex methods, providing robustness and reliability to the system.

In terms of mapping information, we may distinguish three main categories:

- Topological information provides the connectivity between geometry features. Particularly in the field of AD, this is usually the network of roads. This kind of information can allow vehicles to traverse the most energy-efficient route, based on traffic speed, road grade or distance, as well as ensure that ADSs obey traffic regulation orders, such as one-way streets or the corresponding regulatory elements (pedestrian crossing, traffic light, stop signal, etc.).
- Geometric information provides the geometry or shape of other environmental features that can be static (permanent obstruction, such as buildings, bridges or tunnels), temporary (exist for only a limited amount of time, like traffic cones, parked vehicles or temporary road works) and dynamic features (moving people, objects or vehicles). Most of these features are incorporated by means of perception systems, specially in terms of dynamic features, in order to include that information in the HD map for successful motion planning and prediction.
- Semantic information returns the *meaning* of aforementioned features, such as road speed limit, road classification, lane information or even the relational information among the different lanes, *i.e.* how lanes work together, different types of lanes, where vehicles must stop and where vehicles can and cannot turn.

As illustrated, providing rich physical contextual information allows ADSs to make informed decisions in different driving scenarios. In this thesis, we particularly make use of the OpenDrive [87] HD map format, which has been mainly used for two different purposes, as shown in Figure 4.3:

- Global Path Planning, which uses a specific path planner where inputs are the HD map information and the ego-vehicle current location to retrieve an optimal (usually optimized based on the travelled distance) global route towards an specific goal.
- Map monitoring, responsible for monitoring the most relevant static and dynamic map elements around the ego-vehicle at each timestep, such as standard lanes (current, back), intersection lanes (merge, split and cross) and regulatory elements (e.g. give way, stop, pedestrian crossing, traffic light).

In particular, given a pre-defined global route, in this thesis we focus our interest on designing a map monitor module, responsible for retrieving the most relevant lanes around the ego-vehicle to enhance real-time perception and scene understanding requirements.

4.2.2.1. Map Monitor

In a similar way to humans that pay more attention to close obstacles, people walking towards them or upcoming turns rather than considering the presence of building or people far away, the perception layer of a self-driving car must be modelled to focus more on the salient regions of the scene [88] and the more relevant agents to predict the future behaviour of each traffic participant. In that sense, high-fidelity maps have been widely adopted to provide offline (also known as context) information to complement the online information provided by the sensor suite of the vehicle and its corresponding algorithms. Recent learning-based approaches [89] [42] [34] [31], which present the benefit of having probabilistic interpretations of different behaviour hypotheses, require to build a representation to encode the trajectory and map information. [89] assumes that detections around the vehicle are provided and focuses its work on behaviour prediction by encoding entity interactions with ConvNets. Intentnet [31] proposes to jointly detect traffic participants (mostly focused on vehicles) and predict their trajectories using raw LiDAR pointcloud and rendered HD map information. PRECOG [15] aims to capture the future stochasiticity by flow-based generative models. Furthermore, MultiPath [42] uses ConvNets as encoder and adopts pre-defined trajectory anchors to regress multiple possible future trajectories.

As observed, recent DL-based techniques use relatively complicated filters to predict, in an accurate way, the spatial features of the obstacles in the scene, increasing the complexity and computational cost of the system. On the other hand, traditional methods for behaviour prediction are rule-based, where multiple behaviour hypothesis are generated based on constraints from the road maps. As stated above, road maps present some clear advantages over other perception sensors: They have "infinite range", so they can extract information even into occluded areas. Second, they do not fail under challenging environmental conditions, such as intense fog or rain. Third, recent HD maps contain highly refined data (in which many hours or days of human verification and preprocessing to reduce noise and uncertainty), quite useful to perform safe navigation. Then, HD maps can be an additional sensor that cannot fail unless the road infrastructure changes, providing meaningful, accurate and useful information in real-time operation.

Regarding this, we design a Map Monitor in charge of monitoring the surrounding area of the vehicle. The inputs of the Map Monitor are the information provided by the Map Parser module (in charge of getting the information of the map from the HD map file and transform it into custom classes that can be used by other modules like Planning or Perception) and the waypoint route previously obtained by the path planner. The main goal of the Map Monitor is to only monitor the most relevant map elements around the ego-vehicle given the route provided by the global planner (or a new route if the local planer decides to recalculate the route).

First, the path planner returns the route that is divided in segments separated by a given distance and calculates in which segment of the route the ego-vehicle is found, activating a flag in such a way the Map Monitor can start operating. Otherwise, in case the ego-vehicle cannot be located inside the route, the Map Monitor is deactivated. Secondly, a monitor callback is called periodically every time the ego-vehicle status (position, velocity, orientation, etc.) is received, as observed in Figure 4.2. This callback evaluates, if the Map Monitor module is active, calculates the monitored elements frontwards and backwards for a given distance which is proportional to the ego-vehicle velocity given a braking distance linear model that establishes a linear regression between two arrays of velocity and braking distance data. Nevertheless, we make use of a threshold distance to still monitor the environment if the ego-vehicle is stopped.

The monitored elements are:

- **Standard Lanes**: Current, back and the corresponding left and right lanes. Current lane is monitored from current position to a dynamic distance depending on the velocity of the ego-vehicle. Back lane is monitored from current position to back a proportional distance of the dynamic current lane obtained distance. Left and right lanes are monitored the same distance that current and back only if the lane marking from the HD map data allows the lane change.
- **Intersection Lanes**: Other lanes that intersect the current monitored lane are checked. Intersection lanes can have different roles: split (1 lane splits into 2 or more), merge (2 or more lanes merge into 1) and cross (a lane crosses a part of the current lane). To calculate the intersection lanes, each lane of every junction (junctions are areas where more than 2 roads meet) in the current lane is evaluated. The polygon of each lane is calculated and evaluated if is inside the polygon of the current lane. It is important to consider that roundabouts are considered as a set of multiple junctions.
- **Regulatory Elements**: The monitored elements are stops, giveaways, traffic lights, speed limits and crosswalks. The regulatory elements are only monitored for the next intersection affecting the route.

An example of our map monitor module in the CARLA simulator [61] using the RVIZ [90] may be observed in Figure 4.4:

As illustrated in Figure 4.2, once the object detections have been provided and the monitored area has been computed, the Monitored Lanes-based Attention Module helps us to increase the efficiency and robustness of the system to avoid tracking and predicting all obstacles in the environment, which would escalate the computational cost especially in arbitrarily complex urban scenario. This attention module is not focused on DL since the main purpose is to filter non-relevant obstacles in an efficient and interpretable way,

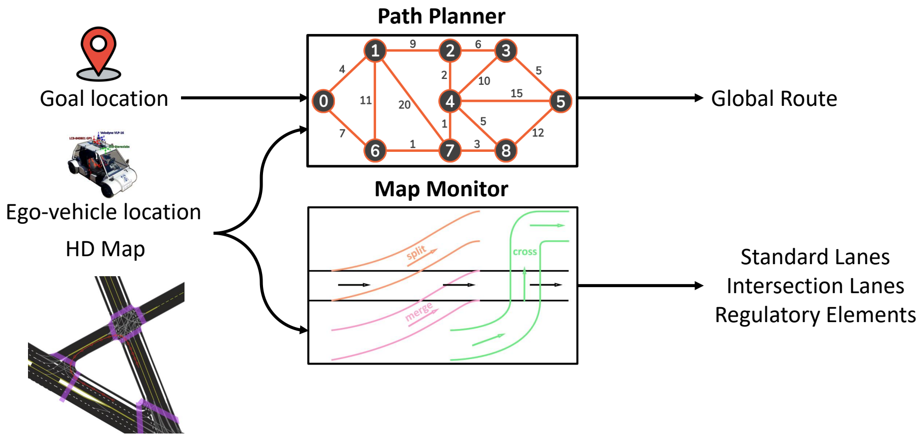


Figure 4.3: Main uses of HD map: Path Planning and Map Monitoring

such as agents driving way in opposite direction lanes, parked vehicles or pedestrians who are chatting on the sidewalk. The filtering process carried out by our Monitored Lanes-based Attention Module is summarized as follows:

Algorithm 2: Jordan's Curve theorem to determine if a point is inside a polygon

Data: point, polygon
Result: isInside
crossings ← 0;
for *i from 0 to (polygon.length - 1)* **do**
  vertex1 ← polygon[i];
  vertex2 ← polygon[(i + 1) mod polygon.length];
  if *(point.y > min(vertex1.y, vertex2.y))* ***and*** *(point.y ≤ max(vertex1.y, vertex2.y))* **then**
    if *point.x ≤ max(vertex1.x, vertex2.x)* **then**
      if *vertex1.y ≠ vertex2.y* **then**
        xIntersection ← (point.y - vertex1.y) * (vertex2.x - vertex1.x) / (vertex2.y - vertex1.y) + vertex1.x;
        if *vertex1.x == vertex2.x* ***or*** *point.x ≤ xIntersection* **then**
          crossings ← crossings + 1;
        end
      end
    end
  end
end
isInside ← (crossings % 2 == 1);
return *isInside*;

1. First, we determine the lanes of interested around the vehicle, till a given threshold. The minimum information will be the current front and back lane information (mandatory for the Adaptive Cruise Control and Unexpected Pedestrian use cases), as well as left and right lane information if lane change is available considering the presence of a discontinuous line. Moreover, if an intersection is near the ego-vehicle, other lanes of interest such as merging, splits and intersections are considered, which

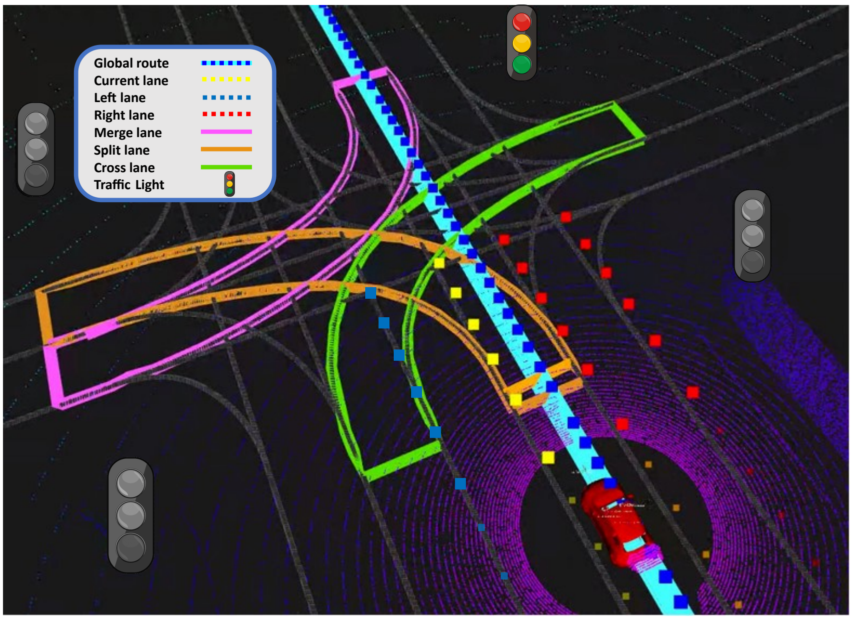


Figure 4.4: Monitored area in the CARLA simulator using the RVIZ tool. It can be appreciated the visualization of the global route, standard lanes, intersection lanes and different traffic lights (regulatory elements). Note that the relevant traffic light is coloured while remaining ones are masked.

are specially useful in urban scenarios when the vehicle faces a roundabout or other vehicles incorporation in an intersection.

2. In order to consider an agent as relevant, we study the presence of this agent in our monitored lanes in an elegant and efficient way. The main idea is to find the polygon segment (two nodes on the left, same on the right) closest to the agent. To do that, we iteratively compute the *L2* distance between the agent position (transformed into global (*a.k.a.* map) coordinates) and the left way nodes (starting from the beginning) of a certain lane. Note that it is irrelevant to take either the left or right lane in terms of observing if the detection is inside a polygon made up by four nodes of the lane, such as there exists the same number of nodes for both ways. For example, in the case of an agent located in front of the vehicle, the distance will decrease since the subsequent left way nodes are closer to the agent.

3. Once the new calculated distance is greater than the previous value, that means the closest segment with nodes N_0 and N_1 are found. Taking the same lane indexes in the right way, we obtain a 4-side polygon in which the detection is evaluated using the Jordan curve theorem [91], as depicted in Algorithm 2. In this theorem, the input parameters are a point and a polygon, where, by means of a simple-yet-accurate ray casting algorithm, a loop is used to iterate over the polygon vertices and performs the necessary checks to determine the number of crossings. The Jordan's Curve Theorem states that a point is inside a polygon if the number of crossings from an arbitrary direction is odd. Consequently, in our particular case, if the object detection lies outside the closest polygon segment, the traffic participant is considered as non-relevant.

4. Nevertheless, despite this proposal is coherent for non-holonomic obstacles with more constrained behaviours like cars, vans or trucks, the behaviour of Vulnerable Road Users (VRUs), such as pedestrians or cyclists, is usually difficult to predict. Hence, we widen the closest segment area a certain threshold L to the sidewalk so as to track the closest VRUs to the road.

4.2.3. BEV Kalman Filter: State Prediction

Once we have obtained the most relevant BEV detections of the environment, a BEV Kalman Filter is used to track the objects. To predict the state of object trajectories from the previous frames to the current frame, we approximate objects inter-frame displacement using a constant velocity model, which is independent of other objects in the scene and of the LiDAR motion. Regarding this, the estimation of the measured variables in the following frame are:

$$x_{px}(\hat{t}) = x_{px}(t) + v_x \quad ; \quad y_{px}(\hat{t}) = y_{px}(t) + v_y$$
$$s(\hat{t}) = s(t) + v_s \quad ; \quad \theta(\hat{t}) = \theta(t) + v_\theta$$

Since we formulate the tracking problem over the BEV plane, we remove all variables related to the third dimension of the object, such as its z coordinate of the 3D bounding box centroid, its associated velocity and the height of the obstacle. On the other hand, since our tracking-by-detection algorithm is inspired by the well-established SORT (Single Online and Real Time) [92] tracking model, originally proposed to track pedestrians using videos as input, some additional variables are included in the object state, such as the aspect ratio and the scale of the bounding box, to help in the tracking stage. The aspect ratio can be defined as the relation between the width and the length of the obstacle. Likewise, the scale represents the area of the target bounding box. Then, the state of each object tracker (usually referred as trajectory tracker in the literature) can be expressed as:

$$\mathbf{T}_t^j = [x_{px}, y_{px}, s, r, \theta, x_{px}^{'}, y_{px}^{'}, s^{'}, \theta^{'}] \tag{4.10}$$

Note that the angular velocity $\theta^{'}$ is used in the state space to improve the prediction of the obstacle in later frames. Furthermore, as shown in [92], the aspect ratio of the bounding box is considered to be constant. As observed in Figure 4.2, at every frame t, a tuple $\mathbf{T}_t = [\mathbf{T}_t^1, \mathbf{T}_t^2, ..., \mathbf{T}_t^M]$ is returned by the data association module, where each element correspond to an association between a detection and a tracker. Note that M represents the current number of trackers. Then, based on these associations between trackers of the previous frame and current detections, and assuming a Kalman Filter of first order (constant velocity model), the tuple $\mathbf{T}_{\hat{t}}$ is calculated, where each element corresponds to the predicted trajectory ($\mathbf{T}_{\hat{t}}^{j}$) in the current frame t expressed as:

$$\mathbf{T}_{\hat{t}}^{j} = [x_{px}(\hat{t}), y_{px}(\hat{t}), s(\hat{t}), r, \theta(\hat{t}), x_{px}^{'}, y_{px}^{'}, s^{'}, \theta^{'}] \tag{4.11}$$

This tuple of predicted trajectories based on the previous frame associations, in addition to the current frame detections, represents the inputs to the data association algorithm at frame t.

4.2.4. Data association

In order to associate the detections $\mathbf{D}_t$ and the trackers information after the Kalman Filter state prediction $\mathbf{T}_{\hat{t}}$, the Hungarian algorithm is applied. The resulting affinity matrix presents N rows (number of detections at frame t) and M columns, which correspond to the number of predicted trajectories based on the information of frame $t-1$. Each element of the matrix corresponds to the Intersection over Union (IoU) in the BEV plane between every pair of predicted trajectory and detection. Then, following the principles stated in the Hungarian algorithm 1 stated in previous sections, we solve the bipartite graph matching problem, rejecting the matching if the BEV-IoU metric is lower than a given hyperparameter IoU_{th}, giving rise to a set of matched detections ($\mathbf{D}_{matched}$) and predicted trackers ($\mathbf{T}_{matched}$) with the same length H (the number of matches), as well as a set of unmatched detections ($\mathbf{D}_{unmatched}$), where $P = N - H$ is the number of unmatched detections, and a set of unmatched trajectories ($\mathbf{T}_{unmatched}$), where $Q = M - H$ is the number of unmatched detections.

4.2.5. BEV Kalman Filter - Object State Update

As observed in Figure 4.2, once we have the corresponding sets of matched detections and trajectories, based on the Kalman Filter prediction-update cycle, we update the state space of each trajectory based on its corresponding matched detection. To do that, we use the weighted average between the matched detection values and the state space of the trajectory tracker, according to [23].

On the other hand, in the same way that [81], we appreciate that this state update step does not work properly for obstacle orientation. The reason is simple: Unless the object detector is based on sensor fusion and vision information is included, the object detector cannot distinguish if the obstacle is rotated 0 or π, $\frac{\pi}{2}$ and $\frac{3\pi}{2}$, and so on, around its z-axis. That is, the orientation may differ by π in two consecutive frames. Then, if no orientation correction is applied, the Kalman Filter associated to the tracker can get easily confused, since it tries to adapts itself to the new orientation value rotating the object by π in following frames, giving rise to a low BEV-IoU between new detections and predicted trajectories. However, regarding the assumption that obstacles must move smoothly and its orientation cannot be modified by π in one frame (0.1 s assuming a frequency of 10 Hz), when this happens the orientation of the corresponding matched detection or matched tracker can be considered wrong. To solve this problem, the detection module only considers angle from 0 to π (that is, if an angle exceeds π, it is substracted to the provided angle). Moreover, if the difference of orientation between a given matched detection and its corresponding matched trajectory is greater than $\frac{\pi}{2}$, as stated before, either the orientation of the detection or the orientation of the tracker is wrong. Finally, we add π to the orientation of the tracker with the aim to be consistent with the matched detection.

4.2.6. Deletion and Creation of Track Identities

When obstacles leave and enter the aforementioned monitored lanes, unique identities must be destroyed or created accordingly. In most tracking algorithms it is known as the Birth and Death Memory, which is based on the set of unmatched trackers and detections provided by the data association algorithm, where the unmatched trackers represent potential objects leaving the monitored area, in the same way that unmatched detections represent potential objects entering in the area of interest.

In order to avoid tracking of false positives or non-relevant obstacles, a new tracker is not created until the unmatched detection has been continuously detected in the next f_{min} frames. Then, the tracker is initialized with the features of the detected bounding box, and the associated velocities set to zero. Note that, as stated in [92], since the velocity associated to the measured variables is unobserved at this moment (i.e., tracker initialization), the covariance initializes the value of the velocities (in the present work, velocity of the x_{px}, y_{px} centroid, scale s and rotation angle θ) with large values, reflecting their uncertainty.

To avoid removing true positives trajectories from the scene, they are not terminated unless they are not detected during consecutive a_{max} frames. This assumption prevents an unbounded growth in the number of localisation errors and trackers due to predictions over long duration where the object detector does not provide any correction. Note that since this work does not consider object re-identification for simplicity, an object should leaves the scene and then reappers, according to the SORT algorithm, if it is initialized with a new tracker under a new identity. As shown in Figure 4.2, the inputs to the Matched Trackers module are the updated matched trajectories from the BEV Kalman Filter and a set of created and deleted trackers, which jointly represent the input trajectories for the prediction step in the following frame.

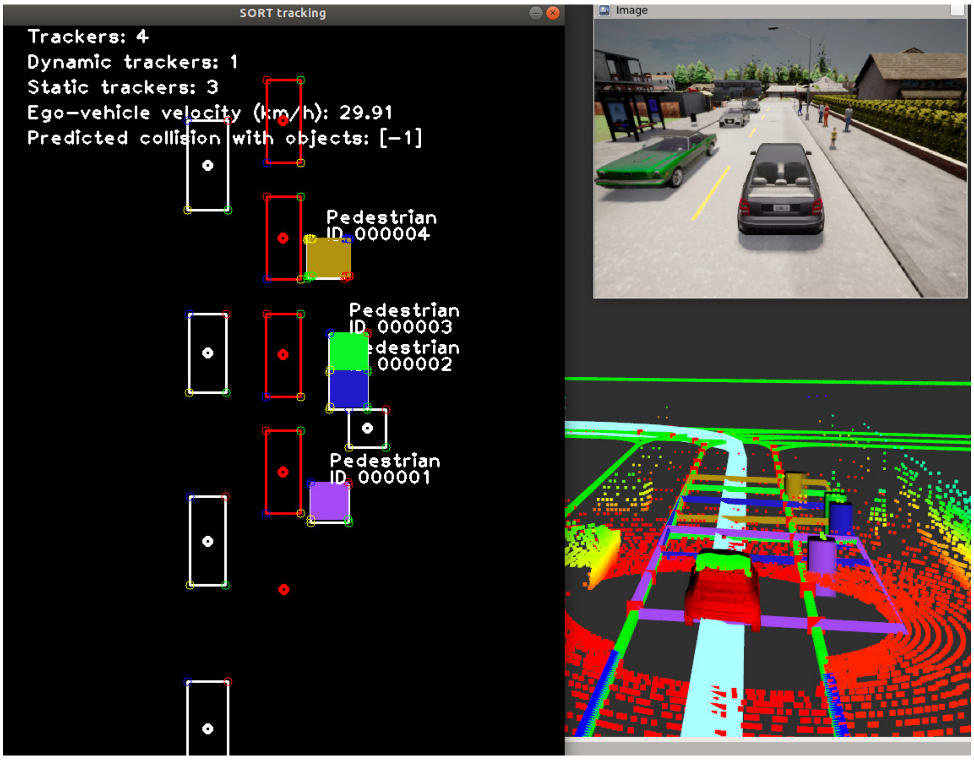


Figure 4.5: Simulator use case of SmartMOT in the CARLA simulator: Vulnerable Road Users (VRUs) are considered on the sidewalk if the they are close enough to the closest segment of the corresponding centerline.

4.2.7. CTRV prediction

The last stage of our SmartMOT pipeline is a physics-based MP model to predict the future behaviour of the agents in the short-term. In particular, we make use of the previously studied CTRV model. Once the tracker information (position, velocity and orientation) has been retrieved in real-world coordinates (instead of BEV image coordinates), we are able to differentiate between static and dynamic agents. Then, given the ego-vehicle status and dynamic agents short-term prediction, we are able to analyze the risk of collision or to carry out the state of the current behaviour (Adaptive Cruise Control, Pedestrian Crossing, etc.) in such a way SmartMOT can send a signal to suddenly stop the car. Further experiments will be detailed in Chapter 4.3. Figure 4.5 illustrates an example of the filtering process, tracking-by-detection paradigm and CTRV prediction.

4.3. Experimental results

As stated in Section 4.2, SmartMOT is a tracking pipeline that leverages HD map information to subsequently conduct a physics-base unimodal prediction. To validate this algorithm, we first validate the proposed tracking algorithm using the KITTI MOT benchmark. Then, we conduct an interesting study of how integrating the monitored area can reduce the risk of collision and/or the impact velocity on the Vulnerable Road User (VRU).

4.3.1. Multi-Object Tracking performance

In order to evaluate our proposed MOT system pipeline, we carry out the evaluation in the KITTI MOT benchmark based on the method proposed by [81]. The KITTI MOT benchmark is composed of 29 testing and 21 training/validation video sequences, where each sequence is provided with the corresponding RGB images (left and right camera of the stereo pair), LiDAR point cloud and the corresponding calibration file. Since KITTI does not provide any annotation (*i.e.*, the groundtruth) for the testing split, we decided to evaluate our system in the training/validation split. Moreover, although KITTI distinguish among eight different classes for the object type, our work focus on the car subset, since it is the class that contains the most number of instances over the whole benchmark.

4.3.1.1. Multi-Object Tracking metrics

Mainstream metrics applied to MOT systems are extracted from CLEAR MOT metrics [93], such as MOTA (Multi-Object Tracking Accuracy), MOTP (Multi-Object Tracking Precision), ML/MT (Number of Mostly Lost/Tracked trajectories), IDS (Number of identity switches), FRAG (Number of fragmentations generated by false negatives) and FN/FP (Number of false negatives/positives). These metrics provide a comprehensive assessment of tracking performance by considering aspects such as accuracy, precision, and overall performance:

4.3.1.1.1. MOTA (Multi-Object Tracking Accuracy) The MOTA metric is commonly used to evaluate the performance of multi-object tracking algorithms. It measures the overall tracking accuracy by considering the false positives (FP), false negatives (FN), and identity switches (IDS) in the tracking results. The formula for calculating MOTA is given as:

$$MOTA = 1 - \frac{\mathrm{FN} + \mathrm{FP} + \mathrm{IDS}}{\mathrm{GT}} \tag{4.12}$$

where:

- FN (False Negatives) represents the number of ground truth objects that were not correctly detected by the tracking algorithm.
- FP (False Positives) represents the number of false detections made by the tracking algorithm.

- IDS (Identity Switches) represents the number of times the algorithm incorrectly switches the identity of a tracked object.
- GT (Ground Truth) represents the total number of ground truth objects in the video sequence.

A higher MOTA value indicates better tracking accuracy, with a perfect tracking result yielding MOTA = 1.

4.3.1.1.2. MOTP (Multi-Object Tracking Precision) The MOTP metric is used to assess the localization accuracy of a multi-object tracking algorithm. It measures the average precision of the tracked object positions by considering the distance between the predicted locations and their corresponding ground truth locations. The formula for calculating MOTP is given as:

$$MOTP = \frac{\sum_{i=1}^{N} d_i}{N} \tag{4.13}$$

where:

- N represents the total number of matched object pairs between the predicted and ground truth locations.
- d_i represents the Euclidean distance between the predicted location and the ground truth location for the i-th matched object pair.

The MOTP metric ranges between 0 and 1, with a higher value indicating better localization accuracy. A perfect tracking result with exact object positions would yield MOTP = 1.

4.3.1.1.3. Integral metrics: AMOTA and AMOTP Nevertheless, these metrics analyze the DAMOT system performance at a given threshold, not taking into account the confidence provided by the object detector and possibly misunderstanding the capability of the method. That means they do not take into account the full spectrum of precision and accuracy over different thresholds. Moreover, these traditional metrics evaluate the performance of the MOT system on the image plane (by projecting the detected 3D bounding box onto the image plane), which does not demonstrate the full strength of 3D DAMOT. In that sense, AB3DMOT [81] recently presented a 3D extension of the KITTI 2D MOT evaluation, known as KITTI-3DMOT, which focuses on the dimensions, orientation and centroid position of the 3D bounding box instead of the projection onto the image plane to evaluate the performance of the MOT system. Moreover, two new integral MOT metrics are introduced in order to solve the problem of evaluating the MOTA and MOTP of the system across all thresholds, known as AMOTA and AMOTP (Average MOTA and MOTP), as shown in Equation 4.14:

$$AMOTA = \frac{1}{L} \sum_{\{\frac{1}{L}, \frac{2}{L}, \dots, 1\}} (1 - \frac{FP + FN + IDS}{num_{gt}}) \tag{4.14}$$

Where L is the number of different recall values. Note that IDS, FP and FN are modified according to the results of each threshold value. Likewise, AMOTP can be estimated by integrating MOTP across all recall values.

4.3.1.2. MOT leaderboard

Table 4.1: Comparative of Multi-Object Tracking pipelines using the KITTI-3DMOT evaluation tool in the validation set (car class). We bold in **black the best results for each category.**

Method	AMOTA [%]	AMOTP [%]	MOTA [%]	MOTP [%]	IDs
mmMOT [94]	33.08	72.45	74.07	78.16	10
FANTrack [95]	**40.03**	75.01	74.30	75.24	35
Monocular 3D [96]	31.37	64.29	62.38	68.26	**1**
Ours (SmartMOT [83] (tracking only))	39.90	**79.31**	**94.20**	82.06	150

We compare our proposed MOT pipeline (PointPillars as 3D object detector [84], BEV Kalman Filter, with the state space specified in Section 4.2 as data estimator and Hungarian algorithm as data association algorithm) against modern open-sourced 3D MOT systems such as mmMOT [94], FANTrack [95] and Monocular3D [96] using the proposed KITTI-3DMOT. Results are observed in Table 4.1, where we achieve results that are on-pair with other SOTA tracking methods. Note that these results were obtained with default values of the hyperparameters in the tracking stage ($age_{max} = 1$, $min_{hits} = 1$, $IoU_{thr} = 0.1$). For a deeper information of these hyperparameters, we refer the reader to the next subsection, where we conduct an ablation study to analyze the influence of maximum age or minimum threshold in the data association cost matrix to achieve the best tracking results.

4.3.1.3. MOT ablation

Once we decide to implement a specific tracking-by-detection configuration, we carry out an ablation study that allows us to observe the performance in function of the tracking hyperparameters. These are:

- age_{max}**:** Maximum number of frames for a tracker (Kalman Filter) to be associated again to a certain detection

- min_{hits}: Minimum number of consecutive frames in which a tentative tracker must be associated to a detection to be considered as an actual tracker
- IoU_{thr}: Threshold to match a predicted trajectory and a detection in the data association module

Table 4.2 shows an ablation study by modifying these parameters. With a threshold IoU_{thr} of 0.01 we get quite similar results in terms of MOTA and MOTP, decreasing by 36 % the number of identity switches (150 to 54). On the other hand, increasing the minimum number of hits allows us to reduce the identity switching noticeably, overcoming one of the main drawbacks associated to the motion metric proposed by SORT. Moreover, modifying the maximum age to consider a tracker has left the scene barely modifies the studied metrics. Finally, we bold in black the best values for each metric and in blue our final configuration (age_{max} = 1, min_{hits} = 3, IoU_{thr} = 0.1) that achieves an impressive number of 2 identity switches and quite acceptable CLEAR and integral metrics, which are key as a preliminary stage to predict the short-term for each trajectory in the motion prediction stage.

Table 4.2: Ablation study of the final tracking stage configuration of SmartMOT using the KITTI-3DMOT evaluation tool in the validation set (car class). We bold the best results in **black** and the second best in **blue** for each metric.

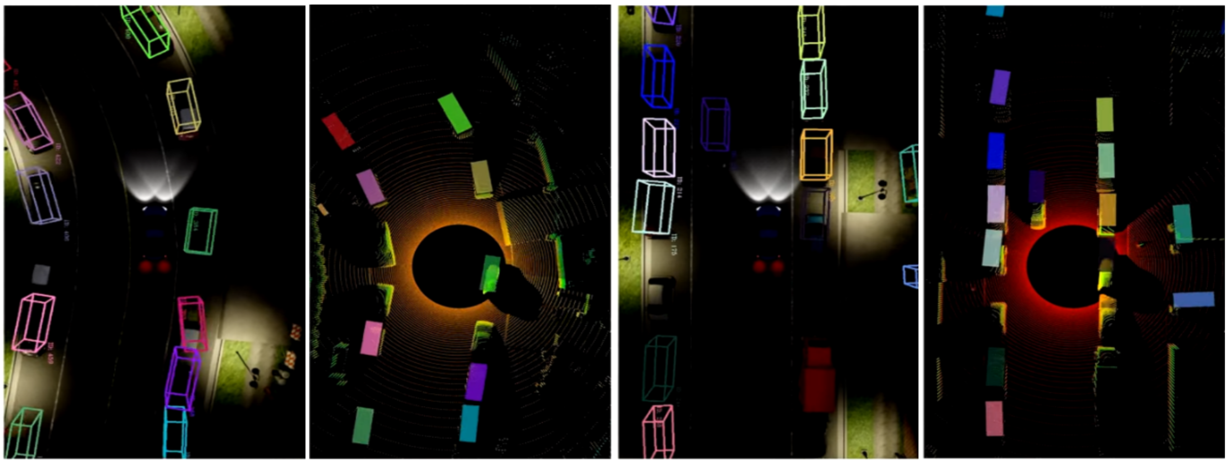
age_{max}	min_{hits}	IoU_{thr}	AMOTA [%]	AMOTP [%]	MOTA [%]	MOTP [%]	IDs
1	1	0.1	**39.90**	79.31	94.20	82.06	150
1	1	0.01	39.84	70.96	95.13	81.84	54
1	1	0.25	39.37	**79.35**	89.10	82.42	682
1	**3**	**0.1**	**39.54**	**71.24**	**91.38**	**83.23**	**2**
1	5	0.1	39.26	71.36	88.84	**83.68**	3
2	1	0.1	39.49	79.24	94.91	81.48	154
3	1	0.1	39.50	79.15	**95.16**	81.15	152

Finally, our final system configuration is as following: We use PointPillars trained over 1,187,840 training steps using the KITTI MOT benchmark database, the BEV Kalman Filter formulated in the previous section, an $IoU_{th} = 0.1$ as the threshold to associate a detection with a tracker the data association module, and $min_{hits} = 3$, $age_{max} = 1$ values for the birth and death module respectively.

4.3.1.4. Qualitative results in CARLA and our campus

In this subsection we may appreciate some qualitative results both in simulation and in our real-world prototype using SmartMOT. One of the best advantages of CARLA is the possibility to create ad-hoc urban layouts by means of an OpenSCENARIO [97] script definition where town, vehicles, climate conditions and also driving behaviours are defined, helpful to validate AD algorithms (specially those focused on the perception layer) under different traffic and weather conditions.

In terms of simulation, we reproduce a very common situation (as observed in the KITTI dataset) which is the ego-vehicle driving in narrow streets full of parked obstacles aside, evaluating its performance in night conditions. Despite this is probably the major disadvantage when using camera information (very poor performance in night conditions), we get impressive results in this situation, as illustrated in Figure 4.6. This is pretty much coherent since LiDAR sensors are not passive sensors like cameras but they supply their own illumination source, which hits objects the reflected energy is detected and measured by the sensor in order to compute the distance to the object.



(a) (b) (c) (d)

Figure 4.6: Detection and Tracking of Multiple Objects in the parked aside vehicles at night traffic scenario considering a curved trajectory (a,b) and straight trajectory (c,d)

On the other hand, in terms of our real-world prototype, we focus on implementing a 360º real-time and power-efficient MOT pipeline in an efficient way. Perception systems in autonomous driving must process a huge amount of information coming from at least one sensor in order to understand the environment. However, the physical space occupied by the processing units in the vehicle or their power consumption are metrics to be deeply analyzed, even more if these processing units will be integrated in an electric vehicle, where the state of the batteries is crucial. In that sense, the current approach is to use powerful but power-efficient AI embedded systems as computation devices for autonomous machines, since they present a remarkable ratio between performance and power consumption in a reduced-size hardware. Regarding the advantage of using neural networks in GPU, these embedded systems present a powerful GPU unit as well as fast storages based on solid state disks and a large RAM memory size. At the time of writing this paper, the best ratio of performance vs power consumption and size is represented by the NVIDIA Jetson embedded computing boards. NVIDIA Jetson is the world's leading AI computing platform for GPU-accelerated parallel processing in mobile embedded systems. These kits allow to implement state-of-the-art frameworks and libraries to conduct accelerated computing, such as CUDA, cuDNN or TensorRT (Tensor RealTime).

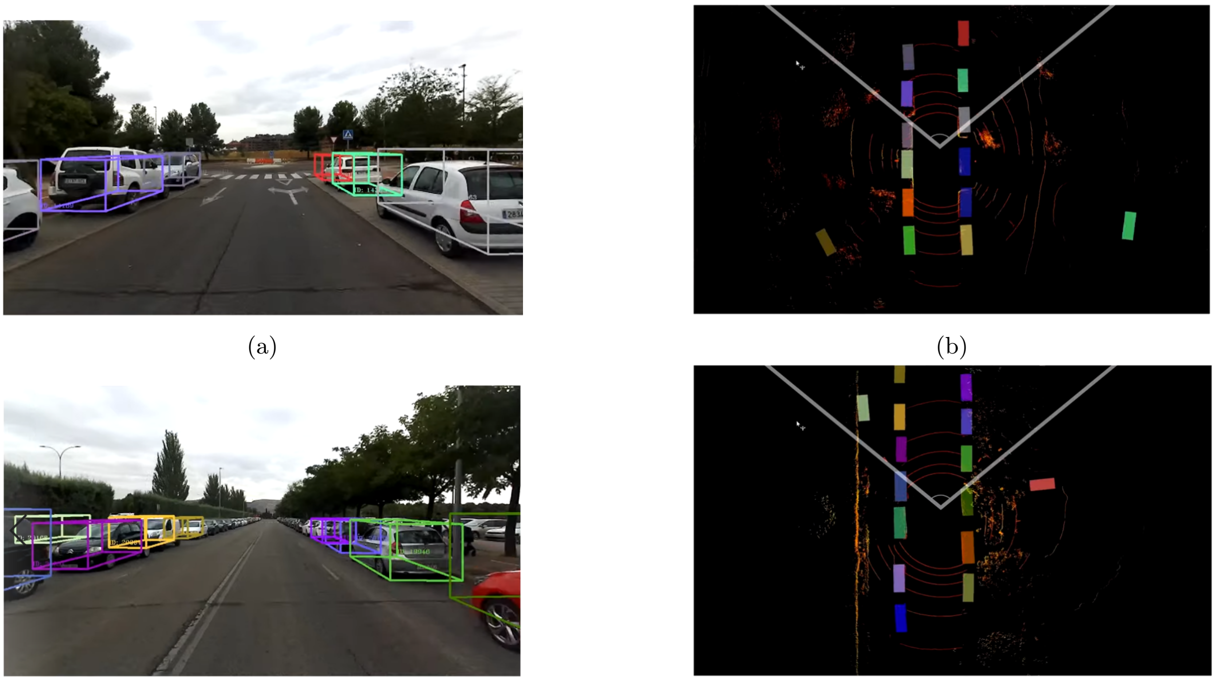
Table 4.3: Comparative of inference frequency for the DAMOT system between the NVIDIA Jetson AGX Xavier and our PC desktop (Intel Core i7-9700, 16GB RAM) with CUDA-based NVIDIA GeForce RTX 1080 Ti 11GB VRAM

Stage	Frequency AGX Xavier (Hz)	Frequency PC desktop (Hz)	Ratio
Detection	7.3	**41.7**	5.7x
Tracking	15	**101.9**	6.7x

In this particular work we make use of the NVIDIA Jetson AGX Xavier, which is as far as we know one of the most powerful AI embedded system specially designed for autonomous machines. Table 4.3 shows a comparative between the embedded system and our PC frequency in the inference stage, where the detection (PointPillars) is reduced by almost 6 times and the tracking by almost 7 times. Nervertheless, although the detection and tracking frequencies are on the border to be considered real-time according to the requirements of the perception systems for autonomous machines, the embedded system consumes 30 W whilst only the 1080 Ti GPU consumes 250 W at full power respectively. Considering that the embedded system computation power is reduced by 6.2 times (average between the detection and tracking frequency ratios) but only the GPU (not considering the whole PC desktop) presents a power consumption 8.3 higher, makes the current NVIDIA Jetson AGX Xavier a better suitable option for large scale-deployment in the autonomous driving field rather than using desktop graphic cards. Distributing several sensor processing across multiple embedded systems for parallelization will result in lower power consumption than using conventional GPUs in future autonomous driving prototypes. Qualitative results of running our DAMOT pipeline in our own vehicle, equipped with a VLP-16 LiDAR instead of the HDL-64 shown in CARLA and KITTI, are illustrated in Figure 4.7. It can be appreciated that although the obtained results are slightly worse than with the KITTI dataset (equipped with a HDL-64 sensor), we obtain quite promising results, validating the pipeline studied in this work both in terms of accuracy and real-time operation.

4.4. Summary

In this chapter we propose SmartMOT, a simple-yet-powerful pipeline that fuses the concepts of tracking-by-detection and HD map information to design a real-time and power-efficient Multi-Object Tracking (MOT) and Motion Prediction pipeline used to track and predict the future trajectories of only the most relevant obstacles around the ego-vehicle, incorporating a Monitored Lanes-based Attention Module to the pipeline, improving the way in which the vehicles are considered as relevant, and an additional module to evaluate different behavioural use cases and background behaviours.



(a) (b) (c) (d)

Figure 4.7: Detection and Tracking of Multiple Objects in our campus with our real-world vehicle

Then, experimental results focus on validating the tracking stage in the KITTI dataset. Once the best hyperparameters are obtain by means of an ablation study, an end-to-end validation of our pipeline in the Unexpected VRU scenario is carried out, following an Euro-NCAP-based validation protocol, illustrating how integrating map information in the pipeline can minimize or at least reduce the impact velocity with the VRU by reducing the computational complexity of the problem. Moreover, a temporal graph is depicted, representing a very intuitive and powerful manner to appreciate how the integration of this extended version of SmartMOT gives the vehicle a valuable time to anticipate the corresponding behaviours. We hope that our distributed pipeline can serve as a solid baseline on which others can build on to advance the state-of-the-art in fusing perception data and map information to perform real-time motion prediction and decision-making evaluation in arbitrarily complex urban scenarios.

Chapter 5

Exploring GAN for Vehicle Motion Prediction

El mundo no es todo alegría y color,
es un lugar terrible y por muy duro que seas
es capaz de arrodillarte a golpes
y tenerte sometido a golpes permanentemente
si no se lo impides.
Ni tú ni yo ni nadie golpea mas fuerte que la vida.
Pero no importa lo fuerte que golpeas,
sino lo fuerte que pueden golpearte
hay que soportar sin dejar de avanzar.
¡Así es como se gana!

Discurso de Rocky a su hijo
Rocky Balboa

5.1. Introduction

Despite the fact that SmartMOT illustrates a simple-yet-powerful tracking and prediction pipeline, traditional methods for MP in the field of AD are based on physical kinematic constraints and road map information with handcrafted rules. Though these approaches are sufficient in many simple situations (*i.e.* vehicles moving in constant velocity or straightforward intersections), they fail to capture the rich behavior strategies and interaction in complex scenarios, in such a way they are only suitable for simple prediction scenes and short-time prediction tasks [18]. In that sense, as commented in Chapter 2.3.2, recently DL based methods have dominated this task and they usually follow an encoder-decoder paradigm.

The main challenge in the MP is the human driver behaviour can neither be modeled and consequently predicted properly, specially in negotiating situations [98] [33] with many participants where considering agent-environment/agent-agent interactions [88] plays a determinant role. Then, resulting trajectories may not be necessarily feasible, not covering the full spectrum of possible trajectories that a vehicle can take. In that sense, a more natural way of capturing the feasible directions [99] is to first compute a set of intermediate target points from a distribution of acceptable positions.

In this chapter we explore the influence of attention mechanisms in generative models, in particular based on GAN [71], to carry out the task of motion prediction. Our model considers both physical context, computing acceptable target points from the driveable area around the target agent, and social context, LSTM [70] based encoder as input to a Multi-head self-attention module, as input of our generator, which combines the scene understanding around the agent vehicle (target agent to predict its trajectory) and the corresponding noise vector associated to generative models to compute the trajectories using a LSTM decoder, as illustrated in Figure 5.1. In this context, the discriminator is applied in order to force the generator model to produce more realistic samples (*i.e.* trajectories), hence, to improve the performance.

Prior knowledge on MP in pedestrian datasets like ETH [100] or UCY [101] usually focuses on deep methods such as LSTMs [70] and GANs [71]. SocialLSTM [32] proposes an LSTM-based model that can jointly predict the paths of all agents in the scene taking into account the common sense rules and social conventions using a social-pooling module. SocialGAN [30] enhances SocialLSTM with a generative adversarial framework, introducing a variety loss that encourage the network to cover the space of plausible paths and proposing a novel pooling global social pooling vector that encodes the subtle cues for all agents involved in the scene. SoPhie [88] considers not only the path history of all agents but also the physical context information (captured by a top-view static image, computing salient regions of the scene), combining physical and social attention mechanisms in order to help the model knows what to extract and where to focus. Goal-GAN [99] predicts the most likely goal points of the agent of interesting, estimating a set of trajectories towards these potential future candidates using both physical and social context, as proposed by [88]. On the other hand, in the context of vehicle prediction [5], [55], prior information takes more importance regarding the risk at certain velocities in urban / highway environments in order to perform safe navigation.

As stated in Chapter 2.3.2, HD maps have been widely adopted to provide a preliminary raw physical context and then apply data-driven approaches. Recent learning-based approaches [31], [89], which present the benefit of having probabilistic interpretations of different behaviour hypotheses, require to build a representation to encode the trajectory and map information. [89] assumes that detections around the vehicle are provided and focuses its work on behaviour prediction by encoding entity interactions with ConvNets. Intentnet [31] proposes to jointly detect traffic participants (mostly focused on vehicles) and predict their trajectories using raw LiDAR pointcloud and rendered HD map information. PRECOG [15] aims to capture the future stochasiticity by flow-based generative models. Furthermore, MultiPath [42] uses ConvNets as encoder and adopts pre-defined trajectory anchors to regress multiple possible future trajectories.

Furthermore, as commented in previous sections, in a similar way to humans that pay more attention to close obstacles, people walking towards them or upcoming turns rather than considering the presence of people or building far away, the perception layer of a self-driving car must be modelled to focus more on the more relevant features of the scene. Social Attention is a mechanism that allows selective interactions within relevant agents. SoPhie [88] computes a different context vector for each agent, in such a way other agents features are sorted in terms of their relative distance to the agent of interest. Then, a soft attention mechanism is used to compute a context feature vector, which represents the social context. Nevertheless, a fixed size (N_{max} agents) list that considers the context of all agents is sensitive to small variations [33] of other agents positions. In that sense, SocialWays [102] presents a hand-crafted relative geometric feature to produce a set of normalized weights, in such a way the context vector represents a convex sum of other feature vectors (context of each agent) that is invariant to the ordering.

However, these attention mechanisms were not designed to model complex interactions, no more than angles and distances due to the inherent problem of pedestrian prediction, in such a way we must find this challenging interactions in the vehicle motion prediction task to account for specific behaviours like overtaking, Adaptive Cruise Control (ACC), emergency braking or yielding. GRIP [103] proposes a graph representation of vehicle neighbours, taking into account local interactions with vehicles that are closer to the target agent than a threshold distance d.

[104] use a dot product attention module (inspired from the attention mechanism proposed by [73] for sentence translation), allowing joint forecast of every agent in the scene without spatial limitations, considering long range interactions regardless the ordering of the input vehicles tracks and the number of vehicles. Moreover, [104] combines this dot product with a spatio-temporal graph representation to take into account temporal and spatial dependencies of the agents, such as their absolute/relative positions and time step movements. [33] present a multi-head extension of this dot attention mechanism, where each agent is embedded by means LSTMs before computing the dot product attention in order to produce social interactions.

5.2. Attention-based GAN

In this work, we aim to develop a model [105] that can successfully predict plausible future trajectories in the context of vehicle prediction, taking into account not only the past trajectory of the corresponding agent but also the HD map information to compute a set of acceptable target points representing the physical constraints for our problem.

When vehicles drive through a traffic scenario, they usually aim to reach partial goals, depending on their predefined navigation route and scene context (both physical and social), until they finally arrive at their final destination. Formally, given a certain goal, vehicles must face different traffic rules and other agents along their way to reach their final destination. Regarding this, our model computes both the social context and acceptable target points for the corresponding agent given its past trajectory and then generates plausible trajectories towards the estimated goals. As illustrated in Figure 5.1, our model consists of three main blocks:

- **Target Points Extractor**: Combines HD Map information and dynamic features of the target agent (speed and orientation) to generate acceptable target points in the driveable area.
- **Attention module**: Computes the agents dynamic features recursively by means a LSTM unit and capture complex social interactions among agents by means of Multi-Head Self Attention.
- **GAN module**: Given the target points and highlighted social features, this module generates plausible and realistic trajectories using a LSTM based decoder, which represents the generator. Discriminator is applied to enhance the performance of the generator by forcing it to compute more realistic predictions.

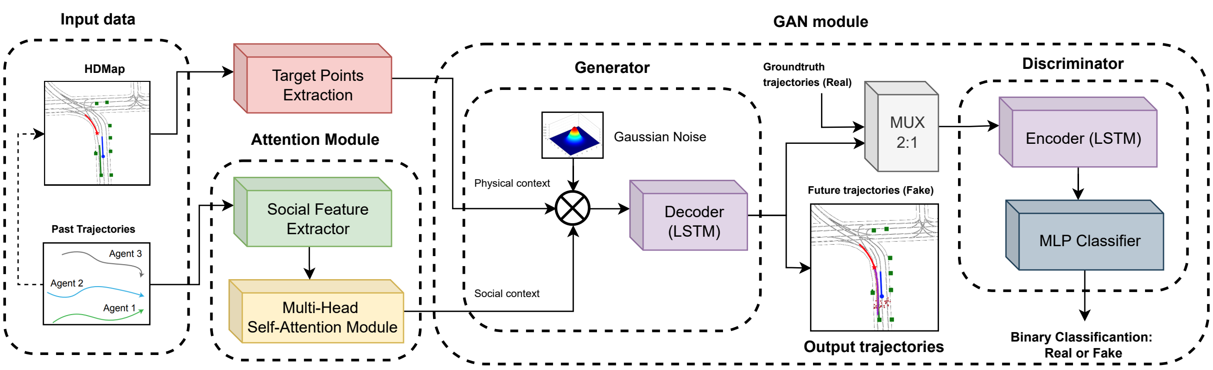


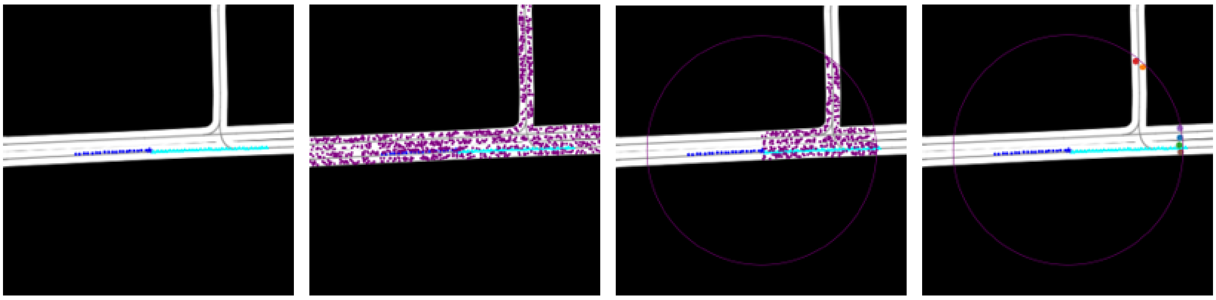
Figure 5.1: Overview of our Attention-based Generative model

Figure 5.1 illustrates an overview of our model. Next, we describe the different blocks of our model.

5.2.1. Target points extraction

Multiple approaches have tried to predict realistic trajectories by means of learning physically feasible areas as heatmaps or probability distributions of the agent future location [8], [88], [99]. These approaches require either a top-view RGB BEV image of the scene, or a HD Map with exhaustive topological, geometric and semantic information (commonly codified as channels). This information is usually encoded using a CNN and fed into the model together with the social agent information [34], [88], [99].

In our model, we propose to estimate the range of motion (360º) using a minimal HD Map representation that includes only the feasible area, where we can discretize the feasible area $\mathcal{F}$ (represented by a discrete grid of the *width* x *height* BEV map image where the pixels are driveable) as a subset of r randomly sampled points $\{p_0, p_1...p_r\}$ from such area in the map (easy to extract from a 1-channel binarized HD image) considering the orientation and velocity in the last observation frame for the agent. This step can be considered as pre-processing of the HD Map, therefore the model never sees the HD map image nor the whole graph of nodes.



(a) (b) (c) (d)

Figure 5.2: Target Points Estimation from the Feasible area process: (a): Agent Past Trajectory (past observations and ground-truth), (b) Feasible area discretization (random points in the driveable area), (c) Non-holonomic-based dynamic filter (both angle and velocity), (d) Multimodal clustering to get the final proposals

Figure 5.2 summarizes step-by-step the whole process. First, we calculate the driveable area (white area in Figure 5.2) around the vehicle considering a hand-defined d threshold.

Then, we consider the dynamic features of the agent of interest in the last observation frame t_{obs} to compute acceptable target points in local coordinates. As we will detail in future sections, the Argoverse 1 Motion Forecasting dataset focuses on estimating the future prediction of a particular target agent. On top of that, the aforementioned dynamic features (orientation and velocity) are not provided, in such a way they must be calculated. Since the trajectory data are noisy with tracking errors, as expected from a real-world dataset, simply interpolating the coordinates between consecutive time steps, assuming constant frequency, results in noisy estimation. Then, in order to estimate the orientation and velocity of the target agent in the last observation frame t_{obs}, we compute a vector for each feature given:

$$\begin{aligned} \theta_i &= \arctan\left(\frac{y_i - y_{i-1}}{x_i - x_{i-1}}\right) \\ v_i &= \frac{X_i - X_{i-1}}{t_i - t_{i-1}} \end{aligned} \tag{5.1}$$

where X_i represents the 2D position of the agent at each observed frame i as state above. Once both vectors are computed, we obtain a smooth estimation as proposed by [106] of the heading angle (orientation) and velocity by assigning less importance (higher forgetting factor) to the first observations, in such a way immediate observations are the key states to determine the current spatio-temporal variables of the agent, as depicted in Equation 5.2, which applies to both the orientation and velocity vector:

$$\hat{\psi}_{T_h} = \sum_{t=0}^{T_h} \lambda^{T_h - t} \psi_t \tag{5.2}$$

where T_h is the number of observed frames, ψ_t is the estimated orientation/velocity at the t_i frame, $\lambda \in (0, 1)$ is the forgetting factor, and $\hat{\psi}_{t_obs}$ is the smoothed orientation/velocity estimation at the last observed frame. After estimating these variables, we calculate the range of motion around the target agent as a circle with radius: $H * \psi_v$ where ψ_v is the estimated velocity using Equation 5.2 and $H = 3$ is the time-horizon of 3s. After that, we randomly sample r points $p \in \mathcal{F}$ in this range considering a constant velocity model during the prediction horizon and the estimated orientation, assuming non-holonomic constraints [51], which are inherent of standard road vehicles, that is, the car has three degrees of freedom, its position in two axes and its orientation, and must follow a smooth trajectory in a short-mid term prediction. Finally, we estimate k target points (one per mode required in the future prediction) by means of the k-means [107] clustering algorithm.

This clustering method, originally from signal processing, that aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean (cluster centers or cluster centroid), serving as a prototype of the cluster. In this particular model, we focus on unimodal prediction, that is, the model must reason the most plausible future trajectory based on the agents past observations, attention-based social interaction and target points as physical context.

This representation not only reunites information about the feasible area around the agent, but also represents potential target points [99] (*i.e.* potential destinations or end-of-trajectory points for the agents). Moreover, this information is *"cheap"* and *interpretable*, therefore, we do not need further exhaustive annotations from the HD Map in comparison with other methods like HOME, which gets as input a 45-channel encoded map [8].

We concatenate this information, as 2D vector $\mathcal{V}$, together with the model social context features to generate more realistic trajectories (see Figure 5.1).

5.2.2. Attention module

Multiple methods [12], [13] consider only the vehicles that are observable at *t=0*, handling those agents that are not observed over the full sequence spectrum (observation length $= obs_{len}$ + prediction length $= pred_{len}$) by concatenating a binary flag b_i^t that indicates if the agent is padded or not.

In our case, we consider the agents that have information over the full history horizon $T_h = obs_{len} + pred_{len}$ (*e.g.* 5s timeframe for Argoverse) as relevant agents, reducing the number of agents to be considered in complex traffic scenarios. Nevertheless, instead of using the past obs_len observations in map (global) coordinates for each agent marked as relevant, we transform to local (relative) coordinates by substracting the last observation of the target agent (considered as the origin of the scene) to the past trajectory of an agent to make the model translation-invariant given the local coordinates the scene. Then of using absolute 2D-BEV (xy plane), the input for the agent i is a series of relative displacements:

$$\Delta\boldsymbol{\nu}_i^t = \boldsymbol{\nu}_i^t - \boldsymbol{\nu}_i^{t-1} \tag{5.3}$$

Where $\boldsymbol{\nu}_i^t$ represents the state vector (in this case, xy position of the agent i at timestamp t.

Unlike other methods, we do not limit nor fix the number of agents per sequence. Given the relative displacements of all different agents, we model the past motion history by means of a single LSTM (Social Feature Extractor in Figure 5.1) is used to compute the temporal information of each agent in the sequence:

$$out, \mathbf{h_{out}}, \mathbf{c_{out}} = \mathrm{LSTM}(\Delta\boldsymbol{\nu}^{obs_{len}}, \mathbf{h_{in}}, \mathbf{c_{in}}) \tag{5.4}$$

This LSTM-based encoder shares the weights for all vehicles in the batch. The input hidden and cell vectors ($\mathbf{h_{in}}, \mathbf{c_{in}}$) are initialized with a tensor of zeros. $\Delta\boldsymbol{\nu}^{obs_{len}}$ represents the relative displacements over the whole past observations obs_{len}.

Then, after computing the social hidden state of each relevant agent in the traffic scenario, we aim to learn complex social interactions, where each agent of the scene should pay attention to specific features around it, while being invariant to their number and ordering, avoiding a fixed size ($N_{\max}$ agents) list which would be sensitive to small variants in the agent positions.

To this end, we make use of a Multi-Head Self-Attention (MHSA) module consists of several heads that given the encoded trajectories produces feature vectors that encode all pairwise relations among agents information, where the input (key, query, value) is the hidden vector ($\mathbf{h_{out}}$) of the previous Social Feature Extractor. The MHSA module consists of several heads that given the encoded trajectories produces feature vectors that encode all pairwise relations among agents information, as stated in Chapter 3.

5.2.3. GAN module

To capture the stochastic nature of motion prediction, state-of-the-art methods leverage the power of generative models, such as Variational Autoencoders (VAEs) and Generative Adversarial Networks (GANs). In our work we use an adversarial framework in order to train our trajectory generator, responsible for generating physically and realistic feasible trajectories. In a GAN, the Generator (which after being trained will be the inference network) and Discriminator networks compete in a two-player min-max game [71], as observed in Equation 5.5. While the generator aims at producing feasible trajectories, the discriminator learns to differentiate between fake and real samples, in other words, ground-truth (which are feasible by definition) and inferred trajectories, in such a way the tasks of the discriminator is to enhance the performance of the generator by forcing it to compute more realistic predictions, more and more similar to the ground-truth trajectory. As a result, the generator should be able to produce outputs which the discriminator cannot discriminate clearly, indicating that the output is realistic.

$$\begin{aligned} \min_{Gen} \max_{Dis} V(Dis, Gen) = E_{X \sim p_{data}(X)}[\log Dis(X,Y)] \\ + E_{z \sim p_z(z)}[\log(1 - Dis(X, Gen(X,z)))], \end{aligned} \tag{5.5}$$

In the present case, the generator, also identified as the routing module, is represented by a decoder LSTM ($LSTM_{gen}$) and the discriminator by a classifier LSTM ($LSTM_{dis}$) so as to estimate the temporally dependent future states. Similar to the conditional GAN proposed by [88], the input to our generator is a concatenation of a noise vector z sampled from a multi-variate normal distribution, being the physical context (goal points in relative coordinates in the last observation frame, $C^{t_{obs}}_{Ph(i)}$) and social context (interactions among agents, $C^{1:t_{obs}}_{So(i)}$) its conditions. Then, the generated future trajectory for a particular agent is modelled as Equation 5.6:

$$\hat{Y}_i^{t_{obs}:t_{pred}} = LSTM_{gen}\Big(C^{t_{obs}}_{Ph(i)}; C^{1:t_{obs}}_{So(i)}; z\Big) \tag{5.6}$$

On the other hand, the input of the discriminator is a randomly chosen trajectory sample from the either predicted future trajectory or ground-truth for the corresponding agent up to $t = t_{obs} + t_{pred}$ frame, i.e. $T_i^{t_{obs}:t_{pred}} \sim p(\hat{Y}_i^{t_{obs}:t_{pred}}, Y_i^{t_{obs}:t_{pred}})$.

$$\hat{L}_i^{t_{obs}:t_{pred}} = LSTM_{dis}(T_i^{t_{obs}:t_{pred}}) \tag{5.7}$$

Then, the discriminator returns a label $\hat{L}_i^{t_{obs}:t_{pred}}$ for the chosen trajectory indicating whether the trajectory is ground-truth (real) $Y_i^{t_{obs}:t_{pred}}$ or predicted (fake) $\hat{Y}_i^{t_{obs}:t_{pred}}$, being the labels 0 and 1 for fake and real trajectories respectively. Equation 5.7 summarizes the discriminator working principles.

5.2.4. Losses

To train our GAN-based model, we use the following losses:

$$\begin{aligned} W^* = \underset{W}{\operatorname{argmin}} \quad & \mathbb{E}_{i,\tau}[\lambda_{gan}\mathcal{L}_{GAN}\left(\hat{L}_i^{t_{obs}:t_{pred}}, L_i^{t_{obs}:t_{pred}}\right) + \\ & \lambda_{ADE}\mathcal{L}_{ADE}(\hat{Y}_i^{t_{obs}:t_{pred}}, Y_i^{t_{obs}:t_{pred}}) + \\ & \lambda_{FDE}\mathcal{L}_{FDE}(\hat{Y}_i^{t_{obs}+t_{pred}}, Y_i^{t_{obs}+t_{pred}})], \end{aligned} \tag{5.8}$$

where W is the collection of the weights of all networks used in our model and the different λ represent the corresponding regularizers between these losses. As stated in Equation 5.5, $\mathcal{L}_{GAN}$ represents the min-max game where the generator tries to minimize the function while the discriminator tries to maximize it. ADE loss function is commonly used to compute the average error between the predicted trajectories and the corresponding ground-truth. Moreover, we add FDE loss function to explicitly optimize the distribution towards the final real point.

5.3. Experimental Results

5.3.1. Dataset

We evaluate this model on the well-established and public available Argoverse 1 Motion Forecasting dataset [5], including the training, validation and testing subsets from its official website [108].

As stated in Section 2.4.1, it consists of 205942 training samples, 39472 validation samples and 78143 test samples. Each sample has a length of 5 seconds, with an observation window of 2 seconds and a prediction window of 3 seconds, including the corresponding labels of the agents ($AGENT$, as the target agent, AV, the vehicle that captures the scene and $OTHER$, representing the remaining relevant obstacles) and a global map from the cities of Pittsburgh and Miami. The sampling frequency is 10Hz. The main goal here is to predict the 3s future position of the target agent in the scene, which is supposed to be the vehicle that faces the most challenging traffic scenarios.

5.3.2. Metrics

Previous works [33], [42], [88] report the minimum Average Displacement Error (minADE_K), which averages the $L2$ distances between the ground truth and predicted output across all timesteps and minimum Final Displacement error (FDE_K), which computes the L_2 distance between the final points of the ground-truth and the predicted final position, taking the best K trajectory sample of each agent compared to the ground truth. In this model, since we do not focus on multimodal prediction but on the predicting the most plausible unimodal trajectory, we use $K = 1$ (unimodal case).

5.3.3. Implementation details

All local test were conducted in a PC desktop (AMD Ryzen 9 5900X, 32GB RAM with CUDA-based NVIDIA GeForce RTX 3090 24GB VRAM, Ubuntu 18.04).

We design our dataloader to sample in each batch a 30/70 proportion of straight and curved trajectories (regarding the target agent whole trajectory). We classify a trajectory as straight or curve estimating a first degree trajectory by means the RANSAC algorithm with the highest number of inliers (tolerance t set to 2m, max trials=30, min samples=60 % total observations). Then, if the actual trajectory presents 20 % or more consecutive points further than t with respect to the closest point of the fitted trajectory, the whole sequence is labelled as curve. We do this to focus in the training process in non-linear prediction, which represents one the key challenges in vehicle motion prediction.

Regarding the ablation study, we train the different models for 150 epochs using the ADAM optimizer with learning rate 0.001 and default parameters, linear LR Scheduler with factor 0.5 decay on plateaus (5k iterations) and batch size 64. The loss function is weighted by setting λ_{gan}=1.4, λ_{ade}=1 and λ_{fde}=1.5, giving more importance to the adversarial loss and the final displacement error. Similar to [88], the LSTM encoder (attention block) encodes trajectories using a single layer MLP with an embedding dimension of 16. We set all LSTM units to have 32 hidden dimensions. The number of target points is set also to 32 in order to compute the physical context. Moreover, in order to calculate these target points we consider the same prediction horizon $t_{pred} = 3s$ to estimate the distance travelled assuming a constant velocity model. To make our model more robust to scene orientation, we augment the training data adding some white noise ($\mu = 0, \sigma = 0.25$, [m]) to the observation data, rotating the scene and also dropping and replacing (with their last frame) some observations of the past trajectory in order to make the trained model general enough so as to perform well on the unseen traffic scenarios in the split test which different scene geometries such as left/right turning or emergency braking.

5.3.4. Statistical study of the baseline in validation

In terms of validation, we conduct a statistical analysis for the ADE and FDE metrics, distinguishing the performance between straight and curved trajectories for our baseline model, *i.e.* without considering target goals as map information and class balance in the batch. To this end, we make use of the Argoverse 1 validation set that consists of 31000 samples where 23012 and 7988 are straight and curved trajectories respectively given the RANSAC-based classification aforementioned. We show in Figure 5.3 the boxplots for the ADE and FDE metrics. As stated before, our method, as most methods, struggles with curved trajectories, the overall ADE and FDE is "always" better for the straight trajectory cases. The median provides a robust estimator of our trajectories error. Note that we detected multiple outliers in our analysis, these are due to the unimodal nature of the model, unable to consider multiple possible hypotheses (multi-modal).

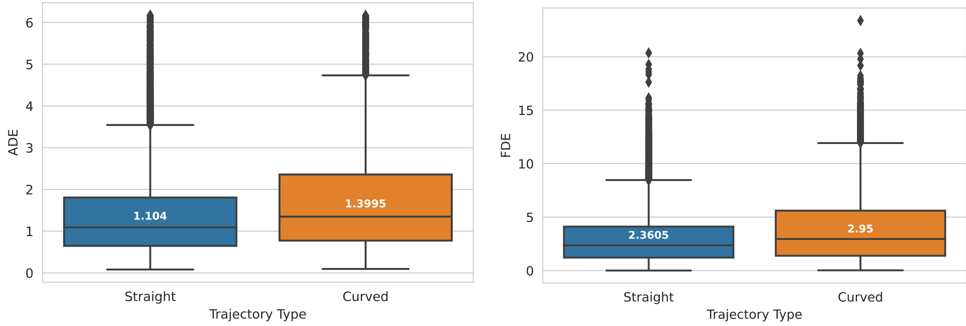


Figure 5.3: Statistical distribution on the Argoverse 1 validation set of regression metrics in straight and curved trajectories. We show the boxplots for ADE and FDE metrics. We distinguish between straight and curved trajectories. We highlight the median (Q2) in each boxplot.

5.3.5. Comparative with the state-of-the-art

In this section, we perform an ablation study and compare our method performance against SOTA methods on the Argoverse 1 Motion Forecasting benchmark test set. Table 5.1 illustrates the comparison with some Argoverse baseline methods. Our baseline (*) is represented by the system pipeline illustrated in Figure 5.1, that is, Attention-based GAN with LSTM as encoder-decoder, without target points extractor and class balance. We conduct an ablation study to observe the influence of incorporating target points and class balance to our baseline. As expected, by explicitly defining the locations an agent is likely to be at a fixed prediction horizon for a given input trajectory and scene geometry, we are able to improve our baseline. Additionally, since nonlinear trajectories are more challenging than standard straight trajectories, we also observe how enforcing the class balance (straight, curve) during training is able to improve performance.

5.3.6. Qualitative results

Figure 5.4 illustrates some qualitative results, all of them considering uni-modal prediction towards the pre-computed target points, meeting the physical and social constraints in the scenes. It can be clearly appreciated that a naive CTRV model could not generalize in these situations, where the vehicle can describe a curved future trajectory given a predominant straight input trajectory and vice-versa. First and second rows show feasible predicted trajectories, whilst the third one shows interesting challenging scenarios in

Table 5.1: Ablation study of our GAN-based unimodal pipeline, and comparison with other relevant methods on Argoverse 1 Motion Forecasting validation set. We can see the improvement using Target points (TP) and Class balance (CB). Our methods are indicated with †.

Model	**ADE (k=1) ↓** [m]	**FDE (k=1) ↓** [m]
Constant Velocity [5]	3.53	7.89
Argoverse Baseline (NN) [5]	3.45	7.88
Argoverse Baseline (LSTM) [5]	2.96	6.81
SGAN [30]	3.61	5.39
TPNet [109]	2.33	5.29
TPNet-map [109]	2.33	4.71
Jean (1st) [5], [33]	1.74	4.24
† Baseline (*)	1.98	4.47
† + TP	1.78	4.13
† + CB	1.82	4.09
† + TP + CB	1.67	3.82

which our current approach is not able to properly reason due to the lack of reasoning and multi-modality (producing a set of K-trajectories towards the set of target points) is revealed, being situations in which predicting accurately the end-point is difficult, and this the Final Displacement Error (FDE) is extremely high.

In that sense, regarding the third row, left image shows how we compute an uni-modal prediction in the wrong direction of a split, even though target points are extracted very close to the ground-truth end point. Center image shows an extreme difficult situation, where the input trajectory is almost in the same place (probably the target agent was stopped in front a traffic light) whilst the ground-truth future trajectory is clearly an acceleration since the last observation frame. Finally, right image shows a deceleration because of the ahead obstacle whilst our model is not able to properly reason the presence of this obstacle in order to meet common sense safety constraints.

5.4. Summary

Forecasting the future trajectories of surrounding actors in the scene is mandatory to achieve a safe planning, and thus, a crucial part of the Autonomous Driving stack. In this Chapter we explore a GAN-based LSTM with MHSA for unimodal vehicle MP using the Argoverse 1 Motion Forecasting Benchmark. Our model considers both the deep physical and social context of the scene to predict the most plausible trajectory using a generative model, and achieves competitive results in comparison to other state-of-the-art methods regarding the case of unimodal prediction. Given the aforementioned results, we realize that this uni-modal method lacks of plausible multi-modality and capacity to model complex traffic scenarios, so the future steps are an enhanced social attention and interaction, high-level and well-structured physical context, specially focusing on

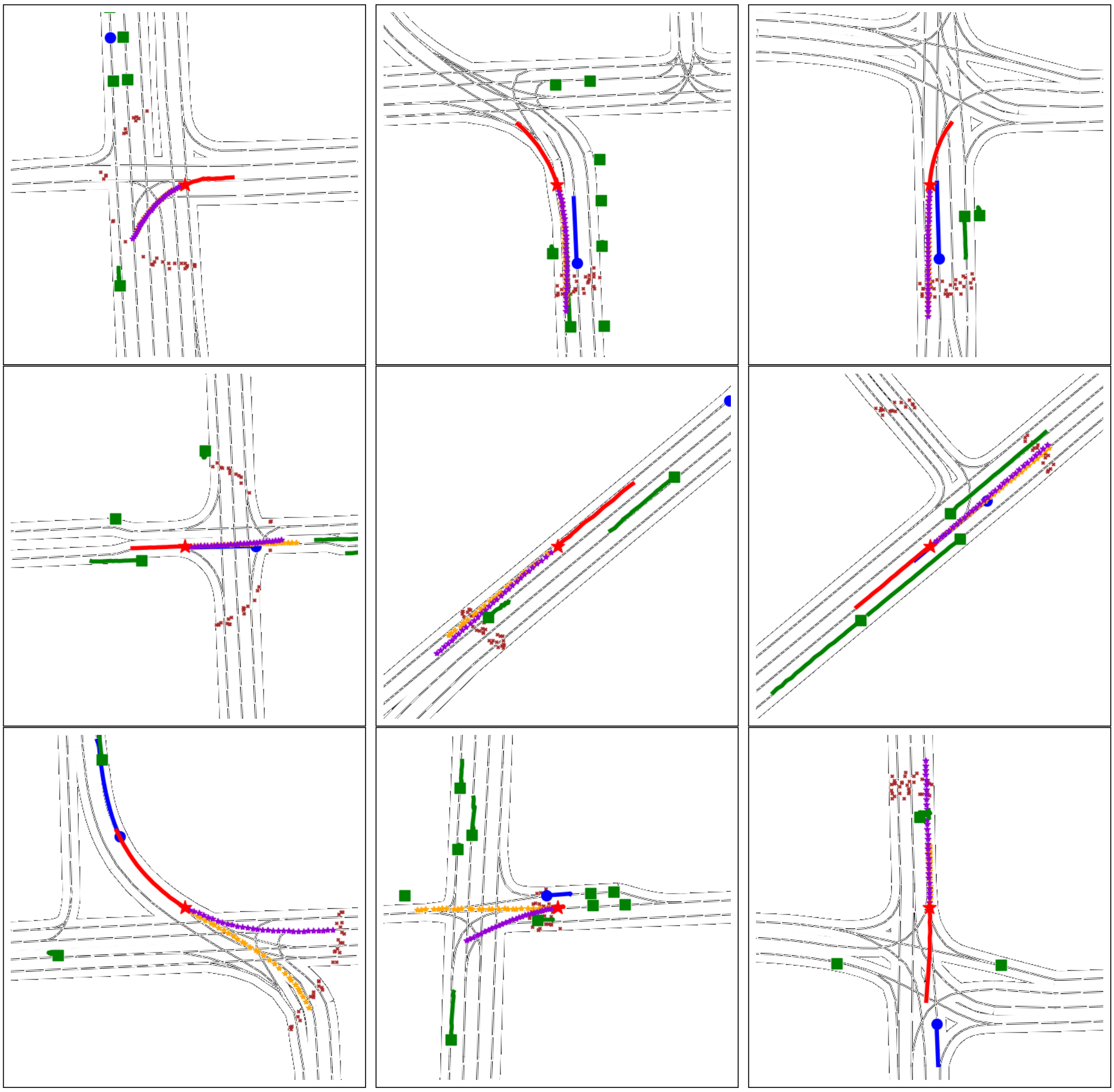


Figure 5.4: Qualitative Results using our Attention-based GAN best model (including target points extraction and class balance). The legend is as follows: our vehicle (ego), the target agent, and other agents. We can also see the real trajectory, the prediction and potential goal-points. Markers are current positions.

the vector features of HD Map, in order to produce feasible and realistic multi-modal trajectories.

Chapter 6

Efficient Baselines for Multi-modal Motion Prediction in Autonomous Driving

La fuerza de tus convicciones
determina tu éxito,
no el número de tus seguidores.

Reamus Lupin
Harry Potter y Las Reliquias de la Muerte, Parte 2

6.1. Introduction

As observed in the previous Chapter, our GAN-based model (more specifically the generator) was able to compute the deep context regarding the agents past observations, attention-based social interaction and target points as physical context, but the prediction was limited to the unimodal case. In other words, the GAN-based model is able to reason more complex interactions and future behaviours than SmartMOT (physics-based prediction), but it lacks one of the main features of a deep learning-based model as a preliminary stage before the local planning or decision-making layers: Multimodality. On top of that, at this point of the thesis the literature was re-visited and despite GANs-based approaches [30], [88], [99], [105] provide certain control since they are focused on more simple methods framed in an adversarial training, most competitive approaches on MP benchmarks in the field of AD, such as Argoverse [5], NuScenes [55] or Waymo [54], do not use adversarial training, where the training complexity is one the main reasons.

6.2. Efficient Baselines

Considering the trade-off between curated input data and complexity, we aim to achieve competitive performance in the MP using powerful DL techniques in terms of prediction metrics (minADE, minFDE), including attention mechanisms and GNNs, while reducing the number of parameters of operations with respect to other SOTA methods. In particular, we propose two baselines, social and map baseline.

The only inputs for the social baseline are the agent past trajectories and their corresponding interactions. On the other hand, for the map baseline, we propose an extension with respect to our previous target points proposals where, based on a simple-yet-powerful map pre-processing algorithm where the corresponding agent trajectory is initially filtered, the feasible area with which the agent can interact is computed. In spite the fact that topological, semantic and geometric information are involved while computing this area due to road connectivity, we only retrieve the geometric information of the feasible area proposals in an efficient and elegant way. Therefore, our models do not require full-annotated (including, topological, geometric and semantic) HD Map information as input or even rasterized BEV representations of the scene to compute the physical context. Figure 6.1 illustrates an overview of our final approach. We distinguish three main blocks: 1) **Encoding module**, which uses plausible HD Map information (specific centerlines and driveable area around) and agents past trajectories to compute the motion and physical latent features, 2) **Social Attention module**, which calculates the interaction among the different agents and returns the most relevant social features, 3) **Decoding module**, responsible for calculating the multimodal prediction by means of an auto-regressive strategy concatenating low-level map features and social features as a baseline, as well as iterating over the different latent centerlines for specific physical information per mode.

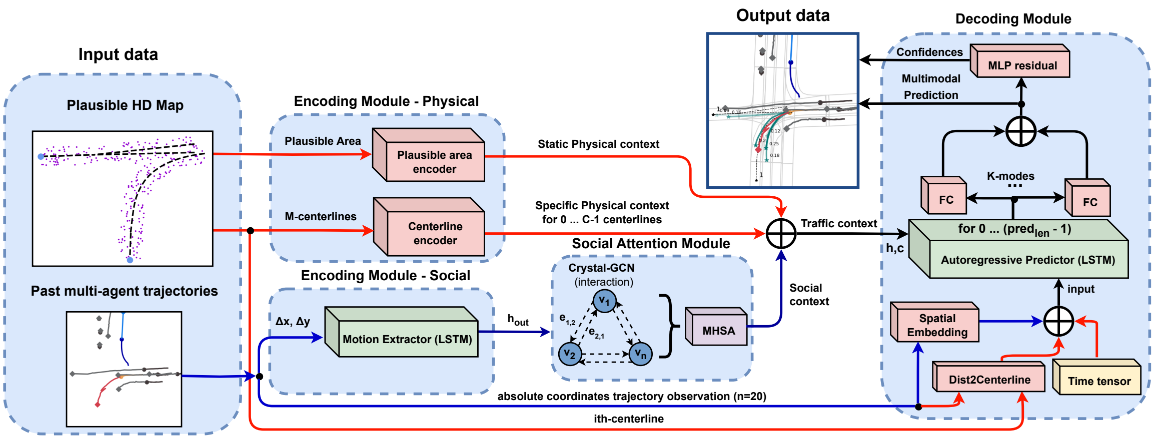


Figure 6.1: Overview of our Social and Map Efficient Baselines
(**Blue links** and **Red links** represent **Social** and **Map** information respectively).

6.2.1. Social Baseline

Our social baseline is inspired in the architecture proposed by [12]. It uses as input the past trajectories of the most relevant obstacles as relative displacements to feed the Encoding Module (Figure 6.1). Then, the social information is computed using a GNN, in particular Crystal-Graph Convolutional Network (**Crystal-GCN**) layers [12], [110], and Multi-Head Self Attention (MHSA) [73] to obtain the most relevant agent-agent interactions. Finally, we decode this latent information using an **autoregressive** strategy where the output at the *i-th* step depends on the previous one for each mode respectively in the Decoding Module. The following sections provide in-depth description of the aforementioned modules.

6.2.1.1. Preprocessing and Encoding of past trajectories

In a similar way to the GAN-based model proposed in Chapter 5, in these efficient baselines we only consider the agents that have information over the full history horizon $T_h = obs_{len} + pred_{len}$, reducing the number of agents to be considered in complex traffic scenarios. Nevertheless, instead of only substracting the origin of the scene (last position of the target agent) to make the model translation-invariant, we also rotate the whole coordinate system, *i.e.* the coordinate system in our model is centered on the target agent at $t = 0$, and we use the orientation from the target location in the same timestamp as the positive x-axis. Note that this representation will benefit the model to have a common representation to enhance the generalization of the model and prevent over-fitting. Once the scene has been translated and rotated, instead of using absolute 2D-BEV (xy plane), the input for the agent i is a series of relative displacements.

Then, as stated in Chapter 5, we do not limit nor fix the number of agents per sequence, and a single LSTM-based encoder computes the deep features of the motion history of the agents. Finally, in order to feed the agent-agent interaction module (Social Attention Module in Figure 6.1), we take the output hidden vector ($\mathbf{h_{out}}$).

6.2.1.2. Social Attention module

After encoding the past history of each vehicle in the sequence, we compute the agent-agent interactions to obtain the most relevant social information of the scene. For this purpose, we construct an interaction graph using Crystal-GCN [110] [12]. , originally developed for the prediction of material properties, allowing to efficiently leverage edge features. Then, MHSA [73] is applied to enhance the learning of agent-agent interactions.

Before creating the **interaction mechanism**, we split the temporal information in the corresponding scenes, taking into account that each traffic scenario may have a different number of agents. The interaction mechanism is defined in [12] as a bidirectional fully-connected graph, where the initial node features $\mathbf{v}_i^{(0)}$ are represented by the latent temporal information for each vehicle $\mathbf{h}_{i,out}$ computed by the motion history encoder. On the other hand, the edges from node k to node l is represented as the vector distance ($\mathbf{e}_{k,l}$) between the corresponding agents at t = obs_{len} in absolute coordinates, where the origin of the sequence ($x = 0, y = 0$) is represented by the position of the target at t = obs_{len}:

$$\mathbf{e}_{k,l} = \boldsymbol{\nu}_k^{obs_{len}} - \boldsymbol{\nu}_l^{obs_{len}}, \tag{6.1}$$

Given the interaction graph (nodes and edges), the Crystal-GCN, proposed by [110], is defined as:

$$\mathbf{v}_i^{(g+1)} = \mathbf{v}_i^{(g)} + \sum_{j=0:j\neq i}^{N} \sigma\left(\mathbf{z}_{i,j}^{(g)}\mathbf{W}_{\mathrm{f}}^{(g)} + \mathbf{b}_{\mathrm{f}}^{(g)}\right) \odot \mu\left(\mathbf{z}_{i,j}^{(g)}\mathbf{W}_{\mathrm{s}}^{(g)} + \mathbf{b}_{\mathrm{s}}^{(g)}\right) \tag{6.2}$$

This operator, in contrast to many other graph convolution operators [49] [13], allows the incorporation of edge features in order to update the node features based on the distance among vehicles (the closer a vehicle is, the more is going to affect to a particular node). As stated by [12], we use $L_g = 2$ layers of the GNN ($g \in 0, \ldots, L_g$ denotes the corresponding Crystal-GCN layer) with ReLU and batch normalization as non-linearities between the layers. σ and μ are the sigmoid and softplus activation functions respectively.

Moreover, $\mathbf{z}_{i,j}^{(g)} = (\mathbf{v}_i^{(g)}||\mathbf{v}_j^{(g)}||\mathbf{e}_{i,j})$ corresponds to the concatenation of two node features in the g_{th} GNN layer and the corresponding edge feature (distance between agents), N represents the total number of agents in the scene and $\mathbf{W}$ and $\mathbf{b}$ the weights and bias of the corresponding layers respectively.

After the interaction graph, each updated node feature $\mathbf{v}_i^{(L_g)}$ contains information about the temporal and social context of the agent i. Nevertheless, depending on their current position and past trajectory, an agent may require to pay attention to specific social information. To model this, we make use of a scaled dot-product Multi-Head Self-Attention mechanism [73] which is applied to the updated node feature matrix $\mathbf{V}^{(L_g)}$ that contains the node features $\mathbf{v}_i^{(L_g)}$ as rows.

Each head $h \in 1, \ldots, L_h$ in the MHSA mechanism is defined as:

$$\mathrm{head}_h = \mathrm{softmax}\left(\frac{\mathbf{V}_{Q_h}^{(L_g)}\mathbf{V}_{K_h}^{(L_g)T}}{\sqrt{d}}\right)\mathbf{V}_{V_h}^{(L_g)}. \tag{6.3}$$

where $\mathbf{V}_{Q_h}^{(L_g)}$ (Query), $\mathbf{V}_{K_h}^{(L_g)}$ (Key) and $\mathbf{V}_{V_h}^{(L_g)}$ (Value) represent the h_{th} head linear projections of the node feature matrix $\mathbf{V}^{(L_g)}$ and d is the normalization factor corresponding to the embedding size of each head. For our purpose, we use $L_h = 4$ as the total number of heads.

The result of the softmax weights multiplied by the node feature matrix $\mathbf{V}_{V_h}^{(L_g)}$ (Value) is often referred as the attention weight matrix, representing in this particular case pairwise dependencies among vehicles.

Finally, the updated node feature matrix **SATT** is computed as the combination of the different attention heads in a single matrix:

$$\mathbf{SATT} = (\text{head}_1|| \dots ||\text{head}_{L_h})\mathbf{W}_\text{o} + \begin{pmatrix} \mathbf{b}_\text{o} \\ \vdots \\ \mathbf{b}_\text{o} \end{pmatrix}. \tag{6.4}$$

Where each row of the final social attention matrix **SATT** (output of the social attention module, after the GNN and MHSA mechanisms) represents the interaction-aware feature of the agent i with surroundings agents, considering the temporal information under the hood, being $\mathbf{W}_\text{o}$ / $\mathbf{b}_\text{o}$ the corresponding weight and bias of the layer that merges the different attention heads. As this model has been developed upon the Argoverse 1 Motion Forecasting benchmark, we **only consider** the row of the final matrix that takes into account the interactions of the target agent with surrounding obstacles.

6.2.2. Map Baseline

As mentioned before, in this work we extend our social baseline using minimal HD map information, from which we discretize the feasible area $\mathcal{P}$ of the target agent as a subset of r randomly sampled points $\{p_0, p_1 ... p_r\}$ (low-level features) around the plausible centerlines (high-level and well-structured features) considering the velocity and acceleration of the corresponding agent in the last observation frame, as observed in Figure 6.2. As stated in Section 5.2, this is a map preprocessing step, therefore the model never sees the HD map (either vectorized or rasterized) image nor the whole graph of nodes.

6.2.2.1. Centerlines proposals and Feasible area points

In a similar way to the target points computed in our GAN-based model, we want to compute the heuristic proposals for each agent. Nevertheless, instead of limiting the map baseline to some discrete target points, now we aim to compute the most plausible future centerlines (that is, the center of the lane) as a connection of nodes (waypoints). Considering lane connectivity, multiple approaches have tried to predict realistic trajectories by means of learning physically feasible areas as heatmaps or probability distributions of the agent's future location [8], [88], [99]. [13] represents the map as a set of lanes and their connectivity (predecessor, successor, right neighbour, left neighbour), taking into account all lanes whose distance from the target agent is smaller than 100 m as the input, regardless the orientation or the velocity of the vehicle. On the other hand, [111] encodes static elements such as crosswalks, lane, road boundaries and intersections that are included in

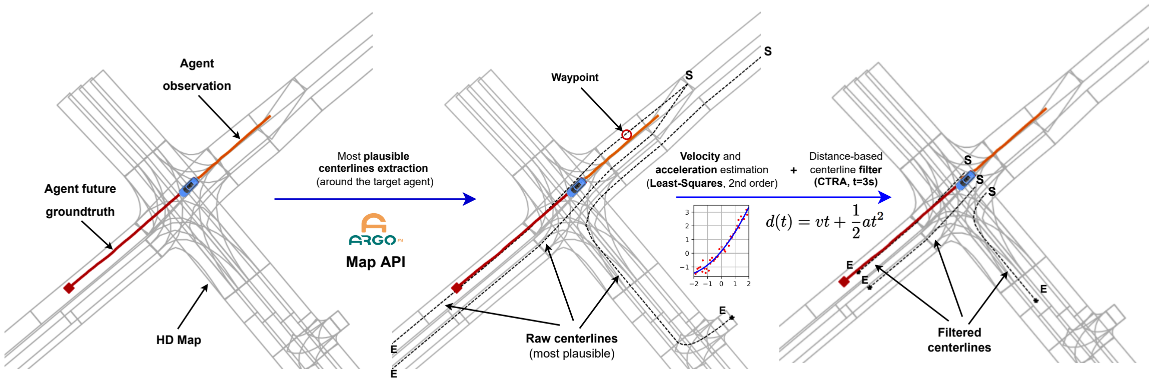


Figure 6.2: Plausible centerlines estimation. Left: General view of the scene, only considering the target agent (**observation (2 s)** and **future ground-truth (3 s)**) and HD Map around its last observation (position of the **blue** vehicle). Center: **Centerlines** proposed by the Argoverse Map API (maximum number of centerlines M set to 3). Right: We filter the input observation by means of Least-Squares (2nd order) algorithm to estimate the velocity and acceleration of the agent. Then, the distance considering a CTRA (Constant Turn Rate Acceleration) model and a prediction horizon of 3 s are used computed to obtain the end-points **E** of the **final proposals**. Start-points **S** are the closest centerlines waypoints to the agent in the last observation frame.

a local map 150m x 100m centered in the corresponding vehicle as a multi-channel image. These approaches require either a top-view RGB BEV image of the scene, or a HD map with exhaustive topological, geometric and semantic information (commonly codified as channels). This information is usually encoded using a CNN and fed into the model together with the social agent's information [34], [88], [99].

As observed, when trying to utilize HD map information, specially in terms of lane proposals, most SOTA methods utilize this physical to enhance the latent information to decode the future trajectories, but heavy computation or raw data features are required.

Effectively and **efficiently** exploiting HD maps is a must for MP models to produce accurate and plausible trajectories in real-time applications, specially in the field of AD, providing specific map information to each agent based on its kinematic state and geometric HD map information. In that sense, we propose to obtain the most plausible M lane candidates, we make use of the pertinent heuristic functions proposed by the Argoverse 1 Motion Forecasting dataset Map API [5] as illustrated by [16] to choose the closest centerlines to the last observation data of the target agent, which are going to represent its most representative future trajectories. On the other hand, as depicted in the MP dataset used to train this model (Argoverse 1), vehicles are the only evaluated object category as the target agent. Then, considering a vehicle as a rigid structure with non-holonomic [51] features (no abrupt motion changes between consecutive timestamps) and the road driving task is usually described as anisotropic [112] (most relevant features are found in a specific direction, in this case the lanes ahead). In other words, the agent should follow a smooth trajectory in a short-mid term prediction. The heuristic to obtain these centerlines is summarized as follows:

1. Trajectory data presents noise associated to the real-world data capturing the exact position of the previously tracked vehices in real-world scenarios. Regarding this, we filter the agent trajectory as a polynomial curve fitting problem by means of the Least Squares (2^{nd} order) per axis and Savitzky-Golais [113] algorithms to obtain a smooth representation of the position vector.

2. By doing so, and assuming the agent is moving with a constant acceleration, we are able to calculate the subsequent derivatives (velocity and acceleration) of the target agent in $t_{obs_{len}}$. Then, a vector of obs_{len} - 1 and obs_{len} - 2 length is computed to estimate the velocity and acceleration respectively as $V_i = \frac{X_i - X_{i-1}}{t_i - t_{i-1}}$ and $A_i = \frac{V_i - V_{i-1}}{t_i - t_{i-1}}$, where $X_i = [x_i, y_i]$ represents the 2D position of the agent at each observed frame i.

3. In order to compute the velocity, acceleration and yaw angle in the last observation frame, we compute a weighted mean by assigning less importance (weight) to the first positions of the corresponding vector and higher importance to the latter states, in such a way immediate past observations are the key states to determine the current spatio-temporal variables of the agent, as depicted in Equation 5.2.

4. We compute the future travelled distance by means of the well-known Constant Acceleration (CA) model:

$$d(t) = x_0 + vt + \frac{1}{2}at^2 \tag{6.5}$$

   where t corresponds to the prediction horizon t_pred, x_0 is equal to 0 since we want to determine the travelled distance from the current position and v and a are the velocity and acceleration in the last observation frame previously calculated. Note that we assume that this is a CTRA model, instead of only CA in a specific direction, since the orientation is implicit in the lane boundaries. That's why it does not make sense to involve the orientation at frame $t = 0$ in the travelled distance calculation.

5. Get all lane candidates within a bubble, given the agent last observation and Manhattan distance.

6. Expand the bubble until at least 1 lane is found.

7. Once some preliminary proposals are found, we employ the Depth First Search (DFS) algorithm to get all successor and predecessor candidates, merging the past and future candidates and removing the overlapping ones.

8. Then, we process these raw candidates so as to use them as plausible physical information. Given these raw lanes, we aim to limit the number of centerlines to a fixed number M. First, given the previously computed smoothed trajectory, we compute the closest centerlines to our current position since they will represent the most realistic future lanes in the traffic scenario. Second, we evaluate the above-mentioned

travelled distance along the raw centerlines. We determine the end-point index p of the centerline m as the waypoint (each discrete node of the centerline) where the accumulated distance (considering the $\mathcal{L}_2$ distance between each waypoint) is greater or equal than the above-computed $d(obs_{len})$:

$$p \quad : \quad d(obs_{len}) \leq \sum_{p=start_{point}}^{centerline_{length}} \mathcal{L}_2(w(p+1), w(p)) \tag{6.6}$$

9. Finally, in order to have the same points (particularly, the number of points matches the prediction horizon t_pred, also referred as $pred_len$) per centerline, we interpolate them using a 1^{st} spline order, considering as start point the last agent observation and as end or goal point the aforementioned travelled distance along the corresponding centerline.

10. Note that if the number of proposed centerlines is lower than a pre-defined number M, a virtual centerline is created and padded with zeros.

Figure 6.2 summarizes this HDMap filtering process, where we are able to estimate the preliminary centerlines proposals as a **simplified version of the HD map**. Moreover, Fig. 6.3 illustrates how after our filtering process the end-points of the plausible centerlines are noticeable closer to the ground-truth prediction at the final timestep.

In addition to these **high-level** and well-structured centerlines, we apply point location perturbations to all plausible centerlines under a $\mathcal{N}(\mu, \sigma)$ [m] distribution [114] in order to discretize the plausible area $\mathcal{P}$ as a subset of r randomly sampled points $\{p_0, p_1 ... p_r\}$ (**low-level** features) around the plausible centerlines. By doing this, we may have a common representation of the plausible area, defined as low-level map features. We make use of a normal distribution $\mathcal{N}$ to calculate these random points as an additional regularization term in a similar way that data augmentation is applied to the past trajectories.

6.2.2.2. Encoding module of map information

In order to calculate the latent map information, we employ a plausible area and centerline encoder (Figure 6.1) to process the low-level and high-level map features respectively. Each of these encoders are represented by a Multi-Layer Perceptron (MLP). First, we flat the information along the points dimension, alternating the x-axis and y-axis information. Then, the corresponding MLP (three layers, with batch normalization, interspered ReLUs and dropout in the first layer) transforms the interpretable absolute coordinates around the origin $(x = 0, y = 0)$ into representative latent physical information. The static physical context (output from the plausible area encoder) will serve as a common latent representation for the different modes, whilst the specific physical context will illustrate specific map information for each mode.

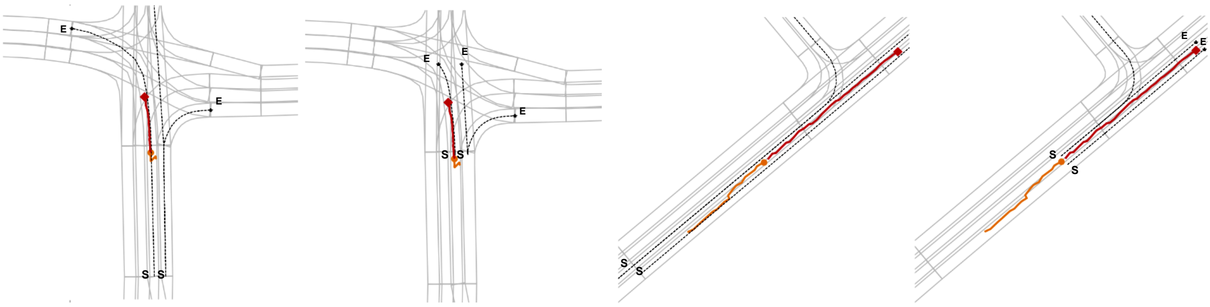


Figure 6.3: Some challenging examples of our preprocessing step to obtain relevant map features. In both scenarios the target agent (observation (2 s) and **future ground-truth (3 s)**) presents a noticeable noisy past trajectory and the provided raw **centerlines** do not consider the current kinematic state of the vehicle. (a) The agent is stopped (maybe due to an stop, pedestrian crossing or red traffic light use case). We estimate a minimum travelled distance of 25 m in these situations to determine the centerline end-points **E**. (b) In this scenario, we can observe how the raw centerlines consider way more distance (both ahead and behind) than required. Our kinematic-based filter is able to minimize these proposals in an interpretable way to serve as prior information to the MP model

6.2.3. Augmented Efficient baseline with Transformer Encoders

Once the social and map baseline encoders are stated, we focus on a more powerful mechanism to encode the spatial and temporal information of inputs by encoding them into feature vectors. In that sense, we focus of designing an effective encoder transformer while keeping its structure as simple and efficient as possible. In a similar way to [115], we adopt the combination of CNN/MLP, attention block and normalization, as observed in Figure 6.4.

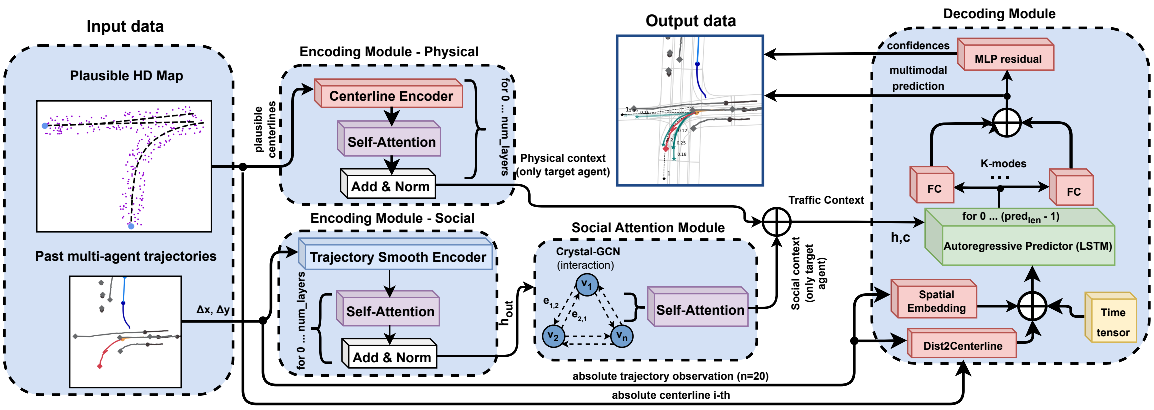


Figure 6.4: Efficient baseline with transformer encoders to process the physical and social input

In order to encode the centerlines, we first use an MLP-based encoder to transform the input vector at each time stamp (d_i^t, which actually represents a plausible position of the target agent) into deep features:

$$f_i^t = MLP_{\text{map}}\left(d_i^t; W_{\text{map}}\right) \tag{6.7}$$

where MLP_{map} is a Multi-Layer (3) Perceptron with a ReLU asnon-linear layer and W_{map} as the weight matrix that is learnable. However, to predict the future trajectory, the separate feature of each vector is insufficient. For example, even if two road segments in the first half have the same structure, the difference in the last half can result in a total difference in geometric meaning. Therefore, we make use of the well-established Multi-Head Self-Attention (MHSA) [73] mechanism to encode the overall set of physical features per agent as a single vector.

To be more specific, we first calculate the query, key and value matrix:

$$q_i^t = W^q f_i^t, k_i^t = W^k f_i^t, v_i^t = W^q f_i^t, \tag{6.8}$$

where W^q, W^k, W^v are the learnable weight matrices. Then, we take these three matrices as the inputs of the weighting block based on softmax:

$$h_i^t = \text{softmax}\left(\frac{q_i^t \cdot k_i^{tT}}{\sqrt{d_k}}\right) v_i^t, \tag{6.9}$$

where d_k is the length of matrix k. Finally, we adopt a 2-layer MLP to aggregate the features of vectors within a road segment:

$$h_i = MLP_{agg}\left(h_i^t; W_{agg}\right) \tag{6.10}$$

where MLP_{agg} is a 2-layer MLP with a ReLU non-linear layer and W_{agg} is the weight matrix that is learnable. Now, we have the feature vector for each target agent, stored as a 2D matrix (M, H), where M is the number of road segments and H is the length of hidden features.

For agents, we use similar techniques to encode and aggregate the information. In particular, we use a trajectory encoder block to encode each vector into the form of a feature vector. Then, similar to roads, even two vehicles have the same movement in the first half of their trajectory, and the differences in the last half of trajectories can lead to a totally different future trajectory. Therefore, we use a MHSA block to encode the overall feature of one trajectory in the observed time period and form a single feature vector for each agent. Finally, a 2-layer MLP-based aggregator is used to construct a single feature vector for each trajectory.

One aspect worth mentioning is the agent encoder. While trajectory data are, unlike roads (well structured) usually non-smooth, as expected from real-world datasets. Then, while we make use of MLP to compute the deep physical features, we use a 1D-CNN based motion encoder in the first stage due to its wider receptive field compared with MLP in such a way the convolutional encoder can smooth the trajectories and reduce the influence of noisy input trajectories.

6.2.4. Decoding module

The decoding module is the third component of our baselines, as observed in Figure 6.1. The decoding module consists of an **LSTM** network, which recursively estimate the relative displacements for the future timesteps, in the same way we studied the past relative displacements in the Motion History encoder. Regarding the social baseline, the model uses the social context computed by the Social Interaction Module, only paying attention to the target agent row. Then, only the social context corresponds to the whole *traffic context* of the scenario, representing the input hidden vector of the autoregressive LSTM predictor. On the other hand, in terms of the map baseline, for a mode m, we identify the latent *traffic context* as the concatenation of the social context, static physical context and specific physical context as stated in Section 6.2.2.2, which will serve as input hidden vector $\mathbf{h}$ of the LSTM decoder. In both cases (social and map baselines), the cell vector $\mathbf{c}$ is initialized with a vector of zeros of the same dimension.

Regarding the LSTM input, in the social case it is represented by the encoded past n relative displacements of the target agent after a spatial embedding, whilst the map baseline adds the encoded vector distance between the current absolute target position and the current centerline, as well as the current scalar timestamp t, as illustrated in Figure 6.1. In both cases (social and map baselines), we process the output of the LSTM using a standard Fully-Connected (FC) layer (one per mode). Once we have the relative prediction in the timestep t, we shift the initial past observation data in such a way we introduce our last-computed relative displacement at the end of the vector, removing the first data. We identify this technique as a *temporal decoder*, where a window of size n is analized by the autoregressive decoder in contrast to other techniques [30], [88], [99] where only the last data is considered. Finally, after performing relative displacements to absolute coordinates operation, we obtain our multimodal predictions $\hat{Y} \in \mathbb{R}^{k \times pred_{len} \times data_{dim}}$, where k represents the number of modes, $pred_{len}$ represents the prediction horizon and $data_{dim}$ represents the data dimensionality, in this case xy, predictions from the BEV perspective). Once the multimodal predictions are computed, they are concatenated and processed by a residual MLP to obtain the confidences (the higher the confidence, the most probable the mode must be, and closer to the ground-truth).

6.2.5. Losses

We use the standard **Negative Log-Likelihood** (NLL) loss to train our social and map baselines in order to compare the ground-truth points $Y \in \mathbb{R}^{pred_{len} \times data_{dim}} = \{(x_0, y_0)...(x_{pred_{len}}, y_{pred_{len}})\}$ with our multimodal predictions ($\hat{Y} \in \mathbb{R}^{k \times pred_{len} \times data_{dim}}$), given k modalities (hypotheses) $\mathbf{p} = \{(\hat{x}_0^1, \hat{y}_0^1)...(\hat{x}_{pred_{len}}^k, \hat{y}_{pred_{len}}^k)\}$, with their corresponding confidences $\mathbf{c} = \{c_1...c_k\}$ using the following equation:

$$\text{NLL} = -\log \sum_{k} e^{\log c^k - \frac{1}{2}\sum_{t=0}^{pred_{len}} (\hat{x}_t^k - x_t)^2 + (\hat{y}_t^k - y_t)^2} \tag{6.11}$$

Similar to [33], we assume the ground-truth points to be modeled by a mixture of multi-dimensional independent Normal distributions over time (predictions with unit covariance). Minimizing the NLL loss maximizes the likelihood of the data for the forecast. Nevertheless, the NLL loss tends to overfit most predictions in a similar direction. As stated above, in the motion prediction task, specially in the Autonomous Vehicles field, we must build a model that not only reasons multimodal predictions in terms of different maneuvers (keep straight, turn right, lane change, etc.) but also different velocity profiles (constant velocity, acceleration, etc.) regarding the same maneuver. For this reason, after the baselines models have been trained, as stated by [57], we add as regularization the Hinge (*a.k.a.*max-margin) and **Winner-Takes-All** (WTA) [13], [57] losses to improve the confidences and regressions respectively.

Algorithm 3 illustrates how we compute the max-margin and WTA losses. First, we determine the closest mode m^* to the ground-truth using the $\mathcal{L}_2$ distance, only considering the end-points. Then, WTA loss is computed using Smooth $\mathcal{L}_1$ distance taking into account in this case the whole prediction horizon between the best mode and ground-truth prediction. Finally, we apply the max-margin loss regarding the confidence of the best mode and a margin (ϵ).

Algorithm 3: Additional regularization: Hinge and WTA loss

input: ground-truth trajectory ($Y \in \mathbb{R}^{pred_{len} \times data_{dim}}$) and output trajectories ($\hat{Y} \in \mathbb{R}^{k \times pred_{len} \times data_{dim}}$), where k, $pred_{len}$, and $data_{dim}$ denote the number of modes, prediction horizon, and data dimensionality for the target agent.

output: classification loss $\mathcal{L}_{Hinge}$ and regression loss $\mathcal{L}_{WTA}$

for m *in* $\{1, 2, \ldots, k\}$ **do**

  $d_{wta}^m \leftarrow$ Euclidean distance between $\hat{Y}_{pred_{len}}^m$ and $Y_{pred_{len}}$;

end

$m^* = \arg\min_m d_{wta}^m$;

$\mathcal{L}_{reg,WTA} \leftarrow$ Smooth $\mathcal{L}_1$ loss between $\hat{Y}^{m^*}$ and Y;

$\mathcal{L}_{class,Hinge} = \frac{1}{(K-1)} \sum_{m=1 \backslash m \neq m^*}^{K} \max(0, c_k + \epsilon - c_{m^*})$;

return $\mathcal{L}_{WTA}$, $\mathcal{L}_{Hinge}$;

Therefore, our loss function is:

$$\mathcal{L} = \alpha \mathcal{L}_{NLL} + \beta \mathcal{L}_{Hinge} + \gamma \mathcal{L}_{WTA} \tag{6.12}$$

6.3. Experimental Results

6.3.1. Implementation details

To validate these efficient baselines (social, map and augmented baseline), we use the Argoverse 1 Motion Forecasting, as discussed in Section 5.3.1. In terms of evaluation metrics, we evaluate the performance of our models using the standard metrics Average Displacement Error (ADE) and FDE. Unlike the GAN-based model, the output of the models in this Chapter is multimodal, then we generate k outputs (also known as modes) per prediction step and report the metrics for the best out of k outputs, regarding the agent i.

We report results for $k = 1$ (unimodal case, only the mode with the best confidence is considered) and $k = 6$ as this is the standard in the Argoverse Motion Forecasting dataset in order to compare with other models.

We train our models to convergence using a single NVIDIA RTX 3090, and validate our results on the official Argoverse 1 validation set [5]. We use Adam optimizer with learning rate 0.001 and default parameters, batch size 1024 and linear LR Scheduler with factor 0.5 decay on plateaus. We rotate the whole scene regarding the orientation in the last observation frame of the target agent to align this agent with the positive y-axis. The hidden dimension for the Motion History encoder is 64, where both the hidden state $\mathbf{h_{in}}$ and cell state $\mathbf{c_{in}}$ are initialized with zeros (dim = 128), whilst the MLP encoder for both the specific centerline and plausible area is 128. Regarding the Social Interaction module, the latent vector of the Crystal-GCN layers is 128 and the number of heads in the MHSA module is $L_h = 4$. In terms of the Autoregressive predictor, the spatial embedding and *dist2centerline* modules encode the past data and distance to the specific centerline using a *window size* of 20. We set the number of plausible centerlines (M) as 3, which cover most cases (if less than 3 plausible centerlines are available, we add padded centerlines as vector of zeros). The time tensor is a single number that represents the current timestep, in such a way the LSTM input is *(2 × window size) + 1 = 41*. The regression head is represented by *k=6* FC layers that map the output latent vector returned by the LSTM to the final output relative displacements (dim = 2, xy). Multimodal predictions are processed by an MLP residual of sizes 60, 60 and 6 with interspersed ReLU activations in order to obtain the corresponding confidences.

6.3.1.0.1. Augmentations (i) Dropout and swapping random points from the past trajectory, (ii) point location perturbations under a $\mathcal{N}(0, 0.2)$ [m] noise distribution [114]. We also apply the well-known hard-mining technique to improve the model's generalization under difficult scenarios. To perform this technique, once we have the social and map baselines, we perform inference on the training set to find the most difficult scenes in terms of minADE. Then, we mine those scenes such that the baselines models perform poorly, and increase their proportion in the batch during training.

6.3.2. Model results

Table 6.1: Ablation Study for map-free MP on the Argoverse 1 validation set. Our methods are indicated with †, our highlighted method indicates our map-free baseline (Best Social Model = BSM). Prediction metrics (minADE, minFDE) are reported in meters.

Method	Number of Parameters	$k=1$		$k=6$	
		minADE	minFDE	minADE	minFDE
TPCN [114]	-	1.42	3.08	0.82	1.32
LaneGCN [13] (w/o map)	$\approx 1M$	1.58	3.61	0.79	**1.29**
WIMP [16] (w/o map)	$> 20M$	1.61	5.05	0.86	1.39
CRAT-Pred (LSTM + GNN + Lin. Residual) [12]	449K	1.44	3.17	0.86	1.47
CRAT-Pred (LSTM + GNN + Multi-Head Self-Attention + Lin. Residual) [12]	515K	**1.41**	**3.10**	0.85	1.44
† LSTM-128 + GNN + MHSA (Baseline social)	351K	1.82	3.72	0.87	1.63
† LSTM-64 + GNN + MHSA	**97K**	1.77	3.68	0.86	1.61
† LSTM-128 + GNN + MHSA + Lin. Residual	552K	2.02	4.16	1.02	1.95
† LSTM-128 (TDec) + GNN + MHSA	365K	1.81	4.04	0.83	1.57
† LSTM-64 (TDec) + GNN + MHSA (Best Social Model)	105K	1.79	4.01	0.81	1.56
† Best Social Model + HardM (10 %)	105K	1.76	3.97	0.80	1.53
† Best Social Model + HardM (10 %) *w/* Loss Hinge + WTA	105K	1.62	3.57	**0.76**	1.43

6.3.2.0.1. Ablation studies As we state in previous Sections, our main goal is to achieve competitive results while being efficient in terms of model complexity, in particular in terms of **FLOPs** (Floating-Point Operations per second) and **parameters** in order to enable these models for real-time operation. For this reason, we have proposed light-weight models, whose main input is the history of past trajectories of the agents, complemented by interpretable map-based features.

In this section we analyze our results and ablation studies, and prove the benefits of our approach for self-driving MP. Table 6.1 and 6.2 illustrate our ablation study regarding our social and map baselines respectively. First, we compare our social model with other SOTA models [13] [16] [12] without map information or with the corresponding module disabled. Our social baseline, trained with NLL loss, presents a number of 351K parameters and 0.87 / 1.63 for minADE and minFDE (k=6) respectively.

We perform the following **ablation studies** in Table 6.1: Reduce social hidden dim (including LSTM, GNN and MHSA modules) from 128 to 64, replace the standard head with residual head, replace only last data (standard autoregressive decoder input) with last N data (temporal decoder). We obtain better results with hidden dim = 64, decreasing the number of parameters. Linear residual, standard in most MP models, presents worse results with a much higher number of parameters, since most works use it in a non-autoregressive way, decoding directly from the latent space. On the other hand, using temporal decoder instead of only the last position as LSTM input achieves better results with a slightly higher number of parameters. Then, we conclude our Best Social Model (BSM), as a preliminary stage before implementing the map features, presents the

Table 6.2: Ablation Study for map-based motion forecasting on the Argoverse 1 validation set. Our methods are indicated with †. We highlight our map-based baseline method, as a reference for future comparisons.

Method	Number of Parameters	$k=1$		$k=6$	
		minADE	minFDE	minADE	minFDE
† **Our Map-free Baseline (BSM, No Hard-mining, Loss = NLL)**	105K	1.79	4.01	0.81	1.56
LaneGCN [13]	3.7M	**1.35**	**2.97**	**0.71**	**1.08**
WIMP [16] (w/o map, NLL loss)	$> 25M$	1.41	6.38	1.07	1.61
WIMP [16] (w/o map, EWTA loss)	$> 25M$	1.45	3.19	0.75	1.14
† BSM + Oracle	**277K**	1.62	3.56	0.77	1.42
† BSM + centerlines=3	307K	1.60	3.53	0.76	1.39
† BSM + centerlines=3 (1D-CNN)	432K	1.63	3.59	0.78	1.43
† BSM + centerlines=3 loop	326K	1.62	3.41	0.76	1.40
† BSM + centerlines=3 loop + Feasible area	458K	1.62	3.40	0.76	1.40
† BSM + centerlines=3 loop + Feasible area + Dist2Centerline (Best Global Model)	459K	1.61	3.40	0.75	1.39
† Best Global model + HardM (10 %)	459K	1.55	3.31	0.75	1.36
† Best Global model + HardM (10 %) *w/* Loss Hinge + WTA	459K	1.46	3.22	0.72	1.28

following modifications: social hidden dim = 64 and temporal decoder. Hard-mining and additional losses (Hinge and WTA) applied to the best social model achieve the best social results (Social Baseline).

To integrate the map features (Table 6.2) we start from the BSM without hard-mining and with the initial loss (NLL), in order to check how implementing these additional regularization terms help the model to generalize better in both experiments (only social and social+map). We perform the following ablations: compute the most plausible centerline (M=1) returned by the **Argoverse API**, consider M=3 centerlines, replace MLP encoder with 1D-CNN encoder in a similar way [33], explicitly iterate over all centerlines as **specific** deep physical context instead of decoding from a common latent space, add low-level features (feasible area) as a common **static** deep physical context for each iteration and finally adding an additional component to the LSTM input determined by the vector distance between the considered input window (last N data) and the corresponding centerline. It can be observed that introducing map features increases the number of parameters in exchange of a noticeable metrics decrement, specially in terms of minFDE ($k = 6$). Considering M=3 centerlines instead of only the most plausible centerline allows the model to compute a more diverse set of predictions, while replacing a standard MLP encoder with 1D-CNN encoder increases the number of parameters achieving worse metrics, according to this experimental setup.

Finally, we include our low-level (static) features as a static deep physical context which is common to all iterations over the different centerlines and an additional vector distance to the corresponding centerline, achieving our best results without additional regularization terms (hard-mining and Hinge / WTA losses). In both cases (social and map baselines), we obtain regression metrics (minADE and minFDE with both $k = 1$ and 6) up-to-pair with other SOTA models with a noticeable lower number of parameters, specially in the ablation study for map-based MP models, demonstrating how focusing on the most important map-features drastically decreases the network complexity obtaining similar results in terms of accuracy. This representation not only gathers information about the feasible area around the agent, but also represents potential goal points [99]

Table 6.3: Results on the Argoverse 1 Motion Forecasting Leaderboard. We borrow some numbers from [5], [8], [9]. We specify the map info for each model: Raster, GNN or polyline, as stated in Table 2.1. We indicate the error difference of our method *w.r.t.* top-25 SOTA methods, in centimeters. Our predictions differ *w.r.t.* top-25 SOTA only 10cm and 15cm for the unimodal and multimodal minADE metric respectively, yet our model is much more efficient.

Model	Map info	K=1		K=6	
		minADE ↓	minFDE ↓	minADE ↓	minFDE ↓
Constant Velocity [5]	-	3.53	7.89		
Argoverse Baseline (NN) [5]	-	3.45	7.88	1.71	3.29
Argoverse Baseline (LSTM) [5]	Polyline	2.96	6.81	2.34	5.44
Argoverse Baseline (NN) [5]	Polyline	3.45	7.88	1.71	3.29
TPNet-map-mm [109]	Raster	2.23	4.70	1.61	3.70
Challenge Winner: uulm-mrm (2nd) [5]	Polyline	1.90	4.19	0.94	1.55
Challenge Winner: Jean (1st) [5], [33]	Polyline	1.74	4.24	0.98	1.42
TNT [17]	GNN	1.77	3.91	0.94	1.54
mmTransformer [116]	Polyline	1.77	4.00	0.84	1.33
HOME [8]	Raster	1.72	3.73	0.92	1.36
LaneConv [117]	Raster	1.71	3.78	0.87	1.36
UberATG [13]	GNN	1.70	3.77	0.87	1.36
LaneRCNN [49]	GNN	1.70	3.70	0.90	1.45
GOHOME [9]	GNN	1.69	3.65	0.94	1.45
State-of-the-art (top-10) [9], [10], [114], [116]		**1.57**±0.06	**3.44**±0.15	**0.79**±0.02	**1.17**±0.04
State-of-the-art (top-25) [9], [10], [114], [116]		**1.63**±0.08	**3.59**±0.20	**0.81**±0.03	**1.22**±0.06
Ours (Social baseline, including HardM and losses)	-	2.57	4.36	1.26	2.67
Ours (Map baseline, including HardM and losses)	Polyline	1.73 (10cm)	3.89 (30cm)	0.96 (15cm)	1.63 (41cm)

(*i.e.* potential destinations or end-of-trajectory points for the agents). Moreover, this information is *"cheap"* and *interpretable*, therefore, we do not need further exhaustive annotations from the HD Map in comparison with other methods like HOME, which gets as input a 45-channel encoded map [8].

6.3.2.0.2. Comparison with the State-of-the-Art The Argoverse Benchmark [5] has over 290 submitted methods, however, the top approaches achieve, in our opinion, essentially the same performance. In order to do a fair comparison, we analyze the *state-of-the-art* performance in this benchmark, we show the results in Table 6.3. Given the standard deviations (in meters) of the most important regression metrics (minADE and minFDE, both in the unimodal and multimodal case), we conclude that there are no significant performance differences for the top-25 models. In fact, as stated in Chapter 2, Argoverse 2 [6] explicitly mentions that there is a *"goldilocks zone"* of task difficulty in the Argoverse 1 test set, since it has begun to plateau.

We prove our best model (Augmented baseline) qualitatively in Figure 6.5, where we can see how the estimated centerlines represent a good guidance for the model and the multi-modal prediction works accordingly specifically in challenging intersections.

6.3.3. Efficiency discussion

In terms of **efficiency discussion**, to the best of our knowledge, very few methods reports efficiency-related information [8], [9], [34], [116]. Furthermore, comparing runtimes is difficult, as only a few competitive methods provide code and models. The Argoverse Benchmark [5] provides insightful metrics about the model's performance, mainly related with the predictions error. However, there are no metrics about efficiency (*i.e.* model complexity in terms of parameters or FLOPs). In the AD context, we consider these metrics as important as the error evaluation because, in order to design a reliable AD stack, we must produce reliable predictions on time, meaning the inference time (related to model complexity and inputs) is crucial. SOTA methods already provide predictions with an error lesser than 1 meter in the multi-modal case. In our opinion, an accident will rarely happen because some obstacle predictions are offset by one or half a meter, this uncertainty in prediction can be acceptable in the design of AV, but rather because lack of coverage or delayed response time. Despite its high accuracy and fast inference time, LaneGCN [13] makes use of multiple GNN layers that can lead to issues with over-smoothing for map-encoders [118]. Moreover, as mentioned in [34], CNN-based models for processing the HD map information are able to capture social and map interactions, but most of them are computationally too expensive. LaneRCNN [49] adds huge number of hyperparameters to the model, making it quite complex since it proposes to capture agent and map interactions with a local interaction graph per agent, not just a single vector.

Table 6.4: Efficiency comparison of our baselines against SOTA methods. We show the number of parameters for each model, FLOPs, minADE (k=6) in the Argoverse test set, and runtime. Works from [34] focus on unimodal predictions (k=1). *N/A* stands for *Not Available*. Time measured on a RTX 2080 Ti (using batch 32). Some numbers are borrowed from [119], [120].

Model	# Par. (M)	FLOPs (G) ↓	minADE (m) ↓	Run (ms) ↓
CtsConv [121]	1.08	0.34	1.85	684
R18-k3-c1-r100 [34]	0.25	0.66	2.21	*N/A*
R18-k3-c1-r400 [34]	0.25	10.56	2.16	*N/A*
VectorNet [34]	**0.072**	0.41	1.66	1103
DenseTNT (w/ 100ms opt.) [122]	1.1	0.763	0.88	2644
DenseTNT (w/ goal set pred.) [122]	1.1	0.763	0.85	531
LaneGCN [13]	3.7	1.071	0.87	173
mmTransformer [116]	2.607	0.177	0.84	*N/A*
MF-Transformer [123]	2.469	0.408	**0.82**	*N/A*
HOME+GOHOME [9]	0.40	0.09	0.94	32
Ours	1.235	**0.038**	0.91	**16**

Similar to image classification, where model efficiency depends on its accuracy and parameters/FLOPs, we use the same criteria to compare models. We show the **efficiency comparison** with other relevant methods in Table 6.4. We calculate FLOPs and parameters using a third-party library [1]. Some minor operations were not supported, yet, their contributions to the number of FLOPs were residual and ignored. The results for the other methods are consulted from [8] [9] [34] [123]. We calculate FLOPs using the rela-

[1] https://github.com/facebookresearch/fvcore

tion: GMACs $\approx 0.5 * $ GFLOPs using https://github.com/facebookresearch/fvcore.

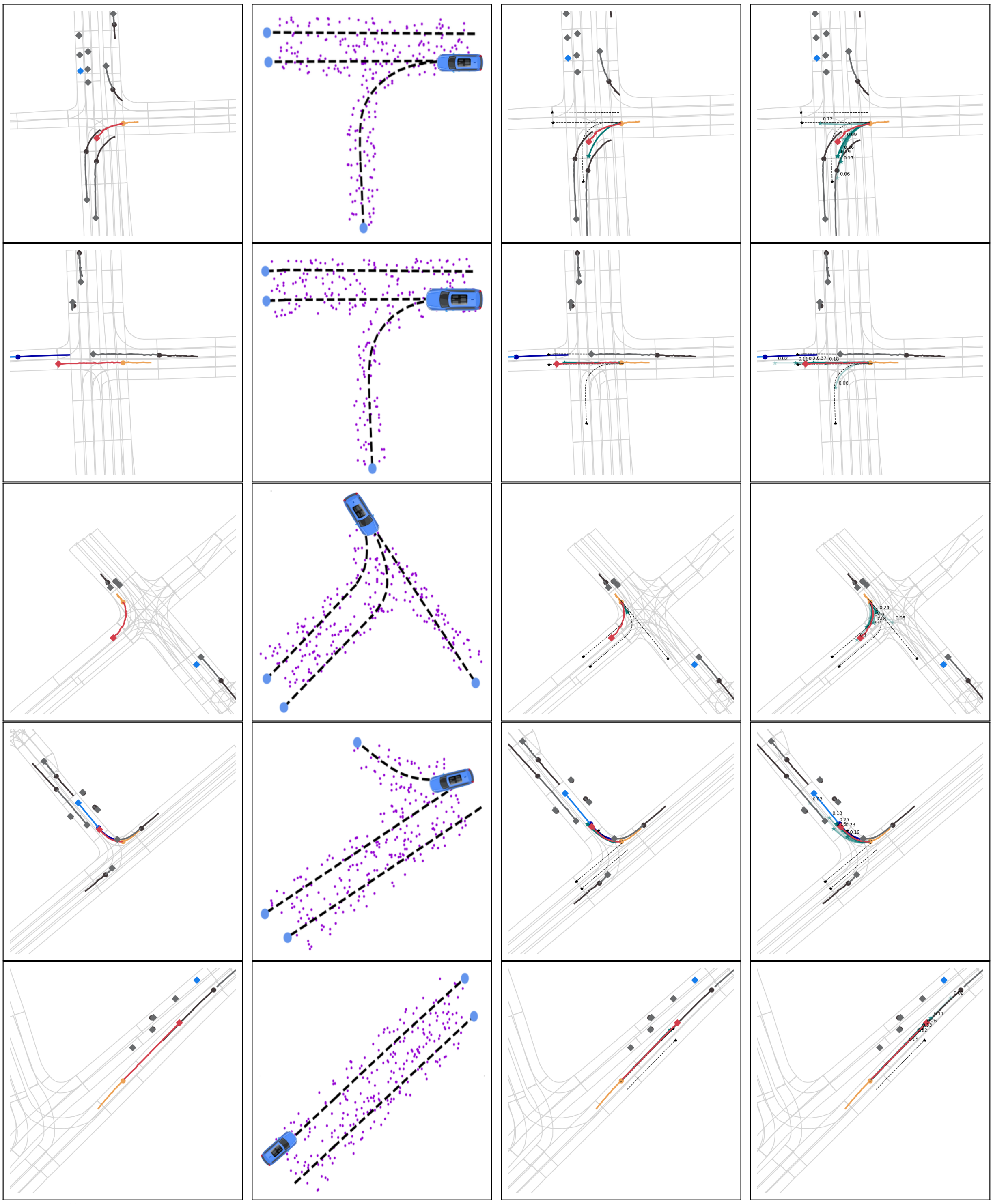
In order to calculate the FLOPs, we follow the common practice [34] [122] [9] of fixing the number of lanes *i.e.* the number of centerlines is limited to 3. [34] compares its GNN backbone with CNNs of different kernel sizes and map resolution to compute deep map features (decoder operations and parameters are excluded, min), demonstrating how CNN based methods noticeably increase the amount of parameters and operations per second. We do not require CNNs to extract features from the HD map since we use our map-based feature extractor to obtain the feasible area (see Section 6.2.2), assuming anisotropic driving (the most important features are ahead) and non-holonomic constraints, in such a way these features are interpretable in comparison with CNNs high-dimensional outputs. Note that, in both variants (social and map baselines), the self-attention module is used with a dynamic number of input agents, this typically implies a quadratic growth in complexity with the number of agents in the scene [73], yet, this only applies to the MHSA layers.

Even though our methods do not obtain the best regression metrics, we achieve up-to-pair results (Table 6.4) against other SOTA approaches whilst our number of FLOPs is several orders of magnitude smaller than other approaches [122] [13], obtaining a good trade-off (specially the map baseline) between model complexity and accuracy (minADE, k=6), making it suitable for real-time operation in the field of AD. In our case, considering the top-25 regression metrics we achieve near SOTA results (just 15 cm, which represents 18.5 %, worse in terms of minADE k=6 regarding our final approach) while achieving an impressive reduction of parameters and FLOPs. As observed in Table 6.4, if we compare our final model, which includes social information, agents interaction and preliminary road information, and the methods with the closest minADE k=6 [m] (LaneGCN [13], HOME [8] and GOHOME [9]), we obtain a reduction of 96 %, 99 % and 48 % respectively in terms of FLOPs. It can be observed how including preliminary road information assuming non-holonomic [51] and anisotropic [112] constraints respectively (that is, we mostly focus on the front driveable area) instead of processing the whole map, as well as computing social interactions via graph convolutional networks, boost our model for further integration edge-computing devices with a minimum accuracy loss acceptable for real-world Autonomous Driving applications.

Moreover, as it is well known in machine learning, the number of parameters is not always proportional to the inference speed. In that sense, our transformer approach also has certain benefits in comparison to LSTM/RNN temporal encoding, since these are non-parallelizable, therefore, despite having more parameters, transformers are faster [73].

6.4. Summary

In this Chapter, we propose several efficient baselines, progressively aggregating contextual information from social info to augmented map via a transformer-based model that does not rely on heavily annotated HD maps, yet it uses past trajectories and minimal map priors. The final proposed method combines the transformer attention mechanisms with GNNs to model agent interactions. We show that it has less parameters than other methods, and it is faster than most previous methods. We achieve near-SOTA results on the Argoverse Motion Forecasting Benchmark while having a low computational cost compared to other state-of-the-art proposals. In future works, we plan to extend our work for multi-agent modal prediction in the Argoverse 2 dataset, taking into account more complex features and interactions in an efficient and powerful way.Our framework is open-sourced.



General view | Plausible HDMap | k=1 prediction | k=6 predictions

Figure 6.5: Qualitative Results on challenging scenarios in the Argoverse 1 validation set using our best model. We represent: our vehicle (**ego**), the **target agent**, and **other agents**. We can also see the **ground-truth** trajectory of the target agent, our **multimodal predictions** (with the corresponding confidences) and **plausible centerlines**. Circles represent last observations and diamonds last future positions. As we can see the plausible HDMap serves as a good guidance to our model, which can predict reasonable trajectories in presence of multiple agents and challenging scenarios. We show, from left to right, a general view of the traffic scenario (including social and map information), our calculated plausible HDMap, unimodal prediction (best mode in terms of confidence) and multimodal prediction (k = 6), including confidences (the higher, the most probable)

Chapter 7

Improving Multi-Agent Motion Prediction with Heuristic Proposals and Motion Refinement

Solo sé que no sé nada.
Sócrates a Platón
Diálogos de Platón

7.1. Introduction

This is the last and most advanced MP algorithm proposed in this thesis. Based on our previously stated map baseline, we aim to get an efficient model that consider more information about the agents, HD map topological and semantic information (in addition to the previously stated geometric features) and contextual interactions. Note that obtaining and fusing this information (*e.g.* actor-to-actor, map-to-actor) is a research topic by itself [10], [13], [49] and a core part in the ADS pipeline. Here we identify a bottleneck for efficient real-time applications [83], [124], as usually, more (complex) data-inputs implies higher model complexity and inference time [34].

Predicting the future trajectories of the agents without considering their nature might not be optimal (*e.g.* predicting on a pedestrian, a cyclist or a vehicle using the same logic). For this reason, we integrate additional features related to the type and properties of agents (also referred as metadata in the literature). Moreover, we also compute heuristic scene understanding to constrain the model predictions towards the real scene geometry (*e.g.* plausible centerlines and lanes), including lane and boundary topological information or presence of an intersection. As stated by [125], only using lane centerline as input to get the embedding feature of vector map nodes is not enough. The lane centerline can only provide the topology of the lanes, and other elements of the vector map also contain rich information. For example, the lane boundary can provide traffic rule constraint information such as whether it is possible to conduct the lane change behaviour or not (dashed vs solid line, yellow vs white, etc.). When considering such amount of information, specifically in terms of physical context and interactions, most SOTA methods require an overwhelming model complexity which can be inefficient in terms of computation [34], [46], [121].

7.2. Our proposal

To address the aforementioned SOTA limitations, we propose a model [126] to achieve accurate motion prediction, yet, using light-weight transformer-based models for social encoding, GNNs for context interaction, enhanced heuristic proposals and motion refinement, reducing notably the complexity of our model with respect to previous methods such as GANet [11] to avoid these possible constraints. Figure 7.1 illustrates the overall pipeline. We make the following contributions:

1. We present a SOTA method on the Argoverse 2 Motion Forecasting Benchmark, one of the most recent and challenging vehicle MP datasets.
2. Our model uses various attention mechanisms with GNNs, and a motion refinement module to further improve temporal consistency.
3. In comparison to previous methods that rely only on past trajectories and HD map, we additionally use information about the agents (*e.g.* type of agent) and the scene geometry (*e.g.* lane distribution and possible goal points).
4. Our method reduces in millions of parameters previous methods such as GANet [11], and improves over LaneGCN [13].
5. Finally, we provide an open-source framework for MP.

Throughout this Chapter we will explain the different modules of the proposed MP method, which are: 1) **Social Encoder**, which uses the agent past trajectories (relative displacements and additional metadata such as the type (*e.g.* car, cyclist, pedestrian) and category, from less to more important) and the corresponding heuristic lane proposals to compute the social features, 2) **Map Encoder**, that constructs a lane graph from the HD Map and uses a LaneConv operator [13] to extract lane node features, 3) **Fusion Cycle**, responsible for fusing agents and map latent features, 4) **Goal Areas estimation**, to predict some goals and their surrounding (area) features are aggregated to the agents, 5) **Multimodal Decoder**, which uses the latent actors with deep area context to generate reliable multi-modal predictions and 6) **Motion Refinement**, in charge of enhancing the quality of the future trajectories taking into account the past trajectories, actors latent features and preliminary predicted trajectories.

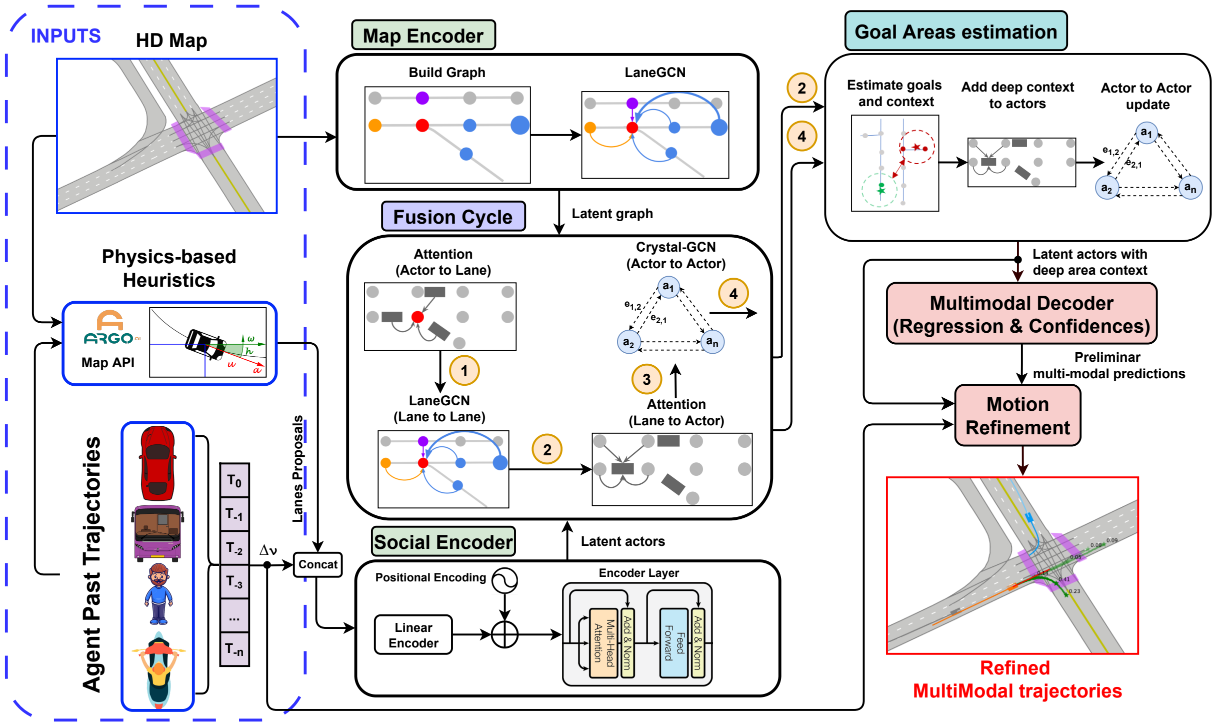


Figure 7.1: Overview of our Motion Prediction model including Fusion Cycle, Heuristic Proposals and Motion Refinement

7.2.1. Preprocessing of past trajectories and heuristic proposals

The social preprocessing step proposed in this model is slightly different to our previous methods where we only considered those agents that have information over the full history horizon. Now, given the complexity of Argoverse 2, an agent that recently appeared in the scene, even if it was not observed over the whole full sequence spectrum, or even it was occluded for a few timestamps, it can be really relevant to determine the future behaviour of another agent. To this end, as proposed by multiple methods [12], [13], we consider all agents that are observable at *t=0*, handling those agents that are not observed over the full sequence spectrum (observation length = obs_{len} + prediction length = $pred_{len}$) by concatenating a binary flag b_i^t that indicates if the agent is padded or not. On top of that, we only consider the dynamic agents of the scene (vehicles, pedestrians, motorcyclists, cyclists and buses) and discard unknown or static objects (background, construction or riderless bicycle), since dynamic (which can be stopped or not) are the most relevant for the MP task. Given these final agents, we follow the same principles of translation and rotation invariant, as well as relative displacements, to compute the input past trajectories. On top of that, we additionally compute and codify the object type (bus, pedestrian, vehicle, cyclist) and agent importance (unscored, scored and focal track) as additional metadata. As described by Argoverse 2, each track is assigned one of the following labels, which dictate scoring behavior in the Argoverse 2 challenges:

1. Track fragment: Lower quality track that may only contain a few timestamps of observations.

2. Unscored track: Unscored track used for contextual input.

3. Scored track: High-quality tracks relevant to the agent.

4. Focal track: The primary track of interest in a given scenario - scored in the single-agent prediction challenge.

We can appreciate how, in addition to the attention mechanisms of the model which are responsible of computing the most relevant features, the aforementioned track category serves as a good guidance as preliminary information to refer the importance of a specific agent with respect to another one.

In terms of physical information, we aim to increase the number of features of the HD Map as well as its corresponding interactions with the social features. Then, we focus on **Graph-based** methods [49] which construct graph-structured representations from the HD maps, which preserve the connectivity of lanes, and therefore the geometry of the scene. VectorNet [34] is one of the first works in this direction, where the authors propose to encode map elements and actor trajectories as polylines and then use a global interactive graph to fuse map and actor features. We find especially related LaneGCN [13], a method that constructs a map node graph and proposes a novel graph convolution. In that sense, we follow the same principles than these well-established baselines by adopting simple form of vectorized map data as our representation of HD maps. In this case, the map data is represented as a set of polylines (lanes) and their connectivity, where each lane contains a centerline (sequence of 2D BEV points), arranged following the lane direction. For any two lanes which are directly reachable, 4 types of connections are given: predecessor, successor, left neighbour and right neighbour.

Moreover, we propose the use of preliminary plausible area information in a similar way to the heuristic proposals illustrated in Section 6.2.2.1. We follow the same heuristic (filter the agent, calculate the future travelled distance by means a CA model, get all candidates within a bubble, given the agent last observation and Manhattan distance, etc) to compute the most plausible future centerlines for the corresponding agent. It must be considered that in Argoverse 2 the number of categories is modified to 5 (vehicles, pedestrians, motorcyclists, cyclists and buses) insteaf of only 1 (vehicle) in the case of most vehicle MP datasets, including Argoverse 1. So, no centerlines proposals are considered (then, they are created virtually and padded as stated in previous sections) since we assume that pedestrians are not walking on the road, but on the pedestrian crossings or sidewalks. In future works we will work on integrating specific physical information depending on the object type as preliminary map features. Nevertheless, in this particular case, thanks to a more realistic representation of the HD map, we include additional metadata such as lane type (bus, bike, vehicle), presence of intersection (binary flag) or boundaries mark type (dash, solid, yellow), along with the aforementioned centerline relative displacements.

7.2.2. Social Encoder

In terms of social encoding, in order to capture more complex features for subsequent features fusion and interaction, we initially adopted GANet [11], based on LaneGCN [13], to encode motion history and scene context for its outstanding performance. In this backbone (given the aforementioned social input: translation and rotation invariant with respect to a target agent, and relative displacements), LaneGCN makes use of use a network with 3 groups/scales of 1D convolutions 1D CNN to process the trajectory input for its effectiveness in extracting multi-scale features and efficiency in parallel computing. The output is a well-structured feature map, whose element at $t = 0$ is used as the actor feature. The network has 3 groups/scales of 1D convolutions. Then, a Feature Pyramid Network (FPN) [127] to fuse the multi-scale features, and apply another residual block to obtain the output tensor. Moreover, GANet applies an LSTM network on FPN output features and use two identical parallel networks to enhance the motion history encoding.

Regarding this, we aimed to improve social encoding taking into account that in spite the fact that LSTMs became popular because they could solve the problem of vanishing gradients, they suffer from *short-term memory* due to the vanishing gradient problem, as well as require a lot of resources to get trained and become ready for real-world applications. In particular, they need high memory-bandwidth because of linear layers present in each cell which the system usually fails to provide for. To solve that, we replace the motion encoder proposed by [11] for a transformer encoder, which is faster than RNN-based models as all the input is ingested once, decreasing the computational complexity. On the other hand, even though the preliminary lane proposals represent physical information, they are quite related to the future intentions of the social information. Then, as depicted in Figure 7.1, we concatenate the agents past trajectories, additional social metadata and heuristic map proposals (including semantic and topological metadata), which is processed by a linear embedding. Then, positional encoding is added to the output embedding explicitly to retain the information regarding the order of past trajectories and future preliminar steps. Finally, these latent features feed the transformer encoder, leveraging the self-attention mechanism and positional encoding to learn complex and dynamic patterns from long-term time series data.

7.2.3. Map preprocessing and encoding

In terms of physical context, we adopt MapNet [13] backbone to encode the scene context for its outstanding performance. While other approaches encode the map as a raster image and apply 2D convolutions to extract features, MapNet consists of two steps:

- Build a lane graph from vectorized map data
- Apply a LaneConv operator to the lane graph to output the map features

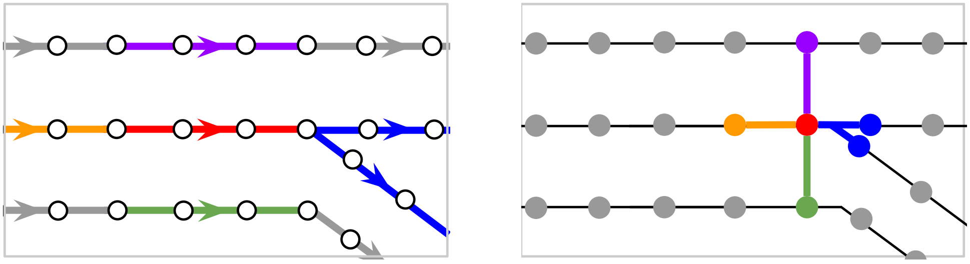


Figure 7.2: Lane graph construction from vectorized map data
Source: *Learning lane graph representations for motion forecasting* [13]

As observed in Figure 7.2, the map data is represented as a set of lanes and their connectivity. Each lane contains a centerline, *i.e.* a sequence of 2D BEV points, which are arranged following the lane direction. For any two lanes which are directly reachable, 4 types of connections are given: *predecessor*, *successor*, *left neighbour* and *right neighbour*. Given a lane A, its predecessor and successor are the lanes which can directly travel to A and from A respectively. Left and right neighbours refer to the lanes which can be directly reached without violating traffic rules. This simple map format provides essential geometric and semantic information for MP, as vehicles generally plan their routes by reference to lane centerlines and their connectivity.

In order to conduct the Lane Graph construction, we first define a lane node as the straight line segment formed by any two consecutive points (grey circles in Figure 7.2) of the centerline. The location of a lane node is the averaged coordinates of its two end points. Following the connections between lane centerlines, we also derive 4 connectivity types for the lane nodes, *i.e. predecessor*, *successor*, *left neighbour* and *right neighbour*. For any lane node A, its predecessor and successor are defined as the neighbouring lane nodes that can travel to A or from A respectively. Note that one can reach the first lane node of a lane l_A from the last lane node of lane l_B if l_B is the predecessor of l_A. Left and right neighbours are defined as the spatially closest lane node measured by ℓ_2 distance on the left and on the right neighbouring lane respectively. We denote the lane nodes with $V \in \mathbb{R}^{N\times 2}$, where N is the number of lane nodes and the i-th row of V is the BEV coordinates of the i-th node. We represent the connectivity with 4 adjacency matrices $\{A_i\}_{i\in\{\text{pre,suc,left,right}\}}$, with $A_i \in \mathbb{R}^{N\times N}$. We denote $A_{i,jk}$, as the element in the j-th row and k-th column of A_i. Then $A_{i,jk} = 1$ if node k is an i-type neighbor of node j.

7.2.3.1. LaneConv Operator

A natural operator to handle lane graphs is the graph convolution [128]. The most widely used graph convolution operator [129] is defined as $Y = LXW$, where $X \in \mathbb{R}^{N\times F}$ is the node feature, $W \in \mathbb{R}^{F\times O}$ is the weight matrix, and $Y \in \mathbb{R}^{N\times O}$ is the output. The graph Laplacian matrix $L \in \mathbb{R}^{N \times N}$ takes the form $L = D^{-1/2}(I + A)D^{-1/2}$, where I, A and D are the identity, adjacency and degree matrices respectively. I and A account for self connection and connections between different nodes. All connections share the same weight W, and the degree matrix D is used to normalize the output. However, this vanilla graph convolution is inefficient in our case due to the following reasons. First, it is not clear what kind of node feature will preserve the information in the lane graphs. Second, a single graph Laplacian can not capture the connection type, *i.e.* losing the directional information carried by the connection type. Third, it is not straightforward to handle long range dependencies within this form of graph convolution. Motivated by these challenges, we introduce our novel specially designed operator for lane graphs, called *LaneConv*.

7.2.3.1.1. Node Feature We first define the input feature of the lane nodes. Each lane node corresponds to a straight line segment of a centerline. To encode all the lane node information, we need to take into account both the shape (size and orientation) and the location (the coordinates of the center) of the corresponding line segment. We parameterize the node feature as follows:

$$\mathbf{x}_i = \text{MLP}_{\text{shape}}\left(\mathbf{v}_i^{\text{end}} - \mathbf{v}_i^{\text{start}}\right) + \text{MLP}_{\text{loc}}\left(\mathbf{v}_i\right) \tag{7.1}$$

where MLP indicates a multi-layer perceptron and the two subscripts refer to shape and location, respectively. $\mathbf{v}_i$ is the location of the i-th lane node, *i.e.*, the center between two end points, $\mathbf{v}_i^{\text{start}}$ and $\mathbf{v}_i^{\text{end}}$ are the BEV coordinates of the node i's starting and ending points, and $\mathbf{x}_i$ is the i-th row of the node feature matrix X, denoting the input feature of the i-th lane node.

7.2.3.1.2. LaneConv The node feature above only captures the local information of a line segment. To aggregate the topology information of the lane graph at a larger scale, we design the following LaneConv operator:

$$Y = XW_0 + \sum_{i \in \{\text{pre,suc,left,right}\}} A_i X W_i \tag{7.2}$$

where A_i and W_i are the adjacency and the weight matrices corresponding to the i-th connection type respectively. Since we order the lane nodes from the start to the end of the lane, A_{suc} and A_{pre} are matrices obtained by shifting the identity matrix one step towards upper right (non-zero superdiagonal) and lower left (non-zero subdiagonal). A_{suc} and A_{pre} can propagate information from the forward and backward neighbours whereas A_{left} and A_{right} allow information to flow from the cross-lane neighbours. It is not hard to see that our LaneConv builds on top of the general graph convolution and encodes more geometric (*e.g.* connection type/direction) information.

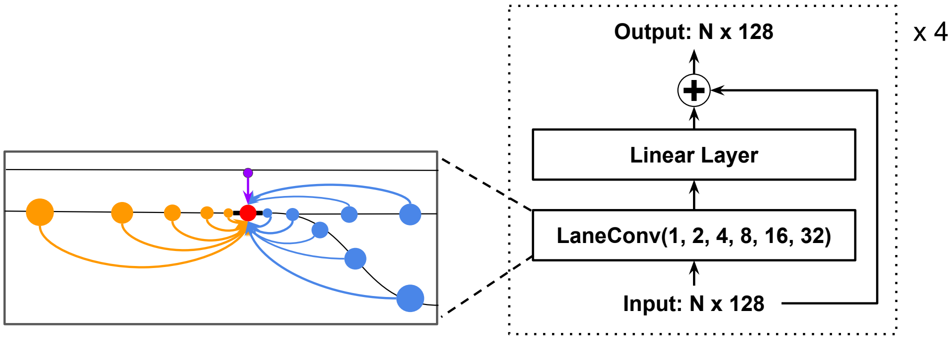


Figure 7.3: LaneGCN architecture. LaneGCN is a stack of 4 multi-scale LaneConv residual blocks, each of which consists of a LaneConv(1,2,4,8,16,32) and a linear layer with a residual connection. Source: *Learning lane graph representations for motion forecasting* [13]

7.2.3.1.3. Dilated LaneConv Since motion forecasting models usually predict the future trajectories of actors with a time horizon of several seconds, actors with high speed could have moved a long distance. Therefore, the model needs to capture the long range dependency along the lane direction for accurate prediction. In regular grid graphs, a dilated convolution operator [130] can effectively capture the long range dependency by enlarging the receptive field. Inspired by this operator, we propose the *dilated LaneConv* operator to achieve a similar goal for irregular graphs.

In particular, the k-dilation LaneConv operator is defined as follows:

$$Y = XW_0 + A^k_{\text{pre}} XW_{\text{pre},k} + A^k_{\text{suc}} XW_{\text{suc},k} \tag{7.3}$$

where A^k_{pre} is the k-th matrix power of A_{pre}. This allows us to directly propagate information along the lane for k steps, with k a hyperparameter. Since A^k_{pre} is highly sparse, one can efficiently compute it using sparse matrix multiplication. Note that the dilated LaneConv is only used for predecessor and successor, as the long range dependency is mostly along the lane direction.

7.2.3.2. LaneGCN

Based on the dilated LaneConv, we further propose a multi-scale LaneConv operator and use it to build our LaneGCN. Combining Eq. (7.2) and (7.3) with multiple dilations, we get a multi-scale LaneConv operator with C dilation sizes as follows:

$$Y = XW_0 + \sum_{i \in \{\text{left,right}\}} A_i XW_i + \sum_{c=1}^{C} \left(A^{k_c}_{\text{pre}} XW_{\text{pre},k_c} + A^{k_c}_{\text{suc}} XW_{\text{suc},k_c} \right), \tag{7.4}$$

where k_c is the c-th dilation size. We denote LaneConv$(k_1, \cdots, k_C)$ this multi-scale layer.

7.2.4. Enhanced Actor-Map Fusion Cycle

Once both the map and social latent features are computed, we obtain a 2D feature matrix X where each row X_i indicates the feature of the i-th actor, and a 2D matrix Y where each row Y_i indicates the feature of the i-th lane node. Then, we make use of the well-established actor-map fusion cycle [13] (also referred as FusionNet in the literature) that transfers and aggregates feature among actors and lane nodes. The behaviour of an actor strongly depends on its context, *i.e.*, other actors and the map. Although the interactions between actors has been explored by previous work, the interactions between the actors and the map, and map conditioned interactions between actors have received much less attention. FusionNet makes use of spatial attention and LaneGCN to capture a complete set of actor-map interactions, as observed in Figure 7.1.

FusionNet makes use of a stack of four fusion modules to capture all information flows between actors and lane nodes, *i.e.*, (1) Actors to Lanes (A2L), (2) Lanes to Lanes (L2L), (3) Lanes to Actors (L2A) and (4) Actors to Actors (A2A). This order is not coincidence. Assuming several agents are on the road and all of them have a web service that provides detailed information about geographical regions and sites worldwide (*e.g.* Google Maps):

- First, agents introduce their real-time traffic information to lane nodes, such as blockage, presence of an accident, etc. In other words, the HD map information is updated with the traffic information of the agents.
- The HD map information of a particular node is propagated to the immediate neighbours, in such a way ...
- ... further agents are updated with the information of surrounding map nodes, which were previously updated by neighboring nodes.
- Finally, A2A concludes the actor-map fusion cycle handling the interactions between actors and produces the output actor features.

We implement L2L using another LaneGCN, which has the same architecture as the one used in our MapNet. Regarding the A2L, L2A and A2A modules, [13] applies a spatial attention layer as defined in Section 3.3.4. Taking the first module (A2L) as en example, given an actor node i, we aggregate the features from its context lane nodes j as follows:

$$\mathbf{y}_i = \mathbf{x}_i W_0 + \sum_j \phi(\text{concat}(\mathbf{x}_i, \Delta_{i,j}, \mathbf{x}_j) W_1) W_2 \tag{7.5}$$

where $\mathbf{x}_i$ represents the feature of the i-th node, W a weight matrix, ϕ the composition of layer normalization and ReLU, and $\Delta_{ij} = \text{MLP}(\mathbf{v}_j - \mathbf{v}_i)$, where $\mathbf{v}$ denotes the node location.

The context nodes are defined to be the lane nodes whose ℓ_2 distance from the actor node i is smaller than a threshold. Each of A2L, L2A and A2A has two residual blocks, which consist of a stack of the proposed attention layer and a linear layer, as well as a residual connection.

Nevertheless, as observed in our model (Figure 7.1), once implemented FusionNet to compute actor-map interactions, we substitute the final module that models Actor to Actor interactions with a Graph Convolution Operator (GCN), inspired in our previous proposed efficient baselines (Figure 6.1).

7.2.5. Goal prediction

In stage two, we predict possible goals for the i-th actor based on X_i. We apply intermediate supervision and calculate the smooth L1 loss between the best-predicted goal and the ground-truth trajectory endpoint to backpropagate, making the predicted goal close to the actual goal as much as possible. The goal prediction stage serves as a predictive test to locate goal areas, which is different from goal-based methods using the predicted goals as the final predicted trajectories' endpoint.

In practice, a driver's driving intent is highly multi-modal. For example, he or she may stop, go ahead, turn left, or turn right when approaching an intersection. Therefore, we try to make a multiple-goals prediction. We construct a goal prediction header as proposed by [11] with two branches to predict E possible goals $G_{n,end} = \{g^e_{n,end}\}_{e\in[0,E-1]}$ and their confidence scores $C_{n,end} = \{c^e_{n,end}\}_{e\in[0,E-1]}$, where $g^e_{n,end}$ is the e-th predicted goal coordinates and $c^e_{n,end}$ is the e-th predicted goal confidence of the n-th actor.

We train this stage using the sum of classification loss and regression loss. Given E predicted goals, we find a positive goal $\hat{e}$ that has the minimum Euclidean distance with the ground truth trajectory's endpoint.

For classification, we use the max-margin loss:

$$L_{cls\_end} = \frac{1}{N(E-1)} \sum_{n=1}^{N} \sum_{e\neq\hat{e}} max(0, c^e_{n,end} + \epsilon - c^{\hat{e}}_{n,end}) \tag{7.6}$$

where N is the total number of actors and $\epsilon = 0.2$ is the margin. The margin loss expects each goal to capture a specific pattern and pushes the goal closest to the ground truth to have the highest score. For regression, we only apply the smooth L1 loss to the positive goals:

$$L_{reg\_end} = \frac{1}{N} \sum_{n=1}^{N} reg(g^{\hat{e}}_{n,end} - a^*_{n,end}) \tag{7.7}$$

where $a^*_{n,end}$ is the ground truth BEV coordinates of the n-th actor trajectory's endpoint, $reg(z) = \sum_i d(z_i)$, z_i is the i-th element of z, and $d(z_i)$ is a smooth L1 loss.

Additionally, we also try to add a "one goal prediction" module at each trajectory's middle position aggregating map features to assist the endpoint goal prediction and the whole trajectory prediction. Similarly, we apply a residual MLP to regress a middle goal $g_{n,mid}$ for the n-th actor. The loss term for this module is given by:

$$L_{reg\_mid} = \frac{1}{N}\sum_{n=1}^{N} reg(g_{n,mid} - a^*_{n,mid}) \tag{7.8}$$

where $a^*_{n,mid}$ is the ground truth BEV coordinates of the n-th actor trajectory's middle position.

The total loss at the goal prediction stage is:

$$L_1 = \alpha_1 L_{cls\_end} + \beta_1 L_{reg\_end} + \rho_1 L_{reg\_mid} \tag{7.9}$$

where $\alpha_1 = 1$, $\beta_1 = 0.2$ and $\rho_1 = 0.1$.

7.2.6. GoICrop

We choose the predicted goal with the highest confidence among E goals as an anchor. This anchor is the approximate destination with the highest possibility that the actor may reach based on its motion history and driving context. Because the actors' motion is highly uncertain, we crop maps within 6 meters of the anchor as the goal area of interest, which relaxes the strict goal prediction requirement. The actual endpoint is more likely to appear in candidate areas compared with being hit by scattered endpoint predictions. Moreover, the actor's behavior highly depends on its destination area's context, i.e., the maps and other actors. Although previous works have explored the interactions between actors, the interactions between actors and maps in goal areas and the interactions among actors in the future have received less attention.

Thus, we retrieve the lane nodes in goal areas and apply a GoICrop module to aggregate these map node features as follows:

$$x'_i = \phi_1(x_i W_0 + \sum_j \phi_2(concat(x_i W_1, \Delta_{i,j}, y_j) W_2)) W_3 \tag{7.10}$$

where x_i is the feature of i-th actor and and y_j is the feature of j-th lane node, W_i is a weight matrix, ϕ_i is a layer normalization with ReLU function, and $\Delta_{i,j} = \phi(MLP(v_i - v_j))$, where v_i denotes the anchor's coordinates of i-th actor and v_j denotes the j-th lane node's coordinates.

GoICrop serves as spatial distance-based attention and updates the goal area lane nodes' features back to the actors. We transpose x_i with W_1 as a query embedding. The relative distance feature between the anchor of i-th actor and j-th lane node are extracted by $\Delta_{i,j}$. Then, we concatenate the query embedding, relative distance feature, and lane node feature. An MLP is employed to transpose and encode these features. Finally, the goal area features are aggregated for i-th actor.

Previous motion forecasting methods usually focus on the interactions in the observation history. However, actors will interact with each other in the future to follow driving etiquette, such as avoiding collisions.

Since we have performed predictive goal predictions and gotten possible goals for each actor, our framework can model the actors' future interactions.

Hence, we utilize the predicted anchor positions and apply a GoICrop module as equation 7.10 to implicitly model actors' interactions in the future. We consider the other actors whose future anchor's distance from the anchor of i-th actor is smaller than 100 meters. In this case, y_j in equation 7.10 denotes the features of j-th actor, v_i denotes the anchor's coordinates of i-th actor, and v_j denotes the anchor's coordinates of j-th actor in $\Delta_{i,j} = \phi(MLP(v_i - v_j))$.

7.2.7. Decoding module

In order to get the future trajectories with corresponding scores, as observed in Figure 7.1, we take the updated actor features X as input to predict K final future trajectories and their confidence scores in stage three. Specifically, we construct a two-branch multi-modal prediction header similar to the goal prediction stage, with one regression branch estimating the trajectories and one classification branch scoring the trajectories.

For each actor, we regress K sequences of BEV coordinates $A_{n,F} = \{(a^k_{n,1}, a^k_{n,2}, ..., a^k_{n,T})\}_{k\in[0,K-1]}$, where $a^k_{n,t}$ denotes the n-th actor's future coordinates of the k-th mode at t-th step.

For the classification branch, we output K confidence scores $C_{n,cls} = \{c^k_n\}_{k\in[0,K-1]}$ corresponding to K modes.

We find a positive trajectory of mode $\hat{k}$, whose endpoint has the minimum Euclidean distance with the ground truth endpoint.

For classification, we use the margin loss L_{cls} similar to the goal prediction stage. For regression, we apply the smooth L1 loss on all predicted steps of the positive trajectories:

$$L_{reg} = \frac{1}{NT} \sum_{n=1}^{N} \sum_{t=1}^{T} reg(a^{\hat{k}}_{n,t} - a^*_{n,t}) \tag{7.11}$$

where $a^*_{n,t}$ is the n-th actor's ground truth coordinates.

To emphasize the importance of the goal, we add a loss term stressing the penalty at the endpoint:

$$L_{end} = \frac{1}{N} \sum_{n=1}^{N} reg(a^{\hat{k}}_{n,end} - a^{*}_{n,end}) \tag{7.12}$$

where $a^{*}_{n,end}$ is the n-th actor's ground truth endpoint coordinates and $a^{\hat{k}}_{n,end}$ is the n-th actor's predicted positive trajectory's endpoint.

The loss function for training at this stage is given by:

$$L_2 = \alpha_2 L_{cls} + \beta_2 L_{reg} + \rho_2 L_{end} \tag{7.13}$$

where $\alpha_2 = 2$, $\beta_2 = 1$ and $\rho_2 = 1$.

7.2.8. Motion refinement

A second-stage motion refinement, as proposed by [131], is introduced to further explore the temporal consistency for predicting more accurate future trajectories. The goal is to reduce the offset between ground truth trajectory Y and predicted trajectory $\hat{Y}$. We define this offset as $\Delta Y = Y - \hat{Y}$. In this stage, we leverage three sources of information: 1. Preliminary predictions computed in the decoding module, 2. Prior latent information to the decoding module and 3. Past observations. Using this approach, an MLP model is trained to minimize the offset by predicting a residual R that is added to the original trajectory *i.e.*we use L_2 loss to optimize the offset as follows:

$$\mathcal{L}_{\text{off}} = ||Y - \hat{Y} - \hat{R}||_2 = ||\Delta Y - \hat{R}||_2. \tag{7.14}$$

Furthermore, we use a cosine function, denoted by Equation 7.15, to explicitly aid the model in learning the turning angle from the last observed position. It measures the difference between the ground truth angle $\theta_t = \text{arctan2}(Y_t - X_0)$ and the predicted angle $\hat{\theta}_t = \text{arctan2}(\hat{Y}_t - X_0)$:

$$\mathcal{L}_{\text{angle}} = \frac{1}{t_f} \sum_{t=1}^{t_f} -cos(\hat{\theta}_t - \theta_t) \tag{7.15}$$

Note that this method can be applied to the pre-trained model from previous stages, which is completely functional, as the main function is to improve the output trajectories.

7.3. Experimental Results

We use the Argoverse 2 [6] dataset described in Section 2.4, where we could appreciate that, compared to Argoverse 1, is a high-quality multi-agent motion prediction dataset. For each real driving scenario we have the corresponding local HD map, past trajectories of the agents, metadata about the agents (*e.g.* the type of agent: cyclist, pedestrian, car), and topological information about the scene. Each scenario is 11 seconds long. We consider five seconds of the past trajectory (also known as motion history), and we predict the next six seconds.

Implementation.

We train our model on 2 A100 GPUs using a batch size of 128 with the Adam optimizer for 42 epochs. The initial learning rate is 1 x 10-3, decaying to 1 x 10-4 at 32 epochs. The latent dimension (regarding map and social features) in most of our experiments is 128. The number of attention heads in the social encoder and motion refinement is 8. The training setup including loss functions follows GANet [11] official implementation as our baseline.

Augmentations (i) Dropout and swapping random points from the past trajectory, (ii) point location perturbations under a $\mathcal{N}(0, 0.2)$ [m] noise distribution [114].

Tables 7.1 and 7.2 present the results obtained on the Argoverse 2 Motion Forecasting validation and test sets, respectively. We achieve near state-of-the-art performance in both sets, which is on-pair with the most promising pipelines, while using notably less parameters. As stated throughout this work, we focus on applying efficient methods to help understand future interactions among the different agents, reducing the number of parameters and inference time.

We can appreciate in Table 7.1 the huge influence of the physical context both in terms of accuracy and runtime. GANet [11] shows the best multimodal prediction metrics, with an approximate amount of 6.2M of parameters of 1612 ms given a batch size of 128 traffic scenarios and an average number of 30 agents per scene. As expected, progressively removing the map influence (remove map from decoder, remove goal areas estimation) in the model we decrease the MP performance with a noticeable parameter decrease.

In our case, we study the influence of substituting the modified ActorNet [11], [13] social encoder proposed by GANet, which uses RNNs. Our proposal replaces these by a Linear embedding, a Positional Encoding and Encoder Transformer. Moreover, we add the aforementioned agent metadata (object type and track category), and we substitute the Actor to Actor attention of the fusion cycle for a GCN [12] operator to enhance agents global interaction. It can be appreciated how we obtained a similar performance (both with 128 latent dimension in *Ours-m*, and 64 latent in *Ours-s*), reducing the parameters and inference time.

Finally, our best model, which includes heuristic proposals that serve as a preliminar multi-modal guidance for the model and motion refinement to improve the quality of the final predictions, obtains a performance on pair with [11], reducing the number of parameters and inference time about 21% and 41% respectively. We can appreciate in Table 7.2 how our model generalizes well in the test set, with results (both in uni-modal and multi-modal prediction) up-to-pair with other state-of-the-art algorithms.

Table 7.1: Comparison of methods in the Argoverse 2 Validation Set. We show the number of parameters for each model, prediction metrics (minADE, minFDE and brier-minFDE) for the multimodal scenario (k=6) and runtime. Runtime was measured on a single GPU A100-SXM4 (using batch 128). Our experiments are indicated using †. We use as baseline method GANet [11].

Method	Map	# Par. (M)	minADE (m) ↓	minFDE (m) ↓	brier-minFDE (m) ↓	Runtime (ms) ↓
GANet [11]	Yes	6.2	0.806	1.402	2.02	1612
GANet w/o Map Decoder [11]	Yes	5.7	0.84	1.55	2.18	1353
GANet w/o Goal Areas [11]	Yes	4.5	0.87	1.66	2.29	1134
GANet w/o map [11]	No	1.79	1.034	2.212	2.825	838
† CRAT-Pred [12]	No	0.53	1.31	2.78	3.65	223
† Ours-base: GANet [11] ActorNet → Attention Transformer	Yes	5.0	0.83	1.45	2.07	923
† Ours-m: Ours-base + A2A → C-GCN + Metadata	Yes	4.74	0.82	1.43	2.05	892
† Ours-s: Ours-base + A2A → C-GCN + Metadata (64 latent size)	Yes	1.2	0.88	1.53	2.15	893
† **Ours:** Ours-m + Proposals + Motion Refinement	Yes	4.92	0.81	1.42	2.04	946

We provide advanced qualitative samples in Figure 7.4, where we show the HD Map of real traffic scenes, heuristic trajectory proposals in the form of centerlines, and the multimodal predictions from our model including their respective confidences (the higher, the most probable).

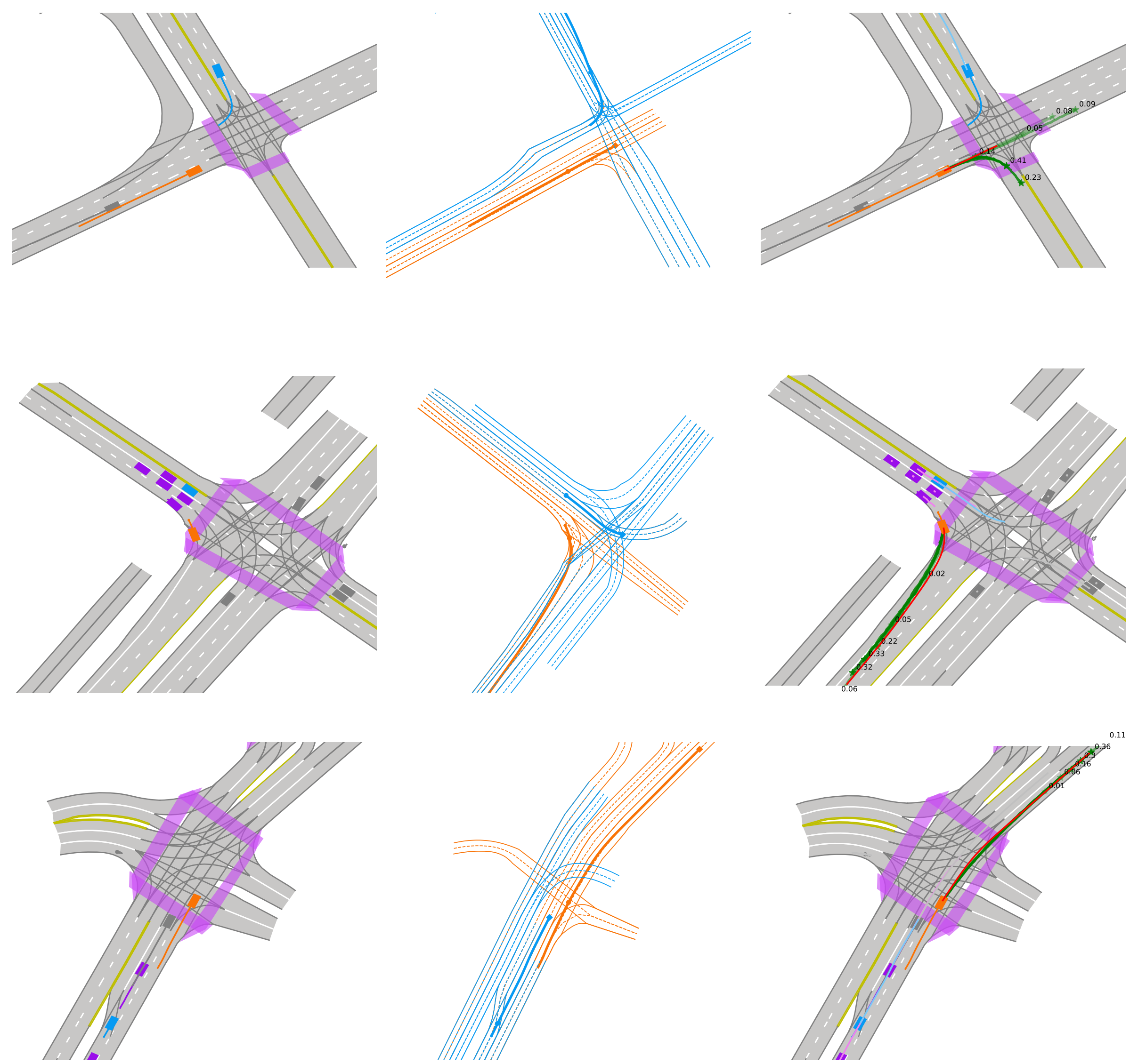
As we discussed throughout this Chapter, we designed our model to ensure realistic predictions. We can appreciate that all the predictions are plausible and constrained to the scene geometry *e.g.* lane distribution and centerlines. We believe our heuristic proposals help to regularize the model and produce realistic predictions that would ensure traffic safety. For simplicity, we only illustrate the heuristic proposals for the focal agent and ego-vehicle. We believe our software for qualitative analysis of MP models on the well-known Argoverse 2 [6] is fundamental and a core contribution.

Table 7.2: Results on the Argoverse 2 Motion Forecasting Leaderboard. The "-" denotes that this result was not reported in their paper. Some numbers are borrowed from [11]. For all the metrics, the lower, the better.

Method	b-minFDE (K=6)	MR (K=6)	minFDE (K=6)	minADE (K=6)	minFDE (K=1)	minADE (K=1)	MR (K=1)
DirEC	3.29	0.52	2.83	1.26	6.82	2.67	0.73
drivingfree	3.03	0.49	2.58	1.17	6.26	2.47	0.72
LGU	2.77	0.37	2.15	1.05	6.91	2.77	0.73
Autowise.AI(GNA)	2.45	0.29	1.82	0.91	6.27	2.47	0.71
Timeformer [132]	2.16	0.20	1.51	0.88	4.71	1.95	0.64
QCNet	2.14	0.24	1.58	0.76	4.79	1.89	0.63
OPPred w/o Ensemble [125]	2.03	0.180	1.389	0.733	4.70	1.84	0.615
TENET w/o Ensemble [133]	2.01	-	-	-	-	-	-
Polkach(VILaneIter)	2.00	0.19	1.39	**0.71**	4.74	1.82	0.61
GANet	**1.969**	**0.171**	**1.352**	0.728	**4.475**	**1.775**	**0.597**
Ours	1.98	0.185	1.37	0.73	4.53	1.79	0.608

7.4. Summary

In this Chapter we solve the challenging problem of Multi-Agent Motion Prediction in real driving scenarios. We present an end-to-end pipeline that combines Deep Learning (DL) and heuristic scene understanding. Our model uses as input the map of the scene, the past trajectories of the agents, and additional information about the scene geometry and agents e.g., type of agent, lane distribution. We propose a model that integrates attention mechanisms with GNNs, heuristic goals, and a motion refinement module to further improve temporal consistency. We achieve SOTA results on the Argoverse 2 Motion Forecasting Benchmark reducing in millions of parameters previous methods such as GANet, and improving over LaneGCN. Our code is publicly available. As future works, we plan to include map-adaptive lane-loss to improve diverse multiple motion prediction and explore knowledge-distillation to improve the efficiency for real-world deployment.



HD Map input Heuristic proposals Multimodal (k=6) predictions

Figure 7.4: Qualitative Results on challenging scenarios in Argoverse 2 using our best model. We represent: our vehicle (**ego**), the **focal agent**, the **relevant agents** in the scene, and **other agents**. We can also see the **ground-truth** trajectory of the target agent, our **multimodal predictions** (with the corresponding **confidences**). We also highlight the most important topology of the road, such as pedestrian crossing and boundaries mark type. We show, from left to right, a general view of the traffic scenario (including and map information), the heuristic proposals for each agent (we only include the **ego** and **focal agent** for simplicity) and the multimodal prediction ($k = 6$) for the **focal agent**, including the corresponding confidences (the higher, the most probable)

Chapter 8

Applications in Autonomous Driving

Trabaja duro en silencio, y deja que tu éxito haga todo el ruido.

Autor original: Frank Ocean

8.1. Introduction

In this Chapter we will detail the experiments carried out to assess the performance of our final proposal (excluding the map data for simplicity), both in terms of accuracy and computational resources (time, Hz) for real-time applications in the field of AD.

First, regarding the decision-making layer we make use of the SMARTS [134] framework based on the SUMO (Simulation of Urban MObility) [135] simulator, where we study the influence of the prediction pipeline when computing the most optimal action for the ego-vehicle in contrast to reactive proposals where only the past observations or current adversaries positions are required.

Second, the prediction pipeline is integrated in the ADS of our research group using the CARLA [61] (CAR Learning to Act) to study how the ego-vehicle can leverage the scene understanding and the future behaviour prediction of the agents to improve the overall score by means of a holistic validation using the CARLA Leaderboard, inspired in the well-established NHTSA typology. We will make use of a simple-yet-powerful concept such us ray-tracing, inspired in [136], to obtain a realistic ground-truth to emulate the input of multi-tracked agents provided by the different datasets, where the main idea is to get those agents which are within a certain radius with respect to the ego-vehicle and are observed by the rays of the LiDAR sensor in such a way a theoretical sensor fusion could detect and track that object. By assuming this, we will be able to deploy and validate (since we have the past and future of the scene) our prediction pipeline in the overall architecture in an efficient and isolated way as a preliminary stage before integrating future object detector or tracking algorithms both in simulation or our real-world vehicle.

8.2. Decision-Making

The increasing popularity of autonomous vehicles (AVs) has brought with it significant challenges in ensuring safe and effective decision-making, particularly in complex urban driving scenarios [137]. Reinforcement learning (RL) techniques have emerged as a promising solution to address these challenges [138]. They enable AVs to learn from their interactions with the driving environment, without relying on pre-defined rules. However, RL-based approaches still face a number of limitations that can hinder their development, including issues related to state representation. In recent years, RL has emerged as a promising approach for developing decision-making policies for AVs [139], outperforming ruled-based approaches that usually cannot solve complex situations [140]. However, a large number of interactions with the environment are required to obtain the desired policy. This is why other approaches such as imitation learning [141] and inverse RL [142], based on human experts' behaviours are also used in the literature.

A key challenge in RL-based AVs is the development of effective state representations that can account for the complexity of urban driving scenarios. Unlike in simpler environments, state representations for urban driving must be able to incorporate a wide range of information, including dynamic features of traffic flows and interactions among different agents. Finding ways to effectively encode this information and develop accurate state representations is essential to enabling RL-based AVs to generalize to various scenarios and make effective driving decisions. Distilling predictive information from scene representations can aid in the development of effective decision-making policies for AVs. By better understanding the potential consequences of different driving actions, RL-based AVs can make more informed decisions that lead to safer and more efficient driving behaviour.

In this application we propose an approach to enhance the efficiency and generalization of RL-based AVs in urban driving scenarios. Specifically, we introduce the use of a Motion Prediction (MP) module to obtain the future positions of the ego-vehicle and the surrounding vehicles (adversaries) in the scenario. These predictions are the input to an RL-based decision-making module that executes high-level actions. Our approach is developed using the Proximal Policy Optimization (PPO) algorithm [143]. We carry out an evaluation in the unsignalized T-intersection scenario shown in Fig. 8.1 with and without the proposed state representation and provide a comparison with some baseline methods.

This application is focused on a critical issue for RL-based AVs, which is the state representation problem. Traditional state representations often focus on low-dimensional features such as distance to obstacles, lane positions, and vehicle velocities [144]. However, these representations may not be sufficient to capture the complex interactions among different agents and road structures in urban driving scenarios. To address the state representation problem, some methods have been proposed that use higher-dimensional or learned representations, such as convolutional neural networks [145] and recurrent neural

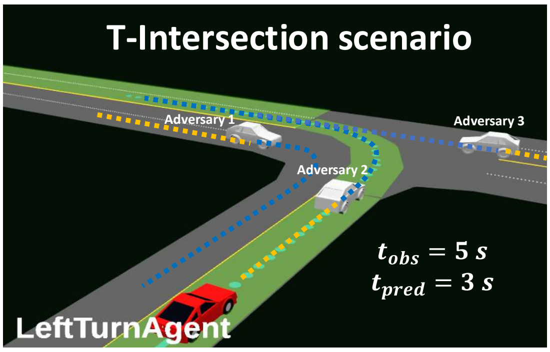


Figure 8.1: T-intersection scenario in the SMARTS simulator. The past positions of the adversaries (yellow) and the predicted trajectories (blue) are represented in the scenario.

networks [146]; other methods have been proposed to use more detailed representations, such as Bird-Eye-View images [147], image augmentation [148] or occupancy grids [149]. These methods have shown promising results in improving the generalization and robustness of the decision-making approaches. Recently, transformer-based approaches have gained increasing attention for their ability to capture long-term dependencies and interactions among different entities in sequential data. In the context of AVs, transformers have been used to reduce the computational load in end-to-end approaches [150] and anticipate future states with prediction-aware planning [151].

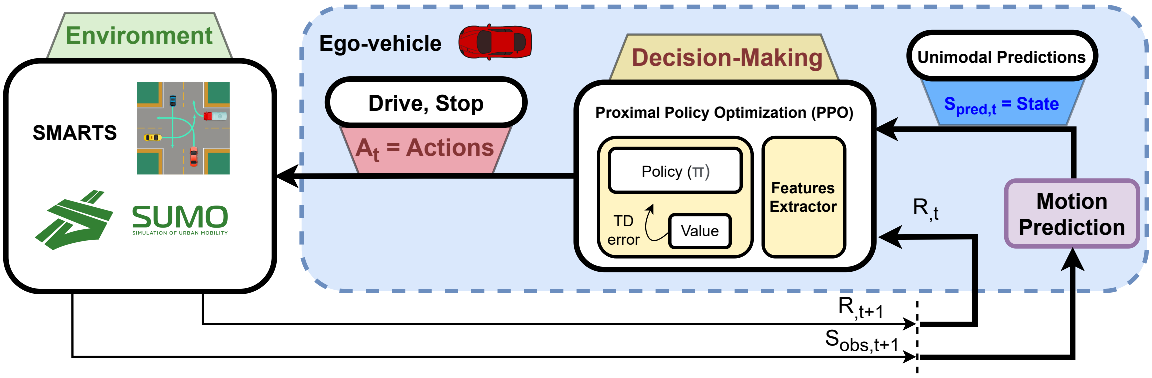


Figure 8.2: An overview of the Augmented Reinforcement Learning with Efficient Social-based Motion Prediction for Autonomous Decision-Making. The observations (both position and ID, so, trackers) of the vehicles in the scenario are obtained from the simulator. The MP module estimates the future positions of these vehicles, taking into account the most plausible score of a multimodal prediction. The decision-making module selects high-level actions based on this information. These actions are executed by the simulator, which provides a new state to the framework.

The objective of this study is to illustrate the efficacy of employing a low-dimensional state representation in conjunction with an MP method. We aim to prove that the proposed framework can lead to good performance in urban scenarios. More specifically, this application presents the following contributions:

- The augmentation of RL techniques with MP to improve state representation. By predicting vehicle trajectories, we can better capture the complex interactions between different agents and road structures in urban driving scenarios.

- Higher explainability than end-to-end methods. Intermediate states are accessible in our approach. This can help to understand the decisions made.

- We provide a comparison with baseline methods in a standard scenario. We demonstrate that our approach leads to some improvements in performance, particularly in scenarios with high velocities.

8.2.1. Our approach

The RL framework proposed in this work, which executes high-level decisions to solve urban driving scenarios, is represented in Figure 8.2. The past observations of the position of adversaries are obtained from the environment. This information is provided to the MP module, which estimates future positions. The PPO algorithm takes these predictions and generates the decision-making output.

We propose two different learning processes: supervised learning for the motion prediction module and a reinforcement learning approach for the decision-making module. These two modules are trained separately, which allows access to the information of the predictions that feed the decision-making module.

8.2.1.1. Efficient Social-based Prediction stage

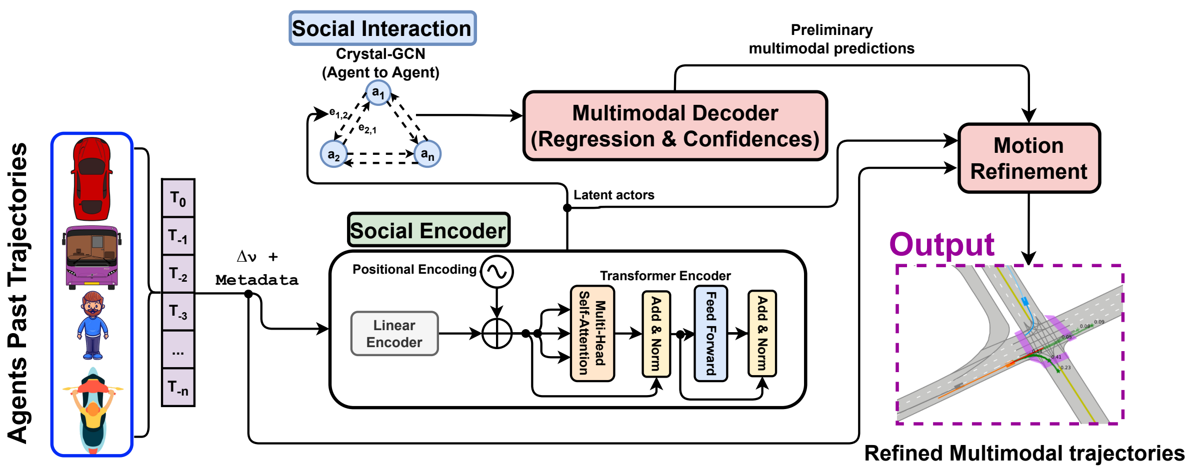


Figure 8.3: Overview of our Efficient Social-based Motion Prediction. The main inputs are the relative displacements and centers (last observations) of the agents in the ego-vehicle frame. The relative displacements, agents centers and additional metadata are encoded through a transformer encoder and interactions are computed by means of a Crystal-GCN. Finally, K final future trajectories (modes) and their confidence scores are computed and refined through a multimodal decoder and motion refinement module respectively.

As observed throughout this thesis, predicting the future behaviour of traffic agents around the ego-vehicle is one of the key unsolved challenges in reaching full self-driving autonomy and it is required to be multi-modal, which means given the past motion of a particular vehicle and its surrounding scene, there may exist more than one possible future behaviour. Therefore, MP models need to cover the different choices a driver could make (*i.e.* going straight or turning, accelerations or slowing down) as a possible trajectory in the immediate future or as a probability distribution of the agents future location. In other words, when an ADS attempts to make a specific action (*e.g.* left turn, brake or accelerate), it must consider the future motion of the other vehicles, since the own future actions (also known as decision-making or behaviour planning) depends on the all possible maneuvers of the other agents of the scene for safe driving.

Since our proposed decision-making and MP pipeline (Figure 8.2) is multi-stage to provide a more interpretable framework, we follow the principles of the Argoverse 2 Motion Forecasting dataset [6] to train our prediction model. In our case, we build an efficient model solely based on past trajectories (motion history, $obs_{len} = 50$) and agents interactions, taking into account the corresponding traffic rules, not requiring fully-annotated HD map information, to predict $obs_{len} = 60$ future steps.

The SMARTS [134] framework provides only the positions of the agents in the timestamp t. Nevertheless, in order to predict the future $pred_{len}$ trajectories of the agents, we require their corresponding obs_{len} trackers over a certain set of observations. Most vehicle prediction datasets [6] aim to predict the future behaviour of a target agent assuming the surrounding agents have been detected and tracked (so, monitored over time) and the map information is also provided. In that sense, since SMARTS provide the agents in the same order for consecutive timestamps (that is, the agent 5, unless it disappears from the scene, will be the agent 5 again in the next frame), we are able to compute a FIFO (*First Input First Output*) for each agent, not requiring data association [67] to perform this task.

On top of that, as proposed by multiple methods [13], [126], we consider only the vehicles that are observable at $t{=}0$, handling those agents that are not observed over the full sequence spectrum (observation length = obs_{len} + prediction length = $pred_{len}$) by concatenating a binary flag b_i^t that indicates if the agent is padded or not. In particular, we filter the static elements and track fragments scored by Argoverse 2 to get only the most relevant traffic agents, reducing the number of agents to be considered in complex traffic scenarios. Furthermore, to make the model translation and rotation invariant, the coordinate system in our model is BEV-centered of a given target agent at $t = 0$, and we use the orientation from the target location given in the same timestamp as the positive x-axis. Note that this representation will benefit the model to have a common representation to enhance the generalization of the model and prevent overfitting. Once the scene has been translated and rotated, instead of using absolute 2D-BEV (xy plane), the input for the agent i is a series of relative displacements, as stated throughout this thesis.

Then, as stated in Chapter 7, we concatenate the agents past trajectories and additional social metadata in order to be processed by a linear embedding. Then, positional encoding is added to the output embedding explicitly to retain the information regarding the order of past trajectories and future preliminar steps. Finally, these latent features feed the transformer encoder, leveraging the self-attention mechanism and positional encoding to learn complex and dynamic patterns from long-term time series data. Once we have the latent vector of the different agents, as observed in Figure 8.3, we learn complex agent-agent interactions by means of a Crystal-GCN, which output is finally introduced into the multi-modal decoder. Taking the final actor features after motion history and agents interaction, a multi-modal prediction header outputs the final motion forecasting. For each agent, it predicts K possible future trajectories and their confidence scores. The header has two branches, a regression branch to predict the trajectory of each mode and a classification branch to predict the confidence score of each mode.

For the m-th actor, a residual block and a linear layer in the regression branch to regress the K sequences of BEV coordinates is obtained:

$$O_{m,\text{reg}} = \{(\mathbf{p}^k_{m,1}, \mathbf{p}^k_{m,2}, ..., \mathbf{p}^k_{m,T})\}_{k\in[0,K-1]} \tag{8.1}$$

where $O_{m,\text{reg}}$ is the whole set of regressions and $\mathbf{p}^k_{m,i}$ is the predicted m-th actor's BEV coordinates of the k-th mode at the i-th time step.

On the other hand, for the classification branch, a MLP to $\mathbf{p}^k_{m,T} - \mathbf{p}_{m,0}$ to get K distance embeddings is applied. Finally, each distance embedding is concatenated with the actor feature, applying a residual block and a linear layer to output K confidence scores, $O_{m,\text{cls}} = (c_{m,0}, c_{m,1}, ..., c_{m,K-1})$.

Finally, as stated in Chapter 7, a motion refinement modules takes into account the preliminary multi-modal predictions computed by the decoder, latent vector before the decoder and past trajectories (including the corresponding meta-data) to fine-tune the final trajectories by means of a regression and orientation loss.

On top of that, in this particular application in the SMARTS simulator, we take the most plausible future trajectory for each agent (both the adversaries and the ego-vehicle) in the following timestamps: *t=0*, *t=10*, *t=20* and *t=30*, which correspond to the current position and the predicted position of the corresponding agent 1, 2 and 3 seconds in the future respectively. Even though we train our prediction model following the principles of Argoverse 2 (5s and 6s of observation and prediction respectively), given the velocities and traffic density of the experiments run in the SMARTS simulator , we believe that predicting 3s in the future is enough for this purpose to evaluate the high-level actions of decision-making layer preventing over-fitting. In future works, we will design more difficult scenarios, up-to-pair with the Argoverse 2 dataset (specially in terms of intersections or lane change behaviours at high speed) where multi-modal predictions with higher prediction horizons will be required.

8.2.1.2. Reinforcement Learning-based Decision Making

A Markov Decision Process (MDP) is a discrete-time stochastic control process that provides a mathematical framework for modelling decision-making environments. An MDP is a tuple (S, A, P, R) in which S is a set of states named state space, A is a set of actions named action space, P is the probability function and R is a reward function. An algorithm with a given state $s \in S$ takes an action $a \in A$ transitioning to s' with a probability $P(s, a, s')$, and getting a reward $R(s, a, s')$ as shown in Figure 8.2. This algorithm iterates through this loop to learn a desired behaviour.

The goal in an MDP is to find a good policy for the decision-making system. The objective is to find the optimal policy $\pi * (s)$, that maximizes the cumulative function of the future reward.

We represent the driving scenario as an MDP to develop our decision-making module. We consider the output of the MP module as an input to this module. The state space, action space, and reward functions are defined in this section.

8.2.1.2.1. State space The state is defined by the predicted trajectories of the ego-vehicle and the five closest vehicles in the scenario.

$$s_t = (K_t^{ego}, K_t^1, ..., K_t^5) \tag{8.2}$$

where $K_t^i = (x_{t_0}^i, y_{t_0}^i, x_{t_1}^i, y_{t_1}^i, x_{t_2}^i, y_{t_2}^i, x_{t_3}^i, y_{t_3}^i)$ contains the future estimations of the positions of the vehicles across a future horizon of three seconds. A representation of a state vector is shown in Fig 8.4, where the vehicles' predicted positions are represented.

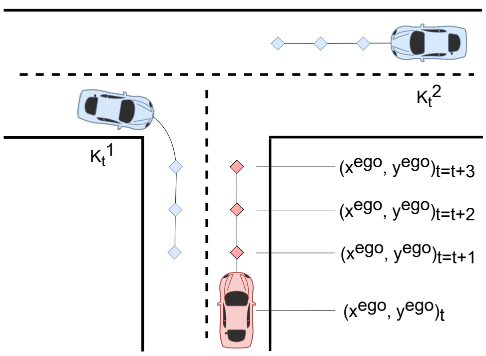


Figure 8.4: Predicted positions of the **ego-vehicle** and the **adversaries** in the next 3s.

8.2.1.2.2. Action space We propose a discrete action space formed by two actions. A low-level controller implemented by the simulator is in charge of performing smooth driving based on these actions. These actions are focused on the ego-vehicle velocity. The first action aims to reach a desired predefined velocity and the second action reduces the velocity until the vehicle stops. The action space is defined as:

$$a = (Drive, Stop) \tag{8.3}$$

8.2.1.2.3. Reward function The reward function is defined in terms of success or failure. A negative reward is given when there is a collision and a positive reward is given when the vehicle reaches the success point, situated at the end of the scenario.

$$r = k_v * v_{ego} + \begin{cases} 1 & if \quad success \\ -1 & if \quad collision \end{cases} \tag{8.4}$$

As shown in Equation 8.4, we add one more factor to the reward function to encourage the ego-vehicle to move. We propose a cumulative reward based on its longitudinal velocity. We use a constant small enough to ensure that the reward per episode is bounded between -1 and 1.

Our approach for the RL implementation (Figure 8.5) builds upon our previous research [152], where we demonstrated that incorporating a feature extractor module to a PPO algorithm yields improved metrics and faster convergence.

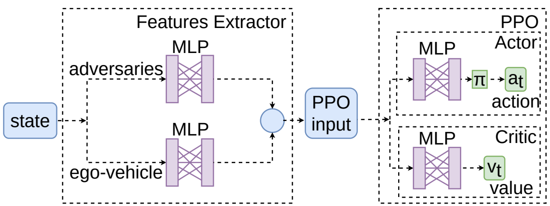


Figure 8.5: Overview of our Reinforcement Learning-based Decision-Making architecture. The neural network architecture consists of two fully connected layers followed by the concatenation of both adversaries and ego vehicle features. The resulting concatenated features are then passed through an actor-critic structure, which comprises two layers, each containing 128 neurons.

In this implementation, we introduce separate feature extractors for adversaries and the ego-vehicle, which are then concatenated into the input for the PPO algorithm. This algorithm consists of two models: the Actor, responsible for selecting an action based on the policy, and the Critic, which estimates the value function.

8.2.2. Experimental results

8.2.2.1. Driving scenario

To validate the performance of our approach, an intersection scenario is implemented in SMARTS, which is a SUMO [135] based simulation platform for research on autonomous driving.

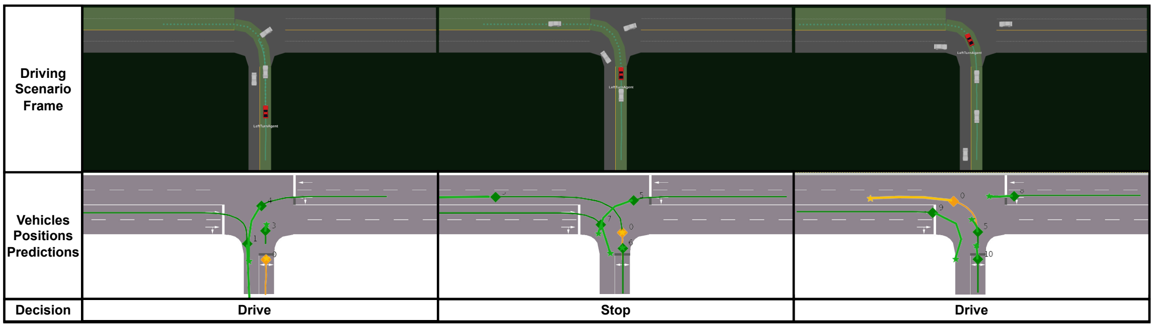


Figure 8.6: Simulation overview of our system behaviour. The red car follows the green path, and each image represents a different frame of the simulation. We also show the predicted positions below each image and the actions taken by the decision-making module.

The scenario is an urban unsignalized T-intersection. The objective is to execute a left turn maneuver in the absence of traffic signal protection, allowing the continuous flow of traffic. Fig. 8.1 illustrates the drivable area (highlighted in green) where the ego-vehicle can navigate to reach the target location. Simulations are reset under three conditions: 1) the ego-vehicle successfully reaches the target, 2) the episodic step surpasses the maximum time steps limit, and 3) the ego-vehicle collides or deviates from the drivable route.

We define different scenario configurations to test the performance of the proposed framework. First, the regular T-intersection scenario which is defined in SMARTS, where a random number of vehicles between [5-10] are spawned every minute, and the maximum velocity of these vehicles is 14 km/h. Then, we propose different configurations increasing the maximum velocity of the adversaries to 30, 60, and 90 km/h.

8.2.2.2. Evaluation Metrics

In decision-making, the success rate serves as a direct measure of the effectiveness of the RL agent in accomplishing the designated task. Besides, the average time of the episode is a common metric used in the literature. These metrics are defined as:

- $success\ [\%] = n_{success}/n_{episodes}$
- $t_e[s] = \sum t_n/n_{episodes}$

where the number of episodes n_e is 100 and simulation time is measured in seconds.

8.2.2.3. Results

To evaluate the performance of our approach we first present a comparison with the existing methods for decision-making in the literature and then an ablation study is conducted.

This first study compares the proposed approach with other existing methods for decision-making. The baseline methods used for comparison are Data-regularized Q-learning (DrQ) [148], Soft Actor-Critic (SAC) [153], and PPO. These methods have different features and serve as reference points for evaluating the proposed approach. The results presented in Table 8.1 demonstrate a higher success rate of our proposal.

Table 8.1: A comparison of the proposed framework against the existing baselines in the T-intersection scenario. The success rate S[%] and the average episode time t_e are presented.

Metric	Ours	PPO	SAC	DrQ
S[%]	**80**	70	68	78
t_e(s)	22.3	36.4	19.2	18.2

Two ablative studies are carried out to see how the use of motion prediction in the state representation can improve the performance of the framework. The first approach is to use just the position of the vehicles as the input to the decision-making module and the second approach is to use the locations over the past five seconds. We test the three approaches under the previously introduced configurations with different adversaries' velocities, from 15km/h to 90km/h. To correctly evaluate the performance of the decision-making system we propose different metrics that aim to provide a better comprehension of the behaviour. We believe that the success rate is still a good indicator, but we slightly modify the average time, only considering the successful episodes to calculate this metric. In addition, we include a new relevant metric: the average ego-vehicle velocity when a collision takes place v_c.

The results presented in Table 8.2 show that the use of the predicted positions in the state vector avoids more collisions as the velocities increase. Besides, the average collision velocity and the average time to complete the scenario are lower for the proposed approach.

Finally, an overview of the behaviour of our system is shown in Figure 8.6. The ego-vehicle in red follows the trajectory defined in green. Each image represents a different frame of the simulation and the respective predictions of the positions are displayed below. Besides, the action executed by the decision-making module for each frame is shown.

8.3. Domain Adaptation in CARLA simulator

In this Section we study the domain adaptation of our efficient social-based prediction model (Figure 8.3) in the CARLA simulator. As stated in previous sections, CARLA

Table 8.2: An ablation study comparing three state representations with different scenario configurations: Current positions, Past positions, and Future positions. The success rate S[%], the episode time t_e in these successful episodes, and the average velocity of collision v_c are presented.

	Metric	15 km/h	30 km/h	60 km/h	90 km/h
Future	S [%]	80	78	78	77
	t_e (s)	22.3	23.3	23.4	23.3
	v_c (km/h)	4.9	5.1	5.6	5.6
Current	S [%]	77	73	70	70
	t_e (s)	25.1	23.4	23.4	23.3
	v_c (km/h)	5.1	5.5	6.1	6.2
Past	S [%]	78	75	72	71
	t_e (s)	24.2	23.3	23.2	23.1
	v_c (km/h)	4.9	5.2	5.9	6.0

provides a realistic sensor simulation environment, including cameras, LIDAR, and radar sensors. On top of that, it accurately models vehicle dynamics, including realistic acceleration, braking, and steering behaviors in complex traffic scenarios with various types of vehicles, pedestrians, and cyclists, enabling motion prediction algorithms to be tested and evaluated in realistic traffic situations, such as sudden imminent collisions with another vehicle, unexpected VRU, intersections or lane change maneuver in a high-way, where the ego-vehicle must pay attention to the surrounding scene and predict its future to take the optimal action.

To this end, we take the ADS provided by the RobeSafe research group (including the control and global planning modules) to move the vehicle around the city given a pre-defined route. In this case, the global routes are pre-defined by the CARLA Autonomous Driving Leaderboard, one of the main competitions around the world to evaluate ADS proposals, either end-to-end or modular-based. Since our MP pipeline require a set of obs_{len} observations per agent, *i.e.* the agent has been previously detected and tracked in such a way a buffer is filled with its past observations, a robust and reliable detection and tracking stage should be implemented to perform 360 °sensor fusion and monitor the most relevant obstacles around the ego-vehicle. Nevertheless, implementing the whole perception pipeline is out of the scope of the thesis, which is mainly focused on the prediction stage.

In that sense, as observed in Figure to validate our prediction pipeline in CARLA in real-time simulation (not in isolated traffic scenarios, as expected from a dataset), we make use of the Autonomous Driving Perception Development Kit (also referred as AD-PerDevKit), partially published in the following conference paper [136]: "Ad perdevkit: an autonomous driving perception development kit using carla simulator and ros", 2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC), p. 4095-4100.

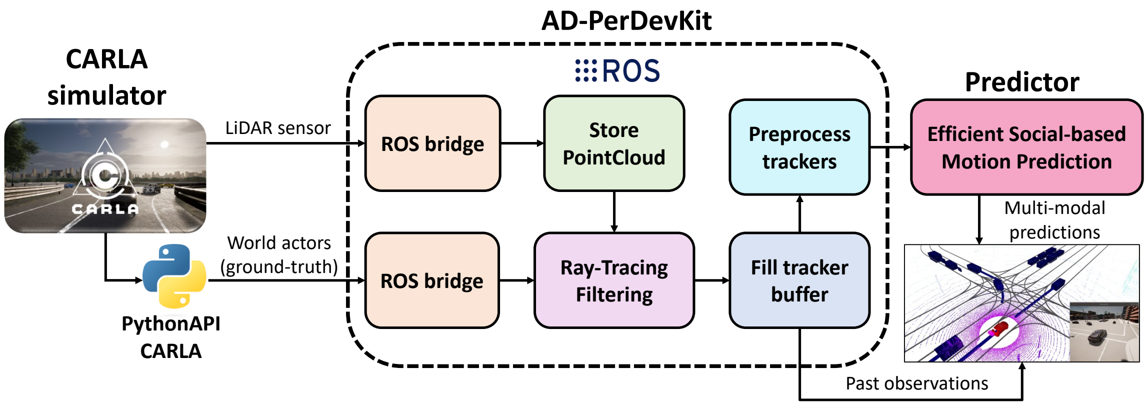


Figure 8.7: Overview of the integration of our prediction pipeline in the CARLA simulator using the AD-PerDevKit tool. The PythonAPI module is used to obtain all actors in the world. Our AD-PerDevKit performs a ray-tracing-based filtering to keep only those obstacles which are hit by at least one LiDAR ray. Past observations are considered over time for these relevant obstacles which are used to feed the social-based prediction pipeline.

8.3.1. Autonomous Driving Perception Development Kit (AD-PerDevKit)

As stated above, most SOTA MP pipelines in the field of AD require 360° DAMOT to get the past observations of the agents and then predict their future actions. Since this thesis is not focused neither in object detection nor tracking, which would involve significant complexity and effort, surpassing the intended focus of this research, utilizing ray-tracing-based ground-truth as a realistic representation of the environment is a justifiable decision.

Autonomous driving systems rely on accurate and reliable input data to make informed decisions. Ray-tracing techniques, which simulate the behavior of light in a virtual environment, provide a highly realistic representation of the surroundings. By employing ray-tracing-based ground-truth, precise information about the positions, shapes, and properties of obstacles in the environment can be generated as preliminary information before conducting the prediction stage. This approach ensures that the ADS receives high-fidelity data that closely resembles real-world conditions.

Moreover, the offline nature of the ground-truth generation process allows for flexibility in experimentation, debugging, and analysis, which can significantly benefit the development and evaluation of the prediction algorithm, for example, by storing the ground-truth of a whole scenario as a text file, do the same with the output predictions, and finally obtain the same metrics that the original dataset where the model was trained to check the domain adaptation (that is, if the model works fine or not without fine-tuning with the current environment data).

The main contribution of the AD-PerDevKit is the creation of a ground-truth generation tool for the surrounding obstacles of the ego-vehicle using CARLA and ROS. For its implementation, the CARLA-ROS brige is used so that the simultaneous execution of CARLA and this tool is not necessary, since the execution of both programs can be very demanding due to the simulator requirements. This way it is possible to record a rosbag (a file with all the ROS messages) with the GT information, so that only an area around the ego-vehicle is analyzed and not the whole obstacles in the CARLA runtime.

The messages created by CARLA contain the information of the different objects of the environment in relation to the map on which it is being used. However, to be used independently, the obstacles must be referenced to the ego-vehicle. Therefore, it is necessary to perform the different transformations to go from a coordinate system based on the map to a coordinate system based on the ego-vehicle.

8.3.1.1. Ray-Tracking-based Filtering

One main issue for the Ground-Truth (GT) calculation is that CARLA always render all the objects, even when they are not visible by the camera, LiDAR or Radar, which is a problem for any training or validation process. In other words, it is necessary to calculate the visibility of the objects from the vehicle, since, otherwise, when proceeding with the evaluation, the precision of the models would be reduced due to not being able to detect some occluded or invisible objects. As aforementioned, to solve that calculation of object visibility (*i.e.* an object that could be preliminarily observed by a SOTA sensor fusion module), we make use of the ray-tracing paradigm.

There are many studies about ray tracing that solve this issue. These techniques [154], [155] are computationally very expensive so it is very difficult to implement them in real-time. In that sense, we propose a method using directly the point-cloud calculated by the simulator. Regarding this, a vehicle will be considered as visible, as long as a point of the LiDAR point-cloud is found inside an object, in the same way as it is done in the nuScenes dataset [55].

Moreover, it is important to note that this tool is designed to work either for offline or online GT generation. On the other hand, one of the most important and delicate parts for the real-time use of this application is efficiency, so it is necessary to perform a method similar to the SOTA in terms of ray-tracing, but with low computational cost. Therefore, it was decided to use the latest point-cloud processed by our Robot Operating System (ROS) bridge from ROS to filter the objects in the environment given a frequency of 10 Hz (standard in the AD industry, and particularly in the CARLA Autonomous Driving Leaderboard). For the implementation of this operation, vectorization of the operations is necessary, as computation needs to be done in less than 10^{-2} seconds. It must be noted that only vehicles and walkers are considered for our purposes, since CARLA also provides the traffic lights and other traffic infrastructure as World agents or actors. The steps to be performed are the summarized in Algorithm 4, where a maximum distance of 120 m is considered to filter the furthest agents:

Algorithm 4: Ray-tracing-based filtering algorithm to perform object visibility

Input : Point cloud provided by the simulator and CARLA world agents

Output: Bounding box parameters of agents with at least one LiDAR point of incidence

1. Transform LiDAR raw data and CARLA World objects into ROS format to enhance matrix operations and interpretability.
2. Remove objects furthest than the maximum LiDAR distance.
3. Delete points in the point cloud with heights higher or lower than the objects in the surroundings.
4. Eliminate points outside the area where the objects are located, considering all possible rotations.

Function $f_{\text{visible\_bb}}(\text{bb}, \text{points})$:;

return

$\text{np.logical\_and}\Big(\text{np.logical\_and}\Big(\text{bb}[0] - \frac{\text{bb}[3]}{2} \leq \text{np.array}(\text{points}[:, 0]), \text{np.array}(\text{points}[:, 0]) \leq \text{bb}$
$\frac{\text{bb}[5]}{2} \leq \text{np.array}(\text{points}[:, 2]) \leq \text{bb}[2] + \frac{\text{bb}[5]}{2}$;

5. Select visible objects having at least one point in the point cloud, considering the rotation of different objects;

$n_{\text{points\_in\_bb}} = 0$;

if *self.pointcloud is not None* **then**

$\quad$ $\textit{points\_in\_bb} =$ $f_{\text{visible\_bb}}\left((\text{obj.position\_x}, \text{obj.position\_y}, \text{obj.position\_z}, \text{obj.l}, \text{obj.w}, \text{obj.h}), \text{self.pointcloud}\right)$;

$\quad$ $n_{\text{points\_in\_bb}} = \text{np.add.reduce}(\textit{points\_in\_bb})$;

end

8.3.2. Experimental results

Once we have calculated the 360 °visible objects in a frame t, as additional post-processing steps the Autonomous Driving Perception Development Kit (AD-PerDevKit) tool fills a buffer with the past positions of the corresponding object (simulating a real-world buffered Multi-Object Tracker). Nevertheless, since the proposed prediction model has been trained in Argoverse 2 using an observation length of $obs_{len} = 50$ (*i.e.* 5s of motion history regarding a frequency of 10 Hz). Then, calculating the predictions with only 2 or 3 observations would not make sense, since even though there are several agents in Argoverse 1 and Argoverse 2 with only a few observations (for example, an object is relevant in the scene since it is relevant to the ADS in $t = 0$, with only has 4 observations because suddenly appeared in the traffic scenario). To this end, we filter those obstacles with a number of observations less than a certain threshold, set to 20 in this case. Figure 8.8 depicts an example of the AD-PerDevKit output while driving our ADS in the CARLA simulator (particularly Town03). Filtered objects and observations are plotted in the well-established RVIZ simulator.

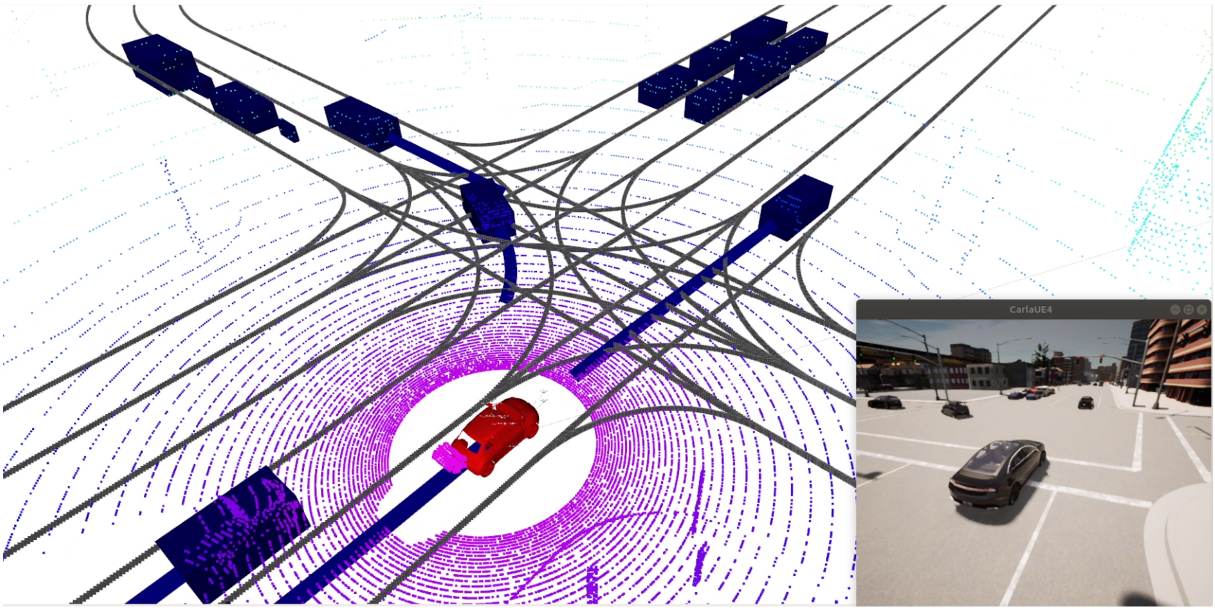


Figure 8.8: Ray-tracing-based filtering example in the RVIZ simulator

The experiments were run in a computer consisting of an Intel i7-9700k (overclocked to 4.9 GHz), NVIDIA 3090 RTX Ti and 32 GB of RAM (3200 MHz). To illustrate the importance of the prediction, we have selected several interesting use cases of the CARLA Autonomous Driving Leadeboard inspired in the National Highway Traffic Safety Administration (NHTSA) where either the prediction module is of vital importance to take the optimal action in the short/long term, or scenarios where the model is able to reason in an intelligent way, even though map information is not included for simplicity, which agents should keep their position in future frames (*e.g.* agents stopped in front of a red traffic light or stop) and which agents, even though they are currently stopped, must be predicted since ahead vehicles have started moving. Dynamic agents, as expected, are predicted in all different scenarios.

For each figure, we illustrate the **past observations** (until $obs_{len} = 50$) of the relevant agents, as well as the **ego-vehicle** observations, taking into account that the model will be able to predict only if the ego-vehicle has at least two observations, since to compute the rotation angle the current and past observations must be considered. Moreover, **multimodal predictions** are calculated for each agent (including the ego-vehicle, treated as a standard agent, not taking into account the CMP paradigm as explained in Chapter 2). Note that for visualization purposes, we avoid plotting those modes with a confidence value lower than a certain threshold, in this case set to 0.2. Note that in CARLA our ego-vehicle is a gray Lincoln MKZ 2017, while in RVIZ, for visualization purposes, it is the red vehicle.

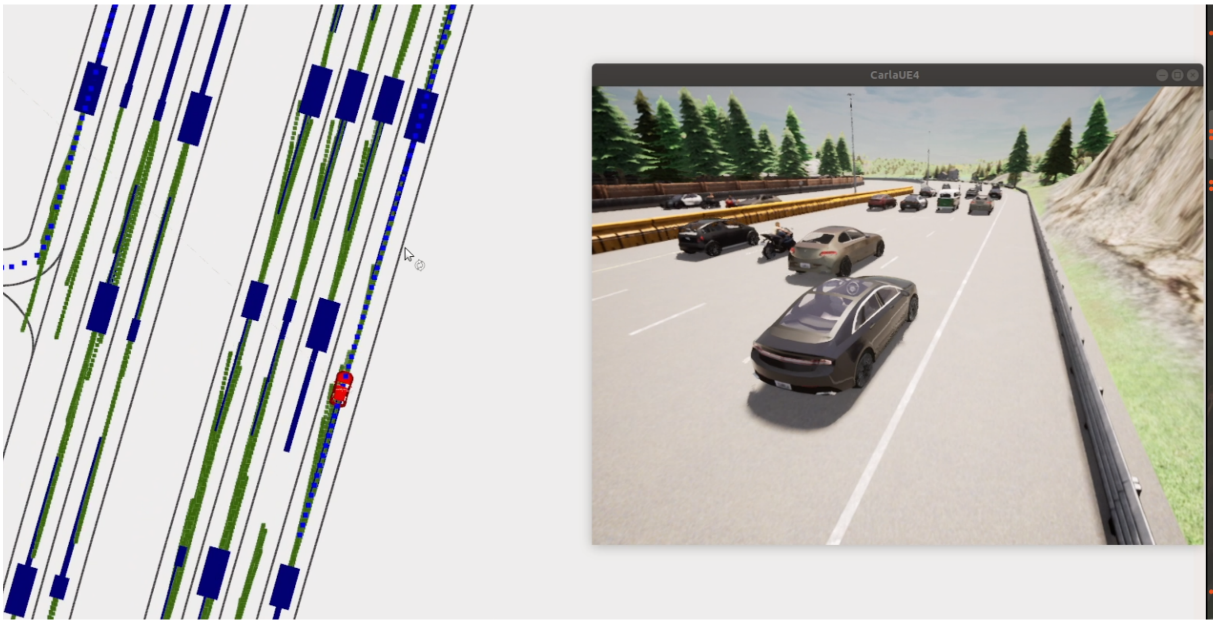
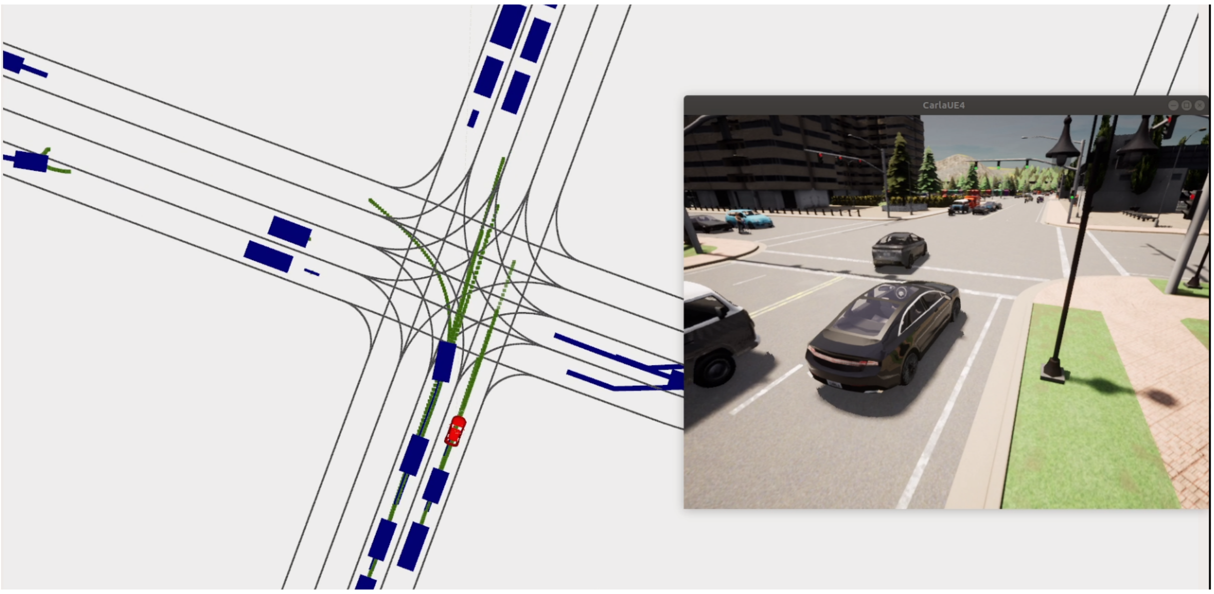


Figure 8.9: Crowded high-way use case

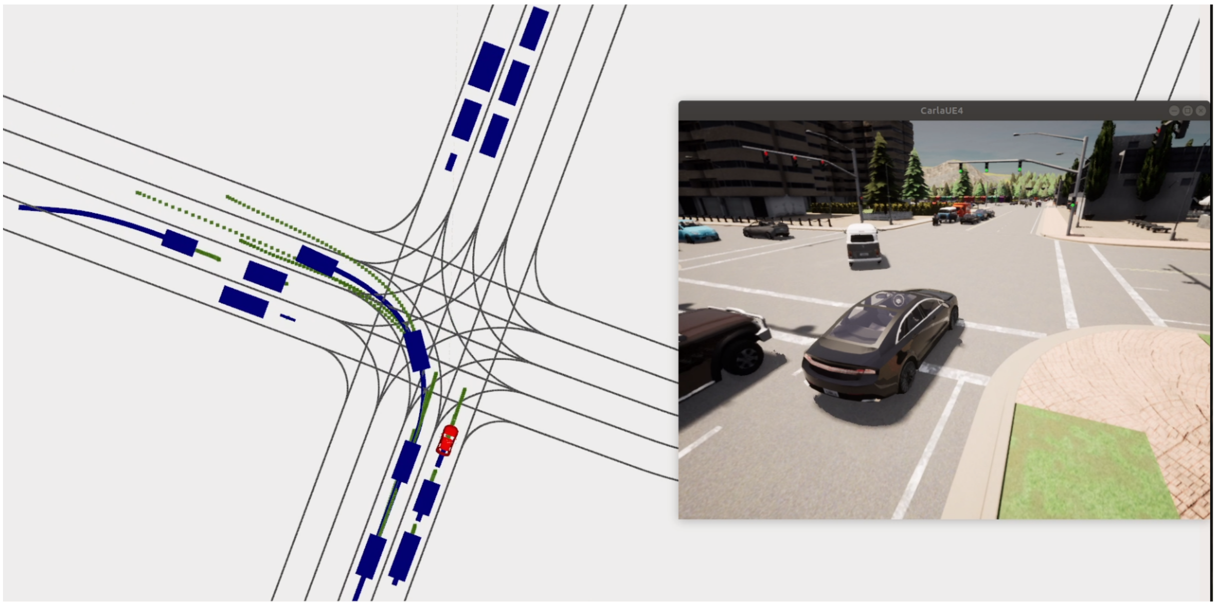
Figure 8.9 illustrates a traffic jam in crowded highway, for example a standard situation when approaching a metropolis in the morning at work time. In this case, we may observe that the model does not compute multi-modal predictions towards different directions, but, as expected, it understands that all vehicles are driving in the same direction (either forward or reverse way) and the multi-modal prediction is computed in terms of different velocity profiles (that is, the agent can continue with the same velocity, suddenly break or accelerate in the mid/long term).

Figure 8.10 shows an intersection where the Toyota Prius in front of the ego-vehicle can conduct a multi-modal prediction (either turn left or keep straight), understanding that it is in an intersection given the surrounding agents position and past observations. Then, after turning left, 2s in the future, the model does not predict different directions, but only keep straights assuming there is not another intersection close to him ahead.

As commented in Chapter 4, predicting the behaviour of VRUs, such as cyclists or pedestrians, is one of the most critical aspect of an ADS to be deployed and scaled to real-world applications in urban scenarios. In this particular case (Figure 8.11), we may observe that a kid suddenly appears behind a CocaCola vending machine, and, after taking into account the minimum number of observations, the model is able to correctly predict the future intentions of the agent. Nevertheless, an interesting point of view for future works could be how to assign the minimum number of observations depending on the agent type or potential risks associated to the traffic situation. For example, in this particular case, even though the model has been trained with way more past observations, the model should predict the agent as fast as possible, even though several modes are computed in non-plausible or non-sense directions.



(a) Intersection at frame t



(b) Intersection at frame $t + 20$

Figure 8.10: Intersection use case

Figure 8.12 illustrates a standard Stop traffic scenario, where the model is clearly able to distinguish among dynamic and static objects, predicting the trajectories in the same point for agents stopped for a while. Moreover, it can be slightly appreciated a multi-modal prediction for the white and red van, but probably the model discards the possibility of turning right given the vehicle that is present in that lane in the current timestamp. Conducting transfer learning and fine-tuning the model with CARLA scenarios could be an option to help the models understand the situation in an enhanced way.

Finally, Figure 8.13 shows one of the most challenging traffic situations in urban scenarios, that is, a roundabout full of agents, considering incoming, outcoming and present

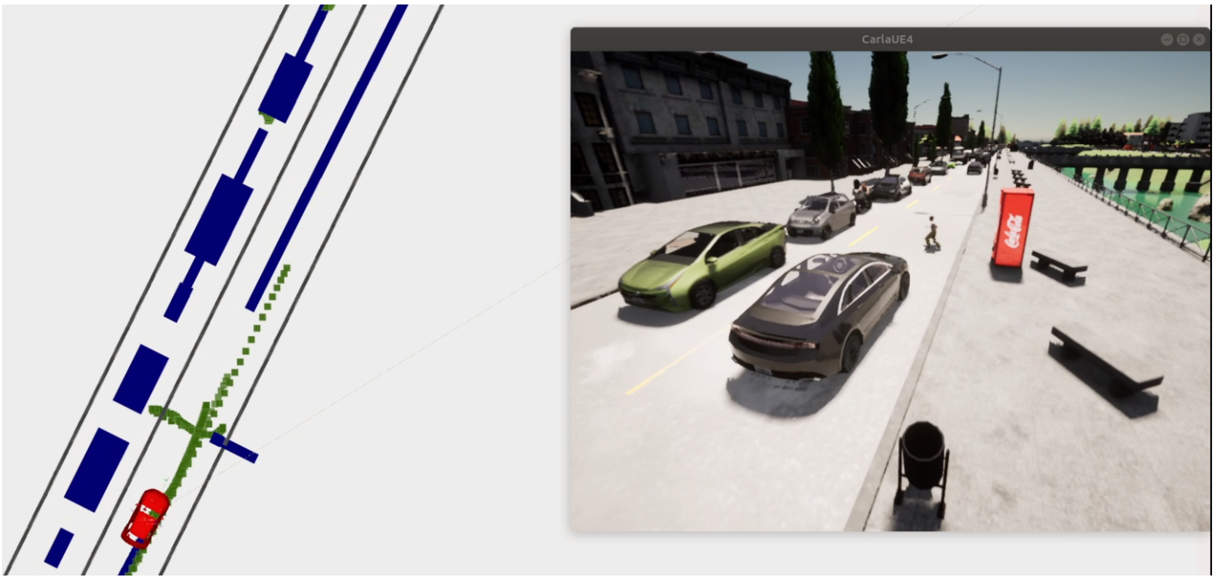


Figure 8.11: Unexpected Vulnerable Road User (VRU) use case

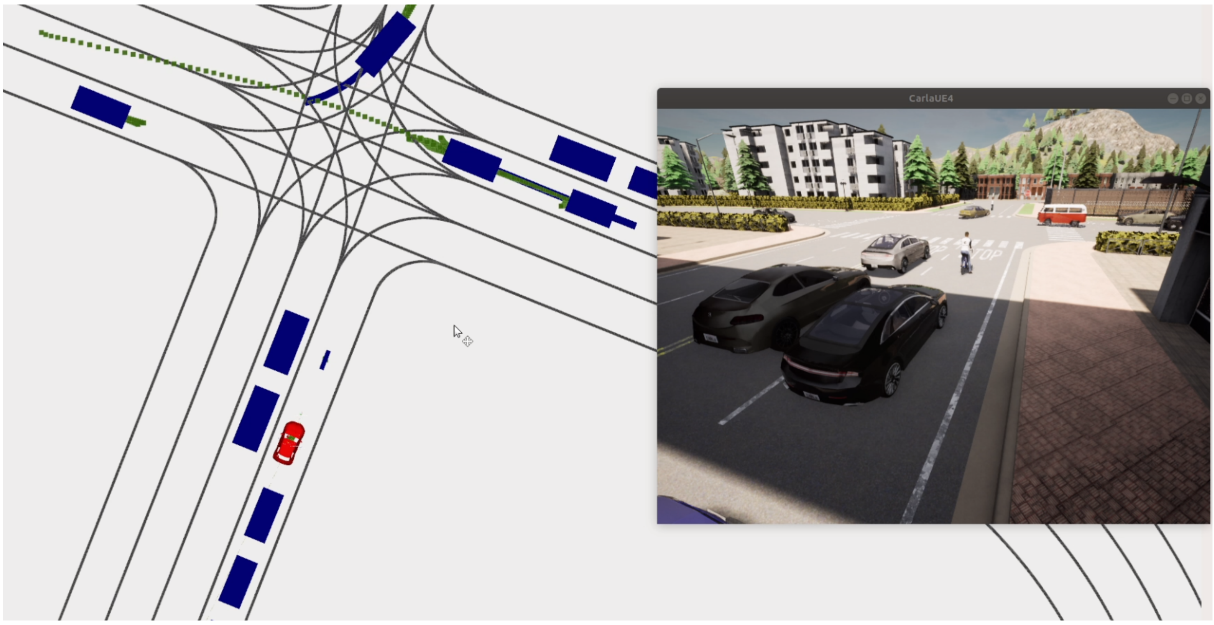


Figure 8.12: Stop use case

agents in the current time-stamp. To help the behavioural and local planning modules to take the optimal action, the model clearly predicts the circular motion of the agents, either for the agents that are in the roundabout with different velocity profiles or even for the incoming agents which are going to get into the roundabout and probably intersect with the trajectory of our ego-vehicle if the adversary does not respect the give-way traffic signal. Then, understanding this complex situation and taking account all different scenarios, from least dangerous to most optimistic, is the key to build a robust and reliable architecture.

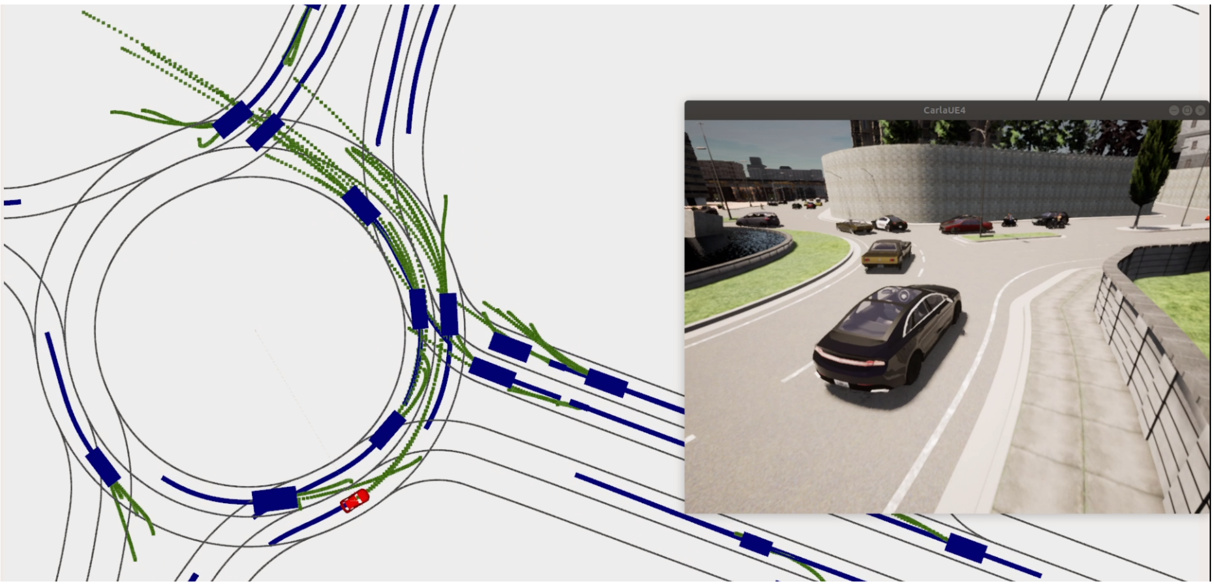


Figure 8.13: Roundabout use case

8.4. Summary

In this Chapter we have evaluated our efficient social-based prediction model, where the incorporation of map information has been avoided for simplicity purposes given the model has been evaluated in two different frameworks, the SMARTS and CARLA simulator, with a graph map structure way different to Argoverse 2.

In terms of the decision-making application, the prediction module must calculate the future positions of vehicles within the scenario to improve a Reinforcement Learning-based Decision Making module. The results of the study demonstrate that our approach achieves significant performance improvements, particularly in scenarios involving high velocities.

On the other hand, we have successfully integrated the prediction algorithm in our ADS in simulation using CARLA and studied the domain adaptation of the algorithm from a static dataset (Argoverse 2) to an hyper-realistic environment, using some scenarios inspired in the well-established NHTSA typology. Given the amount of work and complexity required to develop an efficient and accurate 360 °DAMOT pipeline, we make use of the AD-PerDevKit pipeline to generate a realistic GT (avoiding to store all agents of the simulator, which would not make sense) given a ray-tracing-based filtering algorithm and some post-processing steps to obtain a buffer of past observations for each agent, removing those agents with a number of observations lower than a certain threshold. As observed, the model is able to compute multi-modal predictions in terms of different directions and velocity profiles, or even predicting the trajectories in the same point for agents stopped for a while, understanding the past motion and complex interactions among the agents and corresponding neighbours.

Chapter 9

Conclusions and Future Works

A más ver, mis valientes hobbits.
Mi labor ha concluido. Aquí, al fin, a la orilla del mar,
llega el adiós a nuestra Compañía.
No os diré no lloréis,
pues no todas las lágrimas son amargas.

Discurso de despedida de Gandalf
El Señor de los Anillos: El Retorno del Rey

9.1. Conclusions

In this thesis, a series of interaction-aware motion prediction methods for scene understanding in the field of Autonomous Driving, covering both the single-agent motion prediction and multi-agent motion prediction use cases from the uni-modal and multi-modal perspectives focusing on long-term (from 3 to 6 s) prediction horizon, including social and physical information.

In Chapter 2 we review the contextual factors and the classification of the most important physics-based and Deep Learning (DL)-based methods for Motion Prediction (MP) in the Autonomous Driving (AD) field. Moreover, we study the literature of the State-of-the-Art (SOTA) databases and simulators to validate the algorithms.

In Chapter 3, we study the theoretical background to deeply understand the proposed methods and their validation in the remaining Chapters.

Once the technical background and related works are presented, in Chapter 4 we introduce a simple-yet-powerful tracking-by-detection pipeline, based on traditional techniques such as Kalman Filter (KF) for state estimation, Hungarian Algorithm (HA) for data association and map-based filtering, that computes the trackers over time in the traffic scenario to feed the subsequent predictions. We conclude here that filtering non-relevant objects by means of the monitored area (most relevant lanes around the Autonomous Driving Stacks (ADSs)) can reduce the inference time and computational complexity of the overall pipeline.

Then, the thesis focuses on Deep Learning (DL)-based MP in Chapter 5 and 6, where we validate our algorithms in the Argoverse 1 Motion Forecasting dataset and Chapter 7, where we move to the Argoverse 2 Motion Forecasting, the most challenging and recent vehicle MP dataset in the literature, progressively introducing State-of-the-Art (SOTA) mechanisms to encode the complex traffic scenarios, agents interactions and the most representative features of the map information.

Chapter 5 proposes a conditional Generative Adversarial Network (cGAN) prediction algorithm to compute uni-modal future trajectories which uses as generator a Long Short-Term Memory (LSTM) based encoder-decoder with Multi-Head Self-Attention (MHSA), where the concatenation of the past motion, encoded by a LSTM, and plausible target points, encoded by a Multilayer Perceptron (MLP), represents the condition to the network.

Chapter 6 proposes several efficient baselines, introducing the use of Graph Neural Networks (GNNs) to powerfully encode agent interactions and preprocess preliminary trajectories from the map using a heuristic method to obtain simple-yet-useful centerlines with only geometric information to avoid full HD map preprocessing, that serve as preliminary trajectories. We realize that using encoder transformers instead of recurrent networks or standard MLP presents some benefits, since, as stated in Chapter 6, despite having more parameters, recurrent neural networks such as Long Short-Term Memory (LSTM) or Gated Recurrent Unit (GRU) are non-parallelizable due to their sequential structure so the overall training and hence time is higher than transformers-based approaches.

Taking these previous baselines into consideration, the final model of the thesis improves upon the previous ones by incorporating enhancements in the heuristic method, including topological and semantic information of interest about lanes (such as lane boundaries information, presence of intersection or lane mark type), Deep Learning (DL)-based map encoding by means a Graph Convolutional Network (GCN), fusion-cycle of physical and social features, deep learning-based estimation of final positions on the road, aggregation of the surrounding environment, and refinement of predictions to enhance the quality of the final multi-modal predictions in an elegant and efficient manner.

In all our models, specially in the last one, we can appreciate how they generalize well in unknown traffic scenarios, with up-to-pair regressions metrics (both in uni-modal and multi-modal prediction) while noticeably reducing the inference time and number of parameters compared to other SOTA algorithms.

Then, the final proposal is validated in several applications in Chapter 8, taking into account only its social version, due to the difficulty of adapting the map preprocessing and format in different datasets or simulators), such as decision-making or holistic integration in a hyper-realistic simulator with other vehicle layers, as a preliminary step towards its implementation in an actual autonomous vehicle. Particularly, in our different tests, run in the SMARTS and CARLA simulators, we successfully realize how the overall pipeline can leverage the predictions, specially in challenging situations, to avoid a collision or perform smoother trajectories in complex scenarios where only detecting and tracking the objects over time would not be enough to understand the surrounding scenario and compute the optimal decision.

We hope that our proposals can serve as a solid baseline on which others can build on to advance the state-of-the-art in fusing perception data and map information to perform real-time motion prediction and decision-making evaluation in arbitrarily complex urban scenarios.

9.2. Future Works

The field of vehicle Motion Prediction for Autonomous Driving is continuously evolving, and several promising future directions can be anticipated. These potential advancements aim to enhance the accuracy, reliability, and safety of motion prediction algorithms, ultimately leading to more efficient and robust Autonomous Driving Systems. Although significant research has been conducted in trajectory prediction and path planning in recent years, there are still numerous aspects that require further investigation in the future:

- **Enhanced Scene Representation and Encoding**: Various methods have been proposed to encode driving scenes using different representations, such as graphs or rasterized top-view maps. However, the absence of a unified representation hampers the generalization of prediction methods for large-scale deployment in real-world autonomous vehicles. Graph-based representations show promise in accommodating heterogeneous objects and their inter-dependencies through directed edges. To enhance graph-based scene representation and encoding, three important steps can be pursued and improved in future works: constructing the graph with proper connections, assigning node and edge features appropriately, and designing graph operators to handle scene graph heterogeneity, leveraging advancements in heterogeneous graph neural networks.
- **Probabilistic Modeling**: Incorporating probabilistic methods into vehicle motion prediction algorithms can help quantify uncertainty and improve the reliability of predictions. By estimating probability distributions over future trajectories, autonomous systems can make more informed decisions, taking into account the likelihood of different outcomes. Probabilistic modeling also enables better risk assessment and planning in uncertain and dynamic traffic scenarios.
- **Predictive Planning**: The integration of MP into the planning module (stated as Conditional Behaviour Prediction in Chapter 2) is crucial for improving decision-

making and motion control in ADS. Predictive planning is a worthwhile avenue to explore, addressing challenges such as handling prediction uncertainty, studying the relationship between prediction and planning, and designing scalable predictive planners for predictors with different known uncertainties. Additionally, investigating learning-based motion planners that can be combined with data-driven predictions holds great potential, although the current limitations in explainability and reliability require attention.

- **Interpretability and explainability** are critical aspects of vehicle motion prediction in autonomous driving that require further exploration in future works. While the accuracy and performance of prediction models have improved significantly, understanding and explaining the reasoning behind their predictions remain challenging. Several directions for future research in explainability and interpretability are: Model transparency and visualization (Developing methods to make prediction models more transparent and interpretable is crucial), Rule-based models (Investigating rule-based approaches can provide interpretable predictions in vehicle motion prediction), Feature importance analysis (Conducting feature importance analysis can help identify the most influential factors in vehicle motion prediction), Context-aware explanations (Future research should aim to develop context-aware explanation methods that not only provide predictions but also explain the reasoning behind them in the context of the surrounding environment), Uncertainty estimation and trust assessment (Future works should focus on developing techniques to estimate and communicate prediction uncertainties effectively) and Human-machine interaction (To enhance user trust and acceptance, future research should explore methods for effective human-machine interaction in vehicle motion prediction).
- **Transfer Learning**: Transferring knowledge from one driving scenario to another can significantly improve the efficiency of vehicle motion prediction. By training models on diverse datasets and environments, and then fine-tuning them for specific scenarios, researchers can reduce the reliance on large amounts of scenario-specific training data. Transfer learning allows predictions to generalize across different driving conditions, leading to more robust and adaptable autonomous systems.
- **Human-Centric Approaches**: Understanding human intentions and incorporating social norms in motion prediction is crucial for ensuring safe and harmonious interactions between autonomous vehicles and human drivers or pedestrians. Future works may involve the development of algorithms that can interpret and predict human behavior accurately. This can be achieved by leveraging techniques from computer vision, natural language processing, and social sciences to capture and model human intentions, gestures, and communication cues.
- **Knowledge-Distillation** in vehicle motion prediction is an emerging area that holds promise for reducing reliance on map information and enhancing the generalizabil-

ity of prediction models. By leveraging the knowledge acquired from more complex models or human experts, knowledge distillation enables the transfer of valuable insights to smaller and more lightweight models. Here are several potential future directions for employing knowledge distillation in vehicle motion prediction to reduce dependency on map information: Model compression for map-free prediction (Knowledge-Distillation can be applied to compress large-scale map-based prediction models into smaller models that can make accurate predictions without explicit map information), Expert knowledge transfer (Knowledge-Distillation can facilitate the transfer of expert knowledge to prediction models), Reinforcement learning with model distillation (Reinforcement learning techniques combined with knowledge distillation can be employed to train models that can learn to predict vehicle motions without explicit map information), Data augmentation techniques (By synthesizing additional training samples using map-based models, the prediction models can learn from a more diverse range of scenarios, including situations without map information), Self-supervised learning for map-free prediction (Knowledge distillation can be combined with self-supervised learning techniques to train prediction models without relying on explicit map information).

In conclusion, extensive effort is needed in some interesting areas such as scene representation and encoding, probabilistic modeling, predictive planning, interpretability and explainability, transfer learning, human-centric approaches or even knowledge-distillation to further advance the performance (both in terms of regression and efficiency) of MP algorithms in the field AD as a preliminary stage before feeding the subsequent decision-making, local planning and motion control layers in an ADS, contributing to the development of more efficient and reliable autonomous driving systems.

List of Publications

The following publications correspond to first author or main co-author. For a complete view of all publications derived from the thesis: All publications [1]:

Journal Papers

- **Gomez-Huelamo, C.**, Conde, M. V., Barea, R., Ocala, M, & Bergasa, L. M. (2023). Efficient Baselines for Motion Prediction in Autonomous Driving. Transactions on Intelligent Transportation Systems. In submission.
- **Gomez-Huelamo, C.**, Del Egido, J., Bergasa, L. M., Barea, R., Lopez-Guillen, E., Araluce, J., & Antunes, M. (2022). 360º real-time and power-efficient 3D DAMOT for autonomous driving applications. Multimedia Tools and Applications, 81(19), 26915-26940
- **Gómez-Huélamo, C.**, Del Egido, J., Bergasa, L. M., Barea, R., López-Guillén, E., Arango, F., ... & López, J. (2021). Train here, drive there: Ros based end-to-end autonomous-driving pipeline validation in carla simulator using the nhtsa typology. Multimedia Tools and Applications, 1-28

Conference Contributions

- **Gómez-Huélamo, C.**, Conde, M. V., Gutiérrez-Moreno R, Barea, R., Llamazares A., Antunes M., & Bergasa, L. M. (2022, October). Efficient Context-Aware Graph Transformer for Vehicle Motion Predictio. In 2023 IEEE 26th International Conference on Intelligent Transportation Systems (ITSC). IEEE. In submission.
- Gutiérrez-Moreno R., **Gómez-Huélamo, C.**, Barea. R, López-Guillén E., Arango F., Ortiz, M., & Bergasa, L. M. (2023, October). Augmented Reinforcement Learning with Efficient Social-based Motion Prediction for Autonomous Decision-Making. In 2023 IEEE 26th International Conference on Intelligent Transportation Systems (ITSC). IEEE. In submission.

[1] https://scholar.google.es/citations?user=OWwoG6EAAAAJ&hl=es

- **Gómez-Huélamo, C.**, Conde, M, Barea, R. & Bergasa, L. M. (2023, June). Improving Multi-Agent Motion Prediction with Heuristic Goals and Motion Refinement. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 5322-5331). IEEE

- **Gómez-Huélamo, C.**, Conde, M. V., Ortiz, M., Montiel, S., Barea, R., & Bergasa, L. M. (2022, October). Exploring Attention GAN for Vehicle Motion Prediction. In 2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC) (pp. 4011-4016). IEEE

- Diaz-Diaz, A., Ocaña, M., Llamazares, Á., **Gómez-Huélamo, C.**, Revenga, P., & Bergasa, L. M. (2022, June). HD maps: Exploiting OpenDRIVE potential for Path Planning and Map Monitoring. In 2022 IEEE Intelligent Vehicles Symposium (IV) (pp. 1211-1217). IEEE.

- **Gómez-Huélamo, C.**, Diaz-Diaz, A., Araluce, J., Ortiz, M. E., Gutiérrez, R., Arango, F., ... & Bergasa, L. M. (2022, June). How to build and validate a safe and reliable Autonomous Driving stack? A ROS based software modular architecture baseline. In 2022 IEEE Intelligent Vehicles Symposium (IV) (pp. 1282-1289). IEEE

- Del Egido, J., **Gómez-Huélamo, C.**, Bergasa, L. M., Barea, R., López-Guillén, E., Araluce, J., ... & Antunes, M. (2020, November). 360 real-time 3d multi-object detection and tracking for autonomous vehicle navigation. In Advances in Physical Agents II: Proceedings of the 21st International Workshop of Physical Agents (WAF 2020), November 19-20, 2020, Alcalá de Henares, Madrid, Spain (pp. 241-255). Cham: Springer International Publishing.

- **Gómez–Huélamo, C.**, Bergasa, L. M., Gutiérrez, R., Arango, J. F., & Díaz, A. (2021, July). Smartmot: exploiting the fusion of hdmaps and multi-object tracking for real-time scene understanding in intelligent vehicles applications. In 2021 IEEE Intelligent Vehicles Symposium (IV) (pp. 710-715). IEEE.

- Gutiérrez, R., Arango, J. F., **Gomez-Huélamo, C.**, Bergasa, L. M., Barea, R., & Araluce, J. (2021, July). Validation Method of a Self-Driving Architecture for Unexpected Pedestrian Scenario in CARLA Simulator. In 2021 IEEE Intelligent Vehicles Symposium (IV) (pp. 1144-1149). IEEE.

- **Gómez-Huélamo, C.**, Del Egido, J., Bergasa, L. M., Barea, R., López-Guillén, E., Arango, F., ... & López, J. (2021). Train here, drive there: Simulating real-world use cases with fully-autonomous driving architecture in carla simulator. In Advances in Physical Agents II: Proceedings of the 21st International Workshop of Physical Agents (WAF 2020), November 19-20, 2020, Alcalá de Henares, Madrid, Spain (pp. 44-59). Springer International Publishing

- **Gómez-Huélamo, C.**, Del Egido, J., Bergasa, L. M., Barea, R., Ocana, M., Arango, F., & Gutiérrez-Moreno, R. (2020, September). Real-time bird's eye view multi-object tracking system based on fast encoders for object detection. In 2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC) (pp. 1-6). IEEE.
- **Gómez-Huelamo, C.**, Bergasa, L. M., Barea, R., López-Guillén, E., Arango, F., & Sánchez, P. (2019, October). Simulating use cases for the UAH Autonomous Electric Car. In 2019 IEEE Intelligent Transportation Systems Conference (ITSC) (pp. 2305-2311). IEEE.

Bibliography

[1] S. Taxonomy, “Definitions for terms related to driving automation systems for on-road motor vehicles (j3016)”, Technical report, Society for Automotive Engineering, Tech. Rep., 2016.

[2] J. F. Arango, L. M. Bergasa, P. A. Revenga, *et al.*, “Drive-by-wire development process based on ros for an autonomous electric vehicle”, *Sensors*, vol. 20, no. 21, p. 6121, 2020.

[3] A. Ranga, F. Giruzzi, J. Bhanushali, *et al.*, “Vrunet: Multi-task learning model for intent prediction of vulnerable road users”, *arXiv preprint arXiv:2007.05397*, 2020.

[4] I. Gog, S. Kalra, P. Schafhalter, M. A. Wright, J. E. Gonzalez, and I. Stoica, “Pylot: A modular platform for exploring latency-accuracy tradeoffs in autonomous vehicles”, in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2021, pp. 8806–8813.

[5] M.-F. Chang, J. Lambert, P. Sangkloy, *et al.*, “Argoverse: 3d tracking and forecasting with rich maps”, in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 8748–8757.

[6] B. Wilson, W. Qi, T. Agarwal, *et al.*, “Argoverse 2: Next generation datasets for self-driving perception and forecasting”, *arXiv preprint arXiv:2301.00493*, 2023.

[7] T. Salzmann, B. Ivanovic, P. Chakravarty, and M. Pavone, “Trajectron++: Dynamically-feasible trajectory forecasting with heterogeneous data”, in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XVIII 16*, Springer, 2020, pp. 683–700.

[8] T. Gilles, S. Sabatini, D. Tsishkou, B. Stanciulescu, and F. Moutarde, “Home: Heatmap output for future motion estimation”, in *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, IEEE, 2021, pp. 500–507.

[9] T. Gilles, S. Sabatini, D. Tsishkou, B. Stanciulescu, and F. Moutarde, “Gohome: Graph-oriented heatmap output for future motion estimation”, in *2022 International Conference on Robotics and Automation (ICRA)*, IEEE, 2022, pp. 9107–9114.

[10] B. Varadarajan, A. Hefny, A. Srivastava, *et al.*, “Multipath++: Efficient information fusion and trajectory aggregation for behavior prediction”, in *2022 International Conference on Robotics and Automation (ICRA)*, IEEE, 2022, pp. 7814–7821.

[11] M. Wang, X. Zhu, C. Yu, *et al.*, “Ganet: Goal area network for motion forecasting”, *arXiv preprint arXiv:2209.09723*, 2022.

[12] J. Schmidt, J. Jordan, F. Gritschneder, and K. Dietmayer, “Crat-pred: Vehicle trajectory prediction with crystal graph convolutional neural networks and multi-head self-attention”, in *2022 International Conference on Robotics and Automation (ICRA)*, IEEE, 2022, pp. 7799–7805.

[13] M. Liang, B. Yang, R. Hu, *et al.*, "Learning lane graph representations for motion forecasting", in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16*, Springer, 2020, pp. 541–556.

[14] C. Tang and R. R. Salakhutdinov, "Multiple futures prediction", *Advances in neural information processing systems*, vol. 32, 2019.

[15] N. Rhinehart, R. McAllister, K. Kitani, and S. Levine, "Precog: Prediction conditioned on goals in visual multi-agent settings", in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 2821–2830.

[16] S. Khandelwal, W. Qi, J. Singh, A. Hartnett, and D. Ramanan, "What-if motion prediction for autonomous driving", *arXiv preprint arXiv:2008.10587*, 2020.

[17] H. Zhao, J. Gao, T. Lan, *et al.*, "Tnt: Target-driven trajectory prediction", in *Conference on Robot Learning*, PMLR, 2021, pp. 895–904.

[18] Y. Huang, J. Du, Z. Yang, Z. Zhou, L. Zhang, and H. Chen, "A survey on trajectory-prediction methods for autonomous driving", *IEEE Transactions on Intelligent Vehicles*, vol. 7, no. 3, pp. 652–674, 2022.

[19] N. Kaempchen, B. Schiele, and K. Dietmayer, "Situation assessment of an autonomous emergency brake for arbitrary vehicle-to-vehicle collision scenarios", *IEEE Transactions on Intelligent Transportation Systems*, vol. 10, no. 4, pp. 678–687, 2009.

[20] R. Pepy, A. Lambert, and H. Mounier, "Reducing navigation errors by planning with realistic vehicle model", in *2006 IEEE Intelligent Vehicles Symposium*, IEEE, 2006, pp. 300–307.

[21] R. Miller and Q. Huang, "An adaptive peer-to-peer collision warning system", in *Vehicular Technology Conference. IEEE 55th Vehicular Technology Conference. VTC Spring 2002 (Cat. No. 02CH37367)*, IEEE, vol. 1, 2002, pp. 317–321.

[22] J. Hillenbrand, A. M. Spieker, and K. Kroschel, "A multilevel collision mitigation approach—its situation assessment, decision making, and performance tradeoffs", *IEEE Transactions on intelligent transportation systems*, vol. 7, no. 4, pp. 528–540, 2006.

[23] R. E. Kalman, "A new approach to linear filtering and prediction problems", 1960.

[24] N. Kaempchen, K. Weiss, M. Schaefer, and K. C. Dietmayer, "Imm object tracking for high dynamic driving maneuvers", in *IEEE Intelligent Vehicles Symposium, 2004*, IEEE, 2004, pp. 825–830.

[25] B. Jin, B. Jiu, T. Su, H. Liu, and G. Liu, "Switched kalman filter-interacting multiple model algorithm based on optimal autoregressive model for manoeuvring target tracking", *IET Radar, Sonar & Navigation*, vol. 9, no. 2, pp. 199–209, 2015.

[26] V. Lefkopoulos, M. Menner, A. Domahidi, and M. N. Zeilinger, "Interaction-aware motion prediction for autonomous driving: A multiple model kalman filtering scheme", *IEEE Robotics and Automation Letters*, vol. 6, no. 1, pp. 80–87, 2020.

[27] A. Broadhurst, S. Baker, and T. Kanade, "Monte carlo road safety reasoning", in *IEEE Proceedings. Intelligent Vehicles Symposium, 2005.*, IEEE, 2005, pp. 319–324.

[28] K. Okamoto, K. Berntorp, and S. Di Cairano, "Driver intention-based vehicle threat assessment using random forests and particle filtering", *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 13 860–13 865, 2017.

[29] Y. Wang, Z. Liu, Z. Zuo, Z. Li, L. Wang, and X. Luo, "Trajectory planning and safety assessment of autonomous vehicles based on motion prediction and model predictive control", *IEEE Transactions on Vehicular Technology*, vol. 68, no. 9, pp. 8546–8556, 2019.

[30] A. Gupta, J. Johnson, L. Fei-Fei, S. Savarese, and A. Alahi, "Social gan: Socially acceptable trajectories with generative adversarial networks", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2255–2264.

[31] S. Casas, W. Luo, and R. Urtasun, "Intentnet: Learning to predict intention from raw sensor data", in *Conference on Robot Learning*, PMLR, 2018, pp. 947–956.

[32] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese, "Social lstm: Human trajectory prediction in crowded spaces", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 961–971.

[33] J. Mercat, T. Gilles, N. El Zoghby, G. Sandou, D. Beauvois, and G. P. Gil, "Multi-head attention for multi-modal joint vehicle motion forecasting", in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2020, pp. 9638–9644.

[34] J. Gao, C. Sun, H. Zhao, *et al.*, "Vectornet: Encoding hd maps and agent dynamics from vectorized representation", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 11 525–11 533.

[35] J. Ngiam, V. Vasudevan, B. Caine, *et al.*, "Scene transformer: A unified architecture for predicting future trajectories of multiple agents", in *International Conference on Learning Representations*, 2022.

[36] S. Casas, A. Sadat, and R. Urtasun, "Mp3: A unified model to map, perceive, predict and plan", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 14 403–14 412.

[37] T. Phan-Minh, E. C. Grigore, F. A. Boulton, O. Beijbom, and E. M. Wolff, "Covernet: Multimodal behavior prediction using trajectory sets", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 14 074–14 083.

[38] N. Lee, W. Choi, P. Vernaza, C. B. Choy, P. H. Torr, and M. Chandraker, "Desire: Distant future prediction in dynamic scenes with interacting agents", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 336–345.

[39] F. Marchetti, F. Becattini, L. Seidenari, and A. D. Bimbo, "Mantra: Memory augmented networks for multiple trajectory prediction", in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 7143–7152.

[40] Y. Biktairov, M. Stebelev, I. Rudenko, O. Shliazhko, and B. Yangel, "Prank: Motion prediction based on ranking", *Advances in neural information processing systems*, vol. 33, pp. 2553–2563, 2020.

[41] H. Cui, V. Radosavljevic, F.-C. Chou, *et al.*, "Multimodal trajectory predictions for autonomous driving using deep convolutional networks", in *2019 International Conference on Robotics and Automation (ICRA)*, IEEE, 2019, pp. 2090–2096.

[42] Y. Chai, B. Sapp, M. Bansal, and D. Anguelov, "Multipath: Multiple probabilistic anchor trajectory hypotheses for behavior prediction", *arXiv preprint arXiv:1910.05449*, 2019.

[43] T. Buhet, E. Wirbel, A. Bursuc, and X. Perrotton, "Plop: Probabilistic polynomial objects trajectory prediction for autonomous driving", in *Conference on Robot Learning*, PMLR, 2021, pp. 329–338.

[44] N. Rhinehart, K. M. Kitani, and P. Vernaza, "R2p2: A reparameterized pushforward policy for diverse, precise generative path forecasting", in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 772–788.

[45] H. Cui, T. Nguyen, F.-C. Chou, *et al.*, "Deep kinematic models for kinematically feasible vehicle trajectory predictions", in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2020, pp. 10 563–10 569.

[46] Y. B. Can, A. Liniger, O. Unal, D. Paudel, and L. Van Gool, "Understanding bird's-eye view of road semantics using an onboard camera", *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 3302–3309, 2022. DOI: 10.1109/LRA.2022.3146898.

[47] R. Mahjourian, J. Kim, Y. Chai, M. Tan, B. Sapp, and D. Anguelov, "Occupancy flow fields for motion forecasting in autonomous driving", *IEEE Robotics and Automation Letters*, 2022.

[48] B. Ivanovic, K.-H. Lee, P. Tokmakov, *et al.*, "Heterogeneous-agent trajectory forecasting incorporating class uncertainty", *arXiv preprint arXiv:2104.12446*, 2021.

[49] W. Zeng, M. Liang, R. Liao, and R. Urtasun, "Lanercnn: Distributed representations for graph-centric motion forecasting", in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2021, pp. 532–539.

[50] W. Zeng, W. Luo, S. Suo, *et al.*, "End-to-end interpretable neural motion planner", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8660–8669.

[51] B. Triggs, "Motion planning for nonholonomic vehicles: An introduction", 1993.

[52] W. Zhan, L. Sun, D. Wang, *et al.*, "Interaction dataset: An international, adversarial and cooperative motion dataset in interactive driving scenarios with semantic maps", *arXiv preprint arXiv:1910.03088*, 2019.

[53] H. John, Z. Guido, B. Luca, *et al.*, "One thousand and one hours: Self-driving motion prediction dataset", *arXiv preprint arXiv: 2006.14480*, 2020.

[54] S. Ettinger, S. Cheng, B. Caine, *et al.*, "Large scale interactive motion forecasting for autonomous driving: The waymo open motion dataset", in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 9710–9719.

[55] H. Caesar, V. Bankiti, A. H. Lang, *et al.*, "Nuscenes: A multimodal dataset for autonomous driving", in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 11 621–11 631.

[56] A. Malinin, N. Band, G. Chesnokov, *et al.*, "Shifts: A dataset of real distributional shift across multiple large-scale tasks", *arXiv preprint arXiv:2107.07455*, 2021.

[57] S. Kim, H. Jeon, J. Choi, and D. Kum, "Improving diversity of multiple trajectory prediction based on map-adaptive lane loss", *arXiv preprint arXiv:2206.08641*, 2022.

[58] P. Kaur, S. Taghavi, Z. Tian, and W. Shi, "A survey on simulators for testing self-driving cars", *arXiv preprint arXiv:2101.05337*, 2021.

[59] R. F. Benekohal and J. Treiterer, "Carsim: Car-following model for simulation of traffic in normal and stop-and-go conditions", *Transportation research record*, vol. 1194, pp. 99–111, 1988.

[60] M. Tideman and M. Van Noort, "A simulation tool suite for developing connected vehicle systems", in *2013 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2013, pp. 713–718.

[61] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator", in *Conference on robot learning*, PMLR, 2017, pp. 1–16.

[62] A. Sanders, *An introduction to unreal engine 4*. AK Peters/CRC Press, 2016.

[63] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator", in *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, IEEE, vol. 3, 2004, pp. 2149–2154.

[64] G. Rong, B. H. Shin, H. Tabatabaee, *et al.*, "Lgsvl simulator: A high fidelity simulator for autonomous driving", in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, IEEE, 2020, pp. 1–6.

[65] J. Haas, "A history of the unity game engine", *Diss. WORCESTER POLYTECHNIC INSTITUTE*, 2014.

[66] R. Lattarulo, J. Pérez, and M. Dendaluce, "A complete framework for developing and testing automated driving controllers", *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 258–263, 2017.

[67] H. W. Kuhn, "The hungarian method for the assignment problem", *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[68] G. Bishop, G. Welch, *et al.*, "An introduction to the kalman filter", *Proc of SIGGRAPH, Course*, vol. 8, no. 27599-23175, p. 41, 2001.

[69] R. Schubert, E. Richter, and G. Wanielik, "Comparison and evaluation of advanced motion models for vehicle tracking", in *2008 11th international conference on information fusion*, IEEE, 2008, pp. 1–6.

[70] S. Hochreiter and J. Schmidhuber, "Long short-term memory", *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[71] I. Goodfellow, J. Pouget-Abadie, M. Mirza, *et al.*, "Generative adversarial networks", *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020.

[72] A. Mino and G. Spanakis, "Logan: Generating logos with a generative adversarial neural network conditioned on color", in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, IEEE, 2018, pp. 965–970.

[73] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need", *Advances in neural information processing systems*, vol. 30, 2017.

[74] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks", *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.

[75] D. E. Rumelhart, R. Durbin, R. Golden, and Y. Chauvin, "Backpropagation: The basic theory", *Backpropagation: Theory, architectures and applications*, pp. 1–34, 1995.

[76] H. Robbins and S. Monro, "A stochastic approximation method", *The annals of mathematical statistics*, pp. 400–407, 1951.

[77] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization.", *Journal of machine learning research*, vol. 12, no. 7, 2011.

[78] F. Zou, L. Shen, Z. Jie, W. Zhang, and W. Liu, "A sufficient condition for convergences of adam and rmsprop", in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 11 127–11 135.

[79] M. D. Zeiler, "Adadelta: An adaptive learning rate method", *arXiv preprint arXiv:1212.5701*, 2012.

[80] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization", *arXiv preprint arXiv:1412.6980*, 2014.

[81] X. Weng, J. Wang, D. Held, and K. Kitani, “3d multi-object tracking: A baseline and new evaluation metrics”, in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2020, pp. 10 359–10 366.

[82] H.-k. Chiu, J. Li, R. Ambruş, and J. Bohg, “Probabilistic 3d multi-modal, multi-object tracking for autonomous driving”, in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2021, pp. 14 227–14 233.

[83] C. Gómez–Huélamo, L. M. Bergasa, R. Gutiérrez, J. F. Arango, and A. Díaz, “Smartmot: Exploiting the fusion of hdmaps and multi-object tracking for real-time scene understanding in intelligent vehicles applications”, in *2021 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2021, pp. 710–715.

[84] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, “Pointpillars: Fast encoders for object detection from point clouds”, in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 12 697–12 705.

[85] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection”, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[86] K. Wong, Y. Gu, and S. Kamijo, “Mapping for autonomous driving: Opportunities and challenges”, *IEEE Intelligent Transportation Systems Magazine*, vol. 13, no. 1, pp. 91–106, 2020.

[87] M. Dupuis, M. Strobl, and H. Grezlikowski, “Opendrive 2010 and beyond–status and future of the de facto standard for the description of road networks”, in *Proc. of the Driving Simulation Conference Europe*, 2010, pp. 231–242.

[88] A. Sadeghian, V. Kosaraju, A. Sadeghian, N. Hirose, H. Rezatofighi, and S. Savarese, “Sophie: An attentive gan for predicting paths compliant to social and physical constraints”, in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 1349–1358.

[89] J. Hong, B. Sapp, and J. Philbin, “Rules of the road: Predicting driving behavior with a convolutional model of semantic interactions”, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8454–8462.

[90] M. Quigley, K. Conley, B. Gerkey, *et al.*, “Ros: An open-source robot operating system”, in *ICRA workshop on open source software*, Kobe, Japan, vol. 3, 2009, p. 5.

[91] H. Tverberg, “A proof of the jordan curve theorem”, *Bulletin of the London Mathematical Society*, vol. 12, no. 1, pp. 34–38, 1980.

[92] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, “Simple online and realtime tracking”, in *2016 IEEE international conference on image processing (ICIP)*, IEEE, 2016, pp. 3464–3468.

[93] K. Bernardin and R. Stiefelhagen, “Evaluating multiple object tracking performance: The clear mot metrics”, *EURASIP Journal on Image and Video Processing*, vol. 2008, pp. 1–10, 2008.

[94] W. Zhang, H. Zhou, S. Sun, Z. Wang, J. Shi, and C. C. Loy, “Robust multi-modality multi-object tracking”, in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 2365–2374.

[95] E. Baser, V. Balasubramanian, P. Bhattacharyya, and K. Czarnecki, “Fantrack: 3d multi-object tracking with feature association network”, in *2019 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2019, pp. 1426–1433.

[96] X. Weng and K. Kitani, “Monocular 3d object detection with pseudo-lidar point cloud”, in *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, 2019.

[97] J.-M. Jullien, C. Martel, L. Vignollet, and M. Wentland, “Openscenario: A flexible integrated environment to develop educational activities based on pedagogical scenarios”, in *2009 Ninth IEEE International Conference on Advanced Learning Technologies*, IEEE, 2009, pp. 509–513.

[98] C. Gómez-Huélamo, J. Del Egido, L. M. Bergasa, *et al.*, “Train here, drive there: Ros based end-to-end autonomous-driving pipeline validation in carla simulator using the nhtsa typology”, *Multimedia Tools and Applications*, pp. 1–28, 2021.

[99] P. Dendorfer, A. Osep, and L. Leal-Taixé, “Goal-gan: Multimodal trajectory prediction based on goal position estimation”, in *Proceedings of the Asian Conference on Computer Vision*, 2020.

[100] S. Pellegrini, A. Ess, K. Schindler, and L. Van Gool, “You’ll never walk alone: Modeling social behavior for multi-target tracking”, in *2009 IEEE 12th international conference on computer vision*, IEEE, 2009, pp. 261–268.

[101] A. Lerner, Y. Chrysanthou, and D. Lischinski, “Crowds by example”, *Comput. Graph. Forum*, vol. 26, pp. 655–664, Sep. 2007. DOI: 10.1111/j.1467-8659.2007.01089.x.

[102] J. Amirian, J.-B. Hayet, and J. Pettré, “Social ways: Learning multi-modal distributions of pedestrian trajectories with gans”, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2019.

[103] X. Li, X. Ying, and M. C. Chuah, “Grip: Graph-based interaction-aware trajectory prediction”, in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, IEEE, 2019, pp. 3960–3966.

[104] A. Vemula, K. Muelling, and J. Oh, “Social attention: Modeling attention in human crowds”, in *2018 IEEE international Conference on Robotics and Automation (ICRA)*, IEEE, 2018, pp. 4601–4607.

[105] C. Gómez-Huélamo, M. V. Conde, M. Ortiz, S. Montiel, R. Barea, and L. M. Bergasa, “Exploring attention gan for vehicle motion prediction”, in *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*, IEEE, 2022, pp. 4011–4016.

[106] C. Tang, W. Zhan, and M. Tomizuka, “Exploring social posterior collapse in variational autoencoder for interaction modeling”, *Advances in Neural Information Processing Systems*, vol. 34, pp. 8481–8494, 2021.

[107] M. Ahmed, R. Seraj, and S. M. S. Islam, “The k-means algorithm: A comprehensive survey and performance evaluation”, *Electronics*, vol. 9, no. 8, p. 1295, 2020.

[108] *Argoverse benchmark*, https://eval.ai/web/challenges/challenge-page/454/evaluation, Accessed: 2022-03-19.

[109] L. Fang, Q. Jiang, J. Shi, and B. Zhou, “Tpnet: Trajectory proposal network for motion prediction”, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 6797–6806.

[110] T. Xie and J. C. Grossman, “Crystal graph convolutional neural networks for an accurate and interpretable prediction of material properties”, *Physical review letters*, vol. 120, no. 14, p. 145 301, 2018.

[111] N. Djuric, H. Cui, Z. Su, *et al.*, “Multixnet: Multiclass multistage multimodal motion prediction”, in *2021 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2021, pp. 435–442.

[112] R. S. Ross, “Planning minimum-energy paths in an off-road environment with anisotropic traversal costs and motion constraints”, NAVAL POSTGRADUATE SCHOOL MONTEREY CA, Tech. Rep., 1989.

[113] A. Savitzky and M. J. Golay, “Smoothing and differentiation of data by simplified least squares procedures.”, *Analytical chemistry*, vol. 36, no. 8, pp. 1627–1639, 1964.

[114] M. Ye, T. Cao, and Q. Chen, “Tpcn: Temporal point cloud networks for motion forecasting”, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 11 318–11 327.

[115] Z. Wang, J. Guo, Z. Hu, H. Zhang, J. Zhang, and J. Pu, “Lane transformer: A high efficiency trajectory prediction model”, *IEEE Open Journal of Intelligent Transportation Systems*, 2023.

[116] Y. Liu, J. Zhang, L. Fang, Q. Jiang, and B. Zhou, “Multimodal motion prediction with stacked transformers”, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 7577–7586.

[117] N. Deo and M. M. Trivedi, “Convolutional social pooling for vehicle trajectory prediction”, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2018, pp. 1468–1476.

[118] Q. Li, Z. Han, and X.-M. Wu, “Deeper insights into graph convolutional networks for semi-supervised learning”, in *Thirty-Second AAAI conference on artificial intelligence*, 2018.

[119] Z. Zhou, L. Ye, J. Wang, K. Wu, and K. Lu, “Hivt: Hierarchical vector transformer for multi-agent motion prediction”, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 8823–8833.

[120] P. Bhattacharyya, C. Huang, and K. Czarnecki, “Ssl-lanes: Self-supervised learning for motion forecasting in autonomous driving”, in *Conference on Robot Learning*, PMLR, 2023, pp. 1793–1805.

[121] R. Walters, J. Li, and R. Yu, “Trajectory prediction using equivariant continuous convolution”, *arXiv preprint arXiv:2010.11344*, 2020.

[122] J. Gu, Q. Sun, and H. Zhao, “Densetnt: Waymo open dataset motion prediction challenge 1st place solution”, *arXiv preprint arXiv:2106.14160*, 2021.

[123] B. He and Y. Li, “Multi-future transformer: Learning diverse interaction modes for behaviour prediction in autonomous driving”, *IET Intelligent Transport Systems*, 2022.

[124] C. Katrakazas, M. Quddus, W.-H. Chen, and L. Deka, “Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions”, *Transportation Research Part C: Emerging Technologies*, vol. 60, pp. 416–442, 2015, ISSN: 0968-090X. DOI: https://doi.org/10.1016/j.trc.2015.09.011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0968090X15003447.

[125] C. Zhang, H. Sun, C. Chen, and Y. Guo, “Banet: Motion forecasting with boundary aware network”, *arXiv preprint arXiv:2206.07934*, vol. 2, 2022.

[126] C. Gómez-Huélamo, M. V. Conde, R. Barea, and L. M. Bergasa, “Improving multi-agent motion prediction with heuristic goals and motion refinement”, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 5322–5331.

[127] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, “Feature pyramid networks for object detection”, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2117–2125.

[128] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, “The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains”, *IEEE signal processing magazine*, vol. 30, no. 3, pp. 83–98, 2013.

[129] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks", *arXiv preprint arXiv:1609.02907*, 2016.

[130] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions", *arXiv preprint arXiv:1511.07122*, 2015.

[131] M. Liu, H. Cheng, L. Chen, *et al.*, "Laformer: Trajectory prediction for autonomous driving with lane-aware scene constraints", *arXiv preprint arXiv:2302.13933*, 2023.

[132] T. Gilles, S. Sabatini, D. Tsishkou, B. Stanciulescu, and F. Moutarde, "Thomas: Trajectory heatmap output with learned multi-agent sampling", *arXiv preprint arXiv:2110.06607*, 2021.

[133] Y. Wang, H. Zhou, Z. Zhang, *et al.*, "Tenet: Transformer encoding network for effective temporal flow on motion prediction", *arXiv preprint arXiv:2207.00170*, 2022.

[134] M. Zhou, J. Luo, J. Villella, *et al.*, *Smarts: Scalable multi-agent reinforcement learning training school for autonomous driving*, Nov. 2020. [Online]. Available: https://arxiv.org/abs/2010.09776.

[135] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz, "Sumo - simulation of urban mobility: An overview", in *in SIMUL 2011, The Third International Conference on Advances in System Simulation*, 2011, pp. 63–68.

[136] J. de la Peña, L. M. Bergasa, M. Antunes, F. Arango, C. Gómez-Huélamo, and E. López-Guillén, "Ad perdevkit: An autonomous driving perception development kit using carla simulator and ros", in *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*, IEEE, 2022, pp. 4095–4100.

[137] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A survey of autonomous driving: Common practices and emerging technologies", *CoRR*, vol. abs/1906.05113, 2019. arXiv: 1906.05113. [Online]. Available: http://arxiv.org/abs/1906.05113.

[138] B. R. Kiran, I. Sobh, V. Talpaert, *et al.*, "Deep reinforcement learning for autonomous driving: A survey", *CoRR*, vol. abs/2002.00444, 2020. arXiv: 2002.00444. [Online]. Available: https://arxiv.org/abs/2002.00444.

[139] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Human-level control through deep reinforcement learning", *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015, ISSN: 00280836. [Online]. Available: http://dx.doi.org/10.1038/nature14236.

[140] L. Zhu, F. R. Yu, Y. Wang, B. Ning, and T. Tang, "Big data analytics in intelligent transportation systems: A survey", *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 1, pp. 383–398, 2019. DOI: 10.1109/TITS.2018.2815678.

[141] D. Silver, A. Huang, C. J. Maddison, *et al.*, "Mastering the game of Go with deep neural networks and tree search", *Nature*, vol. 529, no. 7587, pp. 484–489, 2016, ISSN: 0028-0836. DOI: 10.1038/nature16961.

[142] S. Ross, G. J. Gordon, and J. A. Bagnell, "No-regret reductions for imitation learning and structured prediction", *CoRR*, vol. abs/1011.0686, 2010. arXiv: 1011.0686. [Online]. Available: http://arxiv.org/abs/1011.0686.

[143] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms", *CoRR*, vol. abs/1707.06347, 2017. arXiv: 1707.06347. [Online]. Available: http://arxiv.org/abs/1707.06347.

[144] R. Gutiérrez-Moreno, R. Barea, E. López-Guillén, F. Arango, N. Abdeselam, and L. M. Bergasa, "Hybrid decision making for autonomous driving in complex urban scenarios", in *2023 IEEE Intelligent Vehicles Symposium (IV)*, 2023.

[145] C. Hoel, K. Wolff, and L. Laine, "Automated speed and lane change decision making using deep reinforcement learning", *CoRR*, vol. abs/1803.10056, 2018. arXiv: 1803.10056. [Online]. Available: http://arxiv.org/abs/1803.10056.

[146] T. Tram, A. Jansson, R. Grönberg, M. Ali, and J. Sjöberg, "Learning negotiating behavior between cars in intersections using deep q-learning", *CoRR*, vol. abs/1810.10469, 2018. arXiv: 1810.10469. [Online]. Available: http://arxiv.org/abs/1810.10469.

[147] Z. Zhang, A. Liniger, D. Dai, F. Yu, and L. V. Gool, *End-to-end urban driving by imitating a reinforcement learning coach*, 2021. arXiv: 2108.08265 [cs.CV].

[148] I. Kostrikov, D. Yarats, and R. Fergus, *Image augmentation is all you need: Regularizing deep reinforcement learning from pixels*, 2021. arXiv: 2004.13649 [cs.LG].

[149] M. Moghadam and G. H. Elkaim, *A hierarchical architecture for sequential decision-making in autonomous driving using deep reinforcement learning*, 2019. arXiv: 1906.08464 [cs.RO].

[150] G. Li, Y. Qiu, Y. Yang, *et al.*, "Lane change strategies for autonomous vehicles: A deep reinforcement learning approach based on transformer", *IEEE Transactions on Intelligent Vehicles*, vol. 8, no. 3, pp. 2197–2211, 2023. DOI: 10.1109/TIV.2022.3227921.

[151] R. Valiente, M. Razzaghpour, B. Toghi, G. Shah, and Y. P. Fallah, *Prediction-aware and reinforcement learning based altruistic cooperative driving*, 2022. arXiv: 2211.10585 [cs.RO].

[152] R. Gutiérrez-Moreno, R. Barea, E. López-Guillén, J. Araluce, and L. M. Bergasa, "Reinforcement learning-based autonomous driving at intersections in carla simulator", *Sensors*, vol. 22, no. 21, 2022, ISSN: 1424-8220. DOI: 10.3390/s22218373. [Online]. Available: https://www.mdpi.com/1424-8220/22/21/8373.

[153] T. Haarnoja, A. Zhou, K. Hartikainen, *et al.*, *Soft actor-critic algorithms and applications*, 2019. arXiv: 1812.05905 [cs.LG].

[154] J. Knodt, J. Bartusek, S.-H. Baek, and F. Heide, *Neural ray-tracing: Learning surfaces and reflectance for relighting and view synthesis*, 2021. arXiv: 2104.13562 [cs.CV].

[155] S. M. Koksbang and S. Hannestad, "Studying the precision of ray tracing techniques with szekeres models", *Physical Review D*, vol. 92, no. 2, 2015, ISSN: 1550-2368. DOI: 10.1103/physrevd.92.023532. [Online]. Available: http://dx.doi.org/10.1103/PhysRevD.92.023532.