Master's research project

Corner Case Generation for Motion Planning

École Télécom Paris, Valeo AI

Student : Pegah Khayatan
pegah.khayatan@polytechnique.edu

Supervised by : Alexandre Boulch, Eloi Zablocki
alexandre.boulch@valeo.com, *eloi.zablocki@valeo.com*
Academic Advisor: Michel Roux
michel.roux@telecom-paris.fr

Acknowledgments

I would like to express my sincere gratitude to the individuals whose support and guidance have been crucial in the completion of this project.

First and foremost, my deep appreciation goes to my supervisors in this project, Alexandre Boulch and Eloi Zablocki, for their invaluable mentorship, expertise, and continuous encouragement throughout the research process. Their insightful feedback significantly enriched the quality of this work.

I wish to express my profound gratitude to Patrick Pérez and Matthieu Cord, previous and current scientific directors of Valeo.ai, for their unwavering support throughout this experience.

I am grateful to my colleagues and friends, including Amaia, Thibaut, Sophia, and others who provided support, encouragement, and a productive environment.

Finally, I extend my deepest thanks to my partner for his support through this internship.

This project would not have been possible without these individuals, and I am truly grateful for their support.

Contents

1 Introduction

1.1 Context

Advances in machine learning algorithms have contributed immensely to the development of autonomous driving systems in the last few years and have enabled the employment of autonomous cars and taxis on the streets. This fast-growing deployment requires us to take a closer look at their safety-critical evaluation to prevent accidents. However, critical scenarios used for evaluation are rare when extracted from the streets or generated using simulation engines, and they do not provide many corner cases. On the other hand, producing dangerous scenarios in the real world to include in datasets is costly and not a scalable and reasonable solution. Therefore, methods to artificially generate safety-critical driving scenarios have become an important line of work.

These artificially generated scenarios need to be designed in such a way that could be used to improve the performance of the existing systems in real scenarios. Hence, the generated scenarios should be *realistic*- it is not important to refine the performance of the system on cases that do not happen in the real world-; Other than this initial criterion, our interest lies in ensuring the diversity of these scenarios, enabling the exploration of various cases and ensuring transferability. This implies that the scenarios should not exclusively represent challenging cases for a single type of planner.

In this project, we take steps towards generating such scenarios; we investigate the interaction between agents in a simulated environment based on real-world scenarios, NuPlan [Cae+21], and aim to generate corner cases that could potentially expose the agents to the risk of collision.

Figure 1: Examples of accidents involving autonomous vehicles. Left) TESLA crash on San Francisco's BAY BRIDGE, due to sudden break of the autonomous vehicle, Right) high-speed crash involving an autonomous Uber Volvo SUV (not responsible for the crash in this case).

1.2 Contributions

The main contribution of this project is to provide a framework to stress-test vehicle planners, using an optimization method introduced by [Han+22], titled KING: Generating Safety-Critical Driving Scenarios for Robust Imitation via Kinematics Gradients.
We use a realistic simulator for our experiments where the scenes are diverse and the number of involved agents in each scene is not limited, in opposite to Carla [Dos+17] simulator employed in [Han+22]'s experiments. Importantly, the code is built with a modular structure, and on top of the NuPlan [Cae+21] simulator.
We also propose a method to improve upon the estimation of the performed actions by a vehicle from its trajectory.
In a nutshell, we studied the feasibility of the method proposed in [Han+22] within a realistic simulator, and investigated the proposed approach in a different setup in terms of the number of agents and their diverse driving behaviors.

1.3 Timeline of the internship

- **July:** Bibliography on different methods for generating adversary trajectories and stress-testing planners. Getting familiar with Nuplan [Cae+21] simulator and dataset by performing simple tests and training.
- **August:** We decided that our selected method to implement would be based on the paper [Han+22], due to its creativity, simplicity, and explainability. Starting to understand the code of [Han+22] in more depth.
- **September:** Implementation of the code.
- **October:** Performing sanity-check tests using the code, studying the optimization process more closely for different agents.
- **November:** Proposing an equivalent for one of the costly losses used in [Han+22]. Performing empirical investigations to determine optimal hyperparameter values. We identified significant challenges within the reconstruction component of the project, which were more visible in certain scenes than others.
- **December:** Proposing solutions to improve the reconstruction of trajectories. Cleaning the code to be transferred to supervisors.
- **January:** Performing final experiments and documenting the results.

1.4 Related Works

The methods for generating driving adversary samples fall into two main categories:

1) methods that use direct sampling from the dataset to leverage the scenarios that rarely happen, and then utilize a risk metric to filter those that can be considered as corner cases. This category also includes strategies that apply random perturbation on the parameters of the scenarios and the agent. One example of such techniques is KING [Han+22], on which the main approach of this project is based.

2) density estimation methods that consider the driving samples to follow a distribution; they then use real data to learn this distribution. The learned distribution can be leveraged to generate samples that follow the same behavior but could be modified to satisfy extra imposed constraints, such as colliding with other vehicles.

Another categorization of adversary sample-generation methods is based on the target of the attack (which part of the perception-planning-control is targeted?). Some works only attack a single part, and others target the whole pipeline.

[Tu+20] presents a method to perturb the Lidar perception module by generating a physically realizable object that would be invisible to LiDAR detectors, with a success rate of 80% when placed on the roof of a vehicle. [Wan+21] generates critical scenarios for Lidar-based planners and attacks the full stack of planning. Their method consists of applying modifications to actors' trajectories in a heuristic manner. The input to the planner is then updated based on the modified trajectory. In a subsequent step, these modified trajectories are filtered based on their plausibility and a cost that motivates the ego vehicle to deviate from the recorded trajectory, collide with other agents, and have lane violations Figure 2.

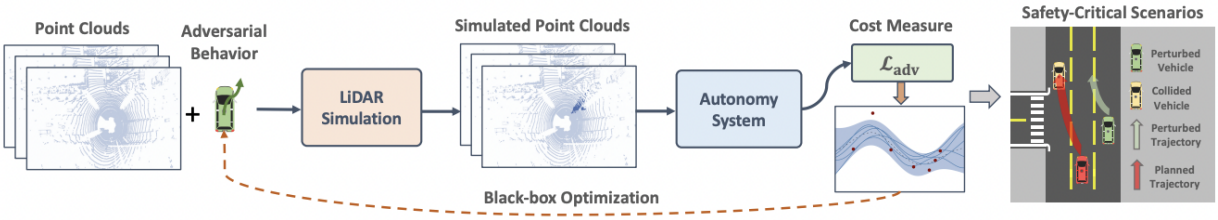


Figure 2: Pipeline of the method proposed by [Wan+21] to generate adversary driving samples; the strategy involves updating the scenario, adjusting LiDAR sensor data based on the new configuration, evaluating the autonomy system's performance, computing an adversarial objective, and iteratively refining the perturbation through a search algorithm.

[Rem+22] proposes to learn the traffic pattern using a neural network -by learning a model of traffic motion in the form of a graph-based conditional VAE-, and to then use this learned representation to parametrize trajectories in a latent space and as prior to evaluate scenario plausibility (how plausible is a generated trajectory according to the learned distribution of trajectories?).
Consequently, they provide an optimization procedure that encourages the *realistic* generation of *challenging* scenarios. One of their contributions is using a proxy for the planner model, to be able to perform an end-to-end optimization also involving the parameters of the planner for the ego vehicle (cf. Figure 3).

In [Cao+22], a dense trajectory from the sampled one in the dataset is constructed at first -along with deriving the dense dynamic parameters for densely sampled trajectory points- and then the actions are optimized to mislead the predictions (of ego, adversary, or the other vehicle) at each time step,

which would lead to unsafe driving behaviors. They put great emphasis on keeping the generated trajectories close to their ground truth values, their fidelity to reality, and following common-sense traffic rules; hence, they consider costs aligned with this objective such as avoiding collision or the trajectory to pass from points close to the original trajectory; instead of directly inducing collisions, their loss aims to mislead the prediction by maximizing the difference between the predicted future trajectory and ground truth. They empirically showed that their approach yields adversary trajectories of greater realism in comparison to [Rem+22]. They stated that this improved realism stems from the dense initialization of the trajectories (cf. Figure 3).

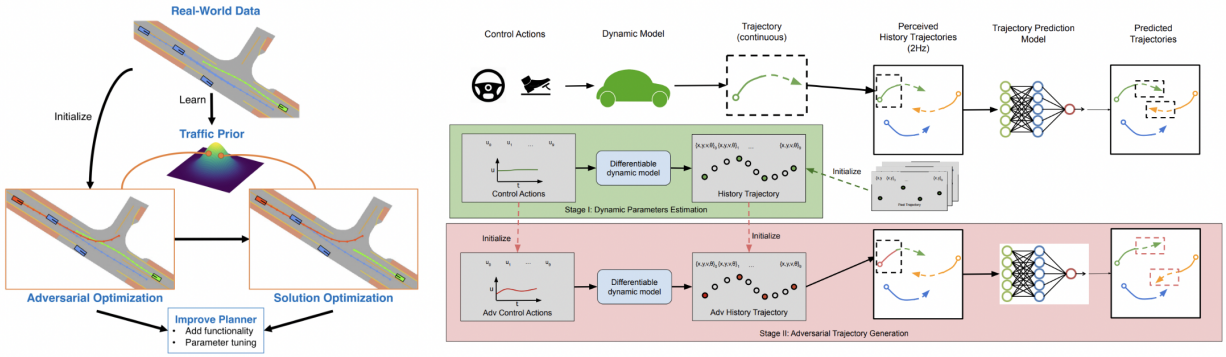


Figure 3: left) An overall schema of the adversary sample generation method proposed by [Rem+22]: Perturbing a real-world scene in the latent space of a learned traffic model to induce a collision between an adversary and the planner. Subsequent optimization seeks a planner trajectory that can avoid collisions, providing a valid solution that can then be used to fine-tune the planner., right) Compositional structure of trajectory reconstruction and adversary sample generation in [Cao+22].

[Din+23] introduces a novel generative model, Causal Autoregressive Flow (CausalAF), which incorporates causality as a prior and focuses on learning cause-and-effect relationships to generate safety-critical scenarios. The generated samples are then shown to be useful to improve data-driven and reinforcement learning-based driving algorithms.

There exists another branch of work that takes a novel view of the problem of generating safety-adverse driving scenarios by attacking the map rather than the trajectory of the agents. In this direction, [Bah+22] introduces a generative model using adversarial scene generation to automatically create realistic scenes that cause even recent planning models to go off-road.

2 Trajectory Prediction, Planning, and Simulation

The precise forecasting of future trajectories for multiple agents is a fundamental requirement in autonomous systems, given the agent interactions and the inherent unpredictability of each agent's future actions.

The task of forecasting the trajectory of an agent is called *trajectory prediction* (we will use it interchangeably with forecasting). Trajectory prediction techniques are used to anticipate the future paths and behaviors of objects or agents. They enable proactive decision-making for a planner and enhance safety and efficiency if aligned with the behavior of agents in the real world.
trajectory prediction methods mainly focus on the behavior of other agents in the short term and do not consider a *goal* for the trajectory (in contrast to trajectory planning). Moreover, prediction is a multi-modal task, meaning that for each agent we predict *several* possible trajectories.
Planning is a task closely related to trajectory prediction, however, it differs from forecasting in two key ways: it focuses on a single trajectory, and its primary purpose is to control vehicle behavior rather than only forecasting trajectories.

It is essential to note that the predominant evaluation approach of forecasting methods primarily seeks to mimic the ground truth of trajectories, which is meaningful for trajectory prediction but is not adapted to the task of planning. To develop a more comprehensive assessment, there is a need to develop evaluation methodologies that account for a broader array of real-world complexities, encompassing unforeseeable behaviors and contextual variables. This approach will better measure the practical applicability and resilience of trajectory planning models in autonomous systems.

In this context, simulation plays an important role in evaluating planning methods. Essentially, when we simulate the interaction of vehicles in a given scene, we complete the planning loop. This means that by incorporating simulation, we not only forecast the individual trajectories of vehicles and plan the trajectory of the autonomous vehicle (often called the ego vehicle) but also account for how they interact with one another within the context of that scene. This comprehensive approach allows us to better understand and anticipate the dynamic behaviors and relationships among vehicles.

In the following two subsections, first, we briefly discuss main trajectory prediction datasets, planning datasets, and a few known driving simulators; its following section is dedicated to a formal definition of each task, and an overview of the main lines of work in trajectory prediction and planning.

2.1 Datasets

2.1.1 Trajectory Prediction Datasets

[Cae+20]

In the past ten years, there has been a notable emergence of driving datasets that have played a crucial role in advancing research related to scene understanding for autonomous vehicles, providing a foundation for the development of robust and sophisticated navigation algorithms. The majority of these datasets have primarily focused on providing 2D annotations, such as bounding boxes and masks, for RGB camera images. Notable datasets in this category include CamVid [Bro+08], Cityscapes [Cor+16], Mapillary Vistas [Neu+17], D2-City [Che+19], BDD100k [Yu+20], and Apolloscape [Hua+18], all of which have continuously expanded their datasets with segmentation masks. Additionally, Vistas, D2-City, and BDD100k have gone further by including images captured under various weather and

lighting conditions. The relative simplicity of capturing and annotating RGB images has facilitated the release of these extensive image-only datasets.

In contrast, multimodal datasets, which typically encompass images, data from Lidars and radars, and GPS/IMU data, pose a greater challenge due to the complexities associated with integrating, synchronizing, and calibrating multiple sensor sources. The KITTI dataset [Gei+13] was the pioneering multimodal dataset, providing dense point clouds from a LiDAR sensor, front-facing stereo images, and GPS/IMU data. This dataset has significantly contributed to the advancement of 3D object detection, offering $200,000$ 3D object annotations across 22 scenes. More recently, the H3D dataset [Pat+19] has emerged, featuring 160 densely populated scenes with a total of 1.1 million 3D object annotations distributed over $27,000$ frames. Notably, H3D provides annotations for objects in a full 360-degree view, which is an improvement from KITTI, where objects are only annotated in the frontal view. The KAIST multispectral dataset [Cho+18] is another multimodal dataset, incorporating RGB and thermal cameras, RGB stereo, 3D LiDAR, and GPS/IMU data, and it even includes nighttime data.

Recently released datasets have started to incorporate map information of the environment as an integral component. For instance, apart from its array of sensors like cameras, LiDAR, and radar, the nuScenes Dataset [Cae+20] stands out by including rasterized top-down semantic maps for the relevant areas. In a similar vein, the Argoverse Dataset [Cha+19] enhances the dataset landscape by providing geometric and semantic maps of the environment. These maps include information about ground height and a vector representation of road lanes along with their interconnections.

Waymo Open dataset [Sun+20] is another recent dataset that stands out for providing a significantly higher volume of annotations, largely owing to its higher annotation frequency (10Hz compared to the 2Hz of other datasets).

2.1.2 Planning Datasets

CommonRoad [AKM17] comprises a variety of vehicle models, cost functions, and scenarios, which encompass diverse sets of goals and constraints. It includes both pre-recorded and interactive scenarios. With a total of 5700 scenarios, the dataset's size falls short of being sufficient for training contemporary deep learning approaches. Sensor data is not provided in this dataset.

2.1.3 Simulation

Simulators have been crucial in achieving significant advancements in planning and reinforcement learning by their capability to replicate physical dynamics, agents, and environmental variables within a controlled simulation environment. Also, they can close the loop of planning, by capturing the response of other agents (other than the vehicle being controlled) and interactively planning the trajectory.
There exist different simulators, using data from real-world scenarios or game engines.

AirSim [Sha+18] is a high-fidelity simulator encompassing both drones and cars. It is built upon Unreal Engine 4, capable of operating at a high frequency, facilitating real-time hardware-in-the-loop simulations.
CARLA ((Car Learning to Act) [Dos+17] is a simulator commonly used in the development, training, and validation of autonomous urban driving systems. It provides various autonomous driving approaches, sensor configurations, and environmental conditions. It is implemented as an open-source layer over Unreal Engine 4. In the CARLA Autonomous Driving Challenge, participants are tasked

with navigating a predefined set of waypoints using sensor data and HD maps. Alternatively, researchers have the option to utilize scene abstraction, bypassing the perception task and concentrating on planning and control aspects.

NupPlan [Cae+21] proposes a closed-loop simulation framework with reactive agents along with a large set of both general and scenario-specific planning metrics. The scenarios in NuPlan are diverse and initialized from real-world data. In this project, we use this simulator, and more details about this latter will be given in subsequent sections.

Nocturne [Vin+22] is a 2D driving simulator (top-down view of the vehicles and roads), where the agents in this simulator only observe an obstructed view of the scene, mimicking human visual sensing constraints at night. The scenes in the simulator are initialized from the Waymo Open dataset [Sun+20], but can be replaced with any other dataset that represents its road features as points or polylines.

Another recently introduced simulator is Waymax [Gul+23], which has many similarities with NuPlan, and notably leverages openly available real-world driving data, such as the Waymo Open Motion Dataset [Sun+20], to initiate or replay a wide range of multi-agent simulated scenarios. This simulator is entirely differentiable; however, unlike NuPlan [Cae+21], it does not provide (partially) the sensor data, and also its scenarios expand over 9s compared to 15s in this latter.

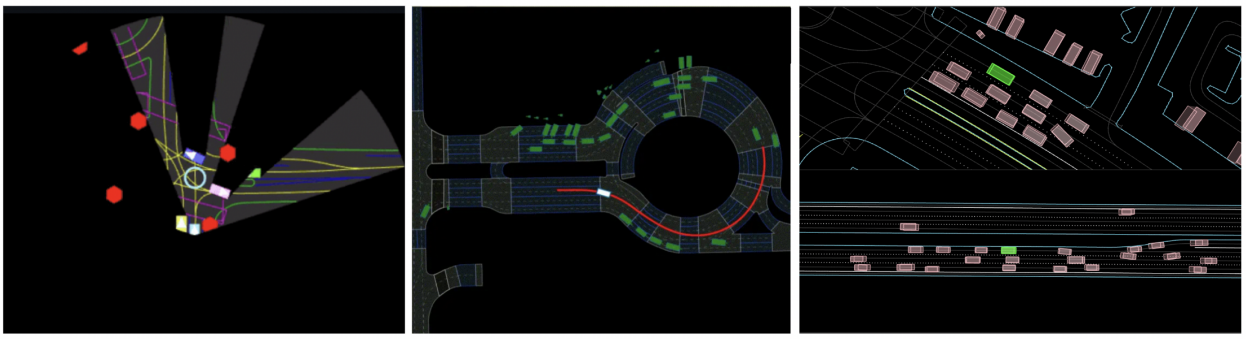


Figure 4: Examples of bird eye view of scenes in [Vin+22], [Cae+21], and [Gul+23] respectively from left to right.

2.2 Main Methods

2.2.1 Trajectory Prediction

A formal definition of the task of trajectory prediction can be given as:

The task is to use the previous states of agents in the scene within a specific scenario to predict their next states: $\mathbf{X} = \{p^1, p^2, ..., p^{t_h}\}$ Where t_h is the length of the observed (history) trajectory, and p^t is the state of the agents at time t: $p^t = \{s_0^t, s_1^t, ..., s_n^t\}$, where s_i^t is the state of the agent i at time t. The state could contain information about the position, heading, velocity, etc. The goal is to predict the trajectory for several next time steps: $Y = \{p^{t_h+1}, p^{t_h+2}, ..., p^{t_h+t_f}\}$ Where t_f is the time step length for prediction. Some methods predict Y directly, and others output an intermediate prediction from which the trajectory is generated [LVL14].

Trajectory prediction methods can be categorized based on their reliance on sensor data. These categories encompass a spectrum from methods that do not use any sensor data to those that use data from few sensors as input.

1. *Non-Sensor-Based Methods* rely solely on historical data, mathematical models, or rule-based

approaches to anticipate future trajectories. They do not incorporate any real-time sensor information, making them computationally efficient and useful in situations where sensors are unavailable or unreliable.

2. *Sparse Sensor-Based Method* Make use of a limited number of sensors. These sensors may provide partial or intermittent information about the environment or objects being tracked. For instance, they may use data from a single sensor type, such as GPS or LiDAR.

3. *Multi-Sensor-Based Methods* leverage data from a variety of sensors. The integration of multiple sensor modalities allows for more accurate and robust trajectory predictions, as it provides a comprehensive view of the surrounding environment; but, at the same time, it requires close supervision to combine different modalities of data.

The most related categorization to this project, in terms of provided input, is *Non-Sensor-Based Methods*, and vision-based techniques are not discussed here.

Another categorization of trajectory prediction methods is *Data-Driven* vs. *Model-Based Methods*. Data-driven methods (classic machine learning-based methods, deep learning-based methods, and reinforcement learning-based methods) rely primarily on historical trajectory data and sensor information to make predictions, while model-based methods employ predefined mathematical or physical models. We discuss each category briefly:

Model-based Methods: Physics models are simplified mathematical representations of the motion of an object that focuses on describing the object's position, velocity, and acceleration without considering the forces or interactions that cause the motion. A few commonly used kinematics models are: Constant-Velocity (CV) and Constant-Acceleration (CA) models [AN09; SRW08; Pol+07], Constant Turn Rate and Velocity (CTRV), Constant Turn Rate and Acceleration (CTRA), etc. [LTA08; BF08; BWB09]. These methods are mostly based on a simplified model of the vehicle's dynamics called bicycle model [Pol+17] that we will discuss in detail in the next sections.

These models are computationally efficient; however, their disadvantage is that They cannot take into account road-related variables, and the current state's unpredictability makes them unreliable for making predictions over an extended period.

One approach to tackle noisy states is to use Kalman Filtering (KF) methods that handle the uncertainty or noise of the current vehicle's state by modeling it as a Gaussian distribution [Kae+04]. Compared to the previous method, these methods predict the uncertainty of the predicted trajectory [Hua+22].

Another drawback of the mentioned works is that they do not take the interaction between different agents and the constantly changing physical model of the agents into account. Different strategies have been proposed to alleviate this issue: [Kae+04] proposed an Interacting Multiple Model (IMM) to output a weighted estimate of parallel kinematics models.

Data-driven Methods: Despite physics-based methods being well-established and providing a deep understanding of the underlying physical processes, they often make simplifying assumptions that can limit their applicability in complex, real-world scenarios. To solve this limitation, data-driven methods leverage vast amounts of observational data to infer patterns and non-linear dynamics that may be challenging to express analytically through physics-based models.

The primary data-driven methods employed for trajectory prediction are the Gaussian Process (GP), Hidden Markov Models (HMM), and Dynamic Bayesian Networks (DBN). As an example of these

methods, we discuss the Gaussian process. This latter provides a probability distribution over possible future positions. When GP is applied to predict trajectory, trajectories are regarded as the samples of GP, sampled along the time axis. The samples are represented by N discrete points to map to the N-dimensional space, where they satisfy the N-dimensional Gaussian distribution in that space. The same approach can be adopted to model interaction-related factors [TK10], or expanded to having several prototypes by dividing the sample trajectories into several subsets and having several prototypes for each class of subsets [Her+09].

Conventional prediction techniques are typically well-suited for straightforward prediction scenarios and short-term forecasting tasks. However, deep learning-based trajectory prediction methods have gained significant importance due to their ability to encompass not just physical and road-related factors, but also factors related to interactions. This latter allows them to excel in more complex environments.

Deep learning methods for trajectory prediction can be categorized into three: Sequential Networks, Graph Neural Networks, and Generative Models [Hua+22].

Sequential Networks serve the purpose of feature extraction from the historical trajectory data. Recurrent Neural Networks (RNN), Convolutional Neural Networks(CNN), and Attention Mechanisms (AM) are all three sequential networks. Different sequential trajectory prediction techniques use one or a combination of these three for the task.

RNNs excel at handling temporal data, yet they face challenges with vanishing or exploding gradients, particularly when dealing with extensive time steps. To mitigate this, Gated RNN models like Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) have been introduced to enhance performance. On the other hand, CNNs have gained popularity due to their proficiency in processing spatial information and capturing spatio-temporal patterns in trajectories. They could be used to extract spatial information from High Definition (HD) maps. These maps, including raster and vector maps, provide semantic insights into roads and interaction-related factors [Hua+22].

Using attention for trajectory prediction leverages the attention mechanism inspired by human cognition. This mechanism filters essential information from vast datasets, making it a valuable tool in deep learning tasks such as NLP, image classification, and speech recognition. In trajectory prediction, attention mechanisms enable the modeling of interactions among traffic participants, extract relevant context features, and achieve parallel calculations on sequential data [Mes+20]. The Transformer-based models, like the Transformer and Bidirectional Transformer, outperform traditional LSTM-based models, especially in long-term predictions and handling missing sensor data, while also effectively modeling interactions between traffic participants and their environment.

Graph Neural Networks: One way to capture the interaction between agents in a scene is to represent them as the nodes in a graph, where the connectivity of different nodes depends on their level of interaction. Graph Neural Networks (GNNs), such as Graph Convolutional Networks (GCN) and Graph Attention Networks (GAT), can be used to model interaction-related factors in trajectory prediction. As an example, a space-based graph convolutional network may treat each vehicle as a node at each sampling time. If two nodes represent the same vehicle at two consecutive sampling times, an edge exists between them, representing the time relationship. Also, if two nodes at the same time represent two vehicles, and the distance between the two vehicles is less than a fixed value, an edge exists between the two nodes, representing the spatial relationship. The graph can then be used to predict the trajectory of an agent, depending on its surroundings and previous trajectory [LYC19].

VectorNet proposed in [Gao+20] suggests utilizing map information by considering vehicles and

vector maps in the scene as nodes. Vector maps use polylines with multiple control points to represent road information. This latter has the advantage of being more computationally efficient than CNNs using rasterized maps. The attention mechanism can be used to extract the most important parts of the data more efficiently [Vel+17; Hua+19].

Generative Neural Networks: A generative model is mainly composed of two parts, namely Generator and Discriminator. The generator is utilized to generate a random sample similar to the real sample, and the discriminator is used to determine whether the data is generated or from the dataset. Through the continuous evolution of the generator and discriminator, GAN can obtain a generator with higher quality and a Discriminator with stronger judgment ability [Goo+14]. When applying GAN to trajectory prediction, the generator is utilized to generate the predicted trajectory, and the discriminator is utilized to judge whether the predicted trajectory is correct and feasible.

2.2.2 Trajectory Planning

As previously noted, trajectory planning differs from trajectory prediction in that its primary purpose is to control the vehicle behavior rather than just forecasting trajectories; also, it often takes as input the destination to be reached.

The recent trajectory planning techniques involve, implicitly [Ngi+21; Alb+21] or explicitly, a component to predict the trajectory of other agents in the scene to avoid collisions.

However, explicitly predicting independent futures for individual agents based on past motion may overlook the potential interactions between agents, resulting in suboptimal planning. In response, [Ngi+21] proposes a model that jointly predicts the behavior of all agents, generating consistent futures that account for inter-agent interactions.

2.2.3 Risk Assessment During Real-Time

In recent years, autonomous vehicles have gained significant attention, with potential deployment in a diverse set of areas. With the development of technology, the focus has been shifted from deploying autonomous vehicles on the roads to ensuring their safe integration.

Safety is evaluated by a set of standards such as succeeding in certain maneuvers without inducing accidents, etc. However, defining a limited set of standards may not reveal all the limitations of an autonomous vehicle, and hence novel strategies should be considered to more effectively tackle this evaluation.

In this project, we are mainly concerned with the planner component of an autonomous vehicle.

One general strategy to challenge a planner is to assess its functionality when placed in scenarios that it has rarely or never been trained on and hence might be lacking the ability to handle them correctly.

A scenario or driving condition can be viewed as rare based on the driving environment (weather conditions, different road surroundings, etc.), or the behavior of other agents in the scene (the driving style might differ from town to town, etc.). We will focus on modifying the second option: modification of the behavior of other agents to create rare and dangerous driving scenarios.

3 Kinematics Model and Controller

3.1 Vehicle Motion Model

In a planning framework, a model for describing the motion of the vehicle is essential to translate planned actions into real-world trajectories. It ensures the execution of planned trajectories, providing a bridge between high-level planning decisions (trajectories) and low-level vehicle dynamics (actions). On the other hand, if the trajectory is planned by the position of its waypoints, a controller is needed to obtain the actions taken by the vehicle to follow that certain path. In most of the studies, the assumption made is that given a constrained planned trajectory, a low-level controller will always be able to track it with a bounded error.

There is a fine relation between motion model and controller, as one translates actions to waypoint positions (and state in general, as it may include other information on the heading of the vehicle, its velocity, etc.), and the other takes as input the trajectory points and provides an approximation of the actions taken to satisfy it.

One of the frequently used motion models is the 3 degrees of freedom kinematic bicycle model, which assumes the two front and two rear wheels are lumped into two wheels respectively, and each is located at the center of its corresponding axle.

The control inputs correspond to the acceleration u_1 and the steering angle u_2 of the front wheels of the vehicle.

$$\dot{X} = V\cos(\psi + \beta(u_2)) \tag{1}$$

$$\dot{Y} = V\sin(\psi + \beta(u_2)) \tag{2}$$

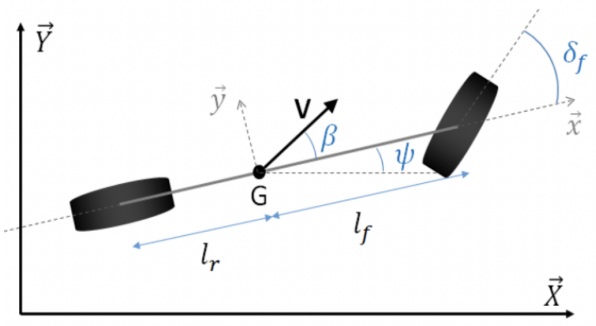
$$\dot{V} = u_1 \tag{3}$$

$$\dot{\psi} = \frac{V}{l_r}\sin(\beta(u_2)) \tag{4}$$

and the slip angle at the center of gravity $\beta(u_2)$ is given by:

$$\beta(u_2) = \arctan\left(\tan(u_2)\frac{l_r}{l_f + l_r}\right) \tag{5}$$

where l_r (resp. l_f) is the distance between the front (resp. rear) axle and the center of gravity.



$$\tan(\beta(u_2)) = \frac{d_r}{l_f + l_r}\tan(u_2)$$

1. **Lateral Distance Ratio:** The numerator d_r represents the lateral distance from the rear axle to the center of gravity, serving as the effective lever arm for the rear axle.
2. **Normalization by Total Wheelbase:** The denominator $l_f + l_r$ represents the total wheelbase, normalizing the lateral distance by considering both front and rear contributions.
3. **Multiplication by Steering Angle:** The entire expression is multiplied by $\tan(u_2)$, accounting for the influence of the front wheel steering angle on the lateral motion of the vehicle.

Conclusion:
The relation $\tan(\beta(u_2)) = \frac{d_r}{l_f+l_r}\tan(u_2)$ captures the geometric aspects of the vehicle dynamics, providing a quantitative link between the slip angle, lateral distance, and steering angle.

Figure 5: Left) Schema of kinematic bicycle model of the vehicle, Right) An intuitive physical explanation for Equation 5.

[Pol+17] compares the kinematic bicycle model with an accurate 9 Degrees of Freedom vehicle model to give bounds on lateral acceleration value below which the bicycle model is consistent and valid for motion planning.

In this project, we apply minor modifications to the introduced bicycle model, without altering its core. These modifications are to make it adaptable to the optimization process which will be discussed in the following sections.

3.2 Controller

In this project, we needed to have access to the value of the actions taken by different agents in the scene. However, these values are not directly given for vehicles other than the ego in the dataset, and hence should be extracted from the trajectories directly. To this end, we use a crucial component in dynamic systems called *controller*. This latter is responsible for adjusting actions to a system so that it can follow a certain series of states.

The *actions* and the *states* can be related in many ways; one of the simplest relations is linear one.

Linear Quadratic Regulator (LQR) is a type of controller where the actions and the states follow this linear relation. It calculates a feedback gain matrix to minimize a cost function, efficiently balancing precision and control effort.

In the following, we formally state the problem solved by the LQR controller, but avoid details of how the system is solved.

Linear system a linear system is often modeled by state-space equations of the form:

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{6}$$

$$y(t) = Cx(t) \tag{7}$$

where $x(t)$ represents the state vector, $u(t)$ is the control input (the actions), $y(t)$ is the output, A is the state matrix, B is the input matrix, and C is the output matrix.

Cost Function of a Linear Controller and Optimization To guide the control of such a system, a cost function is employed to quantify the system's performance. A common choice for the cost function is a quadratic form that penalizes deviations from desired states and control inputs:

$$J = \frac{1}{2} \int_0^\infty \left(x(t)^T Q x(t) + u(t)^T R u(t) \right) \, dt \tag{8}$$

Here, Q and R are positive definite weighting matrices. The objective is to minimize this cost function over an infinite time horizon.

LQR Controller The Linear Quadratic Regulator (LQR) controller is a specific case of a control law that minimizes the cost function. It takes the form:

$$u(t) = -Lx(t) \tag{9}$$

where L is the feedback gain matrix. The matrices Q and R influence the behavior of the system. Specifically, tuning the values in Q and R allows for a trade-off between tracking accuracy and control effort.

Effect of Matrices Q and R The matrix Q weights the importance of minimizing state deviations, while R weighs the cost of control effort. Adjusting these matrices provides a means to tailor the system's behavior. Higher values in Q emphasize state accuracy, resulting in smoother trajectories. Conversely, higher values in R prioritize minimizing control effort, potentially leading to more abrupt responses.

Simplification, Advantages, and Disadvantages The LQR controller simplifies the control problem by providing an analytical solution for the feedback gain matrix L that minimizes the cost function. This allows for efficient and real-time control implementation.
However, the parameters of the controller may have to be fine-tuned when the behavior of the vehicle changes (cf. high vs low speed), or when the scale of the movement is altered (cf. transferring trajectories from world coordinate system to map coordinate system).
We propose an alternative strategy to deal with errors induced by a controller with fixed parameters.

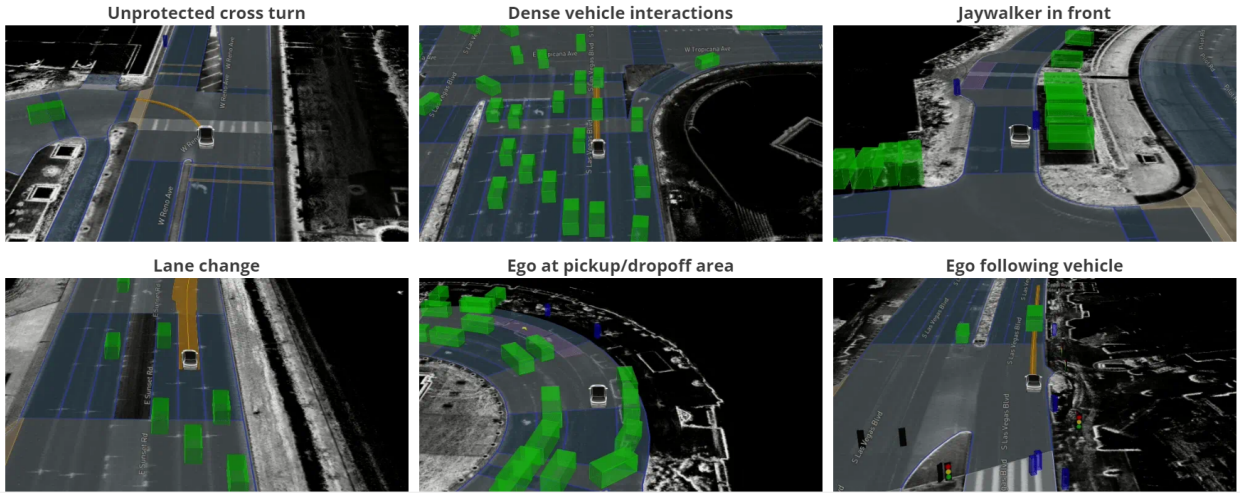


Figure 6: NuPlan provides categorization for its scenarios. A scenario can be labeled by more than one category, from over 70 classes. This categorization is believed to be useful as it also may suggest the type of behavior of its agents, and provide possible insight on the hyperparameters we fix in the controller to reconstruct the trajectories or kinematic adversary attack optimization process. This direction is considered one of the potential areas for future exploration.

4 Project's Main Components

4.1 NuPlan Simulator

As discussed in previous sections, a driving simulator closes the loop of planning by integrating the change in the environment (for example, the response of the other vehicles to the planned trajectory of the ego, and other agents) into the planning of the ego vehicle.

In this project, we opted for *NuPlan* [Cae+21] simulator, the first large-scale planning benchmark for autonomous driving; it provides scenarios of $20s$, longer than those provided in previous benchmarks (Argoverse, Lyft, Waymo) for autonomous vehicle motion prediction. The behavior of agents (ego vehicle, and other agents) can follow an ML-based (such as a trained planner), or a rule-based (such as IDM) model, and the performance of a planner can be evaluated in open-loop or closed-loop setups (which will be detailed later in this section).

Here, we provide details on the key aspects of this simulator: data collection, agent types and road elements, planning framework, metrics in different setups, and visualization tool for simulations.

4.1.1 Data Collection

The dataset provides 1200h of human driving data from 4 cities across the US and Asia with diverse traffic patterns (Boston, Pittsburgh, Las Vegas, and Singapore). Raw sensor data (120h) is released for 10% of the dataset.

4.1.2 Agent Types and Road Elements

Objects and agents in the dataset are categorized into six: 1. *Vehicle* (Includes all four or more wheeled vehicles, as well as trailers), 2. *Bicycle* (Includes bicycles, motorcycles, and tricycles), 3. *Pedestrian* (All types of pedestrians, incl. strollers and wheelchairs), 4. *Traffic cone* (Cones that are temporarily placed to control the flow of traffic), 5. *Barrier* (Solid barriers that can be either temporary or permanent), 6. *Generic object* (Animals, debris, pushable/pullable objects, permanent poles).

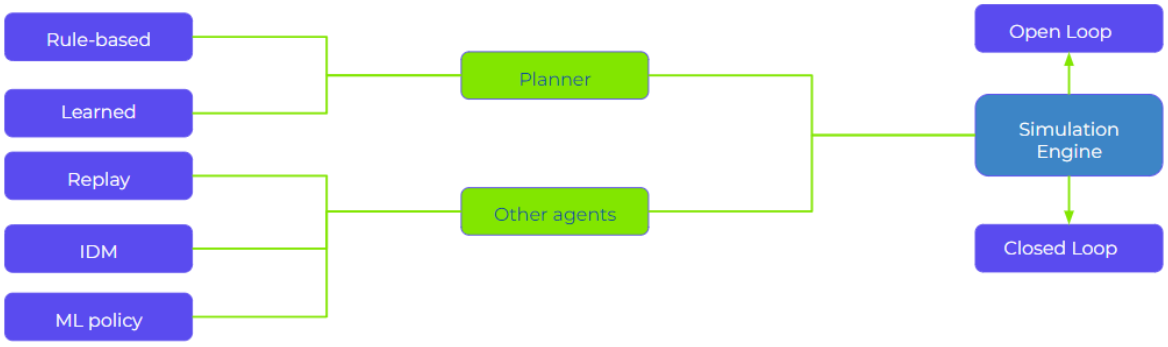


Figure 7: Two main components of the NuPlan simulator: ego vehicle and other agents. Different behavior models can be considered for each of these agents.

Detailed $2d$ high-definition maps (HD maps)annotated by humans with semantic categories, such as roads, sidewalks, crosswalks, lanes, and traffic lights are also provided.

Unlike previous datasets that provide the traffic light statuses as detected from an image-based traffic light detector, NuPlan provides a particular traffic light status for all the time steps of a scenario.

Additionally, NuPlan provides a categorization for scenarios that covers more than 70 classes (cf.Figure 6).

4.1.3 Planning Framework

NuPlan supports *open-loop*, *closed-loop interactive* and *non-interactive* simulation. Closed-loop means that the ego vehicle's perception of its past trajectory is its previously planned trajectory, as opposed to open-loop simulation, where the perception of the ego vehicle of its past trajectory corresponds to the ground truth. Hence, in a closed-loop setup, any errors in the initial stages of planning or execution can potentially propagate and accumulate over time.

Also, other agent vehicles can deviate from what was recorded in the original log in an interactive closed-loop simulation, following a behavior model such as IDM (Intelligent Driver Model).

IDM IDM is the default behavior of other agents in an interactive closed-loop setup in NuPlan, and other models that may be added on top of the simulator.

IDM implemented in NuPlan performs two functions: 1. Finding the longitudinal trajectory along centerline segments, represented as nodes of a graph, using a graph-search algorithm, 2. Driving along the planned centerline segments to avoid colliding into the front vehicle, by adjusting the current acceleration, given The current longitudinal position x, the velocity v, and the distance to the leading vehicle along the centerline s:

$$\frac{dv}{dt} = a\left(1 - \left(\frac{v}{v_0}\right)^{\delta} - \left(\frac{s^*}{s}\right)^2\right) \tag{10}$$

The acceleration limit a, target speed v_0, safety margin s^*, and exponent δ are manually selected.

The provided framework, consisting mainly of a planner for the ego vehicle and behavior model for other agents, allows training ML-based planners and is modular with respect to different components of the simulator (cf. Figure 7)

4.1.4 Metrics

The shift from short-term to long-term motion prediction and planning necessitates an adaptation of the metrics, as $\mathcal{L}2$-based metrics are not suitable for fairly evaluating long-term planning : a satisfactory planned trajectory can deviate from the trajectory given in the ground truth, if it fulfills a number of constraints such as not colliding with other agents and object, and approaching as much as possible the goal point; hence, only evaluating the planned trajectory by comparing it with the ground truth is insufficient.

Different metrics could be considered to evaluate the performance of a planner. Also, a planner can be evaluated in different setups (open-loop, ...), and hence an adapted metric is needed for each setup.

The metrics for the ***open-loop*** setup are Average Displacement Error (ADE), Final Displacement Error (FDE), Average Heading Error (AHE), Final Heading Error (FHE), and Miss Rate. ADE is defined as the average of pointwise $\mathcal{L}2$ distances between the planner trajectory and the ground-truth trajectory starting at that time up to the selected comparison horizon in the future, and AHE is its equivalent for measuring the absolute difference of heading values at each point of the planned trajectory. The Final Displacement/Heading Error measures the error at the final point of the planned trajectory. At each sampled time, the planner's trajectory is considered to have a miss, if its pointwise $\mathcal{L}2$ distance with the ground-truth trajectory is higher than a certain threshold. This assessment is performed for all sampled times in the scenario, and the miss rate is then the ratio of sampled times when the trajectory was flagged as a miss to the total number of sampled times.

To evaluate scenarios in ***Closed-loop non-reactive*** and ***Closed-loop reactive*** setup, traffic rule violation is used to measure compliance with common traffic rules. Also, the rate of collisions with other agents, the rate of off-road trajectories, the time gap to lead agents, and the time to the collision are used as metrics to quantify the safe behavior of the agent. Another aspect of assessing the planner is the rider's comfort and feasibility of a trajectory, which may be quantified by measuring the jerk, acceleration, steering rate, and vehicle oscillation. As opposed to the case of open-loop setup, in planning the objective is not to be as close as possible to the ground-truth trajectory, but to progress towards a goal waypoint on the map, which can be quantified by using $\mathcal{L}2$ distance with the goal.

4.1.5 Visualization

Along with the NuPlan simulator, the *NuBoard* dashboard is provided which is an interactive Bokeh-based simulation viewer that renders the underlying semantic map, the ego vehicle, other agents in the scene, as well as traffic lights. The statistics are also visualized with plots for the simulated scenarios.

4.2 KING: Generating Safety-Critical Driving Scenarios

The main idea of the project is based on the method proposed in [Han+22] to generate safety-critical scenarios.

An alternative approach to adding hand-tuned scenarios to induce safety-critical situations is to adversarially perturb the background traffic trajectories. This perturbation can be applied to the original trajectories in a manner that ensures the modified trajectory exhibits minimal deviation from the original trajectories, thereby preserving the essential characteristics of the latter.

The problem of finding the valid modification can be formulated as an optimization task, wherein the objective is to identify an optimal set of adjustments that satisfy the a set of constraints and

objectives.

In the following, we first discuss the objectives of the optimization and costs considered to attain those objectives; next, the optimization structure is explained, and lastly the results and conclusions of the paper are presented briefly.

4.2.1 Objectives and Losses

The objective of optimization is to induce collisions with the ego vehicle. There exists often a set of more than one modification that could induce a collision, but the simplest and most realistic collision is usually the result of the interaction of ego vehicle with the closest vehicle to it.

In order to ensure the realizability of the modified trajectories, two other constraints should be considered: $1.$ adversary agents should not collide with each other, $2.$ all the trajectories should remain in the drivable area.

Each of these objective and constraints can be formulated into a cost that should be minimized:

$$\phi_{\text{ego,col}}(S) = \min_{i=1,\dots,N} \frac{1}{T} \sum_{t=0}^{T} d(s_t^0, s_t^i) \tag{11}$$

$$\phi_{\text{adv,col}}(S) = -min(\min_{\substack{i,j=1,\dots,N \\ t=0,\dots,T}} d(s_t^i, s_t^j), \tau) \tag{12}$$

$$\phi_{\text{dev}}^{\text{adv}}(S) = \sum_{i=0}^{N} \sum_{t=0}^{T} g(s_t^i, \mathcal{M}) \tag{13}$$

Where s_t^i is the state of the ith agent at time t, T the time horizon of the scenario, τ the minimum distance between two agents in order for the repulsive potential to be active, $\mathcal{M}$ the map of the drivable area, and g a Gaussian potential applied over all agents and time steps.

Each loss respectively encourages collision with the ego, no collision between adversaries, and no deviation from the drivable area for adversary agents (if the ego vehicle strays out of the road to avoid a collision, the attack can be considered as successful).

It should be noted that the first loss is designed to apply attraction potential between the ego and *its closest adversary agent*, and the second loss only applies the repulsive force between the *closest pair of adversarial agents*.

4.2.2 Optimization Structure

The optimization is performed on an initially non-critical scenario, and ego vehicle is iteratively attacked to finally deviate from its normal path, and in the case of a successful attack eventually collide with other agents or stray out of the drivable area.

At each optimization iteration, the scenario's trajectory of adversary agents is modified, to induce such a collision. In other words, the ego vehicle, responding to these adversarial changes, modifies its planning in a closed-loop manner, but may not be able to avoid a collision even when adapting its response.

The following gives a formal formulation of this optimization process:

Let $x_t^i \in \mathcal{R}^2$, $\psi_t^i \in [0, 2\pi]$, and $v_t^i \in \mathcal{R}$ be the ground-plane position, orientation, and speed of the ith agent at time t, which was before noted as the state of the agent, s_t^i. The traffic state is the accumulation of the state of all the agents in the scene. A scenario is composed of agnet states for T

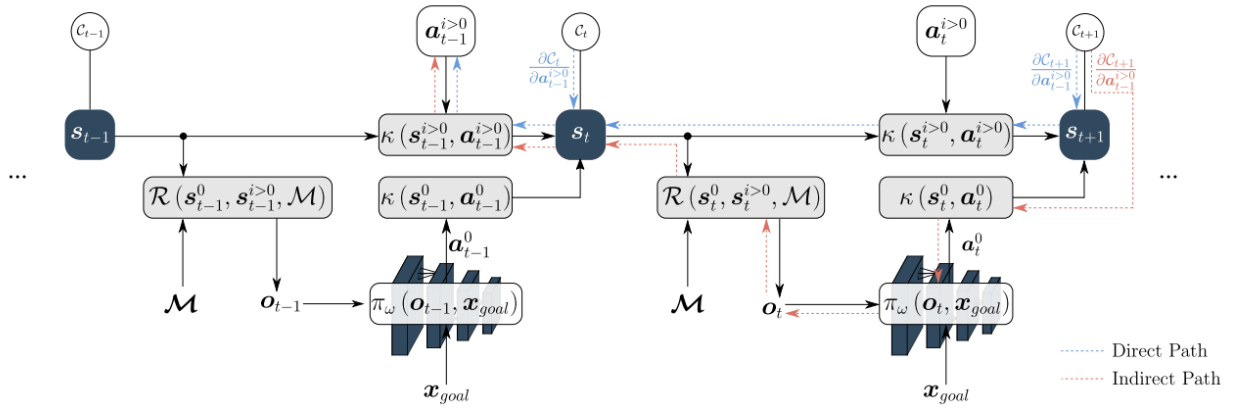


Figure 8: Optimization Structure: To enroll the simulation, actions taken by adversary agents at each time step, a_t, translate into states, s_{t+1}, via a kinematics, $s_{t+1} = \kappa(s_t, a_t)$. The actions taken by the ego vehicle is decided by the planner, $\pi\omega$, which perceives the current state of the scene by an input observation; this observation is the result of a rendering function, $\mathcal{R}$, which may be differentiable or not. To induce plausible collisions, the gradient of the cost at each time step ,with respect actions taken previous to it, can be back-propagated through a direct and indirect path. The direct path consists of the motion/kinematics model, and the indirect path passes through the planner and its rendering function. Back-propagation through the indirect path is considered to be more compute-intensive due to often complex rendering and planning functions.

time steps, the horizon of the scenario; let's call the scenario $S = \{s_t\}_{t=0}^{T}$ where s_t is the state of all agents at time step t.

Using a motion model $\mathcal{M}$, one can advance from the state t to state $t+1$ using the actions taken at time t:

$$s_{t+1} = \mathcal{M}(s_t, a_t)$$

where $a_t = \{a_{i,t}\}_{i=0}^{N}$ are the actions taken by each agent at time step t.

To update the scenario, the states of agents should be modified; however, it is a hard task to ensure plausibility when directly modifying the states. *Instead, the actions taken at each time step by each agent can be updated, and the motion model would provide a strong prior for the generated states to be plausible.*

Actions can be optimized using the discussed losses calculated over the states. This loss can be back-propagated through the motion model to then update the action parameters, which requires the motion model to be differentiable. This is indeed the case for the bicycle model. After updating the trajectory (states) of adversary agents, their states are passed as observation to the ego planner in the simulation. The observation from the state is obtained using a *rendering function*, which can be differentiable or not. In case the planner and its rendering function are differentiable, the gradient of the costs calculated on the agents states can also be back-propagated through them, to contribute to the update of adversary actions. These two back-propagation paths are shown in Figure 8 as direct and indirect paths of back-propagation.

The algorithm of this optimization is given in algorithm 1.

The above algorithm could simply be modified to perform a limited number of optimization steps, and stop in case no collision or deviation of ego has been found. It should be noted that the costs at any timestep are back-propagated to the set of actions at timesteps prior to it.

Algorithm 1: Optimizing a Scenario by Kinematics Gradients

Data: Initial trajectory, actions, and a differentiable motion model $\mathcal{M}$
Result: Plausible safety-critical scenario

If the actions are not given, estimate them using a controller $\{a_1, ..., a_T\} = \mathcal{C}(s_1, ..., s_T)$;

while *no collision or deviation from drivable area by ego* **do**
  for *time step of the simulation* **do**
    Enroll the bicycle model to obtain the next state: $s_{t+1} = \mathcal{M}(s_t, a_t)$;
    if *there is a collision* **then**
      return $\{s_1, ..., s_T\}$
    Compute the cost on the state s_{t+1}:
    $\mathcal{L}(s_{t+1}) = \phi_{\text{ego\_col}}(s_{t+1}) + \lambda\phi_{\text{adv\_col}}(s_{t+1}) + \gamma\phi_{\text{adv\_dev}}(s_{t+1})$;
    Accumulate the loss;
    Provide the current observation to the ego planner, using the currently updated state.
    Update the state of the ego from its planned trajectory.
  Back-propagate the accumulated losses through the bicycle model to update the actions at all the time steps;

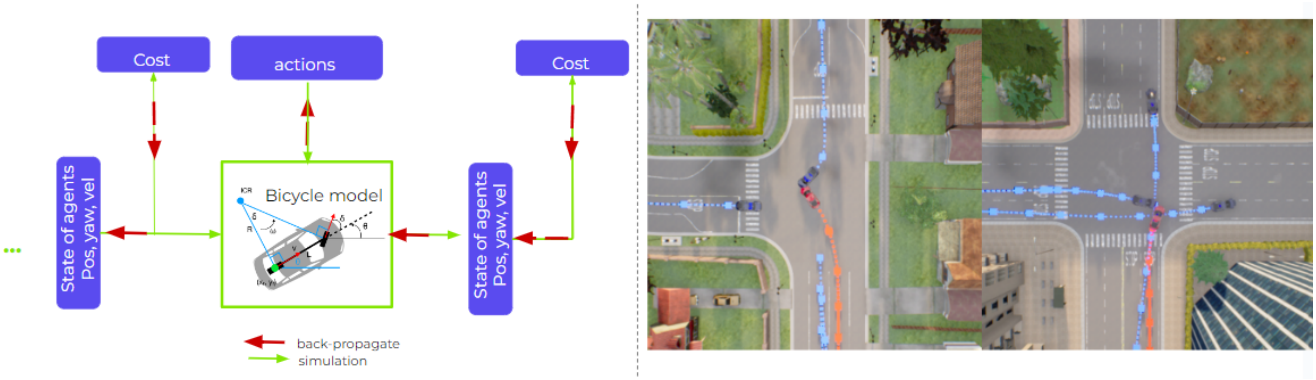


Figure 9: Left) Simplified schematic of action optimization using kinematics gradients throught the bicycle model. Right) Two examples of induced collision from [Han+22]'s experiments.

4.2.3 Experiments, Results, and Conclusion

Authors of [Han+22] test their strategy on two planners: a planner adapted from [Beh+20] and the AIM-VA model in [CPG21], that uses BEV intermediate representation instead of 2D semantics since the BEV is better correlated with vehicle kinematics than the $2D$ image domain. This planner is referred to as AIM-BEV. the second planner tested is the publicly available checkpoint1 released by the authors of TransFuser [Chi+22], that takes as input/observation the RGB image from a front-facing camera and a discretized BEV lidar-histogram. The most important difference between these two planners in the context of this project is the differentiability of their corresponding renderers: the renderer for AIM-BEV is differentiable, whereas the renderer for TransFuser is not.

Comparing the performance of optimizing through direct and indirect paths lead to the conclusion that only back-propagating through the bicycle is enough to induce collisions, and in fact using back-propagting the gradients through the planner does not increase the rate of collisions in the case of Aim-BEV (as we can only back-propagate through the planner in the case of this planner). Having

concluded the sufficiency of this optimization strategy, they perform similar experiment to stress-test TansFuser. The performance of planners of quantitatively reported in terms of Route Completion (RC), Infraction Score (IS), Driving Score (DS), and Collision Rate (CR) (refer to the supplementary material of [Han+22] for exact definition of each score).

Finally, using a privileged rule-based expert algorithm they find solutions (trajectory for the ego vehicle) to the generated safety-critical scenarios (in which the ego vehicle collides with an adversary agent), and fine-tune the planner using these solved scenarios. The improvement of planners is empirically shown by the aforementioned scores, RC, IS, DS, and CR.

4.3 Tuplan-Garage: Winning Planner of NuPlan Challenge

After the release of the Nuplan Simulator [Cae+21], a competition was organized by the team that introduced the simulator. The competition aimed to solicit planner designs suitable for open and closed-loop configurations of the simulator. The outcomes of this competition, recognizing top performers, were publicly announced at the End-to-End Autonomous Driving workshop during the Computer Vision and Pattern Recognition Conference (CVPR) 2023. In our project, focused on testing a planner, we decided to use the winning solution from this competition, presented in the paper *Parting with Misconceptions about Learning-based Vehicle Motion Planning* [Dau+23], and stress-test its performance.

In this section we explain the chosen approach presented in this latter, by first taking a close look at the features important in each setup and the misalignment of planning tasks in open- and closed-loop configurations, and moreover the misalignment of ego-forecasting and planning tasks; at the end of this section, we discuss their proposed planners, with a focus on the one designed for closed-loop simulation.

4.3.1 Misalignment of Open- and Closed-loop Evaluation, Forecasting and Planning

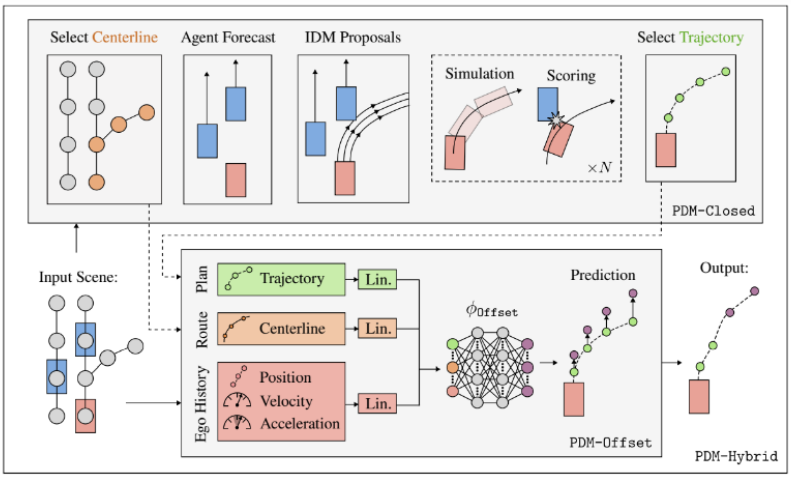
As also mentioned in previous sections, Different metrics are employed to evaluate planners in open- and closed-loop; in the former, the planner seeks to output a trajectory as close as possible to the one logged in the dataset, while in the latter, the planner tries to optimally satisfy/increase a set of metrics such as collision rate and progress.
[Dau+23] goes beyond the intuitive misalignment between these two setups, and empirically shows their negative correlation.

To empirically prove this negative correlation, they compare the performance of two planners, Predictive Driver Model open and IDM, in open-loop and closed-loop setups. Since the metrics for these two setups are not equivalent, they do this comparison by examining the improvement/decline of one metric while improving another:

IDM demonstrates robust closed-loop performance, while PDM-Open outperforms IDM in open-loop scenarios, even with only the current ego state as input. Interestingly, incorporating past ego states yields marginal improvements but results in a decline in closed-loop performance.

Also, they observe that reducing IDM's acceleration improves the scores in open-loop setup, while negatively impacting scores in the closed-loop setup.

They hence argue that there is a misalignment between the goals of ego-forecasting (open-loop) and planning (closed-loop).



PDM-Closed is illustrated on the upper side of the schema: it selects the best set of action by simulating and scoring them.

PDM-Hybrid is the combination of PDM-Closed, along with a learned model to predict the offset that should be applied on top of the PDM-Closed proposition, in order to compensate for the accumulated error in the closed-loop setup.

Figure 10: PDM-Closed and PDM-Hybrid

4.3.2 Pdm-Closed, -Open, and -Hybrid

PDM-Closed is an extension of IDM that provides it with a more flexible maneuver; it does so by by selecting the best combination of target speed and lateral offset, among five distinct speeds, namely, $20\%, 40\%, 60\%, 80\%, 100\%$ of the designated speed limit, and (±1m and 0m) offsets. This selection is done after simulating with theses parameters, and evaluating them with metrics for traffic-rule compliance, progress, and comfort.

PDM-Open is a simple multi-layer perceptron (MLP) to predict future waypoints. The MLP takes two main inputs: the centerline, extracted by the Intelligent Driver Model (IDM), and the ego history, containing the vehicle's positions, velocities, and accelerations over the past two seconds. Both the centerline and ego history are linearly projected into 512-dimensional feature vectors, concatenated, and fed into the MLP , which comprises two hidden layers, each with a dimensionality of 512. The MLP's output consists of future waypoints for an $8-$second horizon, by a frequency of 2Hz.

PDM-Hybrid combines the ideas of the above two planners, in order to merge the ego-forecasting capabilities of PDM-Open with the precise short-term actions of PDM-Closed.

It does so by predicting an offset on top of the waypoints determined by PDM-Closed. The offsets are outputted by a learned model, with a similar structure to PDM-Open, except for an extra linear projection to accommodate w_{Closed} as an additional input (cf. Figure 10).

$$w^t_{\text{Hybrid}} = w^t_{\text{Closed}} + \mathbb{1}[t > C]\phi^t_{\text{Offset}}\left(w_{\text{Closed}}, c, h\right) \tag{14}$$

It should be noted that the offset is applied only when a correction is required : when planning using the true history of the ego vehicle, no correction is applied; the C parameter in the equation is the correction horizon and stands for the time horizon of the input to the planner.

In this project, we mainly stress-test the *PDM-Closed* planner, and experimenting with other planners is left as a future direction for further exploration.

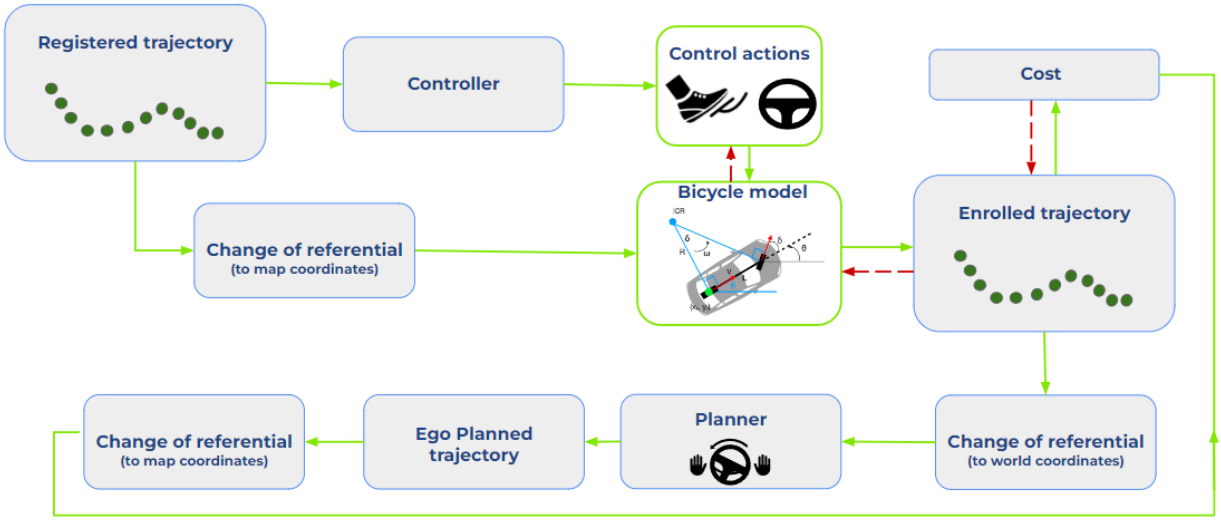


Figure 11: The main structure of the implemented code. The trajectory reconstruction process is omitted here for the sake of simplicity.

5 Implementation, Contribution, and Results

5.1 Code Structure and Implementation Details

The main code is built in a modular manner and on top of the code base of NuPlan simulator devkit. The main structure of the code is shown in Figure 11: The process is mainly composed of two parts:

1. *Action Extraction*: Extracting the actions taken by agents in the scene from their respective trajectories registered in the dataset. These actions would be the optimizable parameters later.

2. *Action Optimization*: The extracted actions are optimized to induce collisions with the ego vehicle. After each round of optimization, the simulation is restarted to capture the change in the behavior of the ego vehicle, its interaction with other agents, and whether a collision is happening.

5.2 Trajectory Reconstruction Optimization

It is expected to obtain a trajectory close to the logged trajectory(registered trajectory) when enrolling the bicycle model with the actions extracted using the LQR controller. However, there is often a discrepancy between these two components, partially because different vehicles may require distinct sets of parameters in LQR due to variations in mass, inertia, or wheelbase, impacting their dynamics. Adapting LQR parameters ensures optimal control tailored to each vehicle's unique characteristics.

Sometimes it is also desirable to have an estimation of dynamic parameters at a higher frequency, by considering short time steps (the sampling time of the trajectory); the sampling rate is limited by the autonomous vehicle stack, but it remains possible to obtain a denser sampling by interpolation.

Inspired by and similar to [Cao+22], we propose to compute an accurate and dense set of actions: Given the sampled trajectory $X = \{X_0, ..., X_T\}$, where T is the time horizon, and the sampling frequency $\mathrm{f} = 1/\Delta t$, the goal is to obtain dense actions, meaning $\{u_0, u_{\Delta t/\alpha}, u_{2\Delta t/\alpha}, ..., u_T\}$ where α is the factor multiplying the sampling rate.

The first step is to densify the trajectory waypoints, by performing an interpolation of X; this provides us with a dense trajectory $D = \{D_0, D_{\Delta t/\alpha}, D_{\Delta t/\alpha}..., X_T\}$. In the second step, the controller

extracts a dense initialization of the actions from D, $\{u_0, u_{\Delta t/\alpha}, u_{2\Delta t/\alpha}, ..., u_T\} = \mathcal{C}(D)$, where $\mathcal{C}$ is the controller.

However, this initial dense approximation of the actions may not satisfy the constraints imposed, such as accuracy, and passing through densely sampled waypoints. Hence, the third step consists of optimizing the actions with a reconstruction loss, $\mathcal{L}_{recon}(u; D_0, \mathcal{M}) = \text{MSE}(D', D)$, where $\mathcal{M}$ is the bicycle model, and D' is the dense trajectory reconstructed from estimated actions and bicycle model, $\mathcal{M}$.

Algorithm 2: Dense Trajectory Reconstruction

Data: Logged trajectory waypoints, sampling multiplier α
Result: Actions for an α times denser trajectory

Interpolate the logged trajectory at the rate $\alpha \rightarrow D = \{D_0, D_{\Delta t/\alpha}, D_{\Delta t/\alpha}..., X_T\}$;
Apply the controller on D to obtain initial dense actions $\rightarrow \{u_0, u_{\Delta t/\alpha}, u_{2\Delta t/\alpha}, ..., u_T\} = \mathcal{C}(D)$
while $\mathcal{L}_{recon} \geq threshold1$ *and optimization step* $\leq threshold2$ **do**
    Construct the trajectory from the current actions using the bicycle model $\rightarrow D' = \mathcal{M}(u, D_0)$
    Compute the reconstruction loss $\rightarrow \mathcal{L}_{recon}(u; D_0, \mathcal{M}) = \text{MSE}(D', D)$
    Back-propagate the loss through the bicycle model $\mathcal{M}$ to update the current actions u.
return u;

A simple version of this algorithm is to only optimize the actions, without increasing the sampling frequency. The optimization in this simple version can be done in different ways:

1. **Optimizing actions initialized with controller**: The actions are first extracted by the controller: at each step of the trajectory, the controller outputs the actions to be taken to reach the next waypoint in the logged trajectory (the current position of the waypoint may be different from its equivalent in the logged trajectory since the controller and the bicycle model are not perfect). When optimizing, the controller is not called again, but at every time step of the trajectory, its corresponding actions are optimized to better reach the next waypoint in the logs.

2. **Optimizing the action estimated by controller at each time step ({Optimization 1)**: Instead of starting from an action initialization for all the time steps, the controller is called alternatively with the optimization: the controller is called at each step to estimate the actions to be taken to reach the next logged waypoint, and the actions are optimized immediately; this latter ensures that the next reachable waypoint is optimized to close to its equivalent in the logs, and hence the next call to the controller probably provides a more exact estimation. There is a slight disadvantage to this approach: we will not be able to use the pre-extracted actions by the controller, and is more compute-intensive than the first strategy if the actions are already extracted. However, the computation overhead is insignificant.

3. **Optimizing the optimized action at the previous time step (Optimization 2)**: In this strategy, we only call the controller once: at the very first time step, the controller is called to obtain an estimation of the actions to be taken to reach the next waypoint in the logs. However, for the next time steps, the actions are initialized with those optimized at the previous time step and then optimized for the current time step. The assumption made is that the actions from one point in time to the next (with an interval of approximately 0.1 seconds) are not substantial.

For example, the formal algorithm of the second variation of optimization is explained in algorithm 3.

Algorithm 3: Trajectory Reconstruction Optimization

Data: Logged trajectory waypoints
Result: Actions optimized with reconstruction loss

while $\mathcal{L}_{recon} \geq threshold1$ *and optimization step* $\leq threshold2$ **do**
  for *time step* t *of the simulation* **do**
    Apply the controller on D, and the current position to obtain an estimation of the actions at the current time step $\rightarrow u_t = \mathcal{C}(D, s_{t-1})$
    Enroll the bicycle model to obtain the next state $\rightarrow s'_t = \mathcal{M}(u_t, s_{t-1})$
    Compute the reconstruction loss $\rightarrow \mathcal{L}_{recon}(s'_t, s_t) = \text{MSE}(s'_t, s_t)$
    Back-propagate the loss through the bicycle model $\mathcal{M}$ to update the action $u_t \rightarrow u'_t$. Enroll the bicycle model to obtain the *optimized* next state $\rightarrow s'_t = \mathcal{M}(u'_t, s_{t-1})$
return u;

It should be noted that in each of the cases, the reconstruction loss is composed of three terms:

$$\text{MSE}(s'_t, s_t) = \text{MSE}(p'_t, p_t) + \text{MSE}(tan\left(\frac{h'_t}{2}\right), \left(\frac{h_t}{2}\right)) + \text{MSE}(v'_t, v_t) \tag{15}$$

where p, h, and v represent respectively the position, heading, and velocity.

We compare these optimization techniques by reporting the reconstruction errors across 6 scenarios in Table 1. The columns of Table 1 represent:

1. **MSE position**: the sum of position squared errors across all agents at all time steps (intersection with the time interval of the presence of an agent in the scene) of the simulation.
2. **MSE** $tan(\frac{yaw}{2})$: the sum of $tan(\frac{yaw}{2})$ squared errors across all agents at all time steps (intersection with the time interval of the presence of an agent in the scene) of the simulation.
3. **MSE** $tan(\frac{yaw}{2})$: the sum of velocity squared errors across all agents at all time steps (intersection with the time interval of the presence of an agent in the scene) of the simulation.
4. **Initial Col**: the number of collisions between adversary agents when using the original states from the dataset. It should be noted that there is a slight discrepancy between the real number of collisions and the reported number of collisions since the collision detection function is simplified.
5. **After BM Cold**: the number of collisions between adversary agents after extraction of their actions, optimizing according to the indicated technique, and enrolling the bicycle model to obtain agents states.
6. **Collision**: whether a collision happens in the reconstruction.

5.3 Exploring Different Loss Combinations

5.3.1 Deviating from Drivable Area Loss

We propose an alternative to the loss proposed in [Han+22] to discourage agents from deviating to the drivable area: Instead of penalizing for the overlap of a Gaussian function centered around each agent

at each time step with the undrivable area, we propose to pre-compute a loss, and more precisely gradient direction with respect to each position in the map (cf. Figure 12); This pre-computation decreases the computation time of this loss significantly.

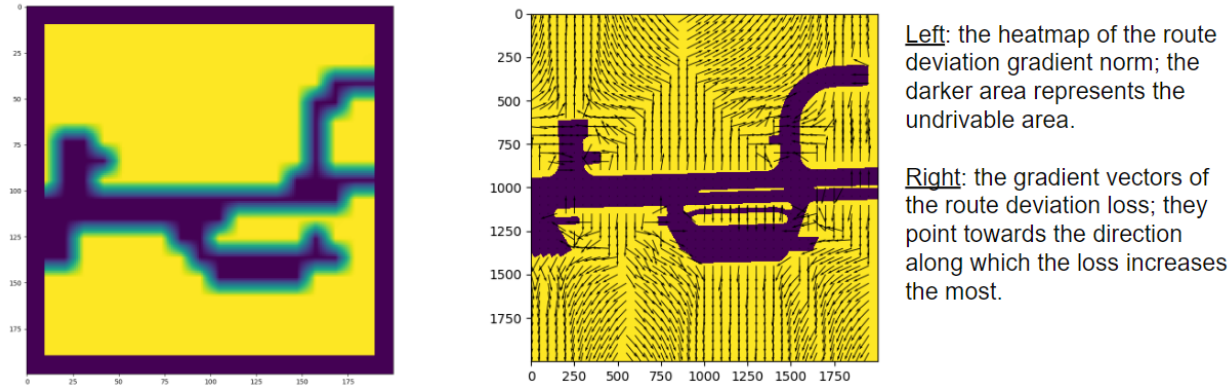


Figure 12: gradient norm and vector heatmaps of the time-efficient route deviation loss

5.3.2 Collision Loss (Multi-Agent Collision Loss)

The collision loss proposed in [Han+22] encourages the closest agent to the ego at each step of the simulation (this closest agent can indeed change), to approach the ego vehicle. Hence, only the actions of a limited set of agents are optimized adversarially.

This design of loss might be a limitation in creating uncommon and irregular adversary scenarios. To diversify the set of scenes where the ego collides with other vehicles, we may consider having more than one closest agent to the ego optimized.

One simple variation of this idea consists of optimizing all the agents present in the scene to be encouraged to collide with the ego vehicle. On the code side, this modification is only a minimum operation apart from the original version, and the back-propagation overhead is negligible.

Of course, the difference with [Han+22]'s collision loss is that adversary agents become more prone to colliding with each other; to alleviate this problem, we propose three different strategies to modify the trajectory of *non-colliding* agents after the collision happens. We discuss these strategies in the next subsection.

5.3.3 Optimizing vs Calling the Planner

The motivation behind considering this variation was to reduce the time needed to induce a collision: calling the planner and unfolding its output is a costly operation since it involves calling a controller and a bicycle model (that are integrated into the NuPlan simulator) after obtaining the output trajectory of the planner. This overhead in time is also partially due to the obligatory change of device between GPU (the optimization operations), and CPU (the controller and the motion model in NuPlan).

As a solution, we propose to only call the planner after several iterations of optimization over the actions, while considering the retained trajectory of the ego as its position.

For a more clear illustration of this strategy, refer to line 14 and Figure 13.

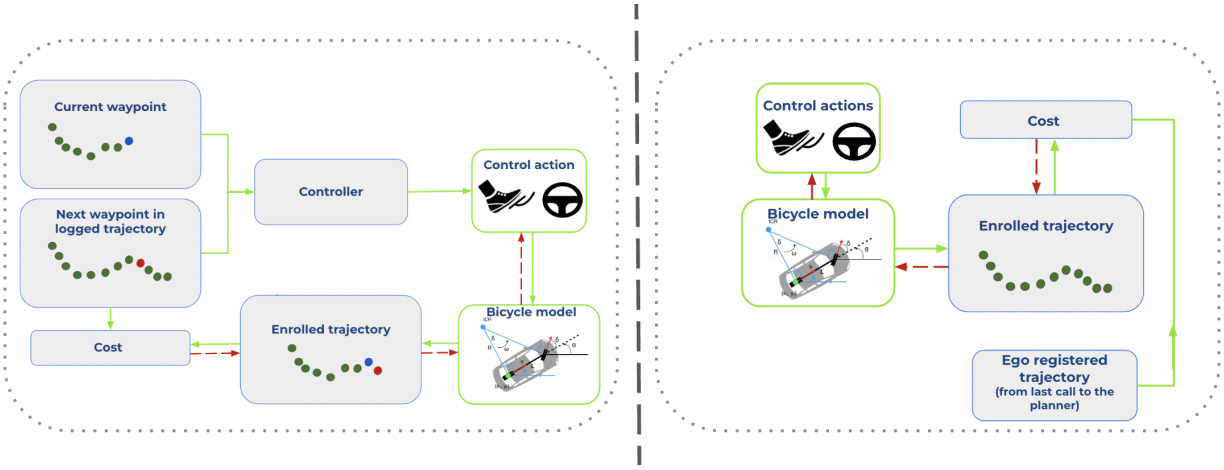


Figure 13: Left) the schema of reconstruction optimization, involving the controller, differentiable bicycle model, and the reconstruction losses. this particular schema corresponds to Optimization 1. Right) the schema corresponding to the strategy discussed in subsubsection 5.3.3, when optimizing several iterations without calling the planner.

5.4 Final Trajectory of Non-colliding Agents

Optimally, the adversarial optimization of actions leads to a collision. Let us assume that this collision happens between the ego vehicle and the agent 1. Depending on the optimization strategy and the losses involved, the actions of agents other than 1 are also modified during the optimization. We have several options on how to modify the trajectory of those agents:

1. **Back to before bicycle model:** Using the ground truth states of other vehicles in the simulation.
2. **Back to after bicycle model:** Resetting the actions to their unoptimized version, unrolling the bicycle, and using their unrolled states in the simulation.
3. **Stopping collider:** Only stopping the agent 1 that has collider with the ego vehicle, and enrolling the bicycle model for other agents to obtain their states in the simulation.

Each of these modifications has certain advantages: the first strategy is useful when there is not much interaction between the collider agent, agent 1, and other vehicles, or when we use a collision loss that affects all the adversary agents instead of only the closest one. The third strategy is useful when including the adversary repulsion loss into the optimization, and provides insights on the effects of this loss.

Algorithm 4: Time-efficient Optimization

Data: Initial trajectory, actions, a differentiable motion model $\mathcal{M}$, and jump value, the number of optimization iteration per each call of the planner for the whole simulation.
Result: Plausible safety-critical scenario

If the actions are not given, estimate them using a controller $\{a_1, ..., a_T\} = \mathcal{C}(s_1, ..., s_T)$;
Optimization_iteration = 0
while *no collision or deviation from drivable area by ego* **do**
  for *time step of the simulation* **do**
    Enroll the bicycle model to obtain the next state: $s_{t+1} = \mathcal{M}(s_t, a_t)$;
    if *there is a collision* **then**
      return $\{s_1, ..., s_T\}$
    Compute the cost on the state s_{t+1}:
    $\mathcal{L}(s_{t+1}) = \phi_{\text{ego\_col}}(s_{t+1}) + \lambda\phi_{\text{adv\_col}}(s_{t+1}) + \gamma\phi_{\text{adv\_dev}}(s_{t+1})$;
    Accumulate the loss;
    if *Optimization_iteration % jump == 0* **then**
      Provide the current observation to the ego planner, using the currently updated state.
      Update the state of the ego from its planned trajectory.
  Optimization_iteration += 1
  Back-propagate the accumulated losses through the bicycle model to update the actions at all the time steps;

5.5 Experiments and Results

We explore different combinations of loss in the experiments (cf. Table 2), across 6 scenarios. Experiments including route deviation loss (cf. subsubsection 5.3.1) are only performed across 2 scenes where the trajectory of the colliding agent passes through the undrivable area before the collision. In the conducted experiments, we mainly combine Multi-agent collision loss (cf. subsubsection 5.3.2) with 'Back to before bicycle model' (cf. item 1) strategy, and king's collision loss (cf. Equation 13) with 'Stopping collider' (cf.item 3) strategy.

The experiments are performed using the actions extracted from the controller and optimized by the optimization 1 strategy. Different columns in Table 2 are:

1. **King Col**: the king collision cost function (attraction of the closest agent at each step to the ego vehicle), **Multi-agent Col**: the cost to encourage the attraction of all the vehicles to the moving ego(cf. subsubsection 5.3.2).
2. **Adv Col**: the cost to encourage repulsion force between the adversary agents, and its weight.
3. **Route Dev**: the cost to avoid adversary agents deviate to undrivable area.
4. **Time to Col**: the optimization time composed of forward in bicycle model+checking the collision at every step of the simulation+computation of cost+backpropagation of the cost until the induced collision.
5. **Opt Iters**: number of optimization iterations performed before inducing the collision.

6. **Out of Drivable**: a measure to quantify how much of the trajectory of the colliding agent has been out of the drivable area. More precisely, for each point of the trajectory, its distance to the closest point in the drivable area is calculated; these values are summed across the waypoints of the trajectory and reported.

7. **Adv Cols**: number of collisions among adversary agents after the optimization + end strategy (cf. subsection 5.4).

8. **Adv Cols**: number of collisions among adversary agents after extraction of actions, optimizing the reconstruction of trajectories, and enrolling the bicycle model.

9. **Collision**: whether the collision is achieved through adversarial optimization.

10. **After Col strategy**: how the state of non-colliding agents is modified at the end of optimization (cf. subsection 5.4).

We can observe from the results that our proposed route deviation loss is a simpler, more efficient, and functioning replacement to the loss proposed in [Han+22].

Choosing to reset the state of non-colliding agents helps to overall stay closer to the initial scenario, but may lead to an unsuccessful collision compared to when the states were obtained through the bicycle model and optimized actions.

The collision loss between adversary agents (repulsion) does not seem to greatly contribute to the reduction of the number of collisions between adversaries, and more investigation is needed to improve or replace this loss.

It is necessary to perform more experiments and gain more insight to interpret the results, but we can claim that the technique of adversarially optimizing actions through the bicycle model works overall and is more interpretable than methods involving deep networks; however, there remains still much room to improve upon our work.

6 Future Work

The following represent potential directions for future work:

6.1 Extension of the Current Study

- Conducting extensive additional experiments on NuPlan scenarios. - Explore the optimization of scenes based on their categorization in NuPlan; possibly find an adaptation of hyperparameters for specific categories.

- The analysis of collision loss among adversary agents and its weighting within the overall loss function shall be examined.

- In the current version of this method, the interaction between adversary agents is only done through adversary collision loss. It may be possible to enable adversary agents to interact in a more controllable manner, for example by adding an IDE as an extra supervision to avoid collisions and ensure realistic behavior.

6.2 Optimization and Efficiency

Future work may focus on optimizing and improving the efficiency of the current code. This could involve: - Improving the implementation of the code. Exploring ways of keeping the modular structure,

but at the same time reducing the number of transfers between CPU and GPU.

Scene/Strategy	MSE position	MSE $tan(\frac{yaw}{2})$	MSE velocity	Initial Col	After BM Col	Collision
scene1						
Controller + BM	681143.25	7.23	24561.67	32	661	-
optimization1	3211.40	165.38	5663.24	32	0	-
optimization2	660.26	0.45	275.20	32	80	-
scene2						
Controller + BM	218982.88	656.57	20888.38	683	1539	✓
optimization1	9749.30	679.87	2143.04	683	458	✓
optimization2	689.28	666.27	122.24	683	459	✓
scene3						
Controller + BM	7958.14	71.05	1076.68	2	497	-
optimization1	805.99	77.52	175.36	2	128	-
optimization2	2138.64	77.65	329.44	2	190	-
scene4						
Controller + BM	82147.18	4.66	4708.01	534	755	-
optimization1	2478.12	0.75	303.65	534	929	-
optimization2	6775.09	1.10	569.85	534	1107	-
scene5						
Controller + BM	5728.74	8.10	1479.86	901	903	-
optimization1	254.14	5.54	40.90	901	952	-
optimization2	207.86	6.58	31.57	901	935	-
scene6						
Controller + BM	1139.28	0.93	234.14	36	64	-
optimization1	274.02	1.60	73.12	36	0	-
optimization2	271.95	1.59	21.03	36	0	-

Table 1: Trajectory reconstruction optimization experiments across 6 scenes from NuPlan simulator. token of scene1: 1f151e15c9cf5c81, scene2: 45a3338a890250de, scene3: 3318937ef0c65127, scene4:bd93ed4bd08b5115, scene5: c5adbbbfc23a5113, scene6: e3f8535c00435041. Total of 166 agents across all the scenes.

Scene	King Col	Multi-agent Col	Adv Col/w	Route Dev	Time to Col	Opt Iters	Out of Drivable	Adv Cols	Initial Col	Collision	After Col strategy
scene1	✓				11.65	1	0	284	0	✓	stop
scene1	✓		✓2		19.53	2	0	533	0	✓	stop
scene1		✓			13.55	1	0	12	0	-	back
scene1		✓	✓2		26.33	1	0	32	0	✓	back
scene1		✓	✓2		21.98	1	0	1535	0	✓	stop
scene2	✓				1.05	0	0	410	458	✓	stop
scene2	✓		✓2		0.098	0	0	410	458	✓	stop
scene2		✓			1.01	0	1.01	701	458	✓	back
scene2		✓	✓2		1.69	0	0	701	458	✓	back
scene2		✓	✓2		1.35	0	0	410	458	✓	stop
scene3	✓				119.14	14	1296.00	114	128	✓	stop
scene3	✓		✓2		127.38	14	1296.00	0	128	✓	stop
scene3		✓			35.65	3	63.81	2	128	✓	back
scene3		✓	✓2		45.86	3	139.15	2	128	✓	back
scene3		✓	✓2		41.65	3	139.15	406	128	✓	stop
scene3	✓			✓	611.31	64	1924.48	114	128	✓	stop
scene3		✓		✓	376.21	37	0	2	128	✓	stop
scene4	✓				8.43	1	0	1409	929	✓	stop
scene4	✓		✓2		11.03	1	0	1386	929	✓	stop
scene4		✓			12.69	1	0	620	929	✓	back
scene4		✓	✓2		14.20	1	0	622	929	✓	back
scene4		✓	✓2		12.31	1	0	2207	929	✓	stop
scene5	✓				8.12	1	0	894	952	✓	stop
scene5	✓		✓2		8.15	1	0	894	952	✓	stop
scene5		✓			9.72	1	0	895	952	✓	back
scene5		✓	✓2		10.28	1	0	895	952	✓	back
scene5		✓	✓2		8.96	1	0	1000	952	✓	stop
scene6	✓				64.85	7	6566.64	0	0	✓	stop
scene6	✓		✓2		65.37	7	6566.64	0	0	✓	stop
scene6		✓			16.04	1	25041.0	18	0	✓	back
scene6		✓	✓2		20.19	1	25041.0	18	0	✓	back
scene6		✓	✓2		15.31	1	25041.0	56	0	✓	stop
scene6	✓			✓	–	200	–	36	0	-	stop
scene6	✓		✓2	✓	1428.78	126	29.20	0	0	✓	stop
scene6		✓		✓	997.28	104	2365.5	18	0	✓	back
scene6		✓	✓2	✓	1308.34	116	335.01	18	0	✓	back
scene6		✓	✓2	✓	1357.49	116	335.01	168	0	✓	stop

Table 2: Adversarial optimization experiments across 6 scenes from NuPlan simulator. token of scene1: 1f151e15c9cf5c81, scene2: 45a3338a890250de, scene3: 3318937ef0c65127, scene4:bd93ed4bd08b5115, scene5: c5adbbbfc23a5113, scene6: e3f8535c00435041

References

[Kae+04] Nico Kaempchen et al. “IMM object tracking for high dynamic driving maneuvers”. In: *IEEE Intelligent Vehicles Symposium, 2004*. IEEE. 2004, pp. 825–830.

[Pol+07] Aris Polychronopoulos et al. “Sensor fusion for predicting vehicles’ path for collision avoidance systems”. In: *IEEE Transactions on Intelligent Transportation Systems* 8.3 (2007), pp. 549–562.

[BF08] Alexander Barth and Uwe Franke. “Where will the oncoming vehicle be the next second?” In: *2008 IEEE Intelligent Vehicles Symposium*. IEEE. 2008, pp. 1068–1073.

[Bro+08] Gabriel J Brostow et al. “Segmentation and recognition using structure from motion point clouds”. In: *Computer Vision–ECCV 2008: 10th European Conference on Computer Vision, Marseille, France, October 12-18, 2008, Proceedings, Part I 10*. Springer. 2008, pp. 44–57.

[LTA08] Panagiotis Lytrivis, George Thomaidis, and Angelos Amditis. “Cooperative path prediction in vehicular environments”. In: *2008 11th International IEEE Conference on Intelligent Transportation Systems*. IEEE. 2008, pp. 803–808.

[SRW08] Robin Schubert, Eric Richter, and Gerd Wanielik. “Comparison and evaluation of advanced motion models for vehicle tracking”. In: *2008 11th international conference on information fusion*. IEEE. 2008, pp. 1–6.

[AN09] Samer Ammoun and Fawzi Nashashibi. “Real time trajectory prediction for collision risk estimation between vehicles”. In: *2009 IEEE 5Th international conference on intelligent computer communication and processing*. IEEE. 2009, pp. 417–422.

[BWB09] Thomas Batz, Kym Watson, and Jurgen Beyerer. “Recognition of dangerous situations within a cooperative group of vehicles”. In: *2009 IEEE Intelligent Vehicles Symposium*. IEEE. 2009, pp. 907–912.

[Her+09] Christoph Hermes et al. “Long-term vehicle motion prediction”. In: *2009 IEEE intelligent vehicles symposium*. IEEE. 2009, pp. 652–657.

[TK10] Peter Trautman and Andreas Krause. “Unfreezing the robot: Navigation in dense, interacting crowds”. In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2010, pp. 797–803.

[Gei+13] Andreas Geiger et al. “Vision meets robotics: The kitti dataset”. In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1231–1237.

[Goo+14] Ian Goodfellow et al. “Generative adversarial nets”. In: *Advances in neural information processing systems* 27 (2014).

[LVL14] Stéphanie Lefèvre, Dizan Vasquez, and Christian Laugier. “A survey on motion prediction and risk assessment for intelligent vehicles”. In: *ROBOMECH journal* 1.1 (2014), pp. 1–14.

[Cor+16] Marius Cordts et al. “The cityscapes dataset for semantic urban scene understanding”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 3213–3223.

[AKM17] Matthias Althoff, Markus Koschi, and Stefanie Manzinger. “CommonRoad: Composable benchmarks for motion planning on roads”. In: *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2017, pp. 719–726.

[Dos+17] Alexey Dosovitskiy et al. “CARLA: An open urban driving simulator”. In: *Conference on robot learning*. PMLR. 2017, pp. 1–16.

[Neu+17] Gerhard Neuhold et al. “The mapillary vistas dataset for semantic understanding of street scenes”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 4990–4999.

[Pol+17] Philip Polack et al. “The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles?” In: *2017 IEEE intelligent vehicles symposium (IV)*. IEEE. 2017, pp. 812–818.

[Vel+17] Petar Veličković et al. “Graph attention networks”. In: *arXiv preprint arXiv:1710.10903* (2017).

[Cho+18] Yukyung Choi et al. “KAIST multi-spectral day/night data set for autonomous and assisted driving”. In: *IEEE Transactions on Intelligent Transportation Systems* 19.3 (2018), pp. 934–948.

[Hua+18] Xinyu Huang et al. “The apolloscape dataset for autonomous driving”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 2018, pp. 954–960.

[Sha+18] Shital Shah et al. “Airsim: High-fidelity visual and physical simulation for autonomous vehicles”. In: *Field and Service Robotics: Results of the 11th International Conference*. Springer. 2018, pp. 621–635.

[Cha+19] Ming-Fang Chang 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.

[Che+19] Zhengping Che et al. “D ̂2-city: a large-scale dashcam video dataset of diverse traffic scenarios”. In: *arXiv preprint arXiv:1904.01975* (2019).

[Hua+19] Yingfan Huang et al. “Stgat: Modeling spatial-temporal interactions for human trajectory prediction”. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 6272–6281.

[LYC19] Xin Li, Xiaowen Ying, and Mooi Choo Chuah. “Grip: Graph-based interaction-aware trajectory prediction”. In: *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE. 2019, pp. 3960–3966.

[Pat+19] Abhishek Patil et al. “The h3d dataset for full-surround 3d multi-object detection and tracking in crowded urban scenes”. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 9552–9557.

[Beh+20] Aseem Behl et al. “Label efficient visual abstractions for autonomous driving”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 2338–2345.

[Cae+20] Holger Caesar et al. “nuscenes: A multimodal dataset for autonomous driving”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 11621–11631.

[Gao+20] Jiyang Gao 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. 11525–11533.

[Mes+20] Kaouther Messaoud et al. “Attention based vehicle trajectory prediction”. In: *IEEE Transactions on Intelligent Vehicles* 6.1 (2020), pp. 175–185.

[Sun+20] Pei Sun et al. “Scalability in perception for autonomous driving: Waymo open dataset”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 2446–2454.

[Tu+20] James Tu et al. “Physically realizable adversarial examples for lidar object detection”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 13716–13725.

[Yu+20] Fisher Yu et al. “Bdd100k: A diverse driving dataset for heterogeneous multitask learning”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 2636–2645.

[Alb+21] Stefano V Albrecht et al. “Interpretable goal-based prediction and planning for autonomous driving”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 1043–1049.

[Cae+21] Holger Caesar et al. “nuplan: A closed-loop ml-based planning benchmark for autonomous vehicles”. In: *arXiv preprint arXiv:2106.11810* (2021).

[CPG21] Kashyap Chitta, Aditya Prakash, and Andreas Geiger. “Neat: Neural attention fields for end-to-end autonomous driving”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 15793–15803.

[Ngi+21] Jiquan Ngiam et al. “Scene transformer: A unified architecture for predicting multiple agent trajectories”. In: *arXiv preprint arXiv:2106.08417* (2021).

[Wan+21] Jingkang Wang et al. “Advsim: Generating safety-critical scenarios for self-driving vehicles”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 9909–9918.

[Bah+22] Mohammadhossein Bahari et al. “Vehicle trajectory prediction works, but not everywhere”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 17123–17133.

[Cao+22] Yulong Cao et al. “Advdo: Realistic adversarial attacks for trajectory prediction”. In: *European Conference on Computer Vision*. Springer. 2022, pp. 36–52.

[Chi+22] Kashyap Chitta et al. “Transfuser: Imitation with transformer-based sensor fusion for autonomous driving”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2022).

[Han+22] Niklas Hanselmann et al. “King: Generating safety-critical driving scenarios for robust imitation via kinematics gradients”. In: *European Conference on Computer Vision*. Springer. 2022, pp. 335–352.

[Hua+22] Yanjun Huang et al. “A survey on trajectory-prediction methods for autonomous driving”. In: *IEEE Transactions on Intelligent Vehicles* 7.3 (2022), pp. 652–674.

[Rem+22] Davis Rempe et al. “Generating useful accident-prone driving scenarios via a learned traffic prior”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 17305–17315.

[Vin+22] Eugene Vinitsky et al. “Nocturne: a scalable driving benchmark for bringing multi-agent learning one step closer to the real world”. In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 3962–3974.

[Dau+23] Daniel Dauner et al. “Parting with Misconceptions about Learning-based Vehicle Motion Planning”. In: *arXiv preprint arXiv:2306.07962* (2023).

[Din+23] Wenhao Ding et al. “CausalAF: causal autoregressive flow for safety-critical driving scenario generation”. In: *Conference on Robot Learning*. PMLR. 2023, pp. 812–823.

[Gul+23] Cole Gulino et al. “Waymax: An Accelerated, Data-Driven Simulator for Large-Scale Autonomous Driving Research”. In: *arXiv preprint arXiv:2310.08710* (2023).

7 Visualizations

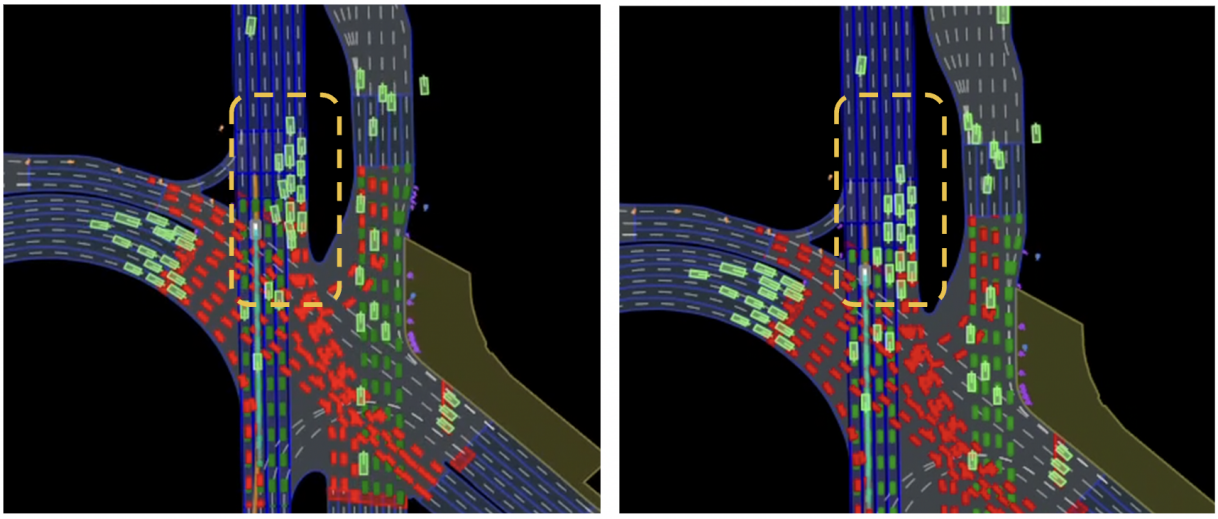


Figure 14: An illustration of the reconstruction optimization: left) the actions of agents are extracted by the controller and directly enrolled with the bicycle model. right) the actions of agents are optimized using the Optimization1. We can observe that several collisions happen between agents on the left case due to the imprecise estimation of their taken actions.

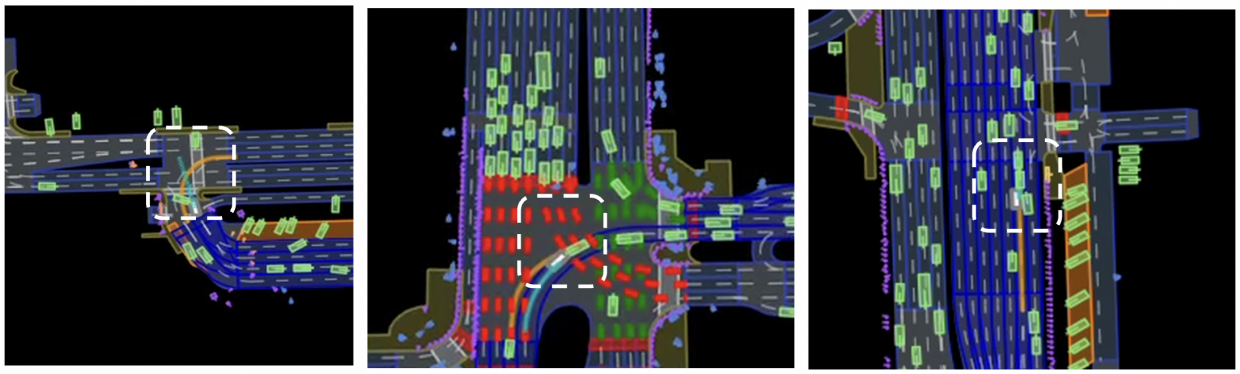


Figure 15: Examples of induced collisions in different scenes. The colliding agent in these scenes accelerates fast toward the ego vehicle from the very beginning. This latter may be considered unrealistic behavior, and solutions to this issue shall be explored in future work.

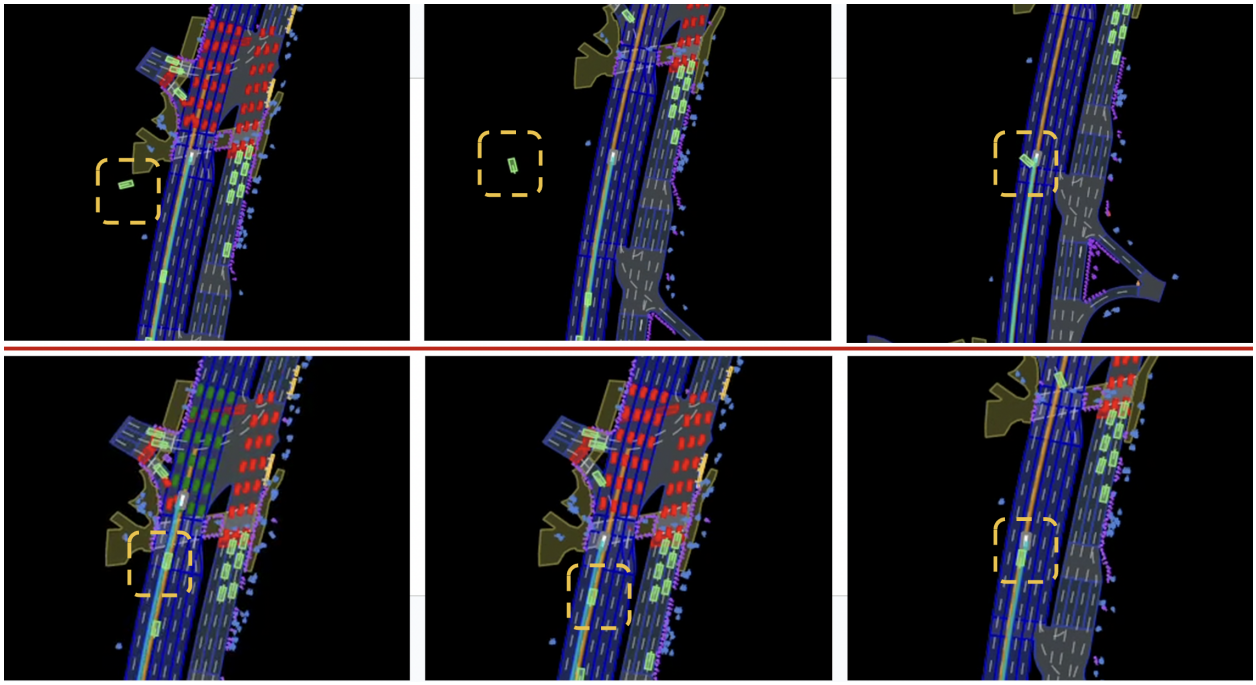


Figure 16: The three stages of a trajectory before collision are shown on each line, from the same scene. In the upper experiment, no route deviation loss is used, and the colliding adversary takes a turn into the undrivable area to then collide with the ego. In the bottom case, employing the route deviation loss helps to keep the adversary inside the drivable area; it instead decelerates in order to make a collision with the ego vehicle.