Learning driving policies for realistic traffic simulations

Yann Koeberle

UNIVERSITÉ PARIS-EST CRÉTEIL

ÉCOLE DOCTORALE MSTIC

PhD dissertation presented to obtain the degree of

Doctor of Philosophy

Learning driving policies for realistic traffic simulations

---

by

Yann Koeberlé

Defended on 14 December 2022

Christophe SABOURIN	Directeur de thèse	Université Paris-Est Créteil
Dzmitry TSISHKOU	Invité	Huawei Technologies France
Hossam AFIFI	Rapporteur	Télécom SudParis
David FILLIAT	Rapporteur	ENSTA Paris
Samia BOUCHAFA	Présidente du jury	Université Paris-Saclay
Moussa BOUKHNIFER	Examinateur	Université de Lorraine
Kurosh MADANI	Examinateur	Université Paris-Est Créteil

LISSI – Laboratory of Images, Signals and Intelligent Systems – EA 3956 University Paris-Est Créteil, IUT Senart-Fontainebleau, 36-37 Rue Georges Charpak, F-77127 Lieusaint

IoV team, Paris Research Center, Huawei Technologies France

Présentation

Le domaine du véhicule autonome a connu un développement rapide au cours des dernières décennies avec l'essor conjoint des technologies d'apprentissage profond et du calcul à grande vitesse. Plusieurs niveaux d'automatisation de la conduite ont été définis pour préparer le déploiement du véhicule autonome (VA) dans le monde réel. Les derniers avancements considèrent un VA évoluant dans des scénarios urbains hautement interactifs avec divers usagers de la route sans supervision humaine. Actuellement, les SDV ne rivalisent avec les conducteurs humains que sur des scénarios restreints tels que les autoroutes et la conduite urbaine reste un défi majeur. La plupart des prototypes avancés ont tendance à être trop prudents en milieu urbain en raison du manque de compréhension de la dynamique des agents du trafic. En effet, la diversité des comportements combinée à l'incapacité de communiquer directement avec les autres usagers de la route pour connaître leur intention rend la prise de décision très compliquée. En conséquence, cela génère des embouteillages ou des prises de risques des conducteurs humains qui effectuent parfois des dépassements inappropriés. Même si plus de 2 millions de Km de données de conduite ont déjà été collectées par la société Waymo, cela reste encore insuffisant pour traiter tous les cas de figure et atteindre une autonomie complète. De plus, il est difficile de garantir que le SDV se comporte en toute sécurité même sur une zone géographique délimitée car le nombre de scénarios à tester est exponentiel en nombre d'agents, topologies du réseau routier, style de conduite etc. Une couverture complète de l'espace des scénarios est impossible surtout si la collecte de données réelles est nécessaire pour chacun d'eux. Toutefois les catégories de scénarios critiques peuvent être identifiées et plusieurs instances peuvent être générées artificiellement de manière automatique. La simulation de trafic apparaît comme un outil prometteur pour le développement incrémental du VA car elle permettrait de tester de façon interactive des modèles sur des scénarios arbitraires à faible coût. La simulation permet de rejouer des scénarios pour mieux comprendre l' origine des erreurs de décision ou d'interprétation, tout en faisant varier les paramètres de la scène. Lors de mise à jour du système, il est également important de pouvoir relancer un ensemble exhaustif de tests pour vérifier que les modifications apportées n'aient pas de con-

séquences indésirables. La simulation apparaît donc essentielle au développement incrémental d'un système de navigation autonome dont l'utilisation est destinée à être progressivement étendue une fois sa sûreté garantie. Les garanties de sécurité peuvent, elles aussi, être apportées de manière quantitative par la simulation, ce qui permet d'envisager la mise en place de contrat d'assurance. Cependant, les cas d'utilisation pratiques nécessitent que le simulateur fournisse des interactions sociales réalistes entre tous les agents du trafic, de sorte que le SDV en cours d'évaluation puisse être testé comme dans des conditions réelles. Cette exigence est capitale car l'évaluation des performances en boucle fermée n'a de valeur que si les agents du trafic simulés se comportent comme le ferait un agent humain. De plus, le simulateur de trafic devrait également fournir des interactions cohérentes dans des situations inhabituelles et rares où les bons réflexes doivent être adoptés. En cas d'urgence, l'agent de conduite est censé s'adapter et éviter les collisions tout en respectant les règles de circulation. La capacité à extrapoler le comportement des situations pour lesquelles on ne dispose pas nécessairement de démonstration de référence est l'un des défis majeurs de la simulation de trafic. Dans ce travail, nous proposons une approche pour apprendre des politiques de navigation de type humaine pour la simulation de trafic capable de généraliser des comportements de conduite sûrs et réalistes pour de nouveaux scénarios interactifs. Nous fondons notre méthode sur la combinaison de l'apprentissage par imitation (AI) et de l'apprentissage par renforcement (AR) pour tirer parti des données de conduite expertes ainsi que des connaissances du domaine. Nous développons d'abord une architecture hiérarchique pour simplifier le processus d'apprentissage de la tâche de conduite. Notre politique de conduite consiste en la combinaison d'un module de routage traditionnel et d'un planificateur de manœuvres implémenté avec un réseau de neurones qui génère une trajectoire à court terme convertie en une séquence de commandes par une sous-couche de contrôle. Le module de routage s'appuie sur le graphe du réseau routier et sur la destination finale pour déterminer un chemin de référence indépendant du trafic tandis que le planificateur de manœuvres génère une trajectoire compatible à la fois avec le chemin de référence et le contexte local de scène. Afin de guider la prise de décision, nous proposons un espace d'action en coordonnées curvilignes basé sur le chemin de référence de sorte que le planificateur de manœuvres puisse spécifier la trajectoire à suivre en termes de déplacement longitudinal et latéral. Nous proposons également un espace d'observation sous forme vectorielle avec plusieurs types de champs de façon à accélérer la simulation par rapport à des formats plus conventionnels comme des images . L'interprétation du contexte de la scène est rendue possible par différentes architecture de réseau de neurones que nous avons conçu pour encoder l'observation dans un espace latent que le planificateur de manœuvre peut ensuite convertir. Nous montrons que ces réseaux peuvent apprendre avec précision

à planifier des trajectoires à court terme en boucle ouverte de la même manière qu'un conducteur humain, mais nous remarquons que les performances en boucle fermée pour de longs horizons de simulation sont limitées. Ce problème est lié au décalage distributionnel (distributionnal shift) puisqu'en boucle fermée, l'agent, en prenant des actions, modifie progressivement sa distribution d'état par rapport à celle de l'expert d'origine ce qui conduit à des erreurs de décisions qui se cumulent sur les nouveaux états visités. Pour limiter la dérive de trajectoire, nous proposons d'intégrer au planificateur une structure autorégressive, de façon à ce que chaque action du plan ait une influence sur le futur état latent qui conditionnera la génération de l' action suivante. Nous montrons que grâce à l' ajout de données synthétiques de simulation, la version auto-régressive du planificateur de trajectoire permet de suivre plus étroitement la trajectoire de l' expert en simulation que la première version qui modélise la séquence d'actions du plan comme une suite de variables aléatoires mutuellement indépendantes. Un autre avantage notable est que la trajectoire est généralement plus régulière car le modèle est mieux capable d'intégrer les contraintes de variation d'états entre deux étapes successives. Toutefois, nous remarquons que notre politique de conduite n'est pas toujours capable de réagir de façon cohérente lorsqu'elle commence à dévier: des collisions avec d'autres agents peuvent parfois survenir mais il arrive aussi que l' agent sous contrôle en évite certaines en sortant de sa voie. Afin d'enseigner à l'agent des connaissances a priori sur la tâche, nous avons également proposé d'entraîner notre politique de conduite par apprentissage par renforcement afin que l'agent puisse au moins éviter les erreures les plus triviales comme les accidents ou la conduite hors route lorsqu'il fait face à de nouvelles situations. En effet, les simulations et le processus d' exploration permettent à l'agent de générer un ensemble de nouvelles expériences qui viennent compléter la compréhension de la tâche. Nous développons plusieurs variantes d'algorithmes de type 'actor-critic policy gradient' pour apprendre des comportements de conduite sûrs pour de long horizon de simulation. Nous expliquons en profondeur comment configurer ces algorithmes de façon à stabiliser l'amélioration graduelle des performances : nous étudions notamment l'influence du processus d' exploration, celle de la fonction d'évaluation et enfin l'influence de prétraitement. Nous expliquons ensuite comment améliorer les performances de navigation sur de nouveaux scénarios non inclus dans la base de scénario d'entraînement. Nous montrons que le découplage de l'entraînement de la fonction d'évaluation et de la politique ainsi que le partage d'un réseau pour encoder l 'observation ont un rôle capital dans la capacité à généraliser de nos politiques de navigation. Il est ainsi possible pour la politique de généraliser des stratégies de conduite sécurisées sur de nouveaux scénarios sans accroître la taille de la base d'entraînement .Étant donné que la politique de conduite apprise avec apprentissage par renforcement ne se comporte pas tou-

jours comme un conducteur humain, il apparaît néanmoins nécessaire d'intégrer des démonstrations d 'expert dans le processus d'apprentissage. Les techniques standards d'apprentissage par imitation qui se limitent à exploiter un ensemble restreint de données expert souffre toutefois du décalage distributionnelle induit par la simulation ce qui se traduit par un dérive progressive qui aboutit souvent sur des collisions. Nous proposons donc d'adapter l'apprentissage par imitation de sorte que la politique puisse bénéficier d' expériences de simulation supplémentaires pour compenser le décalage distributionnelle. Ce principe est au cœur de l'Adversarial Imitation Learning(AIL) qui consiste à apprendre une politique qui doit constamment induire en erreur un discriminateur entraîné à distinguer les transitions expertes de celles générées en simulation par la politique. Alors que le discriminateur maximise une mesure de l'écart entre les trajectoires de l'expert et de la politique, la politique est entraînée avec de l'apprentissage par renforcement guidée par une fonction de récompense obtenue à partir du discriminateur. Nous montrons que les méthodes basées sur l'AIL permettent de réduire largement le décalage distributionnel ce qui se traduit par une meilleure capacité à imiter l' expert a long terme. Toutefois, les algorithmes se basant sur le principe de l'AIL peuvent se révéler extrêmement instable notamment parce que la compétition entre le discriminateur et la politique n'est pas toujours équilibrée. Nous étudions à travers plusieurs expériences l'influence des différents paramètres qui entrent en jeu dans la stabilisation de cette compétition et proposons plusieurs méthodes pour réguler les performances du discriminateurs. Les performances de la politique sont également fortement liées à la forme du signal de récompense dérivé à partir du discriminateur. Il arrive que le discriminateur fonde son jugement sur des caractéristiques non pertinentes de l'environnement ce qui a tendance à désorienter la politique lorsque celle-ci commence à se rapprocher de l'expert. Nous proposons ainsi une méthode qui s'appuie sur les GAN Wasserstein pour fournir un signal de récompense plus cohérent se basant sur une estimation de l'erreur de changement d'état plutôt que sur le couple observation action. Cette méthode permet non seulement d'obtenir de meilleures performances en termes d'imitation mais permet aussi d'apprendre une fonction de récompense facile à interpréter. En complément, nous avons également proposé une nouvelle architecture de planificateurs de trajectoire pour compenser les écarts par rapport aux trajectoires des experts lors des évaluations en boucle fermée. L'idée consiste a prédire la prochaine position à atteindre par l'agent de façon robuste sans être perturbé par les déviations avant de déduire quelle action effectuée en coordonnée curviligne. Cette méthode a pour avantage de pouvoir exploiter non seulement les démonstrations expertes mais aussi des données de simulation arbitraire afin d'ajuster le modèle de transition d'état Laurent, le modèle de décision inverse et le modèle de prédiction de position cible. Enfin, après avoir montré comment

apprendre les règles de base de la circulation à partir d'apprentissage par renforcement et après avoir montré comment imiter les trajectoire d'expert en limitant la dérive nous avons cherché a développé une méthode permettant de bénéficier des avantages des deux techniques simultanément. Pour cela nous proposons un nouvel algorithme d'apprentissage multi-objectifs qui combine deux objectifs basé sur gradient de politique, l'un calculé avec une fonction de récompense provenant d'un discriminateur entraîné sur des données réelles et l'autre provenant d' une récompense synthétique qui code les règles de circulation. Notre méthode appelée Multi Objective Policy Optimization (MOPO) collecte séparément deux ensembles de trajectoires avec notre politique sur des scénarios synthétiques et réels rejoués avant de calculer les deux gradients de politique associés. Ces gradients sont ensuite recombinés grâce à un optimiseur spécifique qui atténue les conflits entre les récompenses synthétiques et celles basées sur les données. La formulation multi-objectifs laisse également la possibilité de combiner différents types d'expériences générées avec plusieurs dynamiques d'environnement de sorte que l'agent ne se suradapte pas au trafic rejoué à partir d'épisodes enregistrés. Nous montrons que compléter les expériences de conduite de notre politique qui évolue à l'origine dans un trafic réel rejoué, avec des expériences synthétiques générées en présence d'agents interactifs, a un effet de régularisation sur la politique, rendant sa stratégie d'évitement des collision plus robustes aux variations de l'environnement. Nous montrons également que des scénarios entièrement synthétiques aident la politique à mieux comprendre la tâche et notamment l'objectif à long terme encodé dans le chemin de référence. Nous avons notamment remarqué que la proportion de scénarios synthétiques et la diversité de ces scénarios ont une grande importance dans le compromis final obtenu entre sécurité et imitation. Afin de déployer des entraînements à grande échelle avec de nombreuses simulations sur divers scénarios en parallèle, ce travail a nécessité la conception de notre propre simulateur de trafic. En effet, une de nos exigences était de pouvoir charger et rejouer des scénarios réels extraits d"Interaction dataset afin de pouvoir extraire des démonstrations d'expert compatible avec notre format d'observation et d'action . Notre travail a permis d'implémenter une 'framework' complète permettant d'apprendre des politiques de conduite réalistes à partir d'un ensemble de scénarios et de démonstrations extraits d'un dataset réel. Notre principale contribution est une méthode d'apprentissage multi-objectifs appelée MOPO qui comble les lacunes des approches purement basées sur l'apprentissage par renforcement ou par imitation sans pour autant recourir à l' intervention d'un expert interactif. Nous pensons que des travaux futurs pourront s'appuyer sur des approches mono-agent comme la nôtre pour initialiser un ensemble de politiques de conduite avant de les améliorer et de les étendre au travers d'une séquences d'entraînements plus sophistiqués définie par un curriculum. Les applications pratiques de la simula-

tion de trafic se heurte encore à d'autres limites comme le manque de diversité au sein de la population des politiques de navigation apprises ou encore les faibles capacités de coordination entre agents qui nécessite l utilisation de techniques plus avancées d'apprentissage multi agents.

Mots clés: Simulation de traffic, Apprentissage par imitation, Apprentissage par renforcement.

Abstract

Self Driving Vehciles(SDV) experienced fast development during the last decades with the joint rise of deep learning and high speed computation technologies. Currently, SDV are only able to drive in restricted locations such as highways and lack safety in interactive urban scenarios. Traffic simulation appears as a promising tool for incremental development of SDV because it makes it possible to run quantitative evaluation in a risk free setting. However, practical use cases require the simulator to provide realistic social interactions among all traffic agents such that the SDV can be tested as in real world conditions. In this work, we propose an approach to learn human-like driving policies for traffic simulation that can generalize safe driving behaviors in new and interactive scenarios. We first developed a hierarchical driving policy based on a routing module combined with a maneuver planner implemented with deep neural networks. We then learned navigation policies capable of imitating long-term expert trajectories thanks to the principle of Adversarial Imitation Learning (AIL). In order to benefit simultaneously from domain knowledge and real data, we propose a new multi objective algorithm that combines two policy gradients one computed with a data-driven reward computed with a discriminator trained with AIL and another based on a synthetic reward that encodes basic traffic rules. Our method resulted in better performances compromise in terms of security and imitation, in particular thanks to the inclusion of synthetic interactions generated with interactive agents.

Keys words Driving simulation, Adversarial Imitation Learning, Reinforcement Learning.

Acknowledgements: I first would like to acknowledge all my family that actively supported my commitment in this research work. I am really grateful to each member of the IoV Paris Research team for their motivations and their advises without which I couldn't have succeeded in completing this project. I address special thanks to Stefano Sabatini who actively contributed to the organization of this work and to publications writing. I also would like to express my special thanks to my two supervisors: Dzmitry Tsishkou from Huawei Paris Research Center and Christophe Sabourin from University Paris Est that trusted and supported me during those three years. Last but not least, I would like to thank Huawei Technology France which made this research work possible.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Several companies have developed Self Driving Vehicles (SDVs) such as, Motional, Magna International,AutoX, Cruise, Waymo, Zoox, Aurora and some of SDVs are already deployed in some federal states in U.S.A (California, Arizona) or in some towns in China (Shenzhen and Beijing). Self-driving systems still need to progress through the 6 levels of driver assistance technology advancements defined by the Society of Automotive Engineers. Currently, the last level of automation where no human supervision is required is far from being reached even if one of the most advanced company (Waymo) already collected more than 2M km of driving data. Providing quantitative safety guarantees on arbitrary locations and situations still constitutes a major challenge, especially if real world tests are systematically required to find out system deficiencies. Traffic simulation has gained a lot of interest in the recent years because it has the potential to considerably ease the development of SDV. In this work, we proposed a method for learning to simulate a realistic background traffic on various driving scenes. In this short introduction, we motivate the necessity to develop a realistic traffic simulator for advancing Autonomous driving.

1.1 Autonomous Driving Development

In a first sec.1.1.1, we summarize how self driving vehicles are currently structured and how the decision making process is implemented. Subsequently in sec.1.1.2, we explain why SDVs still face some major issues which prevents their large scale development in the real world. Finally in sec.1.1.3, we explain why simulation is a key tool for developing future self driving vehicles and highlight what still need to be improved for practical use cases in Autonomous Driving.

1.1.1 Architecture of Self Driving Vehicle

Self driving systems are composed of two main blocks : a perception block and a motion planning block as shown in fig.1.1. Perception usually consists in providing a local representation

of the driving scene localized in a semantic map where routing can be operated. The representation is generally the fusion of several data flow collected by various sensors as LIDARs, Multiple camera, proximity sensors etc [72]. Most advanced solutions provide accurate spatial representation of the ego agent surrounding reaching a centimetric precision at short range for the most advanced systems. However in real world condition, representation can still suffer from substantial issues such as occlusions or inaccurate depth estimation. Another difficultly comes from the estimation of other agents recent trajectories history which can be incomplete or blurred. The scene representation is exploited by motion planning module which combines a prediction module and a planning module.

Motion planning is a challenging problem because it implies to optimize a plan while predicting the short term trajectories of neighboring agents. End-to-End approaches that avoid explicit motion forecasting lacks interpretability and reliability because prediction and planning are closely intertwined[185]. Making consistent plans require to understand the scene context by first predicting the short term goals of the ego agent neighbors before inferring full trajectories which then serve to adapt the plan of the ego agent. Interpretable neural motion planners [213, 163] provide an alternative to end-to-end approaches, by maintaining modularity in the architecture while enabling end-to-end learning. They share a common representation for perception, prediction and planning leveraging multiple subtasks that constrain the decision making process. In order to integrate and select suitable information from the observation i.e change in traffic light and not only rely on arbitrary geometrical features, specific neural network architecture based on Transformers were developed as the TransFuser [151]. Additionally, model-based approaches also enable to enforce dynamical constrains for prediction and planning which can restrict potential future [174]. However, the multi-modal nature of prediction under intention uncertainty makes planning difficult because all agents interact with each other and the exact system dynamic is unknown. The diversity of road user as well as the diversity of road networks increase considerably the complexity of the scene future. Consequently, safety layers are often added on top of data based planner before the last decision is committed such that simple rules can be verified and such that undesirable reaction are avoided [196]. It was shown that adding an hand crafted safety layer considerably reduce the number of collision (by 95%) even if some failure can still happen. Indeed, even if massive amount of data are available, data-based models as well as handcrafted system still suffer from generalization issues for edge cases or unusual situation poorly represented in data [43]. Since appropriate strategies can not always be encoded manually or even found in driving data, we except that the driving system will generalize it trough training. Approaches based on reinforcement learning enables to incorporate basic rules which are sometime ambiguous in driving data [106]. Model based reinforcement learning or imitation learning recently proved that optimal plan can be optimized under sophisticated prediction model [194, 158]. As self driving systems develop, complete performance evaluation also get more difficult because real world test condition are difficult to reproduce.

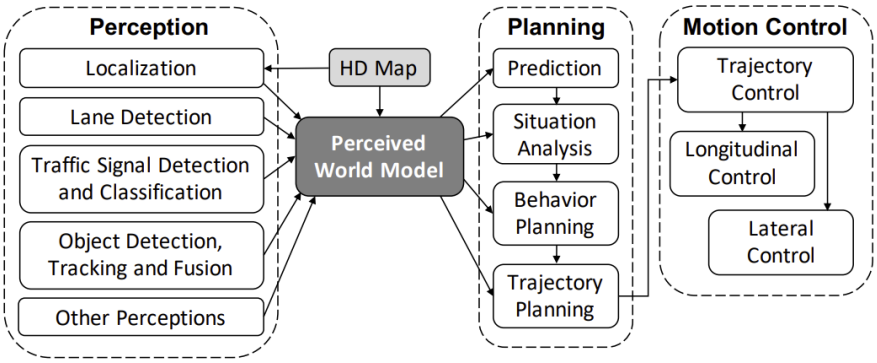

Figure 1.1: Nominal AD functions according to Autoware [84] and Apollo[41]

1.1.2 Challenges in Urban driving

Self driving vehicles are already allowed to be tested on public roads in some specific locations such as California from 2014 and in Germany where level 4 autonomous driving [15] is allowed accord to a federal law. The disengagement reports provided by manufactures shows that large distances can now be traveled by SDV. However, commercial application are still limited to level 2 autopilot mode on highways [35] and the core obstacle towards a large-scale deployment of autonomous vehicles is the long tail of rare events that are poorly represented in the data [224]. Multiple accidents of self driving vehicles or safety critical situations have raised great security concerns. As the performances of self-driving systems become better, it gets also more difficult to find real world situations that still constitute a danger. This makes real-world experiences less attractive after some time of development because more expensive to provide valuable data. Indeed there are exponentially many scenario variations due to the combinatorial number of possible lane topologies, actor configurations, trajectories, velocity profiles, appearance of actors and backgrounds and all variations cannot be collected on purpose. Running the risk to endanger real road user is not worth considering possible. Even if we always use the worst case driving strategy to take action, it can still lead to failure and it may additionally congest the traffic as shown in [221]. A systematic and quantitative analysis of self driving system safety is currently a major concern in addition to the lack of explainability of deep learning based methods. Deficiencies can come from several sources : occlusion or artefacts in perception system can harm the prediction which motivated the development of photo-realictic simulator as Carla [138] or Geosim [24]. Substantial issues also occur during decision making for critical situations where for instance unexpected reactions of some road user can perturb the planning module. Several works started to introduce a comprehensive taxonomy for critical scenario as well as methods to identify them [15, 64]. What represents a suitable risk level is a compromise between performance and safety. Quantifying risk and performances is made difficult because it requires a suitable coverage of the scenario space and criteria to define the completeness of the safety analysis. Systematic methods enable to generate various critical scenarios [36] and simulation appears as a promising solution to deploy massive evaluation with simple access to metric computations. The possibility to replay failures and apply specific modifications on

the scenario is important for incremental development and continuous maintenance. However rolling out realistic and interactive traffic simulations as in real world conditions is still a major challenge. Animating various traffic agent in the scene as human would have behaved implies to master basic driving skills while having access to a deep understanding of human driving styles. The inherent multi-modality of human driving behaviours requires to learn a diversity of traffic agents that can cover diverse situation such that evaluation do not simplify real world complexity. A practical use case consist in simulating a realistic traffic on a new area with few or no real data available which implies that traffic agents can generalizes consistent and robust driving strategies. Data based method and especially the one for sequential decision making are prone to failure when the discrepancy between test distribution and the training distribution. This problem is known as the distributional shift and highly limits generalization of data based models for traffic simulation or autonomous driving [43, 176, 30]. In the next section, we explain what are the limits of existing simulators and motivate our approach.

1.1.3 Traffic simulation for Self Driving Vehicle

Simulators offer the possibility of safe and low-cost development of self driving systems and it has become common practice to test reliability of ADSs in simulation [120]. First traffic simulators such as Sumo [119] or Aisum exhibit naive or stereotypical behaviors for the background traffic because they wew originally designed to study the dynamic of large traffic flows for anticipating traffic congestion. Rule based traffic agents cannot simulate microscopic traffic interactions on arbitrary road-maps in a fine grained manner. Microscopic traffic simulation emerged with the necessity of analysing safety critical scenarios for new self driving systems based on deep learning. Reproducing failures in an interactive context is crucial to identify root cause of consecutive decision with counterfactual analysis for instance. However simulating critical scenarios pose a significant challenge because traffic agents need to extrapolate human like behaviour. More generally rolling out simulation on scenario with no reference data is problematic since no expert behaviour are available to guarantee the realism of interactions. Data driven methods at least enable to evaluate how close the learned traffic agents generalize a human like strategies with respect to human demonstrations on a large test database. Once imitation performances can be obtained, it is reasonable to expect that well trained traffic agents may generalize human like behaviour on new scenarios (assuming scenario are similar) even if we do not have reference trajectories. In any cases we can still use proxy metrics to measure basic performances based on domain knowledge which enables to identify most unrealistic behaviours. Several data-driven traffic simulators recently emerged between 2020 and 2022. Some frameworks such as SMARTS [228], MALib [227] or Nocturne [195] mostly focus on learning safe multi agent interactions on various road-networks and were designed to deploy training massively in parallel. Their approach takes into account the necessity to grow a population of policies in a structured way but their main limitation is related to the fact that

there is no support with large scale driving datasets for comparison with real traffic. In contrast, some simulator were specifically based on data such as the the BARK [13] project which exploits demonstrations from the Interaction Dataset [214]. Unfortunately the framework was not designed to support efficient large scale reinforcement training which require a framework as Rllib [109]. A simulator like CARLA [138] even provide realistic rendering but its main purpose is more oriented on designing self driving systems rather than learning how to animate a background traffic. Even if CARLA now provide simplified Bird Eye View rendering there is currently no support to efficiently load real scenarios and demonstration from large scale driving dataset Note that we designed our simulator during 2019 and 2020 when few efficient interfaces existed. It appears that an efficient framework for learning to simulate a traffic needs an interface to load expert demonstration, an efficient road-map and scenario description as well as the integration of scalable libraries for multi agent learning and deep learning. A consequent part of our work consisted in developing such a pipeline since simulators that offer such functionality such as the Driver simulator[94] or the Nuplan simulator[18] were not available [1]. Additionally, there is also the necessity to properly formulate the learning problem behind traffic simulation. Even if Rule based policies tend to limit unpredictable behaviour, a gap persists between simulated and real driver behaviours [119]. While first data based methods such as PredictionNet [83] or Traffic-sim[180] adopted a centralized perspective for traffic simulation based on graph-based trajectory prediction model, promising works recently embraced a decentralized approach were single agent driving policies are learnt. BITS [206] formulates the driving task as a Bi-level Imitation with a high-level intent inference and a low-level driving behavior imitation. Symphony[74] couple Adversarial Imitation Learning and beam search technics to refine realistic driving behaviour while preserving diversity. The DriveIRL [147] learns a planner which generates a diverse set of trajectory proposals subsequently filtered and scored based on an IRL reward. Modeling human-like driving behaviors with a decentralized perspective offers a large number of possibility to progressively improve individual driving skill as well as coordination [228]. We will explain more in detail in the next chapter why we think that this approach is more promising than an end to end learning method of a centralized traffic model.

In this work, our main contributions for realistic traffic simulation are:

- A method to learn human like diving policies on real and highly interactive scenarios that resulted in a first publication 'Learning human like driving policies from real interactive driving scenes' for the international conference ICINCO 2022.
- A method for learning driving policies that balances safety and human driving imitation on interactive or replayed driving scenarios that resulted in a second publication: 'Exploring the trade off between human driving imitation and safety for traffic simulation' ITSC 2022.

[1] Note that those simulators are still at the prototype stage in 2022.

This manuscript introduces the main steps that led to the develop of those methods. In a first chapter, we first consider traffic simulation from a global perspective and propose an approach to learn a first generation of driving policies. In a second chapter, we introduce our driving simulator and the way we animate traffic agents that are not learning. In a third chapter, we propose different architectures for our driving policy and some pretraining methods on real demonstrations to validate our design. In a fourth chapter we explain how to learn robust driving policies with basic driving skills thanks to reinforcement learning. In a last chapter, we explain how to learn human-like driving policies robust to unusual situations thanks to adversarial imitation learning. Finally, we propose a multi-objective algorithm that optimizes both safety and human imitation for a driving policy on various driving scenarios.

Chapter 2

Driving simulator

Contents

Figures

Tables

In this chapter, we introduce the problem of traffic simulation and propose our approach in sec.2.1 for learning to animate traffic agents. In a second part in sec.2.2 we detail how we set up our simulation environment for traffic simulation.

2.1 Learning to drive in simulation

In the introduction, we explained why traffic simulation is so essential for the large scale deployment of self driving vehicles. A traffic simulator can be decomposed in two main parts: environment components as the road-network, traffic rule items and the pool of driving policies that enables to animate traffic agents and which constitutes the most challenging part of traffic simulation. Simulating a realistic traffic for long horizon is crucial for practical applications but is still not addressed. The main difficulty is to guarantee that all agents can generalize human like driving behaviours in new situations where no demonstrations are available. In this work, we propose to decompose the original multi agent problem into several stages to relieve complexity. We first explain how to model traffic simulation in sec.2.1.1 before defining the task we aim to solve in sec.2.1.2. Finally, we motivate the way we formulated our learning problem in sec.2.1.3.

2.1.1 Traffic simulation

We consider traffic simulation as the process of generating a driving episode on a bounded driving scene for a given temporal horizon H. The driving episode represents the set of trajectories generated in the scene during the temporal range $[0, H]$. Note that a driving episode may contain different kind of agents as vehicles or pedestrians and some dynamic part of the scene as traffic light signals which can also be registered in the episode if necessary.

Definition: driving episode A driving episode on a bounded scene is a set of trajectories $\{\tau^i_{t^i_{start}:t^i_{end}}\}_{i\in[1,N]}$ generated by all the N driving agents that evolved in the driving scene during the temporal range $[0, H]$.

In order to define how to generate a driving episode we introduce the definition of a driving scenario

Definition: driving scenario A driving scenario is a tuple $\mathcal{S} = (\mathcal{M}, \mathcal{F}, \rho, H, \mathcal{G}, \Pi)$ composed of a bounded driving scene $\mathcal{M}$, a traffic flow process $\mathcal{F}$, an initial state distribution ρ, a simulation horizon H, a goal assignment process $\mathcal{G}$ and a pool of driving policies Π.

The driving scenario is provided to the simulator which rolls out the scenario to generate a driving episode. The time at which driving agents are queried to take a decision for moving is called the decision step $\Delta t_d = 100ms$ while the time at which the global simulator state is updated is called the simulation step $\Delta t_s = 25ms$. The decision period is chosen as $\Delta t_d = k.\Delta t_s$ such that traffic agents can be modeled in a hierarchical manner[1]. At the top level there is

[1] A single agent decision can be applied for several simulation steps.

the driving policy that outputs at each decision step an action $a_t^i = \pi(s_t)$ that is a high level representation of the desired trajectory. Subsequently the action is decomposed into control inputs by the agent controller $u_t = f_{control}(a_t, s_t)$ at the simulation period. The agent physical state s_t can then be updated at each simulation step based on the agent physical model $f_{physical}$ and the control: $s_{t+\Delta t_s} = f_{physical}(u_t, s_t)$. We endow each agent of a physical model $f_{physical}$ a controller model $f_{control}$ and a policy π. Depending on the agent, the physical model can be a simple geometrical model, an unicycle or an advanced dynamical model as well as the controller that can be a simple linear interpolator of states or a PID like controller.

2.1.1.1 Driving scene

The driving scene $\mathcal{M}$ of a driving scenario $\mathcal{S}$ is composed of a bounded road map on which traffic agents are supposed to move. A driving scene can be highly complex if it integrates all the real worlds features. In advanced simulation as Carla[138], the world is modeled in three dimensions with realistic textures and sophisticated shapes for buildings, trees, bench but real world details are mainly exploited for application in perception. For decision making problem, most perception are usually provided with a high level representation to simplify the problem. A top view representation of the scene is sufficient for local representation of most of the road-network that does not include important hills that may induces specific driving patterns. As we mainly consider traffic simulation for vehicles even if pedestrian are also included, we focus on designing a representation that enables a car to reach it destination based on traffic rules and the scene context. For this purpose, we introduce a road network representation that includes all elements that enables to infer how to move and where not move. Several road-networks representations exist [11, 138] but they were not necessarily designed for autonomous driving like the OSM format [192]. One representation called the Lanelet2 [150] was specifically introduced to ease the decision making process for automated driving. We describe this representation in detail in sec.2.2.2 since it constitutes the basis of our driving simulator.

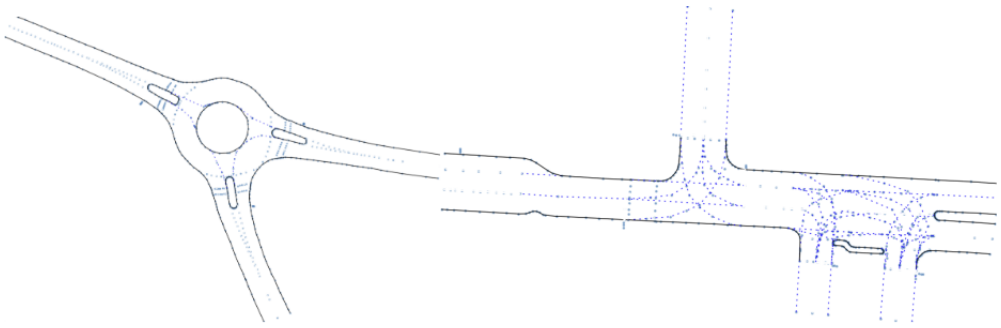

Figure 2.1: Road-maps used for traffic simulations

2.1.1.2 Driving policy

A driving episode is a set of trajectories generated by traffic agents. Trajectories are compact representations that can summarize the behaviour of any kind of agent from pedestrians, cyclist to vehicles. In order to generate individual trajectories we introduce the general notion of muti agent driving policy π^n that maps at each decision step the simulator state s_t and the goal assignment of n agents g_t to a vector of n actions $a_t = [a_t^1, ..., a_t^n]$. In general, we consider single agent driving policy that only consider one goal and output a single action but more general policies that requires coordination among agents may require the definition of the multi agent policy as shown in sec.2.1.3.1. As explained above, any single agent driving policy is associate to a controller that decomposes the action into a sequence of control that enables to update the agent state based on it physical model. Single agent action represents the desired trajectories to follow at least up to the next decision step and can have various form as explained in sec.2.1.2. In order to infer an action, each agent is endowed with an observation model that generates observations extracted from the global simulation state for each agent. For learning agents, an observation consists in an ego-centric partial representation of the driving scene as detailed in the next chapter.3 but for rule based agents that just animate the scene, we can provide more privileged information.
Since some agent trajectories for a driving episode may be used for downstream learning of driving behaviors whereas other trajectories of the same episode will just appear though observations collected by learning agents, we need to introduce another a distinction between two categories of driving agents. A driving agent that generates trajectories for downstream learning of driving behaviours is called an **actor** while a driving agent that just animates the driving scene is called a **traffic worker**[2]. Traffic workers (traffic agents) populate the driving scene but their trajectories: the sequence of consecutive observations and actions are not stored by the learning pipeline.

2.1.1.3 Traffic flow

Traffic simulation requires a mechanism to spawn driving agents on the driving scene $\mathcal{M}$ at each simulation step. We call this process the traffic flow $\mathcal{F}$. To populate a driving scene we first introduce the spatial distribution of initial positions of agents denoted ρ which usually has a fixed and discrete support on a bounded driving scene at the extremity of the map or close to building gates. Empirical distributions can be obtained based on multiple recordings of the same, but the scenarios that we built based on real data and called real scenarios, just extract initial positions of agents that were recorded instead of using a full distribution. In addition to initial locations, the traffic flow specifies which kind of agents to spawn i.e bicyclist, car, bus etc with their own features i.e length, width etc[3]. The Traffic flow also associates

[2]We also call them traffic agents when there is no ambiguity in order to alleviate notations

[3]Note that additional features could be provided as risk aversion or degree of cooperativeness for rule based agents

one driving policy $\pi \in \Pi$ and a goal specification $g \in \mathcal{G}$ to each spawned agent. In order to avoid initial position overlap the traffic flow needs to take into account the current scene context i.e current agent positions to activate some spawning locations at a given time or not. Since we consider bounded driving scenes there is a maximum number of agents denoted N that can appear at the same simulation step. This number can be fixed arbitrarily or obtained empirically if real scenes are available. Formally, the traffic flow can be modeled as a stochastic process that defines the evolution at each simulation step of a random vector $X_t = [X_t^0, ..., X_t^N]$ that describes the birth of at most N agents on the driving scene $\mathcal{M}$ at time step t. Each component $X_t^i = (b_i, x_i, y_i, \pi_i, g_i)$ is a random vector where b_i is a Bernoulli variable representing the potential birth of an i-th agent on the map at time t at location (x_i, y_i), and π_i denotes the policy associated to the agent i. The traffic flow naturally induce the distribution of N agent occupancy conditioned on the current scene context modeled by C_t:

$$\mathcal{F} \sim p(X_t^0, ..., X_t^N | C_t) \tag{2.1}$$

Note that driving simulation on bounded scene also requires to absorb driving agents before they reach map borders or at specific locations as garage entrances. We do not overload the notation with the agent absorbing process as it can be easily modeled the same way. In practice, we considered two kind of traffic flow : the first one is based on real recordings that just consist in replaying episodes with the same initial conditions (initial positions and goals assignment). In addition we also created synthetic traffic flows with predefined spawning location that get activated at specific simulation steps. By extension, we refer to a real episode when the driving episode uses a real traffic flow.

2.1.1.4 Driving mission

A single agent driving policy is conditioned on a goal assignment that specify what the driving agent should ideally achieve for the next H seconds of simulation. Driving behaviour can be quite complex as agents may change their destination or their strategy to reach theirs goals and vehicles can have long term goal far away or the intention to park. Since we are mainly interested in simulated vehicles on bounded scenes, we introduce a very simple goal specification that consists in the final location to reach on the map that is usually far from the initial position of the traffic agent. Note that this specification is not enough to specify a parking maneuver at a given location but our formalism makes possible to extend the goal specification with for example a maneuver descriptor. For our driving simulator, we let the traffic flow assigning to each spawned agent a final destination location as a goal specification and we keep this goal for the whole episode which is a reasonable choice for relatively small maps and short simulation horizons ($H < 20 seconds$).

2.1.2 Driving task

Driving policies are the key elements of the traffic simulator even if other parts like traffic flow process could require a lot of attention to be modeled in a realistic way. Our main objective is to design driving policies that behave as real human driver would do and not just safely. Since the driving task is defined as generating a consistent trajectory for reaching a long term goal, it is interesting to decompose the sequential decision process in different level of complexity.

2.1.2.1 Hierarchical driving policy

The driving task can be decomposed in two different stages that are relatively independent from each other. At the top level considered as the most abstract, any driving agent has to plan a route that leads to the long term goal. To this end, the routing module leverage the road-map structure to infer possible paths without considering dynamical element of the scenes as other road users. Graph search on the road-network can be used to select the sequence of lanes to follow and then a dense polyline can easily be extracted based on the center-lines leveraging the lanelet2Map format introduced in the next chapter.3. Usually the shortest path is preferred and should be selected but other choices based on the current traffic load of the driving scene can be used as a criteria. The route is considered as a sequence of positions and is not indexed by time even if ideal constant speed trajectory based on the scene speed limitation could also be used. Given a goal destination and the current simulation state, the routing module of agent i denoted $\pi_{routing}$ generates a traffic free reference path called **route** p_{route} and sends it to a lower stage of the policy called the **maneuver policy** $\pi_{maneuver}$. The maneuver policy is conditioned on the route to follow and is expected to output an action that will be converted into trajectory up to the next decision step thanks to the controller and the physical model of the agent. The choice of the action space and its associate controller is crucial and will be detailed in chap.3. At this level we consider a general action specification from which the next pose will be computed though the simulator. The action should ideally specify a trajectory whose spatial support is closed to the planned reference route p_{route} but some contextual elements may imply deviations. Concerning the speed along the trajectory, it should be consistent with the local traffic rules but also with safety margins to avoid collisions with other traffic agents. The top down structure composed of an upper routing module $\pi_{routing}$ and a lower level maneuver policy $\pi_{maneuver}$ enables to avoid inconsistent displacements and usually does not require that the routing module depends on the actions generated by the maneuver policy. This assumption is reasonable when the driving scene context does not require to replan for reaching the goal. In some situation like traffic jams or when the vehicle is blocked, it could be necessary to replan but most of the time, the maneuver policy is expected to handle the situation and to find a suitable trajectory. Consequently we decompose any driving policy as follows:

$$\pi = \pi_{routing} \circ \pi_{maneuver} \tag{2.2}$$

The routing module $\pi_{routing}$ that generates the traffic free reference path $\pi_{routing}$ is a relatively simple to implement since several works already proposed robust algorithms [53] but maneuver planning[4] is still an active field of research [229]. Indeed, the maneuver planner has to handle interactions between driving agents which can be hard to reproduce in a realistic way even with full observability. While planning a path on new maps is not prone to major issues assuming the road-structure is well described, finding a socially consistent trajectory on new maps or in new situations can reveal especially challenging because of the diversity of road-users interactions. In the next subsection, we focus our attention on the design of the maneuver planner.

2.1.2.2 Maneuver planning

In order to design a robust maneuver planner able to generalize safe and realistic trajectories on arbitrary contexts, we first review few works to identify the main existing trends. The first methods were fully rule based as the Intelligent Driver Model (IDM) [190] designed for straight highways, or MoByle [87] that can handle lane changes. The major limitation of pure rule based methods is the difficulty to adapt them on arbitrary road topologies. Data based approaches leveraging deep learning now enable to avoid complex hand crafted decision procedures. A lot of progress have been achieved in predicting the short term evolution of the local traffic as shown in [51, 20, 191, 226] which can be extended for planning applications [159, 194]. A General maneuver planer can leverage model predictive control for planning trajectories under constrains for some temporal horizon based on a prediction model. The main difficulty is to learn to dynamically adapt to other agents future trajectories conditioned on the ego agent plan which grows in complexity with the number of neighbors [153]. Concurrently, the driving task can also be formulated as sequential decision making process and several works proposed to apply Reinforcement Learning (RL) and Imitation learning to learn driving policies [229]. RL enables the policy to acquire common sense knowledge thanks to hand crafted rewards [139] while IL helps the policy to reproduce expert trajectories [63]. Most advanced approaches even propose to incorporate prediction model to refine the policy [89].
The question that arises is which approach is the most promising for learning not only one policy but potentially a pool of realistic driving policies for large scale traffic simulation. In the next sections, we propose to analyse two general formulations: one derived from the motion forecasting literature which adopts a centralized perspective in sec.**??** and another based on a Multi Agent Reinforcement Learning (MARL) which adopts a more decentralized perspective.

2.1.3 Learning driving policies

Traffic simulation is an interactive multi agent process that requires to animate all driving agents simultaneously. In order to learn driving policies, we first introduce the two main approaches

[4] In some works, they refer to behavioural planning but since we condition on a command, we call it maneuver planner.

in sec.2.1.3.1 and sec.2.1.3.2 before explaining the one we follow in sec.2.1.3.3.

2.1.3.1 A centralized approach for traffic simulation

Traffic simulation can be considered with a centralized perspective which consists in learning the joint distribution of all actors future states simultaneously [180] based on the driving scene context. Learning the full distribution instead of a deterministic model enables to sample multiple futures and forward multiple driving episodes from the same scenario. Let $\mathcal{Y}^i_{t:} = [Y^i_{t+\Delta t_s}, ..., Y^i_{t+\Delta t_{plan}}]$ denote the future positions of the ith agent present in the scene and let $\mathcal{Y}_{t:} = [\mathcal{Y}^1_{t:}, ..., \mathcal{Y}^N_{t:}]$ denote the futures positions of all the N agents present in the scene. Let $\mathcal{X}_t = [\mathcal{X}^1_{:t}, ..., \mathcal{X}^N_{:t}, \mathcal{C}_{:t}]$ denote the current context of the driving scene composed of individual agent past sate $\mathcal{X}^i_{:t}$ but also static or dynamical elements of the driving scene $\mathcal{C}_{:t}$ across past time steps. The centralized approach aims to learn the joint actor conditional distribution:

$$p(\mathcal{Y}_{t:}|\mathcal{X}_{t:}) \tag{2.3}$$

As this distribution encapsulates all uncertainties about agents intentions, driving style and interactions modes, the distribution cannot easily be expressed in closed from. Instead, the distribution is characterized implicitly via latent variable modes $\mathcal{Z}_t$ that encodes future scene dynamics.

$$p(\mathcal{Y}_t|\mathcal{X}_t) = \int_Z p(\mathcal{Y}_t|\mathcal{X}_t, \mathcal{Z}_t).p(\mathcal{Z}_t|\mathcal{X}_t) \tag{2.4}$$

A future multi-agent plan $\mathcal{Y}_t$ is generated by sampling $Z_t \sim p_\theta(\mathcal{Z}_t|\mathcal{X}_t)$ from an encoder and then decoding Z_t with a deterministic decoder $Y_t = f_\phi(X_t, Z_t)$. Conditional variationnal inference is used to obtain a lower bound of the likelihood of real driving data with respect to the $p(\mathcal{Y}_t|\mathcal{X}_t)$ which can then be maximized through Stochastic Gradient Descent (SGD). The training objective requires the introduction of a posterior model $q_\varphi(\mathcal{Z}_t|\mathcal{X}_t, \mathcal{Y}_t)$ that learns to map the ground truth future to the scene latent space for better reconstruction. Usually the latent space is partitioned for better representation of different source of stochasticity : decomposing the latent space per agent is a commom choice $Z_t = \{Z^1_t, ..., Z^N_t\}$. In order to generate a full driving episode from an initial driving scenario, the joint actor model is sampled sequentially at each time step $p_{\theta,\phi}(\mathcal{Y}_t|\mathcal{X}_t)$ and the new scene context is extracted based on plan. The process is summarized in algorithm 1

Centralized simulation has several convenient properties during training and test. The centralized simulation approach is inspired by the motion forecasting literature which intensively exploits graph neural networks and attention mechanism for learning to predict realistic and socially consistent multi agent trajectories [110, 226]. The performances obtained by several works [51, 20, 164] already proved that the local real traffic around an ego vehicle can be predicted for at least 5 seconds accurately and the main traffic modalities are often all detected. However supervised models still suffer from generalization issues when it comes to closed loop

Algorithm 1 Centralized traffic simulation as formulated in [**TrafficSim**]

INPUTS:

- $\mathcal{X}_0 = [\mathcal{X}_{:0}^1, ..., \mathcal{X}_{:0}^N, \mathcal{C}_{:0}]$ initial context
- Δt_{plan}: planning horizon

OUTPUTS:$[\mathcal{X}_0, ..., \mathcal{X}_T]$ # driving epsiod (multi agent trajectories)
$\Gamma = [\mathcal{X}_0]$
for t in $[t, t + \Delta t_{plan}, ...T]$: **do**
$Z_t \sim p_\theta(\mathcal{Z}_t|\mathcal{X}_t)$
$Y_t = f_\phi(X_t, Z_t)$
$\mathcal{C}_{:t+\Delta t_{plan}} = updateContext(\mathcal{C}_{:t})$
$\mathcal{X}_{t+\Delta t_{plan}} = Y_t$
$\Gamma = cat(\Gamma, [\mathcal{X}_{t+\Delta t_{plan}}, \mathcal{C}_{:t+\Delta t_{plan}}])$
end for

evaluation. While motion forecasting was mainly focused on open loop metrics the traffic simulation problem also requires evaluation in closed loop to attest that simulated agents all behave in a realistic way across time. We defined more precisely those notions for simulation evaluations based on a recent benchmark recently deigned to handle those limitations [18]. In order to evaluate a self driving vehicle we can consider its performances measured in terms of prediction errors, collision rate, traffic rule infractions etc in two different mode.

Definition : open loop evaluation

In open loop evaluation the policy is queried to output a trajectory from states reached by an expert stored in a dataset collected in the real world. The trajectory is evaluated according to specific metrics but is not used to control the vehicle for the next step.

Definition : closed loop evaluation

In closed loop evaluation the policy is queried to output a trajectory from states reached in simulation by the policy. Since the policy partially control the next state it will visit, it may end up in states radically different from the ones represented in the expert dataset.

In traffic simulation open loop and closed loop evaluation not only apply to a single self driving vehicle but to all traffic agents that are planning their next steps. Closed loop simulation with a centralized model requires to re-plan trajectories from states induced by the model which can gradually lead to compounding errors with respect to ground truth demonstrations. This problem is exacerbated by the the fact that multiple agents interacts which each other. A workaround is to resort to back-propagation through time (BPTT) as suggested in [180] but this solution is prone to gradient approximation and is sensitive to the sampling of latent variables [111]. As long as the simulation stays close to the ground truth episode, it is reasonable to think that BPTT can reduce error through time however when the scenario starts to diverge due to the latent variable sampling One agent can take the way instead of waiting at an intersection due to the value taken by a latent component z_i, it can turn unstable. Additionally, BPTT complexity grows linearly in times which requires to predict the next simulation with large horizon as done in Traffic-Sim. Another limitation of real training data is the fact that

they rarely contain safety critical situations that exhibit how to react in case of an emergency. Closed loop evaluation can lead to such situations where appropriate reactions are not represented in the training distributions which can lead to poor performances with for instance collisions or hard braking as shown in [220]. More generally, the **Distributionnal-shift** which is manifested by a gap between the state distribution induced by rolling out the model and the state distribution of the dataset has become a major concern in supervised learning for sequential decision making [43, 122, 30, 176].
Other practical concerns make the centralized approach less attractive as the lack of **modularity**. Learning a centralized model to animate variables number and type of agents on variable locations and for variable horizon is very challenging and the centralized models may be prone to over-fitting which limits its reliability in new situations. Decentralized approaches are mainly motivated by the need of modularity for fast and incremental deployment of simulation on arbitrary scenarios with eventually few expert demonstrations.

2.1.3.2 A decentralized approach of traffic simulation

Decentralized approaches based on Multi Agent Reinforcement Learning (MARL) consider the traffic simulation process as a sequential decision making problem. Instead of designing a centralized model to update the whole driving scenario, we can realise each simulation step with single agent policies that act in parallel for animating each traffic agent. In order to formulate traffic simulation as a MARL problem, we leverage the concept of Markov game with partial observability.
A **Markov Game** [112] with partial observability is defined by the tuple
$\mathcal{G} = (N, S, O_{ii\in N}, A_{ii\in N}, \mathcal{P}, R_{ii\in N})$. The game contains N agents whose global state belongs to $\mathcal{S}$. Each agent is assigned an action space $\{\mathcal{A}_i\}_{i=1}^{n}$ and an observation space which depends on the current global state. The function $\mathcal{P} : \mathcal{S} \times \mathcal{A}_1 \times \cdots \times \mathcal{A}_N \to T(\mathcal{S})$ describes the (stochastic) transition process between states, where $P(\mathcal{S})$ denotes the set of probability distributions over the set $\mathcal{S}$. At each step t, all the agents take actions $(a_1, ..., a_N)$ and the next state s_{t+1} obeys the transitions dynamic $s_{t+1} \sim \mathcal{T}(s_{t+1}|s_t, a_1, ..., a_N)$. Each agent i aims to maximize its own total expected return $R_t^i = \sum_{\tau=t}^{\infty} .\gamma^{t-\tau}.r_\tau^i(s_\tau, a_\tau^1, ..., a_\tau^n)$ where γ is the discount factor, by selecting actions through a (stationary and Markovian) stochastic policy $\pi_i : \mathcal{S} \times \mathcal{A}_i \to [0, 1]$. Each agent is assigned a reward function $r_t^i(s_t, a_t^i, s_{t+1})$ that maps a transition with a scalar value that represents instantaneous utility. The initial states are determined by the initial distribution $\eta : \mathcal{S} \to [0, 1]$.
For decentralized traffic simulation, the joint policy of traffic agents is defined as $\pi((a_1, ..., a_{n_t})|s_t) = \Pi_{i=1}^{n_t}\pi_i(.|s_t)$. The goal of each agent i is to maximize the long-term return J^i calculated by:

$$max_{\pi^i} J^i(\pi^i, \pi^{-i}) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t.r_t^i|s_0, a_t^i \sim \pi^i(.|s_t), a_t^{-i} \sim \pi^{-i}(.|s_t)] \tag{2.5}$$

where $-i$ represents the indices of all agents except agent i, and $\pi^{-i} = \Pi_{j \neq i}^{n_t} \pi_j(.|s_t)$ refers to the joint policy of all agents except agent i. Unlike standard reinforcement learning (RL), the optimal policy of an agent depends on other agents policies in the Markov games because all the J^i are coupled. To formalize this interdependence we use the concept of Nash equilibrium. Informally, a set of policies $\{\pi_*^i\}_{i \in N}$ is a Nash equilibrium if no agent can achieve higher reward by unilaterally changing its policy π_i :

$$\forall i \in [1, N], J^i(\pi_*^i, \pi_*^{-i}) \geq J^i(\pi^i, \pi_*^{-i}) \tag{2.6}$$

Usually, the goal of most MARL problems is to find local Nash equilibrium of the Markov games $\mathcal{G}$ without the knowledge of the dynamic. The main limitation of MARL for realistic traffic simulation is the lack of the 'True' expert reward functions that explain the behaviour of human like traffic agents. Even if most of motion forecasting dataset [70, 40, 19] can be converted into sequences of observations and actions, they do not contain the true reward signals. Hand crafting reward is a common practice, but those rewards can only act as proxy of expert true utility. The induced driving behaviour often exhibits common sense i.e the driving policy avoids crashes and stays on the road but no metrics enables to quantify how sharp is the reality gap with real human drivers [92].

In order to get around this problem, Multi Agent Inverse Reinforcement Learning (MAIRL) proposes to learn not only the policies but also the rewards from real expert demonstrations[175]. Suppose we do not have access to the reward signal r, but only to demonstrations $\mathcal{D}$ provided by N experts. In Markov games, we assume that all N experts operates in the same environment, and the demonstrations $\mathcal{D} = (s_j, a_j)_{j=1}^M$ are collected by sampling $s_0 \sim \eta(s)$, $\underline{a}_t = \pi_E(\underline{a}_t|s_t), s_{t+1} \sim \mathcal{P}(s_{t+1}|s_t, a_t)$. The problem of recovering a reward function r that rationalizes an expert behavior π_e can be formulated as matching two occupancy measures[50] i.e., the distribution over states and actions encountered when navigating the environment with a policy $\rho_\pi(s, a) = \pi(a|s).\sum_{t=0}^{\infty} \gamma^t.\mathbb{P}(s_t = s|\pi)$

$$RL \circ ILR_\psi(\pi_E) = argmin_{\pi \in \Pi} - H(\pi) + \psi^*(\rho_\pi, \rho_{\pi_e}) \tag{2.7}$$

where H denotes the causal entropy and whereψ^* is the convex conjugate of ψ, which could be interpreted as a measure of similarity between the occupancy measures of expert policy and agent's policy. This approach can be extended for Markov games with multiple rewards $\underline{r}$. We use $MARL(r)$ to denote the set of (stationary and Markovian) policies that forms a Nash equilibrium under $\underline{r}$ and maximum causal entropy (among all equilibria):

$$\begin{cases} MARL(\underline{r}) = argmin_{\underline{\pi} \in \Pi, \underline{v} \in \mathbb{R}^{\mathcal{S} \times N}} f_r(\underline{\pi}, \underline{v}) - H(\underline{\pi}) \\ v_i(s) \geq q_i(s, a_i) \forall i \in [N] \forall s \in \mathcal{S}, a_i \in \mathcal{A}_i \end{cases} \tag{2.8}$$

Based on the MARL operator, we need and inverse operator MAIRL, in analogy to IRL, which finds a reward that creates a margin between the expert and every other policy:

$$MARL \circ MAILR_{\psi}(\underline{\pi_E}) \tag{2.9}$$

The most challenging part for implementing a practical operator, consists in maintaining the constraints of the Nash equilibrium $v_i(s) \geq q_i(s, a_i) \forall i \in [N] \forall s \in \mathcal{S}, a_i \in \mathcal{A}_i$. [175] use a Lagrangian formulation which leads to the following optimization problem:

$$MARL \circ MAILR_{\psi}(\underline{\pi_E}) = argmin_{\underline{\pi} \in \Pi} \sum_{i=1}^{N} -\beta.H_i(\pi_i) + \psi_i^*(\rho_{\pi_i, E_{-i}} - \rho_E) \tag{2.10}$$

where ψ_i^* is a convex function measuring occupancy discrepancy and where π_i denotes the policy for agent i and π_E^{-i} expert policies for other agents. Note that in practice, we don't have access to $\rho_{\pi_i, E_{-i}}$ because we usually don't have interactive expert policies that can be rolled out along with π_i but just a fixed expert dataset. In practice [175] considers an alternative approach where they match the occupancy measure between ρ_{π_E} and $\rho_{\underline{\pi}}$. This approximation leads to another common issues in MARL known as non stationarity. As multiple learning agents are driving simultaneously, the environment faced by each individual agent appears non-stationary. In particular, the action taken by one agent affects the reward of other agents, and the evolution of the state. As a result, the learning agent is required to account for how the other agents behave and adapt to the joint behavior accordingly. This invalidates the stationarity assumption for establishing the convergence of single-agent RL algorithms, namely, the stationary Markovian property of the environment such that the individual reward and current state depend only on the previous state and action taken. Empirically, if the agent ignores this issue and optimizes its own policy assuming a stationary environment, which is usually referred to as an independent learner, the algorithms may fail to converge [142]. Beyond the non-stationarity issue, practical MARL algorithms also suffer from the credit assignment problem where individual agents struggle to estimate the impact of their own action over the return with respect to the impact of the joint action on the return [218].
In order to alleviate the learning complexity of the consecutive application of MAIRL and MARL operators, some authors introduce simplifying assumptions. For traffic simulation on highways. PS-GAIL[14] proposed to share parameters of agents policies while assuming that agents have the same action and observation space. Furthermore, they assume that all agents have identical rewards and that agent i reward only depends on the the state and action a_i : $r_{\psi_i}(s, a_i, a_{-i}) = r_{\psi}(s, a_i)$. Such idealization may be sufficient for specific setting but do not hold because expert demonstrations usually come from diverse human drivers with their own driving style.

2.1.3.3 Curriculum for learning driving policies

Learning realistic interactions among heterogeneous traffic participants on various scenarios remains largely unsolved. We previously reviewed two approaches that aim to solve the traffic simulation problem, the first is based on a centralized joint policy learned offline and the second based on decentralized policies learned though simulation with the support of domain knowledge and expert demonstrations. Even if promising results were provided in[14, 180], those methods are still insufficient for robust real world applications. Beyond the difficulty to provide trustworthy metrics to guarantee safety and realism on a arbitrary driving scenarios, current methods lack robustness when facing new scenarios in closed loop evaluation. Additionally there is no methods that enable to compare the diversity of the learned pool of driving policies compared to a real population of human drivers. A recent project called SMARTS [228] revealed a promising way to achieve interactive and realistic traffic simulation. Instead of trying to learn realistic and safe driving policies in one shot, SMARTS decompose the problem into a hierarchy of simpler problems and aim to solve them in a incremental manner. The logic is to reduce the complexity of each learning stage and progressively gain new driving behaviours without catastrophic forgetting. SMARTS builds upon the notion of curriculum that enables to evolve a Multi Agent System(MAS) [209]. We briefly introduce this fundamental mechanism that structures modern MARL training pipelines [227, 109]. From a general point of view, a multi agent system can be seen as a group of adaptive units where each units can itself be composed of adaptive units [104]. For simplicity, we consider atomic adaptive units even if agent can itself be composed of multiple interacting networks as policy or critics and various encoders. One way to evolve the MAS is to let it interact in a simulation environment. The environment where the MAS evolves is not necessarily directly the real ones. We consider that the training environment can change on purpose and we call any change of the environment a challenge. Challenges motivate adaptive units to explore since their primary goal is to maximize their return which has been affected by the environment change. We define a curriculum as a sequence of challenges or as a sequence of tasks to direct learning. Certain curricula may emerge naturally from the non-stationary dynamics of social interactions of the MAS, without any need for environmental engineering and those are called auto-curriculla since each challenge in the sequence is generated by the system itself. From the perspective of an individual learner, the problem is to adapt to a sequence of challenges, i.e. an auto-curriulum. Innovation occurs when following an auto-curriculum leads to a policy that escapes local optima where it would otherwise have been trapped. However, there is no guarantee that novel challenges will be continuously generated in this way. Auto-curricula, may be cyclic, repeatedly learning and unlearning the same information. Care must be taken to prevent from forgetting past policies because if old policies are forgotten, then a newer generation may become unable to outperform an older generation,preventing the productive accumulation of new innovations. In practice, successful self-play algorithms generally play not just against the latest (and strongest) policy, but also against as large and as diverse as possible set of older policies [100].

Therefore, at the heart of the SMARTS project, there is the 'social zoo' which contains various driving behaviours accumulated through the whole training procedure. This enables to avoid policy over-fitting to other agents behaviours and hence more robust behaviour can be learned. However diversity is not enough to understand how to coordinate agents and solve the credit assignment problem. Additionally it may not always be opportune to train all traffic agents simultaneously which brings non stationarity. Consequently SMARTS propose two important mechanisms: the first one is called the bubble and enables to isolate the learning of local interactions among traffic agents. A bubble is a spatiotemporal and conditionally specifiable region in which social agents are expected to be controlled for learning from their experiences. The second mechanism is the background traffic provider which enables to animate specific traffic agents of the scenario that are not supposed to learn from their experiences[5]. Consequently, agents could be trained in a highly focused way by only collecting interaction trajectories in bubbles, while the background traffic continues to supply realistic traffic flow through the outer membrane of the bubble.
Following a bootstrapping strategy, the SMARTS project aims to grow the social zoo for animating realistic traffic with incremental training stages based on increasingly more challenging multi agent interactions. Each training stage requires specific training algorithms, background traffics, driving scenarios and social agents as summarized bellow accordingly to the SMARTS [228] projects.

1. Rule based learning: In this setting, driving policies are designed manually and tested on the fly.
    - Agents are designed to follow specific rules and stick to them regardless of how the environment dynamic may have shifted.
2. Single agent learning (RL): In this setting the environment is considered as stationary and background traffic can be provided either by rule based agents either by frozen agents learned previously.
    - Model free RL : agents can learn to adapt from its online experiences but does not explicitly learn the environment dynamic.
    - Model based RL: agents not only learn to adapt, but also learn to model the dynamic of other road users of the environment. However, there is still no direct information exchange among the learning agents during such a decentralized learning process
3. Multi agent Reinforcement Learning (MARL): In this setting, multiple agents learn simultaneously from their common experiences.
    (a) Independent learning : Multiple agents are learning at the same time in the environment with eventually parameters sharing but independently without direct information exchange among the learning agents.

[5]Note that both background traffic agents and social agents come from the social zoo.

(b) Centralized training and decentralized execution (CTDE): agents start to share information during training and their common experiences are used during training to update each of them consistently but at execution time there is neither centralized control nor direct information exchange.

(c) Centralized training from demonstrations : In this setting MAIRL can be used to learn expert rewards from demonstrations while expert policies can be learn with a centralized MARL algorithm that can additionally enforce global consistency over interactions .

(d) Centralized training with communications: agents are able to communicate either for training purposes either to coordinate locally during execution.

- Local coordination: local group of agents are required to coordinate their learning and are expected to reach equilibrium at execution time i.e at an unprotected intersection, agents align their strategy.
- Global coordination: agents start to consider how their local action may impact the global traffic i.e would a lane change avoid congestion on this part of the road. Global coordination usually requires communication channels [156, 75]

Note that each level of multi agent interactions assumes that the background traffic as well as social agents are able to generate consistent training experiences without naively failing due to lack of skills i.e going off road instead of coordinating at an unprotected intersection. Therefore, organizing the training procedure can become very problematic for the last training stages because acquiring a new skill imply to master another set of skills potentially unknown without which new experiences are not valuable for training [209]. We posit that elementary skills for first generations of social agents can be learned trough single agent learning. In our work, we particularly study how driving policy can be learned both from domain knowledge and human demonstrations. We will show through chap.4 that simple driving skills can be learned with single agent RL while chap.5 will show that expert behaviours can be closely imitated with limited number of failures. Finally, we will also show that a reasonable trade-off can be found between imitating expert and learning rules through a multi task learning process. We posit that our methods could provide a first generation of driving policies with basic driving skill to initialise the social zoo described in the SMARTS project.

2.2 Simulation environment

In the previous section, we motivated our approach for learning single agent driving policies for traffic simulation. In order to evaluate driving policies or to generate data we need a simulation environment. We first explain why and how we designed our own simulator in sec.2.2.1. We discuss important aspects of the simulation and notably the road-network interface in sec.2.2.2

and the environment dynamic in sec.2.2.3. In a last sec.2.2.4, we explain how our framework makes possible to extract and replay real driving scenarios.

2.2.1 Designing a driving simulator

We first motivate our choice of designing our own simulator in sec.2.2.1.1 before detailing the simulation process in sec.2.2.1.2.

2.2.1.1 Limits of existing traffic simulator

Learning a realistic traffic is an incremental procedure as explained in sec:2.1.3.3 that rely on a driving environment not only for evaluation but also for training. Since we aim to use reinforcement learning to learn from interactions on various scenarios we need a scalable pipeline able to perform massively simulation roll-outs in parallel on multiple scenarios which reveals very challenging because of important data sharing between multiple process distributed on multiple CPU cores, high RAM memory consumption[6] necessary to store buffers and training batches and high requirements to deploy multiple neural networks on potentially multiple GPUs. Those issues are addressed in the SMARTS project which provides a scalable and efficient framework for traffic simulation dedicated for MARL. SMARTS is indeed built on top of Ray[127] which offers a distributed system for cluster-computing that enables simulation, training, and serving for RL applications. Contrary to CARLA simulator that can easily be used for RL [117], SMARTS was specifically designed for lightweight control of driving vehicles with better support for MARL which offers more flexibility to grow a diversity of driving polices. For better population management the MAlib project [227] was introduced in concurrence to SMARTS but currently does not offer driving environments support.
The main limitations of those projects are their inability to load real driving scenarios from existing datasets [214, 40, 19, 70] which prevents from comparing the learned driving policies with respect to real human trajectories. SMARTS currently does not provide this functionality probably because it uses the OpenDrive scenario format of SUMO which does not easily enable to convert arbitrary road networks with complex merging or intersections. CARLA simulator recently offers the possibility to load real scenarios but available data are very restricted [138]. Other very recent projects as the DriverGym(2021) [94] which now offers a RL simulator compatible with the Lyft dataset[40] or the Nuplan Project[18](2022) which offers a full pipeline for closed loop evaluation on NuScene Dataset [19] will probably fill this gap in the next years. At the beginning of our work in 2019, the most advanced project to replay episodes from real scenario was the BARK[13] simulator. BARK is built on top of the Interaction Dataset [214] which offers long horizon (up to 20seconds) driving episodes with challenging interactions on various kind of maps. Unfortunately this project is not built upon Ray and its associate Rein-

[6]More than 150GB required for 6 trainings each using 50 parallel simulations collecting 100K transitions for each training batch.

forcement Learning library called Rllib[109] which makes difficult to learn in a highly efficient and distributed way. Consequently, we decided to implement our own simulator combining the bests of the SMART and BARK projects for learning realistic driving policies from real scenarios extracted from the Interaction Dataset [214].

2.2.1.2 Simulation process

Before describing how we designed our simulator, we stress the main guidelines we followed. We designed a simulator such that :

- any driving simulation is initialized and configured with a driving scenario.
- real demonstrations can be integrated with the appropriate observation and action space to replay real episodes.
- the driving scene context can be extracted at each decision step for each agent with its point of view with an observation model
- the decision making process is separated in multiple levels from actions to control sequences that ultimately feed agent physical models.
- various metrics can be computed in closed loop to evaluate safety and imitation errors

The simulation environment is not only necessary for evaluation but also for training which requires that

- massive simulation roll-outs can be performed with multiple driving policies on multiples scenarios in parallel to feed RL algorithms.
- specific feedbacks can be provided at each decision step for each actor based on the global context
- multi agent trajectories coming from parallel simulations can be centralized and stored for training neural networks on appropriate devices (GPUs).
- neural network weights can be transferred to each instance of parallel simulator.

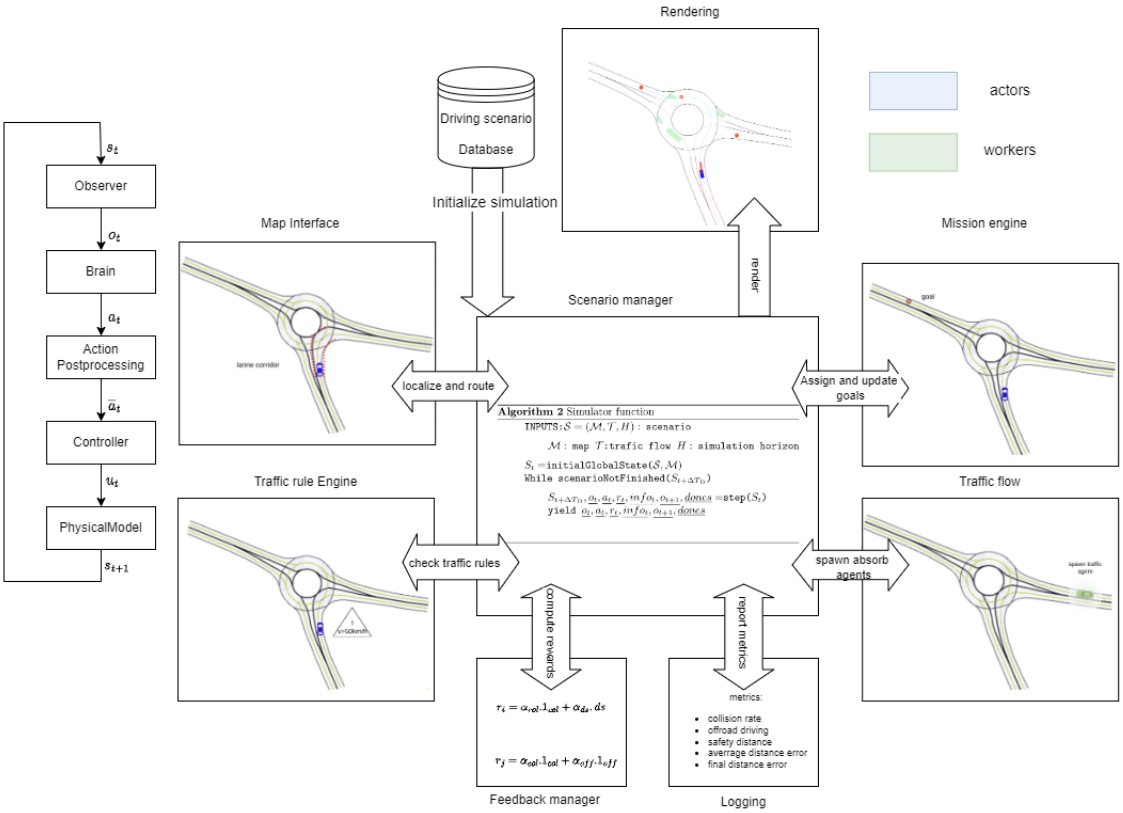

Figure 2.2: Simulation process

Algorithm 2 Simulator function

INPUTS:

- $\mathcal{S} = (\mathcal{M}, \mathcal{T}, H)$: scenario
- $\mathcal{M}$: map
- $\mathcal{T}$traffic flow
- H : simulation horizon

$S_t = initialGlobalState(\mathcal{S}, \mathcal{M})$
while scenarioNotFinished($S_{t+\Delta T_D}$) **do**:
  $S_{t+\Delta T_D}, \underline{o_t}, \underline{a_t}, \underline{r_t}, \underline{info_t}, \underline{o_{t+\Delta T_D}}, \underline{dones}$ =step(S_t)
  yield $\underline{o_t}, \underline{a_t}, \underline{r_t}, \underline{info_t}, \underline{o_{t+\Delta T_D}}, \underline{dones}$
end while

In order to design a suitable simulation environment we first looked for a real dataset from which real driving episodes could be replayed. Numerous dataset for autonomous driving are available [214, 40, 19, 70] but they were not originally collected for simulation purposes. The main limitations, beyond various amount of noise in the data, is the length of demonstrations which usually does not exceed 5 seconds. Learning long term driving strategies that are safe based on such short episodes is very restrictive. Another difficulty comes from observation extraction and metrics computation which both require an access to powerful map interface. Not all datasets contains lightweight hierarchical road-maps that can be exploited by a simulator: usually just raw perceptual data are provided. Our choice turned to the Interaction Dataset[214]

which offers demonstrations on multiple road networks topologies with an efficient Map format called Lanelet2 detailed in sec.**??**. In order to exploit map information to generate observations or to localize agents with respect to the road we designed a centralized Map Interface built upon the Lanelet2 map format Driving scenes entities are decomposed into polygons and polylines with advanced support for various geometric operations . The simulation process depicted on fig.2.2 and summarized in alg.2 starts from a driving scenario $\mathcal{S} = (\mathcal{M}, \mathcal{T}, H)$ which initializes a global simulation state denoted S_t that encompass states of all agents and traffic rule items. The global state will be updated for H steps with the step function that modifies the current global state to the next global state at time $t + \Delta T_d$ where ΔT_d denote the decision period . We summarize the principle of the step function in alg.3 :

Algorithm 3 STEPS function in process

INPUTS:

- T_D : decision time
- T_S : simulation time

OUTPUTS:

- $o_t, a_t, r_t, info, dones$

for $A_t^i \in \mathcal{A}$: **do**
  $o_t^i = a_t^i.observe(S_t)$
  $a_t^i = A_t^i.decide(o_t)$
  for $i \in [\Delta T_S / N_s]$ **do**:
    $u_t = control(a_t^i, S_t)$
    $s_{\tau+i.\Delta T_s} = simulate(u_t, S_t)$
  end for
  checkTrafficRules(S_{t+_D}, S_t)
  r_t =computeFeedbacks($S_{t+\Delta T_D}$)
  updateMissions($S_{t+\Delta T_D}$) # spawn and absorb agents
  $\mathcal{A}.updateTrafficFlow(S_{t+\Delta T_D})$
  for $A_{t+\Delta T_D}^i \in \mathcal{A}$ **do**:
    $o_{t+\Delta T_D}^i = A_t^i.observe(S_t)$
  end for
  info=comuteMetrics($S_{t+\Delta T_D}, S_t$)
  done=IsScenarioFinished($S_{t+\Delta T_D}$)
  return $\underline{o_t}, \underline{a_t}, \underline{r_t}, \underline{o_{t+\Delta T_D}}, \underline{info_{t+\Delta T_D}}, \underline{dones_{t+\Delta T_D}}$

During a decision step, each agent receives an observation o_t in the appropriate observation space built from the global state S_t with the observer model associated to the agent. This observation is then sent to the agent brain which can either be a learnable model or a fixed model depending on the agent type (actor or worker). The brain computes an action a_t based on the observation o_t and on its goal g_t according to the policy model of the agent. The action is then post-processed before being sent to a controller module that will generate a sequence of controls $\underline{u}_{t:t+\Delta T_D}$ to feed the physical model of the agent. The physical state s_t of the agent is updated for several simulation steps ΔT_s according to the sequence of controls and the physical

model : $s_{t+\Delta T_S} = f_{physical}(s_t, u_t)$. Physical model can be simple geometrical model that updates cartesian positions of agents, kinematical model as unicycle, or more advanced model based dynamical model. In practice we mainly worked with model in curvilinear coordinates that we detail in chap.3. Once the last simulation step is reached, the traffic rule engine checks whether traffic rules were respected during the transition from S_t to $S_{t+\Delta T_D}$ and store appropriate information in the new global states $S_{t+\Delta T_D}$. According to the traffic rules and the reward models r_i associated to to each agent, the feedback manager provides a reward $r_i(S_t, a_t, S_{t+\Delta T_D})$ for each agent A_t^i[7]. Subsequently the traffic flow engine spawns new agents at specific locations according to spawning distributions for synthetic episodes or according to a specific locations for real episodes. The traffic flow engine is also responsible to absorb agents that just left the map or that are eliminated due to an absorbing state reached at $s_{t+\Delta T_D}$. Note that replaying a real episode sometimes requires to spawn new agents in the middle of a decision step. The decision step ends with custom metrics computations i.e collision rates, offroad driving rates, imitation errors between for the transition $(S_t, S_{t+\Delta T_D})$ and finally the scenario manager checks whether the simulation should be ended either because the horizon H is reached either because no more learning agent is alive. The step function returns the transitions for all learning agents $\underline{o_t}, \underline{a_t}, \underline{r_t}, \underline{o_{t+\Delta T_D}}, \underline{info_{t+\Delta T_D}}, \underline{dones_{t+\Delta T_D}}$ as implemented in Rllib [109] that extends the usual single agent gym format developed by OpenAI. Note that a critical issue can occur if for a specific decision step no actors are alive because previous were absorbed and new ones will be spawn in the next steps. In this case no transitions can be returned and consequently the simulator should be able to automatically continue the simulation stepping up to the last learning agent(actor) get spawned.

In order to perform efficiently data collection for training, we chose to use the RLlib framework[109]. RLlib enables to compute the action inside a policy model that can eventually be shared among different traffic agents. Rllib also enables to switch easily from evaluation mode on multiple CPUs to training mode on a GPU devices which is critical for using neural networks. Once the policy is updated, Rllib makes easy to synchronize the weights on all the parallel instances of simulators. Another interesting property is the ability to chose which policy model is currently training which makes possible to use fixed neural networks for some agents in the scene instead of just rule based models. Lastly, RLLib offers a power-full trajectory API that enables to store multi agent trajectories of the same episode which can later be decomposed into single agent RL trajectories of variable length. This functionality is crucial to compute various quantity efficiently as the discounted return or Generalized Advantage Estimator for RL algorithms[166]. Lastly the distributed nature of Rllib makes possible to perform massive simulation roll-outs on potentially multiple clusters leveraging the ray framework which is necessary to learn from various experiences on different scenarios. Note that simulation computation can turn highly expensive due to multiple observation computation which requires access to

[7]Note that rewards are computed with the global state to access whatever is necessary and not from observations.

numerous CPU cores[8].
Simulation computation load can become a bottleneck if rendering are necessary to generate observations. Therefore we decided to use a rendering engine only for visualization in evaluation and we use observers that build vectorized outputs instead of high definition top view images of the driving scene. Other expensive computations come from the map interface which enables to localize agents on the road-network, extract various pieces of the road-network or check intersections. The use of a global state object that centralize information for all agents enables to perform individual queries for various services as the feedback manager, observers or control modules through the map interface. Once an operation is done on the global state as for instance localizing an agent A_t^i with respect to its route, it is not necessary to repeat this operation for other services that can just retrieve the result previously computed. The global state also enables to avoid the local copy of information that are shared for all services.

2.2.1.3 Simulation metrics

Since we aim to evaluate the ability to imitate human drivers and the ability to adapt safely on new situations while respecting traffic rules, we introduce two categories of metrics that we will provide at the end of each driving simulation. The first category called imitation metrics aims to measure the average errors of our driving policy with respect to the expert trajectory. Note that imitation metrics can only be computed in simulation with replay agents. Even if replay agents are not interactive, imitation metrics enables to estimate how fast and far the policy diverges from the expert trajectory. Ideally we should have an interactive realistic traffic and an online expert that provides new trajectories from the current state of the policy once it has definitely deviated [9]. In practice, we use the average distance error for a given horizon H on the whole scenario database $\mathcal{S}$ to evaluate imitation performances. Note that we only use scenarios that have at least H steps ($S_i \in \mathcal{S} | H \leq T_i$) to compute the ADE-H . Let $\underline{P}_\tau^\pi$ be the position of the policy at time t on a given scenario

$$ADE - H(meters) = \mathbb{E}_{S_i \in \mathcal{S} | H \leq T_i} \frac{1}{H} \sum_{\tau=1}^{H} ||\underline{P}_\tau^\pi - \underline{P}_\tau||_2 \tag{2.11}$$

In terms of safety, we mostly focus on the presence of collisions and off-road driving which reveals if basic driving skills are mastered. We compute the rate of episode with at least a collision denoted CR on the whole scenario database. Let $s_t^(i)$ be the state at time t on scenario i of the policy .

$$CR(\%) = \mathbb{E}_{S_i \in \mathcal{S}} 1(\exists \tau \in [1, T_i] | iscollided(s_t^i) \tag{2.12}$$

[8]We usually use 50 CPU cores to perform 50 parallel simulations

[9]Once the policy is too far from the original trajectory of the expert, it would be better to query the expert for an updated reference trajectory to analyse the future trajectory of the policy.

We can also focus on front collisions : the ones that occur inside the cone oriented along the ego agent body, with an opening of 60 degrees and starting from the centroid point of the ego agent body. To evaluate if the agent stays on-road, we compute the average amount of time where the policy stays outside the road corridor on each scenario of the database. In the next section, we will explain how to build the corridor associated with a reference thanks to the lanelet strcuture of the road map.

$$Off(\%) = \mathbb{E}_{S_i \in \mathcal{S}} \frac{1}{T_i} \sum_{\tau=1}^{T_i} 1(onroad(s_t^i) \tag{2.13}$$

There are more sophisticated metrics that we practically considered, as the amount of over-speeding, safety distances and jerks but it makes comparisons and analysis more heavy. Over speeding was not a concern in practice because our action space was bounded in contrast to low travel speeds. Since low speeds are often justified iun order to avoid collisions or just to wait for free space we mainly considered the average relative distance travelled with respect to the expert denoted $L(\%)$ measured with curvilinear abscissa relative to the reference path p. Let s_t^π be the curvilinear abscissa of the policy at time t on a given scenario indexed by i and let L_i^{expert} denote the longitudinal distance traveled by the expert on the same scenario.

$$L(\%) = \mathbb{E}_{S_i \in \mathcal{S}} \frac{(s_{T_i}^\pi - s_0^\pi)}{L_i^{expert}} \tag{2.14}$$

2.2.2 The road-network interface

The map interface uses a specific map representation designed for decision making of driving policies. Indeed, a raw picture representation of a road intersection, does not indicate explicitly which traffic light is valid for a particular lane. For motion planning, routing requires knowledge of consecutive lanes and where they lead otherwise the driving agent may move in forbidden directions or change lanes in inappropriate situations. The map representation should also help any driving agent to predict future moves of its immediate neighborhood. Information about the surrounding of a vehicle such as bicycle lanes, sidewalks, markings, traffic signs and curbs must also be available. As the agent is constantly moving, the map representation should also be easily extensible. Such requirements are met by the Lanelet2 map format[150] that we shortly summarize bellow:

2.2.2.1 Lanelet2Map

The lanelet2 map representation depicted in fig.2.3 globally assumes that all elements of the driving scene can be described by a projection onto a flat ground plane at least locally. A lanelet2 map is decomposed in several layers:

1. physical layer : contains real observable elements represented as points and line strings

primitives.

2. relational layer: it contains lanelets, areas and regulatory elements which are composed of items of the physical layer.

3. topological layer: it connects lanelets and defines the graph of the road-network.

All elements have in common that they are identified by a unique ID and attributes in the form of key-value pairs. Areas are regions defined by one or more linestrings of the map in which arbitrary displacements are possible i.e parking areas or forbidden i.e green spaces or buildings borders. A lanelet, defines an atomic section of the map in which agent motion is directed and explicit i.e a lane corridor, sidewalks. Inside a lanelet, traffic rules does not change and the topological relationships with other lanelets does not change either. Both lanelets and areas can have several regulatory elements which define traffic rules, such as speed limits, priority signs or traffic lights.

Definition : Lanelet A lanelet is spatially defined by a polygon encoded with two line strings: one for the left border and another for the right border which defines the lanelet direction.

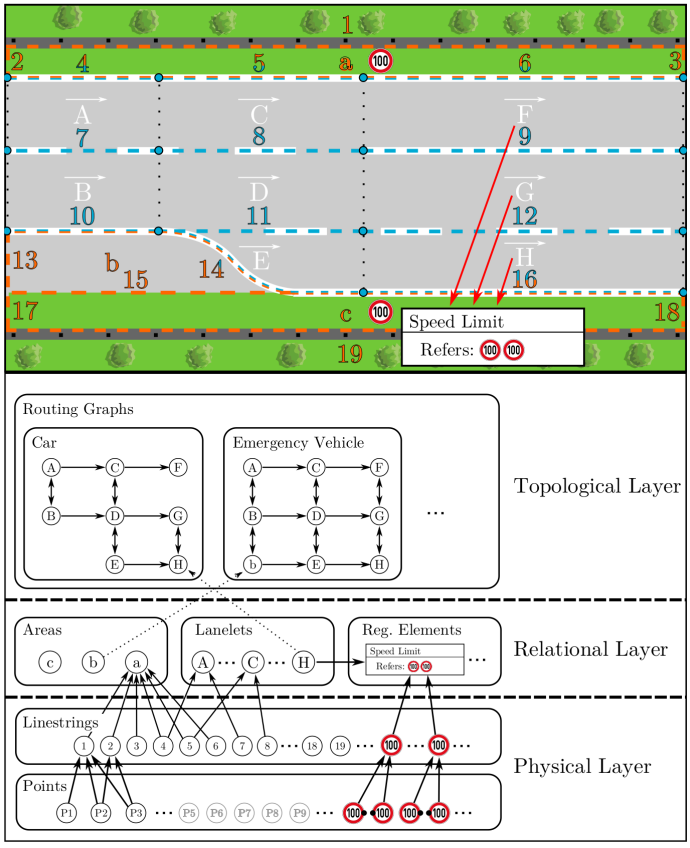

Figure 2.3: Lanelet2 map representation of a motorway borrowed from [150]

The lanelet2 map representation easily enables routing based on the topological layer as long as one can identify the lanelet associated to a given position. As several lanelets can intersect and even overlap, a wrong lanelet can easily be associated to the current position of the agent

which makes sometime impossible to find a route that leads to the desired destination. Hence a planner should take into account the context : the past trajectory to provide continuously a consistent route.

Definition : route A route from position X_t to goal g_t is a sequence of lanelets in the topological layer where the first lanelet matches with agent current configuration X_t and the last one matches with the position of the destination g_t.

The hierarchical policy model $\pi = \pi_{maneuver} \circ pi_{routing}$ use the routing module $\pi_{routing}$ to infer a route that is subsequently converted into a reference path p_{ref} that extends from the current position of the agent X_t to the goal g_t assigned by the mission manager. The reference path is extracted by concatenating the center-lines of each consecutive lanelet of the route.

2.2.2.2 Localization on the map

Given a route and its associate reference path, it can be handful to localize the agent or any other item with respect to it. It is also interesting to define explicit motions along the route. Curvilinear coordinates are very efficient to specify longitudinal and lateral positions along the route but their analytical expression requires parametric curves as Hermite curves which adds additional computational load for the simulator [10]. Using discrete representations for the route based on polylines can lead to discontinuities when the points density of the polyline is not sufficient. On the left side of fig.2.4, we see that naive curvilinear coordinates based on segment normals and length lead to two kind of issues. On the right side of the curve, points P_4, P_5, P_6 have the same abscissa $s = \sum_{i=0}^{1} \left\| \underline{P_i P_{i+1}} \right\|$ but no reference normal enables to define the ordinate. This prevent the conversion from cartesian to cuvilinear coordinates to be bijective which makes impossible to use a physical model that updates the cartesian position from curvilinear motions defined with (ds, n) for instance. On the left side of the curve, we note that curvilinear abscissa of points of segment $[P_1, P_3]$ do not evolve continuously. This come from the fact two normals can be used to define the ordinate of point P_2 because P_2 is on the bisector of cone (C_1, C_2, C_3). As we choose $\underline{n_{23}}$ to be the reference normal it induces a jump in curvilinear abscissa because $|s_2 - s_1| \gg |s_3 - s_3|$ despite $\|P_2P_3\| \approx \|P_1P_2\|$. This side effect is undesirable for control because small motions along the route should be smooth in curvilinear coordinates when the lateral distance with respect to the route does not change too quickly.

[10] Note that coordinates computation of numerous points are necessary for a single simulation step notably for observations.

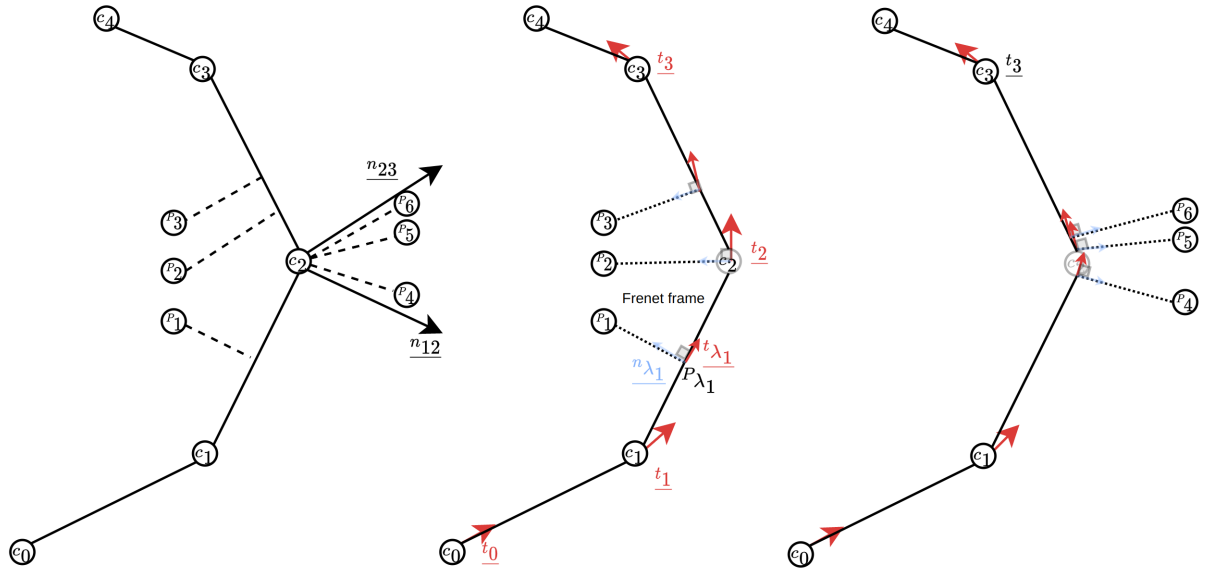

Figure 2.4: Curvilinear coordinates discontinuities: on the left side we show a naive Frenet-Serret frame that leads to coordinate discontinuities while the method that we later propose solve those issues as shown in the middle and on the right side of the figure.

A simple solution is to introduce tangents on each control point of the polyline and to interpolate tangents on each segment from base to tip in order to define a continuous Frenet-Serret frame [65]. We shortly describe the method that enables to compute curvilinear coordinates of point P located close to a polyline defined by the sequence of control points $[C_0, C_1, ..., C_N]$. We found out that the maneuver policy performances highly depend on the curvilinear coordinates smoothness provided by this method. We start with the fundamental principle that enables to check if a point P has its Frenet-Serret frame over the segment $[C_i, C_j]$. of length l. Without loss of generality we consider the segment at the horizontal and re-centered in frame $\mathcal{R} = (C_i, \underline{x}, \underline{y})$.

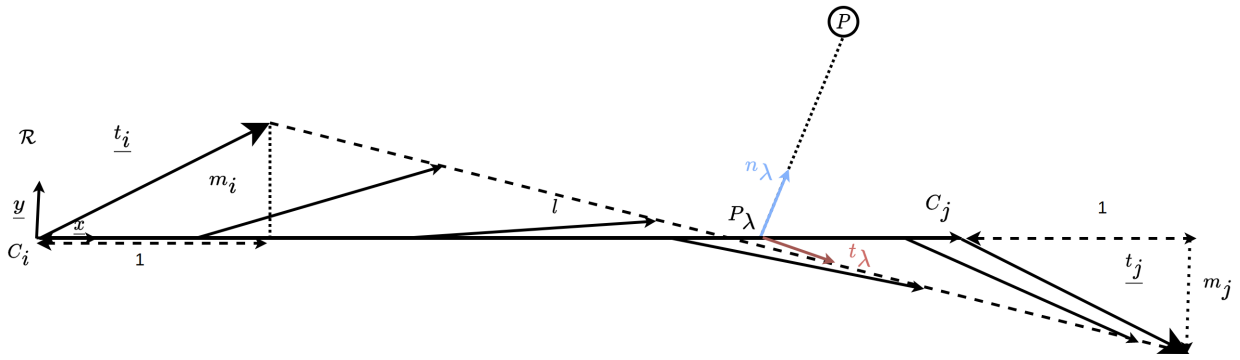

Figure 2.5: Curvilinear coordinate computation

Note that control points C_i and C_j depicted on fig.2.5 have their own tangents $\underline{t_i}$ and $\underline{t_j}$ that

can directly be computed from the polyline control points with the formula $\underline{t_i} = \frac{\underline{C_{i-1},C_i}}{2.\|\underline{C_{i-1},C_i}\|} + \frac{\underline{C_i,C_{i+1}}}{2.\|\underline{C_i,C_{i+1}}\|}$ for $i \in [1, N-1]$ and simply $\underline{t_0} = \frac{\underline{C_0,C_1}}{\|\underline{C_0,C_1}\|}$,$\underline{t_N} = \frac{\underline{C_{N-1},C_N}}{\|\underline{C_{N-1},C_N}\|}$. To simplify calculus, we force the coordinates t_i, t_j to be $t_i = (1, m_i)$ and $t_j = (1, m_j)$ which is easy to implement with a scaling operation. We enforce the Frenet-Serret frame that moves along the segment $\underline{C_i, C_{i+1}}$ to interpolate the tangents:

$$\begin{cases} \underline{t_\lambda} = \lambda.\underline{t_i} + (1-\lambda).\underline{t_{i+1}} \\ \underline{P_\lambda} = \lambda.\underline{P_i} + (1-\lambda).\underline{P_{i+1}} \end{cases} \tag{2.15}$$

Additionally, we have by definition of an orthogonal frame $\underline{t_\lambda}.\underline{n_\lambda} = 0$ so $\underline{t_\lambda}.\underline{P_\lambda P} = 0$.

$$\underline{P_\lambda P} = \underline{P} - \lambda.\underline{P_i} + (1-\lambda).\underline{P_{i+1}} = \begin{bmatrix} x \\ y \end{bmatrix} - \lambda.\begin{bmatrix} 0 \\ 0 \end{bmatrix} - (1-\lambda).\begin{bmatrix} l \\ 0 \end{bmatrix} = \begin{bmatrix} x - l(1-\lambda) \\ y \end{bmatrix} \tag{2.16}$$

$$\left(\lambda.\begin{bmatrix} 1 \\ m_i \end{bmatrix} + (1-\lambda).\begin{bmatrix} 1 \\ m_{i+1} \end{bmatrix}\right) = .\begin{bmatrix} 1 \\ \lambda.m_i + (1-\lambda).m_{i+1} \end{bmatrix} \tag{2.17}$$

$$\left(\begin{bmatrix} 1 \\ \lambda.m_i + (1-\lambda).m_{i+1} \end{bmatrix}\right).\left(\begin{bmatrix} x - l(1-\lambda) \\ y \end{bmatrix}\right) = 0 \tag{2.18}$$

$$(\lambda.m_i + (1-\lambda).m_{i+1}).y + x - l(1-\lambda) = 0 \tag{2.19}$$

$$\lambda.((m_i - m_{i+1}).y + l) = l - x - m_{i+1}.y \tag{2.20}$$

$$\lambda = \frac{l - x - m_j.y}{((m_i - m_{i+1}).y + l)} \tag{2.21}$$

When $\lambda \in [0, 1]$ it means that P lies over the segment $[C_i, C_j]$. In this case, we can compute the curvilinear coordinates of point P with respect to polyline $\mathcal{P}$ with $s = \sum_{j=0}^{i-1} \|C_j C_{j+1}\| + \lambda.l$ and the ordinate is $n = \sqrt{(x - l(1-\lambda)^2 + y^2}$. The conversion is bijective so for any curvilinear coordinate (s, n), we can extract associate cartesian coordinate as follow. First we search lambda by finding the first index i which provides $\lambda = \frac{s - \sum_{j=0}^{i-1} \|C_i, C_j\|}{l_i} \in [0, 1]$.Given λ we can compute $\underline{P_\lambda}$ and $\underline{t_\lambda}$and consequently $\underline{n_\lambda}$ wich is the orthogonal of $\underline{t_\lambda}$. Finally, we get $\underline{P} = \underline{P_\lambda} + n.\underline{n_\lambda}$. Note that the method we introduced still assumes that $\underline{P}$ stays close to the polyline to work properly and additional procedures need to be implemented in case the point lie far before or after the polyline end points. Additionally another problem can occur when the polyline $\mathcal{P}$ tracing the route contains loops i.e a full turn of a roundabout. Using naive localization with the full reference path can lead to errors as shown in fig.2.6

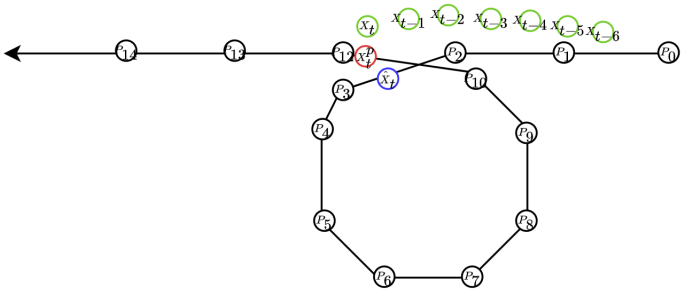

Figure 2.6: localization issue on a polyline including a loop: the curvilinear coordinates of point X_t may be mapped to the wrong curvilinear abscissa if the past trajectory is not considered.

For configuration X_t the naive method detailed before and applied on polyline $[P_0, ..., P_{14}]$ leads to a wrong projection $\underline{P_\lambda}{=}X_t^P$ instead of $\widehat{X_t}$ assuming that the agent already followed route $[P_0, P_1, P_2]$ as showed by its past trajectory in green on fig.2.6. A lightweight method to avoid such mis-localization is to decompose the route into multiple pieces based on the variation of tangents orientations. On fig.2.7 we illustrate this principle with the route decomposed in two pieces depicted in red and in blue based on a threshold on the maximum variation of tangent orientation equal to 180 degrees. We observe that the angle between $\underline{t_0}$ and $\underline{t_7}$ is bigger than 180 which triggered a route decomposition. In this example, there are only two pieces but there could be more depending on the route and on the threshold.

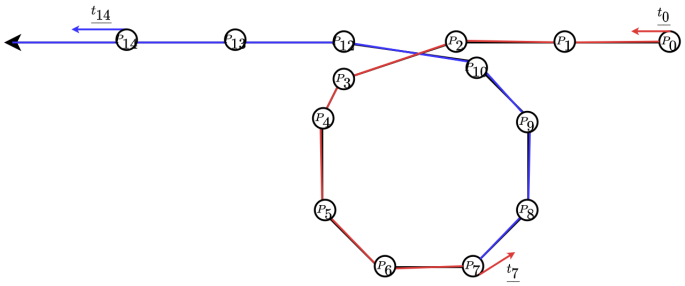

Figure 2.7: Long route decomposition based on a threshold on maximum tangent orientation variation: computing curvilinear coordinates with respect to each piece easily enables to detect discontinuities in curvilinear abscissa between two consecutive position(Assuming that the first one has a correct abscissa)

To localize a point with respect to a polyline decomposed in multiple pieces, we assume that we constantly have an initial guess of at least the last piece on which the point should lie. This assumption is reasonable because we are mainly interested in locating agent with respect to reference path provided as a polyline. The initial pose of the agent with respect to the reference path can be computed without error using the initial lanelet on which the agent is spawned at simulation initialization. This provides an initial guess of which piece of the polyline the agent is currently located and where it is likely to go for the next decision step assuming bounded

motion. When the agent starts to move it can deviate from the route but we still want to locate the agent pose with respect to it. Therefore, we start to locate the agent on the piece of road we guess it should be located on. If we realise that the agent lie outside this piece of route, then we extend our search to neighboring pieces of route up to the good projection (s_{t+1}, n_{t+1}) is found for the pose X_{t+1} . We check that (s_{t+1}, n_{t+1}) is consistent with previous (s_t, n_t) namely : $|s_{t+1} - s_t|$ should not exceed a few meters for a decision step of 100ms.
The localization method we use through our map interface enables to implement various driving policy that can directly exploit the structure, of the road to move. It is particularly useful to define rule based agents that need to reason directly in terms motion along the route instead of relative motion.

2.2.3 Driving scene dynamic

In this section, we describe the dynamic of the driving scene environment. Most of the driving scene is static as the road-network and regulatory elements only obey predefined rules and the environment dynamic mainly depends on behaviour of traffic agents. In our work, our main focus is to learn realistic driving behaviour rather than accurate physical models with high order dynamic. Therefore, we simplify as much as possible the driving scene dynamic and detail how to model traffic agents that populate the scene. We first explain in section2.2.3.1 which hypothesis we made on the transition model by introducing traffic workers in the simulation. Subsequently we describe how we designed two types of rule based traffic workers in sec.2.2.3.2 and in sec.2.2.3.3.

2.2.3.1 Transition model

As detailed in sec.2.2.1.2, we formalized traffic simulation with a Markov game $\mathcal{G}$ with partial observability whose transition model is denoted $\mathcal{T}_{\mathcal{G}}$. Assuming that a single agent is learning, it reduces to a Markov decision process $\mathcal{M}$ with a dynamic $\mathcal{T}_{\mathcal{M}}$ .. Note that the dynamics are not the same because in $\mathcal{G}$ all traffic agents are learning simultaneously while in $\mathcal{T}_{\mathcal{M}}$ part of traffic agents have a fixed behaviour. In $\mathcal{G}$, all actors are changing during learning which makes the environment dynamic non stationary from the point of view of each actor.For each decision step, the simulation process in $\mathcal{G}$ is summarized in alg.5. The transition model $\mathcal{T}_{\mathcal{G}}$ handles all agents physics through $f_{physical}$, collisions detection, all regulatory elements as well as agents spawning and absorbing process. Note that the variable number of agents between two steps encourage a decentralized approach since the number of random variable manipulated o_t^i, a_t^i, s_t^i are changing. In case of single agent learning the, other traffic agents have a fixed policy and are called traffic workers while the learning agent is called an actor. The behaviour of traffic workers is included in the dynamic $\mathcal{T}_{\mathcal{M}}$ as summarized in alg.6. Learning a driving policy with MARL in $\mathcal{T}_{\mathcal{G}}$ or with RL in $\mathcal{T}_{\mathcal{M}}$ is not equivalent because learning agents will not interact the same way and may develop different behaviour to optimize their return even if they share the same reward

Algorithm 5 Transition model in Markov Game

INPUTS:

- $\underline{a_t}$: joint action
- S_t: global state

OUTPUTS:

- $S_{t+\Delta T_D}$: next glocal state

for τ in $[t, t+\Delta T_s, ..., t+\Delta T_D]$: **do**
  for $agent_i$ in $\mathcal{A}_\tau$: **do**
    $u_t^i = agent_i.control(\underline{a_t}[i], s_\tau^i, u_{:\tau-\Delta T_s})$
    $s_{\tau+\Delta T_s}^i = agent_i.f_{physical}(s_\tau^i, u_t^i)$
  end for
  # Check collisions
  $S_{\tau+\Delta T_s}$ =checkCollisions$(s_{\tau+\Delta T_s}^{1:|\mathcal{A}_t|})$
  # spwan and absorb agents
  $S_{\tau+\Delta T_s} = \mathcal{F}(S_\tau)$
end for=0

Algorithm 6 Transition model in a Markov Decision Process $\mathcal{T}_\mathcal{G}$

INPUTS:

- a_t^i: single action
- S_t: global state

OUTPUTS:

- $S_{t+\Delta T_D}$: next glocal state

$\underline{a_t}[i] = a_t^i$ # decision for the actor
# workers decision making
for $agent_j$ in $\mathcal{A}_\tau{}^{-i}$: **do**
  $\underline{a_t}[j] \sim agent_j.\pi(.|S_t)$
end for
for τ in $[t, t+\Delta T_s, ..., t+\Delta T_D]$: **do**
  for $agent_j$ in $\mathcal{A}_\tau$: **do**
    $u_t^j = agent_j.control(\underline{a_t}[j], s_\tau^j, u_{:\tau-\Delta T_s}^j)$
    $s_{\tau+\Delta T_s}^j = agent_j.f_{physical}(s_\tau^j, u_t^j)$
  end for
  # Check collisions
  $S_{\tau+\Delta T_s}$ =checkCollisions$(s_{\tau+\Delta T_s}^{1:|\mathcal{A}_t|})$
  # spwan and absorb agents
  $S_{\tau+\Delta T_s} = \mathcal{F}(S_\tau)$
end for

function. In $\mathcal{G}$ all agent are fully interacting with each other while in $\mathcal{M}$ only a single actor is interacting with traffic workers which may just approximate real traffic interactions. However, since our approach is based on Curriculum learning changing the environment dynamics plays a key role. We posit that progressively changing the dynamic of our learning environment toward a real world traffic dynamic can help to learn robust driving policies with RL. Since

we hypothesize that the first stage of the curriculum consist in learning single agent driving policies, we need to define the dynamic of $\mathcal{T}_{\mathcal{M}}$, namely the dynamic of traffic workers. As the principle of curriculum is to increase complexity of the environment while gaining new skills, we should start with a basic driving task and a dynamic $\mathcal{T}_{\mathcal{M}}$ that is challenging enough for the first generation of driving policies. Since real and interactive driving behaviour are not available for large scale simulations, we have mainly two opposite possibilities. Either we use an interactive rule based agent, either we let the traffic workers replay their respective trajectories according to a real driving episode already recorded. Learning a single agent driving policy with RL in presence of replay worker will be studied in depth in chap.4 but we can already stress interesting properties. Numerous real driving episodes of reasonable temporal length i.e 5 to 20 seconds are already available in the Interaction Dataset[214] which enables to define driving scenarios with replay workers as explained in section 2.2.4. Even if replay workers do not react to the actor behaviour, they still enable to learn a driving strategy for avoiding collisions and in the best case the actor can even recover the expert strategy if the reward is properly defined as studied in chap.5. However, replay workers can also be replaced with more interactive agents such that the actor can acquire more diversified driving skills. Therefore we also introduced two kind of rule based agents in sec. and in sec.2.2.3.3.

2.2.3.2 Decentralized rule based agents

In order to define interactive rule based agents, we first consider a decentralized solution where each worker takes it decision independently from other decisions. Having at our disposal a Map Interface with localization along routes in curvilinear coordinates, we chose to extend the IDM model originally designed for longitudinal control [190] and unable to avoid collisions in case of intersections. Similarly to our hierarchical driving policy, our advanced decentralized IDM model (DIDM) use a planner to build a route leading to the goal it was assigned by the traffic flow. IDM originally just regulates the longitudinal distance with the front neighboor moving in the same lane corridor. IDM instantaneous acceleration $a(t)$, speed $v(t)$ and longitudinal position $s(t)$ with respect to its path p obey the following relations:

$$\begin{cases} a(t) = a_{max}.(a - (\frac{v(t)}{v_{max}})^{delta} - (\frac{d_{desired}(t)}{d(t)})^2) \\ d_{desired} = max(v(t).T - \frac{v(t).\Delta v(t)}{2.\sqrt{(a_{max}.b)}}, 0) + d_0 \\ d(t) = s_n(t) - s(t) - L_n \\ \Delta v(t) = v_n(t) - v(t) \end{cases} \tag{2.22}$$

where a_{max}, v_{max} denote respectively IDM maximal acceleration and speed, b is the comfortable deceleration desired, d_0 is the safety distance gap and T is the desired time gap. The parameters $v_n, s_n(t)$ denote respectively the speed and the longitudinal position of the leading vehicle with respect to IDM path, and L_n represents the half length of the leading vehicle. In

order to handle driving scenarios with intersections we need to regulate the IDM longitudinal speed when it faces a front neighbor that is not moving on the same lane as itself and that is likely to intersect its route. This situation occurs for example on a roundabout when the IDM agent faces a neighboor that is about to enter the roundabout. As any traffic agent (actor or worker) is assigned a path to follow at the beginning of the simulation, any IDM agent can know if a neighbor in a radius r_{inter} is intersecting its own path. On fig.2.8 we show that our advanced IDM agent called DIDM can infer the intersection point denoted P_I between its route and the route of the ith agent. To check that the ith neigboor is moving toward the intersection point we can check that its longitudinal speed reduces the gap between its current position and P_I. If the $DIDM$ agents is closer to P_I as indicated by

$$s(P_I)_{/r_{DIDM}} - s(P_{DIDM})_{/r_{DIDM}} < s(P_I)_{/r_i} - s(P_i)_{/r_i} \tag{2.23}$$

then DIDM is allowed to take the way and does not modify its speed in function of the intersecting neighbor. Otherwise DIDM has to give the way which consists in moving up to a virtual stop point P_i^{stop} and then stop for some time period T_{stop} that we can modulate. The virtual stop point acts as a virtual leading neighbor which would be located on DIDM's route at abscissa $s(P_i^{stop})_{/r_{DIDM}}$and whose longitudinal speed would be zero [11]. The virtual stop point is located at the intersection of DIDM's route and the safety circle centered at P_I as shown on fig.2.8. In case DIDM has several close neighbors likely to intersect its route, it is necessary to calculate which virtual stop point has to be considered to regulate the longitudinal speed of the DIDM agent. For example if DIDM has N neighbors of half length L_i about to intersect its route located at $\{P_i\}_{i\in\{1,N\}}$ and a leading neighbor on the same lane located at P_l then we first compute their associate virtual stop points $\{P_i^{stop}\}_{i\in\{1,N,l\}}$. To select which virtual stop point will be considered, we use the following rule

$$j = argmin_{j\in[1,N,l]}(s(P_I)_{/r_{IDM}} - s(P_j^{stop})_{/r_{IDM}} - L_j) \tag{2.24}$$

which indicates that the closest neighbor along DIDM's route r_{IDM} is considered as the one that regulates DIDM's speed according to formula 2.22.

[11] More advanced model could use a moving virtual stop point

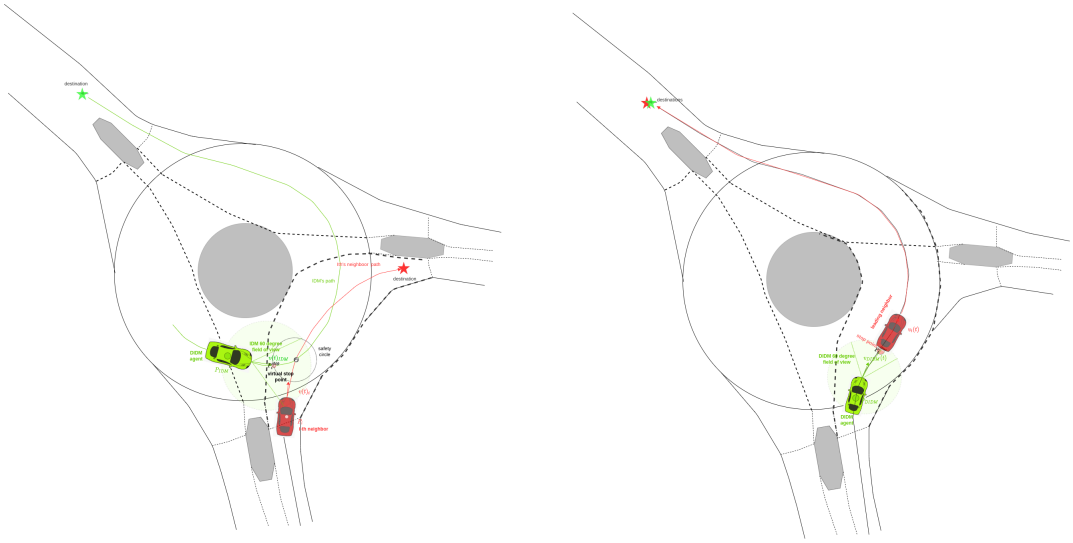

Figure 2.8: Illustration of our advanced IDM decision making process(DIDM).

To validate that our DIDM model can generate an interactive traffic with few collisions, we realised two experiences. We first check that our advanced IDM model has low rate of epsiodes with at least one collision denoted ($CR\%$) in presence of a traffic composed of replayed agents. This experience enables to identify specifically deficiencies of the DIDM because other agents are not reacting. We also consider the rate of episodes with a front collision ($FR\%$) which should be low since the DIDM agent is specifically designed to regulate the gap with neighbors in the front. We ran the evaluation on two scenario databases $Huge\_R\_Basic$ and $Huge\_I\_Basic$ detailed in .2. For comparison, we also replace the DIDM agent we evaluate with either a simple IDM model that do not handle side collisions or a constant speed model that moves forward at 30km/h. We observe that our DIDM agent can reach much lower rate of collisions than a simple

	Huge_R		Huge_I	
	CR%	FR%	CR%	FR%
constant speed	42	21	37	25
IDM	21	2	18	2.5
DIDM	9	1.5	13	2

Table 2.1: Performances of a DIDM agents in presence of a traffic populated with replay agents

IDM baseline or than a constant speed model even if in some cases, it still collides at the level of intersections because the virtual stop point is not always sufficiently far from intersecting agents that are slightly shifted from the center-line. We realised another experiences where we still test our DIDM model on the same set of driving scenarios expect that we replace all replay agents with DIDM models. This experience enables to understand if decentralized interactions among DIDM result in low rate of episode with a collision. We observe that the rate of collisions is lower than in the previous experience because DIDM traffic workers try to avoid collision with other traffic agents. We note that the DIDM agent is still safer than the IDM baseline or than the constant speed model. This can first be explained by the fact that all agents stay on the

	Huge_R		Huge_I	
	CR%	FR%	CR%	FR%
constant speed	12	8	19	13.4
IDM	9	2	13	2.5
DIDM	7	1.5	9	2

Table 2.2: Performances of a DIDM agent in presence of a traffic composed of DIDM agents

center-line which reduces the number of locations where collisions could occur. Qualitatively, we also observed that the traffic tends to evolve in a irregular manner compared the real ones : agents that are likely to intersect slow down, and then progressively re-accelerate once the path is free which generates jerky traffic patterns that do not look realistic.

2.2.3.3 Centralized rule based agents

In order to have a consistent coordination strategy at intersections: a consistent order of passage, it is easier to share information among all agents and take a common joint decision at each decision step of the simulation. Therefore we also developed a centralized version of our advanced IDM agents(CIDM). Since we know the full route of each agent and all intersections areas on the map thanks to the lanelet structure, we can coordinate the passage of each agent for each intersection area that lies on their route. Let's illustrate the principle with the example depicted on fig.2.9

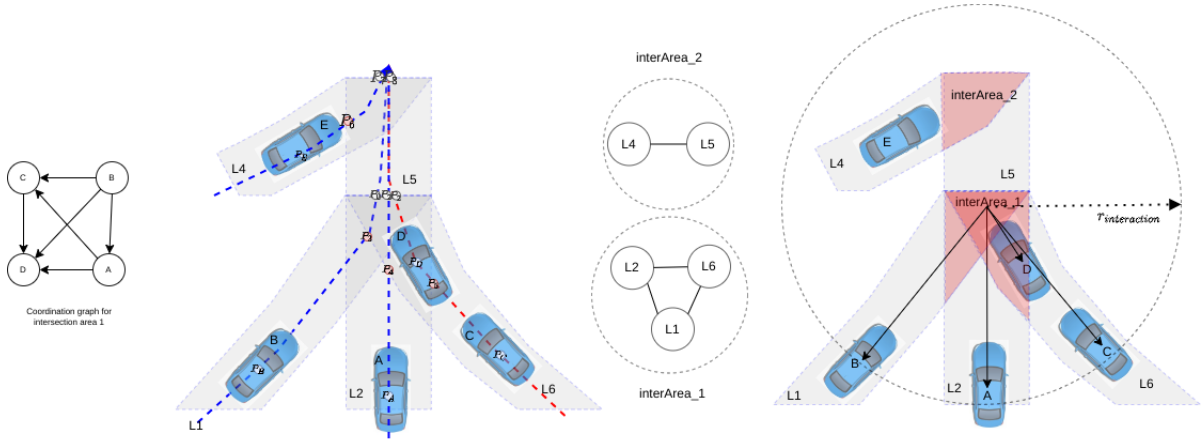

Figure 2.9: Centralized coordination for rule based agents

The roadwork contains two intersections areas where respectively lanelets L_2, L_1, L_6 and lanelets L_4, L_5 intersect . Intersection areas can be pre-computed at simulation initialization and stored in a graph as shown on fig.2.9. For intersection area 1, the route of four (CIDM) agents are intersecting so we need to coordinate their passage at this level. We specify the order of passage as follows $D > C > A > B$ which means that agent D goes first followed by C that is allowed to cross the intersection area once A has finished to cross it etc. In order to specify at which position an agent should stop before crossing the intersection area, we compute the

intersection of its route with the borders of the intersection area . Similarly to (DIDM, we call this intersection point a virtual stop point and it corresponds for instance to point P_3^s for agent B at intersection area 1 as shown on fig.2.9. Having at our disposal curvilinear coordinates, we rank priority of passage based on longitudinal distances that separates each agent from its virtual stopping point with respect to its own route. The first agent allowed to cross the intersection area 1 is the one that verifies:

$$argmin_{i\in\{A,B,C,D\}} s(P_i^s)_{/r_i} - s(P_i)_{/r_i} \tag{2.25}$$

where $s(P)_{/r_i}$ denotes curvilinear abscissa with respect to the route r_i of agent i. Note that the order of priority assumes that all agents move at the same constant speed. To consider the speed in the priority rule, it is possible to divide $s(P_i^s)_{/r_i} - s(P_i)_{/r_i}$ by the current longitudinal speed of the agent along the route. Since IDM principle is to maintain a constant speed along the route and since all rule based agent are supposed to maintain the same constant speed in our simulation, we do not divide by the speed in our framework. To detect if an agent i finished to cross an intersection area, we check if $s(P_i^o)_{/r_i} < s(P_i)_{/r_i}$ where P_i^o denote the intersection point between route r_i and intersection area 1 whose curvilinear abscissa is the biggest[12]. Note that the main changes of CIDM agents with respect to DIDM agents are the way virtual stop points are computed and which virtual stop point each CDIM agent has to consider i.e its order of passage at an intersection at every decision step. In terms of implementation CIDM agents need to share information among agents and notably the coordination graph represented on fig.2.9. The incomming edge $B \rightarrow A$ indicates that agent A has the priority with respect to B and each node stores the virtual stop point of each agent. We summarize the centralized coordination process of CIDM agents in alg.7. Note that in a traffic composed of some CIDM agents and some other kind of agents, CIDM agents assumes that all agents follow the centralized coordination process updated at each decision step even it is not the case in simulation.
In order to validate our CIDM model, we realised the following experiences. On the same set of scenarios as in the previous section, we evaluate a CIDM agent among a traffic of CIDM agents. Our goal is to verify that consistent centralized coordination strategies can be generated in such a traffic. Note that a centralized traffic can be used to teach a learning agent a specific driving strategy: namely a consistent prior strategy to coordinate at an intersection. In the

	Huge_R		Huge_I	
	CR%	FR%	CR%	FR%
RA vs CIDM	37	17.5	34	13
CIDM vs CIDM	4	1	5.5	2

Table 2.3: Performances of CIDM agent or a replay agent(RA) in presence of a traffic composed of CIDM agents

first row of tab.2.3 we see that a replay agent in presence of CIDM agents often tends to collide

[12]Indeed,the intersection area is not necessarily convex and the route is not straight so several intersection points may exist.

with CIDM agents of the traffic which shows that a traffic populated with CIDM cannot adapt to an arbitrary driving behaviour. In contrast, a CIDM agent have almost no collisions with other CIDM agents of the traffic (second row) but interactions are considerably slower than in the previous experience because CIDM are very conservative by definition : they wait their respective turn before committing in the intersection. The CIDM agent usually stop far enough from other agents thanks to the use of intersection areas but it spends considerably more time motionless[13].

Algorithm 7 Centralized coordination at an intersections

INPUTS:

- traffic agents $\mathcal{A}$
- intersection_areas

OUTPUTS:

- virtual stop points

for inter_area_i $\in$ intersection_areas **do**
  # extract all agents likely to collide at intersection area i
  $\mathcal{A}_i = \{\}$
  for a_j in $\mathcal{A}$ **do**
    $\mathcal{A}_i.add(a_j)$ if $s(P_j^o)_{/r_j} > s(P_j)_{/r_j}$and $\|O_i - P_j\| < r_{interaction}$
    #Compute distance to stop points
    for a_j in $\mathcal{A}_i$ **do**
      $\delta s_j = s(P_j^s)_{/r_j} > s(P_j)_{/r_j}$
      # Check the closest agents
      $[\delta s_{k_1}, ...\delta s_{k_{|\mathcal{A}_i|}},]$ =sortDistancesToStopPointInIncreasingOrder$(\mathcal{A}_i, \{\delta s_j\}_{j\in\mathcal{A}_i})$
      # first agents take the way and ignore others in $\mathcal{A}_i$
      for a_j in $\mathcal{A}_i \backslash \{k_1\}$: **do**
        a_j.updateVirtualStopPoint$(P_{k_1}^s)$
      end for
    end for

Contrary to DIDM traffic agents which enables to introduce some stochasticity in the simulation, CIDM traffic agents enable to provide an interactive traffic that always applies the same joint coordination strategy. For a traffic populated with DIDM agents, there is no guarantee that all intersecting agents will cross one by one the intersection area without conflict. In contrast, a traffic composed of CIDM agents is supposed to coordinate the passage of each agent at each intersection area and at each decision step. Leveraging DIDM and CIDM traffic agents, it is possible to simulate various interactions for learning robust driving behaviours as shown in chap.5.

[13]Some collisions remain because we do not explicitly handle speed and acceleration of CIDM agents within the coordination strategy which generates some delay between the expected behaviour at a given time step and the simulated behaviour.

2.2.4 Building driving scenarios

In order to run a simulation, we need provide a driving scenario to the simulator. In this section we just explain how we extracted real driving scenarios from a the Interaction Dataset[214] but we detail the composition of all scenario databases used in this work in annexes .2.

2.2.4.1 Extraction from real epsiodes

A driving scenario can be built in two different ways: either based on real episodes either based on synthetic requirements. Creating synthetic scenarios with synthetic maps as in [107] is also possible in our framework but less interesting since we aim to imitate human drivers in real situations. Even if synthetic scenarios also enable to learn basic driving behaviour in simulation as collision avoidance strategy, they do not enable to compare the trajectory of the learner with a human expert trajectory[145]. Building driving scenarios from real episodes has several advantages over synthetic generation. The goals assignment and the initial configuration of all traffic agents are directly available in real driving dataset as [214] otherwise they need to be computed by some rule based procedures based on the map topology and agents density criteria. Additionally, real scenarios enable to replay trajectories of some agents that we call replay workers in our pipeline.
In order to build a real driving scenario $\mathcal{S} = (\mathcal{M}, \mathcal{F}, \rho, H, \mathcal{G}, \Pi)$ from the interaction dataset we implemented a Scenario Database Editor whose main procedure is shortly summarize in the following. The composition of a scenario database is specified in a configuration file (scenario are scripted) composed of a list of blocks that each details the composition of a subset of scenarios. In each block, we first select the map $\mathcal{M}$ on which the scenario should be built then we select the track file from which the traffic flow $\mathcal{F}$ will be extracted. Given a simulation horizon H, we first search a traffic agent in the track file to play the actor. This traffic agent should last at least H seconds in the recordings. Once the actor is found, we extract all traffic agents recorded during the H seconds of the simulation. At this level, we have the trajectories of each traffic agents of the driving scenario and we extract the last points of their trajectories to define their goal $g \in \mathcal{G}$. Finally, a driving policy $\pi \in \prod$ is assigned to each traffic agents, workers and actor, according to the scenario specifications. For instance, while the actor is assigned a learnable policy, workers can be either DICM,CIDM or replayed agents. Each real driving episode can be represented as depicted in fig.2.10 based on all agents trajectories with color gradients on top of the road-network. For this work, we mainly focused on three kind of road-networks of the Interaction dataset [214]: a roundabout called $DR\_DEU\_Roundabout\_OF$, a map with lane merging $DR\_DEU\_Merging\_MT$ and a complex intersection $DR\_USA\_Intersection\_EP0$. We realised experiences separately on each of them since expert demonstrations are significantly different. We restrict our experiences on those maps to limit the computational load of trainings. In case multiple maps are used simultaneously for training, we need to collect huge amount of transitions (At least more

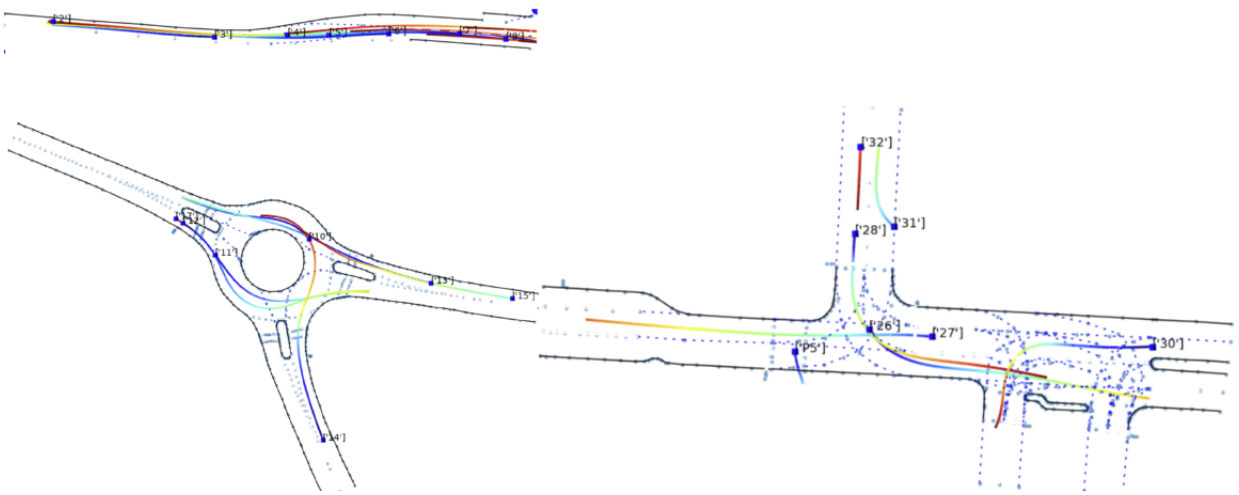

Figure 2.10: Driving episodes on three different maps used in our work: *DR_DEU_Roundabout_OF*, *DR_DEU_Merging_MT* and *DR_USA_Intersection_EP*0.

than 500K per training iterations) to guarantee monotonic improvement of our policy which implies waiting for several days before obtaining final results[14].

2.3 Conclusion

In this chapter, we proposed a general and scalable approach for learning driving policies for traffic simulation that decompose the problem in multiple stages with different levels of complexity. Our goal is to tackle the first step that consists in acquiring basic driving skills similar to human drivers. To this end, we first explained how we structured our driving policy with a routing module that conditions a maneuver planner that can take action in simulation. In order to evaluate our model, we designed our own driving simulator that can load real world scenarios and can be animated by different categories of rule agent agents for interactive training.

[14]More engineering work is necessary to fully exploit the potential of Ray and Rllib with distributed training and simulations, and devices comparable to the one used in DOTA 2 [12] may be required.

Chapter 3

Driving policy architectures

Contents

Figures

Tables

In this chapter, we study how to design the maneuver driving policy that compose our hierarchical driving policy. We first explore how to configure the decision making process and how to efficiently represent the driving scene in sec.3.1. In a second time in sec.3.2, we analyse open loop and closed loop performances of our driving policies built with different neural network architectures and trained with supervised learning to imitate experts short term plans.

3.1 Driving policy design

In chap.2, we explained that the maneuver policy has to exploit the reference path p_t provided by the routing module and the current scene observation to take action. We first explain how to parameterize actions in sec.3.1.1 before detailing how we represent the local driving scene in the observation in sec.3.1.2.

3.1.1 Action space

In this section, we study how the maneuver driving policy should represent the action for sequential decision making. We first explain in sec.3.1.1.1 how the maneuver policy is structured and what are the common action space in traffic simulation in sec.3.1.1.1 before explaining how to use curvilinear coordinates to specify the action in sec.3.1.1.2.

3.1.1.1 Encoding a maneuver

The maneuver policy receives the current observation o_t and the traffic free reference path to follow and should infer the action to perform for the next decision step that last δt_d. The action should be consistent with the latent goal encoded in the traffic free reference path p_t and socially consistent given the context provided in o_t. Since this mapping requires interpretation of the driving scene which may evolve in various ways, neural networks appears as a natural solution. The maneuver policy is hence decomposed in two consecutive blocks : $\pi_\theta = h_{\theta_h} \circ b_{\theta_b}$. The backbone first encodes (o_t, p_t) into a latent representation $z_t = b_{\theta_b}(o_t, p_t)$ as detailed in the next section 3.1.2 and in a second time a policy head infers the action $a_t = h_{\theta_h}(z_t)$. This decomposition separates decision making from representation learning knowing that the latent representation z_t may be reused for various purpose[1]. In the following we analyse how we can represent a maneuver in a_t such that reinforcement learning can be applied downstream. In previous works, driving policies trained with reinforcement learning employed various action spaces [229, 210]. High level maneuver specifications like lane keeping, giving or taking the way, following a front neighbor can be used to define a finite action space but it requires specific

[1]In Reinforcement learning, we commonly use the same latent representation to decouple representation learning and policy or value learning[177]

controllers to perform each discrete action. Those solutions lack flexibility because each maneuver may be executed in various ways for various amount of time and the transition between maneuvers may not always be smooth. In contrast, it is also possible to output directly the full trajectory as done in [8] but this formalism is not adapted for Reinforcement learning since a trajectory is high dimensional $\mathbb{R}^N$ and exploring all of them at a given state is impossible. In motion planning for automated vehicle, plans are usually computed based on dynamical model of the vehicle before being sent to the control stack [53]. Since dynamical models can be difficult to discretize, simple kinematical model are usually preferred for simulation purposes. Various works in reinforcement learning for automated vehicles specify their action space based unicycle or bicycle kinematical model such that policy already integrate limitation of displacement. One convenient specification is to use longitudinal acceleration and brake and control the turn rate the vehicle similarly to trajectory prediction [164, 191]. At each time step the action $a_t = [a_t, \delta_t]$ represents linear acceleration a_t and turn rate δ_t. In simulation, the unicycle kinematical model is discretized and the trajectory is updated at each time step by integrating the action from the previous position at time $t - \Delta T$.

$$v(t) - v(t - \Delta T) = \int_{t-\Delta T}^{t} a(\tau)d\tau \tag{3.1}$$

$$\theta(t) - \theta(t - \Delta T) = \int_{0}^{t} \delta(\tau)d\tau \tag{3.2}$$

$$x(t) - x(t - \Delta T) = \int_{t-\Delta T}^{t} v(\tau).cos(\theta)d\tau \tag{3.3}$$

$$y(t) - y(t - \Delta T) = \int_{t-\Delta T}^{t} v(\tau).sin(\theta)d\tau \tag{3.4}$$

Leveraging a model based approach, it is possible to discretize the unicycle kinematical model and to plan several action for an horizon h_p instead of just a single step. We explain in alg.8 how to implement a fully differentiable planner. For numerical approximation we can use the midpoint approximation of the speed and heading as done in [191] for computing the next position $(x(t), y(t))$:

$$\widetilde{\theta}(t) = \theta(t - \Delta T) + \frac{\delta(t - \Delta T)}{0.5 * \Delta T} \tag{3.5}$$

Even if we can back-propagate through the whole plan, note that the model does not integrate the dynamics of other agent which restricts direct application of this fully differentiable planner to simple driving scene with few traffic interactions. Additionally, actions $[a_t, \delta_t]$ are not defined relative to a route but relative to the current configuration of the vehicle. This is particularly embarrassing during exploration because the maneuver policy is not guided or directed by prior knowledge of the task : the fact that we know exactly which route to follow. In most of the cases once the path to follow is specified, we do not want to follow another one and should stay close to it, except if the situation makes this path irrelevant which should be detected by a separate module. Note that the reference path provided by our routing module is just a spatial

representation: there is no temporal information provided in p_t : the purpose of the maneuver policy is to compute decisions step by step to build a trajectory around the path. In the next section, we propose a method that remains continuous but which expresses actions relative to the traffic free reference path p_t.

Algorithm 8 Planning with discretize unicycle

INPUTS:

- H: planning horizon
- ΔT : time discretization
- $x(t), y(t), \theta(t), v(t)$ initial state
- $a_{t:t+H} = [a(t), a(t+\Delta T), ..., a(t+H)$
- $\delta_{t:t+H} = [\delta(t), \delta(t+\Delta T), ..., \delta(t+H)$

for t in $[\Delta T, 2.\Delta T, ..., H]$ **do**:
  $\widetilde{v} = v(t-\Delta T) + \frac{a(t)}{0.5*\Delta T}$
  $\delta_{cap} = cap(\delta(t-\Delta T)$
  $\widetilde{\theta} = \theta(t-\Delta T) + \frac{\delta_{cap}}{0.5*\Delta T}$
  $x(t) = x(t-\Delta T) + \widetilde{v}.cos(\widetilde{\theta}).\Delta T$
  $y(t) = y(t-\Delta T) + \widetilde{v}.sin(\widetilde{\theta}).\Delta T$
  $\theta(t) = \theta(t-\Delta T) + \delta_{cap}.\Delta T$
  $v(t) = v(t-\Delta T) + a(t-\Delta T).\Delta T$
end for=0

3.1.1.2 Relative displacement in curvilinear coordinates

In order to benefit from the prior information brought by the traffic free reference path, we leverage curvilinear coordinates to parameterize our action. Before detailing our method, we first remind some essential requirements for the action space such that a safe and human like driving policy can be learned. Learning human driving policy requires continuous action space otherwise arbitrary expert trajectory can not be reproduced with arbitrary accuracy. Concerning safety, it is essential to avoid collisions which mainly requires to control forward longitudinal displacements along the path to follow such that the vehicle only smoothly commit to the intersection when the path is free. On fig.3.1, we explained how we configure our action for the ego agent in red with respect to the traffic free reference path $p(g_e)$ that leads to the destination denoted g_e. The ego agent is located at X_t^a and its traffic free reference path $p(g_e)$ is represented in black. The path can be very long and in practice the next action only exploit a piece of it that we represented in violet and that we call the command C_t. Note that the command is composed of points that constitute the traffic free reference path except the first point of the command c_0 that is the projection of X_t^a on $p(g_e)$. We denote the curvilinear coordinates of position X_t^a with respect to path $p(g_e)$ with the curvilinear abscissa s_t^a and the ordinate n_t^a given the Frenet-Serret frame $(O(s_t^a), \underline{t_t}, \underline{n_t})$. The ego agent controls

the longitudinal displacement ds_t^a and the next ordinate n_{t+1}^a at the position corresponding to abscissa $s_t^a + ds_t^a$ according to the local Frenet-Serret frame $(O(s_t^a + ds_t^a), \underline{t_{t+1}}, \underline{n_{t+1}})$. We choose this representation because if we express curvilinear coordinates with respect to the command C_t then the action $a_t = (ds_t^a, n_{t+1}^a)$ corresponds to the curvilinear coordinates of the next position desired X_{t+1}^a. Note that on the fig.3.1, the next position desired X_{t+1}^a corresponds to the next position of a reference trajectory represented in red. As explained before, the action space should enable to reproduce trajectories of experts which is possible with curvilinear coordinates in condition that the expert trajectory stays close to the reference path such that there exists a one-to-one mapping between the two curves. In practice, most trajectories of the Interaction Dataset [214] verifies this condition and what strongly matters is the granularity of the reference path which should be composed of numerous points such that the curvature does not explode when there are abrupt turns. Another problem occurs when the ego agent is ahead of the expert which should be imitated. In this situation, we can allow backward displacement to compensate the advance but the behaviour will turn highly unrealistic. Another possibility is to disable backward displacement and force $ds \in [0, ds_{max}]$ and let the agent adapting its strategy and potentially recover the expert trajectory a bit later in the episode. We choose this solution because our driving scenarios do not require backward displacements along the reference path. Concerning lateral control, we choose to directly control the lateral offset $, n_{t+1}^a$ rather than a shift dn_t because in any case it is necessary to infer the Frenet-Serret frame $(O(s_t^a + ds_t^a), \underline{t_{t+1}}, \underline{n_{t+1}})$ and especially the normal $\underline{n_{t+1}}$ at $O(s_t^a + ds_t^a)$ to understand how far will be the agent from the centerline. Since $a_t = (ds_t^a, n_{t+1}^a)$ only controls the next position, the heading of the vehicle is not fixed. Instead of learning a maneuver policy that outputs $(ds_t, n_{t+1}, d\theta_t) = \pi_{maneuver}(\pi_{planner}(o_t, p_t))$, we chose to use the heading defined by the tangential vector $\underline{t_{t+1}}$ to update the ego agent heading at the next step θ_{t+1}. This approximation is reasonable as long as the curvature of the path is small and the lateral position n_{t+1} is not too big. Controlling a heading knowing that we do not use the bicycle kinematical model to update the state (x, y, θ) is purely artificial and does not bring real skills. Indeed the point of our action space is to decouple longitudinal and lateral control to ease exploration during reinforcement learning trainings.

3.1.2 Observation space

In this section, we explain how we encode the driving scene context such that the maneuver planner can infer the appropriate action. We start to review how driving scenes are commonly represented in autonomous driving in sec.3.1.2.1, thereafter we propose a first method to encode the scene in sec.3.1.2.2 and we explain in a last section 3.1.2.4 how to improve the representation of other agents contexts with a pointNet like architecture.

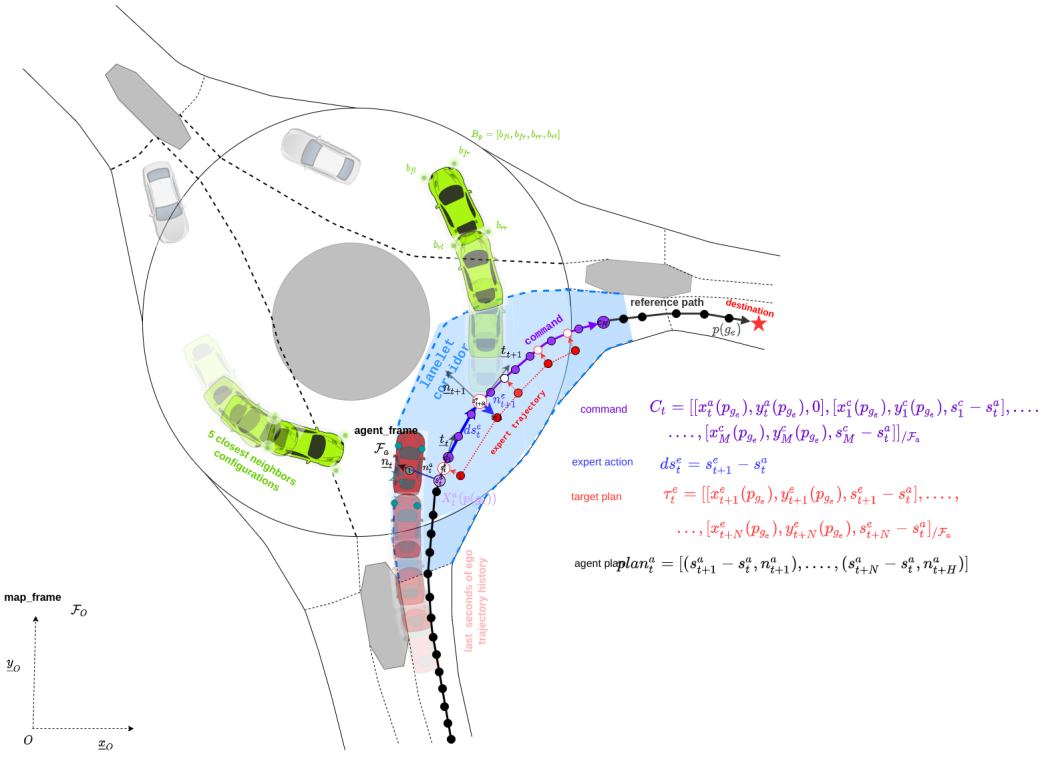

Figure 3.1: Action space in curvilinear coordinates

3.1.2.1 Driving scene representation

We aim to infer the appropriate action $a_t = (ds^a_t, n^a_{t+1})$ based on the path p_t and on a representation of the driving scene context encoded in an observation. Multiple elements of the scene influence the decision and should consequently be integrated in the observation. In the following, we explained how we designed our observation space. As explained in sec.3.1.1.2 any action ds^a_t, n^a_{t+1} is expressed with respect to the command polyline C_t whose points are necessary to infer the next position (x_{t+1}, y_{t+1}). The command should consequently be integrated in the observation but before understanding where an action leads, it is desirable to understand where the agent is allowed to go. Providing information about the local road-network is essential but the level of description can highly impact simulation performances[2]. Rasterized representation of the road-network with a top view perspective are very popular but their accuracy is limited to the image resolution and rendering engines tend to slow down parallel simulation rollouts critical for reinforcement learning [94]. We choose sparse representations based on points as polylines or polygons to represent elements of the road similarly to other works in trajectory prediction [48, 110]. In addition to the command, we provide the lane corridor in which the agent is expected to move during the next decision steps given the command as depicted in blue in fig.3.1. The lane corridor is built with the successive lanelets that are associated to the traffic free reference path. Indeed, the route that leads to the goal is computed with the lanelet graph as explained in chap.2 so the path built upon the centerline of lanelets has an associate sequence of lanelets. We concatenate left and right borders of the consecutive lanelets to build

[2]Speed and memory consumption

left and right borders of our lane corridor. While the lane corridor indicates where the ego agent should move, it does not indicate where other agents are going or at least where they could move. Bringing more information about the road structure can be crucial especially on intersections where other agents can come for the left or right side. Even if other agents goal are not accessible at least from the point of view of the real decision maker, the knowledge about the road-network topology offers a strong inductive bias. Recent histories of neighbors also condition their next displacements and complement the knowledge of the road topology as shown in various works in the field of trajectory prediction [51, 48, 110]. We choose to consider only the five nearest neighbors of the ego agent because most of the interactions on the driving scenes considered do not require more than five agents at a time. For decision making, the ego agent history also matters because other agent may already be influenced by it and ego agent should control its speed based on its past trajectory in order to avoid abrupt acceleration or braking. Since the ego agent should avoid collisions with its neighbors, it should also integrate the information about its own spatial extension as well as other agents spatial extensions that we call footprints[3]. Lastly some meaningful quantities as the current lateral position n_t, the previous longitudinal speed ds_{t-1} and some indicators as off-road driving i_{off} indicators can be useful for decision making.

3.1.2.2 Encoding vectorized observation

In the previous section, we listed which elements of the driving scene context should be integrated to the observation such that the observation backbone can compute a latent representation $z_t = b_{\theta_b}(o_t, p_t)$ for decision making. In the following, we propose a first architecture for b_{θ_b} where each components is first embedded separately by encoders $E_L, E_c, E_{nh}, E_{ec}, E_h, E_f$ as shown in fig.3.2 before being aggregated through a last module called aggregator. All purely geometric components as the lane corridor $L = [[L_j = [(x_i, y_i)]_{i\in[1,20]}]_{j\in[r,l]}$, the agents footprints $F = [(X_{rl}, X_{fl}, X_{rr}, X_{rl})_i]_{i\in[ego,1,\dots,5]}$, and the command $P = [(x_i^p, y_i^p)]_{i\in[1,51]}$ are encoded with separate networks that share the same structure. Note that all geometric points as well as agents positions are beforehand normalized according to spatial bounds specified in the observation space. We get inspired by pointNet architectures that operates on set of geometric points and encode spatial features [152]. Similarly to pointNet, we compute local features for each point and global features for the whole entity and we finaly combine them with a MLP to obtain the final embedding of the geometric entity as depicted on the right in fig.3.2. All agents footprints embeddings are concatenated in increasing order of their distance with respect to the ego agent to form the footprints embedding e_f. Ego agent history and other agent histories are encoded with two separate network that share the same structure : masked position are first projected in a higher dimensional space and then provided to a GRU module [26] which generates history embeddings e_{nh} and e_h. Note that we choose to initialize the hidden state of the GRU with zeros instead of the hidden state at the previous decision step to avoid back-

[3]The footprint is just the four corners of the polygons that defines the vehicle's shape.

propagation through time. History embeddings of each agents are concatenated in increasing order of their distance with respect to the ego agent to form neighboors histories embedding e_{nh}. Concerning the ego agent state, it is encoded in e_{ec} with a simple MLP .
In order to combine all embeddings vectors $e_L, e_c, e_{nh}, e_{ec}, e_h, e_f$ we use a network called aggregator in fig.3.2 which can be implemented with two different architectures. The first one that we called FCBaseline, consists in simply concatenating all embeddings and applying a MLP on top to extract the latent observation vector z_t.

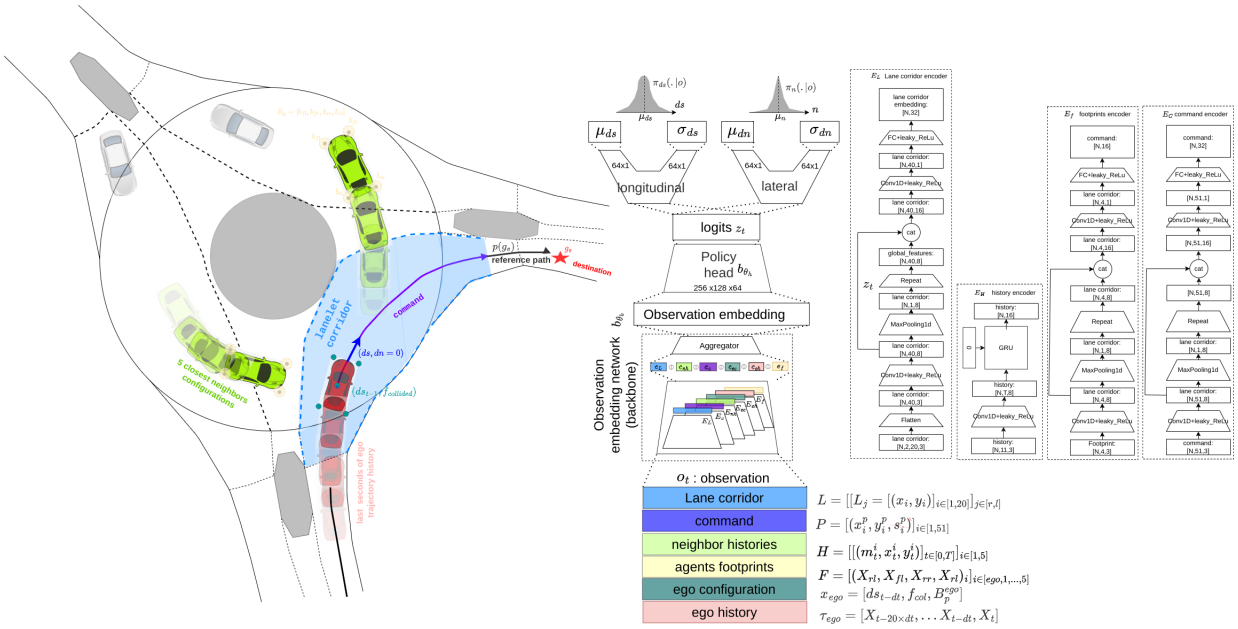

Figure 3.2: Neural network architecture of a basic driving policy with the lightest observation backbone called FCBaseline.

3.1.2.3 Aggregating multiple components

While the FCBaseline architecture offers a simple solution to combine scene components information, it does not enable to understand which components matters for the decision making. Previous works applied spatial attention mechanism to build policy network that actively select important, task-relevant information from visual inputs [128]. The model generates attention maps which can uncover some of the underlying factor used to solve the task. Attention was also used in a multi agent setting to better extract relation between agents that need to cooperate to win a football game [171]. Those applications are similar to ours where we need to interpret vectorized scene components to take actions. We leverage the multi head attention mechanism [193] to build the second architecture for the aggregator that we called $FCBaseline\_Attentive$. More specifically, we use cross attention to build N specific queries based only on the following components $e_L, e_c, e_{eh}, e_{ec}, e_{ef}$ which relates to information of the ego agent as well as the lane corridor while keys and queries are computed based on all embeddings. This enables to understand how the next action that depends on the command and ego information should be shaped based on the whole context. In order to build querries, keys and values from component

embeddings, we proceed the following way. We start to reshape all embeddings into vectors of same size equal to 16 as depicted in fig.3.3 and stack them to form a matrix of size: $[17x16]$. From this matrix, we extract the query matrix $Q \in \mathbb{R}^{7\times16}$ which is composed of the seven first terms while the keys $K \in \mathbb{R}^{17\times16}$ and the values $V \in \mathbb{R}^{17\times16}$ contains all component embeddings. The multi head attention layer denoted MHA operates as follows with $h = 8$ heads:

$$MHA(Q, K, V) = cat(h_1, h_2, ..., h_h).W^O \tag{3.6}$$

$$where\ W^O \in \mathbb{R}^{h.d_k\times d_m} \tag{3.7}$$

$$h_i = Attention(Q.W_i^Q, K.W_i^K, V.W_i^Q) \tag{3.8}$$

$$W_i^Q \in \mathbb{R}^{d_m\times d_k} \tag{3.9}$$

$$W_i^K \in \mathbb{R}^{d_m\times d_k} \tag{3.10}$$

$$W_i^Q \in \mathbb{R}^{d_m\times d_k} \tag{3.11}$$

$$Attention(Q, K, V) = softmax(\frac{Q.K^T}{\sqrt{dk}}).V \tag{3.12}$$

$$d_k : 32, d_m : 16 \tag{3.13}$$

The output $MHA(Q, K, V)$ is then added to initial querries Q to obtain the final latent observation z_t. Note that we applied layer normalization on keys, querries and values as in [171] contrary to transformers. Since the order of the stacked semantic components does not change at each decision step we do not apply positionnal encoding which is often used for language processing where words order contains semantics.

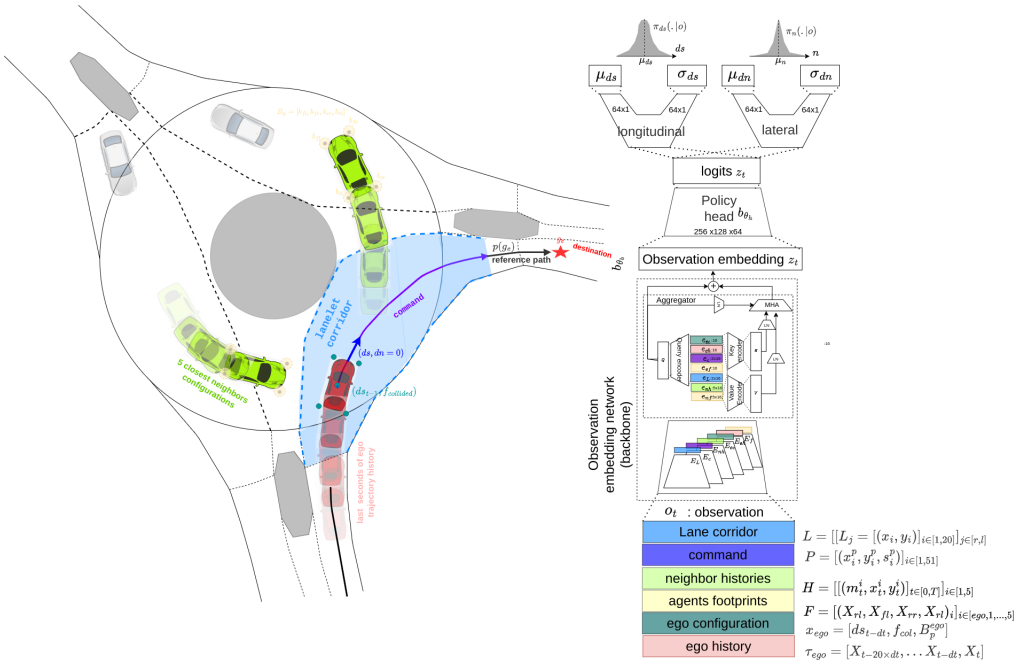

Figure 3.3: Neural network architecture of a basic driving policy with an observation backbone called $FCBaseline\_Attentive$ that uses an aggregator implemented with multi head attention

In sec.3.2.1 we will analyse how the attention mechanism affects the task performances and

we will also investigate if interpretation can be made on the attention weights.

3.1.2.4 Encoding the local map

In the two previous sections, we proposed an architecture that encodes the scene context from the point of view of the ego agent. We used the same architectures to encode spatial features but separate networks were used to encode the command, agents footprints and the ego agent lane corridor. Those architectures do not enable to relate spatial components with each other because embeddings are built separately. Additionally, the representation of the road-network is restricted to the lane corridor which makes difficult for the ego agent to predict how neighbors are likely to move. This can be problematic on intersections where the ego agent should sometime learn to give or take the way. In this section, we propose a second architecture that enables to unify and relate all observation components. This architecture is based on VectorNet [48] which uses a hierarchical graph neural networks to encode the scene decomposed in multiple geometric entities such as as polygons or polylines themselves composed of elementary bi-points. Since the road-networks of our driving scenarios are based on the lanelet map format whose elementary components are points and linestrings, we adapted VectorNet to work directly on sequence of points. Similarly to VectorNet, we use a hierarchical structure that first builds local features for each polylines before exchanging information at a global level between each of them. Features of the global interaction graph will be reused later to decode the next actions for the ego agent.

Before detailing our architetcure, we introduce some modifications in the observation where we replace the lane corridor built upon a sequence of consecutive lanelets, with the set of lanelets that surround the ego agent in a radius lower than $R = 20meters$. The set of lanelets contains the ones that constitute the lane corridor but also incorporates lanelets on which neighbors may move during the next decision steps. For each lanelet, we not only represent left and right borders as polyline but also the associate centerline. Hence all components provided in the observation expect the ego state which is not processed at this level reduced to a set of polylines for geometric entities : lanes, agents footprint, command and agents histories. Polyline are sampled at regular spatial interval for static entities and at a fixed period of 100ms for trajectories.

The new observation backbone called $PointNet\_MHA$ that outputs the latent representation $z_t = b_{\theta_b}(o_t, p_t)$ operates at two stage : it first exploits local subgraphs to extract polylines features and then use a global interaction graphs to relates all polylines with each others. Each polyline defines a local subgraph and each point of the polyline constitute a node which is connected with all other points of the polyline. To build polyline feature we stack three subgraph layers that each transforms node features $v_1^{(l)}, v_2^{(l)}, ...v_{N_{max}}^{(l)}$ of polyline P_i at level l into node features $v_1^{(l+1)}, v_2^{(l+1)}, ...v_{N_{max}}^{(l+1)}$ as follows:

$$v_i^{(l+1)} = \varphi_{rel}(g_{enc}(v_i^{(l+1)}), \varphi_{agg}(\{g_{enc}(v_j^{(l)})\})) \tag{3.14}$$

where $v_i^{(l)}$ is the node feature for the l-th layer of the subgraph network, and $v_i^{(0)}$ is the initial node feature characterized in terms of relative position with respect to ego agent. The function $g_{enc}(\cdot)$ transforms the individual node features while $\varphi_{agg}(.)$ aggregates the information from all neighboring nodes with a max pooling operator, and φ_{rel} is the relational operator between node v_i and its neighbors implemented as a concatenation as depicted in fig.3.4. Finally, to obtain polyline level features, we compute $p = \varphi_{agg_{pol}}(\{v_i^{(L_p)}\})$ where $\varphi_{agg_{pol}}(.)$ is again max-pooling. To model global interactions between node features $\boldsymbol{P}^{(l)} = p_1, p_2, ..., p_P$ we use another GNN that considers full connectivity ($\mathcal{A}$ adjacency matrix is full of ones) of the polylines graph.

$$\{p_i^{(l+1)}\} = GNN(\boldsymbol{P}^{(l)}, \mathcal{A}) \tag{3.15}$$

The GNN can be implemented with self attention such that the future trajectory $\tau_{t:t+T} = \varphi_{traj}(p_i^{(L_t)})$ from the polyline nodes corresponding to the ego agent $p_i^{(L_t)}$ at the last layer L of the global graph, can be decoded with φ_{traj} as in [48] .

$$\boldsymbol{P}^{(l+1)} = softmax(\boldsymbol{P}_Q.\boldsymbol{P}_K^T).\boldsymbol{P}_V \tag{3.16}$$

where $\boldsymbol{P}_Q = \boldsymbol{P^{(l)}}.\boldsymbol{W}_Q$,$\boldsymbol{P}_K = [\boldsymbol{P^{(l)}} \bigcirc \boldsymbol{t}].\boldsymbol{W}_K$ and $\boldsymbol{P}_V = \boldsymbol{P}^{(l)}.\boldsymbol{W}_V$. In our setting, we do not want to infer a trajectory but the next actions conditioned on a reference path represented by the command. In contrast to the Urban Driver[165] our action is not specified relative to ego position in cartesian coordinates but with respect to the reference path in curvilinear coordinates. Therefore, we choose to build the query matrix $\boldsymbol{P}_Q$ based on three polyline features : the ego agent history, the ego agent fooprint and the command. We expect the query to provide meaningful information with the latent observation z_t that will later be used to infer the action. Note that similarly to the Urban Driver, we choose to concatenate the type of each polyline in addition to the polyline features when the keys are built $\boldsymbol{P}_K = [\boldsymbol{P^{(l)}} \bigcirc \boldsymbol{t}].\boldsymbol{W}_K$ such that polylines of different vehicles trajectories, polyline of static entities can be distinguished.

3.2 Imitating human drivers

In this section, we aim to analyse if our observation backbones are powerful enough to learn a driving policy. Before applying reinforcement learning, we propose to analyse the efficacy of our architectures on a supervised learning tasks. Since supervised learning can also be considered at a pretraining step, we propose a method to acquire general representations before applying RL. We propose to pretrain our neural networks to plan multiple decisions instead of a single one. We explore different ways of planning in a model free fashion in section 3.2.1 or with a transition model in sec.3.2.2 and we analyse their performances in open loop and closed loop.

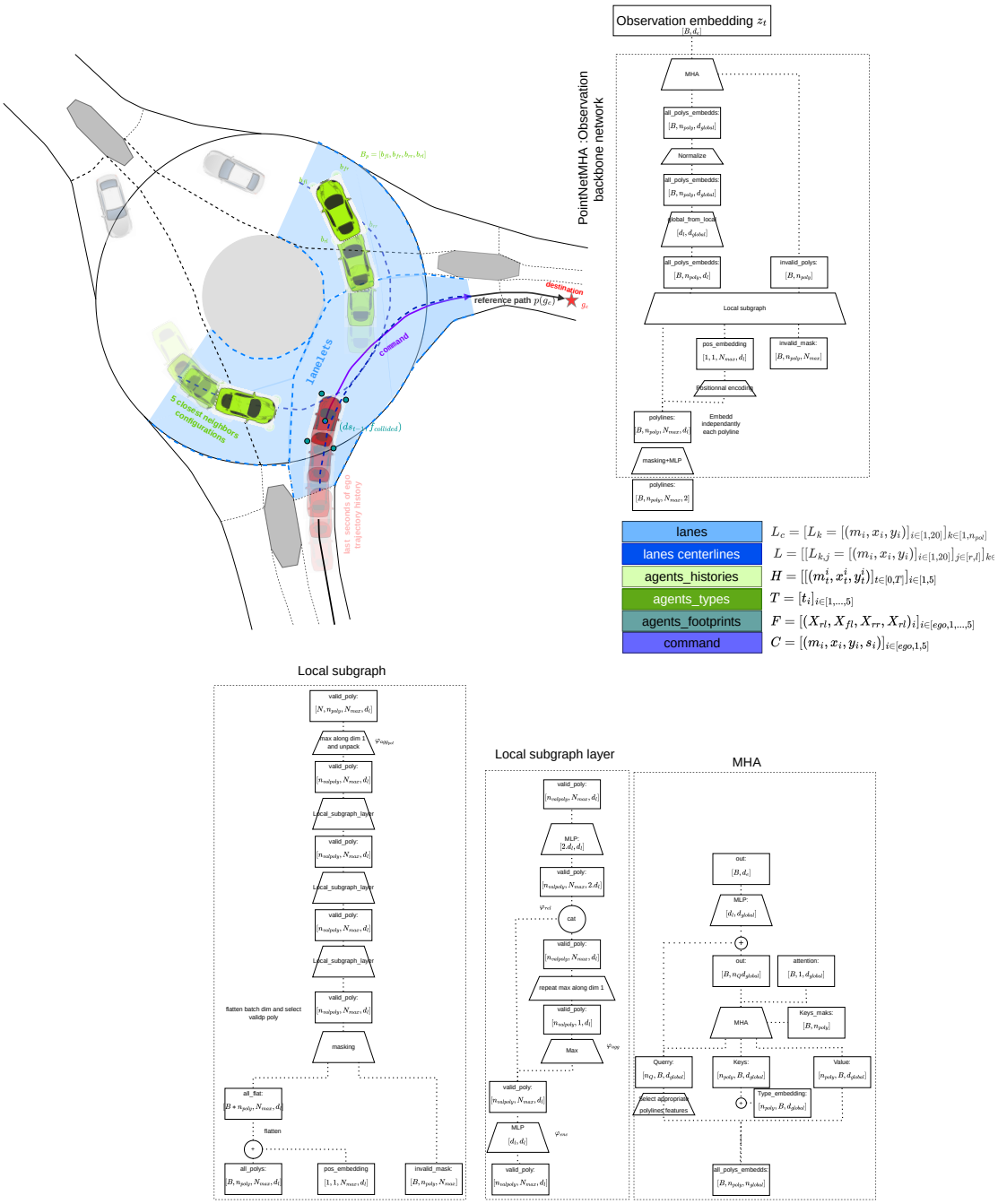

Figure 3.4: Encoding the local scene context with PointNetMHA observation backbone network

3.2.1 Imitative planning

In this section, we first study how to pretrain our networks based on supervised learning in sec.3.2.1.1 and we later propose a planner architecture that can imitate short term trajectories of experts in sec.3.2.1.2.

3.2.1.1 Extending decision making

Training a high capacity model tabula-rasa from random initialization with reinforcement learning such that it outputs robust decision without using any prior knowledge about the environ-

ment seems inefficient [140]. Taking appropriate decisions requires good scene understanding which cannot be acquired when the RL agent starts to learn with a single scalar reward. An action should be characterized by the way it influences state transition therefore state representation that aims to compute actions should account for future state evolution but usually do not for model randomly initialized [22]. The fact that RL optimization objectives can be biased and very noisy due to exploration can consequently rapidly lead to over-fitting because observation embedding get overspecialized for learning a policy with high expected return but not necessarily a policy that performs always well everywhere [216]. Pretraining an encoder $z_t = b_{\theta_b}(o_t, p_t)$ in RL is a common practice that can alleviate over-fitting [37, 208]. Unsupervised pretraining from downstream tasks is common for visual applications and provides task agnostic features which can be used for training [88]. Some pretraining methods also rely on predictions tasks either on some part of the state [103] either in a latent space leveraging amortized variational inference in POMDPs [102]. In multi agent environment, predicting the evolution of the scene is difficult because other agents may behave in very specific way in each situation. In case other agents are replayed in the driving scenario, learning the dynamic only helps locally when the ego agent is not too far from the reference expert. Even if learning reactive behavior with RL still help to avoid collision in this setting, what really matters for long term simulations in presence of replay agent is to closely imitate expert strategy in the long term. In this way the ego agent can not only react to other traffic agent but will also avoid being trapped in some situation far from the expert trajectory where no adaption may be possible. Based on these considerations we propose a pretraining method that consists in planing multiple decisions instead of a single one which is equivalent to apply multi step conditional imitation learning [63]. Since the observation backbone is used to plan multiple decision steps, the features learnt for the first decision will also be refined for taking consistent future decisions and should consequently get more robust to observation artefacts. However we do not intend to use the full plan for closed loop simulation except for the first decision step because at each simulation step we will replan and adapt to unexpected changes.

$$\pi_\theta(o_t, p_t) = h_{\theta_h} \circ b_{\theta_b}(o_t, p_t) = h_{\theta_h}(z_t, p_t) = [a_t, ..., a_{t+H}] \tag{3.17}$$

Contrary to imitative models, we learn directly an imitative planner and not a predictive model of expert like behaviors that requires an additional optimization step in a latent space to build a path that lead to the goal [158].

3.2.1.2 Planning as an expert

In order to extend the single step decision making we choose to plan the trajectory that the ego agent has to follow given the current observation o_t and the traffic free reference path p_t. Since we aim to learn a representation that improves the performances of our agents in closed loop evaluation, the first planning step should corresponds to the action that the agent

will effectively execute in the environment. We expect that pretraining our agent to plan as an expert from offline data, can ease downstream reinforcement learning training with online interactions. Instead of directly planning with a sequential procedure that requires next states predictions or a latent representation [60, 59], we were inspired by simpler models in the motion prediction literature. In order to predict trajectories of some vehicles in a driving scene, some works condition their predictions on anchors $\boldsymbol{a}^k$ that can be either paths [20] or goal locations [226] and then operate trajectory prediction assuming that future states $s_{t:t+H}$ are conditionally independent from each other. To be more specific, in Multipath [20] they made the simplifying assumption that uncertainty is uni-modal given intent $\boldsymbol{a}^k$, and they model the next state as a Gaussian distributions dependent on each waypoint a_t^k of an anchor trajectory:

$$\phi(s_t^k|\boldsymbol{a}^k, o_t) = \mathcal{N}(s_t^k|a_t^k + {}^k_t(o_t), \Sigma_t^k(o_t)) \tag{3.18}$$

Gaussian parameters $\mu_t^k(o_t), \Sigma_t^k(o_t)$ are predicted by the model for each time-step of each anchor trajectory a_t^k. In the Gaussian distribution mean $a_t^k + \mu_t^k(o_t)$, the term $\mu_t^k(o_t)$ represents a scene-specific offset from the anchor state a_t^k. This allows the model to refine the static anchor trajectories to the current context, with variations coming from specific road geometries. The time-step distributions are assumed to be conditionally independent given an anchor, i.e., they write $\phi(s_t^k|a^k, o_t)$ instead of $\phi(s_t^k|a^k, o_t, s_{1:t-1})$. This modeling assumption allows to predict for all time steps jointly with a single inference pass, making the model simple to train and efficient to evaluate. We adapted this approach to the planning setting where we want to plan the next trajectory to follow given a reference path. Since the first step of the trajectory should correspond to the first action we choose to represent the trajectory plan in curvilinear coordinates. The first action parametrize the longitudinal displacement ds_t with respect to the current curvilinear abscissa and the next lateral ordinate n_{t+1}. Instead of representing the next steps of the plan the same way we propose a slight modification for the longitudinal displacement $ds_{t+h'}$ which will always represent the displacement $ds_{t'}$with respect to the initial curvilinear absissa s_t such that $ds_{t'} = s_{t'} - s_t$

$$\pi_\theta(o_t, p_t) = h_{\theta_h} \circ b_{\theta_b}(o_t, p_t) = h_{\theta_h}(z_t) \sim [a_t, ..., a_{t+H}] \tag{3.19}$$

$$= [(ds_t, n_{t+1}), (ds_{t+1}, n_{t+2}), (ds_{t+H-1}, n_{t+H})] \tag{3.20}$$

This choice enables to avoid planning with respect to fixed positions a_t^k sampled uniformly on p_t which corresponds to the anchor $\boldsymbol{a}^k$ in Multipath[20]. Indeed, providing displacement in curvilinear coordinates with respect to different anchor points a_t^k at every decision step t may require to predict negative longitudinal displacement ds_t with respect to a_t^k which are not contained in our action space. Similarly to Multipath, all steps of the plan are considered

independent with each others and consequently from previous steps.

$$p(a_t, ..., a_{t+H}|o_t, p_t) = \prod_{h=0}^{H} p(a_{t+h}|, a_{t+h-1}..., a_t, o_t, p_t) \tag{3.21}$$

$$= \prod_{h=0}^{H} p(a_{t+h}|, o_t, p_t) \tag{3.22}$$

$$a_{t+h} = [ds_{t+h}, n_{t+h+1}] \tag{3.23}$$

$$ds_{t+h} \sim \mathcal{N}(.|\mu_{ds_{t+h}}(o_t, p_t), \sigma_{ds_{t+h}}(o_t, p_t)) \tag{3.24}$$

$$n_{t+h+1} \sim \mathcal{N}(.|\mu_{n_{t+h+1}}(o_t, p_t), \sigma_{n_{t+h+1}}(o_t, p_t) \tag{3.25}$$

$$\tag{3.26}$$

Note that each step of the plan is computed with the same observation embedding z_t as represented in fig.3.5 which helps to compute consistent sequence of displacements from only the current observation. Since each planning step is expressed with curvilinear coordinates $(ds_{t'}, n_{t'+1})$ with respect to p_t it is not easy to make the relation with the associate location in cartesian coordinates $(x_{t'}, y_{t'})$. In order to favor spatial representations that are critical for decision making, the reference path p_t is provided in the form of a polyline with a sequence of cartesian coordinate as well as its curvilinear abscissa in addition to the command component C_t already embedded in z_t as explained in sec.3.1.2.2.

$$p_t = [(x_0, y_0, s_0), ..., (x_{50}, y_{50}, s_{50})] \tag{3.27}$$

The coordinates are expressed with respect to the ego agent current frame and the initial curvilinear abscissa is $s_0 = 0$. Based on p_t and the complementary observation embedding z_t, the planning head outputs a vector that defines the means μ and the standard deviations σ of the longitudinal and lateral components of each of the fifteenth planning steps of the plan. Since decisions are assumed independent as well as the lateral and longitudinal components, we call our planner Independent Longitudinal Lateral (ILL) planner. In order to validate our assumptions and the efficiency of our architecture, we first train our ILL planner on demonstrations of two driving scenario databases $Huge\_R\_Basic$ and $Huge\_I\_Basic$ detailed in annexes.2. We want to understand if the planner is able to generalize expert like plans in curvilinear coordinates on new scenarios. In the following, we first propose to compare open loop performances of the planner with different observation backbones: $FCBaseline\_basic$, $FCBaseline\_Attentive$ and the $PointNet\_MHA$. We observe in tab.3.2.1.2 that the best performances are obtained with the $PointNet\_MHA$ architecture which has more capacity than other architectures and which also benefits from the information of the local road-network contrary to the two other baselines that only have access to the lane corridor. We note that the attention mechanism of the $FCBaseline\_Attentive$ architecture did not considerably improve the test performances compared to $FCBaseline\_Basic$. This can be explained by the fact that the databases are

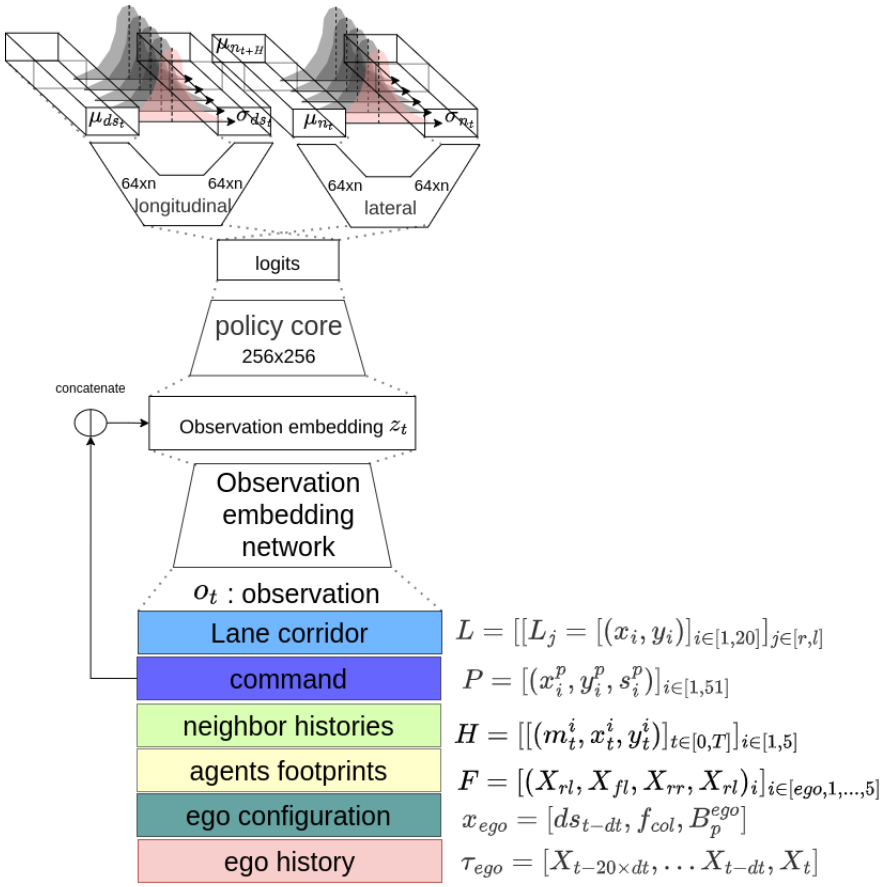

Figure 3.5: Architecture of the independent longitudinal and lateral planner named *Planner_ILL*: on the top we represented the bi-variate Gaussian distributions that defines consecutive steps of the plan where the first is represented in red and corresponds to the action effectively taken in simulation.

	Huge_R		Huge_I	
	ADE-1.5(m)	FDE-1.5(m)	ADE-1.5(m)	FDE-1.5(m)
FCBaseline_basic	0.28	0.38	0.27	0.45
FCBaseline_Attentive	0.26	0.32	0.24	0.41
PointNetMHA	0.23	0.26	0.21	0.37

Table 3.1: Open loop test performances comparison between several IIL planner with different observation backbones.

small because based on a single map with limited number of demonstrations. The restriction on the amount of data is only justified for making fair comparisons between the contribution of real demonstrations on a set of scenarios and the contribution of RL trainings on the same set of scenarios[4]. We posit that pretraining on a larger database would certainly be beneficial but it would become unfair to compare the contribution of supervised and RL trainings since RL is primarily used to compensate unbalanced amount and diversity of demonstrations. Finally, we observe that the smallest backbone *FCBaseline_basic* reached reasonable test performances which suggests that it could be used for downstream RL fine tuning in closed loop.

[4]Using scenarios from multiple maps implies to add several days of training with RL to obtain reasonable results which considerably slows down experiments.

In fig.3.2.1.2, we provide some qualitative test examples generated with $PointNet\_MHA$ which shows that the planner is understanding the scene context. At the entrance of the roundabout, the ego agent's plan shortens because the left agent already committed has the priority whereas the plan extends forward along the center-line when the lane is free. On the map with the intersection, we observe that the planner also interprets the context to take or give the way with respect to other neighbors.

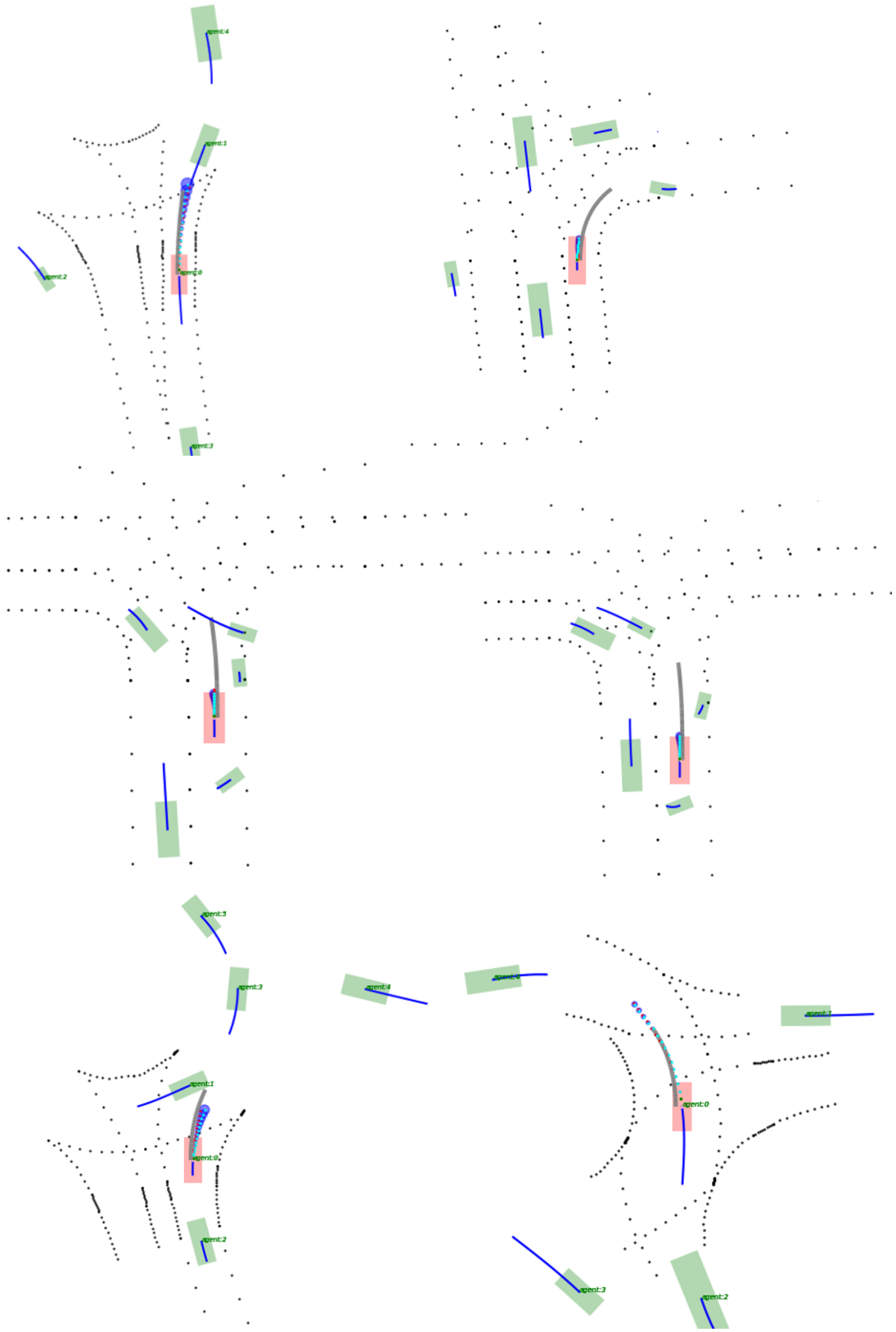

Figure 3.6: Qualitative results of plans generated by *PointNetMHA_Planner_IIL* in open loop by the ego agent represented in red. The observations provided to the backbone are represented on the figure. Plans represented in cyan are projected from curvilinear to cartesian coordinates with respect to the command represented in grey. The longitudinal uncertainty provided by the standard deviation is represented with circles in dark blue. The ground truth trajectory of the associate expert is represented in red.

We choose to focus our attention on a relatively small architecture : $FCBaseline\_basic$ since we intend to apply RL trainings downstream without freezing the backbone which may need to acquire specific features to accurately predict the policy state value or refine actions selection on new situations seen in closed loop simulations. Sophisticated backbone as $FCBaseline\_Attentive$ or $PointNet\_MHA$ requires considerably more interactions and hyper-parameter search to be properly trained with RL as shown in chapter. 4.
Given that the ILL planner is able to plan with reasonable performances in open loop [5], it is interesting to check how it behaves in closed loop : when the first action of the plan is executed at every decision step of the episode. In the next experience, we compare closed loop performances of our ILL planner between different backbones. In contrast to the previous results, we do not compute ADE-1.5 with respect to the expert plan, but we consider the average distance error along the whole episode with respect to the expert trajectory with ADE-5 and ADE-15. Additionally we also report the rate of episodes with at least a collision (CR%) as well as the relative time spent off-road during the episode (Off%) to understand if the policy effectively learned a safe driving strategy.

	Huge_R				Huge_I			
	ADE-5(m)	ADE-15(m)	Off%	CR%	ADE-5(m)	ADE-15(m)	Off%	CR%
FCBaseline_basic_BC	5.20	13.6	10.30	55	5.82	14.1	11	45
FCBaseline_basic_Planner_ILL	4.25	10.64	8	46	4.80	11.62	8.2	42
FCBaseline_Attentive_Planner$_{ILL}$	4.13	11.20	7.5	44	4.83	11.20	8.3	41
PointNetMHA_Planner_ILL	3.82	8.62	6.2	42	4.55	9.41	5.1	40

Table 3.2: Closed loop test performances comparison between several observation backbones.

The results provided in tab.3.2 shows that short term imitation performances as ADE-5 remain relatively low even if the agent is already starting to drift but the long terms imitation performances confirm that the driving policy ends up very far from the expert trajectory and finally fail to adapt as shown by high level of collisions and off road driving. This problem is known as the distributional shift [30, 176, 43] and was mainly pointed out in behavioural cloning which aims to learn single step decisions from expert demonstrations. We also trained a baseline only on the first decision of the expert plan similarly to behavioural cloning and we reported its closed lop test performances in the first row of tab.3.2. We observe that this baseline reached lower results that our planner trained on 15 steps which proves that better features are acquired by our training method compared to standard behavioural cloning. However the $ILL\_planner$ is still suffering from gradual deviations which lead to failure and that cannot be avoided. To illustrate the deficiencies of the ILL planner in closed loop, we provide qualitative results in fig.3.2.1.2 with representative failures that occur in the test database. All situations are considerably different from states visited by the expert which can explain that the policy struggles to adapt on those new situations.

[5] The observation used to process the plan comes from the expert observation,(equivalently state) distribution.

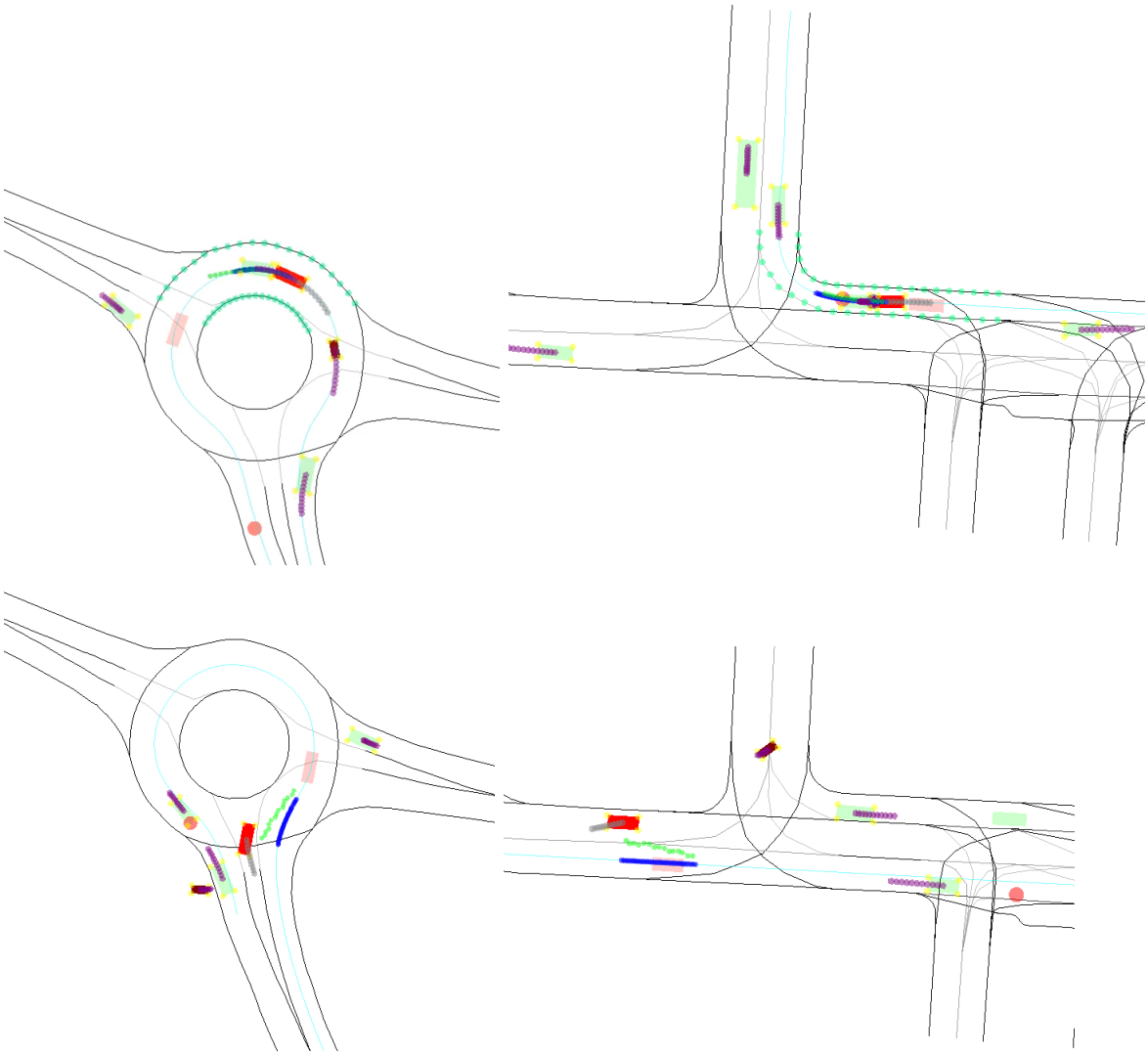

Figure 3.7: Qualitative results of plans generated in closed loop by the *FCBaseline_Planner_ILL*

3.2.2 Model based planning

In the previous section, we explained how to extend single step decision making into planning but our method did not explicitly model the evolution of the ego agent state under the decision making process. This limitation reduces the ability to generalize how to plan outside the expert state distribution and resulted to numerous failures due to distributional shift. We first propose in sec.3.2.2.1 to integrate a transition model in the planner such that it can better predict state evolution and not only optimal actions. We subsequently explain in sec.3.2.2.2 how the planner can leverage additional simulation data during pretraining in conjunction with expert demonstrations.

3.2.2.1 Integrating a transition model

In order to better generalize the impact of consecutive actions on the ego agent future states, we can integrate a state transition model in the planning process. The state transition model should ideally predict the next state s_{t+1} reached by the agent when it takes action a_t in state s_t. In the previous section, all the steps of the plan were computed jointly and they are only linked implicitly based on the observation embedding z_t while in this section we make explicit the fact that action a_{t+h} depends on actions $a_{t:t+h-1}$. The main difficulty of modelling the state transition comes from the fact that the trajectories of other agents surrounding the ego agent need to be predicted. Indeed, the ego agent state or simply its current observation contains information relative to neighbors locations which influence ego agent's decisions. Another minor difficulty comes from the fact that actions in curvilinear coordinates needs to be converted into cartesian coordinates to obtain the future state location and the analytical expression which depends on the reference path p_t should be learnt. In order to avoid to predict all the details of the next state s_{t+1} that do not all necessarily matter for the next decision, our transition model operates in a latent space. As depicted in fig.3.2.2.1, we introduce an auto-regressive planner that generates each action based on the prediction of the current latent state and on the previous action. At the beginning of the planning procedure, the observation embedding is concatenated with the flattened command vector C_t and the ego agent configuration before being embedded into an initial latent state h_o. Note that the ego agent configuration (s_0^c, n_0^c) is defined with respect to the reference path p_t whose curvilinear abscissa origin is taken at the current position of the ego agent hence $s_0^c = 0$ and $n_0^c = n_t$. We choose to concatenate the command and the ego configuration to facilitate the computation of the plan which can easily benefits from the initial lateral offset n_t as well as the curvilinear abscissa of each waypoints of the command. Once a latent state h_t is available we apply a first transformation on h_t to obtain a policy latent state h_t^π that also depends on the previous action a_{t-1}. The policy latent state is later decoded by a policy head which provides the next action a_t to execute through a sampling operation. In order to predict how the action a_t influences the next state, we use a second GRU cell [26] that takes as input a_t and that update h_t^π into the next latent state h_{t+1}. Since we have a new latent state h_{t+1}, we can repeat the procedure to obtain each action of the plan that we store as a vector. In contrast to the previous section, our auto-regressive planner makes each decision a_t based on the previous action a_{t-1} and on a latent representation of the state h_t. The policy latent state denoted h_t^π is just an intermediate representation that enables to derive the next action a_t before predicting the next latent state. Since each action is parametrized by a bi-variate gaussian for the longitudinal displacement and the lateral offset, we can train the planner by maximizing the likelihood of the expert plan as done previously.

$$argmax_\theta \mathbb{E}_{o_t, a_{t:t+h} \sim \rho^*}[log(p_\theta(a_{t+H}, ..., a_t | o_t)] \tag{3.28}$$

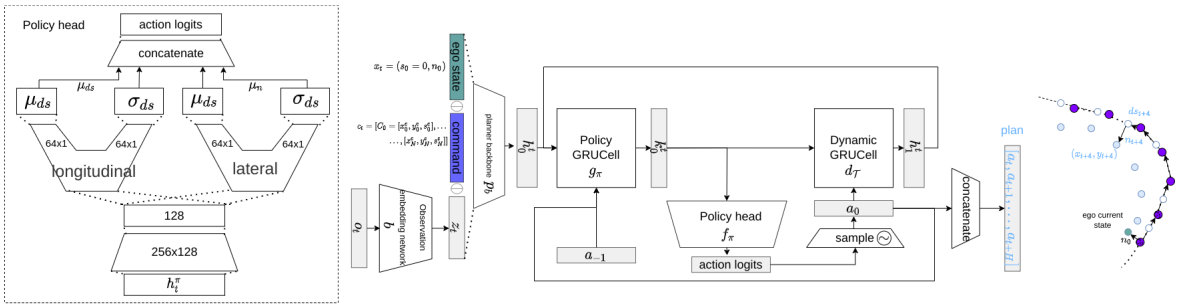

Figure 3.8: Architecture of the auto regressive planner called Planner_AUTOREG

In the following, we first check that the auto-regressive planning procedure does not hurt performances of the planner in open loop with respect to the previous method. The results

	Huge_R		Huge_I	
	ADE-1.5	FDE-1.5	ADE-1.5	FDE-1.5
FCBaseline_basic_Planner_ILL	0.28	0.38	0.27	0.45
FCBaseline_basic_Planner_Autoreg	0.26	0.32	0.25	0.39

Table 3.3: Open loop test performances comparison between several IIL planner with different observation backbones

provided in tab.3.2.2.1 shows that the auto-regressive planner reached slightly better performances in open loop which confirms that the transition model can be trained efficiently even with two consecutive GRU cells. We do not expect the open loop performance to be radically better since our training data are limited but we expect that plans look better in closed loop evaluation since the transition model should at least enforce consistency between consecutive steps of the plan. The test performances in closed loop reported in tab.3.2.2.1 show better

Huge_R					
	ADE-5(m)	ADE-10(m)	ADE-15(m)	Off%	CR%
FCBaseline_basic_Planner_ILL	4.25	8.42	11.64	8.0	46
FCBaseline_basic_Planner_Autoreg	3.95	6.32	9.10	7.0	38
Huge_I					
	ADE-5(m)	ADE-10(m)	ADE-15(m)	Off%	CR%
FCBaseline_basic_Planner_ILL	4.80	7.80	11.62	8.2	42
FCBaseline_basic_Planner_Autoreg	3.96	6.22	10.70	7.2	37

Table 3.4: Closed loop test performances comparison between the ILL planner and the AUTOREG_ILL planner.

performance for the *AUTOREG_Planner_ILL* architecture than the previous method. It appears that the average distance errors given by (ADE-5,ADE-10,ADE-15) grow much faster with the *Planner_ILL* than it does for the *AUTOREG_Planner_ILL* architecture. To illustrate improvements of the auto-regressive method in closed loop simulation, we provide qualitative results in fig.3.9 of plans in situations where the agent is starting to deviate from the expert trajectory. While the *Planner_ILL* tends to generate paths that are not consistent (longitudinal and lateral accelerations are not smooth), the *AUTOREG_Planner_ILL* tries

to compensate smoothly the lateral deviations by generating a trajectory that starts from the current pose. This kind of adaptation enables to compensate distributional shift because it gradually changes the state contrary to abrupt reactions that can be seen with the $Planner\_ILL$ that do not integrates that consecutive states cannot change too quickly.

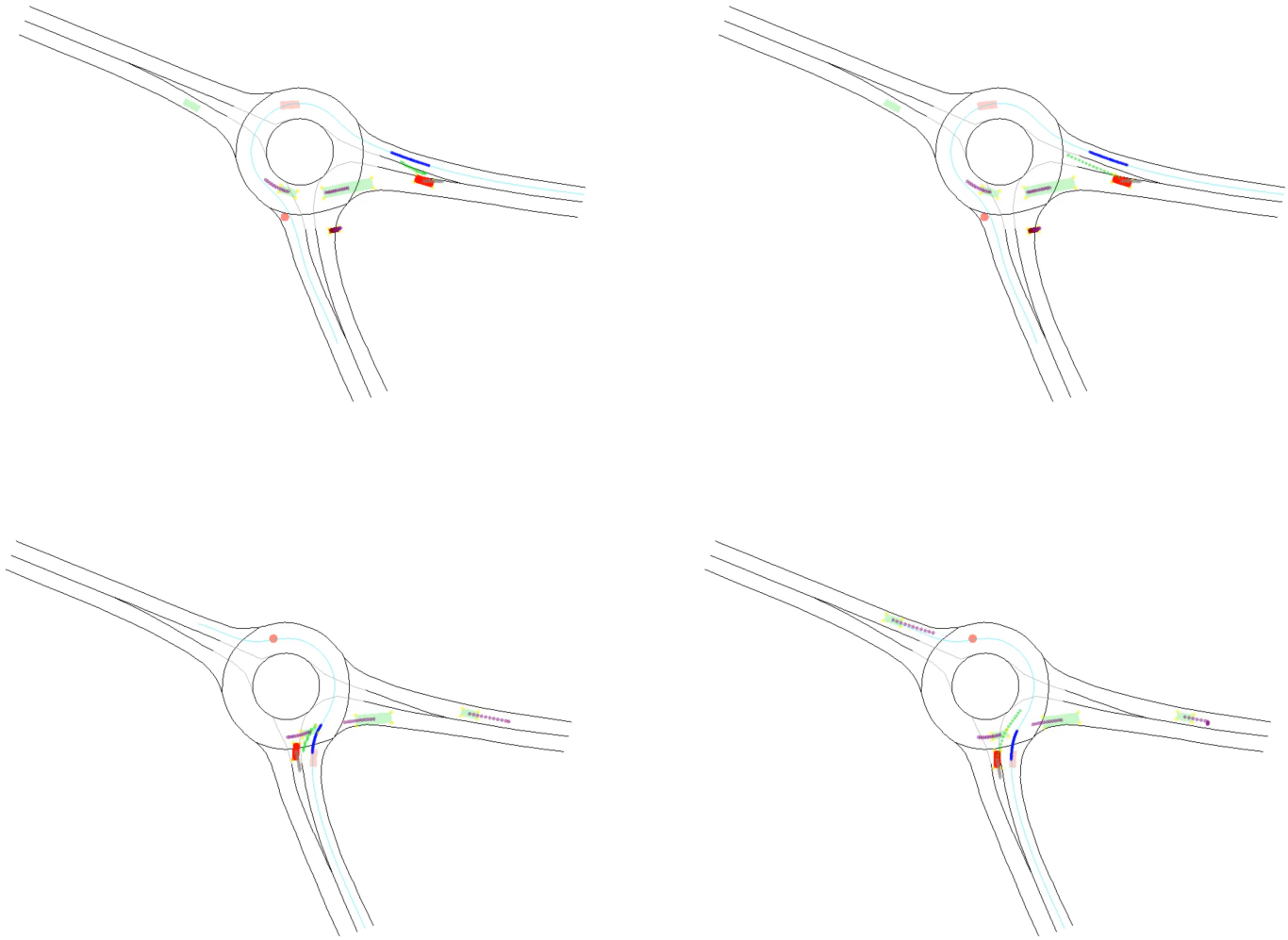

Figure 3.9: Qualitative results of plans generated on test scenarios from the same starting out of expert distribution with an initial lateral offset: on the left side the $Planner\_ILL$ and on the right side $AUTOREG\_Planner\_ILL$.

3.2.2.2 Learning from interactions

In this section, we investigate how to efficiently train the state transition model to improve the closed loop performances of the planner. We know that planning accurately a sequence of actions as an expert does not necessarily enable to understand how a plan influences the future states of the ego agent. Since the driving scenes are generally populated with multiple agents which are also part of the ego agent future states, it is not straightforward to model how ego agent plan $[a_t, ...a_{t+h}]$ is going to influence individual positions of each of its neighbors.

Predicting multi agent interactions for more than one second under a SDV plan in presence of diverse drivers is extremely challenging especially because we ignore the behaviour of other participants as well as their intent [194, 159]. However predicting how actions influence the ego agent configuration is easier because it mainly depends on the ego agent physical model which in our setting reduces to converting curvilinear coordinates into cartesian coordinates. Since our auto-regressive planner uses a latent state representation, it is not directly possible to isolate ego agent future configuration and other agent future configurations. Instead of explicitly predicting the environment evolution we propose to use an auxiliary loss called prediction loss $\mathcal{L}_{\mathcal{T}}$ based on environment transitions in order to improve the latent state representation of the planner. This loss does not directly improve the decision making but aims to improve the transition model used for planning. The loss $\mathcal{L}_{\mathcal{T}}$ can be computed on transitions in the form of $(a_{t-1}, o_t, a_t, o_{t+1})$ collected by an arbitrary behaviour policy in the environment. In case we call our auto-regressive planner at step t on observation o_t it would output the following latent state h^t_{t+1} for the first planning step.

$$z_t = b(o_t)\ h^t_t = b_p(z_t, c_t, x_t)\ k^t_t = g_\pi(h^t_t, a_{t-1})\ h^t_{t+1} = d_{\mathcal{T}}(k^t_t, a_t) \tag{3.29}$$

In case the planner is queried directly at the next step $t+1$ it would output the following latent state to represent the one effectively reached and provided through o_{t+1}.

$$z_{t+1} = b(o_{t+1})\ h^{t+1}_{t+1} = b_p(z_{t+1}, c_{t+1}, x_{t+1}) \tag{3.30}$$

Note that in the transition, actions a_{t-1} and a_t are not necessarily optimal but partially explain how the agent went from o_t to o_{t+1}. The fact that we also consider a_{t-1} to model the transition from o_t to o_{t+1} is reasonable in the sense that previous action normally condition other agents future reactions. Since we want the planner to acquire consistent representations of the influence of an action a_t on next state s_{t+1},we should constrain h^{t+1}_{t+1} to match with h^t_{t+1}.We use the mean square loss on batch of transitions to compute the loss that should be minimized:

$$argmin_\theta \mathcal{L}_{\mathcal{T}}(\theta)\ \mathcal{L}_{\mathcal{T}} = \mathbb{E}_{\mathcal{B}\sim\pi_b}[0.5 * (h^t_{t+1}(\theta) - h^{t+1}_{t+1}(\theta))^2] \tag{3.31}$$

The loss $\mathcal{L}_{\mathcal{T}}$ can be computed on the support of the expert state distribution provided by an expert dataset but the reason why we introduce this loss separately from the planning loss is related to the fact that it does not requires optimal expert actions. Since closed loop evaluation necessarily approximate the environment dynamic either with replayed agents either with rule based agents, we can choose to compute $\mathcal{L}_{\mathcal{T}}$ with transition generated during simulation roll-outs by the current version of the planner. We expect that better predicting state transitions induced by the current planner may help the planner to better understand which action enables to reach a given latent state. The question that remains is related to the way the prediction loss should be back-progapagated : should we freeze one of the two term h^t_{t+1}, h^{t+1}_{t+1} or should we let

the gradient flow twice through h_{t+1}^t and h_{t+1}^{t+1}? Detaching the latest representation h_{t+1}^{t+1} would enforce the transition model $d_\mathcal{T}$ to update h_t^t based on actions a_t and a_{t-1} but the backbone should be detached $h_t^t = b_p(z_t, c_t, x_t)$ in this case.

$$\mathcal{L}_\mathcal{T} = \mathbb{E}_{\mathcal{B}\sim\pi_b}[0.5(h_{t+1}^t.detach() - d_\mathcal{T}(g_\pi(h_t^t.detach(), a_{t-1}), a_t))^2] \tag{3.32}$$

This formulation enforces the planned latent representation h_{t+1}^t to match with the initial latent representation h_{t+1}^{t+1} provided by the backbone but it does not help the backbone to find an optimal representation. In contrast, if we do not detach anything from the computational graph, then the planned representation h_{t+1}^t will still align with the initial latent representation h_{t+1}^{t+1} but latent representations can get arbitrarily big or small since their difference is minimized. We choose to add a L2 regularization terms to prevent the network parameters from changing abruptly in case the backbone is detached.
In the following, we investigate the influence of both version of prediction loss on the planner performances.

In a first experience, we analyse how evolve open loop performances when the prediction

Algorithm 9 Autoreg planner pretraining with the prediction loss

INPUTS:

- expert data: $\mathcal{D}^e$
- $\mathcal{S}$: associate scenario database
- c: cost function for the prediction loss

for i=1,2,...N_{epoch} **do**
  for k=1,2,... **do**
    $B \sim \mathcal{D}^e$ sample expert data
    $\mathcal{L}(\theta) = \mathbb{E}_{o_t, a_{t:t+h}\sim B}[log(p_\theta(a_{t+H}, ..., a_t|o_t)]$ compute planning loss
    if i%$N_{data\,collection} = 0$ **then**
      $\mathcal{D}_i \leftarrow$collectTrajectoriesWithPlanner(π_θ)
    end if
    if $i > N_{data\,collection}$ **then**
      $B \sim \mathcal{D}_i^\pi$ sample policy data
      $\mathcal{L}(\theta) = \mathcal{L}(\theta) + \mathbb{E}_{\mathcal{B}\sim\pi_b}[c(h_{t+1}^t(\theta), h_{t+1}^{t+1}(\theta))]$
    end if
    $\theta_{k+1} = \theta_k + \alpha.\nabla_\theta\mathcal{L}(\theta)$
  end for
end for

loss $\mathcal{L}_\mathcal{T}$ is used in addition to the planning loss during training. The training procedure that explains how simulation data are generated and used to compute the prediction loss is detailed in alg.9.

According to results in tab.3.5, we observe that both version of the prediction loss did not deteriorate final open loop test performances but they did not bring significant improvements

	Huge_R		Huge_I	
	ADE-1.5	FDE-1.5	ADE-1.5	FDE-1.5
FCBaseline_basic_Planner_Autoreg	0.26	0.32	0.25	0.39
FCBaseline_basic_Autoreg_pred_loss_detached	0.25	0.31	0.23	0.36
FCBaseline_basic_Autoreg_pred_loss	0.24	0.28	0.22	0.32

Table 3.5: Open loop test performances of the auto-regressive planner trained with the prediction loss

either. It appears that the prediction loss does not provide more guidance than the planning loss on the expert support. Additionally, we noted that the detached version of the prediction loss is considerably more unstable during training: the prediction loss exhibits peaks at each new trajectory collection which reveals that the latent state representation is still significantly changing. The other version of the prediction loss progressively vanish during training which means that the planner can finally exploit a fixed state representation to compute actions.
In a second experiment, we analyse how the transition loss $\mathcal{L}_{\mathcal{T}}$ influence the closed loop performances of the planner. We also report the average amplitude of the transition loss $\mathcal{L}_{\mathcal{T}}$ during each test episodes. Note that $\mathcal{L}_{\mathcal{T}}$ represents the average discrepancy between the representations of the next state and what the planner expects to reach. The smaller ${\mathcal{L}_{\mathcal{T}}}^h$ the smaller the representation error between the planned latent state h_{t+1}^t and its effective representation h_{t+1}^{t+1} which means a priory that replanning at o_{t+1} won't considerably change the previous plan. The variation in replanning can also be calculated based on the difference between the planned action a_{t+1}^t and the action effectively taken a_{t+1}^{t+1} at the next decision step $t+1$. We chose to compute the re planning error only with the longitudinal components expressed in meters because the longitudinal behaviour is more difficult to learn .

$$e_{replan} = \left| ds_{t+1}^{t+1} - ds_{t+1}^{t} \right| \tag{3.33}$$

Small values of e_{replan} indicates that the planner is adopting a long term strategy because it does not change significantly during consecutive replannings. However small values of e_{replan} does not mean that the driving strategy is optimal in terms of collision avoidance because $\mathcal{L}_{\mathcal{T}}$ is reward free and just indicates that the planner reached the state it intended to reach during the planning which means that it partially control its environment. We observe in tab.3.2.2.2 that both auto-regressive planners trained with the prediction loss outperform the baseline trained only with the planning loss even if the rate of collision is still important. We observe that long term imitation errors tend to reduce which means that the planner better understand how to take action and which state should be reached. This does not mean that the planner can learn the sequence of states that enables to avoid crashes which explains the remaining high rate of collision. We note that the detached prediction loss led to significantly lower performances: more specifically the replanning error e_{replan} is higher and closed to the baseline which means that the planner does not reached what it had planned and consequently replans a different action. In contrast the replanning errors e_{replan} of our best planner trained

Huge_R					
	ADE-5	ADE-15	e_{replan}	Off	CR
FCBaseline_basic_Autoreg	3.95	9.1	0.32	7.0	38
FCBaseline_basic_Autoreg_pred_loss_detached	3.81	8.56	0.16	6.1	36
FCBaseline_basic_Autoreg_pred_loss	3.41	8.20	0.05	4.3	34
Huge_I					
	ADE-5	ADE-15	e_{replan}	Off	CR
FCBaseline_basic_Autoreg	3.96	10.70	0.24	7.2	37
FCBaseline_basic_Autoreg_pred_loss_detached	3.93	8.70	0.14	6.3	34
FCBaseline_basic_Autoreg_pred_loss	3.54	8.32	0.06	3.3	32

Table 3.6: Closed loop test performances comparison when the prediction loss is used to trained the auto-regressive planner.

with the other prediction loss where the backbone is not detached show that during replanning, the action a_t^{t+1} previously planned and the next decision a_{t+1}^{t+1} almost always match which means that decision are taken for more long term.

3.3 Conclusion

In this chapter, we explained how we can efficiently parameterize actions based on curvilinar coordinates with respect to the reference path provided by the routing module. We later explain how to encode the local scene context with different neural network architectures and showed that they enable to learn to plan short term trajectories as human drivers in open loop evaluation. In order to improve the performances of the short term trajectory planner in closed-loop evaluation, we proposed an auto-regressive planner model that better integrates the impact of planned actions on futures states which enable to reduce deviations with respect to the expert trajectory.

Chapter 4

Learning driving policies from domain knowledge

Contents

Figures

Tables

In this chapter, we study how to apply reinforcement learning for basic traffic rules for learning driving policies. In sec.4.2, we explain how to train a driving policy on a large scenario database with actor critic policy gradient algorithms. In a second time, we explain in sec.4.2 how to improve test performances of the driving policy by modifying the training procedure of the actor-critic network.

4.1 Learning to drive with Reinforcement Learning

4.1.1 Principles

We introduce the theory at the basis of all algorithms that we will use in this chapter. We consider an infinite-horizon discounted Markov decision process (MDP) defined by the tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, r, \rho_0, \gamma \rangle$ where $\mathcal{S}$ is the set of states and $\mathcal{A}$ the set of actions, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$ the transition probability function, $r : \mathcal{S} \times \mathcal{A} \to R$ the reward function, ρ the initial distribution over states, and $\gamma \in [0, 1)$ the discount factor. The return of a trajectory $\tau = (s_0, a_o, s_1, ...)$ is denoted $\eta(\tau) = \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)$. In reinforcement learning the objective is to optimizes the expected discounted rewards $J(\pi)$:

$$\pi^* := argmax_{\pi \in \Pi} J(\pi) \tag{4.1}$$

$$J(\pi) = \mathbb{E}_{s_0 \sim \rho, \tau \sim \pi}[\eta(\tau)] \tag{4.2}$$

$$J(\pi) = \mathbb{E}_{s_0 \sim \rho}[V^{\pi}(s_o)] \tag{4.3}$$

$$J(\pi) = \mathbb{E}_{s_0 \sim \rho_0, a_\tau \sim \pi(.|o_t), s_{\tau+1} \sim p(s_{\tau+1}|s_\tau, a_\tau)}[\sum_{\tau=0}^{\infty} \gamma^\tau r(s_\tau, a_\tau)] \tag{4.4}$$

Each policy π induces a stationary discounted state distribution by:

$$d_\gamma^\pi(s) = \int_{\mathcal{S}} \rho(s_o) \sum_{t=0}^{\infty} \gamma^t . \mathbb{P}(s_t = s | s_o, \pi) ds_0 \tag{4.5}$$

as well as value functions Q^π, V^π that quantify mean average return obtained by a policy starting from respectively a given state s or a given state s followed by action a:

$$Q^\pi(s_t, a_t) = \mathbb{E}_{\rho, \pi, \mathcal{T}}[\sum_{l=0}^{\infty} \gamma^l . R(s_{t+l}, a_{t+l}, s_{t+l+1}) | s_t = s, a_t = a] \tag{4.6}$$

$$V^\pi(s_t) = \mathbb{E}_{\rho, \pi, \mathcal{T}}[\sum_{l=0}^{\infty} \gamma^l . R(s_{t+l}, a_{t+l}, s_{t+l+1}) | s_t = s] \tag{4.7}$$

$$\tag{4.8}$$

In order to understand the effects of applying action a from state s on the policy return, we also introduce the advantage function of policy π denoted $A^\pi(s, a)$. Intuitively, it represents

the change in the expected return when we apply action a in state s and then follow policy π compared to directly following policy π from state s. In case the advantage is positive, it means that choosing action a in state s and then following the policy leads to a return that is better than the average return we obtain following the policy directly from state s.

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s_t) \tag{4.9}$$

In the following, we consider a model free setting where the environment dynamic is unknown for the policy in contrast to model based approaches that leverage an environment model to take action [34]. Learning a transition model is possible but relatively difficult because we do not have access to a real interactive simulation environment but only to a restricted set of driving data on specific driving scenes. Therefore, the environment model is expected to quickly overfit the driving data in the dataset without providing consistent guidance for closed loop simulation. Intuitively, during simulation the policy will experience situations largely different from the one provided in the driving dataset because policy actions influence the next states distributions. Instead of learning a full model we chose to directly learn a compact state value function according to model-free reinforcement learning. Two categories of algorithms have to be considered: value-based methods such as Rainbow [66] or policy-based approaches such as TRPO [168]. The main principle of value-based methods is to approximate the fixed point of the Bellman equation through generalized policy iteration (GPI)[182]. Value based methods were mostly used for discrete action settings with successful results but our driving policy operates on a continuous action space for generating trajectories. In contrast Policy-based algorithm were rather used in continuous setting and directly learn a parameterized policy through a policy improvement phase. Policy based methods where later improved based on actor critic architectures that leverage a state value function to reduce high variance of policy gradients. In this work, we investigate how policy gradient actor critic methods can learn driving policies on numerous driving scenarios. Before explaining more in details those algorithms, we first explain how to define a reward function for learning a driving policy.

4.1.2 Reward engineering

In order to learn a driving policy with model free reinforcement learning, we need the specify the associate reward function. We first review common practices in inverse reward design in Sec.4.1.2.1 before detailing how we designed our own driving reward in Sec.4.1.2.2.

4.1.2.1 Inverse reward design

RL basically consists in optimizing the expected discounted return of a policy to learn a target behaviour based on a Markovian reward $r(s_t, a_t, s_{t+1})$ that only depends on the current transition (s_t, a_t, s_{t+1}). In general, the return should be distinguished from the performances because

the reward signal contains not only the true utility signal which reflects the global performance but also some learning guidance called shaping terms. For complex tasks like driving, its is difficult to evaluate a driving behaviour based on a single scalar signal that would define the utility. Usually the true utility is unknown and driving performances are usually evaluated with multiple criterion that act as proxies and which can be integrated in the the reward function. Building a reward function for a desired behaviour is known as the agent alignment problem [58]. There are many different reward specifications under which an optimal policy has the same performance guarantees on the task. This freedom in choosing the reward function, in turn, leads to the fundamental question of reward design: what are the right criteria that one should consider in designing a reward function ?

Specifying a consistent reward value for each transition (s_t, a_t, s_{t+1}) mostly relies on domain knowledge. Since rewards should also stay interpretable, they are usually decomposed in a weighted sum of terms composed of bonus and penalties. For instance, when an agent reach an intersection, it should avoid collisions while crossing the intersection in a short time. Collisions can be penalized by a penalty term r_c while a bonus r_f can be provided when the agent moves forward. Finding appropriate values for r_c and r_f such that agent safely drive on arbitrary intersections is however tricky and time consuming because it requires to run the RL algorithm to check the resulting behaviour. Reward mis-specifications for some transition (s_t, a_t, s_{t+1}) can let the policy trapped in local optima where the exploration mechanism is inefficient.

To compensate exploration deficiencies, the reward often requires some shaping terms so that the agent can be guided along the whole training from states of low utility towards states of higher utility. Sparse reward functions that just indicate achievement of the task does not enable to explore efficiently high dimensional state space. This problem is exacerbated by the fact that the policy or the value estimator can be poorly initialized. Another limitation of exploration, comes when actions are sampled from infinite support distribution such as Gaussian and then clipped during exploration phase [21]. This situation happens for example when we do not allow backward displacements for a driving policy on a straight lane. In this case some action may not have any consequences on the next state which force the reward to depend on the action.

Sometimes the dependencies between states and actions are even more subtle when the outcome of a selected action is delayed. For instance, a collision may be induced by the commitment of a vehicle in an intersection some time before the collision occurred. Discovering which action(s) are really responsible for the delayed outcome is known as the (temporal) Credit Assignment Problem (CAP) [123]. In some situations, the reward cannot stay markovian because it depends not only on the previous action but on part of the action history, hence it should be modeled as $R(.|s_{t+1}, a_t, s_t, ...a_{t-T}, s_{t-T})$. Based on this short insight, we propose a simple yet interpretable reward function for learning a driving policy in order to avoid as much as possible unexpected behaviour after policy optimization.

4.1.2.2 A reward for driving

In our framework, traffic agents are endowed with a hierarchical policy $\pi_{hierarchal}(o_t, g_t) = \pi_{maneuver}(\pi_{routing}(o_t, g_t)) = \pi_{maneuver}(p_t, o_t))$ where p_t represents the traffic free reference path to follow for the next decision steps. The maneuver policy is supposed to select an action a_t that at short term matches with the plan p_t and the social context of the driving scene. In the long term the policy should be able to follow the whole route that is only locally represented by the traffic free reference path p_t at each decision step. The action space already enable to guide the agent along the route to follow because we are using curvilinear coordinates with respect to p_t to specify the action $a_t = [ds_t, n_t]$ where ds_t indicates the desired longitudinal shift from current abscissa s_t and n_{t+1} indicates the next lateral ordinate at abscissa $s_t + ds_t$. The path acts as an abstract sub-goal but the agent also needs to behave in a socially consistent way while respecting traffic rules. Engineering a sophisticated reward is a problem in itself as explained in Sec.4.1.2.1 and it mainly depends on the driving task, lane merging [62], over-passing on highways [130] but our goal in this chapter is to design a basic reward signal to acquire general driving skills for driving. For practical applications, traffic simulations will take place on a bounded map and will have a finite temporal extent denoted H and traffic agents are expected to behave consistently during the whole episode. However, some agents may only be alive for $h < H$ steps because they start closed to the map borders so the episode termination is not triggered by the environment in itself but by simulation restrictions. Therefore, we consider the driving task as a continuing task with a discount factor γ chosen such that the return mainly takes into account the rewards for the first steps. For intrepretability, the reward signal is expressed as a weighed sum of terms that isolate different aspects of the driving :

$$r(s, a, s') = \sum_{i=1}^{n} w_i.r_i(s, a, s') \tag{4.10}$$

socially consistent interactions:

- **collision penalty**: $r_c(s_t, a_t, s_{t+1}) = 1(s_t \in S_{col}\, or\, s_{t+1} \in S_{col}).(1.0 + \frac{ds}{ds_{max}})$ where S_{col} is a subspace of states with ego agent in collision. The penalty corresponds to the event and not to the root cause which highly depends on the context. Note that we can either stop the episode after a collision occurred or let the simulation continue.
- **sophisticated penalties**: More sophisticated signals could be added to avoid abrupt over-passing or to incite the agent to keep safety distances. However those signals highly depend on the context. For instance, keeping a safety distance with a front neighbor moving in the same lane as the ego agent is different from keeping a safety distance at a an intersection area with other intersecting agents. Introducing penalties for safety distances make the global reward less interpretable because it interfere with the collision penalty. In order make the reward as simple as possible we chose to only consider $r_c(s_t, a_t, s_{t+1})$ for socially consistent interactions.

task completion:

- **forward moving bonus**: In order to encourage the agent to move forward, we give a bonus that increases when the longitudinal speed ds gets closer to the optimal speed $ds_{optimal}$ and stagnates above. This bonus is only provided when $ds_{min} \leq ds \leq ds_{max}$ otherwise the agent goes either too fast or too slow.

$$r_{mf}(s_t, a_t, s_{t+1}) = 1(ds_{min} \leq ds \leq ds_{max}).min(\frac{ds}{ds_{optimal}}, 1.0)$$

traffic rule consistency:

- **lateral jerk**:$r_{lj}(s_t, a_t, s_{t+1})$ Since the agent directly controls the lateral position n_{t+1} with respect to the route at the abscissa $s_t + ds_t$ then the lateral position can easily oscillates during exploration if nothing is done. To restrict lateral positions variations we add the following lateral jerk penalty which assigns a negative value configured with the $offset = 0.4$ when $|n_{t+1} - n_t| \leq dn_{max} = 0.3$. $r_{lj}(s_t, a_t, s_{t+1}) = e^{-\frac{|n_{t+1}-n_t|}{dn_{max}}} - offset$

- **lateral distance penalty**: since the agent controls the lateral position with respect to the route, it should not drive too far from the center-line even if some freedom should be given as long as the agent stays in the associated lanelet. We assign the minimum value on the center-line $bonus\_on\_route = 0$ which decreases linearly up to $penalty\,on\,border$ when $|n| = n_{max}$ where n_{max} denotes the maximum distance allowed with respect to the route center-line. In case the agent is farther than $n_{max\,route}$ then the penalty slope gets bigger. Note that the lateral distance to the route cannot be bigger than n_{max} which triggers an episode termination.

$$r_{ldp}(s_t, a_t, s_{t+1}) = 1(n < n_{max\,route}).p_{in}(n) + 1(n \geq n_{max\,route}).p_{out}(n) \tag{4.11}$$

$$p_{in}(n) = \frac{(penalty\,on\,border - bonus\,on\,route)}{n_{max\,route}}.\,|n| + bonus\,on\,route \tag{4.12}$$

$$p_{out}(n) = max(a.\,|n| + b, max\,penalty) \tag{4.13}$$

$$a = \frac{(max\,penalty - penalty\,on\,border)}{(n_{max} - n_{max\,route})} \tag{4.14}$$

$$b = penalty\,on\,border - a * n_{max\,route} \tag{4.15}$$

- **penalty moving too fast**: when the agent moves to quickly, we don't directly penalize it until $ds_{optimal} \leq ds \leq ds_{max}$ but we reduce the bonus and when it exceeds ds_{max} then it gets penalized. This strategy let some freedom to the agent which has to find the optimal speed through exploration.

$$r_{mtf}(s_t, a_t, s_{t+1}) = 1(ds > ds_{max}).\frac{(ds_{max} - ds)}{ds_{optimal}} \tag{4.16}$$

- **penalty moving too slow**: in case the agent moves too slowly on the road, we keep the same formula as r_{mf} but we highlight that the speed is the one provided in the action and not the effective speed between s_t and s_{t+1}. As a consequence, negative speed which does not result to any change of the state can still be penalized smoothly. Since the action space is continuous and since exploration is restricted it is important to avoid sparse reward with no gradient.

$$r_{mts}(s_t, a_t, s_{t+1}) = 1(ds_{min} > ds).\frac{ds}{ds_{optimal}} \tag{4.17}$$

Given all the above terms, we need to choose their respective weights. Before choosing them, we first normalize the reward terms so that their range of values lie in $[-1.0, 1, 0]$ which makes reward weights w_i more interpretable. One important consideration for training is the relative importance of terms with each other. Ideally during training, we first want the agent to start moving forward while it avoids off-road driving. Subsequently the agent should learn to avoid most of collisions and lastly we want it to drive at optimal speed without jerk in a socially consistent way. To achieve this result, we can play with the reward weights by assigning high weights to reward terms of sub-tasks with high priority and low weights to secondary sub tasks. Note that its is also possible to change the weights during training to progressively acquire new skills but we did not explore this solution to simplify the framework. Evaluating and comparing rewards with each other is a problem in itself [202] and we resort to an experimental procedure to check which reward hypothesis is most appropriate to learn a consistent driving behaviour. We do not evaluate the driving behaviour based on the magnitude of the return but based on some driving performances such as the rate of episodes with at least one collision (CR%), the average time spent offroad during an epsiode (Off%), and the averaged distance traveled with respect to an expert(L%). We also evaluate qualitatively driving episodes leveraging the rendering engine of our simulator. Given our reward model we call a reward hypothesis the associate list of weights $[w_{mts}, w_{mtf}, w_{ldp}, w_{lj}, w_{mf}, w_c]$ and we tested few reasonable hypothesis to understand which is the best. We will expose the analysis that consider two of the most important weights in the formula : w_c which weights collisions and w_{mf} which encourage the agent to move forward. Inappropriate values could either lead to a driving behaviour that almost never moves if a collision is likely or to a driving behaviour that moves even if collision occur.

In the following experience, we trained a driving policy on a driving database called $Huge\_R\_basic$ generated on a roundabout from the Interaction Dataset [214] and whose composition is detailed in annexes .2. We restrict this experience on one type of scene to highlight that optimal reward weightings mainly depends on the environment dynamic and choosing appropriate weights for multiple type of scenarios requires massive amount of hyper-parameters search. During our experience, we fine-tuned PPO hyperparameters on the training database and we only provides the best driving training performances obtained for each reward hypothesis. We observe that the best trade off is obtained by reward hypothesis r_2 which enables to move at least as far

hypothesis	w_{mts}	w_{mtf}	w_{ldp}	w_{lj}	w_{mf}	w_c	CR\%	Off\%	L\%
r_1	0.1	0.1	0.5	0.02	0.2	1.0	6.3	2.5	95
r_2	0.1	0.1	0.5	0.02	0.2	2.0	4.5	2.6	120
r_3	0.1	0.1	0.5	0.02	1.0	1.0	7.3	3.3	130
r_4	0.1	0.1	0.5	0.02	1.0	2.0	5.1	4.3	123

Table 4.1: Training performances of PPO trained from scratch for different reward hypothesis

as the expert while avoiding most of the collisions. Other reward hypothesis tend to neglect rare collision incident which can be observed depicted with blue points on fig.4.1 where safety distances with front neighbors are often underestimated which does not let enough time to the ego agent to adapt. For the next sections, we will use the following weights to define our

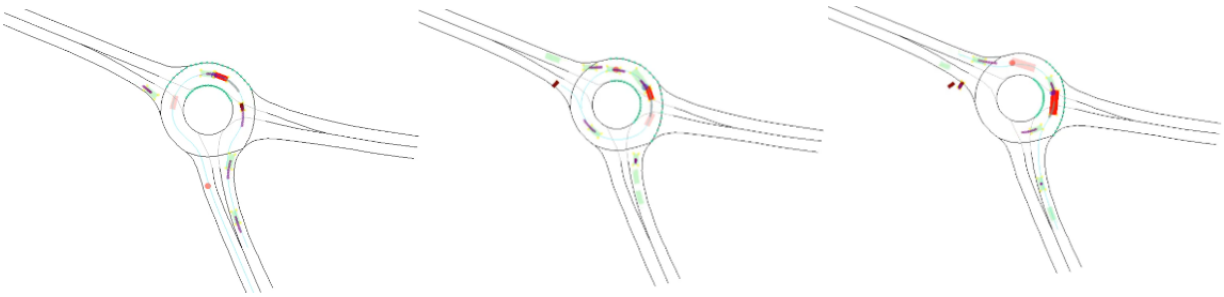

Figure 4.1: Driving behaviours comparison for different reward hypothesis: the agent is represented in dark red while the virtual expert is represented in shaded red. The collisions with replayed agents in green are represented with blue points.

synthetic reward.

$$w_{mts} = 0.1, w_{mtf} = 0.1, w_{ldp} = 0.5, w_{lj} = 0.02, w_{mf} = 0.2, w_c = 2.0$$

4.1.3 Learning with policy gradients

In this section, we introduce the core principles of actor critic policy gradient algorithms. We detail all the algorithms that we use to train train our driving policy and we also stress their strength and deficiencies. Recent works, highlighted that the efficiency of policy gradients highly depends on implementation details[188, 95, 39], so we considered essential to provide them with clarity.

4.1.3.1 Vanilla Policy gradient

Stochastic policy gradient[183] aims to optimize the expected return of a stochastic policy π_θ with the objective $J(\theta) = \mathbb{E}_{s_0 \sim \rho, \tau \sim \pi_\theta}[R(\tau)]$. We consider a smooth class of parameterized stochastic polices $\Pi = \{\pi(.|s;\theta) : s \in \mathcal{S}, \theta \in \Theta\}$. Policy search is achieved by stochastic gradient

ascent on the policy parameters θ:

$$\theta^* = argmax_\theta J(\theta) = \sum_\tau \mathbb{P}(\tau|\theta).R(\tau) \tag{4.18}$$

The general idea consists in increasing the probability $\mathbb{P}(\tau|\theta)$ of trajectories τ of high return collected under policy π_θ. The gradient of the objective $\nabla J(\theta)$ requires the knowledge of $\mathbb{P}(\tau|\theta)$ which expresses as follows in case the state s_{t+1} is independent from $\tau_{0:t} = [s_0, a_0, ..., s_t]$(Strong Markov assumptions):

$$\mathbb{P}(\tau|\theta) = \prod_{t=1}^{H} p(s_{t+1}|s_t, a_t).\pi_\theta(a_t|s_t)\rho(s_0) \tag{4.19}$$

In model free setting we do not have access to the analytical expression of the dynamic $p(s_{t+1}|s_t, a_t)$ so $\nabla_\theta J(\theta)$ should not depend on it. It happens that

$$\nabla_\theta J(\theta) = \nabla_\theta \sum_\tau \mathbb{P}(\tau|\theta).R(\tau) = \sum_\tau \mathbb{P}(\tau|\theta).\frac{\nabla_\theta \mathbb{P}(\tau|\theta)}{\mathbb{P}(\tau|\theta)}R(\tau) \tag{4.20}$$

$$\sum_\tau \mathbb{P}(\tau|\theta).\nabla_\theta ln(\mathbb{P}(\tau|\theta).R(\tau) = \mathbb{E}_\tau[\nabla_\theta ln(\mathbb{P}(\tau|\theta).R(\tau)] \tag{4.21}$$

As $p(s_{t+1}|s_t, a_t)$ and $\rho(s_0)$ do not depend on policy $\nabla_\theta ln(\mathbb{P}(\tau|\theta) = \sum_{t=1}^{H} \nabla_\theta ln(\pi_\theta(a_t|s_t))$ hence we obtain :

$$\nabla J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[\sum_{t=1}^{H} \nabla_\theta ln(\pi_\theta(a_t|s_t).R(\tau)] \tag{4.22}$$

Note that $R(\tau) = \sum_{t=1}^{H} r(s_t^{(i)}, a_t^{(i)})$ contains all trajectory rewards but at time t, past rewards are independent from the current action a_t so we can rewrite :

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[\sum_{t=1}^{H} \nabla_\theta ln(\pi_\theta(a_t|s_t).R(\tau_{t:H})] \tag{4.23}$$

where $R(\tau_{t:H}^{(i)}) = \sum_{k=t}^{H} r(s_k^{(i)}, a_k^{(i)})$. In practice, the expectation can be approximated with m trajectory samples:

$$\frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{H} \nabla_\theta ln(\pi_\theta(a_t^{(i)}|s_t^{(i)}))R(\tau_{t:H}^{(i)}) \tag{4.24}$$

Note that if all rewards of the return are positive then $R(\tau_{t:H}^{(i)})$ will increase the probability of each action within $\tau_{t:H}^{(i)}$ which does not provide real guidance to choose actions that are better than the ones chosen during trajectory collection. One trick is to normalize all the rewards collected during trajectory collection such that only the largest rewards contribute to increase the likelihood. However, maximizing the likelihood of trajectories with high reward does not guarantee to maximize the average return. Additionally, for a pair of state action (s_t, a_t) and a policy π_θ, the returns of trajectories collected by a policy π_{θ_k} starting from (s_t, a_t) can be very different which means that the policy gradient $\frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{H} \nabla_\theta ln(\pi_\theta(a_t^{(i)}|s_t^{(i)}))R(\tau_{t:H}^{(i)})$ will

have a high variance. One possibility is to subtract a state dependant baseline which does not introduce too much bias in the policy gradient while reducing variance. However the objective should still push $\pi_\theta(a_t^{(i)}|s_t^{(i)})$ toward higher returns regions. In case the state baseline does not depend on a_t as for instance the average return over all trajectories it does not introduce bias.

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[\sum_{t=1}^{H} \nabla_\theta ln(\pi_\theta(a_t|s_t).(R(\tau_{t:H}) - b(s_t))] \tag{4.25}$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[\sum_{t=1}^{H} \nabla_\theta ln(\pi_\theta(a_t|s_t).(b(s_t))] = 0 \tag{4.26}$$

$$b(s_t) = \mathbb{E}_{\tau\sim p_\tau(\pi_\theta)}[R(\tau)] = \frac{1}{|\mathcal{D}|}\sum_{\tau^i\in\Gamma} R(\tau_{t:H}^{(i)}) \tag{4.27}$$

Note that this baseline is even independent from the state s_t which prevent in certain situation from detecting if the return collected from s_t denoted $R(\tau_{t:H})$ is getting higher than before. According to the definition of the state action value function

$$\mathbb{E}_{s_0\sim\rho,\tau\sim\pi_\theta}[R(\tau)] = \mathbb{E}_{s_0\sim\rho}[p(s_0).\sum_{a\in\mathcal{A}} \pi_\theta(a_0|s_0)Q^{\pi_\theta}(s_0,a_0)] \tag{4.28}$$

it is possible to replace the sample return $R_{\tau_{t:H}}$ with $Q^{\pi_\theta}(s_t,a_t)$ which gives:

$$\frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{H} \nabla_\theta ln(\pi_\theta(a_t^{(i)}|s_t^{(i)}))(Q^{\pi_\theta}(s_t^{(i)},a_t^{(i)}) - V^{\pi_{old}}(s_t^{(i)})) \tag{4.29}$$

The new term $Q^{\pi_\theta}(s_t^{(i)},a_t^{(i)}) - V^{\pi_{old}}(s_t^{(i)}) = A^{\pi_{old}}(a_t,s_t)$ perfectly fits our desiderata because it represents the advantage of doing $a_t^{(i)}$ at $s_t^{(i)}$ instead of following the mean of the distribution $\pi_{old}(.|s_t)$ and then following π_{old}. Indeed $R(\tau_{t:H}^i) - V^{\pi_{old}}(s_t^{(i)}) \neq Q^{\pi_\theta}(s_t^{(i)},a_t^{(i)}) - V^{\pi_{old}}(s_t^{(i)})$ because sampling $\tau_{t+1:H}^i$ once is not representative of the average performance of the policy for time-steps $t+1:H$. When policy gradient resorts to one of $Q^{\pi_\theta}(s_t^{(i)},a_t^{(i)}), V^{\pi_{old}}(s_t^{(i)})$or both it becomes an actor critic method. However accurately estimating $A^{\pi_\theta}(a_t,s_t)$ while training π_θ remains a challenge especially for long simulations.

$$\frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{H} \nabla_\theta ln(\pi_\theta(a_t^{(i)}|s_t^{(i)}))(A^{\pi_{old}}(a_t,s_t)) \tag{4.30}$$

Vanilla policy gradient methods are prone to instabilities in continuous state and action spaces which has led to development of Trust-Region Learning(TRL) [168] that aims to enforce stable monotonic improvements at each training iteration. A bigger class of algorithm inspired from TRL was recently developed [95] and they reached state-of-the-art performances on a variety of tasks [38, 12]. In the following, we introduce the methods we used to train our driving policies with our synthetic reward model. We extensively compare their performances on our scenario databases in sec.4.1.3.7.

4.1.3.2 Natural policy gradient

We start to explain the main principles that rules all algorithms that we implemented. In vanilla policy gradient, we try to maximize the policy objective (θ) over variables θ while sampling actions from a stochastic policy π_θ. Given the direction provided by the policy gradient $\nabla_\theta J(\theta)$, we should locally modify π_θ by a small amount ϵ to observe a gain in the return. Unfortunately gradient descent in parameter space does not take into account the distance between policy distributions π_θ and $\pi_{\theta_{new}}$. One way to constrain two stochastic policies to stay close is to constrain their KL divergence. Natural policy gradient [82] proposed an update rule which adds a constraint to the original objective

$$\begin{cases} argmax_d J(\theta_{old} + d) \\ \mathbb{D}_{KL}(\pi_{\theta_{old}} || \pi_{\theta_{old}+d}) < \delta \end{cases} \tag{4.31}$$

The Lagrangian of the constrained problem expresses as:

$$L(\theta, \lambda) = J(\theta + d) - \lambda(\mathbb{D}_{KL}(\pi_\theta || \pi_{\theta+d}) - \delta) \tag{4.32}$$

In order to solve numerically this optimization problem we can use the first order Taylor expansion for the loss and second order for the KL in the neighborhood of θ_{old} and we obtain

$$argmax_d J(\theta_{old}) + d.\nabla_\theta J(\theta)|_{\theta=\theta_{old}} - \frac{\lambda}{2}.d^T \nabla^2_\theta(\mathbb{D}_{KL}(\pi_{\theta_{old}} || \pi_\theta)|_{\theta=\theta_{old}}.d + \lambda.\delta \tag{4.33}$$

Note that the Hessian computation $\nabla^2_\theta(\mathbb{D}_{KL}(\pi_{\theta_{old}} || \pi_\theta)|_{\theta=\theta_{old}}$ can be quite expensive for deep neural networks but fortunately the Hessian of a KL divergence matches the definition of a Fisher information matrix which is easier to compute:

$$\nabla^2_\theta(\mathbb{D}_{KL}(\pi_{\theta_{old}} || \pi_\theta)|_{\theta=\theta_{old}} = F(\theta_{old}) = \mathbb{E}_{s\sim\pi_{old}}[\mathbb{E}_{a\sim\pi_{old(.|a)}}[\nabla_\theta ln(\pi_{\theta_{old}}(.|s)).\nabla_\theta ln(\pi_{\theta_{old}}(.|s)^T)]] \tag{4.34}$$

The Fisher Information Matrix defines the local curvature in distribution space for which KL-divergence is the metric. It indicates how much we can change the distribution if you move the parameters θ a little bit in a given direction. In case we substitute the Fisher information matrix, the Lagrangian becomes:

$$L(\theta, \lambda) = J(\theta_{old}) + d.\nabla_\theta J(\theta)|_{\theta=\theta_{old}} - \frac{\lambda}{2}.d^T.F(\theta_{old}).d + \lambda.\delta \tag{4.35}$$

According to Kuhn-Karush-Tucker conditions, the optimum is obtained when $\nabla_\lambda L(d, \lambda) = 0$ and $\nabla_d L(d, \lambda) = 0$

$$\nabla_d(J(\theta_{old}) + \nabla_\theta J(\theta)|_{\theta=\theta_{old}}.d - \frac{\lambda}{2}.d^T F(\theta_{old}) - \delta).d + \lambda.\delta) = 0 \tag{4.36}$$

$$d = \frac{2}{\lambda}.F^{-1}(\theta_{old}).\nabla_\theta J(\theta)|_{\theta=\theta_d} \tag{4.37}$$

The natural gradient is $g_N = F^{-1}(\theta_{old}).\nabla_\theta J(\theta)|_{\theta=\theta_d}$ and the update step should be $\theta_{new} = \theta_{old} + \alpha.g_N$. The step size is chosen based on the fact that $\mathbb{D}_{KL}(\pi_{\theta_{old}}||\pi_\theta)|_{\theta=\theta_{old}}$ should not exceed δ. Due to optimality conditions:

$$\nabla_\lambda(J(\theta_{old}) + \nabla_\theta J(\theta)|_{\theta=\theta_{old}} - \frac{\lambda}{2}.d^T F(\theta_{old}).d + \lambda.\delta) = 0 \tag{4.38}$$

$$\tag{4.39}$$

Consequently, we can express the update step size

$$d^T F(\theta_{old}).d = \delta.2 \tag{4.40}$$

$$\alpha.g_N^T F(\theta_{old}).\alpha.g_N = \delta.2 \tag{4.41}$$

$$\alpha = \sqrt{\frac{\delta.2}{g_N^T F(\theta_{old}).g_N}} \tag{4.42}$$

Note that Natural policy gradient belongs to a broader class of algorithms unified in [148] which uses a first-order approximation of the loss $L(\theta)$ and constrains the step with a quadratic norm. Therefore, each modification $\delta\theta$ of the vector of parameters θ is computed via the solution: $\delta\theta = -\alpha.M(\theta)^{-1}.\nabla_\theta L(\theta)$of the following minimization problem:

$$\begin{cases} min_{\delta\theta}\nabla_\theta L(\theta)^T\delta\theta \\ \delta\theta^T M(\theta).\delta\theta \leq \epsilon^2 \end{cases} \tag{4.43}$$

where $\nabla_\theta L(\theta)$ is the gradient of the loss $L(\theta)$, and $M(\theta)$ a symmetric positive-definite matrix. The methods differ by the matrix $M(\theta)$, which has an effect not only on the size of the steps, but also on the direction of the steps.

Algorithm 10 Natural Policy Gradient

1: Repeat:
2: collect trajectories $\tau_i \sim \pi_{\theta_k}$ and store in Γ_k
3: Estimate adavanatge $\widehat{A}_t^{\pi_{\theta_k}}(a_t^{(i)}, s_t^{(i)})$
4: Form sample estimate for
- natural gradients g_N
- Fisher information matrix

5: compute natural policy update:
6: $\theta_{new} = \theta + \sqrt{\frac{\delta.2}{g_N^T F(\theta_{old}).g_N}}.F^{-1}(\theta_{old}).\hat{\nabla}_\theta J(\theta)|_{\theta=\theta_d}$

Note that the implementation of the Natural Policy Gradient [82] requires to invert the fisher information matrix which can be very expensive for deep neural network with thousands of parameters.

4.1.3.3 Trust region policy optimization

Trust Region Policy Optimization(TRPO)[168] extends the idea of natural gradients with a specific policy objective $J(\theta)$. It has been shown that it is possible to relate the expected return $J(\pi_\theta) = \mathbb{E}_{s_0\sim\rho,\tau\sim\pi_\theta}[\eta(\tau)]$ of policy π_θ and the expected return of another policy π_{old} using advantages [81]:

$$J(\pi_\theta) = J(\pi_{old}) + \mathbb{E}_{s\sim d_\gamma^{\pi_\theta}}[\mathbb{E}_{a\sim\pi_\theta(.|s)}A^{\pi_{old}}(s,a)]] \tag{4.44}$$

$$J(\pi_\theta) = J(\pi_{old}) + \mathbb{E}_{s\sim d_\gamma^{\pi_\theta},a\sim\pi_\theta(.|s)}[A^{\pi_{old}}(s,a)] \tag{4.45}$$

While this relation seems interesting it stays difficult to optimize in practice because the expectation depends on $s \sim d_\gamma^{\pi_\theta}$ and $a \sim \pi_\theta(.|s)$ whereas state action pairs (s,a) used to compute expectations are collected with $d_\gamma^{\pi_{old}}$ and $\pi_{\pi_{\theta_{old}}}$. TRPO does an approximation and uses importance sampling to compensate :

$$J(\pi_\theta) = J(\pi_{old}) + \mathbb{E}_{s\sim d_\gamma^{\pi_{old}},a\sim\pi_{\pi_{\theta_{old}}}(.|s)}[\frac{\pi_\theta(.|s)}{\pi_{\theta_{old}}(.|s)}A_{\pi_{old}}(s,a)] \tag{4.46}$$

Similarly to the Natural policy gradient, TRPO formulates a constrained problem

$$\begin{cases} argmax_\theta J(\pi_\theta)) \\ \mathbb{E}_{s\sim S}[\mathbb{D}_{KL}(\pi_{\theta_{old}}(.|s)||\pi_\theta(.|s))] < \delta \end{cases} \tag{4.47}$$

whose Lagrangian is similar except that the objective is now based on the advantage:

$$L(\theta,\lambda) = J(\theta_{old}) + \Delta\theta.\nabla_\theta J(\theta)|_{\theta=\theta_{old}} - \frac{\lambda}{2}.\Delta\theta^T.F(\theta_{old}).\Delta\theta + \lambda.\delta \tag{4.48}$$

As explained in the previous sec.4.1.3.2, the maximum of L which is quadratic in $\Delta\theta$ is

$$\Delta\theta = \frac{1}{\lambda}.F(\theta_{old})^{-1}.\nabla_\theta J_{\theta_{old}}(\theta) \tag{4.49}$$

Instead of computing the inverse of the Fisher information matrix, which is quadratic with the number of parameters θ, TRPO proposed to use conjugate gradients to iteratively approximate it. After the conjugate gradient optimization step, the constraint $\overline{\mathbb{D}}_{KL}(\pi_{\theta_{old}}||\pi_\theta) = \mathbb{E}_{s\sim S}[\mathbb{D}_{KL}(\pi_{\theta_{old}}(.|s)||\pi_\theta(.|s))] < \delta$ is however not ensured anymore. So TRPO explicitly enforces the trust-region constraint in $\overline{\mathbb{D}}_{KL}(\pi_{\theta_{old}}||\pi_\theta) < \delta$ by a line search: it computes the KL-term for $\theta = \theta_{k+1}$ and checks if it is larger than the threshold δ, in which case, the step size is reduced until the constraint is satisfied. Even if we maintain the constrain, the monotonic improvement of the policy is not fully guaranteed in practice because of advantage estimation errors[95]. Additionally running conjugate gradient descent on deep neural network still requires high number of iterations to reach a good estimate of $x_k = H_k^{-1}.g_k$. In theory it requires at most as many steps as g_k dimensions.

Algorithm 11 TRPO

INPUTS: initial policy paramaters θ_0
for $k = 0, 1, 2, ...$ **do**do:
    collect set of tarjectories Γ_k with π_{θ_k}
    Estimate advantage $\widehat{A}_t^{\pi_{\theta_k}}$
    for $i = 0, 1, ..., N_{TRPO}$ **do**:
        for $\mathcal{B}$ in Γ_k
        compute policy gradient g_n
        KL-divergence Hessian-vector product $H_k.g_k$
        Use CG with n_{cg} iterations to obtain $x_k = H_k^{-1}.g_k$
        estimate the update step $\Delta_k = \sqrt{\frac{\delta.2}{g_k^T H_k.g_k}}.x_k$
        adapt the step size with backtracking line search with exponential decay:
            $\theta_{k+1} = \theta_k + \alpha_j.\Delta k$
    end for
end for

Algorithm 12 Line Search for TRPO

compute proposed policy step $= \sqrt{\frac{\delta.2}{g_N^T F(\theta_{old}).g_N}}.F^{-1}(\theta_{old}).\hat{\nabla}_\theta J(\theta)|_{\theta=\theta_d}$
exponential decay : $[\alpha_0, \alpha_0^1, ..., \alpha_j = \alpha_0^j, ..., \alpha_0^L]$
for $j = 0, 1, ..., L$: **do**
    compute the update $\theta = \theta_k + \alpha_j.\Delta k$
    if $J_{\theta_k}(\theta) \geq 0$ and $\overline{\mathbb{D}}_{KL}(\pi_{\theta_{old}}||\pi_\theta) < \delta$ **then**
        accept the update and set $\theta_{k+1} = \theta_k + \alpha^j.\Delta k$
        break
    end if
end for

4.1.3.4 Proximal policy optimization

Proximal Policy Optimization(PPO)[167] was proposed to overcome the problems of TRPO. They investigate how to obtain a lower bound of TRPO's objective:

$$J(\theta) = \mathbb{E}_{s\sim\rho_{\pi_{\theta_{old}}}, a\sim\pi_\theta(.|s)}[\frac{\pi_\theta(.|s)}{\pi_{\theta_{old}}(.|s)} A_{\pi_{old}}(s,a)] \tag{4.50}$$

Without a constraint, the maximization of $J(\theta)$ would lead to excessively large policy updates. The authors searched how to modify the objective, in order to penalize changes to the policy that make $r_\theta(s) = \frac{\pi_\theta(.|s)}{\pi_{\theta_{old}}(.|s)}$ very different from 1.0, i.e. where the KL divergence between the new and old policies would become high. They first proposed an algorithm called KL-PPO with an adaptive KL penalty :

$$J(\theta) = \mathbb{E}_{s\sim\rho_{\pi_{\theta_{old}}}, a\sim\pi_\theta(.|s)}[\frac{\pi_\theta(.|s)}{\pi_{\theta_{old}}(.|s)} A_{\pi_{old}}(s,a)] + \beta_t \mathbb{E}_{s\sim S}[\mathbb{D}_{KL}(\pi_{\theta_{old}}(.|s)||\pi_\theta(.|s))] \tag{4.51}$$

where the penalty is updated at each PPO mini batch iteration based on the gap with the target $d = \mathbb{E}_{s\sim S}[\mathbb{D}_{KL}(\pi_{\theta_{old}}(.|s)||\pi_\theta(.|s))] < \delta$. The following heuristic was used :

if $d < \delta$ **then**
  $\beta_t \leftarrow \beta_t/2$
else if $d > \delta$ **then**
  $\beta_t \leftarrow \beta_t.2$
end if

The update rule rarely leads to stable improvements in practice. Instead of constraining $\mathbb{D}_{KL}(\pi_{\theta_{old}}(.|s)||\pi_\theta(.|s))] < \delta$ which constrains the policy distribution $\pi_{\theta_{old}}(.|s)$, one can directly constrain $r_\theta(s) = \frac{\pi_{\theta(.|s)}}{\pi_{\theta_{old}}(.|s)}$. Therefore, they proposed the following surrogate loss:

$$J^{clip}(\theta) = \mathbb{E}_{s\sim\rho_{\pi_{\theta_{old}}}, a\sim\pi_\theta(.|s)}[min(r_\theta(s).A_{\pi_{old}}(s,a), clamp(r_\theta(s), 1-\epsilon, 1+\epsilon).A_{\pi_{old}}(s,a)) \quad (4.52)$$

The effect of clipping the importance sampling weight can be illustrated in two interesting situations. In case the advantage is positive then the action a should be made more likely and $r_\theta(s)$ should increase up to $1+\epsilon$. In case the advantage is negative, the action a should be made less likely and should be decreased without getting lower than $1-\epsilon$ which would imply a large policy update.
PPO takes the minimum of the clipped and unclipped objective, so that the final objective gets a lower bound of the unclipped objective. While the left part of the min operator is the surrogate objective of TRPO, the right part restricts the importance sampling weight to stay in $]1-\epsilon, 1+\epsilon[$. Note that clamping remove the gradients which may prevent a lot of samples to contribute to the update. In practice, the sample based objective is estimated on set of trajectories Γ_k.

$$J^{clip}(\theta) = \mathbb{E}_{\tau\in\Gamma_k}[\sum_{t=0}^{T}[min(r_\theta(s).A_{\pi_{old}}(s,a), clamp(r_\theta(s), 1-\epsilon, 1+\epsilon).A^{\theta_k}(s,a)]) \quad (4.53)$$

Instead of sampling full trajectories $\tau \in \Gamma_k$ to compute $J^{clip}(\theta)$, PPO selects mini-batches of samples in Γ_k so that data looks more $i.i.d$ during gradient descent. PPO is prone to breaking

Algorithm 13 PPO

for $k = 0, 1, 2, ...$do **do**;
  collect set of trajectories Γ_k with π_{θ_k}
  Estimate advantage $\widehat{A}_t^{\pi_{\theta_k}}$
  for $i = 0, 1, ..., N_{PPO}$ **do**
    for $\mathcal{B}$ in $shuffle(\Gamma_k)$ **do**
      compute policy gradients g_n with $J^{clip}(\theta)$
      update θ_k with Adam
    end for
  end for
end for

the core principles of trust regions that aims to constrain the update size but a recent unifying

work [95] showed that PPO belongs to a much broader class of algorithms called the Mirror Learning space with convergence guarantees under mild conditions.

4.1.3.5 Mirror descent policy optimization

Previous algorithms PPO, KL-PPO, TRPO try to constrain the consecutive policies to remain close to each other by enforcing a trust region. Those algorithms mainly differ in the way they enforce this trust-region constraint. We could ask to which extent enforcing the hard constrain of TRPO really matters for achieving strong performances. While KL-PPO use a regularization with an adaptive coefficient, TRPO enforces it explicitly through a line-search procedure that ensures that the new policy is selected such that its KL-divergence with the old policy is below a certain threshold. PPO simplifies the problem and use a clipping strategy but it does not prevent the policy ratios to go out of bounds, and only reduces its probability [199, 39]. Mirror Descent Policy Optimization(MDPO) [188] aims to unify those methods leveraging the Mirror Descent algorithm (MD)[131, 9] which is a first order trust-region optimization method for solving constrained convex problems

$$x^* = argmin_{x \in C} f(x) \tag{4.54}$$

where f is a convex function and the constraint set C is convex and compact. In each iteration, MD minimizes a sum of two terms: a linear approximation of the objective function f at the previous estimate x_k, and a proximity term that measures the distance between the updated x_{k+1} and current x_k estimate

$$x_{k+1} \in argmin_{c \in C} \langle \nabla f(x_k), x - x_k \rangle + \frac{1}{t_k} . B_\psi(x, x_k) \tag{4.55}$$

where $B_\psi(x, x_k) = \psi(x) - \psi(x_k) - \langle \nabla \psi(x_k), x - x_k \rangle$ is the Bregman divergence associated with a strongly convex potential function ψ, and t_k is a step-size determined by the MD analysis. Unlike the MD optimization problem, the objective function in policy optimization $\pi^* = argmax_\pi \mathbb{E}_{s_0 \sim \rho}[V^\pi(s_0)]$ is not convex in θ but it was shown that MD-style RL algorithm can still be derived [49].

$$\pi_{k+1} \leftarrow argmax_{\pi \in \Pi} \mathbb{E}_{s \sim \rho^\pi}[\mathbb{E}_{a \sim \pi}[A^{\pi_k}(s, a)] - \frac{1}{t_k} KL(s, \pi, \pi_k)] \tag{4.56}$$

. Since the trust-region optimization problems in the update rule cannot be solved in closed form, its is approximated with multiple steps of stochastic gradient descent (SGD) on the objective functions. The on-policy MDPO update rule for a parameteric policy π_θ consist in

solving :

$$\theta_{k+1} \leftarrow argmax_{\theta \in \Theta} \Psi(\theta, \theta_k) \tag{4.57}$$

$$\Psi(\theta, \theta_k) = \mathbb{E}_{s \sim \rho^{\pi_{\theta_k}}}[\mathbb{E}_{a \sim \pi_{\theta_k}}[\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s,a)] - \frac{1}{t_k} KL(s, \pi_\theta, \pi_{\theta_k})] \tag{4.58}$$

Note that the clipping ratio of PPO does not appear and the KL regularization is weighted by a law motivated by MD theory which sets $t_k = 1 - \frac{k}{K}$ where K is the maximum number of iterations. MDPO uses an annealed schedule to update t_k, starting from 1 and slowly bringing it down to near 0. Contrary to TRPO which enforces a constrain with the forward $KL(\pi_k, \pi)$, MDPO is consistent with the MD update rule in convex optimization and uses the reverse $KL(\pi, \pi_k)$. Since reverse $KL(\pi_\theta, \pi_k)$ is mode-seeking it stops the distribution from collapsing to a very narrow mode of π_k and should consequently less hurt exploration than the forward KL[188]. Since Gaussian distribution are used, KL divergence can be computed in closed form and using the reverse KL does not induce sampling issues. One major difference between PPO and MDPO is the way they apply gradient descent on the objective. PPO use mini batch gradient descent on the training batch Γ_k to perform a policy update while MDPO use the whole training batch. Performing a single gradient step $\nabla_\theta \Psi(\theta, \theta_k)|_{\theta=\theta_k}$ on the whole training batch reduces to vanilla policy gradient because $KL(s, \pi_{\theta_k}, \pi_{\theta_k}) = 0$ which shows that multiple iterations $N_{MDPO} > 1$ on the training bach Γ_k are necessary to approximately enforce the trust-region constraint in MDPO. In practice, the training batch can contain numerous trajectories (more than 100000 transitions) which prevents from computing gradients in one shot. Instead of computing$\nabla_\theta \Psi(\theta, \theta_k)|_{\theta=\theta_k}$ on the whole training batch Γ_k, we use $N_\mathcal{B}$ mini-batches of size $|\mathcal{B}|$ to compute gradients and then average all $N_\mathcal{B}\,|\mathcal{B}|$ gradients before updating the parameters. Note that we don't average all $|\Gamma_k|$ gradients before applying a single gradient step because it would considerably slow down the training.

Algorithm 14 MDPO

for $k = 0, 1, 2, ..., N_{MDPO}$ **do**
  collect set of trajectories Γ_k with π_{θ_k}
  Estimate advantage $\widehat{A}_t^{\pi_{\theta_k}}$
  $\theta_k^{(0)} = \theta_k$
  for $i = 0, ..., N_{MDPO}$ **do**
    $\theta_k^{(i+1)} \leftarrow \theta_k^{(i)} + \eta . \nabla_\theta \Psi(\theta, \theta_k)|_{\theta=\theta_k^{(i)}}$ with Adam
  end for
  $\theta_{k+1} = \theta_{k+1}^{(m)}$
end for

4.1.3.6 Advantage estimation

In order to guarantee policy improvement, previous algorithms highly rely on the quality of advantage estimation which determines if an action should be made more likely based on the

advantage magnitude and sign. Advantage estimation suffers from a bias-variance trade off which penalizes the policy gradient. In case the advantage is estimated with $TD(1)$[181], then the advantage estimates will have a low variance due to the critic low variance but in case the critic is biased then the advantage estimate get biased.

$$\frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{H}\nabla_\theta ln(\pi_\theta(a_t^{(i)}|s_t^{(i)}))(r(s_t^{(i)},a_t^{(i)},s_{t+1}^{(i)})+\gamma.\hat{V}_\psi^\pi(s_{t+1}^{(i)})-\hat{V}_\psi^\pi(s_t^{(i)})) \tag{4.59}$$

In case the advantage estimate use the trajectory return and the average return $b=\mathbb{E}_{\tau\sim p_{\pi_\theta}}[R(\tau)]$ then the policy gradient is unbiased but has high variance because of the single trajectory return $R(\tau_{t:H}^{(i)})$.

$$\frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{H}\nabla_\theta ln(\pi_\theta(a_t^{(i)}|s_t^{(i)}))(R(\tau_{t:H}^{(i)})-b)) \tag{4.60}$$

Note that increasing the parameter k in the following advantage estimate reduces bias of the advantage but increase its variance.

$$\hat{A}_t^{(k)}=\sum_{l=0}^{k-1}\gamma^l.\delta_{t+l}^V=-V(s_t)+\sum_{l=0}^{k-1}\gamma^l.r_{t+l}+\gamma^k.V(s_{t+k}) \tag{4.61}$$

The generalised advantage estimator [166] is built upon this observation and introduces an additional parameter λ to balance bias and variance.

$$\hat{A}_t^{GAE(\gamma,\lambda)}=(1-\lambda).(\sum_{l=0}^{\infty}\lambda^l.\hat{A}_t^{(l)}) \tag{4.62}$$

Using the TD residuals $\delta_t^V=-V(s_t)+r_t+\gamma.V(s_t)$ and the geometric series formula, we can express $\hat{A}_t^{GAE(\gamma,\lambda)}$ efficiently so that it can be computed based on collected transitions (s_t,a_t,s_{t+1}).

$$\hat{A}_t^{GAE(\gamma,\lambda)}=\sum_{l=0}^{\infty}(\gamma\lambda)^l.\delta_{t+l}^{V_\phi} \tag{4.63}$$

Assuming that the value function estimator $V_\phi^{\pi_\theta}$ has a low variance with respect to the true value function V^{π_θ} we have $\mathbb{V}(\hat{A}_t^{GAE(\gamma,\lambda)},\lambda\to 1))>\mathbb{V}(\hat{A}_t^{GAE(\gamma,\lambda)},\lambda\to 0))$ because when $\lambda\to 1$ then $\hat{A}_t^{GAE(\gamma,\lambda)}\to\hat{A}_t^{\infty}$ which has a high number of terms in the summation which usually means higher variance. In contrast $\mathbb{B}(\hat{A}_t^{GAE(\gamma,\lambda)},\lambda\to 1))<\mathbb{B}(\hat{A}_t^{GAE(\gamma,\lambda)},\lambda\to 0))$ because when $\lambda\to 0$ then $\hat{A}_t^{GAE(\gamma,\lambda)}\to\hat{A}_t^{1}$recovers $TD(0)$ which should have less variance but more bias.
Estimating advantages highly relies on the quality of the value function used to compute the TD residual $\delta_t^{V_\phi}$. However estimating the value of states that are less visited or states visited just before episode termination is hard. The value of states less visited is likely to be biased because the value network was trained on very few number of returns. The other issue is induced by episode termination which is necessary due to the structure of the road network which is bounded and not closed. For instance estimating the value of a state at the extremity

of a lane that goes out of the map is problematic because we do not have access to simulation returns for this state since the episode is terminated at this level. Hence the value for those states may be highly biased. We discuss influence of episode termination on the value function in sec.4.1.4.1

4.1.3.7 Performances comparison of policy gradient algorithms

Since TRPO,PPO or MDPO algorithms are still all used for practical applications in various continuous domains [12, 118, 188] for different motivations, we first chose to compare them in our own setting where we simulate a traffic with replayed traffic workers. We train TRPO, PPO and MDPO on the scenario database called *Huge_R_basic* extracted from Interaction dataset whose composition is detailed in annexes .2. We compare the best training performances obtained with the best hyper-parameters found through intensive search. We let each algorithm train for 200 training iterations with a training batch of 100000 transitions which means that 20M millions of transitions will be generated. We chose the lightest observation backbone network called *FCBaseline_basic* introduced in chap.3 to build our actor critic architecture. We select most relevant safety metrics that are expected to be improved according to our simple reward model detailed in sec.4.1.2: the rate of episode with at least one front collision (FCR%), the rate of episode with at least a collision (CR%) and the rate of time spent off-road (Off%) per episode. We first note that TRPO is much more computationally expensive than

	Huge_R			Huge_I			Huge_M		
	FCR%	CR%	Off%	FCR%	CR%	Off%	FCR%	CR%	Off%
MDPO_scratch	2.1	4.0	2.4	1.0	5.0	2.1	1.5	3.2	2.1
PPO_scratch	3.0	4.5	2.6	1.5	6.5	2.5	2.0	3.5	2.2
TRPO_scratch	15.2	23.4	6.6	18.3	28.6	8.3	10.2	15.4	5.2

Table 4.2: Training performances comparison between SOTA actor critic policy gradient algorithms on *Huge_basic_driving* scenario database

other algorithms since the backbone remains relatively large[1] which requires numerous iteration for the conjugate gradient descent[2]. We observe than PPO and MDPO largely outperforms TRPO but we note that performance gains of MDPO are relatively low compared to PPO. We suspect that advantage estimates have a limited accuracy at each epoch which prevents from changing the policy parameters with the appropriate magnitude or orientation. As a consequence enforcing the MDPO regularization with an increasing weight during training makes impossible to improve the performances in case the policy is trapped in a local optimum. We found experimentally that accumulating the gradients of 5 mini-batches of size 50 for the MDPO update led to the best final training performances. Using a bigger batch which means averaging more than 250 gradients per update did not lead to further improvements which

[1]There are much more than 2 hidden layers in our policy network contrary to the original architecture used in [166]

[2]We also tried to freeze a backbone pre-trained with BC but experimental results were not better

confirms a recent theory that explains how to determine the optimal batch size based on an empirical estimate of the gradient noise scale for PPO like algorithms [4]. Additionally we had to clamp MDPO regularization after 100 iterations in case the KL terms gets too big otherwise the KL gradients tends to make training performances unstable[3]. Since MDPO appears less suited for large scale training contrary to PPO as shown for instance in [12], we chose a PPO like implementation for training our driving policy in the next sections.

4.1.4 Influence of the value function

In this section we analyse how training the value function influences the performances of the driving policy. We first explore in sec.4.1.4.1 how episode termination should be handled during training without hindering value estimation. Second, we analyse how the value should be estimated to obtain stable policy improvements in sec.4.1.4.2

4.1.4.1 Episode termination

Intuitively, traffic simulation can be considered as a continuing task with infinite horizon as explained in sec.4.1.1. Since we use a hierarchical driving policy $\pi = \pi_{maneuver} \circ \pi_{planner}$ the maneuver policy has constantly to follow a reference path which only ends in case the vehicle has to park. In practice, we are not interested in parking but rather driving toward a long term goal that will never be reached in simulation. Since the roadmap is bounded the simulator has to end the episode when the agent is about to leave the map or when the agent visits undesired states. For continuing task we can model termination with absorbing states. In case an agent enters an absorbing state then it gets definitely trapped in it whatever action it does. Therefore the agent constantly receives a reward denoted r_a which implies that the true value at s_a for policy π is $V^\pi(s_a) = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k . r_a^{undesired}] = \frac{r_a^{undesired}}{1-\gamma}$.
Absorbing states could be used in different situations: when the agent drives largely off-road or collides with an associate negative reward $r_a < 0$ to encourage the agent to avoid those states. In contrast when the agent leave the map while staying on road we could set $r_a >= 0$ to make such ending more attractive. However this can encourage the agent to leave prematurely the map and absorbing state for collision can make the agent over cautious. Another possibility is to keep bootstrapping at any episode ending as suggested in [143] to evaluate the discounted return of the last state reached. The problem with this technique is that learning accurate estimate $V^\pi(s_T)$ is hard because the value function approximator cannot be trained on s_T since we do not have downstream trajectory to compute value targets for regression.
We analyse experimentally how bootstrapping at the end of episodes or inserting absorbing states influence the driving policy performances. We train a PPO driving policy on $Huge\_R\_basic$ dataset for 100 training iterations, in two different ways. First, we use ex-

[3]Training performances tend to oscillate

clusively absorbing states in case the agent is too far from the road or when it collides and we assign $r_a^{lat} = r_a^{col} = -2.0$. In case the agent passes the limit of the map while staying safe on the road, then it triggers an episode ending and we assign a reward r_a^{end}. Secondly, we launch another experience where we increase the value of the reward provided when the simulation ends r_a^{end} without failure to encourage the agent to finish the episode properly. Finally, we realised an experience where the value function is always bootstrapped at the end of the episodes. We study the final training performances measured with the rate of episode with at least a collision (CR%), the average time spent almost motionless per episode (ML%) [4], and the average distance traveled relative to the expert per episode (L%). In the first experience

	r_a^{lat}	r_a^{col}	r_a^{end}	CR%	ML%	L
PPO_absorbing_col_lat	-2	-2	0	4.5	5	96
PPO_scratch_absorbing_lat	-2	-2	3	5.5	0.0	140
PPO_boostraping	$V_\phi(s_T)$	$V_\phi(s_T)$	$V_\phi(s_T)$	4.5	0.1	120

Table 4.3: Experiment : Influence of episode termination on training performances.

PPO_absorbing_col_lat, we observe that the agent avoids collisions but tends to stay almost motionless close to the end of the map before the simulator triggers a termination. This can be explained by the value assigned to r_a^{end} which makes termination less attractive than staying more time alive. In contrast, in the second experience *PPO_scratch_absorbing_lat* we observe that the agent tends to drive faster than a human expert would do with slightly more collisions. In this setting the value assigned to r_a^{end} seems too high which shows that absorbing state would requires more hyper-parameter search to work as expected. Finally, in the last experience with only value bootstrapping *PPO_boostraping*, we note that collision rate is the lowest and the agent does not stay motion less on the border of the map. This means that the value function can also provides guidance for driving at end of episodes. Even if the distance traveled on the map is still different from the expert, the bootstrapping strategy appears as the best without making harder the implementation.

4.1.4.2 Value objective

The state value function plays a key role in advantage estimation which drives policy improvements. The value approximator V_ϕ^π of policy π is learnt through regression of the episode discounted return. The value target should ideally represent the expected discounted return of the state s.

$$V^{target}(s,t) = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k.r_{t+k}|S_t = s_t]$$

In practice we estimated it based on a trajectory TD return $\eta = \sum_{k=0}^{T-t} \gamma^k.r_{t+k} + V_{\phi_{k-1}}^\pi(s_T)$ or based on the previous value estimate and the advantage estimate [39, 6] as follows:

[4] By motionless we mean that longitudinal speed is lower than 5km.h^{-1}

$$V_t^{target} = V_{\phi_{k-1}}^{\pi}(s_t) + A_{\phi_{k-1}}^{GAE(\gamma,\lambda)}(s_t, a_t)$$

The value objective is computed based on trajectories collected by policy $\pi_{\theta_{k-1}}$ and stored in the training batch $\Gamma_{\pi_{\theta_{k-1}}}$. In case we use the TD return as a target, the objective[5] express as follows:

$$J_V(\phi) = \mathbb{E}_{\tau \sim \Gamma_{\pi_{\theta_{k-1}}}}[\frac{1}{2}.(V_{target}(s,t) - V_{\phi}^{\pi}(s,t)^2) \tag{4.64}$$

$$V^{target}(s_t) = \sum_{k=0}^{T-t} \gamma^k.r_{t+k} + V_{\phi_{k-1}}^{\pi}(s_T) \tag{4.65}$$

In case we use the other value target based on advantage and previous value, the objective is expressed with some clipping to limit the changes of the value target that suffer from the high variance of advantage estimates.

$$J_V(\phi) = max((V_{\phi}(s_t) - V_{target})^2, (clip(V_{\phi}(s_t), V_{\phi_{k-1}}(s_t) - \epsilon_V, V_{\phi_{k-1}}(s_t) + \epsilon_V) - V_{target})^2) \tag{4.66}$$

$$V_{target}(s_t) = V_{\phi_{k-1}}^{\pi}(s_t) + A_{\phi_{k-1}}^{GAE(\gamma,\lambda)}(s_t, a_t) \tag{4.67}$$

Estimating the value function is critical for the policy objective because it determines the sign of the next advantage estimates which will make an action more likely or not. Since the scale of the reward can considerably vary depending on weights of bonus and penalties, it induces a range of trajectory returns with high magnitudes. Instead of normalizing the reward we find it more convenient to normalize the return especially because value targets are computed in a decentralized way. Indeed during data collection after an episode end, there is a post-processing phase which enable to compute discounted return and advantage. Since we need to wait the end of data collection to estimate the mean reward and its standard deviation, this implies to recompute all advantages. In contrast, computing the mean discounted return and its variance on the training batch $\Gamma_{\pi_{\theta_{k-1}}}$ is straightforward. Therefore we normalize the value targets before optimizing the MSE which give:

$$\mu_V = \mathbb{E}_{\tau \in \mathcal{D}}[V^{target}] \tag{4.68}$$

$$\sigma_V = \sqrt{\mathbb{V}_{\tau \in \mathcal{D}}[V^{target}]} \tag{4.69}$$

$$V_{target}(s,t)_N = \frac{(V_{target}(s,t) - \mu_V)}{\sigma_V} \tag{4.70}$$

$$J_V(\phi) = \mathbb{E}_{\tau \sim \mathcal{D}}[\frac{1}{2}.(V_{target}(s,t)_N - V_{\phi,N}^{\pi}(s,t)^2) \tag{4.71}$$

[5]Huber loss can be used to avoid exploding gradients

Once the normalized value is trained, we apply the following affinity transform in order to recover the value at the scale of the simulation rewards.

$$V_{\phi}^{\pi}(s,t) = V_{\phi,N}^{\pi}(s,t)^2.\sigma_V + \mu_V \tag{4.72}$$

In the following experiences, we aim to analyse the influence of different value objectives on the policy driving performances. More precisely we trained a PPO driving policies with the same hyper-parameters on the $Huge\_R\_basic$ scenario database. The first two experiences use the TD return as a value target while the experience is trained with the other value target. For the first experience, we optimize the normalized value function while for the second we optimize an normalized objective. We compare the final training performances after 100 training iterations based on the rate of episode with collision ($CR\%$) and the off-road driving rate $Off\%$ in tab.4.4. We also show in fig.4.2 the evolution of the return during training to illustrate policy improvements.

	CR%	Off%
PPO_scratch_MSE_Return_normalized	4.5	2.6
PPO_scratch_MSE_Return_unnormalized	5.8	3.0
PPO_scratch_MSE_GAE	6.2	3.63

Table 4.4: Influence of the value objective on training performances of PPO.

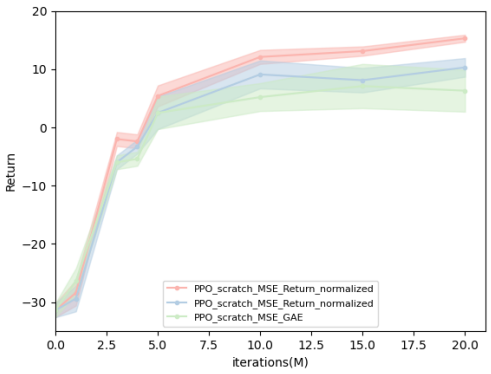

Figure 4.2: Evolution of the return during training for PPO trained with different value objectives

We observe that PPO trained with the GAE value target is very unstable as shown on fig.4.2 since the advantage itself depends on the previous value approximator and errors tends to compound. The other method based on the TD return for value targets led to better performances and we observe that the use of value normalization significantly stabilize policy improvements as shown on fig.4.2. For the remaining sections, we use the TD-return as a value target and we always apply value normalization when we train the value function.

4.1.5 Influence of exploration

During training, the agent needs to explore which actions is the the most advantageous decisions. In model free setting it is difficult to provide efficient exploration guidance because we do not know how the world would have reacted. In sec.4.1.5.1 we study how the policy distribution eanbles to explore. Next we propose to analyse the effect of an entropy regularization on the policy performances in sec.4.1.5.2

4.1.5.1 Policy distribution

In our setting the driving policy parameterize at each decision step t a multivariate gaussian distribution $\mathcal{N}(\underline{\mu_t(o_t)}, \underline{\sigma(o_t)})$ where the mean $\underline{\mu(o_t)} = [\mu_t^{ds}, \mu_t^n]$ is composed of two components the mean longitudinal shift μ_t^{ds} and the mean lateral ordinate μ_t^{dn} desired at $t + \Delta t$. The two components are assumed to be mutually independent to avoid coupling effects during training. In most cases the lateral ordinate has to be reduced up to zero and the policy mainly control forward displacements. The standard deviation $\underline{\sigma(o_t)} = [\sigma_t^{ds}\sigma_t^n]$ specifies how confident the policy is on moving according to the mean action $\underline{\mu(o_t)} = [\mu_t^{ds}, \mu_t^n]$. During training, the policy distribution is sampled in order to explore new actions more or less closed to the mean action while in evaluation we use the mean action to drive. Exploration is done along the whole episode trajectory, so errors related to inappropriate decisions accumulates along time which may gradually lead the agent to undesired part of the state space. Understanding how much to explore around a reference actions provided by the mean $\underline{\mu(o_t)} = [\mu_t^{ds}, \mu_t^n]$ is complex because it depends on the context and how other traffic agents would react. Practically, exploration is mainly required for longitudinal control because policy needs to keep safety distance with front or back neighbor or negotiate with another agent at intersection. During first training stages, exploration is also required to understand that forward displacements at appropriate speed are desirable because we want to follow a reference path. It appears that the context provided by the observation could help to scale the variance. Using a neural network to compute the variance $\underline{\sigma(o_t)} = \sigma_{\theta_\sigma}(o_t)$ from the observation similarly to the mean $\underline{\mu(o_t)} = \mu_{\theta_\mu}(o_t)$ is a straightforward possibility. However it is not guaranteed that the variance is updated the way we expect during policy optimization. Note that the variance for PPO,TRPO,PPG, MDPO appears in three type of terms : the one coming from the policy objective with the probability density, the one coming from the constrain with the KL divergence and eventually the one coming from the entropy. We will analyse the influence of the entropy term in the next sec.4.1.5.2 and restrict our analysis to first two type of terms. As components are considered mutually independent we analyse the gradient of the probability density of one component of the multivariate gaussian: the one for longitudinal shift.

$$\pi_\theta(ds|o_t) = \frac{1}{\sigma_\theta(o_t).\sqrt{2.\pi}} e^{(-0.5*(\frac{ds-\mu_\theta(o_t)}{\sigma_\theta(o_t)})^2)} \tag{4.73}$$

$$\nabla_\theta \pi_\theta(ds|o_t) = \nabla_\sigma \pi(ds|o_t).\nabla_\theta \sigma(\theta, o_t) + \nabla_\mu \pi(ds|o_t).\nabla_\theta \mu(\theta, o_t) \tag{4.74}$$

$$\nabla_\sigma \pi(ds|o_t) = \frac{(\sigma(\theta, o_t)^2 + (ds - \mu(\theta, o_t))^2.}{\sigma(\theta, o_t)^3} \pi_\theta(ds|o_t) \tag{4.75}$$

$$\nabla_\mu \pi(ds|o_t) = \frac{(ds - \mu(\theta, o_t)).}{\sigma(\theta, o_t)^2} \pi_\theta(ds|o_t) \tag{4.76}$$

$$\nabla_\theta \pi_\theta(ds|o_t) = \left[\frac{(\sigma(\theta, o_t)^2 + (ds - \mu(\theta, o_t))^2.}{\sigma(\theta, o_t)^3} . \nabla_\theta \sigma(\theta, o_t) + \frac{(ds - \mu(\theta, o_t))}{\sigma(\theta, o_t)^2} . \nabla_\theta \mu(\theta, o_t) \right] . \pi_\theta(ds|o_t) \tag{4.77}$$

Note that if the standard deviation is allowed to take arbitrary small values when the policy improves and becomes more deterministic then the gradient can get indefinite which stops gradient descent. The same problem occurs with the KL regularization added to the policy objective of KL-PPO and MDPO because as the gaussian behavioral π_{old} policy get more deterministic $\sigma \to 0$ the KL divergence can result in large values which can cause cause numerical instabilities as shown by its analytical expression and the illustration on fig.4.3. Note that the standard deviation tends not only to vanish on states with huge return where it would be desirable but everywhere else [162].

$$\mathbb{D}_{KL}(\pi_\theta(.|o_t), \pi_{old}(.|o_t) \propto ln(\frac{\sigma_{old}(o_t)}{\sigma_\theta(o_t)}) + \frac{\sigma_\phi^2(o_t) + (\mu_\theta(o_t) - \mu_0(o_t))^2}{2.\sigma_{old}^2(o_t)} \tag{4.78}$$

To avoid this situation we can either use a softplus activation with a small positive offset equal

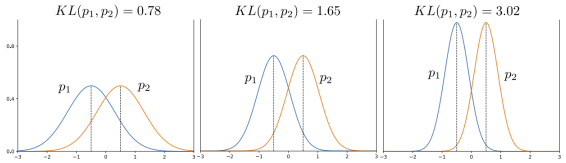

Figure 4.3: Evolution of the KL divergence between two univariate gaussian. The KL divergence is smaller when the variance is larger

to σ_{min} or clamp the variance when it gets smaller than σ_{min}. In the first case, $\nabla_\theta \sigma(\theta, o_t)$ will vanish when $\sigma(\theta, o_t)$ will get close to σ_{min} while in the other case it will be zeroed. In case the variance gets too big which can happen at initialization if weights of neural network are not properly scaled, then exploration will be almost random which tends to slow down the training process. Indeed, GAE can not be properly estimated if the average return of the agent estimated by the value function has very high variance because it behaves randomly. Therefore we also clamp the standard deviation if it gets bigger than a maximal threshold σ_{max}. Note that maximum standard variation for longitudinal shift is chosen such that policy is allowed to accelerate or decelerate around μ_t^{ds} with reasonable amount. In order to avoid interferences induced by $\nabla_\theta \sigma(\theta, o_t)$ during training some works[167] use a free standard deviation parameter for each component such that $\nabla_\theta \sigma(\theta, o_t) = 1$. This choice is also convenient because it enables to set the initial standard deviation $\sigma_{initial}$ at the beginning of training which is not possible when σ depends on observation. In this setting, the policy network just contains an additional free trainable parameters $logStd$ which represents $log(\sigma_{initial})$ where the initial standard deviation can be chosen at the beginning of the training:

$$\sigma = clamp(e^{logStd}, \sigma_{min}, \sigma_{max}) \tag{4.79}$$

$$\tag{4.80}$$

Another issue encountered with gaussian distributions is their infinite support which allows arbitrary big value to be sampled. Even if action are clipped inside the simulator, their are

side effects for the value function estimation of the policy. The value will learn that all actions above the bounds have the same effects which will induce a bias in the policy gradient[27]. A workaround is to resort to squashed gaussian distribution whose KL divergence and entropy can be computed easily[6]. The approach consists in applying a *tanh* activation on top of the gaussian distribution to bound the range of actions. This transformation changes the density of actions : if action u is parameterized as $u = tanh(x)$, where x is a sample from a Gaussian distribution with probability density function p_θ, then the density of u is

$$log(p_u(u)) = logp_\theta(x) - log(tanh'(x)) \tag{4.81}$$

where $x = tanh^{-1}(u)$. This additional $log(tanh'(x))$ term does not affect policy losses because the gradient only use $\nabla_\theta log(p_u(u)) = \nabla_\theta log(p_\theta(x))$. Similarly, this term does not affect the KL divergences which may be used for regularization. The only term affected is the entropy regularization as:

$$H(U) = -\mathbb{E}_u[log(p_u(u))] = \mathbb{E}_x[-log(p_\theta(x) + log(tanh'(x))]. \tag{4.82}$$

This additional $log(tanh'(x))$ term penalizes the policy for taking extreme actions which prevents *tanh* saturation and consequently vanishing gradients. Moreover, it prevents the action entropy from becoming unbounded.
In the following, we propose to analyse experimentally how the policy distribution influence the training performances of a PPO baseline on the database *Huge_R_basic*. We compare two implementations : one using state based standard deviation and one using free standard deviation. We use a standard unbounded gausssian distribution with action clipping set internally in the simulator or a squashed gaussian distribution as detailed above with action clipping enforced in the distribution. We expect to reach improved final performances for the best exploration strategy.As done previously, we consider the rate of episodes with collision (CR%) and the off-road driving rate to evaluate the performances which are reported in tab. 4.5.

	CR%	OFF%
PPO_scratch_variance_unbounded_gaussian	30.4	22.4
PPO_scratch_variance_clamped_gaussian	4.5	2.6
PPO_scratch_variance_clamped_gaussian-free_std	5.5	4.0
PPO_scratch_variance_clamped_squashed_gaussian	5.1	4.1
PPO_scratch_variance_clamped_squashed_free_std	6.1	4.2

Table 4.5: Influence of policy distribution on training performances.

We observe in tab.4.5 that the PPO policies trained with free standard deviations reached lower peak performances on the training set because the standard deviation get quickly clamped at the lowest value which slows down exploration through sampling. In comparison, the two policies trained with state based standard deviation reached better peak performances as shown in tab.4.5 at the cost of instabilities on the return at the end of training because of the gradient

on action densities $\nabla_\sigma \pi(.|o_t)$. It appears on tab.4.5 that the squashed gaussian distribution do not provide quantitative improvements on the performances and this can be explained by the fact that the induced distribution stays symmetric while the probability mass tends to concentrate on extremities of the action supports. As a consequence low speed or high speed become more likely during exploration compared to an unbounded gaussian support which is not necessarily a good strategy in general expect when the driving agent faces an intersection where it should either take or give the way. Since squashed gaussian introduce more computation without strong improvements on the performances we choose to keep a state based variance with an unbounded gaussian support for the remaining part of the work.

4.1.5.2 Entropy regularization

Exploration does not need to be equally large along the whole training. At initialization of the simulation, the decision may be more uncertain than at the end. The main difficulty is to provide enough exploration up to the end of training so that policy can constantly improve. It is known that for continuous action spaces, PPO can prematurely shrink the exploration variance, which leads to slow progress and may make the algorithm prone to getting stuck in local optima [61]. To avoid premature convergence of the policy we leverage the Maximum entropy reinforcement learning framework[231] that optimizes not only the policy expected return but also its expected entropy. In practice we add to the policy objective an entropy regularization term similarly to other works [118, 217]. For a multivariate gaussian distribution $\mathcal{N}(x|\mu, \Sigma)$ the Shannon entropy is given by:

$$H[X] = \frac{1}{2} ln(|\Sigma|) + \frac{D}{2}(1 + ln(2.\pi)) \tag{4.83}$$

In case of a bi-variate distribution whose components are mutually independent it reduces to:

$$H[X] = \frac{1}{2} ln(\sigma_1^2.\sigma_2^2) + (1 + ln(2.\pi)) = H[X_1] + H[X_2] \tag{4.84}$$

Note that the previous expression enables to keep an entropy bonus only for one component. In our setting it is mainly the longitudinal component that requires exploration in order to find efficient collision avoidance strategies in all situation. The entropy bonus is generally weighted by a dynamic coefficient such that if the entropy gets bellow a given threshold then we progressively increase the entropy coefficient whereas the coefficient get decreased when policy entropy get big enough as suggested by the relation bellow that aims to maintain the longitudinal standard deviation between $[0.1, 0.15]$.

$$H[X] < H_{min} = -0.88 \qquad w_{ent} \leftarrow min(w_{ent} * 1.05, 0.05) \tag{4.85}$$

$$H[X] > H_{max} = -0.47 \qquad w_{ent} \leftarrow max(w_{ent} * 0.9, 0.01) \tag{4.86}$$

	CR%	OFF%
PPO_scratch_variance_clamped_entropy_not_regularized	5.3	3.1
PPO_scratch_variance_clamped_entropy_regularized_fixed	4.5	2.6
PPO_scratch_variance_clamped_entropy_regularized_scheduled	4.7	2.9

Table 4.6: Influence of entropy regularization on training performances on *Huge_R_basic* scenario database.

In the following experiences, we investigate if adding an entropy regularization significantly helps to improve driving performance in the long term. We train a PPO policy on the *Huge_R_Basic* database with a state base standard deviation with standard gaussian distribution. We chose to use an entropy regularization only for the longitudinal component of the policy. We start with a fixed entropy bonus weighted by $w_{entropy} = 0.01$ with no entropy target such that the magnitude of the entropy bonus stays smaller than the policy objective. We also train a policy with an adaptive entropy bonus on the longitudinal component as indicated by equ.4.86. The final training results after 100 training iterations are provided in tab.4.6.

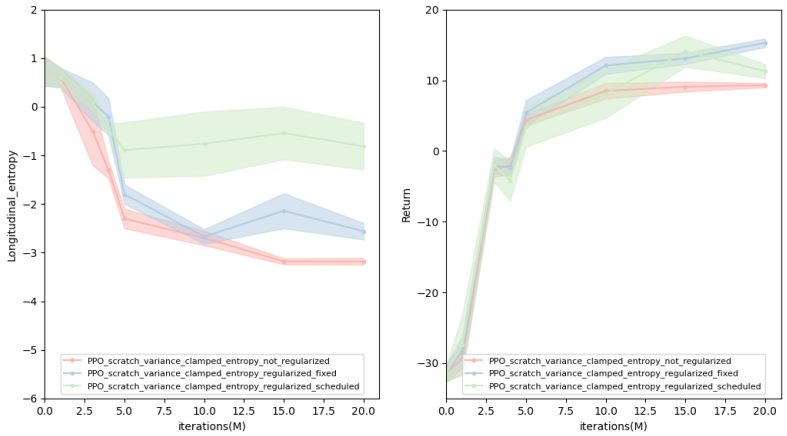

Figure 4.4: Evolution of return and entropy for different PPO policies trained with entropy regularization's strategies on *Huge_R_basic*

We observe on fig.4.4 that without entropy regularization, the longitudinal entropy ends up to stagnate at the minimum value allowed by the minimum variance of the distribution[6] which considerably limit further improvements. When we use a fixed entropy regularization we note that the entropy tends to oscillate but with small amplitudes which helps to improve the return up to a certain extent. When an adaptive entropy regularization is used, we reached slightly better peak performances at the expense of return instabilities during training. Indeed, maintaining the entropy inside bounds increasingly interferes with the policy objective if the constrains are not satisfied after several training iterations. As a conclusion, we keep using a fixed entropy regularization instead of an adaptive regularization strategy which would require

[6]The minimum standard deviation is set to $\sigma_{ds} = 0.01$ which corresponds to an entropy equal to $0.5.log(2.\pi.\sigma_{ds}^2) + 0.5 \approx -3.18$.

more hyper-parameters search to properly scale the entropy weight.

4.1.6 Influence of pretraining

Pretraining can also be used in RL to learn priors over representations or dynamics. We first investigate in sec.4.1.6.2 how pretraining with expert actions can improve the policy performances before proposing another pretraining method in sec.4.1.6.2 that better exploits the sequential nature of decision making.

4.1.6.1 Pretraining a policy with expert data

In the previous chapter we already showed that the driving policy could be trained on expert demonstrations to plan trajectories. Experimental results suggests that the network is able to exploit suitable features from the observation. While open loop test performances are satisfying i.e the learner plans similarly to the human drivers on the expert state support, closed loop test performances which imply simulations and interactions reveal that collisions are numerous. This problem is related to the distributional shift because the pretrained agent progressively deviate from expert trajectory and encounters situations it never saw in the expert dataset. New situations can be critical in terms of safety but since no expert demonstrations show how to properly recover from those situations its is hard for the pretrained agent to adapt [30, 43]. However experiences in the previous chap.3 show that early five seconds of simulation are generally well handled by the pretrained agents which suggests that pretraining could potentially accelerate the RL training. However learning from expert trajectories may lead a policy that exploit radically different features compared to a policy trained on noisy advantage estimates. Fine tuning a pretrained policy with RL may overwrite features learnt from expert demonstrations which is known as catastrophic forgetting [45]. One way to combat catastrophic forgetting is experience replay [160], in which the old data is interspersed with the new data, simulating i.i.d. data such that the network retains the old knowledge. In RL we can replay expert demonstration stored in a separate buffer during online training and add an imitation loss to the policy objective. The imitation loss can be progressively annealed so that RL can adjust its behaviour if necessary when the agent faces situation out of expert distributions [79, 222]. However the two losses may still interfere because imitating the expert on safe situations may not requires the same features as avoiding collision in all unsafe situations that may occur at the beginning of the training. In the following, we first investigate how pretraining by maximizing the likelihood of expert action on expert state distribution influence the final driving performances. We train four baselines on the scenario database called $Huge\_R\_basic$ such that : one is trained with RL from scratch, another one is just pretrained with BC, another is pretrained with BC and then fine-tuned with RL and a last one is pretrained with BC fine-tuned with PPO with an additional imitation loss: $\mathcal{L}(\pi_\theta) = \mathbb{E}_{B\sim\mathcal{D}}[-log(\pi_\theta(a^e|s^e))]$ computed on expert data $\mathcal{D}$ added to the policy loss and decayed every training iteration

$J(\pi_\theta) = J^{PPO}(\pi_\theta) + w_{ILL}.\mathcal{L}(\pi_\theta)$. In all experiences where PPO is used, we report the final training performances after 100 training iterations. While pretraining with BC enables to stay

	ADE-5	CR%	Off%
BC	5.20	55	10.30
PPO_scratch	4.80	4.5	2.6
BC_PPO_finetuned	4.43	4.7	2.8
BC_PPO_finetuned+imitation_loss_annealed	3.60	4.3	2.5

Table 4.7: Influence of pretraining and fine-tuning with PPO on training performances on *Huge_R_basic* scenario database.

relatively close to the expert, it cannot stay safe during closed loop evaluation. We observe in tab.4.7 that final performances of PPO trained from scratch measured in terms of safety (CR%,Off%) are relatively close to other baselines pretrained with BC. We observe that the best safety performances are reached with the initiation loss annealing but the gap with PPO trained from scratch is tight. We think that the low data regime (relatively small amount of training data) penalizes behavioural cloning [7] but we also realise that RL tends to learn new features at the beginning of the training since we noted a moderate rise of collisions in early RL training iterations which indicates that the policy neural network has to be deeply modified before benefiting from RL fine-tuning. This issue is not related to over-fitting during supervised learning because we stopped the BC training before it starts to overfit on the training data. It can come from the value function that may exploit different features to predict the return even if it is first trained on few trajectories before the policy is updated[8]. The imitation loss tends to align features learned with the RL policy and the ones learned through supervised learning during the first 50 training iterations but the last 50 training iterations are completely dominated by the PPO loss. We noticed that keeping the imitation loss for more than 50 training iterations was detrimental for final safety performances. Those results suggests that simply maximizing the likelihood of expert actions in addition to a PPO loss is not enough to significantly improve safety performances reached with an RL based policy which motivates the development of new algorithm introduced in chap.5.

4.1.6.2 Pretraining a planner

Instead of just regressing a single action at each decision step, it is also possible to parameterize a policy as a planner which will just apply the first action given in the plan at each decision step as detailed in chap.3. Learning to plan as an expert from an observation may enable to acquire better representations of the environment dynamic before applying model free RL[169]. In or-

[7] Adding more examples would help to improve the driving performances but cannot solve the distributional shift issue.

[8] Note that the value function is separated from the policy network but we initialize the value backbone with the weights of the policy backbone before starting RL training. Additionally, the first data collection just enables to update the value function.

der to generate a plan, we can first consider that consecutive actions are independent with each other which corresponds to the ILL planner introduced in chap. 3. The ILL planner models the next fifteen actions as independent Gaussian variables where the longitudinal and lateral components are also considered independent at each decision step. This assumption gives more freedom to the network during training but still forces the backbone to extract features useful for planing at every decision step. In the following, we compare the closed loop performances on the scenario database *Huge_R_basic* obtained by a policy fine tuned with PPO for 100 training iterations and eventually pre-trained as an imitative planner. We observe in tab.4.8

	ADE-5	CR%	Off%
Planner_ILL_pretrained	4.25	46	8.0
Planner_ILL_PPO_from_scratch	4.80	4.5	2.6
Planner_ILL_pretrained_PPO_finetuned	4.30	4.4	2.7
Planner_ILL_pretrained_PPO_longitudinal_only_finetuned	4.27	4.4	0.0

Table 4.8: Influence of pretraining with the ILL planner on the training performances.

that just pretraining is not sufficient to learn how to avoid collision and that the agent should stay on-road. Pre-training the policy only marginally improves the final collision rate of the PPO agent even if the PPO agent start with a lower collision rate. We observe that pretraining the planner and then only training the longitudinal part of the policy while setting the lateral ordinate to 0[9] does not result in lower collision rates which indicates that the difficulty comes from the environment dynamic and not necessarily to the additional lateral component.
The main limitation of the ILL planner is the lack of temporal consistency between actions which may be exacerbated with the distributional shift. In contrast our auto-regressive planner (*Planner_ILL_AUTOREG*) introduced in chap.3 enforces dependency between consecutive actions through recursion in a latent state space. Interestingly, the *Planner_ILL_AUTOREG*) also enables to exploit additional simulated data even during the pre-training stage without queering an expert during the data collection. This is important because switching from pre-training to RL finetuning induces a large change in the training data distribution exacerbated by the exploration process. During the pretraining stage we propose to run data collection periodically to build an online dataset of transitions generated by the current pre-trained policy in exploration mode[10]. The online dataset put transitions similar to the ones that the RL training stage will induce which enables to reduce the change in the training data distribution between the pretraining and the RL phases. To leverage the online dataset $\mathcal{D}_{online}$, we propose two losses computed on it as well an entropy terms that maintains exploration after pretraining such that RL can work efficiently[11]. The first one denoted $\mathcal{L}^{pred}_{online}$ consists in a prediction loss and only exploit state transitions (s_t, a_t, s_{t+1}) induced by sampling the policy as explained in sec,3.2.2.2 in chap.3. The prediction loss is expected to enforce consistency between consecu-

[9] the agent stays on the centerline

[10] Note that exploration means that the action will be sampled from the first action distribution parametrized by the planner

[11] If no entropy term is added during pretraining, the RL policy may not explore anymore after the pretraining.

tive observations reached by following the policy. The second loss $\mathcal{L}_{online}^{value}$ is more related to the downstream application of PPO algorithm for finetuning. Since PPO requires a value function to estimate the advantage, the features exploited by the policy should ideally match the ones used to predict the value. In this case the policy is expected to learn actions with high value while the value is expected to accurately learn the value induced by consecutive actions selected by the policy. For each trajectory collected in the online dataset $\mathcal{D}_{online}$ we compute a TD value target and we learn a value approximator with an additional value head that exploit the observation embedding used in conjunction with the planner[12]. As a consequence, after each data collection that are launched every N pretraining iterations the two set of losses expressed bellow can be optimized with Stochastic Gradient Descent(SGD).

$$\mathcal{L}_{offline}^{planner} = \mathbb{E}_{s_t, a_{t:t+H}^e \sim \mathcal{D}_e^{offline}}[\sum_{k=1}^{H} -log(\pi_\theta(a_{t+k}^e|, t+k, s_t)] \tag{4.87}$$

$$\mathcal{L}_{online}^{entropy} = \mathbb{E}_{s_t \sim \mathcal{D}_{\pi_{\theta_{k-1}}}^{online}}[(\mathcal{H}_{target} - \mathcal{H}(\pi_\theta|s_t))^2] \tag{4.88}$$

$$\mathcal{L}_{online}^{value} \quad \mathbb{E}_{s_t, V_{t+1}^{\pi_{\theta_{k-1}}} \sim \mathcal{D}_{\pi_{\theta_{k-1}}}^{online}}[(V_{t+1}^{\pi_{\theta_{k-1}}}(s_t) - V(s_t, \theta, \phi))^2] \tag{4.89}$$

$$\mathcal{L}_{online}^{pred} \quad \mathbb{E}_{s_t, a_t, s_{t+1} \sim \mathcal{D}_{\pi_{\theta_{k-1}}}^{online}}[0.5 * (h_{t+1}^t(\theta) - h_{t+1}^{t+1}(\theta))^2] \tag{4.90}$$

The data collection periodicity N during the pretraining phase is critical for the training stability because if we train for too many epochs the value function learned may get too different from the value function of the current policy. We found experimentally that 5 epochs is the best compromise: it reduces value loss peaks at each new data collection and it avoids to launch data collection too frequently.

In the following, we compare how the planner $Planner\_ILL\_AUTOREG$ performs on the $Huge\_R\_Basic$ scenario database when we apply pretraining and then train the planner with PPO for 100 training iterations. We observe that pretraining the planner $Planner\_ILL\_AUTOREG$

	ADE-5	CR%	Off%
Planner_autoreg_pretrained	3.95	38	7.0
Planner_autoreg_PPO_from_scratch	4.80	4.5	2.6
Planner_autoreg_pretained_PPO_finetuned	4.1	4.5	2.1
Planner_autoreg_pretrained_PPO_finetuned_pred_loss	3.72	4.7	0.8
Planner_autoreg_pretrained_PPO_finetuned_value_loss	4.23	4.1	1.9
Planner_autoreg_pretrained_PPO_finetuned_value_pred_loss	3.74	4.2	1.1

Table 4.9: Influence of pretraining the auto-regressive planner and fine-tuning with PPO on training performances

on the offline dataset $\mathcal{D}_e^{offline}$ enables to slightly reduce the collision rates and the off-road driving rate compared to directly train from scratch. We also observe that during the five first

[12]Note that during downstream RL finetuning the value weights are copied in a separate value networks. The basic neural network architecture for PPO used until here, uses two separate networks for the policy and the value function

second of simulation, the policy reproduce more closely the expert trajectory if we pre-train the planner. We observe that the prediction loss further reduces the off-road driving rate while the value loss reduces moderately the off-road driving rate and the episode collision rate. The influence of the value is notable at the beginning of the RL training because it reduces the initial value error. However the value loss does not enable to significantly improve final training performances because PPO is able to compensate the initial value error after some training iterations. Finally we see that combining all the losses during pretraining with a weighted sum ($w_{offline} = 1, w_{pred} = 1, w_{value} = 1, w_{entropy} = 0.01$) enables to reach the best final training performances. Interestingly, pretraining with the $Planner\_ILL\_AUTOREG$ resulted to better final training performances than pre-training with the $Planner\_ILL$ as shown by the final results in respectively tab.4.9 and tab.4.8.

4.2 Learning basic driving skills

In the previous section, we analysed driving performances on scenarios used for training in order to understand how to properly configure RL training. In this section, we are interested in understanding if the driving policy is able to generalize safe behaviour on new scenarios for real use cases in traffic simulation. We first explain in sec.4.2.1 how our actor critic policy gradient algorithm can be modified to get better test performances on scenarios unseen during training. Secondly, we analyse the influence of backbone architecture on the test performances in sec.4.2.1.3. Finally, we analyse the influence of the environment dynamic during training on the test performances in sec.4.2.1.2.

4.2.1 Toward robust policy optimization

We first explain in sec.4.2.1.1 why generalization in RL does not benefit from the same guarantees as in supervised learning. Subsequently in sec.4.2.1.2, we propose to modify the actor critic architecture of PPO such that the value can better guide the policy during training .

4.2.1.1 Generalization in reinforcement Learning

The sequential nature of RL changes the way an agent can generalize in new scenarios. Generalizing how to take the appropriate action at a given state does not only depends on the state as it would have been in supervised learning but also to the transition dynamic at this state. Since decision errors compounds at every decision step, starting from an unusual context with differences in the environment dynamic can quickly lead to dramatic failures. RL agents are prone to over-fitting to particular training distribution and tend to memorize specific environment dynamic [216]. A simple way to reduce over fitting first consists to introduce more diversity and stochasticity in environment similarly to data augmentation methods in

supervised learning [28]. Regularization methods have also proved their efficiency for policy optimization as for instance L2 regularization, dropout or batch normalization [118]. In order to obtain a policy that is robust to possibly out-of-distribution environments, Ensemble Policy Optimization (EPOpt)[155] maximizes the expected reward over the fraction of environments with worst expected reward which was shown to improve robustness to environment changes [141]. More generally observation features may not transfer to new environments because the agent may mistakenly correlates reward with certain spurious features from the observations generated during training. Learning general representations for measuring behavioral similarity between states without the influence of the reward was studied in [3, 177] but it requires a metric on the state space which is not always easy to define. Another major issue for generalization, is influence of the discount factor on the estimation of the advantage function. In order to balance bias and variance of advantage one can adjust the discount factor as suggested in [186] or using hyperbolic discounting [129] when the length of episodes vary which is often the case when real replayed episodes are used. In the following, we first propose to modify PPO such that policy can better benefit from features learned by the value. Thereafter, we focus our attention on the influence of the training environment on the test performances.

4.2.1.2 Decoupling Policy and Value

In order to guarantee robust policy improvements at each PPO iteration, we need to collect large amount of trajectories on multiple scenarios otherwise the policy is likely to overffit on just a subset of driving scenarios. Policy optimization on large training batch can be challenging for the policy and the value function which may learn significantly different features for their respective observation backbones since their networks are originally separated in PPO[167]. In case the policy does not exploit the same features as the value then the value may struggle to predict the return induced by the policy since it ignores how actions are taken. In case policy and value function use the same observation backbone then policy and value optimization should be done simultaneously but the gradients for the two objectives may be distributed in a significantly different way due to high variance of advantages and value targets[4]. Phasic Policy Gradient(PPG) [29] proposed to preserves the feature sharing between the policy and the value function, while decoupling their training. Decoupling their training is interesting because value learning tolerates higher level of sample reuse than the policy since the value should learn accurately value targets[13] while the policy should only be slightly modified along the policy gradients directions in a trust region.

PPG operates in two alternating phases: the first one called policy phase trains the policy and thereafter the value similarly to PPO. The second phase is called auxiliary phase which distills useful features in the policy network through a value head $V_{\theta_{v_H}}$. The value head is trained on a buffer to regress values generated by a separate value function $V_{\theta_{v_S}}$ as depicted on the right side in fig.4.5. The auxiliary phase occurs after several policy phase and exploits recent trajectories

[13]Low value bias is critical for accurate advantage estimate whose sign is particularly important

stored in a buffer as explained in algorithm 15.

$$L^{V_{head}}(\theta) = \mathbb{E}_{s_t \sim \mathcal{B}}[\frac{1.0}{2.0}.(V_{head}(s_t, \theta_t) - V^{target}(o_t))^2] V^{target}(o_t) = V_{separate_head}(s_t, \phi_t) or TD return \tag{4.91}$$

During the auxiliary phase, the policy backbone learns return based features but it is important to preserve the policy outputs which are not intended to be changed during the auxiliary phase. The auxiliary loss integrates a KL constrain with the last policy trained to avoid policy changes.

$$\mathcal{L}^{joint} = L^{V_{head}}(\theta) + \beta.\mathbb{E}_{s_t \sim \mathcal{B}}[KL(\pi_{old}(.|s_t)|\pi_\theta(.|s_t)))] \tag{4.92}$$

While PPG enables to align policy and value features while reducing interferences, the official

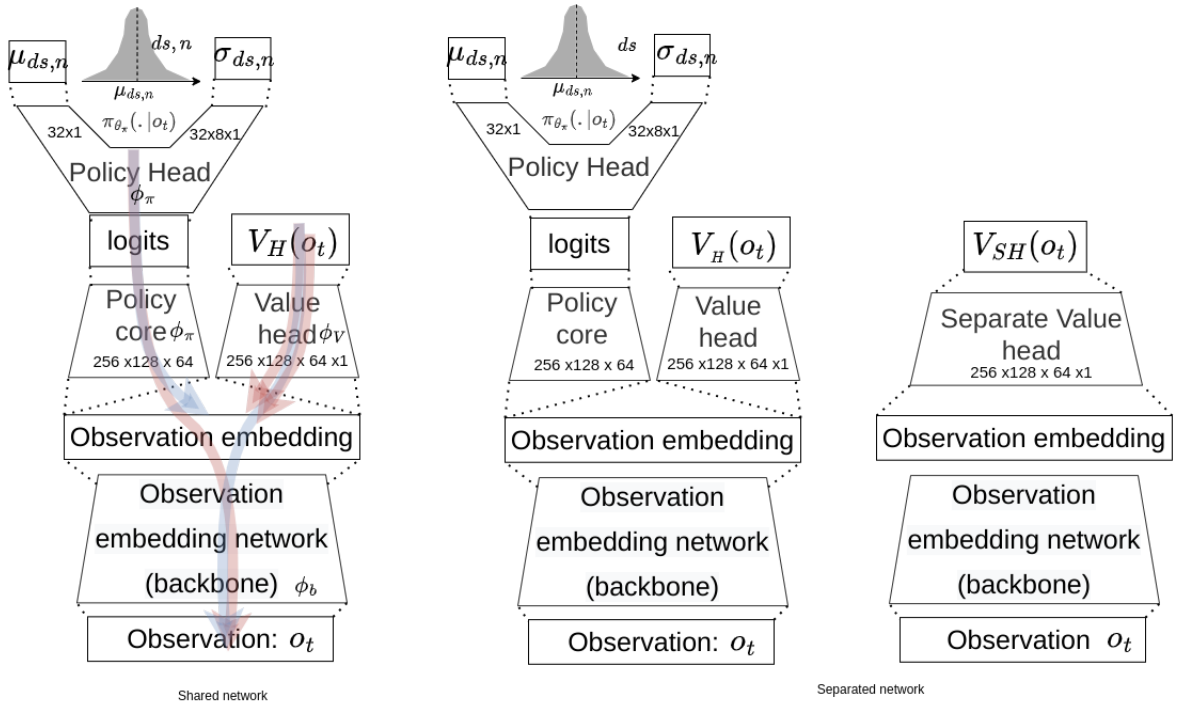

Figure 4.5: Actor critic decoupling

implementation still uses two separate backbones which slows down the training especially when the backbone is very deep. Therefore we propose a shared backbone network $E(s_t)$ depicted in the left side of fig.4.5 with a specific training procedure to handle value and policy gradients that flow through the backbone.

$$\begin{cases} V_{\theta_V}(s_t) = h_{\phi_V}(E(s_t; \phi_b)) \\ \pi(.|s_t) = h_{\phi_\pi}(E(s_t; \phi_b)) \end{cases} \tag{4.93}$$

The above equations suggests that SGD for updating the policy and SGD for updating the value can be done in two consecutive phases. First the policy can be updated $\theta_k \rightarrow \theta_{k+1}$ on the training batch with advantages estimated with fixed values $V\phi_k$. In a second step the value can be updated $\phi_k \rightarrow \phi_{k+1}$ on the same training batch but without letting the gradient

flowing through the backbone. Note that in this case the value function can only exploit the encoding $z_t = E(s_t; \phi_E)$ provided by the encoder updated during the policy phase $E(s_t; \phi_E)$ but predicting accurate value estimate may require specific features not currently provided. For instance, after the backbone get updated by the policy loss, it can happen than two observations o_t and o'_t get close latent representation z_t and $z_{t'}$ because they require the same action whereas their values are significantly different which will induce a bias for the value function. As a consequence it may be more interesting to train first the value ($V \rightarrow \pi$), freeze the observation backbone and then train the policy head. We will analyse the influence of the update order in the next experiences and we propose a training procedure that cycle between the two procedures after each data collection as indicated by the colored arrows in figure 4.5. We call this variation Cyclic Policy Gradient which modifies PPG algorithm at the policy phase as detailed in alg.15. Inspired from the auxiliary phase, we also add a constrain on the policy when we start to train the value and thereafter the policy with the backbone frozen. We apply the same principle when we start to train the policy and we introduce two losses $J^{(\pi \rightarrow V)}, J^{(\rightarrow \pi)}$ that we will replace the standard PPO policy loss $J^{PPO}(\theta)$ in our CPG algorithm. Instead of an auxiliary phase, we just enable the value function head to be fine tuned for some additional epochs E_{fine} on the training batch with a loss denoted $J^{V \rightarrow \pi}$. Note that the actor critic architecture that we propose has backbone parameters denoted ϕ_b, policy head parameters denoted ϕ_π, value head parameters denoted ϕ_V, policy parameters $\theta_\pi = [\phi_b, \phi_\pi]$ and value parameters $\theta_V = [\phi_b, \phi_V]$.

$$J^{(\pi \rightarrow V)} = \mathbb{E}_{(s_t, a_t) \sim \mathcal{B}}[J^{PPO}(\theta) + \beta_V . J^V(\theta)] \tag{4.94}$$

$$J^{(V \rightarrow \pi)} = J^V(\theta) + \beta_\pi . \mathbb{E}_{s_t \sim \mathcal{B}}[KL(\pi_{old}(.|s_t)|\pi_\theta(.|s_t)))] \tag{4.95}$$

In the next experiences, we aim to analyse if PPG and the modified version called CPG can obtain better test performance than the standard variation of PPO thanks to new policy and value features. We first evaluate a standard PPO implementation with two separate backbones and fully separate policy and value training. We evaluate another PPO baseline where the backbone is shared where the policy and the value losses are summed during training with no additional training for the value function. We evaluate a standard PPG implementation with separate backbones. Additionnaly, we also evaluate PPG algorithm with a shared backbone with two training procedures. The first one, trains the whole policy network then freeze the backbone and then trains the value head. The other one trains the whole value network then freeze the backbone and then train the policy head. Finally, we also evaluate CPG by first setting all the constrains to zeros with ($\beta_V = 0.0, \beta_\pi = 0.0$) and then enforcing the constrain with best weight values found experimentally ($\beta_V = 0.01, \beta_\pi = 1.0$). Note that we trained each algorithm on the $Huge\_R\_Basic$ scenario database up to the point where we observe stagnation or degradation of test performances which can be considered as beginning of over-fitting. Regarding the architecture of the policy, we realised the next experiences with the lightest backbone called $FCBaselineBasic$ with the the auto-regressive planner head. We first observe in tab.4.10 that the test performances of $PPO\_scratch\_separate$ are very

Algorithm 15 Cyclic policy gradient

for $i = 1, 2, 3, ..., N_\pi$ **do**
  $\Gamma \leftarrow \text{ParrallelRollouts}(\pi_k)$
  Policy phase
  if $i\%2 == 0$ **then**
    for $k = 1, 2, 3, ..., E_\pi$ **do**
      $\theta_\pi^{k+1} \leftarrow \theta_\pi^k + \alpha.\nabla_{\theta_\pi} J^{\pi \to V}(\theta_\pi^k)$
    end for
    for $k = 1, 2, 3, ..., E_V$ **do**
      $\phi_V^{k+1} \leftarrow \phi_V^k + \alpha_v.\nabla_{\phi_V} J^V(\phi_V^k)$
    end for
  else if $i\%2 == 1$ **then**
    for $k = 1, 2, 3, ..., E_V$ **do**
      $\theta_V^{k+1} \leftarrow \theta_V^k + \alpha.\nabla_{\theta_V} J^{V \to \pi}(\theta_V)$
    end for
    for $k = 1, 2, 3, ..., E_\pi$ **do**
      $\phi_\pi^{k+1} \leftarrow \phi_\pi^k + \alpha_\pi.\nabla_{\phi^\pi} J^{PPO}(\phi_\pi^k)$
    end for
  end if
  Value fine-tuning
  if $i\%N_{fine} == 0$ **then**
    for $k = 1, 2, 3, ..., E_{fine}$ **do**
      $\phi_V^{k+1} \leftarrow \phi_V^k + \alpha_{\phi_V}.\nabla_{\phi_V} J^V(\phi_V^k)$
    end for
  end if
end for

	train:CR%	train:Off%	test:CR%	test:Off%
Planner_autoreg_PPO_from_scratch	4.5	2.6	17.4	5.3
Planner_autoreg_PPO_from_scratch_shared_summed	5.2	3.2	21.1	6.3
Planner_autoreg_PPG_scratch_separate	4.4	2.5	9.6	4.3
Planner_autoreg_PPG_scratch_shared_policy_then_value	4.0	2.8	14.4	5.7
Planner_autoreg_PPG_scratch_shared_value_then_policy	4.4	2.8	16.4	5.8
Planner_autoreg_CPG_scratch_unconstrained	4.6	2.6	13.4	5.3
Planner_autoreg_CPG_scratch_constrained	3.9	2.4	9.4	4.3

Table 4.10: Comparison of training and test performances of CPG with respect to several baselines on *Huge_R_Basic* scenario database.

limited with high collision rates on new test scenarios. Sharing the backbones and summing the policy in value losses is not beneficial as shown by *PPO_scratch_shared_summed* test results. The test results of PPG are better than the PPO baselines but the collision rate is still high for practical use cases. We notice that the versions of PPG that uses a shared backbones obtain competitive results with *PPG_scratch_separate* which shows that reducing interferences between objective enables to use a single backbone and hence fewer parameters. We Note that the unconstrained version of CPG obtained slightly better performances than the PPG baselines with a shared backbone which shows that cycling has a positive effects. Remarkably, the application of CPG with soft constrains brings more significant improvements in

terms of collision rate. The $CPG\_soft\_constrained$ reached competitive results with the best PPG baseline: $PPG\_scratch\_separate$ with only one observation backbone and outperforms $PPG\_shared\_policy\_then\_value$ and $PPG\_shared\_value\_then\_policy$. The main limitation of $CPG\_soft\_constrained$ is related to the soft constrains weights which require some hyper-parameters search to work properly. However PPG also requires search to determine the periodicity of the auxiliary phase and the strength of the KL penalty in the auxiliary loss.

4.2.1.3 Influence of observation backbone architectures

We saw that the policy and the value can share the observation backbone to acquire efficient representation however features extraction mainly depends on the observation encoding which provides all information about the scene context. In this chapter we applied actor-critics policy gradient methods leveraging the lightest backbone we developed : $FC\_baseline\_basic$ since it enables fast forward and backward pass through the computational graph and hence fast training[14]. The main drawback of the observation backbone $FC\_baseline\_basic$ is that the observation does not provide the local road-network but only the lane corridor where the policy is expected to move which deprives the value and the policy from important priors about other agents future displacements. In contrast, we have shown in chap.3 that the observation backbone called $PointNetMHA$ can provide an embedding of the local context represented as a 2D point cloud at the cost of a higher computational load. Since the network ha a considerably more capacity and leverage an attention mechanism to extract appropriate features from the context, it could in theory learn more advanced decision making. In the following we propose to analyse the test performances that can be acquired with different observation backbones when the policy is first pre-trained on real data as detailed in sec.4.1.6.2 and then fine tuned with CPG with soft constrains. We compare the test performances of several observation backbones : the ones that only have access to the lane corridor to represent the map ($FCBaselineBasic$,$FCBaselineAttentive$) and the backbone that has access to a local representation of the map($PointNetNHA$). We realised our experiences on the scenario data-base called $Huge\_R\_basic$ and we fine tune with RL for 200 training iterations. We provide the test performances in tab.**??**. We observe in tab.4.11, that the pretraining phase bring signif-

	Huge_R		Huge_I		Huge_M	
	CR%	Off%	CR%	Off%	CR%	Off%
PointNetMHA_Planner_autoreg_pretrained_CPG	5.4	2.6	8.4	2.7	4.4	3.5
FCBaselineAttentive_Planner_autoreg_pretrained_CPG	6.2	2.2	9.8	2.5	6.2	3.7
FCBaselineBasic_Planner_autoreg_pretrained_CPG	6.4	2.9	9.4	3.2	6.0	3.6
FCBaselineBasic_Planner_autoreg_scratch_CPG	9.4	4.3	13.4	3.8	8.3	4.3

Table 4.11: Comparison of test performances for different observation backbones on $Huge\_R\_basic$ scenario database..

icant improvements in test performances and notably on complex scenarios like the ones on

[14]In general, the training we realised lasted almost 2 days running on a RT2080 GPU with 50 parallel simulations for data collections.

the intersection. This can be explained by the fact that collision avoidance strategy on new scenarios with a replayed traffic are more difficult to handle when the policy start to drift from the expert trajectory because other agents won't adapt their strategy. We observe that the best final test performances were obtained by the pretrained $PointNetMHA$ backbone which has almost no front collisions but can still get collided from the back or from the sides. Note that the pretraining phase is essential for $PointNetMHA$ which cannot be trained efficiently from scratch with RL due to its considerable size (777250 parameters). We observe that remaining crashes are almost all caused by the choice of an inappropriate action made at some critical time step which prevents the agent from adapting to the future traffic that is nonreactive since replayed from real demonstrations. Intuitively the policy enters in a kind of trap where it is often impossible to adapt because there is not enough space between the policy and other traffic agents as shown in fig.4.6. The virtual expert in shaded red chose another strategy compared to the policy in dark red which got trapped between an agent on its left and an agent in the back. In the next section we use interactive rule based traffic agents instead of replayed traffic agents which enables to reduce this kind of issues.

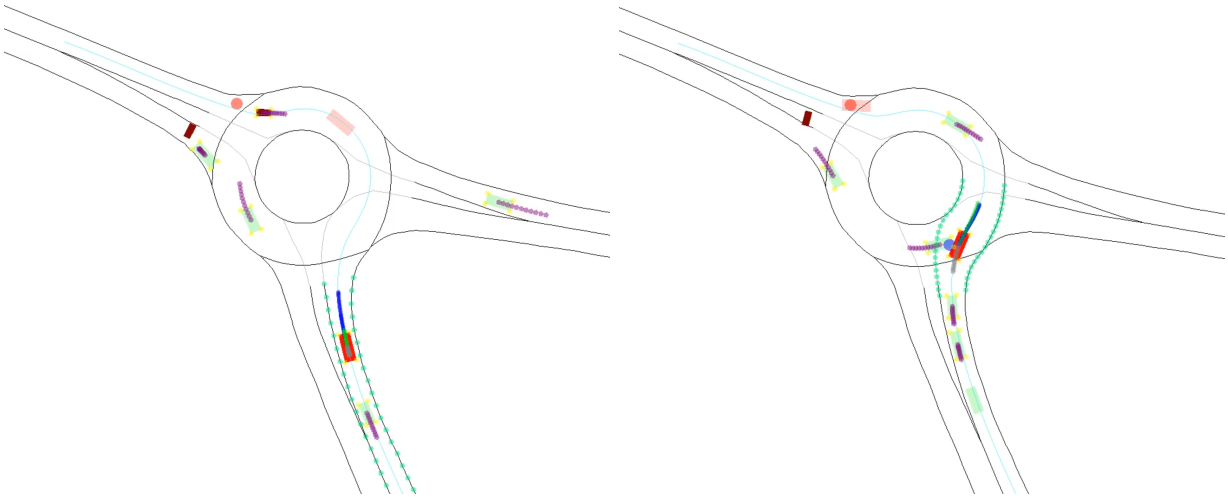

Figure 4.6: A critical situation where the driving policy represented in dark red got trapped between two replayed traffic agents: the image on the left side represents the situation before the trap closes and the image on the right represents the collision with a blue point.

4.2.2 Influence of environment dynamic

In this section, we propose to investigate how the environment dynamic influences the test performances on new scenarios eventually interactive. We first analyse in sec.4.2.2.1 how replacing replayed traffic agents with interactive traffic agents during training influences the test performances on replayed and interactive scenarios. In a second time, we study in sec.4.2.2.2 how increasing the diversity of interactive agents influences the test performances on replayed and interactive scenarios.

4.2.2.1 Replayed or Interactive traffic

In chap.2, we explained how to design interactive agents based on an extension of the rule based IDM model [190] such that it can better handle intersections with fewer collisions. We introduced a decentralized version called DIDM and we have shown in sec.2.2.3.2 that it can evolve more safely than IDM agent for scenarios of the $Huge\_R\_basic$ database. In this section, we propose to train a driving policy with the lightest observation backbone $FCBaselineBasic$ on the basis of all real scenarios taken from the $Huge\_R\_basic$ database but for each scenario originally simulated with replayed, we replace the replay agents with the DIDM agents which gives the scenario database called $Huge\_R\_basic\_synthetic\_DIDM$. Instead of just including replayed agents in the environment dynamic, as in $Huge\_R\_basic\_replayed$ database or just interactive DIDM agents as in the $Huge\_R\_basic\_synthetic\_DIDM$ database, we propose to simulate each real scenario twice: once with replay agents and a second time with a DIDM agents which gives the database called $Huge\_R\_basic\_mixed\_DIDM$. Whereas $Huge\_R\_basic\_replayed$ and $Huge\_R\_basic\_synthetic\_DIDM$ only have 750 training scenarios the $Huge\_R\_basic\_mixed\_DIDM$ database have twice more. We train our policy from sratch on the different scenario databases with the same training batch size equal to 100000 transitions and we report the performances on test set of $Huge\_R\_basic$ once with replayed agents and once with DIDM agents to understand if the driving policy can generalize a safe strategy with various environment dynamic. Regarding the architecture of the policy, we realised the next experiences with the lightest backbone called $FCBaselineBasic$ with the the auto-regressive planner head. As in the previous section we use CPG with soft constrains to train the actor citic architecture. We observe in tab.4.12 that CPG trained only with replay

	test			
	replay workers		DIDM workers	
	CR%	FCR%	CR%	FCR%
Planner_autoreg_CPG_scratch_replayed	9.4	4.2	15.4	10.2
Planner_autoreg_CPG_replayed_pretrained	6.4	4.1	18.4	13.3
Planner_autoreg_CPG_scratch_synthetic_DIDM	34	3.4	6.2	4.0
Planner_autoreg_CPG_scratch_mixed_DIDM	10.8	3.7	12.1	7.1
Planner_autoreg_CPG_pretrained_mixed_DIDM	7.4	3.7	7.6	4.1

Table 4.12: test performances of driving policies trained with different environenement dynamics

agents obtain better test performances on replayed scenarios at the expense of high collision rates on synthetic scenarios which means that the driving policy tends to over adapt to a specific dynamic. When the driving policy is trained from scratch on synthetic scenarios it reaches the best test performances on synthetic scenarios but it gets often collided from the back on replayed scenarios because non reactive replay worker do not stop to avoid collisions in contrast to DIDM agents. We note that a reasonable trade off is reached when the driving policy is trained on both replayed and interactive scenarios. Additional pretraining as done in the experience $CPG\_pretrained\_mixed$ tends to improved test performances on replayed scenario but slightly degrades test performances with DIDM workers. It appears that extending the scenario

database with different environment dynamic during training enables to control the trade off between high test performances in presence of replayed agents and high test performance in presence of interactive agents.

4.2.2.2 Diversity of traffic agents

In this section, we investigate how the diversity of environment dynamics influences the test performances of our driving policy. Instead of just using one category of rule based agents namely DIDM agents, we also propose to use CIDM workers introduced in chap.2 which obeys a centralized strategy to handle intersections. CIDM agents take their decisions at an intersection according to a centralized strategy that defines rule of priorities which enables to reduce the rate of collisions as shown by the results in sec.2.2.3.3. Similarly to previous section, we build a mixed database called $Huge\_R\_basic\_mixed\_DIDM\_CIDM$ from the $Huge\_R\_basic$ database where each scenario is simulated three times : once with exclusively replay agents, once exclusively with CIDM workers, and once exclusively with DIDM agents. As in the previous experiences we use the same CPG hyper-parameters for all trainings: the only difference is the scenario sampling procedure during data collection. In a last experience called $CPG\_pretrained\_mixed\_CIDM\_DIDM\_single$, we also add simulations on each real scenarios where all traffic agents are removed which means that only the actor remains in the scene. On those scenarios, we expect that the policy will move at the optimal speed (40km/h) up to the end of the scenario without stopping which means that the driving task that consists in following the reference path, consistently with the context, is properly understood. In the last column of tab.4.13, we indicate the rate denoted r_{motion_less} of episodes for which the agent spends more than 5% of the time of the episode almost motionless. We consider that the agent should not spend more than 5% motionless to observe a realistic trajectory. Note that the agent can start at low speed or sometime slows down at intersections which explains why we choose the threshold at 5% and not zero. Similarly to the previous section, we observe

	test						
	replay workers		DIDM workers		CIDM workers		single
	CR%	FCR%	CR%	FCR%	CR%	FCR%	$r_{motionless}$
Planner_autoreg_CPG_replayed_pretrained	6.4	4.1	18.4	13.3	12.6	8.1	15
Planner_autoreg_CPG_pretrained_mixed_DIDM	7.4	3.7	7.6	4.1	5.6	4.2	13
Planner_autoreg_CPG_pretrained_mixed_CIDM	7.6	3.8	9.6	5.4	3.2	2.8	12
Planner_autoreg_CPG_pretrained_mixed_DIDM_CIDM	7.3	3.8	8.1	5.3	4.3	3.2	13
Planner_autoreg_CPG_pretrained_mixed_DIDM_CIDM_single	7.3	3.8	7.8	5.3	4.1	3.5	0

Table 4.13: Test performances of driving policies when the environment dynamic get more diverse due to inclusion of multiple traffic agents.

in tab.4.13 that training with interactive agents such as DIDM or CIDM enables to obtain the best test performances when the environment dynamic includes respectively DIDM or CIDM agent but the test performance with replay worker is not improved. When interactive workers are mixed as in experience $CPG\_pretrained\_mixed\_DIDM\_CIDM$ we note that a better trade off on test performances is reached: we gain more improvements in collision rate than we loose in total over all categories of traffic agents. One surprising result is that on some

simple single agent scenario, the policy sometime tends to suddenly decrease its speed as if it was seeing some neighbors coming as depicted on fig.4.7. We observed that simply adding in the training database, scenarios where the actor is alone is sufficient to remove those undesired behaviours as shown by the last column of tab.4.13.

Figure 4.7: Situations where we notice abrupt stopping of the policy whereas the policy is alone: the image on the left represents the starting position and the image on the right represents the position at which the agent stays 5 seconds later(ds<5km/h).

4.3 Conclusion

In this chapter, we explained how we can learn basic traffic rules to the driving policy with reinforcement learning. We first detailed how we designed our synthetic driving reward before studying how to configure actor-critic policy gradient algorithms for stable training on hundreds of driving scenarios. In a second time, we explained how to improve test performances of the driving policy on new scenarios unseen during training. Decoupling the training of the policy and the value function while sharing a common backbone revealed critical and we also showed that synthetic interactions generated with rule based agents considerably improves test performances robustness to environment variations.

Chapter 5

Learning to drive with demonstrations

Contents

Figures

Tables

In this chapter, we explain how we can leverrage human driving demonstrations to learn a realistic behaviour that are able to generalize safe driving strategies in new situations. In section 5.1, we first show how to imitate humane drivers on real driving scenarios based on real demonstrations before investigating to which extent we can generalize and interpret the learned driving behaviours on new scenarios in sec.5.2. In a last section 5.3, we propose new methods to combine human demonstration and domain knowledge to acquire more robust strategies in new situations.

5.1 Learning to imitate human drivers

Massive amount of driving data are available nowadays and it is possible to reconstruct expert trajectories based on the driving scene context for relatively long period. Those offline data coming from real world could be used to teach a policy how to drive with human style and preferences which is particularly interesting for practical applications in traffic simulation. We first review which existing methods better fit our desiderata in 5.1.1 before applying one of the most promising for our use case in sec.5.1.2.

5.1.1 Inverse decision modelling

Understanding and generalizing how human drivers takes decision based on an ego-centric observation is very challenging because agents are often partially rational due to biological, psychological, and computational factors [76]. Obtaining a transparent understanding of existing behaviors is impossible. We first give preliminary definitions in 5.1.1.1, before reviewing few approaches in secs.5.1.1.2.5.1.1.3,5.1.1.4,5.1.1.5,5.1.1.6, that deals with this problem and that share common basis that we will highlight. Based on this study, we will motivate our method for learning from human demonstrations.

5.1.1.1 Preliminaries

We introduce fundamental notions that we will used later in this chapter. We consider an infinite horizon discounted Markov decision process to model our problem $\mathcal{M}$ defined by the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, p, r, \rho_0, \gamma)$ consisting of a continuous state space $\mathcal{A}$, a continuous action space $\mathcal{A}$ and the transition function $p(s'|s, a)$. The agent makes decision through a stochastic policy π and receives a reward from a reward function $r : \mathcal{S} \times \mathcal{A} \to R$ discounted at every step by the discount factor $\gamma \in [0, 1]$. The agent is spawned according to a starting state distribution ρ_0 on $\mathcal{S}$ satisfying $\forall s \in \mathcal{S} \rho_0(s) > 0$.
The stochastic policy $\pi \in \Pi$ induces a Markov chain $\mathcal{M}_\pi = (S_0, A_0, S_1, A_1, ...)$ which can be built by first taking a random starting state $s \sim \rho_0$, then choosing an action $a \sim \pi(.|s)$ and then either restart the chain with probability $1 - \gamma$ or choose the next state $s' \sim p(.|s, a)$ otherwise.

The Markov chain $\mathcal{M}_\pi$ of π induce a stationnary distributions called an occupancy measure:
Definition: Occupancy measures The Markov chain $\mathcal{M}_\pi$ induced by π has a stationnary distribution(that does not change with time):

1. State OM : $\rho_\pi(s) = \sum_{t=0}^{\infty} \gamma^t.P(S_t = s|\pi, p)$ whose normalized (discounted) version is denoted $\overline{\rho}_\pi(s) = (1-\gamma)\rho_\pi(s)$
2. State action OM : $\rho_\pi(s,a) = \pi(a|s)\sum_{t=0}^{\infty} \gamma^t.P(S_t = s|\pi, p)$ whose normalized version is denoted $\overline{\rho}_\pi(s,a) = (1-\gamma)\rho_\pi(s,a)$
3. State transition OM $\rho_\pi(s,s') = \int_{a\in\mathcal{A}} \rho_\pi(s,a).p(s'|s,a)da$
4. joint OM $\rho_\pi(s,a,s') = \rho_\pi(s,a).p(s'|s,a)$

Intuitively, discounted stationary state (state-action) distribution measures the overall "frequency" of visiting a state (state-action).
Theorem:(Syed [184] and [69]) In a single environment, the occupancy measure $\rho_\pi(s,a)$ satisfies the Bellman flow constraint :

$$\forall(s,a,s') \in \mathcal{S}\times\mathcal{A}\times\mathcal{S}\ \rho_\pi(s,a) = \rho_0(s)\pi(a|s) + \gamma.\int_{(s',a')\in\mathcal{S}\times\mathcal{A}} \pi(a|s).p(s|a',s').\rho_\pi(s',a') \quad (5.1)$$

and the policy π whose occupancy measure is ρ_π is unique. That is, the occupancy measure and the policy are in an one-to-one relationship. The mapping $\rho_\pi \mapsto \pi$ defined by $\pi(a|s) := \frac{\rho_\pi(s,a)}{\int_{a'\in\mathcal{A}}\rho_\pi(s',a)})$ is a bijection between Π and the set of measures on $\mathcal{S}\times\mathcal{A}$ satisfying the Bellmann flow constraint.
Note that the expected cumulative reward of $\mathcal{M}_\pi$, i.e. the expected sum of rewards $r(s_t,a_t)$ up to the first restart of the chain, is given by

$$\mathbb{E}[\sum_{t=0}^{\infty}\gamma^t.r(s_t,a_t)] = \frac{\mathbb{E}_{\overline{\rho_\pi}}[r(s,a)]}{(1-\gamma)} = \int_{(s,a)\in\mathcal{S}\times\mathcal{A}} \rho_\pi(s,a).r(s,a) = \langle r,\rho_\pi\rangle \quad (5.2)$$

5.1.1.2 Imitation Learning

Imitation learning is the most straightforward way to learn directly from demonstrations because it removes the burden of a reward function which is often not compatible or unknown. The simplest approach to imitation learning is Behavior Cloning [7] where an agent policy directly regresses on expert actions (but not states) using supervised learning.

$$argmin_\pi \mathbb{E}_{\rho^{exp}(s)}[KL(\pi^{exp}(a|s)||\pi^\pi(a|s)] = argmax_\pi \mathbb{E}_{\rho^{exp}(s,a)}[log(\pi^\pi(a|s)] \quad (5.3)$$

However IL suffers from a fundamental problem called distributional shift[176]. Unlike supervised learning, training and testing states distribution are drawn from different distributions,

induced by the expert and the learned policies respectively. At test time policy actions influence the distributions of the next states so an imitated policy, even with a small training error, may visit a state out of the expert demonstrations, which causes a larger decision error and a transition to further unseen states. Consequently, the policy value gap accumulates along with the planning horizon. It was shown in [207] that given an expert policy π_E and an imitated policy π such that $\mathbb{E}_{\rho^{exp}(s)}[KL(\pi^{exp}(a|s)||\pi^{\pi}(a|s)] \leq \epsilon$,we have for an infinite horizon MDP, a value gap such that :

$$V^{\pi^{exp}} - V^{\pi} \leq \frac{2\sqrt{2}, R_{max}}{(1-\gamma)^2} . \sqrt{\epsilon} \tag{5.4}$$

Therefore, the training objective of imitating expert behavior is not perfectly aligned with the true objective of performing the task correctly. Meanwhile, the policy value gap is only linear w.r.t. the horizon for a concurrent method called GAIL and introduced in sec.5.1.1.6 such that $D_f(\rho^{exp}(a|s)||\rho^{\pi}(a|s)) < \epsilon$, is only $\mathcal{O}(\frac{\sqrt{\epsilon}}{(1-\gamma)})$. Numerous attempts to apply supervised learning methods for learning driving policies reveals that generalization is still insufficient and compounding errors are still a major concern[8, 30, 43]. Disagreement regularized BC[16] operates by training an ensemble of policies on the expert demonstration data, and uses the variance of their predictions as a cost minimized with RL together with a supervised behavioral cloning cost. However prediction uncertainty is not guaranteed to help how to recover from critical situations or how to adapt to new situations : it can just help to avoid drifting from the support of the expert state occupancy measure in some situations. To overcome the distributional shift, methods such as DAgger [161] and Dart [101] assume an interactive access to an expert policy to complement expert demonstrations on new situations . At training iteration n they leverage all the n sets of expert trajectories interventions as well as the initial expert demonstration represented by $\rho^{agg(1:n)}(s) = \frac{1}{n}\sum_{i=0}^{n} \rho^{\pi^{(i)}}(s)$ to optimize the learner policy

$$argmin_{\pi}\mathbb{E}_{\rho^{agg(1:n)}(s)}[KL(\rho^{exp}(a|s)||\rho^{\pi}(a|s)] \tag{5.5}$$

The same principle was applied for learning driving policies with the support of a coach. Roach [225] uses and advisor policy to complement the trajectories of a policy when it fails due to some specific events. Once the event occur, an advisor policy generates a corrective trajectory few steps before the event and the policies is trained with an additional objective to match the advices $\mathbb{E}_{\rho^{advisor}(s)}[KL(\pi^{advisor}(a|s)||\pi^{\pi}(a|s)]$. However this solution is often impractical because it requires the access of an interactive expert which is often human. Additionally expert are not always able to recover a safe strategy from arbitrary state reached by the policy when evaluated in closed loop.

5.1.1.3 Inverse reinforcement learning

Instead of just imitating expert actions, another approach consists in making expert actions optimal under a learned reward function. Given a set of demonstrations from an expert policy

π_e, IRL[133] is the problem of seeking a reward function from which we can recover π_E through RL. Inverse RL interprets the expert as the optimal policy under some unknown reward function. IRL consists in learning a reward function $r \in \mathcal{R}$ or equivalently a cost function $c \in \mathcal{C}$ where $(c = -r)$ from expert demonstrations as an intermediate step before applying Rl on top of it. However, IRL is an ill-defined problem since (1) any constant reward function on expert support 5.1.1.1 can rationalize every expert and (2) multiple rewards meet the criteria of being a solution [133]. Maximum entropy IRL (MaxEntIRL) [230] is capable of solving the first issue by seeking a reward function that maximizes expert's return along with Shannon entropy of expert policy.

$$IRL(\pi_E) = argmax_{r \in \mathbb{R}^{\mathcal{S} \times \mathcal{A}}} min_{\pi \in \Pi} \mathbb{E}_{\pi_E}[r(s,a)] - \mathbb{E}_{\pi}[r(s,a)] - H(\pi)) \tag{5.6}$$

where $\mathcal{H}(\pi)$ denotes the γ-discounted causal entropy of the policy $\mathcal{H}(\pi) = \mathbb{E}_{(s,a) \sim \rho_\pi}[\frac{-log(\pi(a|s)}{(1-\gamma)}]$. Intuitively, MCE-IRL seeks for a reward function that assigns high rewards to expert demonstrations and low rewards to all the others, in favor of high entropy policies. The corresponding RL problem follows to derive a policy from this reward function which is embedded in the inner loop of IRL.

$$RL(c) = argmin_{\pi} - H(\pi) + -\mathbb{E}_{\pi}[r(s,a)] \tag{5.7}$$

Solving an IRL problem involves repeatedly solving the optimal reward for a given policy which makes IRL algorithms prohibitive to learn policies for large MDPs and high dimensional state spaces.

5.1.1.4 Offline reinforcement learning

While imitation learning exploits expert demonstrations in the form of state action pair it also possible to exploit offline data from RL like transitions (s_t, a_t, r_t, s_{t+1}) with Offline Reinforcement Learning [105]. Offline reinforcement learning (RL) algorithms as BCQ[47] or CQL [99], seek to learn an optimal policy without active data collection from a fixed dataset $\mathcal{D}$ composed of (s, a, r, s') generated by a behaviour policy, interacting with the environment. One strong requirement of vanilla offline RL to obtain reasonable performances is uniform coverage of state action space. From a practical standpoint, expert datasets have often a very restricted support which can harm performances of offline algorithms [157]. More problematically, offline RL assumes an access to a reward function which is usually unknown at least for autonomous driving since we ignore what utility human drivers are maximizing. Some works dealing with autonomous driving on highways [124] introduced a custom reward that will be assigned to transitions collected with the behavioural policy but consistent rewards for scenarios with complex road-networks are more difficult to design and more importantly may not lead to realistic driving behaviours [34].

5.1.1.5 Apprenticeship learning

In Apprenticeship Learning [1], rather than learning a cost, its goal is to find a policy π whose performance is close to that of the expert for any possible cost in a known set $\mathcal{R}$.

$$AL(\pi_e) = argmin_{\pi \in \Pi} - H(\pi) + sup_{r \in \mathcal{R}}[\mathbb{E}_{\rho_{\pi_e}}[r(s,a)] - \mathbb{E}_{\rho_\pi}[r(s,a)]] \tag{5.8}$$

This keeps the state-action occupancy measures of the agent and the expert in proximity, requiring the AL agent to find a path back to the expert trajectories in states that are unobserved by the expert. This differs from BC, in which the agent's policy is undetermined in these unobserved states.
Previous approaches enforced some restrictions on the set $\mathcal{R}$ which often relies on linear combination of basis function [184, 1]. Unfortunately for high dimensional state spaces those feature function are often unavailable and need to be learned. To avoid this issue, [204] has shown that apprenticeship learning can be reformulated as minimizing an Integral Probability Metrics. Given two probability measures, p and q defined on a measurable space, S, the integral probability metric (IPM) is defined as:

$$\gamma_{\mathcal{F}}(p,q) = sup_{f \in \mathcal{F}} | \int_{\mathcal{S}} f dp - \int_{\mathcal{S}} f dq | \tag{5.9}$$

where $\mathcal{F}$ is a class of real-valued bounded measurable functions on $\mathcal{S}$. For a class $\mathcal{R}$ of symmetric reward, the IPM is defined by

$$\gamma_{\mathcal{F}}(\rho_{\pi_e}, \rho_\pi) = sup_{r \in \mathcal{R}}[\mathbb{E}_{\rho_{\pi_e}}[r(s,a)] - \mathbb{E}_{\rho_\pi}[r(s,a)]] \tag{5.10}$$

By appropriately choosing $\mathcal{R}$, various popular distance metric between ρ_{π_e} and ρ_π can be obtained. To make our choice on $\gamma_{\mathcal{F}}(\rho_{\pi_e}, \rho_\pi)$ we draw a connection with the field of optimal transport [146] that considers the question of how to transport one distribution to another while minimizing the amount of effort expended. The Wasserstein-p distance between two distributions and ν on metric space (M, d) estimates the amount of work that needs to be done to convert one probability distribution to the other, as measured by the ground metric d.

$$W_d^p(\mu,\nu) = inf_{\xi \in \Pi(\mu,\nu)} \mathbb{E}_{x,y \sim \xi}[d(x,y)^p]^{\frac{1}{p}} \tag{5.11}$$

where $\Pi(\mu,\nu)$ is the space of joint distributions $\triangle(\mathcal{M} \times \mathcal{M})$ whose marginals are μ and ν respectively. If we restrict our attention to the Wasserstein-1 metric, the Kantorovich-Rubinstein duality allows us to express the Wasserstein-1 distance as follows :

$$W(\mu,\nu) = sup_{\phi \in \mathcal{L}_1} \mathbb{E}_{x \sim \mu}[\phi(x)] - \mathbb{E}_{x \sim \nu}[\phi(x)] \tag{5.12}$$

where, $\mathcal{L}_1$is the set of all 1-Lipschitz functions from M to $\mathbb{R}$ such that:

$$f \in \mathcal{L}_1 \Leftrightarrow sup_{(x,y)\in\mathcal{M}\times\mathcal{M}} \frac{|f(x) - f(y)|}{d(x,y)} \leq 1 \tag{5.13}$$

It was shown that the reward function $r(s,a)$ can be treated as a Kantorovich potential in the dual form of optimal transport (OT) problem [204] when the set of reward $\mathcal{R}$ is composed of $1-$Lipshitz function. [219, 204, 23, 172] proposed Wasserstein Adversarial Imitation Learning (WAIL) which considers the IPM $\gamma_{\mathcal{F}}(\rho_{\pi_e}, \rho_\pi) = W_1^d(\rho_\pi, \rho_{\pi_e})$ on a metric space $(\mathcal{S}\times\mathcal{A}, d)$ and solve the problem:

$$argmin_\pi - \mathcal{H}(\pi) + W_1^d(\rho_\pi, \rho_{\pi_e}) \tag{5.14}$$

$$argmin_\pi - \mathcal{H}(\pi) + sup_{\phi\in\mathcal{L}_1} \mathbb{E}_{(s,a)\sim\rho_{\pi_e}}[\phi(s,a)] - \mathbb{E}_{(s,a)\sim\rho_\pi}[\phi(s,a)] \tag{5.15}$$

Note that depending on the distance metric d on $(\mathcal{S}\times\mathcal{A}, d)$ enforcing the constrain $\phi \in Lip(1)$ may get more difficult since $sup_{((s,a),(s',a'))\in(\mathcal{S}\times\mathcal{A})\times(\mathcal{S}\times\mathcal{A})} \frac{|f((s,a)) - f((s',a'))|}{d((s,a),(s',a'))} \leq 1$. In our case $\mathcal{S}\times\mathcal{A}$ reduce to a subset of $\mathbb{R}^m$ so we selected the appropriate norm to enforce $\phi \in Lip(1)$ based on the gradient penalty [57] or spectral normalization [125]. The outer problem for recovering the policy requires to apply RL with the current reward. The choice of the reward function form was studied in [204, 219], but we choose to consider the form proposed in [114] based on a state distribution matching problem instead of the commonly state action distribution matching problem. The original problem can easily be reformulated based on the state transition occupancy measure $\rho_\pi(s,s') = \int_{a\in\mathcal{A}} \rho_\pi(s,a).p(s'|s,a)da$ 5.1.1.1. At every time step, [114] proposed to provide the following reward:

$$r(s_t, s_{t+1}) = \frac{1}{T}[\phi(s_t, s_{t+1}) - \mathbb{E}_{(s,s')\sim\tau_e}[\phi(s,s')] \tag{5.16}$$

In case the MDP is episodic of horizon T we can interpret the reward expression from the policy return expression:

$$J(\pi) = \sum_{t=1}^{T} \mathbb{E}_{s_t,s_{t+1}\sim\pi}[r(s_t,s_{t+1})] = \sum_{t=1}^{T} \frac{\mathbb{E}_{(s_t,s_{t+1})}[\phi(s_t,s_{t+1}) - \mathbb{E}_{(s,s')\sim\tau_e}[\phi(s,s')]]}{T} = -W_1^d(\rho_\pi, \rho_{\pi_e}) \tag{5.17}$$

So optimizing the policy gradient consists in minimizing the 1-Wassertein distance between the state distributions of the expert ρ_e and the one of the learner ρ_π. We will study the efficiency of those methods for our use case in sec.5.2.1.2

5.1.1.6 Imitation learning as divergence minimization

It was shown that imitation learning and inverse reinforcement are mutually connected and can be reformulated as a divergence minimization problem [44, 50] which reveals better to compensate distributional shift than BC and also more sample efficient than IRL. The starting point is

certainly the algorithm known as Generative Adversarial Imitation Learning (GAIL) [69] which proved that applying RL to a cost function learned through IRL with a cost regularizer ψ l is equivalent to

$$IRL(\pi_E) = argmax_{r\in\mathbb{R}^{\mathcal{S}\times\mathcal{A}}} - \psi(c) + (min_{\pi\in\Pi} - \mathbb{E}_{\rho_\pi}[r(s,a)] - H(\pi)) + \mathbb{E}_{\rho_{\pi_e}}[r(s,a)] \tag{5.18}$$

$$RL \circ (IRL_\psi(\pi_E)) = argmin_\pi - H(\pi) + \psi^*(\rho_\pi - \rho_{\pi_E}) \tag{5.19}$$

where ψ^* is the convex conjugate of the regularizer ψ that measures a discrepancy between the induced occupancy measures of the expert and the learned policy. By choosing a specific regularizer ψ, the problem can be reformulated into

$$\pi^* = argmin_\pi \mathbb{D}_{JS}(\overline{\rho}_{\pi_e}, \overline{\rho}_\pi) - \lambda\mathcal{H}(\pi) \tag{5.20}$$

$$argmin_\pi D_{KL}(\rho_\pi|\frac{\rho_{\pi_e}+\rho_\pi}{2}) + D_{KL}(\rho_{\pi_e}|\frac{\rho_{\pi_e}+\rho_\pi}{2}) - \lambda\mathcal{H}(\pi) \tag{5.21}$$

$$=argmin_\pi \mathbb{E}_{\rho_{\pi_e}}[log(\frac{\rho_{\pi_e}}{\rho_{\pi_e}+\rho_\pi})] + \mathbb{E}_{\rho_\pi}[log(\frac{\rho_\pi}{\rho_{\pi_e}+\rho_\pi})] - \lambda\mathcal{H}(\pi) \tag{5.22}$$

$$=argmin_\pi max_D \mathbb{E}_{\rho_{\pi_e}}[log(D(s,a))] + \mathbb{E}_{\rho_\pi}[log(1-D(s,a))] - \lambda\mathcal{H}(\pi) \tag{5.23}$$

We can change the normalized occupancy measure to the normalized version because the constant normalizer is irrelevant in minimization. Concerning the introduction of the discriminator $D(s,a) : S \times A \to (0,1)$ (binary classifier), it is valid because at optimally the discriminator value is given by

$$D^*(s,a) = \frac{\rho_{\pi_e}(s,a)}{\rho_{\pi_e}(s,a) + \rho_\pi(s,a)} \text{ [55]} \tag{5.24}$$

This principle can be extended to a broader class of measure discrepancy called f-divergences. Let P, Q be two distributions with density functions p, q. For any convex, lower-semi continuous function $f : \mathbb{R}^+ \to \mathbb{R}$ satisfying $f(1) = 0$, the f-divergence is defined as:

$$D_f(P||Q) = \int_{\mathcal{X}} q(x) f(\frac{p(x)}{q(x)}) dx \tag{5.25}$$

Different choices of f enables to recover popular divergences KL,RKL,Pearson,Total variation, Jeffrey,Jensen-Shannon as summarized in tab5.1.1.6[136] but this formulation does not allow direct optimization because we do not have access to the densities.[134] leveraged a variational representation of f in the definition of the f-divergence to obtain a lower bound on the divergence. We shortly provide the derivation for clarity. Every convex, lower-semi continuous function f has a convex conjugate function $f^*(t) = sup_{u\in dom(f)}\{u.t - f(u)\}$ and it can be shown that function f can be expressed as follows $f(u) = sup_{t\in dom(f^*)}\{t.u - f^*(u)\}$. Therefore, we can rewrite the f divergence as follows:

$$D_f(P||Q) = \int_{\mathcal{X}} q(x).sup_{t\in dom(f^*)}\{t.\frac{p(x)}{q(x)} - f^*(t)\}dx \tag{5.26}$$

Since f is convex, the Jensen inequality allows to swap integration and supremum:

$$D_f(P||Q) \geq sup_{t \in dom(f^*)} \int_{\mathcal{X}} q(x).\{t.\frac{p(x)}{q(x)} - f^*(t)\}dx \tag{5.27}$$

We now introduce a class of functions $\mathcal{T}_\omega \in \mathcal{T}$ with $T_\omega : X \to \mathbb{R}$ to compute the supremum.

$$D_f(P||Q) \geq sup_{T \in \mathcal{T}}(\int_{\mathcal{X}} p(x).T(x)dx - \int_{\mathcal{X}} q(x).f^*(T(x))dx \tag{5.28}$$

$$D_f(P||Q) \geq sup_{T_w \in \mathcal{T}}(\mathbb{E}_{x\sim P}[T_w(x)] - \mathbb{E}_{x\sim Q}[f^*(T_w(x))]) \tag{5.29}$$

Since we want to follow the generative-adversarial approach [55] for learning a generative model Q_θ of the true distribution P we need this lower bound to be tight. It was shown in [134] that a tight bound can be obtained for

$$T^*(x) = f'(\frac{p(x)}{q(x)}) \tag{5.30}$$

In that special case, we can learn the generative model Q_θ by finding a saddle-point of the following f-GAN objective function:

$$\mathcal{L}(\theta, w) = \mathbb{E}_{x\sim P}[T_w(w)] - \mathbb{E}_{x\sim Q}[f^*(T_w(x))]. \tag{5.31}$$

The associate optimization problem expresses as

$$min_\theta max_{T_w} \mathbb{E}_{x\sim P}[T_w(x)] - \mathbb{E}_{x\sim Q_\theta}[f^*(T_w(x))] \tag{5.32}$$

Note that we recover the same optimization problem with the approach of [69] apart from the entropy regularization term if we take a cost regularizer such that $\psi_f(c) = \mathbb{E}_{\rho^{exp}(s,a)}[f^*(c(s,a)) - c(s,a)]$ as shown in [50]. To optimize on a given finite training data set, we approximate the expectations using mini-batch samples and we alternate between training the generator and the discriminator as detailed in alg.16

f	f^*	g_f	$D_f(\rho^{exp}(s,a)\|\|\rho^\pi(s,a))$	$f'(1)$	$dom(f*)$	$T_w^{\pi,optimal}(s,a)$	f-max
$-log(u)$	$-1-log(-u)$	$-e^{-u}$	$RKL(\rho^\pi(s,a)\|\|\rho^{exp}(s,a))$	-1	$\mathbb{R}_-$	$-\frac{\rho^\pi(s,a)}{\rho^{exp}(s,a)}$	*AIRL*
$u.log(u)$	e^{u-1}	u	$KL(\rho^{exp}(s,a)\|\|\rho^\pi(s,a))$	1	$\mathbb{R}$	$1+log(\frac{\rho^{exp}(s,a)}{\rho^\pi(s,a}$	FAIRL
$u.log(u)-(1+u).log(\frac{1+u}{2})$	$-log(2-e^u)$	$log(2)-log(1+e^{-u})$	$D_{JS}(\rho^{exp}(s,a)\|\|\rho^\pi(s,a)) - \lambda\mathcal{H}^{causal}(\pi)$	0	$t < log(2)$	$log(\frac{2.\rho^\pi(s,a)}{\rho^{exp}(s,a)+\rho^\pi(s,a)})$	GAIL
$\frac{1}{2}\|u-1\|$	u	$\frac{1}{2}.tanh(u)$	$D_{TV}(\rho^{exp}(s,a)\|\|\rho^\pi(s,a))$	1	$\mathbb{R}$	$\frac{1}{2}.sign(\frac{\rho^{exp}(s,a)}{\rho^\pi(s,a)}-1)$	Dagger

To solve this problem for different f-divergences, we also need to respect the domain $dom(f^*)$ for T_w . To this end, we express $T_w(x) = g_f(V_w(x))$ where $V_w : \mathcal{X} \to \mathbb{R}$ is an arbitary function and $g_f : \mathbb{R} \to dom(f^*)$ is an activation function that maps to the domain of f^*. In order to obtain a discriminator with T_w, we enforce g_f to be a monotone increasing function so that a large output $V_w(x)$ corresponds to the belief that the sample x comes from the data distribution P. It appears that the variationnal function $T_w(x)$can be seen as a classifier whose threshold

Algorithm 16 f-GAN training iteration

[1]

sample $X_p = [x_1, ..., x_B]$ from P

sample $X_Q = [x_1^{'}, ..., x_B^{'}]$ from Q_{θ_k}

Update $w_{k+1} = w^t + \eta.\nabla_w \mathcal{L}(\theta_k, w_k)$

Update $\theta_{k+1} = \theta_k - \eta.\nabla_\theta \mathbb{E}_{x\sim Q}[f^*(T_{w_k}(x))]$

is given by $f'(1)$ since $T^*(x) = f'(\frac{p(x)}{q(x)})$. The f-GAN theory was extended to imitation learning with a unified algorithm called $f - max$ [50, 85]. In the context of imitation learning we aim to match the state action occupancy measure of the learner $\rho^\pi(s, a)$ with the state action occupancy measure of the expert $\rho^{exp}(s, a)$.

$$max_{T_w} \mathbb{E}_{(s,a)\sim\rho^{exp}(s,a)}[T_w((s, a))] - \mathbb{E}_{(s,a)\sim\rho^{\pi_\theta}(s,a)}[f^*(T_w((s, a))] \tag{5.33}$$

$$max_{\pi_\theta} \mathbb{E}_{\tau\sim\pi}[\sum_t \gamma^t f^*(T_w(s_t, a_t))] \tag{5.34}$$

The divergence choice matters and leads to different learner policies. Let $p(x)$ be the true distribution density, and $q(x)$ be the approximate distribution learned by minimizing its divergence from $p(x)$. For the KL divergence $D_{KL}(P||Q) = \int_{\mathcal{X}} p(x).log(\frac{p(x)}{q(x)})dx$ we observe that when $p(x) = 0$ then the discrepancy between $q(x) > 0$ and $p(x)$ will be ignored. On the other hand, for the Reverve-KL(RKL) divergence $D_{RKL}(P||Q) = \int_{\mathcal{X}} q(x).log(\frac{q(x)}{p(x)})dx$, when the learner output $q(x) = 0$ it does not capture the discrepancy between $q(x)$ and $p(x) > 0$. Therefore, the KL divergence can be used to better learn multiple modes from a true distribution $p(x)$ (mode-covering), while RKL divergence will perform better in learning a single mode (mode-seeking). GAIL uses the Jensen-Shannon divergence that only discourages the policy from placing more mass than the expert has.

This unified perspective provides some intuition why AIL outperforms BC in the low data regime. Behaviour cloning's objective $\mathbb{E}_{\rho^{exp}(s)}[KL(\rho^{exp}(a|s)||\rho^\pi(a|s)] = -\mathbb{E}_{\rho^{exp}(s,a)}[log(\rho^\pi(a|s)]$ is optimized to match the conditional distribution $\pi(a|s)$, whereas the policy learnt with $f - MAX$ algorithms is explicitly encouraged to match the marginal state distributions as well. Choosing the right divergence for recovering accurately the expert policy is crucial especially because the policy is optimized based on trajectories and not on a single step as in GAN[55] which is much harder. It is even possible to learn an admissible f- divergence by learning a function f with the discriminator that satisfies the convexity constrain and $f(1) = 0$ as proposed in $f - GAIL$ [223]. However this technique does not explain how to shape function f such that policy can better learn the expert policy which is the main concern of AIL algorithms.

The deficiencies of BC suggests that simply trying to match the state marginal distributions $D_f(\rho^{exp}(s)||\rho^\pi(s))$, rather than the state conditioned action distributions could be sufficient to learn an expert policy and even a reward[46]. Recovering expert state marginal distribution

while learning a stationary reward was proposed in f-IRL algorithm [135] however they compute the gradients of the f-divergence based on an objective that requires computing the covariance over agent trajectory distribution $\rho(\tau)$ which can be extremely brittle in high dimensional state space.

5.1.2 Adversarial imitation learning

In this section we explain how we scale up the Adversarial Imitation Learning approach for learning a human-like driving policy on numerous real driving scenarios. We first detail how we designed the neural network architecture of the discriminator-actor-critic in sec.5.1.2.1. Secondly we explain how we set-up the whole pipeline in sec.5.1.2.2 and we provide quantitative results on imitation performances on real driving scenarios. Thereafter, we explain how to consistently improve long term imitation performance thanks to horizon curriculum in sec.5.1.2.3 before studying how to balance policy and discriminator trainings to guarantee stable improvements in sec.5.1.2.4. In a last section.5.1.2.5, we analyse which reward formulation ends up in the best performances and how to properly end an episode that turns unrealistic.

5.1.2.1 Discriminator actor critic architecture

AIL algorithms requires a discriminator in addition to the actor critic network used to train the policy. Since policy and discriminator compete in a two player game[55], they need to share the same observation backbone network such that the policy can exploit similar features than the discriminator and then fool it. To this end, we implemented on one side an actor critic architecture that uses the $FCBasline_basic$ observation backbone network and on the other side a discriminator that use an instance of the backbone. We choose the $FCBasline_basic$ backbone because it is a light architecture that enables fast training with reasonable open loop performances reported in chap.3. The discriminator is supposed to distinguish state-action pairs between expert and learner but as the action representation is considerably smaller than the observation we also propose to feed the discriminator with the states transitions (s_t, s_{t+1}). Optimal discriminator can still be expressed in terms of occupancy measure:[1]

$$D^*(s,a) = \frac{\rho_{\pi_e}(s,a)}{\rho_{\pi_e}(s,a) + \rho_{\pi}(s,a)} \tag{5.35}$$

$$D^*(s,s') = \frac{\rho_{\pi_e}(s,s')}{\rho_{\pi_e}(s,s') + \rho_{\pi}(s,s')} \tag{5.36}$$

We propose two implementations depicted on fig.5.1: the first one represented on the left of fig.5.1 encodes the observation action pair (o_t, a_t) with our observation backbone and an additional action encoder implemented with a simple MLP. Observation and action embedding

[1]Since real scenarios are replayed: the transition dynamic can be considered as deterministic so the optimal discriminator for state transitions can directly be expressed with $\rho_e(s,s')$ and $\rho_\pi(s,s')$

(z_t^D, z_t^a) are then aggregated with a simple concatenation and the discriminator label is obtained after applying the discriminator head on $z_t^D \odot z_t^a$. The main limitation of this simple architecture is related to the fact that an action represented as $a_t = (ds_t, n_t)$ stores less information about the transition than the next observation and the discriminator may just ignore information encoded with the action. Consequently, we propose a second architecture for the discriminator that process the transition based on consecutive observation pairs (o_t, o_{t+1}). We first encode the two observation with the discriminator observation backbone to obtain (z_t^D, z_{t+1}^D) and then aggregate them by substraction $\Delta z_t^D = z_{t+1}^D - z_t^D$ before feeding the discriminator head. We expect that rewards provided by this discriminator will provide better guidance for the policy than the other architecture that may rather focus on how much the current state belong to the expert state distribution which does not provide a strong feedback on how to modify the probability of taking action a_t given observation o_t on the policy side. We will analyse the impact of the different discriminator architectures in sec.5.1.2.5 where we also study the influence of the reward function.

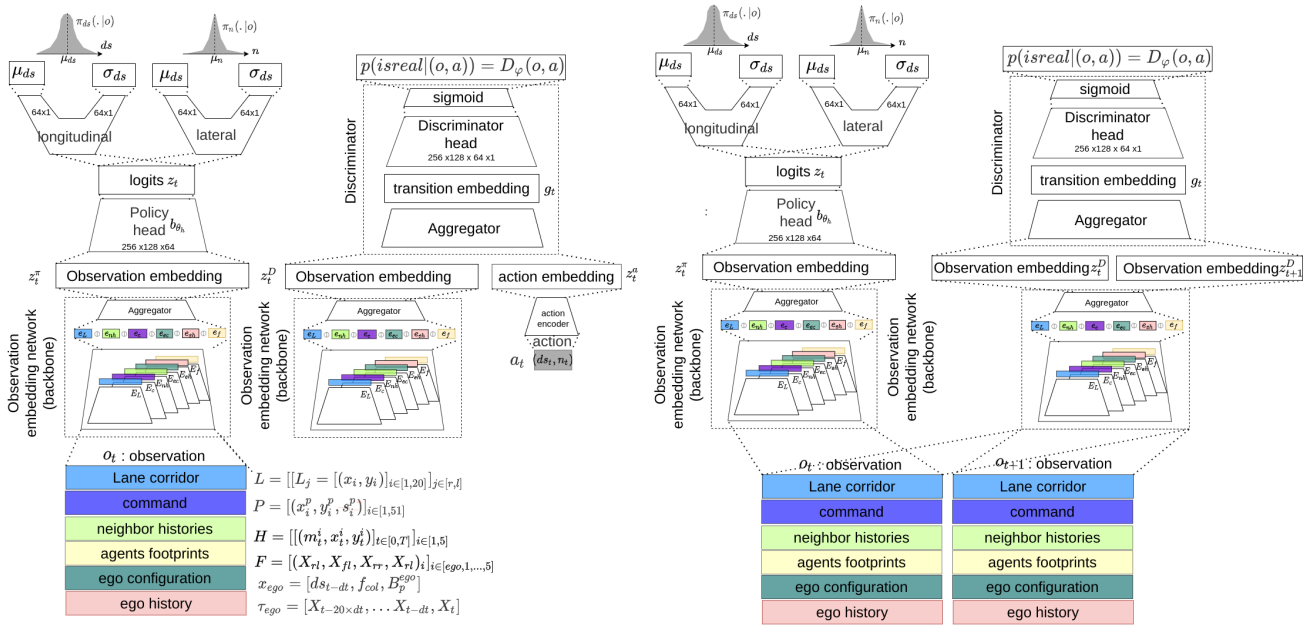

Figure 5.1: Discriminator-Actor-Critic network architectures

5.1.2.2 Large scale training on real driving scenarios

In contrast to most previous applications of AIL algorithms which aimed to imitating synthetic demonstrations generated by RL experts [93] or a very restricted number of real driving demonstrations mainly on highways [98, 14, 10, 71] we want to leverage huge amount of real and highly interactive driving scenarios to learn realistic driving policies. Another main differences with respect to applications in robotic manipulation [200] or character animation [144] is related to the nature of the simulation environment which is multi agent in our case. To better meet the needs of our application we designed our own training framework based on

Rllib reinforcement learning library [109]. The whole framework was specifically designed to handle large scale traffic simulations with thousands of driving scenarios on various maps. In the following, we detail the generic implementation of the AIL training procedure which is illustrated in fig.5.2. AIL algorithm described in 5.1.1.6, consist in repeating a main training operation until the driving performances get satisfying. To maintain stable improvements of the policy we will discuss more in depth several techniques in sec. 5.1.2.4 and more specifically horizon curriculum in sec.5.1.2.3.

At training initialization, the pipeline is fed with an expert database $\mathcal{D}^e$ that is composed of a set of driving scenarios each endowed with a reference expert trajectory τ_e. The database is decomposed in a training set $\mathcal{D}^e_{train}$ and a validation set $\mathcal{D}^e_{valdiation}$ composed of respectively N^s_{train} and N^s_{val} scenarios. The training operation starts with data collection that generates a training batch Γ composed of n_Γ transition (o_t, a_t, o_{t+1}) samples collected by the most recent driving policy $\pi_{\theta_{k-1}}$. This training batch is created by aggregating trajectories provided by N_S instances of simulation that operate in parallel each initialized with a specific list of driving scenarios $[S^{(i)}_j]_{j\in J_i}$. During each episode, a single actor is animated by the same policy π_{θ_k} while others called workers are replayed based on ground truth episodes. As long as the total number of transitions collected is lower than the training batch size each simulator i continues to launch simulation following the list of scenarios $[S^{(i)}_j]_{j\in J_i}$. Depending on the scenario assignment $[[S^{(i)}_j]_{j\in J_i}]_{i\in[1,\dots,N_s]}$ the training batch can contain multiple episodes for the same driving scenario such that advantage estimation can be made more accurate[2]. Usually, we choose a training batch size at least as big as the number of expert transitions stored in the training database ($n_\Gamma = 100K$). Once the training batch is collected, we feed the learner transition buffer denoted $\mathcal{B}_\pi$ with trajectories generated by $\pi_{\theta_{k-1}}$ to train the discriminator $D_{w_{k-1}}$. Note that depending on the size of the learner buffer $|\mathcal{B}_\pi|$ we can store more than a single training batch Γ but to guarantee balanced expert and learner training distributions we enforce $|\mathcal{B}_\pi| = n_\Gamma$. Consequently the expert buffer $\mathcal{B}_e$ is configured such that its size equals the size of the training batch and its content is updated based on the scenario present in the training batch : we load expert demonstrations associated to each learner trajectory in the training batch. At this level the discriminator can be trained for N_D epochs on the expert and learner buffer. The number of epochs the discriminator is trained relies on a stopping criteria detailed in section 5.1.2.4. Intuitively we should avoid saturation of the discriminator but we should guarantee proper guidance for the policy. Once the discriminator is trained, we can compute the reward for each transitions $(o_t, a_t, o_{t+1}, r^{(k)}_t = f(D_{w_k}(o_t, a_t, o_{t+1})))$ collected by the learner π_{θ_k} in the training batch. Since we update the policy with an on-policy actor critic algorithm (PPO[167]), we need to compute the Generalized Advantage Estimator[166] $A^{\pi_{\theta_{k-1}}}_{GAE}(o_t, a_t)$ for each sample $(o_t, a_t, o_{t+1}, r^{(k)}_t)$ of the training batch but since the reward $r^{(k)}_t$) was recently updated by the new discriminator, the value approximator of our policy $V^{\pi_{\theta_{k-1}}}$ should in principle also be updated based on the new rewards. We explain in section5.1.2.4 why it is actually better to reuse the old value function for advantage computation and why

[2]See the VINE version of TRPO [168]

the value function should be trained after the policy. Once the advantage is available, we can train the policy for N_p epochs on the training batch Γ by mini batch gradient ascent. Note that the stability of the training mainly depends on the quality of policy improvements that may sometimes be too conservative or too aggressive as discussed in 5.1.2.4 which can severely harm policy performances on the validation set. Note that off policy methods [93] enables to improve sample efficiency of AIL methods [137] but can quickly turn unstable because the Q function estimation under a changing reward is not guaranteed to provide reliable policy improvements [3] so we chose policy gradient actor critic methods as suggested in the original algorithm [69] and in [56] that studies convergence of AIL. The whole training procedure is repeated for at most N_{train} training iterations with some performance evaluation of the current policy on the test set at a regular frequency. The training is stopped when the test performances do no decrease anymore which means that the policy tends to overffit the training demonstrations. Note that the main limitation of this framework is related to the fact that other agents in the simulation are not really interacting with the learner. As a consequence the learner has either to recover exactly the expert demonstration either to adapt to replay workers trajectories that are fixed. Interacting with replay agent enables to stabilize the global traffic but it is also harder to predict other agents reaction as each replay agent is behaving in a unique way whatever happens on the policy side.

Algorithm 17 AIL Training Procedure

INPUTS:

- $\mathcal{D}^e = (\mathcal{D}_{train}, \mathcal{D}_{eval})$
- n_Γ :traing batch size
- N_D, N_π, N_V :discriminator,policy and value training epochs

$\mathcal{B}_e = \{\}, \mathcal{B}_\pi = \{\}$
for $k = 1, ..., N_{train}$ **do**
  if $k\%n_{evaluation} = 0$ **then**
    E_V =Evaluate(π,$\mathcal{D}^e_{validation}$)
  end if
  $[[S^{(i)}_j]_{j\in J_i}]_{i\in[1,...,N_s]}$ =AssignScenarios($\mathcal{D}^e_{train}$)
  Γ =CollectEpisods($\mathcal{S}^e, \pi_{\theta_{k-1}}$)
  $\mathcal{B}_e, \mathcal{B}_\pi$ =UpdateExpertAndStudentBuffers($\Gamma, \mathcal{S}^e, \mathcal{B}_e, \mathcal{B}_\pi$)
  D_{w_k} ←trainDiscriminator($D_{w_{k-1}}$, $\mathcal{B}_e, \mathcal{B}_\pi$)
  $\Gamma \leftarrow$ UpdateRewardsAndAdvantages(Γ, D_{w_k})
  π_{θ_k} ←trainActor($\pi_{\theta_{k-1}}, \Gamma$)
  $V^{\pi_{\theta_k}}$ ←trainCritic($V^{\pi_{\theta_{k-1}}}, \Gamma$)
  updateSimulationHorizon(Γ)

In the following we provided the best test performances obtained with different AIL algorithms on three different databases extracted from the interaction dataset[214] and each based on a different road-network respectively a roundabout an intersection and a road merging. The

[3]The Q target networks cannot adapt to reward changes.

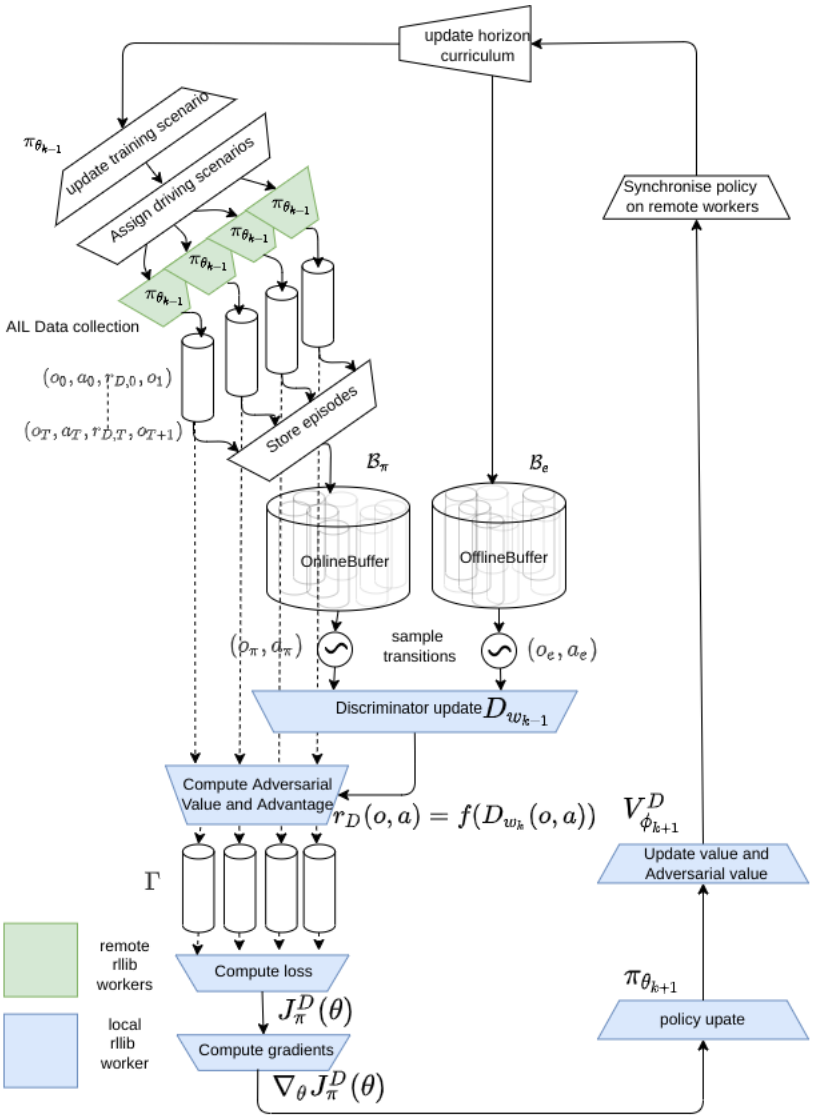

Figure 5.2: AIL Training procedure

detailed composition of the databases can be found in annexes .2.

We observe in tab.5.1 that AIL algorithms AIL,VAIL,WAIL all outperforms BC in terms of imitation performances (ADE-5,ADE-10,ADE-15) but also in terms of safety with significantly less episodes with collisions and less off-road driving. We see that imitation performances of a PPO baseline are also significantly worse than most AIL algorithms which means that PPO trained with the synthetic reward introduced in 4 is effectively learning a different strategy from the expert but it has to be noted that PPO agents tend to drive in a safer way regarding the results provided in tab.5.1 because the synthetic reward was specifically designed for this purpose: avoiding collisions and staying on-road. Concerning AIL algorithms we observe that WAIL is performing worse than GAIL and VAIL especially in the long term where it clearly deviates from experts trajectory. Finally we note that best imitation performances are reached with VAIL that prevents the discriminator to saturate such that the policy stays guided during training. We will analyse more in depth WAIL and VAIL algorithms in the next sections.

	Huge_R_basic					
	ADE-5	ADE-10	ADE-15	CR%	FCR%	Off%
BC	5.20	8.21	13.6	55	23.4	10.30
PPO_R	4.80	6.8	9.67	17.4	7.8	5.3
GAIL	3.6	4.72	5.95	33.2	18.6	4.1
VAIL	2.60	4.51	5.34	28.2	17.3	3.8
WAIL	3.53	6.1	8.5	38.9	18.5	5.2
	Huge_I_basic					
	ADE-5	ADE-10	ADE-15	CR%	FCR%	Off%
BC	5.82	8.67	12.1	45	21.2	11
PPO_R	4.6	7.1	9.3	15.4	7.2	4.5
GAIL	3.8	5.61	7.23	27.2	16.4	3.4
VAIL	3.42	4.82	6.47	25	14.3	3.1
WAIL	4.1	7.8	11.21	28.5	14.3	5.2
	Huge_M_basic					
	ADE-5	ADE-10	ADE-15	CR%	FCR%	Off%
BC	5.2	7.3	12.3	65	35.2	5.3
PPO_R	4.2	7.63	11.31	13.5	7.4	2.2
GAIL	3.8	5.82	6.64	14.4	6.2	3.4
VAIL	3.20	5.41	5.65	12	6.1	3.1
WAIL	3.53	6.7	7.82	15.2	6.7	4.2

Table 5.1: Test performances comparison between AIL algorithms and other baselines.

5.1.2.3 Influence of horizon curriculum

The policy performances are considerably more difficult to optimize than the discriminator performances. Indeed the policy has to imitate realistically the expert during the whole episode while errors compounds at each decision steps whereas the discriminator has to classify expert and learner per decisions step. The driving policy can quickly end up in a completely unrealistic situation during closed loop simulation which may be very easy to detect for the discriminator. As a consequence the bigger the decision step t the more difficult it becomes for the policy to fool the discriminator with transition (s_t^π, a_t^π) because the state s_t accumulates errors and can look significantly different from $(s_t^{\pi_e}, a_t^{\pi_e})$. Since the policy can quickly deviate from the expert reference trajectory it can be beneficial to early stop simulation during training before the agent completely fails. This idea was first proposed in [10] but according to their work, the simulation horizon is increased decision step by decision step without specific criteria based on the current training performances. In the following, we propose several methods to update the horizon based on training statistics before motivating our final choice. Since the discriminator can quickly get too accurate to properly guide the policy it is natural to consider a criteria based on training statistic of the discriminator. At each training step k, as depicted on fig. 17, we compute for each scenario $S \in \Gamma_S$ sampled in the training batch Γ a trajectory score according to the discriminator for the expert policy s_{π_e} and for the associate the learner s_π. We then compute the ratio r_Γ of the trajectory score and average all the ratio on all scenarios

contained in the training batch:

$$s_\pi = \Pi_{t=0}^{H_S} \sigma(D_{\varphi_k}(s_t^\pi, a_t^\pi)) \tag{5.37}$$

$$s_{\pi_e} = \Pi_{t=0}^{H_S} \sigma(D_{\varphi_k}(s_t^{\pi_e}, a_t^{\pi_e})) \tag{5.38}$$

$$r_\Gamma = \mathbb{E}_{S \in \Gamma_S}[\frac{s_\pi}{s_{\pi_e}}] \tag{5.39}$$

The ratio should stay close to one such that the policy keeps challenging the discriminator but in practice it stays smaller than 1 because the discriminator accuracy is also improved at each training step. In order to decide if the simulation horizon should be changed[4], we have to set thresholds on the ratio which is not easy in practice because not directly related to policy performances. A workaround is to use the average distance error δ_Γ at the current horizon H_k to decide if the policy is ready to be trained on longer episodes. Based on policy trajectories in the training batch Γ and their associate expert trajectories we can compute δ_Γ after each data collection as follows:

$$\delta_\Gamma = \mathbb{E}_{(\tau_\pi, \tau_e) \in \Gamma}[\frac{1}{H_k} \sum_{i=1}^{H_k} |\tau_{\pi,i} - \tau_{e,i}|] \tag{5.40}$$

Since thresholds on δ_Γ can easily be determined experimentally based on imitation performances obtained during previous training's we choose δ_Γ as the main criteria to define our horizon curriculum. Given a simulation horizon H_k and an horizon schedule:

$$[(\delta_1^{min}, \delta_1^{max} H_1), (\delta_2^{min}, \delta_2^{max} H_2), ..., (\delta_N^{min}, \delta_N^{max} H_N)] \tag{5.41}$$

the function called *updateSimulationHorizon* in alg.17 update the simulation horizon according to the following procedure detailed in alg.18. Note that horizon curriculum assumes that

Algorithm 18 Horizon scheduler

1: **INPUTS**:
- training batch Γ
- index j of current horizon

2: **OUTPUTS** : index j of new horizon
3: compute $\beta_\Gamma$on the training batch Γ
4: **if** $\delta_\Gamma > r_j^{max}$ **then**
5: $\quad j \leftarrow max(j-1, 1)$
6: **else if** $\delta_\Gamma < r_j^{min}$ **then**
7: $\quad j \leftarrow min(j+1, N)$
8: **end if**
9: return j =0

the return of each episode of length H_j at time t is computed by bootstraping the value function

[4]Changing the simulation horizon impacts data collection at the next training step.

at the end of the episodes.

$$R_t = \sum_{i=t}^{H_j - j} \gamma^i . r(s_{t+i}, a_{t+i}) + V_{\phi_{k-1}}(s_{H_j+1}) \tag{5.42}$$

This enables to reduce the value bias even if the last value estimate is generally poorly estimated due to the lack of data at the last step of the simulation. In order to avoid totally unrealistic transitions as for instance when the agent is largely off-road, we allow simulation terminations at $t < H_j$[5].

In the following experiment we study the influence of the horizon curriculum on long term imitations performances with different horizon schedules detailed in tab.5.2. For those experiences, we built a specific scenario database called $Huge\_R\_horizon\_curriculum$ on the roundabout as detailed in annexes..2. The principle consists in dividing a set of expert episodes into consecutive chunks of given simulation horizons. We repeat this operation for different horizon included in $[2.5, 5, 7.5, 10.0, 12.5, 15]$ and we obtain our final scenario database where several episodes partially overlap. The intuition behind horizon curriculum is that the policy will progressively improves its performances on small episodes chunks before being trained on longer episode chunks in order to limit compounding errors.

	E_1	E_2	E_3	E_4	E_5
candidate horizon	[15]	[2.5, 5, 7.5, 10.0, 12.5, 15]	[5, 10, 15]	[5, 10, 15]	[5, 10, 15]
initial_horizon	15s	2.5s	5s	5s	15s
training_batch_size	100 000	100 000	100 000	100 000	100 000
keep_shorter_horizon	False	False	False	True	True
horizon_step	0	2.5s	5s	5s	5s

Table 5.2: Horizon schedules

To generate the curves in fig.5.3, we ran for each experience, at each training iteration, an evaluation to compute (ADE-5,ADE-10,ADE-15) on the 200 original driving scenarios. This step is not necessary for training but only to compare different curriculum with all the metrics (ADE-5,ADE-10,ADE-15)[6]. Note that during evaluation the policy is sampled from the policy distribution because the horizon schedule is applied on training metrics while the policy is exploring. For all the experiences, the policy is trained from scratch from fixed random weights. We apply the curriculum based on the average distance error criteria detailed above to update the simulation horizon. In the first experience, named E_1, we directly train the agent on complete episodes of 15 seconds. Since the policy is exploring during training, it tends to deviate after the first five seconds and the discriminator gets quickly too accurate which prevents from guiding properly the policy. Training directly on long horizon episodes as in experience E_1 does

[5]We choose a threshold of maximum lateral distance with respect to the center line equal to 5 meters because no expert in the database goes above that distance laterally.

[6]For experience E_2 for instance, ADE-15 is not computed during training at the very beginning because we start to simulate for only 2.5 seconds

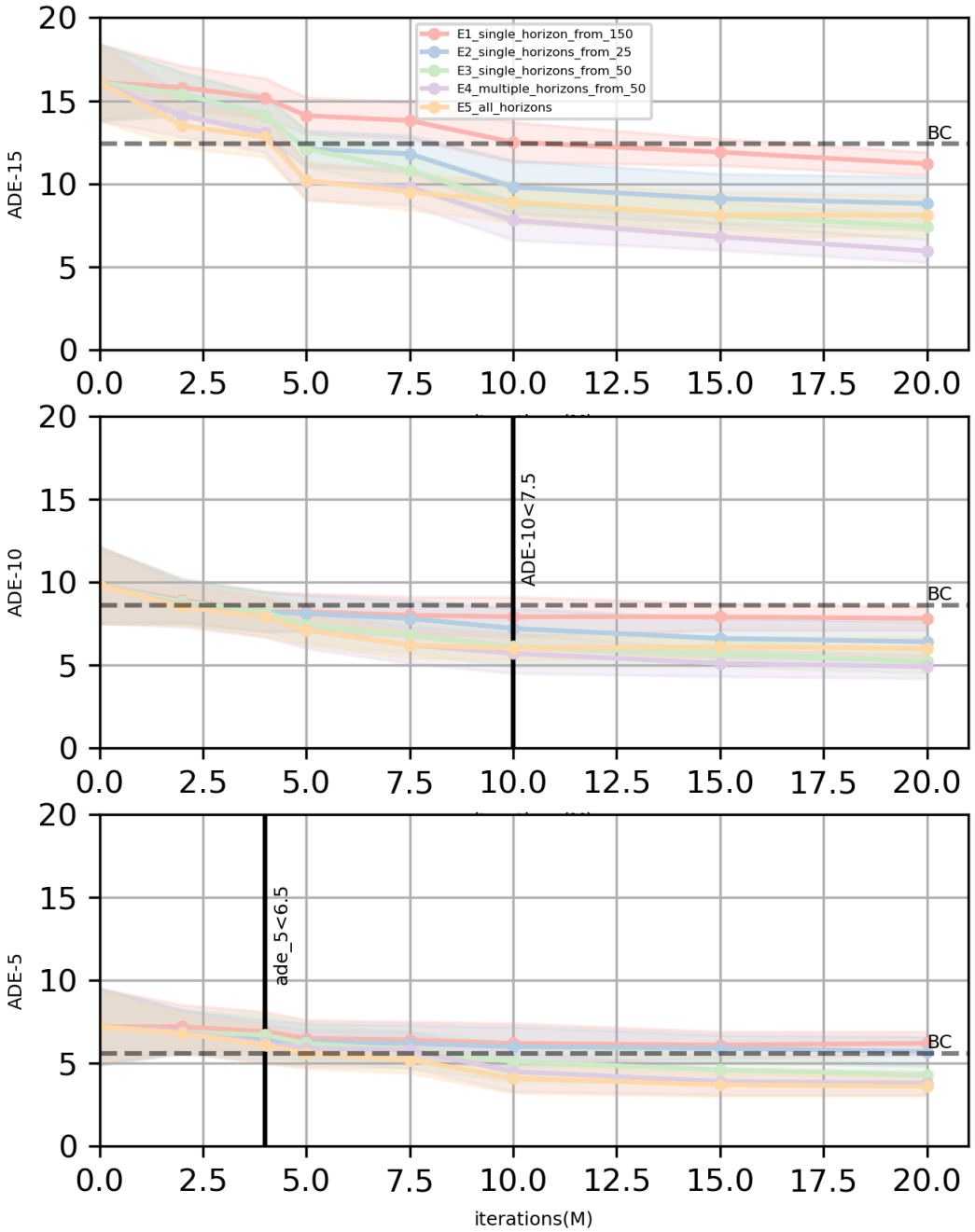

Figure 5.3: Influence of horizon curriculum on training performances: we illustrate when horizon 5s and horizon 10s are passed for experience E_2. On the left side of the vertical line the simulation horizon is equal to respectively 5 and 10 seconds and on the right side it is increased to respectively 7.5 and 12.5 seconds because ADE-5 and ADE-10 reached the thresholds defined in the curriculum.

not lead to good imitation performances as shown by training results on fig.5.3. In case we gradually increase the horizon with a step equal to 2.5 seconds as in experience E_2 [7], we observe improvements in the short term but the performances are unstable. This can be explained by the fact that the policy can start to accumulate errors when the simulation horizon gets bigger

[7]The horizon schedule [(25,4.5),(50,6.5),(75,7.0),(100,7.5),(125,8.0),(150,8.5)] for E_2 just uses lower thresholds δ_{min} at which the horizon can be increased.

but since the shorter episodes are not kept in the training database, the policy tends to forget what it previously learnt and cannot easily recover. In experience E_4 we keep shorter episodes in the training database when a new simulation horizon is reached and we observe that short term performances up to horizon 10 seconds considerably improve compared to experience E_3 where shorter episodes are not kept. The training performances get also more stable because intermediate errors that can come back during training can be quickly corrected since they appear in all episodes that overlap. The main disadvantage of this method is the large number of training episodes that it needs during training. Given an initial set of 200 demonstrations of 15 seconds decomposed according to the following horizon curriculum : $[2.5, 5, 7.5, 10.0, 12.5, 15]$: keeping all episodes at the end of the training implies to sample among (6+3+2+1+1).200=2600 scenarios. It appears that the best imitations performances were obtained in experience E_4 where we reduce the number of horizons from $[2.5, 5, 7.5, 10.0, 12.5, 15]$ to $[5, 10.0, 15]$ and where we also keep shorter episodes. Training with possible scenario from the beginning is insufficient to improve long term performance because long trajectories are generated less frequently in the training batch. One considerable improvement provided by the horizon curriculum is the reduction of long term imitation errors (ADE-10,ADE-15) compared to BC as shown in fig.5.3. However, properly setting the curriculum can require intensive search and depends on the complexity of the road map.

5.1.2.4 Balancing policy and discriminator training

The goal of each training iteration in AIL algorithms is to improve policy performances measured in terms of imitations. For a fixed training horizon H we score the policy with the Average Distance Error at horizon H denoted (ADE-H). We define the policy imitation score (defined earlier in) as the statistic provided by ADE-H and we aim to decrease gradually ADE-H after several training iterations. However the AIL training iterations described in 17 suffers from the same instability issues as GANs [201]. GANs usually suffer from mode collapse which means that they cannot produce diverse samples. Since our policy is conditioned on a high dimensional observation and on a command, the demonstrations get drastically less multi modal and it does not have a major impact on the driving behaviour compared to previous works [108] where policies are not goal based and can turn right or left. The main problem that AIL faces is the unbalanced competition between the generator and the discriminator exacerbated by the sequential nature of the decision process. The discriminator trained on binary targets tends to quickly overfit because its task is much easier to solve than the generator that has to provide realistic trajectories and not only realistic instantaneous transitions (s_t, a_t, s_{t+1}). Although the existence of a global Nash Equilibrium has been proven [55], practical implementations of GANs with neural networks prevents from operating directly in the distributional space and only enables to operate on the parameter space of the the generator. The most challenging part consists in modifying policy parameters such that next trajectories look more realistic than at previous iterations. Our goal in this section,is to investigate how we can adjust the standard

training procedure described in 17 such that the policy imitation score5.1.2.4 gradually improves without resorting to numerous hyper-parameters. To understand why the policy is hard to train, consider how GAN's generator loss is expressed:

$$\mathcal{L}_{GAN}(\theta) = \mathbb{E}_{z\sim p(z)}[log(D(G_\theta(z))] \tag{5.43}$$

The GAN generator training loss enables to back-propagate through the discriminator whereas the policy loss for on-policy AIL algorithms is based on advantage estimates $\widehat{A}_t^{GAE}(s_t, a_t)$ and on probability ratios $\frac{\pi(.|s_t;\theta)}{\pi_{k-1}(a_t|s_t)}$.

$$\mathcal{L}_\pi(\theta) \approx \mathbb{E}_{s\sim p_{\pi_{k-1}}(s), a\sim\pi_{k-1}(.|s)}[\frac{\pi(.|s_t;\theta)}{\pi_{k-1}(a_t|s_t)}\widehat{A}_t^{GAE}(s_t, a_t)] \tag{5.44}$$

The discriminator gradients do not intervene during the policy update because discriminator only comes into play during advantage computation. As a consequence the policy is only guided by the advantage estimate $\widehat{A}_t^{GAE}(s_t, a_t)$ whose scale and sign determines how much the probability of action a_t should be increased or decreased for state s_t. Therefore we should adjust the whole training iteration such that advantage estimates get as accurate as possible[8]. The advantage estimator $\widehat{A}_t^{GAE}(s_t, a_t)$ depends both on discriminator for the reward $r_{D_{w_{k-1}}}(s_{t+l}, a_{t+l}, s_{t+l+1})$ and on the state value function $V_{\phi_{k-1}}^{\pi_{k-1}}(s_{t+l+1})$ through the TD residual δ_{t+l}^V:

$$\widehat{A}_t^{GAE}(s_t, a_t) = \sum_{l=0}^{H-t}(\gamma.\lambda)^l\delta_{t+l}^V \tag{5.45}$$

Given a transition (s_t, a_t, s_{t+1}) of the training batch Γ_k, computing the TD residual δ_{t+1}^V is not as straightforward as in reinforcement learning, because not only the policy is changing but also the reward. In principle, computing δ_{t+l}^V requires the value function of the previous policy $\pi_{\theta_{k-1}}$ but evaluated with the current reward model $r_{D_{w_k}}$ which suggests that we can compute δ_{t+l}^V as follows:

$$\delta_{t+l}^V = r_{D_{w_k}}(s_{t+l}, a_{t+l}, s_{t+l+1}) + \gamma \tag{5.46}$$

$$V_{\phi_k}^{\pi_{k-1}}(s_{t+l+1}) - V_{\phi_k}^{\pi_{k-1}}(s_{t+l}) \tag{5.47}$$

where the value $V_{\phi_k}^{\pi_{k-1}}$ is trained just after the discriminator update on Γ_k but with the new reward $r_{D_{w_k}}$ on each transition. In this case, the TD residuals used for advantage computation may just reduce to zero since we trained the value function on the same transitions as the one used for computing the TD residuals. An alternative method that we call TD-corrected is to consider that the new reward model stays close to the older one with an additional correction

[8]Advantage estimates should have low variance and more importantly low bias

term $(r_{D_{w_k}}(s_{t+l}, a_{t+l}, s_{t+l+1}) - r_{D_{w_{k-1}}}(s_{t+l}, a_{t+l}, s_{t+l+1}))$:

$$\delta^V_{t+l} = r_{D_{w_{k-1}}}(s_{t+l}, a_{t+l}, s_{t+l+1}) + (r_{D_{w_k}}(s_{t+l}, a_{t+l}, s_{t+l+1}) - r_{D_{w_{k-1}}}(s_{t+l}, a_{t+l}, s_{t+l+1})) + \gamma \quad (5.48)$$

$$V^{\pi_{k-1}}_{\phi_{k-1}}(s_{t+l+1}) - V^{\pi_{k-1}}_{\phi_{k-1}}(s_{t+l}) \quad (5.49)$$

In this case, we can keep using the older value function trained on Γ_{k-1} with the older reward model $r_{D_{w_{k-1}}}$ to compute the TD residuals. We apply this approach in our algorithm detailed in alg.19 such that the value can be trained at the end of the training iteration. This method requires that the reward model $r_{D_{w_k}}$ changes slowly to let the policy progressively explore new parts of the state space without suddenly getting lost due to excessive incentive to explore. In order to limit the discriminator changes during an AIL training iteration, we propose a method inspired by PPO clipping strategy [167]. Intuitively, we aim to control the total variation denoted δ_D between the Bernoulli distributions parametrized by the old discriminator $D_{w_{k-1}}$ and the new discriminator:

$$\delta_D = \mathbb{E}_{(s,a)\sim 0.5\rho_\pi + 0.5\rho_{\pi_e}}[D_{TV}(D_w(.|(s,a))|D_{w_{k-1}}(.|(s,a))] = \mathbb{E}_{(s,a)}[\big|D_w(.|(s,a) - D_{w_{k-1}}(.|(s,a)\big|) \quad (5.50)$$

One way to proceed would be to optimize the discriminator with a hard constrain on the total variation similarly to TRPO [168]. Since enforcing a hard constrain considerably increase the computation time, we reformulate the discriminator loss $\mathcal{L}_D$ such that the local changes of density gets restricted:

$$argmax_w \mathcal{L}_D(w) \quad (5.51)$$

$$\mathcal{L}_D(w) = \mathbb{E}_{(o,a)\sim\mathcal{B}_e}[r_\eta(s,a).logD_w(s,a)] + \mathbb{E}_{(o,a)\sim\mathcal{B}_\pi}[r_\eta(s,a).log(1 - D_w(s,a))] \quad (5.52)$$

$$r_\eta(s,a) = 1_{[-\delta_D,\delta_D]}[\left|\frac{D_w(s,a)}{D_{w_k}(s,a)} - 1\right|] + \eta.1_{\mathbb{R}\setminus[-\delta_D,\delta_D]}(\left|\frac{D_w(s,a)}{D_{w_k}(s,a)} - 1\right|) \quad (5.53)$$

$$(5.54)$$

For our experience, we choose to set $\eta = 0.0$ such that the ratio $r_\eta(s,a)$ reduces to the indicator function but we could imagine other variations where η could be set to 0.1 which means that gradients on the discriminator parameters at a state action pair (s,a) with high density changes will be downscaled. An alternative to this clipping strategy would be to monitor estimates of the total variation averaged on mini-batches and stop the SGD once a threshold is exceeded. This strategy is similar to the one used on the policy in [179] but it did not lead to stable improvements when applied on the discriminator. Indeed, stopping the whole discriminator training, potentially at the beginning of SGD, may result to a poor discriminator that cannot efficiently guide the policy. In contrast, our method uses probability ratio clipping that just cancel few samples in mini-batches used to compute the discriminator loss but do not stop SGD. Note that the discriminator ratio threshold δ_D can be chosen the same way as the policy

ratio threshold δ_π which is set considering how much the policy can be changed in average during each AIL training iteration.

$$\delta_\pi \propto \mathbb{E}_{(s,a)\sim\rho_{\pi_{\theta_k}}} \left| \frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} - 1 \right| \tag{5.55}$$

The two criteria (δ_D, δ_π) enable to avoid choosing specific number of training epochs (N_D, N_π) that do not explicitly restrict the amount of change per training iterations in the distribution space. Therefore, we can chose the same number of training epochs for both the discriminator and the policy but we still need to define how fast each of them should learn. According to the Two Time-Scale Update Rule (TTUR) method [67] developed for GANS, it was shown that choosing different learning rates for the generator and discriminator still guarantee convergence under mild assumptions to a stationary local Nash equilibrium. They recommend choosing a higher learning rate for the discriminator and a lower one for the generator such that the generator has to make smaller steps to fool the discriminator. However, we already explained that in AIL a discriminator can quickly overfit especially if trained faster which results in a saturating reward. It appears that using smaller learning rate for the discriminator and a higher learning rate for the policy could also make sense.

So far, we mainly explained how to adjust discriminator and generator trainings but the value function also have its importance and especially through the TD residuals. In contrast to the policy or to the discriminator we do not need to restrict the value function but we want it to generalize as much as possible such that TD residuals do not get too biased which would be detrimental for policy update. To avoid value overfitting we can divide the training batch $\Gamma_k$into a test set Γ_k^{test} and a training set Γ_k^{train} and we should track the value function test error $\Delta V = \mathbb{E}_{s\sim\Gamma_k}[\left|V^{target}(s_t) - V_{\phi_{k-1}}^{\pi_{\theta_{k-1}}}(s_t)\right|]$ after each training epoch. Note that the target return V^{target} is computed with the previous reward model $D_{w_{k-1}}$:

$$V_{target}(s_t) = \sum_{l=0}^{H-t} \gamma^l . r_{t+l}^{D_{w_{k-1}}} + \gamma^{H-t+1} . V_{\phi_{k-1}}^{\pi_{\theta_{k-1}}}(s_H) \tag{5.56}$$

The test error ΔV can be used to launch additional data collection with policy $\pi_{\theta_{k-1}}$ such that the value can be trained with more transition sample. More precisely, in case $\Delta V < \Delta V_{max}$ we allow at most L additional data collections with the old reward model $r_{D_{w_{k-1}}}$to complete the training batch $\Gamma = \bigcup_{l\in[0,L]} \Gamma_{k+l}$ of the value function as described in alg.19. Note that this technique has a limited impact because the reward model $r_{D_{w_{k-1}}}$ may not generalize very well on samples outside Γ_k . Another way to improve the test performances of the value function is to generate multiple episodes of the same scenario in the training batch [9] and to store them in the training batch. Multiple episodes starting from the same state enable to better estimate the stochasticity of the policy for the value function. It is all the more important that for efficient value estimation, the policy standard deviation σ should stay low to avoid that the variance of

[9]This implies using large training batches because we also sample episode on each scenario

the trajectory distribution explode and hence the value error.

In the following, we study how hyper-parameters $\delta_\pi, \delta_D, L, \Gamma_k, \alpha_\pi, \alpha_D$ influence the training performances and which conditions are necessary to obtain stable policy improvements. We choose to set a fixed simulation horizon of 10 seconds such that the policy is enough challenged during training. All experiences of this section are realised on training scenarios of the $Huge\_R\_Basic$ scenario database detailed in annexes .2. We first analyse how clipping the probability ratio of the policy and the discriminator impacts the policy performances measured with $ADE - H$. We choose a maximum number of training epochs equal to $N_{train} = 5$ for the policy, the discriminator and the value function. We first start to train the value function on the same training batch as the one used for the policy. For all experiences, we applied 200 AIL training iterations and we report the final Average Distance Error after 10 seconds (ADE-10) in closed loop evaluation on scenarios of the training database. In experience 0 re-

	(δ_π, δ_D)	(α_π, α_D)	$ADE-5$	$ADE-10$
experience 0	$\delta_\pi = None, \delta_D = None$	$\alpha_\pi = 0.0001, \alpha_D = 0.0001$	8.54	13,45
experience 1	$\delta_\pi = 0.3, \delta_D = None$	$\alpha_\pi = 0.0001, \alpha_D = 0.0001$	3.50	6.70
experience 2	$\delta_\pi = 0.3, \delta_D = 0.1$	$\alpha_\pi = 0.0001, \alpha_D = 0.0001$	3.22	6.21
experience 3	$\delta_\pi = 0.03, \delta_D = 0.1$	$\alpha_\pi = 0.0001, \alpha_D = 0.0001$	3.24	6.59
experience 4	$\delta_\pi = 0.3, \delta_D = 0.05$	$\alpha_\pi = 0.0001, \alpha_D = 0.0001$	2.98	5.73

Table 5.3: Influence of probability ratios clipping for the policy and the discriminator on training performances

ported in tab.5.3, we do not use probability ratios at all and we observe that the policy cannot learn. In experience 1, we start by clipping policy probability ratios according to PPO algorithm[167] which results to large performance improvements for short term imitation but still high deviations in the long term. In case we also clip the discriminator probability ratio as in experience 2, we observe that long term imitation performances start to improve. Reducing the probability ratio of the policy did not lead to better performance as reported in experience 3. In contrast, if the discriminator probability ratio threshold is reduced as in experience 4, we obtain the best training performance. We conclude that sufficiently restricting local changes of the discriminator distribution can also be beneficial for the policy. Since the reward cannot change abruptly between two training iterations, the value function can gradually adapt to the new reward landscape which helps to compute accurate advantage estimates.
However the policy performances also depend on the training speed induced by the learning rates whose scales were fixed in the last experiences. In the following, we try to understand if choosing specific policy and discriminator learning rates can complement ratio clipping. We train the policy in a similar fashion, with the same initial parameters, for 200 AIL iterations with different learning rates. In the nest experiences, we use the best pair of probability ratio thresholds ($\delta_\pi = 0.3, \delta_D = 0.05$) and we search what is the optimal pair of learning rates ($\alpha_\pi \alpha_D$) for keeping balanced performances between the policy and the discriminator for a fixed number

Algorithm 19 Advanced AIL Training Procedure

INPUTS:

- Expert Database $\mathcal{D}^e = (\mathcal{D}_{train}, \mathcal{D}_{eval})$
- n_Γ, N_{train} : training batch size,training epochs for discriminator and policy
- N_V training epochs of the value function
- L: number of additional data collection to train the value
- $(\delta_\pi^{max}, \delta_D^{max})$: early stopping criteria for respectively the policy an d the discriminator ΔV_{min}: minimum value error

for $k = 1, ..., N_{train}$ **do**
  E_V =Evaluate(π,$\mathcal{D}^e_{validation}$)
  $[[S_j^{(i)}]_{j\in J_i}]_{i\in[1,...,N_s]}$ =AssignScenarios($\mathcal{D}^e_{train}$)
  Γ_k =CollectEpisods($\mathcal{S}^e, \pi_{\theta_{k-1}}$)
  $\mathcal{B}_e, \mathcal{B}_\pi$ =UpdateExpertAndStudentBuffers($\Gamma_k, \mathcal{S}^e, \mathcal{B}_e, \mathcal{B}_\pi$)
  for $i \in [1, ..., M]$ **do**
    for i=1,...,N_D **do**
      $(B_\pi, B_e) \sim \mathcal{B}_e, \mathcal{B}_\pi$
      $D_w \leftarrow D_w + \alpha_D \nabla_w \mathcal{L}_D(B_\pi, B_e)$
    end for
    for $\tau \in \Gamma_k$ **do**
      for (s_t, a_t, s_{t+1}) in Γ_k **do**
        $\delta^V_{t+l} = r_{D_{w_k}}(s_{t+l}, a_{t+l}, s_{t+l+1}) + \gamma V^{\pi_{k-1}}_{\phi_{k-1}}(s_{t+l+1}) - V^{\pi_{k-1}}_{\phi_{k-1}}(s_{t+l})$
        $\widehat{A}^{GAE}_t(s_t, a_t) = \sum_{l=0}^{H-t} (\gamma.\lambda)^l \delta^V_{t+l}$
      end for
    end for
    if $k > 1$ **then**
      for i=1,...,N_π **do**
        $B_\pi \sim \Gamma_k$
        $\pi_\theta \leftarrow \pi_\theta + \alpha_\theta \nabla_\theta \mathcal{L}_\pi(B_\pi, \pi_{\theta_{k-1}})$
      end for
    end if
  end for
  $l = 0, \Delta_V = \Delta V_{max}$
  $\Gamma^V_k \leftarrow \Gamma_k$
  while $l < L$ *and* $\Delta_V >= \Delta V_{min}$ **do**
    $\Gamma^V_{k,train} \leftarrow growDataset(\pi_{\theta_k}, D_{w_k}, l)$
    for $B_\pi \sim \Gamma^V_{k,train}$ **do**
      $V_\phi \leftarrow V_{\phi_k} + \alpha_V \nabla_\phi \mathcal{L}_V(B_\pi, V_{\phi_{k-1}})$
    end for
    $\Delta_V \leftarrow evaluateValue(V_\phi, \Gamma^V_{k,test})$
    $l \leftarrow l + 1$
  end while
end for

of training epochs $N_{train} = 5$ for the policy and the discriminator[10].

The first row of tab.5.5 reports the best performances obtained previously with the same

	N_{train}	(δ_π, δ_D)	(α_π, α_D)	$ADE-5$	$ADE-10$
experience 0	5	$\delta_\pi = 0.3, \delta_D = 0.05$	$\alpha_\pi = 0.0001, \alpha_D = 0.0001$	2.98	5.73
experience 1	10	$\delta_\pi = 0.3, \delta_D = 0.05$	$\alpha_\pi = 0.0001, \alpha_D = 0.0001$	4.2	8.23
experience 2	5	$\delta_\pi = 0.3, \delta_D = 0.05$	$\alpha_\pi = 0.0001, \alpha_D = 0.00001$	2.95	5.64
experience 3	5	$\delta_\pi = 0.3, \delta_D = 0.05$	$\alpha_\pi = 0.001, \alpha_D = 0.0001$	3.44	6.23
experience 4	5	$\delta_\pi = 0.3, \delta_D = 0.05$	$\alpha_\pi = 0.001, \alpha_D = 0.001$	4.37	8.61

Table 5.4: Influence of learning rates and training epochs on training performances

learning rates for the discriminator and the policy. Increasing the number of training epochs for the discriminator and the policy does not enable to improve the performances because advantage estimates are only reliable if the optimized policy stay close to the previous which may not be the case after more than 5 epochs. In case we slow down the discriminator training as in experience 2, we do not observe substantial improvements compared to experience 0. Accelerating the policy training as in experience 3 destabilizes the policy whose performances tend to oscillate during training. In the last experience, we accelerate the policy and the discriminator which leads to lower performances. Even if local changes of the policy and the discriminator distributions are restricted with probability ratio clipping, advantage estimates can still be biased which explains that the policy can suffer from high policy learning rates. We conclude that using different learning rates for the discriminator and the policy does not significantly improve performances once suitable probability ratio threshold are found.

Finally, we investigate how the generalization performances of the value function influence the policy performances for a fixed simulation horizon of $H = 10s$. The first row reports the standard configuration with probability ration clipping used previously. In experience 1, we use a training batch size Γ twice bigger to generate multiple episodes on the same scenario for training the value but also the discriminator and the policy. For experience 2 we allow $L = 2$ additional data collections with the old reward model $r_{D_{w_{k-1}}}$ to complete the training batch $\Gamma = \bigcup_{l \in [0,L]} \Gamma_{k+l}$ of the value function[11].

	(δ_π, δ_D)	(α_π, α_D)	L	$\lvert\Gamma_k\rvert$	$ADE-5$	$ADE-10$
experience 0	$\delta_\pi = 0.3, \delta_D = 0.05$	$\alpha_\pi = 0.0001, \alpha_D = 0.0001$	0	100000	2.98	5.73
experience 1	$\delta_\pi = 0.3, \delta_D = 0.05$	$\alpha_\pi = 0.0001, \alpha_D = 0.000$	0	200000	2.82	5.34
experience 2	$\delta_\pi = 0.3, \delta_D = 0.05$	$\alpha_\pi = 0.0001, \alpha_D = 0.000$	2	100000	3.03	6.13

Table 5.5: Influence of value function generalization on training performances

We observe in tab.5.5 that doubling the training batch size is the most efficient way to improve the long term imitation performances. This can be explained by the fact that not only the value see more examples but also the discriminator during its training. Collecting

[10] We could even choose different number of training epochs for the policy and the discriminator but we did not notice significant improvements for long term performances(ADE-10).

[11] Discriminator and policy do not see more training data

multiple training batches that each uses the same discriminator for computing the rewards is less efficient because the discriminator not always generalize very well on new states.

5.1.2.5 Influence of reward form and episode termination

Standard AIL algorithms based on f-divergence minimization as GAIL[69] formulates a surrogate reward function based on the discriminator output. Since the discriminator parameterize a Bernouli distribution, its output is restricted to positive values which can be problematic depending on the expert to imitate. A strictly positive reward function as $r(s,a) = -log(1-D(s,a))$ prevents the agent from solving goal based task in a minimal number of steps because the agent is encouraged to survive in the environment by collecting more rewards. In contrast, a strictly negative reward function $log(D(s,a))$ is not able to emulate a survival bonus and the agent may try to finish the episode too quickly [93, 78, 198]. If we consider our driving task, it is not really goal based because our policy is constantly conditioned on a reference path that has to be followed without failure : there is no final destination that determines the success of the driving task because we aim to stay as close a possible to all states visited by the expert reference trajectory rather than just the last one. Our driving task is not survival based either because it just requires to spend enough time in the environment to visit each state of the expert trajectory. In order to avoid strictly positive or negative rewards we choose a neutral reward function [93] which reduces to the discriminator output before the application of the sigmoid and that as the range $\mathbb{R}$.

$$r(s,a) = log(D(s,a)) - log(1 - D(s,a)) \tag{5.57}$$

If the agent learns with this reward, it will always get fewer rewards than if it follows shorter non-expert trajectories or if it loops as shown in [78]. However, in many environment especially task based there are generally terminal states where agents enter and finish the episode. Depending on the expert behaviour and on the nature of the terminal states, the agent may either try to avoid them or to reach them as soon as possible. In case we define the final return $R_T(s_t, a_t)$ with just the terminal state reward $r(s_T, a_T)$ since the value cannot be bootstrapped [143] without having access to s_{T+1}, the learner cannot access the true value of the terminal state and is therefore still biased. Indeed we did not encourage the survival based agent to avoid the terminal states and we did not encourage the goal based agent to transition to s_T in particular. In order to avoid the introduction of a bias due to terminal states we can consider that after entering a terminal state, the agent visits an absorbing state s_a [181] where it stays whatever action it does. Therefore the return at the terminal state can be written :

$$R_T(s_T, a_T) = r_T(s_T, a_T) + \gamma . \sum_{t=T+1}^{\infty} \gamma^{k-t} . r_a = r_T(s_T, a_T) + \frac{\gamma . r_a}{1-\gamma} \tag{5.58}$$

The value of r_a can be learnt by the discriminator as suggested in [93] that however reports training instabilities. A simple workaround would be to fix manually the value of r_a assuming that we know if the terminal state is detrimental or not for the agent (task completion or failure). If we consider our driving task, adding terminal states could be beneficial for restricting exploration because the policy may visit undesirable states far from the expert state support and the transitions leading to those undesirable states are not very useful for guiding the policy especially after some training iterations when the policy starts to improve. In order to bound exploration in the state space, we can easily use domain knowledge to stop the episode once the agent is too far from the expected configurations. For instance, one can use a threshold n_{max} on the lateral distance with respect to the reference path denoted n to stop the episode once the agent is too far : $n > n_{max}$. We can also stop an episode after that a collision occur but if collisions occur too early in the episode, we may have to learn on very short trajectories which can in turn prevents the agent from discovering interesting strategies. Since we have access to the instantaneous distance error with respect to the reference expert, we could even trigger an episode termination when the distance error gets too big. However, terminating an episode based on a distance error with respect to an expert can also be confusing for the agent since it may have reached a safe configuration on road with appropriate safety distances. Additionally the terminal state reward has to be learned contrary to previous examples where r_a was fixed to a sufficiently negative value. To avoid learning the reward values for absorbing states or setting manually their value based on domain knowledge, we can also consider that arbitrary episode terminations triggered by any kind of events (collision, lateral distance,...) are just time terminations [143]. In case of time termination, the return of the last transition (s_T, a_T, s_{T+1}) reached is estimated by bootstraping the value function.

$$R_T(s_T, a_T) = r_T(s_T, a_T) + \gamma . V_{\phi_{k-1}}(s_{T+1}) \tag{5.59}$$

The main limitation of this technique is that the value may poorly generalize the true value $V^{\pi_{k-1}}(s_{T+1})$ at s_{T+1} because there is no target to train $V_{\phi_{k-1}}$ on the last states s_{T+1}.
In the following we investigate in experiences 1,2,3 how imitation performances evolve during training when we change the form of the reward function. We also analyse the impact of bootstrapping the value at the end of the episode with experiences 3 and 4. Finally we study how the introduction of specific absorbing states for collision abbreviated col, lateral distance with respect to reference path[12] abbreviated n and instantaneous distance error with respect to the expert abbreviated Δd_e could help to improve imitation performance measured by ADE-10. Note that we test two settings, one that consists in manually setting a negative value for absorbing state rewards and one that consists in bootstrapping the value as if termination was just a time termination. We realised our experiences on the database $Huge\_R\_basic$ whose composition is detailed in annexes .2.

[12]We stop the episode if $n > n_{max} = 5m$ meters because the maximum lateral offset of expert does not exceed this value

experience s	reward	aborbing sate	terminal return	ADE-10
experience t 1	$log(D(s,a)$	No	$R_T(s_T,a_T) = r_T(s_T,a_T)$	8.34
experience 2	$-log(1-D(s,a))$	No	$R_T(s_T,a_T) = r_T(s_T,a_T)$	7.94
experience 3	$log(D(s,a) - log(1-D(s,a))$	No	$R_T(s_T,a_T) = r_T(s_T,a_T)$	6.32
experience 4	$log(D(s,a) - log(1-D(s,a))$	No	$R_T(s_T,a_T) = r_T(s_T,a_T) + \gamma.V_{\phi_{k-1}}(s_{T+1})$	5.94
experience 5	$log(D(s,a) - log(1-D(s,a))$	$r_a^{col} = -1$	$R_T(s_T,a_T) = r_T(s_T,a_T) + \frac{\gamma.r_a}{1-\gamma}$	6.95
experience 6	$log(D(s,a) - log(1-D(s,a))$	$r_a^{col} = (1-\gamma)V_{\phi_{k-1}}(s_{T+1})$	$R_T(s_T,a_T) = r_T(s_T,a_T)\gamma.V_{\phi_{k-1}}(s_{T+1})$	6.31
experience 7	$log(D(s,a) - log(1-D(s,a))$	$r_a^{n} = -1$	$R_T(s_T,a_T) = r_T(s_T,a_T) + \frac{\gamma.r_a}{1-\gamma}$	6.82
experience 8	$log(D(s,a) - log(1-D(s,a))$	$r_a^{n} = (1-\gamma)V_{\phi_{k-1}}(s_{T+1})$	$R_T(s_T,a_T) = r_T(s_T,a_T) + \gamma.V_{\phi_{k-1}}(s_{T+1})$	5.64
experience 9	$log(D(s,a) - log(1-D(s,a))$	$r_a^{de} = (1-\gamma)V_{\phi_{k-1}}(s_{T+1})$	$R_T(s_T,a_T) = r_T(s_T,a_T) + \gamma.V_{\phi_{k-1}}(s_{T+1})$	7.68

Table 5.6: Influence of AIL reward form and episode termination on training performances.

We observe in tab.5.6 that using the unbiased reward $log(D(s,a) - log(1-D(s,a))$ enables to reach considerably better performances. We note that using absorbing states for the lateral offset enables to reach better long term performances than just letting the simulation finish at the last step H. We reached the best results with early simulation ending when $n > n_{max}$ with value bootstrapping. It appears that removing totally unrealistic trajectories enables to challenge the discriminator such that it can better guide the policy during the whole training even if sometimes during exploration, the policy starts to deviate too far from the expert reference trajectory.

5.2 Toward robust driving imitations

In this section, we explain how to improve the test performances of AIL algorithms such that the driving policy can imitate human drivers on new scenarios. We first study how to make the policy more robust to distributional shift in sec.5.2.1 before studying to which extent expert driving behaviour can really be explained in sec.5.2.2.

5.2.1 Adapting to distributional shift

In this section we study how the gradual deviation from expert trajectory during closed loop simulation can be reduced. In sec.5.2.1.1 we show that decoupling action selection and short term goal prediction helps the policy to better compensate errors. In sec.5.2.1.2 we show how to provide smoother guidance with different forms of discriminator.

5.2.1.1 Planning target position before action

Displacement in curvilinear coordinates enables to explore easily around a reference path however the agent should still learn to compensate its error with respect to the expert trajectory. If we consider the decision making process, the policy should plan its next target state according to an expert and then infer which action enables to reach this target state. In case the agent starts to drift from the expert trajectory, it should immediately compensate by taking

the appropriate actions. In early stages of deviation, the next target position that can be considered as a short term goal should not change expect in some critical situations where a slight deviation requires full trajectory replanning i.e if a collision is likely to occur, the target is not updated. As a consequence, small variations on the current state should not imply variations in the next target state however it requires to adapt the action. AIL approach intrinsically helps to avoid deviations by guiding the agent with a dense reward but they do not explicitly help to infer the adapted action that leads to the appropriate next state. For traffic simulation, actions are secondary and what primarily matters is how realistic the sequence of states of each agents looks because we aim to generate a realistic traffic and not to control a real vehicle. State only imitation learning focus on recovering the state distribution without direct access to actions in the demonstrations [189] but in our case, we can easily reconstruct accurate sequence of displacements in curvilinear coordinates from expert demonstrations. Some other works [114, 116] build upon state alignment to imitate expert trajectories while reconstructing the action. The algorithm called SAIL[114] relies on a state predictor that first predicts the next desired state $\tilde{s}_{t+1}$ given the current state s_t and then infers the action a_t with an inverse dynamic model $a_t = f(s_t, \tilde{s}_{t+1})$. They leverage $\beta - VAE$ [68] for the state predictor such that the target prediction $\tilde{s}_{t+1}$ remains robust to small variations in the current state s_t while the inverse dynamic is a deterministic function that can be learned on an arbitrary set of transitions (s_t, a_t, s_{t+1}) to get as accurate as necessary. One drawback of this approach is the necessity to predict the entire next target state $\tilde{s}_{t+1}$ which is not fully controllable with the action in our multi-agent setting and which can be very complex because its contains multiple components for ego information, other agents information and the road-network. What really matters to take action is the desired target position Y_{t+1}at $t+1$ relative to the ego agent current frame such that we can extract its curvilinear coordinates with respect to the command polyline and hence infer the action $a_t = (ds_t, n_{t+1})$. As a consequence, we choose to modify the original architecture introduced in [114] such that the predictor only predicts the target position $Y^t_{t+1}$from the next decision step $t+1$ instead of the next state (observation in our case) similarly to motion-forecasting architectures[164]. We define our inverse dynamic such that we do not need to feed $(o_t, \tilde{o}_{t+1})$ but directly an embedding of o_t denoted $h_t = b^\pi_\theta(o_t)$ and the predicted target position Y^t_{t+1} expressed in the same frame as o_t. The planning procedure decomposes as follows : at the current decision step t the observation is first embedded into $h^t_t = b^\pi_\theta(o_t)$. Given the previous actions a_{t-1}, the planner first predicts which target position it aims to reach through a VAE composed of $(D_\theta \circ E_\theta \circ g_\theta)$. The observation embedding h^t_t and the previous actions a_{t-1} are processed by the VAE encoder $(E_\theta \circ g_\theta)$ that generates the latent representation $z^t_t = E_\theta(g_\theta(h^t_t, a_{t-1}))$ associated to the current observation o_t. This latent representation z^t_t is later transformed by the predictor network D_θ into the target position $Y^t_{t+1} = D_\theta(z^t_t)$ and the inverse dynamical model f_θ infers the action $a_t = f_\theta(h^t_t, Y^t_{t+1:t+H})$ the agent should perform to reach the first desired position y^t_{t+1} based on the observation embedding h^t_t . Then a forward dynamical model T_θ updates the observation embedding $h^t_{t+1} = T_\theta(h^t_t, a_t)$ such that we can plan the next decision steps similarly to the auto-regressive planner introduced in chap.3. The

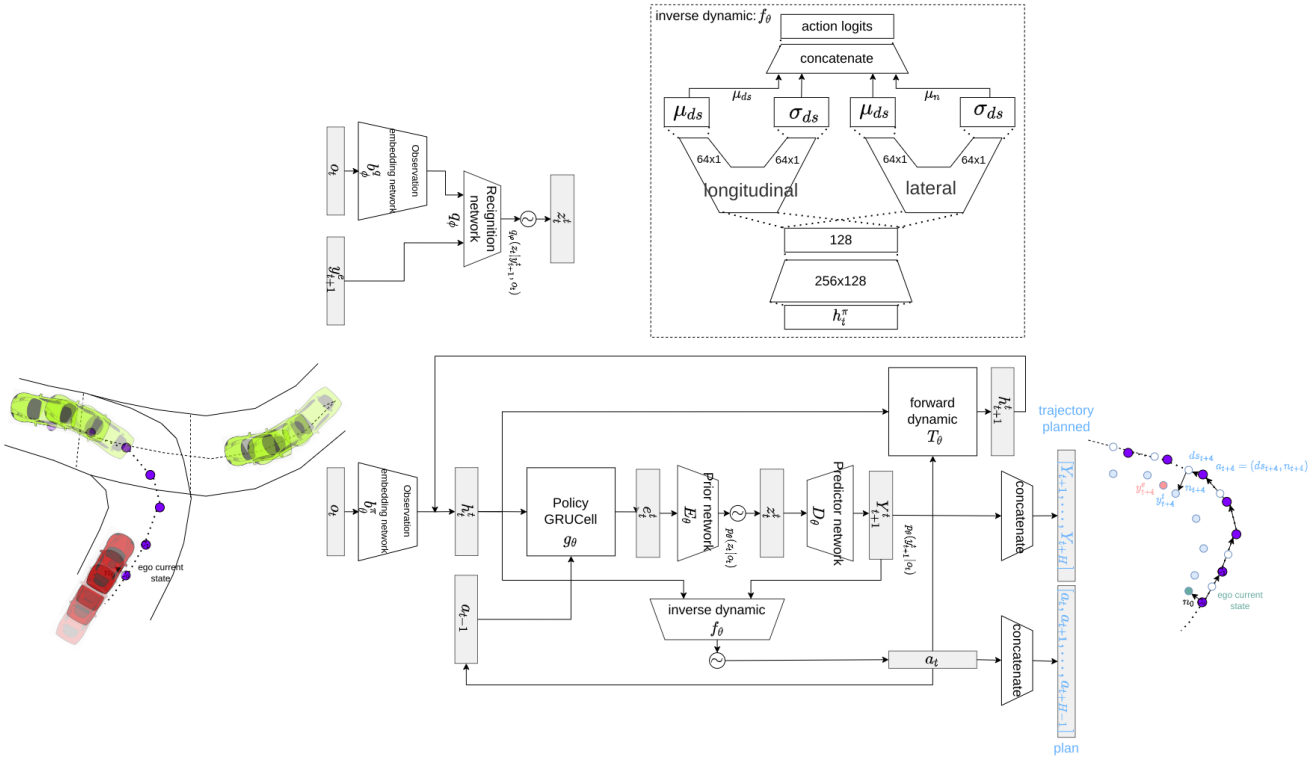

Figure 5.4: Architecture of the Inverse Auto-regressive planner (Autoreg_IDM)

forward model has to implicitly understand how other agents may react to the planned action a_t of the ego agent. The predicted embedding h_{t+1}^t should ideally match the embedding that will be generated by the next observation o_{t+1} at the next decision step : $h_{t+1}^{t+1} = b_\theta^\pi(o_{t+1})$. We propose to train the target position prediction $p_\theta(y_{t+1}^t|o_t)$ with a deep conditional generative model [173]. For a given decision step t, we aim to maximize the conditional log-likelihood $p_\theta(y_{t+1}^t|o_t)$ of the next target position knowing the current observation o_t. In contrast to other works that aims to reproduce dense realistic plans from high dimensional representations [121, 200] with variational inference, we aim to infer a sparse short term goal before taking action. The variational lower bound of the conditional log likelihood is written as follows:

$$log(p_\theta(y_{t+1}^t|o_t)) \geq -KL[(q_\varphi(z_t|y_{t+1}^t, o_t)||p_\theta(z_t|o_t))] + \mathbb{E}_{q_\varphi(z_t|y_{t+1}^t,o_t)}[log(p_\theta(y_{t+1}^t|o_t, z_t)] \quad (5.60)$$

In practice the recognition network $q_\varphi(z_t|y_{t+1}^t, o_t)$ process $h_t^t = b_\theta^\pi(o_t)$ with the same observation backbone used in the planner in order to share representations. This choice is justified by the fact that the VAE encoder is not directly used to extract observation features but rather target features robust to small observation perturbations. The lower bound enables to define the target loss that has to be minimized based on $\beta - VAE$ [68] .

$$\mathcal{L}_{\theta,\varphi}^{CVAE}(y_{t+1}^t, o_t) = \mathbb{E}_{q_\varphi(z_t|y_{t+1}^t,o_t)}[-log(p_\theta(y_{t+1}^t|o_t, z_t)] + \beta.KL[(q_\varphi(z_t|y_{t+1}^t, o_t)||p_\theta(z_t|o_t))] \quad (5.61)$$

$$with\ \beta < 1 \quad (5.62)$$

The expectation is computed with the reparametrization trick : we use L samples to compute $z_t^{(l)} = \mu_\varphi(z_t|y_{t+1}^t, o_t)+\epsilon^{(l)}.\sigma_\varphi(z_t|y_{t+1}^t, o_t)$ where $\epsilon^{(l)} \sim \mathcal{N}(0, I)$. The main drawback of this method is that it requires the following target sequence coming from an expert to train a planner for planning H steps :

$$(o_t^e, y_{t+1}^e, a_t^e, o_{t+1}^e, y_{t+2}^e, a_{t+1}^e, ..., o_{t+H-1}^e, y_{t+H}^e, a_{t+H-1}^e) \tag{5.63}$$

Each sample in the training batch has to store the consecutive H observations $(o_t^e, o_{t+1}^e, ..., o_{t+H-1}^e)$ seen by the expert because the recognition network requires $o_{t+\Delta t}$ for training the planning step $t+\Delta t$. Remember that in chap.3, the auto-regressive planner called $Planner\_ILL\_AUTOREG$ was trained with samples of the following form $(o_t^e, y_{t+1:t+H}^e, a_{t+1:t+H}^e)$ that only contain the current observations which is much easier to store and less computationally intensive. It would have been possible to parameterize the recognition network for multiple planning steps such that it outputs $z_{t:t+H-1}^t$ instead of just z_t^t but this methods makes the lower bound auto-regressive since $z_{t+\Delta t+1}^t$ depends on $z_{t+\Delta t}^t$. Back-propagation is very likely to get unstable with consecutive application of the reparamterization trick for each decision step. Consequently we chose to use the variationnal lower bound $\mathcal{L}_{CVAE}(y_{t+1}^t, o_t, \theta, \varphi)$ to train only the first step of the planner that outputs the first action a_t and its associate target position y_{t+1}^t. In complement to this loss, we keep training the planning loss used in $Planner\_ILL\_AUTOREG$ to learn expert like sequence of actions. Except for the first planning step, we force the latent $z_{t+\Delta t}^t$ to be computed with the mean parametrized by the encoder $(\mu_\theta(e_{t+}^t), (\sigma_\theta(e_{t+\Delta t}^t)) = E_\theta(e_{t+\Delta t}^t)$ instead of being sampled.

$$\mathcal{L}_{planner}(o_t^e, a_t^e, ...a_{t+H-1}^e, \theta) = \mathbb{E}_{o_t \sim \rho^*(o_t), ((a_t^e, ...a_{t+H-1}^e), (y_t^e, ...y_{t+H-1}^e) \sim \pi_e(|, o_t)}[\tag{5.64}$$

$$-log(\pi_{planner}(a_t^e, ...a_{t+H-1}^e|o_t : \theta) + \sum_{k=0}^{H-1}(y_{t+k}^t - y_{t+k}^e)^2] \tag{5.65}$$

In the following, we propose a new training procedure that combines multiple offline and online objectives for training an expert driving policy robust to deviations. Offline objectives are computed on expert data, we optimize $\mathcal{L}_{CVAE}(y_{t+1}^t, o_t, \theta, \varphi)$ such that the agent predicts robustly the first target position y_{t+1}^t as an expert and we also optimize the planning loss to ensure the agent plans a sequence of actions as an expert would do. Since those losses are computed entirely on the expert state distribution they may not be sufficient to improve the robustness to distributional shift. As a complement, we also collect trajectories with behaviour policy that applies the first action of the planner at each decision step and store them in a buffer $\mathcal{B}$. We stop simulation roll-outs when the agent error with respect to expert position $\delta y_t = \|y_t^\pi - y_t^e\|$ gets bigger than a given threshold : the maximum deviation $\delta y_{max} = 3m$. For each step of the trajectory, we compute retrospectively the position effectively reached by the agent denoted y_{t+1}^π after it applied action a_t and we add this information next to the expert target position y_{t+1}^e in each sample stored in $\mathcal{B}$. In order to improve the inverse dynamical model f_θ we leverage

online data stored in $\mathcal{B}$ to improve the inverse inference of the action based on the position really reached y^{π}_{t+1} by applying a^{π}_t from observation o^{π}_t.

$$\mathcal{L}_{inverse}(o^{\pi}_t, a^{\pi}_t, y^{\pi}_{t+1}) = \mathbb{E}_{(o^{\pi}_t, a^{\pi}_t o^{\pi}_{t+1}, y^{e}_{t+1}, y^{\pi}_{t+1})\sim\mathcal{B}}[0.5(a^{\pi}_t - f_\theta(b^{\pi}_\theta(o_t), y^{\pi}_{t+1}))] \tag{5.66}$$

In order to improve the forward dynamical model we reuse the loss introduced in chap.3 that enforces predicted observation embedding $h^t_{t+1} = T_\theta(b^{\pi}_\theta(o^{\pi}_t), a^{\pi}_t)$ and the effective observation embedding $h^{t+1}_{t+1} = b^{\pi}_\theta(o^{\pi}_{t+1})$ to align.

$$\mathcal{L}_{forward}(o^{\pi}_t, a^{\pi}_t, o^{\pi}_{t+1}) = \mathbb{E}_{(o^{\pi}_t, a^{\pi}_t o^{\pi}_{t+1}, y^{e}_{t+1}, y^{\pi}_{t+1})\sim\mathcal{B}}[0.5.(b^{\pi}_\theta(o^{\pi}_{t+1}) - T_\theta(b^{\pi}_\theta(o^{\pi}_t), a^{\pi}_t))^2] \tag{5.67}$$

Algorithm 20 Training procedure of the Inverse Auto-regressive planner

INPUTS:

- $\mathcal{D}$ =$\{(o^e_t, a^e_t, o^e_{t+1} y^e_{t+1})\}$ expert dataset

$\mathcal{B} = \{\}$
for $i = 0, ...N$: **do**
  if $i\%L = 0$ **then**
    collect trajectories $\{\tau_i\}_{i\in I}$ and stop simulations when $\delta y > \delta y_{max} = 2.5m$
    dump $\{\tau_i\}_{i\in I}$ trajectories in buffer: $\mathcal{B} = \mathcal{B}\bigcup\{\tau_i\}_{i\in I}$
    compute first position y^{π}_{t+1}reached by the policy on each trajectory
  end if
  $\mathcal{L}_\theta = 0$
  sample policy transitions $(o^{\pi}_t, a^{\pi}_t o^{\pi}_{t+1}, y^e_{t+1}, y^{\pi}_{t+1})$ in $\mathcal{B}$
  improve target prediction with expert target y^e_{t+1}: $\mathcal{L}_\theta = \mathcal{L}_\theta + \lambda_{target}\mathcal{L}^{target}_\theta(y^e_{t+1}, o^{\pi}_t, \theta, \varphi)$
  improve inverse action extraction : $\mathcal{L}_\theta = \mathcal{L}_\theta + \lambda_{inv}\mathcal{L}_{inverse}(o^{\pi}_t, a^{\pi}_t, y^{\pi}_{t+1}))$
  improve forward model: $\mathcal{L}_\theta = \mathcal{L}_\theta + \lambda_{foward}.\mathcal{L}_{forward}(o^{\pi}_t, a^{\pi}_t, o^{\pi}_{t+1})$
  sample expert transitions : $(o^e_t, a^e_t, ...a^e_{t+H-1}, y^e_{t+1}, ..., y^e_{t+H})$ in $\mathcal{D}$
  imitate expert target prediction:$\mathcal{L}_{\theta,\varphi} = \mathcal{L}_\theta + \lambda_{CVAE}.\mathcal{L}^{CVAE}_\theta(y^e_{t+1}, o^e_t, \theta, \varphi)$
  imitate expert plans beyond the first setp: $\mathcal{L}_{\theta,\varphi} = \mathcal{L}_{\theta,\varphi} + \lambda_{planner}.\mathcal{L}_{planner}(o^e_t, a^e_t, ...a^e_{t+H-1}, \theta)$
  optimize $\mathcal{L}_{\theta,\varphi}$ with respect to θ and φ
end for

To check how much the inverse auto-regressive planner called $Planner\_ILL\_AUTOREG\_INV$ can compensate deviations with respect to expert trajectories we propose the following experiment. We train a driving policy as detailed in alg.20 on the scenario data base called $Huge\_R\_Basic$ detailed in annexes .2 and we analyse the impact of each loss on closed loop test performances and notably imitation errors. We observe in tab.5.7 that only using the planning loss $\mathcal{L}_{planner}$ results to significant deviations with respect to the expert even after 5 seconds. When we add the target prediction loss optimized with a CVAE as in experience 2, we observe general improvements in all imitation metrics but despite good target predictions indicated by low gap δ The gap δ represents the error between the predicted position y^{t+1}_t and position y_{t+1}effectively reached $\delta = |y^{t+1}_t - y_{t+1}|$ we note that actions selected are not always consistent with the target position. In case we add the online loss for the inverse dynamical

		E_0	E_1	E_2	E_3	E_4
Offline Losses						
	$\mathcal{L}_{planner}(o_t^e, a_t^e, ...a_{t+H-1}^e, \theta)$	yes	yes	yes	yes	yes
	$\mathcal{L}_{target-prediction}(y_{t+1}^e, o_t^e, \theta, \varphi)$	no	yes	yes	yes	yes
Online losses						
	$\mathcal{L}_{target-prediction}(y_{t+1}^e, o_t^\pi, \theta, \varphi)$	no	no	yes	yes	yes
	$\mathcal{L}_{inverse}(o_t^\pi, a_t^\pi, y_{t+1}^\pi, \theta)$	no	no	no	yes	yes
	$\mathcal{L}_{forward}(o_t^\pi, a_t^\pi, o_{t+1}^\pi, \theta)$	no	no	no	no	yes

	E_0	E_1	E_2	E_3	E_4
ADE-5	4.82	4.23	3.84	3.44	3.26
ADE-10	6.83	6.40	6.10	5.64	4.63
ADE-15	9.23	9.05	7.83	6.53	5.64

Table 5.7: Influences of offline and online losses on test performances of the inverse auto-regressive planner

model we note significant improvements in ADE-5 and ADE-10 and we observe that actions usually enable to reach the target positions. Finally, we observe marginal improvements when we add the online loss for the forward model. We posit that more advanced prediction models that take into account future trajectories of other agents should be used for training the forward model such that value prediction could be improved for downstream RL trainings.

5.2.1.2 Compensating compounding errors

Adversarial Imitation Learning casts imitation learning as a state action distribution matching between the expert and the RL agent. The adversarial reward can be interpreted as an exploration mechanism for the RL agent rather than the true expert reward. The reward function under the optimally assumption of the discriminator $D^*(s,a) = \frac{\rho_\pi(s,a)}{\rho_\pi(s,a)+\rho_{\pi_e}(s,a)}$ provided by GAIL algorithm [69] expresses as follows:

$$r_{gail}(s,a) = -log(D^*(s,a)) = log(1 + \frac{\rho_{\pi_E}(s,a)}{\rho_\pi(s,a)}) = log(1 + \phi(s,a)) \tag{5.68}$$

Intuitively, r_{gail} encourages the RL agent toward under-visited state-actions, where $\phi(s,a) > 1$, and away from over-visited state-actions, where $\phi(s,a) < 1$. When π_e and π matches exactly, r_{gail} converges to an indicator function for the support of π_e, since $\forall(s,a) \in supp(\pi_e)\, \varphi(s,a) = 1$ and the signal provided by the discriminator $-log(D^*(s,a))$ saturates and cannot be used to guide the learner anymore. Since the policy explores during training it can quickly deviate due to random action sampling and reach regions where the discriminator saturates. The policy gradient computed on those trajectories with rewards saturating to negative values can induce undesirable modifications of the policy weights and considerably limits improvements of the policy. We explained in sec.5.1.1.5 that smoother rewards could be learned based on the 1-Wasserstein distance according to the algorithm called WAIL. The 1-Wasserstein distance allows

to compare the discrepancy between two distributions that have negligible intersections whereas KL based divergences will have troubles because they tend to diverge to infinity on regions not visited by the expert. Since the 1-Wasserstein distance is a proper metric, it is suitable for continuous interpolation of distributions, which is particularly interesting because we want the state-action distribution of the policy to gradually move toward the one of the expert. However learning to quantify deviation for each decision step between expert and policy state action pairs $(s^e, a^e), (s^\pi, a^\pi)$ is not straightforward because the usual distance metric $d((s^e, a^e), (s^\pi, a^\pi))$ on $(\mathcal{S} \times \mathcal{A}, d)$ does not indicate how difficult it will be for the policy to recover an expert like behaviour for future steps. Since we cannot compensate deviations by arbitrary displacements but by transitions that stay close to the expert support it may take several decision steps to recover the expert trajectory and in some cases it may even be impossible. For instance, if at an intersection, the policy does not choose to take the way at the same moment as the expert, it may have to adapt by first giving the way to avoid a collision and then moving forward as the expert would have done. Unfortunately we do not have access to those counterfactual demonstrations to measure how realistic are policy adaptations : for each episode we are forced to imitate the reference expert trajectory. Querying an expert is not a scallable solution and furthermore the expert cannot necessarily recover from arbitrary states. Some works proposed to leverage bissimulation metrics to encourage the agent to choose transitions not too far from the expert support but they assume having access to the rewards in the transitions [31, 215] in contrast to our setting. What primarily matters for practical applications of traffic simulation is to stay sufficiently close to expert trajectory to stay safe in any circumstances. We should not expect the learner to exactly memorize each expert trajectory on each scenario but rather to find transitions that are not unrealistic in expectation. One major limitation of previous WAIL implementations[219, 204] is their dependence on a single binary label to distinguish expert and policy samples which may considerably limit the way the transition features are exploited. A recent work proposed to use an auto encoder to build the discriminator[221]. Auto-Encoding Adversarial Imitation Learning (AEAIL) utilizes the reconstruction error of the auto-encoder as a reward signal. The reward signal based on the reconstruction error significantly retains the information of state-action pairs rather than focusing on the minor differences. The reward tends to get more dense because it is computed based on the full state action reconstruction error instead of just a binary label. Therefore the discriminator gets less overconfident in distinguishing the expert and the generated samples. The method also builds upon the $WAIL$ objective which optimize the reward ϕ as a Kantorovich potentials:

$$sup_{\phi \in \mathcal{L}_1} \mathbb{E}_{(s,a)\sim\rho_{\pi_e}}[\phi(s,a)] - \mathbb{E}_{(s,a)\sim\rho_{\pi}}[\phi(s,a)]$$

This form of the reward signal uses the reconstruction error of an auto-encoder to score the state-action pairs in the trajectories. $\phi(s,a) = \frac{1}{(1+AE_\omega(s,a))}$ where $AE_\omega(s,a) = \|Dec \circ Enc(s,a) - (s,a)\|^2$

$$sup_{\phi \in \mathcal{L}_1} \mathbb{E}_{(s,a)\sim\rho_{\pi_e}}[\frac{1}{(1+AE_\omega(s,a))}] - \mathbb{E}_{(s,a)\sim\rho_{\pi}}[\frac{1}{(1+AE_\omega(s,a))}]$$

The reward $r_w(s,a) = \frac{1}{(1+AE_\omega(s,a))}$ gets high when the reconstruction errors is low which is supposed to happen when state-action pairs belongs to the expert support. However the method does not really scale with high dimensional observation space where numerous components of different types need to be reconstructed. Instead of reconstructing the state action pair we propose to reconstruct the observation embedding as follows:

$$AE_\omega(o_t, o_{t+1}) = \|D_\omega(E_\omega(o_t)) - E_\omega(o_{t+1})\|^2$$

The discriminator is composed of an encoder E_ω that embeds the observation in a latent space and a decoder D_ω that is expected to decode the latent embedding $\hat{z}_{t+1} = D_\omega(E_\omega(o_t))$ of the next observation from the embedding of the current $z_t = E_\omega(o_t)$. In order to enforce the Lipschitz constrain on the discriminator we use a gradient penalty as done in [219]. We expect that using a prediction error in the latent space instead of an explicit reconstruction error in a high dimensional observation space will help the discriminator $D_\omega(o_t, o_{t+1}) = \frac{1}{1+AE_\omega(o_t,o_{t+1})}$ to better guide the policy .

In the following, we investigate how much different implementations of $WAIL$ algorithms could improve long term imitation performances in closed loop compared to $GAIL$. We expect $WAIL$ rewards to provide more informative gradients $\nabla_a r(s,a)$ and $\nabla_s r(s,a)$ when the learning agent leaves the expert state action support which should help to compensate compounding errors. We first try to apply WAIL with a discriminator fed with state action pair as in [219] before trying to use consecutive states (s_t, s_{t+1}) as suggested in [114]. In order to enforce the constrain on WAIL discriminator we implemented two variations : one that use a gradient penalty [57] and one that use spectral normalization [125]. Finally we also evaluated our adaption of the AEAIL algorithm with the latent prediction error. Note that all the baselines are trained on the $Huge\_R\_Basic$ scenario database. We observe in tab.5.8 that WAIL-SN(s,s)

	GAIL	WAIL-GP(s,a)	WAIL-SN(s,a)	WAIL-SN(s,s)	AEAIL
ADE-5	3.6	3.74	3.63	3.53	3.51
ADE-10	4.72	7.31	7.23	6.1	5.34
ADE-15	5.95	8.91	8.89	8.5	7.21

Table 5.8: Comparison of test performances in closed loop of different WAIL implementations

outperforms WAIL-GP(s,a) and WAIL-SN(s,a) notably for long term imitation performances (ADE-10 and ADE-15). We posit that rewarding actions is more complex because actions are implicitly represented with curvilinear coordinated and state-action score also depend on the next state which is not provided in WAIL-GP(s,a) and WAIL-SN(s,a). In contrast, the GAIL baseline that use a standard discriminator successively exploits the action to distinguish expert and policy transitions. It appears that learning a Bernoulli distribution with a GAIL discriminator is less prone to optimization issues than trying to lean a Kantorovitch potential with WAIL algorithm for a distance metric d on $\mathcal{S} \times \mathcal{A}$ which is perhaps not adapted for our problem. Regarding the 1-Lipshitz constrain, we don't observe significant improvements with

the spectral normalization even if the gradient penalty coefficient used in WGAIL(s,a)-GP need to be selected with attention. Concerning our adaptation of the AEAIL algorithm we note that it slightly outperforms all other WAIL algorithms in terms of test performances and gets competitive to GAIL baseline. To better understand how the policy is guided with our AEAIL algorithm, we analysed the reward landscape of AEAIL in situations where the learner deviated from the expert trajectory. We generated synthetic trajectories based on a expert trajectory with some noise injection in the action such that the agent deviates and we analyse how the AEIL rewards react to those perturbations. We created perturbed trajectories with increasingly higher noise injection in the actions which induce important lateral offsets and even a collision as depicted in fig.5.5 We see that AEIL reward globally penalizes all perturbed trajectories with low scores in blue while safe transitions have high score in red and reward variations stay smooth. We observe that the discriminator did not necessarily penalize an agent that goes a bit slower than the expert as long as it does not collide which means that it starts to exploit suitable features to interpret the transition (o_t, o_{t+1}).

5.2.2 Explaining expert driving behaviours

5.2.2.1 Causal confusion

Even if the amount of driving data available is huge, it was shown that test performances of a driving policy in closed loop cannot easily be improved up to an arbitrary level [8]. This issue has deeper origins than standard over-fitting where larger models memorize training data and fail to generalize or optimization difficulties associated with access to more information. To be robust to distributional shift, a policy must exclusively rely on the true causes of expert actions and must ignore spurious correlations between actions and observations features. This issue is called causal confusion and it was shown that it can can occur by simply adding a little bit of additional information to the observation vector [32]. While RL offers a way to conduct interventional queries to alleviate causal confusion by letting the learned model control the system and observe outcomes, Behavioural Clonning cannot and struggles to understand the causal structure of the task [30]. Even if AIL algorithms are more robust to distributionnal shifts, they also suffer from causal confusion with irrelevant decision making in unusual situations. It is particularly the case for high dimensional observations because the discriminator that guides the policy tends to exploit task-irrelevant features (artefacts) which do not provide an informative reward signal, leading to poor imitation performances. A few works [144, 232, 233, 154, 205] have attempted to address the overfitting problem of GAIL but mostly for applications with a single agent in prototypical environments. In the following, we review most interesting approaches and propose some modifications to apply them to our driving task. One way to learn task relevant features is to learn general transferable skills that might help the agent during explorations. [154] proposed to apply reward shaping $\gamma.\phi_\varphi(s') - \phi_\varphi(s)$ with an empowerment-based potential function $\phi(s)$ to guide the agent during training. Intuitively, the empowerment $\phi(s)$ of a state

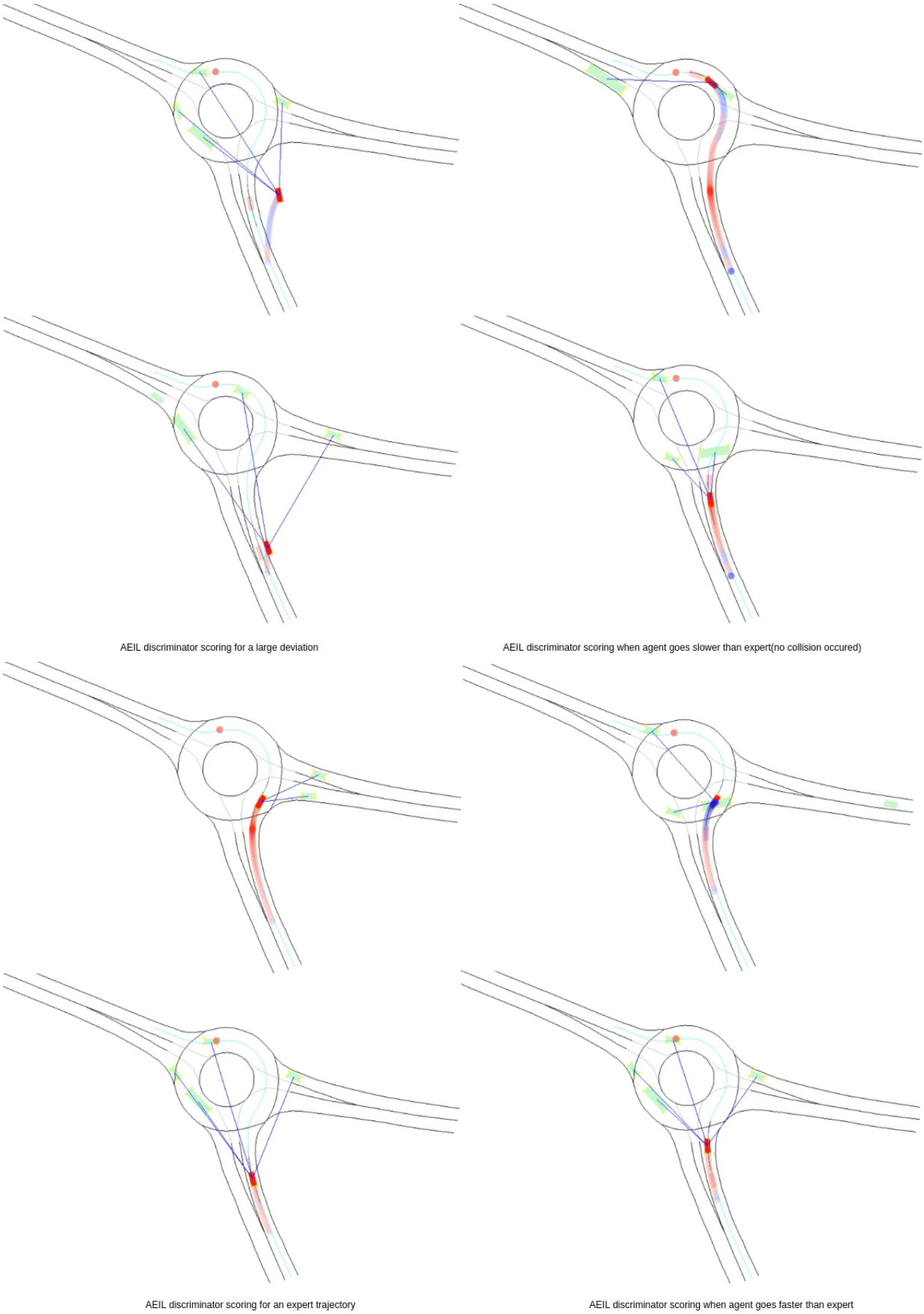

Figure 5.5: Evolution of the AEAIL reward from the expert trajectory toward increasingly more perturbed trajectories.

s quantifies the extent to which an agent can influence its future.

$$\phi(s) = max(I(s', \underline{a}|s)) \tag{5.69}$$

It can be computed based on the mutual information between a sequence of actions $\underline{a}$ and the final state reached s' from state s which can be estimated with variationnal inference [126]. In our setting, the driving policy is already conditioned on a plan which reduces the necessity of learning primary skills like following the road. Additionally, the method does not easily scale for high dimensional setting where the state is not fully controllable [154]. A big picture about AIL algorithms reveals that the drop of policy performances can be explained from the perspective of the discriminator.[232] points out that False negative : realistic transitions generated by the policy can encourage the discriminator to focus on features that are irrelevant for guiding the policy behavior. They propose for instance to filter out some transitions considered as False negative based on prior knowledge of the reward function however the discriminator is not guaranteed to provide better guidance on those transitions. Instead of considering that policy transitions are necessarily unrealistic, [205] proposed to let them unlabeled while expert transitions are labeled as positive because they are assumed optimal. They frame the problems as a positive-unlabeled classification problems, and adapt the empirical risk estimator introduced in [91] to train a discriminator from either expert demonstrations or unlabeled policy samples. However their algorithm requires setting hyper-parmeters that are not stationary during training and hence difficult to determine for large scale experiments (namely the positive class prior η that changes with agents improvements). Instead of considering policy samples as negatively labeled or unlabeled, we can use prior knowledge to refine our judgment. The main problem is that the discriminator ignores driving rules and while policy performances improves it may turn to exploit spurious features instead of task relevant features that become less predictive [232]. As a consequence the discriminator quickly overfitt and the reward saturates preventing the policy from improving. The problem is especially challenging when some observation features are beyond agent control as positions of other agents. Task-Relevant Adversarial Imitation Learning (TRAIL)[233] was designed to prevent the discriminator from forming spurious associations between transition features and labels. The principle consists in regularizing the discriminator on a specific set of transitions (s_t, a_t, s_{t+1}) called a constraining set $\mathcal{I}_e \bigcup \mathcal{I}_\pi$ composed of subsets of expert $\mathcal{I}_e$ and policy $\mathcal{I}_\pi$ subset of transitions. The constraining set is defined based on domain knowledge such that expert and agent transitions contained in $\mathcal{I}_e \bigcup \mathcal{I}_\pi$ can only be identified as belonging to one or the other category using spurious features. Based on the cross entropy objective $G_w((s_e, a_e), (s_\pi, a_\pi)) = log(D_w(s_e, a_e))] + log(1 - D_w(s_\pi, a_\pi))$ used to optimize GAIL discriminator, TRAIL adds a constrain that limits the accuracy of the discriminator on $\mathcal{I}_E \bigcup \mathcal{I}_\pi$ in order to discourages the discriminator from forming spurious associations. Note that the constrain is applied once the discriminator gets overconfident as indicated by

$1_{accuracy(\mathcal{I}_E \bigcup \mathcal{I}_\pi)>0.5}$.

$$\mathcal{L}_D(w) = \mathbb{E}_{(s_e,a_e)\sim\rho_{\pi_e},(s_\pi,a_\pi)\sim\rho_\pi}[G_w((s_e,a_e),(s_\pi,a_\pi))] - 1_{accuracy(\mathcal{I}_E \bigcup \mathcal{I}_\pi)>0.5}. \tag{5.70}$$

$$\mathbb{E}_{(s'_e,a'_e)\sim\mathcal{I}_E,(s'_\pi,a'_\pi)\sim\mathcal{I}_A}[G_w((s'_e,a'_e),(s'_\pi,a'_\pi))] \tag{5.71}$$

We build upon the principle of TRAIL by defining $\mathcal{I}_e \bigcup \mathcal{I}_\pi$ such that all transitions contain no collision, small lateral offsets with respect to the centerline lower than $n_t < n_{max}$ and a distance to neighbors not lower than d_{max}. Additionally, we enforce policy transitions to have a distance error Δd_t with respect to the associate expert demonstration belonging to a fixed range $\Delta d_t \in [\Delta d_{min}, \Delta d_{max}]$. The idea is that those transitions cannot easily be labeled as optimal or unsafe a priori, so the discriminator should become more selective when it exploits features to classify a transition. Similarly to TRAIL, at the beginning of the discriminator update, we evaluate the accuracy of the old discriminator on the new constraining set $\mathcal{I}_e \bigcup \mathcal{I}_\pi$ generated after simulation roll-outs. If the average accuracy exceeds 0.5 then we train the next discriminator with labels set to 0.5 for all transitions contained in $\mathcal{I}_e \bigcup \mathcal{I}_\pi$ while other labels in the buffer remains equal to 1.0 for experts or 0.0 for the policy. This adaptation enables to avoid sampling separate mini-batches from a separate buffer which reduce computational overheads. While the constraining set can be useful at some specific stage of the training it is not guaranteed to help when the policy gets considerably more realistic. Another way to regulate the accuracy of the discriminator without introducing domain knowledge is to constrain the information flow in the discriminator by means of an information bottleneck [144]. Since a discriminator that achieves very high accuracy can produce relatively uninformative rewards (i.e that saturates almost everywhere) while a weak discriminator can also hamper the generator's ability to improve its skills there is a necessity to a find trade-off along the whole training. Starting from the original principle of GAIL, another algorithm called VAIL[144] proposed to structure the discriminator as a variational auto-encoder. An encoder E_w transforms the transition (o_t, a_t, o_{t+1}) into a latent vector z_t distributed as a Gaussian vector $E_w[z|(o] = \mathcal{N}(\mu_E(x), \Sigma_E(x)$ and a decoder D_w outputs the probability that the transition comes from an expert. By enforcing a constraint on the mutual information $I((O_t, A_t, O_{t+1}), Z_t) = D_{KL}(p((O_t, A_t, O_{t+1}), Z_t)|, p((O_t, A_t, O_{t+1}).p(Z_t))$ between the transition and its internal representation z_t, the discriminator enforces the distribution of the latent representation and the distribution of the transitions to stay relatively independent which prevents the discriminator from becoming arbitrarily accurate. Since the mutual information is intractable, an upper bound obtained with variational inference [5] is used to build the objective.

$$max_w \mathbb{E}_{(s,a)\sim\rho_{\pi_e}}[\mathbb{E}_{z\sim E(z|(s,a)}[log(D_w(z))]] - \mathbb{E}_{(s,a)\sim\rho_\pi}[\mathbb{E}_{z\sim E(z|(s,a)}[log(1 - D(z))]] \tag{5.72}$$

$$s.t\ \mathbb{E}_{(s,a)\sim 0.5\rho_{\pi_e}+0.5\rho_\pi}[KL[E(z|(s,a))|r(z)]] < I_C \tag{5.73}$$

$$where\ r(z) = \mathcal{N}(0, I), I_c \leq 1.0 \tag{5.74}$$

To solve this constrained problem, we introduce the Lagrangian and update the Lagrange multipliers β via dual gradient ascent as suggested in [144]. However the choice of the constrain threshold I_c has a strong impact on the performances and it should be determined by hyper-parameter search.
In the following we investigate how the problem of spurious associations in the discriminator network hurts policy test performances. We compare how the test performances of different algorithm discussed above, evolve when we increase the amount of training data. We expect that restricted amount of data severely limits the ability of the policy to generalize while increasing the amount of training data should in theory help. However including more training data can also lead to causal confusions as explained at the beginning of the section and we will analyse how adapted version of TRAIL and VAIL performs with respect to the original GAIL algorithm. For the experiences, we trained GAIL,TRAIL and VAIL on different scenario databases called $Small\_R\_basic$,Big_R_basic and $Huge\_R\_basic$ with increasingly more training scenarios as detailed in annexes.2. We compare the test performances of those algorithms on the test scenario database of $Huge\_R\_basic$ with new driving scenarios in tab.5.9

training scenarios	Small_R_basic			Big_R_basic			Huge_R_basic		
	ADE-5	ADE-15	CR%	ADE-5	ADE-15	CR%	ADE-5	ADE-15	CR%
BC	6.20	15.6	64	5.40	14.1	57	5.20	13.6	55
GAIL	4.2	8.34	40.3	3.96	6.87	36.1	3.6	5.95	33.2
TRAIL	4.85	7.42	37.2	4.61	7.93	26.4	4.2	6.92	18.2
VAIL	4.1	8.12	39.7	3.64	7.34	33.3	2.60	5.34	28.2

Table 5.9: Evolution of test performances of different AIL algorithm when the amount of training data increase.

We observe in tab.5.9 that BC do not understand how to avoid collisions and quickly drifts from the expert trajectory without significant improvements when more training data is provided. The performances of GAIL shows that gradually increasing the amount of data helps to improve test performance and especially short term imitations (ADE-5,ADE-15). The results of TRAIL show improved safety metrics but sightly lower imitations performances. TRAIL discriminator privileges simple observation features to distinguish expert and policy since its accuracy is limited on ambiguous transitions and hence collision are ranked as highly unrealistic while slight speed variations are ignored. We observe that VAIL reached the best performances and this can be explained by the fact that it can gradually regulate the discriminator accuracy in contrast to TRAIL that cannot handle arbitrarily realistic policy. We highlight that those results were obtained with the best hyper-parameters through intensive search.

5.2.2.2 Recovering expert reward function

Learning a policy that imitates expert can be restrictive because driving policies may not easily transfer to new environments with different dynamics especially when other agents are replayed. Ideally we should learn not only a policy but also the associate expert reward function disentangled from the dynamics. The reward could later be used to fine tune the policy on

synthetic scenarios with more interactive agents (Rule based or learned based) in order to improve interactions that are only approximated with replayed agents during training. Learning the reward also enables to analyse how each environment transition (s, a, s') is scored which serves to check that it effectively matches common driving rules as penalizing over-speeding, off road driving or aggressive accelerations. Learning a reward is challenging for several reasons and previous attempts in autonomous driving are restricted to specific settings like highways [203, 178, 73]. Primarily IRL is an ill defined problem since there are many optimal policies that can explain a set of demonstrations so MaxEnt[231] first proposed to learn an expert reward for a policy with maximum entropy which removes this ambiguity. Another problem is that many rewards can explain an optimal policy as shown in [132] and especially rewards shaped by the environment dynamics that cannot easily be transferred at test time when the environment dynamic changes (other agents behaviour change for instance). While GAIL [69] does not attempt to directly recover the expert reward since at optimality, the discriminator is supposed to output 0.5 everywhere on the expert support, a previous work called GAN-GCL proposed a solution inspired from IRL [44] but leveraging an adversarial approach. However their method operates with a GAN at the trajectory level and hence suffered from high variance estimation. Adversarial Inverse Imitation Learning (AIRL)[46], proposed a straightforward extension using only state action pairs to build their discriminator with a specific form:

$$D_w(s, a) = \frac{e^{f_w(s,a)}}{e^{f_w(s,a)} + \pi(a|s)} \tag{5.75}$$

The reward provided to the generator $r_w(s, a) = log(D_w(s, a)) - log(1 - D_w(s, a))$ reduces to $f_w(s, a) - log(\pi(a|s))$ which recovers the entropy-regularised policy objective. It was shown that under the optimal discriminator D^* the reward reduces to the expert advantage $f^*_w(s, a) = A^*(s, a)$ which is yet heavily entangled with the dynamic so this reward won't be robust to changes in environment. According to [46], a reward function $r'(s, a, s')$ is said (perfectly) disentangled with respect to a ground-truth reward $r(s, a, s')$ and a set of dynamics $\mathcal{T}$ if and only if, for all dynamic $T \in \mathcal{T}$, the optimal policy is the same: $\pi^*_{r',T}(a|s) = \pi^*_{r,T}(a|s)$. Consequently AIRL resorts to a technique called reward shaping[132] that states that under the following reward transformation

$$r'(s, a, s') = r(s, a, s') + \gamma\phi(s') - \phi(s) \tag{5.76}$$

the optimal policy remains unchanged, for any function $\phi : \mathcal{S} \to \mathbb{R}$. In order to decouple the reward function from the advantage, AIRL proposed to express the discriminator as follows:

$$D_{w,\varphi}(s, a, s') = \frac{e^{f_{w,\varphi}(s,a,s')}}{e^{f_{w,\varphi}(s,a,s')} + \pi(a|s)} \tag{5.77}$$

where $f_{w,\varphi}(s, a, s') = g_w(s, a) + \gamma.h_\varphi(s') - h_\varphi(s)$ comprises a reward approximator $g_w(s, a)$ and a shaping term h_φ. Note that AIRL algorithm trains the policy and the discriminator the same way as GAIL. Finally it was shown the true reward function r^* can be obtained at optimality,

in case the ground truth reward is state only, the environment is deterministic and additionally satisfies the decomposability condition [52]. The discriminator then reduces to :

$$g^*(s) = r^*(s) + const \tag{5.78}$$

$$h^*(s) = V^*(s) + const \tag{5.79}$$

because $f^*_{w,\varphi}$ should recover the expert advantages. Since the driving scene is far from being deterministic due to other agents behaviour, we cannot fully recover the expert reward in our setting with AIRL. However we choose to investigate to which extent AIRL could effectively recover an interpretable state only reward function.
In the following, we compare the test performances of the state only AIRL(s,s') with respect to a state only WAIL(s,s') baseline which also enables to recover a reward. Since GAIL and AIRL do not optimize the same f-divergence [50] we also compare them to check that AIRL do not obtain lower test performances than GAIL. We trained all algorithms on the scenario database called $Huge\_R\_Basic$ detailed in annexes .2 with horizon curriculum. According to

	ADE-5	ADE-10	Off%	CR%
GAIL	3.6	4.72	4.1	33.2
AIRL(s,s')	3.71	5.31	3.6	37.1
WAIL(s,s')-SN	3.53	6.1	5.2	38.9

Table 5.10: Comparison of best test performances between AIRL,WAIL and GAIL

the test performances in tab.5.10, the state based AIRL is less competitive than GAIL with lower imitation performances. This can be explained by the fact that during discriminator training $D_{w,\varphi}(s,a,s') = \frac{e^{f_{w,\varphi}(s,a,s')}}{e^{f_{w,\varphi}(s,a,s')}+\pi(a|s)}$ tends to saturate to 1.0 because action sampled with low probability during exploration lead to low values for $\pi(a|s)$. As a consequence the guidance provided by the discriminator is hindered by previous policy action probabilities. Note that during discriminator training, the policy parameters are frozen and no gradient is taken in the term $\pi(a|s)$.
We provide qualitative results to interpret the consistency of the state only reward learned by AIRL(s,s') and the state only reward learned WAIL(s,s'). We use stereotypical scenario to point out how much the state only reward is able to guide a policy. We first consider the situation where the last states of the trajectory lead to a collision and we expect the reward to penalize gradually those states such that future training of the policy could avoid it. We also consider the typical case where the policy deviates from the center of the road in a straight line where the reward is also supposed to gradually penalize lateral offsets. Finally, we also propose to analyse how the reward behaves when the agent stay motionless in the middle of the road while there is no neighbor around. The longer the agent stays motionless the more the reward should penalize the agent knowing that the agent has information about its own trajectory history encoded in the observation.

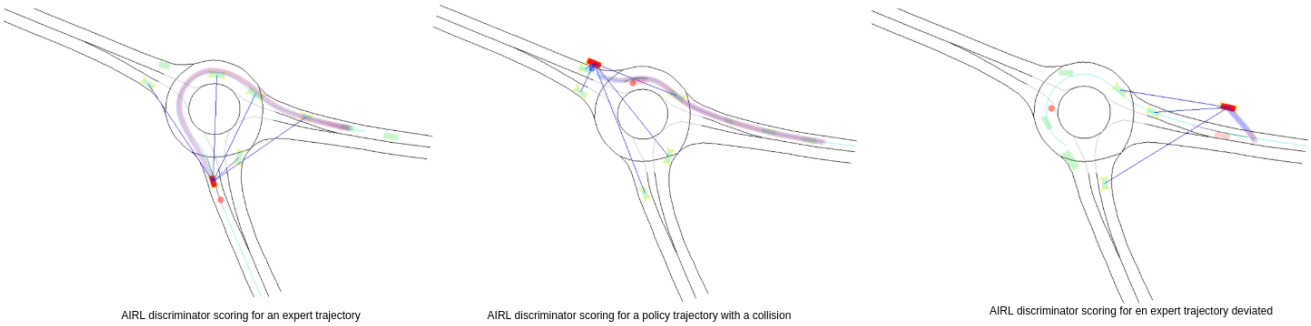

Figure 5.6: AIRL reward landscape for stereotypical scenarios: blue rewards are negative while red rewards are positive. The collision is represented with a dark blue point.

The reward landscape shows that unrealistic behaviours of the learner are conveniently penalized both for the growing lateral offset but not totally for collisions. The collision rate of AIRL given in tab.5.6 reveals that the reward does not sufficiently guide the policy to stay totally safe. We conclude that expert rewards cannot robustly be learnt in replayed driving environments with AIRL which limits the interpretability of the learned driving policies.

5.3 Learning to drive with multiple objectives

We show in the last section that we could learn a driving policy from real demonstrations or from synthetic experiences in chap.4. Therefore we aim to develop a method to learn a policy from multiple experiences that we detail in sec.5.3.1. In a second part sec.5.3.2 we will investigate which experiences are more valuable and how we could combine them to learn robust driving skills.

5.3.1 Multi objective policy optimization

In this section we study how a driving policy can learn simultaneously from multiple objectives such that it can get as safe and realistic as possible. We first review tools of multi task learning that enables to optimize a shared model with multiple targets in sec.5.3.1.1. Subsequently, we explain in sec.5.3.1.2 our own method for training a shared driving policy with reinforcement learning from multiple experiences.

5.3.1.1 Theory of multi task learning

Assuming we have at our disposal several dataset of transitions $\{\mathcal{D}_i\}_{i\in[1,\dots,N]}$ where $\mathcal{D}_i = \{(o_t^j, a_t^j, o_{t+1}^j, r_t^j)\}_{j\in[1,\dots,|\mathcal{D}_i|]}$ each representing different experiences collected by our current policy π_θ. We posit that training our policy jointly on those datasets with different losses enables to

learn more robust features. This formulation reduces to a multi-task learning problem. Multi-task learning exploits similarities between tasks to yield models that are expected to generalize better and require less training data. Several multi task algorithm have been proposed recently UW[86],MGDA[170],PCGrad[212],CAGrad,[113], inspired by various hypotheses about what makes multi-task settings difficult, each motivating a new specialized optimizer. We review the latest approaches and related concepts to motivate our own method detailed in the next section.

Formally, suppose we have K tasks each corresponding to a dataset in $\{\mathcal{D}_i\}_{i\in[1,...,N]}$. A multi task model usually contains two parts of parameters: task-sharing parameters θ_{sh} and task-specific parameters $\{\theta_i\}_{i\in[1,K\}}$. Let $\mathcal{L}_i(\mathcal{D}_i, \theta_{sh}, \theta_i)$, denote per task loss on $\mathcal{D}_i$ with its gradient denoted $\underline{g_i} = \nabla_{\theta_{sh}}\mathcal{L}_i(\mathcal{D}_i, \theta_{sh}, \theta_i)$ then the first question that arise is how to compare a pair of parameters (θ, θ'). Since two multi objective loss vectors $\mathcal{L}(\theta) = (\mathcal{L}_1(\mathcal{D}_i, \theta_{sh}, \theta_1), ..., \mathcal{L}_N(\mathcal{D}_N, \theta_{sh}, \theta_N))$ and $\mathcal{L}(\theta') = (\mathcal{L}_1(\mathcal{D}_i, \theta'_{sh}, \theta'_1), ..., \mathcal{L}_N(\mathcal{D}_N, \theta'_{sh}, \theta'_N))$ can not be ordered we need to introduce some concepts of Pareto optimality to compare parameters (θ, θ').

For two points θ, θ', we say that θ is Pareto dominated by θ', denoted by $L(\theta\prime) \prec L(\theta)$, if $\forall i \in [1, ..., N]\, L_i(\theta\prime) \leq L_i(\theta)$ and $L(\theta\prime) \neq L(\theta)$. A point θ^* is said to be Pareto-optimal if $\forall\theta$ then θ does not dominates θ^*. The set of all Pareto-optimal points θ is called the Pareto set $\mathcal{P}_\theta$ and the image of the Pareto set in the objective function space is called the Pareto front $\mathcal{P}_L = \{\mathcal{L}(\theta)\}_{\theta\in\mathcal{P}_\theta}$. A point θ is called Pareto-stationary if we have $min_{w\in\mathcal{W}} \|g_w(\theta)\| = 0$ where $g_w = \sum_{i=1}^{K} w_i \nabla_\theta L_i(\theta)$ and $\mathcal{W}$ is the probability simplex on $[K]$. A local Pareto optimal point θ is Pareto stationary.

MTL algorithms should ideally lead to a Pareto-optimal point θ for multiple objectives. The Multiple Gradient Descent Algorithm (MGDA)[33] explicitly optimizes towards a Pareto-optimal point and it was shown that a necessary condition for θ to be a Pareto-optimal point is that we could find a convex combination of the task gradients at θ that results in a zeros vector. Therefore, MGDA proposes to minimize the minimum possible convex combination of task gradients:

$$min_{\alpha_1,...,\alpha_T}\{\left\|\sum_{t=1}^{T}\alpha_t.\nabla_{\theta^{sh}}\hat{\mathcal{L}}^t(\theta^{sh},\theta^t)\right\|_2^2 | \sum_{t=}^{T}\alpha_t = 1, \alpha_1, ..., \alpha_T \geq 0\} \tag{5.80}$$

[33] showed that the solution to this optimization problem is either 0 and the resulting point satisfies the Kuhn-Karush-Tucker(KKT) conditions of optimality, or the solution gives a descent direction that improves all tasks. The main limitation of MGDA is the fact that it converges to any point on the Pareto set without explicit control whereas a common objective in MTL is usually to minimize the average loss:

$$\mathcal{L}(\theta) = \sum_{t=1}^{T}\alpha_t.\hat{\mathcal{L}}^t(\theta^{sh},\theta^t) \tag{5.81}$$

where $\alpha_t = \frac{1}{T}$. This gave rise to the development of various methods that directly optimize $\mathcal{L}(\theta)$. Loss balancing methods for instance, studies how to generate appropriate loss weights

$\{\alpha_t\}_{t\in[1,T]}$ in every iteration. Some representative methods focus on using higher loss weights for more difficult tasks measured either by the task uncertainty [86] or measured by the relative loss value [115]. Another line of research named gradient re-balancing, hypothesises that one of the main optimization issues in multi-task learning arises from gradients from different tasks conflicting with one another in a way that is detrimental for making progress. Their analysis assumes that the optimization landscape of each task consists of a deep valley, a property that has been observed in neural network optimization landscapes [54]. Two gradients $\underline{g_i}$ and $\underline{g_j}$ are defined as conflicting when their cosine similarity $cos(\phi_{i,j})$ is negative. The condition at which conflicting gradients can harm the training dynamics are called the tragic triad and implies large differences in gradients magnitudes as well as high curvatures in the multi-task optimization landscape. To illustrate this concept, we borrow the example provided in [212] with a two tasks problem depicted in fig.5.7. We observe that the bottom of the deep valley is characterized by high positive curvature and large differences in the task gradient magnitudes. Under such circumstances, the multi-task gradient is dominated by one task gradient, which comes at the cost of degrading the performances of the other tasks. Furthermore, due to high curvature, the improvement in the dominating task may be overestimated, while the degradation in performance of the non-dominating task may be underestimated. As a result, the ADAM optimizer[90] struggles to make progress on the optimization objective. In order to

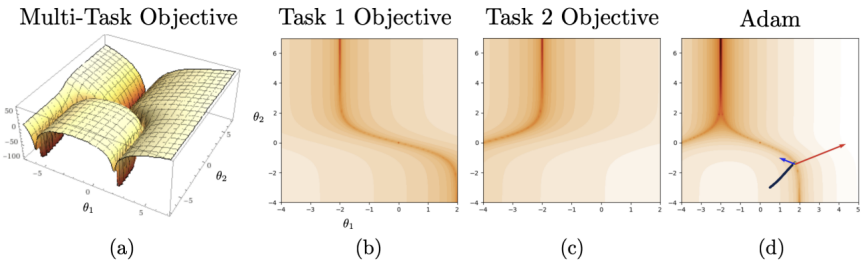

Figure 5.7: Visualization of a multi-task objective landscape. (b) and (c) represent a contour plots of the individual task objectives that compose (a). (d) Trajectory of gradient updates on the multi-task objective using the Adam optimizer. The gradient vectors of the two tasks at the end of the trajectory are indicated by blue and red arrows, where the relative lengths are on a log scale.

break one condition of the tragic triad, PCGrad[212] proposed to directly alter the gradients themselves to prevent conflicts. More specifically, for gradients $\underline{g_i}$ and $\underline{g_j}$ of the i-th and j-th task respectively at a specific training step, PCGrad computes their cosine similarity to determine if they are conflicting, and if the value is negative, it projects g_i onto the normal plane of g_j before combining them together to form the final update vector.:

$$\underline{g}_i' = \underline{g}_i - \frac{\underline{g}_i \cdot \underline{g}_j}{\left\|\underline{g}_j\right\|_2^2} \cdot \underline{g}_j \tag{5.82}$$

The altered gradient $\underline{g_i'}$ replaces the original $\underline{g_i}$ and this whole process is repeated across all tasks in a random order. Note that the gradient cosine similarity will always be zero after the

projection.

$$\underline{g}_i'.\underline{g}_j = (\underline{g}_i - \frac{\underline{g}_i.\underline{g}_j}{\left\|\underline{g}_j\right\|_2^2}.\underline{g}_j).\underline{g}_j = (\underline{g}_i.\underline{g}_j - \frac{\underline{g}_i.\underline{g}_j}{\left\|\underline{g}_j\right\|_2^2}.\left\|\underline{g}_j\right\|_2^2) = 0 \tag{5.83}$$

[212] established the convergence guarantee for PCGrad only under the two-tasks learning setting. Moreover, PCGrad is only guaranteed to converge to the Pareto set without explicit control over which point it will arrive at. This means that the final convergence point of this method may largely depend on the initial model parameters.

This lack of strong convergence guarantee motivated the development of another method called Conflict-Averse Gradient descent (CAGrad)[113], which reduces the conflict among gradients and still provably converges to a minimum of the average loss. The principle is the following : it looks for an update vector that maximizes the worst local improvement of any objectives in a neighborhood of the average gradient. On each optimization step, CaGrads determines the update $\underline{d}$ by solving the following optimization problem:

$$max_{d\in\mathbb{R}^m} min_{i\in[K]} \langle g_i, d\rangle \tag{5.84}$$

$$\left\|d - \underline{g_0}\right\| < c\left\|\underline{g}\right\| \tag{5.85}$$

where $c \in [0, 1[$ is a hyper-parameter that controls the convergence rate. The optimization problem looks for the best update vector $\underline{d}$ within a local ball centered at the averaged gradient $\underline{g_0} = \frac{1}{T}.\sum_{t=1}^{T} \nabla_{\theta^{sh}} \hat{\mathcal{L}}^t(\theta^{sh}, \theta^t)$ while also minimizing the conflict in losses measured by $-min_{i\in[K]} \langle g_i, d\rangle$. It was shown that CAGrad converges to an optimum of $\mathcal{L}(\theta)$, when we choose $c \in [0, 1[$ but instead of solving directly the problem with $\underline{d}$ which has the same dimensionality of the neural network, CaGrad considers the dual problem which only involves solving for a decision variable $\underline{w} \in \mathbb{R}^T$. This enables to leverage standard optimization methods to obtain $\underline{w}$ such that the optimal update direction could be expressed as $d^* = \sum_{t=1}^{T} w_t \nabla_{\theta^{sh}} \hat{\mathcal{L}}^t(\theta^{sh}, \theta^t)$ as detailed in [113]. In the following section we will leverage those multi-task optimizers to train our driving policy with two training objectives jointly.

5.3.1.2 From multi task learning to policy optimization

Domain adaptation in traffic simulation is crucial for practical applications on new driving scenarios. Since we ignore the true expert reward function we can only leverage our domain knowledge and driving demonstrations to learn driving policies. Even if querying an expert could considerably improve the driving policy in critical situations, this solution is not scalable because it would require an expert on arbitrary number of scenarios [145]. In last resort, we could always add a safety shield but this may require at lot of engineering to correct manually unsafe policy decisions[196].

We do not assume having access to the true reward function but only to some proxies associated to our imitation and safety performance metrics. We have at least two reward hypothesis based respectively on domain knowledge and on expert demonstrations. The synthetic reward

based on traffic rules enables to encode traffic rules without ambiguity while the data driven reward induced by demonstrations indicates complex human preferences about various driving aspects. The synthetic reward is supposed to provide minimal guidance for the learner to reach reasonable performances in any situations however it cannot fully explain the expert behaviour. The data driven reward provided by AIL algorithms acts as a guidance toward the expert distribution but this signal may be less informative when the agent goes on states less visited by the expert. It appears that both rewards could complement each other however they may also come into conflict in some cases. The synthetic reward may encourage the agent to go faster than the expert did in the same episode. Similarly, the data driven reward may miss to penalise at the right scale some collisions or large lateral offsets in some unusual situations where the discriminator is not accurate. It is not clear which reward should be privileged during the training because the data driven constantly change which makes difficult to learn preferences on rewards as suggested in [2]. It is also possible to formulate a constrained problem based on a data driven reward to maximize under a set of constrains, leveraging Lagrangian relaxation as proposed in [25]. Constrained reinforcement learning could in theory help to reduce considerably the number of collision in training but at the cost of the data driven reward which may be completely ignored until collisions completely disappear [187]. Instead of enforcing a hard constrain, we can also optimize a soft-robust objective that balances expected performance and a risk measure as proposed by PG-BROIL[77]. The main limitation of this method is that the risk estimation does not scale in high dimensional environment with complex multi agent interactions. The most intuitive solution would be to use a weighted sum of rewards for each transition collected by the policy as suggested in [197]. However the fact that the data driven is constantly changing during training can be problematic.

Assuming that we are using the sum of the reward $r_D + r_S$, then we need to learn the value function of the behaviour policy under the same reward model denoted $V^{\pi}_{r_S+r_d}$. We remind that the value function is necessary to compute advantage estimates [166] for the policy gradient. In practice $V^{\pi}_{r_S+r_d}(s_t = s) = \mathbb{E}_{\tau\sim\pi}[\sum_{l=0}^{\infty} \gamma^l.r_{t+l}|s_t = s]$ is estimated with the trajectory return $G(\tau)$ with bootstrapping at the end of the trajectory $\hat{G}_t(\tau) = \sum_{l=0}^{T-t} \gamma^l r_{t+l} + \gamma^{T+1}.V^{\pi}_{\theta_{k-1}}(s_T)$ for the trajectory $\tau = (s_0, a_0, r_0, s_1, ..., s_{T-1}, a_{T-1}, r_{T-1}, s_T)$ generated by π. The value function is trained to regress the trajectory return by minimizing the following objective :

$$argmin_{\phi}\mathbb{E}_{\tau\sim\pi_\theta}[(V_\phi(s_t) - \hat{G}_t)^2] \tag{5.86}$$

The distribution of the return $G(s_t) = G_s(s_t) + G_d(s_t)$ induced by policy π_θ can be expressed with the synthetic return G_S term and data driven reward G_d. As a consequence the variance of the return:

$$\mathbb{V}(G) = \mathbb{V}(G_s) + \mathbb{V}(G_r) + 2.cov(G_s, G_R) \tag{5.87}$$

may increase[13] due to additional the covariance term $cov(G_s, G_R) = \mathbb{E}_{s\sim\rho^\pi}[(G_s(s)-\mathbb{E}(G_s))(G_r(s)-\mathbb{E}(G_r))]$. The covariance is positive when the differences between the variables (G_s, G_r) and their means tend to be of the same sign according to $cov(G_s, G_R) = \mathbb{E}[(G_s - \mathbb{E}[G_s])(G_r - \mathbb{E}[G_r])]$. Since the two reward signal are rarely in complete contradiction it is reasonable to think that they evolve globally on the same side of their mean so the covariance term can turn positive. At least G_s and G_R cannot be considered as independent since interactions exist between the policy π_θ, the synthetic reward r_s and the data driven reward r_d during training. The policy generates trajectories based on what it learned from previous experiences which are labeled with rewards comprising the synthetic term. Therefore, trajectories generated by the policy π_θ are influenced by the synthetic reward. Those trajectories are later used to update the data driven reward which consequently, gets influenced by the synthetic reward. It is all the more true that this process is repeated for many iteration during the whole training procedure. It appears that the value approximator of $V^\pi_{r_S+r_d}$ may be more difficult to learn accurately than learning separately the two approximators $V^\pi_{r_S}$ and $V^\pi_{r_d}$. This can result in less accurate advantage estimation $A^\pi_{s+r}(a, s)$ which may in turn deteriorate policy updates.

An alternative, would be to leverage all information we have i.e trajectories with synthetic rewards and trajectories with the data driven reward to compute synthetic $A^\pi_s(a, s)$ and data-driven $A^\pi_d(a, s)$ advantages which result to two separate policy objectives $J^{PPO}_s(\theta)$ and $J^{PPO}_d(\theta)$. We propose to train the policy jointly with those two policy objectives based on a multi-task optimizer as detailed detailed in fig.5.8. Since the two objectives $J^{PPO}_s(\theta)$ and $J^{PPO}_d(\theta)$ are computed according to a standard PPO loss that enables to directly update policy parameters, we call our method Multi-Objective Policy Optimization abbreviated (MOPO). During a training iteration of MOPO, the policy first collects a training batch Γ whose transitions are labeled with synthetic rewards r_s. Similarly to GAIL[69], MOPO also learns a data driven reward model from policy samples collected and associate expert demonstrations. Consequently, each transition (s_t, a_t, s_{t+1}) will not only be labeled with a single reward but with two rewards : the synthetic $r_s(s_t, a_t, s_{t+1})$ and the data-driven $r_D(s_t, a_t, s_{t+1})$ rewards. Note that the synthetic reward is the same as the one introduced in chap.4. Subsequently we compute for each transition the two advantage estimators $A^{\pi_{old}}_s(a, s)$ and $A^{\pi_{old}}_d(a, s))$ for respectively the synthetic reward model and the data driven reward model. To this end we also need to train two separate value functions for each reward model V^π_s and V^π_d. In order to update the policy we compute two PPO losses: $J^{PPO}_s(\theta)$ and $\mathcal{J}^{PPO}_r(\theta)$ for a mini batches $\mathcal{B} = \{(s, a, A^{\pi_{old}}_s(a, s), A^{\pi_{old}}_r(a, s))_i\}_{i\in[1,m]}$ sampled in the training batch Γ.

$$J^{PPO}_s(\theta) = \mathbb{E}_{(s,a)\sim\mathcal{B}}[min(\frac{\pi_\theta(.|s)}{\pi_{\theta_{old}}(.|s)}A^{\pi_{old}}_s(a, s), clamp(\frac{\pi_\theta(.|s)}{\pi_{\theta_{old}}(.|s)}, 1-\epsilon, 1+\epsilon).A^{\pi_{old}}_s(a, s)) \quad (5.88)$$

$$J^{PPO}_d(\theta) = \mathbb{E}_{(s,a)\sim\mathcal{B}}[min(\frac{\pi_\theta(.|s)}{\pi_{\theta_{old}}(.|s)}A^{\pi_{old}}_d(a, s), clamp(\frac{\pi_\theta(.|s)}{\pi_{\theta_{old}}(.|s)}, 1-\epsilon, 1+\epsilon).A^{\pi_{old}}_d(a, s)) \quad (5.89)$$

[13]Note that two random variables that are not independent can still have a covariance term equal to 0.

At this level, if we apply a multi-task optimizer on the two policy losses $J_s^{PPO}(\theta)$ and $J_s^{PPO}(\theta)$, we will face a problem because the losses are computed based on the same transitions (s_t, a_t, s_{t+1}). It results that the two gradients computed on the mini-batches $\mathcal{B} = \{(s_t, a_t, A_s^{\pi_{old}}(a,s), A_D^{\pi_{old}}(a,s))\}$ will be co-linear because $\nabla_\theta J_s^{PPO}(\theta) = \nabla_\theta \frac{\pi_\theta(.|s)}{\pi_{\theta_{old}}(.|s)}.A_s^{\pi_{old}}(a,s)$ and $\nabla_\theta J_D^{PPO}(\theta) = \nabla_\theta \frac{\pi_\theta(.|s)}{\pi_{\theta_{old}}(.|s)}.A_D^{\pi_{old}}(a,s)$. Applying the PCGrad[212] optimizer would be equivalent than taking the mean gradient and applying MGDA[170] would result to a zero gradient. It appears that MOPO boils down in the best case to the average loss whose interest depends on the scale of $A_s^{\pi_{old}}(a,s)$ and $A_D^{\pi_{old}}(a,s)$. In order to make the two advantages more comparable, we chose to standardize them with their respective standard deviations. Another solution would have been to use weights (λ_S, λ_D) but we cannot a priory set their values if we ignore their relative importance. Since summing the losses $(J_S^{PPO}(\theta), J_D^{PPO}(\theta))$ is not equivalent to summing the rewards because we learn separate value functions, we chose to analyse this simplified version of MOPO and we call it MOPO Mono-dataset because it collects a single training batch Γ to compute the two policy objectives. In MOPO Mono-dataset we simply sum the losses $J^{PPO}(\theta) = J_S^{PPO}(\theta) + J_D^{PPO}(\theta)$ before computing the policy gradient $\nabla_\theta J(\theta)$. Once the the policy is updated with mini batch SGD, we also update the two value functions with their respective target and we repeat this training procedure as long as test performances improves. While the above algorithm exploits two different reward models, it uses the same environment transitions (s_t, a_t, s_{t+1}) for both reward functions which prevents from obtaining gradients that are not co-linear. Providing more contrastive experiences by not only changing the reward model but also their associate transitions would enable to compute gradients with potentially different directions.

In the following, we propose to collect two training batches Γ_S and Γ_D respectively for the synthetic reward and for the data driven reward based on the same set of driving scenario. This approach is similar to traditional multi-task RL where the policy is conditioned on a task encoding z_i and is trained with multiple losses computed on separate dataset $\{\mathcal{D}_i\}_{i\in\mathcal{T}}$ associated to each task [212]. In our setting, we do not indicate which experience is synthetic and which is real because our goal is to make our policy more robust to changes of dynamics and not to master several tasks. Consequently, the training procedure of the policy is changed. For a given number of epochs we sample a pair of mini-batches $(\mathcal{B}_s, \mathcal{B}_d)$ from the two separate training batches $\Gamma_s \times \Gamma_d$ and we compute for each of them the two PPO losses:

$$J_S^{PPO}(\theta) = \mathbb{E}_{(s,a)\sim B_s}[min(\frac{\pi_\theta(.|s)}{\pi_{\theta_{old}}(.|s)}A_s^{\pi_{old}}(a,s), clamp(\frac{\pi_\theta(.|s)}{\pi_{\theta_{old}}(.|s)}, 1-\epsilon, 1+\epsilon).A_s^{\pi_{old}}(a,s)) \quad (5.90)$$

$$J_D^{PPO}(\theta) = \mathbb{E}_{(s,a)\sim B_d}[min(\frac{\pi_\theta(.|s)}{\pi_{\theta_{old}}(.|s)}A_r^{\pi_{old}}(a,s), clamp(\frac{\pi_\theta(.|s)}{\pi_{\theta_{old}}(.|s)}, 1-\epsilon, 1+\epsilon).A_r^{\pi_{old}}(a,s)) \quad (5.91)$$

Note that the two training batches Γ_S and Γ_D are necessarily different because they are collected while the policy explore: so even if the trajectories are collected on the same scenarios, sampling actions with the stochastic policy for exploration makes each trajectory unique. The gradients $\nabla_\theta J_S^{PPO}(\theta)$ and $\nabla_\theta J_D^{PPO}(\theta)$ will later be combined by the multi task optimizer as

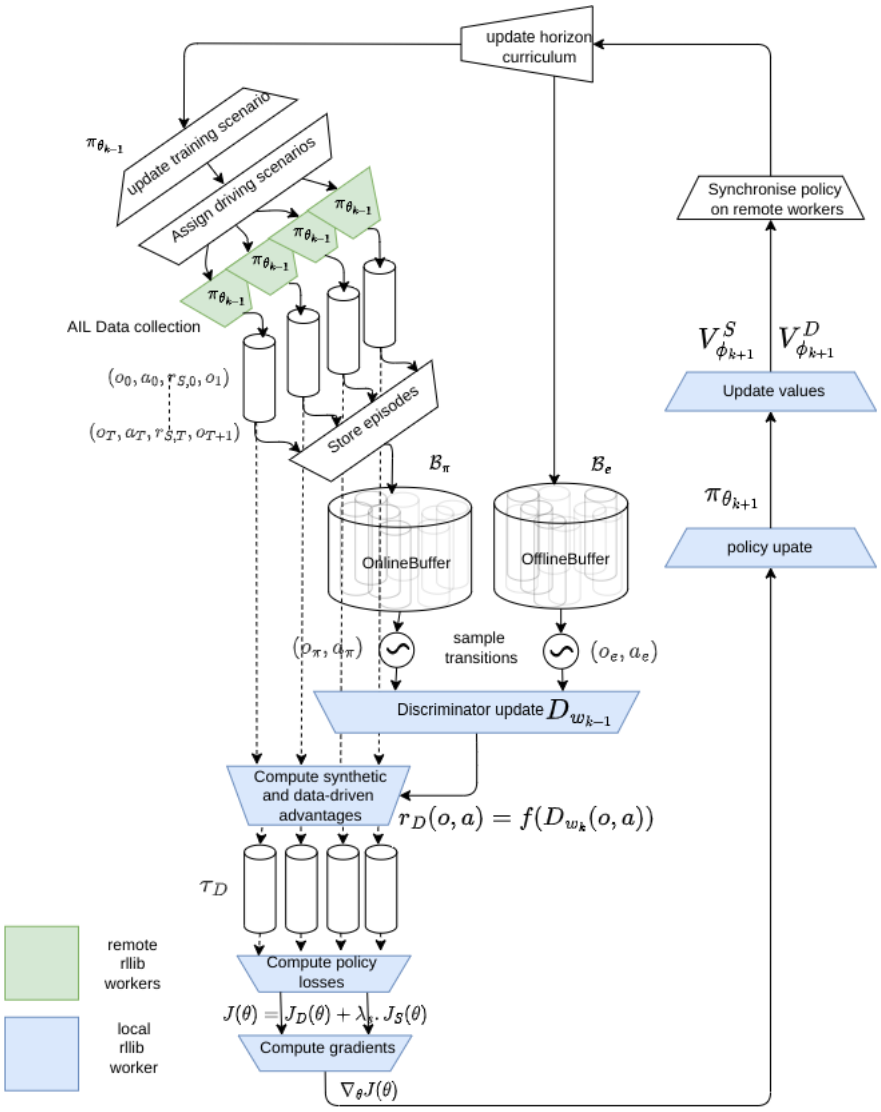

Figure 5.8: Training procedure of MOPO MonoDataset

depicted on fig.5.9. Since the algorithm requires two training batches we call this version MOPO Multi-dataset and we summarize the training procedure in alg.21. The fact that the two gradients $\nabla_\theta J_S^{PPO}(\theta)|_{\theta_k}$ and $\nabla_\theta J_S^{PPO}(\theta)|_{\theta_k}$ are not computed on the same transitions enables to obtain different update directions potentially conflicting at point θ_k. In this case applying a multitask optimizer as PCGrad could enable to avoid detrimental interferences which is expected to help the policy to obtain better performances.

We note that our approach is complementary to the generalist specialist framework proposed in [80]. Their method consists in training a generalist policy on various environments and once it starts to plateaus, specialists policies are initialized with its weights and fine tuned to master a subset of task. Finally, they resume the training of the generalist with auxiliary rewards induced by demonstrations of all specialists and they obtained improved performance for the generalist after several training iterations. Our approach could be used to train a generalist policy and a specialist that both share the same backbone but have separate policy heads. We propose to train the generalist and the specialist jointly with our multi objective approach on

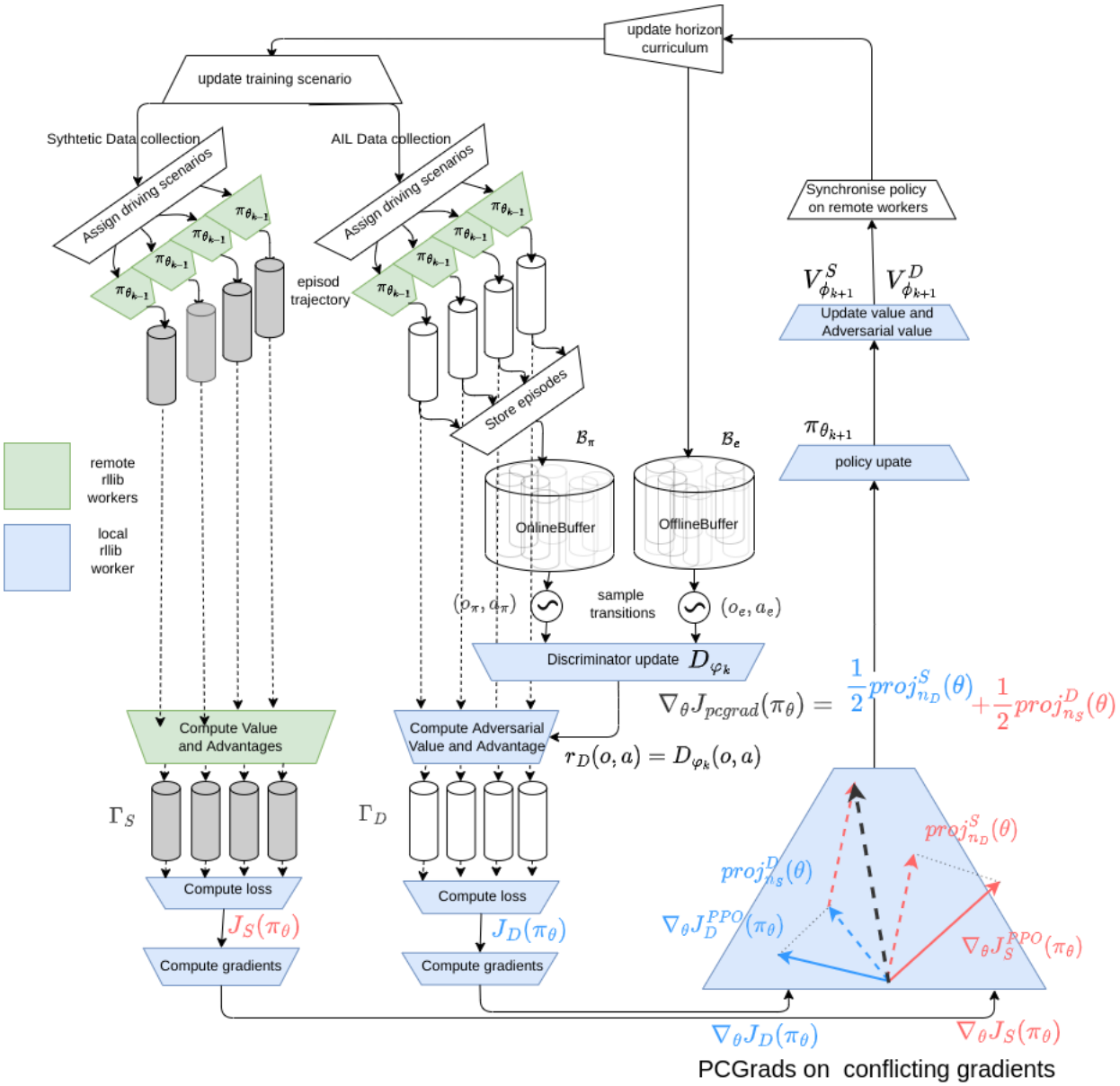

Figure 5.9: Training procedure of MOPO MultiDatasets

different environment. While the generalist is trained on various environments, the specialist is trained on very specific environments where the generalist is currently poorly performing. In the meantime, the specialist continues to generate valuable demonstrations that the generalist can exploit with an auxiliary reward. This technique enables to continuously train the generalist while sharing parameters with a specialist[14].

5.3.2 Learning from multiple experiences

In the following section, we analyse performances of several variations of MOPO. In sec.5.3.2.1 we first examine how using multiple rewards with our multi-objectives algorithm could improve the test performances. In sec.5.3.2.2 we analyse how collecting multiple experiences with various

[14]Multiple specialists can be used but the multi objective optimizer should be selected appropriately

Algorithm 21 Training procedure of MOPO-multi-datasets

INPUTS:

- $\mathcal{S}$: set of training driving scenarios
- π_θ, V_ϕ, D_w :initial policy critic and discriminator
- N_{train}: size of the training batch

while isImproving($\pi_\theta, [m_i]_{i\in I}$) **do**
  $\mathcal{S}_s, \mathcal{S}_d \leftarrow$updateScenarios($\mathcal{S}$,k)
  $\mathcal{D}_s, \mathcal{D}_d \leftarrow$collectDatasets($\mathcal{S}_s, \mathcal{S}_d, \pi_\theta, N_{train}$)
  $\mathcal{B}_\pi, \mathcal{B}_e \leftarrow$UdpateBuffers($\mathcal{B}_\pi, \mathcal{B}_e, \mathcal{D}_d$)
  $D_{w_{k+1}} \leftarrow$trainDiscriminator($\mathcal{B}_\pi, \mathcal{B}_e$)
  $\mathcal{D}_d \leftarrow$computeDataDrivenReward($\mathcal{D}_d, D_{\phi_{k+1}}$)
  $\mathcal{D}_s, \mathcal{D}_d \leftarrow$ComputeAdvantages($\mathcal{D}_s, \mathcal{D}_d$)
  for $i = 1, ..., N_\pi$ **do**
    for (B_s, B_d) sampled in $\mathcal{D}_s \times \mathcal{D}_d$: **do**
      $\nabla_\theta \mathcal{L}^{PPO}(\theta) \leftarrow PCGrad(\nabla_\theta \mathcal{L}_s^{PPO}(\theta), \nabla_\theta \mathcal{L}_d^{PPO}(\theta))$
      $\theta_{k+1} = \theta_k + \alpha.\nabla_\theta \mathcal{L}^{PPO}(\theta)$
    end for
  end for
  $V_{\phi_{k+1}}^S, V_{\phi_k}^D \leftarrow$trainValueFunctions($\mathcal{D}_s, \mathcal{D}_d, V_{\phi_k}^S, V_{\phi_k}^D$)
  if $k\%T_{eval} = 0$ **then**
    m_k=eval(π_θ)
    $k = k + 1$
  end if
end while

environment dynamics could make the policy more robust during test.

5.3.2.1 Learning with multiple rewards

In this section, we will analyse if using jointly a synthetic reward and a data driven reward with MOPO is more efficient than any other baselines. We realised our experiences on a database called *Huge_R_Basic* detailed in annexes.**??** with an environment dynamic composed of replayed workers. We compare performances of $MOPO_{multidataset}$ that use a multi-task optimizer, with respect to three simple baselines named $MOPO_{monodatset}$, $GAIL$ and $GAIL_{aug}$. As a starting point, we choose the simplest multi task optimizer (PCGrad) to implement $MOPO_{multi}$ because we will analyse the impact of the optimizer choice in a second time. We include another baseline called $MOPO_{multidatset-US}$ for Unitary Scalarization (US) and that just consists in taking the gradient of the sum of the two policy objectives: $\nabla_\theta(J_S^{PPO}(\theta) + J_D^{PPO}(\theta))$ to update the policy. This variation is slightly different from $MOPO_{monodataset}$ because the losses are not computed on the same training batches but on two separate training batches labeled respectively with the synthetic reward and the data-driven reward. The standard $GAIL$ implementation use the data driven reward on a single training batch while $GAIL_{aug}$ use the sum of the synthetic reward with the data driven reward on a single training batch[197]. Note

that $GAIL_{aug}$ trains a single value function for the reward sum $V^{\pi}_{r_s+r_D}$ contrary to $MOPO_{mono}$ that trains two separate value functions and sums policy losses computed on the same training batch. We first observe in tab.5.11 that $GAIL_{aug}$ performs better than $GAIL$ especially for

	Training batch	Synthetic experiences	Real experiences	ADE-5	ADE-15	CR%
$MOPO_{multi-PCGrad}$	multi dataset	$\mathcal{D}_s(r_s, \mathcal{T}_{replay})$	$\mathcal{D}_r(r_d, \mathcal{T}_{replay})$	2.70	4.98	16.1
$MOPO_{multi-US}$	multi dataset	$\mathcal{D}_s(r_s, \mathcal{T}_{replay})$	$\mathcal{D}_r(r_d, \mathcal{T}_{replay})$	3.16	5.14	17.2
$MOPO_{mono}$	mono-dataset	$\mathcal{D}_s(r_s, \mathcal{T}_{replay})$	$\mathcal{D}_r(r_d, \mathcal{T}_{replay})$	3.21	5.28	19.7
$GAIL_{aug}$	mono-dataset	None	$\mathcal{D}_r(r_s + r_d, \mathcal{T}_{replay})$	3.36	5.42	21.3
$GAIL_s$	mono-dataset	None	$\mathcal{D}_r(r_d, \mathcal{T}_{replay})$	3.6	5.95	28.2
PPO	mono-dataset	$\mathcal{D}_s(r_s, \mathcal{T}_{replay})$	None	4.80	9.67	17.4

Table 5.11: Test performances comparison between MOPO multidataset-PCGrad with different baselines.

collision avoidance which indicates that the synthetic reward has a positive effects when combined to the data driven reward. As expected, it helps to reduce the number of collision which in turn enables to slightly improve imitation performances. In order to understand if simply summing the reward is a better choice than computing separate advantage estimates as done in $MOPO_{monodataset}$, we compare $GAIL_{aug}$ and $MOPO_{monodataset}$. We note that $MOPO_{monodataset}$ is better than $GAIL_{aug}$ for all performance metrics and especially for imitation performances. To better understand the root causes of the gain of performances we compare to which extent advantage estimation is improved if we use two separate value functions. We compare the bias and the variance of $V^{\pi}_{r_s+r_D}$ against the one of $V^{\pi}_{r_s}$ and $V^{\pi}_{r_D}$ which directly reveals if the advantage is well estimated. We realise that the variance (1.546) but also the bias (0.345) of $V^{\pi}_{r_s+r_D}$ on this experience, is considerably bigger than individual variance (0.241,0.655) and bias (0.015,0.276) for respectively $V^{\pi}_{r_s}$ and $V^{\pi}_{r_s}$[15]. It appears that learning the value function for $GAIL_{aug}$ is harder than learning separate values which can be explained by the covariance term which arise when rewards are summed. The direct consequence is that the $GAIL_{aug}$ policy reached lower performances due to poor advantage estimates compared to $MOPO_{monodataset}$. Finally if we compare $MOPO_{multi-PcGrad}$ against $MOPO_{monodatset}$ or $MOPO_{multi-US}$ we observe that the performances of $MOPO_{multi-PcGrad}$ are better which indicates that using the multi objective optimizer effectively solve conflicts to escape some local minima reached by $MOPO_{monodataset}$ or $MOPO_{multi-US}$. To make a fair comparison, the training batch size of $MOPO_{mono}$ denoted $|\Gamma^{mono}_D|$ is made twice bigger than synthetic or real training batches of $MOPO_{multi}$: $|\Gamma^{mono}_D| = 2.\left|\Gamma^{multi}_D\right| = 2.\left|\Gamma^{multi}_S\right|$ such that $MOPO_{mono}$ and $MOPO_{multi}$ are trained on the same amount of data. In this case, improvements of $MOPO_{multi-PcGrad}$ cannot be attributed to additional data collection but only to the impact of multi task optimizer. Furthermore, the fact that $MOPO_{multi-PCGrad}$ outperforms $MOPO_{multi-US}$ shows that the multi objectives optimizer is more efficient than just applying unitary scalarization on the two policy objectives. It has to be noted that the PPO baseline reached competitive rate of collision with $MOPO_{multi-PCGrad}$ because PPO is focused on improving safety performances while

[15]Remind that the value function is trained to regress the normalized value targets which explains low values for bias and standard deviations.

$MOPO_{multi-PcGrad}$ searches a trade-off between safety and imitation due to the multi-task optimizer. Since imitating the expert and avoiding collisions among replay workers are not necessarily conflicting tasks (It suffices to closely imitate the expert to meet both constrains), $MOPO_{multi-PcGrad}$ finally reached better test performances than other baselines.
After having shown that using a multi task optimizer to update a policy with two sources of rewards provides better performances than other baselines, we examine which multi task optimizer is the most efficient. In the last section we introduced three optimizers: MGDA,PCGrad and CaGrad that we will now use to implement the policy update in $MOPO_{multi}$. We observe

	Optimizer	ADE-5	ADE-15	CR%
$MOPO_{PCGrad}$	PCGRad	2.70	4.98	16.1
$MOPO_{US}$	UnitaryScalarization	3.16	5.14	17.2
$MOPO_{CAGrad}$	CaGrad	2.68	4.95	15.3
$MOPO_{MGDA}$	MGDA	3.46	6.06	19.1

Figure 5.10: Test performances comparison between different multi-objective optimizers.

in fig.5.10 that *CaGrad* reached slightly better results than *PCGrad* which is consistent with experiences realised in [113]. We chose an intermediate value $c = 0.5$ for the CaGrad constrain constant such that it neither reduces to gradient descent neither to MGDA. We also observe that *CAGrad* and *PCGrad* outperform *MGDA* and the unitary scalarization baselines. Since MGDA optimizes the worst local update without explicit constrain on the average loss [113], it may suffer from the fact that the data driven reward is constantly changing. In contrast, it is proved that under mild conditions *CaGrad* and *PCGrad* converge to a local optimum of the average loss when the number of losses is restricted to two. Since PCGrad is much faster in execution in our setting we will continue the next experiences with this optimizer.

5.3.2.2 Learning with agents mixture

To learn a safe driving policy that generalizes how to avoid collisions in new environments, we need to incorporate more realistic interactions in our training batches because other agents populating the scene just replay their trajectories in the previous experiences. While replay agents are necessary to recover an expert driving strategy thanks to AIL algorithms, their benefit are more limited to learn realistic collision avoidance strategies. Since MOPO enables to incorporate synthetic experiences labeled with synthetic rewards, it appears that those synthetic experiences could also include trajectories generated with different environment dynamics. The use of replay workers was justified by the requirements of AIL algorithms whose simulation rollouts should stay as close as possible to the real driving episodes. Since we ignore how to animate other agents of the traffic in a realistic way in new situations we realised that simply replaying other agent trajectories was the best solution to stabilize AIL algorithm. In MOPO, the synthetic training batch is not labelled with the data driven reward but just with the synthetic reward so there is no more restriction on the environment dynamics. To better learn how to

avoid collision with interactive agents we propose to include trajectories generated in presence of interactive workers in the synthetic training batch. We can use centralized and decentralized IDM models introduced in chap.2 that have relatively low collision rates when evaluated with each other. Since we mix experiences generated with different environment dynamics in the synthetic training batch (Non interactive replay agents with Interactive Centralized or Decentralized IDMs), we call this variation $MOPO_{multi-mix}$. We illustrate the differences of $MOPO_{multi-mix}$ during data collection in fig.5.11 where we can see that during synthetic data collection some simulations will collect episodes with interactive workers while other simulations will collect episodes with replay workers according to the scenario specifications. Note that all synthetic episodes contained in the training batch Γ_S are labeled by the same synthetic reward function introduced in chap.4 but since the environment dynamic changes, each trajectory should theoretically be evaluated with a value function associated to the appropriate dynamic : $V_{\phi_k}^{\pi_{\theta_k},\mathcal{T}_{replay}}(s)$ or $V_{\phi_k}^{\pi_{\theta_k},\mathcal{T}_{CIDM}}(s)$. To alleviate the computational load, we use a single value approximator $V_{\phi_k}^{\pi_{\theta_k}}$ and we expect the state value approximation to get more pessimistic which should still be sufficient for advantage estimation[16].

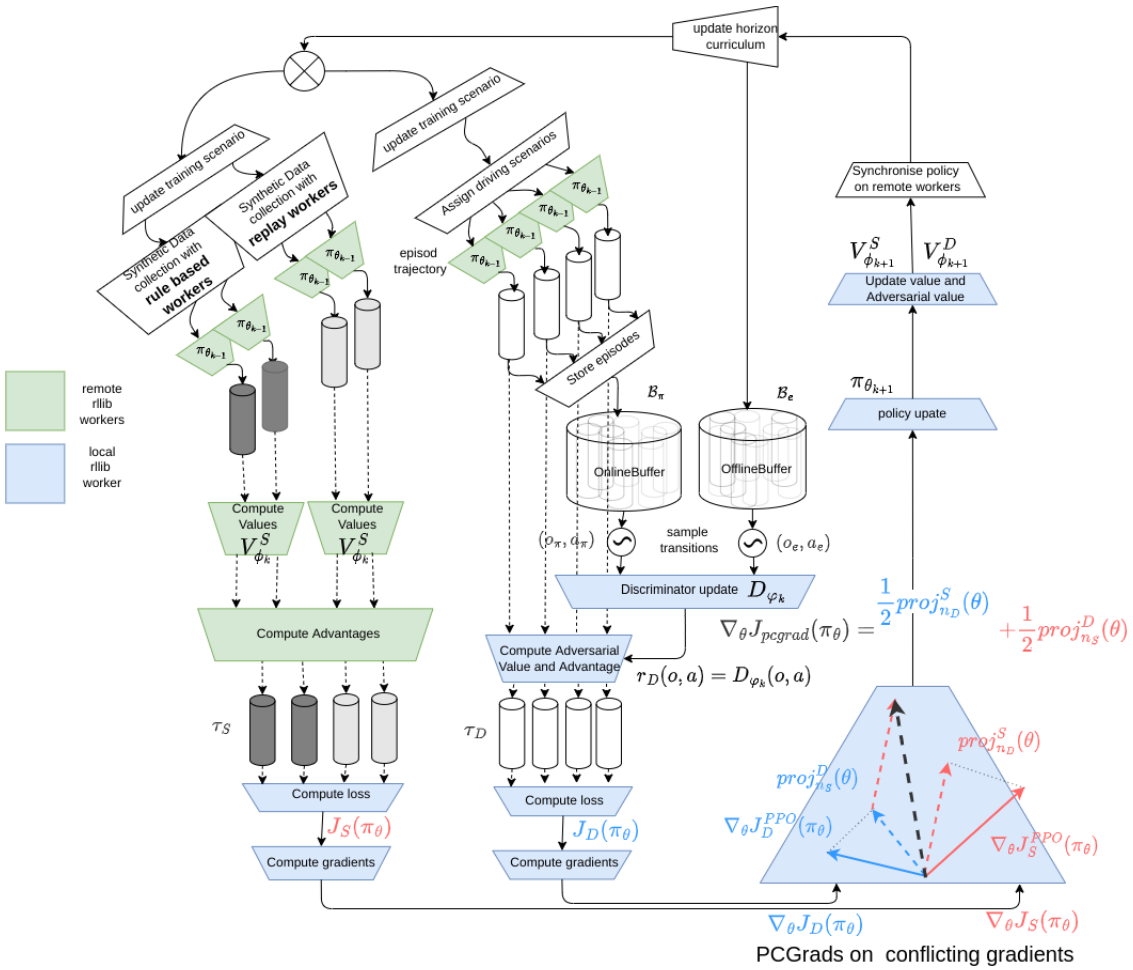

Figure 5.11: Training procedure of $MOPO_{mix}$

[16]In practice we could implement the value loss as follows: $\mathbb{E}_{s\sim\pi}[Relu(V_{target}(s) - V_\phi(s) + (Relu(V_\phi(s) - V_{target}(s)))^2]$ but we found experimentally that the standard loss is sufficient.

In the following, we investigate to which extent synthetic experiences that includes various environment dynamics could improve performances and robustness of our driving policy. Note that in this experience, all driving scenarios are populated with either replay workers or interactive worker :Dencentralized IDM(DCIM) or Centralized IDM(CIDM) but all of them are extracted from real episodes recordings. This means that the agent spawning process as well as the goal assignment are still conform with the real recordings: it is only the behaviour of workers that can change.

In order to measure performances during the test phase, we first created a test dataset composed of 200 real driving scenarios of 15 seconds with replay agents that enables to compute imitation metrics (ADE-5,ADE-15) as well as a rate of episodes with collision in presence of Non Interactive agents denoted CR(NI). In addition we also added the same 200 scenarios but replay workers are replaced with either Centralized IDM(CIDM) or Decentralized workers (DIDM) which enables to compute two associate collision rates CR(DI)% and CR(CI)%. The statistics provided by (CR(NI)%,CR(DI)%,CR(CI)%) show to which extent the driving policy is robust to changes of environment dynamics in terms of collision avoidance. We train all our baselines on 750 real driving scenario $\mathcal{D}_r(r_d, \mathcal{T}_{replay})$ with only replay workers and labeled with the data driven reward r_d and on 750 synthetic scenarios labeled with the synthetic reward r_s that compounds a certain proportion of scenarios with replay workers :α_{replay} CIDM workers :α_{CIDM} or DIDM :α_{DIDM} workers as indicated by $\mathcal{D}_s(r_s, \alpha_{replay}.\mathcal{T}_{replay} + \alpha_{CIDM}.\mathcal{T}_{CIDM} + \alpha_{DIDM}\mathcal{T}_{DIDM})$. The scenario databases used for training and testing are detailed in annexes..2 in paragraph5.3.2.2. We observe in tab.5.12 that using only replay workers does not enable to obtain low collision

	training batch	Synthetic experiences	Real experiences
$MOPO_{multi-unreactive}$	multi-dataset	$\mathcal{D}_s(r_s, \mathcal{T}_{replay})$	$\mathcal{D}_r(r_d, \mathcal{T}_{replay})$
$MOPO_{multi-CIDM}$	multi-dataset	$\mathcal{D}_s(r_s, \frac{1}{2}\mathcal{T}_{replay} + \frac{1}{2}\mathcal{T}_{Inter-centralized})$	$\mathcal{D}_r(r_d, \mathcal{T}_{replay})$
$MOPO_{multi-DIMD}$	multi-dataset	$\mathcal{D}_s(r_s, \frac{1}{2}\mathcal{T}_{replay} + \frac{1}{2}\mathcal{T}_{Inter-decentralized})$	$\mathcal{D}_r(r_d, \mathcal{T}_{replay})$
$MOPO_{multi-mix}$	multi-dataset	$\mathcal{D}_s(r_s, \frac{1}{2}\mathcal{T}_{replay} + \frac{1}{4}\mathcal{T}_{Inter-centralized} + \frac{1}{4}\mathcal{T}_{Inter-decentralized})$	$\mathcal{D}_r(r_d, \mathcal{T}_{replay})$
$MOPO_{multi-Interactive}$	multi-dataset	$\mathcal{D}_s(r_s, \mathcal{T}_{inter-decentralized})$	$\mathcal{D}_r(r_d, \mathcal{T}_{replay})$

	ADE-5	ADE-15	CR(NI)%	CR(DIDM)%	CR(CIDM)%
$MOPO_{multi-unreactive}$	2.70	4.98	16.1	21.2	11.6
$MOPO_{multi-CIDM}$	3.13	5.48	12.2	18.2	4.1
$MOPO_{multi-DIMD}$	2.78	5.24	9.3	7.6	4.3
$MOPO_{multi-mix}$	2.72	5.11	7.1	8.2	3.2
$MOPO_{multi-Interactive}$	2.72	5.17	10.4	7.1	5.9

Figure 5.12: Influence of environment dynamics on test performances of MOPO.

rates in presence of all kind of agents as indicated by the results of $MOPO_{multi-unreactive}$. We see that including trajectories collected in presence of either CIDM workers or DIDM workers enables to improve associate collision rates CR(DI)%,CR(CI)% at the expense of slightly lower imitation performances on real scenarios as indicated by the results of $MOPO_{multi-CIMD}$ and $MOPO_{multi-DIMD}$. In case the synthetic training batch only contains episodes generated with CIDM agents we see that $CR(CI)$ improves a lot at the expense of ADE-5,ADE-15,CR(NI)% and CR(DI)% which means that the policy tends to get overspecialized to another

kind of dynamic. If the synthetic training batch is generated with a full mixture of dynamics $\mathcal{D}_s(r_s, \frac{1}{2}\mathcal{T}_{replay} + \frac{1}{4}\mathcal{T}_{Inter-centralized} + \frac{1}{4}\mathcal{T}_{Inter-decentralized})$ with a majority of trajectories generated in presence of replay workers, we obtain the best trade off as indicated by the results of $MOPO_{multi-mix}$. We observe that $MOPO_{multi-mix}$ reach the best imitation performances and very competitive collision rates compared to the specialized baseline. More importantly $MOPO_{multi-mix}$ clearly outperforms the initial baseline $MOPO_{multi-unreactive}$ that only uses replay workers for training which proves that using diversified experiences with different environment dynamics enables to make the policy more robust in terms of collision avoidance.
In the previous experience, we were restricted to real scenarios Real scenarios have the same initial positions and goal assignments as real episodes recorded in the real world. with very specific agent spawning processes and goal assignments. However those situations do not represent the full spectrum of situations that a basic driving agent is supposed to master. For instance, we do not have driving scenes where the agent is alone and should just reach its goal with almost constant speed. Similarly we do not have an initial state distribution for the ego agent ρ_0 that makes possible to start an episode on every positions of the road network in presence of various density of neighbors. Therefore our driving agent can quickly specialize its driving skills to a restricted set of scenarios that are over-represented in the scenario-database. We propose to include in the set of synthetic scenarios a certain proportion of hand designed scenarios that do not come from the real world. More specifically, we include the following categories of hand crafted scenarios:

1. MonoAgent $\mathcal{S}_{Monoagent}$: the scenario compounds a single agent (actor) which has to join an arbitrary destination on the road-network.

2. MultiAgent $\mathcal{S}_{MultiAgent(N,R)}$: the actor is spawned at an arbitrary yet consistent position on the road and it is surrounded by a fixed number N_a of agents located in a fixed radius R_a around the actor. Each neighbor is assigned a destination and all neighbors initial positions are non overlapping and consistent with the road-network. At the exception of the actor, all other agents are animated by a DIDM model.

In the following, we analyse how the inclusion of handcrafted scenarios can influence the test performance robustness of our driving policy. The test database we use to evaluate our policy in tab.5.12 compounds 200 real scenarios of 15 seconds with replay workers, and the same 200 scenarios with DIDM workers to compute associate collision rate. We also use those 200 test scenarios to evaluate our policy alone in the scene by removing all the traffic. The training database is composed 750 real scenarios with replay workers and 750 synthetic scenarios with a proportions of real world scenario:α_{real} and scenarios of other types as indicated by $\alpha_{real}.\mathcal{S}_{real} + \alpha_{mono}.\mathcal{S}_{Monoagent} + \alpha_{multi}(\mathcal{S}_{MultiAgent(5,3)} + S_{MultiAgent(3,3)} + \mathcal{S}_{MultiAgent(4,3)})$. The scenario database used for training and testing is called $Huge\_R\_Mixed\_H$ and is detailed in annexes..2. We first consider the test results relative to scenarios where the actor is alone and expected to reach its goal without interruption. To check that the policy behaves the

	training batch	Synthetic experiences	type of scenario	Real experiences
$MOPO_{multi-mix-SR}$	multi-dataset	$\mathcal{D}_s(r_s, \frac{1}{2}\mathcal{T}_{replay} + \frac{1}{2}\mathcal{T}_{Inter-decentralized})$	$\mathcal{S}_{real}$	$\mathcal{D}_r(r_d, \mathcal{T}_{replay})$
$MOPO_{multi-mix-SM(1,5)}$	multi-dataset	$\mathcal{D}_s(r_s, \frac{1}{2}\mathcal{T}_{replay} + \frac{1}{2}\mathcal{T}_{Inter-decentralized})$	$\frac{1}{2}\mathcal{S}_{real} + \frac{1}{4}\mathcal{S}_{Monoagent}\frac{1}{4}\mathcal{S}_{MultiAgent(5,3)}$	$\mathcal{D}_r(r_d, \mathcal{T}_{replay})$
$MOPO_{multi-mix-SM(1,5,3,4)}$	multi-dataset	$\mathcal{D}_s(r_s, \frac{1}{2}\mathcal{T}_{replay} + \frac{1}{2}\mathcal{T}_{Inter-decentralized})$	$\frac{1}{2}\mathcal{S}_{real} + \frac{1}{8}\mathcal{S}_{Monoagent} + \frac{1}{8}\mathcal{S}_{MultiAgent(5,3)} + \frac{1}{8}\mathcal{S}_{MultiAgent(3,3)} + \frac{1}{8}\mathcal{S}_{MultiAgent(4,3)}$	$\mathcal{D}_r(r_d, \mathcal{T}_{replay})$

	ADE-5	ADE-15%	CR(NI)%	CR(DI)%	Motionless%
$MOPO_{multi-mix-SR}$	2.78	5.24	9.3	7.6	11.2
$MOPO_{multi-mix-SM(1,5)}$	2.82	5.31	9.1	7.1	0.0
$MOPO_{multi-mix-SM(1,5,3,4)}$	2.86	5.34	8.7	6.2	0.2

Table 5.12: Influence of hand crafted scenarios on test performances of MOPO.

right way, we measure the amount of time it stays motionless during an episode and we consider that it should not exceed 5% of the time to stay realistic. We observe in tab.5.12 that $MOPO_{multi-mix-SR}$ is not always able to reach its destination smoothly: the agent stays in average 11.2 % of the time motionless in an episode while being alone as if neighbors were in the surrounding, blocking its path. In contrast, other versions of $MOPO$ stay less than 5% of the time motionless (sometimes at the level of intersections) and they all reached their destination. Those results show that the driving policy tends to poorly interpret the scene context if it is only trained on real scenarios with numerous neighbors whereas including more diversity at the scenario level helps to disambiguate simple situations.
Secondly, if we consider the collision rate with interactive agents (CR%(DI)), we observe that the more we include diverse scenarios the lower it gets. This prove that domain randomization plays a crucial role to acquire robust driving skills even if it cannot fully address the fundamental problem of causal confusion that prevent from generalizing robust decisions if appropriate features are not used. Lastly we notice that including hand-crafted scenarios in the training set does not significantly deteriorate imitation metrics despite the fact that they are affected. This also reveals the limits of our approach because the environment dynamic induced by replay worker is still very different from the one induced by decentralized IDM workers. As a consequence learning with both dynamic requires to trade off imitation and safety instead of getting cumulative improvements. Another approach such as Robust Reinforcement learning that optimize for the worst case could also help to improve this trade-off[149].

5.4 Conclusion

In this chapter, we study how to leverage expert driving demonstrations to learn realistic driving policies. In order to limit deviations with respect to the expert trajectory we leverage Adversarial Imitation Learning that enables to guide a policy though simulation roll-outs with a data driven reward. We showed that AIL enables to considerably reduce long term imitation errors and we extensively explained how to stabilize the training process which opposes a policy and a discriminator. We notably showed how to better exploit observation features to properly interpret policy environment transitions which revealed critical for constantly improving imitation performances. We also modified our auto-regressive planner in order to limit gradual

deviations with respect to the expert trajectory which resulted to competitive imitation performances. Instead of directly outputting an action, we proposed to first predict a target position before inferring the action in curvilinear coordinates which also enabled to exploit additional simulation data. Finally we proposed a multi objective algorithm called MOPO that combines the bests of AIL and RL in order to control the trade-off between safety and imitation performances. We showed that MOPO can also benefit from synthetic interactions to improve its robustness to environment variations without significantly loosing in imitation performances.

Chapter 6

General conclusion

In this work, we introduced a method to learn realistic and safe driving policies for traffic simulation. We first designed a driving simulator that enables to replay driving episodes extracted from the Interaction Dataset and we developed a salable training framework based on Rllib library which makes possible to simulate driving episodes massively in parallel. In order to animate traffic agents, we proposed a hierarchical driving policy composed of a traditional routing module that generates a traffic free reference path that conditions a maneuver planner implemented with a neural network that can take action in simulation. Actions are specified in curvilinear coordinates with respect to the traffic free path which enables to easily guide the agent during learning. Additionally, we proposed several neural network architectures to encode the local scene context and showed that they enable to learn expert short term plans when we extend decision making for multiple steps with a planner head. Subsequently we improved our maneuver planner architecture such that it can better extract relevant features for long term decision making since closed loop evaluations of the previous planner resulted in poor performances. We proposed an auto-regressive planner augmented with simulation data that better adapts to new situations during closed loop evaluation with compact plans with low jerk. In order to incorporate basic traffic rules to improve safety in closed loop evaluation, we also trained our maneuver planer with Reinforcement Learning(RL) based on a synthetic reward. We show that decoupling the training of the value function and the policy while sharing a common backbone enables to significantly improve test performances. We also noted that including synthetic scenarios with interactive rule based agents enables to make the policy safety more robust to environment variations. Since the maneuver planner trained with RL does not necessarily reproduce expert driving strategies, we leveraged Adversarial Imitation Learning(AIL) to better exploit expert demonstrations. We showed that AIL enables to significantly improve long term imitation performances on various driving scenarios whereas standard imitation learning as behavioural cloning tends to suffer from the distributional shift induced by simulation. We analysed how to stabilize the AIL training by constantly balancing the competition between the discriminator and the policy for maintaining gradual performances

improvements. We also explored another method to reduce deviations with respect to the expert trajectory and found out that predicting a robust target position before inferring the action in curvilinear coordinates with our auto regressive planner augmented with simulation data also led to competitive results. Finally, in order to combine the best of imitation learning and reinforcement learning, we proposed a multi objective formulation of policy optimization. We developed an algorithm called MOPO that enables to combine policy gradients computed respectively on synthetic scenarios with a synthetic reward and on replayed scenarios with a data driven reward learnt with AIL. We showed that using a multi objective optimizer enables to alleviate local conflicts between objectives which led to a better trade off between safety and imitation than other RL and AIL baselines. As a complement, we show that MOPO can obtain balanced imitation and safety performances on a mixture of replayed, synthetic and interactive scenarios while getting more robust to environment variations. As a conclusion, we summarize the main contributions of this work. We first developed a realistic driving simulator that can handle real driving scenes and demonstrations while integrating interactive traffic agents animated with rule based models. With this simulator, we proposed a method to learn humanlike driving policies able to adapt to new situations on several real interactive maps. In order to combine the benefits of domain knowledge and human driving demonstrations, we proposed a multi objective algorithm called MOPO that is able to obtain the best trade off between imitation and safety performances on various scenarios with either replayed traffic agents or interactive traffic agents. We provide an exhaustive comparison in fig.6.1 in terms of imitation and safety performances between all driving policies trained in this work which clearly shows the superiority of our best MOPO variation for all kind of metrics.

Our approach enables to initialize a multi agent traffic simulation with a humanlike driving policy with basic skills but several challenges remain for large scale practical applications. Optimizing imitation and safety metrics with a model free approach is not enough to interpret the outcome of a driving episode. The agent should ideally plan and predict futures states before taking action but standard actor-critic architectures tend to hide this process which makes difficult to understand failures during test. Additionally, existing AIL methods only match expert and policy occupancy measures but do not enable to learn the true expert reward which limits our understanding about expert imitation. Another limitation of our approach is the difficulty to run multi agent simulation from a single agent policy learnt in an another environment dynamic. We showed that our most advanced driving policy can adapt to replay or rule based agents but there is no guarantee that it can adapt to a traffic populated with clones of itself. Consequently, additional multi-agent trainings may be required to properly coordinate agents animated by our policy. Finally, we did not explicitly consider the diversity of traffic users in the demonstrations because we condition our policy on a reference path which considerably reduces the multi-modality of the driving behaviours. However, different driving styles exist in a traffic and should be learned for large scale realistic traffic simulation. As a consequence, future works in traffic simulation may largely benefit from initial driving policies learnt with our method but should focus on the following research directions. Model based approaches that

enable explicit planning look promising for improving the decisions interpretability. Recently, several works[74, 17] proposed to combine AIL methods and tree search algorithms [211] in order to better explore the space of future trajectories. Those methods provides potentially a better policy improvement operator than the policy gradient but stay compatible with our decentralized approach. However, the added value of those methods depend on the access to a realistic and interactive environment which is not possible in practice because the traffic is replayed from logs. Finding a method to relax this strict requirement is a major challenge because contrary to pure control tasks, the dynamic of the traffic is only approximated in the current setting.

A workaround would be to learn all driving polices of each agent in the scene simultaneously as suggested in PS-GAIL[14] but it requires to share parameters among policies and it does not easily converge even to a local optimum. Developing a scalable approach for deploying multi agent imitation leaning on small traffic bubbles before extending it to larger area with complementary MARL technics[96] is a promising research direction but requires considerable engineering work and resources as shown in the SMARTS project[228]. Finally learning diverse driving policies is still an active research direction. While Burn-in GAIL [97] proposed a method to discover latent driving style factors in expert demonstrations, Triple-GAIL [42] proposed a conditional skill selection method, based on multiple expert modalities known a priori. The multi modality of expert driving behaviors can be analysed with mode priors such as level of aggressiveness but more subtle latent factors also come into play such as distractions or stress and are difficult to disentangle from each other. A systematic analysis of the multi modal nature of expert demonstrations is still an open problem.

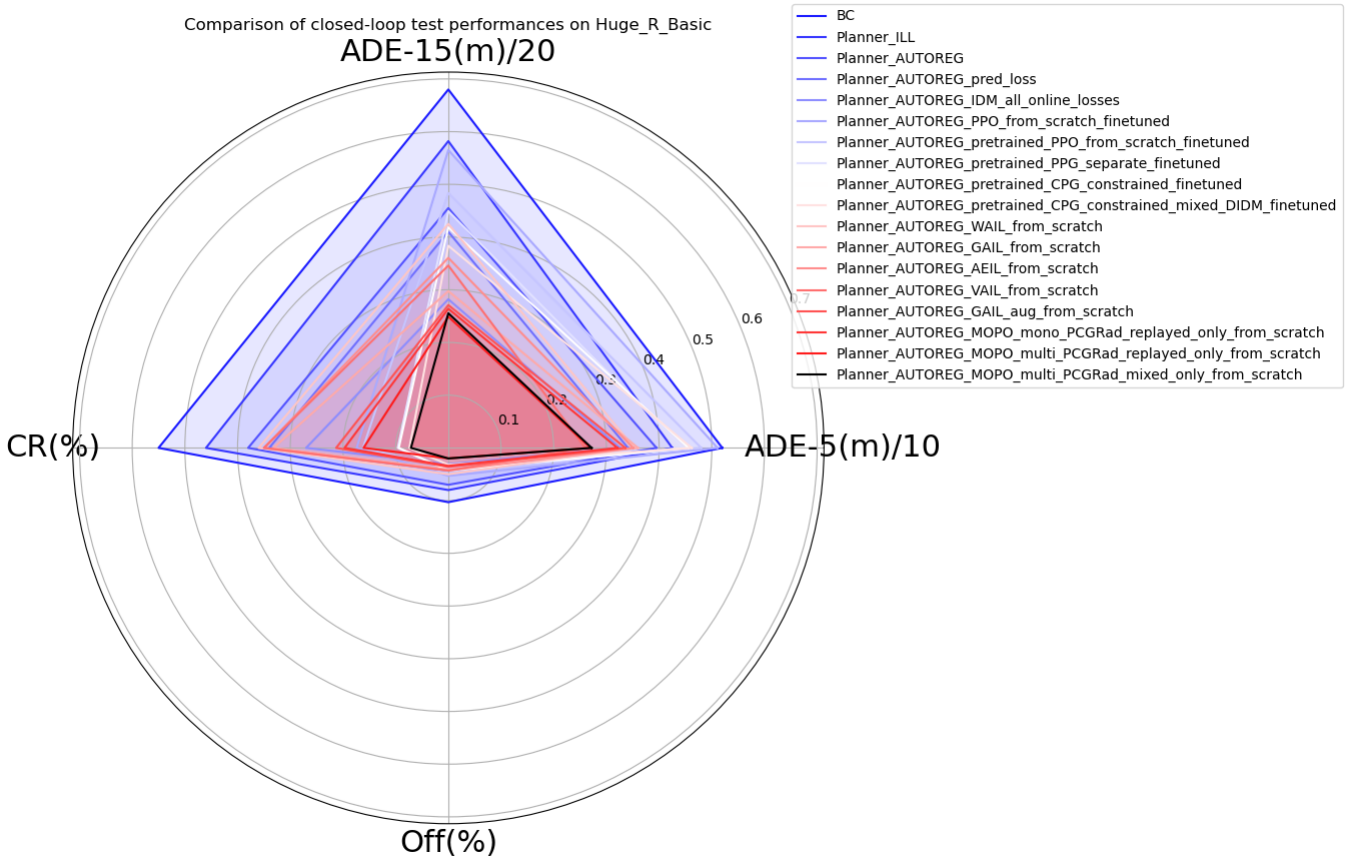

Figure 6.1: Closed loop test performances comparison on Huge_R_Basic with replayed agents between all driving policies trained in this work.

Appendices

.1 Contributions

This work resulted in two publications for international conferences and a validated patent submission.

- Koeberle, Yann, Stefano Sabatini, Dzmitry V. Tsishkou and Christophe Sabourin. “Learning Human-like Driving Policies from Real Interactive Driving Scenes.” International Conference on Informatics in Control, Automation and Robotics (2022).
- Koeberle, Yann, Stefano Sabatini, Dzmitry V. Tsishkou and Christophe Sabourin. "Exploring the trade off between human driving imitation and safety for traffic simulation." 2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC). IEEE, 2022.
- International Patent Application, PCT/EP2021/074878, Title: SIMULATION BASED METHOD AND DATA CENTER TO OBTAIN GEO-FENCED DRIVING POLICY, filed in 2021

.2 Datataset composition

.2.0.1 Huge R basic scenario database

The $Huge\_R\_basic$ databse is generated based on the road map called $DR\_DEU\_Roundabout\_OF$ from the Interaction dataset. The following table specify the composition of the training and testing database. We specify which tracks we used to extract the scenario i.e [0-8], how many time the scenario last in seconds (horizon) based on the associate expert trajectory length. Note that $Huge\_R\_basic$ contains scenarios where traffic workers are replay workers and since scenarios are extracted from real world recordings we also have access to associate demonstrations.

training	number of scenarios	horizon(s)	type of workers	Demonstrations available	tracks
	250	7.5	replay	yes	0-8
	250	10	replay	yes	0-8
	250	15	replay	yes	0-8
evaluation	200	150	replay	yes	9-11

Table 1: Composition of $Huge\_R\_basic$ scenarios database

For experiences on causal confusion, we introduce smaller databases than the $Huge\_R\_basic$ database called respectively $Small\_R\_basic$ database,Big_R_basic database which contains respectively 50 and 150 scenarios for each horizons instead of 250.

.2.0.2 Huge I basic scenario database

Similarly to the $Huge\_R\_basic$ database we also created another scenario database called $Huge\_I\_basic$ based on the intersection map called $DR\_USA\_Intersection\_EP0$.

training	number of scenarios	horizons	type of workers	Demonstrations available	tracks
	250	7.5	replay	yes	0-10
	250	10	replay	yes	0-10
	250	15	replay	yes	0-10
evaluation	200	150	replay	yes	11-14

Table 2: Composition of $Huge\_I\_basic$ scenarios database

.2.0.3 Huge M basic scenario database

Similarly to the $Huge\_R\_basic$ database we also created another scenario database called $Huge\_M\_basic$ based on a map with lane merging called $DR\_DEU\_Merging\_MT$.

training	number of scenarios	horizons	type of workers	Demonstrations available	tracks
	250	7.5	replay	yes	0-8
	250	10	replay	yes	0-8
	250	15	replay	yes	0-8
evaluation	200	150	replay	yes	9-11

Table 3: Composition of$Huge_M_basic$ scenarios database.

.2.0.4 Scenario databases for horizon curriculum

For experiences with horizon curriculum, we created training databases based on 200 driving scenarios of 15 seconds extracted from tracks [0-8] on the roundabout map called $DR\_DEU\_Roundabout\_OF$ from interaction dataset. Depending on the horizon schedule, we subdivide those 200 scenarios, in consecutive chunks of given temporal lengths to build the database. For instance, for the finer curriculum : [2.5,5,7.5,10,12.5,15] we obtain the following scenario database.

training	number of scenarios	horizons	type of workers	Demonstrations available
	1200	2.5	replay	yes
	600	5	replay	yes
	300	7.5	replay	yes
	200	10	replay	yes
	200	12.5	replay	yes
	200	15	replay	yes

Table 4: Composition of $Huge\_I\_basic$ scenarios database

.2.0.5 Huge R mixed scenario database

For experiences of sec.5.3.2.2 we created different scenario databases with a mixture of real scenarios and synthetic scenarios built from from real episodes but where replay agents are replaced either with DIDM workers or CIDM workers. In sec.5.3.2.2, we specified the proportion of synthetic scenarios with replay workers α_{replay}, DIDM workersα_{DIDM}, and CIDM workers α_{CIDM}, we want among our synthetic scenarios. Given those proportions, we can build the scenario databases as follows. Note that we always have $\alpha_{replay} + \alpha_{CIDM} + \alpha_{DIDM} = 1.0$.

training	number of scenarios	horizons	type of workers	Demonstrations available	tracks	type of scenario
	250	7.5	replay	yes	0-8	real
	250	10	replay	yes	0-8	real
	250	15	replay	yes	0-8	real
	α_{replay}.250	7.5	replay	no	0-8	synthetic
	α_{DIDM}.250	7.5	DIDM	no	0-8	synthetic
	α_{CIDM}.250	7.5	CIDM	no	0-8	synthetic
	α_{replay}.250	10.0	replay	no	0-8	synthetic
	α_{DIDM}.250	10.0	DIDM	no	0-8	synthetic
	α_{CIDM}.250	10.0	CIDM	no	0-8	synthetic
	α_{replay}.250	15.0	replay	no	0-8	synthetic
	α_{DIDM}.250	15.0	DIDM	no	0-8	synthetic
	α_{CIDM}.250	15.0	CIDM	no	0-8	synthetic
evaluation	200	150	replay	yes	9-11	real
	200	150	synthetic(CIDM)	no	9-11	synthetic
	200	150	synthetic(DIDM)	no	9-11	synthetic

Table 5: Composition of a generic mixed database with different environment dynamics.

.2.0.6 Huge R mixed H scenario database

For experiences of sec.5.3.2.2 we also created different scenario databases with a mixture of real scenarios and synthetic scenarios where synthetic scenario also contains a certain proportion of hand crafted scenarios : multiagent scenarios $multiagent(N, R)$ where a specific number N of agents surround the ego agent agent at a given distance R , or mono agent scenarios where

the ego agent is alone on the map. Once the proportions of the different categories of synthetic scenarios scenarios are specified, we can build the scenario database as explained in the table bellow. Noe that we always have $\sum_{i=1}^{L} \alpha_{multiagents(N_i,R_i)} + \alpha_{replay} + \alpha_{monoagent} = 1$

training	number of scenarios	horizons	type of workers	Demonstrations available	tracks	type of scenario
	250	7.5	replay	yes	0-8	real
	250	10	replay	yes	0-8	real
	250	15	replay	yes	0-8	real
	α_{replay}.250	7.5	replay	no	0-8	synthetic
	$\alpha_{multiagents(N_1,R_1)}$.250	7.5	DIDM	no	0-8	synthetic
	⋮	⋮	⋮	⋮	⋮	⋮
	$\alpha_{multiagents(N_L,R_L)}$.250	7.5	DIDM	no	0-8	synthetic
	$\alpha_{monoagent}$.250	7.5	none	no	0-8	synthetic
	α_{replay}.250	10	replay	no	0-8	synthetic
	$\alpha_{multiagents(N_1,R_1)}$.250	10	DIDM	no	0-8	synthetic
	⋮	⋮	⋮	⋮	⋮	⋮
	$\alpha_{multiagents(N_L,R_L)}$.250	10	DIDM	no	0-8	synthetic
	$\alpha_{monoagent}$.250	10	none	no	0-8	synthetic
	α_{replay}.250	15	replay	no	0-8	synthetic
	$\alpha_{multiagents(N_1,R_1)}$.250	15	DIDM	no	0-8	synthetic
	⋮	⋮	⋮	⋮	⋮	⋮
	$\alpha_{multiagents(N_L,R_L)}$.250	15	DIDM	no	0-8	synthetic
	$\alpha_{monoagent}$.250	15	none	no	0-8	synthetic
evaluation	200	150	replay	yes	9-11	real
	200	150	synthetic(DIDM)	no	9-11	synthetic
	200	150	none	no	0-8	synthetic

Table 6: Composition of a generic mixed database with different environment dynamics and handcrafted scenarios.

Bibliography

[1] Pieter Abbeel and Andrew Y Ng. "Apprenticeship learning via inverse reinforcement learning". In: *Proceedings of the twenty-first international conference on Machine learning*. 2004, p. 1.

[2] Abbas Abdolmaleki et al. "A distributional view on multi-objective policy optimization". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 11–22.

[3] Rishabh Agarwal et al. "Contrastive behavioral similarity embeddings for generalization in reinforcement learning". In: *arXiv preprint arXiv:2101.05265* (2021).

[4] Mathew Aitchison and Penny Sweetser. "DNA: Proximal Policy Optimization with a Dual Network Architecture". In: *arXiv preprint arXiv:2206.10027* (2022).

[5] Alexander A Alemi et al. "Deep variational information bottleneck". In: *arXiv preprint arXiv:1612.00410* (2016).

[6] Marcin Andrychowicz et al. "What matters in on-policy reinforcement learning? a large-scale empirical study". In: *arXiv preprint arXiv:2006.05990* (2020).

[7] Michael Bain and Claude Sammut. "A Framework for Behavioural Cloning." In: *Machine Intelligence 15*. 1995, pp. 103–129.

[8] Mayank Bansal, Alex Krizhevsky, and Abhijit Ogale. "Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst". In: *arXiv preprint arXiv:1812.03079* (2018).

[9] Amir Beck and Marc Teboulle. "Mirror descent and nonlinear projected subgradient methods for convex optimization". In: *Operations Research Letters* 31.3 (2003), pp. 167–175.

[10] Feryal Behbahani et al. "Learning from demonstration in the wild". In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 775–781.

[11] Michael Behrisch et al. "SUMO–simulation of urban mobility: an overview". In: *Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation*. ThinkMind. 2011.

[12] Christopher Berner et al. "Dota 2 with large scale deep reinforcement learning". In: *arXiv preprint arXiv:1912.06680* (2019).

[13] Julian Bernhard et al. “BARK: Open behavior benchmarking in multi-agent environments”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 6201–6208.

[14] Raunak P Bhattacharyya et al. “Multi-agent imitation learning for driving simulation”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 1534–1539.

[15] Daniel Bogdoll, Stefani Guneshka, and J Marius Zöllner. “One Ontology to Rule Them All: Corner Case Scenarios for Autonomous Driving”. In: *arXiv preprint arXiv:2209.00342* (2022).

[16] Kiante Brantley, Wen Sun, and Mikael Henaff. “Disagreement-regularized imitation learning”. In: *International Conference on Learning Representations*. 2019.

[17] Eli Bronstein et al. “Hierarchical Model-Based Imitation Learning for Planning in Autonomous Driving”. In: *arXiv preprint arXiv:2210.09539* (2022).

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

[19] 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.

[20] Yuning Chai et al. “Multipath: Multiple probabilistic anchor trajectory hypotheses for behavior prediction”. In: *arXiv preprint arXiv:1910.05449* (2019).

[21] Souradip Chakraborty et al. “Dealing with Sparse Rewards in Continuous Control Robotics via Heavy-Tailed Policies”. In: *arXiv preprint arXiv:2206.05652* (2022).

[22] Yash Chandak et al. “Learning action representations for reinforcement learning”. In: *International conference on machine learning*. PMLR. 2019, pp. 941–950.

[23] Minshuo Chen et al. “On computation and generalization of generative adversarial imitation learning”. In: *arXiv preprint arXiv:2001.02792* (2020).

[24] Yun Chen et al. “Geosim: Realistic video simulation via geometry-aware composition for self-driving”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 7230–7240.

[25] Zhihao Cheng et al. “Lagrangian Generative Adversarial Imitation Learning with Safety”. In: (2021).

[26] Kyunghyun Cho et al. “On the properties of neural machine translation: Encoder-decoder approaches”. In: *arXiv preprint arXiv:1409.1259* (2014).

[27] Po-Wei Chou, Daniel Maturana, and Sebastian Scherer. “Improving stochastic policy gradients in continuous control with deep reinforcement learning using the beta distribution”. In: *International conference on machine learning*. PMLR. 2017, pp. 834–843.

[28] Karl Cobbe et al. “Quantifying generalization in reinforcement learning”. In: *International Conference on Machine Learning*. PMLR. 2019, pp. 1282–1289.

[29] Karl W Cobbe et al. “Phasic policy gradient”. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 2020–2027.

[30] Felipe Codevilla et al. “Exploring the limitations of behavior cloning for autonomous driving”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 9329–9338.

[31] Robert Dadashi et al. “Offline reinforcement learning with pseudometric learning”. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 2307–2318.

[32] Pim De Haan, Dinesh Jayaraman, and Sergey Levine. “Causal confusion in imitation learning”. In: *Advances in Neural Information Processing Systems* 32 (2019).

[33] Jean-Antoine Désidéri. “Multiple-gradient descent algorithm (MGDA) for multiobjective optimization”. In: *Comptes Rendus Mathematique* 350.5-6 (2012), pp. 313–318.

[34] Christopher Diehl et al. “UMBRELLA: Uncertainty-Aware Model-Based Offline Reinforcement Learning Leveraging Planning”. In: *arXiv preprint arXiv:2111.11097* (2021).

[35] Murat Dikmen and Catherine M Burns. “Autonomous driving in the real world: Experiences with tesla autopilot and summon”. In: *Proceedings of the 8th international conference on automotive user interfaces and interactive vehicular applications*. 2016, pp. 225–228.

[36] Wenhao Ding et al. “A Survey on Safety-critical Scenario Generation from Methodological Perspective”. In: *arXiv preprint arXiv:2202.02215* (2022).

[37] Andrea Dittadi et al. “The role of pretrained representations for the ood generalization of rl agents”. In: *arXiv preprint arXiv:2107.05686* (2021).

[38] Yan Duan et al. “Benchmarking deep reinforcement learning for continuous control”. In: *International conference on machine learning*. PMLR. 2016, pp. 1329–1338.

[39] Logan Engstrom et al. “Implementation matters in deep policy gradients: A case study on PPO and TRPO”. In: *arXiv preprint arXiv:2005.12729* (2020).

[40] Scott Ettinger et al. “Large scale interactive motion forecasting for autonomous driving: The waymo open motion dataset”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 9710–9719.

[41] Haoyang Fan et al. “Baidu apollo em motion planner”. In: *arXiv preprint arXiv:1807.08048* (2018).

[42] Cong Fei et al. “Triple-GAIL: a multi-modal imitation learning framework with generative adversarial nets”. In: *arXiv preprint arXiv:2005.10622* (2020).

[43] Angelos Filos et al. "Can autonomous vehicles identify, recover from, and adapt to distribution shifts?" In: *International Conference on Machine Learning*. PMLR. 2020, pp. 3145–3153.

[44] Chelsea Finn et al. "A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models". In: *arXiv preprint arXiv:1611.03852* (2016).

[45] Robert M French. "Catastrophic forgetting in connectionist networks". In: *Trends in cognitive sciences* 3.4 (1999), pp. 128–135.

[46] Justin Fu, Katie Luo, and Sergey Levine. "Learning robust rewards with adversarial inverse reinforcement learning". In: *arXiv preprint arXiv:1710.11248* (2017).

[47] Scott Fujimoto, David Meger, and Doina Precup. "Off-policy deep reinforcement learning without exploration". In: *International conference on machine learning*. PMLR. 2019, pp. 2052–2062.

[48] 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.

[49] Matthieu Geist, Bruno Scherrer, and Olivier Pietquin. "A theory of regularized markov decision processes". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 2160–2169.

[50] Seyed Kamyar Seyed Ghasemipour, Richard Zemel, and Shixiang Gu. "A divergence minimization perspective on imitation learning methods". In: *Conference on Robot Learning*. PMLR. 2020, pp. 1259–1277.

[51] Thomas Gilles et al. "Home: Heatmap output for future motion estimation". In: *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*. IEEE. 2021, pp. 500–507.

[52] Adam Gleave and Sam Toyer. "A Primer on Maximum Causal Entropy Inverse Reinforcement Learning". In: *arXiv preprint arXiv:2203.11409* (2022).

[53] David González et al. "A review of motion planning techniques for automated vehicles". In: *IEEE Transactions on intelligent transportation systems* 17.4 (2015), pp. 1135–1145.

[54] Ian J Goodfellow, Oriol Vinyals, and Andrew M Saxe. "Qualitatively characterizing neural network optimization problems". In: *arXiv preprint arXiv:1412.6544* (2014).

[55] Ian J Goodfellow et al. "Generative adversarial networks". In: *arXiv preprint arXiv:1406.2661* (2014).

[56] Ziwei Guan, Tengyu Xu, and Yingbin Liang. "When will generative adversarial imitation learning algorithms attain global convergence". In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2021, pp. 1117–1125.

[57] Ishaan Gulrajani et al. “Improved training of wasserstein gans”. In: *Advances in neural information processing systems* 30 (2017).

[58] Dylan Hadfield-Menell et al. “Inverse reward design”. In: *Advances in neural information processing systems* 30 (2017).

[59] Danijar Hafner et al. “Dream to control: Learning behaviors by latent imagination”. In: *arXiv preprint arXiv:1912.01603* (2019).

[60] Danijar Hafner et al. “Learning latent dynamics for planning from pixels”. In: *International conference on machine learning*. PMLR. 2019, pp. 2555–2565.

[61] Perttu Hämäläinen et al. “PPO-CMA: Proximal policy optimization with covariance matrix adaptation”. In: *2020 IEEE 30th International Workshop on Machine Learning for Signal Processing (MLSP)*. IEEE. 2020, pp. 1–6.

[62] Patrick Hart, Leonard Rychly, and Alois Knoll. “Lane-merging using policy-based reinforcement learning and post-optimization”. In: *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE. 2019, pp. 3176–3181.

[63] Jeffrey Hawke et al. “Urban driving with conditional imitation learning”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 251–257.

[64] Martin Herrmann et al. “Using ontologies for dataset engineering in automotive AI applications”. In: *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2022, pp. 526–531.

[65] Elwan Héry et al. “Map-based curvilinear coordinates for autonomous vehicles”. In: *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2017, pp. 1–7.

[66] Matteo Hessel et al. “Rainbow: Combining improvements in deep reinforcement learning”. In: *Thirty-second AAAI conference on artificial intelligence*. 2018.

[67] Martin Heusel et al. “Gans trained by a two time-scale update rule converge to a local nash equilibrium”. In: *Advances in neural information processing systems* 30 (2017).

[68] Irina Higgins et al. “beta-vae: Learning basic visual concepts with a constrained variational framework”. In: (2016).

[69] Jonathan Ho and Stefano Ermon. “Generative adversarial imitation learning”. In: *Advances in neural information processing systems* 29 (2016).

[70] John Houston et al. “One thousand and one hours: Self-driving motion prediction dataset”. In: *arXiv preprint arXiv:2006.14480* (2020).

[71] Junning Huang et al. “Learning a decision module by imitating driver’s control behaviors”. In: *arXiv preprint arXiv:1912.00191* (2019).

[72] Yu Huang and Yue Chen. "Survey of state-of-art autonomous driving technologies with deep learning". In: *2020 IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. IEEE. 2020, pp. 221–228.

[73] Zhiyu Huang, Jingda Wu, and Chen Lv. "Driving behavior modeling using naturalistic human driving data with inverse reinforcement learning". In: *IEEE Transactions on Intelligent Transportation Systems* (2021).

[74] Maximilian Igl et al. "Symphony: Learning Realistic and Diverse Agents for Autonomous Driving Simulation". In: *arXiv preprint arXiv:2205.03195* (2022).

[75] Shariq Iqbal and Fei Sha. "Actor-attention-critic for multi-agent reinforcement learning". In: *International conference on machine learning*. PMLR. 2019, pp. 2961–2970.

[76] Daniel Jarrett, Alihan Hüyük, and Mihaela Van Der Schaar. "Inverse decision modeling: Learning interpretable representations of behavior". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 4755–4771.

[77] Zaynah Javed et al. "Policy gradient bayesian robust optimization for imitation learning". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 4785–4796.

[78] Rohit Jena, Siddharth Agrawal, and Katia Sycara. "Addressing reward bias in Adversarial Imitation Learning with neutral reward functions". In: *arXiv preprint arXiv:2009.09467* (2020).

[79] Rohit Jena, Changliu Liu, and Katia Sycara. "Augmenting gail with bc for sample efficient imitation learning". In: *arXiv preprint arXiv:2001.07798* (2020).

[80] Zhiwei Jia et al. "Improving Policy Optimization with Generalist-Specialist Learning". In: *International Conference on Machine Learning*. PMLR. 2022, pp. 10104–10119.

[81] Sham Kakade and John Langford. "Approximately optimal approximate reinforcement learning". In: *In Proc. 19th International Conference on Machine Learning*. Citeseer. 2002.

[82] Sham M Kakade. "A natural policy gradient". In: *Advances in neural information processing systems* 14 (2001).

[83] Alexey Kamenev et al. "Predictionnet: Real-time joint probabilistic traffic prediction for planning, control, and simulation". In: *2022 International Conference on Robotics and Automation (ICRA)*. IEEE. 2022, pp. 8936–8942.

[84] Shinpei Kato et al. "An open approach to autonomous vehicles". In: *IEEE Micro* 35.6 (2015), pp. 60–68.

[85] Liyiming Ke et al. "Imitation learning as f-divergence minimization". In: *International Workshop on the Algorithmic Foundations of Robotics*. Springer. 2020, pp. 313–329.

[86] Alex Kendall, Yarin Gal, and Roberto Cipolla. "Multi-task learning using uncertainty to weigh losses for scene geometry and semantics". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 7482–7491.

[87] Arne Kesting, Martin Treiber, and Dirk Helbing. "General lane-changing model MOBIL for car-following models". In: *Transportation Research Record* 1999.1 (2007), pp. 86–94.

[88] Mina Khan et al. "Pretrained encoders are all you need". In: *arXiv preprint arXiv:2106.05139* (2021).

[89] Rahul Kidambi, Jonathan Chang, and Wen Sun. "MobILE: Model-Based Imitation Learning From Observation Alone". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 28598–28611.

[90] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[91] Ryuichi Kiryo et al. "Positive-unlabeled learning with non-negative risk estimator". In: *Advances in neural information processing systems* 30 (2017).

[92] W Bradley Knox et al. "Reward (mis) design for autonomous driving". In: *arXiv preprint arXiv:2104.13906* (2021).

[93] Ilya Kostrikov et al. "Discriminator-actor-critic: Addressing sample inefficiency and reward bias in adversarial imitation learning". In: *arXiv preprint arXiv:1809.02925* (2018).

[94] Parth Kothari et al. "Drivergym: Democratising reinforcement learning for autonomous driving". In: *arXiv preprint arXiv:2111.06889* (2021).

[95] Jakub Grudzien Kuba, Christian Schroeder de Witt, and Jakob Foerster. "Mirror Learning: A Unifying Framework of Policy Optimisation". In: *arXiv preprint arXiv:2201.02373* (2022).

[96] Jakub Grudzien Kuba et al. "Settling the variance of multi-agent policy gradients". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 13458–13470.

[97] Alex Kuefler and Mykel J Kochenderfer. "Burn-in demonstrations for multi-modal imitation learning". In: *arXiv preprint arXiv:1710.05090* (2017).

[98] Alex Kuefler et al. "Imitating driver behavior with generative adversarial networks". In: *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2017, pp. 204–211.

[99] Aviral Kumar et al. "Conservative q-learning for offline reinforcement learning". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 1179–1191.

[100] Marc Lanctot et al. "A unified game-theoretic approach to multiagent reinforcement learning". In: *Advances in neural information processing systems* 30 (2017).

[101] Michael Laskey et al. "Dart: Noise injection for robust imitation learning". In: *Conference on robot learning*. PMLR. 2017, pp. 143–156.

[102] Alex X Lee et al. "Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 741–752.

[103] Kuang-Huei Lee et al. "Predictive information accelerates learning in rl". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 11890–11901.

[104] Joel Z Leibo et al. "Autocurricula and the emergence of innovation from social interaction: A manifesto for multi-agent intelligence research". In: *arXiv preprint arXiv:1903.00742* (2019).

[105] Sergey Levine et al. "Offline reinforcement learning: Tutorial, review, and perspectives on open problems". In: *arXiv preprint arXiv:2005.01643* (2020).

[106] Jinning Li et al. "Hierarchical Planning Through Goal-Conditioned Offline Reinforcement Learning". In: *arXiv preprint arXiv:2205.11790* (2022).

[107] Quanyi Li et al. "Metadrive: Composing diverse driving scenarios for generalizable reinforcement learning". In: *IEEE transactions on pattern analysis and machine intelligence* (2022).

[108] Yunzhu Li, Jiaming Song, and Stefano Ermon. "Infogail: Interpretable imitation learning from visual demonstrations". In: *Advances in Neural Information Processing Systems* 30 (2017).

[109] Eric Liang et al. "RLlib: Abstractions for distributed reinforcement learning". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 3053–3062.

[110] Ming Liang et al. "Learning lane graph representations for motion forecasting". In: *European Conference on Computer Vision*. Springer. 2020, pp. 541–556.

[111] Renjie Liao et al. "Reviving and improving recurrent back-propagation". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 3082–3091.

[112] Michael L Littman. "Markov games as a framework for multi-agent reinforcement learning". In: *Machine learning proceedings 1994*. Elsevier, 1994, pp. 157–163.

[113] Bo Liu et al. "Conflict-averse gradient descent for multi-task learning". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 18878–18890.

[114] Fangchen Liu et al. "State alignment-based imitation learning". In: *arXiv preprint arXiv:1911.10947* (2019).

[115] Liyang Liu et al. "Towards impartial multi-task learning". In: ICLR. 2021.

[116] Minghuan Liu et al. "Plan Your Target and Learn Your Skills: State-Only Imitation Learning via Decoupled Policy Optimization". In: (2021).

[117] Yuqi Liu, Qichao Zhang, and Dongbin Zhao. "A Reinforcement Learning Benchmark for Autonomous Driving in Intersection Scenarios". In: *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE. 2021, pp. 1–8.

[118] Zhuang Liu et al. "Regularization Matters in Policy Optimization–An Empirical Study on Continuous Control". In: *arXiv preprint arXiv:1910.09191* (2019).

[119] Pablo Alvarez Lopez et al. "Microscopic traffic simulation using sumo". In: *2018 21st international conference on intelligent transportation systems (ITSC)*. IEEE. 2018, pp. 2575–2582.

[120] Guannan Lou et al. "An investigation into the state-of-the-practice autonomous driving testing". In: *arXiv e-prints* (2021), arXiv–2106.

[121] Corey Lynch et al. "Learning latent plans from play". In: *Conference on robot learning*. PMLR. 2020, pp. 1113–1132.

[122] Andrey Malinin et al. "Shifts: A dataset of real distributional shift across multiple large-scale tasks". In: *arXiv preprint arXiv:2107.07455* (2021).

[123] Marvin Minsky. "Steps toward artificial intelligence". In: *Proceedings of the IRE* 49.1 (1961), pp. 8–30.

[124] Branka Mirchevska, Moritz Werling, and Joschka Boedecker. "Optimizing Trajectories for Highway Driving with Offline Reinforcement Learning". In: *arXiv preprint arXiv:2203.10949* (2022).

[125] Takeru Miyato et al. "Spectral normalization for generative adversarial networks". In: *arXiv preprint arXiv:1802.05957* (2018).

[126] Shakir Mohamed and Danilo Jimenez Rezende. "Variational information maximisation for intrinsically motivated reinforcement learning". In: *Advances in neural information processing systems* 28 (2015).

[127] Philipp Moritz et al. "Ray: A distributed framework for emerging {AI} applications". In: *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 2018, pp. 561–577.

[128] Alexander Mott et al. "Towards interpretable reinforcement learning using attention augmented agents". In: *Advances in Neural Information Processing Systems* 32 (2019).

[129] Nasik Muhammad Nafi, Raja Farrukh Ali, and William Hsu. "Hyperbolically Discounted Advantage Estimation for Generalization in Reinforcement Learning". In: *Decision Awareness in Reinforcement Learning Workshop at ICML 2022*.

[130] Subramanya Nageshrao, H Eric Tseng, and Dimitar Filev. "Autonomous highway driving using deep reinforcement learning". In: *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*. IEEE. 2019, pp. 2326–2331.

[131] Arkadi Nemirovski. "Efficient methods for large-scale convex optimization problems". In: *Ekonomika i Matematicheskie Metody* 15.1 (1979).

[132] Andrew Y Ng, Daishi Harada, and Stuart Russell. "Policy invariance under reward transformations: Theory and application to reward shaping". In: *Icml*. Vol. 99. 1999, pp. 278–287.

[133] Andrew Y Ng, Stuart Russell, et al. "Algorithms for inverse reinforcement learning." In: *Icml*. Vol. 1. 2000, p. 2.

[134] XuanLong Nguyen, Martin J Wainwright, and Michael I Jordan. "Estimating divergence functionals and the likelihood ratio by convex risk minimization". In: *IEEE Transactions on Information Theory* 56.11 (2010), pp. 5847–5861.

[135] Tianwei Ni et al. "f-irl: Inverse reinforcement learning via state marginal matching". In: *arXiv preprint arXiv:2011.04709* (2020).

[136] Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. "f-gan: Training generative neural samplers using variational divergence minimization". In: *Advances in neural information processing systems* 29 (2016).

[137] Manu Orsini et al. "What matters for adversarial imitation learning?" In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 14656–14668.

[138] Błażej Osiński et al. "CARLA Real Traffic Scenarios–novel training ground and benchmark for autonomous driving". In: *arXiv preprint arXiv:2012.11329* (2020).

[139] Błażej Osiński et al. "Simulation-based reinforcement learning for real-world autonomous driving". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 6411–6418.

[140] Kei Ota, Devesh K Jha, and Asako Kanezaki. "Training larger networks for deep reinforcement learning". In: *arXiv preprint arXiv:2102.07920* (2021).

[141] Charles Packer et al. "Assessing generalization in deep reinforcement learning". In: *arXiv preprint arXiv:1810.12282* (2018).

[142] Georgios Papoudakis et al. "Dealing with non-stationarity in multi-agent deep reinforcement learning". In: *arXiv preprint arXiv:1906.04737* (2019).

[143] Fabio Pardo et al. "Time limits in reinforcement learning". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 4045–4054.

[144] Xue Bin Peng et al. "Variational discriminator bottleneck: Improving imitation learning, inverse rl, and gans by constraining information flow". In: *arXiv preprint arXiv:1810.00821* (2018).

[145] Zhenghao Peng et al. "Safe driving via expert guided policy optimization". In: *Conference on Robot Learning*. PMLR. 2022, pp. 1554–1563.

[146] Gabriel Peyré, Marco Cuturi, et al. "Computational optimal transport". In: *Center for Research in Economics and Statistics Working Papers* 2017-86 (2017).

[147] Tung Phan-Minh et al. "Driving in Real Life with Inverse Reinforcement Learning". In: *arXiv preprint arXiv:2206.03004* (2022).

[148] Thomas Pierrot, Nicolas Perrin-Gilbert, and Olivier Sigaud. "First-order and second-order variants of the gradient descent in a unified framework". In: *International Conference on Artificial Neural Networks*. Springer. 2021, pp. 197–208.

[149] Lerrel Pinto et al. "Robust adversarial reinforcement learning". In: *International Conference on Machine Learning*. PMLR. 2017, pp. 2817–2826.

[150] Fabian Poggenhans et al. "Lanelet2: A high-definition map framework for the future of automated driving". In: *2018 21st international conference on intelligent transportation systems (ITSC)*. IEEE. 2018, pp. 1672–1679.

[151] Aditya Prakash, Kashyap Chitta, and Andreas Geiger. "Multi-modal fusion transformer for end-to-end autonomous driving". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 7077–7087.

[152] Charles R Qi et al. "Pointnet: Deep learning on point sets for 3d classification and segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 652–660.

[153] Xiangjun Qian et al. "Decentralized model predictive control for smooth coordination of automated vehicles at intersection". In: *2015 European control conference (ECC)*. IEEE. 2015, pp. 3452–3458.

[154] Ahmed H Qureshi, Byron Boots, and Michael C Yip. "Adversarial imitation via variational inverse reinforcement learning". In: *arXiv preprint arXiv:1809.06404* (2018).

[155] Aravind Rajeswaran et al. "Epopt: Learning robust neural network policies using model ensembles". In: *arXiv preprint arXiv:1610.01283* (2016).

[156] Tabish Rashid et al. "Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning". In: *International conference on machine learning*. PMLR. 2018, pp. 4295–4304.

[157] Paria Rashidinejad et al. "Bridging offline reinforcement learning and imitation learning: A tale of pessimism". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 11702–11716.

[158] Nicholas Rhinehart, Rowan McAllister, and Sergey Levine. "Deep imitative models for flexible inference, planning, and control". In: *arXiv preprint arXiv:1810.06544* (2018).

[159] Nicholas Rhinehart et al. "Precog: Prediction conditioned on goals in visual multi-agent settings". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 2821–2830.

[160] David Rolnick et al. "Experience replay for continual learning". In: *Advances in Neural Information Processing Systems* 32 (2019).

[161] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. "A reduction of imitation learning and structured prediction to no-regret online learning". In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2011, pp. 627–635.

[162] Tim GJ Rudner et al. "On pathologies in KL-regularized reinforcement learning from expert demonstrations". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 28376–28389.

[163] Abbas Sadat et al. "Perceive, predict, and plan: Safe motion planning through interpretable semantic representations". In: *European Conference on Computer Vision*. Springer. 2020, pp. 414–430.

[164] Tim Salzmann et al. "Trajectron++: Dynamically-feasible trajectory forecasting with heterogeneous data". In: *European Conference on Computer Vision*. Springer. 2020, pp. 683–700.

[165] Oliver Scheel et al. "Urban driver: Learning to drive from real-world demonstrations using policy gradients". In: *Conference on Robot Learning*. PMLR. 2022, pp. 718–728.

[166] John Schulman et al. "High-dimensional continuous control using generalized advantage estimation". In: *arXiv preprint arXiv:1506.02438* (2015).

[167] John Schulman et al. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[168] John Schulman et al. "Trust region policy optimization". In: *International conference on machine learning*. PMLR. 2015, pp. 1889–1897.

[169] Max Schwarzer et al. "Pretraining representations for data-efficient reinforcement learning". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 12686–12699.

[170] Ozan Sener and Vladlen Koltun. "Multi-task learning as multi-objective optimization". In: *Advances in neural information processing systems* 31 (2018).

[171] Wenling Shang et al. "Agent-centric representations for multi-agent reinforcement learning". In: *arXiv preprint arXiv:2104.09402* (2021).

[172] Lior Shani, Tom Zahavy, and Shie Mannor. "Online apprenticeship learning". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 36. 8. 2022, pp. 8240–8248.

[173] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. "Learning structured output representation using deep conditional generative models". In: *Advances in neural information processing systems* 28 (2015).

[174] Haoran Song et al. "Learning to predict vehicle trajectories with model-based planning". In: *Conference on Robot Learning*. PMLR. 2022, pp. 1035–1045.

[175] Jiaming Song et al. "Multi-agent generative adversarial imitation learning". In: *Advances in neural information processing systems* 31 (2018).

[176] Jonathan Spencer et al. "Feedback in imitation learning: The three regimes of covariate shift". In: *arXiv preprint arXiv:2102.02872* (2021).

[177] Adam Stooke et al. "Decoupling representation learning from reinforcement learning". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 9870–9879.

[178] Liting Sun, Wei Zhan, and Masayoshi Tomizuka. "Probabilistic prediction of interactive driving behavior via hierarchical inverse reinforcement learning". In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2018, pp. 2111–2117.

[179] Mingfei Sun et al. "You May Not Need Ratio Clipping in PPO". In: *arXiv preprint arXiv:2202.00079* (2022).

[180] Simon Suo et al. "Trafficsim: Learning to simulate realistic multi-agent behaviors". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 10400–10409.

[181] Richard S Sutton. "Learning to predict by the methods of temporal differences". In: *Machine learning* 3.1 (1988), pp. 9–44.

[182] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[183] Richard S Sutton et al. "Policy gradient methods for reinforcement learning with function approximation". In: *Advances in neural information processing systems* 12 (1999).

[184] Umar Syed, Michael Bowling, and Robert E Schapire. "Apprenticeship learning using linear programming". In: *Proceedings of the 25th international conference on Machine learning*. 2008, pp. 1032–1039.

[185] Ardi Tampuu et al. "A survey of end-to-end driving: Architectures and training methods". In: *IEEE Transactions on Neural Networks and Learning Systems* (2020).

[186] Yunhao Tang et al. "Taylor expansion of discount factors". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 10130–10140.

[187] Chen Tessler, Daniel J Mankowitz, and Shie Mannor. "Reward constrained policy optimization". In: *arXiv preprint arXiv:1805.11074* (2018).

[188] Manan Tomar et al. "Mirror descent policy optimization". In: *arXiv preprint arXiv:2005.09814* (2020).

[189] Faraz Torabi, Garrett Warnell, and Peter Stone. "Generative adversarial imitation from observation". In: *arXiv preprint arXiv:1807.06158* (2018).

[190] Martin Treiber, Ansgar Hennecke, and Dirk Helbing. "Congested traffic states in empirical observations and microscopic simulations". In: *Physical review E* 62.2 (2000), p. 1805.

[191] Balakrishnan Varadarajan et al. "Multipath++: Efficient information fusion and trajectory aggregation for behavior prediction". In: *2022 International Conference on Robotics and Automation (ICRA)*. IEEE. 2022, pp. 7814–7821.

[192] John E Vargas-Munoz et al. "OpenStreetMap: Challenges and opportunities in machine learning and remote sensing". In: *IEEE Geoscience and Remote Sensing Magazine* 9.1 (2020), pp. 184–199.

[193] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[194] Jose L Vazquez et al. "Deep Interactive Motion Prediction and Planning: Playing Games with Motion Prediction Models". In: *arXiv preprint arXiv:2204.02392* (2022).

[195] Eugene Vinitsky et al. "Nocturne: a scalable driving benchmark for bringing multi-agent learning one step closer to the real world". In: *arXiv preprint arXiv:2206.09889* (2022).

[196] Matt Vitelli et al. "Safetynet: Safe planning for real-world self-driving vehicles using machine-learned policies". In: *2022 International Conference on Robotics and Automation (ICRA)*. IEEE. 2022, pp. 897–904.

[197] Pin Wang et al. "Decision making for autonomous driving via augmented adversarial inverse reinforcement learning". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 1036–1042.

[198] Yawei Wang and Xiu Li. "Reward function shape exploration in adversarial imitation learning: an empirical study". In: *2021 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA)*. IEEE. 2021, pp. 52–57.

[199] Yuhui Wang, Hao He, and Xiaoyang Tan. "Truly proximal policy optimization". In: *Uncertainty in Artificial Intelligence*. PMLR. 2020, pp. 113–122.

[200] Ziyu Wang et al. "Robust imitation of diverse behaviors". In: *Advances in Neural Information Processing Systems* 30 (2017).

[201] Maciej Wiatrak, Stefano V Albrecht, and Andrew Nystrom. "Stabilizing generative adversarial networks: A survey". In: *arXiv preprint arXiv:1910.00927* (2019).

[202] Blake Wulfe et al. "Dynamics-Aware Comparison of Learned Reward Functions". In: *arXiv preprint arXiv:2201.10081* (2022).

[203] Markus Wulfmeier et al. "Large-scale cost function learning for path planning using deep inverse reinforcement learning". In: *The International Journal of Robotics Research* 36.10 (2017), pp. 1073–1087.

[204] Huang Xiao et al. "Wasserstein adversarial imitation learning". In: *arXiv preprint arXiv:1906.08113* (2019).

[205] Danfei Xu and Misha Denil. "Positive-unlabeled reward learning". In: *arXiv preprint arXiv:1911.00459* (2019).

[206] Danfei Xu et al. "BITS: Bi-level Imitation for Traffic Simulation". In: *arXiv preprint arXiv:2208.12403* (2022).

[207] Tian Xu, Ziniu Li, and Yang Yu. "Error bounds of imitating policies and environments". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 15737–15749.

[208] Mengjiao Yang and Ofir Nachum. "Representation matters: offline pretraining for sequential decision making". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 11784–11794.

[209] Yaodong Yang et al. "Diverse auto-curriculum is critical for successful real-world multi-agent learning systems". In: *arXiv preprint arXiv:2102.07659* (2021).

[210] Fei Ye et al. "A survey of deep reinforcement learning algorithms for motion planning and control of autonomous vehicles". In: *2021 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2021, pp. 1073–1080.

[211] Zhao-Heng Yin et al. "Planning for Sample Efficient Imitation Learning". In: *arXiv preprint arXiv:2210.09598* (2022).

[212] Tianhe Yu et al. "Gradient surgery for multi-task learning". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 5824–5836.

[213] Wenyuan Zeng et al. "End-to-end interpretable neural motion planner". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 8660–8669.

[214] Wei Zhan et al. "Interaction dataset: An international, adversarial and cooperative motion dataset in interactive driving scenarios with semantic maps". In: *arXiv preprint arXiv:1910.03088* (2019).

[215] Amy Zhang et al. "Learning invariant representations for reinforcement learning without reconstruction". In: *arXiv preprint arXiv:2006.10742* (2020).

[216] Chiyuan Zhang et al. "A study on overfitting in deep reinforcement learning". In: *arXiv preprint arXiv:1804.06893* (2018).

[217] Junwei Zhang et al. "Proximal Policy Optimization via Enhanced Exploration Efficiency". In: *Information Sciences* (2022).

[218] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. "Multi-agent reinforcement learning: A selective overview of theories and algorithms". In: *Handbook of Reinforcement Learning and Control* (2021), pp. 321–384.

[219] Ming Zhang et al. "Wasserstein distance guided adversarial imitation learning with reward shape exploration". In: *2020 IEEE 9th Data Driven Control and Learning Systems Conference (DDCLS)*. IEEE. 2020, pp. 1165–1170.

[220] Qichao Zhang et al. "TrajGen: Generating Realistic and Diverse Trajectories with Reactive and Feasible Agent Behaviors for Autonomous Driving". In: *arXiv preprint arXiv:2203.16792* (2022).

[221] Qingzhao Zhang et al. "On adversarial robustness of trajectory prediction for autonomous vehicles". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 15159–15168.

[222] Xiaoqin Zhang and Huimin Ma. "Pretraining deep actor-critic reinforcement learning algorithms with expert demonstrations". In: *arXiv preprint arXiv:1801.10459* (2018).

[223] Xin Zhang et al. "f-gail: Learning f-divergence for generative adversarial imitation learning". In: *Advances in neural information processing systems* 33 (2020), pp. 12805–12815.

[224] Yangtao Zhang, X Jessie Yang, and Feng Zhou. "Disengagement Cause-and-Effect Relationships Extraction Using an NLP Pipeline". In: *IEEE Transactions on Intelligent Transportation Systems* (2022).

[225] Zhejun Zhang et al. "End-to-end urban driving by imitating a reinforcement learning coach". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 15222–15232.

[226] Hang Zhao et al. "Tnt: Target-driven trajectory prediction". In: *arXiv preprint arXiv:2008.08294* (2020).

[227] Ming Zhou et al. "Malib: A parallel framework for population-based multi-agent reinforcement learning". In: *arXiv preprint arXiv:2106.07551* (2021).

[228] Ming Zhou et al. "Smarts: Scalable multi-agent reinforcement learning training school for autonomous driving". In: *arXiv preprint arXiv:2010.09776* (2020).

[229] Zeyu Zhu and Huijing Zhao. "A survey of deep rl and il for autonomous driving policy learning". In: *IEEE Transactions on Intelligent Transportation Systems* (2021).

[230] Brian D Ziebart. *Modeling purposeful adaptive behavior with the principle of maximum causal entropy*. Carnegie Mellon University, 2010.

[231] Brian D Ziebart et al. "Maximum entropy inverse reinforcement learning." In: *Aaai*. Vol. 8. Chicago, IL, USA. 2008, pp. 1433–1438.

[232] Konrad Zolna et al. "Combating false negatives in adversarial imitation learning". In: *arXiv preprint arXiv:2002.00412* (2020).

[233] Konrad Zolna et al. "Task-relevant adversarial imitation learning". In: *arXiv preprint arXiv:1910.01077* (2019).