UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE AUTOMAÇÃO E SISTEMAS

Gustavo Claudio Karl Couto

Generative Adversarial Imitation Learning with Bird's-Eye View Input Generation for Autonomous Driving on Urban Environments

Florianópolis
2023

Gustavo Claudio Karl Couto

Generative Adversarial Imitation Learning with Bird's-Eye View Input Generation for Autonomous Driving on Urban Environments

Dissertação submetida ao Programa de Pós-Graduação em Engenharia de Automação e Sistemas da Universidade Federal de Santa Catarina para a obtenção do título de mestre em Mestre em Engenharia de Automação e Sistemas.
Supervisor:: Prof. Eric Aislan Antonelo, Dr.

Florianópolis
2023

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Couto, Gustavo Claudio Karl
Generative Adversarial Imitation Learning with Bird's Eye View Input Generation for Autonomous Driving on Urban Environments / Gustavo Claudio Karl Couto ; orientador, Eric Aislan Antonelo, 2023.
69 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico, Programa de Pós-Graduação em Engenharia de Automação e Sistemas, Florianópolis, 2023.

Inclui referências.

1. Engenharia de Automação e Sistemas. 2. Veículos Autônomos. 3. Aprendizagem por imitação. 4. Robótica. I. Antonelo, Eric Aislan. II. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Engenharia de Automação e Sistemas. III. Título.

Gustavo Claudio Karl Couto

Generative Adversarial Imitation Learning with Bird's-Eye View Input Generation for Autonomous Driving on Urban Environments

O presente trabalho em nível de mestrado foi avaliado e aprovado por banca examinadora composta pelos seguintes membros:

Prof. Paulo Fernando Ferreira Rosa, Dr.
Instituto Militar de Engenharia

Prof. Marcelo Ricardo Stemmer, Dr.
Universidade Federal de Santa Catarina

Certificamos que esta é a **versão original e final** do trabalho de conclusão que foi julgado adequado para obtenção do título de mestre em Mestre em Engenharia de Automação e Sistemas.

ICP-Edu
Documento assinado digitalmente
Julio Elias Normey Rico
Data: 25/10/2023 12:59:15-0300
CPF: ***.840.859-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Júlio Elias Normey Rico, Dr.
Coordenador do Programa

ICP-Edu
Documento assinado digitalmente
Eric Aislan Antonelo
Data: 25/10/2023 12:47:28-0300
CPF: ***.163.409-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Eric Aislan Antonelo, Dr.
Orientador

Florianópolis, 2023.

With deep gratitude to those who came before and those who will come after, I present this dissertation as a tribute to the timeless pursuit of knowledge and our shared journey of discovery.

ACKNOWLEDGEMENTS

I extend my deepest gratitude to my advisor, Eric Antonelo, for his invaluable guidance and support throughout this master's dissertation. I am thankful to my parents, Ana Maria and Almir, whose encouragement have been the driving force behind my academic achievements.

To my sister, Christiana, and my companion, Barabara, thank you for your understanding and motivation, which kept me focused on my studies and inspired me.

I'd like to thank Lucia Baruque, Andre Brazil and Allan Brazil for their support as part of my family.

Lastly, I appreciate the support of my friends and those, whose contributions were vital for the success of this research. Thank you all for being a part of this remarkable journey.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001

ABSTRACT

Veículos autônomos representam a próxima fronteira da robótica, envolvendo tarefas complexas de percepção e controle. Esta dissertação investiga algoritmos de aprendizado por imitação para treinar um agente a conduzir um veículo em ambientes urbanos usando sensores como câmeras frontais e um planejador de trajetos baseado no Sistema Global de Navegação por Satélite (GNSS). A arquitetura hierárquica de Aprendizagem por Imitação Adversarial Generativa (hGAIL) é proposta como uma solução completa para a navegação de veículos autônomos, mapeando diretamente percepções sensoriais para ações de baixo nível enquanto aprende uma representação de nível médio do ambiente. Ela é composta por uma arquitetura com dois módulos principais: a Rede Adversarial Generativa Condicional (CGAN) cria uma representação de Visão Aérea (BEV) a partir de imagens da câmera frontal, e a Aprendizagem por Imitação Adversarial Generativa (GAIL) aprende a controlar o veículo com base nas previsões do primeiro módulo. O agente é capaz de aprender a partir de demonstrações de especialistas, tornando-o adequado para tarefas onde os sinais de recompensa são difíceis de definir. Os resultados relatados mostraram que o GAIL, usando exclusivamente câmeras (sem BEV), não consegue sequer aprender a tarefa, enquanto o hGAIL, após treinamento exclusivo em uma cidade, conseguiu navegar autonomamente com sucesso em 98% das intersecções de uma nova cidade que não foi utilizada na fase de treinamento.

Palavras-chave: Veículos Autônomos. Aprendizagem por imitação. Navegação. Robótica.

RESUMO EXPANDIDO

Introdução

A condução autônoma é uma área promissora com o potencial de revolucionar os sistemas de transporte. Veículos autônomos são equipados para perceber seu ambiente através de uma variedade de instrumentos, incluindo câmeras, lidar e outros sensores. Abordagens tradicionais exploraram a divisão do problema em percepção, planejamento e controle para simplificar o problema complexo em uma sequência de tarefas mais gerenciáveis. A dificuldade surge quando, nos módulos de planejamento e controle, se descobre que é impossível prever todas as situações possíveis, além de lidar com as falhas de percepção. Nesse sentido, várias abordagens utilizando aprendizagem de máquina vêm sendo propostas. Ao invés de determinar de antemão o comportamento do veículo para cada situação, são utilizados dados para aprimorar uma função que irá definir o comportamento do veículo.

A clonagem comportamental é um tipo de aprendizado supervisionado onde pares de estado-ação, em que o estado representa a saída de sensores como as imagens de câmeras, e as ações são o controle do veículo, como o ângulo do volante e a aceleração, são utilizados para treinar uma política de controle que imite o comportamento de especialistas. Nesse algoritmo, os especialistas irão gerar uma série de exemplos com pares de estado-ação que o algoritmo utilizará como sinal de supervisão para treinar uma política de controle. Para as mesmas entradas, essa política deverá gerar a mesma resposta que os especialistas. Este algoritmo pressupõe que os especialistas devem gravar uma resposta para todo cenário possível; caso contrário, a política de controle não conseguirá aprender uma resposta correta e gerará uma resposta errada. Essa resposta inadequada leva o agente para um cenário ainda mais distante do fornecido pelos especialistas, gerando uma cascata de erros até um cenário catastrófico para o agente.

A aprendizagem por reforço permite que o agente aprenda interagindo com o ambiente de forma que ele aprenda com os próprios erros e gere políticas de controle robustas. Para isso, utiliza um sinal de recompensa que indica ao agente quais ações levaram a cenários desejados ou indesejados. Gerar esse sinal de recompensa pode ser difícil para várias tarefas, pois é necessário classificar e lidar com todos os cenários possíveis.

Uma maneira do agente aprender com exemplos de especialistas enquanto interage com o ambiente é a Aprendizagem por Imitação Generativa Adversarial (GAIL), onde um sinal de recompensa é fabricado, comparando a similaridade das trajetórias dos especialistas com as trajetórias geradas pelo agente. O objetivo do agente é gerar trajetórias indistinguíveis das trajetórias dos especialistas. Ainda assim, medir a similaridade entre trajetórias e utilizar esse sinal para treinar um agente de direção autônoma é um desafio complexo que está em aberto na ciência. Esta estratégia foi inicialmente investigado nesta dissertação

em um cenário de condução urbana simulada ao longo de rotas fixas, em que o agente recebe como entrada imagens de câmeras frontais.

O sinal de recompensa, apesar de informar o agente sobre pares de estado-ação desejados ou indesejados, transfere para o agente uma quantidade limitada de informação. Por isso, o treinamento de funções matemáticas complexas, como redes neurais mais profundas necessárias para aprender padrões complexos em imagens de câmeras, torna-se inviável. Uma estratégia para contornar essa limitação é a utilização de representações pré-processadas, em que as imagens das câmeras são projetadas para uma vista superior, conhecida como "Visão de Pássaro", e as regiões de interesse da imagem, como pistas de veículos e calçadas, são segmentadas em cores distintas. Assim, a política de controle aprende padrões simples nas imagens e concentra a capacidade de aprendizagem no problema de navegação. Essa estratégia é investigada para testar a GAIL em um cenário de condução urbana mais complexo, onde o agente tem que navegar por rotas dinâmicas em uma cidade simulada. A capacidade do agente de tomar diferentes caminhos é testada, e o agente que aprende a partir da "Visão de Pássaro" é capaz de navegar através de interseções, enquanto que o mesmo agente treinado a partir das imagens das câmeras falha em aprender a tarefa.

Para aprender a gerar a "Visão de Pássaro" a partir das imagens das câmeras, é possível utilizar uma estratégia semelhante à GAIL, em que uma rede neural é treinada para gerar a representação de "Visão de Pássaro" condicionada às imagens das câmeras, de forma que as representações geradas sejam indistinguíveis das representações de "Visão de Pássaro" geradas pelo simulador. Esse algoritmo é conhecido como Redes Neurais Generativas Adversariais Condicionais (CGAN), e quando combinado à GAIL, é capaz de treinar um agente para navegar em rotas dinâmicas a partir das imagens das câmeras. A combinação dos algoritmos é feita de forma hierárquica, em que a saída da CGAN é a entrada da GAIL, formando um novo algoritmo que denominamos hGAIL (GAIL Hierárquica). O algoritmo é testado em um ambiente simulado de direção urbana por rotas dinâmicas em uma nova cidade, diferente da utilizada para treinar o agente. O agente treinado utilizando hGAIL é capaz de navegar por essa nova cidade a partir das imagens, enquanto o agente treinado apenas com a GAIL falha em navegar até mesmo na cidade de treinamento.

Objetivos

O objetivo desta dissertação é investigar o emprego de métodos de aprendizagem por imitação adversarial para realizar tarefas de navegação urbana, além de aprender a gerar representações abstratas como entrada para o controle de veículos autônomos. Para isso,

conduzimos experimentos em simulações realistas com o intuito de avaliar os métodos propostos.

Uma das estratégias centrais da pesquisa é explorar o uso da representação "Visão de Pássaro", visando facilitar o processo de aprendizagem do veículo ao interagir com o ambiente. Além disso, busca-se desenvolver métodos para aprender a criar essa representação abstrata, empregando-a em uma arquitetura modular e hierárquica. A eficácia deste sistema é posteriormente avaliada em uma cidade de teste, objetivando-se verificar a capacidade de generalização do método proposto.

Metodologia

As Redes Neurais Generativas Adversariais Condicionais (CGAN) consistem em duas redes neurais: o discriminador e o gerador. A função da rede geradora é traduzir a informação de uma representação para outro domínio. No contexto desta dissertação, a informação sobre o entorno do veículo, contida nas imagens de três câmeras dianteiras e em uma representação abstrata da trajetória futura, é projetada em uma representação "Vista de Pássaro" que ilustra o ambiente ao redor do veículo como se visto de cima.

Enquanto a rede geradora converte o entorno do veículo na representação "Vista de Pássaro", o discriminador aprende a diferenciar as representações "Vista de Pássaro" originadas no simulador daquelas produzidas pela rede geradora durante seu processo de aprendizado. A saída do discriminador, por sua vez, é utilizada pela rede geradora como feedback para aprimorar suas próprias produções. Assim, ambas as redes são treinadas de maneira interativa até que as representações geradas pelo gerador se tornem indistinguíveis das representações criadas no simulador.

No contexto da Aprendizagem por Imitação Generativa Adversarial (GAIL), o discriminador e o gerador aprendem de maneira interativa, participando do mesmo "jogo" característico das GANs. Aqui, o gerador atua como uma política de controle, que aprende a mapear o estado do ambiente em ações, com o objetivo de imitar um conjunto de trajetórias fornecidas por especialistas. Para isto é necessário o julgamento do discriminador, que é treinado par medir a distância entre a distribuição de probabilidade das trajetórias originadas pelos especialistas e aquelas produzidas pela política de controle em treinamento.

A GAIL se distingue da GAN tradicional pelo conceito de trajetórias, sublinhando que as ações do agente no estado atual influenciam os estados subsequentes. Assim, em vez

de simplesmente replicar pares individuais de entrada e saída, a política de controle, atuando como gerador, deve se empenhar na tarefa mais complexa de mimetizar trajetórias completas. O gerador interage com o ambiente para acumular as trajetórias, que são então avaliadas pelo discriminador. Este, por sua vez, produz sinais de feedback, que são empregados pelo gerador para reforçar ações que recebem uma avaliação positiva.

Nesta dissertação, o algoritmo GAIL é utilizada para ensinar um veículo autônomo a navegar por um cenário urbano. O processo inicia com a condução por rotas predefinidas, recebendo como entrada imagens capturadas por câmeras frontais, e evolui para acomodar rotas variáveis, empregando a representação avançada conhecida como "Visão de Pássaro".

A "Visão de Pássaro" de um veículo ilustra sua posição e movimento através de um sistema de coordenadas de cima para baixo, criando uma representação 2D concentrada nas dimensões navegáveis do veículo. Essa visão superior é configurada de maneira que o ponto de partida do agente permaneça constante em uma posição específica dentro da imagem. A "Visão de Pássaro" do ambiente acompanha o movimento do veículo, possibilitando que o agente mantenha um campo de visão fixo, medido em metros, à sua frente. Nesta representação, a rota desejada, a área navegável e os limites das faixas são simbolizados por cores distintas, constituindo os canais de uma imagem colorida.

Por fim, o método hGAIL é desenvolvido, empregando os conceitos discutidos em uma estrutura hierárquica. Dois módulos são integrados dentro de uma sequência operacional: uma CGAN, que aprende a criar a representação de "Visão de Pássaro" a partir das imagens capturadas pelas câmeras e dos dados da trajetória. As representações produzidas pela CGAN são então fornecidas à GAIL, que, por sua vez, gera as ações direcionais para o veículo. Ambos os processos de aprendizagem ocorrem simultaneamente, utilizando os dados coletados durante interação com o ambiente. Esse método visa ampliar as capacidades da GAIL, mantendo, ao mesmo tempo, o princípio fundamental de aprendizado por interação com o ambiente.

Resultados

Realizamos três experimentos principais durante este estudo. No primeiro, treinamos um agente fim-a-fim para navegar por trajetórias fixas em uma cidade, utilizando Aprendizagem por Imitação Generativa Adversarial (GAIL). O agente conseguiu aprender a trajetória designada, conforme demonstrado por um diagrama que apresenta a redução de erros através do percurso ao longo do treinamento. Inicialmente, os erros eram frequentes, mas diminuíram à medida que o agente progredia, culminando na conclusão bem-sucedida da trajetória pré-definida. Este experimento teve como objetivo verificar a eficácia da GAIL em tarefas de navegação autônoma. Aprendendo a navegar em uma cidade deserta,

o agente processou imagens de alta dimensionalidade capturadas pelas câmeras.

Para ampliar a complexidade da tarefa e aproximar-se mais da realidade da direção autônoma em ambientes urbanos, exploramos uma representação intermediária que desconsidera a altura, focando nas dimensões em que o veículo opera, através do uso de uma visão aérea segmentada. Essa abordagem aliviou a carga sobre a rede neural, que, em vez de processar imagens de um mundo tridimensional complexo, pôde se concentrar na tarefa de navegação.

No segundo experimento, o agente aprendeu a dirigir em rotas dinâmicas, tomando decisões em tempo real sobre quais caminhos seguir nos cruzamentos, baseando-se na representação da "Visão de Pássaro". O agente, ao utilizar a "Visão de Pássaro" como entrada, conseguiu realizar a tarefa, enquanto o mesmo agente, usando imagens das câmeras frontais e a trajetória do veículo, falhou. Isso demonstra a importância de uma representação simplificada do ambiente para aprimorar a tarefa de controle.

No terceiro experimento, treinamos um módulo de percepção para gerar vistas superiores a partir das imagens das câmeras e da rota planejada, utilizando CGAN. Esse módulo foi capaz de criar a representação da "Visão de Pássaro", enquanto o módulo de controle utilizava a GAIL para dirigir o veículo com base na saída do módulo de percepção. A interação entre os dois módulos criou um sistema de aprendizagem generativo adversarial hierárquico (hGAIL), pois o módulo de controle usa a saída do módulo de percepção, e ambos evoluem através da interação com o ambiente.

O módulo de percepção foi capaz aprender a gerar essas vistas com precisão, conforme ilustrado pelas figuras comparativas no estudo, enquanto o módulo de controle aprendia a navegar nas rotas dinâmicas. A performance do sistema integrado foi semelhante à do agente que utilizava as representações geradas pelo simulador, demonstrando a capacidade do agente de aprender a navegar na cidade diretamente a partir de imagens mais complexas da câmera e rotas planejadas, desde que a tarefa de aprendizagem fosse dividida entre percepção e controle, aprendendo o controle a partir de uma representação intermediária que por sua vez também é aprendida.

Discussão e Considerações Finais

Foi proposta uma arquitetura hierárquica, denominada hGAIL, para a navegação autônoma de veículos em ambientes urbanos. A arquitetura hGAIL possibilita o aprendizado simultâneo de uma representação intermediária, a "Visão de Pássaro", e da política de controle

do agente. Aprender essa representação intermediária facilita a execução de tarefas de condução autônoma mais desafiadoras em comparação com o uso exclusivo das imagens brutas das câmeras. As imagens brutas, por serem uma representação menos abstrata, exigem que o agente se concentre excessivamente na tarefa de percepção, dificultando o aprendizado de tarefas de controle mais complexas.

Como próximos passos, pretende-se explorar os limites da capacidade de aprendizado do agente em tarefas mais complexas, em ambientes onde a cidade é habitada por outros veículos e pedestres. O agente terá de aprender a evitar colisões, obedecer às normas de trânsito, como respeitar sinais de parada e semáforos, e interagir de maneira segura e eficiente no ambiente. Outro aspecto que se pretende investigar é a transferência do conhecimento adquirido em simulações para cenários reais, utilizando veículos de controle remoto em pequena escala para navegar por passagens de pedestres.

Palavras-chave: Veículos Autônomos. Aprendizagem por imitação. Navegação. Robótica. Aprenzagem por Imitação Generative Adversarial.

ABSTRACT

Autonomous vehicles represent the next frontier of robotics, involving intricate tasks of perception and control. This dissertation investigates imitation learning algorithms to train an agent to drive a vehicle in urban environments using sensors like forward-facing cameras and a Global Navigation Satellite System (GNSS) based path planner. The hierarchical Generative Adversarial Imitation Learning (hGAIL) architecture is proposed as an end-to-end solution for autonomous vehicle navigation, directly mapping sensory perceptions to low-level actions while learning a mid-level representation of the environment. It comprises an architecture with two main modules: the Conditional Generative Adversarial Network (CGAN) generates a Bird's-Eye View (BEV) representation from frontal camera images, and the Generative Adversarial Imitation Learning (GAIL) learns to control the vehicle based on the predictions from the first module. The agent is able to learn from expert demonstrations, making it suitable for tasks where reward signals are difficult to define. The reported results have shown that GAIL exclusively from cameras (without BEV) fails to even learn the task, while hGAIL, after training exclusively on one city, was able to autonomously navigate successfully in 98% of the intersections of a new city not used in training phase.

Keywords: Autonomous Vehicles. Imitation learning. Navigation. Robotics.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

BC	Behavior Cloning
BEV	Bird's-Eye View
CGAN	Conditional Generative Adversarial Networks
GAIL	Generative Adversarial Imitation Learning
GAN	Generative Adversarial Networks
GNSS	Global Navigation Satellite System
IMU	Inertial Measuring Unit
MDP	Markov Decision Process
PPO	Proximal Policy Optimization

CONTENTS

1 INTRODUCTION

1.1 MOTIVATION

The task of transportation plays a crucial role in our society, enabling the movement of goods, people, and resources from one place to another. As human settlements grew larger, logistical challenges emerged, and ancient civilizations, like the Roman Empire, addressed them by constructing extensive road networks. Over time, transportation methods have evolved, including the advent of trains and automobiles. However, with recent advancements in technology, the next significant milestone in public transportation has emerged: autonomous driving.

Autonomous driving bring raise many economic and social impacts for our society, the first ones that we could cite are decrease of traffic accidents, decrease the cost of logistic as companies would save money from human drivers, increase the general efficiency in traffic as software agents far exceed the human capacity to coordinate themselves and improve traffic efficiency. At the same time it brings ethical challenges as responsabilities and liabilities for traffic accidents caused by the autonomous vehicle. For the last it also raise concerns about a general desemployement, as app drivers, app delivery jobs would cease to exist.

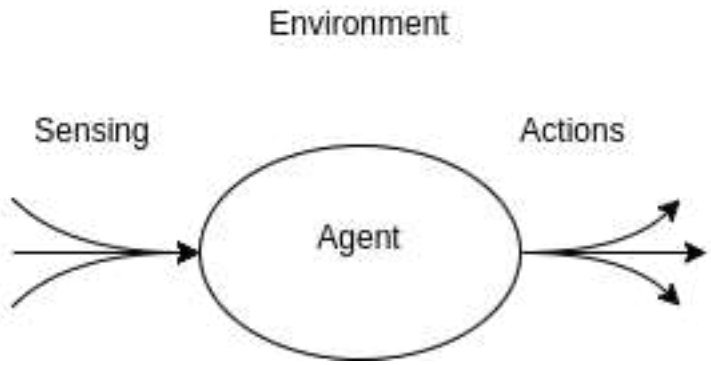


Figure 1 – Agent architecture.

Let's discuss what an autonomous vehicle is. An autonomous vehicle is a rational agent in a multi-agent environment capable of sensing the surroundings, acting upon it, and interacting with other agents to achieve an objective. To sense the environment, the autonomous vehicle uses cameras, lidar, GNSS, odometer, inertial sensors, and other sensors. It gathers information about other agents such as vehicles, pedestrians, cyclists, motorcycles, and the road. Additionally, it analyzes the drivable area and lane boundaries to make informed decisions.

The agent then reasons to determine the actions it will take to execute a task. The reasoning process can be as simple as consulting a table that contains the best action for each sensor response to execute the task. However, creating a comprehensive table for all

possible scenarios is impractical. As the reasoning process becomes more complex, it can be divided into a pipeline of sensing, planning, and decision-making.

Modern driver assistance systems employ complex sensing and predefined rules to build heuristics for actions like driving on a highway. However, this approach still requires human supervision in unknown or dangerous situations. End-to-end learning offers an alternative, where the agent directly transforms sensor outputs into actions using a reward function. Reward functions can stimulate the vehicle to stay centered on the lane, complete the trajectory, and penalize infractions.

Driving is a social task, and the agent interacts with other vehicles and pedestrians during the journey. The desired behavior is for the agent to act similarly to human drivers while avoiding actions that others wouldn't expect, reducing the likelihood of accidents. The strategy of generative adversarial imitation learning trains an agent to imitate human drivers effectively, but it presents challenges as both the agent and the classifier are trained in parallel.

To simplify the agent's task, researchers perform a perspective transformation on the images from the cameras. This perspective transformation projects the three-dimensional space into a two-dimensional top view of the scene, removing the dimension of altitude as the agent is not capable of navigating in that dimension. This makes the navigation task more manageable, focusing only on the two-dimensional navigation while gravity keeps the agent grounded.

1.2 OBJECTIVES

The objective of the present thesis is to investigate the use of online learning and BEV representation learning using methods based on adversarial imitation for the problem of autonomous driving in an urban environment.

The specific aims are as follows:

- Elaborate a review of the problem of learning autonomous driving in urban settings;
- Conduct an in-depth evaluation of the trained controller in a simulated autonomous driving environment;
- Evaluate a mid-level representation to facilitate the driver agent's task.
- Develop a method to generate the mid-level representation for achieving end-to-end learning.
- Evaluate the end-to-end hierarchical architecture on different scenarios to test the generalization of the final controller.

Therefore, this dissertation contributes to the literature by investigating the application of generative adversarial methods for the problem of autonomous driving. It adds

to the study of these methods by developing a new application scenario and highlighting the weaknesses and advantages of the approach. Furthermore, it contributes to the toolbox for autonomous driving, which is still an open problem, by introducing new approaches that, when combined with traditional methods, can improve existing solutions or lay the groundwork for new ones.

1.3 OUTLINE OF THE DISSERTATION

This dissertation contains six chapters, of which the first is this introduction providing the main motivation for carrying out the work, along with the objectives.

Chapter 2 reviews works related to the field of imitation learning for autonomous driving and online learning, which allows the agent to develop robust policies by learning from expert demonstrations and its own experiences. The chapter also explores the use of mid-level input representation to facilitate the autonomous driving task and bridge the simulation-to-real gap through abstractions that are common to both realms. Additionally, it discusses learning methods for the generation of mid-level abstractions from the raw sensor inputs.

Chapter 3 lays out the crucial groundwork for our method. It introduces a supervised learning-based imitation approach, underpinned by detailed definitions and equations. The chapter progresses to explain the core concepts of online learning, emphasizing how agents evolve through hands-on interactions with their surroundings. The chapter then shifts focus to CARLA, the simulator used for agent training, detailing the various sensors assisting the vehicle in navigation.

The following, Chapter 4, focuses on studying online learning from expert demonstrations in an end-to-end fashion for an agent that trains to navigate a fixed trajectory on a high-fidelity simulator. The experiment proves to be successful, as the agent is able to complete the trajectory. The chapter describes the agent's architecture, evaluates the method, and compares it to other solutions.

Further, Chapter 5 purpose a new look into the problem of autonomous driving trading a more complex task with a representation that makes the learning more efficient. We purpose an imitation learning agent that is capable of navigating dynamics routes, a route that is created at spot, instead of the fixed routes from the last chaptar. The agent learns to navigate the city from mostly a segmented top down view of the environment.

In Chapter 6, a solution to generate the segmented top down view representation from raw sensor inputs is presented while the entire hierarchical solution is tested on a challenging generalization scenario, where the vehicle needs to drive a scenario that was not explored during the training phse.

Finally, Chapter 7 concludes and provides reflections on the thesis, in addition to suggestions for future works.

2 RELATED WORKS

In the last chapter we have raised the motivations to this work and detailed how this work is proposed. On this chapter we review how this work is related to other works in the academic society that have been both an inspiration to this work or have proposed the basis from where we start our investigation.

2.1 IMITATION LEARNING

Several studies have explored the integration of deep neural networks in autonomous driving systems. One notable approach, proposed by Codevilla et al. (CODEVILLA et al., 2018), utilizes deep neural networks to guide vehicles towards specific turns at upcoming intersections. This system operates based on high-level command inputs and employs imitation learning in a supervised manner. The network is conditioned to respond to navigational commands, dedicating itself to the task of driving. This conditioning architecture has served as an inspiration for many subsequent works, including ours. The experiments conducted on a 1/5 scale robotic truck and the CARLA simulation platform demonstrate the successful deployment of this approach in real-world scenarios.

In the pursuit of superior generalization capabilities for autonomous driving, CIRL (Controllable Imitative Reinforcement Learning) (LIANG et al., 2018) introduces a novel approach that builds an agent using the same logic of following high-level commands at intersections as the previous work mentioned. By deploying the training policy on CARLA, the CIRL model develops a more robust policy that learns from its own experience while exploring a reasonably constrained action space guided by encoded experiences that imitate human demonstrations. The agent's policy is warmed up by employing BC in a pre-training phase. This innovative approach significantly improves sample efficiency of online learning and outperforms other imitation learning systems on the public CARLA benchmark.

Following the same line of online learning to build generic and robust policies, Jena et al. (JENA; LIU; SYCARA, 2020) tackle the problem of online imitation learning, which aims to recover an expert policy without access to a reward signal. The authors explore GAIL, an algorithm that proposes the use of a reward signal generated by a learning neural network trained to distinguish actions generated by the learning agent from those sampled from expert demonstrations. In an on-policy manner, they combine BC with GAIL, thereby further improving the algorithm's sample efficiency. Their approach demonstrates stability in high-dimensional tasks and provides valuable insights into addressing the issue of covariate shift in imitation learning. The versatility of this approach is demonstrated on a variety of environments, including MuJoCo environments and image-based Car Racing scenarios.

2.2 ONLINE LEARNING WITH MID-LEVEL INPUT REPRESENTATION

The significance of decoupling representation learning from Reinforcement Learning (RL) is highlighted by ResNet RL (OTA; JHA; KANEZAKI, 2021). This study sheds light on the challenges faced by deep RL agents when training with larger networks, leading to instability. While domains like computer vision and natural language processing have benefited from larger networks, the RL community has yet to witness a similar trend. The research underscores the importance of stable training methods for high-performance RL agents.

ChauffeurNet (BANSAL; KRIZHEVSKY; OGALE, 2018) takes a unique approach to autonomous driving by employing mid-level input and output representations. Their perception system processes raw sensor information to produce an abstract inputs: a top-down BEV representation of the environment and a rendering of the road information and traffic lights states. By synthesizing challenging scenarios and augmenting expert demonstrations, the method enhances imitation learning. This alleviates the burden of learning perception and improves the effectiveness of the imitation learning process. One significant advantage of these mid-level representations is that the neural network can be trained on either real or simulated data and easily validated in closed-loop simulations before deployment on a real car. Ultimately, the model is successfully demonstrated driving a car in the real world, showcasing the effectiveness of their approach.

Roach (ZHANG, Z. et al., 2021) trains an RL expert to map BEV images to continuous low-level actions, providing informative supervision signal to train an end-to-end imitation learning agent. The end-to-end apprenditice learns from querying the RL expert on online training set. To enhance the stochastic agents, the method incorporates a beta distribution, effectively representing the low-level actions mapped to a finite space (E.g. 0 to 1). Additionally, an exploration loss is utilized to shape an efficient exploration space. The Roach model sets a new performance upper-bound on the challenging CARLA LeaderBoard, demonstrating high sample efficiency and effective imitation learning.

Muller et al. (MÜLLER et al., 2018) present an approach that focuses on driving policy transfer via modularity and abstraction. Their method encapsulates the driving policy to operate on a semantic map and output waypoints, enabling extensive training in simulation and direct application to a physical vehicle. The transfer of the policy to a 1/5-scale robotic truck on diverse roads and environmental conditions demonstrates its adaptability and robustness.

The paper "Hierarchical Interpretable Imitation Learning for End-to-End Autonomous Driving (HIIL)" (TENG et al., 2023) presents a two-stage end-to-end autonomous driving model. In stage one, a pre-trained BEV semantic masks generator is utilized, which combines raw camera data to generate an interpretable representation. In the second stage, the representation generated by stage one is combined with additional steering angles from the Pure-Pursuit algorithm to drive the autonomous vehicle. Extensive

experiments conducted on the CARLA simulator demonstrate remarkable interpretability, generalization, and robustness in unknown scenarios. Additionally, the semantic masks generated by the first stage provide valuable insights into the ego-vehicle's understanding of the driving scenario.

2.3 MID-LEVEL REPRESENTATION GENERATION

Addressing the sim-to-real gap, Reiher et al. (REIHER; LAMPE; ECKSTEIN, 2020) propose a sim2real deep learning approach to transform segmented images from monocular cameras into semantically segmented BEV representations. When using monocular cameras, estimating distances of elements in the environment becomes challenging. However, transforming the camera perspective to BEV makes distance estimation easier.

By using semantically segmented images as input, the authors reduce the reality gap between simulated and real-world data, as the segmented images from a simulator are already equivalent to segmented images from the real world. They demonstrate how using synthetic datasets and input abstraction to semantically segmented representations enables training a neural network on synthetic data only, while still effectively performing tasks on real-world data. Their method significantly reduces the reality gap between simulated and real-world data, allowing for successful application without manually labeling BEV images.

Finally, Philion et al. (PHILION; FIDLER, 2020) propose a perception approach for autonomous vehicles that extracts semantic representations from multiple sensors and fuses them into a single BEV coordinate frame. The approach employs "lift" and "splat" operations to generate frustums of features for each camera and then aggregates them into a BEV grid. From this grid, a BEV image is generated to be consumed by a motion planning module.

2.4 DISCUSSION

These works provide foundations for our investigation. The works from Section 2.1 provide the basis to justify a deeper study on imitation learning, which is a general method useful for addressing many significant open problems, especially in the context of autonomous driving. The work from Jena et al. (JENA; LIU; SYCARA, 2020) is particularly valuable as they make their source code publicly available, serving as a starting point for our initial experiments to study the application of imitation learning algorithms, especially GAIL, for autonomous driving on the CARLA platform.

In Section 2.2, we present a review of many successful experiments using BEV segmented images to decouple the online training of a learning policy from the raw sensors of autonomous vehicles. Roach (ZHANG, Z. et al., 2021) has also released the code for their experiment, providing an open suite to train an agent to drive from BEV segmented images

using Reinforcement Learning on CARLA. We have successfully utilized this resource to evaluate imitation learning from BEV images for autonomous navigation.

Section 2.3 covers works that focus on generating BEV semantical maps from monocular cameras attached to a vehicle, demonstrating the growing interest in this representation and the feasibility of generating it from the vehicle's frontal cameras.

3 METHODS

In this chapter, we establish the technical framework requisite for the successful implementation of our proposed method. We commence by introducing a supervised learning approach specifically designed for the domain of imitation learning. Subsequently, we elucidate the foundational principles of online learning, which conditions prospective states based on current observations, thereby challenging the conventionally held independent and identically distributed (i.i.d) assumption. Predicated on this paradigm of experiential learning, we propose a method that not only leverages expert demonstrations but also interacts iteratively with the environment. This interaction aims to derive the environment's state transition function and formulate strategies for error rectification.

Further, we present a detailed overview of CARLA, a high-fidelity simulation platform tailored for autonomous driving, which serves as the empirical bedrock for our experimental analyses. We delineate the methodology adopted in CARLA to collect data from a proficient driver, equipped with exact locational data and a detailed trajectory, thereby generating the pertinent training dataset. In addition, we detail the specific input variables that inform our agents' navigational decisions within CARLA. These encompass high-level commands, RGB visual data from frontal cameras, a sparse trajectory representation the agent is expected to pursue, and BEV semantic map representations.

3.1 CONDITIONAL GENERATIVE ADVERSARIAL NETWORKS - CGAN

CGAN are composed of two neural networks, a discriminator D and a generator G. The function of G in a CGAN (ISOLA et al., 2017) is to translate an image x into an image y by mapping both x and a random noise vector z into an output image $G : \{x,z\} -> y$. Both D and C seek to optimize the same objective function:

$$\underset{x,y}{\mathbb{E}}[\log(D(x,y))] + \underset{x,z}{\mathbb{E}}[\log(1 - D(x,G(x,z))], \tag{1}$$

where G tries to minimize it, while D seeks to maximize it. As in traditional Generative Adversarial Networks (GAN), D learns to classify real images from generated ones, and G uses this output of D to direct its own learning. Notice that both D and G are conditioned on the input image x that must be translated.

Additionally, a L1 distance loss function is added to the final objective, making the generator network G also learn from the true label y as it would happen in a supervised learning task (ISOLA et al., 2017).

The L1 loss can model the low-frequency characteristics of images, while the CGAN loss is crucial for modeling high-frequency features. By classifying local patches of the image rather than the entire image, the discriminator can better capture high-frequency correctness, as the assumption is that pixels from different patches are independent.

This method, called PatchGAN (ISOLA et al., 2017), divides the image into multiple patches, and the discriminator classifies each patch individually. This results in a discriminator with fewer parameters and a detailed feedback for the agent

The CGAN is used in our work to generate the BEV image representation from the agent's sensors such as frontal cameras and GNSS, to be detailed later.

3.2 MARKOV DECISION PROCESS

3.2.1 Agent-Environment interaction

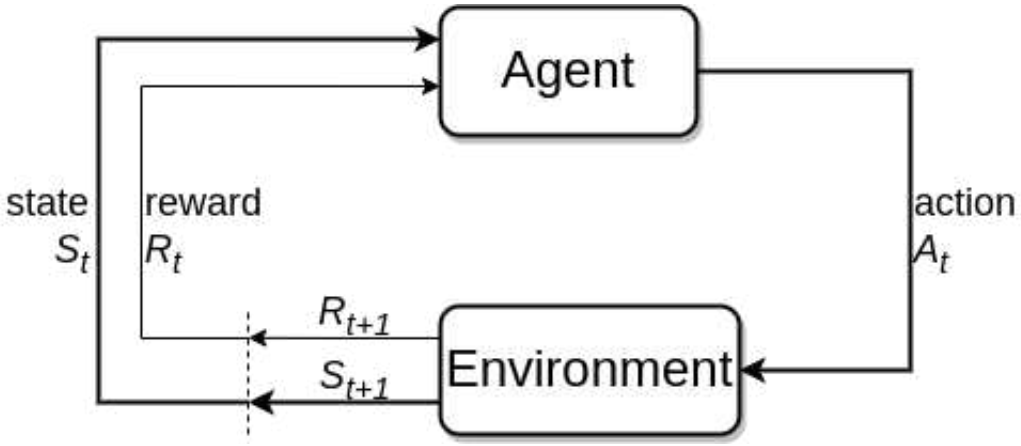


Figure 2 – Agent environment interaction.

Source: Adapted from (SUTTON; BARTO, 2018)

The autonomous driving problem involves an agent that learns and makes decisions within an environment. The agent interacts with the environment through actions and receives information back in the form of a state and a reward. The reward is a numerical value that the agent seeks to maximize over time. This general system is presented in Fig. 2.

3.2.2 Markov Property

An important property that is given to the problem to make it feasible to be solved and control its complexity is known as the Markov property. This property states that, given the present, the future should be independent of the past.

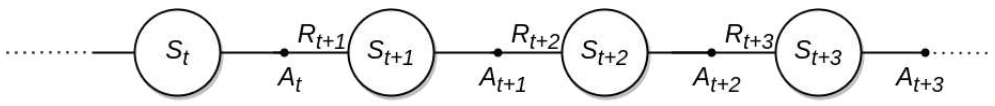


Figure 3 – MDP episode.

Source: Adapted from (SUTTON; BARTO, 2018)

Moving from one state to another is known as a transition, and the transition function defines the probability of moving from one state to another given an action.

Importantly, this transition function depends only on the present state, implying that the past is not important for predicting the future, only the present state matters. This concept is illustrated in Fig. 3, which presents the sequence of states as a linked list, where each state is only connected to the state before it and the state ahead of it.

3.2.3 Return function

In our system, the agent will try to maximize the cumulative reward instead of the reward it receives from the current state. This approach requires the agent to consider the future consequences of its actions and account for the temporal dependence between states and actions. The total sum of reward the agent seeks to maximize is called the return and is defined as follows:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + ... + R_T, \tag{2}$$

Future rewards can be considered either more or less important than the immediate reward, depending on the task. For instance, in autonomous driving, the next 10 states are certainly more important to the current action being taken than a reward from 100 states in the future. To incorporate this importance into the problem, a discount factor γ is introduced. The new discounted return is defined as follows:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{3}$$

where $0 < \gamma < 1$.

3.2.4 Agent policy

A policy is a mathematical abstraction that describes the behavior of the agent. If the agent can be represented as a black box that receives the state from the environment and returns an action to maximize its return, then the agent's behavior can be described by a function that defines the probability of taking an action given a state:

$$\pi(a|s) = \mathbb{P}\left[A_t = a | S_t = s\right] \tag{4}$$

This way, we define a stochastic agent, which, instead of selecting the best action for each state, defines a probability distribution over actions $a \in A$ for each state $s \in S$. In this manner, the agent seeks to maximize the probability of taking the best actions, creating a smooth objective that can be optimized over time.

3.2.5 Value Function

Now that we have defined the policy, we can estimate what to expect as we follow that policy through a sequence of states. This is done by computing the expected

discounted return of a state, given that we pick actions from the action distribution defined by the policy $a \sim \pi$ for this state, and continue doing the same for all the next states. This defines the value function:

$$v_\pi(s) = \mathbb{E}\pi[G_t|S_t = s] = \mathbb{E}\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \Big| S_t = s \right], \tag{5}$$

where $v_\pi(s)$ represents the value function for a state s, which denotes the expected return starting from state s and following policy π thereafter, for all $s \in S$.

3.2.6 Mathmatical Formulation

The problem is formulated as an infinite horizon MDP, defined by the tuple $(\mathcal{S}, \mathcal{A}, P, r, \rho_0, \gamma)$, where $\mathcal{S}, \mathcal{A}$ represents, respectively, the state and action spaces. $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the transition probability distribution, $r : \mathcal{S} \to \mathbb{R}$ is the reward function, $\rho_0 : \mathcal{S} \to \mathbb{R}$ is the initial state distribution and γ is the discount factor.

On MDP problems a sequence of state-action pairs induced by a policy is called a trajectory. The solution to the MDP problem is a stochastic policy that maximizes the value function for entire trajectories:

$$\pi^* = arg \max_{\pi} \mathop{\mathbb{E}}_{s_0 \sim \rho_0} v_\pi(s_0). \tag{6}$$

3.3 POLICY OPTIMIZATION

To find the best policy, an optimization method based on gradient descent is used to interactively reach the best policy starting from a randomly initialized function. The objective function for the method, however, is defined using the advantage function, that we need to first define.

The action-value function is equivalent to the state value function when the agent takes an arbitrary action a_t on state s_t and then starts to act according to it's policy. The function is defined by the following equation:

$$Q_\pi(s_t,a_t) = r(s_t) + \gamma \mathbb{E}[v_\pi(s_{t+1})], \tag{7}$$

where $s_{t+1} \sim P(s_{t+1}|a_t,s_t)$.

Both the value function $v_\pi(s_t)$ and action-value function $Q_\pi(s_t,a_t)$, are used as building blocks to define the advantage function. The function defines how beneficial it is to take the action a_t on the state s_t instead of following the agent's policy and is defined by the equation:

$$A_\pi(s,a) = Q_\pi(s,a) - v_\pi(s). \tag{8}$$

Finally, the objective function for the policy optimization is based on the policy gradient for equation (6). The mathematical details to get to this equation are beyond the scope of this work, a detailed proof, however, can be found on (SUTTON; BARTO, 2018). The objective function, then, is defined as:

$$L(\theta) = \mathop{\mathbb{E}}_{\tau\sim\pi}[log(\pi_\theta(a|s))A_\pi(s,a)], \tag{9}$$

where $\tau = (s_0,a_0,...)$ are trajectories sampled on the environment using the agent's policy.

This objective function builds the basis for the neural networks reinforcement learning methods, in these methods the policy is put to interact with the environment to estimate the advantage function. The estimated function, then, is used to build this objective function and calculates its derivatives with respect to the neural networks parameters θ. These derivatives are then used to interactively improve the policy, until a random initialized policy converges to the optimal policy defined on (6).

3.4 IMITATION LEARNING

The objective of Imitation Learning is to train a policy to perform a task from demonstrations. For that, the learner is trained to mimic the experts' behaviour embedded on example trajectories. There are two main branches of the field. In the first, BC, which is the simpler of the two, the policy is trained to mimic the mapping between states and actions embedded on the experts' demonstrations. The technique, however, ignores the time dependence between states-actions pairs and small deviations are prone to cascade and lead to catastrophic error.

On the second branch of Imitation Learning, Inverse Reinforcement Learning (IRL), a reward function that makes the experts' behaviour uniquely optimal is retrieved from the example trajectories. The technique is very sample efficient on demonstrations, needing just a few demonstrations to generate entire trajectories similar to those on the demonstrations. The algorithm, however, is very hard to train, as it's not efficient on the number of interaction with the environment needed to converge.

For BC, the policy is trained to map states into actions using supervised learning, based on the following loss function:

$$\mathcal{L}_{\mathcal{BC}} = -\mathop{\mathbb{E}}_{\tau_E}[\log(\pi(a|s|))]. \tag{10}$$

The intuition for equation (10) is to maximize the probability for the learner to generate the same actions as the experts on the demonstrations. BC states the problem of learning to perform a task as a problem of regression, i.e. curve adjustment or interpolation.

3.5 GENERATIVE ADVERSARIAL IMITATION LEARNING

GAIL is a method inspired by GAN and IRL. GAIL takes the training method from the latter, i.e. to use reinforcement learning to imitate a series of demonstrations.

From the former it takes the idea of a discriminator network and uses it to generate the reward signal for the policy optimization loop.

The discriminator network generates a signal based on the discrepancy between state-action pairs from experts' demonstrations and learner interactions with the environment. On the proposal of GAIL algorithm, the discriminator is trained as a classifier to identify experts' samples from those generated by the learner.

In more recent works (XIAO et al., 2019; ZHANG, M. et al., 2020), though, the discriminator is trained to generate a measure of the distance from the state-action pairs' distribution from the experts' demonstrations to the one one generated by the learner policy. It was found that this distance provides a more stable feedback signal to the training loop as the used Wasserstein distance function is continuous and differentiable almost everywhere.

The objective function to train the discriminator is described by the following equation:

$$\mathcal{L}_{\mathcal{D}} = \mathop{\mathbb{E}}_{\tau_E}[D(s,a)] - \mathop{\mathbb{E}}_{\tau_\pi}[D(s,a)] - \lambda L_{gp}, \tag{11}$$

where $D(s,a)$ is the discriminator network and L_{gp}, the gradient penalty is a regularization term proposed on (GULRAJANI et al., 2017) to push the discriminator network to 1-Lipschitz function space to satisfy the condition of the Wasserstein distance mathematical construction.

3.6 BC-AUGMENTED GAIL

The GAIL algorithm is a very efficient algorithm capable of generating a policy with only a few demonstrations. It is, however, heavily dependent on interactions with the environment. BC on other hand is dependent on a high volume of experts' demonstrations, and creates policies that are prone to cascading error. It doesn't need, however, to interact with the environment.

In (JENA; LIU; SYCARA, 2020), a method is proposed based on both methods, augmenting GAIL with a supervised training loop that doesn't interact with the environment. Still using GAIL, however, to learn from interactions with the environment.

To construct the new objective function for that method, the BC objective function equation (10) can be rewritten as:

$$\mathcal{L}_{\mathcal{BC}} = -\mathop{\mathbb{E}}_{\tau_\pi}\left[\frac{\rho_E(s,a)}{\rho_\pi(s,a)}\log(\pi(a|s|))\right], \tag{12}$$

where $\rho(s,a)$ denotes the state-action visitation probability and is used here to change the expectation of equation (10) from experts' trajectories to ones generated by the policy. The BC augmented GAIL proposes an objective function that combines BC and GAIL methods defined by the equation:

$$\mathop{\mathbb{E}}_{\tau_\pi}\left[\left(\alpha\frac{\rho_E(s,a)}{\rho_\pi(s,a)} + (1-\alpha)A_\pi(s,a)\right) log(\pi_\theta(a|s))\right], \tag{13}$$

where α is a hyperparameter added to define a weighted sum of equations from GAIL policy optimization (9) and BC loss (12).

The equation can be interpreted to encourage the learner to take the same actions as the experts in states seen in the demonstrations, where $\rho_E(s,a)$ is not zero. For states that are not presented in the demonstrations, i.e. if the expert visitation probability is zero, the learner can still learn an action for them from the advantage function.

The equation can also be reduced to:

$$\alpha L_{bc} + (1 - \alpha) L_\theta, \tag{14}$$

where it is transparent that the method augments the policy optimization objective function with a new term from BC to accelerate the training using supervised learning. As a result the method decreases the total interactions with the environment necessary for the algorithm to converge.

The α term controls the weighted sum for BC and can be found using simulated annealing. In addition, the term can be higher in the initial states when the value function is still not well trained and the policy is not taking meaningful trajectories.

3.7 CAR LEARNING TO ACT: CARLA

CARLA is an open-source simulator developed for research on autonomous vehicles, and it has become a popular platform for evaluating various autonomous vehicle research projects. Built on top of Unreal Engine, which is a game development platform, CARLA offers a rich set of scenarios and characters, including pedestrians, cyclists, motorcycles, cars, and trucks. The simulator provides at least five towns, each with different characteristics, such as residential neighborhoods, larger avenues, tunnels, roundabouts, and highways.

The collection of maps in CARLA is extensive, featuring various buildings and houses, each with unique characteristics. The same attention to detail applies to the vehicles and characters, making the simulation look more realistic.

CARLA utilizes the Unreal 3D physics simulator, providing a comprehensive experience with collision detection and joint simulation.

The simulator comes with a client library in Python, enabling users to control the entire simulation. In asynchronous mode, it allows for precise control over each step of the simulation to ensure synchronization between the agent and the environment, enabling the agent to process all data from each step effectively.

Each actor in the simulation can be equipped with multiple sensors, such as LiDAR (Light Detection and Ranging), monocular cameras, odometers for current speed output, GNSS, Inertial Measuring Unit (IMU) to display acceleration, and magnetometers to output current orientation. The simulator also provides the capability to read the position of moving objects from the map, including the current state of nearby traffic lights.

The monocular cameras in CARLA can be attached to any position and angle on the vehicle, with configurable extrinsics values. Users can simulate lens distortions like chromatic aberration and light falloff. Additionally, the simulator allows access to ground truth data for camera image segmentation, with classes for various objects present in the images.

Furthermore, CARLA offers a semantic map for each town, providing lane and drivable area details. By using the CARLA API, users can build a detailed route with precision down to 50 centimeters. This level of detail makes CARLA a powerful tool for researchers and developers working in the field of autonomous vehicles.

Figure 4 – Images from the three frontal cameras located at the left, central, and right part of the vehicle, respectively. They were taken after the first few interactions of the agent in the CARLA simulation environment considering our defined trajectory. Each camera produces a RGB image with 144 pixels of height and 256 pixels of width. These images are fed to the networks as they are.

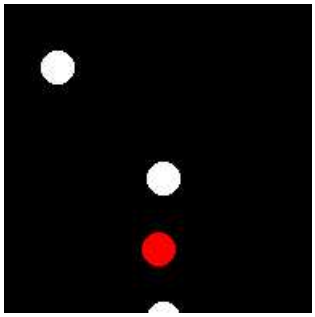


Figure 5 – The sparse trajectory visual input is captured at the same frame as shown in Fig. 4. The points from the sparse trajectory and the highlighted vehicle position are plotted as circles with a radius of 10 pixels, using the same scale (pixels per meter) and perspective as the BEV representation. When the image is fed to the agent, it is represented with only one channel and a size of 192x192 pixels.

3.7.1 CARLA Leaderboard

The team behind CARLA has developed a unified evaluation benchmark known as the CARLA Leaderboard[1]. This platform allows users to submit their code for external evaluation by the CARLA team online, with the results displayed on a public leaderboard.

The CARLA Leaderboard offers several options for autonomous navigation tasks, including tasks with maps, without maps, and with Lidar. These tasks challenge the agents to complete routes on maps that are not publicly available, forcing them to generalize their learned skills and tasks during training.

The test environment in the CARLA Leaderboard includes variable weather conditions, such as fog, night, and heavy rain, which require the agents to successfully complete tasks in low-visibility situations.

Additionally, the Leaderboard simulates a crowded environment with other vehicles like trucks, cars, motorcycles, and cyclists. Pedestrians are also included in the simulation, and they may occasionally cross the road outside of crosswalks while traffic lights are green for vehicles. This aspect adds complexity as the learning agents must not only follow traffic rules but also be attentive to pedestrians who may cross unexpectedly.

For training, the Leaderboard provides a set of routes in each public town of CARLA, ranging in distance from 500 meters to a little over one kilometer. The agent receives a description of the route to follow as a set of high-level points, along with commands for turning or going straight. These points are provided every 50 meters, but there are additional points when there is a change of command, such as entering an intersection.

Using the public set of training routes and the CARLA client, it is possible to create a set of routes described by CARLA coordinates at 50-centimeter intervals. With this set of coordinates and PID control, an autonomous vehicle can accurately follow the designated training routes in an environment empty of other agents, without concerns for collisions or norms like stop signs and traffic lights.

The agents investigated during this dissertation will need to perform the same task of navigating the defined trajectory, but without access to its detailed description. Instead, they will rely only on the high-level description available from the CARLA Leaderboard tasks, challenging them to navigate without explicit route details.

3.7.2 Collected data

The environment and trajectories are obtained from the CARLA Leaderboard evaluation platform. In particular, the *town01* environment from this platform along with ten predefined trajectories are employed to generate the expert training set for the agents trained on this dissertation experiments.

[1] CARLA Autonomous Driving Leaderboard available at: https://leaderboard.carla.org/

The expert dataset is constructed using a deterministic agent that navigates using a dense point trajectory and a classic PID controller (CHEN et al., 2019). The dense point trajectory provides many points at a fine resolution, whereas a sparse point trajectory consists of considerably fewer points, providing only a general sense of direction to the agent. As a result, the dense point trajectory is utilized to generate training data by the expert, whereas the sparse point trajectory is employed by the agent for more general guidance.

In Fig. 6, one of the 10 routes executed by the expert to form the labeled training set of demonstrations is shown, where the line starting in yellow and ending in red represents the desired trajectory (not observable to the agent as it is). The sparse trajectory can be seen as yellow dots, generated every 50 meters traveled or when the vehicle is about to start a different movement (from *straight* to *turn* and vice-versa).

The ten trajectories of the training set were recorded at a rate of 10 hertz, resulting in 10 observation-action pairs per second. For the shortest route of 1480 samples (average route of 2129 samples), it represents 2.5 minutes (3.5 minutes) of simulated driving. All the ten trajectories yielded a total of 21,287 training samples (30 GB of uncompressed data). The total set corresponds approximately to 36 minutes or 8km of driving.

Figure 6 – *Town01* environment of the agent, with one of the routes used to collect data by the expert. The highlighted path has 740 meters, 20 points in the sparse trajectory (shown as yellow dots) and 762 points in the dense point trajectory (not shown).

3.7.3 BEV representation

The BEV of a vehicle represents its position and movement in a top-down coordinate system (BANSAL; KRIZHEVSKY; OGALE, 2018). The vehicle's location, heading, and

Figure 7 – The three channels of the BEV representation image that our agent employs, computed at the same instant shown in Fig. 4. From left to right, the channels correspond to: desired route, drivable area, and lane boundaries. The last image shows all three channels combined in different colors.

speed are represented by p_t, θ_t, and s_t respectively. The top-down view is defined so that the agent's starting position is always at a fixed point within an image (the center of it).

Furthermore, it is represented by a set of images of size $W \times H$ pixels, at a ground sampling resolution of ϕ meters/pixel. The BEV of the environment moves as the vehicle moves, allowing the agent to see a fixed range of meters in front of it. For instance, the BEV representation for the vehicle whose three frontal cameras are shown in Fig. 4 is given in Fig. 7, where the desired route, drivable area and lane boundaries form a set of three images (or a three-channel image). Fig. 5 depicts the sparse trajectory visual representation for the same vehicle position position as shown for the BEV and cameras. It is generated on the same coordinate system as the BEV, centered on the vehicle, with the same resolution of ϕ meters/pixel as the BEV.

Fig. 8 includes various visual representations related to city one, such as the city map, the desired route, drivable area, and lane boundaries. These visual representations are used to generate the BEV of the environment by cropping and rotating these large-scale maps to generate a local representation in the vehicle coordinate system. The BEV provides a top-down view of the environment centered on the vehicle's position and orientation, allowing the agent to perceive its surroundings and make informed decisions based on the local information available to it.

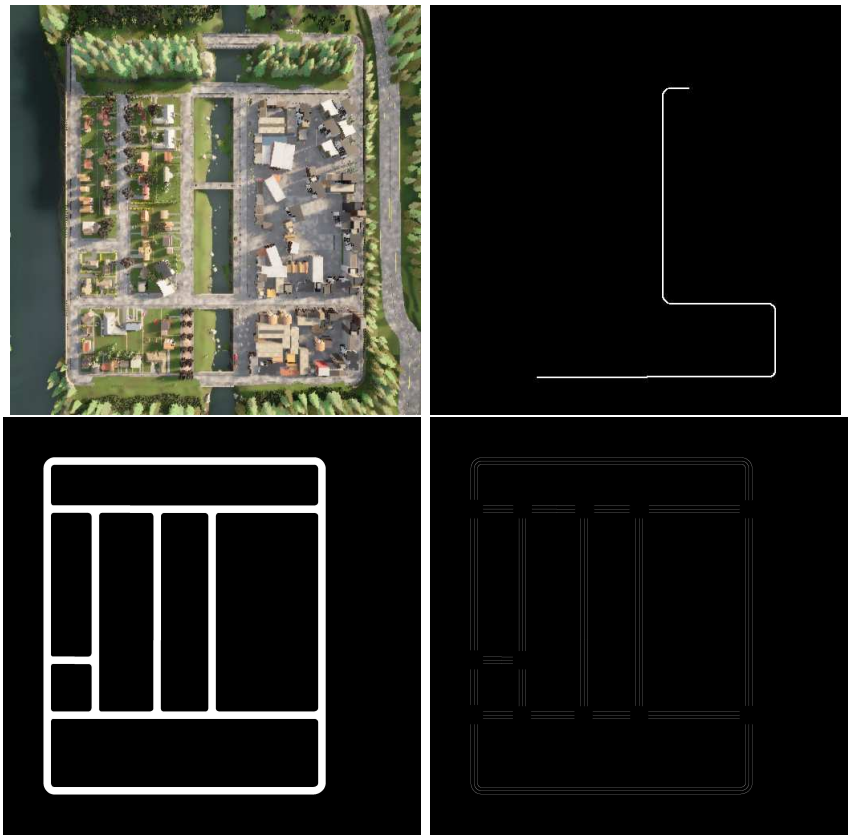


Figure 8 – On the top left the city the map of city one. On the top right the desired route for route one of city one. On the botton left the drivable area of city one. On the botton right the lane boundaries for city one.

4 END-TO-END AGENT

In this chapter, we assess the performance of BC augmented GAIL in an autonomous driving task. BC augmented GAIL is an imitation learning algorithm that derives a policy from expert demonstrations without the necessity of an online agent. Additionally, it produces robust policies that learn from errors during environment exploration. Augmentation with BC directs exploration, thus accelerating and stabilizing the learning process and guiding the policy closer to expert behavior.

To our knowledge, this is the first application of GAIL in a high-fidelity simulator like CARLA for an autonomous driving task. The task was designed to gauge the algorithm's efficacy and learning potential. The chosen task had the agent traverse a pre-set public route from the CARLA Leaderboard, with no other agents present and under uniform weather conditions. The agent successfully completed two variations of this route: the full route and its shortened version.

The results from this chapter were first presented in a conference paper authored by the dissertation author and their advisor (KARL COUTO; ANTONELO, Eric Aislan, 2021).

4.1 AGENT AND NETWORK ARCHITECTURE

4.1.1 Agent

The autonomous car has several sensors, from which we consider: three frontal cameras (Fig. 4), an inertial unit used to compute the vehicle linear speed and angular position, and a GNSS unit for global positioning.

Before training begins, the agent has access to the whole trajectory it must perform, defined as a vector of sparse points and high-level driving commands that characterize the trajectory with no ambiguity. These driving commands can be one from the following in this work:

- LANE_FOLLOW: Continue in the current lane.
- LEFT: Turn left at the intersection.
- RIGHT: Turn right at the intersection.

Thus, the agent can use this trajectory to know which route the car should follow. In practice, this is accomplished by a route planner, that monitors the agent's progress and sends him the next target position in the car's frame of reference as well as the high-level driving command. These two data, totalling 8 dimensions, are given as input to the agent. Notice that the command is input as an one-hot encoded vector.

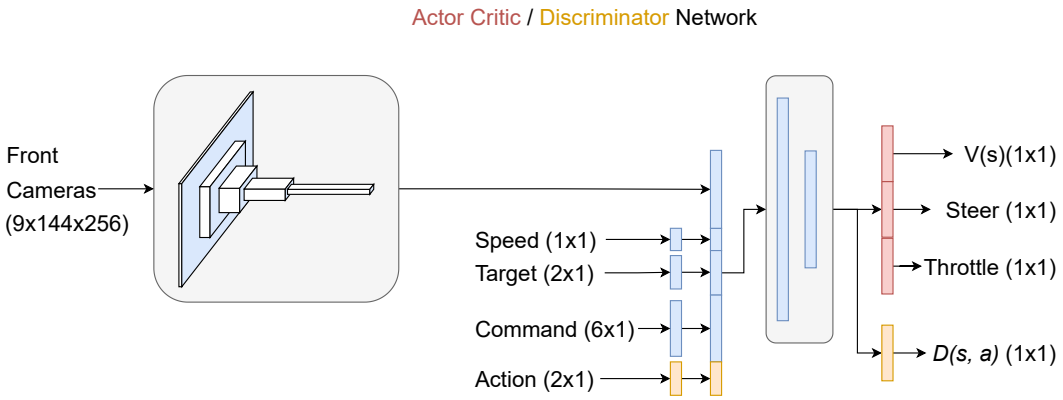


Figure 9 – Architecture of the actor-critic network and discriminator - each of them has its own separate network, with the latter having an additional input for the action, in orange color, and a sigmoidal output $D(s,a)$ instead of the output layer of the actor-critic network which consists of the steering direction, throttle as actions for the actor (policy) and value of the current state $V(s)$ for the critic. The common, though not shared architecture (in blue) is composed of a convolutional block that process the images of the three frontal cameras, whose output features are concatenated with other nine continuous inputs for speed, next target point in the sparse GNSS trajectory, and a high-level driving command. The resulting feature vector is input to a block of two fully-connected (FC) layers.

4.1.2 Networks architecture

The networks represented in Fig. 9 are composed of a convolutional block of four layers, with kernel size of 4 and stride of 2. Each layer in this block is followed by a leaky ReLU activation function, and the numbers of channels starts in 32 on the first layer and is multiplied by 2 on every new layer, ending with 256 channels.

That convolutional block is followed by a fully-connected network block with two layers, with leaky ReLU activation function for the first hidden layer. The second layer represents the output of the architecture.

Both actor-critic and discriminator networks follow that same architecture, although they do not share parameters. The inputs to both networks correspond to 256x140 RGB images from the three frontal cameras. When stacked, these images yield an input with 9 channels, that is fed to the convolution block (Fig. 9). The other continuous input is the car's linear velocity, which is concatenated with the 8-dimensional input from the trajectory as well as to the flattened feature vector from the last convolutional layer. The discriminator has an additional continuous input for the action.

For the actor-critic network, three outputs compose the last layer of the network: a linear unit for the value $V(s)$, a tanh unit for the steering wheel action, and a sigmoid unit for the throttle action, restricting the outputs to the valid domain of these commands (LI; SONG; ERMON, 2017).

4.1.3 Nondeterministic policy

The agent learning process is based on the use of a stochastic policy to calculate action probabilities. This is achieved by using the Gaussian distribution, whose mean is predicted by the policy network, and the standard deviation is fixed to a predefined value (LI; SONG; ERMON, 2017). This was necessary because a variable entropy was shown to be not suitable: the agent with a high entropy is easily disturbed on sensitive moments like a turn, whereas there is not enough exploration during turns if the entropy is too low.

4.2 EXPERIMENTS

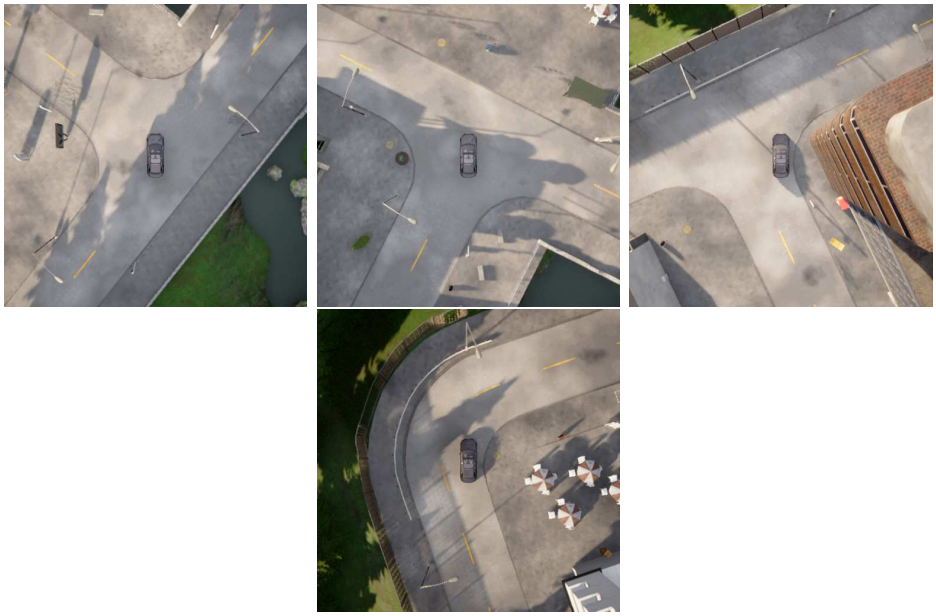


Figure 10 – The top-down view of the simulation with the car in the center and making a turn for the long route. Each picture shows one of the four possible turns, from left to right: left, left, right, and right turns.

The learning navigation experiments are inspired on the CARLA Leaderboard evaluation platform and consists of navigating autonomously on two setups: a short route of 100 meters and one turn (setup 1); and a long route of 2,500 meters and four turns (setup 2). The long route was chosen from the ones available in the CARLA Leaderboard. The short one corresponds to the first 100 meters of the long route.

A top down image from the simulator presenting each turn from the trajectories is displayed on Fig. 10.

4.2.1 Dataset

The expert dataset is built using a deterministic agent that navigates using a dense point trajectory and a classic PID controller (CHEN et al., 2019). While a dense point trajectory provide many points at a finer resolution, a sparse point trajectory is made of

considerably less points to follow, providing just a sense of the right direction to the agent. Thus, the former is used to generate training data by the expert, while the latter is used by the agent for more high-level directions. For instance, the first setup (the short route) considers 80 and 4 points for the dense and sparse trajectories, respectively. The second setup (the long route) uses 760 points in the dense trajectory, and 20 points in the sparse one.

For both setups, 10 complete trajectories were recorded at 10 hertz, i.e., 10 observation-action pairs per second were generated. For the short route, those trajectories correspond to 5 minutes of driving as if in a real scenario, totalling 3,000 training samples (4GB of uncompressed data). For the long route, those trajectories correspond to half an hour of driving, totalling 18,000 training samples (30GB of uncompressed data).

4.2.2 Training

The training was performed using ten parallel actors in a synchronous way, each one running its own CARLA simulator. An eleventh CARLA simulator was also run for evaluation purposes.

In a simulation, every episode starts with the vehicle at zero speed on a particular initial point. The episode ends at every infraction, collision or lane invasion and a new episode starts with the vehicle initially located where the infraction occurred with 90% chance. With 10% chance, the location in the trajectory is randomly chosen, in order to diversify the experience for each policy update.

For the short (long) route, 240 (720) environment interactions or timesteps are recorded for every actor and then the resulting training set of 2,400 (7,200) samples is used to train the parametrized policy in a central computer using ((13)) as loss function. Thus, the episode does not have to end for a policy update to happen. Notice that any one of the ten actors can be interacting with the environment in different parts of the trajectory at a certain moment. Other hyperparameters can be seen in Table 1. For instance, the standard deviation of the Gaussian distribution (σ_1 for steer and σ_2 for throttle) for the policy is fixed to a predefined value. Thus, the output of the policy network only affects the mean of the distribution.

A BC agent is also trained for comparison, using the same dataset available for the GAIL agents, but 70% of the samples are used for training, while 30% for validation. The dataset is not augmented using random rotations or shifting, so that the techniques are compared on their sample efficiency on the same expert trajectories. The BC agent is trained using the loss function from ((10)), and an ADAM optimizer with a learning rate of 3.0×10^{-4}. The network with best validation error is then evaluated in the simulation experiments.

Table 1 – Hyperparameters for training

	Short Route	Long Route
Parallel environments (N)	10	10
Adam step size (lr)	1.0×10^{-4}	1.0×10^{-4}
Number of Proximal Policy Optimization (PPO) epochs (K)	4	4
Mini-batch size (m)	300	900
Discount (γ)	0.99	0.99
GAE parameter (λ)	0.95	0.95
Clipping parameter (ϵ)	0.1	0.1
Value Function coefficient (c_1)	0.5	0.5
Entropy coefficient (c_2)	0.0	0.0
Timesteps per epoch (T)	2400	7200
Log Standard Deviation Steer (σ_1)	-2.0	-2.0
Log Standard Deviation Throttle (σ_2)	-3.2	-3.2

4.3 RESULTS

In order to evaluate the performance of agents, the reward or score metrics is defined as the number of crossed points from the dense trajectory, representing how much of the trajectory the agent has completed without any mistake. Thus, a maximum reward is equivalent to total number of points in the dense trajectory of the particular route.

The learning performance of both GAIL and GAIL augmented with BC (BC_GAIL) can be seen on Figures 11 and 12 for the short and long routes, respectively. In the first setup, BC_GAIL is able to converge significantly faster than GAIL, achieving maximum reward of 80, slightly higher than just BC. On the second more challenging setup which consists of four turns, the learning takes considerably more time. On average, the agent by BC_GAIL was able to complete the route without any mistake much earlier than the GAIL agent, also showing early fast improvement of the policy. This is possible due to the strong influence of the BC term in the loss function in the early part of the training process. Notice that an agent trained only by BC is not able to solve this task (achieved only a reward of 173.8) by training only on the same dataset as GAIL was trained. In addition, after 15×10^5 environment interactions, we can observe that the average reward stabilizes between 500 and 700 for the stochastic policy of both GAIL and BC_GAIL. The spikes seen in Fig. 12 can be caused by random actions of a stochastic policy which can lead to forgetting of some already acquired skills (such as turning at an intersection) or skills that are not well formed yet. For instance, the agent can learn to make a turn at some point and, after some iterations, fail to repeat that behavior, causing a sudden drop of the reward. This happens because turning is a difficult skill to learn, while the reward is proportional to the traveled distance.

The trajectory of the agent for the long route can be viewed in Fig. 13. It shows the early mistakes in red color made by an BC_GAIL agent in the topmost plot. As training proceeds, less and less mistakes are made as it can be noticed in the remaining plots.

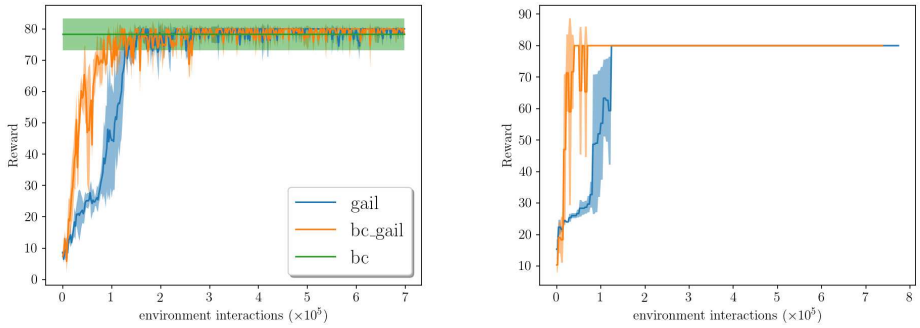


Figure 11 – Average rewards vs environment interactions during training in the short route (setup 1). For each method (GAIL and GAIL with BC), the average performance of three runs (i.e, three agents trained from scratch) is shown with a stochastic policy (top plot) and a deterministic policy (bottom plot). The shaded area represents the standard deviation. The BC agent attains an average reward of 78.3 for ten episodes, while the maximum is at 80, achieved by both GAIL and GAIL augmented with BC.

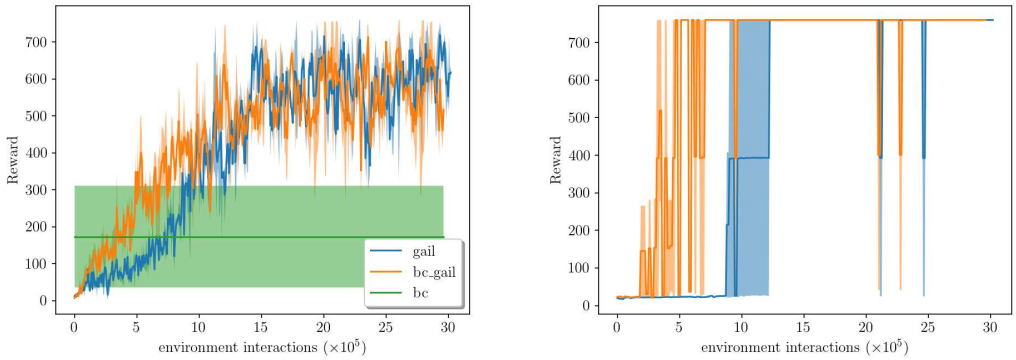


Figure 12 – Average rewards vs environment interactions during training in the long route (setup 2). For each method (GAIL and GAIL with BC), the average performance of two runs (i.e, two agents trained from scratch) is shown with a stochastic policy (left plot) and a deterministic policy (right plot). The shaded area represents the standard deviation. The BC attains an average reward of 173.6 for ten episodes, while the maximum is at 760, achieved by both GAIL and GAIL augmented with BC.

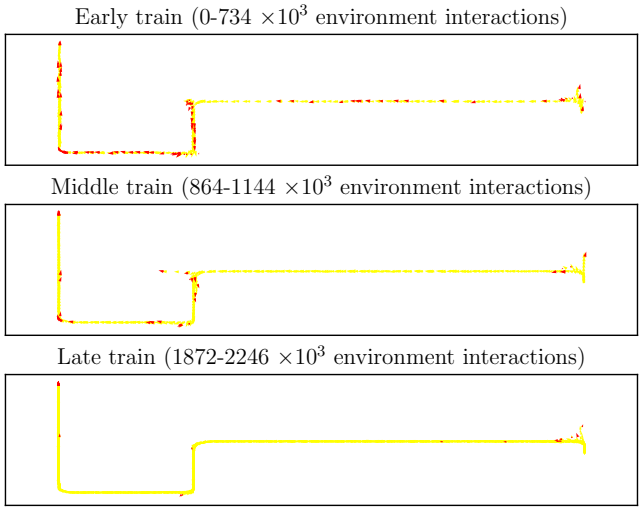


Figure 13 – The vehicle's trajectory, in yellow, for the long route during different moments of the training process. In the early training iterations, errors, marked in red color, are common. As training proceeds, less and less mistakes happen. The trajectory starts at the right side, heading North, and ends at the left side, also heading North.

4.4 DISCUSSION

In this chapter, we have proposed a GAIL-based architecture for end-to-end autonomous driving in urban environments. Despite the known difficulties and learning instabilities of generative adversarial networks, both GAIL and GAIL augmented with BC were able to converge and generate agents able to complete the whole trajectory without mistakes, with the latter able to quickly find a suitable policy when compared to the former. Both of them surpassed BC in performance, which was not able to generate an agent even capable of making more than one turn on average in the long route.

In the following chapter, GAIL augmented with BC is evaluated using a mid-level representation known as the BEV representation. This top-view, semantically-segmented image serves as a more suitable input for driving. By adopting BEV, the emphasis in learning shifts to driving abilities over perception. When using BEV, the agent is subjected to a complex dynamic route task where it must navigate varying paths at given intersections, controlled by the high-level commands and sparse route it receives.

5 BIRD'S-EYE VIEW AGENT

5.1 OVERVIEW

In this chapter, we present an agent trained using a BEV as input. By adopting this approach, the representational capacity of the GAIL agent is honed to focus on mastering driving skills. This is because the agent begins with a more meaningful representation of the environment, enabling it to concentrate on the driving task rather than grappling with intricate perception directly.

Throughout the chapter, we delve into the neural network architecture of the agent, highlighting its input and output representations as well as its stochastic function. Subsequent to this, we introduce an additional exploration loss function that encourages the agent's exploration in specific predetermined directions. This is crucial to narrow down the exploration space and steer the agent towards adopting meaningful behaviors — for example, promoting acceleration if an episode concludes due to the vehicle remaining stationary.

The experimental procedure is then outlined, along with the training setup and hyperparameters. Parallel CARLA environments, utilized to deploy the learning agent, are described. In these CARLA environments, the agent navigates dynamic routes that are generated during training. In terms of evaluation, the agent trained using BEV as input is contrasted against another agent trained with RGB images from cameras instead of BEV.

5.2 POLICY LEARNING WITH GAIL

The generator in the GAIL module iteratively seeks the θ parameters of the policy $\pi_\theta(.|s)$ that minimizes (13), while the discriminator seeks to maximize it. To assist the agent's learning, loss terms for stimulating exploration are added as described after the the representations for the input, output, and architecture are presented.

5.3 INPUT REPRESENTATION

The input s to the agent's policy is a three-channels 192x192 image generated by the environment BEV generator module , corresponding to the mid-level BEV representation of the vehicle in its current position Fig. 7. In addition, the current vehicle's speed and the last value of the policy actuators (last acceleration and steering) are also fed as input further down in the network layers (to the first fully connected layer).

5.4 OUTPUT REPRESENTATION

The vehicle in CARLA has three actuators as: $steering \in [-1,1], throttle \in [0,1]$, and $brake \in [0,1]$. Our agent's action space is $\mathbf{a} \in [-1,1]^2$, where the two components of $\mathbf{a}$ correspond to steering and acceleration. Braking occurs when acceleration is negative. In this way, by modeling brake and throttle with one dimension, the agent is not allowed to brake and accelerate simultaneously (PETRAZZINI; ANTONELO, Eric A, 2021). Instead of using the Gaussian distribution for the policy's actions, common choice in model-free RL, we employ the Beta distribution $\mathcal{B}(\alpha, \beta)$ due to its bounded support, which allows us to model bounded continuous action distributions, usually found in real-world applications such as autonomous driving (PETRAZZINI; ANTONELO, Eric A, 2021), where the action space is not unbounded (i.e., the gas pedal can be actuated up to a certain limit). Besides, the policy loss $\mathcal{L}_P$ can be explicitly computed since clipping or squashing is not used to enforce input constraints (in the case of Gaussian distribution). Furthermore, the Beta distribution allows the policy to act in extreme situations of vehicle driving, where sharp turns and sudden braking are necessary, as its parameters α and β, which are defined as outputs of the policy neural network π_θ and control the shape of the distribution, can be tuned to produce such characteristic vehicle behaviors.

5.5 NETWORKS' ARCHITECTURES

The agent's policy part, has the architecture shown in Fig. 14. The discriminator layers are also shown, even though both network's weights are not shared. The only shared part corresponds to the layers between the Generator and the value function $V_\phi(.)$ until the main branch splits into two heads: one for the actions *steering* and *throttle* for the generator (with 2 *softplus* units that outputs the α and β parameters of a Beta distribution, for each action); and another for the value of state s, given by a linear unit. The discriminator $D(s,a)$ receives an action a in addition to the observation s and maps to a linear output unit, whose output value is employed as reward when training with PPO.

5.6 ENCOURARING EXPLORATION

During training, the agent is encouraged to explore the environment through two objectives, as in (ZHANG, Z. et al., 2021):

$$\mathcal{L}_{\text{ent}} + \mathcal{L}_{\text{exp}} \tag{15}$$

where: the first loss function corresponds to the entropy loss commonly used to promote exploration:

$$\mathcal{L}_{\text{ent}} = -\lambda_{\text{ent}} \cdot \text{H}(\pi_\theta(.|s)), \tag{16}$$

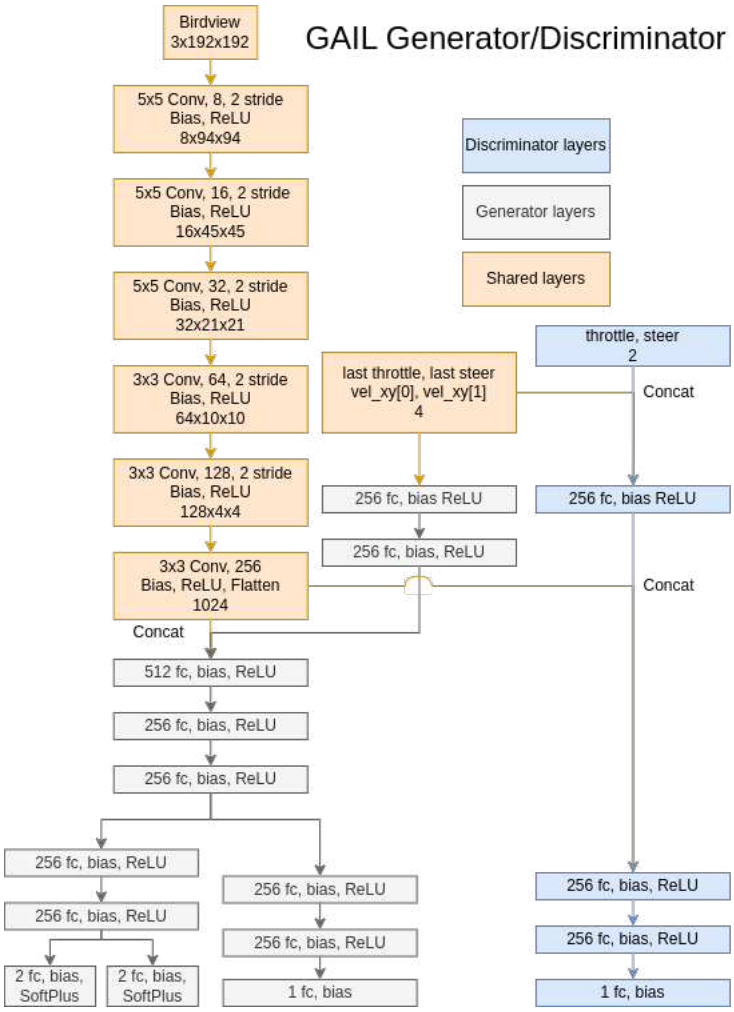


Figure 14 – GAIL architecture for policy learning, corresponding to the Generator network at the right side of Fig. 18 and the Discriminator responsible for producing the reward signal. The Generator, $\pi_\theta(a|s)$, receives the BEV image generated by the environment, the last agent's actions (throttle, steer), and the current speed as input (which forms the observation s of the policy), and outputs the α and β parameters of the Beta distribution for both steering and action with the SoftPlus activation function. The Value function $V_\phi(s)$ shares the Generator network's layers until it branches into a separate head with more two hidden layers and a linear output unit. The Discriminator $D_\omega(s,a)$ receives the actions *throttle* and *steer* in addition to the observation s and has a linear output unit. Notice that features from the last convolutional layer are flattened before they are merged (*concat*) with other information into FC (fully connected) layers.

Minimizing $\mathcal{L}_{\text{ent}}$ means maximizing entropy and thus uncertainty for the policy distribution π_θ, which stimulates the agent try more diverse actions since the policy distribution for a certain state s does not become too certain too quickly in the process. It also drives the action (policy) distribution towards a uniform prior (which represents maximum entropy and uncertainty) since it is equivalent to minimizing the KL-divergence to the uniform distribution defined in the support of the Beta policy $[-1,1]$:

$$\mathrm{H}(\pi_\theta) = -\mathrm{KL}(\pi_\theta || \mathcal{U}(-1,1)), \tag{17}$$

We can also bias the agent's learning with priors that signify meaningful behaviors for an autonomous vehicle and helps to improve and speed up the overall agent's training from scratch. This is accomplished with the following exploration loss $\mathcal{L}_{\text{exp}}$ (ZHANG, Z. et al., 2021):

$$\mathcal{L}_{\text{exp}} = \lambda_{\text{exp}} \cdot \mathbb{1}_{\{T-N_z+1,\ldots,T\}}(k) \cdot \mathrm{KL}(\pi_\theta(.|s) \,||\, p_z), \tag{18}$$

where $\mathbb{1}$ is the indicator function and $z \in \mathcal{Z}$ is the terminal event that finishes the episode. Some examples of events in $\mathcal{Z}$ would be collision, route deviation or the car being still or blocked for too long. $\mathcal{L}_{\text{exp}}$ imposes a prior p_z to the policy during the last N_z steps of an episode ending with one of the events in $\mathcal{Z}$. The indicator function serves as a selection mechanism of the last steps in the episode. This p_z promotes exploration as follows: if z is a collision, $p_z = \mathcal{B}(1,2.5)$ for the acceleration actuator, which encourages slowing down behavior; if the car is still, the acceleration prior is $p_z = \mathcal{B}(2.5,1)$, favoring increasing the vehicle's speed; if the vehicle deviates from the trajectory, a uniform prior $\mathcal{B}(1,1)$ is employed for the steering actuator (ZHANG, Z. et al., 2021).

Thus, uniting (13) and (15), the total loss function for policy learning through PPO for our BC GAIL agent is as follows:

$$\alpha\mathcal{L}_{BC} + (1-\alpha)\mathcal{L}_P + \mathcal{L}_{\text{ent}} + \mathcal{L}_{\text{exp}} \tag{19}$$

5.7 EXPERIMENTAL RESULTS

The goal of the vehicle is to navigate autonomously in the city shown in Fig. 6 using the BC GAIL agent's architecture with mid-level BEV input.

The training was conducted using six parallel actors in a synchronous manner, with each actor running its own instance of the CARLA simulator. In the simulation, each episode begins with the vehicle at zero speed at a random starting point. The episode concludes upon the occurrence of any infraction, collision, or lane invasion, and a new episode begins with the vehicle located at a random point of the map to provide diversified experiences for each policy update.

At every 12,288 environment interactions (steps), the agent's architecture is updated in a central computer: the parametrized policy using loss function (13) is trained for 20

Table 2 – Hyperparameters for GAIL

Description	Value
Parallel environments (N)	6
Initial adam step size (lr)	2.0×10^{-5}
Adam step size exponential decay (λ_{lr})	0.96
Number of PPO epochs (K)	20
Mini-batch size (m)	256
Discount (γ)	0.99
GAE parameter (λ)	0.9
Clipping parameter (ϵ)	0.2
Value Function clipping parameter (ϵ_{vf})	0.2
Value Function coefficient (c_1)	0.5
Entropy coefficient (c_2)	0.01
Exploration coefficient (c_3)	0.05
Timesteps per epoch (T)	12288
GAIL gamma (γ_{gail})	0.004
GAIL gamma decay (λ_{gail})	1.0
Discriminator adam step size (lr)	2.5×10^{-4}
Number discriminator epochs (K)	20

epochs using PPO (K=20), while the GAIL's Discriminator is trained for 2 epochs on these 12,288 samples. This process corresponds to one training cycle of the full BC GAIL. A new cycle will collect the next 12,288 samples from all the actors, and execute the training as described above again. As six parallel actors are used, 2,048 steps or environment interactions per actor are recorded, totalling the 12,288 environment interactions. Thus, the episode does not have to end for a policy update to happen. It is important to note that at any given moment, any of the six actors may be interacting with the environment in different parts of the environment. Additional hyperparameters's values can be found in Table 2.

The training progress can be seen in Fig. 16 for GAIL, and GAIL from cameras agents. The GAIL agent is trained directly on the BEV image computed from the simulator, while the GAIL from cameras one is trained with input coming directly from the three frontal cameras, disregarding any BEV representation. The plot shows the average and standard deviation of the number of infractions for three runs for each agent's stochastic policy.

The evolution of training for the agent in *town1* can also be seen in Fig. 15, where the whole trajectory throughout the city is plot at three different moments in training. Early in the training process, the infractions or errors, given by red triangles, are frequent. These infractions decrease as learning proceeds.

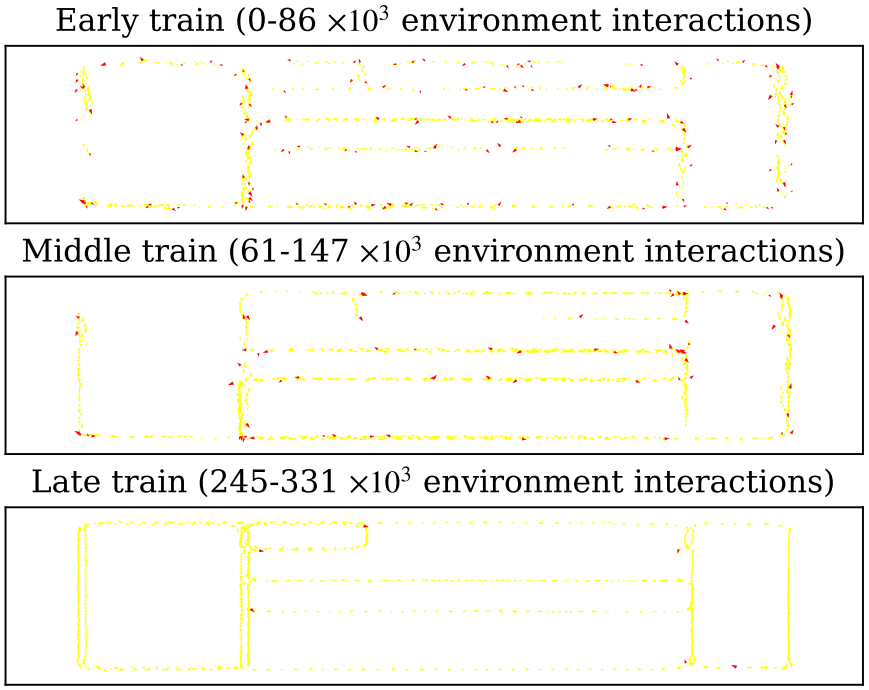


Figure 15 – The vehicle's trajectory, in yellow, during different moments of the training process. In the early training iterations, errors, marked in red color, are common. As training proceeds, less and less mistakes happen.

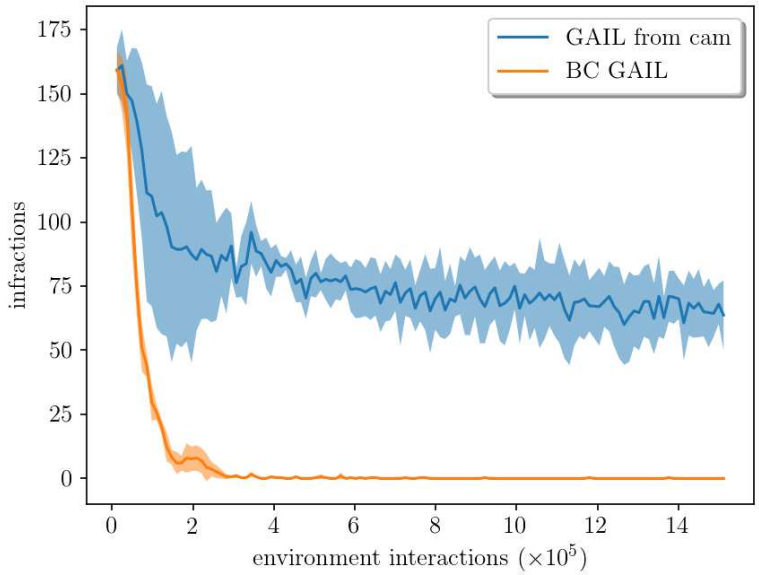


Figure 16 – Number of committed infractions vs. environment interactions during training in *town1* environment. For each method (GAIL from cameras and GAIL from BEV), the average performance of three runs is depicted considering a stochastic policy. The shaded area represents the standard deviation. The GAIL from cameras agent fail to learn the task and keep the sum of committed infractions above zero, while the minimum of zero infractions is achieved by GAIL from BEV.

5.8 DISCUSSION

In this chapter, we have evolved the agent trained using GAIL augmented with BC to utilize a BEV image as input. Additionally, we've adjusted its output stochastic function to better fit a beta distribution and enhanced the agent's loss function to incorporate suggestions for meaningful behavior.

The agent is subsequently trained on a CARLA environment, where it navigates dynamic routes that are generated as the agent progresses through the route. The performance of the GAIL agent, augmented with BC and trained with BEV, is then contrasted with an agent trained using the same algorithm but with RGB images from the vehicle's frontal cameras as input.

We demonstrate that the agent trained with BEV successfully learns to navigate generic routes within the city it was trained in. Conversely, the same agent, when trained using raw camera images as input, struggles to master the task. This highlights the significance of utilizing a mid-level representation and underscores how vital such a representation is for GAIL to triumph in the intricate tasks of autonomous driving.

In the subsequent chapter, we introduce a CGAN based approach to generate BEV images from the raw images captured by the vehicle's frontal cameras. This is aimed at achieving the end-to-end architecture discussed in the previous chapter. Furthermore, we evaluate the agent on public routes from the CARLA Leaderboard in Town 02, demonstrating the agent's capability to choose distinct paths at intersections based on the provided inputs. This is evidenced by an intersection evaluation where the agent, from the same starting point, opts for different routes.

6 HIERARCHICAL GENERATIVE ADVERSARIA IMITATION LEARNING (HGAIL)

In this chapter, we introduce a CGAN that generates semantically segmented BEV images. This generator takes as input RGB images from the vehicle's frontal cameras and the vehicle's sparse trajectory provided by a route manager. We then employ this CGAN to design an end-to-end hierarchical algorithm, which integrates the GAIL from the previous chapter with the CGAN discussed in this chapter. This hierarchical, end-to-end algorithm is subsequently trained on CARLA, using the same setup as the one employed for training the GAIL with BEV in the last chapter.

We then test the generalization of the end-to-end algorithm, which was trained on dynamic routes from CARLA's Town 01, using the CARLA Leaderboard's public routes from Town 02. In the end, the algorithm demonstrates its capability to be directed at intersections—namely, to take varied paths at intersections based on the inputs given to the agent. This is evidenced by an intersection evaluation in Town 02.

6.1 BEV GENERATION WITH CGAN

The CGAN module, used to transform the images from the frontal cameras into a top-down view representation, has two different networks named Discriminator and Generator, whose architectures can be seen in Fig. 17. In this figure, all layers from both networks are presented, and the *common* layers in the orange color refer to layers that exist in both networks' architectures, even though they do not share weights (parameters).

6.1.1 Input representation

The input for the CGAN corresponds to the 192x192 resolution RGB images from the three frontal cameras represented in Fig. 4 and the sparse trajectory visual representation from Fig. 5 that are stacked to generate a 10x192x192 image, i.e., with 10 channels. The goal of the CGAN's generator is to translate this stack of images into the 3-channel BEV representation seen in Fig. 7. In addition to this RGB input image, the discriminator also receives the 3x192x192 BEV image, which can come from either the generator as *fake* or from the training set as *real*.

6.1.2 Generator

It can be seen in this figure and also in Fig. 18 that the CGAN's generator is a U-Net (RONNEBERGER; FISCHER; BROX, 2015), usually employed for image translation or segmentation. Further, while the image is processed by convolution layers, the other perceptual inputs (trajectory and command, second column in the figure) are processed by two fully connected layers followed by two transposed convolution layers which upsample

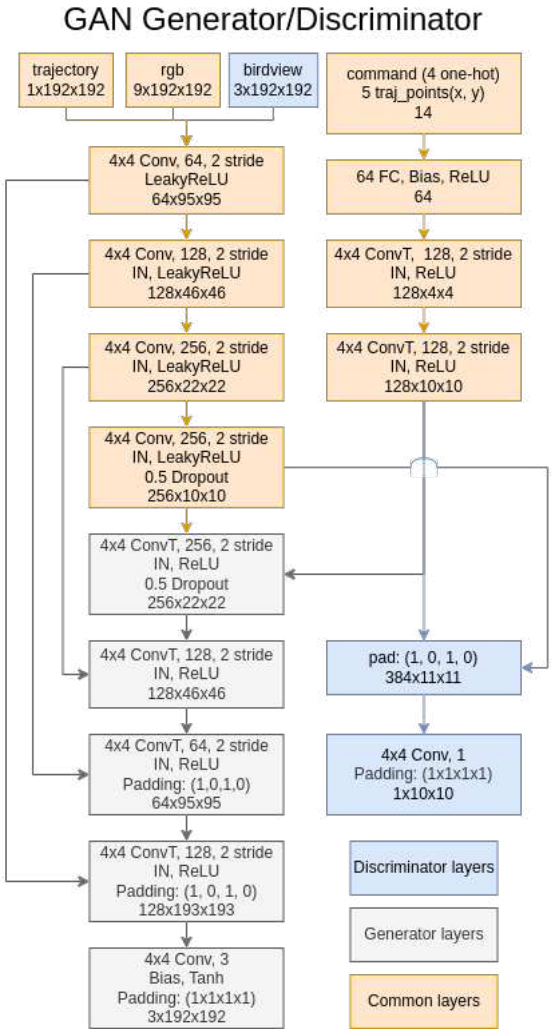


Figure 17 – CGAN architecture for generating the BEV input representation. The Generator and the Discriminator are separate networks which do not share parameters: the figure was made to not repeat equivalent layers when describing both networks. The generator corresponds to the U-net at the left side of Fig. 18 and aims at translating RGB 9x192x192 images from the vehicle's frontal cameras to BEV mid-level input representation (3x192x192 images).

their input to reach the desired resolution so that it can be merged with the last orange 256x10x10 layer in the left column. The next transposed convolution grey layer (256x22x22) merges information coming from the frontal cameras's RGB images (left column) and the trajectory points plus the command (right command) for the generator network. Its final output is 3x192x192, corresponding to the three-channels BEV translated image.

6.1.3 Discriminator

The discriminator is also conditioned on the RGB images from the frontal cameras and the visual trajectory which is merged to the (fake/real) BEV image, totalling 13x192x192 input to the first convolutional layer of the discriminator. The other perceptual inputs (right column) are processed similarly to the generator until it merges in a

new 384x11x11 layer (in blue) with information coming from the images (256x10x10, left column). The final output corresponds to the one given by PatchGAN.

6.1.4 Network Architecture

Both networks' architectures are seen in Fig. 17, where the common layers in orange refer to layers existing in both the generator and the discriminator. Notice that they are separate networks which do not share parameters: the figure was made to not repeat equivalent layers when describing both networks.

6.2 AGENT

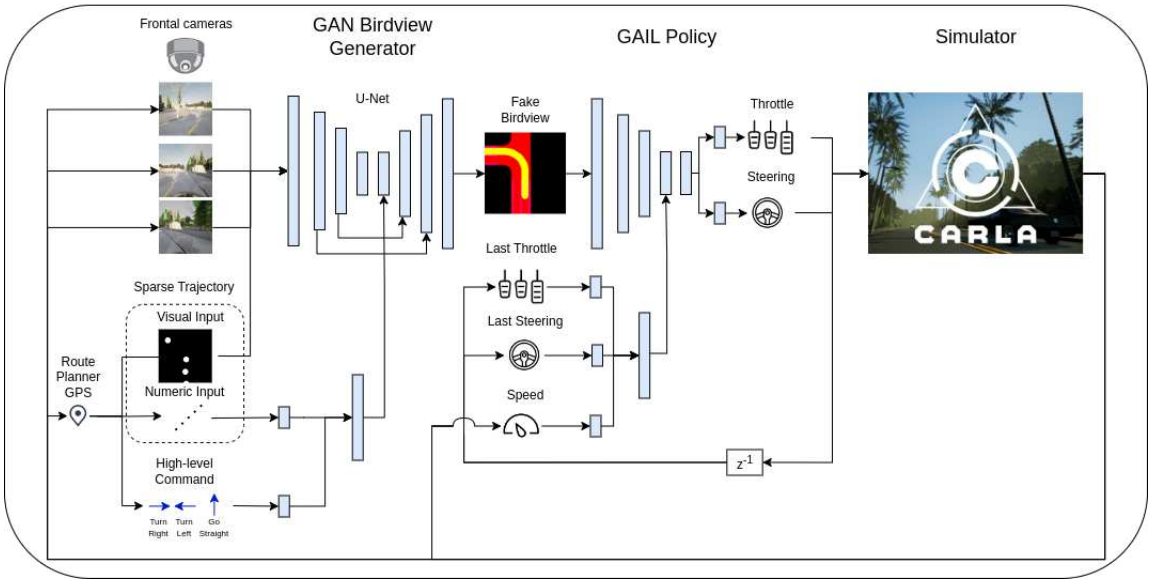


Figure 18 – Hierarchical Generative Adversarial Imitation Learning (hGAIL) for policy learning with mid-level input representation. It basically consists of chained CGAN and GAIL networks, where the first one (CGAN) generates BEV representation from the vehicle's three frontal cameras, sparse trajectory and high-level command, while the latter (GAIL) outputs the acceleration and steering based on the predicted BEV input (generated by CGAN), the current speed and the last applied actions. Both CGAN and GAIL learn simultaneously while the agent interacts to the CARLA environment. The discriminator parts of both networks are not shown for the sake of simplicity.

Our agent's architecture (Fig. 18) is based on hierarchical Generative Adversarial Imitation Learning (hGAIL) for training policy and mid-level representation simultaneously. There are two main parts of hGAIL: the CGAN that generates the BEV representation based on input from the vehicle's frontal cameras, trajectory and high-level command; and the GAIL that learn the agent's policy by imitation learning based on input from the BEV representation generated by the first CGAN module, current vehicle's speed, and the last actuator values.

6.3 TRAINING

The agent is trained on the dyanmic route environment using the Algorithms (1) and (2). It is worth noting that the CGAN's generator of hGAIL is pretrained on the fixed set of the ten expert trajectories (with 21,287 pairs of input and BEV targets) for 4 epochs in a supervised way. Additional hyperparameters's values can be found in Table 3.

The training was conducted using six parallel Carla simulators on a two-node GPU cluster. One node, equipped with an RTX 3060, was dedicated to the simulators, while the main node, with an RTX 3080, was dedicated to training the network. Training hGAIL for 1.5 million environment steps took the cluster a total of 70 hours. For inference, considering the simulator to generate new samples, the cluster achieved a performance of 14 FPS.

Algorithm 1: Conditional Generative Adversarial Network training

Input: Transitions buffer $\mathcal{B}$;
Input parameters: Generator: δ_g and Discriminator: δ_d;
for $k = 1, 2, \ldots, K$ **do**
  // x_t are CGAN $G(.)$'s inputs; y_t is the true BEV
  Sample $\left\{(x^{(i)}, y^{(i)})\right\}_{i=1}^{m}$ from transition buffer $\mathcal{B}$;
  Update CGAN $G_{\delta_g}(x)$ by minimizing (1);
  Update CGAN $D_{\delta_d}(x,y)$ by maximizing (1);
end

In Algorithm (2), x^t are the inputs for CGAN generated by the environment. y^t is the truth ground BEV representation generated by the environment. $G\left(x^{(t)}\right)$ is the BEV representation generated by the CGAN. D_ω is updated using y true ground BEV as state s. π_θ is updated using CGAN BEV $G\left(x^{(t)}\right)$ as state s.

6.4 EVALUATION

The main evaluation environment is *town02*, shown in Fig. 19. It was used to test how well the hGAIL agent can generalize its driving skills to unseen, new environments. All experiments below consider agents trained exclusively in *town01* environment.

The resulting deterministic policies of each agent trained in *town1* for all three runs are also evaluated in *town2* as training evolves, as shown in Fig. 20. It shows the average percentage of completed routes from a total of six Leaderboard routes in *town2* as learning proceeds. Each run uses a different agent trained exclusively in *town1*. Both hGAIL and GAIL with real BEV are able to generalize the learning in *town1* to *town2*.

In addition to the above two agents, GAIL from cameras agent was also tested. The agents is not shown in the plot, since it fails to learn to complete any route (staying at 0% if shown in Fig. 20).

Algorithm 2: Hierarchical Generative Adversarial Imitation Learning

Input: Expert transitions buffer $\mathcal{B}_E$;
Input parameters: Policy Actor: θ, Critic: ϕ, Discriminator: ω ;
Pretrain CGAN (1) using samples from $\mathcal{B}_E$;
for *episode* $= 1, 2, \ldots$ **do**
  `// Collect trajectory samples from the environment`
  for $t = 1, 2, \ldots, T$ **do**
    `//` x_t `are CGAN` $G(.)$`'s inputs;` y_t `is the true BEV`
    Choose action $a_t \sim \pi_\theta\left(G\left(x_t\right)\right)$; $v_t \leftarrow V_\phi\left(G\left(x_t\right)\right)$;
    $x_{t+1}, y_{t+1} \leftarrow act(a_t)$;
    Add $\left(x_t, y_t, G\left(x_t\right), a_t, v_t\right)$ to the buffer $\mathcal{B}_\pi$;
  end
  for $j = 1, 2, \ldots, J$ **do** `/* Update GAIL's discriminator */`
    Sample $\{(y^{(i)}, a^{(i)})^\pi\}_{i=1}^m$ and $\{(y^{(i)}, a^{(i)})^E\}_{i=1}^m$ from policy transitions buffer $\mathcal{B}_\pi$ and expert transitions $\mathcal{B}_E$;
    Update the policy discriminator parameters w to increase (11);
  end
  Compute advantage $A_{t\in\{1,2,\ldots,T\}}$ based on advantage function $A_{\omega,\phi}$ ((8)) and add to policy transitions buffer $\mathcal{B}_\pi$;
  for $k = 1, 2, \ldots, K$ **do** `/* Update agent using PPO */`
    Sample $\left\{\left(G\left(x^{(i)}\right), a^{(i)}, A^{(i)}\right)^\pi\right\}_{i=1}^m$ from policy transitions buffer $\mathcal{B}_\pi$;
    Update the policy generator parameters θ to optimize (19);
  end
  Train CGAN (1) using samples from $\mathcal{B}_\pi$;
end

Table 3 – Hyperparameters for CGAN

Description	Value
Adam step size (lr)	2.0×10^{-4}
Number of GAN epochs (K)	4
Mini-batch size (m)	32
Patch size (γ)	$(10, 10)$
Resize (λ)	$(192, 192)$
Lambda pixel (ϵ)	100

Figure 19 – *Town02* environment of the agent, with one of the routes used to evaluate the agent trained in *town02*. The highlighted path has 1010 meters, 29 points in the sparse trajectory (shown as yellow dots) and 1030 points in the dense point trajectory (not shown).

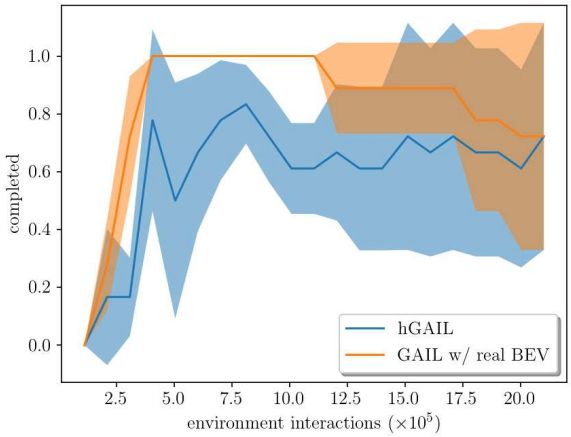


Figure 20 – Evaluation of agents in *town2*, trained exclusively in *town1*. The plot shows the percentage of completed routes from a total of six Leaderboard routes in *town2* vs. environment interactions, averaged over three different runs, where each run entails a different agent trained only in *town1*. For each method (hGAIL, GAIL with real BEV), the average performance of three runs is depicted considering a deterministic policy. The shaded area represents the standard deviation. Both hGAIL and GAIL with real BEV are able to generalize the learning in *town1* to *town2*.

6.4.1 Intersection Evaluation

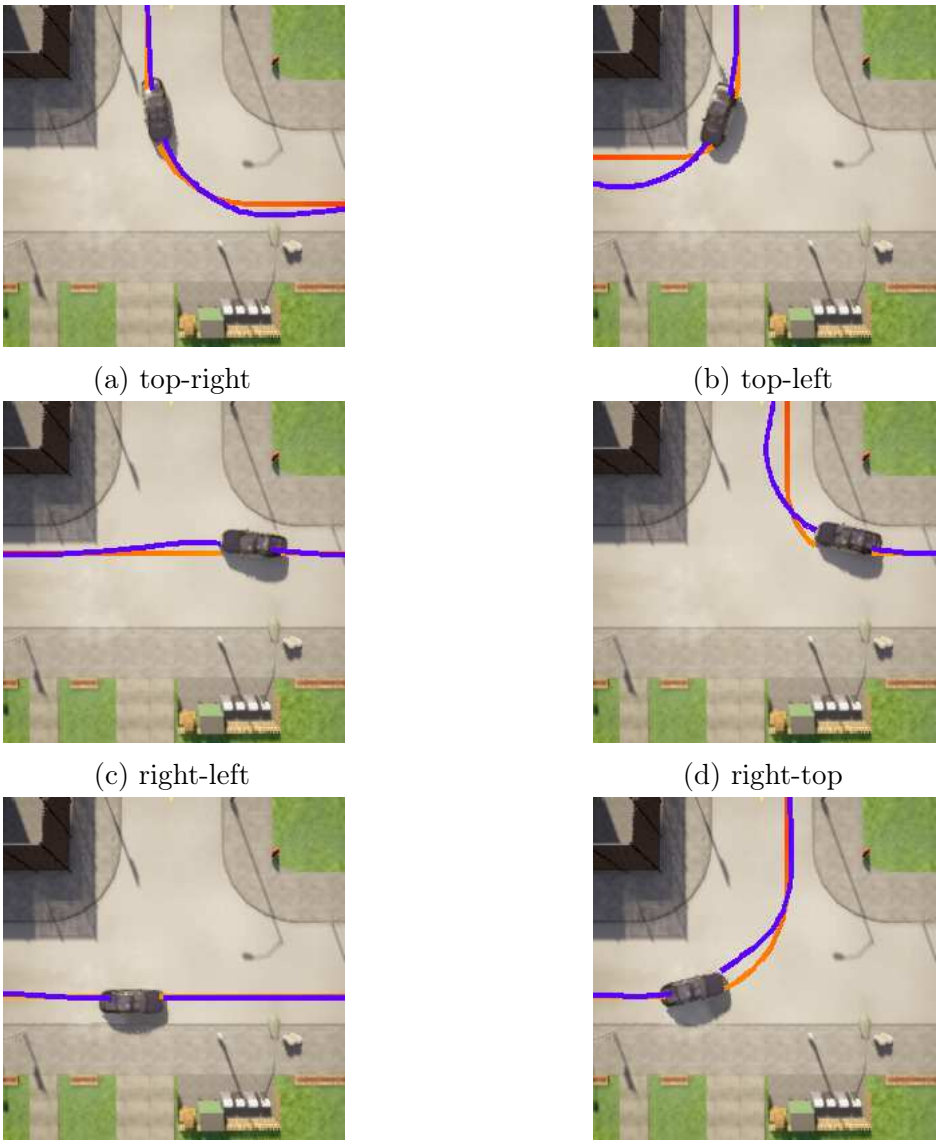
After training, the agent was also evaluated at a given T intersection and compared to the target given by the expert. Fig. 21 shows the resulting trajectories, with blue and orange denoting the agent's and expert's trajectories, respectively. It is worth noting that the policy's network in the hGAIL agent receives as input only the generated (fake) BEV mid-level image, the current speed, and last applied actions for throttle and steering. For instance, this BEV image corresponds to the topdown image with three channels from Fig. 7. It is important to observe that the only information denoting the desired movement for the agent comes from the yellow desired route in the drivable red area. This yellow route occupies the whole lane in the BEV image, which leaves open how the agent will learn to turn at certain intersections. In other words, the agent's policy can not see directly the points in the sparse trajectory, as these points are fed to the CGAN part of the architecture and not to the policy. This means that how we terminate the episode, such as through infractions and lane invasion, will influence to a great extent the type of behavior the agent learns. Such an example can be seen in the turns of Fig. 21, where the agent's trajectory does not match exactly with the expert's one.

6.4.2 Intersection Evaluation Results

The agent trained only on *town1* was also evaluated at every T intersection in *town2* environment, i.e., 8 different T intersections, and compared to GAIL with real BEV and GAIL from cameras. The results are summarized in Table 4, whose lines presents the results for each possible turn out of 6 in total at a given T intersection (as shown in Fig. 21). Thus, each turn, covering around 100 meters, was evaluated in 8 different T intersections, totalling 48 experiments for each agent. The success percentage for each turn type is given in this table, where we can see that hGAIL can turn without failing in all intersections and for all turn types except for one *top-right* intersection, while GAIL with real BEV succeded in all turns without exception and GAIL from cameras fails to learn most of the required driving behavior, succeeding only in 4 turns out of 48. This ablation of the CGAN from hGAIL (which is the GAIL from cameras) shows the need for learning the mid-level input representation to succeed in this complex task. Additionally, notice that only hGAIL and GAIL with real BEV can finish complete trajectories (of around 1 km, as shown in Fig. 20), as GAIL from cameras fail in at least one turn type.

6.4.3 Mid-level representation learning

Here, we present some results of the BEV representation learned by the CGAN's generator of the hGAIL agent's architecture. While this CGAN learns in *town01*, Fig. 22 shows the evolution of the representations of five different vehicle's positions taken in *town02* at different training epochs of the agent in *town1*, where each row corresponds



(a) top-right (b) top-left

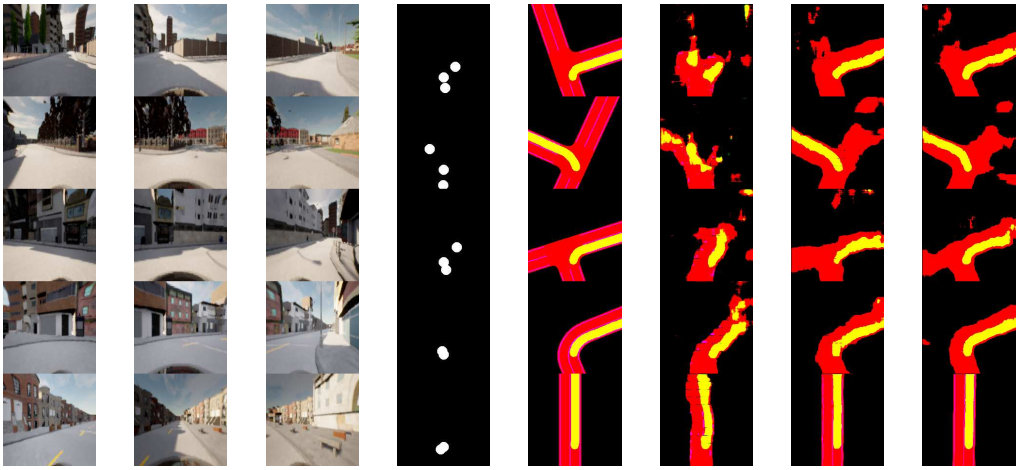
(c) right-left (d) right-top

(e) left-right (f) left-top

Figure 21 – Agent's trajectories in *town2* in blue color generated by the deterministic policy after training in *town1* (at epoch 100) superimposed on the expert trajectory in orange color. At the same T intersection, 6 possible movements are possible: from top to right, top to left, right to left, right to top, left to right and left to top.

Table 4 – Evaluation performance for 8 T Intersections and 6 type of turns in Town2

Turn type	hGAIL	GAIL with real BEV	GAIL from cam.
Top-right	88%	100%	0%
Top-left	100%	100%	0%
Right-left	100%	100%	25%
Right-top	100%	100%	13%
Left-right	100%	100%	13%
Left-top	100%	100%	0%
All types	98%(47)	100%(48)	8%(4)



(a) left (b) cent. (c) right (d) traj. (e) target (f) 1 cy. (g) 16 cy. (h) 32 cy.

Figure 22 – BEV generation as the agent goes through training. The first three colmns shows the images from the cemeras attached to the front of the vehicle. The fourth column show the plot from the point from the route planner The fifth column shows five BEV images computed by the simulator and are considered the target output. The following columns show the BEV images generated by the CGAN from the agent's architecture as it undergoes training, at: 12,288 environment steps (1 cycle), 208,896 environment steps (16 cycles), and 405,504 environment steps (32 cycles). One cycle is similar to the concept of epoch, and consists of the full training of hGAIL using the last 12,288 steps collected; however, each individual network of hGAIL is trained for different number of epochs in one training cycle (see Section 5.7).

to a different particular position of the vehicle in the *town02* environment. The first four columns are the input to the CGAN's generator, consisting of the images from the three frontal cameras and the sparse trajectory given as an image. The fifth column corresponds to the targets (labels), i.e., the BEV generated by the simulator, which is used to train the CGAN's generator. The other columns show the mid-level BEV representation evolving from a poor prediction at cycle 1 (after 12,288 environment steps) to a good enough prediction at cycle 32. It is worth noticing that the CGAN has never seen *town02*, and was trained only on *town01*.

6.5 DISCUSSION

In line with the principles of end-to-end learning discussed in Chapter 4, we introduced the hGAIL framework. This architecture addresses autonomous vehicle navigation using an end-to-end methodology, linking sensory perceptions directly to low-level vehicular actions through neural networks, a process we refer to as sensory-motor coupling. Concurrently, it captures mid-level input representations of the vehicle's surroundings.

Furthermore, we presented two novel evaluation tasks, challenging the agent to traverse an unfamiliar city. In the first evaluation, the agent was tasked with navigating

intricate paths based on the CARLA Leaderboard public routes. Notably, both our agents trained with BEV managed to successfully navigate these routes. This highlights the adeptness of both agents in generalizing their navigation abilities to new and unseen terrains. Additionally, this evaluation confirmed that there was no degradation in agent performance when relying on the BEV created by the CGAN, as compared to the BEV sourced directly from the environment, emphasizing the high-quality BEV produced by the CGAN.

The subsequent assessment examined the agents' proficiency in executing various turns at the eight T-junctions in the new city. During this evaluation, hGAIL distinctly demonstrated its prowess in navigation and adaptability. The value of the mid-level inputs crafted by the CGAN became evident when the GAIL, depending solely on camera inputs and lacking BEV, missed most of the 48 potential turns across the eight T-junctions of that city.

7 CONCLUSION

In this work, we conducted experiments to evaluate imitation learning methods, with a particular focus on those based on GAIL, for autonomous driving tasks. Initially, we tested the limits of an end-to-end algorithm for navigation on CARLA. Although the algorithm successfully solved the task, we had to simplify the general problem of autonomous driving to a fixed trajectory navigation task on a high-fidelity simulator.

Subsequently, we made advancements by adopting a more intelligent approach to leverage GAIL's learning capacity for autonomous driving. Using a BEV semantic map as the input representation helped alleviate the burden of extracting meaningful information from raw sensors for the GAIL agent. As a result, the agent was capable of navigating dynamic trajectories that were created during both training and evaluation.

However, training the GAIL to drive from BEV segmented images raised concerns about obtaining such representations in the real world, where privileged data from simulators is unavailable. To address this, we proposed a CGAN to generate semantic maps from frontal cameras attached to the vehicle. We also developed a hierarchical algorithm to train this CGAN in parallel with the agent, which consumes the images generated by the CGAN. This algorithm was successfully tested in a more challenging scenario where the agent needed to navigate an environment it had never encountered before.

7.1 FUTURE WORK

Despite these achievements, several perspectives of the broader autonomous driving problem were not explored in this dissertation. Notably, the impact of a more dynamic environment with different weather conditions, the implications of operating in a multi-agent environment, where the agent interacts with other agents and adheres to common driving norms, such as stopping at red traffic lights, remain open topics for further investigation.

Additionally, the real-to-sim gap presents an interesting perspective. Using the BEV abstraction to train an agent in simulation and deploy it to the real world, where BEV provides a common ground to make the agent's behavior invariant to the environment, holds potential for future research.

REFERENCES

BANSAL, Mayank; KRIZHEVSKY, Alex; OGALE, Abhijit. **ChauffeurNet: Learning to Drive by Imitating the Best and Synthesizing the Worst**. [S.l.]: arXiv, 2018. Available from: https://arxiv.org/abs/1812.03079.

CHEN, Dian; ZHOU, Brady; KOLTUN, Vladlen; KRÄHENBÜHL, Philipp. **Learning by Cheating**. [S.l.: s.n.], 2019. arXiv: 1912.12294 [cs.RO].

CODEVILLA, Felipe; MÜLLER, Matthias; LÓPEZ, Antonio; KOLTUN, Vladlen; DOSOVITSKIY, Alexey. **End-to-end Driving via Conditional Imitation Learning**. [S.l.: s.n.], 2018. arXiv: 1710.02410 [cs.RO].

GULRAJANI, Ishaan; AHMED, Faruk; ARJOVSKY, Martin; DUMOULIN, Vincent; COURVILLE, Aaron. **Improved Training of Wasserstein GANs**. [S.l.: s.n.], 2017. arXiv: 1704.00028 [cs.LG].

ISOLA, Phillip; ZHU, Jun-Yan; ZHOU, Tinghui; EFROS, Alexei A. Image-to-Image Translation with Conditional Adversarial Networks. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). [S.l.: s.n.], 2017. P. 5967–5976.

JENA, Rohit; LIU, Changliu; SYCARA, Katia. **Augmenting GAIL with BC for sample efficient imitation learning**. [S.l.: s.n.], 2020. arXiv: 2001.07798 [cs.LG].

KARL COUTO, Gustavo Claudio; ANTONELO, Eric Aislan. Generative Adversarial Imitation Learning for End-to-End Autonomous Driving on Urban Environments. In: 2021 IEEE Symposium Series on Computational Intelligence (SSCI). [S.l.: s.n.], 2021. P. 1–7.

LI, Yunzhu; SONG, Jiaming; ERMON, Stefano. **InfoGAIL: Interpretable Imitation Learning from Visual Demonstrations**. [S.l.: s.n.], 2017. arXiv: 1703.08840 [cs.LG].

LIANG, Xiaodan; WANG, Tairui; YANG, Luona; XING, Eric. CIRL: Controllable Imitative Reinforcement Learning for Vision-Based Self-driving. In: FERRARI, Vittorio; HEBERT, Martial; SMINCHISESCU, Cristian; WEISS, Yair (Eds.). **Computer Vision – ECCV 2018**. Cham: Springer International Publishing, 2018. P. 604–620.

MÜLLER, Matthias; DOSOVITSKIY, Alexey; GHANEM, Bernard; KOLTUN, Vladlen. Driving policy transfer via modularity and abstraction. **arXiv preprint arXiv:1804.09364**, 2018.

OTA, Kei; JHA, Devesh K; KANEZAKI, Asako. Training larger networks for deep reinforcement learning. **arXiv preprint arXiv:2102.07920**, 2021.

PETRAZZINI, Irving GB; ANTONELO, Eric A. Proximal Policy Optimization with Continuous Bounded Action Space via the Beta Distribution. In: IEEE. 2021 IEEE Symposium Series on Computational Intelligence (SSCI). [S.l.: s.n.], 2021. P. 1–8.

PHILION, Jonah; FIDLER, Sanja. Lift, splat, shoot: Encoding images from arbitrary camera rigs by implicitly unprojecting to 3d. In: SPRINGER. COMPUTER Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIV 16. [S.l.: s.n.], 2020. P. 194–210.

REIHER, Lennart; LAMPE, Bastian; ECKSTEIN, Lutz. **A Sim2Real Deep Learning Approach for the Transformation of Images from Multiple Vehicle-Mounted Cameras to a Semantically Segmented Image in Bird's Eye View**. [S.l.: s.n.], 2020. arXiv: `2005.04078 [cs.CV]`.

RONNEBERGER, Olaf; FISCHER, Philipp; BROX, Thomas. U-net: Convolutional networks for biomedical image segmentation. In: SPRINGER. INTERNATIONAL Conference on Medical image computing and computer-assisted intervention. [S.l.: s.n.], 2015. P. 234–241.

SUTTON, Richard S.; BARTO, Andrew G. **Reinforcement Learning: An Introduction**. Second. [S.l.]: The MIT Press, 2018.

TENG, Siyu; CHEN, Long; AI, Yunfeng; ZHOU, Yuanye; XUANYUAN, Zhe; HU, Xuemin. Hierarchical Interpretable Imitation Learning for End-to-End Autonomous Driving. **IEEE Transactions on Intelligent Vehicles**, v. 8, n. 1, p. 673–683, 2023.

XIAO, Huang; HERMAN, Michael; WAGNER, Joerg; ZIESCHE, Sebastian; ETESAMI, Jalal; LINH, Thai Hong. **Wasserstein Adversarial Imitation Learning**. [S.l.: s.n.], 2019. arXiv: `1906.08113 [cs.LG]`.

ZHANG, Ming; WANG, Yawei; MA, Xiaoteng; XIA, Li; YANG, Jun; LI, Zhiheng; LI, Xiu. Wasserstein Distance guided Adversarial Imitation Learning with Reward Shape Exploration. **2020 IEEE 9th Data Driven Control and Learning Systems Conference (DDCLS)**, IEEE, 2020.

ZHANG, Zhejun; LINIGER, Alexander; DAI, Dengxin; YU, Fisher; VAN GOOL, Luc. End-to-End Urban Driving by Imitating a Reinforcement Learning Coach. In: 2021 IEEE/CVF International Conference on Computer Vision (ICCV). [S.l.: s.n.], 2021. P. 15202–15212.