Mathematisch-
Naturwissenschaftliche
Fakultät

Lernbasierte Computer Vision

Masterarbeit

Expert Drivers for Autonomous Driving

Eberhard Karls Universität Tübingen
Mathematisch-Naturwissenschaftliche Fakultät
Wilhelm-Schickard-Institut für Informatik
Lernbasierte Computer Vision
Bernhard Jaeger, bernhard.jaeger@student.uni-tuebingen.de, 2021

Bearbeitungszeitraum: 31.03.2021-30.09.2021

Betreuer/Gutachter: Prof. Dr. Andreas Geiger, Universität Tübingen
Zweitgutachter: Prof. Dr. Andreas Zell, Universität Tübingen

Selbstständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbständig und nur mit den angegebenen Hilfsmitteln angefertigt habe und dass alle Stellen, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen sind, durch Angaben von Quellen als Entlehnung kenntlich gemacht worden sind. Diese Masterarbeit wurde in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt.

Bernhard Jaeger (Matrikelnummer 5483684), September 27, 2021

Abstract

A popular approach to self-driving is imitation learning, in which a neural network is trained to predict the actions of an expert driver for a given input. For research performed on simulators, these expert drivers are usually computer programs that have privileged access to the simulation. While these approaches are more cost-efficient than using labels from humans, their quality is worse because existing expert driver approaches are not able to safely drive through urban environments. We improve upon prior work and propose SEED, a Simple and Effective Expert Driver that can safely navigate through urban environments even under challenging adversarial traffic scenarios. We show its utility by training an existing imitation learning architecture to imitate SEED. The resulting TransFuser+ method sets a new state-of-the-art on the challenging NEAT validation routes and outperforms the best prior work on the CARLA leaderboard.

Acknowledgments

I want to thank my parents and grandmother who made this work possible by funding my living costs. Furthermore, I want to thank Kashyap Chitta for supervising the project and his help with generating the dataset. I want to thank Andreas Geiger for helpful discussions and Aditya Prakash for his help with making submissions to the CARLA leaderboard.

Contents

1 Introduction

Autonomous vehicles are a promising technical solution to important problems in transportation. Every year more than a million people die due to traffic accidents [1] primarily caused by human error [2]. Automating driving has the potential to drastically reduce these accidents. Additionally, self-driving cars could improve the mobility of people who are not able to drive themselves. The use of supervised machine learning has become the dominant approach to autonomous driving because it can handle high-dimensional sensor data such as images well. To train machine learning algorithms in an end-to-end fashion, meaning directly optimizing a neural network to perform the full driving task, one needs demonstrations from an expert driver. Industrial research often collects data from human expert drivers, but this approach is expensive in terms of money and time.
Simulations [3, 4] are frequently used to perform research on autonomous driving because new ideas can safely be tested in them. In simulations, an alternative to human experts called privileged experts is available to perform the data collection task. Privileged experts are computer programs that have direct access to the simulator (e.g. knowing the positions of all cars), circumventing the challenging perception task. These privileged experts can generate labeled data faster than human experts and at basically no cost.
However, existing privileged expert approaches are flawed in that they cannot drive safely without causing collisions or violating traffic laws. Existing rule based approaches are too simplistic and lack in implementation quality, whereas learning based approaches fail to generalize to all scenarios. The result is a reduced quality of the labels in the dataset. This leads to reduced performance of the imitation learning agents and as a result the community is currently underestimating existing imitation learning architectures.
In this work, we propose a **S**imple and **E**ffective **E**xpert **D**river (SEED) that can safely navigate in a realistic 3D simulator with adversarial traffic scenarios. We show that it outperforms the prior best privileged expert approaches [5, 6] on the challenging NEAT validation routes [7]. To demonstrate the utility of SEED, we train an existing sensorimotor architecture (TransFuser [6]) with its labels. The improved TransFuser sets a new state-of-the-art on the NEAT validation routes, with no changes made to its inference architecture. It also achieves competitive results on the CARLA leaderboard similar or better than the current top methods [8, 9] which combine reinforcement learning with supervised learning.

Our contributions are as follows:

- We propose an expert driver method that outperforms prior approaches and is able to safely drive in a realistic 3D urban driving simulator with adversarial scenarios.
- We demonstrate its utility by training an existing approach with its data and show that it improves the driving performance of the architecture
- We show how the *inertia problem* [10] can be addressed by a simple controller mechanism, allowing researchers to make and measure progress despite its existence.
- We investigate the impact of the different components in an ablation study.

The rest of the work is structured as follows. In chapter 2 we discuss existing approaches to self-driving, including end-to-end imitation learning approaches and modular pipelines. We also introduce reinforcement learning, its application to autonomous driving, and we discuss existing expert driver approaches. In chapter 3 we describe the problem setting in more detail, introduce the SEED approach and the relevant techniques used. We also review the TransFuser architecture and describe some improvements we have made to its controller and training. In chapter 4 we describe our experimental setup, introduce the baselines and present the results. We show the impact of different components in an ablation study. In chapter 5 we summarize the thesis and outline directions for future research that it points to.

2 Related Work

Autonomous driving systems have been classified into 6 levels/categories by the society of automotive engineers [11]:

0. **Driver Only:** Systems that assist the human driver, by warning him, but do not control the car.
1. **Assisted:** The system exerts lateral **or** longitudinal control. The human driver has to monitor the situation and be able to take over at any time.
2. **Partial Automation:** The system performs lateral **and** longitudinal control. The human driver has to monitor the situation and be able to take over at any time.
3. **Conditional Automation:** The system performs lateral **and** longitudinal control in a specified use case. The system is responsible for monitoring the situation, and the human must be able to take over upon request.
4. **High Automation:** The system controls the car in all situations in a specific use case. A human driver is not required as long as the conditions for the use case are met.
5. **Full Automation:** The system controls the car in all situations. No human driver is required.

Our work focuses on full automation within a realistic 3D self-driving simulator.

Early research on autonomous driving research dates back to the 1980s, with the pioneering work of ALVINN [12]. There they developed a level 1 assisted system that controls the lateral steering of a car based on a camera and laser range sensor. They trained a fully connected 3 layer neural network on simulated road images that was then deployed on a real car and could follow empty roads. This approach where a neural network directly takes the sensor data as input and outputs driving commands was later called end-to-end driving. Another pioneering work was that of Ernst Dickmanns as part of the European PROMETHEUS project [13]. They demonstrated the first long distance drive from Munich to Odense at 95% level 2 partial automation. They used the other dominant approach in self-driving called modular pipeline, wherein the system is separated into individual modular tasks.

Modular pipelines and end-to-end systems and continue to be the dominant approaches to this day and are described in more detail in section 2.1 and 2.2.

Performing research in autonomous driving is complicated because developing an autonomous car is expensive. First, it requires a car equipped with a special sensor suite. Secondly, testing systems is dangerous, as mistakes by the driving system can lead to severe accidents. Therefore, special testing facilities or safety drivers are required to evaluate a system. Finally, to evaluate a system, it has to drive a large amount of kilometers in diverse situations and weather conditions. One way to avoid this problem is to only work on an individual task used in modular pipelines, such as object detection. Another is to evaluate the proposed method only **open loop** meaning that the system's behavior is compared to the recorded decision of a human driver on a particular data sample. This is known to not correlate well with real world driving performance of a system [14]. A third approach that has gained popularity in recent years is to evaluate methods in a driving simulator **closed loop**. This means that the system gets control over the car and is evaluated based on its driving performance. The advantages of simulators are that methods can be cheaply evaluated closed loop, driving conditions can be freely controlled (such as weather) and testing dangerous scenarios is safe as no real damage occurs when the car crashes. Their disadvantage is that they simplify reality to varying degrees, and hence not all complexities of the real world problem are modeled. A particular noteworthy self-driving simulator is **CARLA** (CAR Learning to Act) [3]. It has been specifically designed for developing and testing self-driving approaches and is open source and freely available to anyone at no cost. It can simulate most relevant self-driving sensors and has reasonably realistic graphics based on a modern game engine. Perhaps most importantly, it has a leaderboard [15] where teams can evaluate their approaches on secret test routes on provided third party servers. This allows for objective and fair comparisons of competing methods and gives the community a way to measure its progress. At the time of writing, no method is able to safely drive on the novel conditions of the leaderboard or even close to it. The community is making steady progress though. We work with the CARLA simulator in this thesis. Some examples of how it looks can be seen in figure 2.1. In this work we often show images from a bird's eye view perspective, on which weather effects may look less realistic (because they have been built for 3rd and 1st person view) but which provides a better overview over the situation.

2.1 Modular Pipeline

In the modular pipeline approach, the driving task is separated into individual modules that are trained or programmed individually. Often the tasks of perception, planning and control are considered, but more fine-grained distinctions are also sometimes made. An advantage of the modular pipeline approach is that the intermediate representations (such as semantic segmentation) are chosen by a human designer and hence are interpretable. A disadvantage may be that the intermediate representations could be suboptimal and are not optimized for the final driving task. Failures in earlier modules can propagate into later modules. The

(a) Town 01 EU style intersection

(b) Town 03 Free View

(c) Town 03 US style intersection

(d) Town 10 HD

Figure 2.1: Some example images from different CARLA towns.

different modules can be effectively developed in parallel by a large team. This might be one reason why the modular pipeline approach has become the standard approach in industrial research [16].
MP3 [17] for example, is a recent publication from Uber. Its perception stack predicts dynamic states of actors and an online map of the road from voxelized LiDAR data. The online map consists of BEV representations for drivable areas, intersections, reachable angles and lanes. The dynamic state consists of an BEV occupancy map and a BEV temporal motion field. A motion planning module then converts these representations into trajectories that the car should follow (presumably executed by a controller).
In CARLA research, the modular pipeline has not been particularly popular so far. One notable instance was the modular pipeline approach described in the original release of the CARLA simulator [3]. It decomposed the driving task into perception, planning and continuous control. The perception stack estimates a semantic segmentation of the input camera image and a classification model to determine the proximity to intersections. The semantic segmentation image is then used to estimate lanes, road limits, car and pedestrians and other hazards. The planning module is a state machine that for the different states (road-following, left-turn, right-turn, intersection-forward and hazard-stop) produces waypoints that describe the desired position and orientation of the car in the near future. These waypoints are then converted into the car commands by a PID controller. The modular pipeline performed similar to imitation learning in this evaluation.
A second notable modular pipeline approach is Pylot [16]. Pylot is a modular pipeline framework developed to accurately measure the impact of the runtime of individual components. Algorithms that are slow have in reality a lower accuracy because the world around it has changed by the time the algorithm finishes its prediction. This problem is also called streaming perception [18]. Pylot's components include object detection, object tracking, prediction, planning and control. They provide several baselines for each component. The method is built for a simulator, but can also be deployed on a real car with minimal changes. The modular pipeline has been an under-explored area on openly available simulators. At this point, we do not have conclusive evidence whether it is better or worse than end-to-end approaches. Individual components of the modular pipeline are however well explored. Several computer vision datasets like KITTI [19] or Cityscapes [20] focus on self-driving. Planning and control are also well explored areas, see for example [21, 22].

2.2 End-to-End

End-to-End driving describes approaches in which the entire driving task is done by a single neural network that takes in the raw sensor data and outputs the driving commands. Sometimes also methods are described with this label that have additionally a controller (like a PID) that converts the network output to steering, throttle and brake commands. Advantages of this approach are that the

intermediate representations are jointly optimized for the driving task and that supervision labels can be obtained cheaply (which matters for real world data where supervising intermediate representations such as semantic segmentation is expensive). Its downsides are that the methods are not very interpretable. When the methods make a mistake it is often hard to debug as intermediate representations of a neural network have no semantic meaning for a human. There are two major approaches to train End-to-End models: imitation learning and reinforcement learning.

2.2.1 Reinforcement Learning

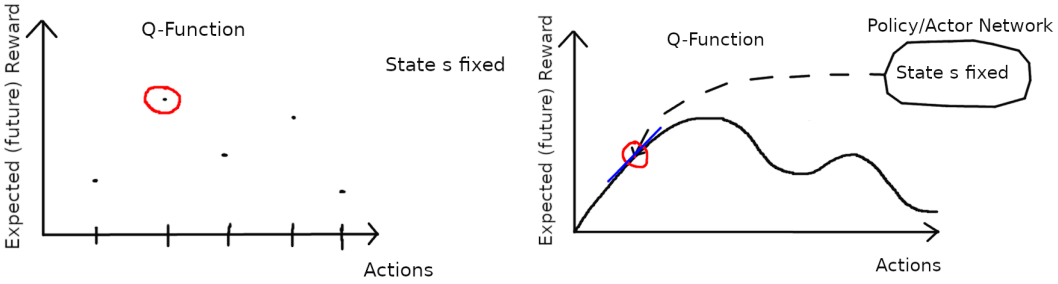
Reinforcement Learning is the paradigm of learning through trial and error. In this field the environment in which our agent (the computer program that is trying to learn) acts is often modeled as a Markov Decision Process (MDP) (S, A, P, R, γ). S is the set of all possible states, A the set of all possible actions, P is the state transition probability matrix describing the probability of the next state given the previous state and action pair, R is the reward function describing the expected return given a state and action pair and γ is a discount factor $\in [0, 1]$ weighting down future rewards. The goal of the agent typically is to find a policy $\pi : S \rightarrow A$ that maximizes the discounted expected return.

Basic reinforcement learning algorithm are often developed in game like simulations that can be executed very fast. The most important one being the **Atari benchmark** [23] in which agents are trained to learn how to play classic Atari games from the input screen. **Deep Q-Learning** (DQN) [24] was the first major success on this benchmark combining convolutional neural networks (CNN) with replay buffers, Q-Learning and target networks. The Q-function $Q(s, a)$ describes the expected return when the agent chooses action a in state s and then follows policy $\pi(s)$ afterwards. It is usually represented by a neural network. The dataset to estimate it is generated by executing the current policy in the environment and storing observation, action, reward and next observation in a replay buffer. The Q-function is then estimated by sampling experiences from the replay buffer and calculating its gradient using temporal difference learning:

$$\nabla Q \leftarrow ((r + \gamma * \max_{a'} Q'(s', a')) - Q(s, a))^2 \tag{2.1}$$

where r is the observed reward in the current sample, γ a parameter to discount future rewards, s' the next state, s the current state, a the current action and Q' the target network (a periodical copy of the Q network used to stabilize learning). The max operator in this equation can be computed because the action space is assumed to be discrete and small enough to evaluate all possible actions. The actual policy π is then implicitly defined:

$$\pi(s) := \operatorname*{argmax}_a Q(s, a) \tag{2.2}$$



(a) Maximizing a 1-Dimensional Q function in the discrete case just amounts to picking the highest point.

(b) With continuous action spaces, a policy network predicts for a given state an action. This prediction can then be improved by following the gradient along the Q-function and updating the network with backpropagation.

Figure 2.2: A conceptual illustration of the Q-function in the 1-dimensional case, keeping the state fixed. Figure a) is for the discrete case, figure b) illustrates the continuous case.

This assumption is of course a limitation because in some domains like autonomous driving the action space is continuous, making the max operation intractable. The typical strategy used by continuous control reinforcement learning algorithms is to represent the policy explicitly. That policy is then represented by a parameterized neural network that takes as input the current state and outputs an action. This policy can then be updated using the insights of the Policy Gradient Theorem [25] by alternately minimizing the following terms:

$$\nabla Q \leftarrow ((r + \gamma * Q'(s', \pi(s'))) - Q(s, a))^2 \tag{2.3}$$

$$\nabla \pi \leftarrow -Q(s, \pi(s)) \tag{2.4}$$

The parameters of the policy π can then be adjusted via backpropagation. This is also illustrated in figure 2.2 where we compare conceptually the two different approaches in the case of a 1-dimensional Q-function.

This approach is called actor (=policy) critic (=Q-function). With the method deep deterministic policy gradient (DDPG) [26] it was demonstrated that this approach works if combined with experience replay and target networks (similar to the discrete case).

One problem of these deterministic algorithms is that they are quite sensitive to hyperparameters and even the seed that they are trained with. Additionally, training a reinforcement learning algorithm can be quite slow in practice (requires many samples and fast simulators) which makes using these algorithms hard. **Soft Actor Critic** [27] is an algorithm that yields competitive performance on continuous control tasks and has been shown to be more robust to hyperparameters and the seed. It

is an actor critic algorithm that expands the standard objective to maximizing the expected future return while acting as randomly as possible (maximizing the entropy of the policy). This changes the update equations to the following terms:

$$\nabla Q \leftarrow ((r + \gamma * (Q'(s', \pi(s')) - \alpha \log(\pi))) - Q(s, a))^2 \tag{2.5}$$

$$\nabla \pi \leftarrow -Q(s, \pi(s)) + \alpha \log(\pi) \tag{2.6}$$

where α is a tradeoff parameter between maximizing entropy and maximizing reward (it can be tuned automatically [28]). The entropy objective incentivizes the policy to explore more during training. The policy also gains the ability to capture multiple optimal strategies. We use this algorithm in our reinforcement learning ablation because of its robustness to hyperparameters and the seed.

2.2.2 Reinforcement Learning for Autonomous Driving

The intersection between autonomous driving and reinforcement learning started being investigated around 2016 [29]. Almost all work is using driving simulations to train and evaluate their methods. One notable exception is [30] where they only used simulation to find appropriate hyperparameters for their method. The training was then done on a real car, with a safety driver preventing collisions and resetting the car after episodes. They were able to successfully train their car to perform lane keeping on a road similar to what ALVINN made possible in 1988 [12] with imitation learning. This work was done in a highly simplified setting. Much more diverse scenarios and tasks need to be able to be performed for reaching full autonomy.

It is of note that at the time of writing, there is no published report that we are aware of that achieves competitive results training a sensorimotor agent purely using reinforcement learning at the complexity of simulators like CARLA. One notable attempt has been made with the publication of the CARLA simulator [3] where a state-of-the-art reinforcement learning algorithm was vastly outperformed by conditional imitation learning and modular pipeline.

A considerable body of research exists on RL for AD [29, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48], but most of it assumes access to privileged information that is not available for sensorimotor agents or is evaluated on much simpler simulators. The reason for this lack of performance of reinforcement learning in autonomous driving, compared to games like Atari, is that autonomous driving requires training deep neural networks like ResNets for perception. This is however beyond the state-of-the-art of RL. Most "Deep" Reinforcement Learning approaches can only train small MLP networks and ConvNets or require an impractical amount of data to converge [49].

This is not to say that reinforcement learning is useless for autonomous driving, only that we can not currently train entire methods solely with reinforcement learning. In

fact, it has been successfully applied in combination with supervised learning.

Controllable Imitative Reinforcement Learning (CIRL)[50] was the first approach to achieve this in CARLA. Here they fine-tuned a fully trained imitation learning agent using a reinforcement learning algorithm with a small learning rate. This improved the performance of the agent on the original CARLA benchmark.

One of the best approaches on the current CARLA leaderboard is **Implicit Affordances** [9] (called **MaRLn** on the submission). Here they separate their training into two stages. They train the perception ResNet part of the architecture on the auxiliary task of semantic segmentation using standard supervised learning. The segmentation head is then cut off and an intermediate layer is used as input for the rest of the network (the implicit affordances). The weights of the ResNet are then frozen during the rest of the training. The remaining fully connected layers in the architecture are then trained using reinforcement learning. Intuitively speaking, they train perception supervised and planning and control with reinforcement learning.

A third successful application of reinforcement learning in CARLA is **World on Rails** [8]. In this work, reinforcement learning was applied in a rather uncommon fashion. They used it to improve the quality of labels on an offline dataset collected by an expert driver. They use a reward function to evaluate all possible actions of the agent for each data point in the dataset. The resulting state is determined by simulating the agent with a bicycle model and all other traffic participant with the world on rails assumption (meaning they do the exact same action as in the dataset, independent of the ego agent's behavior). The Q value for each action is calculated using backward induction and dynamic programming. A standard supervised sensorimotor agent is then trained to predict the Q-scores for each action. At the time of writing, it is the best published approach on the CARLA leaderboard.

2.2.3 Imitation Learning

Imitation learning is the current dominant approach to train end-to-end models. In this approach, the network is trained to predict for a given input the action that an expert driver (that it is imitating) would have done. Formally, it can be described as:

$$\underset{\theta}{\operatorname{argmin}} \mathbb{E}_{s \sim P(s|\pi_\theta)}[\mathcal{L}(\pi^*(s), \pi_\theta(s))] \tag{2.7}$$

where π^* is the expert policy, π_θ the learned policy and $\mathcal{L}$ some loss function. Note that here the states s depends on the current policy π_θ. In the popular variant of behavior cloning, an expert driver collects a static dataset instead by driving around and recording the sensor inputs as data and his actions as labels. Behavior cloning can be described as:

$$\underset{\theta}{\operatorname{argmin}} \mathbb{E}_{(s^*,a^*) \sim P^*}[\mathcal{L}(a^*, \pi_\theta(s^*))] \tag{2.8}$$

where the state and target actions come from the dataset P^* which is collected by the expert policy. The network is then trained with standard supervised learning

methods from this offline dataset. A particular problem this approach has it that the expert usually does not make many mistakes, and therefore there are few or no examples in the dataset on how to recover from mistakes. When the trained network is then evaluated closed loop, it can lead to failure when the network encounters new situations after making a mistake which it does not know how to recover from. A popular approach to alleviate this problem is called Dataset Aggregation (**DAGGER**) [51] which iteratively builds the dataset. Here, a static dataset is first collected by the expert. The model is then trained and evaluated closed loop. When the network makes a mistake the expert will take over and recover from it, adding this particular example to the training dataset. This process is then iteratively repeated.

Another problem of pure imitation learning is that there can be situations where the optimal action is ambiguous. An example is an intersection where both turning left and turning right are possible actions. The dataset might even contain examples of the expert doing both actions. In the worst case, the method then averages the actions and drives straight. The solution to this problem is to condition the input based on a high level driving command or target position created by a navigational planner. In this approach, called **Conditional Imitation Learning** (CIL)[52], inputs come together with a command such as turn right or follow lane. It has been widely adopted in the imitation learning literature since then.

Some examples of conditional imitation learning approaches are:

CILRS [10] improves classic CIL by using a deeper perception architecture (ResNet), image-net pre-training and speed prediction as auxiliary loss. They also introduce the NoCrash benchmark, on which models are trained on one town and evaluated on another. The metric is then how many percent of routes the agent completes without crashing in empty, regular or dense traffic conditions.

Learning by Cheating [53] is the first method to fully solve the original CARLA benchmark, achieving a hundred percent on all test settings. It also performs competitively on the NoCrash benchmark. It is of note here that the original CARLA benchmark was rather simple, containing tasks like driving straight or taking one turn. While this work is good progress, we are still far away from solving full autonomy. The method uses a rule based expert driver to train another neural network based expert driver (the cheater). The cheating network uses bird eye view semantic segmentation ground truth as input. It is then used to train a sensorimotor agent that does not cheat and drives based on camera images. While learning by cheating performs remarkably well on the original benchmark, it does not do so well on more recent benchmarks that feature challenging scenarios, such as the CARLA leaderboard or the NEAT validation routes.

TransFuser [6] is a method that looks at how LiDAR point cloud data can be combined with RGB camera images in an end-to-end architecture. They do so by projecting the point cloud into a bird's eye view image. Both camera and BEV image are then processed by a residual network. Information between the two branches is

then exchanged using Transformer blocks [54]. Another important contribution of this work is the way they condition their network. Instead of a binary command, the next target location is provided as a 2D coordinate in the car's local coordinate system. A recurrent network processes this point and predicts the position of the expert driver multiple time steps into the future. The approach is described in more detail in section 4.4.3. Methods using this waypoint network work remarkably well in lane following / lateral control.

Neural Attention Fields (NEAT) [7] is a very recent method that regresses bird's eye view vectors pointing to the path that the car should follow. It does so by querying an implicit continuous representation of the scene (MLPs) at discrete locations. The inputs are 3 camera images processed as usual with a ResNet and fused together with a transformer. It was the prior state-of-the-art on the NEAT validation routes.

2.2.4 Expert Drivers

Expert drivers are the methods used to generate the labels and collect the data for imitation learning methods. In principle this can be a human driver and this is a common approach in industrial research. In CARLA research however this is rarely done in practice, researchers do not spend weeks of their time playing the CARLA simulator. Instead, methods with privileged access to the simulator are used as an expert to supervise the sensorimotor agents. These methods, while perhaps not as good as the human, can collect data faster than real time and can be executed in parallel. Data quality plays an important role in supervised machine learning. However, these expert drivers are often of secondary interest in publications, briefly described in the appendix and not evaluated quantitatively.

The recent method reinforcement learning coach (**Roach**)[5] is the only work we are aware of that explicitly investigates this topic. They create a birds eye view semantic segmentation like input representation using ground truth access to the simulator and train a convolutional neural network using a popular reinforcement learning algorithm. While it improved over prior work, it is expensive to train, difficult to extend and does not achieve perfect scores either.

We argue in this work that an expert driver achieving maximum score is an important preliminary to correctly evaluate the performance of imitation learning methods. It can also give insights into the necessary components required to solve the task. Lastly, it is of note that since imitation learning methods search for correlations in the input data with the expert actions, only input information that the expert uses can be useful for the learning task. For example, if the expert only uses information from its front, then a method using full 360° cameras just has a lot of data that contains no correlations with the expert's actions. It is therefore important to design the expert to use all relevant information.

3 Method

In this chapter, we explain the problem setting and describe our expert driver method SEED (a **S**imple and **E**ffective **E**xpert **D**river). An overview of SEED can be seen in figure 3.1. The problem setting is introduced in section 3.1. The basic components object-oriented bounding boxes, bicycle model, PID controller and A* planner are introduced in section 3.2. SEED is discussed in detail in section 3.3. We also train a sensorimotor agent to imitate SEED, which is described in section 3.4.

3.1 Problem Setting

We consider the task of driving from a starting point to predefined goal locations. Agents are supposed to drive safely along a route in an urban setting in simulation while following traffic rules. The agents will experience challenging adversarial scenarios along the route, described in more detail in chapter 4. We differentiate between two types of agents.
(1) The **privileged expert drivers** which have direct access to the simulator. Their input can be any representation that the simulator can provide. Their output is the steering, throttle and brake commands of the car. The purpose of privileged expert drivers is to serve as an expert to be imitated by imitation learning methods. Put in other words, they automatically generate the labels for supervised learning approaches. The quality of these labels largely depends on the performance of the expert driver method.
(2) The **sensorimotor agents** solve the same task as the privileged agents, but they only have access to sensor data that would be available in a car deployed in the real world. Such sensors include cameras, LiDARs, GPS, speedometer and inertial measurement units (IMU). Their output again are the steering, throttle and brake controls of the car.

The starting point and destination for the agents are user defined. We follow the standard protocol of CARLA 0.9.10.1 that provides a navigational planner. This navigational planner calculates a route between the starting point and destination. This route is provided as dense GPS waypoints for the privileged agents and as sparse GPS waypoints (potentially hundreds of meters apart) for the sensorimotor agents.

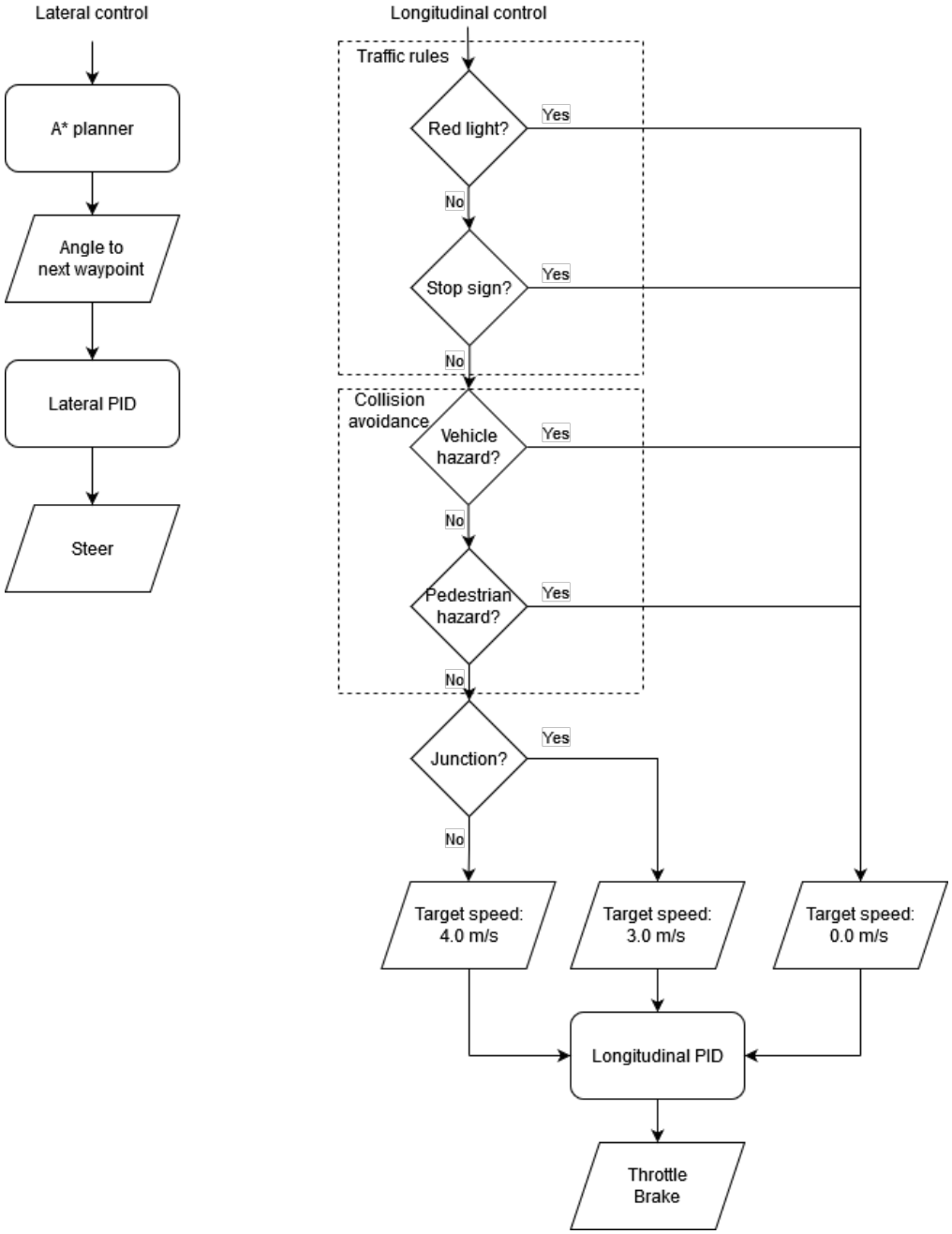


Figure 3.1: Flow chart of SEED

3.2 Preliminaries

3.2.1 Object oriented bounding boxes

Cars are complex objects represented in simulation by meshes that a build of a lot of triangles. This makes answering questions such as whether two cars collide computationally very expensive. Naively, one would have to perform an intersection test between each of the triangles of two cars. Fortunately, cars can be accurately approximated by (object) oriented bounding boxes (OBB). These are rectangular cuboids that entail the car and are oriented in such a way that the least amount of additional volume is added compared to the car object. It is a well studied object in the computer games literature and natively supported by many simulations. Using them, we can represent a car with just 9 numbers. The x, y, z coordinates of its center, the pitch, yaw and roll for orientation and the x, y, z extent along each axis. Compared to their axis aligned versions, they are a tighter fit for the object and can be used to reason about multiple bounding boxes in global coordinate space. We use OBBs to represent all objects of interest: cars, pedestrians, bicycles, light trigger boxes and stop sign trigger boxes. To know whether two cars will collide, we need to know whether the two OBBs representing them intersect. For that we use an oriented bounding box intersection test [55] that given two OBBs in global coordinate space will tell us whether they intersect.

3.2.2 Bicycle model

To predict where a car will be at the next time step, we need to have a model of its dynamics. We choose the popular bicycle model [56] because of its simplicity and the observation that it is an accurate model for cars in the CARLA simulator. The bicycle model is illustrated in figure 3.2. The two front and back wheels are merged into one wheel, and we assume we are driving on a flat plane. Only the front wheel can be steered in this model. The input of the model is the current position x, y, velocity v, orientation ψ, acceleration a and the front wheel steering angle s of the car. Using the following equations it calculates the position x', y', orientation ψ' (yaw) and velocity v' of the car after some time ∇t.

$$\beta(s) = \arctan\left(\tan(s)\frac{l_r}{l_f + l_r}\right) \tag{3.1a}$$

$$x' = x + v\cos(\psi + \beta(s))\nabla t \tag{3.1b}$$

$$y' = y + v\sin(\psi + \beta(s))\nabla t \tag{3.1c}$$

$$v' = v + a\nabla t \tag{3.1d}$$

$$\psi' = \psi + \frac{v}{l_r} sin(\beta(s))\nabla t \tag{3.1e}$$

We estimate the parameters l_r, l_f as well as how throttle, brake and steering translates into acceleration and angle using the procedure from [8]. In short, the procedure

collects a dataset of the car driving around and optimizes the parameters such that the model best predicts the positions of the car.

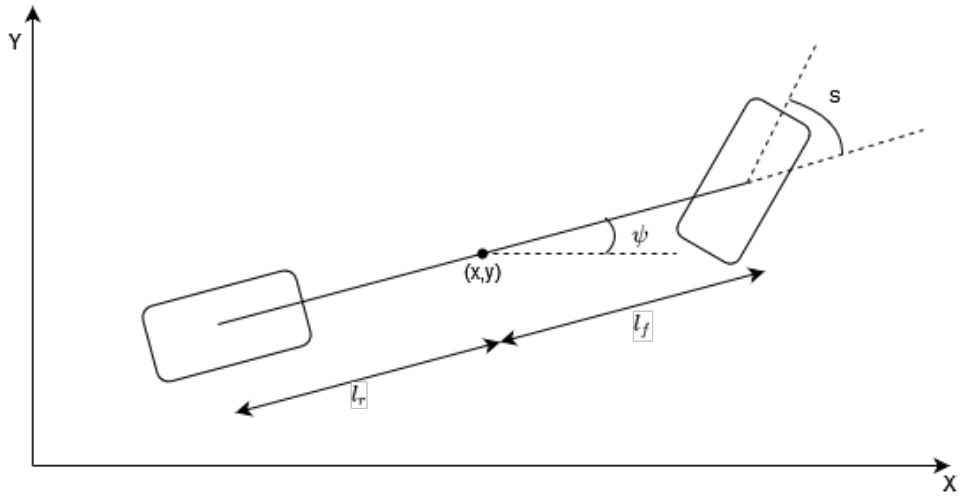


Figure 3.2: An illustration of the bicycle model.

3.2.3 PID controllers

Control is the problem of executing a particular plan. In our case, this means minimizing an error quantity like difference to target speed. Just directly minimizing the error quantity is called proportional control. In practice, it often has the problem of oscillating behavior. To address this, the standard solution is to also consider the integral and derivative of the error function over time. This approach is simply called Proportional Integral Derivative control (**PID**) [57] and is widely used to address control problems. The goal of a PID controller is to minimize the error at time step t (e(t)) using the following equation:

$$y(t) = K_p e(t) + K_i \int_{t-a}^{t} e(\tau)d\tau + K_d \frac{de(t)}{dt} \tag{3.2}$$

Here $y(t)$ is the output and K_p, K_i, K_d are parameters. Since we sometimes drive for a long time, we only calculate the integral for a fixed time window by setting the parameter $a = 40$. The simulation is running at discrete time steps of 50 ms, therefore the integral turns into a sum and can easily be computed. The PID controller is illustrated in the following python style pseudocode:

```
class PIDController(object):
    def __init__(self, K_P, K_I, K_D, a=40):
        self.integral_buffer = deque([0.0 for _ in range(a)], maxlen=a)
        self.last_error = 0.0
        self.K_p = K_P
```

```
        self.K_i = K_I
        self.K_d = K_D

    def step(self, error, delta_time):
        self.integral_buffer.append(error * delta_time)
        integral = sum(self.integral_buffer)
        derivative = (error - self.last_error) / delta_time
        self.last_error = error

        return self.K_p * error + self.K_i * integral + self.K_d * derivative
```

The parameters K_p, K_i, K_d have to be manually tuned for each PID controller. There exist various heuristics for doing so.

3.2.4 A* search algorithm

The A* search algorithm [58] is a classical solution to the problem of finding the shortest route in a graph. It is guaranteed to find the optimal solution and computationally efficient, but has a high memory demand. It starts at a start node and iteratively builds a tree of paths by checking neighboring nodes. It estimates the cost of a path by adding the cost from the start to the current node with a heuristic estimating the cost of the cheapest path from the current node to the goal (such as euclidean distance, for example). The next node to examine is chosen based on the estimated cost. To get the final path after the goal has been reached, each node stores its predecessor. The path is then found by backtracking from the goal to the start in the end.
The CARLA leaderboard client provides an implementation of the A* algorithm that given a start will find a route to a destination point. That route is represented as a set of dense waypoints in the center of the road that follow the route.

3.3 SEED

Our privileged expert has to predict 3 outputs: steering, throttle and brake. We build upon the TransFuser Expert [6, 7] which uses the same basic architecture as shown in figure 3.1. SEED differs from this TransFuser Expert in the way it answers the traffic rules and collision avoidance questions. To explain the system, we start with the observation that lateral control (steering) can be treated independently of the longitudinal control (throttle and brake). Lateral control is especially simple. It is sufficient to follow the path from the start to the goal. The path is determined by a high level A* path planner and is represented as a dense list of 2D waypoints. If the path is blocked by a car or pedestrian, one can simply wait until they move out of the way. We follow the global path by setting the steering error, that our lateral PID controller will minimize, to the angle of the car towards the next waypoint. The

angle is normalized to lie in range $[-2, 2]$. The next waypoint is the closest waypoint that is farther than 4 meters away from the center of the car. The car will then always drive directly towards this waypoint [59]. Since the next waypoint is constantly changing as the car progresses, this leads to the car moving along the predetermined path. The dense path is visualized in the following images as black cubes and the waypoint that the car is currently driving towards is marked white (e.g. figure 3.3). For longitudinal control, we differentiate between three target speeds: Regular driving which is set to 4.0 meters per second (m/s), driving through a junction which is set to 3.0 m/s and stopping the car where the target speed is 0.0 m/s. We always stop the car as fast as possible by performing a full brake. This is of course not ideal in terms of passenger comfort. However, at the current stage of CARLA research, we first need to achieve full safety before improving other metrics like speed or comfort. At the time of writing, there is no sensorimotor approach that can consistently drive in CARLA 0.9.10.1 without collisions. Keeping the car at the desired target speed is achieved with a PID controller that minimizes the difference between the measured speed and the desired target speed. The target speeds are simple parameters in SEED and can be easily adjusted if needed. The current cruising speed is fast enough to clear the routes in the time budged granted by the CARLA leaderboard.

So far, we stayed relatively close to prior work. The actual hard part in designing an expert driver, and this is where prior work fails at, is deciding when to brake. In CARLA there are four causal reasons to slow down: Red lights, stop signs, vehicle hazards and pedestrian hazards. We treat each of them independently and if any of these conditions is true we will set the target speed to zero and stop the car.

3.3.1 Traffic Rules

Traffic lights: The CARLA 0.9.10.1 simulator provides a flag for vehicles whether a red light is currently affecting them. The TransFuser Expert uses that flag to decide whether to brake or not. While this is in principle not a bad idea, in practice we observe that this flag provided by CARLA is unreliable. It has both false positives, telling the car it is affected by a red light even though no traffic light is around (the TransFuser Expert catches these cases), and false negatives, telling the ego vehicle there is no red light affecting him while it is running one. The result is that the TransFuser Expert frequently runs red lights, negatively affecting the quality of the labels generated.

Here we engineer a new approach for red light detection that does not have the aforementioned problems. In CARLA, every traffic light has a so-called trigger box attached to it. This trigger box is an oriented bounding box that is oriented orthogonal to the street and located before the entry to the intersection. We define an additional oriented bounding box following and entailing the agent car that we call light detector OBB. At every time step, we go through the trigger boxes of all traffic lights in the scene. We prefilter them and only consider the once whose centers are

withing a certain distance (set to 15 meters) to the center of the vehicle. For each traffic light, we perform an OBB intersection test between the connected trigger box and the light detector OBB. If an intersection is detected, we look up the traffic light state of the traffic light connected to the trigger box. If the traffic light state is either yellow or red, we consider the red light as detected and the vehicle will come to a stop. The method is visualized in figure 3.3. The light detector OBB is the large red box around the car. It is extended behind the car because the trigger boxes are often some distance before the entry of the intersection. In our visualizations, the traffic light trigger boxes change color according to its associated traffic light state (red, yellow or green). The light detector box changes its color from white to red when a red or yellow light is detected.

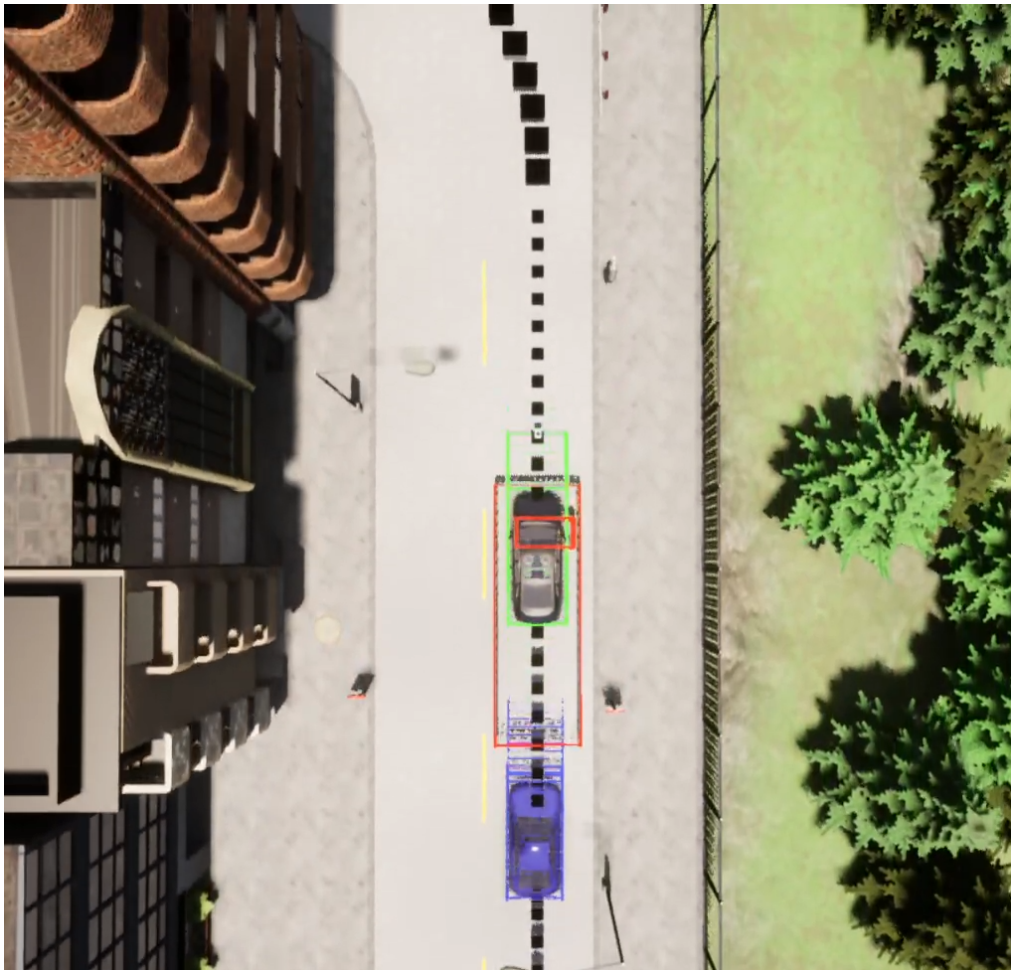


Figure 3.3: The red light detection. The small red box is the trigger box of the traffic light, colored in its state. The large red box around the car is the light detector box that follows the car. When the light detector box intersects with a red or yellow trigger box, it will turn red and the car stops.

Stop signs: We handle stop signs similarly to traffic lights. Each stop sign has a trigger box attached to it in CARLA. After pre-filtering all stop signs analogously to traffic lights, we perform again an OBB intersection test between the trigger box and the standard bounding box of the car. If there is an intersection and the stop sign is not yet cleared, we will mark the stop sign as detected and slow the vehicle down. Once we have reached speed 0 the stop sign is marked as cleared and the car will continue driving. After leaving the intersection, the list of cleared stop signs will be emptied again. An example is shown in figure 3.4.

(a) SEED driving towards a stop sign.

(b) SEED slowing down to clear the stop sign before continuing.

Figure 3.4: Stop sign on the road, represented by its green trigger box. When SEED approaches, it turns red until cleared.

3.3.2 Collision Avoidance

Vehicle Hazard: To explain our vehicle hazard detection algorithm, we first want to outline how the prior work TransFuser Expert handled this problem. The TransFuser Expert checks a static (cone like) area in front of it. The vehicle hazard is marked as detected if a car is in that area. From a basic perspective, this algorithm is answering the question:
"**Is** there something in my way?"
We argue that this is fundamentally the wrong question to ask, and illustrate this with an example in figure 3.5.

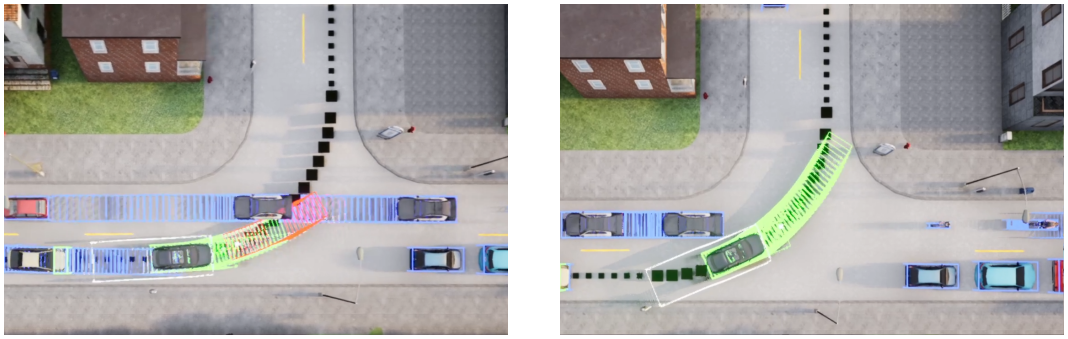
In this figure, the black vehicle at the bottom is representing the expert driver. The vehicle and bicycle are driving straight with constant speed. In a situation like it is depicted in figure 3.5a the collision logic predicts no hazard because there is nothing directly in front of the car. This decision is wrong because there will likely be a collision with the oncoming cyclist. By the time the static cone detects the cyclist it will be too late to stop the car (cars need a certain distance to come to a halt) likely leading to a crash. The car should slow down now to avoid the situation. In another situation like depicted in figure 3.5b the TransFuser Expert collision logic predicts to

(a) False negative. No collision is predicted even though the agent is in a dangerous situation.

(b) False positive. The agent can already start driving again because the cyclist will be gone in time.

Figure 3.5: Illustrative explanation of the old collision logics failure cases.

wait because of the cyclist in front of the car. However, this is unnecessary because the cyclist will have moved forward by the time the car arrives at that location.
We argue that instead the hazard detection algorithm should answer the question: "**Will** there **be** something in my way?".
This means reasoning about where other traffic participants and the agent will be in the future. This is the question we want our agent to answer. To do that, we need an understanding of how cars move over time. For that, we use the previously explained bicycle model. Given the position, orientation, and speed of the car as well as the current action (steer, throttle, brake) the bicycle model can predict the position, orientation and speed at the next time step. Now, the problem is of course that looking one time step (=50ms) into the future does not yet tell us a lot. Since we do not know our action at future time steps, we need to make an assumption at this point. We call the assumption the **action repeat assumption**, namely we assume that we apply the same action at future time steps again. This assumption is of course to some extent violated in reality, but the difference does not matter much in practice, as shown in chapter 4. So now we can iteratively use the output of the bicycle model as input for the next time step and get a prediction of where the car will be at each time step t. We extrapolate 3 seconds into the future at junctions, where paths of cars are frequently crossing. Everywhere else, we extrapolate 1 second. As resolution, we use the same time steps as the simulator, namely 50ms. We apply the same procedure to each vehicle in a radius of 50 meters. To detect vehicle hazards, we then do a OBB intersection test between the ego agent's OBB at time step t and the OBB of each other car at time step t. If an intersection happens, we have predicted a collision on the current course and the vehicle hazard is set to true. An example of this can be seen in figure 3.6 where the agent performs an unprotected left turn.



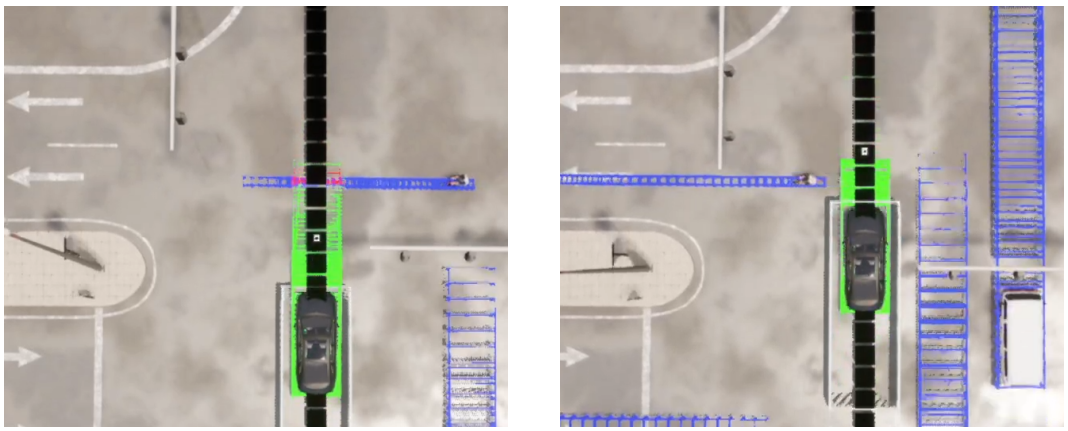
(a) SEED correctly assesses that driving now would lead to a collision (red boxes).

(b) SEED waits until the oncoming traffic has passed and then crosses the intersection.

Figure 3.6: SEED performing an unprotected left turn.

In the figure, the OBBs of other vehicles are rendered blue. The OBBs of the ego vehicle are rendered green. At the time steps where a collision is predicted, the OBBs of the ego vehicle are colored red. As shown, SEED successfully predicts that on the current course of action, there will be a collision in the future. It slows down to let the other vehicles pass first and crosses the intersection when there is enough space.

Another example can be seen in figure 3.7, where a cyclist runs a red light leading to a dangerous situation inside an intersection. SEED correctly predicts the oncoming collision and slows down to let the cyclist pass first. Note that simply checking a static area in front of the car like the TransFuser Expert does would have likely led to a collision in this situation because by the time the cyclist is in front of the car it is too late to stop.



(a) SEED predicts a collision on the current course.

(b) SEED slows down to let the cyclist pass first.

Figure 3.7: A cyclist has run a red light, leading to a dangerous situation at an intersection. SEED successfully predicts it and avoids the collision.

Cars need a certain time to get to a halt, so a minimum distance to preceding vehicles should always be kept. The distance depends on the speed the car is driving at and can be approximated with the following formula:

$$b = (\frac{v}{10.0})^2/2.0 \tag{3.3}$$

where v is the velocity of the car in km/h and b is the resulting braking distance. We realize this by placing a bounding box covering the resulting distance b in front of the car. If it intersects with any other bounding box at time step 0 we stop. An example of this can be seen in figure 3.8 where SEED waits some distance behind the preceding car at a red light.

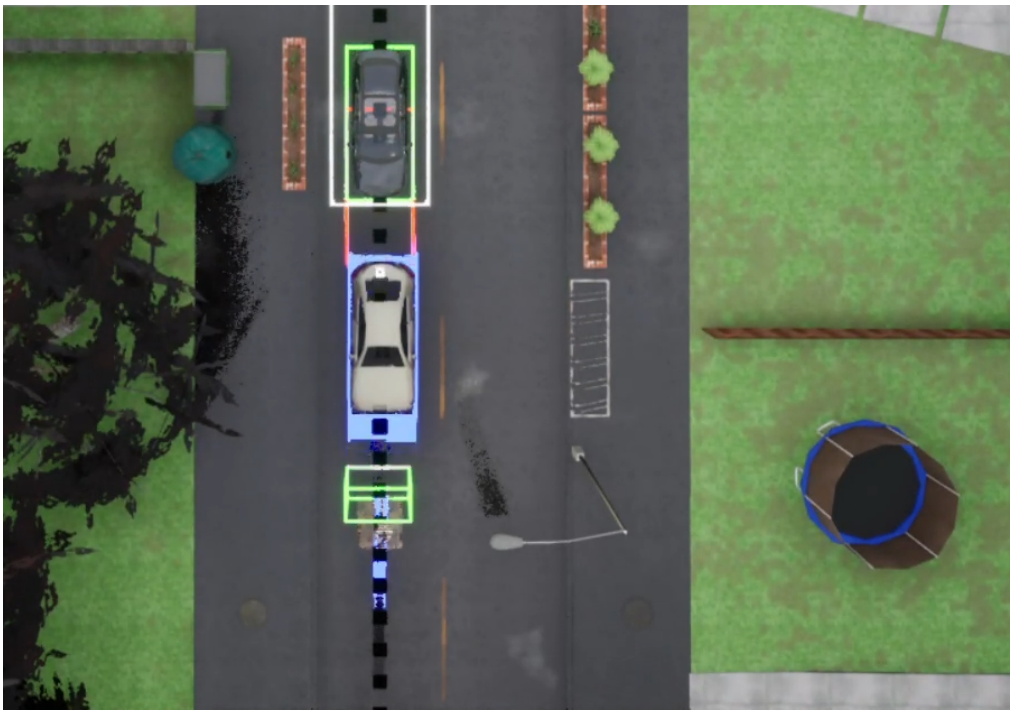


Figure 3.8: SEED keeping a safe distance to the preceding vehicle indicated by the red box.

An additional modification we make to the bicycle model is that we set the brake action of the ego vehicle always to 0. Since we use the algorithm to determine whether to brake we need to know what happens if we do not brake. Another assumption implicit in the algorithm is that there are no obstacles in the vehicle's path (which would lead to a crash and stop it). It can happen that cyclists trying to cross the street drive against a handrail and get stuck. We pre-filter stuck vehicles/cyclists to avoid that their bounding boxes block the agent's path. Avoiding static objects on the street is left for future work, as this situation currently almost never happens in CARLA 0.9.10.1

Pedestrian hazards are handled analogously to vehicle hazards. The only difference is that we use a different dynamics model to describe the pedestrian. Specifically, we treat pedestrians as simple points with a velocity (applied to the direction they are oriented in) and an acceleration. This is sufficient to precisely describe their trajectories, as pedestrians only run in straight direction across the street in the CARLA leaderboard client. An example can be seen in figure 3.9.

(a) SEED a collision with the pedestrian on the current course.

(b) SEED slows down to let the pedestrian pass first.

Figure 3.9: A pedestrian crossing the street. SEED predicts a collision and waits for the pedestrian to cross.

3.4 TransFuser+

To show that SEED improves imitation learning methods, we generated a training dataset with it. We used that dataset to retrain the TransFuser [6] architecture. The dataset was generated with the same training routes to make the dataset comparable. To get the most out of the new dataset, we also made some improvements to the training and controller. The individual contributions are investigated in an ablation study in section 4.6.

3.4.1 TransFuser architecture

We have made no changes to the inference architecture of the TransFuser network. During training, we added two additional decoder heads to the image branch of the architecture. These decoder heads are used for auxiliary tasks, semantic segmentation and depth prediction. It has been shown [7] that these auxiliary tasks help for image only models. We apply the same ConvNet decoders on the image branch and report similar findings for this multi-modal architecture. The architecture design is illustrated in figure 3.10.

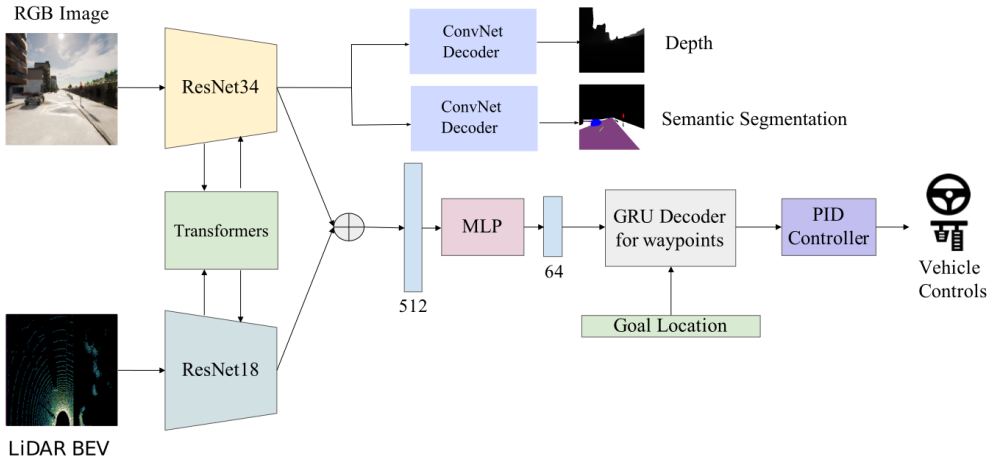


Figure 3.10: TransFuser+ architecture. Adapted from [60] with permission of the author.

The inputs to the method are a 256x256 monocular camera image and a bird's eye view (BEV) image of the LiDAR point cloud. The LiDAR point cloud is voxelized into two channels (higher and lower than 50 cm above the street) and each pixel correspond to the number of LiDAR hits in that area (up till a maximum of 5). An example of the LiDAR can be seen in figure 3.11 and of the image in figure 3.12.

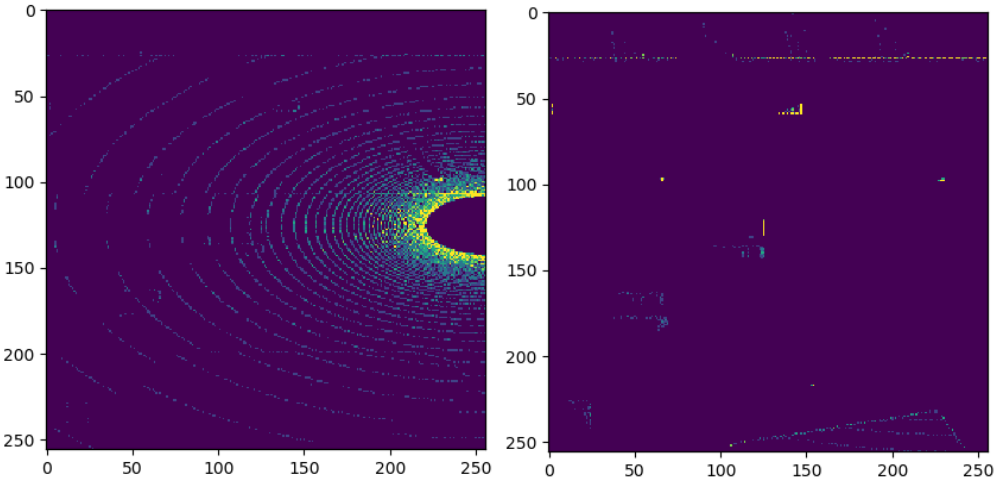


Figure 3.11: An example of the BEV LiDAR input. The two images correspond to the two different channels

Both inputs are processed by standard residual network architectures. In the hidden layers information is exchanged between the two branches using transformer [54] blocks. For the auxiliary losses, the output of the image branch is up-scaled to

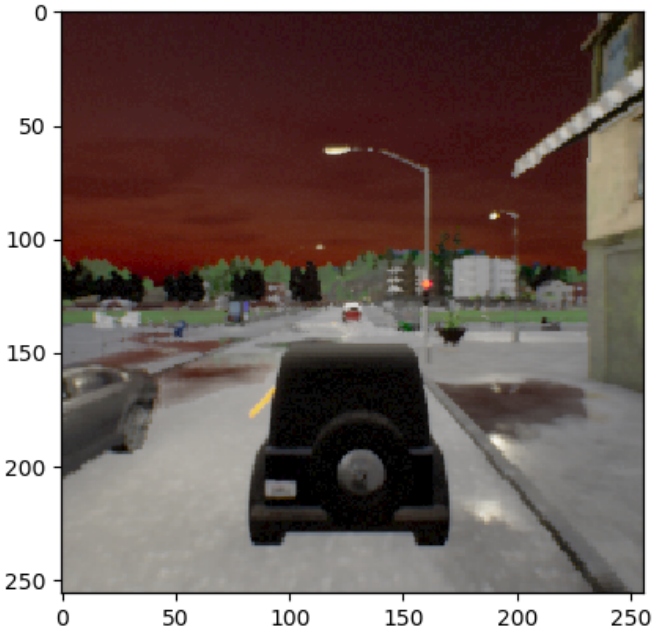


Figure 3.12: An example of the image input.

the original resolution using 2D convolutional layers, ReLU activation functions and bilinear interpolation. The features from the image and LiDAR branch are then concatenated and processed by a multi layer perceptron (MLP). Finally, an autoregressive GRU [61] processes the features together with the goal location, to output waypoints. A PID controller then finally transforms the waypoints into the desired vehicle controls.

The model is trained to predict the position of the (SEED) expert 0.5, 1.0, 1.5 and 2.0 seconds into the future using an L1 loss. Additionally, we leverage multitask learning [62] and predict a 2D semantic segmentation map using the cross entropy loss and a 2D depth map using the L1 loss. We minimize the following equation:

$$\lambda_0 \sum_{t=1}^{4} (x_t - x'_t) + (y_t - y'_t) + \lambda_1 \sum_{i=1}^{H*W} d_i - d'_i + \lambda_2 \mathcal{L}_{CE}(s, s') \tag{3.4}$$

where the lambdas are hyperparameters with $\lambda_0 = 1$, $\lambda_1 = 1$, $\lambda_2 = 10$. Letters with a prime indicate the predicted values. $\mathcal{L}_{CE}$ is the cross entropy loss between the correct semantic class s and the predicted one s', d_i denotes the depth at pixel i and x_t and y_t describe the x and y coordinates of the expert at time step t.

3.4.2 Controller

We make some improvements to the controller. In the official CARLA leaderboard client, the LiDAR sensor is running at a lower frequency (10 Hz) than the simulation (20 Hz). The result is that the received LiDAR data will alternate between the front half of the car and the back half of the car (each 180°). For an example of this, see figure 3.13. Note that we only voxelize the left half for our input. Since our data is stored at even (10) intervals, we only have LiDAR data from the front inside our dataset (odd frames send the back half). To avoid having novel LiDAR data at test time we use an action repeat of 2 meaning we process every second input.

When treating lateral and longitudinal control independently (like we do here) a particular failure case can arise with the use of PID controllers. If the car stops and waits (e.g. at a red light) while the current lateral error is high, this high error can accumulate in the integral of the PID controller. After the car starts moving again the integral can dominate the other terms for a while leading to a wrongly executed maneuver and potentially a crash. For an example, see figure 3.14 where the failure happened after the agent waited at a red traffic light. This is a consequence of the fact that cars need to move forward in order to move laterally (and reduce lateral error). Our solution is to simply set the lateral error to 0 while the car is standing still, as it cannot be minimized anyway.

Imitation learning models are known to suffer from causal confusion problems, in particular the *inertia problem* [10]. The inertia problem describes the phenomenon

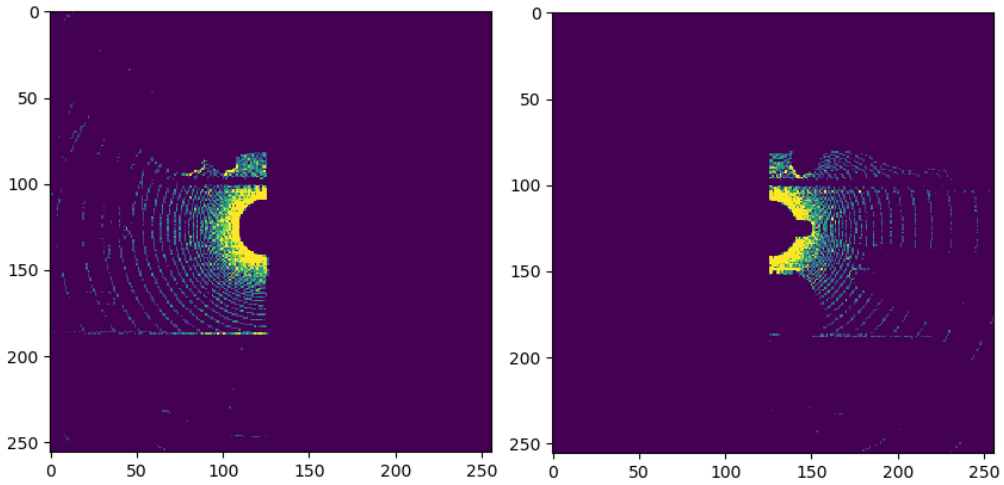


Figure 3.13: LiDAR input at two consecutive time steps projected to BEV. The left image represents even time steps, the right image represents odd time steps

Figure 3.14: Network predicts driving straight, but the PID controller fails to execute and crashes after waiting at a traffic light.

that when a vehicle stays still, the probability that it continues to stay in place (e.g. at a red light) is very high in the training data. This can lead to the trained agent never starting to drive again after having stopped because it learned to attend to these spurious correlations instead of the real causes. The causal confusion problems are even amplified in our situation because SEED has a full 360° field of view (FOV) and uses the velocity and acceleration of other cars in his reasoning process. Some causes for SEED's decisions cannot be discovered by the TransFuser agent because it only has a 180° FOV. The TransFuser architecture also can not estimate the velocity and acceleration because that would require temporal reasoning (tracking objects over time to estimate their velocity and acceleration). The architecture operates on a frame by frame basis without any memory. As shown in section 4.6 the TransFuser+ is heavily affected by the inertia problem.

Using speed prediction as auxiliary task during training has been proposed to alleviate this problem [10]. It is unclear however why his approach helps, it does not fully solve the inertia problem and has not been widely adopted since then. We propose another approach to address the inertia problem that is easy to understand and can be added on top of any method. We start with the observation that the causal confusions we encounter are brittle. Since there is no causal signal for the action, small changes such as creeping forward can already break the agent out of the confused state. Specifically, we add an **unblock controller** to our method. If the car has not moved for more than s (=45) seconds, we force the car to creep forward by setting the target speed of the PID controller to 4 m/s for a short amount of time (1.5 seconds). The value s is chosen to be higher than the amount of time we expect a car to wait at a red light. This creeping forward is enough to make the TransFuser+ agent drive on its own again. This approach alone however would be unsafe because it could happen for example that the agent is stuck in traffic where creeping forward could lead to a collision. Fortunately, this situation is easy to detect with the LiDAR. We additionally implement a **safety controller** that overwrites the unblock controller. It checks whether there are any LiDAR hits in a rectangular area in front of the car. Filtering noise hits is not necessary because CARLA does not simulate realistic LiDAR noise. If any are detected the target speed is set to 0 bringing the car to an immediate stop. The safety controller makes the unblock controller safe. As a nice bonus, the safety controller also sometimes prevents failure cases, where the agent crashes during normal driving mode.

4 Experiments

In this section we will describe our evaluation methodology, introduce the baselines and discuss results.

4.1 Task

We consider the task of driving in an urban setting in the realistic 3D simulator CARLA version 0.9.10.1 [3]. We use routes that cover both highway and residential areas. Agents are supposed to follow routes which are provided as GPS waypoints by an A* navigational planner. The goal is to arrive at the target location in a given amount of time while incurring as little infraction penalties as possible. Infractions that are penalized are:

- Collision with a pedestrian
- Collision with a vehicle
- Collision with a static object
- Running a red light
- Running a stop sign
- Driving on the wrong lane or sidewalk
- Leaving the route specified by the navigational planner

Along the routes, there are dangerous scenarios specified which the agent needs to resolve in order to safely arrive at his destination. There are currently 10 different scenarios specified:

- Rubble on the road leading to a loss of control.
- Leading vehicle suddenly performs an emergency brake.
- A pedestrian hidden behind a static object suddenly runs across the street.
- After performing a turn at an intersection, a cyclist suddenly drives across the street.
- A slow vehicle gets spawned in front of the agent.
- A static object is blocking the street.

- Crossing traffic is running a red light at an intersection
- The agent must perform an unprotected left turn at an intersection with oncoming traffic
- The agent must perform a right turn at an intersection with crossing traffic
- Crossing an unsignalized intersection.

4.2 Routes

To collect the dataset we follow the procedure of [7] collecting data from the 8 publicly available towns. The routes consist of many tiny routes where SEED crosses an intersection and some longer routes where SEED drives around the towns. A set of very long routes from town 1-6 is reserved for open loop validation. The data is stored every 10th frame (at 2 FPS) and the weather is changed at the same frequency. In total, the dataset contains roughly 130 thousand examples.
For our closed loop validation experiments, we follow the evaluation protocol of [7] using 42 routes that consist of 14 routes that are each repeated 3 times under different weather conditions. The weathers consist of combination between weather conditions (Clear, Cloudy, Wet, MidRain, WetCloudy, HardRain and SoftRain) and daylight conditions (Night, Twilight, Dawn, Morning, Noon, Sunset). For the privileged experts we only run one weather as the weather only affect the camera inputs used by the sensorimotor agents. We will call these routes the (NEAT) validation routes in the following. As test set, we use the official CARLA Leaderboard [15] test set. It consists of 10 secret routes evaluated on 2 unknown weather conditions with 5 repetitions (100 routes in total). Submissions are made by submitting the inference code in a docker container that is then secretly tested on a server by a third party. Performance results are substantially lower than on any other CARLA benchmark [7, 10, 3] indicating that the CARLA leaderboard is much more challenging. The CARLA leaderboard does not allow privileged access to the simulator, hence the expert drivers are evaluated on the validation routes only. Teams have a fixed monthly budget of 200 hours for evaluating submission. Because of this, not all our baselines are evaluated on the leaderboard.
Implementation details: We train the TransFuser+ method for 100 epochs using the AdamW [63] optimizer with a learning rate of 0.0001 and pick the epoch with the best waypoint open loop validation loss. In one of the ablations, we use a learning rate schedule and reduce the learning rate by a factor of 10 after 30 epochs.

4.3 Metrics

We measure the performance of the agents using the 3 official metrics of the CARLA leaderboard **Route Completion** (**RC**), **Infractions Score** (**IS**) and **Driving Score**

(**DS**).

4.3.1 Route Completion

Route Completion (RC) is the percentage of the route that is completed by the agent. If the agent drives on a wrong lane or sidewalk, that percentage of the route will not be counted. If the agent left the route specified by the navigational planner or gets blocked, the episode will end, effectively reducing the route completion.

4.3.2 Infraction Score

Infraction Score (IS) is a penalty for infractions where the agent starts with a score of 1.0. For every infraction, the score is multiplied by the penalty coefficient of that infraction type. The penalty coefficients are:

- Collision with a pedestrian: 0.50
- Collision with a vehicle: 0.60
- Collision with a static object: 0.65
- Running a red light: 0.70
- Running a stop sign: 0.80

Note that this means that subsequent infractions will have a lower impact due to the multiplicative nature of the score.

4.3.3 Driving Score

Driving Score is the main metric for performance and a combination of the previous two. It is calculated in the following way:

$$DS = \frac{1}{N}\sum_{i}^{N} RC_i IS_i \tag{4.1}$$

where N is the number of routes, RC_i the route completion of route i and IS_i the infraction score of route i. Note that this is not the same as multiplying the averaged route completion with the averaged infraction score. The driving score is a normalized metric, meaning that the best possible score is 100 and the worst score is 0. For the validation routes, we run them with 3 different seeds and report the mean and standard deviation of the 3 scores averaged across the routes.

4.4 Baselines

We compare our TransFuser+ method with several popular baselines, including the recent state-of-the-art approach World on Rails [8]. SEED is compared to the prior best experts, Roach [5] and TransFuser Expert [6].

4.4.1 Conditional Imitation Learning ResNet Speed

Conditional imitation learning [52] has become the dominant approach in imitation learning. All sensorimotor baseline described below are based on it. It is based on the observation that pure imitation learning has a problem at intersections. At intersections multiple valid paths are possible turning left or turning right for example. A pure imitation approach has no way of deciding between them and in the worst case takes an average between the options such as driving straight. Conditional imitation learning fixes this problem by conditioning the network on a command by the navigational planner (such as turn left or turn right) or supplying the network with a target waypoint. Conditional Imitation Learning extended with ResNet and Speed prediction **CILRS** [10] is such a conditional imitation learning approach that uses residual networks as image backbone and velocity supervision as auxiliary loss during training.

4.4.2 Learning by Cheating

Learning by Cheating **LBC** [53] is an approach in which a second privileged expert based on a neural network is trained to emulate a rule based expert. This second expert (the cheater) closely resembles the network structure of the sensorimotor agent that is trained to imitate it. It uses a semantic segmentation bird's eye view image as input that is processed by a ResNet and outputs waypoints that are processed by a PID controller to produce the driving controls. The sensorimotor agent uses a similar network design but takes as input a frontal camera image, the car's velocity and the conditional command by the navigational planner. Using the DAGGER [51] approach the cheater teaches the sensorimotor agent how to drive. Learning by Cheating was the first approach to solve the original CARLA benchmark.

4.4.3 Auto-regressive IMage-based waypoint prediction

AIM [6] is an improved version of CIRLS that predicts waypoints with a gated recurrent unit (GRU) [61] architecture and uses a PID controller for execution instead of directly predicting the control outputs. The goal is to predict the position of the expert driver 4 steps (a 0.5 seconds) into the future. The GRU unit then takes as input a 2D point and a hidden state. The hidden state is at the first time step the features from the encoded image, and at later steps the output of the previous time step. The input point to the GRU is the predicted waypoint from the last time step +

the target waypoint from the navigational planner. At the first time step, the last predicted waypoint is simply (0/0). The output of the GRU is then fed through a linear layer to be reduced to a 2D point and added to the last predicted waypoint. These predicted waypoints are then processed by a PID controller, which turns them into driving commands.

4.4.4 AIM Multi-Task

AIM-MT [7] is an improved version of AIM that uses multitask learning to improve the training. In particular, we report the version that uses 2D semantic segmentation and 2D depth estimation as auxiliary losses during training.

4.4.5 NEural ATtention fields

NEAT [7] is a conditional imitation method that takes three camera images as input. Like prior work, they are processed to features using ResNets. It aggregates the features from the different images with a Transformer block. For visualization, it can output a 2D bird's eye view map that describes at every pixel the 2D offset vector to the path that the car should follow. During inference, 9 points of this map are queried, and their result is averaged. It achieves this by representing the scene as implicit continuous functions represented by MLPs. The waypoint map is created by querying the network at each position in the map. During querying, the target point from the navigational planner is provided to the MLP. During training, the network is also trained to predict the auxiliary task of BEV semantic segmentation. PID controllers turn the vector to the path into the car's commands. The red light class from the semantic segmentation map is also used to stop at test time.

The results of the previously described baselines have been reported in [7].

4.4.6 TransFuser

The **TransFuser** [6] method is similar to AIM in its basic structure. It differs in that it also uses the LiDAR point cloud as input. The LiDAR point cloud is projected into a bird's eye view image and processed with a ResNet, similar to the image branch. Features between the image ResNet and the LiDAR ResNet are interchanged in the hidden layers using transformer blocks [54]. The TransFuser method is trained with the TransFuser Expert. It is the baseline for our TransFuser+ method that is trained with SEED.

4.4.7 World On Rails

Additionally, we reproduce the current state-of-the-art approach World on Rails (**WOR**) [8]. It is an image based method that learns from prerecorded driving logs.

To do that it makes the world on rails assumption, meaning that the agent's actions do not influence the environment. Similar to our approach, they use a bicycle model to model the ego car. However, the movement of all other cars is determined using the benefit of hindsight from the recorded logs. During training, the optimal action is then determined with the use of a reward function. For each data point in the training set, all possible actions (discretized) are simulated with the bicycle model and their reward is computed. The action of the expert driver is then the one that maximizes future reward (similar to Q-learning in reinforcement learning). The future reward is calculated using backward induction and dynamic programming. An imitation learning agent is then trained to imitate the action of this expert. More specifically, the sensorimotor agent predicts a categorical distribution over the discretized actions. During training, it is regularized with entropy maximization. Semantic segmentation is used as auxiliary loss and the images are augmented. Note that their expert can not be evaluated closed loop like the other experts discussed in this work, as it only works on a prerecorded dataset. In some sense this work is similar to learning by cheating as it uses an expert (the CARLA autopilot that collects the dataset) to build another expert (the Q function) which aims to improve the autopilot's actions using information from a reward function. This second agent then has the advantage of not only providing the optimal action to the sensorimotor policy, but also a score for the suboptimal actions. It is not very clear whether/how errors made by the autopilot propagate. Also, the Q function may not necessarily discover the real best action as it optimizes a local shaping reward and not the desired sparse global reward (the driving score).

4.4.8 TransFuser Expert

As for privileged experts, we report the performance of the expert used by the TransFuser and NEAT architectures [6, 7]. It is the method that our SEED method builds upon and uses a similar basic structure. Lateral control is done by minimizing the angle towards the next waypoint with a PID controller. Longitudinal control runs in two modes. Either the target speed is a fixed constant or zero in case the car needs to stop. The car stops to obey traffic laws (red lights and stop signs) or to avoid collision. Collision hazards are detected by checking if there are cars or pedestrians in a static area in front of the car. A PID controller then controls the car to drive at the desired target speed.

4.4.9 Reinforcement Learning Coach

Additionally, we reproduce the recent **Roach** method, which is the state-of-the-art approach for privileged experts. It uses a privileged bird's eye view input representation (based on semantic segmentation) to train a reinforcement learning agent. The author provided code base [64] uses an entirely new CARLA client for training and evaluation. This new client is unfortunately not compatible with the

scenario runner code base that is used to execute the 10 traffic scenarios that make the validation set challenging. To run the evaluation, we made the inference code run on the CARLA leaderboard client (which is publicly available at [65]). We note that Roach has not experienced the scenarios during training and hence needs to generalize to some extent during validation. Retraining the model weights with scenarios enabled might improve the performance, but would require substantial engineering efforts and training resources. This lack in flexibility is a major downside to reinforcement learning based approaches.

4.4.10 SEED - Soft Actor Critic

To show that the performance improvement compared to Roach does come from better planning and not better control, we train an MLP to do the control using the Soft Actor Critic (SAC) algorithm [28] with planning done using SEED's algorithm. We chose this particular reinforcement learning algorithm because it is designed for continuous control and is known to not require hyperparameter tuning. We confirm these findings, the algorithm worked on our problem with the default hyperparameters. As reward function we use the one from [9] with the implementation details reported in [45]. Like [9] we also encounter the issue of oscillating driving behavior which we also fix using an ensemble (of 6 networks). The input to the controller MLP is the difference to the desired target speed, its integral and derivative, the angle to the next waypoint, its integral and derivative and a binary flag on whether to do an emergency brake. The output is a continuous value for throttle and steer. If throttle is negative, we perform a full brake. The method is trained on tiny and short routes from Town 01 with an action repeat of 4. While we found action repeat to be important for training, it is unnecessary during evaluation, so we set it to 1. While it is possible to achieve the same level of performance with reinforcement learning than with a classical PID controller, the latter is much easier to implement and does not require training. Hence, we do recommend the SEED version that uses PID for control.

4.5 Results

The results for the experts on the NEAT validation routes are shown in table 4.1. The results of the NEAT validation routes for the sensorimotor agents are shown in table 4.2. A snapshot of the publicly visible submissions on the CARLA leaderboard is shown in table 4.4.

The proposed SEED expert outperforms the TransFuser Expert baseline by 28 points in driving score, reducing the remaining error by roughly a factor of 4. It does so primarily through safer driving, which can be observed through the increased infraction score (by 18 percent). The route completion is also increased by roughly 10 percent. This is somewhat surprising as both methods use the same controller for

Method	RC ↑	IS ↑	DS ↑
TransFuser Expert [6]	86.05 ± 2.58	0.76 ± 0.01	62.69 ± 2.36
Roach [5]	91.77 ± 0.75	0.79 ± 0.04	74.21 ± 3.23
SEED - static	90.57 ± 2.57	0.96 ± 0.04	86.18 ± 2.22
SEED - SAC (ours)	100.0 ± 0.0	0.95 ± 0.01	95.16 ± 1.01
SEED (ours)	96.95 ± 1.14	0.94 ± 0.01	91.16 ± 2.17

Table 4.1: Results on the NEAT validation routes for expert drivers.

Method	Aux. Sup.	RC ↑	IS ↑	DS ↑
CILRS [10]	Velocity	35.46 ± 0.41	0.66 ± 0.02	22.97 ± 0.90
LBC [53]	BEV Sem	61.35 ± 2.26	0.57 ± 0.02	29.07 ± 0.67
TransFuser [6]	None	70.22 ± 4.57	0.76 ± 0.02	50.59 ± 2.76
AIM [6]	None	70.04 ± 2.31	0.73 ± 0.03	51.25 ± 0.17
TF + new controller	None	83.19 ± 1.41	0.62 ± 0.01	52.07 ± 1.93
WOR [8]	2D Sem	81.20 ± 1.99	0.74 ± 0.01	60.63 ± 0.22
AIM-MT [7]	Depth+2D Sem	80.81 ± 2.47	0.80 ± 0.01	64.86 ± 2.52
NEAT [7]	BEV Sem	79.17 ± 3.25	**0.82** ± 0.01	65.10 ± 1.75
TransFuser+ (Ours)	Depth+2D Sem	**90.38** ± 3.37	0.78 ± 0.06	**71.65** ± 4.36

Table 4.2: Results on the NEAT validation routes for sensorimotor agents.

lateral control. We hypothesize that this is a result of fewer crashes (which can block the agent and therefore reduce route completion), and fewer false positives of the collision detection algorithm. SEED also outperforms the previous state-of-the-art approach Roach by 16.95 points in driving score, again primarily due to safer driving. We recorded the validation runs with a bird's eye view spectator to visually inspect the infractions. Out of the remaining infractions, none are the agent's fault. They occur due to measurement errors by the simulator. Some examples are other cars actively driving into the waiting expert or driving over the dirt object on a slope is incorrectly attributed as collision. SEED can therefore be said to have achieved perfectly safe driving at the current task complexity. It is still possible to improve the route completion, as SEED will always wait if something is blocking its path. In some situations which we call *route blocked* it is possible to perform an overtaking maneuver instead. On very short routes, *timeouts* can happen because the agent is driving rather slow. At intersections, it can sometimes happen that the agent starts driving too early and ends up stopping midway because oncoming traffic is blocking its way. Both cars are blocking each other's path and cannot move forward anymore to resolve the situation. We call these situations *stareoffs*. Addressing these problems is an interesting direction for future work, though one might need to design validation routes first where these situations occur frequently. On the current validation routes, these situations can occur randomly. To give an intuition how often these situations occur, we show in table 4.3 how often they occurred in each of SEED's 3 repetitions on the NEAT validation routes.

Cause	Repetition 1	Repetition 2	Repetition 3
Route blocked	0	2	0
Timeout	0	0	0
Stareoff	1	0	1

Table 4.3: Counts of how often situations that reduce route completion happened during SEED's validation run.

It can happen that none of these situations occur, as is shown by the SEED - SAC ablation. The resulting higher driving score of SEED - SAC is a result of this randomness in the validation.

On the side of the sensorimotor agents, our method TransFuser+ outperforms all methods trained using the TransFuser Expert. It has a 6.55 better driving score than NEAT, the best method trained against the TransFuser expert, which is a relative improvement of 10%.

#	**Method**	**RC** ↑	**IS** ↑	**DS** ↑
1	GRIAD[1]	61.85	0.60	**36.79**
2	TransFuser+ (ours)	**69.84**	0.56	34.58
3	WOR [8]	57.65	0.56	31.37
4	MaRLn [9]	46.97	0.52	24.98
5	NEAT [7]	41.71	0.65	21.83
6	AIM-MT [7]	67.02	0.39	19.38
7	TransFuser [6]	51.82	0.42	16.93
8	Pylot[2] [16]	48.63	0.50	16.70
9	NEU408[1]	40.40	0.51	15.61
10	LBC [53]	17.54	**0.73**	8.94
11	CILRS [10]	14.40	0.55	5.37
12	CaRINA [66]	23.80	0.41	4.56
13	Cadre v1[1]	65.66	0.07	2.77

Table 4.4: The CARLA leaderboard accessed 03.09.2021.

On the challenging CARLA leaderboard, TransFuser+ doubles the performance of the original TransFuser submission with no changes made to the inference architecture of the network. It outperforms the previous state-of-the-art approach World on Rails (WOR) by 3.21 points in driving score, a relative improvement of 10%. The number 1 ranked method General Reinforced Imitation for Autonomous Driving (GRIAD) is concurrent work that at the time of writing has no public report yet. Out of all the publicly available submissions, TransFuser+ has the highest route completion. The previously best two methods, WOR and MaRLn both use some

[1]No report is available for this submission, at the time of writing.
[2]Uses HD-Map

form of reinforcement learning to supplement the supervised learning. In this work we argue that the expert generating the data, hence the label quality, was what was holding pure imitation learning approaches back and show that they can still achieve competitive performance.

4.6 Ablation Study

The entry SEED - static in table 4.1 gives insight into the contributions of SEED's components. The traffic rules logic of SEED is combined with the static collision avoidance logic of the TransFuser Expert in this experiment. The traffic rules logic contributes 23.49 points in driving score. Running red lights was the biggest problem of the TransFuser Expert. We recorded the validation run and additionally report that 5 avoidable collisions happened in the 3 repetitions. The average infraction score is surprisingly high because no unavoidable infractions happened. The agent also got stuck 5 times due to false positives of the collision logic.
This shows, that the collision avoidance logic of SEED is necessary to reach the highest possible driving score and safety.

To understand the impact of the different components of the sensorimotor architecture TransFuser+, we show multiple submissions on the CARLA leaderboard in table 4.5.

Method	**RC** ↑	**IS** ↑	**DS** ↑
TransFuser [6]	51.82	0.42	16.93
TransFuser + SEED	34.85	**0.72**	21.05
TransFuser + SEED + unblock	63.71	0.58	31.80
TransFuser + SEED + unblock + aux. loss	**72.45**	0.53	33.11
TransFuser + SEED + unblock + aux. loss + lr. schedule	69.84	0.56	**34.58**

Table 4.5: Ablations on the CARLA leaderboard.

First, we can see that training the TransFuser with data from SEED improved the safety of the resulting model. The infraction score went up by 30%. However, the new dataset introduces a new problem where the agent simply stops driving at certain points, reducing the route completion by 16.97. This we can deduce from the additional metrics the leaderboard provides. The submission gets a score of 11004.34 Agent blocked /km. This means that on 11 routes, the agent never started to drive. We hypothesize that this is because the TransFuser architecture can not understand certain decisions made by SEED that would require temporal reasoning or a 360° field of view. It therefore miss attributes the causes for SEED to stop to correlations in its input data, leading to unnecessary stops during evaluation. The unblock controller described in section 3.4.2 fixes this particular issue, improving the overall driving score by 10.75 points and reducing the agent blocked metric to 3.29. The submission now strictly dominates the original TransFuser in all 3 metrics. The unblock controller reduces average infraction score. We think this is because when

the agent drives more, then there are more possibilities for accidents to happen. For example, not driving at all would give you a perfect infraction score of 1.0 but of course the worst driving score of 0.0.
To show that performance improvements come from the combination of better expert and controller and not just from the new controller changes, we run an ablation of the old TransFuser weights with the added controller improvements. It is shown in table 4.2 as the entry TF + new controller. As shown, the controller changes increase the route completion of the TransFuser. The old TransFuser however is worse at driving safely, so driving more led to a lower average infraction score. As a result, the driving score stayed relatively similar, showing that the new SEED expert is the primary reason for the performance improvement.
Adding depth and semantic segmentation auxiliary losses improves the driving score a little by 1.31 points. Lastly, tuning the learning rate schedule improved the performance by another 1.47 points. This submission, which we call TransFuser+ now doubles the score of the original TransFuser submission. This is achieved through better training signals and controller improvements with no changes made to the inference architecture.

Runtime: We evaluate the runtime performance of our methods on a laptop with an Intel Core i7-9750H CPU and an NVIDIA RTX 2070 GPU. To measure the runtime we take the average over the last 100 cycles of the run_step function which provides the sensor data and expects the car commands as return. TransFuser+ runs at a speed of roughly 28 ms per cycle on average. SEED's performance depends on the traffic density around him (it performs collision checks for nearby vehicles). We ran the experiment on 3 validation routes and observed runtime values between 8 ms and 39 ms. Both methods can be considered to run in real time, which allows for fast data collection and evaluation. Note that when one collects data with SEED the time for preprocessing and storing sensor data on the hard-drive needs to be added.

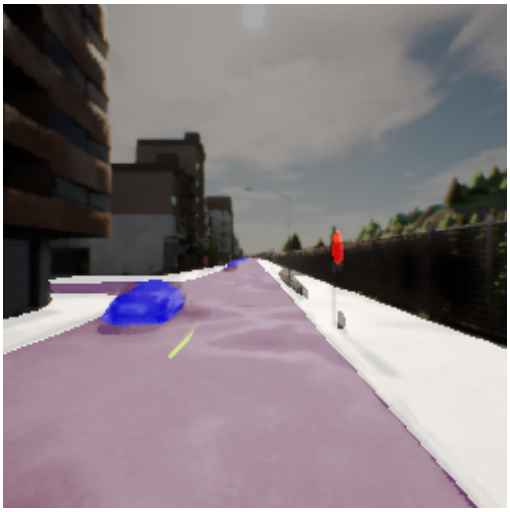
4.7 Visualizations

Examples of SEED and TransFuser+ driving a route can be found in the supplementary videos at [67] and [68].
Another advantage of auxiliary supervision is that it can be used to gain some insight into the end-to-end model by looking at the auxiliary predictions during inference. An example of the different inputs and outputs is shown in figure 4.1. The scene depicts the car waiting at an intersection in the European town 01. The input RGB camera image is shown in figure 4.1a and the two LiDAR channels are depicted in the figures 4.1c and 4.1d. They show the LiDAR point cloud projected to bird's eye view separated at 50 cm above the ground. You can see the depth prediction made by the network in figure 4.1e and the semantic segmentation prediction in figure 4.1f. The semantic segmentation prediction is overlaid with 60% opacity on top of the RGB input in figure 4.1b. The depth prediction seems to be able to decently

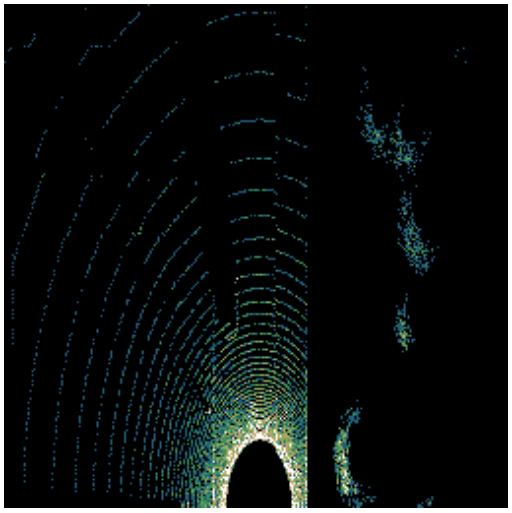
estimate the skyline above the horizon and the sky, but the closer depths are all very close to zero. The reason for this is the normalization of the training data, which is normalized to depth in kilometers. Depth values that are close to the car are almost zero, which in the visualization just look black. The network is only able to roughly estimate the depth, and much structure is lost. This kind of depth supervision has been shown to be beneficial to learning [7] but further improvements might be possible with a more advanced depth prediction architecture.
The segmentation image is better, the road, lane marking, sidewalks and red light are estimated nearly perfectly. This corresponds to our observation that the agent is very good at lane following and waiting at European style red lights. The estimation of the car is somewhat rough, but generally where it should be.

Figure 4.2 shows two different examples of the semantic segmentation at US style intersections in town 03. For the most part, the impressions are similar to the European town. The road, lane markings, cars and sidewalks are roughly fine. In US style towns, the agent does struggle however with detecting the far away red lights. In figure 4.2c only one of the two red lights is detected, and in figure 4.2d the red lights are not detected. At intersection 2 the agent will run the red light. The reason for this might be the rather low resolution (256x256) of the input RGB image. Since US traffic lights are on the other side of the road, they are very hard to spot in the input RGB image. How to scale architectures to work with higher resolution input cameras seems to be an important question going forward.

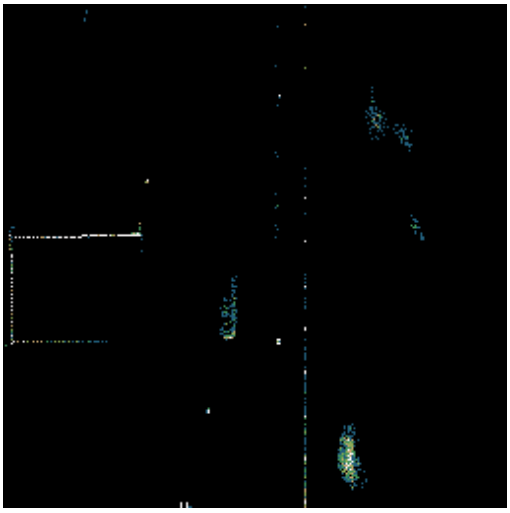
(a) The RGB camera input.



(b) Semantic segmentation overlay

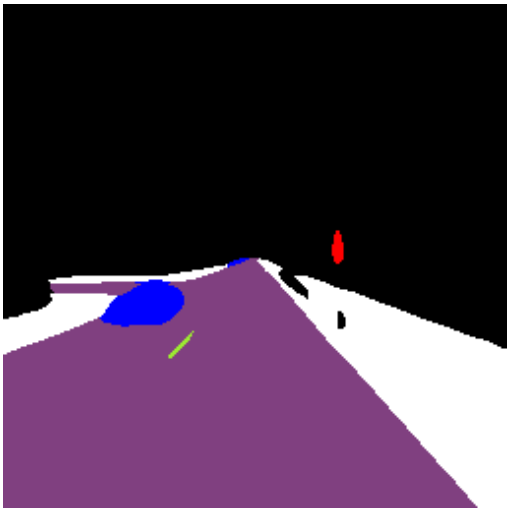


(c) LiDAR input channel 1



(d) LiDAR input channel 2

(e) Depth prediction

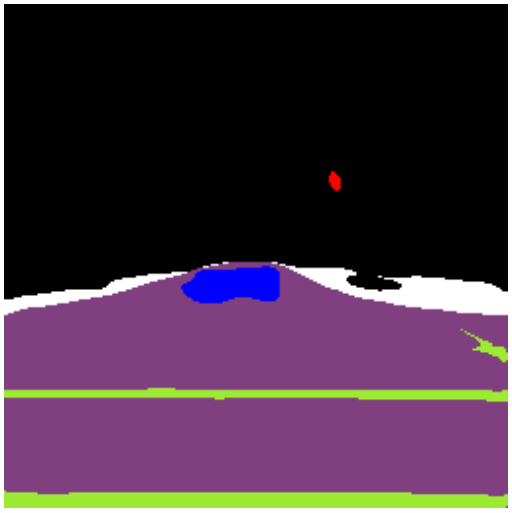


(f) Semantic segmentation prediction

Figure 4.1: Example of the input to the network and corresponding depth and semantic segmentation predictions.

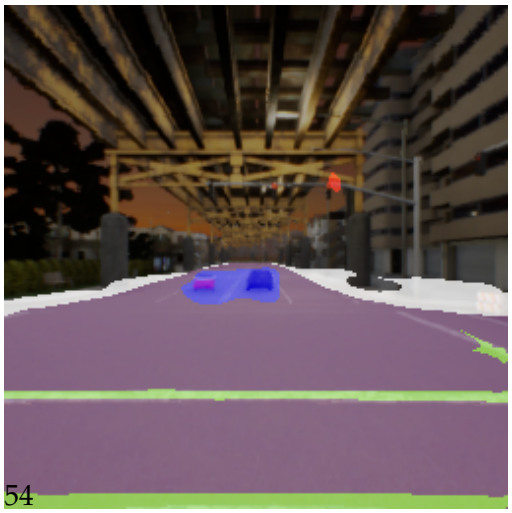
(a) The RGB camera input at intersection 1

(b) The RGB camera input at intersection 2

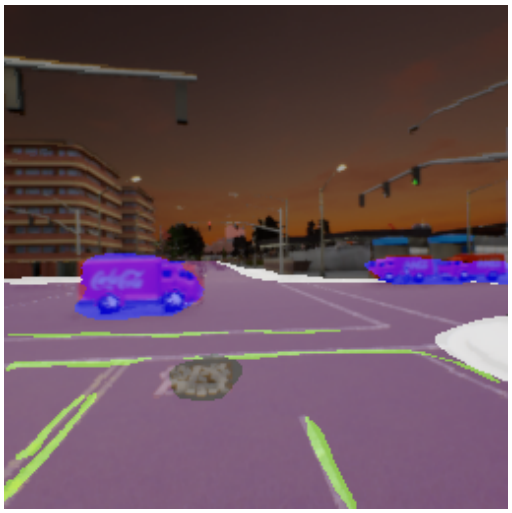


(c) Semantic segmentation prediction at intersection 1

(d) Semantic segmentation prediction at intersection 2



(e) Semantic segmentation overlay at intersection 1



(f) Semantic segmentation overlay at intersection 2

Figure 4.2: Example of the RGB input and semantic segmentation prediction at two different US style intersections.

5 Conclusion

Contemporary imitation learning approaches are held back by the quality of the expert they are imitating. We have shown how to design privileged experts such that they are able to drive safely in the CARLA simulator even in the presence of adversarial scenarios. The resulting improved label quality translates into improvements of existing imitation learning architectures. An important problem we encountered with the new dataset was the inertia problem, wherein the sensorimotor agent does not continue driving after having stopped. We show how a simple unblock controller mechanism is enough to fix this problem, but more elegant solutions might also exist. While the imitation learning agent has been improved, its performance is now lower than that of the expert it is imitating. This reveals some important flaws in contemporary neural network architecture design.
First, all current approaches developed for CARLA only use frontal vision. However, to achieve full self-driving, a full 360° field of view is strictly necessary. Without this, it is impossible to safely perform lane changes. An interesting direction for future work is therefore how to integrate sensor information from the back side of the car that is only relevant in rare but important situations. Since lane changes are rare in current CARLA benchmarks, new validation sets might also need to be developed in order to measure progress.
Secondly, trajectory forecasting, whether done explicitly or implicitly, is an integral part of self-driving. It requires however the use of temporal reasoning to estimate the velocity and acceleration of moving objects in the scene. Only few approaches [69, 9] in CARLA currently make use of temporal data at all. It is an important open question how to effectively integrate temporal data in end-to-end imitation learning architectures. SEED gives an answer to these questions for privileged expert drivers, and we hope it will help the community answer these questions for sensorimotor architectures.
SEED is able to safely drive through CARLA yet occasionally gets stuck by cars blocking the route. Developing an extension to perform overtaking maneuvers would be an interesting direction to further improve it.
Lastly, at the current image resolution, traffic lights on the other side of the intersection (in US towns) can be hard or impossible to spot. Scaling architectures to higher input resolutions is therefore an important open question, necessary to achieve zero red light infractions.

Bibliography

[1] W. H. Organization, "Global health estimates 2019: Deaths by cause, age, sex, by country and by region, 2000-2019." 2020.

[2] S. Singh, "Critical reasons for crashes investigated in the national motor vehicle crash causation survey," 2015.

[3] A. Dosovitskiy, G. Ros, F. Codevilla, A. M. López, and V. Koltun, "CARLA: an open urban driving simulator," in *1st Annual Conference on Robot Learning, CoRL 2017, Mountain View, California, USA, November 13-15, 2017, Proceedings*, ser. Proceedings of Machine Learning Research, vol. 78. PMLR, 2017, pp. 1–16. [Online]. Available: http://proceedings.mlr.press/v78/dosovitskiy17a.html

[4] M. Behrisch, D. Krajzewicz, and M. Weber, Eds., *Simulation of Urban Mobility - First International Conference, SUMO 2013, Berlin, Germany, May 15-17, 2013. Revised Selected Papers*, ser. Lecture Notes in Computer Science, vol. 8594. Springer, 2014. [Online]. Available: https://doi.org/10.1007/978-3-662-45079-6

[5] Z. Zhang, A. Liniger, D. Dai, F. Yu, and L. Van Gool, "End-to-end urban driving by imitating a reinforcement learning coach," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

[6] A. Prakash, K. Chitta, and A. Geiger, "Multi-modal fusion transformer for end-to-end autonomous driving," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*. Computer Vision Foundation / IEEE, 2021, pp. 7077–7087. [Online]. Available: https://openaccess.thecvf.com/content/CVPR2021/html/Prakash_Multi-Modal_Fusion_Transformer_for_End-to-End_Autonomous_Driving_CVPR_2021_paper.html

[7] K. Chitta, A. Prakash, and A. Geiger, "Neat: Neural attention fields for end-to-end autonomous driving," in *International Conference on Computer Vision (ICCV)*, 2021.

[8] D. Chen, V. Koltun, and P. Krähenbühl, "Learning to drive from a world on rails," in *ICCV*, 2021.

[9] M. Toromanoff, É. Wirbel, and F. Moutarde, "End-to-end model-free reinforcement learning for urban driving using implicit affordances," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*. Computer Vision Foundation / IEEE, 2020, pp.

7151–7160. [Online]. Available: https://openaccess.thecvf.com/content_CVPR_2020/html/Toromanoff_End-to-End_Model-Free_Reinforcement_Learning_for_Urban_Driving_Using_Implicit_Affordances_CVPR_2020_paper.html

[10] F. Codevilla, E. Santana, A. M. López, and A. Gaidon, "Exploring the limitations of behavior cloning for autonomous driving," in *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*. IEEE, 2019, pp. 9328–9337. [Online]. Available: https://doi.org/10.1109/ICCV.2019.00942

[11] "Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles ," https://www.sae.org/standards/content/j3016_202104/, 2021, [Online; accessed 25-September-2021].

[12] D. Pomerleau, "ALVINN: an autonomous land vehicle in a neural network," in *Advances in Neural Information Processing Systems 1, [NIPS Conference, Denver, Colorado, USA, 1988]*, D. S. Touretzky, Ed. Morgan Kaufmann, 1988, pp. 305–313. [Online]. Available: http://papers.nips.cc/paper/95-alvinn-an-autonomous-land-vehicle-in-a-neural-network

[13] E. D. Dickmanns, "Dynamic machine vision," http://dyna-vision.de/, 1995, [Online; accessed 27-September-2021].

[14] A. Tampuu, M. Semikin, N. Muhammad, D. Fishman, and T. Matiisen, "A survey of end-to-end driving: Architectures and training methods," *CoRR*, vol. abs/2003.06404, 2020. [Online]. Available: https://arxiv.org/abs/2003.06404

[15] "CARLA Autonomous Driving Leaderboard," https://leaderboard.carla.org/, 2020, [Online; accessed 03-September-2021].

[16] I. Gog, S. Kalra, P. Schafhalter, M. A. Wright, J. E. Gonzalez, and I. Stoica, "Pylot: A modular platform for exploring latency-accuracy tradeoffs in autonomous vehicles," *CoRR*, vol. abs/2104.07830, 2021. [Online]. Available: https://arxiv.org/abs/2104.07830

[17] S. Casas, A. Sadat, and R. Urtasun, "MP3: A unified model to map, perceive, predict and plan," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*. Computer Vision Foundation / IEEE, 2021, pp. 14 403–14 412. [Online]. Available: https://openaccess.thecvf.com/content/CVPR2021/html/Casas_MP3_A_Unified_Model_To_Map_Perceive_Predict_and_Plan_CVPR_2021_paper.html

[18] M. Li, Y. Wang, and D. Ramanan, "Towards streaming perception," in *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part II*, ser. Lecture Notes in Computer Science, A. Vedaldi, H. Bischof, T. Brox, and J. Frahm, Eds., vol. 12347. Springer, 2020, pp. 473–488. [Online]. Available: https://doi.org/10.1007/978-3-030-58536-5_28

[19] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The KITTI dataset," *Int. J. Robotics Res.*, vol. 32, no. 11, pp. 1231–1237, 2013. [Online]. Available: https://doi.org/10.1177/0278364913491297

[20] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2016, pp. 3213–3223. [Online]. Available: https://doi.org/10.1109/CVPR.2016.350

[21] H. Fan, F. Zhu, C. Liu, L. Zhang, L. Zhuang, D. Li, W. Zhu, J. Hu, H. Li, and Q. Kong, "Baidu apollo EM motion planner," *CoRR*, vol. abs/1807.08048, 2018. [Online]. Available: http://arxiv.org/abs/1807.08048

[22] B. Paden, M. Cáp, S. Z. Yong, D. S. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Trans. Intell. Veh.*, vol. 1, no. 1, pp. 33–55, 2016. [Online]. Available: https://doi.org/10.1109/TIV.2016.2578706

[23] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *J. Artif. Intell. Res.*, vol. 47, pp. 253–279, 2013. [Online]. Available: https://doi.org/10.1613/jair.3912

[24] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nat.*, vol. 518, no. 7540, pp. 529–533, 2015. [Online]. Available: https://doi.org/10.1038/nature14236

[25] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. A. Riedmiller, "Deterministic policy gradient algorithms," in *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, ser. JMLR Workshop and Conference Proceedings, vol. 32. JMLR.org, 2014, pp. 387–395. [Online]. Available: http://proceedings.mlr.press/v32/silver14.html

[26] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2016. [Online]. Available: http://arxiv.org/abs/1509.02971

[27] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, ser. Proceedings of Machine Learning Research, J. G. Dy and

A. Krause, Eds., vol. 80. PMLR, 2018, pp. 1856–1865. [Online]. Available: http://proceedings.mlr.press/v80/haarnoja18b.html

[28] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine, "Soft actor-critic algorithms and applications," *CoRR*, vol. abs/1812.05905, 2018. [Online]. Available: http://arxiv.org/abs/1812.05905

[29] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. A. Sallab, S. K. Yogamani, and P. Pérez, "Deep reinforcement learning for autonomous driving: A survey," *CoRR*, vol. abs/2002.00444, 2020. [Online]. Available: https://arxiv.org/abs/2002.00444

[30] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J.-M. Allen, V.-D. Lam, A. Bewley, and A. Shah, "Learning to drive in a day," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 8248–8254.

[31] D. Isele, R. Rahimi, A. Cosgun, K. Subramanian, and K. Fujimura, "Navigating occluded intersections with autonomous vehicles using deep reinforcement learning," in *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*. IEEE, 2018, pp. 2034–2039. [Online]. Available: https://doi.org/10.1109/ICRA.2018.8461233

[32] C. Li and K. Czarnecki, "Urban driving with multi-objective deep reinforcement learning," in *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '19, Montreal, QC, Canada, May 13-17, 2019*, E. Elkind, M. Veloso, N. Agmon, and M. E. Taylor, Eds. International Foundation for Autonomous Agents and Multiagent Systems, 2019, pp. 359–367. [Online]. Available: http://dl.acm.org/citation.cfm?id=3331714

[33] M. Hügle, G. Kalweit, B. Mirchevska, M. Werling, and J. Boedecker, "Dynamic input for deep reinforcement learning in autonomous driving," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2019, Macau, SAR, China, November 3-8, 2019*. IEEE, 2019, pp. 7566–7573. [Online]. Available: https://doi.org/10.1109/IROS40897.2019.8968560

[34] B. Mirchevska, C. Pek, M. Werling, M. Althoff, and J. Boedecker, "High-level decision making for safe and reasonable autonomous lane changing using reinforcement learning," in *21st International Conference on Intelligent Transportation Systems, ITSC 2018, Maui, HI, USA, November 4-7, 2018*, W. Zhang, A. M. Bayen, J. J. S. Medina, and M. J. Barth, Eds. IEEE, 2018, pp. 2156–2162. [Online]. Available: https://doi.org/10.1109/ITSC.2018.8569448

[35] M. Jaritz, R. de Charette, M. Toromanoff, E. Perot, and F. Nashashibi, "End-to-end race driving with deep reinforcement learning," in *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*. IEEE, 2018, pp. 2070–2075. [Online]. Available: https://doi.org/10.1109/ICRA.2018.8460934

[36] P. Cai, Y. Luo, A. Saxena, D. Hsu, and W. S. Lee, "Lets-drive: Driving in a crowd by learning from tree search," in *Robotics: Science and Systems XV, University of Freiburg, Freiburg im Breisgau, Germany, June 22-26, 2019*, A. Bicchi, H. Kress-Gazit, and S. Hutchinson, Eds., 2019. [Online]. Available: https://doi.org/10.15607/RSS.2019.XV.018

[37] J. Chen, B. Yuan, and M. Tomizuka, "Model-free deep reinforcement learning for urban autonomous driving," in *2019 IEEE Intelligent Transportation Systems Conference, ITSC 2019, Auckland, New Zealand, October 27-30, 2019*. IEEE, 2019, pp. 2765–2771. [Online]. Available: https://doi.org/10.1109/ITSC.2019.8917306

[38] J. Wang, Y. Wang, D. Zhang, Y. Yang, and R. Xiong, "Learning hierarchical behavior and motion planning for autonomous driving," in *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2020, Las Vegas, NV, USA, October 24, 2020 - January 24, 2021*. IEEE, 2020, pp. 2235–2242. [Online]. Available: https://doi.org/10.1109/IROS45743.2020.9341647

[39] C. Paxton, V. Raman, G. D. Hager, and M. Kobilarov, "Combining neural networks and tree search for task and motion planning in challenging environments," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2017, Vancouver, BC, Canada, September 24-28, 2017*. IEEE, 2017, pp. 6059–6066. [Online]. Available: https://doi.org/10.1109/IROS.2017.8206505

[40] G. Liu, A. Siravuru, S. P. Selvaraj, M. M. Veloso, and G. Kantor, "Learning end-to-end multimodal sensor policies for autonomous navigation," in *1st Annual Conference on Robot Learning, CoRL 2017, Mountain View, California, USA, November 13-15, 2017, Proceedings*, ser. Proceedings of Machine Learning Research, vol. 78. PMLR, 2017, pp. 249–261. [Online]. Available: http://proceedings.mlr.press/v78/liu17a.html

[41] M. Henaff, A. Canziani, and Y. LeCun, "Model-predictive policy learning with uncertainty regularization for driving in dense traffic," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. [Online]. Available: https://openreview.net/forum?id=HygQBn0cYm

[42] S. Aradi, T. Bécsi, and P. Gaspar, "Policy gradient based reinforcement learning approach for autonomous highway driving," in *IEEE Conference on Control Technology and Applications, CCTA 2018, Copenhagen, Denmark, August 21-24, 2018*. IEEE, 2018, pp. 670–675. [Online]. Available: https://doi.org/10.1109/CCTA.2018.8511514

[43] Y. Tang, "Towards learning multi-agent negotiations via self-play," in *2019 IEEE/CVF International Conference on Computer Vision Workshops, ICCV Workshops 2019, Seoul, Korea (South), October 27-28, 2019*. IEEE, 2019, pp. 2427–2435. [Online]. Available: https://doi.org/10.1109/ICCVW.2019.00297

[44] P. Wang, H. Li, and C. Chan, "Quadratic q-network for learning continuous

control for autonomous vehicles," *CoRR*, vol. abs/1912.00074, 2019. [Online]. Available: http://arxiv.org/abs/1912.00074

[45] W. B. Knox, A. Allievi, H. Banzhaf, F. Schmitt, and P. Stone, "Reward (mis)design for autonomous driving," *CoRR*, vol. abs/2104.13906, 2021. [Online]. Available: https://arxiv.org/abs/2104.13906

[46] K. Min, H. Kim, and K. Huh, "Deep distributional reinforcement learning based high-level driving policy determination," *IEEE Trans. Intell. Veh.*, vol. 4, no. 3, pp. 416–424, 2019. [Online]. Available: https://doi.org/10.1109/TIV.2019.2919467

[47] T. Agarwal, H. Arora, and J. Schneider, "Affordance-based reinforcement learning for urban driving," *CoRR*, vol. abs/2101.05970, 2021. [Online]. Available: https://arxiv.org/abs/2101.05970

[48] A. Brunnbauer, L. Berducci, A. Brandstätter, M. Lechner, R. M. Hasani, D. Rus, and R. Grosu, "Model-based versus model-free deep reinforcement learning for autonomous racing cars," *CoRR*, vol. abs/2103.04909, 2021. [Online]. Available: https://arxiv.org/abs/2103.04909

[49] J. Bjorck, C. P. Gomes, and K. Q. Weinberger, "Towards deeper deep reinforcement learning," *CoRR*, vol. abs/2106.01151, 2021. [Online]. Available: https://arxiv.org/abs/2106.01151

[50] X. Liang, T. Wang, L. Yang, and E. P. Xing, "CIRL: controllable imitative reinforcement learning for vision-based self-driving," in *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part VII*, ser. Lecture Notes in Computer Science, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds., vol. 11211. Springer, 2018, pp. 604–620. [Online]. Available: https://doi.org/10.1007/978-3-030-01234-2_36

[51] S. Ross, G. J. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011*, ser. JMLR Proceedings, G. J. Gordon, D. B. Dunson, and M. Dudík, Eds., vol. 15. JMLR.org, 2011, pp. 627–635. [Online]. Available: http://proceedings.mlr.press/v15/ross11a/ross11a.pdf

[52] F. Codevilla, M. Müller, A. M. López, V. Koltun, and A. Dosovitskiy, "End-to-end driving via conditional imitation learning," in *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*. IEEE, 2018, pp. 1–9. [Online]. Available: https://doi.org/10.1109/ICRA.2018.8460487

[53] D. Chen, B. Zhou, V. Koltun, and P. Krähenbühl, "Learning by cheating," in *3rd Annual Conference on Robot Learning, CoRL 2019, Osaka, Japan, October 30 - November 1, 2019, Proceedings*, ser. Proceedings of Machine Learning Research,

L. P. Kaelbling, D. Kragic, and K. Sugiura, Eds., vol. 100. PMLR, 2019, pp. 66–75. [Online]. Available: http://proceedings.mlr.press/v100/chen20a.html

[54] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, Eds., 2017, pp. 5998–6008. [Online]. Available: https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

[55] D. Eberly, "Dynamic collision detection using oriented bounding boxes," 1999.

[56] P. Polack, F. Altché, B. d'Andréa-Novel, and A. de La Fortelle, "The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles?" in *IEEE Intelligent Vehicles Symposium, IV 2017, Los Angeles, CA, USA, June 11-14, 2017*. IEEE, 2017, pp. 812–818. [Online]. Available: https://doi.org/10.1109/IVS.2017.7995816

[57] T. Hagglund and K. J. Astrom, "Pid controllers: theory, design, and tuning," *ISA-The Instrumentation, Systems, and Automation Society*, 1995.

[58] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. 4, no. 2, pp. 100–107, 1968. [Online]. Available: https://doi.org/10.1109/TSSC.1968.300136

[59] R. C. Coulter, "Implementation of the pure pursuit path tracking algorithm," Carnegie-Mellon UNIV Pittsburgh PA Robotics INST, Tech. Rep., 1992.

[60] "transfuser github repository," https://github.com/autonomousvision/transfuser, 2021, [Online; accessed 12-September-2021].

[61] K. Cho, B. van Merrienboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, A. Moschitti, B. Pang, and W. Daelemans, Eds. ACL, 2014, pp. 1724–1734. [Online]. Available: https://doi.org/10.3115/v1/d14-1179

[62] R. Caruana, "Multitask learning," *Mach. Learn.*, vol. 28, no. 1, pp. 41–75, 1997. [Online]. Available: https://doi.org/10.1023/A:1007379606734

[63] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. [Online]. Available: https://openreview.net/forum?id=Bkg6RiCqY7

[64] "CARLA-Roach github repository," https://github.com/zhejz/carla-roach, 2021, [Online; accessed 05-September-2021].

[65] "Roach inference code made compatible with the CARLA leaderboard client," https://github.com/Kait0/carla-roach, 2021, [Online; accessed 05-September-2021].

[66] L. A. Rosero, I. P. Gomes, J. A. R. da Silva, T. C. dos Santos, A. T. M. Nakamura, J. Amaro, D. F. Wolf, and F. S. Osório, "A software architecture for autonomous vehicles: Team LRM-B entry in the first CARLA autonomous driving challenge," *CoRR*, vol. abs/2010.12598, 2020. [Online]. Available: https://arxiv.org/abs/2010.12598

[67] "An example of SEED driving." https://www.youtube.com/watch?v=rnjPBDBKNlw, 2021, [Online; accessed 25-September-2021].

[68] "An example of TransFuser+ driving." https://www.youtube.com/watch?v=wbe0xa0nqI4, 2021, [Online; accessed 26-September-2021].

[69] A. Sauer, N. Savinov, and A. Geiger, "Conditional affordance learning for driving in urban environments," in *2nd Annual Conference on Robot Learning, CoRL 2018, Zürich, Switzerland, 29-31 October 2018, Proceedings*, ser. Proceedings of Machine Learning Research, vol. 87. PMLR, 2018, pp. 237–252. [Online]. Available: http://proceedings.mlr.press/v87/sauer18a.html