Exploring Supervised and Self-supervised Learning in Autonomous Driving

Partha Ghosh
University of Tübingen

Abstract

Acknowledgements

I want to thank Bernhard Jaeger and Katrin Renz for supervising the project and providing me with the necessary guidance and valuable support. I want to thank Prof. Andreas Geiger for the helpful discussions. Finally, I want to express my very profound gratitude to my parents for their continuous support and encouragement throughout my entire life.

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.

Deep policy learning makes promising progress to many visuomotor control tasks ranging from robotic manipula- tion [20, 22, 25, 39] to autonomous driving [4, 47]. By learning to map visual observation directly to control action through a deep neural network, it mitigates the manual design of controller, lowers the system complexity, and improves generalization ability. However, the sample ef- ficiency of the underlying algorithms such as reinforcement learning or imitation learning remains low. It requires a significant amount of online interactions or expert demon- strations in the training environment thus limits its real- world applications. Many recent works use unsupervised learning and data augmentation to improve the sample efficiency by pre- training the neural representations before policy learning. However, the augmented data in pretraining such as frames with random background videos [16, 17, 43] shifts drasti-cally from the original data distribution, which degrades the overall performance of the model. Also, it remains challenging to generalize the learned weights to the real- world environment as it is hard to design augmentations that reflect the real-world diversity. In this work, we explore pretraining the neural representation on a massive amount of real-world data directly. Figure 1 shows some uncurated YouTube videos, which contain driving scenes all over the world with diverse conditions such as different weathers, urban and rural environments, and various traffic densities. We show that exploiting such real-world data in deep policy learning can substantially improve the generalization ability of the pretrained models and benefit downstream tasks across various domains.

Self-supervised learning (SSL) is an approach of machine learning to learn from unlabeled data. In this learning approach, the learning happens in two steps. First, the task is solved based on pseudo-labels which helps to initialize the network weights. Second, the network

is finetuned with ground truth data. Self-supervised learning (SSL) is a method of machine learning. It learns from unlabeled sample data. It can be regarded as an intermediate form between supervised and unsupervised learning. It is based on an artificial neural network.[1] The neural network learns in two steps. First, the task is solved based on pseudo-labels which help to initialize the network weights.[2][3] Second, the actual task is performed with supervised or unsupervised learning.[4][5][6] Self-supervised learning has produced promising results in recent years and has found practical application in audio processing and is being used by Facebook and others for speech recognition.[7] The primary appeal of SSL is that training can occur with data of lower quality, rather than improving ultimate outcomes. Self-supervised learning more closely imitates the way humans learn to classify objects.[8]

Learning representations from unlabeled data is a popular topic in visual recognition. The learned representations are shown to be generalizable across visual tasks ranging from image classification, semantic segmentation, to object detection [5, 13, 18, 40, 42]. However, these methods are primarily built for learning features for recognition tasks rather than control, where an agent acts in an uncertain environment. Decision-making may not benefit from some visual information, such as the abstraction of appearance and texture. For instance, visual factors like lighting and weather may even become confounders in policy learning, decreasing overall performance [46]. These factors are typically unimportant to the driving job. The properties that are relevant to the output action, on the other hand, must be understood. On the contrary, it is crucial to learn the features that matter to the output action. For example, at the driving junction, the traffic light occupies only a few pixels in the visual observation but has a significant impact on the driver's actions.

In this work, we explore an existing self-supervised learning approach catered to our autonomous driving framework. Moreover, we propose a new self-supervised learning method that outperforms the prior work on the challenging NEAT validation routes [7]. Also, we propose a data cleansing technique for and stratified sampling in the dataloader for effective training that improves the driving performance tremendously.

In summary, the main contributions of this work are:

- We present a novel self-supervised learning approach for autonomous driving.
- +We propose a new data cleaning pipeline and stratified sampling in training.
- We show how the *inertia problem* can be addressed by data cleaning strategy.

We organize the structure of the thesis as follows. We first provide an overview of the learning-based pipeline in SAP and its limitations in Section 2. We then introduce details of our methodology in Section 3, followed by the description of the baseline methods and the effectiveness of our proposed model compared to the baselines both quantitatively and qualitatively in Section 5.

2 Related Work

2.1 Self-supervised Learning

2.1.1 Emerging Properties in Self-Supervised Vision Transformers

2.1.2 Exploring Simple Siamese Representation Learning

2.1.3 Momentum Contrast for Unsupervised Visual Representation Learning

2.1.4 Self-training with Noisy Student improves ImageNet classification

2.2 Self-supervised Learning in Autonomous Driving

2.2.1 SelfD: Self-Learning Large-Scale Driving Policies From the Web

2.2.2 Offline Visual Representation Learning for Embodied Navigation

2.2.3 Action-Conditioned Contrastive Policy Pretraining

3 Problem Setting

Our goal is to facilitate training driving policies at scale.

3.1 Problem Setting

We consider the task of point-to-point navigation in an urban setting where the goal is to complete a given route while safely reacting to other dynamic agents and following traffic rules. To achieve this, we consider the imitation learning approach of learning policy as in self-driving it is easier for an expert to demonstrate the desired behaviour rather than to specify a reward function.

Imitation Learning (IL): The goal of IL is, for an agent to learn a policy π_{θ}, that imitates the behavior of an expert π^{*}. The agent learns to map an input to a navigational decision. In general, the decision may either be a low-level vehicle control action [10] (e.g. steering, throttle and break) or a desired future trajectory relative to the ego-vehicle, i.e., a set of K waypoints [6, 17] in the BEV space. In the latter case, future waypoints may be paired with a hand-specified or learned motion controller to produce the low-level action [6, 17]. In this work, we focus on the later representation due to its interpretability and genralizability. To find the mapping, we consider the Behavior Cloning (BC) approach of IL. An expert policy is first rolled out to collect at each time-step, high-dimensional observations of the environment including front camera image, ego-vehicle position and orientation, high-level navigational command and high-level goal location provided as GPS coordinates etc. From these high-dimensional observations, we derive our dataset $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{w}_i)\}_{i=1}^{N} \in (\mathcal{X}, \mathcal{W})$ of size N, where the input $\mathbf{x}$ is a subset of the observations (see ?) and the corresponding expert trajectory

$\mathbf{w}$, defined by a set of 2D waypoints relative to the coordinate frame of the ego-vehicle in BEV space, i.e., $\mathbf{w} = (x_t, y_t)_{t=1}^{T}$, are calculated from the ego-vehicle positions and orientations from the subsequent frames. Our goal is to find a decision making policy i.e. a waypoint prediction function $\pi_{\boldsymbol{\theta}} : \mathcal{X} \rightarrow \mathcal{W}$ with learnable parameters $\boldsymbol{\theta} \in \mathbb{R}^d$. In BC, the policy $\pi_{\boldsymbol{\theta}}$ is learned by training a neural network in a supervised manner using the dataset, $\mathcal{D}$, with a loss function, $\mathcal{L}$ i.e.

$$\underset{\theta}{\operatorname{argmin}}\ \mathbb{E}_{(\mathbf{x},\mathbf{w})\sim\mathcal{D}}[\mathcal{L}(\mathbf{w}, \pi_\theta(\mathbf{x}))].$$

We use the L_1 distance between the predicted trajectory, $\pi_{\boldsymbol{\theta}}(\mathbf{x})$, and the corresponding expert trajectory, $\mathbf{w}$, as the loss function. We assume access to an inverse dynamic model [3], implemented as a PID controller $\mathbb{I}$, which performs the low-level control, i.e., steer, throttle and brake, provided the future trajectory $\mathbf{w}$. The action are dertermined as $\mathbf{a} = \mathbb{I}(\mathbf{w})$.

Global Planner: We follow the standard protocol of CARLA 0.9.10 and assume that high-level goal locations G are pro- vided as GPS coordinates. Note that these goal locations are sparse and can be hundreds of meters apart, as opposed to the local waypoints predicted by the policy $\pi_{\boldsymbol{\theta}}$. For one of the self-supervised learning methods, we will slightly augment the output space of this function in Section 3.3.

3.2 Input and Output Parameterization

We note that, even though our collected expert demostrations and the youtube driving videos are sequential, we do not use temporal data for training. Contrary to our intuition of getting better generalization in decision-making from sequential observations, the prior works on IL for autonomous driving have shown that using observation histories may not lead to performance gain [15, 16, 2, 18]. Thus, We use a single time-step input.

Input Representation: Following [45, 23], we convert the LiDAR point cloud into a 2-bin histogram over a 2D BEV grid with a fixed resolution. We consider the points within 32m in front of the ego-vehicle and 16m to each of the sides, thereby encompassing a BEV grid of 32m × 32m. We di- vide the grid into blocks of 0.125m × 0.125m which results in a resolution of 256 × 256 pixels. For the histogram, we discretize the height dimension into 2 bins representing the points on/below and above the ground plane. This results in a two-channel pseudo-image of size 256 × 256 pixels. For the RGB input, we consider the front camera with a FOV of 100◦ . We extract the front image at a resolution of 400 × 300 pixels which we crop to 256 × 256 to remove radial distortion at the edges. Output Representation: We predict the future trajectory W of the ego-vehicle in BEV space, centered at the current coordinate frame of the ego-vehicle. The trajectory is repre- sented by a sequence of 2D waypoints, {wt = (xt , yt)}Tt=1 . We use T = 4, which is the default number of waypoints required by our inverse dynamics model.

3.3 Waypoint Prediction Network

As shown in Fig. 2, we pass the 512-dimensional feature vector through an MLP (comprising 2 hidden layers with 256 and 128 units) to reduce its dimensionality to 64 for computational efficiency before passing it to the auto-regressive waypoint network implemented using GRUs [8]. We initialize the hidden state of the GRU with the 64-dimensional feature vector. The update gate of the GRU controls the flow of information encoded in the hidden state to the output and the next time-step. It also takes in the current position and the goal location (Section 3) as input, which allows the network to focus on the relevant context in the hidden state for predicting the next waypoint. We provide the GPS coordinates of the goal location (transformed to the ego-vehicle coordinate frame) as input to the GRU rather than the encoder since it lies in the same BEV space as the predicted waypoints and correlates better with them compared to representing the goal location in the perspective image domain [6]. Following [12], we use a single layer GRU followed by a linear layer which takes in the hid- den state and predicts the differential ego-vehicle waypoints $\{\delta\mathbf{w}_t\}_{t=1}^{T}$ for $T = 4$ future time-steps in the ego-vehicle current coordinate frame. Therefore, the predicted future waypoints are given by $\{\mathbf{w}_t = \mathbf{w}_{t-1} + \delta\mathbf{w}_t\}_{t=1}^{T}$. The input to the first GRU unit is given as $(0, 0)$ since the BEV space is centered at the ego-vehicle's position.

Controller: We use two PID controllers for lateral and lon- gitudinal control to obtain steer, throttle and brake values from the predicted waypoints, $\{\mathbf{w}_t\}_{t=1}^{T}$. The longitudinal controller takes in the magnitude of a weighted average of the vectors between waypoints of consecutive time-steps whereas the lateral controller takes in their orientation. For the PID controllers, we use the same configuration as in the author-provided codebase of [6]. Implementation de- tails can be found in the supplementary.

3.4 Loss Function

Following [8], we train the network using an L1 loss be- tween the predicted waypoints and the ground truth way- points (from the expert), registered to the current coordi- nate frame. Let wtgt represent the ground truth waypoint for time-step t, then the loss function is given by: L= T X

$$\text{wt} - \text{wtgt} \quad 1$$

(5) t=1 Note that the ground truth waypoints {wtgt } which are avail- able only at training time are different from the sparse goal locations G provided at both training and test time.

3.5 Task

We consider the task of navigation along a set of predefined routes in a variety of areas, e.g. freeways, urban areas, and residential districts. The routes are defined by a sequence of sparse

goal locations in GPS coordinates pro- vided by a global planner and the corresponding discrete navigational commands, e.g. follow lane, turn left/right, change lane. Our approach uses only the sparse GPS lo- cations to drive. Each route consists of several scenarios, initialized at predefined positions, which test the ability of the agent to handle different kinds of adversarial situations, e.g. obstacle avoidance, unprotected turns at intersections, vehicles running red lights, and pedestrians emerging from occluded regions to cross the road at random locations. The agent needs to complete the route within a specified time limit while following traffic regulations and coping with high densities of dynamic agents.

We consider the task of driving in an urban setting in the realistic 3D simulator CARLA version 0.9.13. 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 Self-supervised Learning Approaches

In this section we will introduce our proposed method and also explore some prior works in our self-driving framework.

4.1 Learning with Pseudolabels

Contrary to SelfD approach, we do not need to rely on the model's understanding of driving for pseudo-labels

4.1.1 Deep Visual Odometry

In this section, we will introduce our proposed method for self-supervised learning for autonomous driving. Our key idea is to generate pseudo-labels for unlabeled driving scenes by exploiting a learning-based ego-motion estimation method because of its desirable properties of robustness to image noise and camera calibration independence. Moreover, in our expert driving dataset the weather conditions and the time of the day changes from frame to frame. For this reason learning-based method is the best way to estimate ego-motions than classical methods. We use an end-to-end learning approach following [19, 22] to train the model to map directly from input image pairs to an estimate of ego-motion (in our use case, estimate of relative translation is sufficient). The model we used, is a two module Long-term Recurrent Convolutional Neural Networks.

Feature-encoding Module: In order to learn the geometric relationships from two adjacent images, we use the following CNN architechture, inspired by FlowNetSimple architechture [13] ignoring the decoder part of it and only focusing the on the convolutional encoder.

Layer	Kernel Size	Padding	Stride	Max Pool	Number of Channels
Input	-	-	-	-	6
Conv1	3×3	1	0	2×2	64
Conv2	3×3	1	0	2×2	128
Conv3	3×3	1	0	4×4	256
Conv4	3×3	1	0	4×4	512
Conv5	3×3	1	0	4×4	1024

Table 1: Configuration of the CNN

Contrary to DeepVO [19] and PoseConvGRU [22], which uses huge 10-layered CNN architechtures, we use a very simple and lightweight 5-layered CNN architechture as shown in Table 1. Each layer is followed by an application of ReLU nonlinear activation function. We keep the kernel size to 3, padding size to 1 for all the layers. The channel dimension doubles in each subsequent layers. We use maxpool of size 2 in the first 2 layers and use maxpool of size 4 for the rest of the layers. The reason behind this is that, having small receptive field in the first 2 layers encourages the network to learn about the fine-grained geometric details in the pair of images which is essential for relative motion estimation. On the other hand, the large receptive field in the later layers enforces the network to ignore the global context and also helps in reducing the number of learnable parameters as well. The input to the CNN are a sequence of $n+1$, 256×256 RGB driving scenes from CARLA. With $n+1$ sequential driving scenes, we can obtain n sets of image pairs taking two adjacent frames at a time. These image pairs are then fed to the 5-layered CNN to obtain a feature maps of size $1 \times 1 \times 1024$ for each image pair. Contrary to the typical training process with augmented data for CNNs, we only use the original images for accurate relative motion estimation, because performing any pre-processing operation to the images such as blurring, adding noise, random clipping etc., can hamper the network to learn the geometric relationship of the objects in the images.

Memory-propagating Module: Following [22], we use a stacked ConvGRU (convolutional gated recurrent unit) [1] as our memory-propagating module as showns in Fig.. The memory module builds a set of chronological visual representations from the CNN embeddedings of the sequence of image pairs. Because of its ability of remember histories, ConvGRU can capture the geometric relationships coming from the previous frames of images, and then estimate the relative motion for the current frame utilizing the geometric constraint within multiple frames. Also, ConvGRUs are appropriate for this module as they are simple yet powerful. They contain fewer gates compared to ConvLSTMs and thus reduce the number of learnable parameters and also provide similar performance as ConvLSTMs [9]. In our implementation, we flatten the $1 \times 1 \times 1024$-dimensional CNN embeddeding into a feature vector, which we then further process through a fully connected layer and reduce its dimensions to 256 for computational efficiency before passing it to the ConvGRU. Following [12], we use a single layer ConvGRU followed by a linear layer which takes in the hidden state and predicts the relative translation of the ego-vehicle implied by the pair of images. Finally, we accumulate all the relative translations to compute the ego-vehicle trajectory.

should include the math? plot pseudowaypoints

The feature-encoding module encodes the short-term motion feature in an image pair, while the memory- propagating module captures the long-term motion feature in the consecutive image pairs. The visual memory is implemented with convolutional gated re- current units, which allows propagating information over time. At each time step, two consecutive RGB images are stacked together to form a 6 channels tensor for module-1 to learn how to extract motion information and estimate poses. The sequence of output maps is then passed through

a stacked ConvGRU module to generate the relative transformation pose of each image pair. We also augment the training data by randomly skipping frames to simulate the veloc- ity variation which results in a better performance in turning and high-velocity situations. Randomly horizontal flipping and temporal flipping of the sequences is also performed. We evaluate the performance of our proposed approach on the KITTI Visual Odometry benchmark. The experiments show a competitive performance of the proposed method to the geometric method and encourage further exploration of learning based methods for the purpose of estimating camera ego-motion even though geometrical methods demonstrate promising results.

Self-supervised learning aims at gathering more labels to unlabeled inputs so that we can generate better representation of the input. In self-driving task our output is the waypoints. Which can be estimated completely independent of the driving task and we can generate accurate estimation of the waypoints.

There are two ways to generate better representations of the inputs

1. Contrastive learning This kind of learning try to learn a representation that describe the image as a whole. Does not find the representation that is actually important to the downstream task.
2. by generating pseudolabels

it also predicts accurate ego-motion estimation,

4.1.2 SelfD

In this section we will discuss the semi-supervised approach "SelfD" by Zhang et al. [23].

4.1.2.1 Conditional Imitation Learning from Observations

To make use of the unlabeled driving scenes containing diverse navigational experience, the authors propose 'Conditional Imitation Learning from Observations' (CILfO) [23] framework. The framework suggests a way to generate pseudo-labels for the unlabeled driving scenes. We explore the framework in our autonomous driving setting for self-supervised learning. We consider two conditional commands i.e. navigational command and target point separately in our experiments see?. Thus, with this method we recover the waypoints $\hat{\mathbf{w}}$, the conditional command $\hat{c}$ and speed $\hat{v}$ to construct a dataset

$$\hat{\mathcal{D}} = \{(\mathbf{I}_i, \hat{v}_i, \hat{c}_i), \mathbf{w}_i\}_{i=1}^{M}$$

which can be then use to train a policy using behaviour cloning.

4.1.2.2 Initial Data Assumption

The CILfO learning task assumes access to a small labeled dataset to learn an initial policy mapping using human expert demonstrations. We then use this learned policy to gather

pseduo-labels for the unlabeled data. This assumption is reasonable considering that there are several publicly available driving datasets with included action labels [14, 20]. In this work, we use a small subset of a labeled dataset collected from an expert policy roll out in CARLA [11].

4.1.2.3 Self-supervised Training Process

In this section, we discuss the proposed generalized training method for leveraging unconstrained and unlabeled demonstration data. The proposed semi-supervised policy training process, SelfD, can be learned in three summarized steps:

1. Use a small, labeled domain-specific dataset $\mathcal{D}$ to learn an initial observations-to-BEV policy $\pi_{\boldsymbol{\theta}}$ via imitation.
2. Obtain a large pseudo-labeled dataset $\hat{\mathcal{D}}$ by leveraging sampling from $\pi_{\boldsymbol{\theta}}$.
3. Pre-train a generalized policy $\pi_{\boldsymbol{\theta}}$ on $\hat{\mathcal{D}}$ and fine-tune on the clean labels of $\mathcal{D}$.

Note that we re-use the parameter symbol $\boldsymbol{\theta}$ throughout the steps to simplify notation. Our iterative semi-supervised training enables effectively augmenting the knowledge and robustness of an initially trained policy. As described next, our proposed initial step facilitates learning a platform and perspective-agnostic policy during subsequent training steps by directly reasoning in a BEV planning space.

3.3. BEV Plan Network In this section, we propose a suitable output represen- tation to account for arbitrary cameras, viewpoints and scene layouts. Current monocular planners generally pre- dict waypoints in the image plane to align with an input image [13, 49]. The waypoints are then transformed to a BEV plan using carefully calibrated camera intrinsic and extrinsic (e.g., rotation, height) parameters [13]. Thus, pol- icy models are often trained and evaluated within a fixed pre-assumed setup. In contrast, SelfD predicts a future plan parameterized by waypoints in the BEV plan space directly. Based on our experiments in Sec. 4, we demonstrate this choice to be crucial for real-world planning across settings. The predicted generalized BEV waypoints can be paired with a low-level controller, e.g., a PID controller [13, 49]. Due to the difficulty in learning a monocular-to-BEV plan mapping, we follow recent work in confidence-aware learn- ing [38] to train an augmented model fθ : X → Y × R with quality estimates σ □ R. Our training loss function in Eqn. 1 is defined as L = Lplan + λLquality (3) where Lplan is the L1 distance between ground-truth and predicted waypoints, Lquality is a binary cross-entropy loss [38, 78], and the λ hyper-parameter balances the tasks. 3.4. “What If” Pseudo-Labeling of Unlabeled Data Given a set of unlabeled images U, we sample from the trained conditional policy fθ in a semi-supervised training process. While the speed and command inputs to fθ can be recovered through visual odometry techniques [73], these result in highly noisy trajectories in our online video set- tings (discussed in Sec. 4). As the demonstrations in our data may be unsafe or difficult to recover, we propose to leverage a single-frame pseudo-labeling mechanism. Our key insight is to employ the conditional model fθ to gener- ate multiple hypothetical future trajectories in a process

re- ferred to as “what if” augmentation. Beyond resolving the missing speed and command inputs, our proposed augmen- tation provides additional supervision, i.e., a conditional agent that better reasons on what it might need to do, for in- stance, if it had to turn left instead of right at an intersection (Fig. 2). In contrast to related work in policy learning and distilla- tion [13], sampling from the teacher agent is more challenging as the agent is not exposed to extensive 3D per- ception knowledge about the world and is being evaluated outside of its training settings. We repeatedly sample $\hat{v}$ and $\hat{c}$ uniformly and rely on the conditional model to provide pseudo-labels $(\hat{y}, \hat{\sigma}) = f\theta\ (I, \hat{v}, \hat{c})$ for additional supervision across all conditional branches and speed observations. In this manner, querying the “teacher” model $f\theta$ enables us to generate various sce- narios beyond the original demonstration. In particular, as discussed in Sec. 4, we find self-training strategies to pro- vide limited generalization gains without this “what if” data augmentation step. This augmentation strategy enables our single-frame pseudo-labeling approach to significantly out- perform approaches that are more elaborate to train at scale, as they may involve additional modules relying on approxi- mating $\hat{y}$, $\hat{c}$, and $\hat{v}$ from video. Finally, to avoid incorporat- ing potentially noisy trajectories, the corresponding quality estimates $\hat{\sigma}$ can be used to process and filter examples in the pseudo-labeled dataset $\hat{D}$. 3.5. Model Pre-Training and Fine-Tuning As a final training step, we re-train the waypoint network $f\theta$ from scratch over the large and diverse dataset $\hat{D}$. The pre-trained policy can then be further fine-tuned over the original dataset D, thus leveraging the additional knowledge gained from $\hat{D}$ to improve its performance. We note that we employ separate training over the two datasets $\hat{D}$ and D and rely on knowledge transfer through learned represen- tations as it reduces the need for any careful hyperparam- eter tuning beyond the overall learning rate. For instance, Caine et al. [8] empirically demonstrate the importance of delicately op- timizing the ratio of labeled to pseudo-labeled data when mixing the datasets for a 3D object detection task, while also showing it to vary among object categories, e.g., pedestrians vs. vehicles. Thus, through a pre-training mechanism, we avoid the need to carefully mix the cleanly labeled and pseudo-labeled datasets [8, 65, 78]. 3.6. Implementation Details Speed: 0.24 Speed: 16.21 Speed: 2.84 Score: 0.6 Score: 0.09 Score: 0.42 Cmd: 1 Cmd: 2 Cmd: 3 We implement our BEV waypoint prediction network $f\theta$ leveraging a state-of-the-art condi- tional sensorimotor agent [13]. However, as discussed in Sec. 3.2, we do not assume a fixed known BEV perspective transform. Thus, we remove the fixed perspective transformation layer which restricts scalability and replace it with a per-branch BEV prediction module (see supplementary for additional imple- mentation and architecture details). During training we use a learning rate of $1e{-}3$ and resize images to 400×225.

4.2 Self-supervised Representation Learning

4.2.1 Action Conditioned Policy Pretraining

4.2.2 OVRL

In this section, we discuss OVRL[21], a two-stage learning approach proposed by Yadav et. al. and incorporate it in our autonomous driving framework. As an overview, this learning approach includes an encoder pretraining step using DINO, followed by downstream policy learning via behavior cloning.

4.2.2.1 Self-supervised Pretraining

As the first step, a visual encoder is pretrained using DINO [4], a simple but effective self-supervised learning algorithm. DINO uses knowledge distillation as a mechanism for self-training, where the student network $g_{\boldsymbol{\theta}_s}$ is trained to match the output of the teacher network $g_{\boldsymbol{\theta}_t}$. As illustrated in Figure 2(left), it takes an input image x and using the multi-crop strategy introduced in [5], it generates multiple distorted views or crops from it, specifically, it produces two global views (x_1^g and x_2^g) at 224×224 resolution and eight local views x^l at a lower resolution (96×96). All crops are passed through the student network while only the global views are passed through the teacher network. The student and teacher networks both output K dimensional feature vectors for each view, which are converted into probability distributions (P_s and P_t) using a temperature scaled softmax function as follows:

$$P_s(x)^{(i)} = \frac{\exp\left(g_{\boldsymbol{\theta}_s}(x)^{(i)}/\tau_s\right)}{\sum_{k=1}^{K} \exp\left(g_{\boldsymbol{\theta}_s}(x)^{(k)}/\tau_s\right)}$$

where the temperature parameter τ_s controls the sharpness of the output distribution, and a similar formula holds for P_t with temperature τ_t. Now, given a fixed teacher network $g_{\boldsymbol{\theta}_t}$, the student network learns to match the distribution of the teacher network by minimizing the cross-entropy loss:

$$\mathcal{L}(\boldsymbol{\theta}_s) = \sum_{x \in x_1^g, x_2^g} \sum_{x' \in \{x_1^g, x_2^g\} \cup \{x_i^l\}_{i=1}^{8}} P_t(x) \log(P_s(x')).$$

The teacher network parameters are updated as an exponential moving average of the student network parameters.

Representation learning by self-supervised learning algorithm is always prone to collapse, i.e., the network can converge to a trivial solution by predicting the same representation for every image. To avoid collapse, DINO centers and sharpens the teacher's output before the softmax operation. Specifically, the centering operation adds the term c to the teacher's output, which is updated as follows: $c \leftarrow mc + (1-m)\frac{1}{B}\sum_{i=0}^{B} g_{\boldsymbol{\theta}_t}(x_i)$, where $m > 0$ is a momentum parameter and B is the batch size. Sharpening is achieved by setting low value

for the temperature parameter τ_t for the teacher softmax normalization. Thus, centering prevents one dimension to dominate but encourages collapse to the uniform distribution, while the sharpening has the opposite effect. Applying both operations balances their effects which is sufficient to avoid collapse in presence of a momentum teacher.

We use the convolutional layers in a modified ResNet50 [21] architecture com- bined with a projection head for the student and teacher networks. Specifically, we modify the ResNet50 by 1) reducing the number of output channels at every layer by half (i.e. using 32 ResNet baseplanes instead of 64) and 2) using Group- Norm [48] instead of BatchNorm in our backbone, similar to DDPPO [47]. The projection head is a 3-layer MLP (Multi-Layer Perceptron) with 2048 hidden units followed by l2 norm and a weight normalized fully connected layer of K dimension. We keep the BatchNorm in the MLP, however the whole projection head is discarded after training.

4.2.2.2 Downstream Learning

We illustrate DINO in the case of one single pair of views (x1 , x2) for simplicity. The model passes two different random transformations of an input image to the student and teacher networks. Both networks have the same architecture but different parameters. The output of the teacher network is centered with a mean computed over the batch. Each networks outputs a K dimensional feature that is normalized with a temperature softmax over the feature dimension. Their similarity is then measured with a cross-entropy loss. We apply a stop-gradient (sg) operator on the teacher to propagate gradients only through the student. The teacher parameters are updated with an exponential moving average (ema) of the student parameters.

5 Experimental Results

In this section, we describe our experimental setup, com- pare the driving performance of our approach against sev- eral baselines, conduct an infraction analysis to study dif- ferent failure cases, visualize the attention maps of Trans- Fuser and present an ablation study to highlight the impor- tance of different components of our model.

5.1 Implementation Details

5.1.1 Input and Output Parameterization

Input Representation: Following [45, 23], we convert the LiDAR point cloud into a 2-bin histogram over a 2D BEV grid with a fixed resolution. We consider the points within 32m in front of the ego-vehicle and 16m to each of the sides, thereby encompassing a BEV grid of 32m × 32m. We di- vide the grid into blocks of 0.125m × 0.125m which results in a resolution of 256 × 256 pixels. For the histogram, we discretize the height dimension into

2 bins representing the points on/below and above the ground plane. This results in a two-channel pseudo-image of size 256 × 256 pixels. For the RGB input, we consider the front camera with a FOV of 100◦ . We extract the front image at a resolution of 400 × 300 pixels which we crop to 256 × 256 to remove radial distortion at the edges. Output Representation: We predict the future trajectory W of the ego-vehicle in BEV space, centered at the current coordinate frame of the ego-vehicle. The trajectory is repre- sented by a sequence of 2D waypoints, {wt = (xt , yt)}Tt=1 . We use T = 4, which is the default number of waypoints required by our inverse dynamics model.

5.1.2 Dataset

5.1.3 Data Preprocessing Pipeline

5.1.4 Training and Inference

5.2 Dataset

We use the CARLA [11] simulator for training and testing, specifically CARLA 0.9.10 which consists of 8 publicly available towns. We use 7 towns for training and hold out Town05 for evaluation. For generating training data, we roll out an expert policy designed to drive using privileged information from the simulation and store data at 2 FPS. Please refer to the supplementary material for additional details. We select Town05 for evaluation due to the large diversity in drivable regions compared to other CARLA towns, e.g. multi-lane and single-lane roads, highways and exits, bridges and underpasses. We consider two evaluation settings: (1) Town05 Short: 10 short routes of 100-500m comprising 3 intersections each, (2) Town05 Long: 10 long routes of 1000-2000m comprising 10 intersections each. Each route consists of a high density of dynamic agents and adversarial scenarios which are spawned at predefined positions along the route. Since we focus on handling dynamic agents and adversarial scenarios, we decouple this aspect from generalization across weather conditions and evaluate only on ClearNoon weather.

5.3 Evaluation Metrics

For the CARLA Autonomous Driving Leaderboard, the driving performance of an agent is characterized by a set of chosen metrics that considers different aspects of driving. While all routes have the same type of metrics, their respective values are calculated separately. These metrics are as follows:

5.3.1 Route Completion

Route Completion is the percentage of the route that is completed by an agent. If an agent drives off-road, that percentage of the route will not be considered towards the computation of

the route completion score. Additionally, the following events will interrupt the simulation, preventing the agent to continue which will effectively reduce the route completion:

- Route deviation: If an agent deviates more than 30 meters from the assigned route.
- Agent blocked: If an agent doesn't take any actions for 180 simulation seconds.
- Simulation timeout: If no client-server communication can be established in 60 seconds.
- Route timeout: If the simulation of a route takes too long to finish.

5.3.2 Infraction Score

Infraction Score is a penalty for infractions where the agent starts with an ideal base score of 1.0. For every infraction, the score is multiplied by the penalty coefficient of that infraction type. Ordered by their severity, the penalty coefficients are as follows:

- 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.

5.3.3 Driving Score

Driving Score is the main metric for performance, serving as the product between the route completion and the infractions penalty. It is calculated in the following way:

$$\text{Driving Score} = \frac{1}{N}\sum_{i=1}^{N} R_i P_i$$

where N is the number of routes, R_i the route completion percentage of the i-th route and P_i the infraction penalty of i-th route. 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.

5.4 Comparisons to the Baselines Methods

5.5 Ablation Studies

6 Conclusion

We envision broad and easily deployable autonomous navigation systems. However, access to resources and data limits the scope of the brittle autonomous systems today. Our SelfD approaches enables to significantly improve an initially trained policy without incurring additional data col- lection or annotation efforts, i.e., for a new platform, per- spective, use-case, or ambient settings. Crucially, due to the proposed underlying model architecture, we do not in- corporate camera parameters or configuration assumptions into the monocular inference. As SelfD is self-improving, a future direction could be to continue and learn from increasingly larger online datasets beyond what is described in our study. While we emphasized efficient large-scale training in our model development, how to best extend SelfD to more explicitly leverage temporal demonstration data is still an open question which could be studied further in the future. Finally, beyond complex 3D navigation, it would be inter- esting to explore the applicability of our proposed training framework for learning various embodied tasks from unla- beled web data.

In our experiments we have only used front camera for navigational decesion. Even though, it can successful in perfect route completion, it sometimes results in infractions due to invisibilty for example, ego-vehicle changes lane but the rear vehicle collide with it or the passenger starts walking after the car already have gone past it.

References

[1] Nicolas Ballas et al. "Delving Deeper into Convolutional Networks for Learning Video Representations". In: (Nov. 2015). arXiv: 1511.06432 [cs.CV].

[2] Mayank Bansal, Alex Krizhevsky, and Abhijit Ogale. "ChauffeurNet: Learning to Drive by Imitating the Best and Synthesizing the Worst". In: (Dec. 2018). arXiv: 1812.03079 [cs.RO].

[3] RICHARD BELLMAN. *Adaptive Control Processes: A Guided Tour*. Princeton University Press, 1961. ISBN: 9780691079011. URL: http://www.jstor.org/stable/j.ctt183ph6v (visited on 09/10/2022).

[4] Mathilde Caron et al. "Emerging Properties in Self-Supervised Vision Transformers". In: (Apr. 2021). arXiv: 2104.14294 [cs.CV].

[5] Mathilde Caron et al. "Unsupervised Learning of Visual Features by Contrasting Cluster Assignments". In: (June 2020). arXiv: 2006.09882 [cs.CV].

[6] Dian Chen et al. "Learning by Cheating". In: (Dec. 2019). arXiv: 1912.12294 [cs.RO].

[7] Kashyap Chitta, Aditya Prakash, and Andreas Geiger. “NEAT: Neural Attention Fields for End-to-End Autonomous Driving”. In: (Sept. 2021). arXiv: 2109.04456 [cs.CV].

[8] Kyunghyun Cho et al. “Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation”. In: (June 2014). arXiv: 1406.1078 [cs.CL].

[9] Junyoung Chung et al. “Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling”. In: (Dec. 2014). arXiv: 1412.3555 [cs.NE].

[10] Felipe Codevilla et al. “Exploring the Limitations of Behavior Cloning for Autonomous Driving”. In: (Apr. 2019). arXiv: 1904.08980 [cs.CV].

[11] Alexey Dosovitskiy et al. “CARLA: An Open Urban Driving Simulator”. In: (Nov. 2017). arXiv: 1711.03938 [cs.LG].

[12] Angelos Filos et al. “Can Autonomous Vehicles Identify, Recover From, and Adapt to Distribution Shifts?” In: (June 2020). arXiv: 2006.14911 [cs.LG].

[13] Philipp Fischer et al. “FlowNet: Learning Optical Flow with Convolutional Networks”. In: (Apr. 2015). arXiv: 1504.06852 [cs.CV].

[14] Andreas Geiger, Philip Lenz, and Raquel Urtasun. “Are we ready for autonomous driving? The KITTI vision benchmark suite”. In: *2012 IEEE Conference on Computer Vision and Pattern Recognition* (2012), pp. 3354–3361.

[15] Pim de Haan, Dinesh Jayaraman, and Sergey Levine. “Causal Confusion in Imitation Learning”. In: (May 2019). arXiv: 1905.11979 [cs.LG].

[16] Urs Muller et al. “Off-road obstacle avoidance through end-to-end learning”. In: *NeurIPS*. 2006.

[17] Matthias Müller et al. “Driving Policy Transfer via Modularity and Abstraction”. In: (Apr. 2018). arXiv: 1804.09364 [cs.RO].

[18] Dequan Wang et al. “Monocular plan view networks for autonomous driving”. In: *IROS*. 2019.

[19] Sen Wang et al. “DeepVO: Towards End-to-End Visual Odometry with Deep Recurrent Convolutional Neural Networks”. In: (Sept. 2017). DOI: 10.1109/ICRA.2017.7989236. arXiv: 1709.08429 [cs.CV].

[20] Benjamin Wilson et al. “Argoverse 2: Next Generation Datasets for Self-driving Perception and Forecasting”. In: *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks (NeurIPS Datasets and Benchmarks 2021)*. 2021.

[21] Karmesh Yadav et al. “Offline Visual Representation Learning for Embodied Navigation”. In: (Apr. 2022). arXiv: 2204.13226 [cs.CV].

[22] Guangyao Zhai et al. “PoseConvGRU: A Monocular Approach for Visual Ego-motion Estimation by Learning”. In: (June 2019). arXiv: 1906.08095 [cs.CV].

[23] Jimuyang Zhang, Ruizhao Zhu, and Eshed Ohn-Bar. “SelfD: Self-Learning Large-Scale Driving Policies From the Web”. In: (Apr. 2022). arXiv: 2204.10320 [cs.CV].