The Thesis Committee for Jierui Lin
Certifies that this is the approved version of the following Thesis:

PointDrive: A Point-based Self-driving Policy

APPROVED BY
SUPERVISING COMMITTEE:

Philipp Krähenbühl, Supervisor

Yuke Zhu

PointDrive: A Point-based Self-driving Policy

by

Jierui Lin

Thesis

Presented to the Faculty of the Graduate School

of The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Computer Sciences

The University of Texas at Austin

May 2022

Acknowledgments

I would like to express my gratitude to my supervisor, Philipp Krähenbühl, who guided me throughout this project. I would also like to thank Yuke Zhu who offered deep insight into the study.

I would also like to thank all the lab mates in Deep Learning Lab (UT-DL) and Robot Perception and Learning Lab (RPL) for the great time we spent together.

Last but not least, I would like to thank my parents for supporting me over the years and continuing to support me for my future endeavor.

Abstract

PointDrive: A Point-based Self-driving Policy

Jierui Lin, MSCS

The University of Texas at Austin, 2022

Supervisor: Philipp Krähenbühl

Vision-based urban driving is hard since the image representation learned solely from action supervision cannot generalize well to new scenarios. Is there a better representation of the traffic scenarios? We propose PointDrive, a simple driving policy conditioned on feature vectors of nearby agents. With semantic points representation, we can perform realistic and efficient data augmentations to alleviate distributional shift. Also, this point representation is easy to get from existing computer vision algorithms. Experiments show that PointDrive substantially outperforms other offline methods on the CARLA benchmark. Furthermore, we perform extensive ablation studies and show that the current bottleneck of our system is in the perception module, suggesting better model design or learning algorithm is needed to produce a robust vision model for autonomous driving.

Table of Contents

Chapter 1 Introduction **7**
1.1 Introduction 7

Chapter 2 Related Work **9**
2.1 Object detection and tracking 9
2.2 Road and lane detection 9
2.3 Imitation Learning 10
2.4 End-to-End vs. Modular approach in policy learning 10

Chapter 3 Preliminaries **12**
3.1 Partially Observed Markov Decision Process 12
3.2 Behavioral Cloning 12

Chapter 4 System Design **13**
4.1 State representation 13
4.2 Perception Module 14
4.3 Policy Module 17

Chapter 5 Experiments **19**
5.1 Environment 19
5.1.1 CARLA 19
5.1.2 Expert data collection 19
5.2 Implementation Details 20
5.2.1 Data augmentaion in perception module 20
5.2.2 Coordinate transformation 21
5.2.3 Simulate multiple cameras 21
5.3 Baselines 22
5.4 Self-driving peformance 23
5.5 Perception results 23
5.6 Ablations 25
5.7 Offline Evaluation 27

Chapter 6 Conclusions **29**

Bibliography **30**

Chapter 1

Introduction

1.1 Introduction

Self-driving has the potential to reduce accidents, enable the mobility-impaired and reduce emission. However, the technical challenge remains unsolved after decades of research from the first project ALVINN [31], to the recent explosion of self-driving companies. In this thesis, we propose a self-driving system that decouples perception from action in learning vision-based driving policy from cameras.

End-to-end driving, especially from monocular cameras, is hard. It needs to perceive the static lanes information, and dynamic agents, such as vehicles and pedestrians to predict their potential interactions and output actions that satisfy all the traffic rules and users' subtle driving styles. It is unreasonable to expect a deep network to learn all of them with only action supervision [8]. More importantly, even if end-to-end learning solves autonomous driving, it is not likely to commercialize since a crucial requirement for bringing self-driving cars to the market is the ability to reproduce and explain its own decisions, especially mistakes that lead to traffic accidents. Such a regulation by NHTSA makes end-to-end designs not realistic because of the credit assignment problem: it's hard to disentangle the perception error from the control's fault.

On the other hand, a modular approach that decouples perception from action makes training much easier, but the design choices of what high-level representation to use still matters a lot. The most popular representation is a rasterized map with drivable areas, traffic lights, vehicles and pedestrians rendered in a bird's eye view [6; 29; 7]. Such scene context information is often encoded with ConvNets, which have limited receptive fields and fail to reason about global interactions between agents. Recently, VectorNet [15] pioneers to use vectorized representation to encode the scene context and use GNNs to model the interactions among traffic participants. However, they require hierarchical information processing and is inefficient to deploy on real cars. We take a minimalist approach to represent objects as their center points with other informative features. Then, a point encoder is applied to encode the scene before feeding into the action module. The whole system

is computationally efficient and can be parallelized easily with modern GPUs.

With our modular design, different modules can be trained with different data distributions, which improves data efficiency and generalizability. Perception module needs to be robust to different weather, lighting and road conditions. And such data can be easily collected without requiring expert driver or special maneuvers. On the other hand, action module needs to reason about challenging scenarios and learns to recover from its own mistakes. However, collecting such data is expensive since the real-world driving data contains very few interesting cases and thus, it is not a scalable way to increase our model performance with increasingly more fleets of cars. Interestingly, our point-based representation enables many kinds of data augmentations for free. For instance, we can simulate any heading directions of the ego vehicle by performing a matrix transformation on the points **(author?)** [4]. This enables our driving policy to know how to recover from mistakes without requiring dangerous expert maneuvers.

Besides the benefits in training self-driving agents, our modular design also makes extensive offline testing possible. Many previous work reported that offline metrics, such as offline prediction error, are not reliable for autonomous driving and testing in the real world is the golden standard for its performance [9], which leads to many safety and budget concerns. Our modular design makes comprehensive and meaningful offline testing possible. More specifically, the perception module is detection and tracking, which have many well-studied evaluation metrics. And our point representation of local scenes enables us to exhaustively test our policy. For example, we can perform grid search over possible vehicle poses and test whether our policy will react to other traffic entities in a timely manner. Such evaluations of our perception and policy module can be performed entirely offline, which will speed up the development cycle dramatically.

In summary, our contributions are three-fold: First, we design a point-based modular self-driving system that decouples perception from action and achieves the best performance among offline methods on CARLA urban driving benchmark; Second, our choices of high-level scene representation enables many useful data augmentations to cover potential scenarios and alleviate the distributional shift in autonomous driving; Third, our modular design enables extensive offline testing, which is crucial for bringing self-driving cars to the real world.

Chapter 2

Related Work

2.1 Object detection and tracking

Object detection is one of the most popular research areas in computer vision. There are three main approaches of object detection: region classification [17; 16], implicit anchors [35; 25; 26; 34] and keypoint estimation [22; 48; 47].

Object tracking can be divided into two most well-known streams of methods, tracking-by-detection [3; 23] that first uses an off-the-shelf object detector to find all objects in each individual frame and then associate those detected objects across frames and joint detection and tracking [14; 44; 46], which directly performs tracking on frame sequences with either previous bounding boxes or keypoints as additional references.

We directly adopt CenterTrack [46] for its simplicity. In our modular driving pipeline, we only need to track object centers and can use them as a high-level representation of our observations.

2.2 Road and lane detection

Object detection is useful for locating some objects of interest but are inadequate for continuous surfaces like roads. Determining the drivable surface or lane boundary is critical for self-driving and has been specifically researched. While drivable surfaces or lane boundaries can be determined through semantic segmentation, many previous methods require understanding of lanes, and how they are connected through merges and intersections, which remains a challenge from the perspective of perception [2].

In our pipeline, we represent lanes as points to feed into our policy module. Thus, we put fewer constraints on lane geometry, reduce the accuracy requirement in perception and leave the rest of work to our policy module. In our experiments, we use a basic semantic segmentation network, EfficientNet [1], to segment out the lanes, which is sufficient for our self-driving cars to navigate.

2.3 Imitation Learning

Imitation learning is a powerful technique to learn complex decision and control behaviors from observed expert demonstrations. Compared to reinforcement learning [28; 37; 38; 18], imitation learning is more data efficient and does not require exhaustive exploration, which makes it appealing in safety-critical systems, such as autonomous driving.

We mainly focus on the widely used behavioral cloning paradigm, which directly learns a mapping from observations to expert actions. Behavioral cloning usually suffers from distributional shift because small error will compound through time and finally leads to unfamiliar states not supported by the training data.

Our method builds on the high-level representation from tracking enables many data augmentations, such as simulating different camera angles [4], counterfactual data [30], challenging scenarios, almost for free, which is shown to be crucial for tackling the distributional shift [4; 36] and causal confusion [12] problems commonly arised in autonomous driving.

2.4 End-to-End vs. Modular approach in policy learning

Modular policy learning pipelines usually decouple perception from action, which can easily introduce different inductive biases into different modules. More specifically, most autonomous driving stacks decompose the pipeline into HD mapping, localisation, perception, prediction, planning and control [40; 20]. Such a design enables separation of responsibilities, better interpretability, and parallel development of different modules.

On the other hand, end-to-end visuomotor policy learning shifts away from manually defined modules and proposes to learn to control directly from raw observations. These methods rely on the capability of deep networks that perceptual capabilities will arise in the model as needed, as a result of training for specific motor tasks. Although end-to-end visuomotor control has some promising results on challenging tasks, ranging from video games [27], robot control [24] to autonomous driving [10], it's been widely shown that learning from a high-level representation from a well-designed computer vision system still outperforms them by a large margin [45; 41; 29].

Our approach inherits all the advantages of modular design, and by using carefully-designed data augmentations specifically to our point-based representation of the world, we will show a huge performance improvement and better generalization ability across different weathers, cities and time of a day.

Chapter 3

Preliminaries

3.1 Partially Observed Markov Decision Process

The decision process of an agent can be described by a partially observed Markov decision process (POMDP) since the sensors such as cameras or LIDARs can only perceive part of the environment at each time step. A POMDP can be formalized as a tuple $(\mathbf{S}, \mathbf{A}, \mathbf{T}, r, \mathbf{O})$, where $\mathbf{S}$ is the state space of the environment, $\mathbf{A}$ is the action space of the agent, $\mathbf{T}$ is the transition probabilities between states with a given action, $r : \mathbf{S} \times \mathbf{A} \to \mathbb{R}$ is the reward function and $\mathbf{O}$ is the observation space of the agent. At each time step t, the agent does not access to the underlying true state s_t and has to take the action a_t according to the observation o_t.

In imitation learning, we are given a demonstration dataset of observation-action tuples: $\{(o_t, a_t)\}$. They represent expert behaviors, i.e., the trajectories that achieve high accumulated rewards. However, the reward is not provided in the imitation learning setup. Instead, the goal of imitation learning is to learn a function that maps observation histories $\tilde{o}_t$ to the expert action a_t that leads to high accumulated rewards.

3.2 Behavioral Cloning

Among all the variants of imitation learning, we focus on behavioral cloning (BC), a straightforward but powerful approach to mimic expert behaviors from a demonstration dataset $\mathcal{D} = \{(o_i, a_i)\}_{i=1}^{N}$, where N is the number of samples. BC reduces policy learning to supervised learning by maximizing the log-likelihood of the action a_t conditioned on the observation history $\tilde{o}_t$ so that the agent can behave similarly to the expert, i.e.,

$$\theta^* = \arg\max_{\theta} \mathbb{E}_{\mathcal{D}}[\log P(a_t|\tilde{o}_t; \theta)]. \tag{3.1}$$

Chapter 4

System Design

4.1 State representation

State representation is crucial for learning a good driving policy. Many previous work [39] chooses to render the HD maps as color-coded attributes and encode the scene context information with ConvNets, which ignores the structure of the HD map and suffers from limited receptive size. Recently, VectorNet [15] proposes to represent the static scene and the dynamic actors as vectors or polylines, which is more efficient but still suffer from the overhead of hierachical feature encoding and information exchange in GNNs.

We take the minimalist ideology and propose to represent the local scene, including ego vehicle, other vehicles, pedestrians, traffic lights and lanes, as points, which contains the essential information about an agent/object to decide driving actions. More specifically, at time step t, every point $p_i^t \in \mathbb{R}^6$ is defined as

$$[o_x^t, o_y^t, cos(\theta)^t, sin(\theta)^t, s^t, id]$$

, which contains the following attributes: 3D center position (o_x^t, o_y^t), orientation ($cos(\theta)^t$, $sin(\theta)^t$), speed (s^t) and their class label (id). For dynamic agents, such as vehicles and pedestrians, their speed s_t is defined as the distance traveled between two consecutive time steps $t-1$ and t.

$$s_t = \sqrt{(o_x^t - o_x^{t-1})^2 + (o_y^t - o_y^{t-1})^2}$$

For static agents, their speed s_t is set to 0 for all time steps t. Since lanes are continuous, we sample points regularly from them with a resolution of 1m. For traffic light, we assign different ids to different light states (green, yellow and red) to include the necessary information to decide whether to throttle or brake. Note that we do not include vehicle's and pedestrian's width and length in our state representation because we believe the policy can be conservative by assuming a max-sized blob around each center point of the traffic participants. Since these points

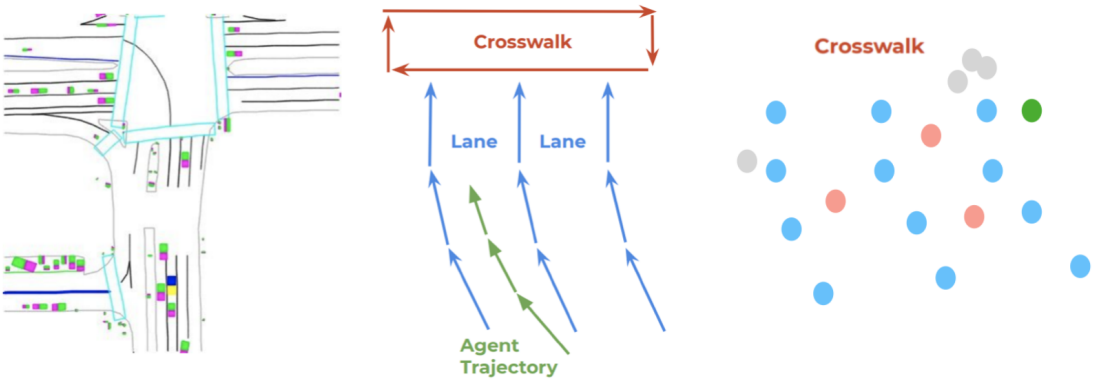

Figure 4.1: Comparisons of different scene representations (Figure partly from [15]). The left and middle figure represent scene as semantic map and vectors respectively. The right figure is our proposed point representation, which uses center points with different semantics to represent traffic agents.

are simply feature vectors, the state can be expanded to contain more attributes, such as vehicle color, brand, traffic light remaining time if needed in specific driving cases.

By representing dynamic agents as point features of their 3D centers and lanes as multiple regularly sampled points, we avoid the need of learning a polyline feature for each agent or object as in [15] and can directly model the high-level interactions between different traffic participants, which further speeds up the training and inference of the driving policy.

4.2 Perception Module

Since our goal is to learn a driving policy from monocular cameras, we need to infer our state representation from images taken by the camera mounted in front of the ego vehicle. With the rapid development of detection, tracking and segmentation in Computer Vision, there are many off-the-shelf models to use. Specifically, we choose CenterTrack [46] for detecting objects/agents, such as vehicles, pedestrians and traffic lights. And we use EfficientNet for lane segmentation [1].

Our detection network F_θ takes in the current image $I^t \in \mathbb{R}^{W\times H\times 3}$, the previous image $I^{t-k} \in \mathbb{R}^{W\times H\times 3}$ and the previous detection results rendered in a class-

Figure 4.2: **Perception module** – Our perception module follows CenterTrack [46]. It takes current and previous observations as well as previous track as input and predicts current object location, orientation and speed (displacement) in pixel coordinates. Then, we project from pixel to world coordinates using known camera intrinsics and extrinsics.

agnostic heatmap $H^{t-1} \in \mathbb{R}^{W \times H \times 1}$ and predicts the state defined in Section 4.1. Following [46], we use intense data augmentation to prevent the detector from overfitting. Our data augmentations include simulating false positive, false negative and inaccurate detections in previous heatmap H^{t-1} and sampling previous image I^{t-k} from $k = 1, 2, 3$ to prevent the detector from overfitting to the frame rate.

We directly predict locations as a class-specific heatmap $\hat{Y}_{xyc} \in \mathbb{R}^{W/\alpha \times H/\alpha \times C}$ where $C = 5$. We use focal loss [25] as shown below.

$$\mathrm{L}_{loc} = \frac{1}{N} \sum_{xyc} \begin{cases} \left(1 - \hat{Y}_{xyc}\right)^{\beta} \log\left(\hat{Y}_{xyc}\right) & \text{if } Y_{xyc} = 1 \\ \left(1 - Y_{xyc}\right)^{\gamma} \left(\hat{Y}_{xyc}\right)^{\beta} \log\left(1 - \hat{Y}_{xyc}\right) & \text{otherwise} \end{cases} \tag{4.1}$$

where $\beta = 2$ and $\gamma = 4$ are hyperparameters.

The orientation θ is a single scalar by default. Following [19], we use an 8-scalar encoding. The 8 scalars are divided into two groups $\left[-\frac{7\pi}{6}, \frac{\pi}{6}\right]$ and $\left[-\frac{\pi}{6}, \frac{7\pi}{6}\right]$, each for an angular bin. Within each bin, 2 of the scalars $b_i \in \mathbb{R}^2$ are used for softmax classification. And the rest 2 scalars $a_i \in \mathbb{R}^2$ are for the sin and cos value of in-bin offset.

The classification are trained with softmax and the angular values are trained

with L1 loss:

$$L_{\text{ori}} = \frac{1}{N}\sum_{k=1}^{N}\sum_{i=1}^{2}\left(\text{softmax}\left(\hat{b}_i, c_i\right) + c_i\left|\hat{a}_i - a_i\right|\right) \tag{4.2}$$

where

$$c_i = \mathbb{1}\left(\theta \in B_i\right)$$

and

$$a_i = \left(\sin\left(\theta - m_i\right), \cos\left(\theta - m_i\right)\right)$$

$\mathbb{1}$ is the indicator function and m_i is the middle of bin i. We do not include the time step index $^{(t)}$ on the variables for clarity.

Moreover, we also predict center offset δ_c, which avoids discretization issues and tracking offset δ_t to provide us with agents' speed. They're both supervised with mean square loss to their ground-truth value.

$$L_{\delta_c^{(t)}} = \frac{1}{N}\sum_{i=1}^{N}\left|\hat{\delta}_{c_i^{(t)}} - \left(\mathbf{o}_i^{(t)} - \text{int}\left(\mathbf{o}_i^{(t)}\right)\right)\right| \tag{4.3}$$

$$L_{\delta_t^{(t)}} = \frac{1}{N}\sum_{i=1}^{N}\left|\hat{\delta}_{t_i^{(t)}} - \left(\mathbf{o}_i^{(t-1)} - \mathbf{o}_i^{(t)}\right)\right| \tag{4.4}$$

where $o^{(t-1)}$ and $o^{(t)}$ are the low-resolution pixel coordinates of objects at time step $t-1$ and t. We avoid the downsampling factor α for simplicity.

Next, we will discuss our lane segmentation model. Since lanes are continuous, a key-point based detector may produce overly sparse detections or inconsistent local geometry. Thus, we use a segmentation model G_ϕ to segment out lanes of different directions and the background. G_ϕ only takes the current image observation $I^t \in \mathbb{R}^{W\times H\times 3}$ as input and is trained with Dice loss:

$$L_{lane} = \frac{1}{N}\sum_{i=1}^{N}\frac{2|\hat{X}_i \cap X_i|}{|\hat{X}_i| + |X_i|} \tag{4.5}$$

where X and $\hat{X}$ are the groud-truth and predicted segmentation map respectively.

The overall loss function is

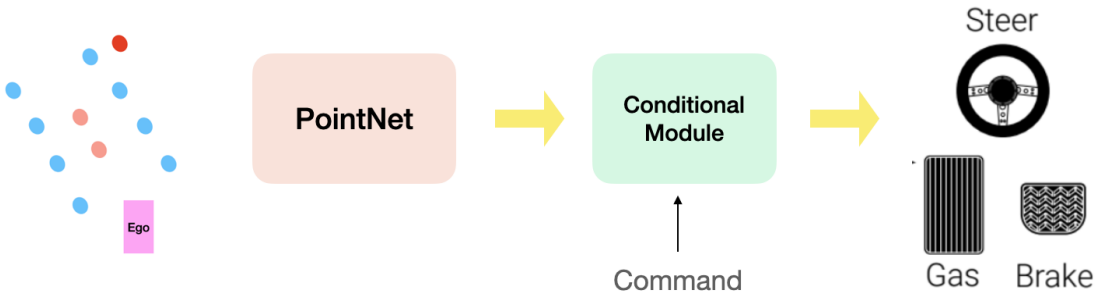

Figure 4.3: **Policy module** – Our policy module acts on detected states and predicts the driving actions conditioned on navigation commands.

$$L_p = L_{loc} + L_{ori} + L_{\delta_c} + L_{\delta_t} + L_{lane} \tag{4.6}$$

4.3 Policy Module

As we represent the local scene as points, a natural choice is to use a permutation invariant point encoder H_ψ to obtain the feature. We use PointNet [33], which has been shown successful in 3D classification, segmentation and reconstruction. We directly predict the low-level controls, including steering, throttle and brake following previous work [10]. Similar to [10], our policy is also conditioned on navigation commands, such as turn left, go straight, turn right and follow., which is proved to be useful especially at intersections.

The loss to train our policy is the following:

$$L_a = \frac{1}{N} \sum_{i=1}^{N} |\hat{a}_i - a_i| \tag{4.7}$$

Since our policy is built upon the high-level point representation of the scene, we can easily simulate different traffic scenarios for better training and testing purposes. For example, we can simulate multiple cameras by rotating all the points, except the ego vehicle point, with simple matrix multiplication. Moreover, we can add, remove or disturb the points to simulate new scenarios to test the robustness and generalization ability of our policy. Such low-cost and highly-realistic simulation is crucial to bringing self-driving cars to the real-world as it greatly speedups

development cycle compared with traditional road testing pipeline.

Chapter 5

Experiments

5.1 Environment

5.1.1 CARLA

CARLA is a photorealistic urban autonomous driving simulator [13], and is commonly adopted for closed-loop evaluation [10; 11; 8; 43; 32]. We collect our own dataset using the autopilot in CARLA, which is about 10 hours of driving. We evaluate all methods in the CARLA *Nocrash* benchmark, which has varying numbers of pedestrians and vehicles in different towns. We pre-define the starting and ending locations of our ego vehicle as shown in Figure 5.1 for evaluating our agent. We use %success as our metric, which is the number of episodes that are fully completed without colliding into other vehicles, pedestrians and lane boundaries.

5.1.2 Expert data collection

CARLA provides a hand-engineered autopilot, which have access to the true state of the simulator. Note that the autopilot is not perfect as it has a success rate of 85%. So, we immediately terminates an episode if the expert crashes to filter out bad maneuvers.

We collect our expert data in CARLA [13]. During collecting, 10% expert actions are perturbed by noise [21] to augment the state. We use three cameras: a forward-facing one and two lateral cameras facing 30 degrees away towards left or right [5] to cover a large field of view. Both noise injection and multiple cameras are common data augmentation techniques to alleviate distributional shift in autonomous driving.

Our dataset is collected in Town01 under 4 different weather conditions: clear noon, hard rain noon, clear sunset and hard rain sunset.

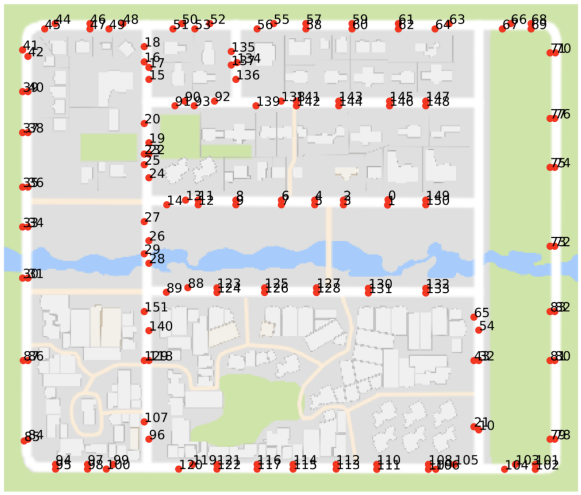

Figure 5.1: **CARLA Town01 map** – Red dots are the pre-defined starting and end points used to test our agent.

5.2 Implementation Details

5.2.1 Data augmentaion in perception module

At inference time, this tracklet heatmap H_{t-k} of previous frame can contain some wrongly localized objects, false negatives and false positives. These errors are not present in ground-truth tracklets provided during training.

Following [46], we simulate this test-time error during training. First, we simulate inaccurate detections by locally jittering each tracklet from the prior frame by adding Gaussian noise to each center. Second, we randomly add false positives near ground-truth object locations by rendering a spurious noisy peak with probability 0.1. Third, we simulate false negatives by randomly removing detections with probability 0.5.

Moreover, similar to [46], we sample the previous frame H_{t-k} from $k = 1, 2, 3$ with equal probability during training to prevent the model from overfitting to the frame rate. And during inference, we only use $k = 1$ as the previous frame.

Note that we do not perform any image data augmentation for detection since we believe our collected dataset is diverse enough, containing scenes collected in different weathers and times of a day.

We follow the default setting in [1] for lane segmentation.

5.2.2 Coordinate transformation

Given a predicted object (o_x^p, o_y^p) in pixel coordinates, we compute its projection onto the ground plane $(o_x^w, o_y^w, 0)$ using the camera's horizontal field of view (fov) $f = \frac{w}{2\tan(fov/2)}$ where w is the width of canvas. We project points onto a constant ground plane $z = 0$ to avoid depth estimation.

$$o_y^c = \frac{f}{h/2 - o_y^p} z_c \tag{5.1}$$

$$o_x^c = \frac{o_x^p - w/2}{h/2 - o_y^p} z_c \tag{5.2}$$

$$\begin{bmatrix} o_x^w \\ o_y^w \\ - \\ - \end{bmatrix} = E \begin{bmatrix} o_x^c \\ o_y^c \\ -z_c \\ 1 \end{bmatrix} \tag{5.3}$$

The camera is placed at a height of $z_c = 1.4$ in the vehicle's reference frame. The camera faces forward and has a resolution of $w \times h = 384 \times 160$ with horizontal $fov = 90^\circ$ and extrinsic matrix E.

Then, we project from global world coordinates to local coordinates in ego vehicle's frame. Assume ego vehicle is at (e_x^w, e_y^w) with orientation e_θ, then

$$\begin{bmatrix} o_x^l \\ o_y^l \end{bmatrix} = R \left(\begin{bmatrix} o_x^w \\ o_y^w \end{bmatrix} - \begin{bmatrix} e_x^w \\ e_y^w \end{bmatrix} \right) \tag{5.4}$$

where

$$R = \begin{bmatrix} \cos e_\theta & -\sin e_\theta \\ \sin e_\theta & \cos e_\theta \end{bmatrix} \tag{5.5}$$

5.2.3 Simulate multiple cameras

The expert drives almost perfectly, which lacks data points about recovering from mistakes. For example, what should I do if I'm oriented 30 degree to the right of the center lane? If we can augment our dataset with scenarios of different ego orientations and their corresponding steerings, we can recover from our mistakes after deployment. As a result, we need to simulate those bad maneuvers to learn

a robust driving policy.

Our point representation gives us an easy way to simulate those scenarios. We can simply let $e_\theta = e_\theta + \delta_\theta$ where δ_θ is the radiance angle offset of the simulated camera sensor. And the coordinate transformation in Section 5.2.2 will take care of the rest.

We can apply the steering physical equation to calculate the corresponding steering of the new orientation.

$$\delta_{steer} = \frac{6}{\pi} \times \arctan \frac{\delta_\theta \times cl}{s + 0.05} \tag{5.6}$$

where cl is the car's length and is set to 6.

$$\delta_{steer} = \min\left(\delta_{steer}, 0.3\right) \tag{5.7}$$

$$a_{steer} = \max\left(1, \min(-1, a_{steer} + \delta_{steer})\right) \tag{5.8}$$

5.3 Baselines

CIRLS [9]. Behavioral cloning from the current observation conditioned on high-level navigation commands, which is shown to perform better than BC with observation histories [12].

LBC [8]. Learning by cheating decouples perception from action by first learning a state-based agent and then distills an image agent. It uses online rollout to bootstrap its performance.

LBC-Offline [8]. Since LBC rolls out the priviliged agent in the environment to generate additional training data for the image agent. We also compare with a version without any online rollout.

Autopilot. We use the autopilot in Carla to generate our training data. Though it is not perfect due to the limitations of a rule-based policy, we treat it as an upper bound of our method.

Task	Weather	CILRS	LBC-Offline	LBC	Autopilot	OURS
Empty	train	87 ± 1	88 ± 2	$\mathbf{100} \pm 0$	$\mathbf{100} \pm 0$	$\mathbf{100} \pm 0$
Regular		83 ± 0	74 ± 4	$\mathbf{99} \pm 1$	$\mathbf{99} \pm 1$	98 ± 1
Dense		42 ± 2	28 ± 4	$\mathbf{95} \pm 2$	86 ± 3	76 ± 2
Empty	test	87 ± 1	91 ± 3	$\mathbf{100} \pm 0$	$\mathbf{100} \pm 0$	$\mathbf{100} \pm 0$
Regular		88 ± 2	71 ± 5	$\mathbf{99} \pm 1$	$\mathbf{99} \pm 1$	85 ± 2
Dense		70 ± 3	24 ± 2	$\mathbf{97} \pm 2$	83 ± 6	60 ± 4

Table 5.1: Success rate of OURS and baselines in training and testing weathers in CARLA NoCrash benchmark.

5.4 Self-driving peformance

As shown in Table 5.4, our method performs the best among offline methods, including an end-to-end approach (CIRLS) and a policy distillation approach (LBC-Offline). CIRLS uses significantly more data (100 hrs driving), but still performs much worse than OURS. LBC-Offline distills an image agent from the semantic map based state agent, which lacks supervising signals to reason about the image and thus, is hard to learn an accurate low-level control. This demonstrates the advantage of our scene representation and our modular design of the self-driving stack.

Compared with an online method (LBC), which uses the state-based agent as expert during online rollout to supervise the image agent, OURS performs worse, especially in dense traffic. This suggests that learning from a broader state distribution is crucial for driving. However, online rollout is expensive and not practical in the real world.

The autopilot, with access to ground-truth state, uses a PID controller to get the driving actions. There is still a 20% gap between OURS and the autopilot in dense traffic. We will investigate where does the gap come from in our ablations.

5.5 Perception results

We use the Pascal VOC metrics to evaluate our object detector. Since we represent the agents by their center points and do not predict their sizes, we heuristically set a bounding box with height and width equal to 10 pixels around the center. Our policy network uses a local region of 80m around the ego vehicle, so we evaluate

Weather	AP (vehicles)	AP (pedestrians)	mAP
train	49.81	18.04	33.92
test	42.33	14.73	28.53

Table 5.2: APs of vehicles and pedestrians in train and test weather.

Distance (m)	10	20	30	40	50
vehicle	4.4	67.31	45.6	33.55	28
pedestrian	2.17	6.95	6.3	24.97	18.74

Table 5.3: APs of vehicles and pedestrians within different distances to the ego vehicle.

APs within the same region. As shown in Table 5.5, the vehicle AP is 49.81 and the pedestrian AP is 18.04. The pedestrians are much harder to detect since they are often in a crowd and are often too small to detect when they are far away from the ego vehicle.

Since nearby objects are more important to our driving policy, we also evaluate APs within different distances relative to the ego vehicle, ranging from 10m to 50m. Table 5.5 shows how APs change with the relative distance of the object. Vehicle detection performance improves when it gets closer to the ego vehicle, but when it is too close (less than 10m), it becomes hard to detect because the center points often fall outside the image. Pedestrian detection performance is poor in close regions. The reason is that in our simulation, it is more likely to spawn more pedestrians nearby the ego vehicle to make the scenario more challenging and thus, it is hard for our detector to correctly identify every pedestrian in the crowd.

Moreover, we show some visualizations of the detected agents' center points in pixel and world coordinates. The green dot is the ego vehicle. The green and blue circles are ground-truth vehicles and pedestrians respectively. The light blue crosses are the detected vehicles while the yellow crosses are the detected pedestrians. These detected agents are also shown in the front-view image with the same color code. The top two examples show some successful detections of vehicles and pedestrians, but it is still hard to detect every pedestrian when they are in a crowd. The bottom example is a common failure case: the vehicle on the opposite lane is extremely close to the ego vehicle so that its center point is out of the canvas frame and is not detected. One potential solution is to render a point on the edge of the

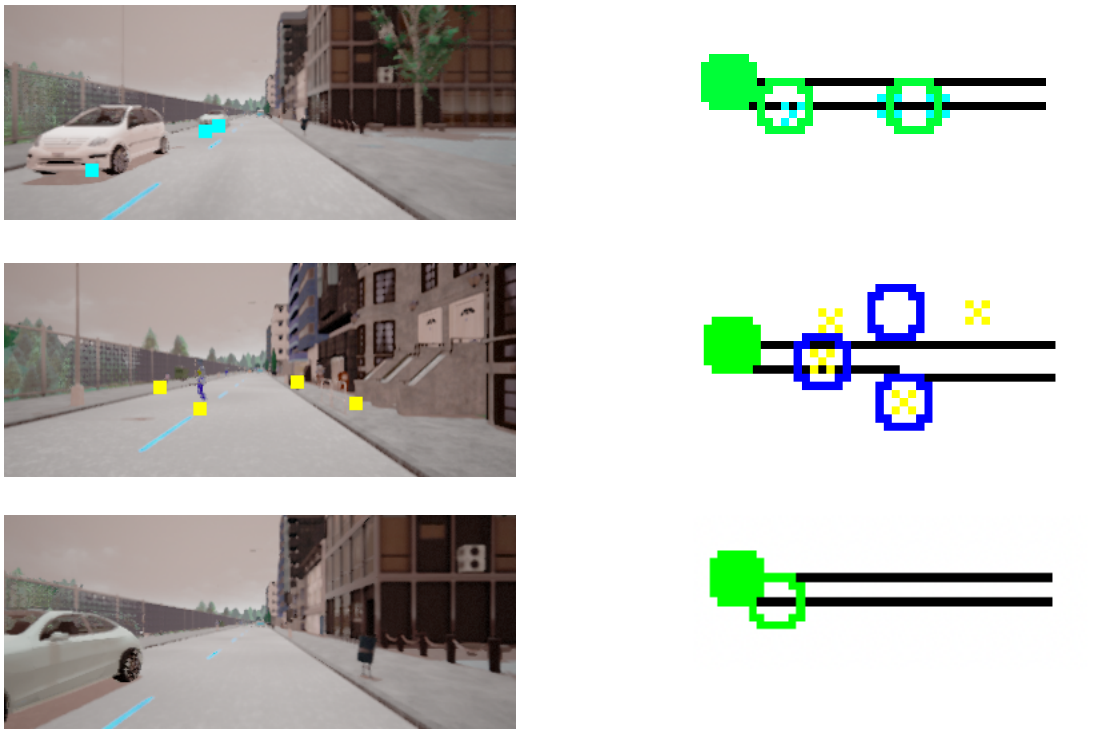

Figure 5.2: Visualizations of our object detector. Solid green dot is the self-driving car. Green and blue circles are ground-truth vehicles and pedestrians. Blue and yellow crosses are detected vehicles and pedestrians. The squares in the image are the detected centers of each object.

canvas and predict an offset vector to handle those extremely close objects, which we leave for future work.

5.6 Ablations

We perform all of our ablations in the most difficult dense traffic setting. In our ablations, we hope to answer the following three questions:

1) Which is the bottleneck of our driving pipeline? The perception module or the policy module?

An advantage of our modular driving system is that we can develop, train and evaluate each module separately, which greatly speeds up the development cycle and is a crucial but often ignored factor to bring self-driving cars to the real-world.

Since **OURS** still has a 20% gap to the autopilot, we would like to identify the bottleneck in our system. We simply replace our detection results by the ground-truth states provided by the simulator and keep the same policy module. We name this ablation as **OURS w/ GT detection**.

OURS w/ GT detection performs even better than the hand-engineered autopilot, achieving 96% success rate across different weathers. This demonstrates that if we have a perfect perception module, a point-based driving policy can be better than a hard-coded policy, which again emphasizes the idea that a good scene representation is important for the performance of a driving model.

2) Does simulating multiple cameras improve driving performance?

Simulating multiple cameras easily and efficiently is an advantage of our point representation. But, how useful is it? In order to answer this question, we compare with an ablation **OURS w/o multi-view simulation** that doesn't augment the point representation with simulated camera transformation.

OURS w/o multi-view simulation has a success rate of 5% drop in the training weather and 8% drop in testing weather.

This is validated by many prior work [4; 10; 11] that data points of different ego orientations are important. However, since these methods are focused on end-to-end driving, they need to mount multiple cameras in order to simulate diverse ego orientations, which increases the hardware cost. In [8], they also simulate multiple ego orientations in the bird's-eye view. However, since they represent the scene as a semantic map, they need to re-render everything from scratch, which is inefficient and time-consuming.

3) Does a rule-based controller help?

Since our perception module is not perfect and our observation is usually partially observed, we want to take the history observations into account. However, many previous work [12; 42] reports that adding historical observations hurts the driving policy's performance.

Since our driving system is modularized, we can utilize historical observations in a very simple way. More specifically, we constrain the self-driving car to stop if a nearby obstacle (vehicle or pedestrian) is detected in any one of the past 10 frames. Thus, our policy is more robust to false negatives of the detector, which is a more severe problem than false positives in real driving scenarios.

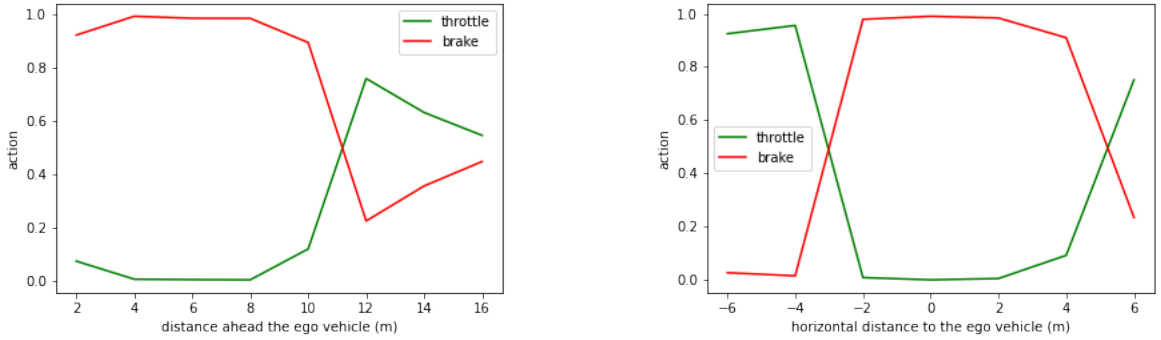

Figure 5.3: Action of the self-driving car when a front other vehicle is at different vertical and horizontal offsets.

We compare with an ablation without the controller, namely **OURS w/o controller**. The performance drops 23% in the training weather and 27% in the testing weather, validating the importance of temporal information to the policy.

4) Does joint training of the perception and policy module help?

Although we decople the perception and policy module in our design, we can also train the whole system end-to-end using both object bounding boxes and expert actions as supervision. However, in practice, we find it difficult to achieve good driving performance with joint training.

5.7 Offline Evaluation

Since we represent agents as points, we can perform extensive offline evaluation by adding, removing or perturbing the points.

Here, we test the scenario when there is a vehicle in front of the ego vehicle. Figure 5.7 shows the ego car's reaction to the other car at different vertical and horizontal offsets. If we fix the horizontal offset of the front car to 0, we can test the self-driving car's reaction to obstacles of varying vertical offset. When the vertical offset is less than 10m, the self-driving car brakes hard and keep throttling if the front car is more than 10m ahead of it. On the other hand, if we fix the vertical offset to 10m, the self-driving car will brake when the horizontal offset is less than 3m.

Such grid-search style test shows that our self-driving car can react to nearby agents in time to avoid collision, which is more reliable than the L1 or L2 error used in prior work [9].

Chapter 6

Conclusions

In this work, we propose a modular driving system that decouples perception from action. Our perception module detects and tracks the dynamic agents in the scene and represents them as points in bird's-eye view. Our policy directly takes in those semantic points and predicts the maneuvers at the current time step. Such a design choice together with our point representation of traffic scene allows many data augmentations for free, such as multi-view simulation, targeted data generation and so on. We extensively evaluate our driving pipeline in the most popular driving benchmark and it achieves state-of-the-art performance compared with offline imitation methods. Our ablation studies show that the current system bottleneck is in the perception module and can be further improved with better model choices or larger training datasets.

Bibliography

[1] Bhakti Baheti, Shubham Innani, Suhas S. Gajre, and Sanjay N. Talbar. Eff-unet: A novel architecture for semantic segmentation in unstructured environment. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1473–1481, 2020.

[2] Aharon Bar-Hillel, Ronen Lerner, Dan Levi, and Guy Raz. Recent progress in road and lane detection: a survey. *Machine Vision and Applications*, 25:727–745, 2011.

[3] Alex Bewley, ZongYuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Simple online and realtime tracking. *CoRR*, abs/1602.00763, 2016.

[4] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016.

[5] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016.

[6] Sergio Casas, Wenjie Luo, and Raquel Urtasun. Intentnet: Learning to predict intention from raw sensor data. *CoRR*, abs/2101.07907, 2021.

[7] Yuning Chai, Benjamin Sapp, Mayank Bansal, and Dragomir Anguelov. Multipath: Multiple probabilistic anchor trajectory hypotheses for behavior prediction. *CoRR*, abs/1910.05449, 2019.

[8] Dian Chen, Brady Zhou, Vladlen Koltun, and Philipp Krähenbühl. Learning by cheating. In *Conference on Robot Learning*, pages 66–75. PMLR, 2020.

[9] Felipe Codevilla, Antonio M. López, Vladlen Koltun, and Alexey Dosovitskiy. On offline evaluation of vision-based driving models. *CoRR*, abs/1809.04843, 2018.

[10] Felipe Codevilla, Matthias Miiller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. End-to-end driving via conditional imitation learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–9. IEEE, 2018.

[11] Felipe Codevilla, Eder Santana, Antonio M López, and Adrien Gaidon. Ex-

ploring the limitations of behavior cloning for autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 9329–9338, 2019.

[12] Pim de Haan, Dinesh Jayaraman, and Sergey Levine. Causal confusion in imitation learning. *Advances in Neural Information Processing Systems*, 32:11698–11709, 2019.

[13] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[14] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. Detect to track and track to detect. *CoRR*, abs/1710.03958, 2017.

[15] Jiyang Gao, Chen Sun, Hang Zhao, Yi Shen, Dragomir Anguelov, Congcong Li, and Cordelia Schmid. Vectornet: Encoding HD maps and agent dynamics from vectorized representation. *CoRR*, abs/2005.04259, 2020.

[16] Ross B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015.

[17] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

[18] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[19] Hou-Ning Hu, Qi-Zhi Cai, Dequan Wang, Ji Lin, Min Sun, Philipp Krähenbühl, Trevor Darrell, and Fisher Yu. Joint monocular 3d vehicle detection and tracking. *CoRR*, abs/1811.10742, 2018.

[20] Ashesh Jain, Luca Del Pero, Hugo Grimmett, and Peter Ondruska. Autonomy 2.0: Why is self-driving always 5 years away? *CoRR*, abs/2107.08142, 2021.

[21] Michael Laskey, Anca Dragan, Jonathan Lee, Ken Goldberg, and Roy Fox. Dart: Optimizing noise injection in imitation learning. In *Conference on Robot Learning (CoRL)*, volume 2, page 12, 2017.

[22] Hei Law and Jia Deng. Cornernet: Detecting objects as paired keypoints. *CoRR*, abs/1808.01244, 2018.

[23] Laura Leal-Taixé, Cristian Canton-Ferrer, and Konrad Schindler. Learning by tracking: Siamese CNN for robust target association. *CoRR*, abs/1604.07866,

2016.

[24] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *CoRR*, abs/1504.00702, 2015.

[25] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *CoRR*, abs/1708.02002, 2017.

[26] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015.

[27] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei Rusu, Joel Veness, Marc Bellemare, Alex Graves, Martin Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–33, 02 2015.

[28] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[29] Jonah Philion and Sanja Fidler. Lift, splat, shoot: Encoding images from arbitrary camera rigs by implicitly unprojecting to 3d. *CoRR*, abs/2008.05711, 2020.

[30] Silviu Pitis, Elliot Creager, and Animesh Garg. Counterfactual data augmentation using locally factored dynamics. *CoRR*, abs/2007.02863, 2020.

[31] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in neural information processing systems*, pages 305–313, 1989.

[32] Aditya Prakash, Kashyap Chitta, and Andreas Geiger. Multi-modal fusion transformer for end-to-end autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7077–7087, June 2021.

[33] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *CoRR*, abs/1612.00593, 2016.

[34] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*,

abs/1506.02640, 2015.

[35] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[36] Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. *Journal of Machine Learning Research*, 15:627–635, 2011.

[37] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

[38] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[39] Rainer Stiefelhagen, Keni Bernardin, Rachel Bowers, John Garofolo, Djamel Mostefa, and Padmanabhan Soundararajan. The clear 2006 evaluation. volume 4122, pages 1–44, 04 2006.

[40] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[41] Dequan Wang, Coline Devin, Qi-Zhi Cai, Philipp Krähenbühl, and Trevor Darrell. Monocular plan view networks for autonomous driving. *CoRR*, abs/1905.06937, 2019.

[42] Chuan Wen, Jierui Lin, Trevor Darrell, Dinesh Jayaraman, and Yang Gao. Fighting copycat agents in behavioral cloning from observation histories. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

[43] Chuan Wen, Jierui Lin, Jianing Qian, Yang Gao, and Dinesh Jayaraman. Keyframe-focused visual imitation learning. In *Proceedings of the 38th International Conference on Machine Learning*, Proceedings of Machine Learning Research. PMLR, 18–24 Jul 2021.

[44] Zheng Zhang, Dazhi Cheng, Xizhou Zhu, Stephen Lin, and Jifeng Dai. Integrated object detection and tracking with tracklet-conditioned detection. *CoRR*, abs/1811.11167, 2018.

[45] Brady Zhou, Philipp Krähenbühl, and Vladlen Koltun. Does computer vision matter for action? *CoRR*, abs/1905.12887, 2019.

[46] Xingyi Zhou, Vladlen Koltun, and Philipp Krähenbühl. Tracking objects as

points. *CoRR*, abs/2004.01177, 2020.

[47] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as points. *CoRR*, abs/1904.07850, 2019.

[48] Xingyi Zhou, Jiacheng Zhuo, and Philipp Krähenbühl. Bottom-up object detection by grouping extreme and center points. *CoRR*, abs/1901.08043, 2019.