Stochastic Navigation Command Matching for Imitation Learning of a Driving Policy

Xiangning Meng, Jianru Xue(✉), Kang Zhao, Gengxin Li, and Mengsen Wu

Institute of Artificial Intelligence and Robotics, College of Artificial Intelligence, Xi'an Jiaotong University, Xi'an, China
jrxue@mail.xjtu.edu.cn

Abstract. Conditional imitation learning provides an efficient framework for autonomous driving, in which a driving policy is learned from human demonstration via mapping from sensor data to vehicle controls, and the navigation command is added to make the driving policy controllable. Navigation command matching is the key to ensuring the controllability of the driving policy model. However, the vehicle control parameters output by the model may not coincide with navigation commands, which means that the model performs incorrect behavior. To address the mismatching problem, we propose a stochastic navigation command matching (SNCM) method. Firstly, we use a multi-branch convolutional neural network to predict actions. Secondly, to generate the probability distributions of actions that are used in SNCM, a memory mechanism is designed. The generated probability distributions are then compared with the prior probability distributions under each navigation command to get matching error. Finally, the loss function weighted by matching and demonstration error is backpropagated to optimize the driving policy model. The significant performance improvement of the proposed method compared with the related works has been verified on the CARLA benchmark.

Keywords: Autonomous driving · Driving policy · Imitation learning

1 Introduction

Driving policy has a pivotal role in autonomous driving system, which builds a bridge from perception to control. Many researches effort within the field of intelligent vehicles have been focused on learning a driving policy. Different from traditional motion planning system [1–4] which realizes autonomous driving from high level to low level (as shown in Fig. 1 left), learning-based method uses a deep neural network to parameterize the driving policy and trains through imitation learning (IL) or reinforcement learning (RL). The learned driving policy model directly maps sensor observations to vehicle controls.

Learning-based methods provide a concise framework for autonomous driving. A series of researches [5–8] for learning driving policy model follow the conditional imitation learning (CIL) [9] which leverages navigation command generated from global

This work is supported by the National Natural Science Foundation of China Projects 62036008.

S. Yu et al. (Eds.): PRCV 2022, LNCS 13536, pp. 192–203, 2022.
https://doi.org/10.1007/978-3-031-18913-5_15

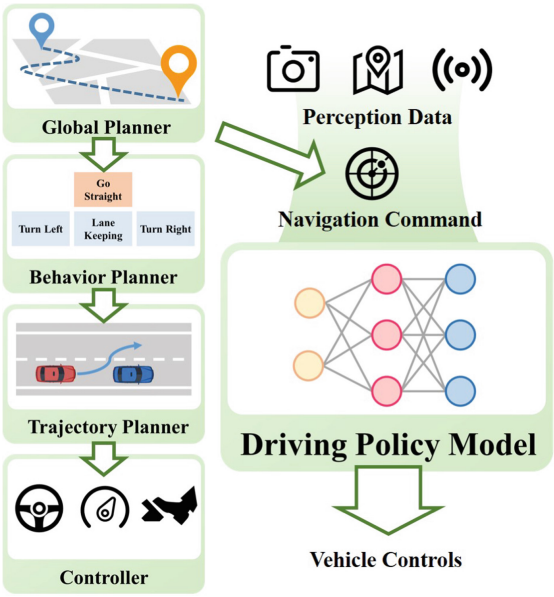

Fig. 1. Overview of the modular pipeline (left) and driving policy model in CIL framework (right). CIL parameterizes driving policy with a neural network and selects the behavior with navigation commands generated from global planner.

route planner to guide motion planning (as shown in Fig. 1 right). The navigation command provides guidance for the vehicle at the intersection and reinforces the controllability of imitation learning. However, during the test of CIL, we find that the vehicle may take wrong actions that are inconsistent with the navigation command in some cases, such as the example shown in Fig. 2. We define this problem as "navigation command mismatch", which most of the existing methods didn't attach importance to as far as we know. Navigation command mismatch may cause the vehicle to spend more time than the optimal global path planning and even make the global planner have to replan the global path.

One possible reason behind the problem of navigation command mismatch in CIL is that it only uses the navigation command as gating function and ignores its strong influence on action generation. The essence of CIL is an end-to-end solution, which may lead learned model only capture weak navigation information. Moreover, CIL only use lowdimensional control parameters as supervision data, effective exploration of the supervision information implied in demonstration data could be used to further improve the performance of the driving policy model. Thus, navigation command matching (NCM) is very important for training an efficient driving policy model.

An NCM model [8] was proposed to generate a smooth reward for reinforcement learning of driving policy model. The model uses conditional probability to measure the matching degree between trajectory (state-action pairs) and navigation commands. Motivated by this work, we further find that relationship between actions and navigation

commands could be used in supervised learning. Thus, we propose stochastic navigation command matching (SNCM) which measures matching error by a metric between two probability distributions of actions under each navigation command.

As illustrated in Fig. 3, we adopt a multi-branch architecture-based convolutional neural network (CNN) (as shown in Fig. 4) as driving policy model. Firstly, images are fed into the multi-branch CNN to compute proposed actions, which are steering angle, throttle and brake. Secondly, we use probability distributions of actions under each kind of navigation commands to describe the matching degree between actions and navigation commands and propose a memory mechanism to compute the distributions. The statistical distributions generated from the model output are compared to the prior distribution to calculate matching error, and computed actions are compared with demonstration data to get demonstration error. Finally, the weighted summation of matching error and demonstration error is backpropagated into the driving policy model to optimize the weights of the network.

The rest of this paper is organized as follows. Section 2 discusses the related works. Section 3 introduces the architecture of our driving policy model and the novel SNCM method for model training. Finally, Sect. 4 presents the experimental results, and Sect. 5 concludes the paper and discusses the future work.

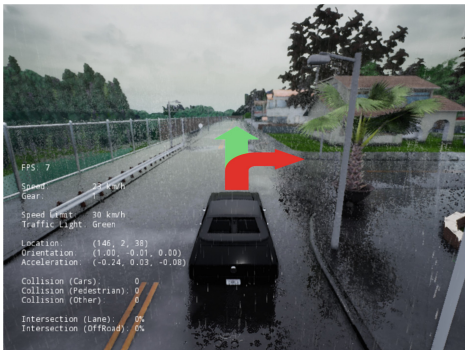

Fig. 2. An example of navigation command failure. The navigation command given at the intersection is *Go Straight* (as shown by the green arrow in the figure), but the motion controls output by the driving policy model make the vehicle turn right at the intersection (as shown by the red arrow in the figure). (Color figure online)

2 Related Works

In learning-based autonomous driving, the mainstream method [9–11] adopts camera image as environment observation, and the methods roughly fall into two categories: imitation learning and reinforcement learning. Our work shares the idea of training a vision-based driving policy model by imitation learning.

Imitation learning enables the agent to learn how to perform a given task through the demonstration of human experts [12]. Bojarski *et al.* [10] did pioneering work in applying the imitation method in learning an end-to-end driving policy, and they trained a CNN through human driving data to control the steering wheel angle so that the vehicle can complete lane following task. Pan *et al.* [13] presented a similar IL system to achieve high-speed off-road autonomous driving in the real world. Similar work e.g. [14–16] focused on basic driving tasks like lane-keeping and obstacle-avoiding. Based on previous work, CIL [9] improved traditional policy network with multi decision branches and activated different branches through navigation command. The introduction of navigation command has been proven to improve the controllability of the driving policy model at intersections and improve the performance of autonomous cars in complex urban environments.

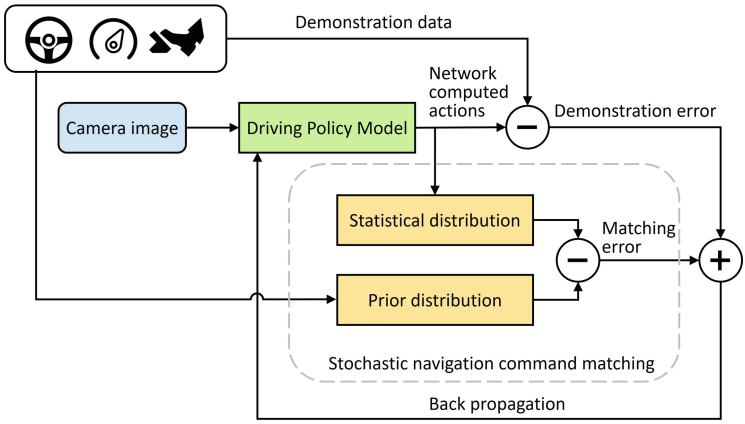

Fig. 3. The process of training driving policy model through stochastic navigation command matching.

As for imitation learning of a driving policy, model optimization uses a loss function. Most studies on driving policy model [7,9,16] adopt mean square error (MSE) between the predicted action value and the ground-truth value as loss functions. Different loss functions are designed along with different factors considered in the driving policy model. Uncertainty-aware imitation learning [6] suggested considering the uncertainty of the model output and proposed an uncertainty-aware loss function, which enabled the vehicle to learn a safer driving policy in unfamiliar scenarios. Li *et al.* [17] proposed a driving approach that splits the driving policy model into a perception module and a driving module. In their approach, softmax categorical cross-entropy and binary cross-entropy are used for perception module training, and MSE is used for driving module training. Conditional affordance learning [18] added the class-weighted categorical cross-entropy and mean average error to its loss function.

Our approach differs from existing methods by introducing NCM into the imitation learning of driving policy model, and we design matching loss to measure the matching degree between generated actions with navigation commands.

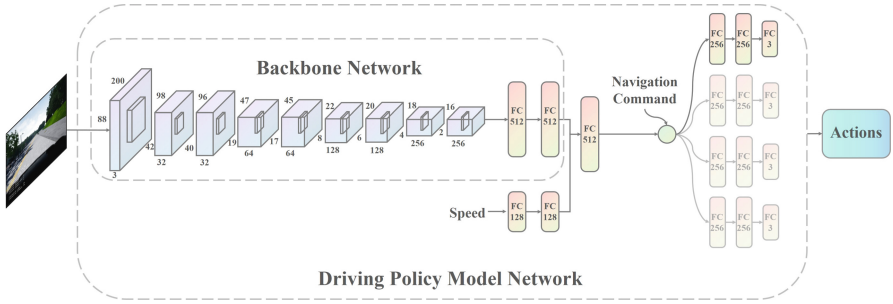

Fig. 4. Network architecture of our driving policy model. Image and speed measurement is fed into the network as input. The four branches follow, and navigation command activates relevant branch which outputs proposed actions.

3 Method

3.1 Problem Formulation

To realize effective driving policy imitation learning, we propose a multi-branch CNN driving policy model trained via SNCM, which generates steering, acceleration and braking commands from camera images, vehicle speeds and navigation commands. Our network architecture is shown in Fig. 4. The driving policy model is built via learning a mapping:

$$\pi(s_t, v_t, c_t, \theta) : s_t \rightarrow a_t \tag{1}$$

where $s_t \in S$ is the observation state of the environment, $a_t \in A$ represents the actions the car will take, v_t is speed, $c_t \in C$ is the navigation command and θ is the policy parameter. At each time step t, the agent will receive an observation s_t and take an action a_t.

3.2 Backbone Network

As the most popular network architecture for driving policy model, CNN has shown good performance in vision-based autonomous driving. Our backbone network is a CNN, which is composed of four convolutional layers and four max-pooling layers sequentially and alternatively. The first layer has 32 5×5 filters with stride 2 followed by three other layers which have respectively 64, 128 and 256 kernels of size 3×3 and stride 1. Through the backbone network, each input image will result in a feature map. The feature map will be flattened and transformed into a feature vector of size 512 through 2 fully connected layers.

3.3 Navigation Command

In conditional imitation learning [9], navigation commands provide guidance for vehicle's action at intersections, which include:

- $Follow\ Lane$: lane keeping
- $Turn\ Left$: turn left at the next intersection
- $Turn\ Right$: turn right at the next intersection
- $Go\ Straight$: go straight at the next intersection

In driving policy model, navigation commands are generated from a global path planner and equivalent to the behavioral decision in the modular framework.

3.4 Multi-branch Architecture

To make driving policy model output actions respond to navigation commands, our model adopts a multi-branch structure in policy network. We use gating function $G(c_i)$ to activate the appropriate branch via control command c_i. In each branch, the feature vector f_i acts as input to three fully connected layers. At the output of the network, the activated branch delivers the action a_i, which consists of steering angle, throttle and brake.

3.5 Stochastic Navigation Command Matching

In the course of interaction with environment, at time step t, the agent receives current observation s_t, speed measurement v_t and navigation command c_t, and then learns to perform an action a_t.

Navigation command has a strong influence on steering angle. To qualitatively describe the relationship between them, we obtain the distribution of steering angles for each navigation command according to the data set in [9] and we use them to describe the relationship between navigation commands and steering angles. Therefore, the navigation command not only activates the branch but also guides generating vehicle actions.

We use the standard distribution as benchmark and compare the similarity between the probability distribution of the steering angle output by the model and the standard distribution. The closer the statistical distribution is to the standard distribution, the more actions output by the model matches the navigation commands.

Memory Mechanism. To make statistics on the steering angle output by the model, we need to sample the steering angle under each navigation command. For the supervised training, it is not available to get a large number of prediction results from the model at every moment.

In deep Q-learning [19], a mechanism known as *replay memory* is used for more efficient sampling. We consider applying *replay memory* in steering angle sampling. Specifically, we collect the pair of navigation command and steering angle $\{(c_i, a_i^s)\}_{i=1}^N$ output by the model at each training step, where N is batch size. It is worth noting that we have set four memory units for four different navigation commands (e.g. *Follow Lane*, *Turn Left*, *Turn Right*, *Go Straight*), the sampling result a_i^s will be stored in its corresponding memory unit according to its navigation command c_i (as shown in Fig. 5). Because the memory unit has a capacity limit, it is necessary to check whether the capacity is exceeded after storing the data every time. If it is exceeded, the earliest data stored in the unit will be deleted automatically.

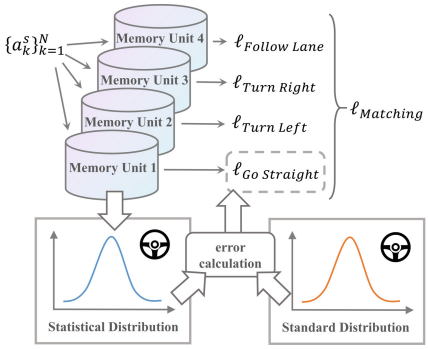

Fig. 5. The process of data storage and matching loss calculation. A batch of steering angles are assigned to memory units according to their corresponding navigation commands respectively, then the memory units update. For each memory unit, a batch of data is sampled to calculate probability distribution. The loss under each navigation command is obtained according to the error between the statistical distribution and prior distribution, and matching loss is the average of them.

Matching Loss. In SNCM, we need to measure the similarity between two distributions. Based on the data set, we get the standard discrete distribution $\mathcal{X} = \{\mathbf{x}^c : c \in C\} = \{(x_1^c, x_2^c, ..., x_n^c) : c \in C\}$ (since we divided the steering angle into 200 units, $n = 200$) of the steering angle under the four navigation commands c ($c \in C, C$ =$\{$*Follow Lane*, *Turn Left*, *Turn Right*, *Go Straight*$\}$). At each training step, we sample in each memory unit separately, and obtain the discrete distribution of steering angle under each navigation command $\mathcal{Y} = \{\mathbf{y}^c : c \in C\} = \{(y_1^c, y_2^c, ..., y_n^c) : c \in C\}$.

We use Bhattacharyya distance to measure the similarity of the two sets of distributions. Bhattacharyya distance is the most common distance metric for measuring the similarity of two probability distributions. For discrete probability distributions, its definition is:

$$D_B(p, q) = -\ln(BC(p, q)) \tag{2}$$

where

$$BC(p, q) = \sum_{x \in X} \sqrt{p(x)q(x)} \tag{3}$$

and p, q are probability distributions of x.

The matching loss calculated by Bhattacharyya distance is:

$$\ell_{matching} = -\frac{1}{|C|} \sum_{c \in C} \ln(\sum_{i=1}^{n} \sqrt{x_i^c \cdot y_i^c}) \tag{4}$$

3.6 Training

We use imitation learning method to train our driving policy network π_θ. The demonstration data $D = \{(o_i, c_i, v_i, a_i)\}_{i=1}^N$ consists of observation image o_i, control command c_i, speed v_i and demonstration action a_i.

The complete training process is shown in Fig. 3. Observation o_i and speed v_i are the main input, and we use gating function $G(c_i)$ to activate appropriate branch via navigation command c_i. The parameter optimization of the driving policy network π_θ through imitation learning is to minimize the target loss function ℓ_{π_θ}, which is composed of action loss ℓ_{action} and matching loss $\ell_{matching}$ weighted:

$$\ell_{\pi_\theta} = \ell_{action} + w_\alpha \cdot \ell_{matching} \tag{5}$$

where the action loss is the MSE of the model's predictive value and the ground-truth value:

$$\ell_{action} = \sum_{i=1}^{N} \ell(\pi_\theta(o_i, v_i, G(c_i)), a_i) \tag{6}$$

and the matching loss is calculated according to (4).

4 Experiments

We evaluate our driving policy model in the open-source urban autonomous driving simulator CARLA [20], which provides a dynamic and open environment for research, development and testing of autonomous driving systems. In this section, we verify the effectiveness of our SNCM training in four challenging driving tasks proposed by CARLA.

4.1 Experiment Setting

Dataset. [9] provides an imitation learning dataset which contains more than ten hours of human driving data in CARLA simulator. The dataset mainly consists of RGB images, state measurements, control commands and navigation commands *et al*. We train our driving policy model on the dataset with the form as $D = \{(o_i, c_i, v_i, a_i)\}_{i=1}^{N}$ which is mentioned in Sect. 3.6.

Evaluation Benchmark. The experimental benchmark includes two experimental conditions: training and test, which are different in map and weather settings. The detailed information about the conditions can be seen in Table 1.

Table 1. Summary of experimental condition.

Condition	Map	Weather
Training	Town1	clear noon, clear sunset, hard rain noon, noon after rain
Test	Town2	cloudy noon after rain, soft rain at sunset

The benchmark provides four tasks with increasing difficulty. In each task, the agent car is randomly initialized in a start point and needs to reach a destination point. The tasks include:

- $Straight$: the start point and the destination point are in a straight line;
- $One\ turn$: there is a turn between the start point and the destination point;
- $Navigation$: there is no special restriction between the start point and the destination point, the path usually includes several turns;
- $Navigation\ with\ dynamic\ obstacles$: same as *Navigation*, but there are dynamic objects in the scenario.

For each combination of a task, a town, and a weather set, the paths are carried out over 25 episodes. In each episode, the target of driving agent is to reach a given goal location. An episode is considered successful if the agent reaches the goal within a time budget, which is set to reach the goal along the optimal path at a speed of 10 km/h.

Implementation Details. Our model was trained using the Adam solver [21] with batch size of 64 samples and an initial learning rate of 0.0001. The capacity of each memory unit is 1000. Training is completed on NVIDIA Titan XP GPUs. Other parameter settings are the same as [9].

4.2 Quantitative Comparison

We compare our SNCM with modular pipeline (MP) [20], reinforcement learning (RL) [20] and conditional imitation learning (CIL) [9] by success rate of autonomous driving tasks on CARLA benchmark. From Table 2, we can observe that our method outperforms all baseline methods with the highest average success rate of about 300 episodes.

Table 2. Average success rate of different methods of all autonomous driving tasks.

Method	Success rate
MP [20]	69.19%
RL [20]	27.44%
CIL [9]	72.19%
SNCM (ours)	**76.13%**

Table 3 reports the quantitative comparisons with baseline methods by the percentage of successful episodes in each task. As we can see, the agent trained via SNCM outperforms the baselines in most tasks, especially in *Straight* and *One Turn* tasks, e.g. 70% of SNCM vs. 50%, 20% and 48% of MP, RL and CIL. Besides, SNCM particularly excels in generalizing to the new town & new weather, the condition where most baselines did not perform well, but the average performance of our method is almost 70% better than the best baseline.

Table 3. Quantitative evaluation on goal-directed navigation tasks, i.e. *Straight* (T1), *One turn* (T2), *Navigation* (T3) and *Navigation with dynamic obstacles* (T4), measured in percentage of successfully completed episodes of the driving tasks. Comparison results include four conditions consisting of different maps and weather settings. The best is in bold.

Method	Training conditions				New town				New weather				New town & weather			
	T1	T2	T3	T4	T1	T2	T3	T4	T1	T2	T3	T4	T1	T2	T3	T4
MP [20]	98	82	80	77	92	61	24	24	**100**	95	**94**	**89**	50	50	47	44
RL [20]	89	34	14	7	74	12	3	2	86	16	2	2	68	20	6	4
CIL [9]	95	89	86	83	97	59	40	38	98	90	84	82	80	48	44	42
SNCM (ours)	**100**	**93**	**87**	78	**98**	**63**	**48**	**42**	**100**	**96**	72	62	**100**	**70**	**58**	**50**

4.3 Qualitative Comparison

We do comparison between SNCM and CIL because CIL was chosen as baseline by most driving policy learning methods. In addition, CIL has the most similar training framework and network structure of SNCM. Figure 6 provides some examples of navigation command mismatch that CIL fails and SNCM successfully avoids.

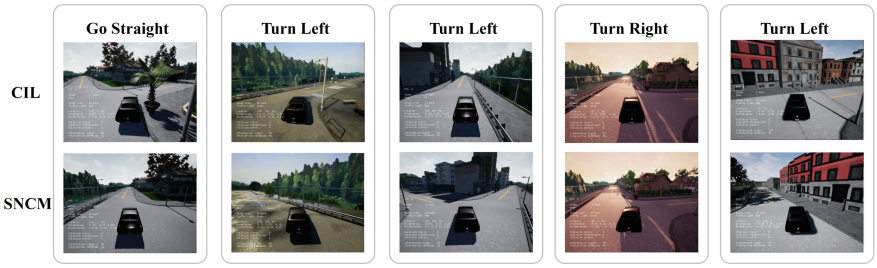

Fig. 6. Comparison between conditional imitation learning (first line) and our stochastic navigation command matching (second line) in some driving cases. The CIL fails with navigation command mismatch while our method successfully completes the driving tasks. Navigation commands of the cases are shown at the top of the images.

4.4 Visualization Results

To understand the input processing of the driving policy model, we extracted and composed the first feature map layer of the CNN to generate a heatmap. The heatmap demonstrates whether the network detects useful features for decision-making.

Figure 7 shows the examples of visualization results in some scenarios. We can observe that the driving policy model trained via SNCM perceives the lane marking and road boundary more clearly, and both static and dynamic targets (e.g. the traffic light and car in the second and third line of Fig. 7) in the environment have more noticeable feature map activations. Contrarily, the model trained without SNCM distracts attention to the background unrelated to the driving task.

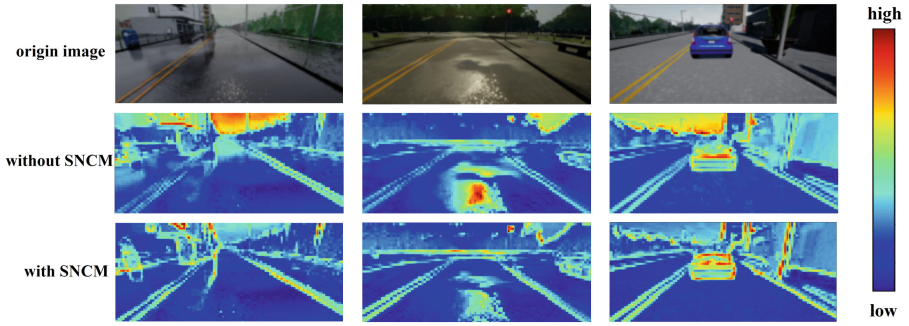

Fig. 7. Visualization of the heatmaps. The first row is origin images, the second and third row are the heatmaps of driving policy model trained with and without stochastic navigation command matching.

5 Conclusions

In this paper, we proposed a novel driving policy training method, stochastic navigation command matching (SNCM), utilizing the correlation between the actions and the navigation commands for model optimization. By considering the matching degree between actions and navigation commands, more reasonable actions are proposed by the learned driving policy model. Experimental results show our method can make driving policy model better overcome the navigation command mismatch problem and improve performance in challenging autonomous driving tasks. Matching degree optimization can be migrated into other hierarchical models, future work will explore its application in more complicated hierarchical autonomous driving system.

References

1. Urmson, C., et al.: Autonomous driving in urban environments: boss and the urban challenge. J. Field Robot. **25**(8), 425–466 (2008)
2. Montemerlo, M., et al.: Junior: the Stanford entry in the urban challenge. J. Field Robot. **25**(9), 569–597 (2008)
3. Bohren, J., et al.: Little Ben: the Ben Franklin racing team's entry in the 2007 Darpa urban challenge. J. Field Robot. **25**(9), 598–614 (2008)
4. Bacha, A., et al.: Odin: team VictorTango's entry in the Darpa urban challenge. J. Field Robot. **25**(8), 467–492 (2008)
5. Liang, X., Wang, T., Yang, L., Xing, E.: CIRL: controllable imitative reinforcement learning for vision-based self-driving. In: Ferrari, V., Hebert, M., Sminchisescu, C., Weiss, Y. (eds.) ECCV 2018. LNCS, vol. 11211, pp. 604–620. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-01234-2_36
6. Tai, L., Yun, P., Chen, Y., Liu, C., Ye, H., Liu, M.: Visual-based autonomous driving deployment from a stochastic and uncertainty-aware perspective. In: 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 2622–2628. IEEE (2019)
7. Cultrera, L., Seidenari, L., Becattini, F., Pala, P., Del Bimbo, A.: Explaining autonomous driving by learning end-to-end visual attention. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, pp. 340–341 (2020)

8. Pan, Y., Xue, J., Zhang, P., Ouyang, W., Fang, J., Chen, X.: Navigation command matching for vision-based autonomous driving. In: 2020 IEEE International Conference on Robotics and Automation (ICRA), pp. 4343–4349. IEEE (2020)
9. Codevilla, F., Müller, M., López, A., Koltun, V., Dosovitskiy, A.: End-to-end driving via conditional imitation learning. In: 2018 IEEE International Conference on Robotics and Automation (ICRA), pp. 4693–4700. IEEE (2018)
10. Bojarski, M., et al.: End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316 (2016)
11. Xu, H., Gao, Y., Yu, F., Darrell, T.: End-to-end learning of driving models from large-scale video datasets. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2174–2182 (2017)
12. Osa, T., Pajarinen, J., Neumann, G., Bagnell, J.A., Abbeel, P., Peters, J.: An algorithmic perspective on imitation learning. arXiv preprint arXiv:1811.06711 (2018)
13. Pan, Y., et al.: Agile autonomous driving using end-to-end deep imitation learning. arXiv preprint arXiv:1709.07174 (2017)
14. Muller, U., Ben, J., Cosatto, E., Flepp, B., Cun, Y.L.: Off-road obstacle avoidance through end-to-end learning. In: Advances in Neural Information Processing Systems, pp. 739–746. Citeseer (2006)
15. Song, S., Hu, X., Yu, J., Bai, L., Chen, L.: Learning a deep motion planning model for autonomous driving. In: IEEE Intelligent Vehicles Symposium (IV), pp. 1137–1142. IEEE (2018)
16. Jiang, H., Chang, L., Li, Q., Chen, D.: Deep transfer learning enable end-to-end steering angles prediction for self-driving car. In: IEEE Intelligent Vehicles Symposium (IV), pp. 405–412. IEEE (2020)
17. Li, Z., Motoyoshi, T., Sasaki, K., Ogata, T., Sugano, S.: Rethinking self-driving: multi-task knowledge for better generalization and accident explanation ability. arXiv preprint arXiv:1809.11100 (2018)
18. Sauer, A., Savinov, N., Geiger, A.: Conditional affordance learning for driving in urban environments. In: Conference on Robot Learning, pp. 237–252 (2018)
19. Mnih, V., et al.: Playing Atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602 (2013)
20. Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., Koltun, V.: Carla: an open urban driving simulator. In: Conference on Robot Learning, pp. 1–16. PMLR (2017)
21. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)