Towards Practical Hierarchical Reinforcement Learning for Multi-lane Autonomous Driving

Anonymous Author(s)
Affiliation
Address
email

Abstract

In this paper, we propose an approach for making hierarchical reinforcement learning practical for autonomous driving on multi-lane highway or urban structured roads. While this approach follows the conventional hierarchy of behavior decision, motion planning, and control, it introduces an intermediate layer of abstraction that specifically discretizes the state-action space for motion planning according to a given behavioral decision. This hierarchical design allows principled modular extension of motion planning, in contrast to relying on either monolithic behavior cloning or a large set of hand-written rules. We show that this design enables significantly faster learning than a flat design, when using both value-based and policy optimization methods (DQN and PPO). We also show that this design allows transferring of the trained models, without any retraining, from a simulated environment with virtually no dynamics to one with significantly more realistic dynamics. Overall, our proposed approach is a promising way to allow reinforcement learning to be applied to complex multi-lane driving in the real world. In addition, we introduce and release an open source simulator for multi-lane driving that follows the OpenAI Gym APIs and is suitable for reinforcement learning research.

1 Introduction

Developing autonomous cars that reliably assist humans in everyday transportation is a grand research and engineering challenge. Mundane as it may seem, cruising on a multi-lane highway effectively and safely while taking full advantage of available driving space has proved challenging for existing autonomous cars. What makes multi-lane cruising significantly more challenging than the single-lane adaptive cruise control (ACC) is the fact that the multi-vehicle interaction happens both laterally (i.e. perpendicular to the lanes) and longitudinally (i.e. parallel to the lanes) and requires coordination between lateral and speed control.

The current research focuses on multi-lane cruising as a special case of driving on structured roads in general. Different from classical feedback control scenarios, driving on structured roads is heavily regulated by signs, signals, and rules that come to apply at various points in time, space, or even history (e.g. all-way stop intersections). In the case of multi-lane cruising, lane markings dictate that driving takes place mostly within the boundaries of a single lane. Lane change is thus a short-lived, transitional event in continuous motion space that links two distinct states — driving in one lane vs. driving in an adjacent lane. Furthermore, because driving on structured roads in general is thus *symbolically punctuated*[1], it is *naturally hierarchical* — higher level decisions on discrete state transitions are coordinated with lower level motion control in continuous state space. Because multi-lane cruising shares this hierarchical character, we propose using a hierarchical design for autonomous driving systems (similar to [16]), trusting that lessons we learn from this case can be generalized to other aspects of autonomous driving on structured roads. For this research, we take multi-lane

[1]The symbols here could include traffic lights, lane markings, speed limit signs, fire truck sirens, signals of other vehicles, and so on.

Submitted to 32nd Conference on Neural Information Processing Systems (NIPS 2018). Do not distribute.

cruising problem, requiring (a) lane changes, (b) in-lane maneuvers, and (c) speed control in order to maintain desirable speed, avoid obstacles, and stay safe. Furthermore, we have operationalized these essential aspects of the multi-lane cruising task in a small video game called *SimpleTraffic*[2], which follows the OpenAI Gyms API conventions [2] and is conducive to reinforcement learning research.

Autonomous driving systems typically consist of modules for perception and prediction, localization and mapping, and planning and control. Our study focuses on the planning and control part, where a hierarchy of behavior decision, motion planning, and control is widely adopted [10]. Classical methods for implementing the behavior planner are largely rule-based with finite state machines being a common choice [10]. Classical motion planner methods typically require optimization according to explicitly defined cost functions with the behavior decision expressed as constraint terms in the cost function [4, 18]. While rule-based behavior planner is extremely hard to maintain and does not scale well in complex dynamic scenarios, explicit cost functions for motion planner are hardly general enough and very difficult to tune for complex dynamic interactions. To avoid the limitations of traditional planning solutions, many recent studies attempted learning-based approaches. Bojarski et al. [1] proposed an end-to-end supervised learning scheme that directly maps images to steering commands. Sun et al. [17] in contrast use a mapping from state features to trajectories and then use an execution layer to further guarantee short term feasibility and safety. These approaches leverage expert experience for training. However, by directly cloning the expert's driving strategy, they are limited to the expert's performance and experience, failing to adequately explore the parts of the state-action space that may be less critical for safety and performance. In addition, planning and control is largely implemented as one monolithic network, which makes debugging, failure analysis, and incorporation of domain knowledge all very difficult.

With the recent great success of reinforcement learning (RL) methods in decision making and control (*e.g.* [3, 8, 19]), RL methods are now frequently applied to autonomous driving tasks [6, 12, 13]. In such cases, RL agents learn driving tasks by exploring typically simulated driving environments while attempting to improve performance along some metrics such as safety, average speed, smoothness of maneuvers, and smoothness of speed profile. The hope here is that such RL agents will outperform classical solutions through scaling significantly better for complex dynamic scenarios, and thereby making competent autonomous driving available much more broadly.

In the following, we aim to show that our design allows effective integration of RL for autonomous driving, by (1) eliminating the dependency on explicit cost functions in motion planning, (2) enabling significantly faster learning performance than a non-hierarchical or flat design, (3) allowing principled modular extension of motion planning in contrast to using either monolithic behavior cloning or a large set of hand-written rules. This in turn suggest that our proposed hierarchical architecture is a promising way to allow reinforcement learning to be applied to complex multi-lane cruising in the real world, which we demonstrate through transferring the trained models *without retraining* from a simulated environment with virtually no dynamics to one with significantly more realistic dynamics.

To summarize, the main contributions of this paper are: (1) proposing a modularized skill-based planning framework with two layers of hierarchy (behavioral and motion planner) for cruising in multi-lane roads; (2) proposing a higher level of abstraction in the state-action space of driving in multi-lane roads; and (3) introducing an open source simulator, *SimpleTraffic*, for RL and autonomous driving research community. In Section II, we review the related state-of-the-art methods in autonomous driving. In Section III, we present the details of our planning framework. Section IV introduces *SimpleTraffic* simulator. In Section V we evaluate our approach comprehensively and conclude our work in Section VI.

2 Related works

Recent studies have utilized RL for higher level decision making [7, 9]. Mukadam et al. [9] trained a Q-network to issue only high level commands, *e.g.* switch left/right, while the execution and collision avoidance rely on the low-level rule-based controller. Mirchevska *et al.* [7] proposed an RL-based approach for autonomous driving in highway scenarios using the fitted Q-iteration with extremely randomized trees as a function approximator. Both of these approaches have utilized RL for high level decision making (BP in our case) only and adopted hand-crafted rule-based approaches for MoP [7, 9] and collision avoidance [9]. In contrast, in addition to behavioral planner, motion planner and collision avoidance are also handled by our hierarchical learning-based approach which

[2]To be open sourced upon the publication of this paper.

is more generalizable compared to its rule-based counterparts. Shalev-Shwartz *et al.* [15] proposed to decompose the autonomous driving problem into two phases. First, they apply supervised learning for predicting the near future states based on the current states. Second, they model a full trajectory of the agent using a recurrent neural network, where unexplained factors are modeled as (additive) input nodes. As this method uses fully supervised learning approach, it relies on the training data too much. Paxton *et al.* [11] learns both low-level control policies and high level task policies by integrating a learning-based approach (Monte Carlo Tree Search) with a heuristic search algorithm to achieve a complex task. Their learned low-level policies sometimes can have oscillatory behavior that makes the car unstable. Given well established controllers such as PID and MPC, we believe that learning-based methods are more effective in the high and mid levels (*e.g.* BP and MoP) of a decision making rather than in the low-level controllers.

3 Technical approach

3.1 The Planning Hierarchy

Driving is a symbolically punctuated behavior. Different from classical feedback control scenarios, driving is heavily punctuated by signs and rules on top of what is largely a continuous control task. The symbols here include lane markings, traffic lights, speed limit signs, fire truck sirens, turning signals of the heading vehicle, and so on. As an example, lane markings dictates that most driving happen within a single lane. Lane changes are thus short-lived, transitional events that link forward driving in one lane to forward driving in an adjacent lane – two discrete states at a higher level of abstraction in the state space of driving. Because driving is thus symbolically punctuated, it is naturally hierarchical: higher level decisions on discrete state transitions with lower level execution in continuous state space, which suggests a hierarchical structure in the design of planning systems for autonomous driving. Figure 1 illustrates our proposed hierarchical decision making architecture for cruise in multi-lane roads. As shown in Figure 1, the proposed decision making framework includes *behavioral planner* that makes high level decisions about transitions between discrete states, and *motion planner* that generates a target spatio-temporal trajectory with a target speed according to the decisions made by BP. The target trajectory is then fed to the controller to follow the trajectory by controlling the steering wheel, throttle, and brake in continuous state space. The hierarchical structure of our planning framework facilitates analysis of the decisions that are made during driving. In addition, structured layers allows for convenience for modularizing different *skills*, *e.g.* adaptive cruise control, lane switching, etc. Each modularized skill acts as an independent entity and forms a comprehensive maneuver function considering its own constraints and safety internally. This also enables modifing and replacing sub-modules according to new requirements and conditions.

3.2 Behavior Planner

Behavior decision is about transitioning between states that are discrete only at a higher level of abstraction. BP is responsible to drive the car to the destination safely and as fast as possible. In our current setting, BP makes high level decisions including *keep lane*, *switch to the left lane*, and *switch to the right lane* subject to the following conditions: (1) following the mission planner (*drive to the destination*); (2) navigating the ego-car to less busy lanes so the car can drive to the maximum speed limit (*drive as fast as possible*); and (3) avoiding collisions (*drive safely*).

BP takes the full set of states as input which includes:, target lane suggested by Mission planner (MiP), current lane, current speed, speed and distance to the nearest heading car for current and neighboring lanes, and speed and distance to nearest car in the back for current and neighboring lanes. We design a *coarse-grained* reward function and avoid any *fine-grained* rules in our reward feedback. This way, we give the RL agent a chance to explore the state space and to come up with solutions that possibly outperform classical rule-based planners. We design the reward for BP in a way that it gets more reward if the car drives closer to the lane specified by the mission planner and closer to the speed limit and it gets a large negative reward if it collides with an obstacle.

3.3 Motion Planner

Motion planner's main task is to provide a safe and collision-free path towards its destination, while taking into account road boundaries, the vehicle dynamics, its maneuver capabilities in the presence of obstacles along with traffic rules or other constraints dictated by BP. In our design, motion planner generates a target trajectory defined as a set of 2D points (*i.e.* path) coupled with a target *speed* value. We propose a new level of road abstraction, through which each lane consists of N_c corridors, as depicted in Figure 2. Corridors are defined in the Frenét coordinate frame along center lines provided

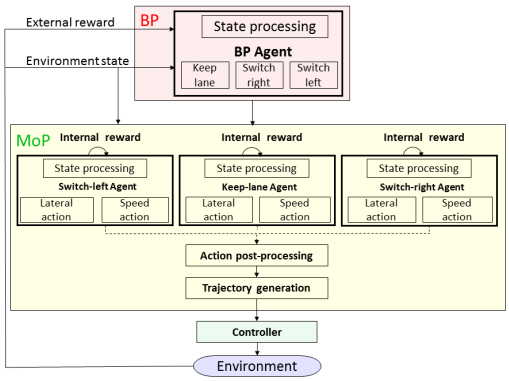

Figure 1: Overview of our hierarchical planning framework: Behavioral planner (BP) and Motion planners (MoP).

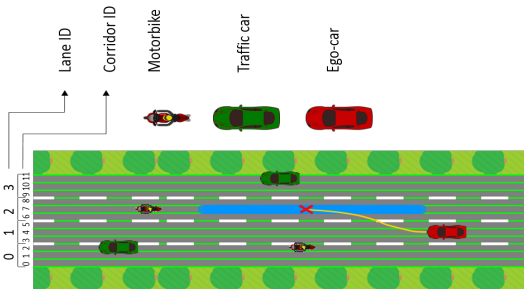

Figure 2: Corridor abstraction for structured roads. Here *Green lines* separate corridors, *Blue line* is the corridor selected by MoP, and *Yellow line* is the generated trajectory corresponding to the selected corridor (Red car is the ego-car).

by mission planner, constructed based on road centers (waypoints) or path planning algorithms for unstructred environments. As corridors are defined in the Frenét coordinate frame, tracking performance remains invariant to transformation [18]. As such, motion planner can be considered to be agnostic to lanes, intersections, and other road structures. As a result, there is no difference between straight roads, curvy roads, and roundabout. This road abstraction reduces the the continuous trajectory generation problem to the discrete corridor selection.

An MoP agent in our framework selects two sets of actions: 1) A *lateral action* identifying the target corridor; and 2) A *speed action* which selects the target speed. Corridor selection is equivalent to selecting a path among a set of predefined paths (clothoids or splines) along the mission planner routes. An MoP agent chooses the best target corridor based on the behavioral planner decisions: **BP == Keep lane**: MoP_{keep_lane} should select a corridor within the current lane while avoiding any collision. This enables MoP to maneuver around small objects in the lane without switching lane; and **BP == Switch left/right**: $MoP_{switch\_left/right}$ should select a corridor in the adjacent lanes. We included the nearest corridor to the target lane to MoP choices to consider prepare-for-merge action. Figure 2 illustrates corridor selection by an MoP agent.

MoP agents also select target speed based on BP actions and physical boundaries (*e.g.* heading cars, or any interfering object on the road) while ensuring safety and smoothness. By selecting a target speed, cruise in lane, emergency stop, vehicle following, and other BP-speed dependent actions can be handled. By adjusting the target speed, MoP agents can avoid collision while keeping the lane or switching to adjacent lanes. The target corridor and speed that are selected by MoP are relative quantities. The absolute corridor and speed are calculated in the *Action post-processing* module in Figure 1 and fed into the *Trajectory generator* module. Trajectory generator is a non-learning-based module implemented using a simple curve fitting from point A to point B (Yellow line in Figure 2). The generated trajectory is extended along the target corridor as the vehicle moves and is guaranteed to be smooth and feasible for the controller node.

MoP must also be a function of the complete set of environment states because it must always be safe no matter what the current BP decision is. MoP input states include: current lane and corridor, current speed, speed and distance to heading cars for each corridor in current and neighboring lanes, and speed and distance to cars in the back for each corridor in current and neighboring lanes. Similar to BP reward function, MoP reward function also need to avoid using any *fine-grained* rules and should remain *coarse-grained*. A *Keep lane* agent is responsible for adaptive cruise control (ACC). Keep lane agent is responsible to remain in the current lane, maintain the distance to the heading car and the car in the back (if it is too close to the ego-car), and maneuver around small obstacles within the current lane. The reward for MoP_{switch_left} and MoP_{switch_right} was designed such that the agent gets positive reward as it gets closer to the lane specified by BP. The reward function for MoP_{keep_lane} is designed to encourage the agent to keep a *desired distance* from the heading cars

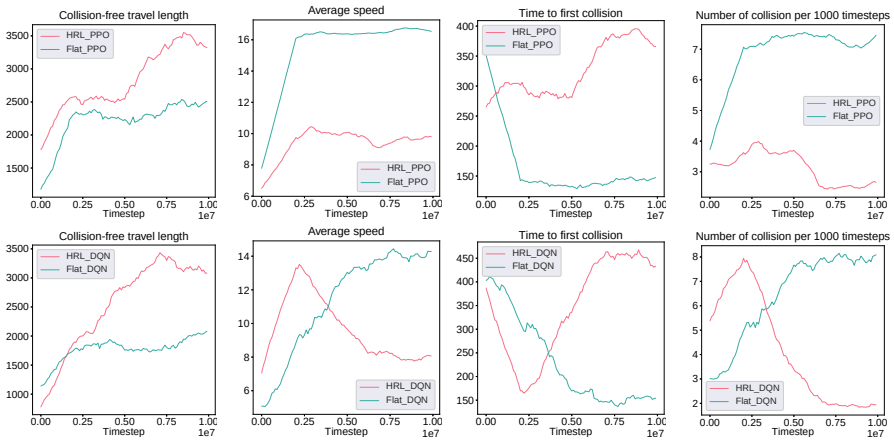

Figure 3: HRL vs. Flat agent. Top row: PPO agent. Bottom row: DQN agent.

if there exist any heading car or maintain safe front and back distance when the car in the back is too close to the vehicle. All MoP agents get penalized with a negative value if they collide with obstacles.

3.4 Training

In the proposed hierarchical RL framework, BP issues a high level command which is executed by the corresponding MoP. As opposed to the other hierarchical frameworks (*e.g.* [5]), BP does not wait until its command gets executed. Considering any fixed lifetime for BP commands is dangerous for autonomous driving. In fact, BP should be able to update its earlier decisions (at every time step) according to the new states. MoP is designed to prioritize safety over BP decisions. Our framework is flexible when it comes to choosing RL algorithms to be applied for BP and MoPs. We tested our framework with *DQN* [8], and *PPO* [14]. Potentially, we can train the HRL agent in two phases. In the first phase, we initialize the exploration of BP and MoP agents to 1, and train BP coupled with a rule-based MoP, and MoP coupled with a rule-base BP. This leads to pre-training BP and MoPs. In the second phase, we may jointly train BP and MoPs. However, in this study, we only present the results corresponding to the first phase.

4 Simulator

We developed a simulator, called *SimpleTraffic*, for cruising in multi-lane roads. *SimpleTraffic* has been developed in Python and is compatible with OpenAI Gym environments. We will open source *SimpleTraffic* for research community under GPL[3]. Concepts such as lane and corridor as well as ego-car and traffic obstacles (*e.g.* other cars or motorbikes) have been implemented in the simulator and can be customized for different scenarios. A user can set the number of lanes, number of corridors per lane, maximum and minimum speed of the ego-car and traffic obstacles. Currently, two sets of traffic obstacles have been implemented in *SimpleTraffic*: cars and motorbikes. In the default setting, each car occupies three corridors and each motorbike occupies one corridor only. Having these two sets of obstacles enables agent to distinguish between small objects that can be bypassed without changing the lane and larger objects that can only be bypassed by a lane change.

SimpleTraffic uses discrete time steps. Therefore, neither the speed of computers nor a slow architecture (*e.g.* huge neural network) influences the result. This way, researchers can debug their RL systems easier and more effectively. At every time step, the *SimpleTraffic* environment expects an action in the form of a tuple $(corridor, speed)$ where $corridor$ specifies the corridor index, Figure 2, and $speed$ specifies the target speed. After executing the action *SimpleTraffic* returns the next state, reward, and a flag indicating whether a terminal state is reached. The simulator states include: total number of lanes and corridors, current ego-car lane and corridor id, current ego-car speed, current ego-car lateral position, target lateral position (specified by the previous action), distance to heading cars and cars in the back at each corridor, and speed of heading cars and cars in the back at each corridor. A trajectory generator module has been developed inside the simulator that generates a smooth trajectory from the current position of the ego-car to the selected corridor.

[3]To be released on `http://www.github.com`

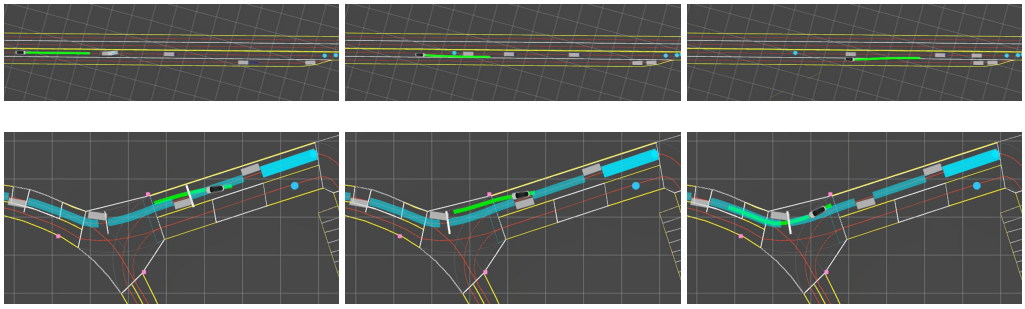

Figure 4: Corridor selection of the transferred model in a new environment with Lincoln MKZ simulated dynamic model.

5 Experiments

We tested our framework with both Q-learning-based and policy-gradient-based techniques and chose popular algorithms from each of these categories, namely DQN and PPO, respectively.

***SimpleTraffic*:** We trained both HRL and flat agents for 10 million time steps and deployed them on *SimpleTraffic* simulator. During evaluation, each model was deployed on the simulator 10 times for 1000 time steps and recording the *average speed*, the *time to first collision*, and the *number of collisions per* 1000 *steps*. During our experiments, we noticed that some agents lean towards driving too slowly to avoid any collision. This is not a desirable behavior, especially in a highway. On the other hand, other agents drive more aggressively (*i.e.* too fast) to maximize the speed reward, which may result in higher tendency to crash into other vehicles. In such case considering *average speed* and *time to first collision* separately is not fully informative. However, combining them can be more informative. Therefore, we define another metric we call *collision-free travel length* which is a multiplication of time to first collision and average speed. Figure 3 compares the performance of the HRL agent with the flat agent over time for PPO and DQN algorithms. These figures illustrate the aforementioned metrics over time; from initial agent (not trained) to trained agent (after 10 million time steps). Figures 3(a) and 3(b) confirm that both HRL and flat agents have learned to maximize the collision-free travel length, however, HRL agents managed to obtained better results by controlling their speed and sacrificing speed reward in order to drive safer. We emphasize that the same rewards were used for both HRL and flat agents.

Transfer to real vehicles: In section III we introduced our road abstraction and state representation for an RL agent that not only facilitates the learning procedure, but also helps to transfer the models to similar environment including real vehicle. As a proof of concept, we tested the transferability of the trained models by deploying them on a new ROS environment and using a Lincoln MKZ 2015 model as the vehicle dynamic model. Figure 4 illustrates few snapshots of the deployed model on the new environment. Our trained agents managed to successfully bypass static and dynamic obstacles by choosing appropriate corridors and cruise safely in multi-lane roads.

6 Conclusion & future work

We proposed an RL-based hierarchical framework for autonomous multi-lane cruising. We introduced a key intermediate abstraction within the motion planner to discretize the state-action space according to high level behavioral decisions. Furthermore, we showed that the hierarchical design for an autonomous vehicle system enables significantly better learning performance than a non-hierarchical design, when using both a value-based method (DQN) and a policy optimization method (PPO). The proposed framework allows for principled modular extension of motion planning, which is not the case in rule-based or monolithic behavior cloning-based approaches. Moreover, we experimentally showed that our state-action space abstraction allows transferring of the trained models from a simulated environment with virtually no dynamics to the one with significantly more realistic dynamics without a need for retraining. We also introduced our open source OpenAI Gym compatible simulator, called *SimpleTraffic*, to the reinforcement learning research community. Although training BP and MoP individually could sufficiently address the cruising in multi-lane problem, as our future work, we aim to train the BP and MoP agents jointly (in an end-to-end fashion) to acquire higher level of performance.

References

[1] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[2] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[3] Yu Fan Chen, Miao Liu, Michael Everett, and Jonathan P How. Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 285–292. IEEE, 2017.

[4] Xuemin Hu, Long Chen, Bo Tang, Dongpu Cao, and Haibo He. Dynamic path planning for autonomous driving on various roads with avoidance of static and moving obstacles. *Mechanical Systems and Signal Processing*, 100:482–500, 2018.

[5] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in neural information processing systems*, pages 3675–3683, 2016.

[6] Richard Liaw, Sanjay Krishnan, Animesh Garg, Daniel Crankshaw, Joseph E Gonzalez, and Ken Goldberg. Composing meta-policies for autonomous driving using hierarchical deep reinforcement learning. *arXiv preprint arXiv:1711.01503*, 2017.

[7] Branka Mirchevska, Manuel Blum, Lawrence Louis, Joschka Boedecker, and Moritz Werling. Reinforcement learning for autonomous maneuvering in highway scenarios.

[8] 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, 2015.

[9] Mustafa Mukadam, Akansel Cosgun, Alireza Nakhaei, and Kikuo Fujimura. Tactical decision making for lane changing with deep reinforcement learning. 2017.

[10] Brian Paden, Michal Čáp, Sze Zheng Yong, Dmitry Yershov, and Emilio Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on intelligent vehicles*, 1(1):33–55, 2016.

[11] Chris Paxton, Vasumathi Raman, Gregory D Hager, and Marin Kobilarov. Combining neural networks and tree search for task and motion planning in challenging environments. *arXiv preprint arXiv:1703.07887*, 2017.

[12] Etienne Perot, Maximilian Jaritz, Marin Toromanoff, and Raoul De Charette. End-to-end driving in a realistic racing game with deep reinforcement learning. In *International conference on Computer Vision and Pattern Recognition-Workshop*, 2017.

[13] Ahmad EL Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 2017(19):70–76, 2017.

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

[15] Shai Shalev-Shwartz, Nir Ben-Zrihem, Aviad Cohen, and Amnon Shashua. Long-term planning by short-term prediction. *arXiv preprint arXiv:1602.01580*, 2016.

[16] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. On a formal model of safe and scalable self-driving cars. *arXiv preprint arXiv:1708.06374*, 2017.

[17] Liting Sun, Cheng Peng, Wei Zhan, and Masayoshi Tomizuka. A fast integrated planning and control framework for autonomous driving via imitation learning. *arXiv preprint arXiv:1707.02515*, 2017.

[18] Moritz Werling, Julius Ziegler, Sören Kammel, and Sebastian Thrun. Optimal trajectory generation for dynamic street scenarios in a Frenet frame. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 987–993. IEEE, 2010.

[19] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3357–3364. IEEE, 2017.