ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/339015776

Deep Reinforcement Learning for Autonomous Driving: A Survey

Preprint · February 2020

CITATIONS
0

READS
485

7 authors, including:

Bangalore Ravi Kiran
Navya
45 PUBLICATIONS **419** CITATIONS
SEE PROFILE

Ibrahim Sobh
Valeo
14 PUBLICATIONS **69** CITATIONS
SEE PROFILE

Victor Talpaert
ENSTA ParisTech
7 PUBLICATIONS **24** CITATIONS
SEE PROFILE

Patrick Mannion
National University of Ireland, Galway
47 PUBLICATIONS **355** CITATIONS
SEE PROFILE

Some of the authors of this publication are also working on these related projects:

DATMO-CARLA View project

Tessellations and Partitions View project

All content following this page was uploaded by Bangalore Ravi Kiran on 15 February 2020.

The user has requested enhancement of the downloaded file.

Deep Reinforcement Learning for Autonomous Driving: A Survey

B Ravi Kiran[1], Ibrahim Sobh[2], Victor Talpaert[3], Patrick Mannion[4], Ahmad A. Al Sallab[2], Senthil Yogamani[5], Patrick Pérez[6]

arXiv:2002.00444v1 [cs.LG] 2 Feb 2020

***Abstract*—With the development of deep representation learning, the domain of reinforcement learning (RL) has become a powerful learning framework now capable of learning complex policies in high dimensional environments. This review summarises deep reinforcement learning (DRL) algorithms, provides a taxonomy of automated driving tasks where (D)RL methods have been employed, highlights the key challenges algorithmically as well as in terms of deployment of real world autonomous driving agents, the role of simulators in training agents, and finally methods to evaluate, test and robustifying existing solutions in RL and imitation learning.**

***Index Terms*—Deep reinforcement learning, Autonomous driving, Imitation learning, Inverse reinforcement learning, Controller learning, Trajectory optimisation, Motion planning, Safe reinforcement learning.**

I. Introduction

Autonomous driving (AD)[1] systems constitute of multiple perception level tasks that have now achieved high precision on account of deep learning architectures. Besides the perception, autonomous driving systems constitute of multiple tasks where classical supervised learning methods are no more applicable. First, when the prediction of the agent's action changes future sensor observations received from the environment under which the autonomous driving agent operates, for example the task of optimal driving speed in an urban area. Second, supervisory signals such as time to collision (TTC), lateral error w.r.t to optimal trajectory of the agent, represent the dynamics of the agent, as well uncertainty in the environment. Such problems would require defining the stochastic cost function to be maximized. Third, the agent is required to learn new configurations of the environment, as well as to predict an optimal decision at each instant while driving in its environment. This represents a high dimensional space given the number of unique configurations under which the agent & environment are observed, this is combinatorially large. In all such scenarios we are aiming to solve a sequential decision process, which is formalized under the classical settings of Reinforcement Learning (RL), where the agent is required to learn and represent its environment as well as act optimally given at each instant [1]. The optimal action is referred to as the policy.

In this review we shall cover the notions of reinforcement learning, the taxonomy of tasks where RL is a promising solution especially in the domains of driving policy, predictive perception, path and motion planning, and low level controller design. We also focus our review on the different real world deployments of RL in the domain of autonomous driving expanding our conference paper [2] since their deployment has not been reviewed in an academic setting. Finally, we motivate users by demonstrating the key computational challenges and risks when applying current day RL algorithms such imitation learning, deep Q learning, among others.

The rest of the paper is organized as follows. Section II provides an overview of components of a typical autonomous driving system. Section III provides an introduction to reinforcement learning and briefly discusses key concepts. Section IV discusses more sophisticated extensions on top of the basic RL framework. Section V provides an overview of RL applications for autonomous driving problems. Section VI discusses challenges in deploying RL for real-world autonomous driving systems. Section VII concludes this paper with some final remarks.

II. Components of AD System

Figure 1 comprises of the standard blocks of an AD system demonstrating the pipeline from sensor stream to control actuation. The sensor architecture in a modern autonomous driving system notably includes multiple sets of cameras, radars and LIDARs as well as a GPS-GNSS system for absolute localisation and inertial measurement Units (IMUs) that provide 3D pose of the vehicle in space.

The goal of the perception module is the creation of an intermediate level representation of the environment state (for example bird-eye view map of all obstacles and agents) that is to be later utilised by a decision making system that ultimately produces the driving policy. This state would include lane position, drivable zone, location of agents such as cars & pedestrians, state of traffic lights and others. Uncertainties in the perception propagate to the rest of the information chain. Robust sensing is critical for safety thus using redundant sources increases confidence in detection. This is achieved by a combination of several perception tasks like semantic segmentation [3], [4], motion estimation [5], depth estimation [6], soiling detection [7], etc which can be efficiently unified into a multi-task model [8], [9].

[1]Navya, Paris. ✉ ravi.kiran@navya.tech
[2]Valeo Cairo AI team, Egypt. ✉ ibrahim.sobh@valeo.com
[3]U2IS, ENSTA Paris & AKKA Technologies. ✉ victor.talpaert@ensta.fr
[4]National University of Ireland, Galway. ✉ patrick.mannion@nuigalway.ie
[5]Valeo Vision Systems. ✉ senthil.yogamani@valeo.com
[6]Valeo.ai. ✉ patrick.perez@valeo.com

[1]For easy reference, the main acronyms used in this article are listed in Appendix (Tables III and IV).

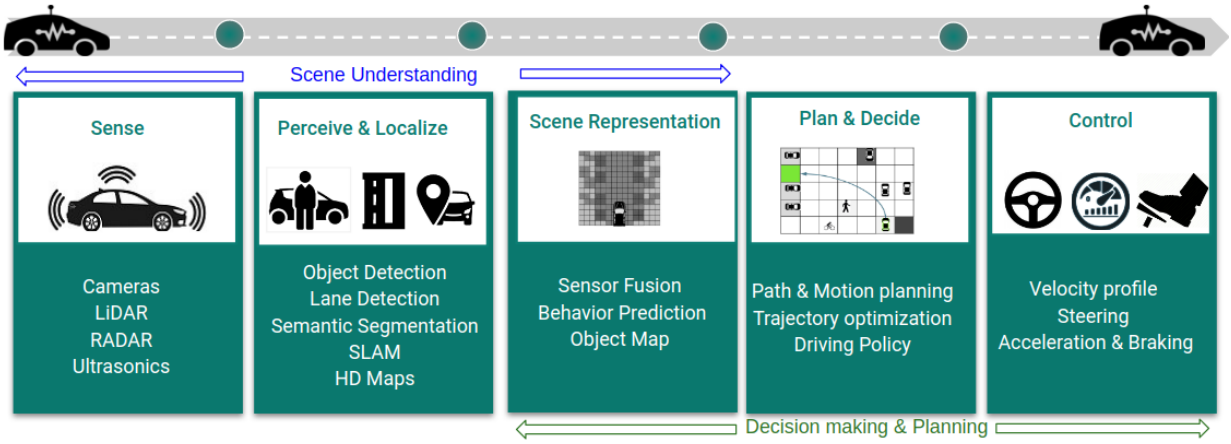

Fig. 1. Standard components in a modern autonomous driving systems pipeline listing the various tasks. The key problems addressed by these modules are Scene Understanding, Decision and Planning.

A. Scene Understanding

This key module maps the abstract mid-level representation of the perception state obtained from the perception module to the high level action or decision making module. Conceptually, three tasks are grouped by this module: Scene understanding, Decision and Planning as seen in figure 1 module aims to provide a higher level understanding of the scene, it is built on top of the algorithmic tasks of detection or localisation. By fusing heterogeneous sensor sources, it aims to robustly generalise to situations as the content becomes more abstract. This information fusion provides a general and simplified context for the Decision making components.

Fusion provides a sensor agnostic representation of the environment and models the sensor noise and detection uncertainties across multiple modalities such as LIDAR, camera, radar, ultra-sound. This basically requires weighting the predictions in a principled way.

B. Localization and Mapping

Mapping is one of the key pillars of automated driving [10]. Once an area is mapped, current position of the vehicle can be localized within the map. The first reliable demonstrations of automated driving by Google were primarily reliant on localisation to pre-mapped areas. Because of the scale of the problem, traditional mapping techniques are augmented by semantic object detection for reliable disambiguation. In addition, localised high definition maps (HD maps) can be used as a prior for object detection.

C. Planning and Driving policy

Trajectory planning is a crucial module in the autonomous driving pipeline. Given a route-level plan from HD maps or GPS based maps, this module is required to generate motion-level commands that steer the agent.

Classical motion planning ignores dynamics and differential constraints while using translations and rotations required to move an agent from source to destination poses [11]. A robotic agent capable of controlling 6-degrees of freedom (DOF) is said to be holonomic, while an agent with fewer controllable DOFs than its total DOF is said to be non-holonomic. Classical algorithms such as A^* algorithm based on Djisktra's algorithm do not work in the non-holonomic case for autonomous driving. Rapidly-exploring random trees (RRT) [12] are non-holonomic algorithms that explore the configuration space by random sampling and obstacle free path generation. There are various versions of RRT currently used in for motion planning in autonomous driving pipelines.

D. Control

A controller defines the speed, steering angle and braking actions necessary over every point in the path obtained from a pre-determined map such as Google maps, or expert driving recording of the same values at every waypoint. Trajectory tracking in contrast involves a temporal model of the dynamics of the vehicle viewing the waypoints sequentially over time.

Current vehicle control methods are founded in classical optimal control theory which can be stated as a minimisation of a cost function $\dot{x} = f(x(t), u(t))$ defined over a set of states $x(t)$ and control actions $u(t)$. The control input is usually defined over a finite time horizon and restricted on a feasible state space $x \in X_{\text{free}}$ [13]. The velocity control are based on classical methods of closed loop control such as PID (proportional-integral-derivative) controllers, MPC (Model predictive control). PIDs aim to minimise a cost function constituting of three terms current error with proportional term, effect of past errors with integral term, and effect of future errors with the derivative term. While the family of MPC methods aim to stabilize the behavior of the vehicle while tracking the specified path [14]. A review on controllers, motion planning and learning based approaches for the same are provided in this review [15] for interested readers. Optimal control and reinforcement learning are intimately related, where optimal control can be viewed as a model based reinforcement learning problem where the dynamics of the vehicle/environment are modeled by well defined differential equations. Reinforcement learning methods were developed to handle stochastic control

problems as well ill-posed problems with unknown rewards and state transition probabilities.

III. Reinforcement Learning

Machine learning (ML) is a process whereby a computer program learns from experience to improve its performance at a specified task [16]. ML algorithms are often classified under one of three broad categories: supervised learning, unsupervised learning and reinforcement learning (RL). Supervised learning algorithms are based on inductive inference where the model is typically trained using labelled data to perform classification or regression, whereas unsupervised learning encompasses techniques such as density estimation or clustering applied to unlabelled data. By contrast, in the RL paradigm an autonomous agent learns to improve its performance at an assigned task by interacting with its environment. Russel and Norvig define an agent as "anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators" [17]. RL agents are not told explicitly how to act by an expert; rather an agent's performance is evaluated by a reward function R. For each state experienced, the agent chooses an action and receives an occasional reward from its environment based on the usefulness of its decision. The goal for the agent is to maximize the cumulative rewards received over its lifetime. Gradually, the agent can increase its long-term reward by exploiting knowledge learned about the expected utility (i.e. discounted sum of expected future rewards) of different state-action pairs. One of the main challenges in reinforcement learning is managing the trade-off between exploration and exploitation. To maximize the rewards it receives, an agent must exploit its knowledge by selecting actions which are known to result in high rewards. On the other hand, to discover such beneficial actions, it has to take the risk of trying new actions which may lead to higher rewards than the current best-valued actions for each system state. In other words, the learning agent has to exploit what it already knows in order to obtain rewards, but it also has to explore the unknown in order to make better action selections in the future. Examples of strategies which have been proposed to manage this trade-off include ϵ-greedy and softmax. When adopting the ubiquitous ϵ-greedy strategy, an agent either selects an action at random with probability $0 < \epsilon < 1$, or greedily selects the highest valued action for the current state with the remaining probability $1-\epsilon$. Intuitively, the agent should explore more at the beginning of the training process when little is known about the problem environment. As training progresses, the agent may gradually conduct more exploitation than exploration. The design of exploration strategies for RL agents is an area of active research (see *e.g.* [18]).

Markov decision processes (MDPs) are considered the de facto standard when formalising sequential decision making problems involving a single RL agent [19]. An MDP consists of a set S of states, a set A of actions, a transition function T and a reward function R [20], i.e. a tuple $< S, A, T, R >$. When in any state $s \in S$, selecting an action $a \in A$ will result in the environment entering a new state $s' \in S$ with a transition

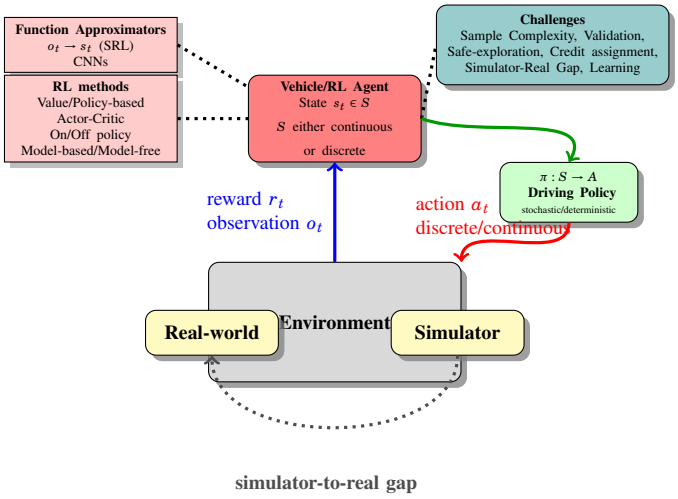

Fig. 2. A graphical decomposition of the different components of an RL algorithm. It also demonstrates the different challenges encountered while training a D(RL) algorithm.

probability $T(s,a,s') \in (0,1)$, and give a reward $R(s,a)$. This process is illustrated in Fig. 2. The stochastic policy $\pi : S \rightarrow \mathscr{D}$ is a mapping from the state space to a probability over the set of actions, and $\pi(a|s)$ represents the probability of choosing action a at state s. The goal is to find the optimal policy π^*, which results in the highest expected sum of discounted rewards [19]:

$$\pi^* = \underset{\pi}{\operatorname{argmax}} \underbrace{\mathbb{E}_\pi \left\{ \sum_{k=0}^{H-1} \gamma^k r_{k+1} \mid s_0 = s \right\}}_{:= V_\pi(s)}, \tag{1}$$

for all states $s \in S$, where $r_k = R(s_k, a_k)$ is the reward at time k and $V_\pi(s)$, the 'value function' at state s following a policy π, is the expected 'return' (or 'utility') when starting at s and following the policy π thereafter [1]. An important, related concept is the action-value function, a.k.a.'Q-function' defined as:

$$Q_\pi(s,a) = \mathbb{E}_\pi \left\{ \sum_{k=0}^{H-1} \gamma^k r_{k+1} \mid s_0 = s, a_0 = a \right\}. \tag{2}$$

The discount factor $\gamma \in [0,1]$ controls how an agent regards future rewards. Low values of γ encourage myopic behaviour where an agent will aim to maximise short term rewards, whereas high values of γ cause agents to be more forward-looking and to maximise rewards over a longer time frame. The horizon H refers to the number of time steps in the MDP. In infinite-horizon problems $H = \infty$, whereas in episodic domains H has a finite value. Episodic domains may terminate after a fixed number of time steps, or when an agent reaches a specified goal state. The last state reached in an episodic domain is referred to as the terminal state. In finite-horizon or goal-oriented domains discount factors of (close to) 1 may be used to encourage agents to focus on achieving the goal, whereas in infinite-horizon domains lower discount factors may be used to strike a balance between short- and long-term rewards. If the optimal policy for a MDP is known, then V_{π^*} may be used to determine the maximum expected discounted sum of rewards available from any arbitrary initial

state. A rollout is a trajectory produced in the state space by sequentially applying a policy to an initial state. A MDP satisfies the Markov property, i.e. system state transitions are dependent only on the most recent state and action, not on the full history of states and actions in the decision process. Moreover, in many real-world application domains, it is not possible for an agent to observe all features of the environment state; in such cases the decision-making problem is formulated as a partially-observable Markov decision process (POMDP). Solving a reinforcement learning task means finding a policy π that maximises the expected discounted sum of rewards over trajectories in the state space. RL agents may learn value function estimates, policies and/or environment models directly. Dynamic programming (DP) refers to a collection of algorithms that can be used to compute optimal policies given a perfect model of the environment in terms of reward and transition functions. Unlike DP, in Monte Carlo methods there is no assumption of complete environment knowledge. Monte Carlo methods are incremental in an episode-by-episode sense. Upon the completion of an episode, the value estimates and policies are updated. Temporal Difference (TD) methods, on the other hand, are incremental in a step-by-step sense, making them applicable to non-episodic scenarios. Like Monte Carlo methods, TD methods can learn directly from raw experience without a model of the environment's dynamics. Like DP, TD methods learn their estimates based on other estimates.

A. Value-based methods

Q-learning is one of the most commonly used RL algorithms. It is a model-free TD algorithm that learns estimates of the utility of individual state-action pairs (Q-functions defined in Eqn. 2). Q-learning has been shown to converge to the optimum state-action values for a MDP with probability 1, so long as all actions in all states are sampled infinitely often and the state-action values are represented discretely [21]. In practice, Q-learning will learn (near) optimal state-action values provided a sufficient number of samples are obtained for each state-action pair. If a Q-learning agent has converged to the optimal Q values for a MDP and selects actions greedily thereafter, it will receive the same expected sum of discounted rewards as calculated by the value function with π^* (assuming that the same arbitrary initial starting state is used for both). Agents implementing Q-learning update their Q values according to the following update rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a' \in A} Q(s',a') - Q(s,a)], \quad (3)$$

where $Q(s,a)$ is an estimate of the utility of selecting action a in state s, $\alpha \in [0,1]$ is the learning rate which controls the degree to which Q values are updated at each time step, and $\gamma \in [0,1]$ is the same discount factor used in Eqn. 1. The theoretical guarantees of Q-learning hold with any arbitrary initial Q values [21]; therefore the optimal Q values for a MDP can be learned by starting with any initial action value function estimate. The initialisation can be optimistic (each $Q(s,a)$ returns the maximum possible reward), pessimistic (minimum) or even using knowledge of the problem to ensure faster convergence. Deep Q-Networks (DQN) [22] incorporates a variant of the Q-learning algorithm [23], by using deep neural networks (DNNs) as a non-linear Q function approximator over high-dimensional state spaces (e.g. the pixels in a frame of an Atari game). Practically, the neural network predicts the value of all actions without the use of any explicit domain-specific information or hand-designed features. DQN applies experience replay technique to break the correlation between successive experience samples and also for better sample efficiency. For increased stability, two networks are used where the parameters of the target network for DQN are fixed for a number of iterations while updating the parameters of the online network. Readers are directed to sub-section III-E for a more detailed introduction to the use of DNNs in Deep RL.

B. Policy-based methods

The difference between value-based and policy-based methods is essentially a matter of where the burden of optimality resides. Both method types must propose actions and evaluate the resulting behaviour, but while value-based methods focus on evaluating the optimal cumulative reward and have a policy follows the recommendations, policy-based methods aim to estimate the optimal policy directly, and the value is a secondary if calculated at all. Typically, a policy is parameterised as a neural network π_θ. Policy gradient methods use gradient descent to estimate the parameters of the policy that maximise the expected reward. The result can be a stochastic policy where actions are selected by sampling, or a deterministic policy. Many real-world applications have continuous action spaces. Deterministic policy gradient (DPG) algorithms [24] [1] allow reinforcement learning in domains with continuous actions. Silver *et al.* [24] proved that a deterministic policy gradient exists for MDPs satisfying certain conditions, and that deterministic policy gradients have a simple model-free form that follows the gradient of the action-value function. As a result, instead of integrating over both state and action spaces in stochastic policy gradients, DPG integrates over the state space only leading to fewer samples in problems with large action spaces. To ensure sufficient exploration, actions are chosen using a stochastic policy, while learning a deterministic target policy. The REINFORCE [25] algorithm is a straight forward policy-based method. The discounted cumulative reward $g_t = \sum_{k=0}^{H-1} \gamma^k r_{k+t+1}$ at one time step is calculated by playing the entire episode, so no estimator is required for policy evaluation. The parameters are updated into the direction of the performance gradient:

$$\theta \leftarrow \theta + \alpha \gamma^t g \nabla \log \pi_\theta(a|s), \quad (4)$$

where α is the learning rate for a stable incremental update. Intuitively, we want to encourage state-action pairs that result in the best possible returns. Trust Region Policy Optimization (TRPO) [26], works by preventing the updated policies from deviating too much from previous policies, thus reducing the chance of a bad update. TRPO optimises a surrogate objective function where the basic idea is to limit each policy gradient update as measured by the Kullback-Leibler (KL) divergence between the current and the new proposed policy. This method results in monotonic improvements in policy performance.

While Proximal Policy Optimization (PPO) [27] proposed a clipped surrogate objective function by adding a penalty for having a too large policy change. Accordingly, PPO policy optimisation is simpler to implement, and has better sample complexity while ensuring the deviation from the previous policy is relatively small.

C. Actor-critic methods

Actor-critic methods are hybrid methods that combine the benefits of policy-based and value-based algorithms. The policy structure that is responsible for selecting actions is known as the 'actor'. The estimated value function criticises the actions made by the actor and is known as the 'critic'. After each action selection, the critic evaluates the new state to determine whether the result of the selected action was better or worse than expected. Both networks need their gradient to learn. Let $J(\theta) := \mathbb{E}_{\pi_\theta}[r]$ represent a policy objective function, where θ designates the parameters of a DNN. Policy gradient methods search for local maximum of $J(\theta)$. Since optimization in continuous action spaces could be costly and slow, the DPG (Direct Policy Gradient) algorithm represents actions as parameterised function $\mu(s|\theta^\mu)$, where θ^μ refers to the parameters of the actor network. Then the unbiased estimate of the policy gradient gradient step is given as:

$$\nabla_\theta J = -\mathbb{E}_{\pi_\theta}\Big\{(g-b)\log\pi_\theta(a|s)\Big\}, \tag{5}$$

where b is the baseline. While using $b \equiv 0$ is the simplification that leads to the REINFORCE formulation. Williams [25] explains a well chosen baseline can reduce variance leading to a more stable learning. The baseline, b can be chosen as $V_\pi(s)$, $Q_\pi(s,a)$ or 'Advantage' $A_\pi(s,a)$ based methods. Deep Deterministic Policy Gradient (DDPG) [28] is a model-free, off-policy (please refer to subsection III-D for a detailed distinction), actor-critic algorithm that can learn policies for continuous action spaces using deep neural net based function approximation, extending prior work on DPG to large and high-dimensional state-action spaces. When selecting actions, exploration is performed by adding noise to the actor policy. Like DQN, to stabilise learning a replay buffer is used to minimize data correlation. A separate actor-critic specific target network is also used. Normal Q-learning is adapted with a restricted number of discrete actions, and DDPG also needs a straightforward way to choose an action. Starting from Q-learning, we extend Eqn. 2 to define the optimal Q-value and optimal action as Q^* and a^*.

$$Q^*(s,a) = \max_\pi Q_\pi(s,a), \tag{6}$$

$$a^* = \operatorname*{argmax}_a Q^*(s,a). \tag{7}$$

In the case of Q-learning, the action is chosen according to the Q-function as in Eqn. 7. But DDPG chains the evaluation of Q after the action has already been chosen according to the policy. By correcting the Q-values towards the optimal values using the chosen action, we also update the policy towards the optimal action proposition. Thus two separate networks work at estimating Q^* and π^*.

Asynchronous Advantage Actor Critic (A3C) [29] uses asynchronous gradient descent for optimization of deep neural network controllers. Deep reinforcement learning algorithms based on experience replay such as DQN and DDPG have demonstrated considerable success in difficult domains such as playing Atari games. However, experience replay uses a large amount of memory to store experience samples and requires off-policy learning algorithms. In A3C, instead of using an experience replay buffer, agents asynchronously execute on multiple parallel instances of the environment. In addition to the reducing correlation of the experiences, the parallel actor-learners have a stabilizing effect on training process. This simple setup enables a much larger spectrum of on-policy as well as off-policy reinforcement learning algorithms to be applied robustly using deep neural networks. A3C exceeded the performance of the previous state-of-the-art at the time on the Atari domain while training for half the time on a single multi-core CPU instead of a GPU by combining several ideas. It also demonstrates how using an estimate of the value function as the previously explained baseline b reduces variance and improves convergence time. By defining the *advantage* as $A_\pi(a,s) = Q_\pi(s,a) - V_\pi(s)$, the expression of the policy gradient from Eqn. 5 is rewritten as $\nabla_\theta L = -\mathbb{E}_{\pi_\theta}\{A_\pi(a,s)\log\pi_\theta(a|s)\}$. The critic is trained to minimize $\frac{1}{2}\left\|A_{\pi_\theta}(a,s)\right\|^2$. The intuition of using advantage estimates rather than just discounted returns is to allow the agent to determine not just how good its actions were, but also how much better they turned out to be than expected, leading to reduced variance and more stable training. The A3C model also demonstrated good performance in 3D environments such as labyrinth exploration. Advantage Actor Critic (A2C) is a synchronous version of the asynchronous advantage actor critic model, that waits for each agent to finish its experience before conducting an update. The performance of both A2C and A3C is comparable. Most greedy policies must alternate between exploration and exploitation, and good exploration visits the states where the value estimate is uncertain. This way, exploration focuses on trying to find the most *uncertain* state paths as they bring valuable information. In addition to advantage, explained earlier, some methods use the entropy as the uncertainty quantity. Most A3C implementations include this as well. Two methods with common authors are energy-based policies [30] and more recent and with widespread use, the Soft Actor Critic (SAC) algorithm [31], both rely on adding an entropy term to the reward function, so we update the policy objective from Eqn. 1 to Eqn. 8. We refer readers to [31] for an in depth explanation of the expression

$$\pi^*_{MaxEnt} = \operatorname*{argmax}_\pi \mathbb{E}_\pi\Big\{\sum_t [r(s_t,a_t) + \alpha H(\pi(.|s_t))]\Big\}, \tag{8}$$

shown here for illustration of how the entropy H is added.

D. Model-based (vs. Model-free) & On/Off Policy methods

In practical situations, interacting with the real environment could be limited due to many reasons including safety and cost. Learning a model for environment dynamics may reduce the amount of interactions required with the real environment. Moreover, exploration can be performed on the learned

models. In the case of model-based approaches (*e.g.* Dyna-Q [32], R-max [33]), agents attempt to learn the transition function T and reward function R, which can be used when making action selections. Keeping a model approximation of the environment means storing knowledge of its dynamics, and allows for fewer, and sometimes, costly environment interactions. By contrast, in model-free approaches such knowledge is not a requirement. Instead, model-free learners sample the underlying MDP directly in order to gain knowledge about the unknown model, in the form of value function estimates for example. In Dyna-2 [34], the learning agent stores long-term and short-term memories, where a memory is defined as the set of features and corresponding parameters used by an agent to estimate the value function. Long-term memory is for general domain knowledge which is updated from real experience, while short-term memory is for specific local knowledge about the current situation, and the value function is a linear combination of long and short term memories.

Learning algorithms can be on-policy or off-policy depending on whether the updates are conducted on fresh trajectories generated by the policy or by another policy, that could be generated by an older version of the policy or provided by an expert. On-policy methods such as SARSA [35], estimate the value of a policy while using the same policy for control. However, off-policy methods such as Q-learning [23], use two policies: the behavior policy, the policy used to generate behavior; and the target policy, the one being improved on. An advantage of this separation is that the target policy may be deterministic (greedy), while the behavior policy can continue to sample all possible actions, [1].

E. Deep reinforcement learning

Tabular representations are the simplest way to store learned estimates (of *e.g.* values, policies or models), where each state-action pair has a discrete estimate associated with it. When estimates are represented discretely, each additional feature tracked in the state leads to an exponential growth in the number of state-action pair values that must be stored [36]. This problem is commonly referred to in the literature as the "curse of dimensionality", a term originally coined by Bellman [37]. In simple environments this is rarely an issue, but it may lead to an intractable problem in real-world applications, due to memory and/or computational constraints. Learning over a large state-action space is possible, but may take an unacceptably long time to learn useful policies. Many real-world domains feature continuous state and/or action spaces; these can be discretised in many cases. However, large discretisation steps may limit the achievable performance in a domain, whereas small discretisation steps may result in a large state-action space where obtaining a sufficient number of samples for each state-action pair is impractical. Alternatively, function approximation may be used to generalise across states and/or actions, whereby a function approximator is used to store and retrieve estimates. Function approximation is an active area of research in RL, offering a way to handle continuous state and/or action spaces, mitigate against the state-action space explosion and generalise prior experience to previously unseen state-action pairs. Tile coding is one of the simplest forms of function approximation, where one tile represents multiple states or state-action pairs [36]. Neural networks are also commonly used to implement function approximation, one of the most famous examples being Tesuaro's application of RL to backgammon [38]. Recent work has applied deep neural networks as a function approximation method; this emerging paradigm is known as deep reinforcement learning (DRL). DRL algorithms have achieved human level performance (or above) on complex tasks such as playing Atari games [22] and playing the board game Go [39].

In DQN [22] it is demonstrated how a convolutional neural network can learn successful control policies from just raw video data for different Atari environments. The network was trained end-to-end and was not provided with any game specific information. The input to the convolutional neural network consists of an $84 \times 84 \times 4$ where 4 consecutive frames are used to capture the temporal information. The first hidden layer consists of 32 filters of 8×8 with stride 4 and applies a rectifier non linearity. The second hidden layer consists of 64 filters of 4×4 with stride 2, followed by a rectifier non-linearity. This is followed by a third convolutional layer that of 64 filters of 3×3 with stride1 followed by a rectifier. The final hidden layer is fully connected and consists of 512 rectifier units. The output layer is a fully-connected linear layer with a single output for each valid action. For DQN training stability, two networks are used while the parameters of the target network are fixed for a number of iterations while updating the online network parameters. For practical reasons, the $Q(s,a)$ function is modeled as a deep neural network that predicts the value of all actions given the input state. Accordingly, deciding what action to take requires performing a single forward pass of the network. Moreover, in order to increase sample efficiency, experiences of the agent are stored in a replay memory (experience replay), where the Q-learning updates are conducted on randomly selected samples from the replay memory. This random selection breaks the correlation between successive samples. Experience replay enables reinforcement learning agents to *remember* and reuse experiences from the past where observed transitions are stored for some time, usually in a queue, and sampled uniformly from this memory to update the network. However, this approach simply replays transitions at the same frequency that they were originally experienced, regardless of their significance. An alternative method is to use two separate experience buckets, one for positive and one for negative rewards [40]. Then a fixed fraction from each bucket is selected to replay. This method is only applicable in domains that have a natural notion of binary experience. Experience replay has also been extended with a framework for prioritising experience [41], where important transitions, based on the TD error, are replayed more frequently, leading to improved performance and faster training when compared to the standard experience replay approach.

The max operator in standard Q-learning and DQN uses the same values both to select and to evaluate an action resulting in over optimistic value estimates. In Double DQN (D-DQN) [42] the over estimation problem in DQN is tackled where the

greedy policy is evaluated according to the online network and uses the target network to estimate its value. It was shown that this algorithm not only yields more accurate value estimates, but leads to much higher scores on several games.

In Dueling network architecture [43] the state value function and associated advantage function are estimated, and then combined together to estimate action value function. The advantage of the dueling architecture lies partly in its ability to learn the state-value function efficiently. In a single-stream architecture only the value for one of the actions is updated. However in dueling architecture, the value stream is updated with every update, allowing for better approximation of the state values, which in turn need to be accurate for temporal difference methods like Q-learning.

DRQN [44] applied a modification to the DQN by combining a Long Short Term Memory (LSTM) with a Deep Q-Network. Accordingly, the DRQN is capable of integrating information across frames to detect information such as velocity of objects. DRQN showed to generalize its policies in case of complete observations and when trained on Atari games and evaluated against flickering games, it was shown that DRQN generalizes better than DQN.

IV. Extensions to reinforcement learning

This section introduces and discusses some of the main extensions to the basic single-agent RL paradigms which have been introduced over the years. As well as broadening the applicability of RL algorithms, many of the extensions discussed here have been demonstrated to improve scalability, learning speed and/or converged performance in complex problem domains.

A. Reward shaping

As noted in Section III, the design of the reward function is crucial: RL agents seek to maximise the return from the reward function, therefore the optimal policy for a domain is defined with respect to the reward function. In many real-world application domains, learning may be difficult due to sparse and/or delayed rewards. RL agents typically learn how to act in their environment guided merely by the reward signal. Additional knowledge can be provided to a learner by the addition of a shaping reward to the reward naturally received from the environment, with the goal of improving learning speed and converged performance. This principle is referred to as reward shaping. The term shaping has its origins in the field of experimental psychology, and describes the idea of rewarding all behaviour that leads to the desired behaviour. Skinner [45] discovered while training a rat to push a lever that any movement in the direction of the lever had to be rewarded to encourage the rat to complete the task. Analogously to the rat, a RL agent may take an unacceptably long time to discover its goal when learning from delayed rewards, and shaping offers an opportunity to speed up the learning process. Reward shaping allows a reward function to be engineered in a way to provide more frequent feedback signal on appropriate behaviours [46], which is especially useful in domains with sparse rewards. Generally, the return from the reward function is modified as follows: $r' = r + f$ where r is the return from the original reward function R, f is the additional reward from a shaping function F, and r' is the signal given to the agent by the augmented reward function R'. Empirical evidence has shown that reward shaping can be a powerful tool to improve the learning speed of RL agents [47]. However, it can have unintended consequences. The implication of adding a shaping reward is that a policy which is optimal for the augmented reward function R' may not in fact also be optimal for the original reward function R. A classic example of reward shaping gone wrong for this exact reason is reported by [47] where the experimented bicycle agent would turn in circle to stay upright rather than reach its goal. Difference rewards (*D*) [48] and potential-based reward shaping (*PBRS*) [49] are two commonly used shaping approaches. Both *D* and *PBRS* have been successfully applied to a wide range of application domains and have the added benefit of convenient theoretical guarantees, meaning that they do not suffer from the same issues as the unprincipled reward shaping approaches described above (see *e.g.* [49]–[53]).

B. Multi-agent reinforcement learning

In multi-agent reinforcement learning (MARL), multiple RL agents are deployed into a common environment. The single-agent MDP framework becomes inadequate when multiple autonomous agents act simultaneously in the same domain. Instead, the more general stochastic game (SG) may be used in the case of a Multi-Agent System (MAS) [54]. A SG is defined as a tuple $< S, A_{1\ldots N}, T, R_{1\ldots N} >$, where N is the number of agents, S is the set of system states, A_i is the set of actions for agent i (and A is the joint action set), T is the transition function, and R_i is the reward function for agent i. The SG looks very similar to the MDP framework, apart from the addition of multiple agents. In fact, for the case of $N = 1$ a SG then becomes a MDP. The next system state and the rewards received by each agent depend on the joint action a of all of the agents in a SG, where a is derived from the combination of the individual actions a_i for each agent in the system. Each agent may have its own local state perception s_i, which is different to the system state s (i.e. individual agents are not assumed to have full observability of the system). Note also that each agent may receive a different reward for the same system state transition, as each agent has its own separate reward function R_i. In a SG, the agents may all have the same goal (collaborative SG), totally opposing goals (competitive SG), or there may be elements of collaboration and competition between agents (mixed SG). Whether RL agents in a MAS will learn to act together or at cross-purposes depends on the reward scheme used for a specific application.

C. Multi-objective reinforcement learning

In multi-objective reinforcement learning (MORL) the reward signal is a vector, where each component represents the performance on a different objective. The MORL framework was developed to handle sequential decision making problems where tradeoffs between conflicting objective functions must be considered. Examples of real-world problems with multiple

objectives include selecting energy sources (tradeoffs between fuel cost and emissions) [55] and watershed management (tradeoffs between generating electricity, preserving reservoir levels and supplying drinking water) [56]. Solutions to MORL problems are often evaluated using the concept of Pareto dominance [57] and MORL algorithms typically seek to learn or approximate the set of non-dominated solutions. MORL problems may be defined using the MDP or SG framework as appropriate, in a similar manner to single-objective problems. The main difference lies in the definition of the reward function: instead of returning a single scalar value r, the reward function $\mathbf{R}$ in multi-objective domains returns a vector $\mathbf{r}$ consisting of the rewards for each individual objective $c \in C$. Therefore, a regular MDP or SG can be extended to a Multi-Objective MDP (MOMDP) or Multi-Objective SG (MOSG) by modifying the return of the reward function. For a more complete overview of MORL beyond the brief summary presented in this section, the interested reader is referred to recent surveys [58], [59].

D. State Representation Learning

State Representation Learning (SRL) refers to feature extraction & dimensionality reduction to represent the state space with its history conditioned by the actions and environment of the agent. A complete review of SRL for control is discussed in [60]. In the simplest form SRL maps a high dimensional vector o_t into a small dimensional latent space s_t. The inverse operation decodes the state back into an estimate of the original observation $\hat{o}_t$. The agent then learns to map from the latent space to the action. Training the SRL chain is unsupervised in the sense that no labels are required. Reducing the dimension of the input effectively simplifies the task as it removes noise and decreases the domain's size as shown in [61]. SRL could be a simple auto-encoder (AE), though various methods exist for observation reconstruction such as Variational Auto-Encoders (VAE) or GANs, as well as forward models for predicting the next state or inverse models for predicting the action given a transition. A good learned state representation should be Markovian; i.e. it should encode all necessary information to be able to select an action based on the current state only, and not any previous states or actions [60], [62].

E. Curriculum Learning & Hierarchical RL

Better learning performance can be achieved when the examples are organised in a meaningful order which illustrates more concepts gradually. Authors in [63] formalised training strategies in the context of machine learning that show that improvements in generalisation and in speed of convergence of the training. However, even humans do not agree about the order in which concepts should be introduced. Curriculum learning can be seen as a special form of transfer learning where the initial tasks are used to guide the agent to perform better on the target task. Curriculum learning is related to active learning [64] where the learning agent can focus on the interesting examples that stand near frontier of its knowledge and abilities. In such case, the pace at which the agent would move along a the curriculum is automated. Contrary to conventional or flat RL, HRL refers to the decomposition of complex agent behavior using temporal abstraction, such as the options framework [65]. Readers are directed towards a complete and compact review on advances in hierarchical reinforcement learning [65].

F. Learning from Demonstrations

Learning from Demonstrations (LfD) is used by humans to acquire new skills in an expert to learner knowledge transmission process. LfD is important for initial exploration where reward signals are too sparse or the input domain is too large to cover. In LfD, an agent learns to perform a task from demonstrations, usually in the form of state-action pairs, provided by an expert without any feedback rewards. However, high quality and diverse demonstrations are hard to collect, leading to learning sub-optimal policies. Accordingly, learning merely from demonstrations can be used to initialize the learning agent with a good or safe policy, and then reinforcement learning can be conducted to enable the discovery of a better policy by interacting with the environment. Combining demonstrations and reinforcement learning has been conducted in recent research. AlphaGo [39], combines search tree with deep neural networks, initializes the policy network by supervised learning on state-action pairs provided by recorded games played by human experts. Additionally, a value network is trained to tell how desirable a board state is. By conducting self-play and reinforcement learning, AlphaGo is able to discover new stronger actions and learn from its mistakes, achieving super human performance. More recently, AlphaZero [66], developed by the same team, proposed a general framework for self-play models. AlphaZero is trained entirely using reinforcement learning and self play, starting from completely random play, and requires no prior knowledge of human players. AlphaZero taught itself from scratch how to master the games of chess, shogi, and Go game, beating a world-champion program in each case. In [67] it is shown that given the initial demonstration, no explicit exploration is necessary, and we can attain near-optimal performance. Measuring the divergence between the current policy and the expert policy for optimization is proposed in [68]. DQfD [69] pre-trains the agent and uses expert demonstrations by adding them into the replay buffer with additional priority. Moreover, a training framework that combines learning from both demonstrations and reinforcement learning is proposed in [70] for fast learning agents. Two policies close to maximizing the reward function can still have large differences in behaviour. To avoid degenerating a solution which would fit the reward but not the original behaviour, authors [71] proposed a method for enforcing that the optimal policy learnt over the rewards should still match the observed policy in behavior. Behavior Cloning (BC) is applied as a supervised learning that maps states to actions based on demonstrations provided by an expert. On the other hand, Inverse Reinforcement Learning (IRL) is about inferring the reward function that justifies demonstrations of the expert. IRL is the problem of extracting a reward function given observed, optimal behavior [72]. A

key motivation is that the reward function provides a succinct and robust definition of a task. Generally, IRL algorithms can be expensive to run, requiring reinforcement learning in an inner loop between cost estimation to policy training and evaluation. Generative Adversarial Imitation Learning (GAIL) [73] introduces a way to avoid this expensive inner loop. In practice, GAIL trains a policy close enough to the expert policy to fool a discriminator. This process is similar to Generative Adversarial Networks (GANs) [74], [75]. The resulting policy must travel the same MDP states as the expert, or the discriminator would pick up the differences. The theory behind GAIL is an equation simplification: qualitatively, if IRL is going from demonstrations to a cost function and RL from a cost function to a policy, then we should altogether be able to go from demonstration to policy in a single equation while avoiding the cost function estimation.

V. Reinforcement learning for Autonomous driving tasks

Autonomous driving tasks where RL could be applied include: controller optimization, path planning and trajectory optimization, motion planning and dynamic path planning, development of high-level driving policies for complex navigation tasks, scenario-based policy learning for highways, intersections, merges and splits, reward learning with inverse reinforcement learning from expert data for intent prediction for traffic actors such as pedestrian, vehicles and finally learning of policies that ensures safety and perform risk estimation. Before discussing the applications of DRL to AD tasks we briefly review the state space, action space and rewards schemes in autonomous driving setting.

A. State Spaces, Action Spaces and Rewards

To successfully apply DRL to autonomous driving tasks, designing appropriate state spaces, action spaces, and reward functions is important. Leurent *et al.* [76] provided a comprehensive review of the different state and action representations which are used in autonomous driving research. Commonly used state space features for an autonomous vehicle include: position, heading and velocity of ego-vehicle, as well as other obstacles in the sensor view extent of the ego-vehicle. To avoid variations in the dimension of the state space, a Cartesian or Polar occupancy grid around the ego vehicle is frequently employed. This is further augmented with lane information such as lane number (ego-lane or others), path curvature, future trajectory of the ego-vehicle, longitudinal information such as Time-to-collision (TTC), and finally scene information such as traffic laws and signal locations. Using raw sensor data such as camera images, LiDAR, radar, etc. provides the benefit of finer contextual information, while using condensed abstracted data reduces the complexity of the state space. In between, a mid-level representation such as 2D bird eye view (BEV) is sensor agnostic but still close to the spatial organization of the scene. Fig. 3 is an illustration of a top down view showing an occupancy grid, past and projected trajectories, and semantic information about the scene such as the position of traffic lights. This intermediary format retains the spatial layout of roads when graph-based representations would not. Some simulators offer this view such as Carla or Flow (see Table V-C). A vehicle policy must control a number of different actuators. Continuous-valued actuators for vehicle control include steering angle, throttle and brake. Other actuators such as gear changes are discrete. To reduce complexity and allow the application of DRL algorithms which work with discrete action spaces only (*e.g.* DQN), an action space may be discretised uniformly by dividing the range of continuous actuators such as steering angle, throttle and brake into equal-sized bins. Discretisation in log-space has also been suggested, as many steering angles which are selected in practice are close to the centre [77]. Discretisation does have disadvantages however; it can lead to jerky or unstable trajectories if the step values between actions are too large. Furthermore, when selecting the number of bins for an actuator there is a tradeoff between having enough discrete steps to allow for smooth control, and not having so many steps that action selections become prohibitively expensive to evaluate. As an alternative to discretisation, continuous values for actuators may also be handled by DRL algorithms which learn a policy directly, (*e.g.* DDPG). Temporal abstractions options framework [78]) may also be employed to simplify the process of selecting actions, where agents select *options* instead of low-level actions. These options represent a sub-policy that could extend a primitive action over multiple time steps. How to design reward functions to train DRL agents for autonomous driving is still very much an open question. Examples of criteria which have been integrated into reward functions when training DRL agents for AD tasks include: distance travelled towards a destination [79], speed of the ego vehicle [79]–[83], keeping the ego vehicle at a standstill [82], collisions with other road users or scene objects [79], [80], [82], infractions on sidewalks [79], keeping in lane [79], [81]–[83], maintaining comfort and stability (avoiding extreme acceleration, braking or steering) [80], [82], [83], and following traffic rules [80]. Many applications of DRL for AD use a combination of more than one criterion in the reward function; a weighted sum is often used to linearly scalarise these components into a single reward term to be used during learning (see *e.g.* [79]). As AD is essentially a multi-objective problem, methods from the field multi-objective RL such as thresholded lexicographic ordering may be easily applied and have been demonstrated to work well (see *e.g.* [80]).

B. Motion Planning & Trajectory optimization

Motion planning is the task of ensuring the existence of a path between target and destination points. This is necessary to plan trajectories for vehicles over prior maps usually augmented with semantic information. Path planning in dynamic environments and varying vehicle dynamics is a key problem in autonomous driving, for example negotiating right to pass through in an intersection [86], merging into highways. Recent work by authors [91] contains real world motions by various traffic actors, observed in diverse interactive driving scenarios. Recently, authors demonstrated an application of DRL (DDPG) for AD using a full-sized autonomous vehicle

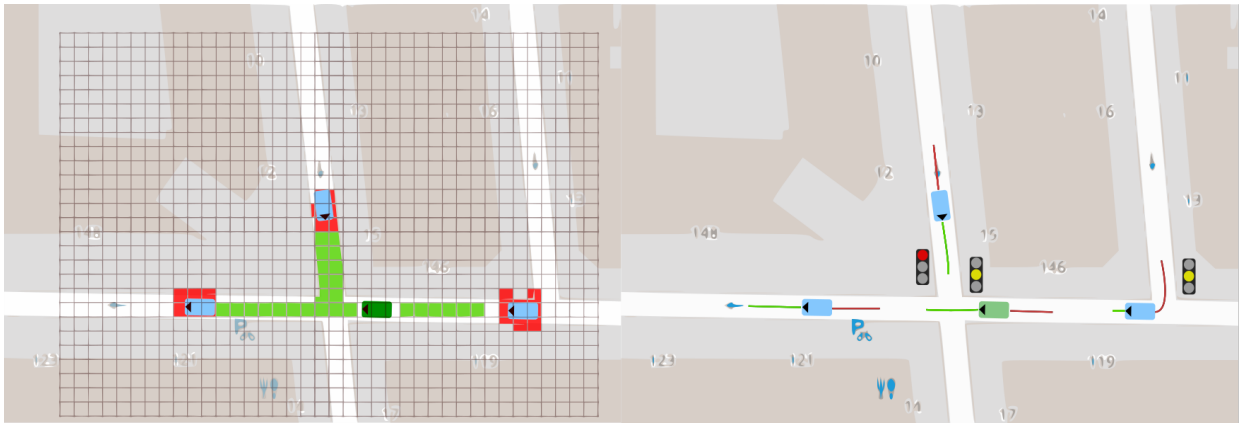

Fig. 3. Bird Eye View (BEV) 2D representation of a driving scene. Left demonstrates an occupancy grid. Right shows the combination of semantic information (traffic lights) with past (red) and projected (green) trajectories. The ego car is represented by a green rectangle in both images.

AD Task	Description and Utilization of (D)RL
Motion Planning	Learn to plan trajectories dynamically and optimize cost function to provide smooth control behavior of vehicle. Inverse RL is utilized to learn optimal reward function (or shaping) from experts. Authors propose to learn a heuristic function for the A^* algorithm using a DQN over image-based input obstacle map [84]
Overtaking	Authors [85] propose Multi-goal RL (MGRL) framework to learn overtaking policy while avoiding collisions & maintain steady speed.
Intersections/Merging	Ego-vehicle required to negotiate intersections and merges into highways [86], Ramp merging is tackled in [87], where DRL is applied to find an optimal driving policy using LSTM for producing an internal state containing historical driving information and DQN for Q-function approximation.
Lane Change	Learn a policy that decides whether the vehicle performs no operation, lane change to left/right, accelerate/decelerate. Authors [88] use Q-learning, whereas traditional approaches consist in defining fixed way points, velocity profiles and curvature of path to be followed by the ego vehicle.
Lane Keep	Ego-vehicle follows the lane. Authors [89] propose a DRL system for discrete actions (DQN) and continuous actions (DDAC) using the TORCS simulator (see Table V-C), study concludes that continuous actions provide smoother trajectories, while more restricted termination conditions lead to the slower convergence time to learn.
Automated parking	Learn policies to automatically park the vehicle [90].

TABLE I
LIST OF AD TASKS THAT REQUIRE D(RL) TO LEARN A POLICY OR BEHAVIOR.

[92]. The system was first trained in simulation, before being trained in real time using on board computers, and was able to learn to follow a lane, successfully completing a real-world trial on a 250 metre section of road. Model-based deep RL algorithms have been proposed for learning models and policies directly from raw pixel inputs [93], [94]. In [95], deep neural networks have been used to generate predictions in simulated environments over hundreds of time steps. RL is also suitable for Control. Classical optimal control methods like LQR/iLQR are compared with RL methods in [96]. Classical RL methods are used to perform optimal control in stochastic settings, for example the Linear Quadratic Regulator (LQR) in linear regimes and iterative LQR (iLQR) for non-linear regimes are utilized. A recent study in [97] demonstrates that random search over the parameters for a policy network can perform as well as LQR.

C. Simulator & Scenario generation tools

Autonomous driving datasets address supervised learning setup with training sets containing image, label pairs for various modalities. Reinforcement learning requires an environment where state-action pairs can be recovered while modelling dynamics of the vehicle state, environment as well as the stochasticity in the movement and actions of the environment and agent respectively. Various simulators are actively used for training and validating reinforcement learning algorithms. Table V-C summarises various high fidelity perception simulators capable of simulating cameras, LiDARs and radar. Some simulators are also capable of providing the vehicle state and dynamics. A complete review of sensors and simulators utilised within the autonomous driving community is available in [98] for readers. Learned driving policies are stress tested in simulated environments before moving on to costly evaluations in the real world. Multi-fidelity reinforcement learning (MFRL) framework is proposed in [99] where multiple simulators are available. In MFRL, a cascade of simulators with increasing fidelity are used in representing state dynamics (and thus computational cost) that enables the training and validation of RL algorithms. Experiments conducted on a remote controlled car show that MFRL transfers heuristics to guide exploration in high fidelity simulators. Moreover, it allows RL algorithms to find near optimal policies for the real world with fewer expensive real world samples using a remote controlled car. Besides the MFRL setup, simulators are built to model realistic perception streams from camera, LIDAR, radar and other sensor suites.

Decision making simulators require much lesser fidelity in perception while focusing vehicle dynamics and modelling the environment for path planning and trajectory opmization tasks. CARLA Challenge [100], is a Carla simulator based autonomous driving system challenge with pre-crash scenarios characterized in a National Highway Traffic Safety Administration report [101]. The systems are evaluated in critical scenarios such as: Ego-vehicle loses control, ego-vehicle reacts to unseen obstacle, lane change to evade slow leading vehicle among others. The scores of agents are evaluated as a function of the aggregated distance travelled in different circuits, and total points discounted due to infractions. Recent large scale data collection on human-driven cars have lead to a data driven approach using time series data available from the GPU and IMU which were later used to extract driving primitives using unsupervised learning methods such as clustering or Bayesian optimisation [102].

D. LfD and IRL for AD applications

Early work on Behavior Cloning (BC) for driving cars in [112], [113] presented agents that learn form demonstrations (LfD) that tries to mimic the behavior of an expert. BC is typically implemented as a supervised learning, and accordingly, it is hard for BC to adapt to new, unseen situations. An architecture for learning a convolutional neural network, end to end, in self-driving cars domain was proposed in [114], [115]. The CNN is trained to map raw pixels from a single front facing camera directly to steering commands. Using a relatively small training dataset from humans/experts, the system learns to drive in traffic on local roads with or without lane markings and on highways. The network learns image representations that detect the road successfully, without being explicitly trained to do so. Authors of [116] proposed to learn comfortable driving trajectories optimization using expert demonstration from human drivers using Maximum Entropy Inverse RL. Authors of [117] used DQN as the refinement step in IRL to extract the rewards, in an effort learn human-like lane change behavior.

VI. Challenges for real-world RL

A. Validating RL systems

Henderson *et al.* [118] described challenges in validating reinforcement learning methods focusing on policy gradient methods for continuous control algorithms such as PPO, DDPG and TRPO as well as in reproducing benchmarks. They demonstrate with real examples that implementations often have varying code-bases and different hyper-parameter values, and that unprincipled ways to estimate the top-k rollouts could lead to incoherent interpretations on the performance of the reinforcement learning algorithms, and further more on how well they generalize. Authors concluded that evaluation could be performed either on a well defined common setup or on real-world tasks. Authors in [119] proposed automated generation of challenging and rare driving scenarios in high-fidelity photo-realistic simulators. These adversarial scenarios are automatically discovered by parameterising the behavior of pedestrians and other vehicles on the road. Moreover, it is shown that by adding these scenarios to the training data of imitation learning, the safety is increased.

B. Bridging the simulation-reality gap

Training deep networks requires collecting and annotating a lot of data which is usually costly in terms of time and effort. Using simulation environments enables the collection of large training datasets. However, the simulated data usually do not have the same data distribution compared to the real data. Accordingly, models trained on simulated environments often fail to generalise well on real environments. Domain adaptation allows a machine learning model trained on samples from a source domain to generalise on a target domain. Feature-level domain adaptation focuses on learning domain-invariant features. In [120], the decisions made by deep neural networks are based on features that are both discriminative and invariant to the change of domains. Pixel level domain adaptation focuses on stylizing images from the source domain to make them *similar* to images of the target domain, based on image conditioned GANs. In [121], the model learns a transformation in the pixel space from one domain to the other, in an unsupervised way. GAN is used to adapt simulated images to look like as if drawn from the real domain. Both feature-level and pixel-level domain adaptation are combined in [122], where the results indicate that including simulated data can improve the vision-based grasping system, achieving comparable performance with 50 times fewer real-world samples. Another relatively simpler method is introduced in [123], by randomizing the dynamics of the simulator during training, policies are capable of generalising to different dynamics without any training on the real system. A model trained in a virtual environment is shown to be workable in real environment [124]. Virtual images rendered by a simulator are first segmented to scene parsing representation and then translated to synthetic realistic images by the proposed image translation network. The proposed network segments the simulated image input, and then generates a synthetic realistic images. Accordingly, the driving policy trained by reinforcement learning can be easily adapted to real environment. Authors in [125] addressed the issue of performing imitation learning in simulation that transfers well to images from real world. They achieved this by unsupervised domain transfer between simulated and real world images. Additionally, the auxiliary task of predicting the steering control of the vehicle is added. By jointly training on real world images translated to simulation and the controller applied over them, a controller is learns steering in the real world domain without any ground truth from the real world. Authors remark that there were no pairwise correspondences between images in the simulated training set and the unlabelled real-world image set. Similarly, [126] performs domain adaptation to map real world images to simulated images. In contrast to sim-to-real methods they handle the reality gap during deployment of agents in real scenarios, by adapting the real camera streams to the synthetic modality, so as to map the unfamiliar or unseen features of real images back into the simulated style, which the agents have already learned how to deal with during training in the simulation.

Simulator	Description
CARLA [79]	Urban simulator, Camera & LIDAR streams, with depth & semantic segmentation, Location information
TORCS [103]	Racing Simulator, Camera stream, agent positions, testing control policies for vehicles
AIRSIM [104]	Camera stream with depth and semantic segmentation, support for drones
GAZEBO (ROS) [105]	Multi-robot physics simulator employed for path planning & vehicle control in complex 2D & 3D maps
SUMO [106]	Macro-scale modelling of traffic in cities motion planning simulators are used
DeepDrive [107]	Driving simulator based on unreal, providing multi-camera (eight) stream with depth
Constellation [108]	NVIDIA DRIVE Constellation^{TM} simulates camera, LIDAR and radar for autonomous driving (Proprietary)
MADRaS [109]	Multi-Agent Autonomous Driving Simulator built on top of TORCS
Flow [110]	Multi-Agent Traffic Control Simulator built on top of SUMO
Highway-env [111]	A gym-based environment that provides a simulator for highway based road topologies
Carcraft	Waymo's simulation environment (Proprietary)

TABLE II
SIMULATORS FOR RL APPLICATIONS IN ADVANCED DRIVING ASSISTANCE SYSTEMS (ADAS) AND AUTONOMOUS DRIVING.

C. Sample efficiency

Animals are usually able to learn new tasks in just a few trials, benefiting from their prior knowledge about the environment. However, one of the most important challenges for reinforcement learning is sample efficiency, where the learning process requires too many samples for learning a decent policy. This issue becomes more noticeable when collection of samples is expensive or even risky. For example in the case of robot control and autonomous driving. Sample efficiency is a difficult issue due to the delayed and sparse rewards found in typical settings, in addition to the large size of the state space. Accordingly, some approaches enable the agent to learn intermediate goals via reward shaping by designing a more frequent reward function to encourage the agent to learn faster from fewer samples. Efficiency can be achieved by conducting imitation learning, where the agent is learning offline an initial policy from trajectories provided by an expert. Next, the agent can self-improve by applying RL while interacting with the environment. Transfer learning is another approach for sample efficiency, which enables the reuse of previously trained policy for a source task to initialize the learning of a target task. Policy composition presented in [127] propose composing previously learned basis policies to be able to reuse them for a novel task, which leads to faster learning of new policies. A survey on transfer learning in RL is presented in [128]. Actor Critic with Experience Replay (ACER) [129], is a sample-efficient policy gradient algorithm that makes use of a replay buffer, enabling it to perform more than one gradient update using each piece of sampled experience, as well as a trust region policy optimization method. Multi-fidelity reinforcement learning (MFRL) framework [99] showed to transfer heuristics to guide exploration in high fidelity simulators and find near optimal policies for the real world with fewer real world samples. Moreover, model-based RL agents are known to have a competitive edge over model-free agents, in terms of sample efficiency, where the agent can plan ahead utilizing its own model of the environment. World models proposed in [130] are trained quickly in an unsupervised way, via a variational autoencoder (VAE), to learn a compressed spatial and temporal representation of the environment. This approach leads to learning a compact and simple policy directly from the compressed representation. Moreover, the agent can while utilizing the internal learned environment model and transfer the policy back into the actual environment. Authors in [131] transferred policies learnt to handle simulated intersections to real world examples between DQN agents. Meta-learning algorithms enable agents adapt to new tasks and learn new skills rapidly from small amounts of experiences, benefiting from their prior knowledge about the world. Authors of [132] addressed this issue through training a recurrent neural network on a training set of interrelated tasks, where the network input includes the action selected in addition to the reward received in the previous time step. Accordingly, the agent is trained to learn to exploit the structure of the problem dynamically and solve new problems by adjusting its hidden state. A similar approach for designing RL algorithms is presented in [133]. Rather than designing a "fast" reinforcement learning algorithm, it is represented as a recurrent neural network, and learned from data. In Model-Agnostic Meta-Learning (MAML) proposed in [134], the meta-learner seeks to find an initialisation for the parameters of a neural network, that can be adapted quickly for a new task using only few examples. Continuous adaptation in dynamically changing and adversarial scenarios is presented in [135] via a simple gradient-based meta-learning algorithm. Additionally, Reptile [136] is mathematically similar to first-order MAML, making it consumes less computation and memory.

D. Exploration issues with Imitation

In imitation learning, the agent makes use of trajectories provided by an expert. However, the distribution of states the expert encounters usually does not cover all the states the trained agent may encounter during testing. Furthermore imitation assumes that the actions are independent and identically distributed (i.i.d.). One solution consists in using the Data Aggregation (DAgger) methods [137] where the end-to-end learned policy is executed, and extracted observation-action pairs are again labelled by the expert, and aggregated to the original expert observation-action dataset. Thus, iteratively collecting training examples from both reference and trained policies explores more valuable states and solves this lack of exploration. Following work on Search-based Structured Prediction (SEARN) [137], Stochastic Mixing Iterative Learning (SMILE) trains a stochastic stationary policy over several iterations and then makes use of a geometric stochastic mixing of

the policies trained. In a standard imitation learning scenario, the demonstrator is required to cover sufficient states so as to avoid unseen states during test. This constraint is costly and requires frequent human intervention. More recently, Chauffeurnet [138] demonstrated the limits of imitation learning where even 30 million state-action samples were insufficient to learn an optimal policy that mapped bird-eye view images (states) to control (action). The authors propose the use of simulated examples which introduced perturbations, higher diversity of scenarios such as collisions and/or going off the road. The *featurenet* includes an agent RNN that outputs the way point, agent box position and heading at each iteration.

E. Intrinsic Reward functions

In controlled simulated environments such as games, an explicit reward signal is given to the agent along with its sensor stream. However, in real-world robotics and autonomous driving deriving, designing a *good* reward functions is essential so that the desired behaviour may be learned. The most common solution has been reward shaping [139] and consists in supplying additional well designed rewards to the agent to encourage the optimization into the direction of the optimal policy. Rewards as already pointed earlier in the paper, could be estimated by inverse RL (IRL) [140], which depends on expert demonstrations. In the absence of an explicit reward shaping and expert demonstrations, agents can use intrinsic rewards or intrinsic motivation [141] to evaluate if their actions were good or not. Authors of [142] define *curiosity* as the error in an agent's ability to predict the consequence of its own actions in a visual feature space learned by a self-supervised inverse dynamics model. In [143] the agent learns a next state predictor model from its experience, and uses the error of the prediction as an intrinsic reward. This enables that agent to determine what could be a useful behavior even without extrinsic rewards.

F. Incorporating safety in DRL

Deploying an autonomous vehicle in real environments after training directly could be dangerous. Different approaches to incorporate safety into DRL algorithms are presented here. For imitation learning based systems, Safe DAgger [144] introduces a safety policy that learns to predict the error made by a primary policy trained initially with the supervised learning approach, without querying a reference policy. An additional safe policy takes both the partial observation of a state and a primary policy as inputs, and returns a binary label indicating whether the primary policy is likely to deviate from a reference policy without querying it. Authors of [145] addressed safety in multi-agent Reinforcement Learning for Autonomous Driving, where a balance is maintained between unexpected behavior of other drivers or pedestrians and not to be too defensive, so that normal traffic flow is achieved. While hard constraints are maintained to guarantee the safety of driving, the problem is decomposed into a composition of a policy for desires to enable comfort driving and trajectory planning. The deep reinforcement learning algorithms for control such as DDPG and safety based control are combined in [146], including artificial potential field method that is widely used for robot path planning. Using TORCS environment, the DDPG is applied first for learning a driving policy in a stable and familiar environment, then policy network and safety-based control are combined to avoid collisions. It was found that combination of DRL and safety-based control performs well in most scenarios. In order to enable DRL to escape local optima, speed up the training process and avoid danger conditions or accidents, Survival-Oriented Reinforcement Learning (SORL) model is proposed in [147], where survival is favored over maximizing total reward through modeling the autonomous driving problem as a constrained MDP and introducing Negative-Avoidance Function to learn from previous failure. The SORL model was found to be not sensitive to reward function and can use different DRL algorithms like DDPG. Furthermore, a comprehensive survey on safe reinforcement learning can be found in [148] for interested readers.

G. Open-source frameworks

Reinforcement learning results are usually difficult to reproduce and are highly sensitive to hyper-parameter choices, which are often not reported in detail. Both researchers and practitioners need to have a reliable starting point where the well known reinforcement learning algorithms are implemented, documented and well tested. OpenAI Baselines [149] provide a set of high-quality implementations of different reinforcement learning algorithms. The main goal for these Baselines is to make it easier for the research community to replicate, refine and create good baselines to build research on top of. Moreover, Unity Machine Learning Agents Toolkit implements core RL algorithms, games, simulations environments for training RL or IL based agents [150]. Reinforcement Learning Coach by Intel AI Lab [151] enables easy experimentation with state of the art RL algorithms. Coach allows simple integration of new environments. Extending and reusing existing components is enabled through the decoupling of basic RL components. Tensorflow Agents (TF-Agents) [152] is a versatile RL library for TensorFlow where an agent encompasses two main responsibilities: defining a policy to interact with the environment, and how to learn that policy from collected experience. TF-Agents enable developers to directly using the code from the package itself. TF-Agents is well tested and provides Jupyter notebooks example. Additionally, rlpyt [153] implements deep Q-learning, policy gradients and Q-value policy gradients algorithm families on top of a shared, optimized infrastructure, in a single repository in Python. rlpyt is designed as a high throughput code base for small to medium scale research in deep RL. More recently, DeepMind has released the Behaviour Suite for Reinforcement Learning (bsuite) [154] which is a collection of experiments that investigate capabilities of agents with two objectives: First, to collect the key issues in the design of general and efficient learning algorithms; Second, to study agent behaviour through their performance. The evaluation and analysis are automated for any agent on bsuite.

VII. Conclusion

Reinforcement learning is still an emerging area in real-world autonomous driving applications. Although there are a few successful commercial applications, there is very little literature or large-scale public datasets available. Thus we were motivated to formalize and organize RL applications for autonomous driving. Autonomous driving scenarios involve interacting agents and require negotiation and dynamic decision making which suits RL. However, there are many challenges to be resolved in order to have mature solutions which we discuss in detail. We hope that this overview paper encourages further research and applications.

Appendix

A2C	Advantage Actor Critic
A3C	Asynchronous A2C
BC	Behavior Cloning
DDPG	Deep DPG
DP	Dynamic Programming
DPG	Deterministic PG
DQN	Deep Q-Network
DRQN	Deep Recurrent Q-Network
DRL	Deep RL
GAIL	Generative Adversarial IL
HRL	Hierarchical RL
IL	Imitation Learning
IRL	Inverse RL
LfD	Learning from Demonstration
MAML	Model-Agnostic Meta-Learning
MARL	Multi-Agent RL
MDP	Markov Decision Process
MFRL	Multi-Fidelity RL
MGRL	Multiple Goal RL
MOMDP	Multi-Objective MDP
MORL	Multi-Objective RL
MOSG	Multi-Objective SG
PG	Policy Gradient
POMDP	Partially Observed MDP
PPO	Proximal Policy Optimization
RRT	Rapidly-exploring Randomm Trees
SG	Stochastic Game
SORL	Survival-Oriented RL
SMDP	Semi-Markov Decision Process
SRL	State Representation Leaning
TD	Time Difference
TRPO	Trust Region Policy Optimization

TABLE III
Acronyms related to Reinforcememt learning (RL).

AD	Autonomous Driving
AE	Auto-Encoder
ADAS	Advanced Driving Assistance System
BEV	Bird Eye View
CNN	Convolutional Neural Network
DNN	Deep Neural Network
DoF	Degree of Freedom
GAN	Generative Adversarial Network
iLRQR	iterative LQR
IMU	Inertial Measurement Unit
KL	Kullback-Leibler
LQR	Linear Quadratic Regulator
LSTM	Long-Short Term Memory
ML	Machine Learning
MPC	Model Predictive Control
PID	Proportional-Integral-Derivative
TTC	Time-to-Collision
VAE	Variational AE

TABLE IV
Other acronyms.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction (Second Edition)*. MIT Press, 2018. 1, 3, 4, 6

[2] V. Talpaert., I. Sobh., B. R. Kiran., P. Mannion., S. Yogamani., A. El-Sallab., and P. Perez., "Exploring applications of deep reinforcement learning for real-world autonomous driving systems," in *Proceedings of the 14th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 5 VISAPP: VISAPP,*, INSTICC. SciTePress, 2019, pp. 564–572. 1

[3] M. Siam, S. Elkerdawy, M. Jagersand, and S. Yogamani, "Deep semantic segmentation for automated driving: Taxonomy, roadmap and challenges," in *2017 IEEE 20th international conference on intelligent transportation systems (ITSC)*. IEEE, 2017, pp. 1–8. 1

[4] K. El Madawi, H. Rashed, A. El Sallab, O. Nasr, H. Kamel, and S. Yogamani, "Rgb and lidar fusion based 3d semantic segmentation for autonomous driving," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2019, pp. 7–12. 1

[5] M. Siam, H. Mahgoub, M. Zahran, S. Yogamani, M. Jagersand, and A. El-Sallab, "Modnet: Motion and appearance based moving object detection network for autonomous driving," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2018, pp. 2859–2864. 1

[6] V. R. Kumar, S. Milz, C. Witt, M. Simon, K. Amende, J. Petzold, S. Yogamani, and T. Pech, "Monocular fisheye camera depth estimation using sparse lidar supervision," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2018, pp. 2853–2858. 1

[7] M. Uřičář, P. Křížek, G. Sistu, and S. Yogamani, "Soilingnet: Soiling detection on automotive surround-view cameras," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2019, pp. 67–72. 1

[8] G. Sistu, I. Leang, S. Chennupati, S. Yogamani, C. Hughes, S. Milz, and S. Rawashdeh, "Neurall: Towards a unified visual perception model for automated driving," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2019, pp. 796–803. 1

[9] S. Yogamani, C. Hughes, J. Horgan, G. Sistu, P. Varley, D. O'Dea, M. Uricár, S. Milz, M. Simon, K. Amende *et al.*, "Woodscape: A multi-task, multi-camera fisheye dataset for autonomous driving," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 9308–9318. 1

[10] S. Milz, G. Arbeiter, C. Witt, B. Abdallah, and S. Yogamani, "Visual slam for automated driving: Exploring the applications of deep learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2018, pp. 247–257. 2

[11] S. M. LaValle, *Planning Algorithms*. New York, NY, USA: Cambridge University Press, 2006. 2

[12] S. M. LaValle and J. James J. Kuffner, "Randomized kinodynamic planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001. 2

[13] Y. Kuwata, J. Teo, G. Fiore, S. Karaman, E. Frazzoli, and J. P. How, "Real-time motion planning with applications to autonomous urban driving," *IEEE Transactions on Control Systems Technology*, vol. 17, no. 5, pp. 1105–1118, 2009. 2

[14] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Transactions on intelligent vehicles*, vol. 1, no. 1, pp. 33–55, 2016. 2

[15] W. Schwarting, J. Alonso-Mora, and D. Rus, "Planning and decision-making for autonomous vehicles," *Annual Review of Control, Robotics, and Autonomous Systems*, no. 0, 2018. 2

[16] T. M. Mitchell, *Machine learning*, ser. McGraw-Hill series in computer science. Boston (Mass.), Burr Ridge (Ill.), Dubuque (Iowa): McGraw-Hill, 1997. 3

[17] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach (3rd edition)*. Prentice Hall, 2009. 3

[18] Z.-W. Hong, T.-Y. Shann, S.-Y. Su, Y.-H. Chang, and C.-Y. Lee, "Diversity-driven exploration strategy for deep reinforcement learning," *arXiv preprint arXiv:1802.04564*, 2018. 3

[19] M. Wiering and M. van Otterlo, Eds., *Reinforcement Learning: State-of-the-Art*. Springer, 2012. 3

[20] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, 1st ed. New York, NY, USA: John Wiley & Sons, Inc., 1994. 3

[21] C. J. Watkins and P. Dayan, "Technical note: Q-learning," *Machine Learning*, vol. 8, no. 3-4, 1992. 4

[22] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, 2015. 4, 6

[23] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, King's College, Cambridge, 1989. 4, 6

[24] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *ICML*, 2014. 4

[25] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, pp. 229–256, 1992. 4, 5

[26] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International Conference on Machine Learning*, 2015, pp. 1889–1897. 4

[27] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017. 5

[28] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015. 5

[29] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning*, 2016. 5

[30] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, "Reinforcement learning with deep energy-based policies," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ser. ICML'17. JMLR.org, 2017, pp. 1352–1361. 5

[31] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine, "Soft actor-critic algorithms and applications," *CoRR*, vol. abs/1812.05905, 2018. 5

[32] R. S. Sutton, "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming," in *Machine Learning Proceedings 1990*. Elsevier, 1990. 6

[33] R. I. Brafman and M. Tennenholtz, "R-max-a general polynomial time algorithm for near-optimal reinforcement learning," *Journal of Machine Learning Research*, vol. 3, no. Oct, 2002. 6

[34] D. Silver, R. S. Sutton, and M. Müller, "Sample-based learning and search with permanent and transient memories," in *Proceedings of the 25th international conference on Machine learning*. ACM, 2008, pp. 968–975. 6

[35] G. A. Rummery and M. Niranjan, "On-line Q-learning using connectionist systems," Cambridge University Engineering Department, Cambridge, England, Tech. Rep. TR 166, 1994. 6

[36] R. S. Sutton and A. G. Barto, "Reinforcement learning an introduction–second edition, in progress (draft)," 2015. 6

[37] R. Bellman, *Dynamic Programming*. Princeton, NJ, USA: Princeton University Press, 1957. 6

[38] G. Tesauro, "Td-gammon, a self-teaching backgammon program, achieves master-level play," *Neural Computing*, vol. 6, no. 2, Mar. 1994. 6

[39] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016. 6, 8

[40] K. Narasimhan, T. Kulkarni, and R. Barzilay, "Language understanding for text-based games using deep reinforcement learning," *arXiv preprint arXiv:1506.08941*, 2015. 6

[41] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015. 6

[42] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning." in *AAAI*, vol. 16, 2016, pp. 2094–2100. 6

[43] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," *arXiv preprint arXiv:1511.06581*, 2015. 7

[44] M. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," *CoRR, abs/1507.06527*, 2015. 7

[45] B. F. Skinner, *The behavior of organisms: An experimental analysis*. Appleton-Century, 1938. 7

[46] E. Wiewiora, "Reward shaping," in *Encyclopedia of Machine Learning and Data Mining*, C. Sammut and G. I. Webb, Eds. Boston, MA: Springer US, 2017, pp. 1104–1106. 7

[47] J. Randløv and P. Alstrøm, "Learning to drive a bicycle using reinforcement learning and shaping," in *Proceedings of the Fifteenth International Conference on Machine Learning*, ser. ICML '98. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998, pp. 463–471. 7

[48] D. H. Wolpert, K. R. Wheeler, and K. Tumer, "Collective intelligence for control of distributed dynamical systems," *EPL (Europhysics Letters)*, vol. 49, no. 6, p. 708, 2000. 7

[49] A. Y. Ng, D. Harada, and S. J. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *Proceedings of the Sixteenth International Conference on Machine Learning*, ser. ICML '99, 1999, pp. 278–287. 7

[50] S. Devlin and D. Kudenko, "Theoretical considerations of potential-based reward shaping for multi-agent systems," in *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2011. 7

[51] P. Mannion, S. Devlin, K. Mason, J. Duggan, and E. Howley, "Policy invariance under reward transformations for multi-objective reinforcement learning," *Neurocomputing*, vol. 263, 2017. 7

[52] M. Colby and K. Tumer, "An evolutionary game theoretic analysis of difference evaluation functions," in *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM, 2015, pp. 1391–1398. 7

[53] P. Mannion, J. Duggan, and E. Howley, "A theoretical and empirical analysis of reward transformations in multi-objective stochastic games," in *Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2017. 7

[54] L. Buşoniu, R. Babuška, and B. Schutter, "Multi-agent reinforcement learning: An overview," in *Innovations in Multi-Agent Systems and Applications - 1*, ser. Studies in Computational Intelligence, D. Srinivasan and L. Jain, Eds. Springer Berlin Heidelberg, 2010, vol. 310. 7

[55] P. Mannion, K. Mason, S. Devlin, J. Duggan, and E. Howley, "Multi-objective dynamic dispatch optimisation using multi-agent reinforcement learning," in *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2016. 8

[56] K. Mason, P. Mannion, J. Duggan, and E. Howley, "Applying multi-agent reinforcement learning to watershed management," in *Proceedings of the Adaptive and Learning Agents workshop (at AAMAS 2016)*, 2016. 8

[57] V. Pareto, *Manual of political economy*. OUP Oxford, 1906. 8

[58] D. M. Roijers, P. Vamplew, S. Whiteson, and R. Dazeley, "A survey of multi-objective sequential decision-making," *Journal of Artificial Intelligence Research*, vol. 48, pp. 67–113, 2013. 8

[59] R. Rădulescu, P. Mannion, D. Roijers, and A. Nowé, "Multi-objective multi-agent decision making: A utility-based analysis and survey," *Autonomous Agents and Multi-Agent Systems*, 2019 (in press). [Online]. Available: https://arxiv.org/abs/1909.02964 8

[60] T. Lesort, N. Diaz-Rodriguez, J.-F. Goudou, and D. Filliat, "State representation learning for control: An overview," *Neural Networks*, vol. 108, pp. 379 – 392, 2018. 8

[61] A. Raffin, A. Hill, K. R. Traoré, T. Lesort, N. D. Rodríguez, and D. Filliat, "Decoupling feature extraction from policy learning: assessing benefits of state representation learning in goal based robotics," *CoRR*, vol. abs/1901.08651, 2019. 8

[62] W. Böhmer, J. T. Springenberg, J. Boedecker, M. Riedmiller, and K. Obermayer, "Autonomous learning of state representations for control: An emerging field aims to autonomously learn state representations for reinforcement learning agents from their real-world sensor observations," *KI-Künstliche Intelligenz*, vol. 29, no. 4, pp. 353–362, 2015. 8

[63] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th annual international conference on machine learning*. ACM, 2009, pp. 41–48. 8

[64] D. A. Cohn, Z. Ghahramani, and M. I. Jordan, "Active learning with statistical models," *Journal of artificial intelligence research*, 1996. 8

[65] A. G. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete Event Dynamic Systems*, vol. 13, no. 4, pp. 341–379, 2003. 8

[66] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017. 8

[67] P. Abbeel and A. Y. Ng, "Exploration and apprenticeship learning in reinforcement learning," in *Proceedings of the 22nd international conference on Machine learning*. ACM, 2005, pp. 1–8. 8

[68] B. Kang, Z. Jie, and J. Feng, "Policy optimization with demonstrations," in *International Conference on Machine Learning*, 2018, pp. 2474–2483. 8

[69] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, I. Osband *et al.*, "Deep q-learning from demonstrations," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018. 8

[70] S. Ibrahim and D. Nevin, "End-to-end framework for fast learning asynchronous agents," in *the 32nd Conference on Neural Information Processing Systems, Imitation Learning and its Challenges in Robotics workshop*, 2018. 8

[71] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the twenty-first international conference on Machine learning*. ACM, 2004, p. 1. 8

[72] A. Y. Ng, S. J. Russell *et al.*, "Algorithms for inverse reinforcement learning." in *ICML*, 2000. 8

[73] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *Advances in Neural Information Processing Systems*, 2016, pp. 4565–4573. 9

[74] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems 27*, 2014, pp. 2672–2680. 9

[75] M. Uřičář, P. Křížek, D. Hurych, I. Sobh, S. Yogamani, and P. Denny, "Yes, we gan: Applying adversarial techniques for autonomous driving," *Electronic Imaging*, vol. 2019, no. 15, pp. 48–1, 2019. 9

[76] E. Leurent, Y. Blanco, D. Efimov, and O.-A. Maillard, "A survey of state-action representations for autonomous driving," *HAL archives*, 2018. 9

[77] H. Xu, Y. Gao, F. Yu, and T. Darrell, "End-to-end learning of driving models from large-scale video datasets," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017, pp. 3530–3538. 9

[78] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999. 9

[79] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16. 9, 12

[80] C. Li and K. Czarnecki, "Urban driving with multi-objective deep reinforcement learning," in *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2019, pp. 359–367. 9

[81] S. Wang, D. Jia, and X. Weng, "Deep reinforcement learning for autonomous driving," *arXiv preprint arXiv:1811.11329*, 2018. 9

[82] J. Chen, B. Yuan, and M. Tomizuka, "Model-free deep reinforcement learning for urban autonomous driving," *arXiv preprint arXiv:1904.09503*, 2019. 9

[83] S. Kardell and M. Kuosku, "Autonomous vehicle control via deep reinforcement learning," Master's thesis, 2017. 9

[84] A. Keselman, S. Ten, A. Ghazali, and M. Jubeh, "Reinforcement learning with a* and a deep heuristic," *arXiv preprint arXiv:1811.07745*, 2018. 10

[85] D. C. K. Ngai and N. H. C. Yung, "A multiple-goal reinforcement learning method for complex vehicle overtaking maneuvers," *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, no. 2, pp. 509–522, 2011. 10

[86] D. Isele, R. Rahimi, A. Cosgun, K. Subramanian, and K. Fujimura, "Navigating occluded intersections with autonomous vehicles using deep reinforcement learning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 2034–2039. 9, 10

[87] P. Wang and C.-Y. Chan, "Formulation of deep reinforcement learning architecture toward autonomous driving for on-ramp merge," in *Intelligent Transportation Systems (ITSC), 2017 IEEE 20th International Conference on*. IEEE, 2017, pp. 1–6. 10

[88] P. Wang, C.-Y. Chan, and A. de La Fortelle, "A reinforcement learning based approach for automated lane change maneuvers," in *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2018, pp. 1379–1384. 10

[89] A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani, "End-to-end deep reinforcement learning for lane keeping assist," *arXiv preprint arXiv:1612.04340*, 2016. 10

[90] A.-E. Sallab, M. Abdou, E. Perot, and S. Yogamani, "Deep reinforcement learning framework for autonomous driving," *Electronic Imaging*, vol. 2017, no. 19, pp. 70–76, 2017. 10

[91] W. Zhan, L. Sun, D. Wang, H. Shi, A. Clausse, M. Naumann, J. Kümmerle, H. Königshof, C. Stiller, A. de La Fortelle, and M. Tomizuka, "INTERACTION Dataset: An INTERnational, Adversarial and Cooperative moTION Dataset in Interactive Driving Scenarios with Semantic Maps," *arXiv:1910.03088 [cs, eess]*, 2019. 9

[92] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J.-M. Allen, V.-D. Lam, A. Bewley, and A. Shah, "Learning to drive in a day," *arXiv preprint arXiv:1807.00412*, 2018. 10

[93] M. Watter, J. Springenberg, J. Boedecker, and M. Riedmiller, "Embed to control: A locally linear latent dynamics model for control from raw images," in *Advances in neural information processing systems*, 2015. 10

[94] N. Wahlström, T. B. Schön, and M. P. Deisenroth, "From pixels to torques: Policy learning with deep dynamical models," *arXiv preprint arXiv:1502.02251*, 2015. 10

[95] S. Chiappa, S. Racaniere, D. Wierstra, and S. Mohamed, "Recurrent environment simulators," *arXiv preprint arXiv:1704.02254*, 2017. 10

[96] B. Recht, "A tour of reinforcement learning: The view from continuous control," *Annual Review of Control, Robotics, and Autonomous Systems*, 2008. 10

[97] H. Mania, A. Guy, and B. Recht, "Simple random search provides a competitive approach to reinforcement learning," *arXiv preprint arXiv:1803.07055*, 2018. 10

[98] F. Rosique, P. J. Navarro, C. Fernández, and A. Padilla, "A systematic review of perception system and simulators for autonomous vehicles research," *Sensors*, vol. 19, no. 3, p. 648, 2019. 10

[99] M. Cutler, T. J. Walsh, and J. P. How, "Reinforcement learning with multi-fidelity simulators," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 3888–3895. 10, 12

[100] F. C. A. M. L. German Ros, Vladlen Koltun, "Carla autonomous driving challenge," https://carlachallenge.org/, 2019, [Online; accessed 14-April-2019]. 11

[101] W. G. Najm, J. D. Smith, M. Yanagisawa *et al.*, "Pre-crash scenario typology for crash avoidance research," United States. National Highway Traffic Safety Administration, Tech. Rep., 2007. 11

[102] J. Zhu, W. Wang, and D. Zhao, "A tempt to unify heterogeneous driving databases using traffic primitives," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2018, pp. 2052–2057. 11

[103] B. Wymann, E. Espié, C. Guionneau, C. Dimitrakakis, R. Coulom, and A. Sumner, "Torcs, the open racing car simulator," *Software available at http://torcs. sourceforge. net*, vol. 4, 2000. 12

[104] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and Service Robotics*. Springer, 2018, pp. 621–635. 12

[105] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *In IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2004, pp. 2149–2154. 12

[106] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wießner, "Microscopic traffic simulation using sumo," in *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE, 2018. 12

[107] C. Quiter and M. Ernst, "deepdrive/deepdrive: 2.0," Mar. 2018. [Online]. Available: https://doi.org/10.5281/zenodo.1248998 12

[108] Nvidia, "Drive Constellation now available," https://blogs.nvidia.com/blog/2019/03/18/drive-constellation-now-available/, 2019, [accessed 14-April-2019]. 12

[109] A. S. et al., "Multi-Agent Autonomous Driving Simulator built on top of TORCS," https://github.com/madras-simulator/MADRaS, 2019, [Online; accessed 14-April-2019]. 12

[110] C. Wu, A. Kreidieh, K. Parvate, E. Vinitsky, and A. M. Bayen, "Flow: Architecture and benchmarking for reinforcement learning in traffic control," *CoRR*, vol. abs/1710.05465, 2017. 12

[111] E. Leurent, "A collection of environments for autonomous driving and tactical decision-making tasks," https://github.com/eleurent/highway-env, 2019, [Online; accessed 14-April-2019]. 12

[112] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," in *Advances in neural information processing systems*, 1989. 11

[113] D. Pomerleau, "Efficient training of artificial neural networks for autonomous navigation," *Neural Computation*, vol. 3, no. 1, 1991. 11

[114] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016. 11

[115] M. Bojarski, P. Yeres, A. Choromanska, K. Choromanski, B. Firner, L. Jackel, and U. Muller, "Explaining how a deep neural network trained with end-to-end learning steers a car," *arXiv preprint arXiv:1704.07911*, 2017. 11

[116] M. Kuderer, S. Gulati, and W. Burgard, "Learning driving styles for autonomous vehicles from demonstration," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 2015, pp. 2641–2646. 11

[117] S. Sharifzadeh, I. Chiotellis, R. Triebel, and D. Cremers, "Learning to drive using inverse reinforcement learning and deep q-networks," *arXiv preprint arXiv:1612.03653*, 2016. 11

[118] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018. 11

[119] Y. Abeysirigoonawardena, F. Shkurti, and G. Dudek, "Generating adversarial driving scenarios in high-fidelity simulators," in *2019 IEEE International Conference on Robotics and Automation (ICRA)*. ICRA, 2019. 11

[120] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky, "Domain-adversarial training of neural networks," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 2096–2030, 2016. 11

[121] K. Bousmalis, N. Silberman, D. Dohan, D. Erhan, and D. Krishnan, "Unsupervised pixel-level domain adaptation with generative adversarial networks," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 1, no. 2, 2017, p. 7. 11

[122] K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. Pastor, K. Konolige *et al.*, "Using simulation and domain adaptation to improve efficiency of deep robotic grasping," *arXiv preprint arXiv:1709.07857*, 2017. 11

[123] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," *arXiv preprint arXiv:1710.06537*, 2017. 11

[124] Z. W. Xinlei Pan, Yurong You and C. Lu, "Virtual to real reinforcement learning for autonomous driving," in *Proceedings of the British Machine Vision Conference (BMVC)*, G. B. Tae-Kyun Kim, Stefanos Zafeiriou and K. Mikolajczyk, Eds. BMVA Press, September 2017. 11

[125] A. Bewley, J. Rigley, Y. Liu, J. Hawke, R. Shen, V.-D. Lam, and A. Kendall, "Learning to drive from simulation without real world labels," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 4818–4824. 11

[126] J. Zhang, L. Tai, P. Yun, Y. Xiong, M. Liu, J. Boedecker, and W. Burgard, "Vr-goggles for robots: Real-to-sim domain adaptation for visual control," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1148–1155, 2019. 11

[127] R. Liaw, S. Krishnan, A. Garg, D. Crankshaw, J. E. Gonzalez, and K. Goldberg, "Composing meta-policies for autonomous driving using hierarchical deep reinforcement learning," *arXiv preprint arXiv:1711.01503*, 2017. 12

[128] M. E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey," *Journal of Machine Learning Research*, vol. 10, no. Jul, pp. 1633–1685, 2009. 12

[129] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas, "Sample efficient actor-critic with experience replay," *arXiv preprint arXiv:1611.01224*, 2016. 12

[130] D. Ha and J. Schmidhuber, "Recurrent world models facilitate policy evolution," in *Advances in Neural Information Processing Systems*, 2018. 12

[131] D. Isele and A. Cosgun, "Transferring autonomous driving knowledge on simulated and real intersections," *arXiv preprint arXiv:1712.01106*, 2017. 12

[132] J. X. Wang, Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos, C. Blundell, D. Kumaran, and M. Botvinick, "Learning to reinforcement learn," *arXiv preprint arXiv:1611.05763*, 2016. 12

[133] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel, "Fast reinforcement learning via slow reinforcement learning," *arXiv preprint arXiv:1611.02779*, 2016. 12

[134] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," *arXiv preprint arXiv:1703.03400*, 2017. 12

[135] M. Al-Shedivat, T. Bansal, Y. Burda, I. Sutskever, I. Mordatch, and P. Abbeel, "Continuous adaptation via meta-learning in nonstationary and competitive environments," *arXiv preprint arXiv:1710.03641*, 2017. 12

[136] A. Nichol, J. Achiam, and J. Schulman, "On first-order meta-learning algorithms," *CoRR, abs/1803.02999*, 2018. 12

[137] S. Ross and D. Bagnell, "Efficient reductions for imitation learning," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010, pp. 661–668. 12

[138] M. Bansal, A. Krizhevsky, and A. Ogale, "Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst," *arXiv preprint arXiv:1812.03079*, 2018. 13

[139] A. Y. Ng, D. Harada, and S. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *ICML*, vol. 99, 1999, pp. 278–287. 13

[140] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the Twenty-first International Conference on Machine Learning*, ser. ICML '04. ACM, 2004. 13

[141] N. Chentanez, A. G. Barto, and S. P. Singh, "Intrinsically motivated reinforcement learning," in *Advances in neural information processing systems*, 2005, pp. 1281–1288. 13

[142] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," in *International Conference on Machine Learning (ICML)*, vol. 2017, 2017. 13

[143] Y. Burda, H. Edwards, D. Pathak, A. Storkey, T. Darrell, and A. A. Efros, "Large-scale study of curiosity-driven learning," *arXiv preprint arXiv:1808.04355*, 2018. 13

[144] J. Zhang and K. Cho, "Query-efficient imitation learning for end-to-end simulated driving," in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, San Francisco, California, USA.*, 2017, pp. 2891–2897. 13

[145] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "Safe, multi-agent, reinforcement learning for autonomous driving," *arXiv preprint arXiv:1610.03295*, 2016. 13

[146] X. Xiong, J. Wang, F. Zhang, and K. Li, "Combining deep reinforcement learning and safety based control for autonomous driving," *arXiv preprint arXiv:1612.00147*, 2016. 13

[147] C. Ye, H. Ma, X. Zhang, K. Zhang, and S. You, "Survival-oriented reinforcement learning model: An effcient and robust deep reinforcement learning algorithm for autonomous driving problem," in *International Conference on Image and Graphics*. Springer, 2017, pp. 417–429. 13

[148] J. Garcıa and F. Fernández, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 1437–1480, 2015. 13

[149] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov, "Openai baselines," https://github.com/openai/baselines, 2017. 13

[150] A. Juliani, V.-P. Berges, E. Vckay, Y. Gao, H. Henry, M. Mattar, and D. Lange, "Unity: A general platform for intelligent agents," *arXiv preprint arXiv:1809.02627*, 2018. 13

[151] I. Caspi, G. Leibovich, G. Novik, and S. Endrawis, "Reinforcement learning coach," Dec. 2017. [Online]. Available: https://doi.org/10.5281/zenodo.1134899 13

[152] Sergio Guadarrama, Anoop Korattikara, Oscar Ramirez, Pablo Castro, Ethan Holly, Sam Fishman, Ke Wang, Ekaterina Gonina, Neal Wu, Chris Harris, Vincent Vanhoucke, Eugene Brevdo, "TF-Agents: A library for reinforcement learning in tensorflow," https://github.com/tensorflow/agents, 2018, [Online; accessed 25-June-2019]. [Online]. Available: https://github.com/tensorflow/agents 13

[153] A. Stooke and P. Abbeel, "rlpyt: A research code base for deep reinforcement learning in pytorch," *arXiv preprint arXiv:1909.01500*, 2019. 13

[154] I. Osband, Y. Doron, M. Hessel, J. Aslanides, E. Sezener, A. Saraiva, K. McKinney, T. Lattimore, C. Szepesvári, S. Singh, B. Van Roy, R. Sutton, D. Silver, and H. van Hasselt, "Behaviour suite for reinforcement learning," 2019. 13

B Ravi Kiran is a technical lead of machine learning team at Navya and works in designing and deploying realtime deep learning architectures for autonomous shuttles. He worked in the domain of machine learning for the past 5 years working on applications of deep learning architectures for video anomaly detection, online time series anomaly detection, hyperspectral image processing for tumor detection. He finished his PhD at Paris-Est in 2014 entitled Energetic lattice based optimization which was awarded the MSTIC prize. He has worked in academic research for over 6 years in the domain of computer vision and machine learning, and in industry for 2 years in embedded programming and 2 years in autonomous driving. During his career he has published over 40 articles and journals.

Ibrahim Sobh Ibrahim has more than 20 years of experience in the area of Machine Learning and Software Development. Ibrahim received his PhD in Deep Reinforcement Learning for fast learning agents acting in 3D environments. He received his B.Sc. and M.Sc. degrees in Computer Engineering from Cairo University Faculty of Engineering. His M.Sc. Thesis is in the field of Machine Learning applied on automatic documents summarization. Ibrahim has participated in several related national and international mega projects, conferences and summits. He delivers training and lectures for academic and industrial entities. Ibrahim's publications including international journals and conference papers are mainly in the machine and deep learning fields. His area of research is mainly in Natural language processing, Speech processing, Computer vision and autonomous driving. Currently Ibrahim is a Senior Expert of AI, Valeo.

Victor Talpaert is a PhD student at U2IS, ENSTA Paris, IP Paris, France. His PhD is directed by Bruno Monsuez since 2017. The lab speciality is robotics and complex systems. His PhD is co-sponsored by AKKA Technologies in Gyuancourt, France, through the guidance of AKKA's Autonomous Systems Team and Guillaume Trehard. This team has a large focus on Autonomous Driving (AD) and the automotive industry in general. His PhD subject is learning decision making for AD, with assumptions such as a modular AD pipeline, learned features compatible with classic robotic approaches and hierarchical temporal abstractions.

Patrick Mannion is a Lecturer in Computer Science at National University of Ireland Galway (NUIG). He previously served as an Assistant Lecturer in Computing at Galway-Mayo Institute of Technology (GMIT). In 2018, he visited the Collaborative Robotics and Intelligent Systems Institute, Oregon State University as a Fulbright-TechImpact Scholar. He completed his PhD in Machine Learning at NUIG and also holds a BEng in Civil Engineering (NUIG), a HDip in Software Development (NUIG), and a PgCert in Teaching and Learning (GMIT). His research interests include reinforcement learning, multi-agent systems, game theory, multi-objective optimisation and evolutionary computation.

Ahamd El Sallab Ahmad El Sallab is the Senior Chief Engineer of Deep Learning at Valeo Egypt, and Senior Expert at Valeo Group. Ahmad has 15 years of experience in Machine Learning and Deep Learning, where he acquired his M.Sc. and Ph.D. on 2009 and 2013 in the field. He has worked for reputable multi-national organizations in the industry since 2005 like Intel and Valeo. He has over 35 publications and book chapters in Deep Learning in top IEEE and ACM journals and conferences, in addition to 30 patents, with applications in Speech, NLP, Computer Vision and Robotics.

Senthil Yogamani is a computer vision architect and technical leader at Valeo Vision systems. He is currently focused on research and design of the overall computer vision algorithm architecture for surround-view camera visual perception in autonomous driving systems. He has over 14 years of experience in computer vision and machine learning including 11 years of experience in industrial automotive systems. He is an author of 80 publications and 50 patents. He serves in the editorial board of various leading IEEE automotive conferences including ITSC and ICVES and advisory board of various industry consortia including Khronos, Cognitive Vehicles and IS Auto. He is a recipient of best associate editor award at ITSC 2015 and best paper award at ITST 2012.

Patrick Pérez is Scientific Director of Valeo.ai, a Valeo research lab on artificial intelligence for automotive applications. He is currently on the Editorial Board of the International Journal of Computer Vision. Before joining Valeo, Patrick Pérez has been Distinguished Scientist at Technicolor (2009-2918), researcher at Inria (1993-2000, 2004-2009) and at Microsoft Research Cambridge (2000-2004). His research interests include multi-sensor perception, audio-visual analysis and computer vision.

View publication stats