The Effects of Entropy Regularization and Lyapunov Stability Constraint on Multi-Agent Reinforcement Learning for Autonomous Driving.

Thesis report

by

Mohamed Madi

to obtain the degree of
Master of Science
in Mechanical Engineering
at the Delft University of Technology

August 23, 2022

Student number: 5136539
Project duration: September 2021 - July 2022
Supervisors: Dr. Wei Pan

Abstract

High level decision making in Autonomous Driving (AD) is a challenging task due to the presence of multiple actors and complex driving interactions. Multi-Agent Reinforcement Learning (MARL) has been proposed to learn multiple driving policies concurrently to solve AD tasks. In the literature, multi-agent algorithms have been shown to outperform single-agent algorithms and rule-based algorithms. Also several techniques have been employed to facilitate convergence in policy learning such as parameter sharing and local reward design. Further, functional safety in AD has been addressed with techniques such as unsafe action-masking. However, there is a gap in the literature on the study of the effects of entropy regularization and on policies learned with closed-loop stability guarantee to solve AD tasks in MARL. In this thesis, research gaps are addressed in entropy regularization and Lyapunov stability constrained policy objectives applied to Autonomous Driving in MARL. Specifically, it is demonstrated on the lane-keeping task with 2 agents that entropy regularization improves training stability. It was also shown that in stochastic multi-agent algorithms on the lane-keeping task, a Lyapunov constrained policy objective performs better in average episode returns, success rate and collision rate than a policy objective without a Lyapunov constraint with low measurement noise perturbation. However, an algorithm with a stochastic actor performs worse than that with a deterministic actor in stability and lane center proximity on the lane-keeping task.

Acknowledgements

I would like to thank my thesis supervisor Dr. Wei Pan for his continued support and expert advice throughout the duration of my project. His advice and guidance in research direction has been invaluable.

I would also like to thank my friends and family, especially my parents, Twfik Madi and Mariann Gellért who have supported and motivated me in my pursuit of a Master of Science degree in Mechanical Engineering. I would also like to thank my fiancée Edna Ivonne Martínez Corona for her support and motivation throughout the duration of my project.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

Autonomous Driving (AD) has been proposed as a solution to transportation issues such as road accidents, access to mobility and fuel savings. AD is a promising solution to road accidents since 94% of road accidents are attributable to human-error [1]. An AD system constitutes several modules including perception, localization and mapping, navigation, motion planning and control. According to [2], perception level tasks are well understood in the AD domain but the notion of driving policies remains less understood. Moreover, AD systems have achieved high precision on perception level tasks due to advancements made in deep learning architectures [3]. The investigation of high-level decision making in autonomous driving is motivated by the nature of the driving task. The driving task involves navigating in evolving configurations of the environment and interacting with different road users. A machine learning approach can be used to learn effective driving policies that are capable of handling the complexity of the driving task. Supervised learning algorithms are not suitable for tasks such as learning driving strategies and high-level decision making [3]. In driving tasks, an AV agent is required to take optimal actions in evolving configurations of the environment at consecutive time-steps. Learning to take optimal actions in sequential decision processes is the focus of Reinforcement Learning. Furthermore, learning driving policies in the presence of multiple road users is a multi-agent problem that requires multi-agent learning [4].

1.2 Preliminaries

1.2.1 Single-Agent Reinforcement Learning

In a single agent setting, an RL agent interacts with its environment through trial and error in a sequence of time-steps to learn a policy that maximizes the agent's sum of future rewards. The agent observes its environment state and takes an action according to a policy for which it receives a reward. Single-agent reinforcement learning problems can be mathematically formalized as Markov Decision Processes (MDPs). An MDP is a tuple (S, A, P, R, γ) where:

- S is the set of all valid states.
- A is the set of all valid actions.
- $P : S \times A \rightarrow P(s)$ is the transition probability function. $P(s_{t+1}|s_t, a_t)$ that denotes the probability of transitioning into state s_{t+1} by taking action a_t in state s_t.
- $R : S \times R \rightarrow \mathbb{R}$ is the reward function that is given by $r_t = R(s_t, a_t)$.
- $\gamma \in (0, 1]$ is the discount factor which discounts future rewards.

The assumption in an MDP is that the state is fully observable by the agent, and the system has the Markov property which means that state transitions are only dependent on previous time-step states and actions and no prior history. An agent selects an action $a_t \in A$ at time-step t when in state $s_t \in S$ according to a policy π and transitions into state $s_{t+1} \in S$ with transition probability $P(s_{t+1}|s_t, a_t)$ and receives a reward $r_t = R(s_t, a_t)$. A stochastic policy is a probability distribution over actions $\pi(a|s)$. The goal of a reinforcement learning agent is to learn a policy

that maximizes the expected return. The infinite-horizon discounted return is the discounted sum of all rewards obtained from the start of an episode:

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t \tag{1}$$

τ denotes a trajectory, which is the sequence of states and actions encountered by an agent in an episode:

$$\tau = (s_0, a_0, s_1, a_1, ...) \tag{2}$$

The objective function $J(\pi)$ to be maximized is the expected return, where the expectation is with respect to the probability distribution over trajectories generated using the policy π, $P(\tau|\pi)$:

$$J(\pi) = \int_{\tau} P(\tau|\pi) R(\tau)$$

$$J(\pi) = \mathbb{E}_{\tau\sim\pi}[R(\tau)]$$

The goal of an RL agent is then to find the optimal policy π^*[5]:

$$\pi^* = \arg\max_{\pi} J(\pi) \tag{3}$$

An important concept in Reinforcement learning is the value function which is a measure of the value of a state or state-action pair. The state-value function $V^\pi(s)$ denotes the value of starting in state s and taking actions according to policy π forever after:

$$V^\pi(s) = \mathbb{E}_{\tau\sim\pi}[R(\tau)|s_0 = s] \tag{4}$$

The state-action value function $Q^\pi(s, a)$ denotes the value of starting in state s and taking an arbitrary action a, then following the policy π forever after:

$$Q^\pi(s, a) = \mathbb{E}_{\tau\sim\pi}[R(\tau)|s_0 = s, a_0 = a] \tag{5}$$

RL algorithms can be categorized into model-free and model-based algorithms. In model-free algorithms, the agent does not learn or have access to a model of its environment. In other words, the agent does not have access to state-transition functions. Model-based algorithms on the other hand require learning a model of the environment dynamics. Another branching point for algorithms is what agents learn. In value-based methods, agents learn an approximator $Q_\phi(s, a)$ for the optimal action-value function $Q^*(s, a)$. The actions are selected according to:

$$a = \arg\max_{a} Q_\phi(s, a) \tag{6}$$

An example algorithm that utilizes Q-learning is DQN [6].

In policy-based methods, the policy $\pi_\theta(a|s)$ is parameterized by θ . These algorithms typically optimize the parameters θ by gradient ascent on the objective $J(\pi)$. They can also use an approximator $V_\phi(s)$ which is used in updating the policy. An example of a policy based algorithm is A2C [7]. Actor-Critic algorithms are hybrid algorithms that combine the two families. In these

algorithms, the actor is the policy π that is used to select actions. The critic is the value function that "criticizes" the actions produced by the actor. In policy-gradient methods, the policy parameters are updated according to:

$$\theta_{k+1} = \theta_k + \alpha \nabla_{\theta_k} J(\pi_{\theta_k}) \tag{7}$$

The gradient of the objective function $\nabla_{\theta_k} J(\pi_{\theta_k})$ can be analytically formulated as an expectation:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[\sum_{t=0}^{T} \nabla_\theta \log(\pi_\theta(a_t|s_t)) R(\tau)] \tag{8}$$

The gradient can be estimated using sample trajectories:

$$\widehat{\nabla_\theta J(\pi_\theta)} = \frac{1}{|D|} \sum_{\tau \in D} \sum_{t=0}^{T} \nabla_\theta \log(\pi_\theta(a_t|s_t)) R(\tau) \tag{9}$$

where D is a set of trajectories $\tau_{i i=1,..N}$.

1.2.2 Multi-agent Reinforcement Learning

The multi-agent reinforcement learning problem can be formalized using a Markov Game (MG) under the assumption that the environment state is fully observable by the agent [5]. A Markov Game is defined by a tuple $(\mathcal{N}, S, \{A^i\}_{i \in N}, P, \{R^i\}_{i \in N}, \gamma)$ where

- $\mathcal{N} = \{1, .., N\}$ denotes the set of N agents in a game.
- S is the state space observed by all agents.
- A^i denotes the action space of the i^{th} agent. The total action space A of all agents is $A := A^1 \times A^2 \times ... \times A^N$.
- $P : S \times A \rightarrow P(s)$ is the transition probability function $P(s_{t+1}|s_t, a_t)$ that denotes the probability of transitioning into state s_{t+1} by agents taking the **joint** action a_t in state s_t.
- $R^i : S \times A \rightarrow \mathbb{R}$ is the reward function for the i^{th} agent and is given by $r_t^i = R^i(s_t, a_t)$.
- $\gamma \in (0, 1]$ is the discount factor which discounts future rewards.

In a Markov Game, each agent $i \in \mathcal{N}$ chooses its action $a_t^i \in A^i$ according to its own policy $\pi_i(a^i|s)$ based on its observation of state $s \in S$. Let a^{-i} denote the actions of all agents other than agent i. The system transitions into state s_{t+1} with transition probability $P(s_{t+1}|s_t, a_t^i, a_t^{-i})$ and each agent receives its own reward that is a function of its own actions and the actions of all other agents $r_t^i = R^i(s_t, a_t^i, a_t^{-i})$. The joint policy π is given by $\pi := (\pi^1(a^1|s), \pi^2(a^2|s), ..., \pi^N(a^N|s))$. Each agent aims to maximize its own objective that is a function of the joint policy π. The objective to be maximized is given by

$$J^i(\pi) = \mathbb{E}_{s \sim p, a \sim \pi}[\sum_{t=0}^{\infty} \gamma^t R^i(s_t, a_t)] \tag{10}$$

where $a_t = (a_t^i, a_t^{-i})$ is sampled from the joint policy π and s_t is sampled from the start-state distribution p. Consequently, the state-action and state value functions are given by

$$Q_\pi^i(s,a) = \mathbb{E}_{s\sim p, a\sim\pi}[\sum_{t=0}^{\infty} \gamma^t R^i(s_t,a_t)|s_0 = s, a_0 = a] \tag{11}$$

$$V_\pi^i(s) = \mathbb{E}_{s\sim p, a\sim\pi}[\sum_{t=0}^{\infty} \gamma^t R^i(s_t,a_t)|s_0 = s] \tag{12}$$

An important concept in multi-agent reinforcement learning is the Nash Equilibrium [5]. The Nash Equilibrium is the joint policy $\pi^* = (\pi^{1*}, ..., \pi^{N*})$ such that for any $s \in S$ and $i \in \mathcal{N}$

$$V^i_{\pi^{i*},\pi^{-i*}}(s) \geq V^i_{\pi^i,\pi^{-i*}}(s) \tag{13}$$

The Nash equilibrium is an equilibrium point from which no agent deviates if any algorithm converges. In multi-agent reinforcement learning, the performance of each agent is not only dependent on the actions taken by the agent, but also on the performance of all other agents. Several challenges arise in the MARL setting [5]. For instance, value based methods fail to converge to the Nash equilibrium in general-sum Markov games [8]. There are several other goals which have been formulated for MARL such as learning to communicate. Another challenge in MARL is the non-stationary environment. The reward and the current state not only depend on the previous state and action taken, but also on the action taken by other agents whose policies are constantly evolving over time. MARL algorithms also suffer from scalability issues, since the joint action space increases exponentially with the number of agents. Several approaches have been proposed to deal with the non-stationary environment problem in MARL [9]. These include using memory based models that save transitions in a long term memory, allowing agents to communicate with each other by message sharing, sharing a common policy function by all agents in the environment and gradient information sharing between agents during training.

Multi-agent reinforcement learning algorithms can be categorized as follows: Fully centralized (FC), fully decentralized (FD), centralized training and decentralized execution (CTDE), and networked agent learning (NL) [4]. In the fully centralized training paradigm, each agent is controlled by a central controller. The central controller has access to each agent's actions, observations and rewards. Agents can have access to these parameters as well as local policies of each agent [5]. In the centralized training decentralized execution paradigm, each agent shares information with all agents globally during training, but each agent executes its own policy locally. In the fully decentralized paradigm, there is no information exchange between agents during training and execution, and each agent learns its own policy without access to other agent policies. In Networked agent learning, each agent shares information only with its neighbors in the network. This constitutes a special case of decentralized execution/training [5].

2 Related Work

2.1 Single-Agent Reinforcement Learning Algorithms and Applications

Single-agent reinforcement learning algorithms have been explored in different driving scenarios and tasks such as lane keeping, lane changing, ramp merging, overtaking, intersections and motion planning. In [10], the authors employ a Twin Delayed Deep Deterministic Policy Gradient

algorithm **TD3** [11] as an RL baseline and compare its performance to the rule-based algorithms Intelligent Driver Model **IDM** [12] and Autonomous Emergency Braking Model **AEB** [10] in the single-agent setting. The authors compare the performance of these agents in an unsignalized intersection scenario with Human Driven Vehicles (HDVs) with evaluation metrics *success rate* and *average duration time*, to test safety and efficiency, respectively. It was concluded that the TD3 algorithm after training performed better than the rule-based algorithms with at least a 90% success rate and better average duration time in a deterministic test. In a stochastic test, the TD3 algorithm achieved a higher success rate than IDM and AEB but a higher average duration time compared to AEB in 2 of the 3 tested tasks (left turn, right turn, going straight).

To optimize traffic flow at unsignalized intersections, the authors in [13] implement a single-agent RL algorithm based on **PPO** [14] that serves as an intersection management control unit in a mixed traffic setting with Connected and Autonomous Vehicles (CAVs) and Human Driven Vehicles (HDVs). The RL agent sends stop commands to CAVs on routes with higher priority to allow for HDVs on lower priority routes to cross an intersection optimizing overall performance. This work differs from DRL based approaches that deal with individual AV navigation and control such as [10]. Their proposed algorithm termed Courteous Virtual Traffic Signal Control (CVTSC) is evaluated on a 3-way intersection against road signs and adaptive traffic light controllers in the SUMO [15] traffic simulator on the traffic management task. The CVTSC algorithm outperforms the baselines in terms of mean and standard deviation of *travel time* of vehicles on simulated and real traffic. The throughput of intersections is also shown to increase with increasing CAV percentage among the traffic participants.

In [16], the authors employ a single-agent Double Deep Q Network **DDQN** [17] algorithm for AV navigation through an unsignalized intersection in the presence of pedestrians. The algorithm also utilizes a belief update LSTM (Long Short-Term Memory) model and a future collision prediction LSTM model. The belief update network is used to output a perceived state representation, given imperfect observations of pedestrian states. The imperfect observations consist of noisy inputs that are generated by adding Gaussian noise to the input features. The future collision prediction model utilises an LSTM network to predict the future location of pedestrians in the neighborhood of the ego vehicle. The future locations are used to extract trajectories which are compared to the ego vehicle trajectory and if a collision results from the ego vehicle action, the unsafe action is masked and a deceleration action is output by a high-level controller. The authors compared the performance of their algorithm SRL to a rule based agent that utilizes Time To Collision TTC in the CARLA simulator on a 4-way unsignalized intersection scenario. Compared to the rule-based agent, the SRL agent achieved 0% collision episodes and 92% successful episodes compared to 62% collision episodes and 38% successful episodes for the rule-based agent on an intersection scenario that was similar to the one seen in training. However, the rule-based agent was able to achieve a higher average speed and a faster average intersection crossing time. The test results also show that an RL-agent without the future collision predictor and the belief update model cannot guarantee collision free navigation, compared to the SRL agent with 0% collision episodes.

In [18], authors test the **PPO** [14] and **SAC** [19] algorithms in different maps that are composed of road blocks including straight roads, roundabouts and intersections in random configurations.

It was shown that increasing the number of scenes in which the agents were trained increases the generalizability of the trained agents. Also in [18], the Reward Shaping and Lagrangian variants of both PPO and SAC (**PPO-RS**, **SAC-RS**, **PPO-Lag**, **SAC-lag**) and Constrained Policy Optimization (**CPO**) [20] algorithms were trained and tested in safe exploration tasks in diverse scenarios with static and movable obstacles. It was shown that SAC baselines achieve higher *success rates* compared to PPO baselines.

2.2 Multi-Agent Reinforcement Learning Algorithms and Applications

MARL algorithms have been applied to scenarios such as car-following, lane-overtaking, highway merging and intersection navigation. In [21], the authors propose a decentralized-execution MARL algorithm **MA2C** for a highway on-ramp merging scenario. The authors formulated the on-ramp merging problem as a Partially Observable MDP with networked agents. Their proposed algorithm uses an Actor and Critic Network for each agent that share a FC layer. Furthermore, the policy network parameters are shared among all agents during training. Learning is therefore centralized by parameter sharing. The authors test their algorithms against several MARL baselines, **MAA2C** [22], **MAPPO** [23] and **MAACKTR** [23] that share parameters of the policy among all agents in mixed traffic, which means in the presence of AVs and Human Driven Vehicles (HDVs). The algorithm also utilizes a local reward function, action masking and a priority-based safety supervisor. The reward for each agent is an average reward dependent on the vehicles in the neighborhood of the ego-vehicle. The authors argue that averaging global rewards of all vehicles leads to a large communication overhead and the credit assignment problem. Invalid action masking is used in this algorithm to assign close to zero probabilities to invalid/unsafe actions to prevent undesirable system behaviors and invalid policy updates. The authors observed better performance in the shared network design compared to the separate actor and critic network design. The performance difference was attributed to the fact that in a separate network, the actor may not produce optimal actions until the critic network is well trained. Tests in different traffic densities shows better performance of MA2C compared to benchmarks, evaluated using *collision rate* and *average speed* of vehicles which are indicative of safety and efficiency.

In [24], Parameter Sharing DDPG **PS-DDPG** is used to learn lane keeping and over-taking behaviors in autonomous vehicles in an all AV agents setting. The proposed network is a single Actor-Critic Network that is shared for all agents. The proposed algorithm also uses a single replay buffer that is shared for all agents. The authors argue that sharing a single Actor-Critic network in this way allows for homogeneous agents to benefit from the learning experience of other agents, where the network is updated by each agent for every time step. The authors argue that this would increase the speed of training. Moreover, a single shared replay buffer would allow the agents to learn from a diverse set of experiences, which is possible due to the homogeneous nature of the trained agents. The authors compared the performance of PS-DDPG against vanilla **DDPG** [25] for multiple agents, in the Gym-TORCS [26] simulation platform. In the lane-keeping task, the rewards obtained by PS-DDPG were robust to the increase in the number of agents in training, as opposed to DDPG which showed a decrease in the rewards obtained with an increase in the number of trained agents. PS-DDPG also required a fewer number of episodes to learn lane-keeping and overtaking behavior.

In [27], the authors propose a multi-agent advantage actor critic **MA2C** algorithm for the lane-

changing task that is similar to the algorithm used in [21]. The algorithm was evaluated in a modified gym-based highway-env simulator with mixed traffic participants. The algorithm uses parameter sharing and a multi-objective reward function that takes into account driver comfort. The HDVs utilize IDM [12] for longitudinal control and the MOBIL [28] model for lateral control. Similar to [21], the MAA2C algorithm in [27] uses an Actor-Critic network where the Actor and Critic networks share a hidden layer. The reward function contains an additional term for driving comfort that penalizes rapid accelerations/decelerations. The authors also evaluate their proposed algorithm with varying human driver aggressiveness and varying traffic densities. Their results show that parameter sharing achieved higher episode rewards and lower variance compared to the non-parameter sharing variant of the algorithm. The authors also evaluated their algorithms against **MADQN** [29], **MAACKTR** [30] and **MAPPO** [14]. MA2C outperformed all baselines in low traffic density. MADQN achieved higher average rewards in medium and high traffic densities compared to MA2C but with a higher variance of rewards which can cause unstable training.

In [31], the authors propose a multi-agent delayed A3C **MAD-A3C** algorithm to train multiple homogeneous agents (all AVs) in a signalized intersection scenario. Their algorithm utilizes two sub-modules for predicting acceleration and steering angle separately. Each sub-module is composed of a convolutional neural network that encodes information from a series of images that represent the navigable space, path, obstacles and traffic signs. The traffic signs and obstacles (other traffic participants) are color-coded in these images based on their state and their priorities. Each sub-module outputs a mean value and state value, and accelerations or steering angles are sampled from Gaussian distributions centered at those mean values. The algorithm was trained and evaluated in different difficulty level intersections and also compared to the **TTC** [32] algorithm, which is a rule-based algorithm. The evaluation metrics used are *reaches*, *crashes*, *off-roads* and *time-overs*. The TTC algorithm computes the TTC of vehicles approaching from the left and right adjacent lanes based on the time to reach an imaginary line that coincides with the longitudinal axis of the ego-vehicle. It was shown that with a higher number of simultaneous traffic participants, the MAD-A3C algorithm performs better than the rule-based TTC algorithm. Experiment results in [31] show that the proposed network is able to learn the right of way rule based on the traffic signs and the priority to the right rule. This was validated using an *infraction* evaluation metric, which measures the percentage of incidents in which a vehicle crosses an intersection before vehicles with a higher priority.

It is worth mentioning the work in [33], where the authors propose a game theoretically optimal auction, a game-theory based approach to navigating unsignalized intersections. The **GamePlan** algorithm proposed determines which vehicles at an intersection should move first based on driver observed vehicle trajectories and velocities which indicate aggressiveness. The authors argue that DRL based methods such as in [31] cannot guarantee collision free and deadlock free navigation and they do not generalize well to unseen environments. The assumptions made in the GamePlan algorithm are that agents take turns navigating intersections one at a time and that they do not communicate with each other. The authors evaluated GamePlan against DRL based methods including [31] and other RL and game-theoretic approaches in a 4 way multi-lane intersection, using *success rate*, *deadlock rate* and *collision rate* as evaluation metrics. GamePLan was shown to have zero collisions and deadlocks. Limitations in this paper are that GamePLan does not plan beyond the turn-based ordering.

In addition to algorithms proposed for navigating challenging driving scenarios, there are simulators that have been proposed for MARL research in driving scenarios. One such platform is **SMARTS** [4]. The goal of SMARTS is to provide a MARL dedicated research platform that provides diverse driver behavior models for social vehicles that result in diverse interactions. SMARTS provides challenging driving scenarios such as un-protected left turns, intersections, roundabouts and double merge. The authors also provide a classification of MARL algorithms applied to AD **M0-M5**, with M0 agents following hard coded rules to M5 agents that reason about the repercussions of local actions on global traffic state such as congestion. SMARTS utilizes "bubbles" where control of social vehicles is handed over from background traffic provider to social agents to produce diverse and realistic interactions at locations of interest. The authors of SMARTS compared the performance of the following algorithms: Independent learning algorithms **DQN** [34], **PPO** [14], centralized training algorithms **MAAC**, **MF-AC** [35], **MADDPG** [36], **Networked Fitted-Q** [37] and fully centralized **CommNet** [38]. These algorithms were tested in the following scenarios: two-way traffic, double-merge and unprotected intersection. The authors evaluate the performance of these algorithms using *collision rate* and *completion rate* and it was shown that MADDPG performed better than most algorithms especially in the intersection scenario. SMARTS also provides behavioral performance metrics such as safety, stability, agility and control diversity.

The double-merge scenario was also used in [2] to test a multi-agent algorithm that decomposes policy learning into learning a policy for Desires and trajectory planning with hard constraints which is not learned. The authors propose the decomposition of the policy in this form to mitigate the large variance of the gradient estimate that results from a low probability of accident occurrence. The desires policy $\pi^D s$ is a mapping from states to a discrete set of desires, which are used in a cost function. The trajectory planning function π^T selects a trajectory that minimizes the aforementioned cost function. This decomposition also leads to driving trajectories that provide comfort whilst maintaining functional safety.

To address the problems of generalization to unseen environments and safe exploration, the authors in [18] propose the **MetaDrive** simulation platform. It differs from SMARTS [4] in the additional areas of generalization and safe exploration tasks which are designed to test single-agent algorithms and in the map generation process which enables the generation of many different driving scenarios. MetaDrive provides the following MARL benchmarking scenarios: toll-gate, bottleneck, parking lot, roundabout, intersection. The MARL algorithms Independent Proximal Policy Optimization **IPPO** [14], and centralized critic algorithms **Concat-CCPPO** [39] and Mean Field **MF-CCPPO** [35] were trained and evaluated in these scenarios and the success rates of the algorithms were compared. It was shown that MF-CCPPO outperformed IPPO in most scenarios and that Concat-CCPPO performed poorly compared to the other baselines. This was attributed to the concatenation method which makes learning difficult by expanding input dimensions.

In [40], the author presents the Multi-Agent Connected Autonomous Driving **MACAD-Gym** simulation platform to enable research on multi-agent reinforcement learning algorithms in the autonomous driving domain. The author trained independent RL agents that utilize the **IMPALA** [41] algorithm to navigate a partially observable 3-way signalized (stop-sign controlled)

intersection with homogeneous agents. The algorithm utilizes raw camera observations as input to a Convolutional Neural Network followed by a fully connected layer with weight sharing of this layer among all agents. Training is centralized in this manner due to the sharing of layer weights among all agents. The MACAD-Gym platform also provides training in heterogeneous environments under different settings such as cooperative and competitive settings. The performance of the trained agents was evaluated using cumulative mean episodic rewards.

In [42], a parameter-sharing Soft Actor Critic **PS-SAC** algorithm is used to learn driving policies for homogeneous agents AVs navigating in a roundabout scenario. All agents in the environment share the same copy of a policy that is updated using the experiences collected by all agents from a shared experience replay buffer. During execution, each agent uses a copy of the trained policy to map its local observations to actions. The authors in [42] classify their algorithm as centralized-training decentralized-execution as the learning is centralized with parameter sharing. The performance of the algorithm was evaluated using *success rate* and *collision rate* in the roundabout scenario with different traffic densities. The robustness of the algorithm was evaluated by varying noise added to the accelerations of other agents and varying road geometry. The success rate of the PS-SAC algorithm was higher in less dense traffic situations in experiments with varying noise and road geometry. Additionally, the success rate of the trained policies in the evaluation experiments did not decrease significantly due to the addition of Gaussian noise which suggests that the policy is robust against deviations from expected behavior of other vehicles seen during training.

In [43], authors use a **Graphical Convolutional Network** combined with **DDPG** to train connected and autonomous vehicles (CAVs) for bottleneck congestion mitigation. Position and velocity information of both HDVs and CAVs is fed into a fully-connected network encoder to generate node embeddings. The node embeddings are then fed into a graphical convolutional network along with an adjacency matrix that contains information on the network topology. The GCN outputs embeddings that are then input to an actor-critic network that utilizes the DDPG algorithm. The actor network outputs individual accelerations for each CAV. This algorithm falls under the fully centralized paradigm due to the presence of a central controller. The authors compared the performance of the GCN-DDPG algorithm with that of the rule-based IDM model on two highway bottleneck networks, one moderately congested and the other severely congested. It was observed that the DRL based controller achieved higher episode rewards compared to the rule-based controller. Furthermore, the duration and intensity of the congestion was smaller for the DRL based controller compared to the rule-based controller. This was done by plotting a time-space heat map of *mean traffic speeds*. The *throughput* of the DRL based controller was also higher than that of the rule-based controller.

In [44], the authors also implement a **Graphical Convolutional Network** to train CAVs to navigate in mixed traffic settings. Their algorithm utilises **PPO** with critic networks sharing parameters and actor networks sharing parameters for the CAV agents. The relationship between CAVs and neighboring vehicles is captured using a dynamic adjacency matrix. Furthermore, an attention mechanism is added to selectively integrate features from neighboring CAVs. This algorithm differs from the one used in [43] which utilizes one network to predict the actions of all agents collectively and the observations of all agents are fed into one network. This constitutes a

fully centralized scheme. On the other hand, the algorithm in [44] has separate actors and critics for each agent where only the local observations of neighbors in the network are input to each agent network. Therefore, the execution of this algorithm is decentralized . The authors compared the performance of their algorithm to that of DDPG, PPO, MADDPG, and MAPPO [14] on the car-following, unsignalized intersection navigation and merging scenarios in the flow simulation platform. The car following task was evaluated on a ring network where it was shown that CAVG achieved the lowest accelerations and highest returns. This was indicative of better shock-wave mitigation compared to the other algorithms. In the intersection navigation task, a figure eight network was used to test the algorithms for safety and efficiency. The CAVG algorithm did not achieve the highest velocity but had the lowest accelerations and highest returns which showed that the algorithm is able to sacrifice speed for safer navigation of intersection. The authors also conducted experiments with increasing CAV penetration rates. It was shown that the performance of CAVG was robust to increased penetration rates compared to MADDPG and DDPG which showed an initial increase in return followed by a drop in return. The authors attributed this behavior to a a larger policy space to explore with increasing number of actors.

In [9], the author presents centralized-training decentralized-execution algorithms, Multi-Agent Message Sharing Network **MA-MeSN** and Multi-Agent Broadcast Network **MA-BoN**, that use **DQN** as the baseline model. These algorithms both use a message generating network and a behavior policy network. For each agent, the message network in MA-MeSN generates message actions from other agents private observations. These messages are concatenated and fed to the policy network along with the agent's private observations to evaluate a state-action-message value function. In MA-BoN, the partial observations of all agents are used to generate a single broadcast message that is used by all agents to generate a behavior action. To achieve decentralized execution, two techniques are implemented. In the first, behavior cloning is used to train the decentralized agent action policies, using the centralized policy that was trained with the message sharing network. In the second technique, the behavior policy is used with a memory module that is local to each agent. The memory module is an LSTM network that maps each agent's private observation to messages generated by other agents. The trained LSTM networks are used during execution to simulate messages generated by other agents in the environment. The performance of these algorithms is compared to **DIAL** [45] and **CommNet** in terms of the cumulative reward achieved per episode in a highway driving scenario that was simulated using robots on a treadmill. MA-MeSN and MA-BoN achieved higher cumulative rewards compared to DIAL and CommNet in training on the highway environment. MA-BoN also shows better scalability (slower drop in average cumulative rewards) compared to CommNet and DIAL as the number of agents in the environment increases. In terms of the decentralized execution techniques, Cooperative Distributed Behavior Cloning achieved higher cumulative rewards compared to other techniques such as DQN with Stabilized Experience Replay. With CoDBC, each agent's independent policy is trained iteratively while using the centralized cooperative policy obtained during training for other agents. In contrast, the memory module is able to achieve higher cumulative rewards compared to CoDBC in a fewer number of episodes. For a classification of algorithms surveyed in the literature, refer to Table 2 in the Appendix.

2.3 Algorithms overview

Many multi-agent reinforcement learning algorithms utilized in experiments build upon single-agent variants. Deep Q Network (DQN) [34] is an off-policy value-based method, where a Q-function is learned by minimizing the following loss function:

$$L(\theta) = \frac{1}{|D|} \sum_{d} (r(s,a) + \gamma \max_{a'} Q_{\theta'}(s',a') - Q_{\theta}(s,a))^2 \tag{14}$$

where D refers to a set of transition tuples used to update the Q-function. The actions can be taken based on epsilon-greedy action selection, where the action that maximizes the Q-function approximation is selected with a probability of (1 - ϵ). Multi-agent algorithms that are extensions of this algorithm are MADQN, evaluated in [27] and MA-MeSN and MA-BoN, evaluated in [9]. Deep Deterministic Policy Gradient (DDPG) [25] is an off-policy actor-critic algorithm that learns a Q-function and a deterministic policy. The policy is updated by gradient ascent on the policy parameters with the following gradient estimate:

$$\nabla_{\theta} \frac{1}{|D|} \sum_{\tau \in D} Q_{\phi}(s, \mu_{\theta}(s)) \tag{15}$$

The Q-function is learned by gradient descent on the Q-function parameters with the following gradient estimate:

$$\nabla_{\phi} \frac{1}{|D|} \sum_{\tau \in D} (Q_{\phi}(s,a) - y)^2 \tag{16}$$

where y denotes the target and is given by:

$$y(r, s', d) = r + \gamma(1-d) Q_{\phi_{targ}}(s', \mu_{\theta_{targ}}(s')) \tag{17}$$

where $targ$ denotes target networks whose parameters are updated using the main network parameters by polyak averaging. Multi-agent variants of DDPG, namely PS-DDPG and GCN-DDPG were evaluated in [21] and [43] respectively. The TD3 [11] algorithm is also an off-policy actor-critic algorithm that learns a deterministic policy in continuous action spaces. The algorithm learns two Q-functions and uses the smaller of the two in the targets used to update the Q-functions. The target is given by:

$$y(r, s', d) = r + \gamma(1-d) \min_{i=1,2} Q_{\phi_{i,targ}}(s', a'(s')) \tag{18}$$

where a' is the action obtained from a target actor network with the addition of Gaussian noise that serves to smooth out the Q-function values (target policy smoothing). The policy is updated by gradient ascent on the policy parameters using one of the Q-functions as done in (15). Proximal Policy Optimization [14] is an on-policy algorithm that utilizes a stochastic actor. The policy parameters are updated by maximizing the following objective via gradient ascent:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|D_k T|} \sum_{\tau \in D_k} \sum_{t=0}^{T} \min\left(\frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t))\right) \tag{19}$$

where A is an advantage estimate, ϵ is a hyperparameter and g is piece-wise defined function that is a function of A and ϵ. The objective function aims to take the largest step possible in updating policy parameters without causing a collapse in performance. Multi-agent variants of PPO,

namely MF-CCPPO and Concat-CCPPO were evaluated in [18], and GCN-PPO was utilized in [44]. SAC [19] is an off-policy actor-critic algorithm that utilizes maximum entropy reinforcement learning. The policy objective function is given by:

$$J(\pi) = \mathbb{E}_{\tau\sim\pi}[\sum_{t=0}^{\infty} \gamma^t (R(s_t, a_t, s_{t+1}) + \alpha\mathcal{H}(\pi(a_t|s_t)))] \tag{20}$$

where $\mathcal{H}$ denotes the entropy of the policy and is given by:

$$\mathcal{H}(\pi(a_t|s_t)) = \mathbb{E}_{a\sim\pi}[-\log(\pi(a_t|s_t))] \tag{21}$$

The policy objective maximizes the expected future rewards and expected future entropy of the policy. This prevents policies from converging early to bad local optima and ensures sufficient exploration. The policy update step is given by:

$$\nabla_\theta \frac{1}{|D|} \sum_{\tau\in D} (\min_{i=1,2} Q_{\phi_i}(s, a_\theta(s)) - \alpha \log(\pi_\theta(a_\theta(s)|s))) \tag{22}$$

where $a_\theta(s)$ is an action sample. The action sample is obtained by a deterministic function of state, policy parameters and Gaussian noise. SAC also learns two Q-functions and the Q-function update is given by:

$$\nabla_\phi \frac{1}{|D|} \sum_{\tau\in D} (Q_\phi(s, a) - y)^2 \tag{23}$$

where the target y is give by:

$$y(r, s', d) = r + \gamma(1 - d)(\min_{i=1,2} Q_{\phi targ,i}(s', a'_\theta(s') - \alpha \log(\pi_\theta(a'_\theta(s'), s'))) \tag{24}$$

In [21], the authors propose a Multi-Agent Actor Critic (MA2C) algorithm with a shared actor critic network. The objective function to be maximized for the network is given by:

$$J(\theta_i) = J^{\pi_{\theta_i}} - \beta_1 J^{V_{\phi_i}} + \beta_2 \mathcal{H}(\pi_{\theta_i}(s_t)) \tag{25}$$

where β_1 and β_2 are weighting coefficients for the value-function objective and the entropy regularization terms, and J^π and J^V denote the policy and value function objectives respectively. The policy objective that is maximized is given by:

$$J^{\pi_{\theta_i}} = \mathbb{E}_{\pi_{\theta_i}}[\log(\pi_{\theta_i}(a_{i,t}|s_{i,t}))A_{i,t}^{\pi_{\theta_i}}] \tag{26}$$

where A is an advantage function. The Value function objective that is to be minimized is given by:

$$J^{V_{\phi_i}} = \min_{\phi_i} \mathbb{E}_{D_i}[r_{i,t} + \gamma V_{\phi_i}(s_{i,t+1}) - V_{\phi_i}(s_{i,t})]^2 \tag{27}$$

This differs from the SAC [19] algorithm as the policy objective does not explicitly maximize the entropy of the policy, and the entropy term is weighted against the value function objective. Also, the value function target does not include the entropy term. The authors in [21] do not perform an ablation study on the effect of including the entropy term in the overall objective function. Further, the authors in [27] utilize a multi-agent advantage actor critic algorithm that has

the same value objective and policy objective functions utilized in [21], and no entropy regularization term. The MAA2C algorithm evaluated in [21] is a parameter sharing variant of Advantage Actor Critic (A2C) [7]. A2C is an on-policy actor-critic algorithm that utilizes an n-step advantage estimate in policy updates. The n-step advantage estimate is given by:

$$A_\phi(s,a) = \sum_{k=0}^{n-1} \gamma^k r_{t+k+1} + \gamma^n V_\phi(s_{t+n+1}) - V_\phi(s_t) \tag{28}$$

The policy parameters of the actor are updated using the following gradient estimate:

$$\nabla_\theta J(\theta) = \sum_t \nabla_\theta \log(\pi_\theta(s,a)) A_\phi(s,a) \tag{29}$$

The value-function parameters are updating with the following loss function:

$$L(\phi) = \sum_t (A_\phi(s,a))^2 \tag{30}$$

A3C [7] is similar to A2C except the actor and critic networks can be updated asynchronously. A multi-agent variant of this algorithm, MAD-A3C, was evaluated in [31]. In A3C, multiple copies of a global actor-critic network are trained in parallel on multiple instances of the environment. The gradients obtained by the different learners are used to update a global copy of the actor and critic networks. In A3C, the learners can update the global copy as soon as the gradients are computed, and hence the gradient updates for each learner can be asynchronous. Further, A3C utilizes entropy regularization, where the policy update is given by:

$$\nabla_\theta J(\theta) = \sum_t \nabla_\theta \log(\pi_\theta(s,a))(A_\phi(s,a) + \beta \mathcal{H}(\pi_\theta(s_t)) \tag{31}$$

Parameter sharing variants of single agent algorithms MAA2C, MAPPO and MAACKTR [23], DDPG, SAC were evaluated in [21, 24, 42] which involves learning a single shared policy for all agents during training. The policy update rules are the same for the single agent variants of these algorithms except that a single policy is shared among homogeneous agents, and each agent's experiences are used to update the shared policy during learning. The algorithms in [21] and [27] can be classified as centralized-training decentralized-execution algorithms with networked agents. In [23], parameter-sharing is described as the most centralized learning method. Although, the policy parameters are shared among all agents, each agent only shares its local observations with neighbors in its network. In [4], the authors evaluate Multi-Agent Deep Deterministic policy Gradient (MADDPG)[36] algorithm, a multi-agent off policy actor-critic algorithm that utilizes a centralized critic learned for each agent and separate actors that are executed independently during test time. Each agent learns a Q-function by minimizing the following loss:

$$L(\theta_i) = \frac{1}{S} \sum_j (y^j - Q_i^\mu(\mathbf{x}^j, a_1^j, .., a_N^j))^2 \tag{32}$$

where $\mathbf{x}$ denotes the full state information and a_i^j denotes the i^{th} agents actions in a sample $j \in S$ where S is the set of samples. The target y is computed using:

$$y^j = r_i^j + \gamma Q_i^{\mu'}(\mathbf{x}^{j\prime}, a_1', .., a_N')|_{a_k' = \mu_{\mathbf{k}'}(o_k^j)} \tag{33}$$

where μ' denotes the target actor, k denotes the agent index and o denotes agent local observation. Each agent updates its actor using the following gradient estimate:

$$\nabla_{\theta_i} J = \frac{1}{S} \sum_j \nabla_{\theta_i} \mu_i(o_i^j) \nabla_{a_i} Q_i^{\mu}(\mathbf{x}^j, a_1^j, .., a_N^j)|_{a_k = \mu_{\mathbf{k}}(o_k^j)} \tag{34}$$

Other Multi-Agent algorithms that have been explored in the previous works mentioned include CommNet [38] and Networked-fitted Q [37]. CommNet is a fully centralized multi-agent algorithm that learns a mapping from all agent states to all agent actions. It additionally utilizes a communication channel which the agents can access.

2.4 Stability in MARL

Multi-Agent RL algorithms implemented in previous experiments on driving tasks neglect the guarantee of closed-loop stability of the learned control policies. A dynamical system is stable if the system state trajectory starts in the vicinity of an equilibrium point and stays close to it [46]. Stability is closely related to safety, robustness and reliability and should therefore be considered when deploying agents in safety-critical applications such as the driving task. Furthermore, it is necessary for policies in driving tasks to be stable and resilient against disturbances and measurement noise. It can be hypothesized that policies learned for AD tasks with closed-loop stability guarantee are resilient to external disturbances and measurement noise. In [47], the authors utilize Lyapunov's method in control theory to propose a single-agent actor-critic algorithm that can guarantee the closed-loop stability of learned policies. Their proposed framework (Lyapunov Actor-Critic) utilizes a Lyapunov critic function and the following policy objective that is based on Lyapunov's energy decreasing condition $L_{t+1} - L_t < 0$:

$$J(\theta) = \mathbb{E}_{(s,a,s',c)\sim\mathcal{D}}[\beta(\log(\pi_\theta(f_\theta(\epsilon, s|s))) + \mathcal{H}_t) + \lambda(L_c(s', f_\theta(\epsilon, s')) - L_c(s, a) + \alpha c)] \tag{35}$$

where β and λ are temperature parameters, $\mathcal{H}_t$ is an entropy lower bound. L_c denotes the Lyapunov critic function and f_θ denotes a deep neural network parametrized by θ that is a function of state s and Gaussian noise ϵ. In [47], the performance of LAC was compared to SAC and Lyapunov constrained PPO on five robotic control problems including CartPole and HalfCheetah. It was shown that LAC converged stably in all experiments with the lowest variance. To evaluate stability, the state trajectories were observed in relation to reference trajectories on the problem of Gene Regulatory Networks. It was shown that LAC was better able to track the reference trajectories whereas SAC either diverged or oscillated around the reference trajectories. In another experiment, LAC outperformed the other algorithms in robustness to external disturbances. In, [46], the authors utilize a Lyapunov constrained policy objective formulated for a decentralized multi-agent Soft Actor Critic algorithm and is given as:

$$\begin{aligned} J_{\pi_i}(\phi_i) = \mathbb{E}_{s_{i,t}\sim D}[\mathbb{E}_{\pi_{\phi_i}}[\alpha_i \log(\pi_{\phi_i}) - Q_{\theta_i}(s_{i,t}, a_{i,t}, \bar{\mathbf{a}}_{\mathcal{N}i,t})] \\ + \beta \mathbb{E}_{\pi_{\phi_i}}[-Q_{\theta_i}(s_{i,t+1}, a_{i,t+1}, \bar{\mathbf{a}}_{\mathcal{N}i,t+1}) + l_\delta ||a_{i,t}^{\pi_{\phi_i}} - a_{i,t}||_2 + Q(s_{i,t}, a_{i,t}, \bar{\mathbf{a}}_{\mathcal{N}i,t}) + \psi(s_t)]] \end{aligned} \tag{36}$$

where β is a temperature parameter that controls the Lyapunov term and $\bar{\mathbf{a}}_{\mathcal{N}i,t+1}$ denotes the mean action of neighbors of agent i in a Network. It was shown in [46] that the policies learned with the Lyapunov constrained objective were resilient to measurement noise in the form of Gaussian noise added to the states of the agents in a consensus control problem. In another exper-

iment, external disturbances were added to the control actions of agents trained with the Lyapunov constrained policy objective in the form of Gaussian noise. On the consensus control problem, the agents could still maintain consensus even under external disturbances [46].

3 Research Goals

3.1 Research Findings

Based on an investigation into the different algorithms proposed to solve driving tasks for Autonomous Vehicles, the following key findings are present. Firstly, Reinforcement Learning algorithms outperform rule-based algorithms in both the single-agent and multi-agent frameworks. In general, Reinforcement Learning algorithms achieve higher success rates and lower collision rates than rule-based algorithms as noted in [10, 16]. However, it was shown that rule-based algorithms outperform Reinforcement Learning algorithms in terms of average speed, which indicates higher efficiency. This conclusion drawn on efficiency however can be contested by the results in [43], which show that Reinforcement Learning algorithms are better able to mitigate bottleneck congestion compared to rule-based algorithms as measured by mean traffic speeds and throughput. Moreover, Reinforcement Learning algorithms scale better than rule-based algorithms as the number of traffic agents increases [31]. Another key finding is that multi-agent algorithms in general outperform single-agent algorithms that are applied to multi-agent settings as observed in [4, 18, 24, 44], which are usually referred to in the literature as independent learners. The assumption made in independent learning is that the other agents constitute part of the environment, and their policies are not taken into account when taking actions in the environment. Independent learners suffer from the non-stationarity problem in multi-agent settings. To overcome the non-stationarity problem, multiple algorithms utilize a parameter sharing variant of single-agent algorithms as was done in [21, 24, 27, 40, 42, 44]. Furthermore, it was also shown in [31, 40] that having a single actor-critic network with parameter sharing results in better performance compared to separate actor and critic networks due to the delayed convergence of critic networks which can impede the performance of these algorithms. An important issue that was addressed in RL algorithms used to solve navigation tasks was functional safety [2]. This is due to the fact that RL algorithms optimize an objective function by estimating a gradient and low-accident probabilities result in high variance gradient estimates which prevents multi-agent algorithms from converging to optimal policies [2]. Further, it was shown in [16, 33] that reinforcement learning algorithms alone cannot guarantee collision free navigation. In [16], the RL algorithm had to be augmented with a future collision detector to mask unsafe actions. Unsafe action masking was also used in [21].

3.2 Research Gaps

From the previous discussion, it is evident that MARL algorithms outperform single-agent algorithms when multiple agents learn concurrently in their environment. Also, several techniques have been employed to facilitate convergence of these algorithms to an optimal policy such as parameter sharing. There are also several techniques used in the literature to improve safety of learned policies such as unsafe action masking. However, there is a gap in the literature on the study of the stability of learned policies for autonomous driving tasks. In this report, stability of algorithms during training is used to refer to the variance in the performance of algorithms dur-

ing training observed across different random seeds or hyper-parameters. Stability in testing is used to refer to the stabilizing characteristic of learned policies when agents are subject to measurement noise or external disturbances. In [21, 27], parameter-sharing of actor-critic networks was shown to achieve lower variance across runs with different random seeds compared to a non-parameter sharing variant of the MA2C algorithm. It was shown in [21, 27] that a local reward design achieved lower variance across different training instances compared to a global reward design. In [4], the authors compare the performance of several state-of-the-art MARL algorithms evaluated on scenarios such as the double merge scenario and compute several behavior metrics including a stability metric. The stability metric implemented in [4] is the average variance in distance to the center of the lane across all agents over all episodes. In [42], the authors evaluate the robustness of a parameter-sharing multi-agent SAC algorithm by adding Gaussian noise to the accelerations of other agents in a round-about scenario. They evaluate the performance of PS-SAC in success rate across varying traffic densities and varying standard deviation of noise and observe that the success rate does not decrease significantly with increasing standard deviation of Gaussian noise. In [44], it was shown that MARL algorithms were able to stabilize vehicle velocities and mitigate shock-waves in a car-following scenario in a ring road.

Furthermore, there exists a research gap in the study of the effect of entropy regularization in multi-agent algorithms on performance in driving tasks. Soft Actor Critic (SAC) is a single agent algorithm that utilises maximum entropy reinforcement learning and was used to learn driving in random road configurations in [18] for a single-agent. According to [19], the entropy augmented policy objective incentivizes the policy to explore more widely and it can capture multiple modes of optimal behavior. It was shown in [19] that SAC outperforms other single-agent algorithms DDPG, PPO and TD3 in continuous control tasks in terms of learning speed and average episode returns. The authors in [19] also show that utilizing a stochastic actor with entropy regularization improves stability in training compared to a deterministic actor. Stability in this case refers to stability across different random seeds. SAC performed more consistently than a deterministic counterpart which does not maximize the entropy of the policy. In [21], the authors augment the policy objective with an entropy regularization term, but the effects of this term on performance of learned policies was not evaluated. Further, a multi-agent variant of A3C [7] is evaluated in [31] to train AV agents in a signalized intersection scenario. According to [7], inclusion of the entropy regularization term prevents early convergence to sub-optimal deterministic policies. The authors in [31] do not perform a study to investigate the effect of the entropy term in the policy objective on algorithm performance during training. These aforementioned gaps give rise to the following thesis questions:

1. Can the performance of a multi-agent reinforcement learning algorithm be improved with entropy regularization on the lane-keeping task?

In addition, a research gap exists in the study of stable control policies in MARL algorithms applied to autonomous driving tasks. Inspired by the work in [47, 46] on RL algorithms with stability guarantee utilizing a Lyapunov-constrained policy objective, the following thesis questions can be asked:

2. Can policies trained with a Lyapunov constraint show stabilizing behavior when subject to

measurement noise on the lane-keeping task?

3. Can Lyapunov constrained policy learning show improved performance during training on the lane keeping task?

4 Methods

4.1 Simulation Environment

For the experiments, the SMARTS [4] simulation platform was chosen since it is a dedicated multi-agent reinforcement learning research platform for autonomous driving with diverse scenarios and customizable observation and action spaces. For the experiments, the straight road scenario was chosen as the driving environment due its simplicity and an all AV agents setting was chosen to limit the non-stationarity introduced from social vehicles interacting with AV agents in the environment. The goal of the AV agents is to reach a target position at the end of a 200 m straight-road segment at the center of each agent's lane in the agents' respective lanes while avoiding collisions and off-road incidents. The straight-road segment is composed of 3 lanes with a lane width of 3.2 m each. 2 agents are instantiated on either side of the road (far-left and far-right lanes) with one lane left empty between the agents (middle lane). Fig 1 shows an illustration of the simulation environment.

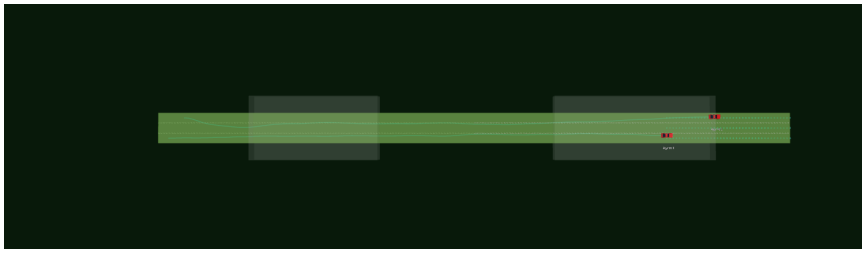


Figure 1: Straight road scenario in SMARTS simulator.

4.2 Observation Space

The observation space for each agent consists of a vector of continuous values with the following features:

1. $dist_g$: the euclidean distance from the ego vehicle position to the goal position normalized by the distance from start position to goal position
2. $dist_c$: the signed lateral error from the center of the lane closest to the ego vehicle normalized by half the lane width
3. ego_{rel-h}: the difference between the lane heading and the ego vehicle heading in radians.
4. $neighbor_{rel-h}$: the difference between the heading of the closest neighbor vehicle and ego vehicle heading in radians.

5. $steering_{ego}$: the steering angle of the front wheels of the ego vehicle in radians.

6. rel_x: the relative x position between the closest neighboring vehicle and the ego vehicle in meters.

7. rel_y: the relative y position between the closest neighboring vehicle and the ego vehicle in meters.

4.3 Action Space

At every time-step, the agents output continuous values for acceleration, deceleration and steering rate. The output values range [0,1] $\frac{m}{s^2}$ for acceleration and deceleration and [-1,1] $\frac{rad}{s}$ for steering rate. The steering rate is the amount of change in steering angle in radians per second.

4.4 Reward Scheme

The reward received by each agent from the environment is calculated according to the following formula:

$$r_{total} = c_1 r_d + c_2 r_g + c_3 r_h + r_{col} + r_{off} + r_{reach-goal} + r_{reach-max} \tag{37}$$

where c_1, c_2, c_3 denote constants that are used to weigh different reward components.

1. r_d: This denotes the reward received by the agent based on distance traveled in meters in one step.

2. r_g: This denotes the reward received by the agent based on distance to goal in current and next time-steps and is given by:

$$r_d = dist_{goal_{curr}} - dist_{g_{next}} \tag{38}$$

   Based on this formula, the agent receives a negative reward if the next time-step distance is larger than the current time-step distance.

3. r_h: This denotes the reward received based on the heading error of the ego vehicle with respect to the lane heading in current and next time-steps and is given by:

$$r_h = rel_{h_{curr}} - rel_{h_{next}} \tag{39}$$

   The agent receives a positive reward for heading closer to the lane heading and a negative reward for deviating further from the lane heading.

4. r_{col}: The agent receives a negative reward of -100 in the event of a collision.

5. r_{off}: The agent receives a negative reward of -100 if the agent is off-road.

6. $r_{reach-goal}$: The agent receives a positive reward of $+100$ for reaching the goal position.

7. $r_{reach-max}$: The agent receives a penalty of -5 if the maximum episode steps are reached, otherwise a reward of 0.5 is received.

4.5 Framework

The multi-agent decision making problem can be modeled using a **Partially Observable Markov Game** (POMG) [36]. Unlike in a Markov Game, in a POMG, each agent can only observe its local environment and there is no explicit communication between agents. A POMG is defined by a tuple $(\mathcal{N}, S, \{A^i\}_{i\in N}, \{O^i\}_{i\in N}, \{\Omega^i\}_{i\in N}, P, \{R^i\}_{i\in N}, \gamma)$, where $\mathcal{N} = \{1, ..., N\}$ denotes the set of N agents in a game, S denotes the environment state for all agents, A_i denotes the action space of the i^{th} agent and O_i denotes the local observation space of the i^{th} agent. Ω is an observation function that maps the environment state S to agent local observation O_i, $\Omega_i : S \to O_i$. P denotes the state transition function that represents the probability density of transitioning from state s_t to state s_{t+1} given the joint action of all agents $a_t \in A_t$, $P : S \times A \times S \to [0, \infty)$. Each agent receives a reward value given its own reward function $R_i : S \times A \times S \to \mathbb{R}$. The objective for each agent is to learn a policy π_i that is a mapping $\pi_i : O_i \times A_i \to [0, \infty)$.

4.6 Algorithms

MADDPG [36]: Multi-Agent Deep Deterministic Policy Gradient is a multi-agent off policy actor-critic algorithm that utilizes a centralized critic learned for each agent and separate actors that execute actions independently during test time. Each agent learns a Q-function by minimizing the following loss:

$$L(\phi_i) = \frac{1}{S}\sum_j (y^j - Q_i^\mu(\mathbf{x}^j, a_1^j, .., a_N^j))^2 \tag{40}$$

where $\mathbf{x}$ denotes the full state information and a_i^j denotes the i^{th} agent's action in a sample $j \in S$ where S is the set of samples. The target y is computed using:

$$y^j = r_i^j + \gamma Q_i^{\mu'}(\mathbf{x}^{j\prime}, a_1', .., a_N')|_{a_k' = \mu_{\mathbf{k}'}(o_k^{j\prime})} \tag{41}$$

where μ' denotes the target actor, k denotes the agent index and o' denotes the next agent local observation. Each agent updates its actor using the following gradient estimate:

$$\nabla_{\theta_i} J \approx \frac{1}{S}\sum_j \nabla_{\theta_i}\mu_i(o_i^j)\nabla_{a_i} Q_i^\mu(\mathbf{x}^j, a_1^j, .., a_N^j)|_{a_k = \mu_{\mathbf{k}}(o_k^j)} \tag{42}$$

MADSPG [48]: Multi-Agent Deep Stochastic Policy Gradient is similar to the MADDPG [36] algorithm except that a stochastic actor is used to output actions for each agent. In [48], the authors utilize MADSPG to learn a policy for the application of Dynamic Spectrum Access where IoT devices compete for time slots to transmit event information. The algorithm proposed in this thesis is the same except for the policy objective. The policy objective utilized in this report maximizes $Q_\phi^i(\mathbf{x}, \tilde{a_1}, ..., \tilde{a_N})$ whereas the gradient of the policy objective in [48] is given by $\nabla_{\theta_i} J \approx \frac{1}{S}\sum_j \nabla_{\theta_i} \log \pi_{\theta_i}(a_i|o_i) Q_\phi^i(\mathbf{x}, \tilde{a_1}, ..., \tilde{a_N})|_{\tilde{a_k}\sim\pi_{\theta k}(a_k|o_k)}$. Furthermore, an entropy-regularization term is added to the policy objective. The policy parameters are updated by maximizing the following policy objective for each agent:

$$J(\theta_i) = \mathop{\mathbb{E}}_{o_i, \mathbf{x}\sim D, \zeta\sim\mathcal{N}}[Q_\phi^i(\mathbf{x}, \tilde{a_1}, ..., \tilde{a_N}) - \alpha \log(\pi_\theta^i(\tilde{a_i}|o_i))]|_{\tilde{a_k}\sim\pi_{\theta k}(a_k|o_k)} \tag{43}$$

where the actions $\tilde{a_k}$ are sampled from a Normal distribution. The reparameterization trick is used to compute a deterministic function of state, policy parameters and independent Gaussian

noise. The Q-value parameters are updated by gradient descent on the following loss function:

$$L(\phi_i) = \frac{1}{S}\sum_j (y^j - Q_i(\mathbf{x}^j, a_1^j, .., a_N^j))^2 \tag{44}$$

where a^j denotes an action from a sample j from the replay buffer. The target y^j is given by:

$$y^j = r_i^j + \gamma Q_{\phi_i}^{'}(\mathbf{x}^{j\prime}, \tilde{a_1}', .., \tilde{a_N}')|_{\tilde{a_k}' \sim \pi_{\mathbf{k}}'(o_k^{j\prime})} \tag{45}$$

where $Q_{\phi_i}^{'}$ denotes the target Q-value function and $\tilde{a_k}'$ is an action sample from a reparameterized Normal distribution, for which the mean and standard deviation are output by the target actor π'.

L-MADSPG: Lyapnov MADSPG is an extension of the MADSPG algorithm with the addition of a Lyapunov-constraint in the policy objective to guarantee the closed-loop stability of the learned policies. The mathematical formulations that follow are based on the work in [46].

The following definition holds for the stability of stochastic discrete-time systems:

Definition 1: Let the origin $s_t = 0$ be the equilibrium point. The origin is said to be stable in probability if $\lim_{s_0 \to 0} \mathcal{P}[\sup_{t>0} ||s_t|| > \epsilon] = 0$ for any $\epsilon > 0$. The origin is asymptotically stable with probability one if it is stable in probability and $\lim_{s_0 \to 0} \mathcal{P}[\lim_{t\to\infty} ||s_t|| = 0] = 1$

The following Lemma was proposed by the authors in [46]:

Lemma 1: Let L be a continuous positive definite and radially unbounded function. Define a set $\mathcal{S}_\lambda = \{s : 0 \leq L < \lambda\}$ with $\lambda > 0$. Assuming:

$$\mathbb{E}_{a_t}[L(s_{t+1}|s)] - L(s_t) \leq -\psi(s_t), \forall t \tag{46}$$

where $\psi(s_t) \geq 0$ is continuous for any $s_t \in \mathcal{S}_\lambda$. Then the following statements apply according to [46]:

For any $s_0 \in \mathcal{S}_\lambda$, s_t converges to $\{s_t \in \mathcal{S}_\lambda : \psi(s_t) = 0\}$ with probability at least $1 - \frac{L(s_0)}{\lambda}$. Moreover, if $\psi(s_t)$ is positive definite on $\mathcal{S}_\lambda$, and there are two $\mathcal{K}$ class functions h_1 and h_2 such that $h_1(||s_t||_2) \leq L(s_t) \leq h_2(||s_t||_2)$, the system state s_t will converge to 0 with probability one.

A cost function for each agent is introduced and is defined as follows:

Definition 2: A cost function is feasible, if and only if, $c_i(s_{i,t}, a_{i,t}) > 0 \; \forall s_{i,t} \neq s_{i,T}, \forall a_{i,t} \neq a_{i,T}$ and $c_i(s_{i,T}, a_{i,T}) = 0$, where $s_{i,T}$ is the target state and $a_{i,T}$ is the target action.

Based on this definition of a feasible cost function, a performance function L_i is proposed as follows:

$$L_i(s, a_i, \mathbf{a}) = \sum_{t=0}^{\infty} \gamma_L^t c_i(s_{i,t}, a_{i,t}, \mathbf{a}_t) \tag{47}$$

where c_i is a cost function for agent i, $\gamma_L \in [0, 1)$ is a discount factor and $\mathbf{a}$ denotes the actions of agents other than agent i in the learning environment. The following assumption is made for the stabilization of agents in a Markov Game:

Assumption 1: Assuming that all agents can be stabilized based on Definition 1, there exists a control policy for each agent i such that L_i is a Lyapunov function candidate.

In the following section, the state s is replaced by $\mathbf{x}$ to denote the full-state information where $\mathbf{x} = \{o_1, ..., o_N\}$ is a set of local observations o_k. Based on Lemma 1, Assumption 1, (47) and the policy objective in (43), the following constrained policy improvement is formulated:

$$\pi_i = \arg\max_{\pi_i} \mathop{\mathbb{E}}_{o_i, \mathbf{x}\sim D, \zeta\sim\mathcal{N}} [Q^i_\phi(\mathbf{x}, \tilde{a_1}, ..., \tilde{a_N}) - \alpha \log(\pi^i_\theta(\tilde{a_i}|o_i))]|_{\tilde{a_k}\sim\pi_{\theta\,k}(a_k|o_k)}$$
$$s.t \mathop{\mathbb{E}}_{o_i, \mathbf{x}\sim D} [L_i(\mathbf{x}_{t+1}, a_{i,t+1}, \mathbf{a}_{t+1})] - L_i(\mathbf{x}_t, a_{i,t}, \mathbf{a}_t) \leq -\psi(\mathbf{x}_t) \quad (48)$$

Furthermore, $\psi(\mathbf{x}_t)$ is a positive definite function where $\psi(\mathbf{x}_t) = \kappa_i c_i(\mathbf{x}_t, a_{i,t}, \mathbf{a}_t)$ and $\kappa > 0$ is a constant. It is possible to transform the hard constraint in (48) into a soft constraint using the Lagrangian method:

$$\pi_i = \arg\max_{\pi_i} \mathop{\mathbb{E}}_{o_i, \mathbf{x}\sim D, \zeta\sim\mathcal{N}} [Q^i_\phi(\mathbf{x}, \tilde{a_1}, ..., \tilde{a_N}) - \alpha \log(\pi^i_\theta(\tilde{a_i}|o_i))]|_{\tilde{a_k}\sim\pi_{\theta\,k}(a_k|o_k)}$$
$$+ \beta_i \mathop{\mathbb{E}}_{o_i, \mathbf{x}\sim D} [L_i(\mathbf{x}_{t+1}, a_{i,t+1}, \mathbf{a}_{t+1}) - L_i(\mathbf{x}_t, a_{i,t}, \mathbf{a}_t) + \psi(\mathbf{x}_t)] \quad (49)$$

(49) can always result in a stable control policy if it is assumed that the agents start with a feasible control policy. Instead of assuming that agents start with a feasible control policy, Lyapunov's energy decreasing condition is used to guarantee the closed loop stability of the learned policies. Also in (49), the next state $\mathbf{x}_{t+1}$ is determined by the current policy and in order to use a replay buffer, (49) has to be modified. Assume that the Lyapunov function L_i is Lipschitz continuous with respect to $a_{i,t}$. The following assumption is introduced for deterministic discrete-time systems:

Assumption 2: Consider the deterministic discrete-time system $\mathbf{x}_{t+1} = f(\mathbf{x}_t, a_t, \mathbf{a}_t)$. The nonlinear dynamics f is Lipschitz continuous with respect to $a_{i,t}$ if $||f(\mathbf{x}_t, a^2_{i,t}, \mathbf{a}_t) - f(\mathbf{x}_t, a^1_{i,t}, \mathbf{a}_t)||_2 \leq l_f||a^2_{i,t} - a^1_{i,t}||_2$ where l_f is a Lipschitz constant.

The Lyapunov function can be parameterized with a DNN and since the DNN must be bounded, it is assumed that the Lyapunov function L_i is also Lipschitz continuous with respect to the state $\mathbf{x}_t$ with Lipschitz constant l_L. Let a feasible policy be one in which the system is stable and a Lyapunov function exists. Suppose Assumption 2 holds, then the following Theorem is introduced to guarantee that the new policy following the policy improvement step is also feasible [46].

Theorem 1: Let $\pi_{i,old}$ be a feasible policy for data collection and $L_{\pi_{i,old}}(\mathbf{x})$ is the Lyapunov function. A new policy $\pi_{i,new}$ will also be a feasible policy if there exists:

$$L_{\pi_{i,old}}(\mathbf{x}_{t+1}) + l_L l_f ||a_t^{\pi_{i,new}} - a_t^{\pi_{i,old}}||_2 - L_{\pi_{i,old}}(\mathbf{x}_t) \leq 0 \quad (50)$$

Proof of Theorem 1 is provided in [46]. The negative cost function is chosen as the reward and hence the negative Q value function is used as the Lyapunov candidate. Based on this and following Theorem 1, the following Lyapunov constrained policy objective is minimized:

$$J_{\pi_i} = \mathop{\mathbb{E}}_{o_i,\mathbf{x}\sim D,\zeta\sim\mathcal{N}}[\alpha\log(\pi^i_\theta(\tilde{a_i}|o_i)) - Q^i_\phi(\mathbf{x},\tilde{a_1},...,\tilde{a_N})]|_{\tilde{a_k}\sim\pi_{\theta k}(a_k|o_k)} + \mathop{\mathbb{E}}_{o_i,\mathbf{x}\sim D,\zeta\sim\mathcal{N}}\beta_{\rho_i}[-Q'_{\phi_i}(\mathbf{x}_{t+1},\tilde{a}_{1,t+1},...\tilde{a}_{N,t+1})|_{\tilde{a'_k}\sim\pi^k_{\theta_{targ}}(a'_k|o'_k)} + l_\delta||\tilde{a}^{\pi_{\theta_i}(|o_i)}_{i,t} - a_{i,t}||_2 + Q_{\phi_i}(\mathbf{x}_t,a_{1,t},...a_{N,t})|_{a_k\sim\mathcal{D}} + \psi(\mathbf{x}_t)] \quad (51)$$

where l_δ is a constant, and $\psi(\mathbf{x}_t)$ is a positive-definite function. An intuitive choice for ψ is the reward value. However, since the reward in this implementation can assume negative values, a sigmoid activation σ was applied to the rewards $\psi(\mathbf{x}_t) = \sigma(r^i(\mathbf{x_t},a_{i,t},\mathbf{a}_t)) + \epsilon$. β_{ρ_i} denotes a temperature that controls the Lyapunov term and is parameterized by a Deep Neural Network with parameters ρ. The DNN, takes as input local observations and actions sampled from a replay buffer $(o^i,a^i)\sim D$. β_{ρ_i} is updated to minimize the following objective:

$$J_{\beta_i} = -\mathop{\mathbb{E}}_{o_i,\mathbf{x}\sim D,\zeta\sim\mathcal{N}}\beta_{\rho_i}(o^i,a^i)[-Q'_{\phi_i}(\mathbf{x}_{t+1},\tilde{a}_{1,t+1},...\tilde{a}_{N,t+1})|_{\tilde{a'_k}\sim\pi^k_{\theta_{targ}}(a'_k|o'_k)} + l_\delta||\tilde{a}^{\pi_{\theta_i}(|o_i)}_{i,t} - a_{i,t}||_2 + Q_{\phi_i}(\mathbf{x}_t,a_{1,t},...a_{N,t})|_{a_k\sim\mathcal{D}} + \psi(\mathbf{x}_t)] \quad (52)$$

In (51), a clipped value for β is used and the value is clipped in the range (0,1) to prevent large values which can affect the convergence of the algorithm. Large values of β can result from unstable control policies in training since there is no assumption made that the starting control policies must be feasible. The Q-value function parameters are updated according to (44)

4.7 Training

For training, 5 trials are run with 2 agents for each of the learning algorithms for 2000 episodes per trial and a maximum episode limit of 300 steps. An initial exploration phase was implemented during training to encourage early exploration where actions are randomly sampled from a uniform distribution over the action space for the first 50 episodes. The agents are instantiated at the start of every episode at 5m and 10m from the start of their lanes respectively. An episode terminates if either of the agents is involved in a collision or off-road incident or if the maximum steps per episode limit is reached. An episode also terminates and is a success if either of the agents reach their goal position at the end of the road while the other agent remains on the road. The average episode rewards obtained over the course of training are observed for agents trained in self-play with MADDPG, MADSPG and L-MADSPG in separate trials to observe the effects of entropy regularization and the Lyapunov-stability constraint on the performance of agents during training and on the convergence of the policies.

4.8 Evaluation

For the evaluation scenario, the policies learned from each of the training trials are run for 500 episodes. The performance of the learned policies is evaluated by measuring success rate, collision rate, off-road rate, time-out rate, average lateral error per episode and average standard deviation of average lateral error per episode (averaged for the 2 agents). The evaluation trials are run with

model weights saved during training that achieved the highest episode returns averaged over the last 10 episodes. The stability of the learned policies is assessed by adding mean-zero Gaussian measurement noise to two of the observation dimensions, namely $dist_c$ and $steering_{ego}$ with varying standard deviations of noise as seen in Table 1 and comparing the performance of the learned policies.

Noise level	$dist_c$	$steering_{ego}$
Easy	0.0	0.0
Medium	$\zeta \sim (0, 10^{-4})$	$\zeta \sim (0, \pi * 10^{-4})$
Hard	$\zeta \sim (0, 10^{-3})$	$\zeta \sim (0, \pi * 10^{-3})$

Table 1: Difficulty levels corresponding to mean-zero Gaussian noise added to observation features $dist_c$ and $steering_{ego}$ with varying standard deviations of noise.

4.9 Implementation Details

Feed forward neural networks are used for the actor and critic networks with 2 hidden layers and units [200,100] each with ReLU activation for the hidden layers. For the stochastic actors in the MADSPG and L-MADSPG algorithms, the output vector of the hidden layer is fed to two FC layers which output mean and standard deviation vectors. The mean and standard deviation outputs are used to construct a Normal distribution from which raw actions are sampled. A *Sigmoid* function is used for the acceleration and deceleration dimensions and a *tanh* function for the steering rate dimension for raw actions after sampling. Furthermore, a feed forward neural network is also used for the Lyapunov network with 3 hidden layers and units [64,64,16] each. The objectives are optimized using Stochastic Gradient Descent with the ADAM optimizer and a batch size of 1024 samples. The learning rates for the actor, critic, and Lyapunov networks are set to 1e-3, 1e-2 and 5e-3 respectively. The training was done on an NVIDIA GeForce 840M GPU.

5 Results

5.1 Training

The learning curves for each of the trained algorithms can be seen in Fig 2. In comparison with MADDPG, MADSPG is more stable in training and achieves faster convergence of learned policies. The greater stability in training is indicated by lower variance in average episode returns over 5 trials. However, MADDPG achieves higher peak average episode returns compared with MADSPG. Furthermore, L-MADSPG achieves higher peak average episode returns in training compared with MADSPG indicating better performance and comparable average episode returns in comparison with MADDPG. However, the variance of the average episode returns over 5 trials are larger for L-MADSPG compared to MADSPG indicating lower stability in training over the course of all trials. The variance of the average episode returns of L-MADSPG is lower than that of MADDPG indicating better stability over the course of all trials. Based on the raw data from each of the trials in Fig 5, Fig 6, Fig 7 in the Appendix section, the results show that the policies trained demonstrate two different behaviors in average episode returns in separate trials.

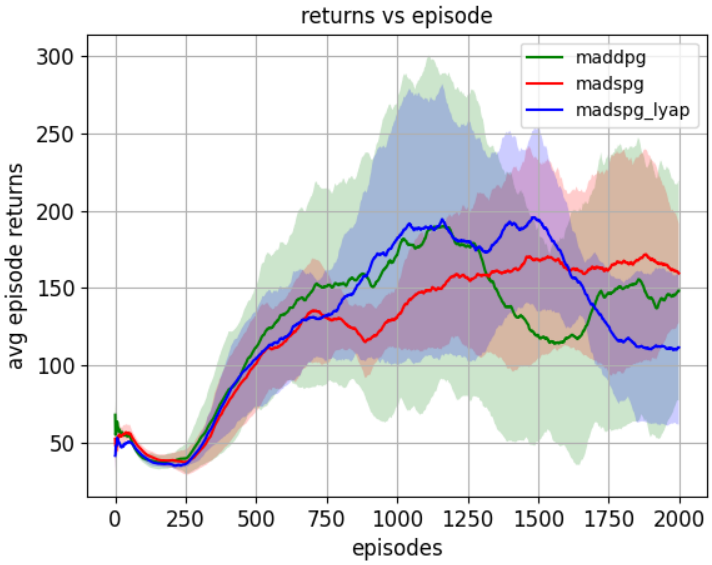


Figure 2: Learning curves for agents trained with MADDPG, MADSPG, and MADSPG-LYAP algorithms. The figure shows the mean and 1 standard deviation from the mean of 5 trials per algorithm. The episode returns for each trial are averaged over the last 200 episodes.

In some trials, it can be seen that the average episode returns increase slowly followed by convergence. In others, the average episode returns climb rapidly achieving considerably higher episode returns followed by a collapse in performance. The results in Fig 3 show the mean and variance of average episode returns of trials where convergence is observed.

Based on the results shown in Fig 3, it can be observed that MADSPG and L-MADSPG show better performance in training compared with MADDPG. Furthermore, MADSPG and L-MADSPG are more stable in training as shown by lower variance achieved in comparison with MADDPG. The lowest variance and highest average episode returns are seen for L-MADSPG. When comparing L-MADSPG with MADSPG, L-MADSPG achieves higher average episode returns and is more stable as observed by lower variance.

5.2 Evaluation

The evaluation results for models trained in all 5 trials with MADDPG, MADSPG and L-MADSPG with varying standard deviation of noise can be seen in Fig 4. Across all trials, MADDPG achieves lower average lateral error in comparison with MADSPG and L-MADSPG over all difficulty levels as can be seen in Fig 4a. The average lateral error of MADSPG and L-MADSPG are comparable. In Fig 4b, MADDPG achieves the lowest average of standard deviation of the average lateral error across all difficulty levels of noise. MADSPG achieves the highest average standard deviation of average lateral error with low variation in results across noise levels. Compared to MADSPG however, L-MADSPG achieves considerably lower average standard deviation of av-

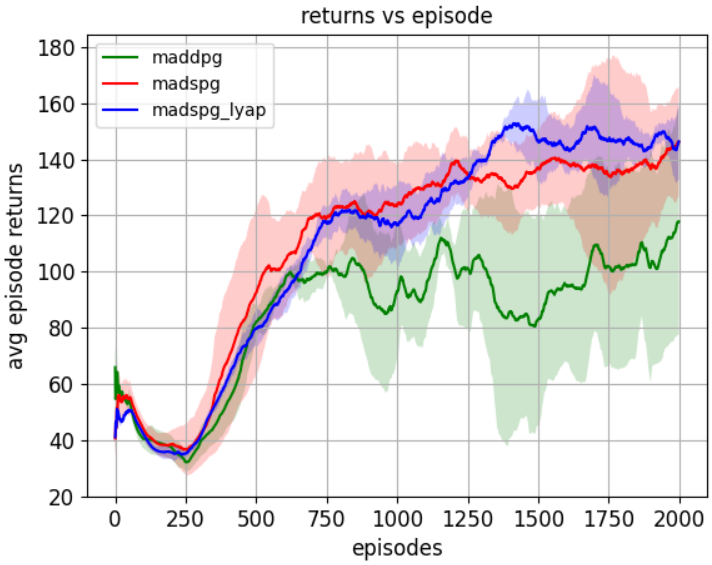
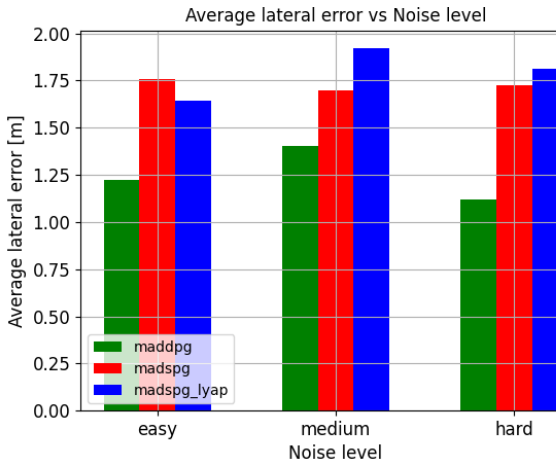
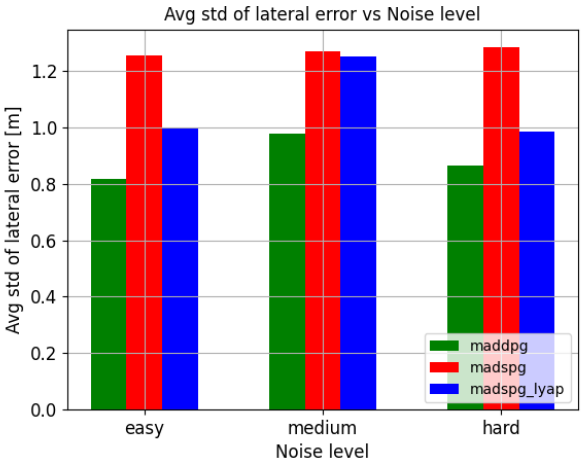


Figure 3: Learning curves for agents trained with MADDPG, MADSPG, and MADSPG-LYAP algorithms. The figure shows the mean and 1 standard deviation from the mean of only trials where convergence is observed. The episode returns for each trial are averaged over the last 200 episodes.

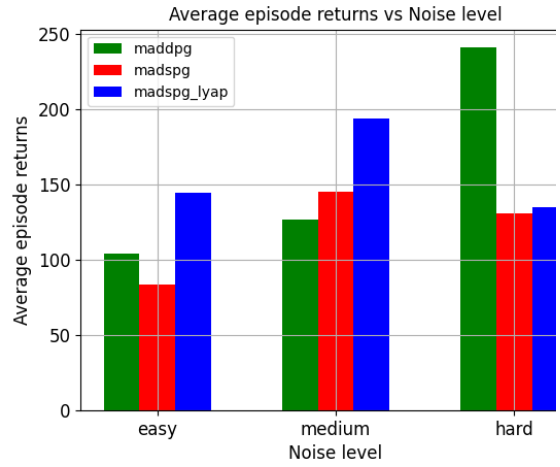
erage lateral error on the easy and hard noise levels. On the medium noise level, MADSPG and L-MADSPG perform similarly. Based on these results, MADDPG achieves better proximity to lane center as indicated by lower average lateral error compared to MADSPG and L-MADSPG. With the addition of Gaussian noise to the input observations, MADDPG is most stable as indicated by lowest average standard deviation of lateral error. Furthermore, L-MADSPG is more stable than MADSPG on the easy and hard noise levels, and both achieve comparable stability on the medium noise level. The average episode returns are highest for L-MADSPG on the easy and medium difficulty noise levels as seen in Fig 4c. On the hard noise level however, MADDPG achieves relatively higher average episode returns compared to L-MADSPG and MADSPG. The highest success rates were achieved with L-MADSPG seen in Fig 4d, considerably outperforming MADDPG and MADSPG on the easy and medium noise levels. However, the success rate for L-MADSPG was lower on the hard noise level, where MADDPG achieved the highest success rate. From Fig 4e, it can be observed that MADDPG across all difficulty levels achieved the highest collision rates, with L-MADSPG achieving the lowest collision rates on the easy and medium difficulty levels while performing comparably to MADSPG on the hard noise level. The off-road rates for MADDPG were lowest across-all noise levels with higher rates for MADSPG and L-MADSPG as seen in Fig 4f. In Fig 4g, it can be observed that MADDPG has the highest time-out rates compared to MADSPG and L-MADSPG on the easy and medium difficulty levels. On the hard noise level, The time-out rate was considerably higher for L-MADSPG compared with MADDPG and MADSPG.



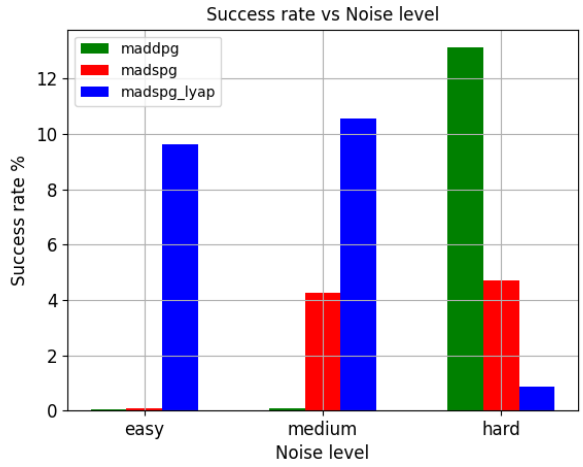
(a) Average lateral error vs varying noise levels.



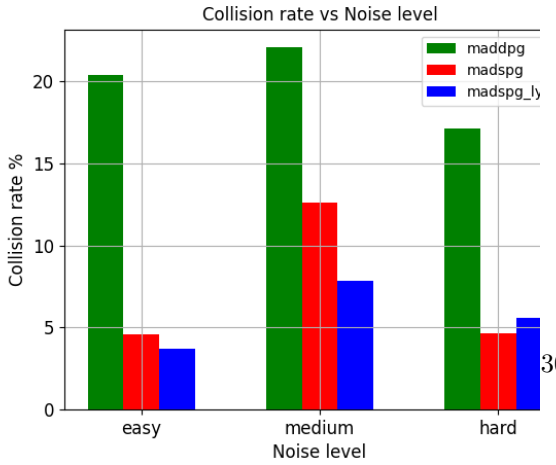
(b) Standard deviation of average lateral error vs varying noise levels.



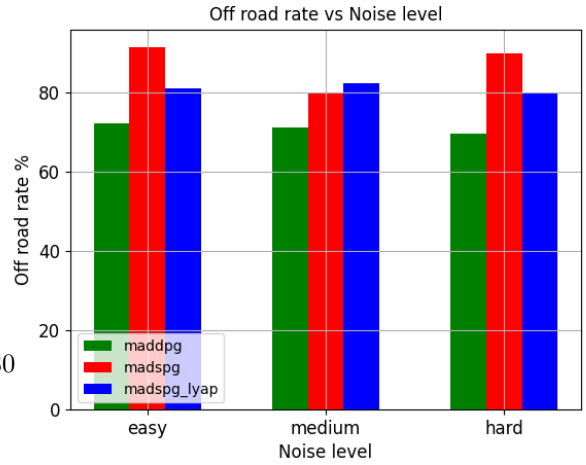
(c) Average episode returns vs varying noise levels.



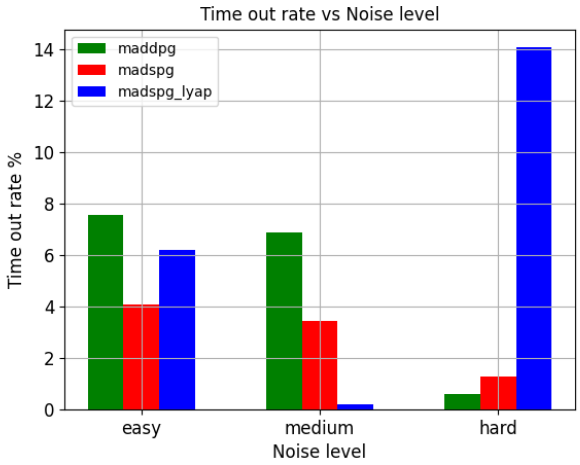
(d) Success rate vs varying noise levels.



(e) Collision rate vs varying noise levels.



(f) Off-road rate vs varying noise levels.



(g) Time-out rate vs varying noise levels.

Figure 4: Evaluation results of MADDPG, MADSPG and L-MADSPG for metrics based on all 5 trials per algorithm. The noise difficulty level is increased by increasing standard deviation of Gaussian noise added to input observations of agents.

6 Discussion

6.1 Training

Based on the results from Fig 2 and Fig 3, MADSPG is more stable in training compared to MADDPG. This is indicated by the reduced variance over the training trials. Therefore, it can be stated that a stochastic actor with entropy regularization is more stable in training compared to a deterministic actor with no entropy regularization. Furthermore, based only on trials where learning converged, MADSPG achieved faster convergence and higher average episode returns compared with MADDPG. Moreover, based on converged trials, a Lyapunov constrained policy objective achieves more stable training compared to a policy objective without a stability constraint as indicated by the lower variance in training. A Lyapunov constrained policy objective performs similarly to a policy objective without Lyapunov constraint in terms of average episode returns. This is based on a comparison between the performance of L-MADSPG and MADSPG.

6.2 Evaluation

From the evaluation results in Fig 4a and Fig 4b, MADDPG achieves significantly lower average lateral error and average standard deviation of average lateral error compared with MADSPG and L-MADSPG. This indicates that a stochastic actor with entropy regularization reduces lane-tracking performance and is less robust to measurement noise. A stochastic actor with entropy regularization is also less stable as indicated by the higher average standard deviation of lateral error compared with a deterministic actor. This is expected as a stochastic actor can sample from more actions which leads to more deviations from the lane center. However, a Lyapunov

constrained policy objective leads to improved stability compared to a policy objective without a stability constraint as indicated by the significantly lower average standard deviation of average lateral error of L-MADSPG compared with MADSPG on two of the three noise difficulty levels. This suggests that the Lyapunov constraint leads to policies learned that mitigate the effects of increased stochasticity of learned policies. It can also be concluded that the addition of increasing measurement noise does not lead to significant degradation of lane-tracking performance and stability in all three algorithms.

Furthermore, a Lyapunov constrained policy objective achieves the highest returns and highest success rates on two of the three noise difficulty levels (easy and medium). Counter-intuitively, this indicates that a policy learned with Lyapunov constraint sacrifices stability and lane-tracking performance for higher returns and successful episode completion. However, at high noise level, MADDPG significantly outperforms MADSPG and L-MADSPG in average returns and success rate. This indicates that a stochastic actor with entropy regularization does not perform well with high measurement noise. It could be stated that since stochastic actors can sample non-ideal actions, at a higher noise level, there is a higher chance of deviating further from the lane center since the actor can sample non-deal actions based on non-accurate observations which in turn degrades performance. At a low noise level however, the stochasticity can lead agents to explore more actions and help agents travel further towards their goal distance. Furthermore, at high level of measurement noise, a Lyapunov constrained policy objective performs similarly to that without the stability constraint on average returns, and even reduced performance on success rate. MADDPG significantly outperforms MADSPG and L-MADSPG in terms of success rate at high noise level. This means that more often, a deterministic policy guides agents to reach their goal positions at the end of the road. However, it should be noted that agents can also reach the end of the road segment but be slightly off the goal position resulting in "non-successful" episode completion. Moreover, a stochastic actor with entropy regularization significantly reduces collision rate compared to a deterministic actor without entropy regularization. This could be due to the fact that a stochastic policy has more actions to sample from when the agents are on a collision trajectory, and more often are able to avoid collisions. It can also be observed that the addition of a Lyapunov constraint achieves similar or lower collision rates compared to a stochastic algorithm without the Lyapunov constraint. This suggests that the Lyapunov constraint can lead to safer learned policies in algorithms with stochastic actors. Also a stochastic actor with entropy regularization and Lyapunov constrained policy objective reduce time-out rates on low difficulty noise levels. On the high noise level, Lyapunov constrained policy objective suffers high time-out rates compared to policies without Lyapunov stability constraint. This indicates that the policies learned with the Lyapunov constraint exploit the reward scheme by exhausting the episode steps limit. It is possible that a reward scheme that penalizes slow moving agents or higher penalties for exceeding the maximum episode steps can result in lower time-out rates and possibly higher average returns on high noise level.

7 Conclusion

The work in this thesis is aimed at investigating the effects of entropy regularization and a multi-agent policy objective with Lyapunov stability constraint on the performance of agents on lane-keeping. From the discussion above, it can be concluded that compared to a deterministic pol-

icy, an algorithm with a stochastic actor, and further, an algorithm with Lyapunov constrained policy objective achieve higher average episode returns and are more stable in training. This is based on results from trials where training converged. From the evaluation metrics for a scenario with added measurement noise, a Lyapunov constrained policy objective achieves better stability compared to a similar algorithm without the Lyapunov constraint as indicated by the reduced average standard deviation of average lateral error of L-MADSPG vs MADSPG. However, a deterministic actor is better able to stay close to the lane center and avoid deviations indicating more stable behavior compared to the two algorithms with stochastic actors. Furthermore, the addition of Lyapunov constraint results in higher average returns and success rates compared to algorithms without stability constraints on low and medium measurement noise levels. However, the performance of the algorithm with Lyapunov constraint suffers at high noise levels in terms of success rate. Also, it was demonstrated that a stochastic actor with entropy regularization significantly reduces collision rates and moreover, a Lyapunov constraint further reduces collision rates. In summary, the addition of entropy regularization achieves more stable training and although a Lyapunov constrained policy objective (L-MADSPG) is less stable with addition of measurement noise compared with MADDPG, L-MADSPG is still as stable or more stable compared with MADSPG. L-MADSPG also outperforms other algorithms in average returns, success rate and collision rate at low measurement noise levels.

References

[1] Santokh Singh. Critical reasons for crashes investigated in the national motor vehicle crash causation survey. Technical report, 2015.

[2] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. Safe, multi-agent, reinforcement learning for autonomous driving. *arXiv preprint arXiv:1610.03295*, 2016.

[3] B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[4] Ming Zhou, Jun Luo, Julian Villella, Yaodong Yang, David Rusu, Jiayu Miao, Weinan Zhang, Montgomery Alban, Iman Fadakar, Zheng Chen, et al. Smarts: Scalable multi-agent reinforcement learning training school for autonomous driving. *arXiv preprint arXiv:2010.09776*, 2020.

[5] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *Handbook of Reinforcement Learning and Control*, pages 321–384, 2021.

[6] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[7] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[8] Martin Zinkevich, Amy Greenwald, and Michael Littman. Cyclic equilibria in markov games. *Advances in Neural Information Processing Systems*, 18:1641, 2006.

[9] Sushrut Bhalla, Sriram Ganapathi Subramanian, and Mark Crowley. Deep multi agent reinforcement learning for autonomous driving. In *Canadian Conference on Artificial Intelligence*, pages 67–78. Springer, 2020.

[10] Yuqi Liu, Qichao Zhang, and Dongbin Zhao. A reinforcement learning benchmark for autonomous driving in intersection scenarios. *arXiv preprint arXiv:2109.10557*, 2021.

[11] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

[12] Arne Kesting, Martin Treiber, and Dirk Helbing. Enhanced intelligent driver model to access the impact of driving strategies on traffic capacity. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 368(1928):4585–4605, 2010.

[13] Shengchao Yan, Tim Welschehold, Daniel Büscher, and Wolfram Burgard. Courteous behavior of automated vehicles at unsignalized intersections via reinforcement learning. *arXiv preprint arXiv:2106.06369*, 2021.

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

[15] Pablo Alvarez Lopez, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Flötteröd, Robert Hilbrich, Leonhard Lücken, Johannes Rummel, Peter Wagner, and Evamarie Wießner. Microscopic traffic simulation using sumo. In *2018 21st international conference on intelligent transportation systems (ITSC)*, pages 2575–2582. IEEE, 2018.

[16] Kasra Mokhtari and Alan R Wagner. Safe deep q-network for autonomous vehicles at unsignalized intersection. *arXiv preprint arXiv:2106.04561*, 2021.

[17] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[18] Quanyi Li, Zhenghao Peng, Zhenghai Xue, Qihang Zhang, and Bolei Zhou. Metadrive: Composing diverse driving scenarios for generalizable reinforcement learning. *arXiv preprint arXiv:2109.12674*, 2021.

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

[20] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *International conference on machine learning*, pages 22–31. PMLR, 2017.

[21] Dong Chen, Zhaojian Li, Yongqiang Wang, Longsheng Jiang, and Yue Wang. Deep multi-agent reinforcement learning for highway on-ramp merging in mixed traffic. *arXiv preprint arXiv:2105.05701*, 2021.

[22] Kaixiang Lin, Renyu Zhao, Zhe Xu, and Jiayu Zhou. Efficient large-scale fleet management via multi-agent deep reinforcement learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1774–1783, 2018.

[23] Justin K Terry, Nathaniel Grammel, Ananth Hari, Luis Santos, and Benjamin Black. Revisiting parameter sharing in multi-agent deep reinforcement learning. *arXiv preprint arXiv:2005.13625*, 2020.

[24] Meha Kaushik, K Madhava Krishna, et al. Parameter sharing reinforcement learning architecture for multi agent driving behaviors. *arXiv preprint arXiv:1811.07214*, 2018.

[25] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[26] Gym-torcs. `https://github.com/ugo-nama-kun/gym_torcs`, 2016.

[27] Wei Zhou, Dong Chen, Jun Yan, Zhaojian Li, Huilin Yin, and Wanchen Ge. Multi-agent reinforcement learning for cooperative lane changing of connected and autonomous vehicles in mixed traffic. *arXiv preprint arXiv:2111.06318*, 2021.

[28] Arne Kesting, Martin Treiber, and Dirk Helbing. Connectivity statistics of store-and-forward intervehicle communication. *IEEE Transactions on Intelligent Transportation Systems*, 11(1):172–181, 2010.

[29] Guanglin Ji, Junyan Yan, Jingxin Du, Wanquan Yan, Jibiao Chen, Yongkang Lu, Juan Rojas, and Shing Shin Cheng. Towards safe control of continuum manipulator using shielded multiagent reinforcement learning. *IEEE Robotics and Automation Letters*, 6(4):7461–7468, 2021.

[30] Yuhuai Wu, Elman Mansimov, Roger B Grosse, Shun Liao, and Jimmy Ba. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. *Advances in neural information processing systems*, 30, 2017.

[31] Alessandro Paolo Capasso, Paolo Maramotti, Anthony Dell'Eva, and Alberto Broggi. End-to-end intersection handling using multi-agent deep reinforcement learning. *arXiv preprint arXiv:2104.13617*, 2021.

[32] Richard Van Der Horst and Jeroen Hogema. Time-to-collision and collision avoidance systems. 1993.

[33] Rohan Chandra and Dinesh Manocha. Gameplan: Game-theoretic multi-agent planning with human drivers at intersections, roundabouts, and merging. *arXiv preprint arXiv:2109.01896*, 2021.

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

[35] Yaodong Yang, Rui Luo, Minne Li, Ming Zhou, Weinan Zhang, and Jun Wang. Mean field multi-agent reinforcement learning. In *International conference on machine learning*, pages 5571–5580. PMLR, 2018.

[36] Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, 30, 2017.

[37] Kaiqing Zhang, Zhuoran Yang, Han Liu, Tong Zhang, and Tamer Başar. Finite-sample analysis for decentralized batch multiagent reinforcement learning with networked agents. *IEEE Transactions on Automatic Control*, 66(12):5925–5940, 2021.

[38] Sainbayar Sukhbaatar, Rob Fergus, et al. Learning multiagent communication with backpropagation. *Advances in neural information processing systems*, 29, 2016.

[39] Avik Pal, Jonah Philion, Yuan-Hong Liao, and Sanja Fidler. Emergent road rules in multi-agent driving environments. *arXiv preprint arXiv:2011.10753*, 2020.

[40] Praveen Palanisamy. Multi-agent connected autonomous driving using deep reinforcement learning. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE, 2020.

[41] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International conference on machine learning*, pages 1407–1416. PMLR, 2018.

[42] Fabian Konstantinidis, Ulrich Hofmann, Moritz Sackmann, Jörn Thielecke, Oliver De Candido, and Wolfgang Utschick. Parameter sharing reinforcement learning for modeling multi-agent driving behavior in roundabout scenarios. In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, pages 1974–1981. IEEE, 2021.

[43] Paul Young Joun Ha, Sikai Chen, Jiqian Dong, Runjia Du, Yujie Li, and Samuel Labi. Leveraging the capabilities of connected and autonomous vehicles and multi-agent reinforcement learning to mitigate highway bottleneck congestion. *arXiv preprint arXiv:2010.05436*, 2020.

[44] Tianyu Shi, Jiawei Wang, Yuankai Wu, Luis Miranda-Moreno, and Lijun Sun. Efficient connected and automated driving system with multi-agent graph reinforcement learning. *arXiv preprint arXiv:2007.02794*, 2020.

[45] Jakob Foerster, Ioannis Alexandros Assael, Nando De Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. *Advances in neural information processing systems*, 29, 2016.

[46] Qingrui Zhang, Hao Dong, and Wei Pan. Lyapunov-based reinforcement learning for decentralized multi-agent control. In *International Conference on Distributed Artificial Intelligence*, pages 55–68. Springer, 2020.

[47] Minghao Han, Lixian Zhang, Jun Wang, and Wei Pan. Actor-critic reinforcement learning for control with stability guarantee. *IEEE Robotics and Automation Letters*, 5(4):6217–6224, 2020.

[48] Rahif Kassab, Apostolos Destounis, Dimitrios Tsilimantos, and Mérouane Debbah. Multi-agent deep stochastic policy gradient for event based dynamic spectrum access. In *2020 IEEE 31st Annual International Symposium on Personal, Indoor and Mobile Radio Communications*, pages 1–6. IEEE, 2020.

8 Appendix

8.1 Algorithms Classification

Scenario	Rule-based	SA	FC	FD	CTDE	Networked	Traffic	Simulator
Unsignalized Intersection	IDM[10] AEB[10]	TD3[10]					hetero	RL-CIS[10]
		PPO[13]					hetero	SUMO[13]
	TTC[16]	DDQN[16]					ped	CARLA[16]
		DQN[4] PPO[4]	Comm-Net [4]		MAAC[4] MADDPG[4] MF-AC[4]	Net-Fitted Q[4]	hetero	SMARTS[4]
		PPO[18]			MF-CCPPO [18] Concat-CCPPO [18]		homo	Meta-Drive[18]
	IDM[44]	DDPG[44] PPO[44]			MAPPO[44] (PS) MADDPG[44] GCN-PPO[44]		hetero	flow[44]
signalized Intersection	TTC[31]				MAD-A3C[31]		homo	CAIRO[31]
					IMPALA[40] (PS)		homo	MACAD-Gym[40]
Random Config		PPO[18] SAC[18]					hetero	Meta-Drive[18]
		PPO[18]			MF-CCPPO [18] Concat-CCPPO [18]		homo	Meta-Drive[18]
on-ramp merging					MA2C[21] MAA2C [21] MAPPO [21] MAA CKTR [21] (PS)		hetero	MARL-CAVs[21]
	IDM [44]	DDPG[44] PPO[44]			MAPPO[44] (PS) MADDPG [44] GCN-PPO [44]		hetero	flow[44]
lane-keeping		DDPG[24]			PS-DDPG [24]		homo	Gym-TORCS[24]

	IDM [44]	DDPG[44] PPO[44]			MAPPO[44] (PS) MADDPG [44] GCN-PPO [44]		hetero	flow[44]
			Comm-Net[9]		MA-MeSN [9] MA-BoN [9] DIAL[9]		homo	VREP[9]
lane-changing		DDPG[24]			PS-DDPG [24]		homo	Gym-TORCS[24]
					MA2C[27] MADQN [27] MAA CKTR [27] MAPPO [27]		hetero	highway-env[27]
double-merge		DQN[4] PPO[4]	Comm-Net[4]		MAAC[4] MADDPG[4] MF-AC[4]	Net-Fitted Q[4]	hetero	SMARTS[4]
two-way traffic		DQN[4] PPO[4]	Comm-Net[4]		MAAC[4] MADDPG[4] MF-AC[4]	Net-Fitted Q[4]	hetero	SMARTS[4]
tollgate		PPO[18]			MF-CCPPO [18] Concat-CCPPO [18]		homo	Meta-Drive[18]
bottleneck		PPO[18]			MF-CCPPO [18] Concat-CCPPO [18]		homo	Meta-Drive[18]
			GCN-DDPG [43]				hetero	SUMO[43]
roundabout		PPO[18]			MF-CCPPO [18] Concat-CCPPO [18]		homo	Meta-Drive[18]
	IDM [42]				PS-SAC[42]		homo	NA
parking		PPO[18]			MF-CCPPO [18] Concat-CCPPO [18]		homo	Meta-Drive[18]

Table 2: A summary of the algorithms employed to tackle multiple driving scenarios. The algorithms are categorized based on driving scenario, algorithm type, traffic setting and simulator used. The algorithm types are rule-based, single agent (SA) RL, Fully Centralized (FC) MARL, Fully Decentralized (FD) MARL, Centralized Training Decentralized Execution (CTDE) MARL and Networked Agent Learning. The traffic setting homo refers to training in an all AV environment, whereas hetero refers to training in a mixed setting with both AV and HDVs. The algorithms are cited based on the papers in which they are utilized.

8.2 Detailed Training Results

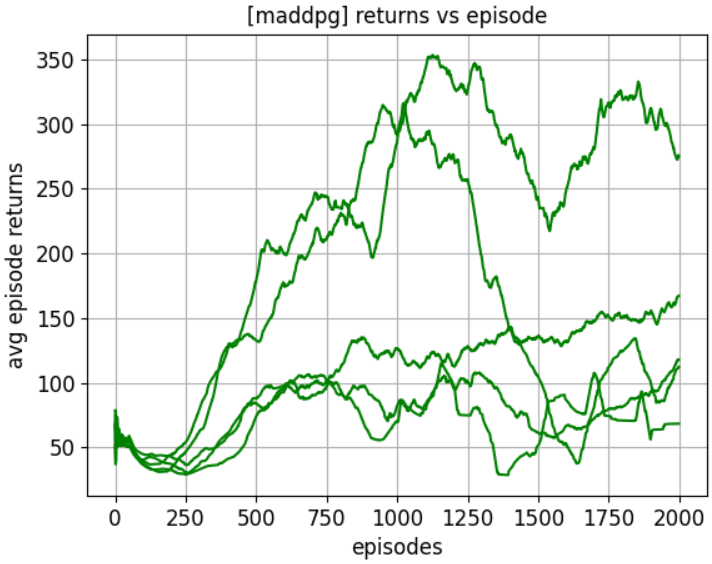


Figure 5: Learning curves for agents trained with MADDPG for 5 trials

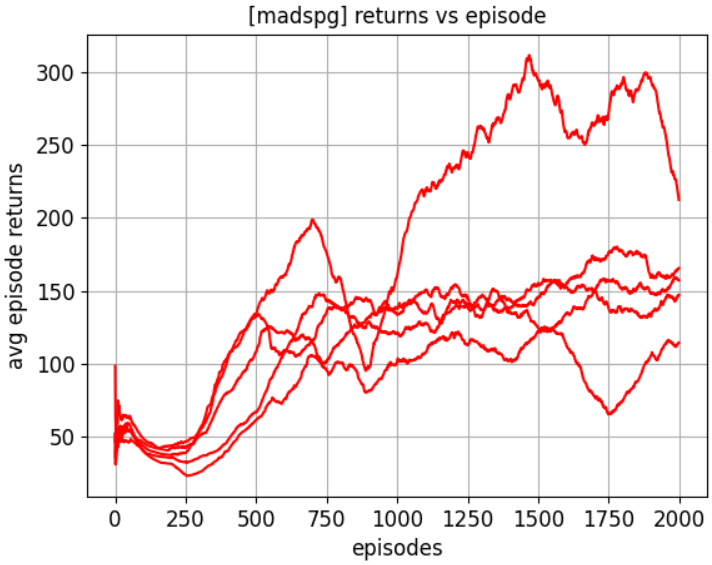


Figure 6: Learning curves for agents trained with MADSPG for 5 trials

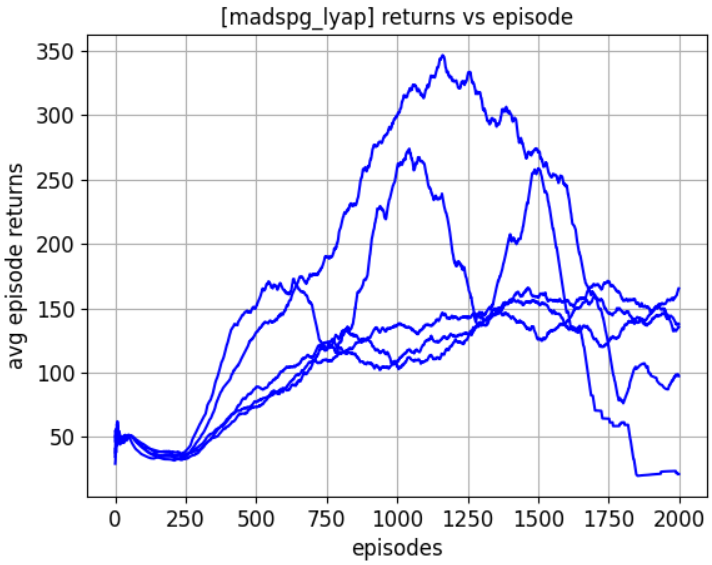


Figure 7: Learning curves for agents trained with L-MADSPG for 5 trials

8.3 Additional equations

The average lateral error per step is computed using:

$$avg\ l.e = \frac{error_{agent1} + error_{agent2}}{2} \tag{53}$$

The average lateral error per episode is computed using:

$$avg\ l.e = \frac{avg\ l.e_1 + avg\ l.e_2 + ... + avg\ l.e_n}{n} \tag{54}$$

The average standard deviation over all episodes is computed using:

$$avg\ S.D. = \sqrt{\frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2 + (n_3 - 1)s_3^2...}{n_1 + n_2 + n_3... - k}} \tag{55}$$

where n_i is the number of steps in episode i, s_i is the standard deviation of average lateral error in episode i and k is the number of episodes.