Centralizing State-Values in Dueling Networks for Multi-Robot Reinforcement Learning Mapless Navigation

Enrico Marchesini*, Alessandro Farinelli

***Abstract*— We study the problem of multi-robot mapless navigation in the popular Centralized Training and Decentralized Execution (CTDE) paradigm. This problem is challenging when each robot considers its path without explicitly sharing observations with other robots and can lead to non-stationary issues in Deep Reinforcement Learning (DRL). The typical CTDE algorithm factorizes the joint action-value function into individual ones, to favor cooperation and achieve decentralized execution. Such factorization involves constraints (e.g., monotonicity) that limit the emergence of novel behaviors in an individual as each agent is trained starting from a joint action-value. In contrast, we propose a novel architecture for CTDE that uses a centralized state-value network to compute a joint state-value, which is used to inject global state information in the value-based updates of the agents. Consequently, each model computes its gradient update for the weights, considering the overall state of the environment. Our idea follows the insights of Dueling Networks as a separate estimation of the joint state-value has both the advantage of improving sample efficiency, while providing each robot information whether the global state is (or is not) valuable. Experiments in a robotic navigation task with 2 4, and 8 robots, confirm the superior performance of our approach over prior CTDE methods (e.g., VDN, QMIX).**

I. Introduction

Multi-Agent Reinforcement Learning (MARL) [1] considers multiple learning agents (or robots) that have to optimize either an individual or a joint reward signal accumulated over time. In particular, we are interested in Multi-Robot navigation as it recently gained attention due to the wide range of applicability (e.g., multi-robot search [2]).

Multi-robot navigation has been previously considered as a cooperative MARL problem and requires the learning of an efficient collision avoidance policy [3]. Among the different approaches that can address MARL, *centralized learning* aims at reducing the task to a single-robot problem that can be solved with standard DRL algorithms. However, these centralized solutions require comprehensive knowledge about all the robots' observations (e.g., positions, velocities) and their workspace (e.g., a grid map), which are concatenated to form a joint observation. Intuitively, centralized methods do not scale well within the number of robot and consistently fail in practice due to the "lazy" agent problem [4] (i.e., an individual that is discouraged from learning the policy as its exploration can hinder other successful agents).

Conversely, *independent learning* consists of individual agents that have to face a non-stationary problem, due to the dynamics of the environment that changes within the other unobserved policies. In practice, the non-stationarity can be addressed by using recurrent units (e.g., Long Short-Term Memory), but the lack of shared information does not favor cooperative behaviors. However, independent learning is typically used as a baseline as it represents a simple yet good-performing solution for a variety of MARL tasks [5].

Both the scalability and the non-stationarity issues have been considered by the recent *Centralized Training with Decentralized Execution* paradigm, which has been used to design value-based MARL algorithms [4], [6], [7] that showed state-of-the-art results on several multi-agent tasks (e.g., multi-agent particle envs [8]). CTDE involves the training of robots' policies that use global information in a centralized way and rely only on local observations for the action selection, hence enabling the decentralized execution. In more detail, to address the combinatorial action space and the dimensionality of the joint observations, VDN [4], QMIX [6], and QTRAN [7] propose different mechanisms to factorize the joint (or global, interchangeably) action-value function, which we refer to as Q_G for notation. However these approaches suffer from a common limitation as they introduce structural constraints (e.g., additivity and monotonicity) to ensure the factorization of Q_G into the agent individuals Q_i-values (with $i \in [1,..,N]$, where N is the number of robots) that are used for the decentralized action selection process. Such constraints severely limit the joint action-value function class that VDN and QMIX can represent, while QTRAN showed poor practical performance despite its strong theoretical factorization guarantees [7].

To address these limitations, we present a novel value-based CTDE approach (Fig. 1), called *multi-agent Global Dueling Q-learning* (GDQ), that in contrast to prior work

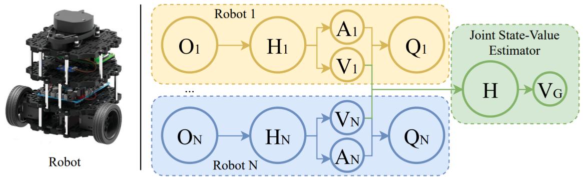


Fig. 1: Left: Robotic platform. Right: Architecture of GDQ. Each robot uses local observation O_i (and hidden layers H_i) to compute the Q_i-values for a decentralized execution. Individuals Dueling Architecture compute the state-values V_i that are used as input for the joint state-value estimator.

*Contact author: enrico.marchesini@univr.it
Authors are with the Department of Computer Science, University of Verona, 37135 Verona, Italy.

exploits the state-values V[1] of each agent to estimate a joint state-value for the system, which is used in the update of the agents (as detailed in Section III). In more detail, GDQ centralizes the individuals state-values of the agents, that are estimated using Dueling Networks [9], to compute a joint state-value V_G with a Double Deep Q-Network (DDQN) [10] (which is trained using the cumulative reward signal). Our goal is to avoid the issues of factorizing Q_G by exploiting the separate state-value V and advantage A functions of each agent. As showed by Dueling Networks, the combination of V and A results in the agent's action-value Q, from which the action of an agent is sampled. Intuitively, the state-value function measures how good it is to be in a state, whether the advantages are the relative importance of the actions. The combination of these values $Q = V + A$ is then used to measure the value of choosing a particular action in the state. Hence, we use the agents' state-values as input for the centralized DDQN, which is used to learn a joint state-value V_G. This V_G replaces V in the estimation of the temporal difference target of each agent, and also has the advantage of addressing the overestimation problem of Deep Q-Network-based algorithms that typically requires a target network [10]. Our idea is to inject information on the global environment state in each agent update rule, such that the robots optimize their behaviors while considering the state of the system.

We first evaluate the performance of GDQ on the explanatory *Cooperative navigation* task of the multi-agent particle envs benchmark suite [8], to confirm the superior performance of GDQ with respect to prior value-based CTDE algorithms (i.e., VDN and QMIX). Hence, we present a Turtlebot3[2] multi-agent navigation task with 2, 4, and 8 robots, to highlight the scalability and the beneficial effects of GDQ in increasingly complex collision avoidance situations.

II. Background and Related Work

Similar to previous CTDE approaches, we can describe our multi-robot navigation scenario as a Dec-POMDP [11] (we remind the interested reader to prior work for details about the Dec-POMDP formalization [6], [12]). Given the partial observability setting of the tasks, the Q-functions are defined over individual observation histories, hence we incorporate recurrent units (i.e., an LSTM layer) in the robots' networks. For each agent, we also consider a common setup with Prioritized buffers [13] and n-step returns [14] to speed up the training.

A. Deep Reinforcement Learning for Robotic Navigation

Robotic navigation is a well-known problem in recent DRL literature [15], [16], [17], [18]. This class of problems has been naturally extended to the MARL domain, due to the wide range of multi-robot navigation applications, such as search and rescue [2], or collision avoidance solutions [3]. Typical solutions to these problems consider policy-gradient algorithms with continuous high-dimensional action spaces. However, recent work [19], [20] show that value-based methods with discrete action spaces achieve similar performance while drastically reducing training time. Furthermore, several different methods are renewing the interest in the use of discrete action spaces, showing that value-based DRL algorithms can also handle high-dimensional actions. In more detail, the Branching Architecture [21] presents an adaptation of the Dueling Network (that we consider in our work) and obtain competitive results over continuous algorithms in classical MuJoCo locomotion tasks [22], with several other value-based methods that confirm these results [23], [24]. Hence, similarly to the CTDE baselines [4], [6], we use discrete action spaces for our tasks in Section III, IV.

B. Centralized Training with Decentralized Execution

CTDE has recently attracted attention as multi-agent DRL [4], [6], [7] paradigm. The main idea introduced in one of the first DRL CTDE approach, the actor-critic MADDPG [5], is that the individuals are trained using centralized information (e.g., Q_G) or the global state (e.g., positions and goals of the other robots) during the training phase. However, the decision-making process of each agent is strictly independent as it is based only on the local action-observation history, guaranteeing a decentralized execution.

Since MADDPG (and the actor-critic COMA [25]), the main research trend is to develop CTDE value-based approaches, that aim at satisfying the *Individual-Global-Max* (IGM) requirement, which state that the optimal joint action induced from Q_G is equivalent to the collection of the individuals Q_i-values. Hence, this asserts that $Q_G(\tau, \mathbf{a})$ (where τ is the joint action-observation history, and $\mathbf{a} \equiv [a_i]_{i=1}^N$ is the joint action) is factorizable if and only if exists $[Q_i : \mathbf{T} \times \mathbf{A} \rightarrow \mathbb{R}]_{i=1}^N$ such that $\forall \tau \in \mathbf{T}$:

$$\underset{\mathbf{a} \in \mathbf{A}}{\arg\max}\, Q_G(\tau, \mathbf{a}) = \begin{pmatrix} \arg\max_{a_1 \in A} Q_1(\tau_1, a_1) \\ \vdots \\ \arg\max_{a_N \in A} Q_N(\tau_N, a_N) \end{pmatrix} \tag{1}$$

we can summary Eq. 1 saying that a global $\arg\max$ on Q_G returns the same result as a set of $\arg\max$ on each Q_i (with $i \in [1, .., N]$). Two different factorization strategies for satisfying the IGM have been proposed: (i) *additivity* by VDN [4], and (ii) *monotonicity* by QMIX [6]. These factorizations are sufficient for the IGM principle. However, they limit the representation expressiveness of the joint action-value function Q_G [26]. QMIX also considered additional *hypernetworks* [27] to add additional global state information to the robot individual observations, introducing overhead. QTRAN [7] uses a relaxation of the IGM constraint in the form of a linear constraint and soft regularization, which despite the theoretical guarantee result in poor performance.

In contrast to this research trend that aims at factorizing Q_G, and use additional global state information to favor cooperation, GDQ is more similar to the *independent learning* paradigm, in the sense that the robots do not share any additional global observation among each other (to favor

[1] We omit the state s and action a parameters for notation simplicity.

[2] https://www.turtlebot.com/

cooperation), but they only rely on an additional estimation of a joint state-value V_G, which uses the individuals V as input (i.e., it is a form of centralized training). Crucially, V_G is only used in the temporal difference target computation of each agent, hence GDQ is also based on the CTDE paradigm.

III. Methods

In this section, we first introduce our multi-agent Global Dueling Q-learning algorithm in detail. We also discuss the benefits of computing a joint state-value with respect to prior CTDE approaches that factorize the joint action-value to satisfy the IGM, which is the main limitation of such methods. We use a multi-agent benchmark, the *Cooperative Navigation* task of the particle envs suite [8] as a motivating example to show the superior performance of our approach.

A. Multi-Agent Global Dueling Q-learning

GDQ uses a DDQN algorithm for each agent[3], enhanced with priority buffer [13], n-step returns [14], and more importantly the dueling architecture [9]. The insights of Dueling Networks, in fact, represents the foundation of our value-based CTDE approach. In detail, the agents' Q-networks maintain two streams to represent both the individual state-value function $V(s)$, and the individual advantage function $A(s,a)$. The streams are combined with an aggregation layer to produce the agent's action-value function $Q(s,a)$, which in its simplest form computes the Q-values as follows:

$$Q(s,a) = V(s) + A(s,a) \tag{2}$$

The use of a separate representation for the state-value and the action advantages allows to learn whether the considered state s is either valuable or not, and crucially $V(s)$ does not influence the action selection process of an agent, which is strictly dependent on the advantage values:

$$\arg\max_a Q(s,a) \equiv \arg\max_a A(s,a) \tag{3}$$

The action-value decomposition improves the sample efficiency of the training process [9]. For example, in some states, it is crucial to know which action to take (e.g., to avoid collisions), but in many other states (e.g., where obstacles are far from the agents) the difference between various actions can be negligible. This is mainly related to two factors: (i) the direct estimation of the Q-values (as in standard Deep Q-Networks) requires to calculate the value of each action at each state, which is a non-negligible overhead when the whole set of possible actions do not affect the environment in a relevant way. For example, in a navigation scenario, we could move to the left or to the right only when there is a risk of collision, otherwise, the choice of action has no major effect on what happens. As detailed by prior work [9], this also aggregates similar actions, which further improves the sample efficiency. (ii) In a Dueling architecture, where state-value is estimated independently by a stream of the network, $V(s)$ is learned more efficiently as its separate estimation stream is updated with every update of the Q-values.

Note that Eq. 2 is unidentifiable, in the sense that we can not uniquely recover the state-value and the advantages function from given Q-values. For example, if we multiply the state-value and divide the action advantages by the same constant value, we will obtain the same Q-values. This causes poor practical performance and has been addressed using different aggregation functions to combine the two streams. Hence, we consider the the best operator identified in the original work [9] for the aggregation layer of each agent:

$$Q(s,a) = V(s) + \left(A(s,a) - \frac{1}{|A(s,a)|} \sum_{a' \in A(s,a)} a' \right) \tag{4}$$

where $|A(s,a)|$ is the cardinality of the advantage function's stream, i.e., the number of possible actions. Given the weights of each agent θ_{a_i} and the estimated joint next state-value V'_G (described in the next section), GDQ optimizes at iteration t the following loss function for each agent:

$$L_t(\theta_{a_i}) = \mathbb{E}_{s,a,r,s',V'_G} \left[\left(y_t(r,s',V'_G|\theta_{a_i}) - Q(s,a|\theta_{a_i}) \right)^2 \right] \tag{5}$$

where s,a,r,s' is a tuple of state, action, reward, next state (at iteration t) sampled from the agent's memory. In detail, the target is computed by replacing the next state-value, with the one computed by our centralized joint state-value estimator:

$$\begin{aligned} y_t(r,s',V'_G|\theta_{a_i}) &= r + \gamma \ \arg\max_{a'} Q(s',a'|\theta_{a_i},V'_G) \\ Q(s',a'|\theta_{a_i},V'_G) &= V'_G + A(s',a'|\theta_{a_i}) \end{aligned} \tag{6}$$

As previously discussed, this does not influence the action selection process of an agent as it depends on the advantage function, but it serves to inject learned information on the global state value. The other natural contribution of the introduction of V'_G is that it naturally mitigates the overestimation bias of the standard Deep Q-Networks, and it is possible to remove the requirements of a target network for each agent (as it is typically used to compute $Q(s',a')$ to reduce the overestimation and improve the stability of the training).

B. Joint State-Value Estimator

We use a standard DDQN algorithm to compute the joint state-value V'_G for the agents. This network takes as input all the agents' next state-values $[V_1(s'_t),\ ...,\ V_N(s'_t)]$ computed by their separate stream to learn an estimation of the joint state-value V'_G. The training procedure for this centralized component resembles the standard DDQN algorithm [10] (which we refer the interested reader for further details), but considers state-values instead of action-values and the cumulative reward of the agents. In more detail, for each environment iteration t we store in a memory buffer the concatenated agents' state-values $\mathbf{v} = [V_1(s_t),\ ...,\ V_N(s_t)]$, the next state-values $\mathbf{v}' = [V'_1(s'_t),\ ...,\ V'_N(s'_t)]$ (that are computed

[3] Here we refer only to local state/action information, where s could be interpreted as a local observation history in the partial-observable scenario.

with an additional forward step of the agents' networks) and the agents' cumulative rewards $\mathbf{r}$. Hence, given the weights of both the centralized joint state-value estimator θ_c and its target network $\theta_{c'}$ and a sample from the memory, it aims at optimizing the following loss function:

$$L_t(\theta_c) = \mathbb{E}_{\mathbf{v},\mathbf{v}',\mathbf{r}}\left[\left(y_t(\mathbf{r},\mathbf{v}'|\theta_{c'}) - V(\mathbf{v}|\theta_c)\right)^2\right] \tag{7}$$

with $V(\mathbf{v}|\theta_c)$ computed by a forward pass of the joint state-value estimator with input $\mathbf{v}$, and target y_t is computed as:

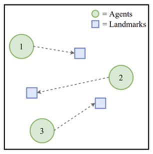
$$y_t(\mathbf{r},\mathbf{v}'|\theta_{c'}) = \mathbf{r} + \gamma\, V(\mathbf{v}'|\theta_{c'}) \tag{8}$$

Besides the benefits of GDQ, the decoupled nature of the state-value stream could also address competitive MARL, using separate joint state-value estimator for each agent group. We intend to explore this direction as future work.

C. Motivating Example

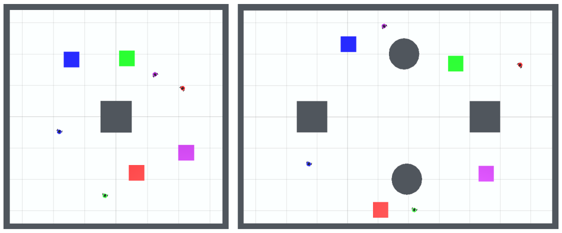
To evaluate the performance of GDQ over previous similar CTDE baselines (VDN [4] and QMIX [6], as QTRAN typically result in poor performance [7]), and *independent learners* (IQL) based on DDQN with priority buffer and n-step updates (i.e., similarly to a GDQ agent), we consider the *Cooperative navigation* task of the *multiagent particle envs* suite [8], as a benchmarking example. All the networks share a similar architecture with 2 hidden layers with 64 *tanh* neurons and Adam optimizer with default learning rate (for algorithmic specific hyper-parameters, we refer the interested reader to the original work of VDN [4] and QMIX [6] as we used the authors' implementation). In this cooperative problem depicted in Fig. 2 on the left, 3 agents have to cover the same number of landmarks and are rewarded based on how far each one is from each landmark, while they are penalized if they collide with other agents.

Results in Fig. 2 right shows the average reward, average distance from the landmarks, and collision percentage collected in this evaluation over 5 independent runs. Training time where similar for all the considered approaches and was ≈100 minutes. These values confirm the superior performance of GDQ over all the considered baselines as it is able to obtain the highest average reward (i.e., minimum average distance), while also favoring information exchange, resulting in the lowest percentage of collisions. Moreover, as detailed in Section I, IQL obtained good performance despite the simplicity of the approach, while the superior performance of QMIX over VDN and IQL confirm the original authors' results [6].



	Reward	Distance	Collisions (%)
IQL	-2.41 ± 0.31	1.39 ± 0.11	34.2 ± 2.2
VDN	-2.45 ± 0.21	1.43 ± 0.13	32.5 ± 1.9
QMIX	-2.23 ± 0.24	1.25 ± 0.09	25.6 ± 1.7
GDQ	-2.06 ± 0.21	1.02 ± 0.10	18.7 ± 1.7

Fig. 2: Left: *Cooperative navigation* environment. Right: Average reward, Distance from the landmarks, and Collision percentage for IQL, VDN, QMIX, and GDQ.



(a) Training environment

(b) Testing environment

Fig. 3: Left: Training environment for our MARL navigation. Right: Testing environment for the trained policies. Explanatory view with $N = 4$ robots in different colors.

IV. Evaluation

Here we introduce our multi-robot mapless navigation for our training and testing experiments with 2, 4, and 8 robots. We then present the results of our evaluation, discussing the benefits of GDQ in the considered robotic navigation tasks.

A. Mapless Navigation Environment Description

Mapless navigation is a well-known benchmark in recent DRL literature [15], [19], [16], and aims at navigating a robot towards a random target using only local observation and the target position, without a map of the surrounding environment or obstacles. Given the wide range of real-world task that involves robot navigation, this problem has been naturally extended to the multi-robot domain [2], [28]. In this paper, we consider a setup similar to the previous single-robot navigation scenarios [19], [16], adapted for MARL.

In more detail, we consider a Turtlebot3 indoor navigation environment with $N \in [2,4,8]$ robots and fixed obstacles, where each robot has to navigate to its target avoiding collisions. Given the robot specifications, we consider a max angular velocity of 90 deg/s and a max linear velocity $= 0.2\ m/s$. The decision-making frequency of the robot in both the training and testing environments is set to 20Hz, to reflect the update rate of the equipped LDS-01 lidar sensor. This training environment is depicted in Fig. 3a and is built with the Unity ml-agents toolkit [29]. As demonstrated in [19], Unity enables rapid development of simulation environments, which also allows the export of trained models to the more conventional Robot Operating System (ROS). To detect collisions among the robots and the obstacles, we used Unity's mesh collider system, applied to the 3D models of the environment. This allows triggering a collision event when an intersection between the bounds of two or more colliders is detected. The target goals of each randomly spawn in the scenario and are guaranteed to be obstacle-free and with a minimum distance between each other (set to $0.5m$). Each agent receive a separate reward r_t that is structured as follows:

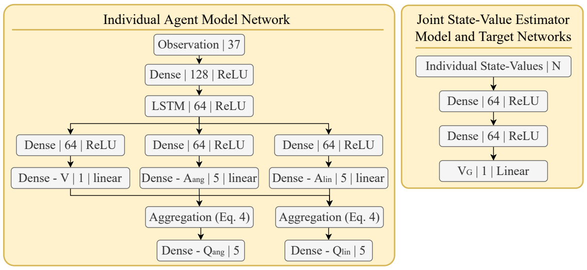


Fig. 4: Left: robot network architecture. Right: join state-value estimator network architecture

$$r(s_t) = \begin{cases} \alpha(d_{t-1} - d_t) - 0.005, \text{ if } d_t > 0.1 \\ 1, \text{ if } d_t <= 0.1 \\ -1, \text{ if a collision is detected} \end{cases} \tag{9}$$

hence we have two sparse value in case of reaching the target within a threshold distance $d_t = 10cm$ from the robot's goal (i.e., $r_t = 1$), or crashing within the obstacles or other robots (i.e., $r_t = -1$) which terminates an episode resetting the robot to its starting position and generating a new set of goals. A dense component is used during the travel: $\alpha(d_{t-1} - d_t) - 0.005$, where d_{t-1}, d_t is euclidean distance between a robot and its goal at two consecutive time steps and α is a multiplicative factor that is used for normalization. A penalty of 0.005 is applied at each time-step, to encourage the robots to perform the shortest path possible.

1) Network Architecture: The input layer of each network contains a 35-dimensional vector with sparse laser scan values sampled in $[-120, 120]$ degrees in a fixed angle distribution, and the individual target position is expressed as the distance from the robot and relative heading. We considered a similar setting of previous DRL literature for robotic navigation [19], [16]. The output layer considers the Branching Architecture [21] to output two streams of Q-values: one for angular velocities $v_{ang} \in [-90, -45, 0, 45, 90]$ deg/s and one for linear velocities $v_{lin} \in [0, 0.05, 0.1, 0.15, 0.2]$ m/s. Hence, following the GDQ architecture described in Section III, each network maintains three separate streams: one for the advantage of v_{ang}, one for the advantage of v_{lin}, and one to estimate the state-value. The two advantage streams are then combined with the state-value using Eq. 2. Given the importance of the statistical significance of the data, highlighted by several recent work [30], [31], we performed an initial evaluation in the training environment with multiple trials on different network sizes and seeds. The outcome led us to use the network architectures depicted in Fig. 4, with one *ReLU* hidden layer with 128 neurons, a *ReLU* LSTM layer of size 64, plus an additional *ReLU* hidden layer of 64 neurons for each stream. The joint state-value estimator, is a simple feed-forward network with two *ReLU* hidden layers of 64 neurons each and a *linear* output.

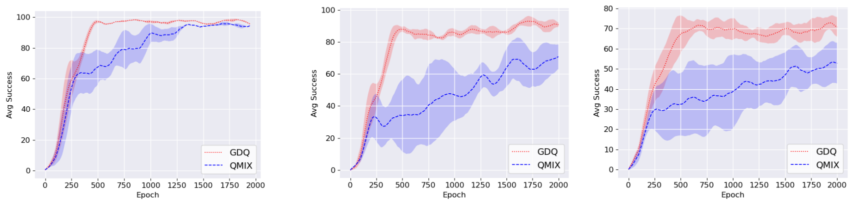
B. Empirical Evaluation

Given the results in our preliminary evaluation in Section III-C, we evaluate the performance of the two best-performing algorithms for our multi-robot mapless navigation: QMIX, and the proposed GDQ. The goal of this evaluation is to investigate whether GDQ can successfully address a robotic task of real interest while favoring cooperation as detailed in Section III. Data are collected on an i7-9700k, using a network architecture for QMIX similar to the one described in the previous section in terms of layer numbers and sizes (while we refer the interested reader to the original work to further details about QMIX *hypernetworks* [6]). Every experiment with N robots $\in [2, 4, 8]$ is performed over ten different independent runs with different seeds, for statistical significance of the collected data. For each experiment, we report the following curves that show mean and standard deviation over the runs, smoothed over 100 epochs. In detail, for each setting, we show the average success rate: how many successful obstacles-free trajectories are performed over a sequence of 100 epochs. In the evaluation in an unseen scenario (Tab. I considers average reward: the cumulative reward for each epoch, and average path length: represented as the number of steps in the environment for each epoch. All these metrics are averaged over the N robots.

1) Training Results: Fig. 5 shows the results of GDQ and QMIX in our robotic navigation tasks, where the robots have to learn how to reach different target positions, avoiding collisions with the fixed obstacles and with each other.

For each task with $N \in [2, 4, 8]$ Turtlebot3 robots we show the average success rate. While the results are comparable when considering only 2 robots (i.e., Fig. 5a), with a growing number of robots GDQ clearly offers better performances. In more detail, with $N = 4$ (i.e., Fig. 5b) the percentage of successes stabilizes at over $\approx$90% in $\approx$1700 epochs, i.e., 150 minutes of training. QMIX, in contrast, reaches $\approx$80% successes in the same epochs that correspond to 190 minutes of training. It naturally follows that GDQ offers better performance even in terms of reward and, as shown in the following evaluation, it performs on average a lower number of steps, which means shorter paths. The evaluation with $N = 8$ confirms the superior performance of GDQ.

2) Testing Results: Crucially, after the training phase, the converged models can navigate in the environment in a decentralized fashion, exploiting the minimal information in the input layer of the network. Moreover, the individuals generalize crucial aspects of robot navigation such as (i) robot starting position, (ii) target position, (iii) velocity. The laser scan-based navigation also allows our robots to navigate in previously unknown environments with different obstacles, which is a key feature for motion planning. To confirm the successful generalization of our models, we run an additional evaluation with $N \in [4, 8]$ in a previously unseen scenario, depicted on the right Fig. 3. Results in Tab. I confirm the trend highlighted during the training, where GDQ offers superior performance over the baseline in terms of average reward and path length when the same sequence of goals is considered for both algorithms. In this evaluation, we also collect the percentage of collisions detected, which further confirms a better generalization of the task with multiple robots in the case of GDQ.



(a) $N = 2$ (b) $N = 4$ (c) $N = 8$

Fig. 5: Average Success Rate of the GDQ and QMIX robots in our training phase.

TABLE I: Average Reward, Steps and Collision percentage in the evaluation over ten runs on different targets in the unseen scenario depicted in the right Fig. 3

	N	Reward	Steps	Collisions (%)
QMIX	4	26.3 ± 3.1	295 ± 23	1.4 ± 2.6
GDQ	4	30.1 ± 2.7	242 ± 18	0.8 ± 1.8
QMIX	8	18.2 ± 3.9	361 ± 29	34.2 ± 4.1
GDQ	8	24.3 ± 2.8	285 ± 22	17.3 ± 2.3

V. Discussion

We presented GDQ, a novel value-based CTDE approach for multi-agent scenarios that exploit state-values information to favor cooperation. We performed an initial evaluation of GDQ and several baselines (i.e., IQL, VDN, QMIX) in the *Cooperative Navigation* benchmark to highlights the superior performance of our approach. Then, we presented a multi-robot robotic navigation scenario built in Unity to evaluate the navigation performance of GDQ with 2, 4, and 8 robots.

Our empirical evaluation shows that GDQ significantly outperforms prior CTDE approaches in multi-robot robotic navigation, especially with a growing number of robots. Crucially, such superior performance is obtained only considering the individual's state-values and no additional centralized information (such as in QMIX), which allows scaling the number of robots without major overhead for the training process (each robot only adds an input node in our centralized joint state-value estimator).

This work paves the way for several interesting research directions which include the exploration of competitive multi-agent tasks using separate joint state-value estimators.

References

[1] K. Tuyls and G. Weiss, "Multiagent learning: Basics, challenges, and prospects," in *AI Magazine*, 2012.

[2] J. L. Baxter, E. K. Burke, J. M. Garibaldi, and M. Norman, "Multi-robot search and rescue: A potential field based approach," in *Autonomous Robots and Agents*, 2007.

[3] P. Long, T. Fan, X. Liao, W. Liu, H. Zhang, and J. Pan, "Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning," in *ICRA*, 2018.

[4] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. F. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, and T. Graepel, "Value-decomposition networks for cooperative multi-agent learning," in *AAMAS*, 2018.

[5] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *NIPS*, 2017.

[6] T. Rashid, M. Samvelyan, C. S. de Witt, G. Farquhar, J. N. Foerster, and S. Whiteson, "QMIX: monotonic value function factorisation for deep multi-agent reinforcement learning," in *ICML*, 2018.

[7] K. Son, D. Kim, W. J. Kang, D. Hostallero, and Y. Yi, "QTRAN: learning to factorize with transformation for cooperative multi-agent reinforcement learning," in *ICML*, 2019.

[8] I. Mordatch and P. Abbeel, "Emergence of grounded compositional language in multi-agent populations," in *arXiv*, 2017.

[9] Z. Wang, N. de Freitas, and M. Lanctot, "Dueling network architectures for deep reinforcement learning," in *ICML*, 2016.

[10] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *AAAI*, 2016.

[11] F. A. Oliehoek and C. Amato, "A concise introduction to decentralized pomdps," in *SpringerBriefs in Intellingece Systems*, 2016.

[12] A. Castellini, E. Marchesini, G. Mazzi, and A. Farinelli, "Explaining the influence of prior knowledge on pomcp policies," in *Multi-Agent Systems and Agreement Technologies*, 2020.

[13] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *ICLR*, 2016.

[14] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in drl," in *AAAI*, 2018.

[15] J. Zhang, J. T. Springenberg, J. Boedecker, and W. Burgard, "Deep reinforcement learning with successor features for navigation across similar environments," in *IROS*, 2017.

[16] L. Tai, G. Paolo, and M. Liu, "Virtual-to-real drl: Continuous control of mobile robots for mapless navigation," in *IROS*, 2017.

[17] D. Corsi, E. Marchesini, and A. Farinelli, "Formal verification of neural networks for safety-critical tasks in deep reinforcement learning," in *UAI*, 2021.

[18] E. Marchesini, D. Corsi, and A. Farinelli, "Genetic soft updates for policy evolution in deep reinforcement learning," in *ICLR*, 2021.

[19] E. Marchesini and A. Farinelli, "Discrete deep reinforcement learning for mapless navigation," in *ICRA*, 2020.

[20] E. Marchesini and A. Farinelli, "Genetic deep reinforcement learning for mapless navigation," in *AAMAS*, 2020.

[21] A. Tavakoli, F. Pardo, and P. Kormushev, "Action branching architectures for deep reinforcement learning," in *AAAI*, 2018.

[22] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *IROS*, 2012.

[23] T. Wiele, D. Warde-Farley, A. Mnih, and V. Mnih, "Q-learning in enormous action spaces via amortized approximate maximization," in *NeurIPS Workshop*, 2018.

[24] G. Dulac-Arnold, R. Evans, P. Sunehag, and B. Coppin, "Reinforcement learning in large discrete action spaces," in *CoRR*, 2015.

[25] J. N. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," in *AAAI*, 2018.

[26] A. Mahajan, T. Rashid, M. Samvelyan, and S. Whiteson, "MAVEN: multi-agent variational exploration," in *NeuIPS*, 2019.

[27] D. Ha, A. M. Dai, and Q. V. Le, "Hypernetworks," in *ICLR*, 2017.

[28] P. Wang, C. Chan, and A. de La Fortelle, "A reinforcement learning based approach for automated lane change maneuvers," in *IEEE Intelligent Vehicles Symposium*, 2018.

[29] A. Juliani, V. Berges, E. Vckay, Y. Gao, H. Henry, M. Mattar, and D. Lange, "Unity: A platform for intelligent agents," in *CoRR*, 2018.

[30] C. Colas, O. Sigaud, and P.-Y. Oudeyer, "A hitchhiker's guide to statistical comparisons of rl algorithms," in *arXiv*, 2019.

[31] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *AAAI*, 2018.