Guided Soft Actor Critic: A Guided Deep Reinforcement Learning Approach for Partially Observable Markov Decision Processes

Mehmet Haklidir\textsuperscript{1,2}, Hakan Temeltas\textsuperscript{1}, Member, IEEE
\textsuperscript{1}Department of Control and Automation Engineering, Istanbul Technical University, Maslak, Istanbul, TURKEY
\textsuperscript{2}Information Technologies Institute, TÜBİTAK Informatics and Information Security Research Center, Gebze, Kocaeli, TURKEY

Corresponding author: Mehmet Haklidir (e-mail: mhaklidir@itu.edu.tr).

ABSTRACT Most real-world problems are essentially partially observable, and the environmental model is unknown. Therefore, there is a significant need for reinforcement learning approaches to solve them, where the agent perceives the state of the environment partially and noisily. Guided reinforcement learning methods solve this issue by providing additional state knowledge to reinforcement learning algorithms during the learning process, allowing them to solve a partially observable Markov decision process (POMDP) more effectively. However, these guided approaches are relatively rare in the literature, and most existing approaches are model-based, meaning that they require learning an appropriate model of the environment first. In this paper, we propose a novel model-free approach that combines the soft actor-critic method and supervised learning concept to solve real-world problems, formulating them as POMDPs. In experiments performed on OpenAI Gym, an open-source simulation platform, our guided soft actor-critic approach outperformed other baseline algorithms, gaining 7\textasciitilde20\% more maximum average return on five partially observable tasks constructed based on continuous control problems and simulated in MuJoCo.

INDEX TERMS Deep reinforcement learning, guided policy search, POMDP

I. INTRODUCTION
Recently, deep learning techniques have been implemented in reinforcement learning (RL) as function approximators, yielding deep RL, to solve high-dimensional, continuous state space problems [1]-[4]. In Markov decision process (MDP)-based deep RL approaches, it is generally assumed that the agent can fully observe the true state of the environment and that the environment's next state is solely determined by the current state and the chosen action. This ensures that the agent can act in the best way by simply reacting to the current state of the environment. However, this assumption is not true for many real-world problems. In real-world environments, agents often perceive the state of the environment to be partially and noisily. Another way of saying that the Markov property is seldom found in real-world environments. A reactive policy found by the MDP cannot be said to be optimum if the observations do not hold the Markov property. In such cases, the partially observable Markov decision process (POMDP) approach is used. POMDP extends the MDP by allowing an agent to represent a subsequent decision-making process under uncertainty.

The development of RL algorithms for solving POMDP poses many challenges. The agent explicitly observes state transitions in the MDP, and the generative model is easily estimated using empirical estimators. In POMDP, partial and noisy observations must be used for the transition and reward models. POMDPs require a range of observation probabilities outside of direct observation and a probability distribution across all possible situations based on MDP to predict the next state. To compute the optimal policy for a POMDP with known parameters, an augmented MDP based on a continuous domain of belief must be solved. Finally, it is not trivial to integrate prediction and planning with guarantees in a discovery, and no regret strategy is currently known. Guided RL methods solve this issue by providing additional state knowledge to RL algorithms during the learning process, allowing them to solve POMDPs more effectively.

In the literature, most deep RL approaches have been applied to Markov decision processes. The deep RL for
POMDPs has not been sufficiently studied. Following the successful performance of the deep Q-network (DQN), several approaches have been developed for POMDPs, mainly focusing on remembering some important observations, implementing recurrent neural networks (RNNs) [5–8]. However, especially in real-world scenarios, there are also partially observable environments in which the agent must consider account information gathering and therefore engage in much more exploration and interaction with these environments to learn how to collect and use the information. There are very few studies on these scenarios [9–16].

In this study, we propose a novel approach that combines the soft actor-critic method [17] and supervised learning concepts to solve real-world problems, formulating them as POMDPs. In experiments performed on OpenAI Gym [18], an open-source simulation platform, our guided SAC approach outperformed other baseline methods trained directly on five partially observable tasks constructed based on continuous control problems, simulated in the MuJoCo [19]. The main contribution of this study is as follows: Our proposed guided SAC approach outperforms the comparison methods on partially observable continuous control experiments owing to our proposed novel architecture and interaction loop for the classic soft actor–critic. Unlike the original concept of the SAC, in the training phase, the guided-SAC architecture consists of two actors and a critic. To generate the samples, two actors iteratively interact with the same environment. The final control actor chooses actions based on partial observations, while the guiding actor chooses actions by perceiving the full-state observations. Both the final control and guiding policies are trained to maximize both the expected reward and entropy. Furthermore, a DNN is trained to teach the control policy actor to act as a guiding policy actor.

This paper is organized into five sections, including the introduction. The second section provides a summary of the deep reinforcement learning taxonomy, a brief review of the guided policy search methods, and a detailed literature survey of the deep reinforcement learning approaches for partially observable Markov decision processes. The third section provides the background information necessary to comprehend our novel approach. Section gives a detailed explanation of the proposed methodology. The fifth section provides the results obtained and discusses the results. The sixth section presents the conclusion and future work plan of this study.

II. RELATED WORK
In this section, we first summarize the taxonomy of the DRL algorithms. Second, we provide a brief review of guided policy search methods from a historical perspective. In the last part of the section, we provide a detailed literature survey of deep reinforcement learning approaches for a partially observable Markov decision process.

A. TAXONOMY OF DEEP REINFORCEMENT LEARNING ALGORITHMS
Reinforcement learning is a machine learning subfield in which an agent attempts to learn actions by maximizing the reward received through interaction with the environment [20]. Conventional RL policies such as tabular, linear, and radial basis networks have a number of drawbacks, especially in terms of high-level behavior representation and the computational cost of updating parameters. Combining RL and deep learning can extract more information from raw inputs by passing them through multiple neural layers, which can have a large number of parameters and represent highly nonlinear problems [10]. By using deep learning as a function approximator in RL [21], deep RL methods can approximate optimal policies and/or value functions.

All methods in deep reinforcement learning can be divided into the following categories: model-free/model-based, value-based/policy-based, and on-policy/off-policy. Figure 1 shows the taxonomy of deep reinforcement learning methods. In real-world problems, interaction with the physical world may be restricted for a variety of reasons, such as safety and cost. The amount of interaction required in the real world can be reduced by learning a dynamic model of the environment. Further, exploration can be performed on the learned models. With model-based approaches, the agent tries to learn the transition and reward functions, which are utilized when taking action. Gathering information on the dynamics of the environment is required to form a model approach. Unlike model-based approaches, this type of information is not required in model-free approaches. Model-free approaches directly illustrate the underlying MDP to learn about an unknown model. Model-free policy search methods require more training data and a longer training time than model-based methods [22].

Value-based methods estimate value functions directly and infer policies that should be implemented. These value functions are calculated by computing the immediate reward and discounting the next state value with the previous state. Policy-based methods specifically discover policies that can take action with no value function values [22]. The distinction between value-based and policy-based methods is based on the trade-off between better performance and more complex strategies. Both methods need to propose actions and evaluate the outcome of the actions. However, value-based methods emphasize determining the maximum cumulative return and have a policy that is based on recommendations, policy-based methods try to directly establish the optimal policy that maximizes the cumulative discounted reward. Furthermore, actor-critic approaches are also known as hybrid approaches that combine value and policy-based methods [22].
FIGURE 1. Taxonomy of Deep Reinforcement Learning Algorithms

On-policy algorithms are RL algorithms that train agents based solely on current policy experience. They are sample-inefficient in the context of changing the agent's policy and actions through training. This ensures that prior experience cannot be considered for training purposes. Off-policy algorithms can employ the experience gained from other policies [66].

B. GUIDED POLICY SEARCH
Guided policy search algorithms assist policy learning with additional information and mechanisms. The type of assistance can vary depending on the domain of the problem or the policy search method used.

Levine and Koltun [67] proposed a guided policy search method using trajectory optimization to learn policy directly and prevent poor local optima. Levine and Koltun [67] employed an importance-sampled variation of the likelihood ratio estimator to incorporate the off-policy guiding samples created using DP into the policy search. Guided policy search algorithms have been applied to various domains, such as controlling locomotion tasks, robotic manipulation, and control [68-70]. Levine and Abbeel [71] performed a guided policy search under unknown dynamics with a hybrid method and optimized trajectory distributions for large-continuous problems using iteratively refitted local linear models. Zhang et al. [72] proposed an approach to decompose the policy search problem into two phases: trajectory optimization and supervised learning, in order to acquire policies with effective memory and recall procedures in continuous control tasks. In the context of guided policy search, Zhang et al. [73] presented an approach that integrated model predictive control (MPC) and RL. To supervise the learning of the neural network policy, Zhang et al. [73] employed MPC, which only received the full state of the system during the training phase. Zhang et al. [73] used a model-based technique to create guiding samples for guided policy search by combining MPC with offline trajectory optimization and assuming that a system dynamics approximation model is obtained in the training phase.

To improve their effectiveness in solving POMDPs, guided RL techniques use additional state information throughout the learning process. There are only a few prominent examples, such as the studies by [67] or [73] on guided policy search.

C. DEEP REINFORCEMENT LEARNING METHODS FOR POMDPs
Deep RL methods have been applied to partially observable Markov decision processes for both discrete [5-10] and continuous [11-16] control tasks. (Table 1).

Egorov [5] suggested the extension of the DQN approach and mapped an action-observation history to an optimal action to solve POMDPs. Egorov [5] showed that DQNs can learn good policies but require substantially more computational resources, and while the Q values converge, policies are vulnerable to minor disturbances. Another approach, called the deep recurrent Q-network (DRQN), can be implemented...
in a model-free representation using additional LSTM layers [6]. The qualified policy network can capture all historical data in the LSTM layer, and the output actions are based on all historical observations. After performing extensive analysis on DRQN, Foerster et al. [7] proposed deep distributed recurrent Q-networks (DDDRQN), which used both prior observations and actions as inputs. The action-specific deep recurrent Q-network (ADRQN) uses action-observation pairs to obtain an optimal policy in partially observable domains [8]. Jin et al. [9] proposed a novel algorithm called ARM that does not require access to a Markovian state and is based on the notion of regret minimization. They demonstrated that regret estimating approaches are better at handling partial observability than regular RL.

Jin et al. [9] proposed a novel algorithm called ARM that does not require access to a Markovian state and is based on the notion of regret minimization. They demonstrated that regret estimating approaches are better at handling partial observability than regular RL. Le et al. [10] proposed a hierarchical deep RL approach for learning in a hierarchical POMDP. Instead of using the deep deterministic policy gradient (DDPG) for continuous regulation, Heess et al. [11] proposed a recurrent deterministic policy gradient (RDPG) for the POMDP, which involves using recurrent neural networks (RNNs) to approximate the actor and critic. The autonomous navigation of an unmanned aerial vehicle (UAV) was modeled in [12] under uncertainty in a large-scale complex environment as a POMDP. Wang et al. [12] proposed a fast-RDPG using an actor-critic-based method with a function approximation. [13] extended RDPG to solve partially observable bipedal locomotion tasks; however, only one was investigated. The fast-RDPG generates policy updates by directly maximizing the expected long-term accumulated discounted reward. Igl et al. [14] proposed deep variational RL (DVRL) using inductive biases in the network to learn a generative model that can imitate belief updates. DVRL is a state-of-the-art deep RL method for POMDPs. DVRL uses the evidence lower bound (ELBO) loss to estimate belief states directly for recurrent RL policies, while probabilistic inference for learning control (PILCO) uses dynamic models learned from beliefs rather than observations. Previous studies have focused on partial observability, but none have discussed state observations. Pajarinen et al. [15] proposed Compatible Policy Search (COPOS), a variant of trust region policy optimization (TRPO), which has a transformer network using Adam optimization and applies to machine translation tasks. Although the original TRPO and proximal policy optimization (PPO) were planned for an MDP with an infinite horizon, they have recently been expanded for a finite horizon [16].

In this study, we propose a novel architecture and interaction loop for SAC [17], which is one of the state-of-the-art methods on MDP, inspired by [73] to solve partially observable continuous control tasks more efficiently in much more exploration and interaction with the environment by using a guided policy search approach to optimize the policy.

### III. PRELIMINARIES

This section provides background information necessary to comprehend our guided soft actor–critic approach. We first briefly introduce partially observable Markov decision process (MDP) fundamentals and then present the classic soft actor–critic method.

### A. PARTIALLY OBSERVABLE MARKOV DECISION PROCESS

In many real-world problems, the agent cannot fully observe the state of the current environment, which is necessary for MDP-based learning methods. The Markov property for MDPs does not hold in real-world scenarios in which observations are incomplete or noisy, such as autonomous driving, where the agent can only observe parts of the environment and does not have a complete view of the world. Under uncertainty, the results of the actions depend more on the current state signal. POMDPs generalize MDPs to decision-making processes under partial observability.
POMDP captures the dynamics of many real-world environments better because it explicitly acknowledges that the sensations acquired by the agent are simply fragmentary glimpses of the underlying system state. [6] Formally, a POMDP is defined [74] by the tuple \((S, \mathcal{A}, O, T, O, R, \gamma)\), where \(S\) is the state space, \(\mathcal{A}\) is the action space, \(O\) is the observation space, \(T\) is the transition model, \(R\) is the reward function, \(O\) is the observation model, and \(\gamma\) is the discount factor, consisting of:

- a set of states \(S \in S\),
- a set of actions \(A \in \mathcal{A}\),
- a set of observations \(O \in O\) the agent can experience of its world,
- a transition function defined as \(T: S \times \mathcal{A} \times S \rightarrow [0,1]\), where the probability of ending up in the next state \(s'\) after action \(a\) in state \(s\) is denoted \(T(s,a,s')\),
- an observation function defined as \(O: S \times \mathcal{A} \times O \rightarrow [0,1]\), where \(O(s',a,o)\) is the probability of making observation \(o\) given that the agent took action \(a\) and landed in state \(s'\),
- \(r\), a reward function defined as \(r: S \times \mathcal{A} \rightarrow \mathbb{R}\),
- the initial state distribution \(\mu_0(s)\) specifying the probability of state \(s\) being the initial state of the decision process,
- the history \(h_t = (o_0,a_0,...,a_{t-1},o_t)\) as a vector containing all past observations and taken actions.

A POMDP is distinguished from a fully observable MDP by the agent’s perception of an observation \(o \in O\) rather than actual observation of the state of the environment. As a result, POMDPs have two more elements in their description than MDPs. The set of observations \(O\) represents all possible sensors that the agent can receive. The observation function \(O\) indicates which particular observation the agent receives depending on the next state \(s'\) and maybe also on its action \(a\) [70].

### B. Soft Actor Critic (SAC)

SAC, which is defined for RL tasks involving continuous actions, is an off-policy optimization algorithm for stochastic policies. Entropy regularization is a key element of the SAC. The policy is trained to maximize the trade-off between expected returns and entropy, which is a measure of the randomness of the policy [18]. This is strongly related to the exploration-exploitation trade-off. More exploration occurs when entropy increases. Thus, it has the potential to accelerate learning. It can also avoid the policy from prematurely convergent to a bad local optimum. The objective function of the SAC comprises both a reward term and an entropy term \(\mathcal{H}\) weighted by \(\alpha\)

\[
J(\pi) = \sum_{t=0}^{\infty} \mathbb{E}_{(s_t,a_t) \sim \rho_{\pi}}[r(s_t,a_t) + \alpha \mathcal{H}(\pi(\cdot | s_t))] 
\]  

SAC concurrently learns by using three networks: a state value function \(V\) parametrized by \(\psi\), a soft Q-function \(Q\) parameterized by \(\theta\), and a policy function \(\pi\) parametrized by \(\phi\) [17].

The Value network is trained to minimize the squared residual error [17]:

\[
J_V(\psi) = \mathbb{E}_{s_t \sim \mathcal{D}} \left[ \frac{1}{2} (V_\psi(s_t) - V_\pi(s_t, a_t) - \log \pi_\phi(a_t | s_t))^2 \right] 
\]  

where \(\mathcal{D}\) is the distribution of the replay buffer. The gradient of Eq. 2 can be estimated using an unbiased estimator [17]:

\[
\hat{\nabla}_\psi J_V(\psi) = \nabla_\psi V_\psi(s_t) \left( V_\psi(s_t) - Q_\theta(s_t, a_t) + \log \pi_\phi(a_t | s_t) \right) 
\]  

where the actions are sampled according to the current policy instead of the replay buffer.

The Q network is trained by minimizing the soft Bellman residual [17]:

\[
J_Q(\theta) = \mathbb{E}_{(s_t,a_t) \sim \mathcal{D}} \left[ \frac{1}{2} (Q_\theta(s_t, a_t) - \hat{Q}(s_t, a_t))^2 \right] 
\]  

with

\[
\hat{Q}(s_t, a_t) = r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim p} \left[ V_\psi(s_{t+1}) \right] 
\]  

which again is optimized with stochastic gradients

\[
\nabla_\theta J_Q(\theta) = V_\theta Q_\theta(s_t, a_t) \left( Q_\theta(s_t, a_t) - r(s_t, a_t) - \gamma V_\psi(s_{t+1}) \right) 
\]  

Finally, the policy parameters are learned by directly minimizing the expected KL-divergence [17]:

\[
J_\pi(\phi) = \mathbb{E}_{s_t \sim \mathcal{D}} \left[ D_{KL} \left( \pi_\phi(\cdot | s_t) \parallel \frac{\exp(q_\phi(s_t))}{Z_\theta(s_t)} \right) \right] 
\]  

the policy is reparameterized by using a neural network transformation

\[
a_t = f_\phi(\epsilon_t; s_t) 
\]  

where \(\epsilon\) is an input noise vector. The objective in Equation 7 can be rewritten as follows:

\[
J_\pi(\phi) = \mathbb{E}_{s_t \sim \mathcal{D}, \epsilon_t \sim N} \left[ \log \pi_\phi(f_\phi(\epsilon_t; s_t) | s_t) - Q_\theta(s_t, f_\phi(\epsilon_t; s_t)) \right] 
\]  

The gradient of Equation 9 can be approximated with

\[
\nabla_\phi J_\pi(\phi) = \nabla_\phi \log \pi_\phi(a_t | s_t) + \left( \nabla_{a_t} \log \pi_\phi(a_t | s_t) - \nabla_{s_t} Q_\theta(s_t, a_t) \right) \nabla_\phi f_\phi(\epsilon_t; s_t) 
\]  

where \(a_t\) is evaluated at \(f_\phi(\epsilon_t; s_t)\) [17].
IV. GUIDED DEEP REINFORCEMENT LEARNING APPROACH FOR PARTIALLY OBSERVABLE MARKOV DECISION PROCESS

Guided policy search (GPS) approaches [67]–[73] provide an idea for combining RL and supervised learning to solve POMDP problems by including previous shortened observation-action pairs into current state-action representations. Model-based approaches need to learn a dynamic model of the environment to reduce the amount of interaction required in the real world. However, this can be challenging for complex tasks and requires extensive computation. On the other hand, model-free approaches learn a policy directly from interactions with the environment and have much more flexibility because they do not need an exact representation of the environment. To achieve effective outcomes using model-free approaches, an adequate number of interactions with the environment must be provided, depending on the complexity of the task. For this reason, we focus on a model-free approach, as opposed to most traditional GPS methods. We present a new approach to solving continuous control POMDP problems by combining a model-free guided framework with SAC.

Our guided SAC, like that of [73], has two phases: training and testing. In the training phase, the guided-SAC architecture consists of two actors and a critic. To generate samples stored in replay buffer $\mathcal{D}$, two actors interact with the same environment iteratively. Figure 2 shows the guided-SAC architecture and the interaction loop of the guided SAC in the training phase. The final control actor chooses actions based on partial observations, while the guiding actor chooses actions by perceiving the full-state observations. The critic's job is to assess the policy and, as a result, update the action-value function, which is a measure of the total discounted reward, using data gathered through environmental observations. Targets for the Q functions are computed using

$$y(r,s',d) = r + \gamma(1 - d) \left( \min_{i=1,2} Q_{\phi_i}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}' | s') \right),$$

and Q-functions are updated by one step of gradient descent

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left( Q_{\phi_i}(s, a) - y(r,s',d) \right)^2 \text{ for } i = 1,2 \quad (12)$$

Then, the actors are fed the estimated discounted reward. Each policy gradient of the actors is updated by one step of gradient ascent using

$$\nabla_{\theta_i} \frac{1}{|B|} \sum_{s \in B} \left( \min_{i=1,2} Q_{\phi_i}(s, \tilde{a}_\theta(s)) - \alpha \log \pi_\theta(\tilde{a}_\theta(s) | s) \right) \quad (13)$$

$$\nabla_{\theta_i} \frac{1}{|B|} \sum_{s \in B} \left( \min_{i=1,2} Q_{\phi_i}(s, \tilde{a}_\theta(s)) - \alpha \log \pi_\theta(\tilde{a}_\theta(s) | s) \right) \quad (14)$$

where $\tilde{a}_\theta(s)$ is a sample from $\pi_\theta(\cdot | s)$ which is differentiable w.r.t $\theta$ via the reparametrization trick.

In turn, the actor chooses an action based on the current state and interacts directly with the environment using the estimated action-value function. For each update, the guiding policy is trained with the history of observation-action pairs along with the current state from $\mathcal{D}$ and the control actor updates its policy with only the history of observation-action pairs from $\mathcal{D}$. Both are trained to maximize the expected reward while also maximizing entropy like in the original concept of the SAC [17]. Through $\alpha$ known as the entropy temperature, the agent

**FIGURE 2.** Architecture and interaction loop of Guided-SAC in the training phase. The numbers indicate the order of steps.
selects random actions to explore the environment through \( \alpha \), known as the entropy temperature. The parameter is automatically adjusted with gradient descent to guarantee adequate exploration at the beginning of learning and, thereafter, a larger emphasis on maximizing the predicted reward. By decreasing the Kullback–Leibler divergence, the policy distribution is updated towards the softmax distribution for the current Q function for the policy improvement step. [75]

In our approach, to use samples generated from different policies, the distributions of the two policies should be similar. We use the adjustable entropy temperature \( \alpha \) as it improves the performance and stability of the method [75] and guarantees that they converge to the same policy.

\[
\alpha \leftarrow \alpha - \lambda \hat{v}_c(\alpha)
\]  

Update target networks with

\[
\phi_{\text{arg},i} \leftarrow \rho \phi_{\text{arg},i} + (1-\rho)\phi_i \text{ for } i = 1,2
\]

We also trained the DNN using the stored data in \( \mathcal{D} \) to teach the control policy actor to act as a guiding policy actor. Using this iterative interaction approach, the same trajectory is generated by both the guiding and control policies.

Algorithm 1 shows the Guided SAC approach. The numbers, which indicate the order of steps in Figure 2, are referenced in the Algorithm 1.

Algorithm 1: Guided SAC

```plaintext
1: Input: initial policy parameters \( \theta_\pi, \theta_\rho \), Q-function parameters \( \phi_i, \phi_1 \);
2: empty replay buffer \( \mathcal{D} \);
3: Set target parameters equal to main parameters \( \phi_{\text{arg},1} \leftarrow \phi_1, \phi_{\text{arg},2} \leftarrow \phi_2 \);
4: while \( \text{not convergence} \) do
5: Run the guiding policy network \( \pi_{\text{arg},i} (a_t | h_t, s_t) \) and the final control policy \( \pi_{\text{arg},1} (a_t | h_t) \) iteratively
6: Store in replay buffer \( \mathcal{D} = \{ (a_0^s, s_0^s, r_0^s, h_0^s, a_1^s, s_1^s, r_1^s, h_1^s, a_2^s, ...) \}
7: if \( s' \) is terminal, reset environment state.
8: for \( j \) in range(\text{however many updates}) do
9: Randomly sample a batch of transitions, \( B = \{ (s, a, r, s', d) \} \) from \( \mathcal{D} \)
10: Compute targets for the Q functions with eq. (11)
11: Update Q-functions by one step of gradient descent using (12)
12: Update policy by one step of gradient ascent using eq. (13), (14)
13: Adjust entropy temperature with eq. (15)
14: Update target networks with eq. (16)
15: end for
16: end if
17: until convergence
```

The final control actor interacts with the environment only during the testing phase when only observations are available. (Figure 3)
Based on the original tasks, have a fully observable state-space, we built a modification to simulate partial observability in real applications. The modification details of the Mujoco tasks for POMDP are given in Appendix A.

B. EXPERIMENTAL SETTINGS
To compare our results, the original implementations of SAC and TD3, as well as the OpenAI baselines, were used. We trained the algorithms for a total of 1 million time steps on Mujoco tasks. One core on Intel® Xeon® Gold 6148 Processor and 16 GB RAM has been used for the computation of each job. It took about 12 hours for each task. All trials were carried out with a total of ten random seeds in each. Running the mean policy without action noise for ten trajectories once per 10,000 steps was used to assess performance, and the average return over ten trajectories was reported. The results of our experiments were plotted as learning curves for the average cumulative reward over the last 10 episodes. Feedforward neural networks are used to represent policies and/or value functions in all evaluation techniques. The SAC and TD3 algorithms use two hidden neural network layers of size $(256, 256)$ with rectified linear units (ReLU) as the activation function [40], [17]. For the baseline algorithms, default hyperparameters from OpenAI SpinningUp [68] were used. The hyperparameters for our proposed algorithm were empirically set to those for SAC, and the network architectures of the guided SAC were selected to have the same number of parameters as the networks in SAC. (See Appendix A). The observation history has a fixed length of four, which means that the input for the RL algorithm includes the current observation as well as the four previous observations.

C. RESULTS
To compare our guided SAC approach with SAC and TD3, we ran all tasks for full observability and POMDP. Figure 5 shows the learning curves for the sampled tasks, with the first column representing fully observable performance and the second column representing POMDP findings. Tables 2 and 3 summarize the results.

Guided SAC performed best on the four POMDP Mujoco tasks. Guided SAC outperformed the state-of-the-art DRL baseline algorithm by more than $\sim 20\%$ on each task. TD3 could perform better than guided SAC on POMDP InvertedDoublePendulum-v2 Task. In general, guided SAC achieves significantly better performance on difficult POMDP tasks than the baselines.

### TABLE II
MAXIMUM AVERAGE RETURN ON FULLY OBSERVABLE

| Task                | SAC       | TD3       |
|---------------------|-----------|-----------|
| HalfCheetah-v2      | 12610.32±120.89 | 10050.08±127.51 |
| Hopper-v2           | 3650.64±3.4    | 3786.79±4.95    |
| InvertedDoublePendulum-v2 | 9359.85         | 9359.66         |
| Walker2d-v2         | 4470.69±78.31  | 5857.5±34.78    |
| Swimmer-v2          | 45.95±1.69    | 55.16±9.36     |

### TABLE III
MAXIMUM AVERAGE RETURN ON POMDP Mujoco TASKS

| Task                | SAC       | TD3       | Guided SAC | $\Delta$ |
|---------------------|-----------|-----------|------------|----------|
| HalfCheetah-v2      | 2508.82±148.46 | 3431.44±193.51 | 3658.26±123.59 | 7%       |
| Hopper-v2           | 1444.73±21    | 1586.35±460  | 1784.8±11.45  | 10%      |
| InvertedDoublePendulum-v2 | 6313.95±3763   | 9306±1.16    | 9120.46±688.6  | -2%      |
| Walker2d-v2         | 2269.17±431.78 | 897±172.52   | 2616.89±34.68  | 15%      |
| Swimmer-v2          | 32.19±5.3     | 34.42±2.15   | 41.46±1.83     | 20%      |
VI. CONCLUSION

To summarize our contribution, we propose a novel approach called guided SAC, which combines a model-free guided framework with SAC for POMDPs. We proposed a new agent environment interaction method for generating training samples in which the guiding and control actors interact with the same environment iteratively. The control actor selects actions based on partial observations, whereas the guiding actor can take action by perceiving full-state observations. We evaluated our approach using modified, partially observable MuJoCo tasks. The proposed guided SAC was compared to the state-of-the-art DRL baseline algorithms (SAC and TD3) on the POMDP versions of the sampled continuous control tasks in MuJoCo. We learned the optimal policy successfully and obtained good empirical results, showing that guided SAC outperforms the baselines on POMDPs. Our guided soft actor-critic approach outperformed the state-of-the-art DRL baseline algorithm, gaining 7~20% more maximum average return on four out of five partially observable tasks.

In future studies, we aim to apply our proposed algorithm to autonomous driving for partially observable environments, such as pedestrian collision avoidance problems.

APPENDIX

A. HYPERPARAMETERS

The implementation of the algorithms was based on OpenAI Spinningup [18]. Table IV details the hyperparameters used in this study.

| Hyperparameter          | TD3   | SAC   | Guided SAC |
|-------------------------|-------|-------|------------|
| discount factor:        | 0.99  |       |            |
| batch size:             | 100   |       |            |
| replay buffer size:     | 10^6  |       |            |
| random start step:      | 10000 |       |            |
| update after:           | 1000  |       |            |
| target NN update rate:  | 0.005 |       |            |
| optimizer:              | Adam  |       |            |
| actor learning rate:    | 10^-3 | -     | -          |
| critic learning rate:   | 10^-3 | -     | -          |
| actor NN structure:     | [256, 256] | [256, 256] | [256, 256] |
| critic NN structure:    |       |       |            |
| actor exploration noise:| 0.1   | -     | -          |
| target actor noise:     | 0.2   | -     | -          |
| target actor noise clip boundary: | 0.5 | - | - |
| policy update delay:    | 2     | -     | -          |
| entropy regulation coefficient: | 0.2 | - | 0.2 |

B. ENVIRONMENTS AND TASKS DETAILS

To create partially observable environments, we modified the MuJoCo tasks listed above by deactivating the parts of the observation. A number of dimensions of the observation vector were selected and overwritten with zeros. We chose dimensions whose deactivation had a noticeable impact on overall performance. (Table 5)
REFERENCES

[1] V. Mnih et al., “Human-level control through deep reinforcement learning,” Nature, vol. 518, no. 7540, pp. 529–33, Feb. 2015.

[2] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, “Trust Region Policy Optimization,” in International Conference on Machine Learning, 2015, pp. 1889–1897.

[3] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, et al., “Continuous control with deep reinforcement learning,” CoRR, vol. abs/1509.02971, 2015.

[4] K. Azizzadenesheli, A. Lazaric, and A. Anandkumar, “Experimental results: Reinforcement Learning of POMDPs using Spectral Methods,” in IMLR: Workshop and Conference Proceedings, 2016, no. vol.49, pp. 1–64.

[5] M. Egorov, “Deep reinforcement learning with POMDPs,” Stanford University, Rep., 2015. [Online]. Available: http://cs229.stanford.edu/proj2015/363_report.pdf

[6] M. Hausknecht and P. Stone, “Deep recurrent q-learning for partially observable MDPs,” AAAI Fall Symp. - Tech. Rep., vol. FS-15-06, pp. 29–37, 2015.

[7] J. Foerster, Y. Assael, N. de Freitas, and S. Whiteson, “Learning to communicate to solve riddles with deep distributed recurrent q-networks,” in IJCAI 2016 Deep Learning Workshop, 2016.

[8] P. Zhu, X. Li, and P. Poupart, “On improving deep reinforcement learning for POMDPs,” 2017. [Online]. Available: arXiv:1704.07978.

[9] P. H. Jin, S. Levine, and K. Keutzer, “Regret minimization for partially observable deep reinforcement learning,” 2017. [Online]. Available: arXiv:1710.11424, 2017.

[10] T. P. Le, N. A. Vien, and T. Chung, “A deep hierarchical reinforcement learning algorithm in partially observable Markov decision processes,” IEEE Access, vol. 6, pp. 49089–49102, 2018.

[11] N. Heess, J. J. Hunt, T. P. Lillicrap, and D. Silver, “Memory-based control with recurrent neural networks,” 2015. [Online]. Available: arXiv:1512.04455.

[12] C. Wang, J. Wang, X. Zhang, and X. Zhang, “Autonomous navigation of uav in large-scale unknown complex environment with deep reinforcement learning,” in Proc. IEEE Global Conf. Signal Inf. Process. (GlobalSIP), Nov. 2017, pp. 858–862.

[13] D. R. Song, C. Yang, C. McGreavy, and Z. Li, “Recurrent deterministic policy gradient method for bipedal locomotion on rough terrain challenge,” 2017. [Online]. Available: arXiv:1710.02896.

[14] M. Igl, L. Zintgraf, T. A. Le, F. Wood, and S. Whiteson, “Deep variational reinforcement learning for POMDPs,” 35th Int. Conf. Mach. Learn. ICML 2018, vol. 5, pp. 3359–3375, 2018.

[15] J. Pajarinen, H. L. Thai, R. Akhour, J. Peters, G. Neumann, “Compatible natural gradient policy search”, Machine Learning, 108(8), 1443-1466, 2019.

[16] K. Azizzadenesheli, M. K. Bera, and A. Anandkumar, “Trust Region Policy Optimization for POMDPs,” 2018. [Online]. Available: arXiv:1810.07900.

[17] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” in Proc. Int. Conf. Mach. Learn., vol. 80, 2018, pp. 1861–1870.

[18] OpenAI. Spinning Up. [Online]. Available: https://spinningup.openai.com

[19] E. Todorov, T. Erez, and Y. Tassa, “MuJoCo: A physics engine for modelbased control,” in Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., 2012, pp. 5026–5033.

[20] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.

[21] C. C. Aggarwal, Neural Networks and Deep Learning: A Textbook. Cham, Switzerland: Springer, 2018.

[22] B. R. Kiran et al., “Deep Reinforcement Learning for Autonomous Driving: A Survey,” in IEEE Transactions on Intelligent Transportation Systems, doi: 10.1109/TITS.2021.3054625.

[23] H. Van Hasselt, A. Guez, and D. Silver, “Deep Reinforcement Learning with Double Q-Learning,” in Proceedings of the 30th AAAI Conference on Artificial Intelligence, 2016, pp. 2094–2100.

[24] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, “Dueling network architectures for deep reinforcement learning,” 2015, arXiv:1511.06581. [Online]. Available: http://arxiv.org/abs/1511.06581

[25] J. Oh, V. Chockalingam, S. Singh, and H. Lee. (2016). Control of Memory, Active Perception, and Action in Minecraft. [Online]. Available: https://arxiv.org/abs/1605.09128

[26] H. Lee. Deep Reinforcement Learning with Forward Prediction, Memory, and Hierarchy. [Online]. Available: https://dl.berkeley.edu/deeprlcoursesp17/docs/honglak_lecture.pdf

[27] B. R. Kiran et al., “Deep Reinforcement Learning for Autonomous Navigation of uav in Large-Scale Unknown Complex Environment with Deep Reinforcement Learning,” in Proc. IEEE Global Conf. Signal Inf. Process. (GlobalSIP), Nov. 2017, pp. 858–862.

[28] M. Houseknecht and P. Stone, “Deep recurrent q-learning for partially observable MDPs,” AAAI Fall Symp. - Tech. Rep., vol. FS-15-06, pp. 29–37, 2015.

[29] J. Foerster, Y. Assael, N. de Freitas, and S. Whiteson, “Learning to communicate to solve riddles with deep distributed recurrent q-networks,” in IJCAI 2016 Deep Learning Workshop, 2016.

[30] P. Zhu, X. Li, and P. Poupart, “On improving deep reinforcement learning for POMDPs,” 2017. [Online]. Available: arXiv:1704.07978.

[31] P. H. Jin, S. Levine, and K. Keutzer, “Regret minimization for partially observable deep reinforcement learning,” 2017. [Online]. Available: arXiv:1710.11424, 2017.

[32] T. P. Le, N. A. Vien, and T. Chung, “A deep hierarchical reinforcement learning algorithm in partially observable Markov decision processes,” IEEE Access, vol. 6, pp. 49089–49102, 2018.

[33] N. Heess, J. J. Hunt, T. P. Lillicrap, and D. Silver, “Memory-based control with recurrent neural networks,” 2015. [Online]. Available: arXiv:1512.04455.

[34] C. Wang, J. Wang, X. Zhang, and X. Zhang, “Autonomous navigation of uav in large-scale unknown complex environment with deep reinforcement learning,” in Proc. IEEE Global Conf. Signal Inf. Process. (GlobalSIP), Nov. 2017, pp. 858–862.

[35] D. R. Song, C. Yang, C. McGreavy, and Z. Li, “Recurrent deterministic policy gradient method for bipedal locomotion on rough terrain challenge,” 2017. [Online]. Available: arXiv:1710.02896.

[36] M. Igl, L. Zintgraf, T. A. Le, F. Wood, and S. Whiteson, “Deep variational reinforcement learning for POMDPs,” 35th Int. Conf. Mach. Learn. ICML 2018, vol. 5, pp. 3359–3375, 2018.

[37] J. Pajarinen, H. L. Thai, R. Akhour, J. Peters, G. Neumann, “Compatible natural gradient policy search”, Machine Learning, 108(8), 1443-1466, 2019.

[38] K. Azizzadenesheli, M. K. Bera, and A. Anandkumar, “Trust Region Policy Optimization for POMDPs,” 2018. [Online]. Available: arXiv:1810.07900.

[39] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” in Proc. Int. Conf. Mach. Learn., vol. 80, 2018, pp. 1861–1870.

[40] OpenAI. Spinning Up. [Online]. Available: https://spinningup.openai.com

[41] E. Todorov, T. Erez, and Y. Tassa, “MuJoCo: A physics engine for modelbased control,” in Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., 2012, pp. 5026–5033.

[42] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.

[43] C. C. Aggarwal, Neural Networks and Deep Learning: A Textbook. Cham, Switzerland: Springer, 2018.

[44] B. R. Kiran et al., “Deep Reinforcement Learning for Autonomous Driving: A Survey,” in IEEE Transactions on Intelligent Transportation Systems, doi: 10.1109/TITS.2021.3054625.
This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/