Effective Skill Learning via Autonomous Goal Representation Learning

Constantinos G. Tsakonas[1], and Konstantinos I. Chatzilygeroudis[1]

***Abstract*—A long standing goal of robotics researchers is to develop robots that are able to develop in an autonomous open-ended manner through lifelong learning and interactions. If we are to see robots learning in an autonomous and open-ended manner, we need to develop methods for incremental and autonomous skill discovery and trial-and-error learning. In other words, we want our robots to be able to autonomously select their goals according to their current capabilities and learn controllers or policies to achieve those goals. In this paper, we take a step towards solving this challenge and propose a novel pipeline, called AGRL, that effectively combines deterministic simulations, Variational Auto-Encoders (VAEs) and Reinforcement Learning (RL) and enables robots to learn goal-conditioned policies suited to their capabilities. Our main intuition is that we can use effective exploration strategies in order to learn a good goal representation and distribution, and then use this distribution to generate effective and reachable goals for fast skill learning. We extensively evaluate the proposed method in simulation with a 7DOF manipulator and a differential drive mobile robot.**

I. Introduction

One of the goals from the first inception of Artificial Intelligence (AI) is to create agents that are able to learn autonomously in an open-ended manner [1]. This type of learning refers to the training of an agent which is decoupled from a specific objective or task. During this type of learning process, the agent is free to discover skills and/or objectives by itself without explicit external feedback or guidance. In this setting, AI agents also adapt and improve their performance over time by interacting with the environment. Agents are able to refine their skills and capabilities without relying on explicit instructions. We are yet to see systems with such capabilities.

Reinforcement Learning (RL) [2] is a field of artificial intelligence that focuses on training agents to make sequential decisions by interacting with an environment. RL is a tool that can be used for autonomous open-ended learning as it can allow agents to learn without explicit supervision. RL has seen a renowned interest lately and has provided effective solutions for robotic applications [3], [4]. Furthermore, RL's ability to handle sequential decision-making makes it well-suited for scenarios where tasks are interconnected or have a long-term dependency [5]. Agents can learn to break down complex problems into smaller subtasks and develop strategies to tackle them in a coherent manner. This hierarchical approach to learning enables RL agents to tackle increasingly difficult challenges, leading to continuous improvement and autonomous growth. One of the initial approaches to address this type of learning was by using intrinsic rewards [6]. Another approach is through Universal Value Function Approximators [7], which aims to develop versatile value functions to be utilized in a wide range of tasks rather than a specific objective. Nevertheless, we are yet to see an RL agent that can truly learn in an open-ended manner [1].

One approach to overcome the challenges of autonomous open-ended learning is information-theoretic skill learning, which is a product of the advancements in the field of RL and leverages principles from the information theory to develop an efficient and effective learning framework [8]. The conceptualization around this idea is to exploit unstructured raw data by employing properties from information theory along with unsupervised learning to create compact skill representations. However, these methods are strongly coupled with the exploration's strategy productivity to discover feasible and diverse skills.

In this work, we propose a novel pipeline for autonomous skill learning by exploiting Go-Explore as an exploration strategy in order to efficiently explore the environment and construct a dataset of states that can produce meaningful compact skill representations. This allows us to develop continuous skill representations by learning a goal distribution and sample new skills from it, in contrast with other approaches that discover a finite number of skills. We call this pipeline Autonomous Goal Representation Learning (AGRL). Finally, we learn these skills by training policies for two types of tasks, mobile robot navigation and controlling a 7-DOF robotic manipulator.

II. Related Work

A. Information-Theoretic Skill Learning

Derived from information theory and taking advantage of mutual information and Shannon entropy, information-theoretic skill discovery tries to find and optimal policy that maximizes the mutual information between skills and states. There have been many attempts to utilize information-theoretic skill learning. Skew-Fit [9], introduces a self-supervised objective and a policy learning objective for unsupervised RL. The self-supervised objective involves predicting state visitation order, and encouraging exploration. The policy learning objective maximizes expected cumulative rewards. The method alternates between training these objectives, promoting state-covering exploration during the training of the RL algorithm,

[1]Computational Intelligence Laboratory (CILab), Department of Mathematics, University of Patras, GR-26110 Patras, Greece, tsakonas_k@upnet.gr, costashatz@upatras.gr

979-8-3503-1806-7/23/$31.00 ©2023 IEEE

and improving sample efficiency and generalization. Experimental results demonstrate interesting performance compared to existing exploration methods. Moreover, curiosity-driven exploration [10], where a robot learns to predict the consequences of its actions and explores states that are difficult to predict, is based upon the information-theoretic framework. While the approach demonstrates promising results, it has limitations in scenarios where the prediction model fails to capture complex dynamics, leading to suboptimal exploration. DIAYN [11], is a proposed methodology for learning skills without any explicit reward function by maximizing an information-theoretic objective based on mutual information between states and actions. While it achieves skill diversity, it heavily relies on random exploration and may struggle to find optimal solutions efficiently due to its inherent exploration-exploitation trade-off. In general, the most decisive part of the successful utilization of information-theoretic skill learning comes down to the quality of the exploration.

The *Explore, Discover, Learn* (EDL) [8] framework combines unsupervised exploration and hierarchical clustering to identify distinct and reusable skills that enable effective behavior in a given environment while disentangling the process in three phases, exploration, skill discovery and skill learning. By leveraging variational inference it achieves to discover a diverse set of skills that cover different states of the environment. The discovered skills are then clustered using VQ-VAE model, allowing for the creation of a skill repertoire, where each skill sets the agent capable to reach a region in the operating environment. Experimental results demonstrate that the proposed approach leads to improved performance and enhanced exploration capabilities compared to prior methods in a range of challenging tasks. Inspired by the idea of separating each phase and treating it as an independent component, we want to explore more effective exploring strategies and create policies that can reach every possible state of the environment.

B. State-Space Exploration

Exploration plays a vital role in sequential decision-making tasks, including reinforcement learning (RL) and count-based methods. Count-based methods use state visitation counts to guide exploration, favoring less-visited states [12]. On the other hand, intrinsic motivation methods, such as information-theoretic approaches, encourage agents to seek novel and informative experiences [11]. However, count-based methods can struggle with sparse rewards [13], and designing effective intrinsic reward functions can be challenging [14]. Balancing exploration and exploitation is a difficult task in RL and remains an ongoing interest of research.

Quality-Diversity (QD) algorithms [15], [16] are evolutionary algorithms that attempt to illuminate the whole search space. Indeed, instead of searching for a unique global optimum while trying to devise some intelligent exploration policy, QD algorithms' goal is to provide a holistic view of how high-performing solutions are distributed throughout the search space. They do this by keeping an archive of previous good solutions. QD algorithms have had great success in evolutionary robotics and have been used to generate effective robot controllers in numerous studies [17]–[20].

Go-Explore [21], is a QD algorithm that addresses the challenge of efficiently exploring large state spaces with sparse rewards. It exploits the deterministic nature of simulators combined with a memory-based approach to overcome limitations in traditional exploration methods [6], [21]. By maintaining an archive of promising states and periodically resetting the agent to these states, Go-Explore enables more efficient exploration. It has achieved impressive performance in complex environments, such as video games and robotics applications, by effectively leveraging deterministic simulators and memory-based exploration. Go-Explore offers a novel solution to the exploration problem, facilitating the discovery of effective behaviors in challenging domains. Go-Explore can be employed to construct a strong goal distribution over the environment, hence providing better goal representation in the skill discovery phase.

III. Background

A. Reinforcement Learning

At its core, RL involves an agent that interacts with an environment over a sequence of discrete time steps. The agent receives observations from the environment, which include information about the current state, and takes actions that affect the environment. The environment provides feedback to the agent in the form of a reward signal, which indicates how well the agent is doing.

To formalize the RL problem, we use the framework of Markov Decision Processes (MDPs). An MDP is defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where:

- $\mathcal{S}$ is the set of possible states in the environment; it can also be a continuous space.
- $\mathcal{A}$ is the set of possible actions the agent can take; it can also be a continuous space.
- $\mathcal{P}$ is the state transition probability function, which gives the probability of transitioning to a new state $\boldsymbol{s}'$ when the agent takes action $\boldsymbol{a}$ in state $\boldsymbol{s}$.
- $\mathcal{R}$ is the reward function, which specifies the immediate reward the agent receives when it takes action $\boldsymbol{a}$ in state $\boldsymbol{s}$.
- γ is the discount factor that determines the importance of future rewards compared to immediate rewards.

At each time step t, the agent observes the current state $\boldsymbol{s}_t$, selects an action $\boldsymbol{a}_t$ based on its policy π, and receives a reward r_t from the environment. The state then transitions to a new state $\boldsymbol{s}_{t+1}$ according to the state transition probability function $\mathcal{P}(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$.

The goal of RL is to find an optimal policy π^* that maximizes the expected cumulative reward. The cumulative reward is also known as the return. The return at time step t is defined as the sum of discounted future rewards:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}. \tag{1}$$

In many practical applications, the state and action spaces are continuous. This poses a challenge for RL algorithms, which typically operate in discrete state and action spaces. To address this challenge, RL algorithms often use function approximators, such as neural networks, to represent the value function or policy.

The value function is a function that estimates the expected cumulative reward of being in a given state and following a given policy. The value function can be defined as follows:

$$V^{\pi}(\boldsymbol{s}) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid \boldsymbol{s}_t = \boldsymbol{s}\right], \\ = \mathbb{E}_{\pi}\left[G_t \mid \boldsymbol{s}_t = \boldsymbol{s}\right]. \tag{2}$$

The action-value function, or Q-function, is a function that estimates the expected cumulative reward of taking a given action in a given state and following a given policy. The action-value function can be defined as follows:

$$Q^{\pi}(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid \boldsymbol{s}_t = \boldsymbol{s}, \boldsymbol{a}_t = \boldsymbol{a}\right], \\ = \mathbb{E}_{\pi}\left[G_t \mid \boldsymbol{s}_t = \boldsymbol{s}, \boldsymbol{a}_t = \boldsymbol{a}\right]. \tag{3}$$

In the general case, the policy π is parameterized by parameters $\boldsymbol{\theta}$, and $\pi(\boldsymbol{a}|\boldsymbol{s}, t, \boldsymbol{\theta})$ outputs a distribution (e.g., a Gaussian) that is sampled in order to get the action to apply; i.e., we have *stochastic policies*. Most algorithms utilize policies that are not time-dependent (i.e., they drop t), but we include it here for completeness. Several algorithms use *deterministic policies*; a deterministic policy means that $\pi(\boldsymbol{a}|\boldsymbol{s}, t, \boldsymbol{\theta}) \Rightarrow \boldsymbol{a} = \pi(\boldsymbol{s}, t|\boldsymbol{\theta})$.

There are different approaches to solving RL problems, but one commonly used method is the actor-critic algorithm. The actor-critic algorithm is a model-free RL algorithm that combines the strengths of policy-based and value-based methods. The algorithm consists of two main components: an actor network that learns a policy, and a critic network that learns the value function.

The actor-critic algorithm updates the policy and value function parameters based on the observed rewards and states. The actor network updates the policy by following the gradient of the expected cumulative reward with respect to the policy parameters, while the critic network updates the value function by minimizing the temporal difference error between the estimated value and the observed reward.

B. Go-Explore

The core idea behind Go-Explore is to divide the learning process into two main phases: the *exploration phase* and the *exploitation phase*. During the exploration phase, the agent aims to gather as much information as possible about the environment by exploring diverse regions. It uses a random or stochastic policy to sample actions and record promising states and trajectories in a memory-based archive.

Once the exploration phase is complete, the exploitation phase begins. In this phase, the agent leverages the collected knowledge from the archive to focus on the most promising regions of the environment. It uses a more deterministic policy to exploit the learned information and maximize the cumulative reward.

In this work, we use only the exploration part of the Go-Explore algorithm that can be summarized in Algo. 1.

Input: Environment E, Archive A
Initialize A with empty archive;
while *not done* **do**
  Exploration Phase: for i *from 1 to* N **do**
    Initialize environment E;
    Initialize state $\boldsymbol{s}_0$;
    while *not done* **do**
      Sample action $\boldsymbol{a} \sim \pi(\boldsymbol{s})$;
      Execute action $\boldsymbol{a}$, observe next state $\boldsymbol{s}'$ and reward r;
      Add $(\boldsymbol{s}, \boldsymbol{a})$ to archive A;
      Update state $\boldsymbol{s} = \boldsymbol{s}'$;
    end
  end
end

Algorithm 1: Go-Explore: Exploration Phase

The Go-Explore algorithm has shown promising results in challenging RL environments, including complex video games and robotic tasks [21], [22]. By combining effective exploration with knowledge reuse, it provides a powerful approach for discovering optimal solutions in large state spaces with sparse rewards.

IV. Autonomous Skill Discovery and Learning

Following the path of the EDL method [8], we split the overall process of autonomous skill discovery and learning into three distinct sub-components: (a) state-space exploration, (b) skill discovery, and (c) skill learning (Fig. 1). Our intuition lies in the fact that each component is mostly disentangled from the others and can be tackled individually.

In AGRL, we tackle the:

- **state-space exploration** part using the Go-Explore algorithm;
- **skill discovery** part using Variational Auto-Encoders with continuous latent space;
- **skill learning** part using an off-policy goal-conditioned RL algorithm.

In the following subsections, we detail our choices and interconnections.

A. Exploration with Go-Explore

In the state-space exploration part, we are interested in finding a set of state vectors that describe as accurately as possible the capabilities of the robot at hand. In essence, we would like to find all the states that the robot can realistically be at given the nature of its physical and control characteristics as well as the characteristics of the environment. For this task, we turn to QD algorithms as they have been successfully used to find both diverse and high-performing behaviors even for complex continuous tasks [16], [17], [20]–[22]. In particular,

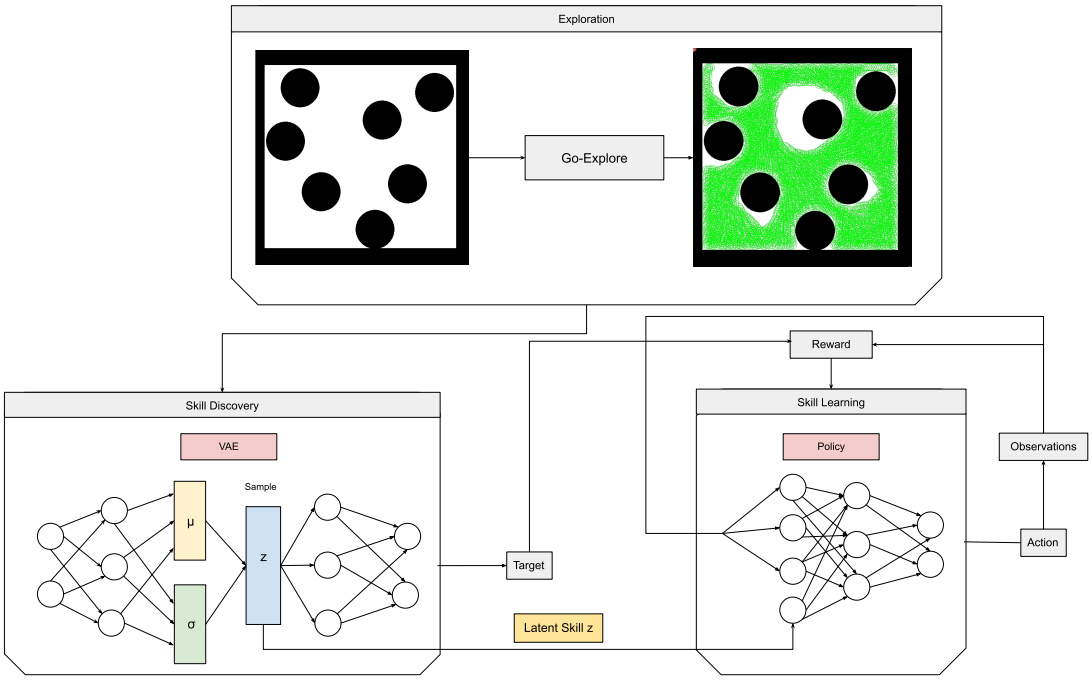

Fig. 1. Autonomous skill learning pipeline overview.

we will use the Go-Explore algorithm as it possesses a few key features that we can exploit in our setup:

- it exploits effectively deterministic simulators;
- it does not require a policy parameterization that can affect/bias the exploration process;
- it can scale to high-dimensional state spaces (usually the case in robotics applications);
- it keeps an archive of different states visited as opposed to policies which is usually done in QD;
- it is effective in rapidly exploring state-spaces even when no reward signal is available.

In this work, we use our own custom C++ implementation, where we have implemented effective parallel batch computations to speed up the exploration process.

B. Skill Discovery

We denote skills as a continuous vector $\boldsymbol{z} \in \mathbb{R}^{N_{\text{latent}}}$, and the skill space as $\mathcal{Z}$: $\forall \boldsymbol{z}, \boldsymbol{z} \in \mathcal{Z}$. In the skill discovery part of AGRL, we are interested in solving two tasks:

1) making the skill space $\mathcal{Z}$ a compact representation of the state space $\mathcal{S}$;
2) learning a skill space distribution that we can use to generate targets/goals in the skill learning phase.

Throughout the paper we refer to *skill space* and *goal space* interchangeably.

For the first task, we wish to map every point of state space into a latent skill $\boldsymbol{z}$. To achieve that we need to calculate the posterior $p(\boldsymbol{z}|\boldsymbol{s})$. Thus, based on Bayes' theorem, we need to compute

$$p(\boldsymbol{z}|\boldsymbol{s}) = \frac{p(\boldsymbol{s}|\boldsymbol{z})p(\boldsymbol{z})}{\int p(\boldsymbol{s}|\boldsymbol{z})p(\boldsymbol{z})d\boldsymbol{z}} = \frac{p(\boldsymbol{s},\boldsymbol{z})}{\int p(\boldsymbol{s}|\boldsymbol{z})p(\boldsymbol{z})d\boldsymbol{z}} \tag{4}$$

where $p(\boldsymbol{s},\boldsymbol{z})$ is the probability distribution over the states and latent skills, and the denominator is the evidence. However, computing the above in the general case is intractable. Hence, we turn our attention to variational inference to approximate the posterior, and more specifically to Variational Autoencoders (VAE) neural networks [23].

The VAE topology consists of two parts: the encoder and the decoder network. The encoder network parameterizes a probabilistic mapping from the input data $\boldsymbol{s}$ to a latent space $\boldsymbol{z}$, where $\boldsymbol{z} \sim p(\boldsymbol{z}|\boldsymbol{s})$. The encoder produces the parameters of the approximate posterior distribution $p(\boldsymbol{z}|\boldsymbol{s})$, typically in the form of mean $\boldsymbol{\mu}_z$ and variance $\boldsymbol{\sigma}_z^2$. We denote $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{s}) \approx p(\boldsymbol{z}|\boldsymbol{s})$, where $\boldsymbol{\phi}$ are the parameters of the encoder network. The decoder network, parameterized as $q_{\boldsymbol{\psi}}(\boldsymbol{s}|\boldsymbol{z}) \approx p(\boldsymbol{s}|\boldsymbol{z})$, reconstructs the original data $\boldsymbol{s}$ given samples from the latent space $\boldsymbol{z}$ ($\boldsymbol{\psi}$ are the parameters of the decoder network). Similar to the encoder, the decoder is probabilistic and generates mean $\boldsymbol{\mu}_s$ and variance $\boldsymbol{\sigma}_s^2$ for each output variable. During training, the VAE aims to maximize the evidence lower bound (ELBO), which is a variational approximation of the log-likelihood of the data. The ELBO is given by:

$$\mathcal{L}_{\text{VAE}}(\boldsymbol{\phi},\boldsymbol{\psi}) = \mathbb{E}\left[\log q_{\boldsymbol{\psi}}(\boldsymbol{s}|\boldsymbol{z})\right] - \text{KL}\left[q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{s})||p(\boldsymbol{z})\right] \tag{5}$$

where KL denotes the Kullback-Leibler divergence and $p(\boldsymbol{z})$ is the prior distribution over the latent space[1]. The first term encourages faithful reconstruction, while the second term regularizes the latent space to follow the prior distribution. In practice, we train the VAE networks using the data from

[1]Which we assume to be $\mathcal{N}(0,1)$.

the exploration step using the Adam optimizer [24] and we compute Eq. 5 in batches.

When the training of the VAE networks is complete, we have in our possession approximations of the distributions $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{s}) \approx p(\boldsymbol{z}|\boldsymbol{s})$ and $q_{\boldsymbol{\psi}}(\boldsymbol{s}|\boldsymbol{z}) \approx p(\boldsymbol{s}|\boldsymbol{z})$. Practically, we can use the encoder network to find the latent space representation of a specific state, while we can use the decoder network to reconstruct the full state from a specific latent vector.

C. Skill Learning

One interesting fact that we exploit in AGRL is that we can view the process of sampling a latent vector $\boldsymbol{z}'$ from the prior distribution $p(\boldsymbol{z})$ of the latent space as a way of sampling a novel goal or skill for our algorithm to learn. This means that we can use the prior distribution of the latent space and the decoder network to create a goal sampling distribution that will effectively generate goals/skills in a continuous spectrum.

As such at this final stage, the agent learns a goal-conditioned policy so that it can perform any skill generated by the sampling procedure described just above. To train the policy, we utilize the Twin Delayed DDPG (TD3) algorithm [25]. TD3 is an off-policy actor-critic RL algorithm that is suitable for continuous control tasks, and is directly inspired by DDPG [3]. In comparison with DDPG, TD3 uses two Q-functions and chooses the smallest Q-value in order to resolve the Q-value overestimation. Moreover, the policy networks are updated less frequently than the Q-functions, and noise is added to the target network's output to achieve generalization during training.

One of the most important parts of RL algorithms is the reward function. In this paper, we experiment with two different reward functions. At the beginning of each episode we sample a new skill/goal that we want to learn $\boldsymbol{z}' \sim p(\boldsymbol{z})$. The first reward function exploits the full decoder network and we compute:

$$r_{\log}(\boldsymbol{s}, \boldsymbol{z}') = \log q_{\psi}(\boldsymbol{s}|\boldsymbol{z}'), \boldsymbol{z}' \sim p(\boldsymbol{z}) \tag{6}$$

In the second reward function, we pass the sampled latent vector $\boldsymbol{z}'$ through the decoder network and we only keep the mean prediction, $\boldsymbol{s}' = \boldsymbol{\mu}_s(\boldsymbol{z}')$. We now define a reward function that computes the mean squared error (MSE) between the desired target and the current state:

$$r_{\text{mse}}(\boldsymbol{s}, \boldsymbol{z}') = -\|\boldsymbol{s} - \boldsymbol{\mu}_s(\boldsymbol{z}')\|^2, \boldsymbol{z}' \sim p(\boldsymbol{z}) \tag{7}$$

Finally, since our trained policies are conditioned on a skill/goal, the $\boldsymbol{z}'$ is fixed over the duration of an episode, and it is a part of the policy's inputs along with the agent's state.

V. EXPERIMENTS

With the experiments in this section, we aim at answering the following questions:

1) Is using Go-Explore more effective than uniform sampling for learning an effective goal distribution?
2) Do we benefit from using a continuous latent space versus a discrete one which is usually used in the context of skill discover?
3) How does the choice of the reward function affect the skill learning performance?

In order to answer the above questions, we devise two

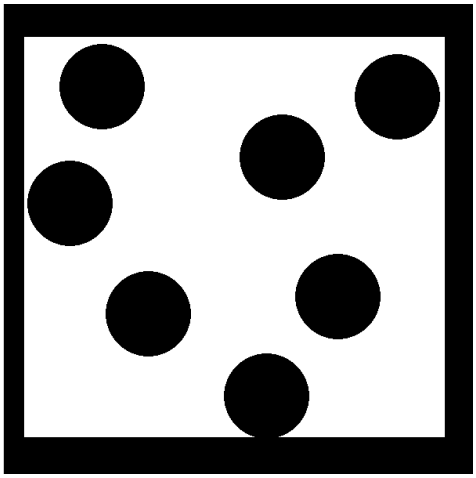

Fig. 2. The mobile robot environment. Black regions are obstacles.

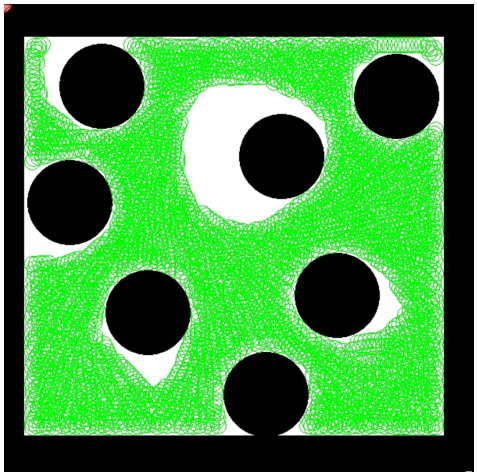

Fig. 3. Go-Explore on mobile robot's environment. The green circles showcase the states that Go-Explore has explored.

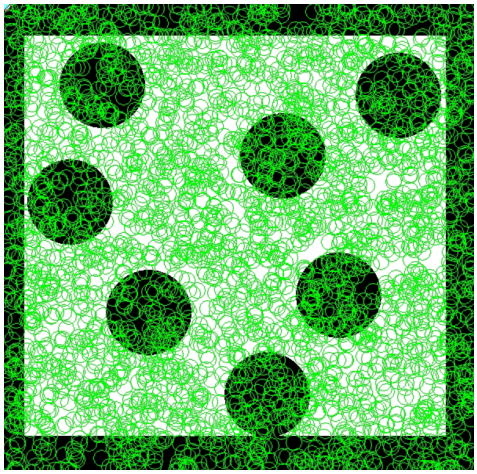

Fig. 4. Uniform sampling on mobile robot's environment. The green circles correspond to uniformly random sampled points.

learning scenarios:

1) A 7DoF manipulator joint-space reaching task. Here the main idea is to devise an experiment that highlights the importance of the initial exploration strategy. For this reason, we create an environment where the end-effector of the manipulator is constrained to move inside a "capsule" (Fig. 5).
2) A differential drive mobile robot experiment, where the mobile robot needs to navigate a map with obstracles and be able to reach any desired goal state.

1) Baselines and Ablations: In order to answer to the above questions, we devise some baselines and ablations of AGRL. In particular, in the manipulator experiment we compare the following setups:

- **AGRL**: our proposed method that consists of first exploring with Go-Explore, learning a continuous latent/skill space and learning a goal-conditioned policy;
- **AGRL**$^{-}$: this is an ablation of our proposed method that replaces the Go-Explore part with uniform random exploration. In essence, given the fact that we have a deterministic simulator and we know the kinematics of our robots and objects, we can sample inside the valid state space a huge amount of data;
- **Joint-Space Goal-Conditioned RL**: this is the traditional approach where we would directly learn a goal-conditioned policy and at each episode we choose a target/goal randomly from the joint space. We use this as a baseline.

It is important to note that neither Go-Explore nor the uniform random process is aware of the constrained nature of the environment. However, Go-Explore interacts with the actual environment while the uniform random process does not interact with the environment.

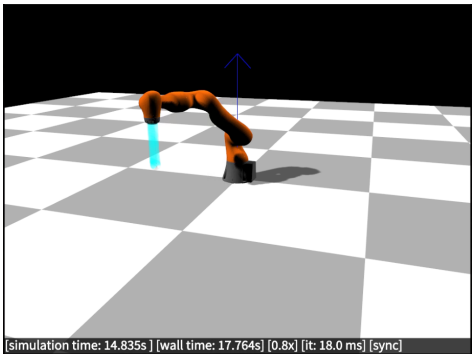

Fig. 5. Go-Explore on robotic arm's environment.

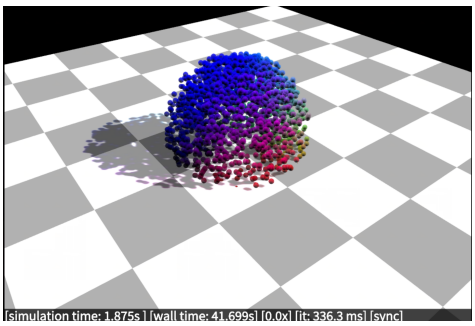

Fig. 6. Uniform sampling on robotic arm's environment.

In the mobile robot experiment we compare the following setups:

- **AGRL**: our proposed method that consists of first exploring with Go-Explore, learning a continuous latent/skill space and learning a goal-conditioned policy;
- **AGRL**$^{\text{discrete}}$: this is an ablation of our proposed method that learns a discrete latent/skill space.

To create a discrete number of skills for the skill-conditioned policy, we deploy the VQ-VAE architecture during the skill discovery phase [26]. VQ-VAE consists of a lookup table, named codebook, and each place in this table contains a vector $\boldsymbol{c}$. Each $\boldsymbol{c}$ represents the center of a cluster, and during the training of the model, skills are assigned to these clusters. By the end of the VQ-VAE's training, we retrieve a discrete number of skills equal to the number of clusters, and the latent representation of each skill is the $\boldsymbol{c}$. To simplify the above procedure, VQ-VAE basically creates skills that can reach a unique region of the environment.

2) Environment Details: For the navigation task, the agent is placed in an environment containing multiple objects, which cover a big portion of the environment. The objective is for the agent to reach any location of the environment using the learned policy without sticking to the obstacles. The environment is depicted in Fig. 2. For the robotic manipulator, we introduce an artificial constraint and we allow the end effector of the robotic manipulator to move only inside a "capsule" along the z-axis (Fig. 5). Therefore, the manipulator needs to learn a policy that can reach any joint configuration while the end effector moves only inside this capsule. In the manipulator task, the skill space dimensions are 1D and the state space is 7D, while in the mobile robot the skill space is 2D and the state space is 3D.

Regarding the navigation task, we use the fastsim simulator [27], whereas, for the 7DOF robotic arm we use the DART simulator [28] (we use the robot_dart wrapper). To provide some intuition about the exploration of the environment, we provide an overview of the states sampled for the two tasks using Go-Explore exploration and random sampling over the state space. In Fig. 3 and Fig. 4 are the results of the exploration concerning the navigation task, and in Fig. 5 and Fig. 6 the results of the robotic arm environment. In both tasks, we evaluate the policies in 200 unseen valid targets and report the mean value of the sum of the mean squared errors per episode. A video showcasing the learned policies can be found at the following url: https://youtu.be/x-j5mid6jxM.

A. Exploration strategy's impact

The results showcase that AGRL finds an effective goal distribution that fosters effective and rapid learning (Fig. 7). The variation of our method that uses uniform sampling for the exploration part, fails to identify the underlying structure of the reachable/interesting states of the environment and the learning

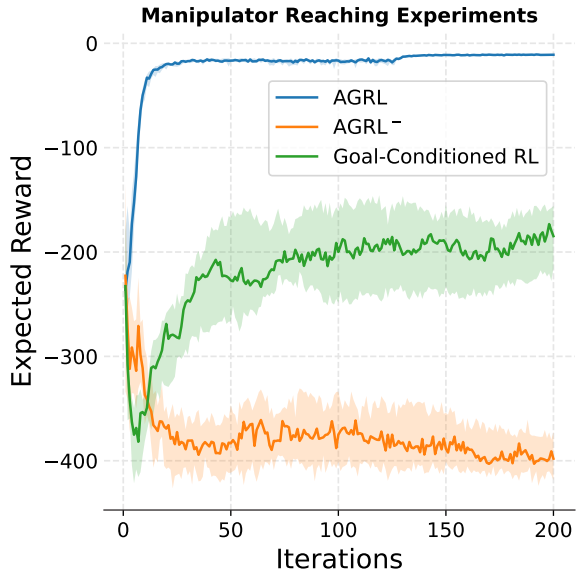

Fig. 7. Manipulator reaching experiments: comparison of different exploration strategies. Creating the latent space with data generated from Go-Explore provides a superior goal distribution that fosters effective and rapid learning. On the contrary, doing a simple uniform random sampling can create a sub-optimal goal distribution. We include a baseline of using the full joint-space as a goal and learning by uniformly sampling over this space. Solid lines are the median over 20 replicates and the shaded regions are the regions between the 25-th and 75-th percentiles.

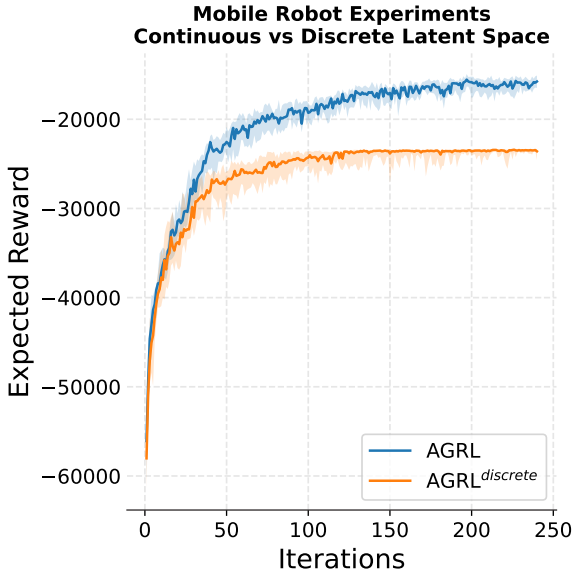

Fig. 8. Mobile robot task: comparison of continuous vs discrete latent skill space. Solid lines are the median over 20 replicates and the shaded regions are the regions between the 25-th and 75-th percentiles.

diverges. The classical goal-conditioned RL is learning but at a much slower rate than AGRL.

B. Goal-Conditioned vs Skill-Conditioned Policy

The results showcase that while learning a discrete latent space gives us the ability to reach the center of unique regions of the environment, it is difficult to reach any target accurately (Fig. 8). On the contrary, AGRL using a continuous latent skill representation is able to learn a goal conditioned policy that can reach any target.

C. Reward Functions

Since skill-conditioned policies are the most common policies that are created when referring to skill learning, it is frequently assumed due to their discrete nature, that the standard deviation of the decoder is fixed. Hence, under this assumption the Eq. 6 is equivalent to the mean squared error (aka, Eq. 7). On the contrary, we implement a goal-conditioned policy, and we are able to acquire both $\boldsymbol{\mu}_s$ and $\boldsymbol{\sigma}_s^2$ as the decoder's output, so we can construct a distribution and use Eq. 6. Thus, in order to compare how effective maximizing the log probability of the agent's state is in contrast to minimizing the distance between the target and agent, we learn two policies, one using a distance-based reward and one using the log probability-based reward. We observe that by using both rewards the policy converges, and even though the MSE reward performs better in this task, maximizing the log probability could seem useful in different scenarios (Fig. 9).

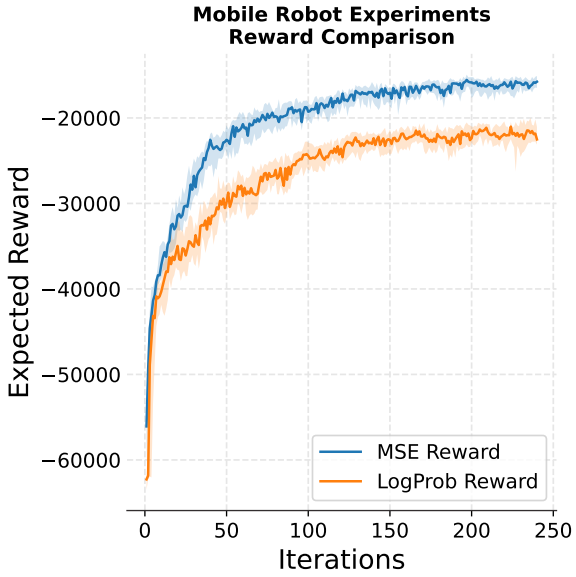

Fig. 9. Mobile robot task: comparison of different reward functions. Solid lines are the median over 20 replicates and the shaded regions are the regions between the 25-th and 75-th percentiles.

D. Verdict

Overall, our experiments showcase two important outcomes:

1) In order to learn an effective goal distribution for skill learning, it is important to create a dataset that contains states that are reachable by the robot and useful for the task at hand. We advocate in this paper that QD algorithms are an effective way of doing so.
2) It is possible to learn a continuous latent/skill space and use it in an autonomous skill discovery and learning pipeline. A crucial part for making this possible is that the learned latent distribution serves as the *goal generator* for the skill learning part.

It is important to note that in AGRL we do not use any explicit prior information about the task or constraints of the

environment. This is autonomously extracted with environment interaction both in the exploration phase as well as in the skill learning phase. Overall, AGRL makes a step forward towards an effective autonomous skill discovery and learning pipeline.

VI. CONCLUSION

In this work, we proposed the AGRL framework, an autonomous skill-discovery pipeline, which extends the EDL [8] pipeline. Our approach consists of three phases: state-space exploration, skill discovery, and skill learning. We proved that a robust exploration strategy plays a crucial role on the quality of the final learned policy. Regarding skill discovery and skill learning, we mainly focus on goal-conditioned policies by treating each possible state as a goal rather than learning discrete behaviors. We evaluated our proposed methodology in two types of tasks: a mobile robot navigation task in an environment containing obstacles and a 7DOF robotic manipulator reaching task. Additionally, using a different skill discovery approach, we show the limitations of skill-conditioned policies introduced in such tasks. Finally, we explore the impact a probabilistic reward function has on the learning procedure rather than a distance-based reward function.

Despite the success AGRL has, it comes with some limitations. Our approach is bounded to the exploration performance which affects the following stages since the quality of the exploration determines the quality of the latent representation of the goals. However, even if the quality of the exploration is sufficient, the training of the VAE model could be suboptimal, and the possible causes are extreme dimensionality reduction or the entanglement of latent space. This will result in less representative goal representation and will degrade learning.

To advance our work on AGRL, we want to extend this autonomous learning approach, including the goal representation, to exploit observations from sensors rather than the agent's state directly. Sensor data require different handling, however, and if implemented effectively, it can be an important step towards implementing AGRL in a real world environment.

ACKNOWLEDGMENTS

This work was supported by the Hellenic Foundation for Research and Innovation (H.F.R.I.) under the "3rd Call for H.F.R.I. Research Projects to support Post-Doctoral Researchers" (Project Acronym: NOSALRO, Project Number: 7541).

REFERENCES

[1] S. Doncieux, N. Bredeche, L. L. Goff, B. Girard, A. Coninx, O. Sigaud, M. Khamassi, N. Díaz-Rodríguez, D. Filliat, T. Hospedales *et al.*, "DREAM architecture: a developmental approach to open-ended learning in robotics," *arXiv preprint arXiv:2005.06223*, 2020.

[2] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998.

[3] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *International Conference on Learning Representations (ICLR)*, 2016.

[4] K. Chatzilygeroudis, V. Vassiliades, F. Stulp, S. Calinon, and J.-B. Mouret, "A survey on policy search algorithms for learning robot controllers in a handful of trials," *IEEE Transactions on Robotics*, vol. 36, no. 2, pp. 328–347, 2019.

[5] K. Tsinganos, K. Chatzilygeroudis, D. Hadjivelichkov, T. Komninos, E. Dermatas, and D. Kanoulas, "Behavior policy learning: Learning multi-stage tasks via solution sketches and model-based controllers," *Frontiers in Robotics and AI*, vol. 9, 2022.

[6] P.-Y. Oudeyer, F. Kaplan, and V. V. Hafner, "Intrinsic motivation systems for autonomous mental development," *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 2, pp. 265–286, Apr. 2007.

[7] T. Schaul, D. Horgan, K. Gregor, and D. Silver, "Universal value function approximators," in *International Conference on Machine Learning (ICML)*, 2015.

[8] V. Campos, A. Trott, C. Xiong, R. Socher, X. Giró-i-Nieto, and J. Torres, "Explore, discover and learn: Unsupervised discovery of state-covering skills," in *International Conference on Machine Learning (ICML)*, 2020.

[9] V. H. Pong, M. Dalal, S. Lin, A. Nair, S. Bahl, and S. Levine, "Skew-fit: State-covering self-supervised reinforcement learning," in *International Conference on Machine Learning (ICML)*, 2020.

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

[11] B. Eysenbach, A. Gupta, J. Ibarz, and S. Levine, "Diversity is all you need: Learning skills without a reward function," in *International Conference on Learning Representations (ICLR)*, 2019.

[12] M. G. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, "Unifying count-based exploration and intrinsic motivation," in *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[13] G. Ostrovski, M. G. Bellemare, A. van den Oord, and R. Munos, "Count-based exploration with neural density models," in *International Conference on Machine Learning (ICML)*, 2017.

[14] R. Houthooft, R. Y. Chen, P. Isola, B. C. Stadie, F. Wolski, J. Ho, and P. Abbeel, "Evolved policy gradients," in *Conference on Neural Information Processing Systems (NeurIPS)*, 2018.

[15] A. Cully and Y. Demiris, "Quality and Diversity Optimization: A Unifying Modular Framework," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 2, pp. 245–259, 4 2018.

[16] K. Chatzilygeroudis, A. Cully, V. Vassiliades, and J.-B. Mouret, "Quality-diversity optimization: a novel branch of stochastic optimization," in *Black Box Optimization, Machine Learning, and No-Free Lunch Theorems*. Springer, 2021, pp. 109–135.

[17] A. Cully, J. Clune, D. Tarapore, and J.-B. Mouret, "Robots that can adapt like animals," *Nature*, vol. 521, no. 7553, pp. 503–507, 2015.

[18] V. Vassiliades, K. Chatzilygeroudis, and J.-B. Mouret, "Using centroidal Voronoi tessellations to scale up the multi-dimensional archive of phenotypic elites algorithm," *IEEE Transactions on Evolutionary Computation*, 2017.

[19] K. Chatzilygeroudis, V. Vassiliades, and J.-B. Mouret, "Reset-free Trial-and-Error Learning for Robot Damage Recovery," *Robotics and Autonomous Systems*, vol. 100, pp. 236–250, 2018.

[20] M. Allard, S. C. Smith, K. Chatzilygeroudis, B. Lim, and A. Cully, "Online Damage Recovery for Physical Robots with Hierarchical Quality-Diversity," *ACM Transactions on Evolutionary Learning and Optimization*, 2023.

[21] A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune, "First return, then explore," *Nature*, vol. 590, no. 7847, pp. 580–586, Feb. 2021.

[22] C. Lu, R. Georgescu, and J. Verwey, "Go-Explore Complex 3D Game Environments for Automated Reachability Testing," *IEEE Transactions on Games*, 2022.

[23] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[24] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference for Learning Representations (ICLR)*, 2015.

[25] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International Conference on Machine Learning (ICML)*, 2018.

[26] A. Van Den Oord, O. Vinyals *et al.*, "Neural discrete representation learning," *Advances in Neural Information Processing Systems (NIPS)*, 2017.

[27] J.-B. Mouret and S. Doncieux, "Encouraging behavioral diversity in evolutionary robotics: an empirical study," *Evolutionary Computation*, vol. 20, no. 1, pp. 91–133, 2012.

[28] J. Lee, M. X. Grey, S. Ha, T. Kunz, S. Jain, Y. Ye, S. S. Srinivasa, M. Stilman, and C. K. Liu, "DART: Dynamic animation and robotics toolkit," *Journal of Open Source Software*, vol. 3, no. 22, p. 500, 2018.