Shaped Policy Search for Evolutionary Strategies using Waypoints*

Kiran Lekkala[1] and Laurent Itti[2]

***Abstract*— In this paper, we try to better exploration in Blackbox methods, particularly Evolution strategies (ES), when applied to Reinforcement Learning (RL) problems where intermediate waypoints/subgoals are available. Since Evolutionary strategies are highly parallelizable, instead of extracting just a scalar cumulative reward, we use the state-action pairs from the trajectories obtained during rollouts/evaluations, to learn the dynamics of the agent. The learnt dynamics are then used in the optimization procedure to speed-up training. Lastly, we show how our proposed approach is universally applicable by presenting results from experiments conducted on Carla driving and UR5 robotic arm simulators.**

I. Introduction

Reinforcement learning (RL) is one of the most popular methods used in a wide range of Robotic applications. Although RL has shown promising results in the past, one of the main challenges that arise in RL is that it requires evaluating a large number of samples on the environment. Evolutionary strategies are one of the least widely used RL methods, which, although are highly parallelizable, have poor sample efficiency. In situations where we have access to waypoints/intermediate subgoals, we could further improve the performance of these methods, apart from the better parallelization.

A common practice to run RL algorithms involves single-threaded sequential policy learning. Algorithms on scaling these sequential learning methods to distributed settings [21] are getting popular as they can be linearly scaled to the number of machines/cores. These methods are widely used where the agent is simulated or where there is access to multiple robots. However, all these works focus on policy optimization and neglect, learning useful information like dynamics from the collected experiences.

AllReduce [22] is the most common algorithm used to extend policy gradient methods to distributed settings. Existing solutions like [12], [4] and [1] better RL by including expert demonstrations, using sub-goals, and learning from failed distributions.

Our method shapes the policy search such that the sampling procedure is biased towards the optimal parameters (Figure 2). To achieve this, we generate a noisy action label and evaluate a surrogate/noisy estimate of the gradients of the policy parameters. We can then sample parameters based on their alignment with these gradients, unlike uniform sampling.

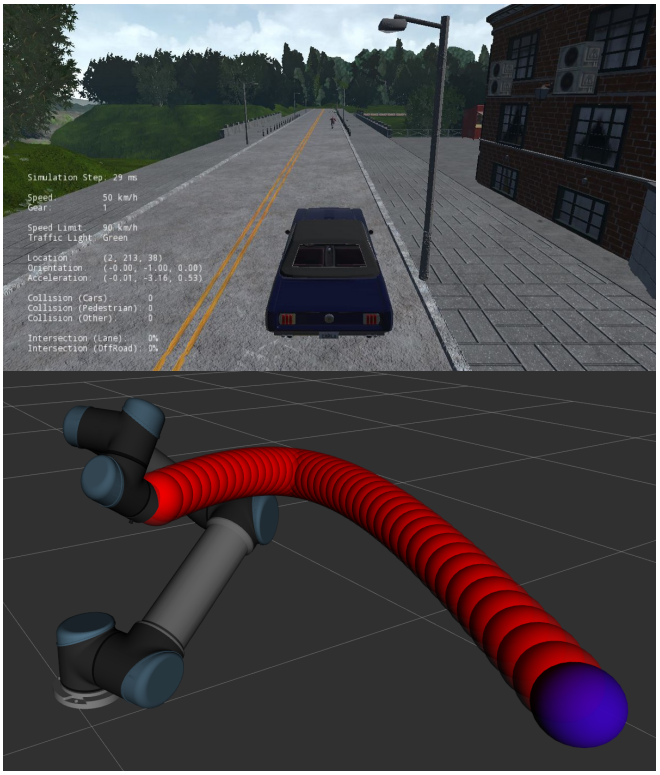

Fig. 1. We test our method on the (a) Carla and the (b) UR5 Gazebo simulators. In Carla simulator, the agent had to solve a point goal navigation task, where it has to reach a destination location from a randomly spawned location. In UR5 Gazebo simulator, the task for the agent was to move the fixed end-effector in alignment with the trajectory consisting of the target pose (blue) and the waypoints (red).

II. Related Work

Standard RL methods rely on reward signals from the environment to find the optimal parameters for a policy. Two of the most active research areas in RL are overcoming sparse rewards and exploration. In a lot of practical scenarios, environments have long horizon and sparse rewards, and it becomes challenging for RL methods to optimize the reward function. [1] proposes a novel approach to alleviate sparse rewards by learning from failed trajectories, which dovetails with our motivation. Although shaped rewards can be used these scenarios, they often lead to sub-optimal solutions.

Derivative-free optimization methods have been used in the past for problems where computation of a gradient is

*This work was supported by C-BRIC (one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA), National Science Foundation (grants CCF-1317433 and CNS-1545089) and Intel Corporation. The authors affirm that the views expressed herein are solely their own, and do not represent the views of the United States government or any agency thereof.

[1]Kiran Lekkala is with the ILab, Department of Computer Science, University of Southern California, 90089, USA klekkala@usc.edu

[2]Laurent Itti is with the ILab, Department of Computer Science, Psychology and NGP, University of Southern California itti@usc.edu

Manuscript 1771 submitted to 2021 IEEE International Conference on Robotics and Automation (ICRA). Received November 2, 2020.

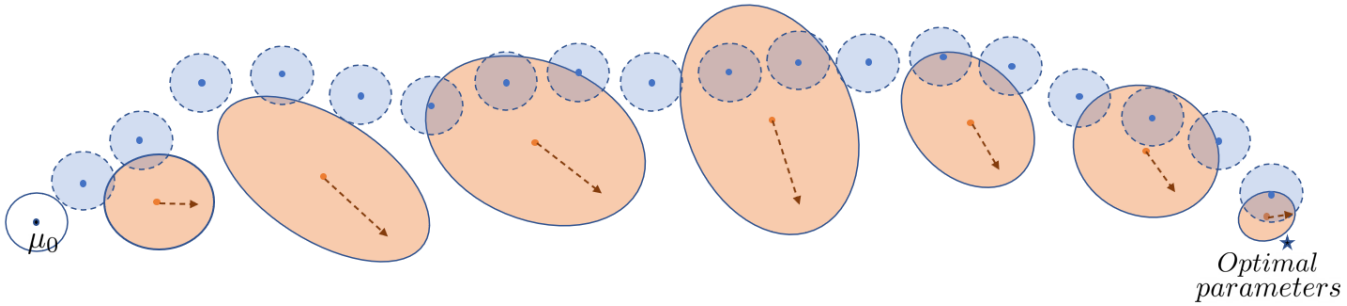

Fig. 2. Pictorial overview of our method. Blue represents the uniform sampling distributions used by vanilla-ES method, and orange represents the sampling done by our method. Center dots in both the sampling distributions represents the mean. Instead of sampling parameters from a standard Gaussian of a fixed covariance, like that of vanilla-ES, we extend the search space towards the optimum using the surrogate gradients (brown).

not feasible, like that of RL. Most popular *BlackBox methods* like evolution strategies work by sampling parameters from a distribution and evaluating them to take an update step. *Covariance-based Evolution strategy (CMA-ES)* [8] and *Cross-Entropy* [15] are two most prominent algorithms widely used in RL and in they perform surprisingly well when compared to the state of the art policy gradient methods. Other versions of ES like PEPG [18] and NES [20] carry out parameter search with novel techniques inspired from Reinforce, which is one of the earliest policy gradient algorithm [7].

Imitation learning (IL) involves learning a policy from a set of expert demonstrations without explicitly designing any reward function. The most naive form of Imitation learning, sometimes referred to as *Behaviour Cloning* requires fitting a model to the expert trajectories. Although behaviour cloning is a seemingly trivial problem, a significant challenge involves distribution mismatch of the expert and the agent, which leads to compounding errors. Furthermore, the trained agent would have obtained sub-optimal behaviour compared to the expert, and so the agent can never be better than the expert. Recent advances like enabling the use of *Inverse Reinforcement Learning (IRL)* also fall under the broad umbrella of IL. These methods use expert demonstrations to learn the reward function, which could then be used to optimize the policy using RL.

Expert demonstrations have also been used to supplement exploration in RL. Some of the prominent ones include fusing demonstrations in the optimization procedure with Q-learning as an auxiliary objective [5] and hybrid policy gradient [12]. Although the methods mentioned above show promising results on some benchmarks, there are inherent issues. First, expert demonstrations are off-policy, which makes the optimization challenging. Second, while trying to learn a policy using RL, the agent could forget the expert's trajectories, which nullifies the reason for bringing in the demonstrations.

Imitation from Observation (IfO) is an ongoing research area which deals with policy learning from expert observations, without the need of the expert actions. The policy is then learnt with the help of RL, by taking help from the expert observations, which could be same as the waypoints/intermediate goals. Unlike using demonstrations in RL, since there are no actions for the expert, there would not rise a problem of fusing in off-policy data. However, to determine the actions and reachable states is still a challenge. Recent IfO works, like [6] and [11] use an *Inverse Dynamics* model.

Although Blackbox methods, like CMA, are not constrained to environments having *Markov Decision Process (MDP)* assumption, and so can be used in a lot more applications, they are highly sample-inefficient. However, if we have access to intermediate waypoints, these BlackBox methods could be improved by learning an inverse dynamic model from the experiences generated. Since plenty of rollouts are generated by parallelizing across multiple machines, the inverse dynamics model could be trained efficiently.

In the ideal case, if these actions generated were the actual goal/task-specific action labels, this entire formulation would be reduced to a trivial problem of behaviour cloning. However, to obtain the action vectors is itself the problem we are trying to solve, i.e., policy estimation, we could only get the inverse dynamics model, that is conditioned on the intermediate waypoint, to infer a rough estimate of the action. We would then get noisy gradients evaluated on the data consisting of the noisy action labels, which could be used to bias the sampling process in the evolution strategy. This approach would favour the regions of the search space closer to the optimal parameters.

Unlike [11], where the policy is the same as the inverse dynamics model, our method is more practical as the policy does not depend on any intermediate waypoints. The inverse dynamics model is only used to speed-up training and does not play any role during test-time. Our method could be used in *Sim2Real* transfer scenarios, with training the policy in a simulator, where we have access to data in abundance. Following are our contributions:

We propose a methodology to extend and apply Guided Evolutionary strategy [10] to RL settings where the agent has access to the waypoints in an environment. This also involves, using an inverse dynamics model to generate noisy action labels, from which gradients are evaluated. In the experimental section, we outline how our method can be scaled and deployed on a Beowulf cluster.

Manuscript 1771 submitted to 2021 IEEE International Conference on Robotics and Automation (ICRA). Received November 2, 2020.

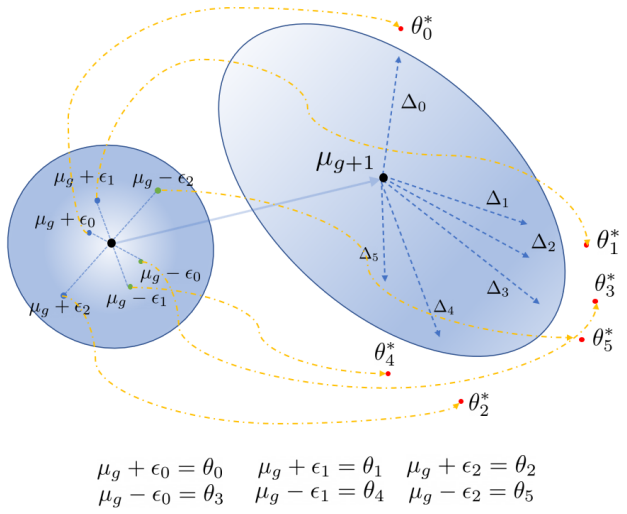

Fig. 3. Above figure shows how our method changes the uniform sampling to favour samples closer to the global optimum. For simplicity, we consider population size to be 6, where half of them have positive perturbations (in blue), and the other half have negative perturbations (green). These perturbations are with respect to the mean μ (in black). After evaluating these samples on the environment, they are updated to θ_i^* (in red) using the noisy actions from the inverse-dynamics model by behaviour cloning. Yellow dot-broken arrow represents gradient descent when done by each worker on the sample. The sampling distribution is then modified using the surrogate gradients Δ_i pointing towards θ_i^*.

III. Problem Formulation and Preliminaries

Reinforcement learning methods are based on an MDP formulation, characterized by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{S}^{'})$, where $\mathcal{S}$ and $\mathcal{A}$ are the set of states and actions. $\mathcal{R}(s,a)$ is the reward function when the agent is at state s and takes action a. $\mathcal{S}^{'}$ is the set of possible next states reached from $\mathcal{S}$. As a baseline comparison to our method, we use Proximal Policy Optimization (PPO) [17] an on-policy based method, which is the state of the art policy gradient method on continuous control tasks.

Unlike the MDP based methods, we use black-box based evolutionary strategies for policy search. CMA-ES, one of the most widely used variants of evolutionary strategy which is known for its robustness to multiple local minima and has proved to be efficient in solving RL problems in comparison to the state of the art policy gradient methods [16]. The core idea is to sample policy parameters from a parameterized distribution like Gaussian and evaluate those parameters on the environment. The mean μ and the covariance matrix Σ of the Gaussian distributions is then updated using the M_{best} (highest rewards in our case) parameters, in the most standard case is 25, over every g^{th} generation. Although, the covariance matrix calculation during every generation is expensive as it is $\mathcal{O}(N^2)$, and might not be feasible for parameters more than 10000, there are some practical tricks to mitigate these complexities [7].

$$\mu = \frac{1}{M_{best}} \sum_{k=1}^{M_{best}} \Theta_k \tag{1}$$

$$\Sigma_{ij} = \frac{1}{M_{best}} \sum_{k=1}^{M_{best}} (\Theta_k^i - \mu^i)(\Theta_k^j - \mu^j)^T \tag{2}$$

In the above equation, Θ_k is the k^{th} sampled parameter and Θ^i is the i^{th} index of all sampled parameters. Σ_{ij} refers to the i^{th} column and j^{th} row in a matrix. In the remainder of the paper, parameters refer to that of the neural network after flattening. We use the vanilla version of the ES algorithm, which involves updating the mean of the Gaussian distribution alone and having the covariance fixed. We improve this by using the dynamics to estimate the covariance matrix. Vanilla-ES method is based on finite differences and falls roughly into the category of *Zeroth order* methods, which is used in applications, where gradients are not available. Compared to CMA-ES, these methods supposedly fail in a lot of applications, because of their inability to escape local minima. Similar to CMA-ES, these methods sample perturbations from standard Gaussian $\mathcal{N}(\mu, \sigma^2 I)$, where I and σ represents identity matrix and variance respectively, and use the evaluated values to move the mean towards a direction. We use antithetic sampling, which involves sampling P points and P mirror images of those points with respect to the mean of the distribution. Antithetic sampling has proven to reduce the variance of the *Monte-Carlo estimator* as it draws correlated rather than independent samples [13]. All the members in a *population* are then generated by $\mu + \epsilon\sigma$. These members are evaluated in the simulator. In the remainder of the paper, the term population represents a set of sampled policy parameters.

Now we present the notations and definitions which are used in this paper. We denote $\pi_\theta(s_t)$ as the policy which is a neural network, parametrized by θ, which receives a state s_t to output a_t at every time-step t. To generate actions for each state using intermediate waypoints, apart from the policy, we also have the inverse dynamics model $\mathcal{M}_\phi(s_t, \hat{s}_t)$, which is parametrized by ϕ. During the train-time, we also have access to the local waypoint $\hat{s}_t$, along with a state s_t. The gradient of the policy is represented by $\nabla\mathcal{L}(\theta)$, where $\mathcal{L}$ is the loss function, in our case is the Mean squared error of the actions predicted by the policy π_θ to that of the inverse dynamics model $\mathcal{M}_\phi$. n is the total number of parameters in the policy neural network, and N is the total number of workers in our distributed setup. Lastly, note that the bold counterpart of a symbol corresponds to an array of scalars or vectors, depending on that symbol.

Also note that there are global waypoints denoted by $s_t^{'}$ which form the basis for the reward function, i.e., the agent gets a positive reward when it passes through a global-waypoint. The global waypoint is also used to estimate the heading angle in the state vector which the agent receives at every time-step. Since the global waypoint can have a varying resolution, i.e., sparser or denser, local waypoints are computed so that the agent can reach the next local waypoint and get closer to the global waypoint. During train-time, the agent has access to these local waypoints but not during test-

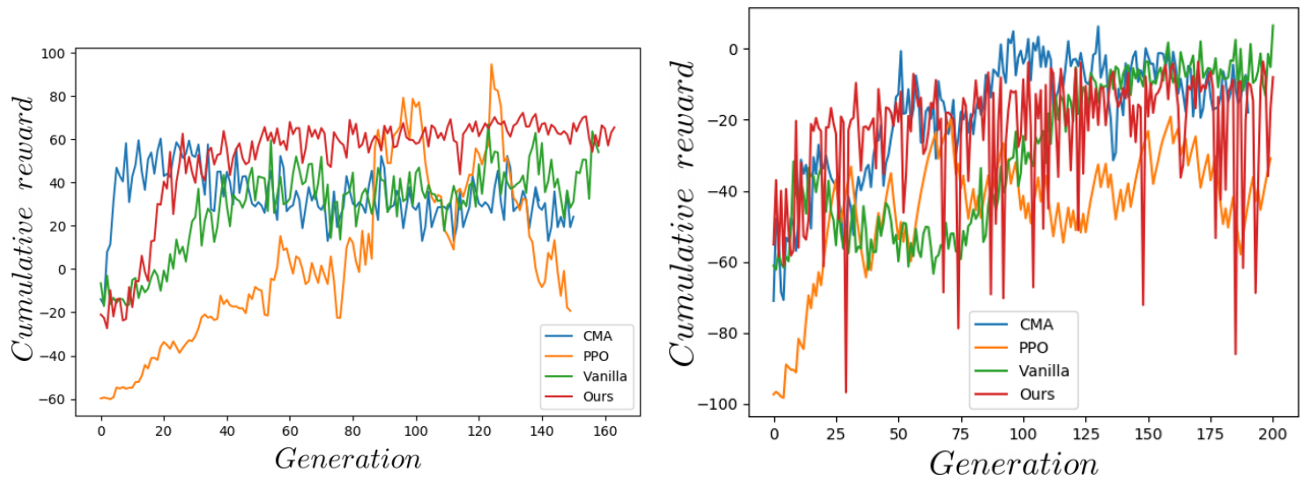

Fig. 4. Comparison of our method with the benchmarks CMA-ES, Vanilla-ES, PPO on the (a) CARLA driving simulator and (b) UR5 Gazebo simulator. x axis represents the generation and the y axis represents the cumulative reward. We provide the mean reward of the best memberin the population averaged on 15 test episodes, in the case of ES benchmarks. Our method reaches the global optimum faster and performs better than all the baselines.

time, which corroborates the use of inverse dynamics model only during training. Please note that M is the max number of frames which the agent can see, whereas the episode length or horizon H is set by the simulator. Since, it might not be possible for the simulator to find episodes which are of fixed size, the simulator is given a range of values to find an episode.

IV. Proposed Method

In this section, we outline the main contributions of our method. The detailed implementation details are given in the experiments section and note that our method along with the other evolutionary baselines are run on a cluster of machines, as a distributed, centralized setup, where a Master node generates the parameters and worker nodes evaluates them on the simulator.

A. Estimating actions

We use an inverse dynamics model $\mathcal{M}_\phi$ to estimate the action using the current state and the locally generated waypoint. As mentioned before, a simulator provides a global waypoint and a local waypoint is generated with the constraint of agent's reachability. At the end of each generation, with the exception of the first generation, we aggregate all the rollouts until the current generation and train the inverse dynamics model for every iteration. Since this process happens at different worker nodes in a cluster, each node only has access to its own history of rollouts, and this procedure occurs in parallel at different nodes.

Since the actions which are estimated by the inverse dynamics model corresponding to each state in a trajectory are on-policy, we can use behaviour cloning using DAgger [14] to fit the policy on the data collected. To achieve that, the worker estimates the gradient of the policy parameters by using the following Mean squared error (MSE) loss function and does gradient descent.

$$\mathcal{L}(\theta) = \frac{1}{M}\sum_{j=0}^{j=M}(\pi_\theta(s_t) - \mathcal{M}_\phi(s_t, \hat{s}_t))^2 \tag{3}$$

In the above equation, π_θ is the policy, $\mathcal{M}_\phi$ is the inverse dynamics model, and M is the episode length. After every generation, each worker would fit the policy π_{θ_i} using the aggregated data by taking successive gradient steps, after which we obtain θ_i^*, where i is the index of the worker. θ_i^* along with the cumulative reward R_i are then sent to the master. The reader is reminded that, for the sake of simplicity, Unlike $\mathbf{\Theta_g}$ (array of policy parameters), g (generation) is omitted out from θ_i (policy parameters received by i^{th} worker. This is the same in case of $\mathbf{R_g}$ and R_i.

The parameters of $\mathcal{M}_\phi$ are updated at the master node by training on the trajectories recorded from all workers, during a particular generation. Another thread, meanwhile, waits for the rewards $\mathbf{R_g}$ and $\mathbf{\Theta_g}$, which are a vector of rewards R_i and θ_i^* respectively. Both of them happen concurrently, to speed-up training.

B. Incorporating gradients

As stated in the preliminaries, for vanilla-ES algorithm, we sample some perturbations $\boldsymbol{\epsilon}$ around the mean of the Gaussian μ, and update it as follows:

$$\mu_{g+1} \leftarrow \mu_g - \gamma \cdot \frac{\beta}{\sigma^2 2P}\sum_{i=0}^{P}\epsilon_i \cdot [F(\mu_g + \epsilon_i) - F(\mu_g - \epsilon_i)] \tag{4}$$

In the above equation, $2P$ is the number of members in a population, since we are doing antithetic sampling. F is the fitness function or the reward function. β and σ are the normalizing coefficient and variance, respectively. γ is the learning rate of the gradient ascent (since we maximize the reward function). $\mathbf{\Theta}_g$ is an array of policy parameters, that

are sent to the workers are estimated using $(\mu + \epsilon)$[1]. ϵ_i is a perturbation sampled from $\mathcal{N}(0, I_n)$, where n is the length of the policy parameters and so the identity matrix I_n is of dimension $n \times n$.

Recently, [10] proposed a method to improve training of the Vanilla-ES in situations where surrogate/noisy gradient is available. Instead of having a search covaraince of σI_n, the solutions could then be drawn from a Gaussian having Σ as follows:

$$\Sigma = \frac{\alpha}{n} I_n + \frac{1-\alpha}{k} UU^T \tag{5}$$

In the above equation, U is a low-rank, orthogonal basis of the subspace spanned by the surrogate gradients. The orthogonal basis can be obtained easily from QR decomposition of the gradients, which is an $k \times n$ matrix, where n is the number of policy parameters/ size of the gradient and k is the number of gradients obtained, which is equal to the number of workers. α is the coefficient which weights the gradient perturbations over the random ones. Taking inspiration from [10], we also apply a similar concept in our case, where the surrogate gradients could be pointing towards the optimal parameters. From the previous subsection, we could use:

$$\mathbf{\Delta_g} = \frac{\mathbf{\Theta_g}^* - \mu_{g+1}}{\eta} \tag{6}$$

In the above equation, $\mathbf{\Delta_g}$ represents an array of surrogate gradients. Again, since μ_{g+1} is a single vector and $\mathbf{\Theta_g}^*$ is an array of vectors (θ_i^*), by broadcasting we get an array of surrogate gradients. The normalization constant η is to ensure that the covariance matrix does not become large, to make the sampling inaccurate. The CMA-ES algorithm does not explicitly control the step-size of the distribution, but at any given step or generation, it increases or decreases the scale only in a single direction for each selected step [8]. But in our case, since the gradient subspace consists of multiple optimum points, we can induce the distribution towards the direction of the optimal policy, as shown in Figure 3. The entire algorithm is described in the figure on this page, with steps corresponding to master or worker nodes in blue and red, respectively.

Algorithm 1 Policy shaping using surrogate gradients

Require: Master and worker steps in blue and red
Require: U as the gradient subspace
Require: Δ as the surrogate gradient
Require: N as the number of workers
Require: $\{\mathcal{D}_g^i\}_{i=0}^N$ as data-buffers for each worker
Initialize $\{\mathcal{D}_i\}_{i=0}^N$ to empty set
Initialize μ_0 to 0
Initialize U to $n \times n$ identity matrix
while not done **do**
  Estimate U from `QRDecompose(`$\mathbf{\Delta_g}$`)`
  Estimate solutions $\mathbf{\Theta_g} \sim \mathcal{N}(\mu_g, UU^T)$
  `comm.send(`$\mathbf{\Theta_g}$`)`
  `comm.recv(`θ_i`)`
  Evaluate θ_i to obtain $\{s_j, \hat{s}_j\}_{j=0}^M$
  Save $\{s_j, \hat{s}_j, a_j\}_{j=0}^M$ onto NFS
  Aggregate data: $\mathcal{D}_g^i \leftarrow \{s_j, \hat{a}\}_{j=0}^M \cup \mathcal{D}_{g-1}^i$
  Estimate $\hat{a}$ from $\mathcal{M}_\phi(s_j, \hat{s}_j)$
  Train on $\mathcal{D}_g^i$ to obtain θ_i^*
  `comm.send(`R_i`,` θ_i^*`)`
  `comm.recv(`$\mathbf{R_g}$`,` $\mathbf{\Theta_g^*}$`)`
  $\mu_{g+1} \leftarrow \mu_g$ - $\gamma \cdot \frac{\beta}{2\sigma^2 P} \sum_{i=0}^{P} \epsilon_i \cdot [F(\mu_g + \epsilon_i) - F(\mu_g - \epsilon_i)]$
  Calculate $\mathbf{\Delta_g} = \frac{\mathbf{\Theta_g}^* - \mu_{g+1}}{\eta}$
  Train $\mathcal{M}_\phi$ using experiences collected
end while

V. Experimental Results

Our method was evaluated on two distinct simulators involves long horizon tasks with sparse rewards. At this point, we would like to remind the reader that our method is not only limited to simulators; we could use the same setup in cases where there are multiple robots to speed-up training. Many popular works use simulators like OpenAI [2] and Mujoco [19], where the simulations are run as a wrapper inside the worker scripts which could be parallelized. Unlike them, our entire experimental setup was based on a distributed client-server setup. We use docker [2] containers to run the simulations, as our simulators involve high-end graphics and so have to be run as standalone servers. We used 13 machines of different hardware configurations and set them up as a Beowulf cluster. We use a common Network File system (NFS) shared amongst all the machines in the cluster, onto which all the trajectories are written to, and are used by the master node to train the inverse dynamics model. All the distributed methods were implemented in OpenMPI[3]. Experimental details irrespective of the simulator used are as follows.

At the end of every generation, the fitness/reward values are rank transformed to the range of -1 to +1 to eliminate any outliers in the population dominating the update procedure. In the case of CMA-ES, weight decay of 0.05 was used to prevent overfitting. We used a population size of 64, evaluated by 64 train workers. Additionally, we used 15 evaluation workers to evaluate the best member of the population. Each worker evaluates the weight vector two times using different seeds to obtain the mean cumulative reward and the gradients. For both the simulators, we followed a similar reward strategy similar to the CarRacing-v0 simulator from the OpenAI gym [2]. The agent receives a reward of $(\frac{M}{H})$ if it passes through a waypoint, where M and H are the max-frames and horizon, respectively. In case of a collision, a negative reward of -50 is received and the episode is terminated. To encourage the agent to reach the goal faster, it also receives a negative reward of -0.1 irrespective of any state (positive or negative rewarded state). In regards to

[1]Note that ϵ is an array of vectors and μ is a single vector and so μ would get broadcasted over the array

[2]https://hub.docker.com/

[3]https://www.open-mpi.org/

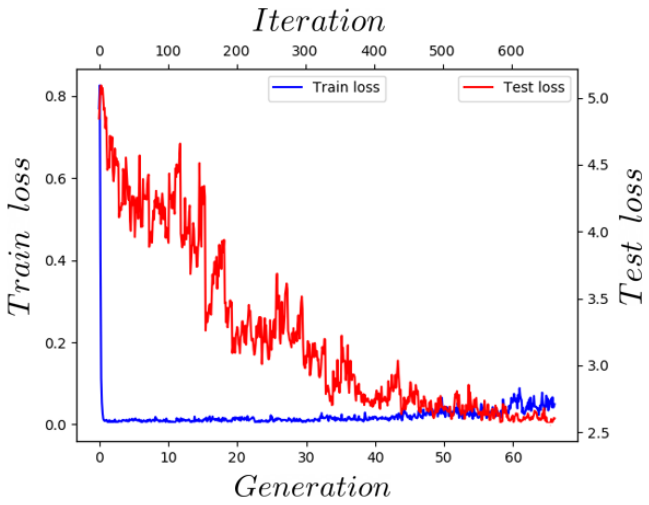

Fig. 5. Accuracy of the inverse dynamics model with generation in the Carla simulator. x axis represents the training iteration which is folded in with the generation. Test Loss corresponds to the euclidean distance between the actions and the ground-truth actions. Overall, we can see that the accuracy of the action prediction improves with iteration.

the training of the inverse-dynamics model, please note that trajectories which do not have a cumulative positive reward of 5 are discarded.

A. Carla Driving simulator

We used the Carla simulator [3], which is a High-end graphical simulator based on Unreal Engine. The task was to navigate from a source to destination location which are both sampled randomly in a specific environment/town (Town07). The agent receives a 38-dimensional state consisting of heading angle, odometry information, linear and angular velocity of the vehicle and the 32-dimensional lidar scan at every time-step. The agent then outputs an action vector of 2 dimensions, i.e., steering and throttle. To improve stability, we set a speed limit of 25 Kmph (whenever the agent crosses it, a brake is automatically applied). The resolution of the rewards is also a hyperparameter, which is set 1.5. As the resolution increases, the rewards tend to become more sparse. Performance of our method in comparison with other benchmarks are shown in Figure 4.

All the episodes had the number of waypoints between 800 and 1200. The maximum number of frames which the agent could see in a specific episode was set to 1000. Based on these values, when the agent is within the radius of 0.5m of the next waypoint, it receives a reward of 3.16. Other than this, there are no other positive rewards, which the agent could receive. The inverse dynamics model $\mathcal{M}_\phi$ is a two-layer neural network with 20 neurons in each layer with ReLU activation at each layer. We used a learning rate of 0.001 for training and a batch size of 64.

B. UR5 Robot arm in ROS Gazebo

Similar to the Navigation task mentioned above, we further tested our method on a robotic arm simulator developed by [9]. The task was to make the end-effector reach a goal coordinate from a fixed configuration (C-space), as shown in Figure 1. We modified the environment by computing Bezier curves from the source to the target end-effector position, which is randomly sampled from the upper hemisphere. We used a quadratic Bezier curve which uses a second-order Bernstein polynomial:

$$\mathbf{B}(\mathbf{t}) = \sum_{i=0}^{n} \binom{N}{k} (1-t)^{n-i} t^i \mathbf{P}_i, \qquad 0 \leq t \leq 1 \qquad (7)$$

The above equation involves 1 control point $\mathbf{P_1}$ which in our case were $(.75, 0.15, 0.45)$, apart from the source $(\mathbf{P_0})$ and the target $(\mathbf{P_2})$ end-effector poses. All the points belonging to this curve are obtained using different values of t ranging from 0 to 1. The reward mechanism was similar to the one used in the navigation task, i.e., -0.1 for every time-step and positive reward if the end-effector is within a distance of 0.1 units from the next local waypoint. In total, our method was able to achieve a cumulative reward of -2.8 after training for 200 generations, as shown in Figure 4. Unlike the Carla simulator, since we now have access to the exact number of points which we can sample from a Bezier curve, the horizon was equal to the maximum number of frames which was set to 100. A similar inverse dynamics model which was used on the Carla simulator was used here, with the modification of adding another layer, while keeping all the other details same. The input to the inverse dynamics model are joint velocities, Cartesian coordinates of the next waypoint with respect to the end-effector position in 3D space. The action vector consists of 5D joint torques.

Train-loss and test-loss of the inverse dynamics model $\mathcal{M}_\phi$ of the Carla simulator are shown in Figure 5. We use the actions from a fully trained policy as the ground-truth data. Train-loss corresponds to the loss obtained when training $\mathcal{M}_\phi$ on the rollouts. We compute Euclidean distance between the actions generated by $\mathcal{M}_\phi$ and the ground-truth actions, as the test-loss.

VI. Discussion and Conclusion

In this paper, we presented a method which could improve the performance of distributed evolutionary strategies for RL in scenarios where we have access to waypoints. By incorporating generated actions using an inverse-dynamics model, we have shown that the training process can be improved without no additional cost, as the experiences are generated during training.

We evaluated our method on the Carla driving simulator and the UR5 Gazebo simulator. Our results indicate the validity of the hypothesis. We also show the performance of the inverse dynamics model with each generation, which is trained on the rollouts generated on a distributed cluster of machines.

REFERENCES

[1] Marcin Andrychowicz et al. "Hindsight Experience Replay". In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*. Ed. by Isabelle Guyon et al. 2017, pp. 5048–5058. URL: http://papers.nips.cc/paper/7090-hindsight-experience-replay.

[2] Greg Brockman et al. "OpenAI Gym". In: *CoRR* abs/1606.01540 (2016). arXiv: 1606.01540. URL: http://arxiv.org/abs/1606.01540.

[3] Alexey Dosovitskiy et al. "CARLA: An Open Urban Driving Simulator". In: *1st Annual Conference on Robot Learning, CoRL 2017, Mountain View, California, USA, November 13-15, 2017, Proceedings*. Vol. 78. Proceedings of Machine Learning Research. PMLR, 2017, pp. 1–16. URL: http://proceedings.mlr.press/v78/dosovitskiy17a.html.

[4] Ben Eysenbach, Ruslan Salakhutdinov, and Sergey Levine. "Search on the Replay Buffer: Bridging Planning and Reinforcement Learning". In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*. Ed. by Hanna M. Wallach et al. 2019, pp. 15220–15231. URL: http://papers.nips.cc/paper/9660-search-on-the-replay-buffer-bridging-planning-and-reinforcement-learning.

[5] Yang Gao et al. "Reinforcement Learning from Imperfect Demonstrations". In: *CoRR* abs/1802.05313 (2018). arXiv: 1802.05313. URL: http://arxiv.org/abs/1802.05313.

[6] Xiaoxiao Guo et al. "Hybrid Reinforcement Learning with Expert State Sequences". In: *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*. AAAI Press, 2019, pp. 3739–3746. DOI: 10.1609/aaai.v33i01.33013739. URL: https://doi.org/10.1609/aaai.v33i01.33013739.

[7] David Ha. "A Visual Guide to Evolution Strategies". In: *blog.otoro.net* (2017). URL: https://blog.otoro.net/2017/10/29/visual-evolution-strategies/.

[8] Nikolaus Hansen. "The CMA Evolution Strategy: A Tutorial". In: *CoRR* abs/1604.00772 (2016). arXiv: 1604.00772. URL: http://arxiv.org/abs/1604.00772.

[9] Matteo Lucchi et al. "robo-gym - An Open Source Toolkit for Distributed Deep Reinforcement Learning on Real and Simulated Robots". In: *CoRR* abs/2007.02753 (2020). arXiv: 2007.02753. URL: https://arxiv.org/abs/2007.02753.

[10] Niru Maheswaranathan et al. "Guided evolutionary strategies: augmenting random search with surrogate gradients". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 4264–4273. URL: http://proceedings.mlr.press/v97/maheswaranathan19a.html.

[11] Brahma S. Pavse et al. "RIDM: Reinforced Inverse Dynamics Modeling for Learning from a Single Observed Demonstration". In: *IEEE Robotics Autom. Lett.* 5.4 (2020), pp. 6262–6269. DOI: 10.1109/LRA.2020.3010750. URL: https://doi.org/10.1109/LRA.2020.3010750.

[12] Aravind Rajeswaran et al. "Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations". In: *Robotics: Science and Systems XIV, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, June 26-30, 2018*. Ed. by Hadas Kress-Gazit et al. 2018. DOI: 10.15607/RSS.2018.XIV.049. URL: http://www.roboticsproceedings.org/rss14/p49.html.

[13] Hongyu Ren, Shengjia Zhao, and Stefano Ermon. "Adaptive Antithetic Sampling for Variance Reduction". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 5420–5428. URL: http://proceedings.mlr.press/v97/ren19b.html.

[14] Stéphane Ross, Geoffrey J. Gordon, and Drew Bagnell. "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011*. Ed. by Geoffrey J. Gordon, David B. Dunson, and Miroslav Dudík. Vol. 15. JMLR Proceedings. JMLR.org, 2011, pp. 627–635. URL: http://proceedings.mlr.press/v15/ross11a/ross11a.pdf.

[15] Reuven Y. Rubinstein and Dirk P. Kroese. *The Cross Entropy Method: A Unified Approach To Combinatorial Optimization, Monte-Carlo Simulation (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2004. ISBN: 038721240X.

[16] Tim Salimans et al. "Evolution Strategies as a Scalable Alternative to Reinforcement Learning". In: *CoRR*

abs/1703.03864 (2017). arXiv: 1703.03864. URL: http://arxiv.org/abs/1703.03864.

[17] John Schulman et al. "Proximal Policy Optimization Algorithms". In: *CoRR* abs/1707.06347 (2017). arXiv: 1707.06347. URL: http://arxiv.org/abs/1707.06347.

[18] Frank Sehnke et al. "Parameter-exploring policy gradients". In: *Neural Networks* 23.4 (2010), pp. 551–559. DOI: 10.1016/j.neunet.2009.12.004. URL: https://doi.org/10.1016/j.neunet.2009.12.004.

[19] Emanuel Todorov, Tom Erez, and Yuval Tassa. "MuJoCo: A physics engine for model-based control". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2012, Vilamoura, Algarve, Portugal, October 7-12, 2012*. IEEE, 2012, pp. 5026–5033. DOI: 10.1109/IROS.2012.6386109. URL: https://doi.org/10.1109/IROS.2012.6386109.

[20] Daan Wierstra et al. "Natural evolution strategies". In: *J. Mach. Learn. Res.* 15.1 (2014), pp. 949–980. URL: http://dl.acm.org/citation.cfm?id=2638566.

[21] Erik Wijmans et al. "DD-PPO: Learning Near-Perfect PointGoal Navigators from 2.5 Billion Frames". In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL: https://openreview.net/forum?id=H1gX8C4YPr.

[22] Huasha Zhao and John F. Canny. "Sparse Allreduce: Efficient Scalable Communication for Power-Law Data". In: *CoRR* abs/1312.3020 (2013). arXiv: 1312.3020. URL: http://arxiv.org/abs/1312.3020.