Augmenting Reinforcement Learning Policies using Human Samples

Ashvin Iyer

***Abstract*— As the interest and use cases in complex, high-dimensional robots increase, traditional control methods have struggled to keep up with these. As a result, learning-based solutions have become common to overcome this hurdle. Reinforcement Learning aims to set up the task in a simulated environment, and learn an optimal policy through trial and error. Imitation Learning uses human-curated motion samples for a robot to learn, imitating human behavior. Both methods have their merits and issues, and recent papers look to combine both methods. I look to implement and improve some existing methods that aim to utilize the strengths of both Reinforcement and Imitation Learning.**

I. Introduction

Over the last few years, there has been a growing interest in utilizing high-dimensional robotics to perform complex tasks: robot manipulators for precise surgical tasks, quadrupeds for exploration of difficult terrain, humanoids for autonomous manufacturing, and more.

In traditional robotics, model-based approaches are used to design closed-loop feedback systems and are often achieved using trajectory optimization [1]. Alternatively, some approaches plan in latent space (using graph [2] or sampling-based planning [3]) and utilize inverse kinematics [4] to convert between latent space and control space. Both of these methods rely on rigorous system identification and hand-tuned controllers and often fail to generalize in complex and nuanced environments.

With the recent boom of Machine Learning over the last decade, several learning-based solutions have been proposed for complex robot planning; namely, Imitation Learning and Reinforcement Learning.

Imitation Learning [5] is an increasingly popular solution for robot manipulation tasks. Imitation Learning learns by taking in examples of expert trajectories and learns to mimic similar behavior. Samples for Imitation Learning are usually collected through recording human tele-operation of the given task, and converting them into state-action pairs for learning. Imitation Learning is often done through Inverse Reinforcement Learning (IRL) [6] or Behavior Cloning (BC) [7]. Inverse Reinforcement Learning. Inverse Reinforcement Learning attempts to learn a reward function from expert samples and then trains a policy using the learned function. Behavior Cloning instead learns a state-action pair mapping directly, using some neural network architecture. Imitation Learning has great success on scenarios very similar to those in the examples but can struggle to generalize beyond that. Although this can be solved with a large and robust dataset, that may not be feasible for some applications.

Reinforcement Learning [8] works by learning a motion policy through a hand-engineered reward function. A robot simulation environment is used to generate state-action trajectories, and the policy learns the positive and negative actions through feedback from the reward function. Deep-reinforcement learning with a well-defined reward function can generalize incredibly well; however, it is challenging to define an effective reward. RL often gets stuck in local minima or does not converge at all, depending on the complexity of the task and the definition of the reward. There is also the challenge of Sim2Real [9], where the policy learned in a simulated environment may not be feasible in the real world due to noise or safety factors. In such cases, it may be useful to follow a human-like trajectory, which is usually done through IL.

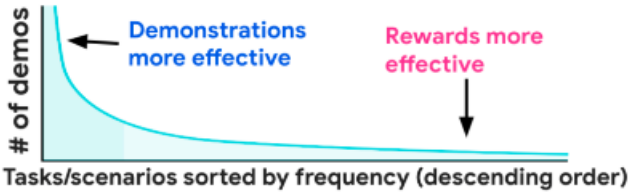

Fig. 1. Curve demonstrating tradeoffs of IL and RL from [10]

This paper aims to integrate Imitation and Reinforcement Learning to As demonstrated in Fig. 1, I want to utilize Imitation Learning to get peak performance when samples are available, but still get the benefits of RL, which is more generalizable. By combining both methods, I hope to achieve the following:

1) (Quickly) Converged Policy: Utilizing samples can help guide RL exploration in the correct direction, finding global optima.

2) Generalized Policy: RL will be able to generalize behavior in all configurations through repeated trial-and-error.

3) Human-Like Policy: Guiding RL using human samples will help the policy to converge to human-like weights, rather than some other solution.

II. Related Work

Several methods exist on augmenting RL with other approaches to improve sample efficiency and promote the search for global optima. Some approaches integrate sampling-based methods, such as RRT* [11] or PRM [12] into RL methods. These methods improve the sample efficiency of RL by enabling RL to sample in random tree's latent space, and handling conversion to control space through classical planning and kinematics.

Constrained Policy Optimization [13] enables configuring constraints in the RL problem directly, preventing the policy from taking invalid action and reducing the search size.

There also exist methods that look to combine expert demonstrations with Reinforcement Learning. Deep Q-Learning from Demonstrations (DqfD) [14] initializes a TD-policy by pretraining the network weights with behavior cloning. This helps guide the training of the RL policy towards the expert behavior, and the remaining exploration helps generalize the policy.

Demo-Augmented Policy Gradient (DAPG) [15] also integrates the Natural Policy Gradient (NPG) [16] algorithm with Behavior Cloning. They pre-train their policy with Behavior Cloning, similarly to DqfD. However, they also integrate BC loss into their optimization function directly, which ensures a human-like behavior for their policy.

LOKI [17] also uses policy gradients, but alternates between using an imitation gradient and a reinforcement learning gradient to learn an effective policy. This method focuses more on helping RL policies converge and does not necessarily guarantee behavior similar to the examples.

Nuro takes a different approach with CIMRL [18], which modifies Recovery-RL [19] to take a discrete number of trajectories and output a score for each one. For their specific case, they utilized Imitation Learning to generate the inputs for Reinforcement Learning, but theoretically any trajectory could be inputted.

Waymo [10] proposes combining Behavior Cloning with Reinforcement Learning by integrating a BC [20] loss function into a SAC [21] actor loss. The following methods will go into more detail about BC-SAC, as well as modifications to improve this method.

III. Methods

In the Methods below, I will look at a Soft Actor-Critic [21] family of methods for combining reinforcement learning with human samples. Although there are several techniques for using RL in robotics applications, I chose to look at specifically SAC models because of its great performance in complex, continuous environments, which are common for robotics applications.

Specifically, the performance of SAC and BC will be individually evaluated against BC-SAC [10], in addition to other novel modifications introduced in this project.

A. Soft Actor-Critic

Soft Actor-Critic (SAC) is a state-of-the-art off-policy method for training reinforcement learning tasks. SAC works by simultaneously training an Actor and Critic network. The actor network is designed to learn the policy at hand, and the critic network learns how to effectively evaluate the actor, using the reward function for the environment.

The objective function of the critic is defined as the following

$$\min_{Q} \mathbb{E}_{s,a,s' \sim \pi}[(Q(s,a) - \hat{Q}(s,a,s'))^2]$$

$\hat{Q}$ is the target value of the critic Q, and the critic learns to minimize this error. The target value is defined as

$$\hat{Q}(s,a,s') = r(s,a) + \gamma \mathbb{E}_{a' \sim \pi}[Q(s',a') - \log \pi(a'|s')]$$

This target is essentially the reward for a specific state action pair, plus the expected value of all possible results of the future step as well.

The actor network has the simple objective of maximizing the following

$$\max_{A} \mathbb{E}_{s,a \sim \pi}[Q(s,a) + H(\cdot|s)]$$

The goal is to maximize the Q value from the critic, with an additional H term, which maximizes the entropy of the SAC. The output of the actor is an action distribution, rather than a singular action. When sampling during exploration, the action taken is chosen from the distribution the policy provides, which promotes exploration. The goal of the term H is to increase the variance of the distribution to promote exploration. During inference, the expected value of the distribution is used as the model's output.

The high sample efficiency and improved exploration make SAC ideal for integrating with human samples, as the model needs to perform exploration to be able to generalize effectively. However, SAC is generally noisy and difficult to train efficiently. With expert demonstrations to help guide the SAC, it can avoid local-minima traps and quickly find the generally optimal search space. Here, SAC's efficient exploration can help fine-tune the policy to improve results.

B. Behavior Cloning

Behavior Cloning is a specific method of Imitation Learning, where the policy learns a mapping between states and actions via supervised learning. Specifically, given a set of state-action pairs $D = \{(s_0, a_0), (s_1, a_1), ...\}$, the objective function is maximized as

$$\mathbb{E}_{s,a \sim D}[\log \pi(a|s)]$$

Here, the goal is to maximize the probability that expert actions are taken given the expert states. This can be accomplished using any standard Neural Network Architecture.

Behavior Cloning can often struggle to generalize, especially to complex tasks, due to the simple nature of how the method is trained. The direct mapping from state to action prevents the model from performing long-term planning, meaning that the policy cannot recover from any mistakes. Other methods such as Direct Policy Learning or Inverse-RL can be more effective but also require a much more complex setup. Instead, RL can be used to integrate imitation in a simpler and smarter way.

C. Behavior Cloning Soft Actor-Critic

Behavior Cloning Soft Actor-Critic (BC-SAC) [10] combines expert demonstrations with the SAC architecture to improve performance. Specifically, they modify the actor objective function to the following

$$\max_{A} \mathbb{E}_{s,a \sim \pi}[Q(s,a) + H(\cdot|s)] + \lambda \mathbb{E}_{s,a \sim D}[\log \pi(a|s)]$$

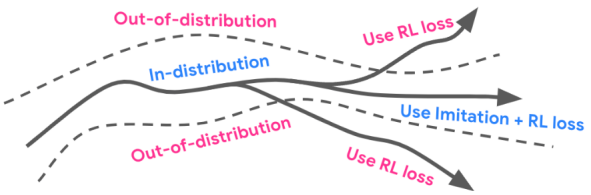

Fig. 2. Figure demonstrating when IL and RL error is used for BC-SAC [10]

Within the actor itself, a behavior cloning objective is added by evaluating how likely the current policy is to perform actions similar to the human samples (The distribution D). The λ term is a scaling factor that determines how much to weight the RL error versus the IL error, and can be tuned as needed.

The goal of the authors is to utilize behavior cloning loss to improve trajectories within the distribution D, but to still obtain adequate performance in out-of-distribution (OOD) regions through RL loss. They expect the OOD performance to be similar to that of a normal RL (as shown in Fig. 2), as the BC error will be large regardless of an OOD scenario.

D. Discounted Behavior Cloning Soft Actor-Critic

Discounted Behavior Cloning Soft Actor-Critic (DBC-SAC) is a modification to BC-SAC introduced in this project. Specifically, one small but crucial change is introduced to the actor objective function

$$\max_A \mathbb{E}_{s,a\sim\pi}[Q(s,a) + H(\cdot|s)] + \lambda_0 \alpha^n \mathbb{E}_{s,a\sim D}[\log \pi(a|s)]$$

Instead of using a constant λ to balance the importance between RL and IL loss, λ becomes an exponentially decaying function. The function starts with some initial λ_0, and has an exponential decay factor of $\alpha \in [0, 1]$ (n is the number of iterations carried out in training so far).

Using standard BC-SAC, the value of λ is very important, as it determines if you under or over-fit to the expert samples. However, the desired behavior is to allow RL to freely explore around the IL distribution, which will allow the policy to generalize effectively. With BC-SAC, a balance between RL and IL is utilized, but the behavior explained above is not actually achieved. By using a decaying function, the actor initially mimics the IL policy and over time promotes the RL loss to modify the policy around the IL distribution.

E. Self-Imitation Demonstrations

Self-Imitation Learning (SIL) [22] is a method that helps improve Reinforcement Learning Policies. The method works by taking the best outputs of a SAC model, and inputting them into a separate replay buffer that is used to perform periodic offline training. This helps the model learn from good results more often and converge to a better solution faster.

Self-Imitation Demonstrations (SID) is a method that takes inspiration from SIL, but is compatible with BC-SAC. Similarly to SIL, SID utilizes the good policy outputs to help enhance the final policy. However, these outputs are added to the distribution D of expert samples, rather than running offline training with them. Since the outputs of the policy will cover some OOD scenarios, adding them to the expert distribution can improve the coverage provided for the IL error, leading to an overall more positive result.

1) Imitation Learning Demonstrations: Imitation Learning Demonstrations (ILD) is very similar to SIDs, but rather than using the same model, the extra demonstrations are generated directly from some IL model. This may help the policy to behave more closely to the expert demonstrations itself.

F. Demo Augmented Soft Actor-Critic

Demo Augmented Soft Actor-Critic (DA-SAC) is another new method introduced in this project and takes inspiration from SIL. The core idea of periodically using good examples to promote learning positive rewards remains the same. However, instead of utilizing previously run examples, the expert demonstrations are added to the replay buffer, to periodically re-align the policy with the imitation objective. Details of this are shown in Algorithm 1.

Algorithm 1 DA-SAC: Modified SAC [21] implementation

procedure DA-SAC
  $R \leftarrow \{\}$
  $R_d \leftarrow \{\}$ ▷ Demo Replay Buffer
  for each trajectory $T \in D$ **do**
    for each timestep $t \in T$ **do**
      $R_d \leftarrow R_d \cup \{s_t, a_t, r_t, s_{t+1}\}$
    end for
  end for
  for $t \in [0, n]$ **do**
    $a_t \leftarrow \pi(s_t)$
    $s_{t+1}, r_{t+1} \leftarrow \text{Env}(a_t)$
    $R \leftarrow R \cup \{s_t, a_t, r_t, s_{t+1}\}$
    $d \leftarrow\sim R$ ▷ Sample from Replay Buffer
    sac.update(d)
    if $t \mod f = 0$ **then** ▷ Demo Frequency
      $d \leftarrow\sim R_d$
      sac.update(d)
    end if
  end for
end procedure

IV. Experiments

A. Experiment Setup

All the methods explained above are tested using the Hand Manipulation Suite from RoboHive [23], which utilizes Mujoco [24] as the backend simulator. Specifically, all the models are trained on the $door - v1$ task, which a high-dof robot arm opening a door, as shown in Fig. 3.

The state space comprises the position of the hand, the angles of the joints and the position of the doors / angles in $\mathbb{R}^{38}$. The action space consists of the torque inputs

Fig. 3. Environment from Hand Manipulation Suite from Robohive. [23]

for each joint / position of the robot hand in $\mathbb{R}^{28}$. The mujoco simulation handles all the physics of progressing between states. The suite also provides a set of 75 human demonstrations that will be used to integrate IL into models.

A default reward function is provided. All the experiments below are trained using this default reward function, as the goal is to test how well each of these models perform generally, rather than attempting to fine-tune the reward. The reward function is defined as:

$$R = \alpha_1 Re + \alpha_2 O + \alpha_3 B$$

Re defines the distance between the hand and the handle, O defines how close the handle is to being open (angular squared distance from $90°$), and B sets individual rewards for reaching certain angle thresholds. By default $\alpha_1 = \alpha_2 = -0.1$, $\alpha_3 = 1$.

$$Re = ||p_{hand} - p_{handle}||_2$$

$$O = (\theta_{door} - 90°)^2$$

$$B = \begin{cases} 0 & \text{if } \theta_{door} < t_1 \\ b_1 & \text{if } t_1 \leq \theta_{door} < t_2 \\ ... & ... \\ \sum_{i=1}^{n-1} b_i & \text{if } t_{n-1} \leq \theta_{door} < t_n \\ \sum_{i=1}^{n} b_i & \text{if } t_n \leq \theta_{door} \end{cases}$$

The reward for reacing certain goals is cumulative (get rewards for all previous goals as well). By default, the values of b and t are $b = [2, 8, 10]$ and $t = [0.2, 1, 1.35]$ (rad).

B. Model Implementation

All the methods are trained using some variation of a SAC. To implement this, I utilized an existing pytorch SAC implementation [25] that uses a Double-Q Critic with Multi-layer Perceptions and a Diagonal Gaussian Actor, and made necessary modifications to train all the models. For training details, the following configuration was used:

- Actor Model Dimensions: $38 \times 1024 \times 1024 \times 56$ (output mean and variance)
- Critic Model Dimensions: $66 \times 1024 \times 1024 \times 1$
- Learning Rate: 1^{-5}
- Discount: 0.99
- Seed Steps: 10000
- Training Steps: 100000

C. Models

The following models are trained and tested:

1) SAC Baseline: A traditional SAC without any BC integration trained as a baseline comparison.

2) BC Baseline: Similarly to SAC, a BC model without any RL is trained as another baseline comparison.

3) BC-SAC λ Variation: Various BC-SAC models are trained to observe the impact of λ. The values of $\lambda = 0.1, 1, 10$ are used.

4) DBC-SAC: A DBC-SAC model with $\lambda_0 = 1$ and $\alpha = 0.999$ is trained to compare to BC-SAC and see improvements.

5) ILD: ILDs are added to both BC-SAC and DBC-SAC to observe improvements. The IL policy for generating experts is the same BC model trained for comparison.

6) SID: SIDs are added to BC-SAC and DBC-SAC to observe improvements.

7) DA-SAC: DA-SAC model is trained with a frequency of $0.5hz$ (every other iteration).

D. Metrics

Each model is run over 1000 different variations of the same door-opening environment. From this, the following metrics are computed:

1) Average Reward: : Gives a general idea of how well the task is performed

2) Average Door Angle: : Gives an understanding of how well the door is opened. Two averages are computed: 1 across all samples and 1 across only samples where the door was successfully opened.

3) Door Opening Angles: : The percentages of reaching certain angle thresholds are also computed. There is a threshold for being able to open the door at all, in addition to the 3 angles specified in the reward function ($0.2, 1, 1.35$ rad or $11.5°, 57.3°, 77.3°$).

E. Results

All the numerical metrics can be found in Table I and Table II. Overall, the results show that while SAC and BC struggle on their own to obtain good solutions, combining both is very effective in getting a more generalized and effective policy.

1) SAC Performance: SAC struggled a lot to converge to a good solution. Without any guidance from expert samples, the SAC model learned d to open the door using the dorsum (back of the hand), rather than grabbing the handle with the fingers (as seen in Fig. 4). Sometimes, it would get lucky and pull the door slightly open despite this technique, but this is mostly due to artifacts of the simulation and would likely not work in the real world.

TABLE I

EVALUATION OF AVERAGE REWARD AND AVERAGE ANGLE FOR EACH MODEL

Model	Average Reward	Average Door Angle° (w/ failures)	Average Door Angle° (w/o failures)
SAC	4.1	2.2°	12.3°
BC	461	41.1°	55.9°
BC-SAC ($\lambda = 0.1$)	65.3	7.9°	51.1°
BC-SAC ($\lambda = 1$)	955.8	83.1°	84°
BC-SAC ($\lambda = 10$)	868.8	77.7°	81.3°
DBC-SAC ($\lambda = 1$, $\alpha = 0.999$)	966.7	86.4°	86.5°
SID-BC-SAC ($\lambda = 1$)	910.7	82°	82.7°
SID-DBC-SAC ($\lambda = 1$, $\alpha = 0.999$)	**966.8**	**86.7°**	**86.7°**
ILD-BC-SAC ($\lambda = 1$)	704.5	62.1°	62.3°
ILD-DBC-SAC ($\lambda = 1$, $\alpha = 0.999$)	614	54.3°	56.2°
DA-SAC	46	5.2°	11°

TABLE II

EVALUATION OF DOOR OPENING ANGLE FREQUENCY FOR EACH MODEL

Model	(Door Angle > 0)%	(Door Angle > 0.2)%	(Door Angle > 1)%	(Door Angle > 1.35)%
SAC	18.2%	4.9%	1%	0.2%
BC	73.6%	64.8%	34.9%	27.6%
BC-SAC ($\lambda = 0.1$)	15.5%	11.8%	10.1%	0%
BC-SAC ($\lambda = 1$)	98.9%	97.5%	92.5%	83.9%
BC-SAC ($\lambda = 10$)	95.6%	93.8%	88.1%	78%
DBC-SAC ($\lambda = 1, \alpha = 0.999$)	99.9%	99.3%	98.2%	94.8%
SID-BC-SAC ($\lambda = 1$)	99.1%	99%	98.2%	93.5%
SID-DBC-SAC ($\lambda = 1, \alpha = 0.999$)	**100%**	**100%**	**99.9%**	**97.1%**
ILD-BC-SAC ($\lambda = 1$)	96.6%	95.1%	42.5%	36.1%
ILD-DBC-SAC ($\lambda = 1, \alpha = 0.999$)	99.7%	96.2%	55.5%	42%
DA-SAC	47.9%	11.7%	0.8%	0%

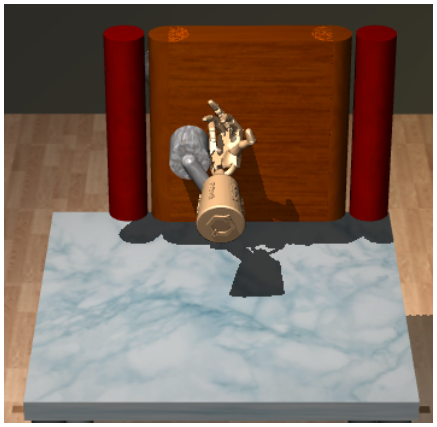

Fig. 4. SAC approaching door using dorsum

This performance is due to the reward function giving a small reward for coming close to the handle, and the SAC model learned that this is the easiest way to approach the handle. Without any guidance from samples, the SAC failed to find the global optima and instead converged to a local minima. The large search space makes it nearly impossible for SAC to find the global optima without any guidance.

2) BC Performance: BC definitely performed better than SAC and is able to open with better success. However, the BC model would often release the door handle too early, preventing the model from fully opening the door, which can be observed in the metrics. Sometimes, it would swipe at the door and completely miss the handle.

The BC model struggles to learn actions on a longer horizon, as there is a larger space of actions to learn from due to accumulating errors. Furthermore, since BC does not have any long-term planning, it would behave erratically once there was enough deviation from the distribution, preventing it from consistently swinging the door open.

3) BC-SAC Performance: BC-SAC showed very promising results. With the right values of λ, BC-SAC is able to fully open the door in 83% of the time, much better than the performance of BC or SAC individually.

With $\lambda = 0.1$, there was not enough emphasis on BC loss, so it still struggled to converge. It learned the same grasping motion as BC but simply attempted to make contact with the handle, rather than actually trying to grab it.

With $\lambda = 1$, BC-SAC performed quite well. It would usually open the door successfully once it grabbed the door handle properly. However, it would sometimes fail to grab with good contact to maintain strong hold. This is likely due

Fig. 5. BC missing handle when grabbing

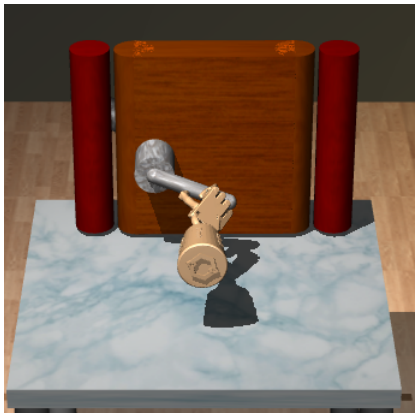

Fig. 6. BC-SAC ($\lambda = 0.1$) grasping and making contact, not grabbing

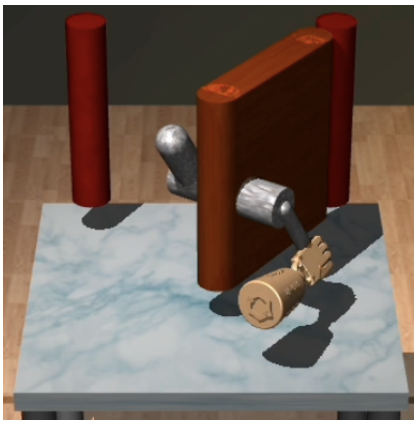

Fig. 7. BC-SAC ($\lambda = 1$) stuck due to bad grab angle

Fig. 8. BC-SAC ($\lambda = 10$) grabbing door knob instead of handle

to BC-SAC trying to balance grasping (from BC loss) with approaching the door handle (RL loss), and the end result is not the best hold on the door.

With $\lambda = 10$, there was too much emphasis on the BC component, which caused the model to let go of the handle at times, similar to what the BC model itself did. However, it still performed fairly well.

Overall, BC-SAC is quite effective in balancing RL rewards with IL loss to obtain a globally optimal policy. However, this policy still has some struggle to generalize to all scenarios. Although it is possible to potentially find the perfect value of λ to reach convergence, that is a tedious process. The methods below will help address this without requiring fine hyper-parameter tuning.

4) DBC-SAC performance: DBC-SAC performed similarly to BC-SAC but was generally more consistent in its performance. It did not get stuck as often as BC-SAC due to bad grasping of the door handles. DBC-SAC learned all the steps of grasping and opening the door: Grab the handle and the correct location, turn the handle $90°$, pull in an arc, and do not let go.

Since DBC-SAC increases the importance of RL exploration over time, it was able to generalize better than BC-SAC. While BC-SAC looks to find a balance between RL reward and BC loss, it always looks for the same middle ground, rather than allowing for the best exploration. DBC-SAC allows RL to fully explore the distribution around BC over enough time, allowing it to learn the subtlety of how to properly grasp the handle.

5) ILD performance: Adding ILDs to both BC-SAC and DBC-SAC actually decreased the performance of both models. ILDs resulted in similar performance in getting the door to open, but performance significantly dropped in actually swinging the door fully open. This is consistent with the original BC model, which also struggled to get the door fully open.

This is likely because the original BC model from which the examples were extracted does not have generalized performance to begin with. Although examples with good rewards were extracted, they may have used a technique that is not consistent or optimal for actually opening the door. This is because BC does not have any long-term planning, so it was only able to open the door to those specific configurations.

6) SID performance: In contrast to the performance of the ILD, the SID helped improve the overall consistency of both models, especially in helping the door fully open.

The general behavior and strategy of both models remained similar, but the extra data helped the BC loss cover a larger amount of the distribution, which in turn gave a

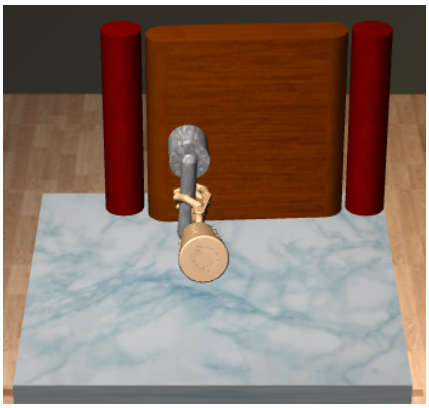

Fig. 9. DBC-SAC effectively grabbing handle and pulling

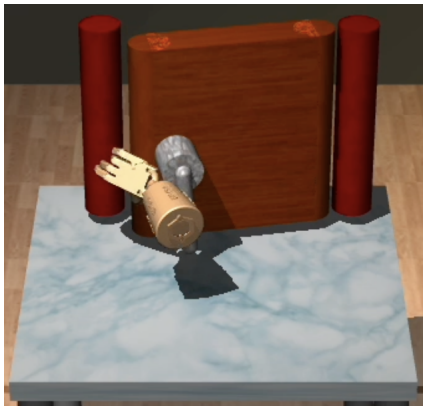

Fig. 10. DA-SAC opening door from left

more generalized model. Specifically, SID especially helped improve the long-term planning needed to fully swing the door open, as more samples helped guide the loss to the true global optima faster.

SID-DBC-SAC perfromed especially well, being the only model that actually opened the door 100% of the time and fully opened the door 97. 1% of the time.

7) DA-SAC performance: DA-SAC developed a rather unusual strategy: It learned to pull the door handle down to unlock the door, but would then choose to let go and pull the door from the left (as seen in Fig. 10). Although the model had some success in opening the door, it was unable to properly swing the door open with this sub-optimal strategy.

It appears that the model learned the first initial aspect of unlocking the door from the BC samples, but was unable to actually converge to the correct solution, and found its own local minima it got stuck in. It is possible that with a smarter method for injecting demonstrations and a different frequency, this issue would not occur.

V. Conclusion

In conclusion, the integration of Reinforcement Learning with human expert demonstrations helped massively improve the performance of the models. Compared to pure Reinforcement Learning, the family of BC-SAC models avoided getting stuck in local minima and quickly converged to the optimal solution. Furthermore, they retained a human-like policy from the given demonstrations while also generalizing much better than a pure BC policy.

The integration of BC loss into the actor objective function, initially proposed by Waymo [10], has been shown to be extremely effective in combining IL and RL. Furthermore, in Discounting λ and Self-Imitation Demonstrations, I introduced two additions that help further improve the performance of their method to generalize better.

In the future, I would be interested to see how Sim2Real transfer of such models would work. Although the BC-SAC policies are definitely better in simulation, they have not been subjected to real world noise, and may struggle with such cases. I also think trying different decay functions for λ could be useful. Exponential decay was a simple and effective method, but other functions may have properties that allow for even better performance. Lastly, I would be interested in further researching DA-SAC and trying to integrate demonstrations in a smarter manner that more closely resembles the SIL paper [22].

References

[1] K. M. Lynch and M. T. Mason, "Dynamic nonprehensile manipulation: Controllability, planning, and experiments," *The International Journal of Robotics Research*, vol. 18, no. 1, pp. 64–92, 1999. [Online]. Available: https://doi.org/10.1177/027836499901800105

[2] M. Gillepsi, L. Davis, S. Jones, and A. Choudhary, "Review of graph-based motion planning algorithms," 08 2024.

[3] L. Zhang, K. Cai, Z. Sun, Z. Bing, C. Wang, L. Figueredo, S. Haddadin, and A. Knoll, "Motion planning for robotics: A review for sampling-based planners," 2024. [Online]. Available: https://arxiv.org/abs/2410.19414

[4] A. Aristidou and J. Lasenby, "Inverse kinematics: a review of existing techniques and introduction of a new fast iterative solver," 09 2009.

[5] S. Adams, T. Cody, and P. A. Beling, "A survey of inverse reinforcement learning," *Artificial Intelligence Review*, vol. 55, no. 6, pp. 4307–4346, 2022.

[6] S. Arora and P. Doshi, "A survey of inverse reinforcement learning: Challenges, methods and progress," 2020. [Online]. Available: https://arxiv.org/abs/1806.06877

[7] F. Torabi, G. Warnell, and P. Stone, "Behavioral cloning from observation," 2018. [Online]. Available: https://arxiv.org/abs/1805.01954

[8] C. Tang, B. Abbatematteo, J. Hu, R. Chandra, R. Martín-Martín, and P. Stone, "Deep reinforcement learning for robotics: A survey of real-world successes," 2024. [Online]. Available: https://arxiv.org/abs/2408.03539

[9] M. Kaspar, J. D. M. Osorio, and J. Bock, "Sim2real transfer for reinforcement learning without dynamics randomization," 2020. [Online]. Available: https://arxiv.org/abs/2002.11635

[10] Y. Lu, J. Fu, G. Tucker, X. Pan, E. Bronstein, B. Roelofs, B. Sapp, B. White, A. Faust, S. Whiteson *et al.*, "Imitation is not enough: Robustifying imitation with reinforcement learning for challenging driving scenarios," *arXiv preprint arXiv:2212.11419*, 2022.

[11] G. Khandate, S. Shang, E. T. Chang, T. L. Saidi, Y. Liu, S. M. Dennis, J. Adams, and M. Ciocarlie, "Sampling-based exploration for reinforcement learning of dexterous manipulation," 2023.

[12] D. Lawson and A. H. Qureshi, "Control transformer: Robot navigation in unknown environments through prm-guided return-conditioned sequence modeling," *arXiv preprint arXiv:2211.06407*, 2022.

[13] J. Achiam, D. Held, A. Tamar, and P. Abbeel, "Constrained policy optimization," 2017. [Online]. Available: https://arxiv.org/abs/1705.10528

[14] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, G. Dulac-Arnold, I. Osband, J. Agapiou, J. Z. Leibo, and A. Gruslys, "Deep q-learning from demonstrations," 2017.

[15] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Overcoming exploration in reinforcement learning with demonstrations," *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6292–6299, 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:3543784

[16] S. M. Kakade, "A natural policy gradient," in *Neural Information Processing Systems*, 2001. [Online]. Available: https://api.semanticscholar.org/CorpusID:14540458

[17] C.-A. Cheng, X. Yan, N. Wagener, and B. Boots, "Fast policy learning through imitation and reinforcement," 2018. [Online]. Available: https://arxiv.org/abs/1805.10413

[18] J. Booher, K. Rohanimanesh, J. Xu, V. Isenbaev, A. Balakrishna, I. Gupta, W. Liu, and A. Petiushko, "Cimrl: Combining imitation and reinforcement learning for safe autonomous driving," 2024. [Online]. Available: https://arxiv.org/abs/2406.08878

[19] B. Thananjeyan, A. Balakrishna, S. Nair, M. Luo, K. Srinivasan, M. Hwang, J. E. Gonzalez, J. Ibarz, C. Finn, and K. Goldberg, "Recovery rl: Safe reinforcement learning with learned recovery zones," 2021. [Online]. Available: https://arxiv.org/abs/2010.15920

[20] S. Ross, G. J. Gordon, and J. A. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," 2011.

[21] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," *CoRR*, vol. abs/1801.01290, 2018. [Online]. Available: http://arxiv.org/abs/1801.01290

[22] J. Oh, Y. Guo, S. Singh, and H. Lee, "Self-imitation learning," 2018. [Online]. Available: https://arxiv.org/abs/1806.05635

[23] "Robohive – a unified framework for robot learning," https://sites.google.com/view/robohive, 2020. [Online]. Available: https://sites.google.com/view/robohive

[24] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 5026–5033.

[25] D. Yarats and I. Kostrikov, "Soft actor-critic (sac) implementation in pytorch," https://github.com/denisyarats/pytorch$_sac$, 2020.