Guided Uncertainty-Aware Policy Optimization: Combining Learning and Model-Based Strategies for Sample-Efficient Policy Learning

Michelle A. Lee [*] [†] **Carlos Florensa** [*] [‡] **Jonathan Tremblay** [§] **Nathan Ratliff** [§]

Animesh Garg [§] **Fabio Ramos** [§] **Dieter Fox** [§]

Abstract

Traditional robotic approaches rely on an accurate model of the environment, a detailed description of how to perform the task, and a robust perception system to keep track of the current state. On the other hand, Reinforcement Learning (RL) approaches can operate directly from raw sensory inputs with only a reward signal to describe the task, but are extremely sample-inefficient and brittle. In this work we combine the strengths of model-based methods with the flexibility of RL methods to obtain a general algorithm that is able to overcome inaccuracies in the robotics perception/actuation pipeline, while requiring minimal training time in the environment. This is achieved by leveraging uncertainty estimates to divide the space in regions where the given model-based policy is reliable, and regions where it may have flaws or not be well defined. In these uncertain regions, we show that a local RL policy can be learned directly from raw sensory inputs. We test our algorithm, Guided Uncertainty-Aware Policy Optimization (GUAPO), on a real-world robot performing tight-fitting peg insertion. Videos are available[5].

1 INTRODUCTION AND RELATED WORKS

There have been great advances in making robots learn to act from raw sensory data leveraging techniques like deep Reinforcement Learning (RL). These methods have successfully been applied to contact-rich manipulation tasks like object insertion, pushing, and grasping (1; 2; 3; 4). However, RL methods tend to be sample inefficient, requiring too many training interactions with the environment to be practical for real world applications. To mitigate this limitation, many prior works carefully tune densely shaped rewards, which often requires explicit knowledge of the state of the world, such as a goal location (1). On the other hand, given a precise model and current state of the world, there are many algorithms to plan, engineer, or search for policies to accomplish the task (5; 6). This kind of model-based (MB) strategies have been shown for many manipulation tasks, such as peg insertion, grasping, and reaching (7; 8; 9; 10). Nevertheless, MB strategies can be crippled by model bias and state estimation errors, and often reach lower asymptotic performance (11; 12).

In this work, we want to combine the sample efficiency from a Model-Based (MB) policy, but overcome the errors in dynamics and perception with a Reinforcement Learning (RL) policy that closes the loop on raw sensory data. Recent works have also combined model-based and RL

[*]Equal contribution as well as work performed during an internship at NVIDIA

[†]Department of Computer Science at Stanford University, mishlee@stanford.edu

[‡]Department of Computer Science, University of California at Berkeley, florensa@berkeley.edu

[§]NVIDIA, [jtremblay, nratliff, animeshg, fabior, dieterf]@nvidia.com

[5]https://sites.google.com/view/guapo-rl

NeurIPS 2019 Workshop on Robot Learning: Control and Interaction in the Real World, Vancouver, Canada

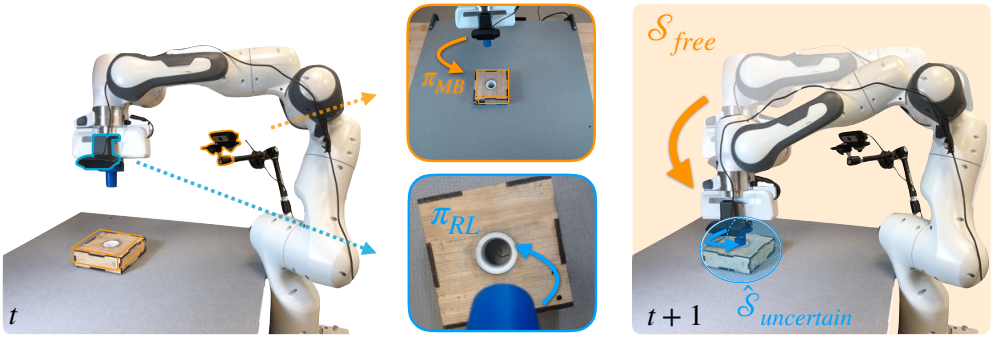


Figure 1: Real-world setup for peg insertion: one perception system (in orange) gives the approximate position of the relevant objects. The *model-based* method drives the system within the uncertainty area (in blue). Once inside this area the model can't be trusted, and a *reinforcement learning* policy is learned directly from the raw sensory inputs of another "local" camera (in blue) that gives enough information to complete the task.

approaches, Johannsmeie *et al.* uses a learning algorithm to find the best parameters that describe the behavior of the agent based on a model-based template (13). The learning is very efficient, but at the cost of an extremely engineered pre-solution that also relies on an accurate perception system. Another line of work that allows to combine model-based and RL methods is Residual Learning (14; 15), where RL is used to learn an additive policy that can potentially fully over-write the original model-based policy. Nevertheless, these methods are hard to tune, and hardly preserve the benefits of the underlying model-based method once trained.

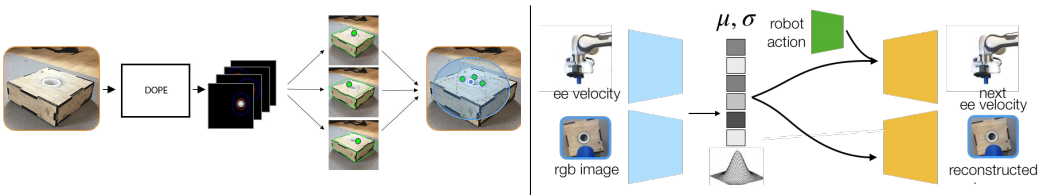
In our work, we first leverage the efficiency of a model-based method to move in a free space environment (6; 16; 17; 18), space where collisions with solid environment or human beings are impossible. Then, we use the capacity of a learning-based method to learn from its environment and a loosely define goal. In order to accomplish such system we also introduce a perception system that can predict pose uncertainties to help the system fuse the MB and RL policies. Figure 1 shows an overview of our system, the task is initialized with a MB policy (π_{MB}) where it moves the robot within the range of the uncertainties of the object of interest, *e.g.*, the box where to insert the peg. Once we have reached the uncertainty region, we switch to a RL policy (π_{RL}) to finish the task. At learning time, we leverage information from π_{RL} to reduce the perception system's uncertainties.

Our system effectively uses uncertainty to mix MB and RL policies to accomplish complicated tasks that require environment interactions which would be quite challenging when either method is used alone. Therefore, we call our algorithm Guided Uncertainty Aware Policy Optimization (GUAPO). We demonstrate our novel algorithm is sample efficient on real world robots on a peg insertion task, and compare our algorithm to learning-based and model-based approaches.

2 PROBLEM STATEMENT

We tackle the problem of learning to perform an operation, unknown *a-priori*, in an area of which we only have an estimated location and no accurate model. We formalize this problem as solving a Markov Decision Process (MDP), where we want to maximize a certain reward $r : \mathcal{S} \to \mathbb{R}$ by learning a policy $\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}_+$ that is a probability distribution over actions $a \in \mathcal{A}$, given a state $s \in \mathcal{S}$. The first assumption we leverage in this work can be expressed as having a partial knowledge of the transition probability $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}_+$ dictating the probability over next states when applying a certain action to the current state. Specifically, we assume this transition is available only within a sub-space of the state-space $\mathcal{S}_{free} \subset \mathcal{S}$. This is a common case in robotics, where it is perfectly known how the robot moves while it is in open-space, but there are no reliable and accurate models of general contacts and interactions with its surrounding. This partial model can be combined with well established methods able to plan and execute trajectories that traverse $\mathcal{S}_{free}$, but these methods are hard to use for successfully completing tasks that require acting in $\mathcal{S}_{uncertain} = \mathcal{S} \setminus \mathcal{S}_{free}$.

The tasks we consider consist on reaching a particular state or configuration through interaction with the environment, like peg-insertion, switch-toggling, or grasping. These tasks are conveniently defined by a binary reward function, $r(s) = \mathbb{1}[s \in \mathcal{S}_g]$, indicating the goal set is reached successfully,



(a) DOPE uncertainty estimation of $\hat{\mathcal{S}}_{uncertain}$ (b) π_{RL} variational autoencoder architecture

Figure 2: Perception modules for the model-based component (left) and learning-based component (right)

$\mathcal{S}_g \subset \mathcal{S}_{uncertain}$. Unfortunately this reward is extremely sparse, and standard exploration strategies from a random initial configuration might take an unreasonable amount of samples to discover it. Furthermore, we do not assume knowledge of the exact location of $\mathcal{S}_{uncertain}$, but rather only access to a noisy estimate of it. Therefore the question we address in this paper is how to leverage the partial model described above to efficiently learn to solve the full task, overcoming an imperfect perception system and dynamics.

3 METHOD

Given that the dynamics are not known precisely around the area where the desired configuration is, a *learning-based method* needs to be used to perform the task. Nevertheless, learning a RL policy from scratch using raw image inputs is extremely inefficient and brittle because it needs to learn how to control the robot everywhere, and every time the position of the goal changes, the task may look completely different. The main insight in our work is that a partial model can still be leveraged with a *model-based method* that only acts in the regions where the model is trusted, hence considerably off-loading the learning process of the RL algorithm and making it more invariant to the absolute goal location.

In this section, we propose a method to generate a super-set $\hat{\mathcal{S}}_{uncertain}$ of $\mathcal{S}_{uncertain}$ based on the perception system uncertainty estimation. Then we describe a MB algorithm that can now confidently be used outside of $\hat{\mathcal{S}}_{uncertain}$ to bring the robot within that set. We also define the parameters of our RL policy, and how the learning can be more efficient by making its inputs local. Finally we present initial experimental results.

From coarse perception to the learning-based workspace. Coarse perception systems are usually cheaper and faster to setup because they might require simpler hardware like RGB cameras, and can be used out-of-the box with excessive tuning and calibration efforts. If we use this system to directly localize $\mathcal{S}_{uncertain}$, the perception errors might misleadingly indicate that a certain area belongs to $\mathcal{S}_{free}$, hence trying to apply the MB policy and potentially not being able to learn how to recover from there. Instead, we propose to use a perception system that also gives an *uncertainty estimate*. For example, many methods can represent the uncertainty by a nonparametric distribution, with n possible poses of the region $\{\mathcal{S}^i_{uncertain}\}_{i=1}^n$ and their associated weights $p(\mathcal{S}^i_{uncertain})$. Interpreting these weights as the likelihoods, we can express the likelihood of a certain state s belonging to $\mathcal{S}_{uncertain}$ as:

$$p(s \in \mathcal{S}_{uncertain}) = \sum_{i=1}^{n} \mathbb{1}[s \in \mathcal{S}^i_{uncertain}] p(\mathcal{S}^i_{uncertain}) \tag{1}$$

We now define $\hat{\mathcal{S}}_{uncertain} = \{s : p(s \in \mathcal{S}_{uncertain}) > \epsilon\}$, where ϵ is set by the user. A more accurate perception system would make $\hat{\mathcal{S}}_{uncertain}$ a tighter super-set of $\mathcal{S}_{uncertain}$, hence further reducing the area where the *learning-based* method is required. Defining $\alpha = \mathbb{1}[s \in \hat{\mathcal{S}}_{uncertain}]$, the overall policy we use is:

$$\pi(a|s) = (1 - \alpha) \cdot \pi_{MB}(a|s) + \alpha \cdot \pi_{RL}(a|s), \tag{2}$$

where $\pi_{MB}(a|s)$ and $\pi_{RL}(a|s)$ are the model-based and learning-based policies respectively. We now detail how each of these policies are obtained.

Model-based method. In the previous section we have defined the region $\hat{\mathcal{S}}_{uncertain}$, the region where we know there is a certain reward to achieve the task, but not how to do so. In our problem statement we assume that outside that region, the environment model is well known, and therefore it is amenable to use a *model-based* approach. Therefore, whenever we are outside of $\hat{\mathcal{S}}_{uncertain}$,

Table 1: Real World Peg Insertion Results out of 30 Trials. The first row indicates percentage of full peg insertion. The second is the speed of insertion (in terms of steps) of the trained policy. The last two indicates how likely is the method to enter $\mathcal{S}_{uncertain}$ and $\hat{\mathcal{S}}_{uncertain}$

	MB-Perfect	MB-DOPE	MB-RA Perfect	MB-RA DOPE	SAC	RESIDUAL (14)	GUAPO (ours)
Success Rate	100%	0%	86.67%	26.6%	0%	0%	93%
Avg. Steps for Task Completion	158.3	n/a	554.1	925.4	n/a	n/a	469.6
In $\mathcal{S}_{uncertain}$	100%	0%	100%	70.0%	0%	0%	100%
In $\hat{\mathcal{S}}_{uncertain}$	100%	100%	100%	93.3%	0%	100%	100%

the model-based approach brings the robot back into it. The most straight-forward way is to pick a specific point within $\mathcal{S}_{uncertain}$, like its centroid, which can be found as the maximum likelihood estimate from the perception system, and set that location as a target for the *model-based* method.

Our formulation can be extended to account for multiple goals or task requirements. For example, if there is an obstacle needs to be avoided, and there is an uncertainty about its location, we can describe $\hat{\mathcal{S}}_{uncertain} = \hat{\mathcal{S}}^{goal}_{uncertain} \sqcup \hat{\mathcal{S}}^{obst}_{uncertain}$, where $\mathcal{S}_g \subset \hat{\mathcal{S}}^{goal}_{uncertain}$ and $r(s) = -1 \; \forall s \in \hat{\mathcal{S}}^{obst}_{uncertain}$. Then an obstacle-avoiding MB policy can be used to get to the area where to goal is while avoiding the regions where the obstacle might be, as shown in our videos[6].

Learning-based method. Once π_{MB} has brought the system within $\hat{\mathcal{S}}_{uncertain}$, the controlled is handed-over to π_{RL} as expressed in Eq. 2. Note that our switching definition goes both ways, and therefore if π_{RL} takes exploratory actions that move it outside of $\hat{\mathcal{S}}_{uncertain}$, then the MB will act again to funnel the state to the area of interest. This also provides a framework for safe learning (19) in case there are obstacles to avoid as introduced in the section above. There are several advantages to having a more restricted area where the RL needs to learn how to act: first the exploration becomes easier, and second the policy can be local. In particular, we only feed to π_{RL} the images from a wrist-mounted camera and its current velocities, as depicted in Fig. 2b. Not having global information like the one provided by the perception system in Fig. 2a can make the RL policy generalize better across locations of $\hat{\mathcal{S}}_{uncertain}$. Finally, we propose to use an off-policy RL algorithm such that all the observed transitions can be added in the replay buffer, no matter if they come from π_{MB} or π_{RL}.

Closing the RL-MB loop. This framework also allows to use any newly acquired experience to reduce $\hat{\mathcal{S}}_{uncertain}$ such that successive rollouts can use the MB policy in larger areas of the state-space. For example, in the peg-insertion task, once the reward of fully inserting the peg is received the location of the opening is immediately known, and therefore we can update $\hat{\mathcal{S}}_{uncertain} = \mathcal{S}_{uncertain}$, where now the RL policy only needs to do the actual insertion and not keep looking for the opening.

Experimental Design. In order to test our algorithm, we set up a peg insertion task in the real world (see Fig. 1), on the Franka Panda robot, a 7-DoF torque-controlled robot. Because our algorithm is agnostic to the model-based algorithm and perception system, the details of our set up can be found in Appendix A. We compare our method (GUAPO) with a model-based policy that has a perfect estimate of where the hole is (MB-Perfect), a model-based policy using our perception system's goal estimate (MB-DOPE), model-based policy with additive Gaussian random action noise with perfect perception (MB-RA-Perfect) and with our perception goal estimate (MB-RA-DOPE), model-free soft-actor critic (SAC), and a residual policy (Residual) (14). GUAPO uses sparse rewards, while SAC and Residual use a dense reward described in Appendix A. For GUAPO, SAC, and Residual, we ran the learning for 60 training episodes (1000 steps per episode), which took roughly 90 minutes to train. To evaluate each method, we ran 30 rollouts per policy, and recorded the success rate in Table 1.

[6]https://sites.google.com/view/guapo-rl

4 DISCUSSION AND CONCLUSIONS

Our preliminary results in the real world show learning of a tight-fitting peg insertion task five times faster than recent prior work (1), by properly leveraging a coarse perception system and a sparse reward.

References

[1] M. A. Lee, Y. Zhu, K. Srinivasan, P. Shah, S. Savarese, L. Fei-Fei, A. Garg, and J. Bohg, "Making Sense of Vision and Touch: Self-Supervised Learning of Multimodal Representations for Contact-Rich Tasks," Tech. Rep. [Online]. Available: https://arxiv.org/pdf/1810.10191.pdf

[2] S. Levine and C. Finn, "End-to-End Training of Deep Visuomotor Policies," vol. 17, pp. 1–40, 2016.

[3] Y. Zhu, Z. Wang, J. Merel, A. Rusu, T. Erez, S. Cabi, S. Tunyasuvunakool, J. Kramár, R. Hadsell, N. de Freitas *et al.*, "Reinforcement and imitation learning for diverse visuomotor skills," *arXiv preprint arXiv:1802.09564*, 2018.

[4] A. Zeng, S. Song, S. Welker, J. Lee, A. Rodriguez, and T. Funkhouser, "Learning synergies between pushing and grasping with self-supervised deep reinforcement learning," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 4238–4245.

[5] T. Schmidt, R. Newcombe, and D. Fox, "DART: Dense Articulated Real-Time Tracking," Tech. Rep. [Online]. Available: https://www.cc.gatech.edu/~afb/classes/CS7495-Fall2014/readings/dart.pdf

[6] C.-A. Cheng, M. Mukadam, J. Issac, S. Birchfield, D. Fox, B. Boots, and N. Ratliff, "RMPflow : A Computational Graph for Automatic Motion Policy Generation," Tech. Rep. [Online]. Available: https://arxiv.org/pdf/1811.07049.pdf

[7] K. Van Wyk, M. Culleton, J. Falco, and K. Kelly, "Comparative Peg-in-Hole Testing of a Force-Based Manipulation Controlled Robotic Hand," *IEEE Transactions on Robotics*, vol. 34, no. 2, pp. 542–549, 2018.

[8] C. H. Kim and J. Seo, "Shallow-Depth Insertion: Peg in Shallow Hole Through Robotic In-Hand Manipulation," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 383–390, 4 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8598749/

[9] G. Thomas, M. Chien, A. Tamar, J. A. Ojea, P. Abbeel, and R. O. Mar, "Learning Robotic Assembly from CAD," *International Conference on Robotics and Automation*, 2018.

[10] C. Borst, M. Fischer, and G. Hirzinger, "A fast and robust grasp planner for arbitrary 3d objects," in *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C)*, vol. 3. IEEE, 1999, pp. 1890–1896.

[11] T. Wang, X. Bao, I. Clavera, J. Hoang, Y. Wen, E. Langlois, S. Zhang, G. Zhang, P. Abbeel, and J. Ba, "Benchmarking model-based reinforcement learning," *CoRR*, vol. abs/1907.02057, 2019. [Online]. Available: http://arxiv.org/abs/1907.02057

[12] M.-f. Fine-tuning, A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, "Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning."

[13] L. Johannsmeier, M. Gerchow, and S. Haddadin, "A framework for robot manipulation: Skill formalism, meta learning and adaptive control," *arXiv preprint arXiv:1805.08576*, 2018.

[14] T. Johannink, S. Bahl, A. Nair, J. Luo, and A. Kumar, "Residual Reinforcement Learning for Robot Control," pp. 1–9.

[15] T. Silver, K. Allen, J. Tenenbaum, and L. Kaelbling, "Residual Policy Learning," 2018. [Online]. Available: https://k-r-allen.github.io/residual-policy-learning/.http://arxiv.org/abs/1812.06298

[16] S. Chitta, I. Sucan, and S. Cousins, "Moveit![ros topics]," *IEEE Robotics & Automation Magazine*, vol. 19, no. 1, pp. 18–19, 2012.

[17] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[18] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "Chomp: Gradient optimization techniques for efficient motion planning," 2009.

[19] J. F. Fisac, A. K. Akametalu, M. N. Zeilinger, S. Kaynama, J. Gillula, and C. J. Tomlin, "A General Safety Framework for Learning-Based Control in Uncertain Robotic Systems," 5 2017. [Online]. Available: http://arxiv.org/abs/1705.01292

[20] J. Tremblay, T. To, B. Sundaralingam, Y. Xiang, D. Fox, and S. Birchfield, "Deep object pose estimation for semantic robotic grasping of household objects," *arXiv preprint arXiv:1809.10790*, 2018.

[21] T. To, J. Tremblay, D. McKay, Y. Yamaguchi, K. Leung, A. Balanon, J. Cheng, and S. Birchfield, "NDDS: NVIDIA deep learning dataset synthesizer," 2018, https://github.com/NVIDIA/Dataset_Synthesizer.

[22] R. Martín-Martín, M. A. Lee, R. Gardner, S. Savarese, J. Bohg, and A. Garg, "Variable impedance control in end-effector space: An action space for reinforcement learning in contact-rich tasks," *arXiv preprint arXiv:1906.08880*, 2019.

[23] N. D. Ratliff, J. Issac, D. Kappler, S. Birchfield, and D. Fox, "Riemannian Motion Policies," 1 2018. [Online]. Available: http://arxiv.org/abs/1801.02854

[24] T. Haarnoja, A. Zhou, P. Abbeel, S. Levine, and C. Sciences, "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," *Internation Conference in Machine Learning*, pp. 1–15, 2018.

[25] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, A. Lerchner, and G. Deepmind, "β-VAE: LEARNING BASIC VISUAL CONCEPTS WITH A CONSTRAINED VARIATIONAL FRAMEWORK," *International Conference in Learning Representations*, pp. 1–22, 11. [Online]. Available: https://openreview.net/forum?id=Sy2fzU9gl

A Implementation details

A.1 Perception Module

On the real robot system, we use the Deep Object Pose Estimator (DOPE) (20) as our perception system. DOPE uses a simple neural network architecture that can be quickly trained with synthetic data and domain randomization using NDDS (21). The algorithm first finds the object cuboid keypoints using a local peak on the map. With the cuboid real dimensions, camera intrinsics, and the keypoint locations, DOPE runs the PnP algorithm to find the final object pose in the camera frame. For this work we extended DOPE perception system to predict uncertainty estimates with the object pose. This extension augments the peak estimation algorithm by fitting a 2d gaussian around the found peak. We then run the PnP algorithm multiple times where each keypoint is sampled from its respective 2d gaussian. This gives us the pose of the object, as well as an uncertainty estimate with each dimension of the pose as depicted in Fig. 2a. We set the camera for DOPE mounted on top of our workspace, as seen in Fig. 1.

A.1.1 Model-based Controller Design

We use target attractors with impedance control in end-effector space, an action space which has been shown to improve sample efficiency for policy learning (22). Specifically on the real robot, we use target attractors defined by Riemannian Motion Policies (RMPs) (23) to move the robot towards a desired end-effector location. The RMPs take in a desired end-effector position $\mathbf{x}_x \in \mathbb{R}^3$ in Cartesian space. Both MB and RL policies are sending end-effector position commands at 20Hz. The RMPs are computing desired joint positions $\mathbf{q}_d$ at 1000Hz.

For the peg insertion task, the pure model-based policy goes to the goal location and once it reaches within a threshold (the L2 distance of 0.005cm), the policy pushes down to attempt to insert.

A.1.2 Learning-based Policy Control

We use a state-of-the-art model-free off-policy Reinforcement Learning (RL) algorithm, Soft Actor Critic (24). The RL policy acts directly from raw sensory inputs. This consists on joint velocities and images from a wrist-mounted camera (64x64x3 RGB images from a Logitech Carl Zeiss Tessar) on the robot (see Fig. 1). As illustrated in Fig. 2b, all inputs are fed into a β-VAE (25), which gives us a low-dimensional latent-space representation of the state. The parameters of this VAE are trained before-hand on a data-set collected off-line. The only part that is learned by the RL algorithm is a 2-layer MLP that takes as input the 64-dimensional latent representation given by the VAE, and produces 3D position displacement $\Delta\mathbf{x}$ of the robot end-effector.

A.2 Rewards

For our GUAPO policy, we use a sparse reward when the policy finishes the task (inserts the peg). The policy gets -1 everywhere, and 0 when it finishes the task. For SAC and Residual, we use a negative L2 norm to the perception estimate of the goal location, a 0 reward when it reaches $\hat{\mathcal{S}}_{uncertain}$, and 1 when it finishes the task.