Learning Attentive Neural Processes for Planning with Pushing Actions

Atharv Jain[1], Seiji Shaw[1], Nicholas Roy[1]

Abstract— Our goal is to enable robots to plan sequences of tabletop actions to push a block with unknown physical properties to a desired goal pose. We approach this problem by learning the constituent models of a Partially-Observable Markov Decision Process (POMDP), where the robot can observe the outcome of a push, but the physical properties of the block that govern the dynamics remain unknown. A common solution approach is to train an observation model in a supervised fashion, and do inference with a general inference technique such as particle filters. However, supervised training requires knowledge of the relevant physical properties that determine the problem dynamics, which we do not assume to be known. Planning also requires simulating many belief updates, which becomes expensive when using particle filters to represent the belief. We propose to learn an Attentive Neural Process that computes the belief over a learned latent representation of the relevant physical properties given a history of actions. To address the pushing planning problem, we integrate a trained Neural Process with a double-progressive widening sampling strategy. Simulation results indicate that Neural Process Tree with Double Progressive Widening (NPT-DPW) generates better-performing plans faster than traditional particle-filter methods that use a supervised-trained observation model, even in complex pushing scenarios.

I. Introduction

In scenarios where robots operate in unstructured, dynamic environments, planning under uncertainty is paramount. Some relevant properties for manipulation, such as object geometry, can be easily observed through sensors. Many other properties, such as inertial properties and friction, require interaction with the object [1]. The robot may not even know what the relevant physical properties are to determine action outcomes. One way robots can learn these inertial properties is to iteratively interact with objects and observe their behavior.

We focus on an instance of this problem where a robot must compose a sequence of pushes to guide a block from a starting configuration to a goal configuration, without prior knowledge of the center of mass. Our goal is to develop a technique capable of inferring the true center of mass while simultaneously predicting outcomes of pushing actions.

We model our planning challenge as a Partially Observable Markov Decision Process (POMDP), where unobservable physical properties remain constant over time [2]. Many POMDP solutions rely on Bayesian state estimation techniques (e.g., Kalman or particle filters) to update their beliefs as new observations become available. However, the linear assumptions required for Kalman filters do not hold in this scenario, since varying the center of mass can nonlinearly

influence rotational outcomes. More general methods, such as particle filters, must be used, despite their high computational cost [3].

To leverage these particle-based belief representations to solve our planning problem, a common method is to use *Particle Filter Trees with Double Progressive Widening* (PFT-DPW), a Monte Carlo tree search algorithm that performs a randomized search to identify the plan yielding the highest expected reward. In POMDP planning, it is crucial to balance exploration and exploitation [4]. PFT-DPW and similar Monte Carlo tree search methods manage this trade-off effectively by systematically exploring uncertain aspects of the system while exploiting known information to achieve high rewards [5].

However, PFT-DPW also has significant limitations due to the use of a particle filter. First, training an observation model used to update the particle filter requires knowing what the relevant physical properties of the object for the task dynamics are. Second, the runtime cost of updating a particle belief scales linearly with the number of particles when searching for long-horizon plans.

To overcome these problems, we propose a model called the Pushing Neural Process (PNP), which jointly trains an inference network that estimates physical parameters, and a prediction network that learns to output a distribution of push outcomes based on physical properties. The inference network and prediction network are set up in an encoder-decoder architecture, where a representation of the physical properties is learned in the intermediate latent space. Notably, the PNP does not require any information about which physical properties are relevant to the problem, but instead encodes any relevant information about the representation into a multivariate gaussian outputted by the encoder.

To compare the performance of both models, we derive a new planner that integrates the PNP, called NPT-DPW (Neural Process Tree with Double Progressive Widening). NPT-DPW replaces the particle filter used to maintain belief in PFT-DPW with a neural network that consumes the history of a given search and learns a representation for the object's unknown physical properties. Our results show that, on average, NPT-DPW reaches up to twice as close to the goal location as PFT-DPW and avoids catastrophic failures (such as pushing into prohibited zones) far more often. We additionally demonstrate that the PNP's latent space embeds a representation of the true COM by showing that the model becomes more accurate with greater amounts of previous pushes.
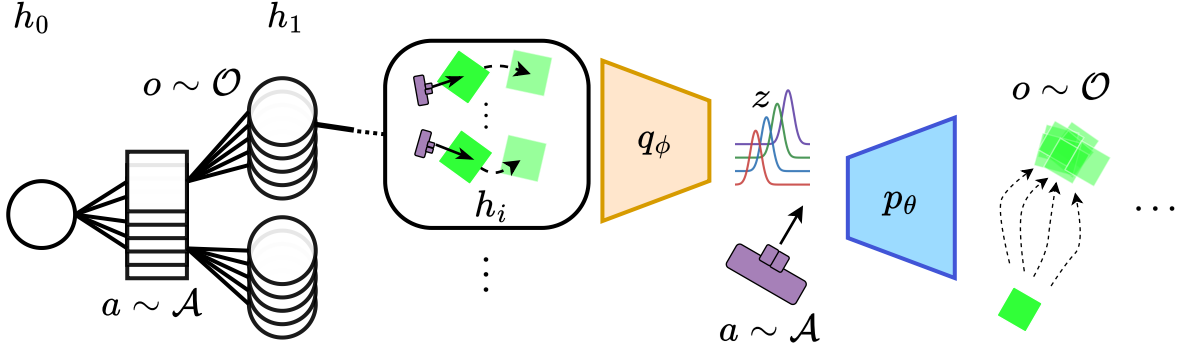
Fig. 1. The Push Neural Process architecture and its use as a subroutine within NPT-DPW. On the right of the image, the search tree over actions and outcomes is pictured, where at each step an action is selected. We then pass in the history of outcomes (h) that we have already observed into the encoder (q_ϕ) of the model to obtain the distribution over the latent parameters, which we sample to get (z). We then pass z to the decoder with our new action a to get a distribution over outcomes. We sample from that distribution as an outcome (o) for the action a. We then add that outcome to the potential outcomes in the search tree.
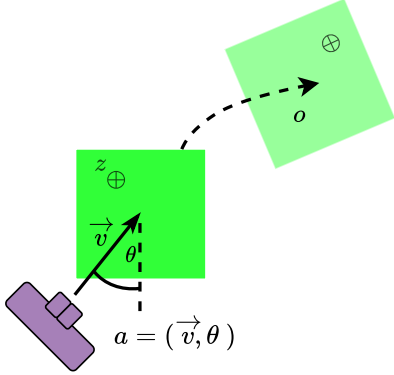

Fig. 2. A depiction of a push action $a \in \mathscr{A}$, parameterized by push angle $\theta \in [0, 2\pi)$ and velocity $\vec{v} \in \mathbb{R}^2$. The center of mass z (denoted as $\oplus$) is not directly observed.

II. PROBLEM SETTING

For the objects we aim to push, we categorize their properties into two distinct sets: observable properties, denoted by $x \in \mathscr{X} \subset \mathbb{R}^n$, and latent physical properties, denoted by $z \in \mathscr{Z} \subset \mathbb{R}^m$, where m is decided empirically based on what results in the strongest model. The observable properties x include features such as the block's resting orientation, while z captures physical properties such as center of mass. These latent properties are unknown to the robot *a priori* but play a critical role in determining the outcome of a push. We define a push in terms of an action $a \in \mathscr{A}$, characterized by an approach angle and a push velocity, where the angle is in the range $[0, 2\pi)$. Upon completing a push, the robot observes the resulting pose of the block $o \in SE(2)$. A schematic of the push action is depicted in Figure 2.

Within our Partially Observable Markov Decision Process (POMDP) framework, the *transition function T* models the object's dynamics. It maps the current state—which encompasses both the object's observable pose and its latent physical properties—and an action to a probability distribution over future object poses. Importantly, the object's pose remains observable to the robot at all times, while the latent properties remain constant throughout the interaction.

Consequently, our *observation function O* simply reflects the observable portion of the state provided by the transition function T. The *reward function R* penalizes the robot proportionally to its distance from the target goal, offering a substantial reward upon successfully reaching the goal (see Section IV-A, Equation 1 for explicit details). A discount factor γ further encourages shorter plans aimed at efficiently reaching the goal.

Collectively, we formulate our pushing task as a POMDP defined by the tuple

$$(\mathscr{S}, \mathscr{A}, \Omega, T, O, R, \gamma)$$

where $\mathscr{S} = \mathscr{X} \times \mathscr{Z}$ represents all possible states and $\omega \in SE(2)$ represents all possible observations.

III. METHODS

A. Pushing Neural Process

To solve the POMDP, we require probabilistic models that describe transitions in the partially observable state of the system and the observations made by the robot. We use an Attentive Neural Process, which we call a *Pushing Neural Process*, to jointly model push state estimation and prediction [6]. This model consists of two primary components: an *encoder* that infers latent representations from historical data, and a *decoder* that predicts future push outcomes given these representations.

We denote a sequence of pushes and their outcomes as the dataset:

$$D_t = \{(a_1, o_1), \ldots, (a_t, o_t)\}.$$

The encoder consumes D_t (of arbitrary length) along with the observable properties x (object geometry) and infers a probability distribution over the latent parameters z. Although the latent space is uninterpretable, we hypothesize that the model will implicitly learn what useful information can be learned from pushing. To this end, we train a neural network, parameterized by ϕ, to produce an approximate posterior $q_\phi(z|x,D_t)$ that closely matches the true posterior $p(z|x,D_t)$.

Given the latent representation from the encoder, the decoder—parameterized by θ—then predicts push outcomes. Mathematically, prediction of a new outcome o given action a, object properties x, and history D_t can be expressed using the law of total probability:

$$p(o|a,x,D_t) = \int p_\theta(o|a,x,z)p(z|x,D_t)\,dz.$$

Algorithm 1 NPT-DPW

---

procedure SIMULATEACTION(h,a)
    $b \leftarrow$ PNPENCODER(h)   ▷ PNP inference step.
    $p \leftarrow$ PNPDECODER(b, a)   ▷ PNP prediction step.
    $o \leftarrow p.sample()$
    $r \leftarrow$ Reward($o.x, o.y$)   ▷ As defined in Equation 1
    return o,r
end procedure
procedure SIMULATE(h,d)
    if $d = 0$ **then**
        return 0
    end if
    $a \leftarrow$ ACTIONPROGWIDEN(h)
    if $|C(ha)| \leq N(ha)^{\alpha_0}$ **then**
        $o,r \leftarrow$ SIMULATEACTION(h,a)
        $C(ha) \leftarrow C(ha) \cup \{o\}$
        $total \leftarrow r + \gamma \cdot$ROLLOUT($hao, d-1$)
    else
        $o,r \leftarrow$ sample from $C(ha)$
        $total \leftarrow r + \gamma \cdot$SIMULATE($hao, d-1$)
    end if
    return $total$
end procedure

Since a single sample from $p(z|x,D_t)$ is an unbiased estimator of the distribution, we sample a single $z \sim q_\phi(z|x,D_t)$. The decoder consumes z, the observable object properties, and the new push to predict a distribution over the observation (block displacement) $p_\theta(o|a,x,z)$.

During training, we maintain a collection of pushes for each object in a set of objects with varying physical properties. We provide the partial dataset D_t as input to the inference network (encoder), and apply the prediction network (decoder) to the collection of *all* pushes D_T for the given object:

$$D_T = D_t \cup \{(a_{t+1}, o_{t+1}), \ldots, (a_T, o_T)\}.$$

We optimize the Evidence Lower Bound (ELBO)

$$\mathbb{E}\Big[\log p_\theta(o|x,a,z) - D_{KL}\big(q_\phi(z|D_T) \,\|\, q_\phi(z|D_t)\big)\Big],$$
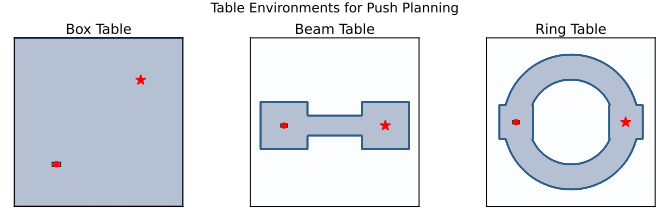

Fig. 3. The pushing surfaces used in our experiments. The rectangle denotes the starting position, and the star the goal.

which promotes accurate push outcome predictions while reducing the discrepancy between the full and partial posterior distributions. The cross-entropy term increases the likelihood of accurately predicting outcomes over time, and the KL term mitigates overfitting.

B. NPT-DPW

Our planner is an extension of PFT-DPW introduced by Sunberg and Kochenderfer [5]. In PFT-DPW, a particle filter is used to represent the set of hypotheses over the partially observable state of the problem, and the belief is updated as new push outcomes are observed. The PF is used as a belief state in Monte Carlo tree search, which branches at new actions and observations. Since both the action and observation spaces are continuous, double progressive widening is used to limit the number of actions and observations sampled at each tree node to ensure reliable reward estimates.

Instead of using a particle filter, our planner, NPT-DPW, uses the Pushing Neural Process to infer the underlying physical parameters and predict outcomes of future actions. To expand the tree, we pass the history of actions and outcomes through the encoder to infer the physical parameters. This history of actions implicitly represents a belief of the true properties, as the history of actions can be passed into the encoder to generate a distribution of the true physical properties.

Actions are sampled as they normally would be in PFT-DPW. Observations are sampled from the distribution computed by the decoder from the inferred parameters and the push under consideration. Thus, every step of the tree search will have runtime complexity of $O(h^2)$, where h is the size of the history (due to the quadratic complexity of self-attention), rather than $O(n)$, where n is the number of particles used in PFT-DPW. We hypothesize a significant improvement in plan quality within a fixed computational budget, since $h \ll n$ in practice. As stated previously, an additional advantage of this method is that it works in general scenarios where the relevant properties are not known *a priori* to the robot.

The NPT-DPW algorithm is described in Algorithm 1; any functions not described in this block are the same as those specified for PFT-DPW in Sunberg and Kochenderfer [5]. Furthermore, a diagram of the Neural Process and its integration into the planner can be seen in Figure 1.

IV. EXPERIMENTS

In our experiments, we seek to evaluate the performance of NPT-DPW relative to the baseline method PFT-DPW, given a
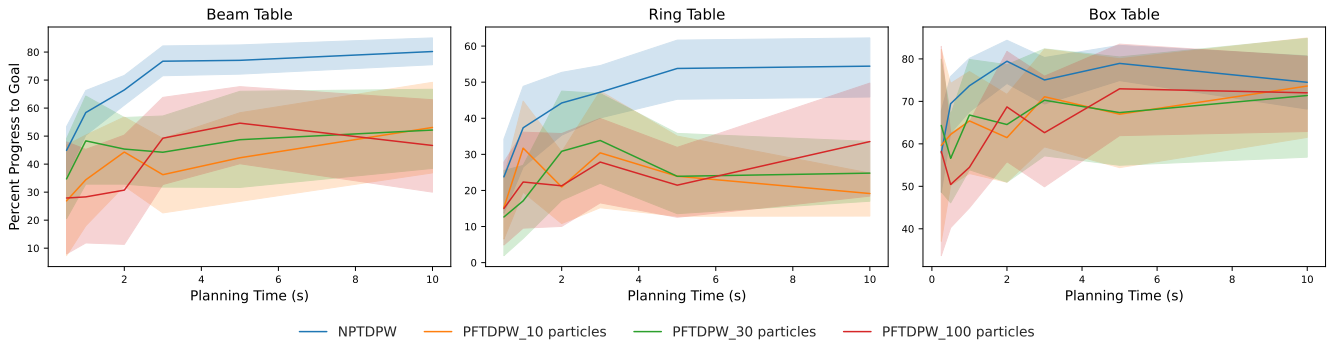
Fig. 4. This figure demonstrates the effectiveness of the NPT-DPW versus PFT-DPW (with 10/30/100 particles). Additionally, we conducted trials with 0.5,1,2,3,5, and 10 seconds of time to plan per step. The results plot the average percentage traveled towards the goal as a function of planning time allocated per action. For example, if the block started 3 meters away, and was pushed to being 1 meter way, that would be considered 66% progress. Fallen blocks are marked with the percentage distance corresponding to when they fell off. Therefore, the percentage from the goal varies a lot, since times the block will fall off almost immediately will be marked with an approximately 0% success.

fixed computational runtime budget. The problem scenarios are designed to evaluate whether an MCTS-DPW planning scheme can leverage the efficient inference updates of the Neural Process for a deeper tree search, yielding better plans.

A. Scenarios

The experiments will be run using three tables with geometry designed to vary the hardness of the pushing problem (Figure 3). Each of these tables defines a different planning problem. The first is a blank-surface table designed to check whether the planner can produce plans that drive the block to the goal. In the second and third scenarios, the movement of the block must be constrained along certain paths to prevent it from falling off the pushing surface. The second table requires pushing in a straight line, while the third requires a push trajectory along a curve.

Solving these instances requires the planner to accurately infer the object's physical properties. For the second and third tables, an intelligent planner must sequence information-gathering actions on a wider area before pushing the block along the narrow sections of these tables.

After each push, the robot obtains a reward for the block's location (x,y) relative to the goal (g_x, g_y). We define the reward function

$$
\begin{aligned}
\text{Reward}(x,y) = &-10\sqrt{(x-g_x)^2 + (y-g_y)^2} \\
&-100\cdot\texttt{fallen}(x,y) \\
&+10000\cdot\texttt{inRange}(x,y) \quad (1)
\end{aligned}
$$

`fallen` is a function that returns 1 if the block is off the table, and 0 otherwise. `inRange` is a function that is 1 if the block is within 0.2 meters of the goal, and 0 otherwise. We set the discount factor $\gamma = 0.8$ to incentivize the planner to prioritize not push the block off the table. An example of a plan generated by NPT-DPW is depicted in Fig. 5. While the metric used as the model to plan will be the reward function, we do not evaluate performance based on that, but instead use percent progress towards the goal, as can be seen later in the results section.

We assess performance by fixing the computation time t allowed at each step of the plan. Then for each t in $[0.5, 1, 2, 3, 5, 10]$ seconds, we ran 15 trials on each map for each planner, where we recorded the distance of the object from the goal after 30 steps. If the object fell off the table, or reached the goal early, we stopped then. We sampled a new center of mass for the block in each trial. For each tree search, we looked for the best action for depth equaling 3, and we only stored the first 10 actions for NPT-DPW, since empirically that was enough to gain a strong belief over the block.

We also varied the number of particles in PFT-DPW, initializing it with $n = 10, 30$, or 100, to evaluate NPT-DPW against a range of particle filter–based planners and assess the impact of particle count on performance. Additionally, while maintaining the particle filter, if the weight associated with a particle became smaller than 10^{-9}, then we turn off that particle, which should encourage the particle filter to become faster as plans become deeper, since low weighted particles will no longer be counted.

These simulations are run in the PyBullet simulator. We implement an end-effector PD-controller to ensure the gripper's end-effector motion velocity is consistent throughout the motion. To sample actions for progressive widening, we uniformly sample an angle from $[0, 2\pi]$ in world coordinates and push at the velocity of 0.10 m/sec. Each of the blocks is a rectangular prism with a fixed geometry of 2.5 cm x 2.5 cm x 1.5 cm. The mass of the block is set constant at 0.2 kg. The center of mass of the block is sampled uniformly within the convex hull of the object's geometry.

Our neural network architecture is based on Attentive Neural Processes [6]. Each push is first encoded by an MLP and processed through multiple layers of self-attention. After that, the outputs of the self-attention layers are aggregated into outputs consisting of a mean and covariance. These represent the mean and diagonal covariance of a multivariate Gaussian over z, representing the distribution over latent physical properties. Note that for our problem, the only latent physical properties are the x and y coordinates of the center
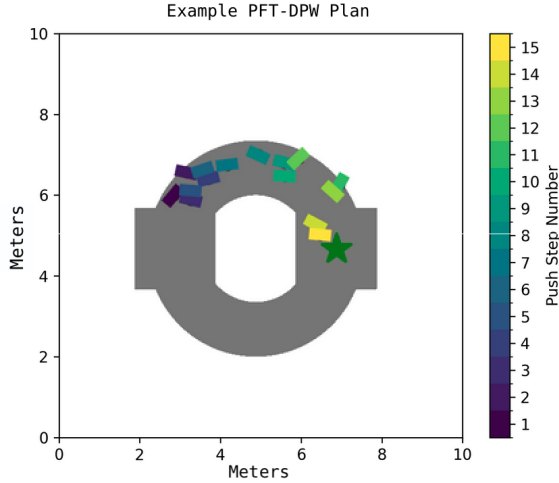
Fig. 5. This is an example of a plan created by NPT-DPW and run by the simulation on the "ring" map. Important features that show signs of intelligent planning are the pushes in the beginning in the corner, signifying learning more about the block before trying to move towards the goal. Note the figure has been re-scaled slightly from the real experiments to be more visually clear.
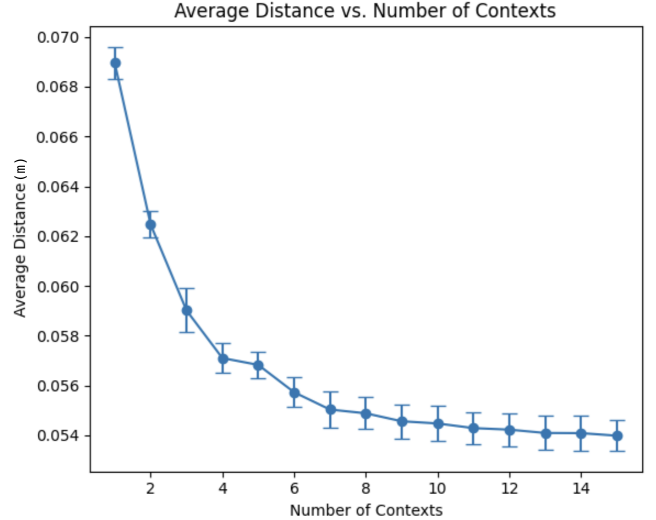

Fig. 6. This plot shows how the average error of the predicted location of the block from the Push Neural Process to the true result of the location of that push decreases as the model receives more past interactions, or context pushes. The error bars represent an interval of one standard error around the mean distance.

of mass, but we still set $z \in \mathbb{R}^5$ since it empirically led to improved performance. The decoder is structured as multiple linear layers that consume a sample from the latent space and outputs a distribution over possible outcome poses of the block after the push is finished.

TABLE I

NUMBER OF SIMULATIONS DONE WITH DIFFERENT AMOUNTS OF PLANNING TIME, AVERAGED OVER 10 PLANS

Planner	1 s	5 s	10 s
NPTDPW	325	2291	4807
PFTDPW with 10 Particles	158	807	1910
PFTDPW with 30 Particles	63	646	1410
PFTDPW with 100 Particles	30	322	977

B. Results

Our results, shown in Figure 4, highlight the effectiveness of using a learned latent space for efficient planning. NPT-DPW consistently outperforms PFT-DPW across multiple scenarios when allocated at least one second per action. This performance gain suggests that the latent space learned by the Attentive Neural Process provides a compact yet expressive understanding of the block's unknown physical properties, which NPT-DPW was able to leverage to achieve greater success planning.

Figure 6 demonstrates that knowing the result of previous interactions with a block (called "context pushes") enables the robot to make more accurate pushes, since the robot is able to process those previous outcomes in the encoder. Because of this, we can infer that the PNP's latent space is able to learn a representation of the block's center of mass.

In addition, maintaining belief through the latent space reduces the computational overhead of having runtime pro-

portional to the number of particles of explicit particle representations, which is what we hypothesize allows NPT-DPW to more rapidly explore the space of potential action sequences than PFT-DPW. In specific, in Table I, we have shown that the number of planning iterations the PNP can process is significantly higher than the particle filter even when the number of particles is 10.

While all these results are extremely promising, there are a few caveats. In scenarios where very limited planning time (e.g., half a second per action) was available, both models struggled similarly, underscoring the necessity for adequate computation time to fully leverage the benefits of learned latent spaces. NPT-DPW, however, starts out performing PFT-DPW when the amount of computation becomes greater than a second.

Another interesting result of the data is the lack of difference in performance between the varied number of particles in PFT-DPW. We hypothesize this is due to the nature of the pushing task, where subtle differences in the center of mass have limited impact because push outcomes primarily depend on the relative positions of the center of mass and the point of contact. As a result, a small number of particles might already capture most of the informative uncertainty, while increasing particle counts does not significantly enhance prediction quality but rather increases computational load, limiting the planner's capacity for deeper exploration.

Our findings reinforce the potential of latent spaces in the Attentive Neural Process to maintain a useful learned representation of the unobservable properties of an object. Additionally, we showed that using the Attentive Neural Process in a planner leads to significantly improved performance compared to PFT-DPW. These results indicate promising directions for future exploration into richer representation

learning, aiming to enhance generalization and robustness in robotic planning tasks.

V. Related Work

Our pushing POMDP is an instance of a Hidden-Parameter POMDP [2], where the unobserved state variables remain static but influence the transition dynamics of the planning problem. Several previous works have introduced Bayesian models for rapid estimation and adaptation of the unobserved state in this class of problems. The difference between these methods largely depends on the representation of uncertainty. Killian et al. [7] use Bayesian neural nets; others use Gaussian processes [2], [8]. Ensembles of neural networks are a valid choice as well [9].

Other work has focused on learning object dynamics with unobservable object properties outside the POMDP framework. Many authors develop ways to estimate unobserved properties when the dynamics are known [10], [11], [12], [13]. Other works, like our method, do not assume the dynamics are known [14], [15], [16], [17]. However, these works assume they have access to a dataset of interactions from a new instance of the system to estimate the unobserved variables.

Estimating unobserved properties through interaction is a common strategy in many manipulation tasks [1]. These strategies are applied to adapt grasping classifiers [18], [19], [20], [21], [22], disentangle object geometry without visual input [23], [24], and obtain object articulation constraints [25], [26]. Bauza and Rodriguez [27] develop an alternate method to compute uncertainty of push outcomes based on uncertainty of physical parameters of known relevance. Many of these methods tackle information gathering by developing problem-specific solutions [21], [22], or relying on bandit-style exploration approaches [19], [20] or information-gain heuristics [18], [24] for short, single-step estimation tasks. Our work leverages POMDP solvers to tackle the information-gathering problem when multiple actions must be composed together.

VI. Conclusion

We present: the Pushing Neural Process, an Attentive Neural Process architecture to learn push outcome prediction of a novel object instance, and NPT-DPW, a new planner that integrates the PNP to produce good plans under reasonable computational runtime budgets. Additionally, we demonstrated a method of learning a representation of an object's properties, and how we can leverage that representation to maintain belief with a general use case planner.

Our initial experiments demonstrate promising capabilities; however, we acknowledge certain limitations that we aim to address in subsequent research. For instance, the PNP exhibited underfitting, which we attribute to the complexity and inherent instability of the simulation environment used to generate the training data. To address this, we hope to apply the NPT-DPW to a broader spectrum of tasks, including scenarios with simpler dynamics and transition models to yield clearer insights. Secondly, we observed occasional erratic behavior in generated plans, indicating opportunities for further model improvement. Addressing these issues may improve planning performance in both PFT-DPW and NPT-DPW. Finally, we hope to deploy the PNP on a physical robot platform to comprehensively validate our approach and demonstrate its applicability.

ACKNOWLEDGMENTS

References

[1] J. Bohg, K. Hausman, B. Sankaran, O. Brock, D. Kragic, S. Schaal, and G. S. Sukhatme, "Interactive Perception: Leveraging Action in Perception and Perception in Action," *IEEE Transactions on Robotics*, vol. 33, no. 6, pp. 1273–1291, Dec. 2017.

[2] F. Doshi-Velez and G. Konidaris, "Hidden parameter markov decision processes: A semiparametric regression approach for discovering latent task parametrizations," in *Proceedings of the International Joint Conferences on Artificial Intelligence*, 2016.

[3] L. Zhang and J. C. Trinkle, "The application of particle filtering to grasping acquisition with visual occlusion and tactile sensing," in *Proceedings of the IEEE International Conference on Robotics and Automation*, May 2012, pp. 3805–3812.

[4] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, no. 1, pp. 99–134, 1998. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S000437029800023X

[5] Z. Sunberg and M. Kochenderfer, "Online Algorithms for POMDPs with Continuous State, Action, and Observation Spaces," *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 28, pp. 259–263, Jun. 2018. [Online]. Available: https://ojs.aaai.org/index.php/ICAPS/article/view/13882

[6] H. Kim, A. Mnih, J. Schwarz, M. Garnelo, A. Eslami, D. Rosenbaum, O. Vinyals, and Y. W. Teh, "Attentive Neural Processes," in *NeurIPS Workshop on Bayesian Deep Learning*, Sep. 2018. [Online]. Available: https://openreview.net/forum?id=SkE6PjC9KX

[7] T. W. Killian, S. Daulton, G. Konidaris, and F. Doshi-Velez, "Robust and Efficient Transfer Learning with Hidden Parameter Markov Decision Processes," in *Proceedings of the 31st Conference on Neural Information Processing Systems*, 2017.

[8] S. Sæmundsson, K. Hofmann, and M. P. Deisenroth, "Meta reinforcement learning with latent variable gaussian processes," in *Uncertainty in Artificial Intelligence*, 2018.

[9] S. Belkhale, R. Li, G. Kahn, R. McAllister, R. Calandra, and S. Levine, "Model-Based Meta-Reinforcement Learning for Flight with Suspended Payloads," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, 2021.

[10] P. W. Battaglia, J. B. Hamrick, and J. B. Tenenbaum, "Simulation as an engine of physical scene understanding," *Proceedings of the National Academy of Sciences*, vol. 110, no. 45, 2013.

[11] J. Wu, I. Yildirim, J. J. Lim, B. Freeman, and J. Tenenbaum, "Galileo: Perceiving Physical Object Properties by Integrating a Physics Engine with Deep Learning," in *Advances in Neural Information Processing Systems*, 2015, p. 9.

[12] J. Wu, J. Lim, H. Zhang, J. Tenenbaum, and W. Freeman, "Physics 101: Learning Physical Object Properties from Unlabeled Videos," in *Procedings of the British Machine Vision Conference 2016*, 2016.

[13] L. Zhang and J. C. Trinkle, "The application of particle filtering to grasping acquisition with visual occlusion and tactile sensing," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2012.

[14] P. Y. Lu, S. Kim, and M. Soljačić, "Extracting interpretable physical parameters from spatiotemporal systems using unsupervised learning," *Physical Review X*, vol. 10, no. 3, p. 031056, 2020.

[15] R. Veerapaneni, J. D. Co-Reyes, M. Chang, M. Janner, C. Finn, J. Wu, J. Tenenbaum, and S. Levine, "Entity Abstraction in Visual Model-Based Reinforcement Learning," in *Proceedings of the Conference on Robot Learning*, 2019.

[16] Z. Xu, J. Wu, A. Zeng, J. B. Tenenbaum, and S. Song, "DensePhysNet: Learning Dense Physical Object Representations via Multi-step Dynamic Interactions," in *Proceedings of Robotics: Science and Systems*, 2019.

[17] D. Zheng, V. Luo, J. Wu, and J. B. Tenenbaum, "Unsupervised Learning of Latent Physical Properties Using Perception-Prediction Networks," in *Uncertainty in Artificial Intelligence*, 2018.

[18] M. Noseworthy, S. Shaw, C. C. Kessens, and N. Roy, "Amortized Inference for Efficient Grasp Model Adaptation," *International Conference on Robotics and Automation*, 2024.

[19] M. Danielczuk, A. Balakrishna, D. Brown, and K. Goldberg, "Exploratory Grasping: Asymptotically Optimal Algorithms for Grasping Challenging Polyhedral Objects," in *Proceedings of the Conference on Robot Learning*, 2021.

[20] L. Fu, M. Danielczuk, A. Balakrishna, D. S. Brown, J. Ichnowski, E. Solowjow, and K. Goldberg, "LEGS: Learning Efficient Grasp Sets for Exploratory Grasping," in *Proceedings of IEEE International Conference on Robotics and Automation*, 2022.

[21] Z. Zhao, X. Li, C. Lu, and Y. Wang, "Center of mass and friction coefficient exploration of unknown object for a robotic grasping manipulation," in *Proceedings of the IEEE International Conference on Mechatronics and Automation*, 2018, pp. 2352–2357.

[22] D. Ćehajić, S. Hirche *et al.*, "Estimating unknown object dynamics in human-robot manipulation tasks," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2017, pp. 1730–1737.

[23] A. Herzog, P. Pastor, M. Kalakrishnan, L. Righetti, J. Bohg, T. Asfour, and S. Schaal, "Learning of grasp selection based on shape-templates," *Autonomous Robots*, vol. 36, no. 1, pp. 51–65, Jan. 2014.

[24] S. Dragiev, M. Toussaint, and M. Gienger, "Uncertainty aware grasping and tactile exploration," in *Proceedings of the IEEE International Conference on Robotics and Automation*, May 2013.

[25] S. Y. Gadre, K. Ehsani, and S. Song, "Act the part: Learning interaction strategies for articulated object part discovery," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021.

[26] R. Martín-Martín and O. Brock, "Coupled recursive estimation for online interactive perception of articulated objects," *The International Journal of Robotics Research*, vol. 41, no. 8, pp. 741–777, Jul. 2022.

[27] M. Bauza and A. Rodriguez, "GP-SUM. Gaussian Process Filtering of non-Gaussian Beliefs," in *Algorithmic Foundations of Robotics XIII*, M. Morales, L. Tapia, G. Sánchez-Ante, and S. Hutchinson, Eds. Cham: Springer International Publishing, 2020, pp. 508–525.