2024 IEEE International Conference on Robotics and Automation (ICRA)
May 13-17, 2024. Yokohama, Japan

Task-Oriented Active Learning of Model Preconditions for Inaccurate Dynamics Models

Alex LaGrassa[1], Moonyoung Lee[1], Oliver Kroemer[1]

***Abstract*— When planning with an inaccurate dynamics model, a practical strategy is to restrict planning to regions of state-action space where the model is accurate: also known as a *model precondition*. Empirical real-world trajectory data is valuable for defining data-driven model preconditions regardless of the model form (analytical, simulator, learned, etc...). However, real-world data is often expensive and dangerous to collect. In order to achieve data efficiency, this paper presents an algorithm for actively selecting trajectories to learn a model precondition for an inaccurate pre-specified dynamics model. Our proposed techniques address challenges arising from the sequential nature of trajectories, and potential benefit of prioritizing task-relevant data. The experimental analysis shows how algorithmic properties affect performance in three planning scenarios: icy gridworld, simulated plant watering, and real-world plant watering. Results demonstrate an improvement of approximately 80% after only four real-world trajectories when using our proposed techniques. More material can be found on our project website: https://sites.google.com/view/active-mde.**

I. Introduction

Many planning and control frameworks used in robotics rely on dynamics models, whether analytical or learned, to reason about how the robot's actions affect the state of the environment [1], [2]. However, robots deployed in the real world frequently encounter unfamiliar environments and complex interactions, e.g., with deformable objects, where assumptions of simplified dynamics break, making the models deviate from reality. In such situations, previous works [3]–[6] show how predicting model deviation with a Model Deviation Estimator (MDE) can be a powerful tool to restrict model-based planners to planning in regions of state-action space where the model is reliable, which we call *model preconditions*. Although using model preconditions defined by MDEs can improve the reliability of plans computed with inaccurate models, MDEs require real-world data, which can be expensive or dangerous, e.g., robot welding or pouring water, and drastically increase the cost of exploration.

We illustrate the intuition of our problem setting and approach in Fig. 1. Given a dynamics model that is accurate in only some combinations of states and actions, the objective is to estimate model deviation in state-action space to define the model precondition. During each iteration, the robot collects data and updates its model precondition based on that data, which is a form of active learning. Although we can draw from existing active learning techniques in other robotics applications [7], collecting a useful MDE dataset is challenging because it needs to contain a diverse set of trajectories to both identify the limits of the model and reliably solve planning problems. Furthermore, model error in earlier states of a selected trajectory can lead the robot to be unable to gather data from the later states due to the sequential nature of the problem.

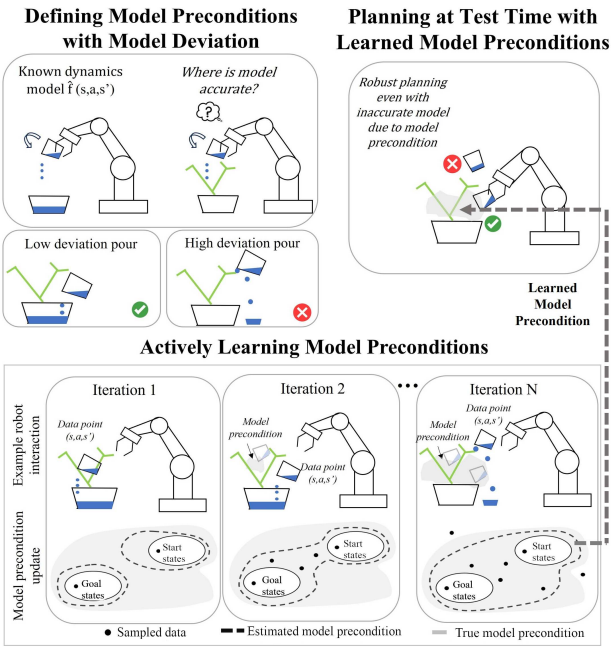


Fig. 1: Illustrative example of using a planner and acquisition function to iteratively select informative trajectories to define where the model is accurate to compute plans to the goal. In this example, the known dynamics model on the upper left ($\hat{f}(s,a,s')$) reasons only about the containers but not about the plant. The problem is to define where the model is accurate enough to compute plans to the goal. The resulting learned model precondition is then used at test time to only perform actions in the model precondition.

To address these problems, we describe active learning techniques for efficiently selecting trajectories to learn the MDE. We propose a task-oriented approach for generating trajectory candidates, as well as multi-step acquisition functions that compute a single utility value from the sequence of transitions in a trajectory. Our approach enables sample-efficient learning of an MDE for solving tasks from a given distribution. We analyze how variations on the active learning algorithm affect the dataset, and subsequently the quality of the model preconditions at test time.

This paper makes the following contributions: (1) a novel problem formulation and approach for active learning of

A, LaGrassa, M. Lee, O. Kroemer are with Carnegie Mellon University Robotics Institute, Pittsburgh PA, USA {alagrass, moonyoul, okroemer,}@andrew.cmu.edu

This work was supported by NSF Grants No. CMMI-1925130 and IIS-1956163, ARL Grant No. W911NF-18-2-0218 as part of the A2I2 Program, and NSF/USDA NIFA AIIRA AI Research Institute 2021-67021-35329.

model preconditions defined by MDEs to improve the accuracy and robustness of plans for manipulation tasks with variable-accuracy models; (2) analysis of the effect of acquisition function choices on trajectory selection during training and the resulting test-time reliability of executed plans.

II. Related Work

This work focuses on planning with variable-accuracy models, where assuming globally high accuracy would lead to failures. Existing techniques to mitigate the impact of model deviation on task performance such as adaptive control [8], [9], and reasoning about uncertainty with probabilistic models [10]–[12] are still susceptible to inaccuracy [13]. Furthermore, some models, including simulators and many analytical models, lack the capacity to represent uncertainty. Similarly, dynamics models learned from data [14]–[17] can be inaccurate for a variety of reasons such as scenarios outside the training data distribution and limited model capacity for complex interactions. Despite these limitations, such models demonstrate practical utility in various planning tasks.

Our approach does not intend to replace learned or uncertain dynamics models, but rather to complement them to address persistent model inaccuracies. Other work has shown that estimating model deviation can be more data-efficient than learning a dynamics model with an equivalent amount of data, and lead to higher reliability [3], [5], [18]. Despite some data-efficiency improvements, current approaches to estimating model deviation lack active learning capabilities, limiting their use to scenarios where the inherent randomness of the planning process and environment can sufficiently cover the data space [4], [19].

Though we address a different problem, we use similar tools as broader active learning techniques used in other areas of robotics, such as using probabilistic models to select informative samples for dynamics models, skill preconditions, and policies [7], [20], [21]. Active dynamics learning [22], [23] approaches sometimes address the additional challenge of sequential dependence of selecting informative points, but these works put strong assumptions on the form of the dynamics model, whereas we extend our scope to allow model preconditions over various types of models such as analytical models and simulators. Model preconditions are different but potentially more generalizable than skill preconditions since multiple model-based behaviors can use the same model preconditions.

Real-world fluid manipulation particularly benefits from efficient exploration. Existing works tend to be conservative in action space by limiting pours to a small region, such as directly over a target container, which greatly limits the set of observable dynamics [24]–[26]. Furthermore, other approaches are largely constrained to scenarios with simple dynamics [26]–[29] where failure tends to be over-pouring or under-pouring. To our knowledge, this is the first experimental setting that uses a commonly-used 7 DOF manipulator to perform actions that can often spill water into the workspace.

III. Problem Statement

In this work, we actively learn model preconditions for planning with inaccurate dynamics models of the form $\hat{s} \leftarrow \hat{f}(s, a)$. We do *not* make additional assumptions on the implementation or source of the model (e.g. analytical model, simulator, learned model). A model precondition, denoted as $\mathrm{pre}(\hat{f})$, is a region where a planner may use a given $\hat{f}(s, a)$.

The planning problems in our setting are defined by sampling a start state and goal function $g(s)$ that outputs whether or not s is a goal state. Goals are achieved by planning and executing a trajectory defined by actions $a_{1:T-1}$ and predicted states $\hat{s}_{1:T}$ such that $g(s_T)$ holds. We assume that $\hat{f}(s, a)$ is sufficient for solving the planning problems. At test time, the planner uses the learned model precondition to reject transitions where $(s, a) \notin \mathrm{pre}(\hat{f})$.

The concrete form of model preconditions we use describes (s, a, s') transitions where the deviation between predicted states and next states $d(\hat{s}, s')$ stay within a threshold tolerance, $d_{\max}$, that the system can tolerate or correct. $d(s_i, s_j)$ is a distance function, such as Euclidean distance, that outputs a scalar. The constraint can then be defined as $\mathrm{pre}(s, a) = \{s, a \,|\, d(s', \hat{f}(s, a)) < d_{\max}\}$. Since $d(s', \hat{f}(s, a))$ is impossible to compute without knowing s', we instead estimate $d(s', \hat{f}(s, a))$ given (s, a), denoted as $\hat{d}(s, a)$ to indicate that it is estimated for a state and action.

The active learning problem in this work is to select a set of (variable-length) trajectories to form a dataset $\mathcal{D}$ of (s, a, s') tuples on which an MDE is trained. Each trajectory is denoted by τ, and we describe the set of candidate trajectories that different searches generate for the same problem as $\boldsymbol{\mathcal{T}}$. The agent may use the planner during training time and sample from the same *distribution* of planning problems that will be seen at test time, but not the same problems. We assume access to sufficiently accurate state estimation to compute meaningful deviations between all observed points on the trajectory.

IV. Learning a Model-Deviation Estimator

As defined in Section III, an MDE predicts the deviation $d(\hat{s}, s')$ for a particular model $\hat{f}(s, a)$ between a predicted state $\hat{s}$ and the true next state s'. We denote the output of an MDE as $\hat{d}(s, a)$. By directly predicting $d(\hat{s}, s')$, the MDE is agnostic to the source of model deviation. The MDE in this paper is a Gaussian Process (GP) model with a Matérn kernel and heteroscedastic noise model, specifying a Gaussian distribution for the deviation with mean $\mu(\hat{d}(s, a))$ and standard deviation $\sigma(\hat{d}(s, a))$. A heteroscedastic noise model enables input-dependent noise, which is important to capture when sources of model deviation differ.

Data collected for the MDE is in the form of (s, a, s') tuples from executing action a in the target environment (i.e., the real world) from state s, and then observing s'. The input to the MDE is (s, a) and the label is $d(\hat{f}(s, a), s')$.

A learned MDE then defines a model's precondition as: $\mathrm{pre}(\hat{f}) = \{(s, a) | P(\hat{d}(s, a) > d_{\max}) < \delta\}$ for some small probability δ. The constraint can be written as $\mu(s, a) + \beta\sigma(s, a) < d_{\max}$, where higher β lowers the risk tolerance.

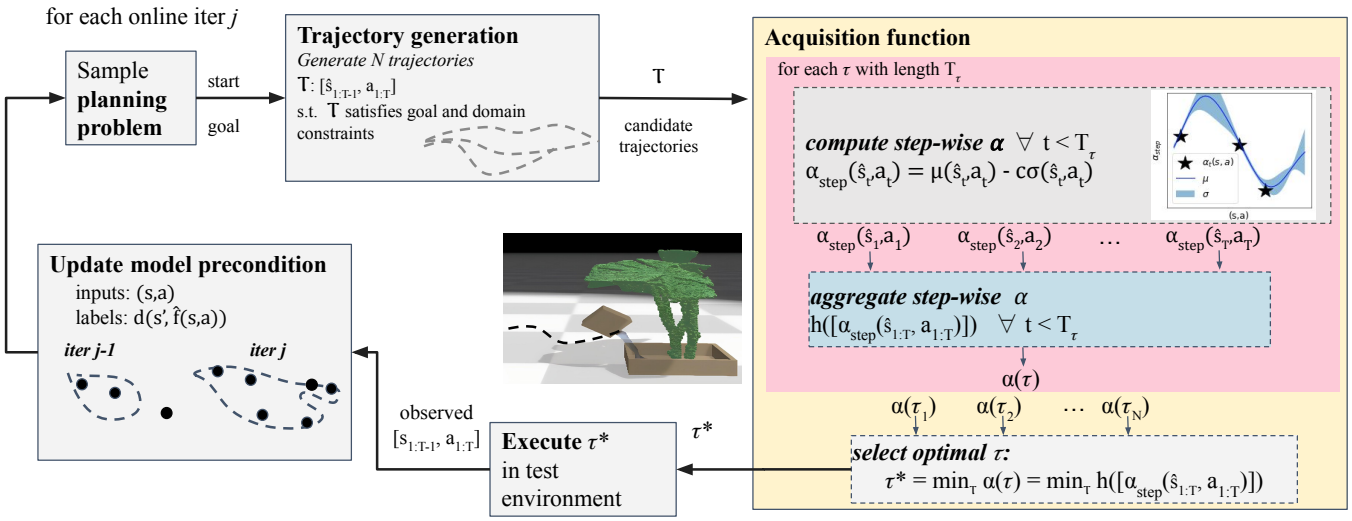


Fig. 2: Overview of our method: Each iteration j starts with sampling a planning problem and generating candidate trajectories that satisfy domain constraints and reach the goal. We outline the acquisition function computation for *each* trajectory in the pink box, including the step-wise acquisition function values, $\alpha_{\text{step}}(s_t, a_t)$ for each state-action pair in the trajectory. These values are then aggregated by a function h to yield the trajectory's utility: $\alpha(\tau)$. The final step is selecting and executing $\tau*$, in the test environment to collect the ground truth $[s_{[1:T_\tau]}, a_{[1:T_\tau-1]}]$. The MDE is updated every M trajectories.

V. Active Learning

The algorithm for actively learning MDEs is illustrated in Fig. 2. First, the agent samples a planning problem and then uses a motion planner to generate candidate trajectories, $\boldsymbol{\mathcal{T}}$. The search adds transitions to the tree if they satisfy planning constraints such as joint limits, a collision check, and the model precondition as defined in Section IV. To encourage exploration, a zero-mean prior is used for the MDE during the learning phase. We use a rapidly exploring random tree (RRT) planner [30] to encourage a diverse set of solutions in $\boldsymbol{\mathcal{T}}$. Then, the robot executes the trajectory that minimizes an acquisition function $\alpha(\tau)$, which is a heuristic for the utility of τ to the MDE. After a batch of M executed trajectories for M problems, the robot adds the observed (s, a, s') tuples to $\mathcal{D}$ and updates the MDE using the training method described in Section IV. Ideally, the model precondition region between the start and goal states expands with more data (Fig. 1).

At test time, the robot generates a trajectory to the goal using the same planner, MDE, and constraints as during training, but with a more conservative model precondition. The β parameter as described in Section III sets the tolerance for deviations outside the model precondition; for example, $\beta = 2$ specifies a 98% confidence interval.

A. Acquisition Function

Now, we explain how we define the acquisition function $\alpha(\tau)$, which guides our selection of the trajectory to execute in each iteration: $\tau^* \leftarrow \text{argmin}_{\tau \in \boldsymbol{\mathcal{T}}}; \alpha(\tau)$. The procedure is illustrated in the rightmost box of Fig. 2.

Step-wise utilities: First, we compute utilities for each step in the trajectory, shown in the dotted box. Trajectories can vary in length, denoted as T_τ, and are comprised of states and actions: $s_{1:T_\tau}, a_{1:T_\tau-1}$. We denote the utility for each step t as $\alpha_{\text{step}}(s_t, a_t)$ and define it using a form inspired by Lower Confidence Bound: $\mu(x) - c\sigma(x)$ where c controls exploration.

Aggregating individual step-wise utilities for a trajectory: As shown in the blue dotted box (Fig. 2), we next define a function h that aggregates single-step utilities in trajectories of different lengths, $\alpha_{step}(s_{1:T_\tau}, a_{1:T_\tau})$, to a single trajectory utility, $\alpha(\tau)$.

The general form of a trajectory-based acquisition function using the lower confidence bound-based α_{step} is thus:

$$\alpha(\tau) \leftarrow h([\mu(\hat{d}(s_t, a_t)) - c\sigma(\hat{d}(s_t, a_t))] \; \forall t < T_\tau) \quad (1)$$

Since later transitions may not be reached when the trajectory is planned using an inaccurate dynamics model, we introduce $h_{\max}$ and h_{sum}. $h_{\max} = \max_{t<T_\tau} \gamma^t \alpha_{\text{step}}(s_t, a_t)$ and $h_{\text{sum}} = \sum_{t<T_\tau} \gamma^t \alpha_{\text{step}}(s_t, a_t)$. Multiplying each $\alpha_{\text{step}}(s_t, a_t)$ by γ^t approximates the idea that nearer steps are more useful in the trajectory.

B. Candidate trajectory generation

To set the risk tolerance during training, we propose a schedule for the MDE that gradually reduces risk tolerance as the robot accumulates more data. Since δ is determined by $\mu(s, a) + \beta\sigma(s, a)$, δ can be set by modifying β using the inverse CDF of a Gaussian distribution: $\beta = \Phi^{-1}(1 - \delta)$. At iteration j of J total iterations, $\beta_j \leftarrow \frac{2k_1}{1+\exp\left(-k_2(j-\frac{J}{2})\right)} - k_1$ results in a transition from $-k_1$ to k_1 where a lower k_2 causes a smoother transition.

VI. Experimental Setup

The first scenario, `Icy GridWorld` (Fig. 3a), is a grid-world also used in [31] where the robot can move in four cardinal directions, but if it moves left or right over an icy state, it slips by moving two cells backwards. The robot cannot move through the obstacle. The dynamics model assumes the robot moves to the intended location. The start

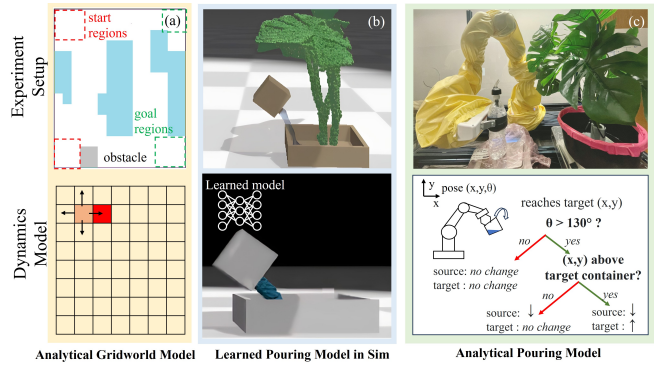
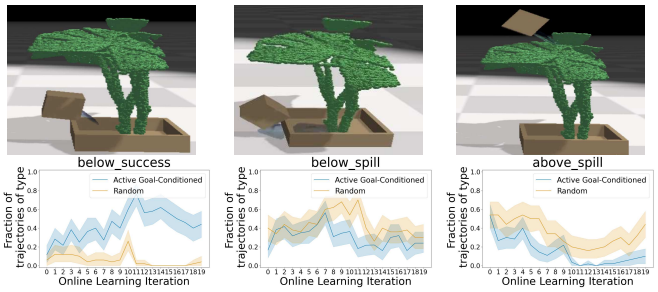


Fig. 3: Scenarios and their corresponding dynamics models. (a) Slippery grid world where movement may result in slipping backwards over ice (blue) or not moving (grey). The analytical dynamics model assumes unimpeded movement within grid bounds. (b) Simulated plant watering using a learned dynamics model trained on a scenario without a plant. (c) Real-world plant watering with a rule-based analytical dynamics model.

and goal state are selected randomly for each planning problem.



(a) *below* leaves success (b) *below* leaves and spills (c) *above* leaves and spills

Fig. 4: Ratio of trajectory types executed during training (examples shown above) for our method and the *Random* ablation over training iterations.

The second scenario, `Water Plant(sim)` (Fig. 3b), is a plant-watering domain where the goal is to pour a specified amount of water from a source container into a target container without spilling more than 2%. The state space is the poses of both containers and their liquid volumes. The actions are specified as a target pose for the source container. The actions must satisfy constraints unrelated to the dynamics model including being collision-free according to an approximate collision checker, and either translating or rotating in one motion, but not both. The model is a neural network dynamics model trained in a simpler environment (shown in the bottom half of Fig. 3) with no plant and a wider source container. This scenario tests the algorithm's ability to learn model preconditions caused by multiple sources of model error such as geometry mismatch, obstruction by leaves, and unexpected collisions. The start state is a random pose left of the target container.

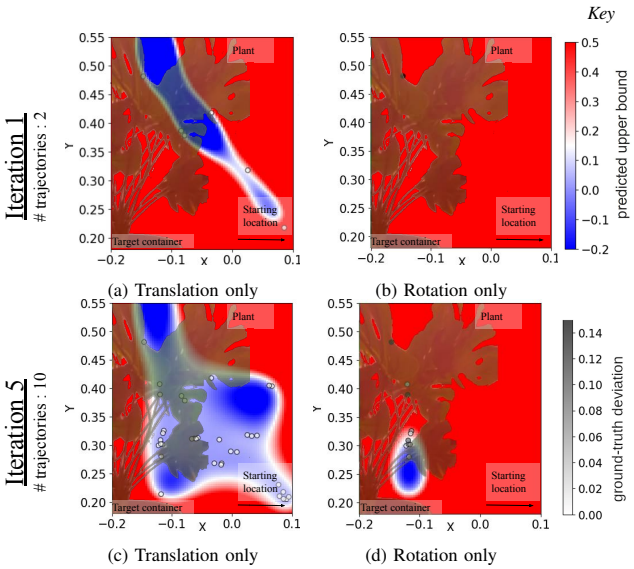
Lastly, the third scenario, `Water Plant(real)` (Fig. 3c), is a real-world variation of the previous plant-watering domain. A measured pourer dispenses 15 mL of water when tilted above 130 degrees. The action space and state space representations are consistent with the simulated scenario, but we restrict the MDE input to only the action to reduce dimensionality. The branches can move, but the base stays fixed relative to the container. Since the plant state is only measured by the container pose, variations that affect the dynamics cause noise. This scenario demonstrates that reliable performance can be reached in a small number of trajectories (less than a dozen) in the real world where there is considerably more noise and variation. The analytical model we use assumes that 15 mL is dispensed for rotational actions above 130 degrees and that the water enters the target container if poured above the area of the container. The start state is fixed and the goal is for 15 mL to be in the target container without spilling more than 5 mL.

Evaluation methodology: On simulated domains, we evaluate each variation using 10 seeds for 20 learning iterations. The simulated domains use five training trajectories per iteration, and the real domain uses two. Setup, planning and execution for each training trajectory takes 15 s in `Icy GridWorld`, 1.5 min in `Water Plant(sim)`, and 3 m in `Water Plant(real)`. At test time, we evaluate the model preconditions for each iteration by using the model preconditions where $\hat{d}(s,a) < d_{\max}$ for each transition with 98% confidence. In the simulated scenarios, we sample 20 planning problems per iteration, and in the real-world scenario, we sample 5 per iteration. In `Water Plant(real)`, we only evaluate the effect of using active learning and goal-conditioned candidate trajectories. $d_{\max} = 0.1$ for all scenarios and represents the sum of all position distances over all objects. We use a risk-tolerance schedule (Section V-B) with $k_1 = 2$ and $k_2 = \frac{1}{2}$ In both watering scenarios, the average volume deviation of both containers is added to the position error. For consistency, we scale the volume units between simulation and the real-world such that one unit is poured out.

The metrics that we evaluate are as follows. First, we evaluate model precondition accuracy on a cross-validation dataset of trajectories from the other seeds. We measure both the true negative rate (TNR) and the true positive rate (TPR) of whether an individual (s, a, s') data point is in the model precondition. A higher rate is better for both metrics, but the TNR is more important than the TPR because the model precondition only needs to cover enough state-action space to compute a plan. Second, we test whether the model precondition is sufficient to reliably compute plans by measuring the success rate of the planner within a fixed timeout of 5000 extensions. Finally, we evaluate the end-to-end success rate in achieving the goal, which measures the effect of using the estimated model preconditions over the entire trajectory.

VII. Results

We first show a qualitative analysis of data collected during the active learning. We then quantitatively evaluate the effects of different algorithmic variations using the metrics we previously described in Section VI. Two baselines using an MDE but different methods to generate $\mathcal{T}$ measure the impact of algorithmic choices. *Random* selects random



(a) Translation only (b) Rotation only

(c) Translation only (d) Rotation only

Fig. 5: MDE with plant overlay over training iterations for translation-only actions and rotation-only actions. Color scales indicate ground-truth deviation (right, top) and upper bound of the predicted deviation $\mu(\hat{d}(s,a)) + \beta\hat{d}(\sigma(s,a))$ for $\beta = 2$. $d_{\max} = 0.1$, so the blue region indicates the model precondition.

actions, effectively removing the goal-conditioning. *Goal-Conditioned* eliminates the acquisition function by selecting a random trajectory to the goal.

Online learning analysis: Here, we analyze the qualities of the MDE dataset when using our active learning approach described in Section V (*Active Goal-Conditioned*) compared to selecting trajectories that do not reach the goal, but use the same samplers from the planner, which we call *Random*. The first row of Fig. 4 shows labels of trajectories selected during training to analyze the makeup of the MDE dataset. `below_success` is the label for pours below the leaves where all the water reaches the target, which is what would ideally be well represented in the dataset. Since data where the model is inaccurate is also necessary to quantify the limitations of the model, we also track the portion of the dataset that is `below_spill`. The final trajectory type we track is `above_spill`, which is the least useful because the water-container-leaf dynamics are typically high-deviation and unnecessary to complete tasks.

Fig. 4 shows a faster increase in desirable `below_success` pours from *Active Goal-Conditioned* in iterations 0 to 9. We note that there is an increase in successful pours around iterations 10-12, which then decreases. As we show in the included video, this effect can be attributed to responses in both aleatoric and epistemic uncertainty, which affects the exploration bonus: $c\sigma(\hat{d}(s,a))$.

Model preconditions in `Water Plant(real)`: Here, we visually analyze how regions of model preconditions change in the real-world pouring task. These regions are low dimensional so it can be visualized in 2D by splitting θ_d into two-cases: one for rotation-only actions, and one for translation-only actions. We show the model precondition by plotting $\hat{d}(s,a)$ over a region of $\mathcal{A}$ (Fig. 5). Aligned with our intuition of how model preconditions should evolve, the area of the model precondition expands as more data is collected. The boundary becomes more precise with more data points (Fig. 5). By iteration 5, the model precondition for rotational actions is in a region above the target container but below the leaves. Note that despite a low-deviation point at iteration 5 at (-0.12, 0.39), that area is not in the model precondition because there is a nearby point that is high-deviation.

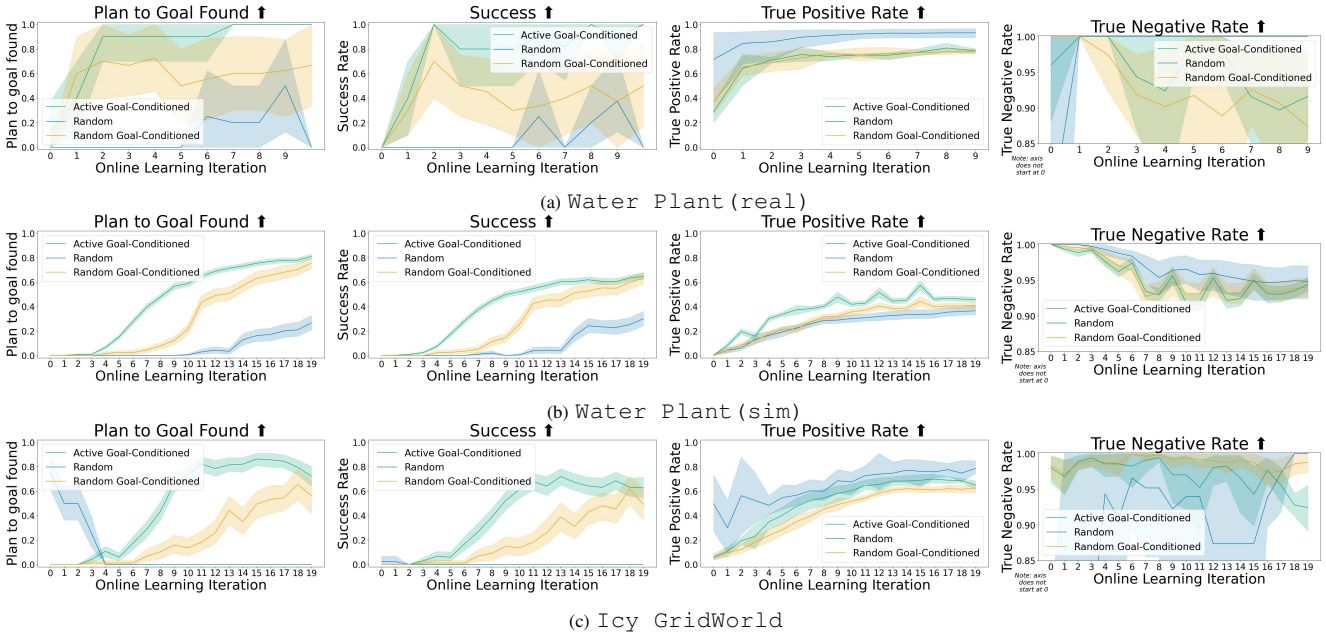
A. Candidate Trajectory Set Generation

Here, we evaluate how the method to generate $\mathcal{T}$ affects performance metrics. Fig. 6, shows higher data efficiency when using *Active Goal-Conditioned* or *Goal-Conditioned* in all scenarios. In `Water Plant(sim)`, we also see significantly higher data efficiency when using active learning in addition to conditioning on trajectories that reach the goal, indicated by higher performance after fewer iterations when using *Active Goal-Conditioned* over using *Goal-Conditioned*, which does not use an acquisition function. In `Water Plant(real)`, we see a clear improvement when using goal-conditioning, and a modest improvement in success when using the acquisition function in later iterations. The improvement of *Active Goal-Conditioned* over *Goal-Conditioned* in finding plans to the goals is matched by a significant increase in the TPR, which is more significant in the simulated environments than in `Water Plant(real)`. *Active Goal-Conditioned* shows an improvement in success rate reflected by a higher TNR. Earlier iterations in `Icy GridWorld` using *Random* have overly broad model preconditions, as seen by a high TPR, high success in finding goals, but low success in reaching them. Overall, we see a positive affect when using both goal-conditioning and our active learning method.

Model precondition accuracy is not necessarily indicative of high performance. As shown in the first row of Fig. 6, although *Random* has both a high TNR and high TPR, it never finds plans to the goal because of insufficient task-relevant data. Additional results on the risk-tolerance schedule and candidate trajectory diversity can be found on our website.

B. Effect of acquisition function on performance

We analyze the impact of the aggregation function and discount factor on performance. We test $\gamma = 0.9$ and compare to no discounting with $\gamma = 1$. We observe an slight improvement in performance in both `Icy GridWorld` and `Water Plant(sim)` when using $\gamma = 0.9$ and $h_{\max}$, which is consistent with our intuition that discounting can account for the dependence of later states on reaching earlier states. The improvement is smaller for `Icy GridWorld` and more apparent in the success rate in finding plans. Overall, we find that the choice of γ and h does not have a major impact on performance, but there may be some benefit in using a discount factor or other forms of aggregation of step-wise acquisition function values.

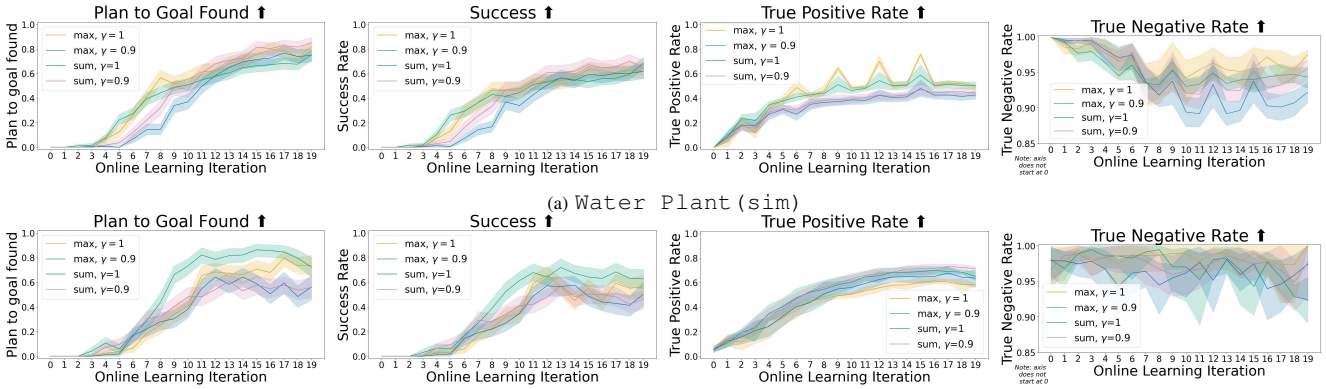


(a) `Water Plant (real)`

(b) `Water Plant (sim)`

(c) `Icy GridWorld`

Fig. 6: Success rate in computing a plan, success rate in reaching the goal, and model precondition accuracy over training iterations for three scenarios. These plots measure the effect of whether trajectories are selected actively, goal conditioned, or not goal conditioned on performance.



(a) `Water Plant (sim)`

(b) `Icy GridWorld`

Fig. 7: Success rate in computing a plan, success rate in reaching the goal, and model precondition accuracy over training iterations for three scenarios. These plots measure the effect of the aggregation function on performance.

VIII. Limitations

Our proposed active learning approach is limited to scenarios where dynamic variables that significantly affect deviation are represented in the state. Unobserved dynamic variables are implicitly modelled as noise, which may lead to overly restrictive model preconditions. This issue can be mitigated by prioritizing recent data or by incorporating these variables into the state, if feasible. Our MDE implementation with a GP does not directly scale to high-dimensional state spaces. Future work will explore using pre-trained general-purpose models to address this, potentially using cross-task information for added efficiency.

IX. Conclusions

This paper formulates the problem of active learning of model preconditions then presents a novel class of techniques designed to generate and select candidate trajectories. We evaluate the performance of variations on our active learning approach on two simulated scenarios and one real-world task with learned models and analytical models. Our experimental results demonstrate the effect of algorithmic choices in candidate trajectory selection and acquisition function on data efficiency. This work enables empirical estimation of model preconditions with minimal data, a capability we plan to extend to high-dimensional deformable object scenarios where the use of model preconditions can be particularly beneficial.

REFERENCES

[1] C. J. Bates, I. Yildirim, J. B. Tenenbaum, and P. Battaglia, "Modeling human intuitions about liquid flow with particle-based simulation," *PLoS computational biology*, vol. 15, no. 7, p. e1007210, 2019.

[2] T. Pfaff, M. Fortunato, A. Sanchez-Gonzalez, and P. Battaglia, "Learning mesh-based simulation with graph networks," in *International Conference on Learning Representations*, 2020.

[3] P. Mitrano, D. M^{c}Conachie, and D. Berenson, "Learning Where to Trust Unreliable Models in an Unstructured World for Deformable Object Manipulation," *Science Robotics*, 2021.

[4] D. McConachie, T. Power, P. Mitrano, and D. Berenson, "Learning When to Trust a Dynamics Model for Planning in Reduced State Spaces," *IEEE Robotics and Automation Letters*, 2020.

[5] T. Power and D. Berenson, "Keep it simple: Data-efficient learning for controlling complex systems with simple models," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1184–1191, 2021.

[6] A. L. LaGrassa and O. Kroemer, "Learning model preconditions for planning with multiple models," in *Conference on Robot Learning*, pp. 491–500, PMLR, 2021.

[7] Z. Wang, C. R. Garrett, L. P. Kaelbling, and T. Lozano-Pérez, "Learning compositional models of robot skills for task and motion planning," *The International Journal of Robotics Research*, vol. 40, no. 6-7, pp. 866–894, 2021.

[8] K. S. Narendra and J. Balakrishnan, "Adaptive control using multiple models," *IEEE transactions on automatic control*, vol. 42, no. 2, pp. 171–187, 1997.

[9] J. Fu, S. Levine, and P. Abbeel, "One-shot learning of manipulation skills with online dynamics adaptation and neural network priors," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4019–4026, IEEE, 2016.

[10] E. Páll, A. Sieverling, and O. Brock, "Contingent contact-based motion planning," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6615–6621, IEEE, 2018.

[11] L. P. Kaelbling and T. Lozano-Pérez, "Integrated task and motion planning in belief space," *The International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1194–1227, 2013.

[12] S. Levine and P. Abbeel, "Learning neural network policies with guided policy search under unknown dynamics," *Advances in neural information processing systems*, vol. 27, 2014.

[13] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," in *International conference on machine learning*, pp. 1321–1330, PMLR, 2017.

[14] R. Takano, H. Oyama, and Y. Taya, "Robot skill learning with identification of preconditions and postconditions via level set estimation," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 10943–10950, 2022.

[15] A. Nagabandi, K. Konolige, S. Levine, and V. Kumar, "Deep dynamics models for learning dexterous manipulation," in *Conference on Robot Learning*, pp. 1101–1112, PMLR, 2020.

[16] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson, "Learning latent dynamics for planning from pixels," in *International conference on machine learning*, pp. 2555–2565, PMLR, 2019.

[17] P. Wu, A. Escontrela, D. Hafner, P. Abbeel, and K. Goldberg, "Daydreamer: World models for physical robot learning," in *Conference on Robot Learning*, pp. 2226–2240, PMLR, 2023.

[18] H. Liu and G. M. Coghill, "A model-based approach to robot fault diagnosis," in *Applications and Innovations in Intelligent Systems XII: Proceedings of AI-2004, the Twenty-fourth SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence*, pp. 137–150, Springer, 2005.

[19] P. Mitrano and D. Berenson, "Data augmentation for manipulation," *Robotics: Science and Systems 2022*, 2022.

[20] I. Abraham and T. D. Murphey, "Active learning of dynamics for data-driven control using koopman operators," *IEEE Transactions on Robotics*, vol. 35, no. 5, pp. 1071–1083, 2019.

[21] B. Eysenbach and S. Levine, "Maximum entropy rl (provably) solves some robust rl problems," in *International Conference on Learning Representations*, 2021.

[22] A. Capone, G. Noske, J. Umlauft, T. Beckers, A. Lederer, and S. Hirche, "Localized active learning of gaussian process state space models," in *Learning for Dynamics and Control*, pp. 490–499, PMLR, 2020.

[23] M. Buisson-Fenet, F. Solowjow, and S. Trimpe, "Actively learning gaussian process dynamics," in *Learning for dynamics and control*, pp. 5–15, PMLR, 2020.

[24] N. Correll, N. Arechiga, A. Bolger, M. Bollini, B. Charrow, A. Clayton, F. Dominguez, K. Donahue, S. Dyar, L. Johnson, *et al.*, "Indoor robot gardening: design and implementation," *Intelligent Service Robotics*, vol. 3, pp. 219–232, 2010.

[25] J. Stückler and S. Behnke, "Adaptive tool-use strategies for anthropomorphic service robots," in *2014 IEEE-RAS International Conference on Humanoid Robots*, pp. 755–760, 2014.

[26] C. Schenck and D. Fox, "Visual closed-loop control for pouring liquids," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2629–2636, IEEE, 2017.

[27] M. Kennedy, K. Schmeckpeper, D. Thakur, C. Jiang, V. Kumar, and K. Daniilidis, "Autonomous precision pouring from unknown containers," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2317–2324, 2019.

[28] Y. Noda and K. Terashima, "Modeling and feedforward flow rate control of automatic pouring system with real ladle," *Journal of Robotics and Mechatronics*, vol. 19, no. 2, pp. 205–211, 2007.

[29] J. C. Vaz and P. Oh, "Model-based suppression control for liquid vessels carried by a humanoid robot while stair-climbing," in *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*, pp. 1540–1545, IEEE, 2020.

[30] S. LaValle, "Rapidly-exploring random trees: A new tool for path planning," *Research Report 9811*, 1998.

[31] A. Vemula, Y. Oza, J. A. Bagnell, and M. Likhachev, "Planning and execution using inaccurate models with provable guarantees," in *Robotics: Science and Systems XVI, Virtual Event / Corvalis, Oregon, USA, July 12-16, 2020* (M. Toussaint, A. Bicchi, and T. Hermans, eds.), 2020.