Planning with Learned Model Preconditions for Water Manipulation

Alex LaGrassa
Robotics Institute
Carnegie Mellon University
Pittsburgh, USA
alagrass@andrew.cmu.edu

Oliver Kroemer
Robotics Institute
Carnegie Mellon University
Pittsburgh, USA
okroemer@andrew.cmu.edu

***Abstract*—Although models are useful for long-horizon reasoning, the complex dynamics of deformable objects in many real-world tasks limit the applicability of many model-based planning algorithms. In this work, we address the difficulty of modeling deformable object dynamics by learning where a set of given high-level dynamics models are accurate: a *model precondition*. Model preconditions are then used to find trajectories using states and closed-loop actions where the dynamics models are accurate. We demonstrate the efficacy of our approach on pouring and water transport using both analytical and learned models of water movement. We also show qualitative results to provide intuition for what model preconditions might be for the models used in those tasks.**

***Index Terms*—planning, manipulation**

I. Introduction

Deformable object manipulation is difficult due to the high number of degrees of freedom typically used to represent such objects [1]. General-purpose fluid dynamics models require particularly expensive computational models. Furthermore, dynamics models do not accurately reflect real-world dynamics, especially when key dynamical properties of the liquid and objects it interacts with are partially observable [1]. Although data-driven models for pouring exist [2], specific interactions between objects and water can be modelled accurately using computationally efficient analytical models that do not require as much data as general-purpose neural-network models [3, 4]. These models rely on assumptions on factors such as the geometry of the container, viscosity of the liquid and how it is moved. The accuracy of data-driven models depends on the training data and model representation. The best model to use depends on the constraints of the task.

In this work, we propose an approach that addresses the complementary benefits and drawbacks of different types of deformable object models by predicting model error, and then using that information during planning to focus on states and actions where the current models are accurate. When models are updated using new data, the model preconditions are updated to reflect this. To describe model error, we use a distance function that collapses high-dimensional state information into a measure of how different two states are. By applying this assumption, we can express model preconditions using Model Deviation Estimators (MDE), which predict the error of an existing model on a given state and action. MDEs constrain planning to dynamics that are predicted to be accurate.

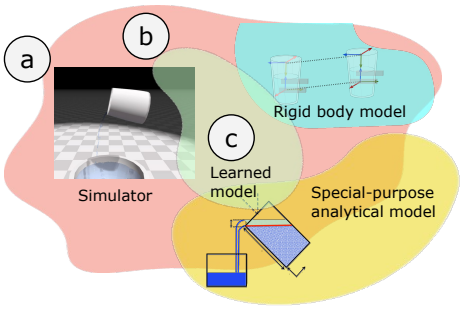


Fig. 1: Each colored region in $\mathcal{S} \times \mathcal{A}$ space represents a model precondition, which we learn in this work. We show situations where applications of our approach can improve planning. For example, a) cannot be modelled with the given set of models and should be avoided, b) can be modelled using a simulator, and c) can be accurately modeled using a simulator, but a learned model could confer planning speed advantages.

The main contribution of this work is demonstrating the effectiveness of using model preconditions defined using MDEs to increase the reliability of inaccurate learned and analytical models used for motion planning with deformable objects. In our experiments, we evaluate the effect of applying our approach on success rate in finding plans, and achieving task success when executing those plans.

II. Related Work

The proposed approach in this paper builds on previous research in planning using inaccurate models, model error prediction, and online model adaptation.

One common way to address modeling error of deformable objects is to use system identification to infer their dynamical properties, to then use the more accurate model for control. Only using MPC to address model error fails when errors cannot be corrected locally. Prior work has avoided states with inaccurate dynamics by predicting where the model is inaccurate from data [5, 6, 7, 8] or biasing controllers to states similar to the training data [9, 10]. Our work is focused on computing plans using multiple actions which avoid model

This work was supported by the Office of Naval Research Grant No. N00014-18-1-2775, Army Research Laboratory grant W911NF-18-2-0218 as part of the A2I2 program, and National Science Foundation Grant No. CMMI-1925130 and IIS-1956163. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of our sponsors.

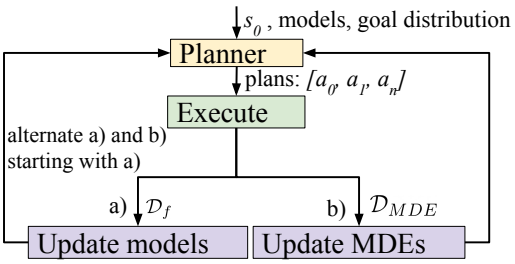


Fig. 2: Overview of the proposed approach. Goals are sampled and the planner computes plans to the goal using the current models. a) The system first trains the learned models to collect $\mathcal{D}_f$ b) Then, MDEs are learned using transitions calculated with the new plans, $\mathcal{D}_{MDE}$.

error by estimating it using a small amount of real-world data. We also address the issue of models being inaccurate for most state-action space by adding a learned model to the framework in [11].

Other approaches to model-based water manipulation assume accurate models after a calibration phase and account for model error using model predictive control [12, 13, 14]. If these models are insufficiently accurate, they can be improved by collecting more data. However, this can require a lot of data to be accurate across all possible state-action pairs [15, 16], we aim to only learn the dynamics of transitions required for planning. Our approach only requires learning a specific set of high-level dynamics, which avoids many of the challenges in trajectory optimization using approximate low-level models [17, 18].

III. Approach

In this section, we overview our proposed approach to learning and planning with model preconditions, which is shown graphically in Figure 2. First, we describe how we collect data to train the models and corresponding MDEs. Next, we describe how MDEs are used during planning. The planner and MDE formulation are adapted from [11], which contains a detailed description of the planning algorithm. The eventual goal is for the robot to search for a high-level plan of actions $[a_0, a_1, \ldots, a_{T-1}]$ that visit a sequence of states $[s_0, s_1, \ldots, s_T]$ such that the final state is a goal state: $s_T \in \mathcal{S}^g$

The actions used in our planner are closed-loop options with a precondition, controller $\pi(s)$, and termination condition [19]. The precondition describes the set of states from which it is valid to execute the controller. The effects of an action are modelled by M, which is a *Skill Effect Model* (SEM) [20]. Learned SEMs describe the state after executing a parameterized skill to termination. Data is collected by first computing N_f plans to a set of sampled goals using the current models, then executing them in the target environment, typically the real world, to obtain (s, a, s') tuples. First, data is collected to train the learned models (Figure 2a). The dataset collected this way is denoted as $\mathcal{D}_f$, as it is meant to represent a small subset of the real-world dynamics: $s' = f(s, a)$. During model update, learned models are trained by minimizing MSE on $\mathcal{D}_f$. We use Random Forest Regression for its data efficiency.

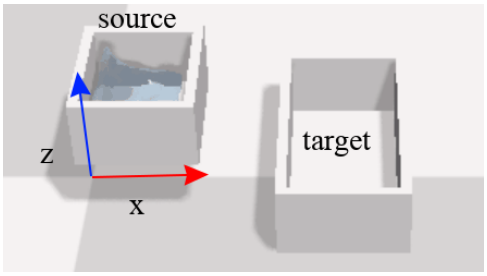


Fig. 3: Experimental setup in SoftGym. The box on the left is controllable in x, z, and rotation about the XZ plane

We express model preconditions using an MDE, which given a distance function d predicts $d(s', M(s, a))$. This prediction is the error of model M on taking action a from state s. MDEs are trained from data in the target domain by using the learned models to collect a new dataset, $\mathcal{D}_{MDE}$, by executing N_{MDE} trajectories to N_{MDE} different goals. The procedure to collect $\mathcal{D}_{MDE}$ is the same as for $\mathcal{D}_f$ but with updated models.

If a transition is in a model's precondition, then that model can be used to accurately model that transition. A model precondition is defined using MDEs $\text{pre}(M_i) = \{(s, a) \,|\, P(\hat{d}(M_i(s, a), s') < d_{\max}) < \delta\}$.

We implement MDEs using a Gaussian Process Regression with a Matern kernel for k and additive Gaussian Noise [21]. The posterior $\mu_{GP}(s, a)$ and $\sigma_{GP}(s, a)$ after observing inputs (s, a) and labels are $d(M(s, a), s')$ for all $(s, a, s') \in \mathcal{D}_{MDE}$ represent the MDE for a model M_i. For data efficiency, each transition is used even if that model was not used to compute the successor of (s, a).

Finally, we define the model precondition using the MDE. $P(\hat{d}(M_i(s, a), s') < d_{\max}) < \delta$ holds if $\mu_{GP}(s, a) + \beta\sigma_{GP}(s, a) < d_{\max}$ where β is a hyperparameter that specifies an upper bound on $P(\hat{d}(s, a) > d_{\max})$ according to the GP. For a normal distribution, $\beta = 1$ and $\beta = 2$ corresponds to approximately 84% and 98% probability respectively. Thus, the model precondition can be written as: $\text{pre}(M_i) = \{(s, a) \,|\, \mu_{GP}(s, a) + \beta\sigma_{GP}(s, a) < d_{\max}\}$

We use the planner from [11] but with only one model at a time, which avoids edges that have high predicted model error.

IV. Experiments

For experiments, we use SoftGym as a stand-in for the real world [22]. The initial poses of the objects are fixed. The action space for controls in SoftGym is $(dx, dz, d\theta)$. One task is to move the control box to a target location. The other task is to pour a desired amount of water into the target box. (Figure 3). The state space used in planning is a vector of $[x_{cc}, z_{cc}, \theta_{cc}, w_{cc}, w_{tc}]$ where cc denotes the control box and tc denotes the target box. $(x_{cc}, z_{cc}, \theta_{cc})$ is the pose of the control box. The initial poses of both objects are fixed for simplicity. The box is initialized with a fixed volume of liquid.

Tasks: The first task, `BoxAtLocation` is to move the box to a target pose (x_d, z_d) sampled uniformly from a 0.5 m by 0.6 m box within a 0.05 m error. The second task,

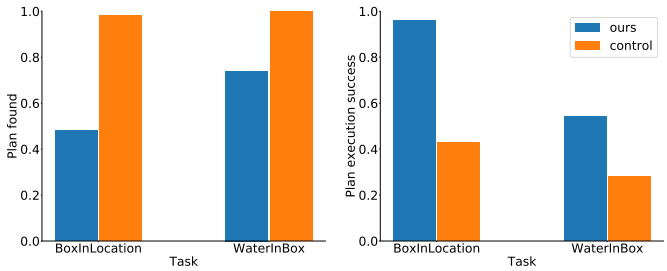


Fig. 4: Plan performance of planning using model preconditions (ours) and without (control). Left shows success rate finding plans, right shows success executing the plan and achieving the goal given that a plan was found

`WaterInBox`, is to pour at least a given volume sampled from $\mathcal{U}(20, 40)\%$ of water in the target container within 5% of the total liquid volume. The agent is permitted to spill no more than 1% of the total liquid for both tasks.

Skills: *MovePourer* moves the box to (x_d, z_d) along a linear trajectory of a fixed number of steps. *Pour* requires that the base of the pourer be within a pre-defined region, and rotates the control box to θ_d then back to 0.

Models: In this work, we use one model per skill, primarily using model preconditions for reliable trajectories rather than to reduce planning time. The model used for *MovePourer* is a kinematic model that assumes that the container moves to the desired pose, and the amount of water in and out of the container does not change. We use the scikit-learn implementation of Random Forest Regression using a maximum depth of 20 and 100 trees [23] to learn a model for *Pour*.

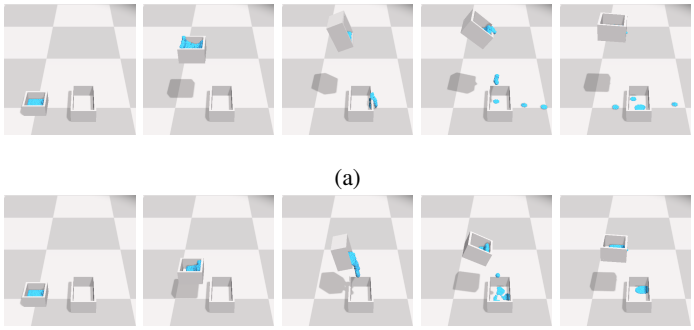
Data collection: For our experiments, we set $N_f = 20$ and $N_{MDE} = 100$, leaving 10% of samples for validation. For the first round of data collection, the model for *Pour* predicts that half the glass is filled to ensure plans can be found. Each action is unit cost.

We test our planner using model preconditions against a control that does not use model preconditions but uses a learned model. 50 goals for each task are sampled for the test set. $\beta = 1$ for all skills and $d_{\max} = 0.08\ m$ for *MovePourer* and $d_{\max} = 0.3$ for *Pour*.

V. Results

Our results in Figure 4 show improvement in task success rate for both tasks when using model preconditions, though the planner that uses model preconditions is less likely to compute a plan than the control is. The trajectories found using our planner predict that for all (s, a) in the result trajectory, $(s, a) \in \text{pre}(M)$, so M is predicted to model $M(s, a)$ accurately. Thus, the space of possible trajectories is smaller, but the trajectories found are more likely to be executed successfully. This is especially apparent for `BoxInLocation` since the goal needs to be within 0.05 m and poses are sampled randomly.

There is still much room for improvement in the plan success rate using model preconditions, as these results show only one round of MDE and model learning. We attribute the MDE inaccuracy as due to data distribution shift. As



(a)

(b)

Fig. 5: Example trajectories where model is inaccurate (top) and accurate (bottom). Water is shown as particles for clarity.

model preconditions are applied in planning, the state-action distribution changes to parts of state-space that are far from $\mathcal{D}_{MDE}$. When using the GP, (s, a) pairs that are very different from those seen in training are predicted to be the mean, which is high deviation for our dataset.

Qualitatively, we see that the model preconditions hold for actions with a smaller distance covered and selects pours at a shorter height. This is because the dynamics are more accurate for the available models for those actions. Spillage is more difficult to model for the learned pour model, and not modelled at all by the rigid-body model for *MovePourer*. For instance, refer to Figure 5, which shows a trajectory where the model inaccurately predicts the particles will end up in the target container and not spill to the floor.

Conclusion

We show a proof-of-concept implementation of planning using learned model preconditions in two water transportation tasks that use analytical models and learned models. Our preliminary results show improved task completion success when plans are found even with MDEs that make inaccurate estimates of the model error. In future work, we will employ better online data collection to more efficiently gather data for MDEs and learned dynamics models. As the model improves, the set of state-action pairs that satisfy the model precondition will increase. As a result, the success rate in finding plans should improve. When the planner is not finding plans, selecting samples can be improved by building on the SEM data generation procedure in [20]. Additionally, we will explicitly model uncertainty in d, which would enable explicit reasoning about dynamics uncertainty. The target domain for this work in predicting model error is to improve the reliability of long-horizon deformable object manipulation problems executed in the real world.

Acknowledgments

We thank Peter Mitrano, Ada Taylor, and Anirudh Vemula for feedback and discussions.

References

[1] H. Yin, A. Varava, and D. Kragic, "Modeling, learning, perception, and control methods for deformable

object manipulation," *Science Robotics*, vol. 6, no. 54, p. eabd8803, 2021.

[2] T. Chen, Y. Huang, and Y. Sun, "Accurate pouring using model predictive control enabled by recurrent neural network," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 7688–7694, 2019.

[3] 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.

[4] Z. Pan and D. Manocha, "Motion planning for fluid manipulation using simplified dynamics," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4224–4231, 2016.

[5] P. Mitrano, D. McConachie, and D. Berenson, "Learning where to trust unreliable models in an unstructured world for deformable object manipulation," *Science Robotics*, vol. 6, no. 54, 2021.

[6] 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*, vol. 5, no. 2, pp. 3540–3547, 2020.

[7] A. Vemula, Y. Oza, J. A. Bagnell, and M. Likhachev, "Planning and execution using inaccurate models with provable guarantees," in *Proceedings of Robotics: Science and Systems (RSS '20)*, July 2020.

[8] 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.

[9] G. Chou, N. Ozay, and D. Berenson, "Model error propagation via learned contraction metrics for safe feedback motion planning of unknown systems," *ArXiv*, vol. abs/2104.08695, 2021.

[10] C. Knuth, G. Chou, N. Ozay, and D. Berenson, "Planning with learned dynamics: Probabilistic guarantees on safety and reachability via Lipschitz constants," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5129–5136, 2021.

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

[12] T. L. Guevara, N. K. Taylor, M. U. Gutmann, S. Ramamoorthy, and K. Subr, "Adaptable pouring: Teaching robots not to spill using fast but approximate fluid simulation," in *Proceedings of the Conference on Robot Learning (CoRL)*, vol. 2, 2017.

[13] T. Tsuji and Y. Noda, "High-precision pouring control using online model parameters identification in automatic pouring robot with cylindrical ladle," in *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 2563–2568, IEEE, 2014.

[14] 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.

[15] Y. Li, J. Wu, R. Tedrake, J. B. Tenenbaum, and A. Torralba, "Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids," in *International Conference on Learning Representations*, 2018.

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

[17] Z. Li and A. B. Farimani, "Graph neural network-accelerated lagrangian fluid simulation," *Computers & Graphics*, vol. 103, pp. 201–211, 2022.

[18] P. Holl, V. Koltun, and N. Thuerey, "Learning to control pdes with differentiable physics," *arXiv preprint arXiv:2001.07457*, 2020.

[19] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artificial intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[20] J. Liang, M. Sharma, A. LaGrassa, S. Vats, S. Saxena, and O. Kroemer, "Search-based task planning with learned skill effect models for lifelong robotic manipulation," *CoRR*, vol. abs/2109.08771, 2021.

[21] C. K. Williams and C. E. Rasmussen, *Gaussian processes for machine learning*, vol. 2. MIT press Cambridge, MA, 2006.

[22] X. Lin, Y. Wang, J. Olkin, and D. Held, "Softgym: Benchmarking deep reinforcement learning for deformable object manipulation," in *Conference on Robot Learning*, pp. 432–448, PMLR, 2021.

[23] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.