Learning to Plan with Optimistic Action Models

Caris Moses, Leslie Pack Kaelbling, and Tomás Lozano-Pérez

***Abstract*— Planning for and successfully executing manipulation tasks require accurate dynamics models. Existing methods for engineering these models often fail to capture the underlying complex object interactions present in many tasks such as tool-use. Therefore, in this work we leverage a data-driven approach to acquiring action models. Data collection on a robotic platform can be time and cost prohibitive, and randomly executing actions is unlikely to elicit informative behavior useful for learning. Therefore, we propose an active learning strategy which aids the robot in learning action models quickly, with the ultimate goal of using them within a planner. Additionally, we supply the robot with initial optimistic action models which are a relaxation of the underlying unknown action model. Optimistic models have the added benefit of being easier to specify than fully accurate action models. In this work, we present an active learning strategy which leverages optimism and give results in a tool-use domain.**

I. Introduction

Planning for long-horizon manipulation tasks requires models which accurately capture the preconditions and effects of a robot's actions. This poses a challenge due to the underlying complexity of the world and robotic system. Issues such as poorly modeled object properties and interactions can lead to a model mismatch between the planning model and the real world. To this end, integrating machine learning (ML) techniques with planning methods has become an effective approach [1], [2], [3], as ML can be leveraged to learn accurate models of complex dynamics. By augmenting a planning system with the ability to learn from data, we can develop robots capable of overcoming issues associated with model mismatch.

The space of plans that a robot has available to explore is prohibitively large, and acquiring data on both real and simulated platforms can be time and cost prohibitive. We leverage an active learning strategy which enables an agent to be intentional in how training data is collected in order to learn efficiently. We also provide additional structure to the learning problem in the form of *optimistic* action models. Specifically, an optimistic model has the property that the set of states reachable under an optimistic model is larger than the set of states reachable under the true unknown action model. These models can also be thought of as a relaxation on the true underlying dynamics. For example, an optimistic pick action model might predict that the robot is able to pick up an object from any location, ignoring kinematic constraints or object properties such as weight. Optimistic models are easier to specify than fully accurate transition models as they allow one to ignore the complex conditions under which an action can be successfully executed. We propose to learn feasibility models which predict the probability that the optimistic model is correct.

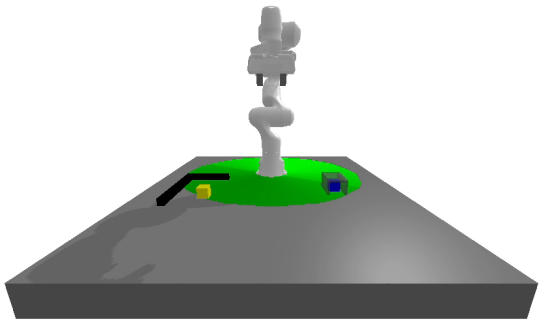


Fig. 1: Tool-Use Domain – The robot is given a tool to interact with a yellow block, whose weight impacts the robot's actions, and a blue block, which begins in an unmodeled tunnel. Details of the domain are given in Section IV-B.

In this work we (1) give a formalism of optimistic action models and how they can be augmented with feasibility models to generate feasible plans, (2) evaluate an active learning strategy for learning feasibility models by generating a rich search space of plans leveraging optimism and goal-directed planning, and (3) give results in a tool-use domain (shown in Figure 1).

II. Problem Formulation

A robot operates in an observable state space S and can take actions in action space A. It operates under an *unknown* true transition function $f_T : S \times A \rightarrow S$ and is given an *optimistic* transition function, $f_O : S \times A \rightarrow S$.

The set of states reachable under a transition function, f, from a state s_{i-1} are

$$S(f, s_{i-1}) = \{s_i \mid \exists a \text{ s.t. } s_i = f(s_{i-1}, a)\}. \quad (1)$$

When we say that f_O is an optimistic version of f_T, we mean that $S(f_T, s) \subseteq S(f_O, s)$. Intuitively, f_O believes that the robot can transition into more states than it actually can within a single action step. Planning with f_O may allow plans to be constructed that will not actually be executable under the true transition dynamics.

Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, USA

{carism, lpk, tlp}@mit.edu

We gratefully acknowledge the funding support of Accenture and the Honda Research Institute.

The authors would like to thank Michael Noseworthy for invaluable discussions relating to the ideas presented.

We would ultimately like to find a plan, $a^*_{1:N}$, which achieves a given goal state, $s_{goal} \in S$, from an initial state, s_0, using the following objective function,

$$a^*_{1:N} = \underset{a_{1:N} \in P}{\operatorname{argmax}} \mathbb{1}_{\{s=s_{goal}\}}(s_N). \quad (2)$$

Generating plan search space P using the optimistic model, f_O, would generate many infeasible plans leading to likely infeasible solutions. To solve Equation 2 effectively, we need to know which plans in P are actually feasible. To do this we need the help of an additional model, a probabilistic classifier, which indicates the probability that the optimistic model's prediction is correct. We refer to this classifier as a feasibility model, $f_\Theta : S \times A \to [0, 1]$.

With both an optimistic model, f_O, and feasibility model, f_Θ, at our disposal, we can weigh plans by their feasibility, resulting in solutions which are both feasible and achieve the goal.

$$a^*_{1:N} = \underset{a_{1:N} \in P}{\operatorname{argmax}} \left(\prod_{i=1}^{N} f_\Theta(s_{i-1}, a_i) \right) \mathbb{1}_{\{s=s_{goal}\}}(s_N) \quad (3)$$

Note that we often reference plan space P made up of plans of length N. However in our work this space of plans consists of plans of varying lengths.

A. Learning Problem

We propose to learn an approximation of the feasibility model, $\hat{f}_\Theta : S \times A \to [0, 1]$, by performing supervised learning. The prediction is probabilistic to account for our epistemic uncertainty, or the uncertainty we have over our predictions. We also maintain distributions over the feasibility models' learnable parameters, $\Pr(\Theta)$, which comes in handy when deciding what data to acquire, and will be discussed in Section III.

During data collection the robot takes an action a_i, from state s_{i-1}, resulting in a state, s^T_i, under the true unknown dynamics, and receives feasibility label

$$\phi = \begin{cases} 1 \text{ if } \textsc{close}(s^T_i, s^O_i) \\ 0 \text{ otherwise.} \end{cases}$$

where s^O_is the state which the optimistic model expected to transition to.

For continuous state spaces such as object poses, $\textsc{close}(s^T, s^O)$ evaluates to True if the states are within some distance ϵ of each other. For logical states, $\textsc{close}(s^T, s^O)$ evaluates to True if the logical states match, for example whether or not a robot is holding a given object. The dataset of samples (s, a, ϕ) are used to train the feasibility model, $\hat{f}_\Theta$, which is then used to weight plans in the search space as shown in Equation 3.

A limitation of this problem formulation is that $\hat{f}_\Theta$ is unable to learn the true resulting state when the optimistic model's prediction is incorrect. New action effects not covered by the optimistic model can never be learned.

III. Method

Active learning methods aim to learn an unknown function with as few labeled data points as possible by selecting what model inputs to label based on some acquisition function.

A. Bayesian Active Learning by Disagreement (BALD)

Bayesian Active Learning by Disagreement [4], or BALD, is one such strategy which guides sampling towards parts of the input space where we have both high uncertainty over our predictions and high uncertainty over our model parameters. This requires maintaining a distribution over both model predictions and model parameters. In our work these two quantities are $\Pr(\phi)$ and $\Pr(\Theta)$ respectively. The BALD strategy finds the action, a, which maximizes the following objective, given a state, s,

$$\begin{aligned} \textsc{bald}(s, a) =& \mathcal{H}(\phi \mid \mathcal{D}, s, a) - \\ & \mathbb{E}_{\Theta \sim \Pr(\cdot|\mathcal{D})} \left[\mathcal{H}(\phi \mid s, a; \Theta) \right]. \end{aligned}$$

The derivation of this objective function relies on the underlying problem being submodular [5], or for the information gain of specific actions to have diminishing returns. However, information gain is often not submodular in sequential manipulation tasks. For example, in our tool-use domain (described in Section IV-B), the blue block must be pushed out of the tunnel before it can be manipulated in any other way (e.g. picked). Once the robot has learned that it can feasibly push the blue block out of the tunnel, it can use that action to set up other experiments, even though it will no longer gain information from the initial push.

B. Sequential Actions

One potential solution to adapting the BALD objective to sequential problems which are not submodular would be to sum the information gain over a sequence of actions given initial state, s_0,

$$\underset{a_{1:N} \in P}{\operatorname{argmax}} \sum_{i=1}^{N} \textsc{bald}(s_{i-1}, a_i).$$

However, if all we have access to is the optimistic model for generating plan space P, then this objective ignores the fact that some of these actions may never be experienced due to a subplan being infeasible. In sequential domains where reaching parts of the state space are dependent on the successful execution of previous actions, it is important to consider subplan feasibility. Noseworthy et al. [1] extend BALD to handle this scenario using the following **Sequential** objective,

$$\underset{a_{1:N} \in P}{\operatorname{argmax}} \sum_{i=1}^{N} \left[\left(\prod_{j=1}^{i} f_\Theta(s_{j-1}, a_j) \right) \textsc{bald}(s_{i-1}, a_i) \right], \quad (4)$$

where s_0 is given. When $P = P_{act}$, we refer to it as the **Sequential Actions** objective. P_{act} consists of plans generated with random roll-outs under the optimistic transition model,

$$P_{act} = \{a_{1:N} | \exists s_{0:N} \text{ s.t. } s_i = f_O(s_{i-1}, a_i)\}. \quad (5)$$

C. Sequential Goals

In many manipulation domains, executing random roll-outs to represent the space of potential plans may not provide enough coverage to elicit interesting training data. Random roll-outs of actions often lead to a robot mostly moving around in free space, and rarely in plans which have the robot making contact with other objects, grasping objects, etc. Noseworthy et al. [1] are able to generate a rich search space, Equation 5, with the help of a useful action space abstraction (a single stack action in a block stacking domain). However for problems which cannot be reduced to an abstract action space of a single action, more sophisticated plan space generation methods are required.

As such, the **Sequential Goals** method uses a different plan space, $P = P_{goal}$, in Equation 4. Now we supply the robot with a space of goals, $S_{goal} \subset S$, which it can sample from and use within a planner to generate a rich optimization landscape for active learning

$$P_{goal} = \{a_{1:N} | \exists s_{goal} \in S_{goal}. \text{ s.t. } a_{1:N} = \text{PLAN}(f_O, s_0, s_{goal})\}. \quad (6)$$

A set of goals are sampled, and the optimistic model is used in a planner to generate plans which provide the search space for optimization of the **Sequential** objective.

IV. Implementation

A. Planning

Due to the limitations of most current planners in coming up with a set of diverse plans for achieving a given goal, we use a skeleton-based planner to implement PLAN in Equation 6. Our skeleton-based planner takes in a transition model defined in the Planning Domain Description Language (PDDL), an initial and goal state both represented with logical predicates, and a plan skeleton. While we do not directly use PDDLStream [6] for planning, many of its tools were useful in developing our skeleton-based planner.

B. Tool-Use Domain

We give results in a tool-use domain, implemented in PyBullet [7], shown in Figure 1. This domain was inspired by related works in robotic manipulation [8], [9]. The robot is given a tabletop environment with a hook and a yellow and blue block. The yellow block is heavy and cannot be picked up or grasped outside of the green circle. As such, the tool must be used to interact with the yellow block outside of the green circle. The blue block begins inside of an unmodeled tunnel, and the robot must first use the tool to push it out of the tunnel before it can be directly manipulated.

The robot is given the following optimistically modeled actions:

- `MoveContact` allows the robot to manipulate a block using the tool. The action begins with the tool and block in contact with each other. Then the robot believes that the block will move relative to the tool in whichever direction it attempts to push/pull it, ignoring the friction cone constraints which enable successful tool-use.
- `MoveHolding` allows the robot to move between configurations while holding an object. It assumes that once an object is held it can be moved anywhere. While this model does include kinematic constraints, it does not account for object properties which may make moving while holding an object infeasible (e.g. the weight of the yellow block).
- `Pick` allows the robot to pick an object from the table. It assumes that an object can be picked from anywhere which is kinematically within reach. This fails when the robot attempts to pick up the yellow block outside of the green circle, and when it attempts to pick the blue block from its initial position inside of the unmodeled tunnel.

We learn action feasibility models $\hat{f}_{contact}$, $\hat{f}_{holding}$, and $\hat{f}_{pick}$ for these optimistically modeled actions. We also give the planner action models for placing (`Place`) and moving through free space (`MoveFree`) and assume that they are accurately modeled.

For all of our training and evaluation instances the initial state is the same, with the yellow block in the same position within the green circle and the blue block in the tunnel. The goal space, S_{goal}, consists of goal poses on the table for either the yellow or blue block.

The following (abbreviated) plan skeletons are used to generate plans to achieve goal states sampled from S_{goal}. Ungrounded parameters (beginning with a `#`) are grounded at planning time.

- `MoveFree -> Pick(Tool) -> MoveHolding(Tool) -> MoveContact(Tool, GoalObj, GoalPose)`
- `MoveFree -> Pick(GoalObj) -> MoveHolding(GoalObj) -> Place(GoalObj, GoalPose)`
- `MoveFree -> Pick(Tool) -> MoveHolding(Tool) -> MoveContact(Tool, GoalObj, #p0) -> MoveHolding(Tool) -> Place(Tool, #p1) -> MoveFree -> Pick(GoalObj) -> MoveHolding(GoalObj) -> Place(GoalObj, GoalPose)`

The full descriptions of the action models, initial and goal states, and skeletons are given in the APPENDIX.

C. Learning

The input to $\hat{f}_{contact}$ is the (x, y) position of the intended final push pose of the pushed block. The input to $\hat{f}_{holding}$ is the (x, y) position of the intended final pose of the held

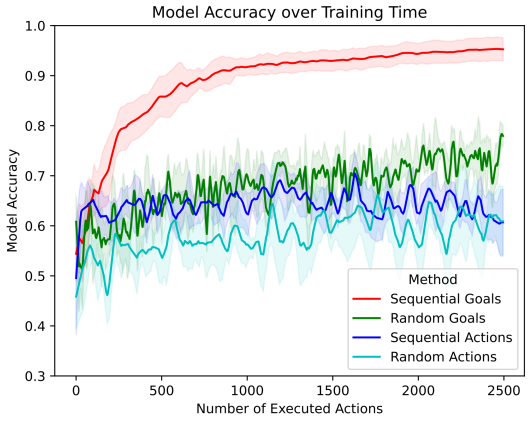


Fig. 2: The joint accuracy of all feasibiliy models being learned. Each line consists of 5 runs. The x-axis is all actions executed by the robot including ones which were not modeled optimistically.

block within the robot's gripper. The input to $\hat{f}_{pick}$ is the (x, y) position of the picked block's initial pose.

We learn a separate model for each of the blocks. In addition, the contact configuration between the pushed block and the tool, as well as the grasp of the tool in hand can impact the success of the `MoveContact` action. As such we learn a separate model for every combination of block, contact configuration, and grasp for the `MoveContact` actions, and just a single model for each block's `Pick` and `MoveHolding` action.

The feasibility models are each represented by an ensemble of Multi-Layer Perceptions (MLPs). We found an ensemble of 20 MLPs to be effective at modeling a distribution over the predictive space.

V. Evaluation

A. Random Baselines

We compare the sequential methods to random baselines. **Random Goals** randomly samples from P_{goal} and **Random Actions** randomly samples from P_{act}.

B. Learning Efficiency

Figure 2 shows the accuracy of the learned feasibility models for all methods. **Sequential Goals** is able to quickly learn the true underlying feasibility of the optimistic models. The performance of both **Sequential Goals** and **Random Goals** shows the usefulness of S_{goal} and using goal-directed plans to generate the optimization search space. The benefits of using the **Sequential** objective to actively acquire trajectories is evidenced by the jump in performance between **Sequential Goals** and **Random Goals** which randomly samples from P_{goal}. **Sequential Actions** and **Random Actions** perform poorly because the plans in P_{act} are not goal-directed and thus consist of many actions which are not useful, such as repeated `MoveFree` actions.

Figures 3 and 4 in the APPENDIX give images of the mean and standard deviation of the predictions made by some of the feasibility model's ensembles learned under the **Sequential Actions** and **Sequential Goals** objectives.

C. Plan Success

To evaluate using our learned feasibility models to plan to achieve goals, we sample a goal state, $s_{goal} \in S_{goal}$, and optimize Equation 3 with the following plan search space

$$P_{goal}(s_{goal}) = \{a_{1:N} | a_{1:N} = \text{PLAN}(f_O, s_0, s_{goal})\}.$$

After executing 2500 actions, each trained feasibility model was given a set of 19 randomly generated goals from S_{goal} and planned to achieve them using Equation 3. The success rate of planning is shown in Table I for each method. While **Sequential Goals** does not perform perfectly it is still markedly better than the baselines. Many of the learned models for **Random Goals**, **Sequential Actions**, and **Random Actions** were unable to find the small feasibly subspace for pushing the blue block (shown in Figure 3). This region opens up the robot to being able to reach a wide range of goal poses for the blue block. This accounts for the majority of the differing performance between our method and the baselines.

Method	Plan Success
Sequential Goals	0.73 ±0.44
Random Goals	0.48 ±0.50
Sequential Actions	0.48 ±0.50
Random Actions	0.20 ±0.40

TABLE I: Planning performance of **Sequential Goals** and all baselines.

VI. Related Work

Methods for learning symbolic action models which are either deterministic [10] or noisy [11] have been explored via various learning strategies such as random goal sampling [10]. The work of Wang et al. [2] and Noseworthy et al. [1] are most similar to ours in that they assume a task and motion planning architecture and learn constraints on continuous parameters of hybrid action models. Silver et al. [3] learn full probabilistic hybrid action models by clustering observed transitions.

A closely related body of work to learning accurate action models for planning, is learning models to aid in planning *given* accurate action models [12], [8]. These methods learn mappings from action types and goals to successful action parameterizations to speed up planning. In Xu et al.'s [8] work this is referred to as an actions' affordance.

In our work we explore learning from *optimsitic* action models, but learning from models which are inaccurate in other ways has also been explored. Lagrassa et al. [13] use model-free reinforcement learning (RL) to augment a given inaccurate planning domain when an unexpected transition occurs. Other methods have been used to learn residual controllers on top of inaccurate controllers, where the inaccuracies stem from either simulation approximations to the real world [14], or poorly learned RL-base controllers [15].

REFERENCES

[1] M. Noseworthy, C. Moses, I. Brand, S. Castro, L. Kaelbling, T. Lozano-Pérez, and N. Roy, "Active learning of abstract plan feasibility," in *Robotics: Science and Systems (RSS)*, 2021.

[2] Z. Wang, C. R. Garrett, L. P. Kaelbling, and T. Lozano-Pérez, "Active model learning and diverse action sampling for task and motion planning," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 4107–4114.

[3] T. Silver, R. Chitnis, J. Tenenbaum, L. P. Kaelbling, and T. Lozano-Pérez, "Learning symbolic operators for task and motion planning," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 3182–3189.

[4] N. Houlsby, F. Huszar, Z. Ghahramani, and M. Lengyel, "Bayesian Active Learning for Classification and Preference Learning," in *NeurIPS Workshop on Bayesian optimization, experimental design and bandits: Theory and applications*, 2011.

[5] D. Heckerman, J. S. Breese, and K. Rommelse, "Troubleshooting under uncertainty," *International Workshop on Principles of Diagnosis*, 1994.

[6] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "PDDLStream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning," in *International Conference on Automated Planning and Scheduling (ICAPS)*, 2020.

[7] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016–2019.

[8] D. Xu, A. Mandlekar, R. Martín-Martín, Y. Zhu, S. Savarese, and L. Fei-Fei, "Deep affordance foresight: Planning through what can be done in the future," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 6206–6213.

[9] M. A. Toussaint, K. R. Allen, K. A. Smith, and J. B. Tenenbaum, "Differentiable physics and stable modes for tool-use and manipulation planning," *Robotics: Science and Systems (RSS)*, 2018.

[10] R. Chitnis, T. Silver, J. Tenenbaum, L. P. Kaelbling, and T. Lozano-Pérez, "Glib: Efficient exploration for relational model-based reinforcement learning via goal-literal babbling," *AAAI Conference on Artificial Intelligence (AAAI)*, 2021.

[11] H. M. Pasula, L. S. Zettlemoyer, and L. P. Kaelbling, "Learning symbolic models of stochastic domains," *Journal of Artificial Intelligence Research*, vol. 29, pp. 309–352, 2007.

[12] A. Curtis, M. Xin, D. Arumugam, K. Feigelis, and D. Yamins, "Flexible and efficient long-range planning through curious exploration," in *International Conference on Machine Learning (ICML)*, 2020, pp. 2238–2249.

[13] A. Lagrassa, S. Lee, and O. Kroemer, "Learning skills to patch plans based on inaccurate models," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 9441–9448.

[14] A. Ajay, J. Wu, N. Fazeli, M. Bauza, L. P. Kaelbling, J. B. Tenenbaum, and A. Rodriguez, "Augmenting physical simulators with stochastic neural networks: Case study of planar pushing and bouncing," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 3066–3073.

[15] T. Silver, K. Allen, J. Tenenbaum, and L. Kaelbling, "Residual policy learning," *arXiv preprint arXiv:1812.06298*, 2018.

APPENDIX

A. Model Prediction Figures

Figures 3 and 4 give visualizations of the feasibility model ensembles after learning from 2500 actions generated under the **Sequential Actions** and **Sequential Goals** objectives.

B. Domain Details

1) Predicates: The parameters for our actions and predicates are the following: `?o` is an object, `?p` is an object pose, `?g` is a grasp, `?q` is a robot arm configuration, `?t` is a robot arm trajectory in joint space, `?c` is the relative pose between the tool and an object it is in contact with.

The predicates `AtPose`, `AtConf`, and `HandEmpty` model the changing pose of objects, configuration of the robot arm, and state of the robot gripper. When the robot is holding something, the grasp is modeled with `AtGrasp`. `Block`, `Tool`, and `Table` simply mean that the given object is a block tool, or table, respectively. `On` indicates that `?o1` is resting stable on top of `?o2`.

`PickKin` indicates that an object can be picked up without violating kinematic and collision constraints. `HoldingMotion` indicates that a trajectory is collision-free while the robot has an object in its grasp. `ContactMotion` indicates that a pushing action will successfully move the pushed object from one pose to another. The following domain description defines the optimistic transition model, f_O.

2) Optimistic Action Models: The optimistically modeled actions are given here

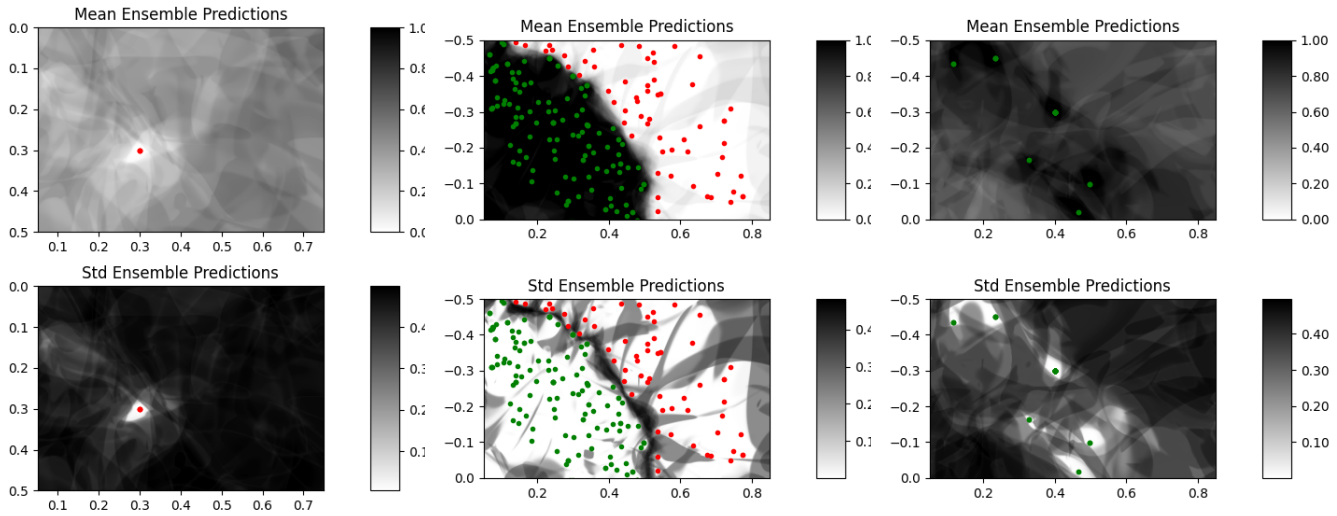
```
(:action Pick
  :param (?o1 ?p1 ?o2 ?g ?q1 ?q2 ?t)
  :pre (and (PickKin ?o1 ?p1 ?g ?q1 ?q2 ?t)
            (AtPose ?o1 ?p1)
            (On ?o1 ?o2)
            (HandEmpty))
  :effect (and (AtGrasp ?o1 ?g)
               (not (AtConf ?q1))
               (not (AtPose ?o1 ?p1))
               (not (HandEmpty))
               (not (On ?o1 ?o2)))
(:action MoveHolding
  :param (?o ?g ?q1 ?q2 ?t)
  :pre (and (HoldingMotion ?o ?g ?q1 ?q2 ?t)
            (AtGrasp ?o ?g))
(:action MoveContact
  :param (?o1 ?c ?o2 ?p1 ?p2 ?g ?q1 ?q2 ?t)
  :pre (and (ContactMotion ?o1 ?c ?o2 ?p1
                           ?p2 ?g ?q1 ?q2 ?t)
            (AtPose ?o1 ?p1)
            (AtGrasp ?o2 ?g)
            (Block ?o1))
  :effect (and (AtPose ?o1 ?p2)
               (not (AtPose ?o1 ?p1)))
```

These actions have additional preconditions and effects which state that the robot configuration changes from `(AtConf q1)` to `(AtConf q2)`, but we have removed them for brevity.

3) Problem Specification: A problem is specified by an initial and goal state. For all of our training and evaluation instances the following initial state is the same with the yellow block in the same position within the green circle, and the blue block in the tunnel,

```
((AtPose YellowBlock YellowBlockInitPose) ∧
(AtPose BlueBlock BlueBlockInitPose) ∧
(AtPose Tool ToolPose) ∧
(Block YellowBlock) ∧
(Block BlueBlock) ∧
(Tool Tool) ∧
(Table Table)).
```

The same goal space, S_{goal}, is used during training and evaluation, and consists of `(AtPose GoalObj GoalPose)` predicates where `GoalObj` can be either `YellowBlock` or `BlueBlock` and `GoalPose` is a randomly sampled pose on the table.



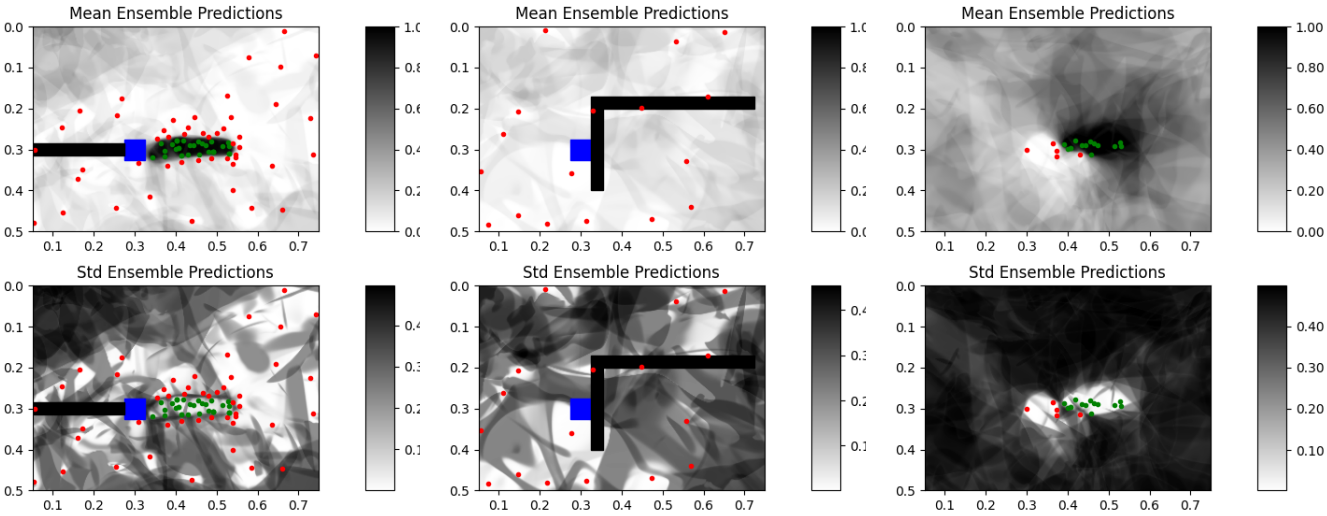
(a) Blue Block $\hat{f}_{pick}$ (b) Yellow Block $\hat{f}_{holding}$ (c) Yellow Block $\hat{f}_{pick}$

Fig. 3: These plots show the ensemble predictions of the feasibility models learned from a single run of the **Sequential Actions** strategy. The grayscale color indicates the model's prediction for the given (x, y) input (see Section IV-C for the model inputs). In the top mean prediction plots, a value of 1.0 indicates feasible action parameters, and 0.0 indicates infeasible action parameters. In the bottom standard deviation plots the value indicates the standard deviation of the ensemble's prediction for a given feasibility model input. The plotted points are the action parameters which were executed by the learning strategy and were found to be either feasible (green) or infeasible (red). Only the actions that were used to train the feasibility models are visualized. Many other actions (such as `MoveFree`) were executed, but not used to learn an feasibility model and thus are not visualized here. For the blue block, the robot attempts to pick it from its initial position, but finds that that action is infeasible. It never explores plans to first push the blue block out of the tunnel, therefore no other pick actions are ever successful. For the yellow block the robot is able to successfully pick it up from its initial position. It is then able to learn that some `MoveHolding` actions are feasible while others are not. The **Sequential Actions** strategy leads the robot to execute a plan which involve picking, placing, then picking up the yellow block again. This is shown by the `Pick` attempts which do not correspond to the yellow block's initial position. No `MoveContact` actions are ever attempted due to the difficulty of randomly sampling a successful tool use plan when generating P_{act}.

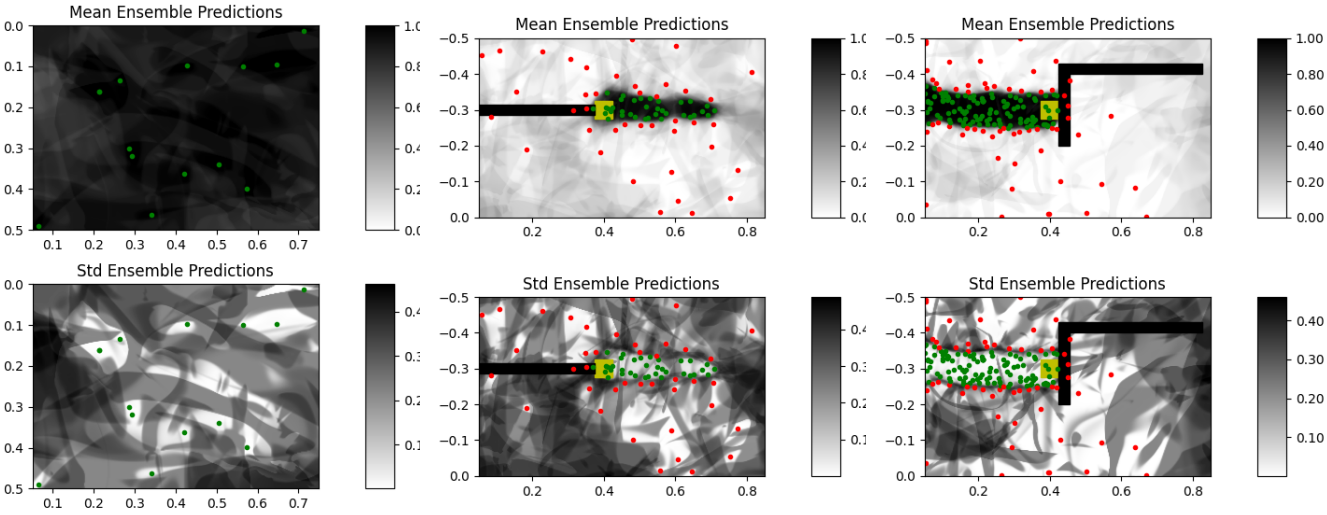
4) Skeletons: A plan skeleton is a sequence of actions where some parameters are ungrounded (denoted with a # symbol). The ungrounded variables are grounded at planning time. In the tool-use domain, the following plan skeletons are used to generate plans to achieve goal states sampled from S_{goal}

- `MoveFree -> Pick(Tool, ToolPose, Table, #g) -> MoveHolding(Tool, #g) -> MoveContact(Tool, #c, GoalObj, GoalObjInitPose, GoalPose, #g)`
- `MoveFree -> Pick(GoalObj, GoalObjInitPose, Table, #g) -> MoveHolding(GoalObj, #g) -> Place(GoalObj, GoalPose, Table, #g)`
- `MoveFree -> Pick(Tool, ToolPose, Table, #g1) -> MoveHolding(Tool, #g1) -> MoveContact(Tool, #c, GoalObj, GoalObjInitPose, #p1, #g1) -> MoveHolding(Tool, #g1) -> Place(Tool, #p2, Table, #g1) -> MoveFree -> Pick(GoalObj, #p1, Table, #g2) -> MoveHolding(GoalObj, #g2) -> Place(GoalObj, GoalPose, Table, #g2).`

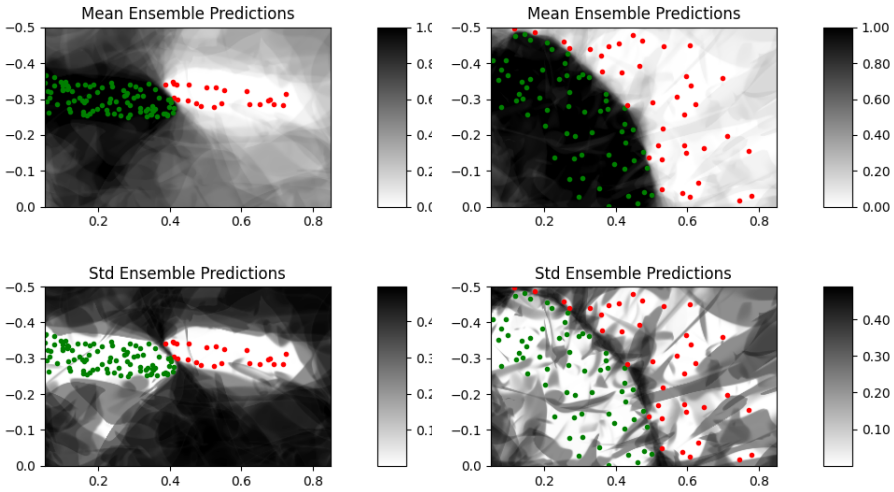
We omit the configuration and trajectory parameters of each action for readability, but these parameters are also grounded at planning time.



(a) Blue Block $\hat{f}_{contact}$ (b) Blue Block $\hat{f}_{contact}$ (c) Blue Block $\hat{f}_{pick}$



(d) Blue Block $\hat{f}_{holding}$ (e) Yellow Block $\hat{f}_{contact}$ (f) Yellow Block $\hat{f}_{contact}$



(g) Yellow Block $\hat{f}_{pick}$ (h) Yellow Block $\hat{f}_{holding}$

Fig. 4: These plots show the ensemble predictions of the feasibility models learned from a single run of the **Sequential Goals** strategy. The grayscale color indicates the model's prediction for the given (x, y) input (see Section IV-C for the model inputs). In the top mean prediction plots, a value of 1.0 indicates feasible action parameters, and 0.0 indicates infeasible action parameters. In the bottom standard deviation plots the value indicates the standard deviation of the ensemble's prediction for a given feasibility model input. The plotted points are the action parameters which were executed by the learning strategy and were found to be either feasible (green) or infeasible (red). For the $\hat{f}_{contact}$ feasibility models we also visualize the contact configuration between the tool, shown in black, and the block, shown in its color in its initial position. From this contact configuration the robot attempts to move it to the different (x, y) positions in the plot. For the blue block $\hat{f}_{contact}$ feasibility model shown in (a), the robot is able to explore feasible pushes which move the block out of the tunnel (past $x = 0.4$). Now, when it attempts picking the blue block, it is occasionally successful. These pick attempts are shown in (c). The $\hat{f}_{contact}$ feasibility model shown in (b) is never feasible due to the tool colliding with the tunnel before it is ever able to reach the desired contact configuration. Once the robot has successfully picked up the blue block, all attempted `MoveHolding` actions are feasible, as shown in (d). For the yellow block the robot is able to see some positive labels for both $\hat{f}_{contact}$ feasibility models shown in (a) and (b). Finally, both $\hat{f}_{pick}$ and $\hat{f}_{holding}$ learn how the weight of the yellow block impacts the success of the `Pick` and `MoveHolding` actions.