# Learning constraint-based planning models from demonstrations

João Loula[1], Kelsey Allen[1], Tom Silver[2], Josh Tenenbaum[1]

***Abstract*— How can we learn representations for planning that are both efficient and flexible? Task and motion planning models are a good candidate, having been very successful in long-horizon planning tasks—however, they've proved challenging for learning, relying mostly on hand-coded representations. We present a framework for learning constraint-based task and motion planning models using gradient descent. Our model observes expert demonstrations of a task and decomposes them into modes—segments which specify a set of constraints on a trajectory optimization problem. We show that our model learns these modes from few demonstrations, that modes can be used to plan flexibly in different environments and to achieve different types of goals, and that the model can recombine these modes in novel ways.**

## I. Introduction

To take a simpler case: suppose we want, from a few demonstrations, to learn a model that can efficiently plan to pick a block up and place it in a given position. One easy thing to do is to learn a step-wise model of the task, and plan by rolling that model forward until we arrive at the goal state—when state and action spaces are large, or planning horizons are long, this is like searching for a needle in a haystack. Another approach [1] involves having our model be differentiable, so that we can optimize our trajectory by doing gradient descent through it. The problem with that is that the task itself is non-differentiable as it involves creating contacts, and so any action around the initial state has no influence on the block's position, giving us no gradient information.

How then can we represent models in a way that is more amenable to planning? An avenue that has proved promising is to define a model in pieces, each of which is smooth, such that planning can happen by first defining the sequence of pieces to use and then optimizing a trajectory based on them. [3], for instance, do so by having the model pieces be handcoded constraints that aid in a trajectory optimization procedure.

This approach, though it has found great success in long-horizon planning, presents a challenge for learning: whereas learning a step-wise model comes down to simply collecting state-action pairs $(s_t, a, s_{t+1})$ and building a self-supervised regressor $f(s_t, a) = s_{t+1}$, learning a hybrid representation like the one handcoded in [3] has proved more difficult. In this paper, we present a model that learns these piece-wise constraint representations from expert demonstrations. We show that representations of this form allow our model to learn from few examples and to plan in out-of-distribution tasks, as well as creatively recombine pieces of solutions to previous problems.

[1]João Loula, Kelsey Allen, and Josh Tenenbaum are with the Department of Brain and Cognitive Sciences, MIT, Cambridge, MA 02139, USA jloula@mit.edu, krallen@mit.edu, jbt@mit.edu

[2]Tom Silver is with the Department of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, USA tslvr@mit.edu

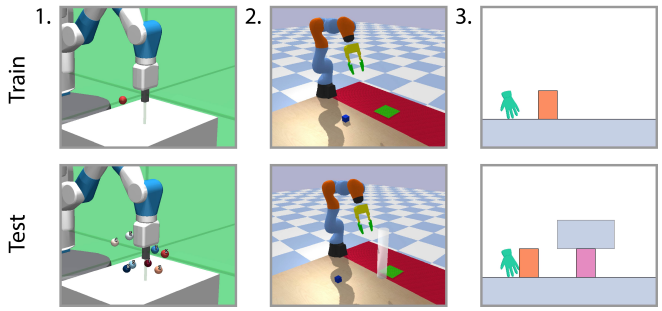


Fig. 1: The three experiments used in this paper, requiring the model to learn the gist of an action in order to adapt it at test time. **1:** At train time, the model observes demonstrations of an expert reaching for a goal in OpenAI Gym's *Reach* environment [2]. At test time, the model has to plan for reaching a sequence of goals at specific times. **2:** At train time, the model observes demonstrations of an expert picking up a block and placing it on a green mat. At test time, the model must plan to place the block on the mat while avoiding an obstacle. **3:** At train time, the model observes demonstrations of an expert either picking up or pushing a block. At test time, the agent must plan to place one block above another by combining pushes and picks.

## II. Previous work

**Task and motion planning:** describes high-level representations of skills or models which are then parametrized [4], [5], [6], [7]. Planning happens by first searching for a subset of handcoded representations to be used and then optimizing the continuous parameters of these representations. These are the family of planning models we're trying to learn from data, specifically the mode-based approach of [3], which we explain in detail in the next section.

**Reconciling learning and symbolic planning:** [8] use deep learning as a way of performing search over symbolic spaces, without dealing with the low-level action space. [9] constrain the learning problem by introducing symbolic representations, such as deictic references, but do not show that their learned models are useful for planning. [10] similarly use a supervision signal in order to break a task into smaller parts that can be recomposed, but their components are fixed policies rather than pieces of a model, which restricts the flexibility with which they can be used in planning.

[11] also unsupervisedly segment demonstrations in order to learn piecewise representations of them, but their pieces are dynamic motion primitives (DMPs) fit to the specific trajectory observed, and their planning procedure is restricted to finding the most similar demonstration and using its DMPs.

**Learning constraints:** previous work on learning physics constraints has mostly relied upon using pre-built sets of constraints. Seminal work by [12] introduced a model that learned rule-like representations for actions in a blocks world by enumerating primitives from a pre-specified grammar. More recently, [13] introduce a model for one-shot learning of geometric constraints, but unlike our work it does so by matching a given trajectory to existing constraints stored on a pre-built knowledge base. The approach most similar to our work is [14], where a planning model also learns optimization constraints using gradient descent. However, the authors' approach is limited to box constraints, i.e. scalar bounds on a single feature,which excludes all the constraints our model attempts to learn—such as contacts and spatiotemporal continuity—which consist of relationships between multiple features.

**Alternative approaches:** Contact Invariant Optimization [15] can be seen as similar in spirit to task and motion planning, with the high-level representations being the scalar variables that indicate which constraints are active, and planning happening through the gradual enforcement of constraints. In the literature attempting to learn to do model-based planning, there are approaches based on using differentiable forward models and planning by backpropagating through them [1], [16], which hinges upon the environment not having non-smooth transitions such as contacts. A different line of work has attempted to solve this problem with forward model rollouts not by introducing higher-level representations and symbolic plans, but rather by learning to shape value functions so as to make search more efficient [17], [18]. Lastly, it's important to note that model-free algorithms have enjoyed success in domains where it is hard to conceive of an approach to planning using stable modes, such as complex dexterous manipulations [19].

## III. Background: Planning

The main contribution of this paper is a framework for learning from demonstrations the representations required to plan with the model introduced by [3] (in that work, like most in the task and motion planning literature, the representations were hand-coded). The learning framework will be presented in the next section; in this section, we briefly introduce the planning model itself.

In the spirit of constraint-based task and motion planning approaches, we formalize our approach to planning as a trajectory optimization problem, subject to initial, goal, and mode constraints. More precisely, we are searching for a trajectory $\hat{x} \in R^{kT}$—where $k$ is the number of features (such as object positions, velocities, and gripper opening) and $T$ is the number of timesteps in the trajectory—that minimizes an optimization problem of the following form:

$$\hat{x} := argmin_x \int_0^T C(x_t)\, dt$$
$$\text{subject to initial constraints } f_i(x_0) \leq 0,$$
$$\text{goal constraints } f_g(x) \leq 0,$$
$$\text{and mode constraints } f_m(x_{[t_m:t_{m+1}]}) \leq 0,\ \forall m,$$

where we suppose some smoothness condition on the $f$s such that, even though dynamics may not be smooth between modes (for instance in the moment we start pushing an object), they will be smooth inside any given mode, allowing for efficient trajectory optimization.

Thus, planning proceeds at two levels, as shown in figure III: at the high level, the model decides on a sequence of modes using breadth-first tree search; at the low-level, each node in the tree becomes a trajectory optimization problem, whose constraints are defined by sequence of modes in that node. The planner returns the first sequence of nodes that arrives at a feasible low-level trajectory.

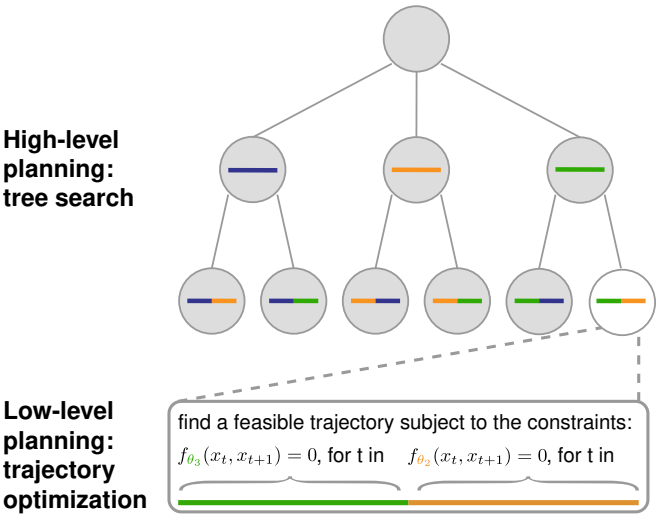


Fig. 2: A schematic of the planning procedure used in this paper, which is the one introduced by [3] with a few small changes. Planning happens at two levels: at the high level, the model decides on a sequence of modes using breadth-first tree search; at the low-level, each node in the tree becomes a trajectory optimization problem, whose constraints are defined by sequence of modes in that node. The planner returns the first node that arrives at a feasible low-level trajectory.

One important difference between the planning model we implement here and that of [3] is that we do away with preconditions for mode transitions. Interestingly, for the tasks we present here, this choice doesn't alter the generated plans. To see why, imagine a pushing task consisting of two modes—**Mode 1** constrains the block to be at rest, and **Mode 2** specifies a horizontal contact between gripper and block. A model can specify that a precondition for **Mode 2** is that

the gripper be next to the block, but it can also leave it to the trajectory optimization procedure to verify that the only feasible trajectories are the ones where **Mode 1** ends with the gripper next to the block. This makes learning easier, as it spares us from dealing with the hard task of learning preconditions, but it makes planning harder, as we lose the possibility of pruning many mode sequences, and the work of verifying preconditions is in practice still done, but offloaded to the optimizer. For the kinds of tasks we're interested in in this paper (our experiments include plans with at most five modes), this is a trade-off we're happy to accept.

## IV. Learning mode-based models

This paper's main contribution is a framework for learning a model like the one in section III from demonstrations: see figure 3 for an overview.

To start out, we note that by parametrizing a smooth function that goes from the space of observation pairs $(x_t, x_{t+1})$ to the real numbers, we can express a parametrized constraint as

$$f_\theta(x_t, x_{t+1}) = 0.^1$$

Though there are many other possible choices of domains for the learned constraints, our choice of observation pairs is motivated in a few different ways. On the one hand, it defines a very expressive space, encompassing for instance all stepwise forward models $x_{t+1} = g(x_t)$ (it suffices to set $f(x_t, x_{t+1}) = x_{t+1} - g(x_t)$) while also being able to express more abstract relations such as collision avoidance, contacts, and joints. On the other hand, since each such constraint is time-independent, our learning problem can benefit from weight sharing through time, greatly reducing the number of parameters to be learned.

In our model, these parametrized functions will stand in for the mode constraints presented in section III—this represents the smooth part of the optimization problem. The non-smooth part comes from the mode changes, making it so that different constraints hold at different points in time. At train time, we assume that all the observed demonstrations follow the same sequence of modes. We'll call *switch times* the timesteps when a change in mode occurs, and use $s_j$ to denote the switch time between modes $m_{j-1}$ and $m_j$. Thus the constraints that mode $m_j$ places on the trajectory $x$ are given by

$$f_{\theta_{m_j}}(x_t, x_{t+1}) = 0, \forall t \in [s_j, s_{j+1}]$$

(in practice, we learn multiple constraints for each mode—we omit this here for clarity of notation.) Finally, our goal is to find the constraint parameters $\theta$ that minimize, across all trajectory demonstrations $x^k_{0:T}$, the following loss function:

$$\mathcal{L}_{reconstruction}(x, \hat{x}) = \sum_k ||x^k - \hat{x}^k(\theta)||_2,$$

$$\text{where } \hat{x}^k(\theta) := \; argmin_x \int_0^T C(x_t)\, dt$$

$$\text{subject to initial constraints } f^k_i(x_0) \leq 0,$$

$$\text{goal constraints } f^k_g(x) \leq 0,$$

$$\text{and mode constraints } f_{\theta_{m_j}}(x_t, x_{t+1}) = 0,$$

$$\forall j,\ \forall t \in [s^k_j, s^k_{j+1}].$$

Solving for $\theta$ is a hard optimization problem, as it requires doing inference through an $argmin$ procedure. In order to do so, we use the approach introduced by [20], where, by restricting the optimization problems solved at train time to quadratic programs we can obtain analytical gradients and learn $\theta$ using gradient descent. Note that, though this approach restricts us to quadratic program optimization problems at train time, at test time we can use the learned constraints in an off-the-shelf optimizer to solve any type of optimization problem we wish—see the experiments section for examples solving quadratically-constrained quadratic programs using SNOPT [21], and mixed-integer quadratic programs using Gurobi [22].

This approach requires inferring the mode switch times $s_j$: though in principle this could also be learned by gradient descent, in practice the fact that it is a discrete decision makes it challenging. Since the problems we are interested in here have simple dynamics involving only a few objects, infer switch times using a standard top-down segmentation algorithm from the time-series analysis literature [23]. The basic idea is to pick a regression function class that well approximates the smooth parts of your data, but poorly approximates the non-smooth parts. Then, greedily add a split point $s$ such that regressing each resulting segment will result in the lowest overall error, and repeat until a certain error threshold is met. Since we're interested in learning linear constraints, we chose linear functions as our regression class, which in practice worked well for inferring segments in our experiments—we stress that inferring mode switches in more complex environments would require more general approaches to segmentation, which is an interesting and open problem in itself and beyond the scope of this paper. When planning at test time, we follow [3] and set the segments to have equal lengths.

## V. Why choose constraints as a high-level representation?

The benefits of hierarchical representations when planning are clear, allowing for long-horizon plans and better generalization. However, many different choices of high-level representation are possible. In this section, we discuss the trade-offs that come with choosing constraints as a high-level representation, and spell out why we think this approach is a good idea.

**Pros**:

[1] We choose to learn only equality constraints in this paper, as they suffice for the problems we are interested in and allow us to reduced the number of learned constraints by half, but our framework can use inequality constraints just as well.

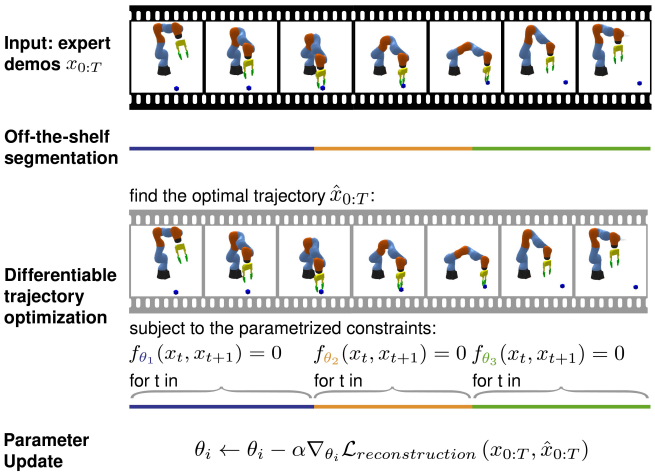


Fig. 3: Our learning framework: the model receives as input a set of expert demonstrations of a multi-step action. Then, an off-the-shelf segmentation algorithm is used to infer the split times between the steps in the demonstration. Next, a differentiable procedure finds the optimal trajectory given the initial state, the goal, the inferred split times and the currently hypothesized constraints for each segment. Finally, the constraint parameters are updated by comparing the reconstructed trajectory to the expert demonstrations.

- By having the goal represented as a constraint, the same model can learn from and generalize to many different types of goals (see our experiments where the model can generalize at test time to multiple goals (experiment 2), or to creating spatial relations between two objects (experiment VI-C).)
- Constraints are easy to specify and easy to compose, allowing us to take our learned representations and use them in an optimization problem written in any off-the-shelf optimizer (see experiment VI-B where we can add obstacles at test time and the previously learned representations can still be used solve the task.)
- Learning constraints is data efficient: a single expert demonstration of a mode lasting T timesteps offers T examples from which to learn the relevant constraints, whereas for other kinds of representations that might yield only one example (see our experiments, all involving fewer than five demonstrations.) [2]

**Cons**:

- Learning constraints is hard, as it involves inferring parameters through an $argmin$ operation.
- A significant part of the burden of modeling the task is offloaded to the optimizer, which means that planning at test time can be expensive for complex tasks.

[2]It's worth noting however that the examples provided by the T timesteps are not i.i.d. with respect to the underlying mode, as they will share spurious correlations due to having the same initial state and goal—that's the reason why we are able to learn from few demonstrations but not from a single, longer demonstration.

## VI. Experiments

We test our approach to learning mode constraints by using the OpenAI Gym Robotics environment [2], a PyBullet [24] environment with a Kuka robot with a WSG 50 gripper, and a custom 2d block manipulation environment (see figure 1, top row.) For all experiments, we train the model on expert demonstrations from a hand-coded expert policy. At both train and test time, the model has access to the initial state constraints $f_i$ and the goal constraints $f_g$, and the cost function is kept fixed throughout as the sum of squared velocities of the gripper. Trajectories are represented as a sequence of discrete timesteps $x_t$, each of which contains features such as the position and velocity for the objects in the environment, including the gripper. The loss function being optimized at train time is the one presented in section IV, and at test time we plan as described in section III.

### A. Multireach: Learning and generalizing a single mode

In this experiment, we first train our model on expert trajectory demonstrations of the *Reach* task (see figure 1, top left). This task consists of a single mode: that means we don't do any segmentation during training, and all the constraints we learn are applied at all timesteps in the trajectory. At test time, the model must do only low-level planning (since the high-level plan consists of a single mode), and use the constraints $f_\theta$ it has learned for this mode to plan in an out-of-distribution *Multireach* task that requires maneuvering to reach multiple spots at different times in the trajectory (see figure 1, bottom left).

***Train:*** the model is trained on 5 expert demonstrations. Each timestep's feature space $x_t$ has dimension 6 (gripper 3d position and velocity), and the trajectories have length 20. The expert is a hand-coded policy that moves towards the goal with constant velocity. The goal is encoded as a constraint enforcing that the gripper be at the specified position on the final timestep.

***Test:*** we test the model's learned constraints in a *Multireach* task, which requires the gripper to do multiple reaches to different goals in sequence. This goal is encoded as a set of constraints enforcing that the gripper be, at evenly-spaced timesteps, at positions sampled from the same distribution as the train goals. Following the scoring in the Gym *Reach* environment, we consider a goal to be reached if the gripper's distance to it is less than 5cm. The total trajectory time is scaled linearly with the number of goals: we test the model on 1, 2, 4, and 8 goals. We solve this optimization problem using OptNet to obtain the full predicted trajectory $x_{0:T}$ and extract from it the control variables—the gripper velocities $v_t$—which we use as a velocity controller in the Multireach environment.

***Results:*** for each number of goals (1, 2, 4, and 8), we test the model across 100 randomly-sampled runs. Each trial's score is the proportion of goals reached. The results are shown in figure 5, on the left. We see that the model that was trained on *Reach* can solve *Multireach* with a large number of goals, and that increasing the number of goals has almost no effect on performance—the small decrease in score comes

from the accumulation of errors, as the controller is not reactive.

We can also inspect the learned parameters for the constraint $f_\theta$ (figure 4). We can see that, from the few trajectories it observes, the model learns a representation of spatiotemporal continuity, i.e. $x_t + v_t - x_{t+1} = 0$, across the 3 spatial dimensions.

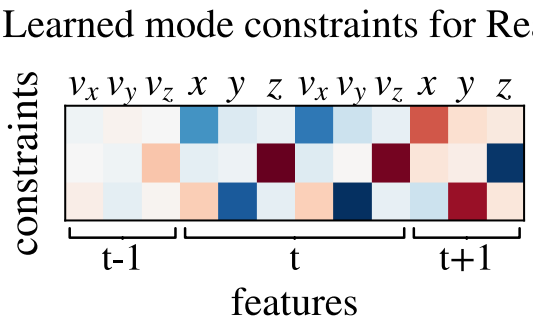


Fig. 4: Learned weights for the mode constraint $f_\theta$ in the *Reach* task, represented on a sparse basis—blue colors represent learned positive weights on a given feature, while red colors represent negative ones (we omit the scale of the values here: it is meaningless as these constraints are linear with zero bias.)

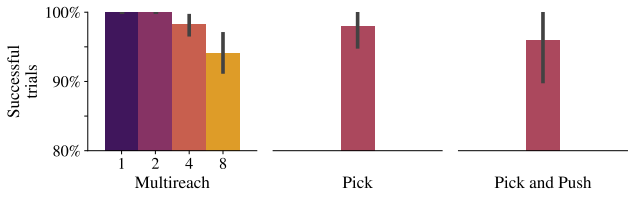


Fig. 5: Model out-of-distribution test performance on the Multireach (experiment VI-A) , Pick (experiment VI-B, and Pick and Push (experiment VI-C) tasks. For Multireach, we plot performance separately for the 1 goal, 2 goals, 4 goals and 8 goals conditions. Error bars are bootstrapped 95% confidence intervals.

| | Multireach | Pick | Pick + Push |
|---|---|---|---|
| # demonstrations | 5 | 5 | 8 |
| # modes in test plan | 1 | 5 | 4 |

TABLE I: Task-by-task breakdown of the number of demonstrations given to the model at train time and the number of modes in the sequence used to solve the test environment.

### *B. Pick: Learning and generalizing a complex mode sequence*

In this experiment, we first train our model on expert trajectory demonstrations of a PyBullet [24] pick and place task (see figure 1, top center). At test time, the model is tasked with solving the pick task in the presence of an obstacle that precludes the kinds of trajectories observed at train time.

***Train:*** the model is trained on 5 expert demonstrations. Each timestep's feature space $x_t$ has dimension 7 (gripper and block 3d positions and the gripper finger opening), and the trajectories have length 50. The expert is a hand-coded policy that moves to a grasping position, closes its fingers, moves towards the goal, and then opens its fingers and moves back to the starting position. The goal is encoded as two constraints: one enforcing that the block be at a uniformly drawn position on the right side of the table (depicted by the green mat on figure 1) on the final timestep, and one enforcing that the gripper be at the starting position on the final timestep.

***Test:*** we test the model's learned constraints in an obstacle pick task, which requires the gripper to maneuver around an obstacle as its moving the block towards the goal position. The presence of the obstacle is encoded as a set of handcoded collision avoidance constraints between the gripper and the obstacle and the block and the obstacle. The obstacle is placed such that it always impedes a straight path between the block's starting position and the mat's center. The goal constraints are the same as in the train setting. Since the sequence of modes for solving the task at train and test time is the same, and it is rather the low-level trajectories that change, we do not perform high-level planning at test time and instead just use the mode sequence inferred at train time as the high-level test plan. We solve this optimization problem as a sequential quadratic program to obtain the full predicted trajectory $x_{0:T}$ and extract from it the control variables—the gripper positions $x_t$ and finger openings—which we use as a position controller for the task.

***Results:*** we test the model across 100 randomly-sampled runs. Each trial's score is the proportion of goals reached. We consider a goal to be reached if the block is within the mat, which is a 2cm-wide square (see figure 1). The results are shown in figure 5, in the center.

### *C. Pick and Push: recombining modes*

In this experiment, we train our model separately on expert trajectory demonstrations of a pick task and a push task (see figure 6, top). At test time, the model is tasked with placing one block on top of another, in an environment where doing so requires combining both the pick and the push modes and navigating around an obstacle. In order to do so, the model must successfully learn the respective mode constraints, and be able to make a successful high-level plan at test time, that combines these modes in a way it has not seen at train time, in order to arrive at a feasible low-level plan.

***Train:*** the model is trained twice on 8 expert demonstrations total: once on 4 demonstrations of the pick action and separately on 4 demonstrations of the push action (see figure 6, top). Each timestep's feature space $x_t$ has dimension 9 (gripper and block 2d positions and velocities and a variable that indicates when the gripper is grasping a block), and the trajectories have length 21. The push expert is a hand-coded policy that moves towards the block and then pushes it towards the goal; the pick expert is a hand-coded policy that moves to the top of the block, turns on the grasping

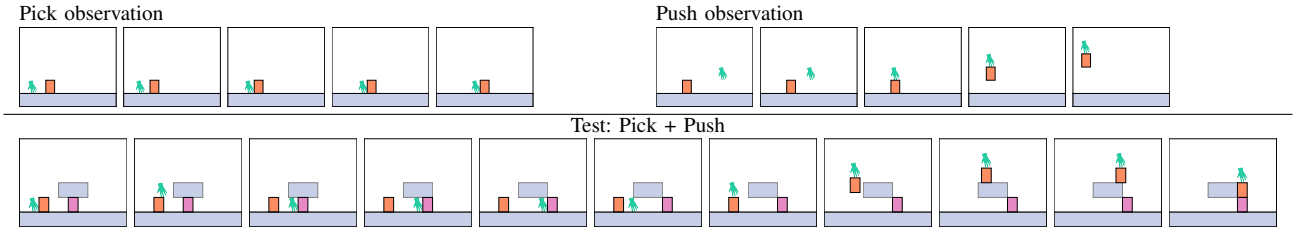


Fig. 6: The model observes an expert trajectory in a pick demonstration (top left) and a push demonstration (top right). It can then plan in a new environment where achieving the goal (placing the orange block above the pink one) requires executing both the pick and the push actions: the trajectory shown at the bottom is a plan executed by the model at test time. Trajectories images were downsampled for illustration.

variable, and then moves the block towards the goal. The goal is encoded as a constraint enforcing that the block be at a uniformly drawn position on the final timestep—for the push demonstrations, that position's y-coordinate is always 0 and to the right of the gripper, such that the gripper is always shown pushing the block from left to right. On this task, we also give the model the spatiotemporal continuity constraint for both the gripper and the block as a fixed constraint that holds on all timesteps.

***Test**:* we take the union of the constraints learned in the pick and in the push train tasks and test them in a novel task, which requires the gripper to maneuver around an obstacle and use both the pick and the push actions to successfully attain the goal of placing the orange block above the pink block. The goal is encoded as a constraint enforcing that the orange block be directly above the pink block; the presence of the obstacle is encoded as a set of handcoded collision avoidance constraints between the gripper and the obstacle and the two blocks and the obstacle[3]. The high-level planning is done by doing breadth-first search over the modes. The low-level optimization problem is solved as a mixed-integer quadratic program, with the model returning the first high-level node in the tree that allows for a feasible low-level trajectory. The gripper velocities and grasp control in the predicted trajectory are used as a controller for the task.

***Results**:* we test the model across 100 randomly-sampled runs. Each trial's score is the proportion of goals reached. Akin to the Multireach environment, we consider the goal to be reached within a 5cm tolerance of the goal constraint. The results are shown in figure 5, in the right. We also show display a successful test trial in figure 6, in the bottom.

## VII. Discussion

We showed that our model can learn, a representation that generalizes to out-of-distribution goals (experiment VI-A), that carves out the smooth segments of a demonstration and can adapt it to a novel environment (experiment VI-B), and that can be composed to create novel high-level plans (experiment VI-C.) It does so from less than 10 demonstrations for each environment (see Table 1)—which is many orders of magnitude lower than most work on learning task and motion planning skills from demonstrations (e.g. [10], [25]), and approaches the data efficiency of methods that deal only with the high-level aspects of skills (e.g. [8].) This is possible due to the choice of constraints as a building-block for high-level representations: building on the work of [3], which established the power of hand-coded constraints for long-horizon hierarchical planning, we showed that they are also a great candidate for learning, being data efficient and allowing for plans that generalize at both the high and the low-level.

## VIII. Limitations and Future Work

A significant challenge in our framework is the necessity of a segmentation procedure to infer the mode switch times. Though our simple environments yielded reliable segmentation results when using the standard time-series segmentation algorithm [23], this is unlikely to hold true when scaling to more complex scenes. Future work could introduce segmentation into the optimization procedure and have it be learned as well.

Another challenge is how to generalize constraints to scenarios involving different objects. The way constraints are represented in our framework doesn't differentiate between information about the kinds of kinematic relationships that are possible (e.g. picking, pushing) and specific geometric properties of objects (e.g. the size of the gripper or the shape of a block): this means that if we swapped a block for a cylinder, for instance, we'd have to learn the relevant constraints all over again. Disentangling these two aspects of constraints and reasoning about them separately would be necessary for this kind of generalization.

Finally, having analytical gradients with which to learn constraints comes at the cost of restricting the space of optimization problems we can learn from at train time to quadratic programs. Though this is a fairly broad class, many interesting problems involve constraints that violate these assumptions. Future work could try to get around this by moving from analytical gradients to gradient estimation methods.

[3]In order to translate the constraints learned at train time, where each timestep's feature space $x_t^{train}$ had dimension 9, to the test environment, where each timestep's feature space $x_t^{test}$ has dimension 13 (because there is an additional block), we set $f_\theta^{test}(x_t^{test}, x_t^{test}+1) = \left(f_\theta^{blockA}(x_t^{blockA}, x_{t+1}^{blockA}), f_\theta^{stationary}(x_t^{blockB}, x_{t+1}^{blockB})\right)$, where $f_{stationary}$ is a constraint enforcing zero velocity for a given block—this means that each constraint learned at train time translates into two constraints at test time, one for each block.

## REFERENCES


[1] E. Todorov, "Convex and analytically-invertible dynamics with contacts and constraints: Theory and implementation in mujoco," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 6054–6061.

[2] M. Plappert, M. Andrychowicz, A. Ray, B. McGrew, B. Baker, G. Powell, J. Schneider, J. Tobin, M. Chociej, P. Welinder, *et al.*, "Multi-goal reinforcement learning: Challenging robotics environments and request for research," *arXiv preprint arXiv:1802.09464*, 2018.

[3] M. Toussaint, K. Allen, K. Smith, and J. Tenenbaum, "Differentiable physics and stable modes for tool-use and manipulation planning," *Proceedings of Robotics: Science and Systems, Pittsburgh, PA*, 2018.

[4] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical planning in the now," in *Workshops at the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[5] M. Toussaint, "Logic-geometric programming: An optimization-based approach to combined task and motion planning." in *IJCAI*, 2015, pp. 1930–1936.

[6] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki, "Incremental task and motion planning: A constraint-based approach." in *Robotics: Science and Systems*, 2016, pp. 1–6.

[7] A. Jain and S. Niekum, "Efficient hierarchical robot motion planning under uncertainty and hybrid dynamics," *arXiv preprint arXiv:1802.04205*, 2018.

[8] D.-A. Huang, S. Nair, D. Xu, Y. Zhu, A. Garg, L. Fei-Fei, S. Savarese, and J. C. Niebles, "Neural task graphs: Generalizing to unseen tasks from a single video demonstration," *arXiv preprint arXiv:1807.03480*, 2018.

[9] V. Xia, Z. Wang, and L. P. Kaelbling, "Learning sparse relational transition models," *arXiv:1810.11177 [cs, stat]*, Oct. 2018, arXiv: 1810.11177. [Online]. Available: http://arxiv.org/abs/1810.11177

[10] J. Andreas, D. Klein, and S. Levine, "Modular multitask reinforcement learning with policy sketches," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 166–175.

[11] S. Niekum, S. Osentoski, C. G. Atkeson, and A. G. Barto, "Online Bayesian changepoint detection for articulated motion models," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 1468–1475.

[12] 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.

[13] C. Pérez-D'Arpino and J. A. Shah, "C-learn: Learning geometric constraints from demonstrations for multi-step manipulation in shared autonomy," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 4058–4065.

[14] B. Amos, I. Jimenez, J. Sacks, B. Boots, and J. Z. Kolter, "Differentiable mpc for end-to-end planning and control," in *Advances in Neural Information Processing Systems*, 2018, pp. 8289–8300.

[15] I. Mordatch, E. Todorov, and Z. Popović, "Discovery of complex behaviors through contact-invariant optimization," *ACM Transactions on Graphics (TOG)*, vol. 31, no. 4, p. 43, 2012.

[16] A. Srinivas, A. Jabri, P. Abbeel, S. Levine, and C. Finn, "Universal planning networks," *arXiv preprint arXiv:1804.00645*, 2018.

[17] A. Tamar, G. Thomas, T. Zhang, S. Levine, and P. Abbeel, "Learning from the hindsight planepisodic mpc improvement," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 336–343.

[18] K. Lowrey, A. Rajeswaran, S. Kakade, E. Todorov, and I. Mordatch, "Plan online, learn offline: Efficient learning and exploration via model-based control," *arXiv preprint arXiv:1811.01848*, 2018.

[19] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine, "Learning complex dexterous manipulation with deep reinforcement learning and demonstrations," *arXiv preprint arXiv:1709.10087*, 2017.

[20] B. Amos and J. Z. Kolter, "Optnet: Differentiable optimization as a layer in neural networks," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 136–145.

[21] P. E. Gill, W. Murray, and M. A. Saunders, "Snopt: An sqp algorithm for large-scale constrained optimization," *SIAM review*, vol. 47, no. 1, pp. 99–131, 2005.

[22] G. Optimization, "Inc.,gurobi optimizer reference manual, 2015," 2014.

[23] E. Keogh, S. Chu, D. Hart, and M. Pazzani, "Segmenting time series: A survey and novel approach," in *Data mining in time series databases*. World Scientific, 2004, pp. 1–21.

[24] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," *GitHub repository*, 2016.

[25] M. B. Chang, A. Gupta, S. Levine, and T. L. Griffiths, "Automatically composing representation transformations as a means for generalization," *arXiv preprint arXiv:1807.04640*, 2018.