TAMPC: A Controller for Escaping Traps in Novel Environments

Sheng Zhong\textsuperscript{1}, Zhenyuan Zhang\textsuperscript{1}, Nima Fazeli\textsuperscript{1}, and Dmitry Berenson\textsuperscript{1}

Abstract—We propose an approach to online model adaptation and control in the challenging case of hybrid and discontinuous dynamics where actions may lead to difficult-to-escape “trap” states. We first learn dynamics for a given system from training data which does not contain unexpected traps (since we do not know what traps will be encountered online). These “nominal” dynamics allow us to perform tasks under ideal conditions, but when unexpected traps arise in execution, we must find a way to adapt our dynamics and control strategy and continue attempting the task. Our approach, Trap-Aware Model Predictive Control (TAMPC), is a two-level hierarchical control algorithm that reasons about traps and non-nominal dynamics to decide between goal-seeking and recovery policies. An important requirement of our method is the ability to recognize nominal dynamics even when we encounter data that is out-of-distribution w.r.t the training data. We achieve this by learning a representation for dynamics that exploits invariance in the nominal environment, thus allowing better generalization. We evaluate our method on simulated planar pushing and peg-in-hole as well as real robot peg-in-hole problems against adaptive control and reinforcement learning baselines, where traps arise due to unexpected obstacles that we only observe through contact. Our results show that our method significantly outperforms the baselines in all tested scenarios.

Index Terms—Machine Learning for Robot Control, Reactive and Sensor-Based Control.

I. INTRODUCTION

In this paper, we study the problem of controlling robots in environments with unforeseen traps. Informally, traps are states in which the robot cannot make progress towards its goal and is effectively “stuck”. Traps are common in robotics and can arise due to a wide variety of factors including geometric constraints imposed by obstacles, frictional locking effects, and nonholonomic dynamics leading to dropped degrees of freedom \cite{1}, \cite{2}, \cite{3}. In this paper, we consider instances of trap dynamics in planar pushing with walls and peg-in-hole with unmodeled obstructions to the goal.

Developing generalizable algorithms that rapidly adapt to handle the wide variety of traps encountered by robots is important to their deployment in the real-world. Two central challenges in online adaptation to environments with traps are the data-efficiency requirements and the lack of progress towards the goal for actions inside of traps. In this paper, our key insight is that we can address these challenges by explicitly reasoning over different dynamic modes, in particular traps, together with contingent recovery policies, organized as a hierarchical controller. We introduce an online modeling and controls method that balances naive optimism and pessimism when encountering novel dynamics.

1) Introduce a novel representation architecture for generalizing dynamics and show how it allows our method to achieve good performance on out-of-distribution data;

2) Introduce Trap-Aware Model Predictive Control (TAMPC), a novel control algorithm that reasons about non-nominal dynamics and traps to reach goals in novel environments with traps;

3) Evaluate our method on real robot and simulated peg-in-hole, and simulated planar pushing tasks with traps where adaptive control and reinforcement learning baselines achieve less than 5% success rate while our method achieves 75% success rate (85% with tuning) averaged across environments and tasks.

Our approach addresses limitations in state-of-the-art techniques \cite{4}, \cite{5} that cannot be expected to perform well in these scenarios because they have little to no contingencies for handling traps – action sequences that escape traps incur high short-term costs and are much less likely to be discovered.

\textsuperscript{1}Robotics Institute, University of Michigan, MI 48109, USA \{zhsh, cryscan, nfz, dmitryb\}@umich.edu

For code, see: https://github.com/UM-ARM-Lab/tampc
II. PROBLEM STATEMENT

Let \( x \in \mathcal{X} \) (\( N_x \) dimensional) denote the state, \( u \in \mathcal{U} \) (\( N_u \) dimensional) denote the control, and \( \Delta x \in \mathcal{X} \) denote the change in state. We assume that we are given a state distance function \( d : \mathcal{X} \times \mathcal{X} \to [0, \infty) \) and a control similarity function \( s : \mathcal{U} \times \mathcal{U} \to [0, 1] \). The state distance function measures changes in state and progress towards the goal implies \( d > 0 \). Let \( f \) denote the nominal dynamics (i.e. for the system under ideal conditions) and \( f_n \) denote the novel dynamics in a novel environment. We assume they are discrete-time in the form:

\[
\Delta x = f_n(x, u) = f(x, u) + e(x, u)
\]

where \( e \) denotes the error dynamics. We assume the error dynamics are relatively small (w.r.t. to the nominal) except for non-nominal regions \( \bar{X} \subset \mathcal{X} \) for which:

\[
|e(x, u)| \sim |f(x, u)|, \quad \forall x \in \bar{X}, \quad \exists u \in \mathcal{U}
\]

We are given some goal-directed cost function \( C \) which induce trap states \( T \subset \mathcal{X} \). These states are basins of attraction for \( C \) inside of which no action sequence below some length \( N > 1 \) can decrease the cost where \( N \) can be considered a given trap’s difficulty. Note that traps are defined by both the environment and cost function.

In trap states, the controller benefits from considering a horizon greater than \( N \) and may need to incur high short-term cost to escape and have a chance at eventually reaching the goal. This is especially difficult when the trap dynamics are unknown; i.e. when \( T \subset \mathcal{X} \). We consider the case where traps are unforeseen and novel environments where \( e \) may be discontinuous, such as in the case of unexpected contact during manipulation. Partial observability and limited sensing significantly increase the difficulty of the problem because they prevent the anticipation and preemptive avoidance of novel traps. The key challenges to accomplishing a task in this scenario are 1) the identification of traps; and 2) a reactive control scheme that can escape from traps and make progress toward the goal.

To learn the nominal dynamics \( f \), we assume that some data has been collected in a nominal environment. We are given \( M \) transition sequences from the nominal environment \( \tau = \{\tau_1, ..., \tau_M\} \). \( \tau_i = \{(x, u, \Delta x)_1, ..., (x, u, \Delta x)_T\} \). Starting from an initial state \( x_0 \) in a novel environment, which may contain traps that were not present in the nominal environment, our objective is to minimize the number of steps \( T \) to navigate to a given goal set \( \mathcal{G} \subset \mathcal{X} \):

\[
\begin{align*}
\arg\min_{u_0, ..., u_{T-1}} & \quad T \\
\text{s.t.} & \quad x_{t+1} = x_t + f_n(x_t, u_t), \quad t = 0, ..., T-1 \\
& \quad x_T \in \mathcal{G}
\end{align*}
\]

III. RELATED WORK

In this section, we review related work to the two main components of this paper: handling traps and generalizing models to out-of-distribution (OOD) dynamics.

Handling Traps: Traps can arise due to many factors including nonholonomic dynamics, frictional locking, and geometric constraints [1], [2], [3]. In the control literature, viability control [6] can be applied to nonholonomic systems in the case where the dynamics of entering traps (leaving the viability set) is known. While they enforce staying inside the viability set as a hard constraint, our method can be interpreted as online learning of the non-viable set with a policy for returning to the viable set.

Another way to handle traps is with exploration, such as through intrinsic curiosity (based on model predictability) [7], [8], [9], state counting [10], [11], or stochastic policies [12]. However, trap dynamics can be difficult to escape from and can require returning to dynamics the model predicts well (so receives little exploration reward). We show in an ablation test how random actions are insufficient for escaping traps in the tasks we consider. Similar to [13], we remember interesting past states and attempt to recover to them before resuming our control algorithm. They require a simulator to reset to the previous state and design domain-specific state interest scores. We do not require resetting, and effectively allow for online adaptation of the state score based on how much movement the induced policy generates while inside a trap.

Generalizing models to OOD Dynamics: Actor-critic methods have long been used to control nonlinear systems with unknown dynamics online [14], [15]. However, they tend to do poorly in discontinuous dynamics and are sample inefficient compared to our method as we show in experimentation. Another approach to handle novel environments online is with locally-fitted models [16], which [4] shows could be mixed with a global model and used in model predictive control (MPC). Similar to this approach, our method adapts a nominal model to local dynamics; however, we do not always exploit the dynamics to reach the goal.

One goal of our method is to generalize the nominal dynamics to OOD novel environments. A popular approach for doing this is explicitly learning to be robust to expected variations across training and test environments. This includes methods such as meta-learning [17], [18], domain randomization [19], [20], Sim-to-real [21], [22], and other transfer learning [23] methods. These methods are unsuitable for this problem because our training data contains only nominal dynamics, whereas they need a diverse set of non-nominal dynamics. Instead, we learn a robust, or “disentangled” representation [24] of the system under which models can generalize. This idea is active in computer vision, where learning based on invariance has become popular [25], [26]. Using similar ideas, we present a novel architecture for learning invariant representations for dynamics models.

IV. METHOD

Our approach is composed of two components: offline representation learning and online control. First, we present how we learn a representation that allows for generalization by exploiting inherent invariances inferred from the nominal data, shown in Fig. 2. Second, we present Trap-Aware Model Predictive Control (TAMPC), a two-level hierarchical MPC method shown in Fig. 3. The high-level controller explicitly reasons about non-nominal dynamics and traps, deciding
These two mechanisms yield the following expressions:

\[ x \]

when to exploit the dynamics and when to recover to familiar ones by outputting the model and cost function the low-level controller uses to compute control signals.

A. Offline: Invariant Representation for Dynamics

In this section, our objective is to exploit potential underlying invariances in state-control space when predicting system dynamics to achieve better generalization to unseen data. More formally, our representation of dynamics is composed of an invariant transformation and a predictive module, shown in Fig. 2. The invariant transformation \( h_x \) maps the state-action space into a latent space \( (z \in \mathbb{R}^{N_z}) \) that the predictive module \( g \) operates on to produce a latent output \( (v \in \mathbb{R}^{N_v}) \) that is then mapped back into the state space using \( h_r \). We parameterize the transformations with neural networks and build in two mechanisms to promote a meaningful and descriptive latent space:

First, we impose \( N_z < N_x + N_u \) to create an information bottleneck. This encourages the latent input to throw out information not relevant for predicting dynamics and to discover invariances—variations in the original state-action space that can be safely ignored. Further, we limit the capacity of \( g \) to be significantly smaller than that of \( h_x \) such that the dynamics take on a simple form.

Second, we partially decouple \( z \) and \( v \) by learning \( v \) in an autoencoder fashion: we require that \( v \) can reconstruct \( \Delta x \) with partial information from \( x \). Passing in \( x \) at reconstruction allows the dynamics pathway to ignore some information in \( x \). Much like the autoencoder architecture, we require the encoded \( v \) to match the \( v \) output from dynamics. To further improve generalization, we restrict information passed from \( x \) to \( v \) when reconstructing \( \Delta x \) with a dimension reducing transform \( h_\omega : \mathcal{X} \rightarrow \mathbb{R}^{N_x} \). This mechanism has the effect of reducing compounding errors when \( x, u \) is OOD. These two mechanisms yield the following expressions:

\[ \tilde{v} = g(z) \quad \Delta x \approx h_r(v, h_\omega(x)) \quad v = h_\eta(\Delta x, h_\omega(x)) \approx \tilde{v} \]

and their associated batch-based reconstruction loss and matching loss:

\[
L_r = \frac{\mathbb{E}[||\Delta x - h_r(v, h_\omega(x))||_2]}{\mathbb{E}[||\Delta x||_2]} \quad L_m = \frac{\mathbb{E}[||v - \tilde{v}||_2]}{\mathbb{E}[||v||_2]}
\]

\[
L_\beta(\tau_i) = \lambda_r L_r(\tau_i) + \lambda_m L_m(\tau_i)
\]

These losses are ratios relative to the norm of the quantity we are trying to match to avoid decreasing loss by scaling the representation. In addition to these two losses, we apply Variance Risk Extrapolation (V-REx [26]) to explicitly penalize the variance in loss across the \( M \) trajectories:

\[
\mathcal{L}(\tau) = \beta \text{var} \{L_\beta(\tau_1), ..., L_\beta(\tau_M)\} + \sum_{i=1}^{M} L_\beta(\tau_i) \quad (4)
\]

We train on Eq. (4) using gradient descent with an annealing strategy for \( \beta \) suggested by [26]. For minibatches, we adjust \( L_r \) to be over only the trajectories that are in the batch.

After learning the transforms, we replace \( g \) with a higher capacity model and fine-tune it on the nominal data with just \( L_m \). Learning the invariant representation this way avoids compounding errors from OOD inputs while allowing our model to be robust to variations unnecessary for dynamics. For further details, please see App. C.

B. Online: Trap-Aware MPC

Online, we require a controller that has two important properties. First, it should incorporate strategies to escape from and avoid detected traps. Second, it should iteratively improve its dynamics representation, in particular when encountering previously unseen modes. To address these challenges, our approach uses a two-level hierarchical controller where the high-level controller described in Alg. 1 explicitly reasons about non-nominal dynamics and traps, outputting the dynamics model and cost function that the low level controller uses to compute control. This structure allows a variety of low-level MPC designs that can be specialized to the task or dynamics if domain knowledge is available.

TAMPC operates in either an exploit or recovery mode, judiciously chosen based on the type of dynamics encountered. When in nominal dynamics, the algorithm exploits its confidence in predictions. When encountering non-nominal dynamics, it attempts to exploit a local approximation built online until it detects entering a trap, at which time recovery
When exploiting dynamics to navigate towards the goal, we evaluate the nominal model prediction error against observed states (“nominal model accuracy” block in Fig. 3 and lines 5 and 13 from Alg. 1):  
\[
\frac{|\langle \Delta x - f(x, u) \rangle / E \rangle| > \epsilon_T \quad (5)
\]
where \( \epsilon_T \) is a designed tolerance threshold and \( E \) is the expected movement (from one-step prediction) since the end of last recovery.

The first step in striking this balance is identifying non-nominal dynamics. Here, we evaluate the nominal model prediction error against observed states (“nominal model accuracy” block in Fig. 3 and lines 5 and 13 from Alg. 1):  
\[
\frac{|\langle \Delta x - f(x, u) \rangle / E \rangle| > \epsilon_T \quad (5)
\]
where \( \epsilon_T \) is a designed tolerance threshold and \( E \) is the expected movement (from one-step prediction) since the end of last recovery.

We add to it the transition with the lowest ratio of actual to expected movement (from one-step prediction) since the end of last recovery:  
\[
\tau = \min_a \frac{d(x, x_{a+1})}{d(x, x_{a+1})} \quad \mathcal{T}_c \leftarrow \mathcal{T}_c \cup \{(x, u)\} \quad (9)
\]

To mitigate over-generalizing local non-nominal dynamics is triggered. This approach attempts to balance between a potentially overly-conservative treatment of all non-nominal dynamics as traps and an overly-optimistic approach of exploiting all non-nominal dynamics assuming goal progress is always possible.

The first step in striking this balance is identifying non-nominal dynamics. Here, we evaluate the nominal model prediction error against observed states (“nominal model accuracy” block in Fig. 3 and lines 5 and 13 from Alg. 1):  
\[
\frac{|\langle \Delta x - f(x, u) \rangle / E \rangle| > \epsilon_T \quad (5)
\]
where \( \epsilon_T \) is a designed tolerance threshold and \( E \) is the expected movement (from one-step prediction) since the end of last recovery.

We add to it the transition with the lowest ratio of actual to expected movement (from one-step prediction) since the end of last recovery:  
\[
\tau = \min_a \frac{d(x, x_{a+1})}{d(x, x_{a+1})} \quad \mathcal{T}_c \leftarrow \mathcal{T}_c \cup \{(x, u)\} \quad (9)
\]

To handle traps close to the goal, we only penalize revisiting trap states if similar actions are to be taken. With the control similarity function \( s : \mathcal{U} \times \mathcal{U} \rightarrow [0, 1] \) we formulate the cost  
\[
C_T(x, u) = \sum_{x', u' \in \mathcal{T}_e} \frac{s(u, u')}{d(x, x')^2} \quad (10)
\]

We switch from exploit to recovery mode when detecting a trap, but it is not obvious what the recovery policy should be. Driven by the online setting and our objective of data-efficiency: First, we restrict the recovery policy to be one induced by running the low-level MPC on some cost function other than one used in exploit mode. Second, we propose hypothesis cost functions and consider only convex combinations of them. Without domain knowledge, one hypothesis is to return to one of the last visited nominal states. However, the dynamics may not always allow this. Another hypothesis is to return to a state that allowed for the most one-step movement. Both of these are implemented in terms of the following cost, where \( S \) is a state set and we pass in either \( X_0 \), the set of last visited nominal states, or \( X_f \), the set of

---

**Algorithm 1:** TAMPC high level control loop

```
Given : C(x, u) cost, x_0, MPC, parameters from Tab. II
1 s ← NOM, t ← 0, d_0 ← 0, x ← x_0, \mathcal{T}_c \leftarrow \emptyset, \alpha ← 1, w ← 0
2 MAB arms ← N_a random convex combs.
3 while C(x, 0) > acceptable threshold do
   if s is NOM then
      if not nominal from Eq. 5 then
         s ← NONNOM
         initialize GP \( \hat{\phi} \) with \((x_{t-1}, u_{t-1}, \Delta x_{t-1})\)
      else
         \( \alpha ← \alpha \cdot \alpha_o \) // anneal trap cost
         \( d_0 ← \max(d_0, d(x_{t-N_a}, x)/N_a) \)
      \end{if}
   \end{if}
   \end{while}
   \end{algorithm}
```

---
A. Experiment Environments

Our two tasks are quasi-static planar pushing and peg-in-hole. Both tasks are evaluated in simulation using PyBullet [28] and the latter is additionally evaluated empirically using a KUKA LBR iiwa arm depicted in Fig. 1. In planar pushing, the goal is to push a block to a known desired position. In peg-in-hole, the goal is to place the peg into a hole with approximately known location. In both environments, the robot has access to its own pose and senses the reaction force at the end-effector. Thus the robot cannot perceive the obstacle geometry visually, it only perceives contact through reaction force. During offline learning of nominal dynamics, there are no obstacles or traps. During online task completion, obstacles are introduced in the environment, inducing unforeseen traps. See Fig. 4 for a depiction of the environments and Fig. 6 for typical traps from tasks in these environments, and App. B for environment details.

In planar pushing, the robot controls a cylindrical pusher restricted to push a square block from a fixed side. Traps introduced by walls are shown in Fig. 6. Frictional contact with the wall limits sliding along the wall and causes most actions to rotate the block into the wall. The state is $x = (x, y, \theta, r_x, r_y)$ where $(x, y, \theta)$ is the block pose, and $(r_x, r_y)$ is the reaction force the pusher feels, both in world frame. Control is $u = (p, \delta, \alpha)$, where $p$ is where along the side to push, $\delta$ is the push distance, and $\alpha$ is the push direction relative to side normal. The state distance is the 2-norm of the pose, with yaw normalized by the block’s radius of gyration. The control similarity is $s(u_1, u_2) = \max(0, \cos(\theta_2))$ where $\cos(\theta_2)$ is cosine similarity.

In peg-in-hole, we control a gripper holding a peg (square in simulation and circular on the real robot) that is constrained to slide along a surface. Traps in this environment geometrically block the shortest path to the hole. The state is $x = (x, y, \delta, \alpha)$ and control is $u = (\delta, \alpha)$, the distance to move in $x$ and $y$. We execute these on the real robot using a Cartesian impedance controller. The state distance is the 2-norm of the position and the control similarity is

$$s(u_1, u_2) = \max(0, \cos(\theta_1)).$$

The goal-directed cost for both environments is in the form $C(x, u) = x^T Q x + u^T R u$. The MPC assigns a terminal multiplier of 50 at the end of the horizon on the state cost. See Tab. III for the cost parameters of each environment.

The nominal data $\tau$ for simulated environments consists of $M = 200$ trajectories each with $T = 50$ transitions (all motion is collision-free as there are no obstacles). For the real robot, we use $M = 20$, $T = 30$. We generate them by uniformly sampling starts from $[-1, 1] \times [-1, 1]$ (for planar pushing is also uniformly sampled) and applying actions uniformly sampled from $U$.

B. Offline: Learning Invariant Representations

In this section we evaluate if our representation can learn useful invariances from the offline training data. We expect nominal dynamics (in freespace) in both environments to be invariant to translation. Since the training data has positions around the range $[-1, 1]$, we evaluate this as the learning losses on the validation set translated by $(10, 10)$. As Fig. 5 shows, we achieve good performance on the translated validation set even with ablations learned without REx. This could be due to our low dimensional state space whereas REx was developed for high dimensional image representations. All representations (for both planar pushing and peg-in-hole) use $N_c = 5, N_n = 5, n_\omega = 2$, and implement the transforms with shallow fully connected networks. For learning details see App. C.

Fig. 5 shows performance on a test set with OOD reaction forces, which we do not expect invariance over. Since the nominal data has no obstacles, we expect dynamics to do poorly (high match loss), but we see that with the $h_\omega$ transform we still learn to reconstruct the output.

C. Online: Tasks in Novel Environments

We evaluate TAMPC against baselines and ablations on the tasks shown in Fig. 1 and Fig. 6. For TAMPC’s low-level MPC, we use a modified model predictive path integral controller (MPPI) [29] where we take the expected cost
no obstacles. Visual sensing and is pre-trained only on environments with planar pushing and peg-in-hole tasks. Our method has no

Fig. 6: (top) Initial condition and (bottom) typical traps for planar pushing and peg-in-hole tasks. Our method has no visual sensing and is pre-trained only on environments with no obstacles.

dynamics, we increase exploration by increasing $N_d$ with the trade-off of staying longer in traps. Independently, we control the expected trap difficulty $N$ with the MPC horizon $H$. Intuitively, we increase $H$ to match higher $N$ (more difficult), as in Peg-I, at the cost of more computation. Lastly, $\alpha_a \in (0, 1]$ controls how quickly we decrease the weight of the trap cost. Too low a value prevents escape from difficult traps while values close to 1 leads to waiting longer in cost function local minima. For Peg-U, the “TAMPC tuned” runs used $N_d = 15, H = 15$, while for Peg-I, the “TAMPC tuned” runs used $H = 20, \alpha_a = 0.95$.

On the real robot, the Cartesian impedance controller did not account for joint limits; instead, these limits were handled naturally as traps by TAMPC. The only scenario in which the baselines achieved some level of success was the real Peg-U, where these approaches generated controls signals that always ended up sliding the peg along the same side of the U. This may be due to possibly imperfect construction leading one corner of the obstacle to be be closer to the goal, or that the joint configuration favoured moving in that direction. This movement bias may have contributed to the adaptive baseline’s success. In contrast, the non-adaptive baseline did not exhibit any movement biases and stayed at the bottom of the U, while TAMPC trajectories explored both sides.

A common trend we identify from Fig. 7 is that the baselines tend to plateau after initially decreasing distance to goal. Through inspection, we found that this plateau did indeed correspond to being caught in traps. We highlight that the adaptive baseline also gets caught in traps. This may be due to a combination of insufficient knowledge of dynamics around traps, over-generalization of trap dynamics, and using too short of a MPC horizon. SAC likely stays inside of traps because no action immediately decreases cost, and action sequences that eventually decrease cost have high short-term cost and are thus unlikely to be sampled in 500 steps. TAMPC escapes traps because it uses the signal from state distance to explicitly reason about traps and switch to a recovery policy. This added structure is useful in tasks with traps because it decreases the degree of learning and planning the algorithm has to do.

In more detail, the ablation variants demonstrate the value of TAMPC components. Our structured recovery policy is highlighted on the block tasks. TAMPC random (recovery policy is uniform random actions until dynamics is nominal) performs significantly worse than our full method here because escaping traps in planar pushing requires long sequences exploiting the local non-nominal dynamics to rotate the block against the wall. This is unlike the peg environment where the gripper can directly move away from the wall and the random recovery policy performs well. The Peg-T(T) task highlights our learned dynamics representation. It is a copy of Peg-T translated 10 units in $x, y$. Using the invariant representation, we maintain similar performance to Peg-T whereas performance degrades under a nominal model in the original space. This is because annealing the trap set cost requires being in recognized nominal dynamics, without which it is easy to get stuck in local minima.

Fig. 7 shows that TAMPC or its variants outperforms baselines on all tasks in median distance to goal after 500 control steps. Our method is empirically robust to parameter value choices, as we use the same values for different tasks, with many also shared across environments; see Tab. II. We further improve performance on Peg-U and Peg-I by tuning only three independent parameters. We control the exploration of non-nominal dynamics with $N_d$. For cases like Peg-U where the goal is surrounded by non-nominal invariance representation, we maintain similar performance to Peg-T whereas performance degrades under a nominal model in the original space. This is because annealing the trap set cost requires being in recognized nominal dynamics, without which it is easy to get stuck in local minima.

VI. DISCUSSION

Fig. 5: Learning curves on the (left) validation, (mid) translated validation, and (right) test set for planar pushing representation. Mean across 10 runs is solid while ±1 standard deviation is shaded.

Baselines: We compare against three baselines. The first is online model adaptation from [4], which does iLQR control on a linearized global model mixed with a locally-fitted linear model. This represents the overly-optimistic approach of always exploiting non-nominal dynamics to head towards the goal. iLQR (code provided by [4]’s author) performs poorly in freespace of the planar pushing environment, possibly due to the non-linearity and noise from reaction force dimensions. We instead do control with MPPI using a locally-fitted GP model (effectively an ablation of TAMPC with control mode fixed to NONNOM). This is termed “adaptive baseline++”. Our second baseline is model-free reinforcement learning with Soft Actor-Critic (SAC) [5]. Here, a nominal policy is learned offline for 1000000 steps on the nominal environment, which is used to initialize the policy at test time. Online, the policy is retrained after every control on the dense environment reward. Lastly, our “non-adaptive” baseline runs MPPI on the nominal model.
A. Limitations and Future Work

The main failure mode of TAMPC is oscillation between previously visited traps. Future work could break the symmetry between traps by weighting them on time of visit. A limitation of our method is that we need to be given a state space with a distance function. Future work could attempt to learn this from nominal data, for example by assuming linearity of state distance to control effort. Our experiments focus on rigid-body interactions that can be succinctly represented with low-dimensional state-spaces. Future work could apply TAMPC to non-rigid manipulation problems, where the higher dimensional state space could be computationally challenging for the local GP model.

Finally, future work could improve the detection of non-nominal dynamics. For certain classes of problems, there could be more justified measures than the 2-norm on model prediction error.

APPENDIX A: PARAMETERS AND ALGORITHM DETAILS

Making U-turns in planar pushing requires \( H \geq 25 \). We shorten the horizon to 5 and remove the terminal state cost of 50 when in recovery mode to encourage immediate progress. \( \rho \in (0, \infty) \) depends on how accurately we need to model trap dynamics to escape them. Increasing \( \rho \) leads to selecting only the best sampled action while a lower value leads to more exploration by taking sub-optimal actions.

Nominal error tolerance \( \epsilon_N \) depends on the variance of the model prediction error in nominal dynamics. A higher variance requires a higher \( \epsilon_N \). We use a higher value for peg-in-hole because of simulation quirks in measuring reaction force from the two fingers gripping the peg. Block pushing has greater stability in measuring reaction force since there is only one point of contact between the pusher and block.

APPENDIX B: ENVIRONMENT DETAILS

The planar pusher is a cylinder with radius 0.02m to push a square with side length \( a = 0.3m \). We have \( \theta \in [-\pi, \pi], p \in [-a/2, a/2], \delta \in [0, a/8], \) and \( \alpha \in [-\pi/4, \pi/4] \). All distances are in meters and all angles are in radians. The state distance function is the 2-norm of the pose, where yaw is normalized by the block’s radius of gyration. \( d(x_1, x_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + \sqrt{a^2/6(\theta_1 - \theta_2)^2}} \). The sim peg-in-hole peg is square with side length 0.03m, and control

Algorithm 2: EnteringTrap

Given: \( x_0, ..., x_t \), since end of last recovery or start of local dynamics, whichever is more recent, \( d_0 \), max nominal per-step distance, \( N_d \), \( \epsilon_T \)

1. for \( a \leftarrow 0 \) to \( t - N_d \) do
2.   if \( d(x_t, x_a)/(t - a) < \epsilon_T d_0 \) then
3.     return True
4. return False

Fig. 7: Minimum distance along the trajectory to the goal accounting for walls (computed with Djikstra’s algorithm). Median across 10 runs is in solid while the 20th–80th percentile is shaded. Task success on block tasks is achieving distance < 0.5 and achieving 0 distance for peg-in-hole.

| Parameter | block | peg | real peg |
|-----------|-------|-----|---------|
| \( R \) samples | 500 | 500 | 500 |
| \( H \) horizon | 40 | 10 | 15 |
| \( R \) rollouts | 10 | 10 | 10 |
| \( \lambda \) | 0.01 | 0.01 | 0.01 |
| \( u' \) | diag [0.2, 0.4, 0.7] | diag [0.2, 0.2] | diag [0.2, 0.2] |
| \( \Sigma \) | diag [0.2, 0.4, 0.7] | diag [0.2, 0.2] | diag [0.2, 0.2] |

TABLE II: TAMPC parameters for different environments.

| Parameter | block | peg | real peg |
|-----------|-------|-----|---------|
| \( \alpha_n \) trap cost annealing rate | 0.97 | 0.9 | 0.8 |
| \( \rho \) recovery cost weight | 2000 | 1 | 1 |
| \( \epsilon_N \) nominal error tolerance | 10 | 15 | 15 |
| \( \epsilon_T \) trap tolerance | 0.6 | 0.6 | 0.6 |
| \( N_R \) min dynamics window | 5 | 5 | 2 |
| \( N_w \) nominal window | 3 | 3 | 3 |
| \( N_s \) steps for bandit arm pulls | 3 | 3 | 3 |
| \( N_p \) number of bandit arms | 100 | 100 | 100 |
| \( N_R \) max steps for recovery | 20 | 20 | 20 |
| \( N_s \) local model window | 50 | 50 | 50 |
| \( \epsilon_c \) converged threshold | 0.05T | 0.05T | 0.05T |
| \( \epsilon_m \) move threshold | 1 | 1 | 1 |

TABLE III: Goal cost parameters for each environment.

| Term | block | peg | real peg |
|------|-------|-----|---------|
| \( Q \) | diag [10, 10, 0, 0] | diag [1, 1, 0, 0, 0] | diag [1, 1, 0, 0, 0] |
| \( R \) | diag [0.1, 0.1, 0.1] | diag [1, 1] | diag [1, 1] |

TABLE I: MPPI parameters for different environments.
Algorithm 3: Recovered

Given: \( x_0, \ldots, x_t \) since start recovery, \( d_0 \), parameters from Tab. I

1. if \( t < N_d \) then
2. \( \text{return False} \)
3. else if \( t > N_R \) then
4. \( \text{return True} \)
5. converged \( \leftarrow d(x_t, x_{t-N_d})/N_d < \epsilon_c d_0 \)
6. away \( \leftarrow d(x_t, x_0) > \epsilon_a d_0 \)
7. \( \text{return converged and away} \)

Algorithm 4: MPPI with multiple rollouts. Differences from standard MPPI are highlighted.

Given: \( C(x, u), f_n(x, u), x, U, \), Tab. I parameters
1. \( \epsilon \leftarrow \mathcal{N}(\mu, \Sigma) \) // \( u \) perturbation for \( H \) steps
2. \( U, \epsilon \leftarrow \text{clip} \ U \) to control bounds
3. \( x_0 \leftarrow x \)
4. \( c \leftarrow 0 \) // \( R \times K \)
5. for \( r \leftarrow 0 \) to \( R-1 \) do
6. \( \text{for} \ t \leftarrow 0 \) to \( H-1 \) do
7. \( X_{t+1} \leftarrow f_n(x_t, u_t) \) // sample rollout
8. \( c_{r,t} \leftarrow C(x_{t+1}, U_t) \)
9. \( c_r \leftarrow c_r + c_{r,t} \)
10. \( \epsilon \leftarrow \text{mean} \ c \ across \ R \)
11. \( U \leftarrow \text{softmax mix} \ c \ across \ R \)
12. \( \text{return} \ U \)

is limited to \( \delta_x, \delta_u \in [0, 0.03] \). These values are internally normalized so MPPI outputs control in the range \([-1, 1]\).

APPENDIX C: REPRESENTATION LEARNING & GP

Each of the transforms is represented by 2 hidden layer multilayer perceptrons (MLP) activated by LeakyReLU and implemented in PyTorch. They each have (16, 32) hidden units except for the simple dynamic \( g \) which has (16, 16) hidden units. We optimize for 3000 epochs using Adam with default settings (learning rate 0.001), \( \lambda_r = \lambda_u = 1 \), and \( \beta = 1 \) for the first 1000 epochs, then \( \beta = 1000 \) afterwards. For training with V-REx, we use a batch size of 2048, and a batch size of 500 otherwise. We use the GP implementation of gpytorch with an RBF kernel, zero mean, and independent output dimensions. On every transition, we retrain for 15 iterations on the last 50 transitions since entering non-nominal dynamics to only fit non-nominal data.

REFERENCES

[1] J. Borenstein, G. Granosik, and M. Hansen, “The omnitread serpentine robot: design and field performance,” in *Unmanned Ground Vehicle Technology VII*, vol. 5804. SPIE, 2005, pp. 324–332.
[2] I. Fantoni, R. Lozano, and R. Lozano, *Non-linear control for underactuated mechanical systems*. Springer Science, 2002.
[3] D. E. Koditschek, R. J. Full, and M. Buehler, “Mechanical aspects of legged locomotion control,” *Arthropod structure & development*, vol. 33, no. 3, pp. 251–272, 2004.
[4] J. Fu, S. Levine, and P. Abbeel, “One-shot learning of manipulation skills with online dynamics adaptation and neural network priors,” in *IROS*. IEEE, 2016, pp. 4019–4026.
[5] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” in *ICML*, 2018, pp. 1861–1870.
[6] D. Panagou and K. J. Kyriakopoulos, “Viability control for a class of underactuated systems,” *Automatica*, vol. 49, no. 1, pp. 17–29, 2013.
[7] I. Schrammhuber, “A possibility for implementing curiosity and boredom in model-building neural controllers,” in *Proc. of the international conference on simulation adaptive behavior: From animals to humans*, 1991, pp. 222–227.
[8] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, “Curiosity-driven exploration by self-supervised prediction,” in *CVPR Workshops*, 2017, pp. 16–17.
[9] Y. Burda, H. Edwards, A. Storkey, and O. Klimov, “Exploration by random network distillation,” *arXiv preprint arXiv:1810.12894*, 2018.
[10] H. Tang, R. Houthooft, D. Foote, A. Stooke, O. X. Chen, Y. Duan, J. Schulman, F. DeTurck, and P. Abbeel, “# exploration: A study of count-based exploration for deep reinforcement learning,” in *NeurIPS*, 2017, pp. 2753–2762.
[11] M. Belllemare, S. Srivivasan, G. Ostrovski, T. Schaaf, D. Saxton, and R. Munos, “Unifying count-based exploration and intrinsic motivation,” in *NeurIPS*, 2016, pp. 1471–1479.
[12] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy, “Deep exploration via bootstrapped dqn,” in *NeurIPS*, 2016, pp. 4026–4034.
[13] I. E. Coelho, J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune, “Go-explore: a new approach for hard-exploration problems,” *arXiv preprint arXiv:1901.10995*, 2019.
[14] T. Dierks and S. Jagannathan, “Online optimal control of affine nonlinear discrete-time systems with unknown internal dynamics by using time-based policy update,” *IEEE Trans Neural Netw Learn Syst.*, vol. 23, no. 7, pp. 1118–1129, 2012.
[15] S. Bhasin, R. Kanalamurkar, M. Johnson, K. G. Vamvoudakis, F. L. Lewis, and W. E. Dixon, “A novel actor–critic–identifier architecture for approximate optimal control of uncertain nonlinear systems,” *Automatica*, vol. 49, no. 1, pp. 82–92, 2013.
[16] S. Levine and P. Abbeel, “Learning neural network policies with guided policy search under unknown dynamics,” in *NeurIPS*, 2014, pp. 1071–1079.
[17] C. Finn, P. Abbeel, and S. Levine, “Model-agnostic meta-learning for fast adaptation of deep networks,” in *ICML*. JMLR. org, 2017, pp. 1126–1135.
[18] D. Li, Y. Yang, Y.-Z. Song, and T. M. Hospedales, “Learning to generalize: Meta-learning for domain generalization,” in *AAAI*, 2018.
[19] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, “Domain randomization for transferring deep neural networks from simulation to the real world,” in *ICROS*. IEEE, 2017, pp. 23–30.
[20] S. J. Pan, I. W. Tsang, J. T. Kwok, and Q. Yang, “Domain adaptation via transfer component analysis,” *IEEE Transactions on Neural Networks*, vol. 22, no. 2, pp. 199–210, 2010.
[21] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, “Sim-to-real transfer of robotic control with dynamics randomization,” in *ICRA*. IEEE, 2018, pp. 1–8.
[22] S. James, P. Wohlhart, M. Kalakrishnan, D. Kalashnikov, A. Irpan, J. Ibarz, S. Levine, R. Hadsell, and K. Bouman, “Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks,” in *CVPR*, 2019, pp. 12627–12637.
[23] A. Zhang, H. Saito, and J. Pineau, “Decoupling dynamics and reward for transfer learning,” *arXiv preprint arXiv:1804.10089*, 2018.
[24] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel, “Infogan: Interpretable representation learning by information maximizing generative adversarial nets,” in *NeurIPS*, 2016, pp. 2172–2180.
[25] M. Arjovsky, L. Bottou, I. Gulrajani, and D. Lopez-Paz, “Invariant risk minimization,” *arXiv preprint arXiv:1907.02893*, 2019.
[26] D. Krueger, E. Caballero, J.-H. Jacobsen, A. Zhang, J. Binns, R. L. Priol, and A. Courville, “Out-of-distribution generalization via risk extrapolation (rex),” *arXiv preprint arXiv:2003.00658*, 2020.
[27] D. McConachie and D. Berenson, “Bandit-based model selection for deformable object manipulation,” in *Algorithmic Foundations of Robotics XII*. Springer, 2020, pp. 704–719.
[28] E. Coumans and Y. Bai, “Pybullet, a python module for physics simulation for games, robotics and machine learning,” 2016.
[29] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou, “Information theoretic mpc for model-based reinforcement learning,” in *ICRA*. IEEE, 2017, pp. 1714–1721.