Multi-Modal MPPI and Active Inference for Reactive Task and Motion Planning

Zhang, Yuezhe; Pezzato, Corrado; Trevisan, Elia; Salmi, Chadi; Corbato, Carlos Hernandez; Alonso-Mora, Javier

DOI
10.1109/LRA.2024.3426183

Publication date
2024

Document Version
Final published version

Published in
IEEE Robotics and Automation Letters

Multi-Modal MPPI and Active Inference for Reactive Task and Motion Planning

Yuezhe Zhang, Corrado Pezzato, Elia Trevisan, *Graduate Student Member, IEEE*, Chadi Salmi, Carlos Hernández Corbato, and Javier Alonso-Mora, *Senior Member, IEEE*

***Abstract*—Task and Motion Planning (TAMP) has made strides in complex manipulation tasks, yet the execution robustness of the planned solutions remains overlooked. In this work, we propose a method for reactive TAMP to cope with runtime uncertainties and disturbances. We combine an Active Inference planner (AIP) for adaptive high-level action selection and a novel Multi-Modal Model Predictive Path Integral controller (M3P2I) for low-level control. This results in a scheme that simultaneously adapts both high-level actions and low-level motions. The AIP generates alternative symbolic plans, each linked to a cost function for M3P2I. The latter employs a physics simulator for diverse trajectory rollouts, deriving optimal control by weighing the different samples according to their cost. This idea enables blending different robot skills for fluid and reactive plan execution, accommodating plan adjustments at both the high and low levels to cope, for instance, with dynamic obstacles or disturbances that invalidate the current plan. We have tested our approach in simulations and real-world scenarios.**

***Index Terms*—Manipulation planning, task and motion planning.**

I. Introduction

TASK and Motion Planning (TAMP) is a powerful class of methods for solving complex long-term manipulation problems where logic and geometric variables influence each other. TAMP [1], [2], [3] has been successfully applied to domains such as table rearrangement, stacking blocks, or solving the Hanoi tower. However, the plan is often executed in open-loop in static environments. Recent works [4], [5], [6] recognized the importance of robustifying the execution of TAMP plans to be able to carry them out in the real world reliably. But they either rely only on the adaptation of the action sequence in a plan [6], [7], [8], [9] or only on the motion planning problem in a dynamic environment given a fixed plan [4], [5]. Unlike typical TAMP planners that focus on solving static and complex tasks offline and then execute the solution, this letter aims to achieve reactive execution by simultaneously adapting high-level actions and low-level motions.

Reactive TAMP faces the challenge of accommodating unforeseen geometric constraints during planning, such as the need to pull rather than push a block when it's in a corner, complicating high-level planning without complete scene knowledge. Additionally, scenarios like pick-and-place tasks with dynamic obstacles and human disturbances demand varied grasping poses for different objects and obstacles, requiring TAMP algorithms to adapt to such configurations dynamically.

We address these challenges by proposing a control scheme that jointly achieves reactive action selection and robust low-level motion planning during execution. We propose a high-level planner capable of providing alternative actions to achieve a goal. These actions are translated to different cost functions for our new Multi-Modal Model Predictive Path Integral controller for motion planning. This motion planner leverages a physics simulator to sample parallel motion plans that minimize the given costs and computes one coherent control input that effectively blends different strategies. To achieve this, we build upon two of our recent works: 1) an Active Inference planner (AIP) [7] for symbolic action selection, and 2) a Model Predictive Path Integral (MPPI) controller [10] for motion planning. The AIP computes a sequence of actions and state transitions through backchaining to achieve a sub-goal specified in a given Behavior Tree (BT). The BT guides the search and allows real-time high-level planning within the AIP framework [7]. In this work, we extend the previous AIP to plan possible alternative action plans, and we propose a new Multi-Modal Model Predictive Path Integral controller (M3P2I) that can sample in parallel these alternatives and smoothly blend them considering the geometric constraints of the problem.

A. Related Work

To robustly operate in dynamic environments, reactive motion planners are necessary. In [4], the authors provided a reactive Model Predictive Control (MPC) strategy to execute a TAMP plan as a given linear sequence of constraints. The reactive nature of the approach allows coping with disturbances and dynamic collision avoidance during the execution of a TAMP plan. Authors in [5] formulated a TAMP plan in object-centric Cartesian coordinates, showing how this allows coping with perturbations such as moving a target location. However, both [4], [5] do not consider adaptation at the symbolic action level if a perturbation invalidates the current plan.

Manuscript received 22 December 2023; accepted 17 June 2024. Date of publication 10 July 2024; date of current version 18 July 2024. This article was recommended for publication by Associate Editor Rahul Shome and Editor Hanna Kurniawati upon evaluation of the reviewers' comments. This work was supported in part by Ahold Delhaize, in part by the Netherlands Organization for Scientific Research (NWO), domain Science (ENW), TRiLOGy project, and in part by European Union through ERC, under Grant 101041863 (INTERACT). *(Corresponding author: Yuezhe Zhang.)*

The authors are with the Cognitive Robotics Department, TU Delft, 2628 CD Delft, The Netherlands (e-mail: yuezhezhang_bit@163.com; corrado.pezzato@gmail.com; e.trevisan@tudelft.nl; salmi.chadi@gmail.com; c.h.corbato@tudelft.nl; j.alonsomora@tudelft.nl).

Project website: https://autonomousrobots.nl/paper_websites/m3p2i-aip.

This letter has supplementary downloadable material available at https://doi.org/10.1109/LRA.2024.3426183, provided by the authors.

Digital Object Identifier 10.1109/LRA.2024.3426183

Several letters focused on adapting and repairing high-level action sequences during execution. In [11], robot task plans are represented as robust logical-dynamical systems to handle human disturbances. Similarly, [12] coordinates control chains for robust plan execution through plan switching and controller selection. A recent letter [13] suggests employing Monte Carlo Tree Search with IsaacGym to accelerate task planning for multi-step object retrieval from clutter involving intricate physical interactions. While promising, [13] only supports high-level reasoning with predefined motions in an open loop. Recent works [6], [14] combined BTs and linear temporal logic to adapt the high-level plan to cope with cooperative or adversarial human operators, environmental changes, or failures. In our previous work [7], AIP and BT were combined to provide reactive action selection in long-term tasks in partially observable and dynamic environments. This method achieved hierarchical deliberation and continual online planning, making it particularly appealing for the problem of reactive TAMP at hand. In this letter, we extend [7] by bridging the gap to low-level reactive control by planning cost functions instead of symbolic actions.

At the lower level, MPC is a widely used approach [15], [16], [17]. However, manipulation tasks often involve discontinuous contacts that are hard to differentiate. Sampling-based MPCs, such as MPPI [18], [19], can handle non-linearities, non-convexities, or discontinuities of the dynamics and costs. MPPI relies on sampling control input sequences and forward system dynamics simulation. The resulting trajectories are weighted according to their cost to approximate an optimal control input. In [20], the authors proposed an ensemble MPPI to cope with model parameters uncertainty. Sampling-based MPCs are generally applied for single-skill execution, such as pushing or reaching a target point. As pointed out in the future work of [21], one could use a high-level agent to set the cost functions for the sampling-based MPC for long-horizon cognitive tasks. We follow this line of thought and propose a method to reactively compose cost functions for long-horizon tasks. Moreover, classical MPPI approaches can only keep track of one cost function at a time. This means the task planner should propose a single plan to solve the task. However, some tasks might present geometric ambiguities for which multiple plans could be effective, and selecting what strategy to pursue can only be determined by the motion planner based on the geometry of the problem.

B. Contributions

The main contribution of this work is a reactive task and motion planning algorithm based on the following:

- A new Multi-Modal MPPI (M3P2I) capable of sampling in parallel plan alternatives to achieve a goal, evaluating them against different costs. This enables the smooth blending of alternative solutions into a coherent behavior instead of switching based on heuristics.
- An enhanced Active Inference planner (AIP) capable of generating alternative cost functions for M3P2I.

We demonstrate the method in several scenarios in simulations and real robots for pushing, pulling, picking, and placing objects under disturbances.

II. Background

In this section, we present the background knowledge about the Active Inference planner and Model Predictive Path Integral Control to understand the contributions of this letter. We refer the interested reader to the original articles [7], [22], [23] for a more in-depth understanding of the techniques.

A. Active Inference Planner (AIP)

AIP is a high-level decision-making algorithm that relies on symbolic states, observations, and actions [7]. Each independent set of states in AIP is a factor, and the planner contains a total of n_f factors. For a generic factor f_j where $j \in \mathcal{J} = \{1, \ldots, n_f\}$, it holds:

$$s^{(f_j)} = \left[s^{(f_j,1)}, s^{(f_j,2)}, \ldots, s^{(f_j,m^{(f_j)})}\right]^\top, \quad \mathcal{S} = \left\{s^{(f_j)} | j \in \mathcal{J}\right\} \tag{1}$$

where $m^{(f_j)}$ is the number of mutually exclusive symbolic values a state factor can have, each entry of $s^{(f_j)}$ is a real value between 0 and 1, and the sum of the entries is 1. This represents the current belief state.

The continuous state of the world $x \in \mathcal{X}$ is discretized through a symbolic observer such that the AIP can use it. Discretized observations o are used to build a probabilistic belief about the symbolic current state. Assuming one set of observations per state factor with $r^{(f_j)}$ possible values:

$$o^{(f_j)} = \left[o^{(f_j,1)}, o^{(f_j,2)}, \ldots, o^{(f_j,r^{(f_j)})}\right]^\top, \quad \mathcal{O} = \left\{o^{(f_j)} | j \in \mathcal{J}\right\} \tag{2}$$

The robot has a set of symbolic actions that can act then their corresponding state factor:

$$a_\tau \in \alpha^{(f_j)} = \left\{a^{(f_j,1)}, a^{(f_j,2)}, \ldots, a^{(f_j,k^{(f_j)})}\right\}, \quad \mathcal{A} = \left\{\alpha^{(f_j)} | j \in \mathcal{J}\right\} \tag{3}$$

where $k^{(f_j)}$ is the number of actions that can affect a specific state factor f_j. Each generic action $a^{(f_j,\cdot)}$ has associated a symbolic name, *parameters*, *pre-* and *postconditions:*

Action $\mathbf{a^{(f_j,\cdot)}}$	**Preconditions**	**Postconditions**
`action_name`(par)	$\texttt{prec}_{a^{(f_j,\cdot)}}$	$\texttt{post}_{a^{(f_j,\cdot)}}$

where $\texttt{prec}_{a^{(f_j,\cdot)}}$ and $\texttt{post}_{a^{(f_j,\cdot)}}$ are *first-order logic predicates* that can be evaluated at run-time. A logical predicate is a boolean-valued function $\mathcal{B} : \mathcal{X} \to \{\texttt{true}, \texttt{false}\}$. Finally, we define the logical state $l^{(f_j)}$ as a one-hot encoding of $s^{(f_j)}$. The AIP computes the posterior distribution over p plans $\boldsymbol{\pi}$ through free-energy minimization [7]. The symbolic action to be executed by a robot in the next time step is the first action of the most likely plan, denoted with $\pi_{\zeta,0}$:

$$\zeta = \max(\underbrace{[\boldsymbol{\pi}_1, \boldsymbol{\pi}_2, \ldots, \boldsymbol{\pi}_p]}_{\boldsymbol{\pi}^\top}), \; a_{\tau=0} = \pi_{\zeta,0}. \tag{4}$$

B. Model Predictive Path Integral Control (MPPI)

MPPI is a method for solving optimal stochastic problems in a sampling-based fashion [22], [23]. Let us consider the following discrete-time systems:

$$x_{t+1} = f(x_t, v_t), \quad v_t \sim \mathcal{N}(u_t, \Sigma), \tag{5}$$

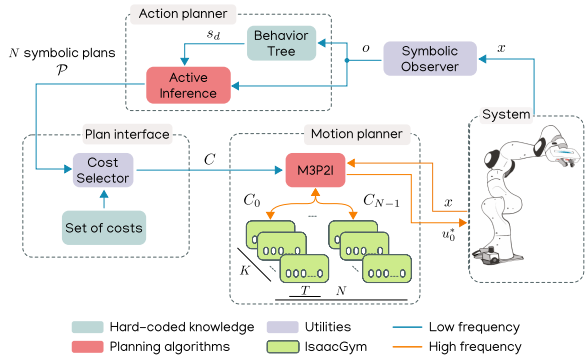

Fig. 1. Proposed scheme. Given symbolic observations o of the environment, the action planner computes N different plan alternatives linked to individual cost functions C_i. M3P2I samples control input sequences and uses an importance sampling scheme to approximate the optimal control u_0^*.

Algorithm 1: Overview of the Method.

Input: action templates and inputs from Algorithms 2 to 4
$AIP.task = AIP.agent(ActionTemplates)$
while task not completed **do**
  $o \leftarrow GetSymbolicObservation(x)$
  / Get current desired state */*
  $AIP.s_d \leftarrow BT(o)$ or be manually set ▷ from [7]
  / Get current action plans from Active Inference */*
  $\mathcal{P} \leftarrow AIP.parall\_act\_sel(o)$ ▷ Algorithm 2
  / Translate action plan to cost function */*
  $C \leftarrow Interface(\mathcal{P})$
  / Compute motion commands */*
  $M3P2I.command(C)$ ▷ Algorithm 4
end while

where f, a nonlinear state-transition function, describes how the state x evolves over time t with a control input v_t. u_t and Σ are the commanded input and the variance, respectively. K noisy input sequences V_k are sampled and then applied to the system to forward simulate K state trajectories $Q_k, k \in [0, K-1]$, over a time horizon T. Given the state trajectories Q_k and a designed cost function C to be minimized, the total state-cost S_k of an input sequence V_k is computed by evaluating $S_k = C(Q_k)$. Finally, each rollout is weighted by the importance sampling weights w_k. These are computed through an inverse exponential of the cost S_k with tuning parameter β and normalized by η. For numerical stability, the minimum sampled cost $\rho = \min_k S_k$ is subtracted, leading to:

$$w_k = \frac{1}{\eta} \exp\left(-\frac{1}{\beta}(S_k - \rho)\right), \quad \sum_{k=1}^{K} w_k = 1 \tag{6}$$

The parameter β is called *inverse temperature*. The importance sampling weights are finally used to approximate the optimal control input sequence U^*:

$$U^* = \sum_{k=1}^{K} w_k V_k \tag{7}$$

The first input u_0^* of the sequence U^* is applied to the system, and the process is repeated. At the next iteration, U^* is used as a warm-start, time-shifted backward of one timestep. Specifically, the second last input in the shifted sequence is also propagated to the last input. In this work, we build upon our previous MPPI approaches [10], [24], where we employed IsaacGym as a dynamic model to forward simulate trajectory rollouts and allow for arbitrary sampling distributions.

III. Methodology

The proposed method is depicted in Fig. 1. After a general overview, we discuss the three main parts of the scheme: *action planner*, *motion planner*, and *plan interface*.

A. Overview

The proposed scheme works as follows. First, the *symbolic observers* translates continuous states x into discretized symbolic observations o, which are then passed to the action planner. The current desired state s_d for Active Inference can be manually set or be encoded as the skeleton solution of a BT as previous work [7]. The AIP computes N alternative symbolic plans based on the current symbolic state and the available symbolic actions. The symbolic actions are encoded as action templates with pre-post conditions that Active Inference uses to construct action sequences to achieve the desired state. After the plans are generated, the plan interface links the first action $a_{0,i},\ i = 0 \ldots N-1$ of each plan to a cost function C_i. The cost functions are sent to M3P2I, which samples $N \cdot K$ different control input sequences. The input sequences are forward simulated using IsaacGym, which encodes the dynamics of the problem [10]. The resulting trajectories are evaluated against their respective costs. Finally, an importance sampling scheme calculates the approximate optimal control u_0^*. All processes are running continuously during execution at different frequencies. The action planner runs, for instance, at 1 Hz while the motion planner runs at 25 Hz. An overview can be found in Algorithm 1.

B. Action Planner - Active Inference Planner (AIP)

In contrast to our previous work [7] where only one action a_τ for the next time step is computed, we modify the AIP to generate action alternatives. In particular, instead of stopping the search for a plan when a valid executable action a_τ is found, we repeat the search while removing that same a_τ from the available action set $\mathcal{A}$. This simple change is effective because we are looking for alternative actions to be applied at the next step, and the AIP builds plans backward from the desired state [7]. The pseudocode is reported in Algorithm 2. The algorithm will cease when no new actions are found, returning a list of possible plans $\mathcal{P}$. This planner is later integrated with M3P2I to evaluate different alternatives in real-time. This increases the robustness at run-time and, at the same time, reduces the number of heuristics to be encoded in the action planner. Specifically, one does not need to encode when to prefer a symbolic action over another based on the geometry of the problem.

C. Motion Planner - Multi-Modal MPPI (M3P2I)

We propose a Multi-Modal MPPI capable of sampling different plan alternatives from the AIP. Traditional MPPI approaches consider *one* cost function and *one* sampling distribution. In this work, we propose keeping track of N separate control input

Algorithm 2: Generate Alternative Plans Using Active Inference.

1: **Input:** available action set: $\mathcal{A}$
2: $a_\tau \leftarrow AIP.act\_sel(o)$ ▷ from [7]
3: Set $\mathcal{P} \leftarrow \emptyset$
4: **while** $a_\tau != \text{none}$ **do**
5: $\mathcal{P}.append(a_\tau)$
6: $\mathcal{A} = \mathcal{A} \setminus \{a_\tau\}$
7: $a_\tau \leftarrow AIP.act\_sel(o)$ ▷ from [7]
8: **end while**
9: **Return** $\mathcal{P}$

sequences corresponding to N different plan alternatives/costs. This is advantageous because it offers a general approach to exploring different strategies in parallel. We perform N separate sets of importance weights, one for each alternative, and only ultimately, we combine the weighted control inputs in one coherent control. This allows the smooth blending of different strategies. Assume we consider N alternative plans, a total of $N \cdot K$ samples. Assume the cost of plan $i, i \in [0, N)$ to be formulated as:

$$S_i(V_k) = \sum_{t=0}^{T-1} \gamma^t C_i(x_{t,k}, v_{t,k}) \tag{8}$$

$\forall k \in \kappa(i)$ where $\kappa(i)$ is the integer set of indexes ranging from $i \cdot K$ to $(i+1) \cdot K - 1$. State $x_{t,k}$ and control input $v_{t,k}$ are indexed based on the time t and trajectory k. The random control sequence $V_k = [v_{0,k}, v_{1,k}, \ldots, v_{T-1,k}]$ defines the control inputs for trajectory k over a time horizon T. The trajectory $Q_i(V_k) = [x_{0,k}, x_{1,k}, \ldots, x_{T-1,k}]$ is determined by the control sequence V_k and the initial state $x_{0,k}$. C_i is the cost function for plan i. Finally, $\gamma \in [0, 1]$ is a discount factor that evaluates the importance of accumulated future costs. As in classical MPPI approaches, given the costs $S_i(V_k)$, we can compute the importance sampling weights associated with each alternative as:

$$\omega_i(V_k) = \frac{1}{\eta_i} \exp\left(-\frac{1}{\beta_i}\left(S_i(V_k) - \rho_i\right)\right), \forall k \in \kappa(i) \tag{9}$$

$$\eta_i = \sum_{k \in \kappa(i)} \exp\left(-\frac{1}{\beta_i}\left(S_i(V_k) - \rho_i\right)\right) \tag{10}$$

$$\rho_i = \min_{k \in \kappa(i)} S_i(V_k) \tag{11}$$

We use the insight in [10] to 1) sample Halton splines instead of Gaussian noise for smoother behavior, 2) automatically tune the inverse temperature β_i to maintain the normalization factor η_i within certain bounds. The latter is helpful since η_i indicates the number of samples to which significant weights are assigned. If η_i is close to the number of samples K, an unweighted average of sampled trajectories will be taken. If η_i is close to 1, then the best trajectory sample will be taken. We observed that setting η_i between 5% and 10% of K generates smooth trajectories. As opposed to [10], we update η *within a rollout* to stay within bounds instead of updating it once per iteration, see Algorithm 3.

We use μ_i to denote the action sequence of plan i over a time horizon $\mu_i = [\mu_{i,0}, \mu_{i,1}, \ldots, \mu_{i,T-1}]$. Each sequence is

Algorithm 3: Update Inverse Temperature β_i.

1: **Input:** parameters: η_l, η_u
2: **while** $\eta_i \notin [\eta_l, \eta_u]$ **do**
3: $\rho_i \leftarrow \min_{k \in \kappa(i)} S_i(V_k)$ ▷ (11)
4: $\eta_i \leftarrow \sum_{k \in \kappa(i)} \exp(-\frac{S_i(V_k) - \rho_i}{\beta_i})$ ▷ (10)
5: **if** $\eta_i > \eta_u$ **then** ▷ greater than upper bound
6: $\beta_i = 0.9 * \beta_i$
7: **else if** $\eta_i < \eta_l$ **then** ▷ smaller than lower bound
8: $\beta_i = 1.2 * \beta_i$
9: **end if**
10: **end while**
11: **Return** ρ_i, η_i, β_i

weighted by the corresponding weights leading to:

$$\mu_i = \sum_{k \in \kappa(i)} \omega_i(V_k) V_k \tag{12}$$

At every iteration, we add to μ_i the sampled noise from *Halton splines* [19]. Then, we forward simulate the state trajectories $Q_i(V_k)$ using IsaacGym as in [10]. Finally, given the state trajectories corresponding to the plan alternatives, we need to compute the weights and mean for the overall control sequence. To do so, we concatenate the N state-costs $S_i(V_k), i \in [0, N)$ and represent it as $\tilde{S}(V)$. Therefore, we calculate the weights for the whole control sequence as [19]:

$$\tilde{\omega}(V) = \frac{1}{\eta} \exp\left(-\frac{1}{\beta}\left(\tilde{S}(V) - \rho\right)\right) \tag{13}$$

Similarly, η, ρ are computed as in (10) and (11) but considering $\tilde{S}(V)$ instead. The overall mean action over time horizon T is denoted as $u = [u_0, u_1, \ldots, u_{T-1}]$. For each timestep t:

$$u_t = (1 - \alpha_u) u_{t-1} + \alpha_u \sum_{k=0}^{N \cdot K - 1} \tilde{\omega}_k(V) v_{t,k} \tag{14}$$

where α_u is the step size that regularizes the current solution to be close to the previous u_{t-1}. The optimal control is set to $u_0^* = u_0$. Note that through (13), we can smoothly fuse different strategies to achieve a goal in a general way.

The pseudocode is summarized in Algorithm 4. After the initialization, we sample Halton splines and forward simulate the plan alternatives using IsaacGym to compute the costs (Lines 8-18). The costs are then used to update the weights for each plan and update their means (Lines 20–24). Finally, the mean of the overall action sequence is updated (Line 28), and the first action from the mean is executed.

D. Plan Interface

The plan interface is a component that takes the possible alternative symbolic actions in $\mathcal{P}$ and links them to their corresponding cost functions, forwarding the latter to M3P2I. For every symbolic action a robot can perform, we store a cost function in a database that we can query at runtime, bridging the output of the action planner to the motion planner.

IV. Experiments

We evaluate the performance of our method in two different scenarios. The first is a *push-pull scenario* for non-prehensile

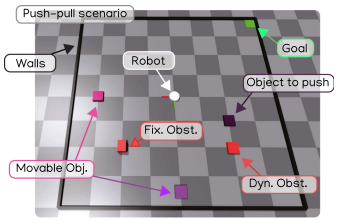

Fig. 2. Push-pull scenario. The dark purple object has to be placed on the green area. The robot can pull or push the object while avoiding dynamic and fixed obstacles. The objects and goals can have different initial positions.

Algorithm 4: Multi-Modal Model Predictive Path Integral Control (M3P2I).

Input: cost functions: $C_i, \forall i \in [0, N)$
Parameters: N, K, T
Initial sequence: $\mu_i = \mathbf{0}, u = \mathbf{0}, \in \mathbb{R}^T \; \forall i \in [0, N)$
while task not completed **do**
  $x \leftarrow GetStateEstimate()$
  $InitIsaacGym(x)$
  / Begin parallel sampling of alternatives */*
  for $i = 0$ to $N-1$ **do**
    for $k \in \kappa(i)$ **do**
      $S_i(V_k) \leftarrow 0$
      Sample noise $\mathcal{E}_k \leftarrow SampleHaltonSplines()$
      $\mu_i \leftarrow BackShift(\mu_i)$
      for $t = 0$ to $T-1$ **do**
        $Q_i(V_k) \leftarrow ComputeTrajIsaacGym(\mu_i + \mathcal{E}_k)$
        $S_i(V_k) \leftarrow UpdateCost(C_i, Q_i(V_k))$ ▷ (8)
      end for
    end for
  end for
  / Begin computing trajectory weights */*
  for $i = 0$ to $N-1$ **do**
    $\rho_i, \eta_i, \beta_i \leftarrow UpdateInvTemp(i)$ ▷ Algorithm 3
    $\omega_i(k) \leftarrow \frac{1}{\eta_i} \exp(-\frac{1}{\beta_i}(S_i(V_k) - \rho_i)), \forall k$ ▷ (9)
    $\mu_i = \sum_{k \in \kappa(i)} \omega_i(V_k) V_k$ ▷ (12)
  end for
  / Begin control update */*
  $\tilde{\omega}(V) = \frac{1}{\eta} \exp(-\frac{1}{\beta}(\tilde{S}(V) - \rho))$ ▷ (13)
  for $t = 0$ to $T-1$ **do**
    $u_t = (1-\alpha_u)u_{t-1} + \alpha_u \sum_{j=0}^{N \cdot K-1} \tilde{\omega}_k v_{t,k}$ ▷ (14)
  end for
  $ExecuteCommand(u_0^* = u_0)$
  $u = BackShift(u)$
end while

manipulation of an object with an omnidirectional robot. The second is a *object stacking scenario* with a 7-DOF manipulator with dynamic obstacles and external disturbances at runtime.

A. Push-Pull Scenario

This scenario is depicted in Fig. 2. One object has to be placed to a goal, situated in one of the corners of an arena. The object can have different initial locations, for instance, in the middle of the arena or on one of the corners. There are also static and dynamic obstacles, and the robot can push or pull the object.

We define the following action templates for AIP and the cost functions for M3P2I.

1) Action Templates for AIP: The AIP for this task requires one state $s^{(goal)}$ and a relative symbolic observation $o^{(goal)}$ that indicates when an object is at the goal. This is defined as:

$$o^{(goal))} = \begin{cases} 0, ||p_G - p_O|| \leq \delta \\ 1, ||p_G - p_O|| > \delta \end{cases} \tag{15}$$

where p_G, p_O represent the positions of the goal and the object in a 3D coordinate system. δ is a constant threshold determined by the user. The mobile robot can either push, pull, or move. These skills are encoded in the action planner as follows:

Actions	Preconditions	Postconditions
`push(obj,goal)`	-	$l^{(goal)} = [1\,0]^\top$
`pull(obj,goal)`	-	$l^{(goal)} = [1\,0]^\top$

The postcondition of the action `push(obj, goal)` is that the object is at the goal, similarly for the pull action. Note that we do not add complex heuristics to encode the geometric relations in the task planner to determine when to push or pull; instead, we will exploit parallel sampling in the motion planner later. The desired state s_d of this task is set as a preference for $l^{(goal)} = [1\,0]^\top$. The BT would contain more desired states for pushing or pulling several blocks. Our approach can be extended to multiple objects in different locations, for instance, and accommodate more involved pre-post conditions and fallbacks since it has the same properties as in [7].

2) Cost Functions for M3P2I: We need to specify a cost for each symbolic action. The cost function for pushing object O to the goal G is defined as:

$$C_{push}(R, O, G) = C_{dist}(R, O) + C_{dist}(O, G) + C_{ori}(O, G) + C_{align\_push}(R, O, G) \tag{16}$$

where minimizing $C_{dist}(O, G) = \omega_{dist} \cdot ||p_G - p_O||$ makes the object O close to the goal G. $C_{ori}(O, G) = \omega_{ori} \cdot \phi(\Sigma_O, \Sigma_G)$ defines the orientation cost between the object O and goal G. We define ϕ for symmetric objects as:

$$\phi(\Sigma_u, \Sigma_v) = \min_{i,j \in \{1,2,3\}} (2 - ||\vec{u}_1 \cdot \vec{v}_i|| - ||\vec{u}_2 \cdot \vec{v}_j||) \tag{17}$$

where $\Sigma_u = \{\vec{u}_1, \vec{u}_2, \vec{u}_3\}, \Sigma_v = \{\vec{v}_1, \vec{v}_2, \vec{v}_3\}$ form the orthogonal bases of two coordinates systems. Minimizing this cost makes two axes in the coordinate systems of the object and goal coincide. The orientation cost for asymmetric objects can be extended from (17) by aligning the corresponding axes.

The align cost $C_{align\_push}(R, O, G)$ is defined as:

$$C_{align\_push}(R, O, G) = \omega_{align\_push} \cdot h(cos(\theta)), \tag{18}$$

$$\cos(\theta) = \frac{(p_R - p_O) \cdot (p_G - p_O)}{||p_R - p_O|| \cdot ||p_G - p_O||}, \tag{19}$$

$$h(\cos(\theta)) = \begin{cases} 0, \cos(\theta) \leq 0 \\ \cos(\theta), \cos(\theta) > 0 \end{cases} \tag{20}$$

This makes the object O lie at the center of robot R and goal G so that the robot can push it, as illustrated in Fig. 3.

Similarly, the cost function of making the robot R pull object O to the goal G can be formulated as:

$$C_{pull}(R, O, G) = C_{dist}(R, O) + C_{dist}(O, G) + C_{ori}(O, G) + C_{align\_pull}(R, O, G) + C_{act\_pull}(R, O, G) \tag{21}$$

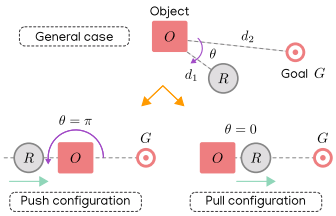

Fig. 3. Push and pull ideal configurations. The robot R has to push or pull the object O to the goal G.

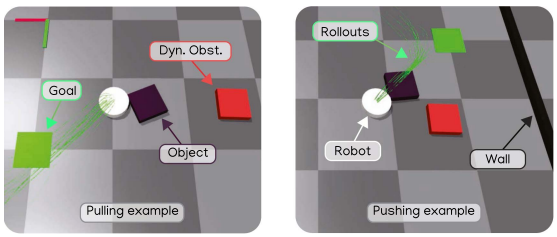

Fig. 4. Illustrative example of pulling and pushing a block to a goal. The strategy differs according to the object, goal location, and dynamic obstacle position. What action to perform is decided at runtime through multi-modal sampling.

where the align cost $C_{align\_pull}(R, O, G)$ makes the robot R lie between the object O and goal G, see Fig. 3. While pulling, we simulate a suction force in IsaacGym, and we are only allowed to sample control inputs that move away from the object through $C_{act\_pull}(R, O, G)$. Mathematically:

$$C_{align\_pull}(R, O, G) = \omega_{align\_pull} \cdot h(-\cos(\theta)) \tag{22}$$

$$C_{act\_pull}(R, O, G) = \omega_{act\_pull} \cdot h\left(\frac{(p_O - p_R) \cdot \vec{u}}{||p_O - p_R|| \cdot ||\vec{u}||}\right) \tag{23}$$

An example can be seen in Fig. 4. We also consider an additional cost $C_{dyn\_obs}(R, D)$ to avoid collisions with (dynamic) obstacles while operating. The dynamic obstacle is assumed to move in a certain direction with constant velocity. We use a constant velocity model to predict the position of the dynamic obstacle D in the coming horizon and try to maximize the distance between the latter and the robot:

$$C_{dyn\_obs}(R, D) = \omega_{dyn\_obs} \cdot e^{-||p_R - p_{D_{pred}}||} \tag{24}$$

where $p_{D_{pred}}$ is the predicted position of dynamic obstacle.

3) Results: We test the performance of our approach in two configurations: a) the object is in the middle of the arena, and the goal is to one corner, and b) both the object and the goals are in different corners. For each arena configuration, we test three cases: the robot can either only push, only pull, or combine the two through our M3P2I. The AIP plans for the two alternatives, pushing and pulling, and forwards the solution to the plan interface. Then, M3P2I starts minimizing the costs until the AIP observes the completion of the task. We performed 20 trials per case, per arena configuration, for a total of 120 simulations. By only pulling an object, the robot cannot tightly place it on top of the goal in the corner; on the other hand, by only pushing, the robot cannot retrieve the object from the corner. Using multi-modal motion, we can complete the task in every tested configuration. Table I shows that the multi-modal case outperforms push and pull in both arena configurations. It presents lower position and orientation errors and a shorter planning and execution time.

TABLE I
SIMULATION RESULTS OF PUSH AND PULL

Case	Skill	Mean(std) pos error	Mean(std) ori error	Mean(std) time (s)
Middle-corner	Push	0.1061 (0.0212)	0.0198 (0.0217)	6.2058 (6.8084)
	Pull	0.1898 (0.0836)	0.0777 (0.1294)	25.1032 (13.7952)
	Multi-modal	**0.1052** (0.0310)	**0.0041** (0.0045)	**3.7768** (0.8239)
Corner-corner	Push	7.2679 (3.2987)	0.0311 (0.0929)	time-out
	Pull	0.3065 (0.1778)	0.1925 (0.2050)	32.8838 (7.9240)
	Multi-modal	**0.1375** (0.0091)	**0.0209** (0.0227)	**9.9473** (3.4591)

The bold is used to indicate optimal values.

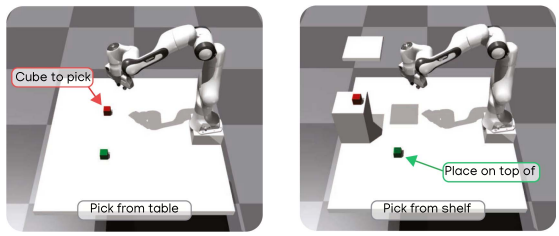

Fig. 5. Pick-place scenarios. The red cube has to be placed on top of the green cube. The red cube can be either on the table or a constrained shelf, requiring different pick strategies from the top or the side, respectively.

B. Object Stacking Scenario

We address the challenge of stacking objects with external task disruptions, necessitating adaptive actions like re-grasping with different pick configurations (e.g., top or side picking in Fig. 5). We showcase the robot's ability to rectify plans by repeating actions or compensating for unplanned occurrences, such as unexpected obstacles obstructing the path. We benchmark against the cube-stacking task outlined in [25].

1) Action Templates for AIP: For this task, we define the following states $s^{(reach)}$, $s^{(hold)}$, $s^{(preplace)}$, $s^{(placed)}$, and their corresponding symbolic observations. The robot has four symbolic actions, summarized below:

Actions	Preconditions	Postconditions
`reach(obj)`	-	$l^{(reach)} = [1\ 0]^\top$
`pick(obj)`	`reachable(obj)`	$l^{(hold)} = [1\ 0]^\top$
`prePlace(obj)`	`holding(obj)`	$l^{(preplace)} = [1\ 0]^\top$
`place(obj)`	`atPreplace(obj)`	$l^{(placed)} = [1\ 0]^\top$

The symbolic observers to estimate the states are defined as follows. To estimate whether the gripper is close enough to the cube, we define the relative observation $o^{(reach)}$. We set $o^{(reach)} = 0$ if $\delta_r \leq \delta$, where $\delta_r = ||p_{ee} - p_O||$ measures the distance between the end effector ee and the object O. $o^{(reach)} = 1$ otherwise. To estimate whether the robot is holding the cube of size 0.06 m,

we define:

$$o^{(hold)} = \begin{cases} 0, \delta_f < 0.06 + \delta \text{ and } \delta_f \geq 0.06 - \delta \\ 1, \delta_f \geq 0.06 + \delta \text{ or } \delta_f \leq 0.06 - \delta \end{cases} \quad (25)$$

where $\delta_f = ||p_{ee\_l} - p_{ee\_r}||$ measures the distance between the two gripper's fingers. To estimate whether the cube reaches the pre-place location, we define:

$$o^{(preplace)} = \begin{cases} 0, C_{dist}(O,P) < \delta \text{ and } C_{ori}(O,P) < \delta \\ 1, C_{dist}(O,P) \geq \delta \text{ or } C_{ori}(O,P) \geq \delta \end{cases} \quad (26)$$

where $C_{dist}(O,P)$ and $C_{ori}(O,P)$ measure the distance and the orientation between the object O and the pre-place location P as in (16). The pre-place location is a few centimeters higher than the target cube location, directly on top of the green cube. We use the same logic as (26) for $o^{(placed)}$ where the place location is directly on top of the cube location. The desired state for this task is set to be $l^{(placed)} = [1\ 0]^T$, meaning the cube is correctly placed on top of the other. Note that in more complex scenarios, such as rearranging many cubes, the BT can guide the AIP as demonstrated in [7].

2) Cost Functions for M3P2I: At the motion planning level, the cost functions for the four actions are formulated as:

$$C_{reach}(ee, O, \psi) = \omega_{reach} \cdot ||p_{ee} - p_O|| + \omega_{tilt} \cdot \left(\frac{||\vec{z}_{ee} \cdot \vec{z}_O||}{||\vec{z}_{ee}|| \cdot ||\vec{z}_O||} - \psi \right) \quad (27)$$

$$C_{pick}(ee) = \omega_{gripper} \cdot l_{gripper} \quad (28)$$

$$C_{preplace}(O,P) = C_{dist}(O,P) + C_{ori}(O,P) \quad (29)$$

$$C_{place}(O,P) = \omega_{gripper} \cdot (1 - l_{gripper}) \quad (30)$$

$C_{reach}(ee, O, \psi)$ moves the end effector close to the object with a grasping tilt constraint ψ. As ψ approaches 1, the gripper becomes perpendicular to the object; as it nears 0, the gripper aligns parallel to the object's supporting plane.

3) Results - Reactive Pick and Place: We first consider the pick-and-place under disturbances. We model disturbances by changing the position of the cubes at any time. We compare the performance of our method with the off-the-shelf RL method [25]. This is a readily available Actor-Critic RL example from IsaacGym, which considers the same tabletop configuration and robot arm. We compare the methods in a *vanilla* task without disturbances and a *reactive* task with disturbances. It should be noticed that the cube-stacking task in [25] only considers moving the cube on top of the other cube while neglecting the action of opening the gripper and releasing the cube. In contrast, our method exhibits fluent transitions between pick and place and shows robustness to interferences such as repick during the long-horizon task execution. Results are available in Table II, with 50 trials per case. While the RL agent shows a slightly lower position error in the vanilla case, our method outperforms it in the reactive task. Planning and execution time for smooth pick-and-place with our method is approximately 5 to 10 s.

4) Results - Multi-Modal Grasping: In this case, we consider grasping the object with different grasping poses by sampling two alternatives in parallel. That is, pick from the top or the side to cover the cases when the object is on the table or the constrained shelf with an obstacle above. To do so, we use the proposed M3P2I and incorporate the cost functions of $C_{reach}(ee, O, \psi = 0)$ and $C_{reach}(ee, O, \psi = 1)$ as shown in (27). This allows for a smooth transition between top and side grasp according to the geometry of the problem, see Fig. 6.

TABLE II
SIMULATION RESULTS OF REACTIVE PICK AND PLACE

Task	Method	Training epochs	Mean(std) pos error
Vanilla	Ours	0	0.0075 (0.0036)
	RL	1500	**0.0042** (0.0019)
Reactive	Ours	0	**0.0117** (0.0166)
	RL	1500	0.0246 (0.0960)

The bold is used to indicate optimal values.

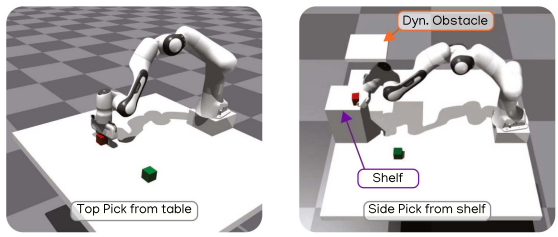

Fig. 6. Example of different picking strategies computed by our multi-modal MPPI. The obstacle on top of the shelf can be moved, simulating a dynamic obstacle.

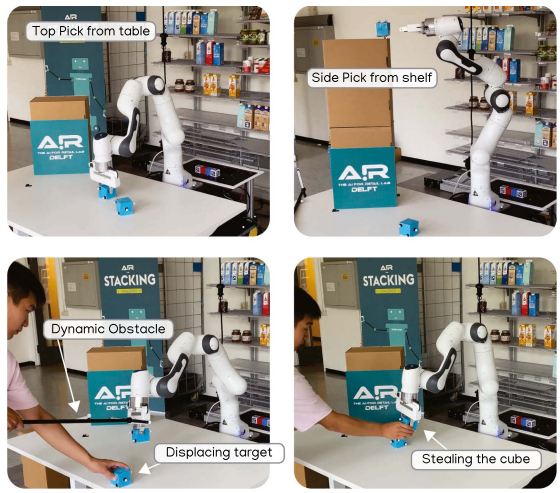

Fig. 7. Real-world experiments of picking a cube from the table or the shelf while avoiding dynamic obstacles and recovering from task disturbances.

5) Results - Real-World Experiments: Our real-world validation of reactive pick-and-place, depicted in Fig. 7, involves avoiding a moving stick and disturbances such as movement and theft of the cube. M3P2I enables smooth execution and recovery while using different grasp configurations.

V. DISCUSSION

In this section, we discuss key aspects of our solution and potential future work. The main strength of M3P2I is its ability to reason over discrete alternative actions at the motion planning level. This is enabled by sampling different control sequences for each alternative symbolic action and then blending them through importance sampling. We thus alleviate the task planning burden

by eliminating logic heuristics to switch between these actions. Sampling alternatives at the motion planning level increases robustness during execution, at the price of slightly degrading the performance since the control distribution is also slightly biased towards less effective strategies, as shown in [24]. The performance of M3P2I also depends on the weight tuning of the cost functions. In this case, implementing auto-tuning techniques can reduce manual effort [26]. The cost functions also need to capture the essence of the skills. The AIP requires manually defined symbolic action templates and a set of discrete states. The discrete desired states need to be encoded in a sequence in a BT or can be as simple as encoding the end state for a task, as in our examples. To transfer from simulation to the real world, we considered randomization of object properties in the rollouts [10]. Online system identification could be added to achieve better performance with uncertain model parameters [20].

VI. Conclusion

In this letter, to address the runtime geometric uncertainties and disturbances, we proposed a method to combine the adaptability of an Active Inference planner (AIP) for high-level action selection with a novel Multi-Modal Model Predictive Path Integral Controller (M3P2I) for low-level control. We modified the AIP to generate plan alternatives that are linked to costs for M3P2I. The motion planner can sample the plan alternatives in parallel, and it computes the control input for the robot by smoothly blending different strategies. In a push-pull task, we demonstrated how our proposed framework can blend both push and pull actions, allowing it to deal with corner cases where approaches only using a single plan fail. With a simulated manipulator, we showed our method outperforming a reinforcement learning baseline when the environment is disturbed while requiring no training. Simulated and real-world experiments demonstrated how our approach solves reactive object stacking tasks with a manipulator subject to severe disturbances and various scene configurations that require different grasp strategies.

References

[1] C. R. Garrett et al., "Integrated task and motion planning," *Annu. Rev. Control Robot. Auton. Syst.*, vol. 4, pp. 265–293, 2021.

[2] M. Toussaint, "Logic-geometric programming: An optimization-based approach to combined task and motion planning," in *Proc. 24th Int. Joint Conf. Artif. Intell.*, 2015, pp. 1930–1936.

[3] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "PDDLStream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning," in *Proc. Int. Conf. Autom. Plan. Scheduling*, 2020, pp. 440–448.

[4] M. Toussaint, J. Harris, J.-S. Ha, D. Driess, and W. Hönig, "Sequence-of-constraints MPC: Reactive timing-optimal control of sequential manipulation," in *Proc. IEEE Int. Conf. Intell. Robots Syst.*, 2022, pp. 13753–13760.

[5] T. Migimatsu and J. Bohg, "Object-centric task and motion planning in dynamic environments," *IEEE Robot. Automat. Lett.*, vol. 5, no. 2, pp. 844–851, Apr. 2020.

[6] S. Li, D. Park, Y. Sung, J. A. Shah, and N. Roy, "Reactive task and motion planning under temporal logic specifications," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2021, pp. 12618–12624.

[7] C. Pezzato, C. Hernandez, S. Bonhof, and M. Wisse, "Active inference and behavior trees for reactive action planning and execution in robotics," *IEEE Trans. Robot.*, vol. 39, no. 2, pp. 1050–1069, Apr. 2023.

[8] N. Castaman, E. Pagello, E. Menegatti, and A. Pretto, "Receding horizon task and motion planning in changing environments," *Robot. Auton. Syst.*, vol. 145, 2021, Art. no. 103863.

[9] M. Colledanchise, D. Almeida, and P. Ögren, "Towards blended reactive planning and acting using behavior tree," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2019, pp. 8839–8845.

[10] C. Pezzato, C. Salmi, M. Spahn, E. Trevisan, J. Alonso-Mora, and C. Hernández, "Sampling-based model predictive control leveraging parallelizable physics simulations," 2023, *arXiv:2307.09105*.

[11] C. Paxton, N. Ratliff, C. Eppner, and D. Fox, "Representing robot task plans as robust logical-dynamical systems," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2019, pp. 5588–5595.

[12] J. Harris, D. Driess, and M. Toussaint, "FC^3: Feasibility-based control chain coordination," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2022, pp. 13769–13776.

[13] B. Huang, A. Boularias, and J. Yu, "Parallel Monte Carlo tree search with batched rigid-body simulations for speeding up long-horizon episodic robot planning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2022, pp. 1153–1160.

[14] Z. Zhou, D. J. Lee, Y. Yoshinaga, S. Balakirsky, D. Guo, and Y. Zhao, "Reactive task allocation and planning for quadrupedal and wheeled robot teaming," in *Proc. IEEE Int. Conf. Automat. Sci. Eng.*, 2022, pp. 2110–2117.

[15] M. Bangura and R. Mahony, "Real-time model predictive control for quadrotors," *IFAC Proc. Volumes*, vol. 47, no. 3, pp. 11773–11780, 2014.

[16] N. Scianca, D. De Simone, L. Lanari, and G. Oriolo, "MPC for humanoid gait generation: Stability and feasibility," *IEEE Trans. Robot.*, vol. 36, no. 4 , pp. 1171–1188, Aug. 2020.

[17] M. Spahn, B. Brito, and J. Alonso-Mora, "Coupled mobile manipulation via trajectory optimization with free space decomposition," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2021, pp. 12759–12765.

[18] G. Williams et al., "Information theoretic MPC for model-based reinforcement learning," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2017, pp. 1714–1721.

[19] M. Bhardwaj et al., "STORM: An integrated framework for fast joint-space model-predictive control for reactive manipulation," in *Proc. Conf. Robot Learn.*, PMLR, 2022, pp. 750–759.

[20] I. Abraham, A. Handa, N. Ratliff, K. Lowrey, T. D. Murphey, and D. Fox, "Model-based generalization under parameter uncertainty using path integral control," *IEEE Robot. Automat. Lett.*, vol. 5, no. 2, pp. 2864–2871, Apr. 2020.

[21] T. Howell, N. Gileadi, S. Tunyasuvunakool, K. Zakka, T. Erez, and Y. Tassa, "Predictive sampling: Real-time behaviour synthesis with MuJoCo," 2022, *arXiv:2212.00541*.

[22] G. Williams, A. Aldrich, and E. A. Theodorou, "Model predictive path integral control: From theory to parallel computation," *J. Guid. Control Dyn.*, vol. 40, no. 2, pp. 344–357, Feb. 2017.

[23] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Information-theoretic model predictive control: Theory and applications to autonomous driving," *IEEE Trans. Robot.*, vol. 34, no. 6, pp. 1603–1622, Dec. 2018.

[24] E. Trevisan and J. Alonso-Mora, "Biased-MPPI: Informing sampling-based model predictive control by fusing ancillary controllers," *IEEE Robot. Automat. Lett.*, vol. 9, no. 6, pp. 5871–5878, Jun. 2024.

[25] V. Makoviychuk et al., "Isaac gym: High performance GPU-based physics simulation for robot learning," 2021, *arXiv:2108.10470*.

[26] M. Spahn and J. Alonso-Mora, "Autotuning symbolic optimization fabrics for trajectory generation," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2023, pp. 11287–11293.