Task Decomposition for Iterative Learning Model Predictive Control

Charlott Vallon and Francesco Borrelli

Abstract— A task decomposition method for iterative learning model predictive control is presented. We consider a constrained nonlinear dynamical system and assume the availability of state-input pair datasets which solve a task $\mathcal{T}1$. Our objective is to find a feasible model predictive control policy for a second task, $\mathcal{T}2$, using stored data from $\mathcal{T}1$. Our approach applies to tasks $\mathcal{T}2$ which are composed of subtasks contained in $\mathcal{T}1$. In this paper we formally define subtasks and the task decomposition problem, and provide proofs of feasibility and iteration cost improvement over simple initializations. We demonstrate the effectiveness of the proposed method on autonomous racing and robotic manipulation experiments.

I. INTRODUCTION

Control design for systems repeatedly performing a single task has been studied extensively. Such problems arise frequently in practical applications [1], [2] and examples range from autonomous cars racing around a track [3]–[5] to robotic system manipulators [6]–[9]. Iterative Learning Controllers (ILCs) aim to autonomously improve a system's closed-loop reference tracking performance at each iteration of a repeated task, while rejecting periodic disturbances [1], [10]. In classical ILC, the controller uses tracking error data from previous task iterations to better track a provided reference trajectory during the current iteration. Recent work has also explored reference-free ILC strategies for tasks whose goals are better defined in terms of an economic metric, rather than a reference trajectory. The controller again uses previous iteration data to improve closed-loop performance with respect to the chosen performance metric. Examples include autonomous racing tasks (e.g. "minimize lap time") [5], [11], or optimizing flight paths for tethered energy-harvesting systems (e.g. "maximize average power generation") [12].

The aforementioned iterative learning methods require either a reference trajectory to track (classical ILC) or a feasible trajectory with which to initialize the iterative control algorithm (reference-free ILC). If the task changes, a new trajectory needs to be redesigned to match the new task. This can be challenging for complex tasks.

There are many methods that allow controllers to use data collected while solving a task to efficiently solve variations of that task, including model-based and model-free methods. A complete review of the literature is outside the scope of this paper. Here, we focus specifically on model-based methods

for using stored trajectories from previous tasks in order to find feasible trajectories for new tasks. The authors in [13] propose running a desired planning method in parallel with a retrieve and repair algorithm that adapts reference trajectories from previous tasks to the constraints of a new task. Retrieve and repair was shown to decrease overall planning time, but requires checking for constraint violations at each point along a retrieved trajectory. In [14], environment features are used to divide a task and create a library of local trajectories in relative state space frames. These trajectories are then pieced back together in real-time according to the features of the new task environment. A trajectory library built using differential dynamic programming is used in [15] to design a controller for balance control in a humanoid robot. At each time step, a trajectory is selected from the library based on current task parameter estimates and a k-nearest neighbor selection scheme. A similar method is explored in [16], where differential dynamic programming is combined with receding horizon control. While these methods can decrease planning time, they verify or interpolate saved trajectories at every time step, which can be inefficient and unnecessary.

The authors in [17] propose piecing together stored trajectories corresponding to discrete system dynamics only at states of dynamics transition. However, this method only applies to discontinuities in system dynamics, and does not generalize to other task variations. In [18], a static map is learned between a given reference trajectory and the input sequence required to make a linear time invariant system track that reference trajectory. Once learned, this map can be used to determine an input sequence that lets the linear system track a new reference trajectory.

In this paper, our objective is to find a feasible trajectory to smartly initialize an Iterative Learning Model Predictive Controller (ILMPC) [19] for a new task, using data from previous tasks. ILMPC is a type of reference-free ILC that uses a *safe set* to design a model predictive control (MPC) policy for an iterative control task. The ILMPC safe set is initialized using a feasible task trajectory.

We consider a constrained nonlinear dynamical system and assume the availability of a dataset containing states and inputs corresponding to multiple iterations of variations of a task $\mathcal{T}1$. This dataset can be stored explicitly (e.g. by human demonstrations [20] or an iterative controller like ILMPC [19]) or generated by roll-out of a given policy (e.g. a hand-tuned controller). We introduce a Task Decomposition for ILMPC algorithm (TDMPC), and show how to use the stored $\mathcal{T}1$ dataset to efficiently construct a non-empty ILMPC safe set for task $\mathcal{T}2$ (a new variation of $\mathcal{T}1$), containing feasible trajectories for $\mathcal{T}2$.

The contributions of this paper are twofold.

1) We first present how to build the aforementioned $\mathcal{T}2$ safe set using the TDMPC algorithm. TDMPC reduces the complexity to adapt trajectories from $\mathcal{T}1$ to a new task $\mathcal{T}2$ by decomposing task $\mathcal{T}1$ into different modes of operation, called subtasks. The stored $\mathcal{T}1$ trajectories are adapted to $\mathcal{T}2$ only at points of subtask transition, by solving one-step controllability problems.

2) We prove that the resulting safe set based ILMPC policy is feasible for $\mathcal{T}2$, and the corresponding closed-loop trajectories have lower iteration cost compared to an ILMPC initialized using simple methods.

II. PROBLEM DEFINITION

A. Safe Set Based ILMPC

We consider a system

$$x_{k+1} = f(x_k, u_k), \tag{1}$$

where $f(x_k, u_k)$ is the dynamical model, subject to the constraints

$$x_k \in \mathcal{X}, \ u_k \in \mathcal{U}. \tag{2}$$

The vectors x_k and u_k collect the states and inputs at time step k. We pick a set $\mathcal{P} \subset \mathcal{X}$ to be the target set for an iterative task $\mathcal{T}$, performed repeatedly by system (1) and defined by the tuple

$$\mathcal{T} = \{\mathcal{X}, \mathcal{U}, \mathcal{P}\}. \tag{3}$$

Assumption 1: $\mathcal{P}$ is a control invariant set [21, Sec 10.9]:

$$\forall x_k \in \mathcal{P}, \ \exists u_k \in \mathcal{U} \ : \ x_{k+1} = f(x_k, u_k) \in \mathcal{P}.$$

Each task execution is referred to as an iteration. The goal of an ILMPC is to solve at each iteration the optimal task completion problem:

$$V_{0 \to T}^{\star}(x_0) = \min_{T, u_0, \ldots, u_{T-1}} \sum_{k=0}^{T} h(x_k, u_k) \tag{4}$$

$$\text{s.t.} \quad x_{k+1} = f(x_k, u_k),$$
$$x_k \in \mathcal{X}, u_k \in \mathcal{U} \ \ \forall k \geq 0,$$
$$x_T \in \mathcal{P},$$

where $V_{0 \to T}^{\star}(x_0)$ is the optimal cost-to-go from the initial state x_0, and $h(x_k, u_k)$ is a chosen stage cost.

At the j-th successful task iteration, the vectors

$$E^j(\mathcal{T}) = [\mathbf{x}^j, \mathbf{u}^j] \tag{5a}$$
$$\mathbf{x}^j = [x_0^j, x_1^j, \ldots, x_{T^j}^j], \ x_k^j \in \mathcal{X} \ \ \forall k \in [0, T^j],$$
$$x_{T^j}^j \in \mathcal{P}, \tag{5b}$$
$$\mathbf{u}^j = [u_0^j, u_1^j, \ldots u_{T^j}^j], \ u_k^j \in \mathcal{U} \ \ \forall k \in [0, T^j], \tag{5c}$$

collect the inputs applied to system (1) and the corresponding state evolution. In (5), x_k^j and u_k^j denote the system state and control input at time k of the j-th iteration, and T^j is the duration of the j-th iteration.

After J number of iterations, we define the *sampled safe state set* and *sampled safe input set* as:

$$\mathcal{SS}^J = \left\{ \bigcup_{j=1}^{J} \mathbf{x}^j \right\}, \ \mathcal{SU}^J = \left\{ \bigcup_{j=1}^{J} \mathbf{u}^j \right\}, \tag{6}$$

where $\mathcal{SS}^J$ contains all states visited by the system in previous task iterations, and $\mathcal{SU}^J$ the corresponding inputs applied at each of these states. Hence, by construction of the safe set, for any state in $\mathcal{SS}^J$ there exists a feasible input sequence contained in $\mathcal{SU}^J$ to reach the goal set $\mathcal{P}$ while satisfying state and input constraints (2).

Similarly, we define the *sampled cost set* as:

$$\mathcal{SQ}^J = \left\{ \bigcup_{j=1}^{J} \mathbf{q}^j \right\} \tag{7a}$$
$$\mathbf{q}^j = [V^j(x_0^j), V^j(x_1^j), \ldots, V^j(x_{T^j}^j)], \tag{7b}$$

where $V^j(x_k^j)$ is the realized cost-to-go from state x_k^j at time step k of the j-th task execution:

$$V^j(x_k^j) = \sum_{i=k}^{T^j} h(x_i^j, u_i^j). \tag{8}$$

The safe set based ILMPC policy tries to solve (4) by using state and input data collected during past task iterations, stored in the sampled safe sets. At time k of iteration $J+1$, we solve the optimal control problem:

$$V^{\text{ILMPC}, J+1}(x_k^{J+1}) = \tag{9}$$

$$\min_{u_{k|k}, \ldots, u_{k+N-1|k}} \sum_{t=k}^{k+N-1} h(x_{t|k}, u_{t|k}) + V^J(x_{t+N|k})$$

$$\text{s.t.} \quad x_{t+1|k} = f(x_{t|k}, u_{t|k}), \forall t \in [k, k+N-1],$$
$$x_{t|k} \in \mathcal{X}, \ u_{t|k} \in \mathcal{U}, \ \ \forall t \in [k, k+N-1],$$
$$x_{k|k} = x_k^j,$$
$$x_{k+N|k} \in \mathcal{SS}^J \cup \mathcal{P},$$

which searches for an input sequence over a chosen planning horizon N that controls the system (1) to the state in the sampled safe state set or task target set $\mathcal{P}$ with the lowest cost-to-go (8). We then apply a receding horizon strategy:

$$u(x_k^j) = \pi^{\text{ILMPC}}(x_k^j) = u_{k|k}^{\star}. \tag{10}$$

A system (1) in closed-loop with (10) leads to a feasible task execution if $x_0 \in \mathcal{SS}^J$. At each time step, the ILMPC policy searches for the optimal input based on previous task data, leading to performance improvement on the task as the sampled safe sets continue to grow with each subsequent iteration. For details on ILMPC, we refer to [22].

The sampled safe sets used in the ILMPC policy (10) must first be initialized to contain at least one feasible task execution. The aim of TDMPC is to use data collected from a task $\mathcal{T}1$ in order to efficiently design sampled safe sets for a new task $\mathcal{T}2$. Such a set will induce an ILMPC policy (10) that can be used to directly solve or initialize an ILMPC for

$\mathcal{T}2$. We approach this using *subtasks*, formalized in Sec. II-B, and the concept of controllability.

Definition: A system (1) is *N-step controllable* from an initial state x_0 to a terminal state x_P if there exists an input sequence $[u_0, u_1, ..., u_{N-1}] \in \mathcal{U}$ such that the corresponding state trajectory satisfies state constraints (2) and $x_N = x_P$. A system is *controllable* from x_0 to x_P if there exists an $N > 0$ such that the system is N-step controllable to x_P. [21, Sec 10.1]

B. Subtasks

Consider an iterative task $\mathcal{T}$ (3) and a sequence of M subtasks, where the i-th *subtask* $\mathcal{S}_i$ is the tuple

$$\mathcal{S}_i = \{\mathcal{X}_i, \mathcal{U}_i, \mathcal{R}_i\}. \tag{11}$$

We take $\mathcal{X}_i \subseteq \mathcal{X}$ as the subtask workspace, $\mathcal{U}_i \subseteq \mathcal{U}$ the subtask input space, and $\mathcal{R}_i$ the set of transition states from the current subtask $\mathcal{S}_i$ workspace into the subsequent $\mathcal{S}_{i+1}$ workspace:

$$\mathcal{R}_i \subseteq \mathcal{X}_i = \{x \in \mathcal{X}_i : \exists u \in \mathcal{U}_i,\ f(x,u) \in \mathcal{X}_{i+1}\}.$$

A successful *subtask execution $E(\mathcal{S}_i)$* of a subtask $\mathcal{S}_i$ is a trajectory of inputs and corresponding states evolving according to (1) while respecting state and input constraints (2), ending in the transition set. We define the j-th successful execution of subtask $\mathcal{S}_i$ as

$$E^j(\mathcal{S}_i) = [\mathbf{x}_i^j, \mathbf{u}_i^j], \tag{12a}$$
$$\mathbf{x}_i^j = [x_0^j, x_1^j, ..., x_{T_i^j}^j],\ x_k \in \mathcal{X}_i\ \ \forall k \in [0, T_i^j],$$
$$x_{T_i^j}^j \in \mathcal{R}_i, \tag{12b}$$
$$\mathbf{u}_i^j = [u_0^j, u_1^j, ..., u_{T_i^j}^j],\ u_k \in \mathcal{U}_i\ \ \forall k \in [0, T_i^j],$$

where the vectors $\mathbf{u}_i^j$ and $\mathbf{x}_i^j$ collect the inputs applied to the system (1) and the resulting states, respectively, and x_k^j and u_k^j denote the system state and the control input at time k of subtask execution j. T_i^j is the duration of the j-th execution of subtask i. The final state of each successful subtask execution is in the subtask transition set, from which it can evolve into new subtasks. For the sake of notational simplicity, we have written all subtask executions as beginning at time step $k = 0$.

We say the task $\mathcal{T}$ is an ordered sequence of the M subtasks ($\mathcal{T} = \{\mathcal{S}_i\}_{i=1}^M$) if the j-th successful task execution (5) is equal to the concatenation of successful subtask executions:

$$E^j(\mathcal{T}) = [E^j(\mathcal{S}_1), E^j(\mathcal{S}_2), ..., E^j(\mathcal{S}_M)] = [\mathbf{x}^j, \mathbf{u}^j],$$
$$\mathbf{x}^j = [\mathbf{x}_1^j, \mathbf{x}_2^j, ..., \mathbf{x}_M^j],$$
$$\mathbf{u}^j = [\mathbf{u}_1^j, \mathbf{u}_2^j, ..., \mathbf{u}_M^j],$$
$$f\left(x_{T_{[1\to i]}^j}^j, u_{T_{[1\to i]}^j}^j\right) \in \mathcal{X}_{i+1},\ i \in [1, M-1],$$
$$x_{T_{[1\to M]}^j}^j \in \mathcal{R}_M,$$

where $T_{[1\to i]}^j$ is the duration of the first i subtasks during the

j-th task iteration. When the state reaches a subtask transition set, the system has completed subtask $\mathcal{S}_i$, and it transitions into the following subtask $\mathcal{S}_{i+1}$. The task is completed when the system reaches the last subtask's transition set, $\mathcal{R}_M$, which we consider as the task's control invariant target set (referred to as $\mathcal{P}$ in the previous section).

III. Task Decomposition for ILMPC

In this section we describe the intuition behind TDMPC and provide an algorithm for the method. We prove feasibility and iteration cost reduction of policies output by TDMPC.

A. TDMPC

Let Task 1 and Task 2 be different ordered sequences of the same M subtasks:

$$\mathcal{T}1 = \{\mathcal{S}_i\}_{i=1}^M, \quad \mathcal{T}2 = \{\mathcal{S}_{l_i}\}_{i=1}^M, \tag{14}$$

where the sequence $[l_1, l_2, ..., l_M]$ is a reordering of the sequence $[1, 2, ..., M]$. Assume non-empty sampled safe sets $\mathcal{SS}_{[1\to M]}^J$, $\mathcal{SU}_{[1\to M]}^J$, and $\mathcal{SQ}_{[1\to M]}^J$ (6, 7) containing task data from $\mathcal{T}1$.

The goal of TDMPC is to use the state trajectories stored in the sampled safe sets from Task 1 in order to find feasible trajectories for Task 2, ending in the new target set $\mathcal{R}_{l_M}$, again assumed to be control invariant. The key intuition of the method is that all successful subtask executions from Task 1 are also successful subtask executions for Task 2, as this definition only depends on properties (12) of the subtask itself, not the subtask sequence. Based on this intuition, Algorithm 1 proceeds backwards through the new subtask sequence $[l_1, l_2, ..., l_M]$. The key steps of Algorithm 1 are discussed below.

- Consider subtask $\mathcal{S}_{l_M}$. We know that all states from $\mathcal{S}_{l_M}$ stored in the Task 1 executions are controllable to $\mathcal{R}_{l_M}$ using the stored inputs, i.e. there exists a stored input sequence that can be applied to the state such that the system evolves to be in $\mathcal{R}_{l_M}$. We next look for stored states from the preceding subtask, $\mathcal{S}_{l_{M-1}}$, which are also controllable to $\mathcal{R}_{l_M}$. (Algorithm 1, Lines 4-6) Define the sampled guard set of $\mathcal{S}_{l_{M-1}}$ as

$$\mathcal{SG}_{l_{M-1}} = \left\{ \bigcup_{j=1}^J x_{T_{[1\to l_{M-1}]}^j}^j \right\}. \tag{15}$$

The sampled guard set for subtask l_{M-1} contains the states in $\mathcal{S}_{l_{M-1}}$ from which the system transitioned into another subtask during one of the previous J executions of $\mathcal{T}1$. Only controllability from the sampled guard set will be important in our approach.

- We search for the set of points in $\mathcal{SG}_{l_{M-1}}$ that are controllable to stored states in $\mathcal{S}_{l_M}$. This problem can be solved using a variety of numerical approaches. (Algorithm 1, Lines 9-14)

- For any stored state x in $\mathcal{SG}_{l_{M-1}}$ for which the controllability analysis failed, we remove the stored $\mathcal{SS}_{[l_{M-1}]}^J$ subtask execution ending in x as candidate controllable states for Task 2. All remaining stored states in $\mathcal{S}_{l_{M-1}}$

Algorithm 1 TDMPC algorithm

1: **input** $\mathcal{SS}_{[1 \to M]}^J$, $\mathcal{SU}_{[1 \to M]}^J$, $\mathcal{SQ}_{[1 \to M]}^J$, $[l_1, l_2, ..., l_M]$
2: **do** guard set clustering($\mathcal{SS}_{[1 \to M]}^J$) (15)
3: **initialize empty** $\hat{\mathcal{SS}}, \hat{\mathcal{SU}}$
4: $\hat{\mathcal{SS}}_{l_M} \leftarrow \bigcup_{j=1}^J \mathbf{x}_{l_M}^j$
5: $\hat{\mathbf{u}}_{l_M}^j \leftarrow \mathbf{u}_{l_M}^j \; \forall j \in [1, J], \quad \hat{\mathcal{SU}}_{l_M} \leftarrow \bigcup_{j=1}^J \hat{\mathbf{u}}_{l_M}^j$,
6: $\hat{\mathcal{SQ}}_{l_M} \leftarrow \bigcup_{j=1}^J V^j(\mathbf{x}_{l_M}^j)$
7: **for** $i \in [l_{M-1} : -1 : l_1]$ **do**
8: $\quad$ $\hat{\mathcal{SS}}_i \leftarrow \bigcup_{j=1}^J \mathbf{x}_i^j$
9: $\quad$ $\hat{\mathbf{u}}_i^j \leftarrow \mathbf{u}_i^j \; \forall j \in [1, J]$
10: $\quad$ **for** $x \in \mathcal{SG}_i$ **do**
11: $\quad\quad$ $k = \{k : x \in \mathbf{x}_i^k\}$
12: $\quad\quad$ **initialize empty** $Q^\star$
13: $\quad\quad$ **for** $j : \mathbf{x}_{i+1}^j \in \hat{\mathcal{SS}}_{i+1}$ **do**
14: $\quad\quad\quad$ $\hat{\mathbf{q}}_{i+1}^j = V^j(\mathbf{x}_{i+1}^j)$
15: $\quad\quad\quad$ **solve** $(Q_j^\star, u_j^\star) = \text{Ctrb}(x, \mathbf{x}_{i+1}^j, \hat{\mathbf{q}}_{i+1}^j)$ (17)
16: $\quad\quad$ **if** $Q^\star$ not empty **then**
17: $\quad\quad\quad$ $j_* = \arg\min_j Q_j^\star$
18: $\quad\quad\quad$ $\hat{\mathbf{u}}_i^k[-1] \leftarrow u_{j_*}^\star$
19: $\quad\quad$ **else**
20: $\quad\quad\quad$ $\hat{\mathcal{SS}}_i \leftarrow \hat{\mathcal{SS}}_i \backslash \mathbf{x}_i^k, \quad \hat{\mathcal{SU}}_i \leftarrow \hat{\mathcal{SU}}_i \backslash \hat{\mathbf{u}}_i^k$
21: $\quad$ $\hat{\mathcal{SU}}_i \leftarrow \bigcup_{j=1}^J \hat{\mathbf{u}}_i^j, \quad \hat{\mathcal{SQ}}_i \leftarrow \bigcup_{j=1}^J V^j(\hat{\mathbf{x}}_i^j)$
22: **Return** $\mathcal{SS}_{[l_1 \to l_M]}^0 = \bigcup_{i=l_1}^{l_M} \hat{\mathcal{SS}}_i$
$\quad\quad\quad$ $\mathcal{SU}_{[l_1 \to l_M]}^0 = \bigcup_{i=l_1}^{l_M} \hat{\mathcal{SU}}_i$
$\quad\quad\quad$ $\mathcal{SQ}_{[l_1 \to l_M]}^0 = \bigcup_{i=l_1}^{l_M} \hat{\mathcal{SQ}}_i$
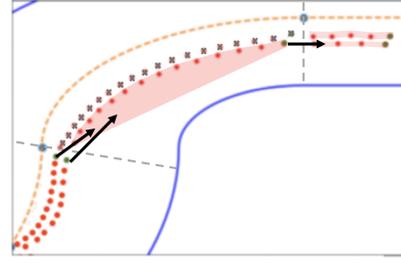

Fig. 1: Algorithm 1 checks controllability from states in the sampled guard set (green) to the convex hulls of safe trajectories through the next subtask (light red). If the controllability fails for a point in the sampled guard set, the backwards reachable states are removed from the safe set (grey). The centerlane is plotted in dashed yellow.

are controllable to stored states in $\mathcal{S}_{l_M}$, and therefore to $\mathcal{R}_{l_M}$. (Algorithm 1, Lines 15-20)

Algorithm 1 iterates backwards through the remaining subtasks, connecting points in subtask sampled guard sets to verified trajectories in the next subtask. Fig. 1 depicts this process across three subtasks from an autonomous racing task detailed in Section IV. The algorithm terminates when it has iterated through the new subtask order, or when no states in a subtask's sampled guard set can be shown to be controllable to $\mathcal{R}_{l_M}$. The algorithm returns sampled safe sets for Task 2 that have been verified through controllability to contain feasible executions of Task 2.

TDMPC can improve on the computational complexity of existing methods in two key ways: *(i)* by verifying stored trajectories only at states in the sampled guard set, rather than at each recorded time step, and *(ii)* by solving a data-driven, one-step controllability problem to adapt the trajectories, rather than a multi-step or set-based controllability method.

B. Properties of TDMPC-Derived Policies

We prove feasibility and iteration cost reduction of ILMPC policies (10) initialized using TDMPC.

Assumption 2: Task 1 and Task 2 are defined as in (14),

where the subtask workspaces and input spaces are given by $\mathcal{X}_i = \mathcal{X}$, $\mathcal{U}_i = \mathcal{U}$ for all $i \in [1, M]$.

Theorem 1: (Feasibility) Let Assumptions 1-2 hold. Assume non-empty sets $\mathcal{SS}_{[1 \to M]}^J$, $\mathcal{SU}_{[1 \to M]}^J$, $\mathcal{SQ}_{[1 \to M]}^J$ containing trajectories of system (1) for Task 1. Assume Algorithm 1 outputs non-empty sets $\mathcal{SS}_{[l_1 \to l_M]}^0$, $\mathcal{SU}_{[l_1 \to l_M]}^0$, $\mathcal{SQ}_{[l_1 \to l_M]}^0$ for Task 2. Then, if $x_0 \in \mathcal{SS}_{[l_1 \to l_M]}^0$, the policy $\pi_{[l_1 \to l_M]}^{\text{ILMPC}}$, as defined in (10), produces a feasible execution of Task 2.

Proof: At every state x_k, the ILMPC policy (10) searches for a sequence of inputs $[u_k, u_{k+1}, ..., u_{k+N-1}]$ such that, when applied to the system (1), the resulting state x_{k+N} is in $\mathcal{SS}_{[l_1 \to l_M]}^0$ or the target set $\mathcal{R}_{l_M}$.

Since all states in $\mathcal{SS}_{[l_1 \to l_M]}^0$ are either stored as part of feasible trajectories to $\mathcal{R}_{l_M}$ or are directly in $\mathcal{R}_{l_M}$, such a sequence of inputs can always be found, and (9) always has a solution:

$$\forall \, x_k \in \mathcal{SS}_{[l_1 \to l_M]}^0, \; \exists \, [u_k, u_{k+1}, ..., u_{k+N-1}] \in \mathcal{U} :$$
$$x_{k+N} \in \mathcal{SS}_{[l_1 \to l_M]}^0 \subseteq \mathcal{X}.$$

As the terminal constraint set in 9 is itself an invariant set, recursive feasibility follows from standard MPC arguments [21]. It follows that the policy $\pi_{[l_1 \to l_M]}^{ILMPC}$ produces feasible trajectories for Task 2.

∎

The above Theorem 1 implies that the safe sets designed by the TDMPC algorithm induce an ILMPC policy that can be used to successfully complete Task 2 while satisfying all input and state constraints.

Assumption 3: Consider Task 1 and Task 2 as defined in (14). The trajectories stored in $\mathcal{SS}_{[1 \to M]}^J$ and $\mathcal{SU}_{[1 \to M]}^J$ correspond to executions of Task 1 by a nonlinear system (1). One stored trajectory corresponds to an execution of (1) in closed-loop with a policy $\pi^0(\cdot)$ that is feasible for both Task 1 and Task 2.

Theorem 2: (Cost Improvement) Let Assumptions 2-3 hold. Then, Algorithm 1 will return non-empty sets $\mathcal{SS}_{[l_1 \to l_M]}^0$, $\mathcal{SU}_{[l_1 \to l_M]}^0$, $\mathcal{SQ}_{[l_1 \to l_M]}^0$ for Task 2. Furthermore, if $x_0 \in \mathcal{SS}_{[1 \to M]}^0$, an ILMPC initialized using $\mathcal{SS}_{[l_1 \to l_M]}^0$ will incur no higher iteration cost during an execution of

Task 2 than an ILMPC initialized using a trajectory corresponding to (1) in closed-loop with $\pi^0(\cdot)$.

Proof: Define the vectors

$$\mathbf{x}^1 = \mathcal{SS}^1_{[1 \to M]} \subseteq \mathcal{SS}^J_{[1 \to M]}, \tag{16a}$$

$$\mathbf{u}^1 = \pi^0(\mathbf{x}^1) = \mathcal{SU}^1_{[1 \to M]} \subseteq \mathcal{SU}^J_{[1 \to M]}, \tag{16b}$$

to be the stored state and input trajectory associated with the implemented policy $\pi^0(\cdot)$. Since $\pi^0(\cdot)$ is also feasible for Task 2, when Algorithm 1 is applied, the entire task execution can be stored as a successful execution for Task 2 without adapting the policy. It follows that $\mathcal{SS}^1_{[1 \to M]} \subseteq \mathcal{SS}^0_{[l_1 \to l_M]}$ and $\mathcal{SU}^1_{[1 \to M]} \subseteq \mathcal{SU}^0_{[l_1 \to l_M]}$, and the returned sample safe sets for Task 2 are non-empty.

At the initial state x_0, the ILMPC policy (10) optimizes the chosen input so as to minimize the remaining cost-to-go. Consider an MPC planning horizon of $N = 1$ (though this extends directly for any $N \geq 1$). Trivially,

$$\min_u \; h(x_0, u) + V^j(x^j_p) \quad \leq \quad h(x_k, \pi^0(x_0)) + V^j(x^j_p)$$
$$\text{s.t. } u \in \mathcal{U}, \qquad\qquad \text{s.t. } f(x_0, \pi^0(x_0)) = x^j_p.$$
$$f(x_0, u) = x^j_p.$$

It follows that the cost incurred by a Task 2 execution with $\pi^{ILMPC}_{[l_1 \to l_M]}$ is no higher than an execution with $\pi^{ILMPC}_{\pi^0}$. ∎
The proof that the improved closed-loop iteration cost follows from the improved ILMPC cost (9) is not presented here. However, the result shown holds for the examples presented in Sec. IV- V.

C. Discussion

In the results presented in this paper, we implement the search for controllable points (Algorithm 1, Line 15) by solving a one-step controllability problem, $(Q^\star, u^\star) = \mathrm{Ctrb}(x, \mathbf{z}, \mathbf{q})$, where

$$u^\star, \lambda^\star = \underset{u, \lambda}{\arg\min} \; h(x, u) + \lambda^\top \mathbf{q} \tag{17}$$
$$\text{s.t. } f(x, u) = \lambda^\top \mathbf{z},$$
$$\sum \lambda_i = 1, \; \lambda_i \geq 0,$$
$$u \in \mathcal{U}_i,$$
$$Q^\star = \lambda^{\star\top} \mathbf{q}, \tag{18}$$

where $\mathbf{z}$ is a previously verified state trajectory through the next Task 2 subtask, and $\mathbf{q}$ the sampled cost vector associated with the trajectory. (17) aims to find an input $u \in \mathcal{U}_i$ that connects the sampled guard state x to a state in the convex hull of the trajectory [21, Sec 4.4.2]. If an input is found, the new cost-to-go (8) for the state x is taken to be the convex combination of the stored cost vector (18). Solving the controllability analysis to the convex hull is an additional method for reducing computational complexity of TDMPC and is exact only for linear systems with convex constraints.

The points of subtask transition should be defined as is most useful, given the two tasks. Subtask transition points simply indicate which segments of the stored trajectories are certain to remain feasible in $\mathcal{T}2$ using the stored policies -
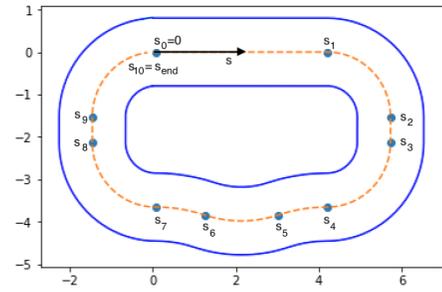

Fig. 2: Each subtask of the racing task corresponds to a segment of the track with constant curvature. The vehicle state s tracks the distance traveled along the centerline.

but this can change depending on how exactly $\mathcal{T}2$ differs from $\mathcal{T}1$. The TDMPC method is therefore not limited in applicability to a predetermined number of reshuffled tasks.

IV. APPLICATION 1: AUTONOMOUS RACING

A. Task Formulation

Consider an autonomous racing task, in which a vehicle is controlled to minimize lap time driving around a race track with piecewise constant curvature (Fig. 2). We model this task as a series of ten subtasks, where the i-th subtask corresponds to a section of the track with constant radius of curvature c_i. Tasks with different subtask order are tracks consisting of the same road segments in a different order.

The vehicle is modeled in the curvilinear abscissa reference frame [23], with states and inputs at time step k

$$x_k = [v_{x_k} \; v_{y_k} \; \dot{\psi}_k \; e_{\psi_k} \; s_k \; e_{y_k}]^\top,$$
$$u_k = [a_k \; \delta_k]^\top,$$

where v_{x_k}, v_{y_k}, and $\dot{\psi}_k$ are the vehicle's longitudinal velocity, lateral velocity, and yaw rate, respectively, at time step k, s_k is the distance travelled along the centerline of the road, and e_{ψ_k} and e_{y_k} are the heading angle and lateral distance error between the vehicle and the path. The inputs are longitudinal acceleration a_k and steering angle δ_k. The system dynamics (1) are described using an Euler discretized dynamic bicycle model [5]. Accordingly, the system state and input spaces are

$$\mathcal{X} = \mathbb{R}^6, \; \mathcal{U} = \mathbb{R}^2.$$

We formulate each subtask according to (11), with:
1) Subtask Workspace $\mathcal{X}_i$:

$$\mathcal{X}_i = \left\{ x : \begin{bmatrix} 0 \\ -\frac{\pi}{2} \text{ rad} \\ s_{i-1} \\ -\frac{l}{2} \end{bmatrix} \leq \begin{bmatrix} v_x \\ e_\psi \\ s \\ e_y \end{bmatrix} \leq \begin{bmatrix} 3 \text{ m/s} \\ \frac{\pi}{2} \text{ rad} \\ s_i \\ \frac{l}{2} \end{bmatrix} \right\},$$

where s_{i-1} and s_i mark the distances along the centerline to the start and end of the curve, and $l = 0.8$ is the lane width in meters. $s_{10} = s_{\text{end}}$ is the total length of the track. These bounds indicate that the vehicle can only drive forwards on the track, up to a maximum velocity, and must stay within the lane.

2) Subtask Input Space $\mathcal{U}_i$:

$$\mathcal{U}_i = \begin{bmatrix} -1 \text{ m/s}^2 \\ -0.5 \text{ rad/s}^2 \end{bmatrix} \leq \begin{bmatrix} a \\ \delta \end{bmatrix} \leq \begin{bmatrix} 1 \text{ m/s}^2 \\ 0.5 \text{ rad/s}^2 \end{bmatrix}.$$

The input limits are a function of the vehicle, and do not change between subtasks.

3) Subtask Transition Set, $\mathcal{R}_i$: Lastly, we define the subtask transition set to be the states along the subtask border where the track's radius of curvature changes:

$$\mathcal{R}_i = \{x \in \mathcal{X}_i : \exists u \in \mathcal{U}_i, \text{ s.t. } s^+ = s_{i+1}\},$$

where $x^+ = f(x, u)$. The task target set is the race track's finish line,

$$\mathcal{R}_M = \{x : s \geq s_{\text{end}}\}.$$

The task goal is to complete a lap and reach the target set as quickly as possible. Therefore we define the stage cost as:

$$h(x_k, u_k) = \begin{cases} 0, & x_k \in \mathcal{P} \\ 1, & \text{otherwise.} \end{cases}$$

B. Simulation Setup

An ILMPC (10) is used to complete $J = 5$ executions of Task 1, the track depicted in Fig. 2. The vehicle begins each task iteration at standstill on the centerline at the start of the track. The J executions and their costs are stored in $\mathcal{SS}_{[1 \to M]}$, $\mathcal{SU}_{[1 \to M]}$, and $\mathcal{SQ}_{[1 \to M]}$. An initial trajectory for the ILMPC safe sets is executed using a centerline-tracking, low-velocity PID controller, Π_0.

TDMPC then uses these sampled safe sets to design initial policies for a new track composed of the same track segments. Two ILMPCs are designed for the reconfigured track: one initialized with TDMPC, and another initialized with Π_0. Each ILMPC completes $J = 10$ laps around the new tracks. In this examples, the reconfigured track is not continuous, and should be considered to be a segments of larger, continuous track.

C. Simulation Results

Fig. 3 compares the first and tenth trajectories around the track of the two ILMPCs, plotted as black and red lines. The Π_0-initialized ILMPC (in dashed black) initially stays close to the centerline, taking nearly 18 seconds to traverse the new track. The TDMPC-initialized ILMPC, however, traverses the new track more efficiently starting with the first lap. The first lap completed using the TDMPC-initialized ILMPC (in solid black) begins closer to the final locally optimal policies (in red) that both ILMPCs eventually converge to. In this example, the TDMPC method is able to leverage experience on another track in order to complete sections of the new track in a locally optimal way, even on the first iteration of a new task.

The lap time of each of the ten ILMPC iterations is plotted in the bottom of Fig. 3. As expected, the TDMPC-initialized ILMPC completes the first several laps faster than the Π_0-initialized ILMPC. The TDMPC-initialized ILMPC requires fewer task iterations and less time per iteration to reach a locally optimal trajectory.
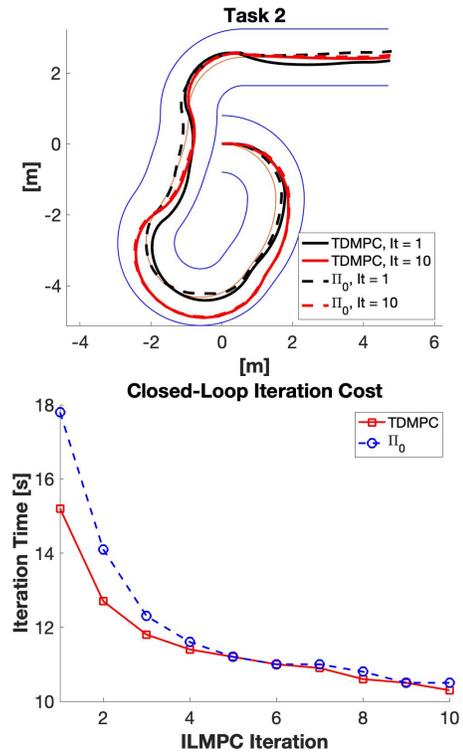

Fig. 3: The TDMPC-initialized ILMPC converges to a locally optimal trajectory faster than the PID-initialized one.
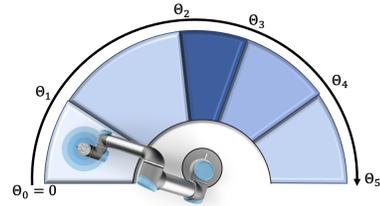

Fig. 4: Topview of the robotic path planning task. Each subtask corresponds to an obstacle in the environment with constant height.

V. APPLICATION 2: ROBOTIC PATH PLANNING

TDMPC can also be used to combine knowledge gained from solving a variety of previous tasks. For example, if D ILMPCs as in (10) complete J iterations of D different tasks, TDMPC can be used to design a policy for a task $(D+1)$. The algorithm draws on subtask executions collected over D different tasks in order to build safe sets for Task $D+1$. We evaluated this approach in a robotic path planning example.

A. Task Formulation

Consider a task in which a UR5e [1] robotic arm needs to move an object to a target without colliding with obstacles (Fig. 4). The obstacles are modeled as extruded disks of varying heights above and below the robot, leaving a workspace space between $h_{\text{min},i}$ and $h_{\text{max},i}$. Here, each subtask corresponds to the workspace above a particular

---

[1]https://www.universal-robots.com/products/ur5-robot/

obstacle. Different subtask orderings correspond to a rearranging of the obstacle locations.

The end-effector reference tracking accuracy of the UR5e allows us to use a simplified model in robot experiments, in place of a discretized second-order model as in [24]. We solve the task in the reduced state space:

$$x_k = [q_{0_k} \; \dot{q}_{0_k} \; z_k \; \dot{z}_k]^\top,$$
$$u_k = [\ddot{q}_{0_k} \; \ddot{z}_k]^\top,$$

where z_k is the height of the robot end-effector at time step k, calculated from the joint angles via forward kinematics, and $\dot{z}_k$ the upward velocity. We control $\ddot{q}_{0_k}$ and $\ddot{z}_k$, the accelerations of q_0 and z, respectively. The system state and input spaces are

$$\mathcal{X} = \mathbb{R}^4, \; \mathcal{U} = \mathbb{R}^2.$$

We model the simplified system as a quadruple integrator:

$$x_{k+1} = \begin{bmatrix} 1 & dt & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & dt \\ 0 & 0 & 0 & 1 \end{bmatrix} x_k + \begin{bmatrix} 0 & 0 \\ dt & 0 \\ 0 & 0 \\ 0 & dt \end{bmatrix} u_k, \quad (23)$$

where $dt = 0.01$ seconds is the sampling time. This simplified model holds as long as we operate within the region of high end-effector reference tracking accuracy, characterized in previous experiments.

We formulate each subtask according to (11).

1) Subtask Workspace $\mathcal{X}_i$:

$$\tilde{\mathcal{X}}_i = \begin{bmatrix} \theta_{i-1} \\ -\pi \text{ rad/s} \\ h_{\min,i} \\ -1 \text{ m/s} \end{bmatrix} \leq \begin{bmatrix} q_0 \\ \dot{q}_{0k} \\ z_k \\ \dot{z}_k \end{bmatrix} \leq \begin{bmatrix} \theta_i \\ \pi \text{ rad/s} \\ h_{\max,i} \\ 1 \text{ m/s} \end{bmatrix}$$

where θ_{i-1} and θ_i mark the cumulative angle to the beginning and end of the i-th obstacle, as in Fig. 4. States $\dot{q}_{0k}$ and $\dot{z}_k$ are constrained to lie in the experimentally determined region of high end-effector tracking accuracy. The robot end-effector is constrained not to collide with the subtask obstacle.

2) Subtask Input Space $\mathcal{U}_i$:

$$\tilde{\mathcal{U}}_i = \begin{bmatrix} -\pi \text{ rad/s}^2 \\ -0.6 \text{ m/s}^2 \end{bmatrix} \leq \begin{bmatrix} \ddot{q}_{0k} \\ \ddot{z}_k \end{bmatrix} \leq \begin{bmatrix} \pi \text{ rad/s}^2 \\ 0.6 \text{ m/s}^2 \end{bmatrix},$$

where $\ddot{q}_{0k}$ and $\ddot{z}_k$ are constrained to lie in the experimentally determined region of high end-effector tracking accuracy.

3) Subtask Transition Set, $\mathcal{R}_i$: We define the subtask transition set to be the states along the subtask border where the next obstacle begins:

$$\mathcal{R}_i = \{x \in \mathcal{X}_i : \exists u \in \mathcal{U}_i, \text{ s.t. } q_0^+ \geq \theta_i \},$$

where $x^+ = f(x, u)$. The task target set is the end of the last mode:

$$\mathcal{R}_M = \{x : q_0 = \theta_M, \; h_{\min,M} \leq z \leq h_{\max,M} \}.$$

The task goal is to reach the target set as quickly as possible:

$$h(x_k, u_k) = \begin{cases} 0, & x_k \in \mathcal{P} \\ 1, & \text{otherwise.} \end{cases}$$

B. Experimental Setup

An ILMPC (10) was used to complete $J = 10$ executions of five different training tasks, where each training task corresponded to a reordering of the obstacles. In each task, the ILMPC tries to reach the target set as quickly as possible while avoiding the obstacles. Each ILMPC was initialized with a trajectory resulting from executing a policy Π_0 that tracks the center height of each mode with the end-effector, while the robot rotated at a low constant joint velocity $\dot{q}_0$. TDMPC was then applied to the combined sampled safe sets of the five training tasks, and used to design an initial policy for a new ILMPC on an unseen ordering of obstacles, shown in Fig. 5. The white space corresponds to environment obstacles, so that the ILMPC task is to reach the end of the last mode as quickly as possible while controlling the end-effector to remain within the safe (green) part of the state space. A second ILMPC was initialized with the center-height tracking Π_0, for comparison. After initialization, the two ILCMPs completed $J = 20$ iterations of the new task. These iterations were executed in simulation using the simplified model (23), and the first and last trajectories of each ILMPC were then tracked by a real UR5e robot using end-effector tracking.

C. Experimental Results

The measured robot trajectories are plotted in Fig. 5. The Π_0-initialized ILMPC follows the center-height of each mode closely during the first task iteration (plotted in dashed black). After ten iterations of the task, the resulting trajectory (plotted in dashed red) has only diverged from the center-height trajectory slightly. Correspondingly, after ten iterations the Π_0-initialized ILMPC still requires more than four seconds to complete the task.

The TDMPC-initialized ILMPC, however, draws on knowledge gathered over many previous tasks in order to solve the task efficiently right away. Already on the first trajectory (plotted in solid black), the TDMPC-initialized ILMPC solves the task in under three seconds. This is a 30% improvement over the Π_0-initialized ILMPC. As in the autonomous driving task, the first trajectory completed by the TDMPC-initialized ILMPC is very close to the ultimate locally optimal trajectory.

Because of the nonconvex obstacles, this task is nonconvex, and there are many locally optimal trajectories. At various iterations of the task, both the TDMPC-initialized and the Π_0-initialized ILMPCs get stuck at such local minima, so that the ILMPCs performance metric remained constant over several iterations before improving again (Fig. 5). At these performance plateaus, the realized trajectories continue to change. We believe that the variability in the mixed integer solver used in the ILMPC led the ILMPC to follow different trajectories with the same iteration cost, as if encouraging
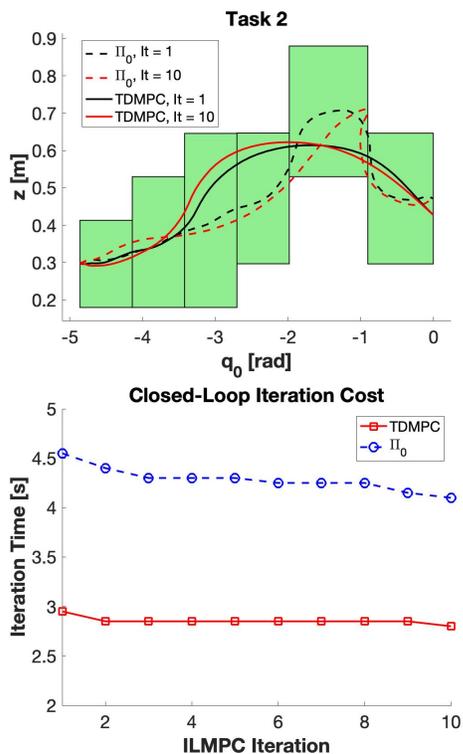
Fig. 5: The TDMPC-initialized ILMPC solves $\mathcal{T}2$ much faster than the ILMPC initialized with a center-height tracking policy Π_0.

exploration. Some of these different trajectories then allowed for performance improvement in the next iteration.

VI. CONCLUSION

A task decomposition method for ILMPC was presented. The TDMPC algorithm uses stored state and input trajectories from executions of a task, and efficiently designs policies for executing variations of that task. TDMPC breaks tasks into subtasks and performs controllability analysis at sampled safe states between subtasks. The algorithm can improve upon other methods by only needing to verify and adapt the original task policy at points of subtask transition, rather than along the entire trajectory. We evaluate the effectiveness of the proposed algorithm on autonomous racing and robotic manipulation tasks. Our results confirm that TDMPC allows an ILMPC to converge to an optimal minimum-time trajectory faster than finding an initial feasible trajectory using simple methods.

REFERENCES

[1] D. A. Bristow, M. Tharayil, and A. G. Alleyne, "A survey of iterative learning control," *IEEE Control Systems Magazine*, vol. 26, no. 3, pp. 96–114, 2006.

[2] Y. Wang, F. Gao, and F. J. Doyle III, "Survey on iterative learning control, repetitive control, and run-to-run control," *Journal of Process Control*, vol. 19, no. 10, pp. 1589–1600, 2009.

[3] K. Kritayakirana and J. C. Gerdes, "Using the centre of percussion to design a steering controller for an autonomous race car," *Vehicle System Dynamics*, vol. 50, no. sup1, pp. 33–51, 2012.

[4] J. V. Carrau, A. Liniger, X. Zhang, and J. Lygeros, "Efficient implementation of randomized mpc for miniature race cars," in *2016 European Control Conference (ECC)*. IEEE, 2016, pp. 957–962.

[5] U. Rosolia, A. Carvalho, and F. Borrelli, "Autonomous racing using learning model predictive control," in *2017 American Control Conference (ACC)*. IEEE, 2017, pp. 5115–5120.

[6] R. Horowitz, "Learning control of robot manipulators," *Journal of Dynamic Systems, Measurement, and Control*, vol. 115, no. 2B, pp. 402–411, 1993.

[7] S. Arimoto, M. Sekimoto, and S. Kawamura, "Task-space iterative learning for redundant robotic systems: Existence of a task-space control and convergence of learning," *SICE Journal of Control, Measurement, and System Integration*, vol. 1, no. 4, pp. 312–319, 2008.

[8] J. Van Den Berg, S. Miller, D. Duckworth, H. Hu, A. Wan, X.-Y. Fu, K. Goldberg, and P. Abbeel, "Superhuman performance of surgical tasks by robots using iterative learning from human-guided demonstrations," in *2010 IEEE International Conference on Robotics and Automation*. IEEE, 2010, pp. 2074–2081.

[9] Y. C. Wang, C. J. Chien, and C. N. Chuang, "Backstepping adaptive iterative learning control for robotic systems," in *Applied Mechanics and Materials*, vol. 284. Trans Tech Publ, 2013, pp. 1759–1763.

[10] J. H. Lee and K. S. Lee, "Iterative learning control applied to batch processes: An overview," *Control Engineering Practice*, vol. 15, no. 10, pp. 1306–1318, 2007.

[11] J. Kabzan, L. Hewing, A. Liniger, and M. N. Zeilinger, "Learning-based model predictive control for autonomous racing," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3363–3370, 2019.

[12] M. K. Cobb, K. Barton, H. Fathy, and C. Vermillion, "Iterative learning-based path optimization for repetitive path planning, with application to 3-d crosswind flight of airborne wind energy systems," *IEEE Transactions on Control Systems Technology*, pp. 1–13, 2019.

[13] D. Berenson, P. Abbeel, and K. Goldberg, "A robot path planning framework that learns from experience," in *2012 IEEE International Conference on Robotics and Automation*. IEEE, 2012, pp. 3671–3678.

[14] M. Stolle and C. Atkeson, "Finding and transferring policies using stored behaviors," *Autonomous Robots*, vol. 29, no. 2, pp. 169–200, 2010.

[15] C. Liu and C. G. Atkeson, "Standing balance control using a trajectory library," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Citeseer, 2009, pp. 3031–3036.

[16] Y. Tassa, T. Erez, and W. D. Smart, "Receding horizon differential dynamic programming," in *Advances in neural information processing systems*, 2008, pp. 1465–1472.

[17] C. G. Atkeson and J. Morimoto, "Nonparametric representation of policies and value functions: A trajectory-based approach," in *Advances in Neural Information Processing Systems*, 2003, pp. 1643–1650.

[18] K. Pereida, M. K. Helwa, and A. P. Schoellig, "Data-efficient multirobot, multitask transfer learning for trajectory tracking," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 1260–1267, 2018.

[19] U. Rosolia and F. Borrelli, "Learning model predictive control for iterative tasks. a data-driven control framework," *IEEE Transactions on Automatic Control*, vol. 63, no. 7, pp. 1883–1896, 2017.

[20] A. Coates, P. Abbeel, and A. Y. Ng, "Learning for control from multiple demonstrations," in *Proceedings of the 25th international conference on Machine learning*. ACM, 2008, pp. 144–151.

[21] F. Borrelli, A. Bemporad, and M. Morari, *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.

[22] U. Rosolia, X. Zhang, and F. Borrelli, "Simple policy evaluation for data-rich iterative tasks," in *2019 American Control Conference (ACC)*. IEEE, 2019, pp. 2855–2860.

[23] R. Rajamani, *Vehicle dynamics and control*. Springer Science & Business Media, 2011.

[24] M. Spong and M. Vidyasagar, *Robot Dynamics And Control*, 01 1989.