Task Decomposition for MPC: A Computationally Efficient Approach for Linear Time-Varying Systems ⋆

Charlott Vallon * **Francesco Borrelli** *

* *University of California at Berkeley, Berkeley, CA 94701, USA (e-mail: charlott, fborrelli@berkeley.edu).*

Abstract: A Task Decomposition method for iterative learning Model Predictive Control (TDMPC) for linear time-varying systems is presented. We consider the availability of state-input trajectories which solve an original task $\mathcal{T}1$, and design a feasible MPC policy for a new 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 the task decomposition problem, and provide a feasibility proof for the resulting policy. The proposed algorithm reduces the computational burden for linear time-varying systems with piecewise convex constraints. Simulation results demonstrate the improved efficiency of the proposed method on a robotic path-planning task.

Keywords: Data-based control, Iterative and Repetitive Learning control, Linear Model Predictive Control, Convex Optimization

1. INTRODUCTION

Classical Iterative Learning Controllers (ILCs) aim to improve a system's closed-loop reference tracking performance at each iteration of a repeated task (Bristow et al. (2006); Lee and Lee (2007)). Recent work has also explored reference-free ILC for applications whose goals are more appropriately defined via a performance metric, rather than a reference trajectory to track. Examples include autonomous racing tasks (e.g. "minimize lap time") (Rosolia et al. (2017); Kabzan et al. (2019)), or optimizing flight paths for tethered energy-harvesting systems (e.g. "maximize average power generation") (Cobb et al. (2019)).

In both classical and reference-free ILC, the controller uses data from previous iterations to improve future closed-loop performance with respect to the appropriate performance metric. At the very first iteration, these methods require either a reference trajectory to track or a feasible trajectory with which to initialize the control algorithm. If the task changes, a new trajectory must be designed, which can be difficult for complex tasks.

A variety of model-based methods have been suggested for finding feasible trajectories or policies for new tasks using stored data from related tasks. One approach relies on building and adapting trajectory libraries. For example, Nguyen et al. (2016) design a walking gait across stepping stones for a bipedal robot by linearly interpolating trajectories from a library of asymptotically stable periodic walking gaits. The authors of Yang et al. (2019) consider a set of actions and corresponding motion primitives for iterative teleoperative tasks. Given a new user-provided input, probabilistic inferences are made over the respective set of locally feasible trajectories. Similar approaches considering probabilistic distributions over trajectory libraries are proposed for robotic manipulators and autonomous vehicles in Paraschos et al. (2013) and Zhi et al. (2019), respectively. In Stolle and Atkeson (2010), 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 according to the features of the new task environment. The authors in Berenson et al. (2012) propose running a desired path planning method in parallel with a retrieve and repair algorithm that adapts a reference trajectory from a previous task to the constraints of a new task. While these methods decrease planning time, they require verifying or interpolating saved trajectories at each new time step, and cannot a priori guarantee constraint satisfaction.

Other approaches learn generalizable policies from stored task data. The authors of Dai and Sznaier (2018) consider linear hybrid systems. Data is collected in individual modes, and a polynomial optimization problem is formulated to find a stabilizing controller for arbitrary switching sequences. In Pereida et al. (2018), a fixed map is learned between a given reference trajectory and the input sequence required to track the trajectory with a linear system. This defines a policy given a new reference trajectory, but does not provide the trajectory itself. The authors of Fitzgerald et al. (2019) learn a mapping between a robot gripper pose performing the same task with different tools, based on online human corrections. This method was effective in demonstrations, but required human supervision and cannot guarantee safety.

In this paper, our objective is to efficiently find a feasible trajectory to smartly initialize an Iterative Learning

⋆ This research was sustained in part by fellowship support from the National Physical Science Consortium and the National Institute of Standards and Technology.

Model Predictive Controller (ILMPC) (Rosolia and Borrelli (2017)) for a new task. ILMPC is a reference-free ILC that uses a *safe set* to design an MPC policy for an iterative control task. This safe set is initialized using a feasible task trajectory, and collects states from which the task can be completed. In Vallon and Borrelli (2019), a Task Decomposition for ILMPC (TDMPC) algorithm was introduced for nonlinear, constrained dynamical systems. TDMPC is data-efficient, requires no human supervision, and, if the algorithm converges, produces trajectories that are guaranteed to satisfy all constraints for the new task. TDMPC decomposes an initial task $\mathcal{T}1$ into different modes of operation, called *subtasks*, and adapts stored $\mathcal{T}1$ trajectories to a new task $\mathcal{T}2$ only at points of subtask transition, by solving one-step controllability problems. The main contributions of this paper are as follows:

1. We present an extension to the TDMPC algorithm in Vallon and Borrelli (2019). We introduce a new formulation for linear time-varying systems with piecewise convex state and input constraints. The new formulation further reduces the computational burden of finding feasible trajectories for a new task $\mathcal{T}2$ by formulating the goal as a convex optimization problem, and simultaneously increases the size of the resulting $\mathcal{T}2$ safe set.
2. We prove that the induced safe set based MPC policy is feasible for $\mathcal{T}2$. This policy can be used to initialize an iterative learning control algorithm, or to directly obtain a suboptimal execution of $\mathcal{T}2$.

2. PROBLEM DEFINITION

2.1 Tasks and Subtasks

Consider a discrete-time system with linear, time-varying dynamics

$$x_{k+1} = A_k x_k + B_k u_k, \tag{1}$$

subject to state and input constraints

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

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

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

In this work we consider tasks $\mathcal{T}$ that can be decomposed into an ordered sequence of subtasks with piecewise linear dynamics and convex constraint sets. Specifically, the i-th subtask $\mathcal{S}_i$ is the tuple

$$\mathcal{S}_i = \{A_i, B_i, \mathcal{X}_i, \mathcal{U}_i, \mathcal{R}_i\}. \tag{3}$$

Within $\mathcal{S}_i$, the system is subject to linear dynamics

$$x_{k+1} = A_i x_k + B_i u_k, \tag{4}$$

and convex state and input constraints

$$x_k \in \mathcal{X}_i,\ u_k \in \mathcal{U}_i, \tag{5}$$

where $\mathcal{X}_i \subseteq \mathcal{X}$ and $\mathcal{U}_i \subseteq \mathcal{U}$ are convex sets. $\mathcal{R}_i$ is the set of transition states from $\mathcal{S}_i$ into the next subtask $\mathcal{S}_{i+1}$:

$$\mathcal{R}_i \subseteq \mathcal{X}_i = \{x \in \mathcal{X}_i : \exists u \in \mathcal{U}_i,\ A_i x + B_i u \in \mathcal{X}_{i+1}\}. \tag{6}$$

A successful *subtask execution E($\mathcal{S}_i$)* of a subtask $\mathcal{S}_i$ is a trajectory of states and inputs evolving according to (4), respecting state and input constraints (5), and ending in the subtask transition set (6). We define the j-th successful execution of subtask $\mathcal{S}_i$ as

$$E^j(\mathcal{S}_i) = [\mathbf{x}_i^j, \mathbf{u}_i^j], \tag{7a}$$

$$\mathbf{x}_i^j = [x_0^j, x_1^j,\ ..., x_{T_i^j}^j],\ x_k \in \mathcal{X}_i\ \ \forall k \in [0, T_i^j],$$

$$x_{T_i^j}^j \in \mathcal{R}_i, \tag{7b}$$

$$\mathbf{u}_i^j = [u_0^j, u_1^j,\ ..., u_{T_i^j}^j],\ u_k \in \mathcal{U}_i\ \ \forall k \in [0, T_i^j], \tag{7c}$$

where 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. Fig. 1 depicts three feasible subtask executions from a robotic path-planning example detailed in Sec. 4. Again, the final state of each successful subtask execution is in the subtask transition set $\mathcal{R}_i$, from which it can evolve into the subsequent subtask. In order to keep notation simple, we have written all subtask executions as beginning at time step $k = 0$.

Let task $\mathcal{T}$ be an ordered sequence of M subtasks, $\mathcal{T} = \{\mathcal{S}_i\}_{i=1}^M$. The j-th successful task execution is the concatenation of the corresponding subtask executions:

$$E^j(\mathcal{T}) = [E^j(\mathcal{S}_1), E^j(\mathcal{S}_2), ..., E^j(\mathcal{S}_M)] = [\mathbf{x}^j, \mathbf{u}^j], \tag{8}$$

$$\mathbf{x}^j = [\mathbf{x}_1^j, \mathbf{x}_2^j,\ ..., \mathbf{x}_M^j],$$

$$\mathbf{u}^j = [\mathbf{u}_1^j, \mathbf{u}_2^j,\ ..., \mathbf{u}_M^j],$$

$$x_\alpha^j \in \mathcal{R}_i, \qquad i \in [1, M-1],$$

$$A_i x_\alpha^j + B_i u_\alpha^j \in \mathcal{X}_{i+1}, \quad i \in [1, M-1],$$

where $\alpha = T_{[1\to i]}^j$ 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, $\mathcal{P} = \mathcal{R}_M$, the task's target set.

Definition: A set $\mathcal{P} \subset \mathcal{X}$ is a *control invariant set* for the system (1) subject to the constraints (2) if:

$$x_k \in \mathcal{P} \implies \exists u_k \in \mathcal{U}\ :\ A_k x_k + B_k u_k \in \mathcal{P},\ \forall k \geq 0. \tag{9}$$

Assumption 1: $\mathcal{P}$ is a control invariant set (9).

The optimal task $\mathcal{T}$ completion problem is given by:

$$V_{0\to T}^\star(x_0) = \min_{T, u_0, ..., u_{T-1}} \sum_{k=0}^{T} h(x_k, u_k) \tag{10}$$

$$\text{s.t.} \quad x_{k+1} = f(x_k, u_k),$$

$$x_k \in \mathcal{X}, u_k \in \mathcal{U}\ \ \forall k \geq 0,$$

$$x_T \in \mathcal{P},$$

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.

2.2 Safe Set Based ILMPC

In Rosolia and Borrelli (2017), a data-driven formulation for approximating the optimal control task (10) for a linear time-invariant system with convex constraints (2) is introduced. Here we propose a formulation for linear *time-varying* systems with *piecewise* convex constraints.

Each execution of task $\mathcal{T}$ is referred to as an iteration. After J number of task iterations, we define the time-indexed *sampled subtask safe set* of subtask $\mathcal{S}_i$ as:

$$\mathcal{KS}_{i,k} = \left\{ \bigcup_{j=1}^{J} x^j_{T_i^j - k} \right\}, \; k \in [0, \max_j T_i^j - 1]. \tag{11}$$

For given k and i, $x^j_{T_i^j-k}$ is the k-to-last state visited in $\mathcal{S}_i$ during the j-th task iteration. Thus the set (11) is the collection of states from which the system reaches the subtask transition set $\mathcal{R}_i$ in exactly k steps during a previously recorded task iteration, while satisfying subtask constraints (5). We similarly define a time-indexed *sampled subtask input set* of subtask $\mathcal{S}_i$ as:

$$\mathcal{KU}_{i,k} = \left\{ \bigcup_{j=1}^{J} u^j_{T_i^j - k} \right\}, \; k \in [0, \max_j T_i^j - 1].$$

Because $\mathcal{X}_i$ and $\mathcal{U}_i$ are convex, there also exists a feasible k-step input sequence to $\mathcal{R}_i$ for each convex combination of elements in $\mathcal{KS}^J_{i,k}$. We define *convex subtask safe sets* and *convex subtask input sets* as:

$$\mathcal{CK}_{i,k} = \left\{ \sum_{p=1}^{|\mathcal{KS}_{i,k}|} \lambda_p z_p : \lambda_p \geq 0, \sum_{p=1}^{|\mathcal{KS}_{i,k}|} \lambda_p = 1, z_p \in \mathcal{KS}_{i,k} \right\},$$
$$\mathcal{CU}_{i,k} = \left\{ \sum_{p=1}^{|\mathcal{KU}_{i,k}|} \lambda_p w_p : \lambda_p \geq 0, \sum_{p=1}^{|\mathcal{KU}_{i,k}|} \lambda_p = 1, w_p \in \mathcal{KU}_{i,k} \right\}, \tag{12}$$

where $|\mathcal{KS}_{i,k}|$ is the cardinality of $\mathcal{KS}_{i,k}$. Fig. 1 depicts these sets for three trajectories ($J = 3$) through a subtask from a robotic path-planning detailed in Sec. 4. We define a barycentric cost-to-go over the convex subtask safe sets:

$$\begin{aligned} v(x) = \min_{\lambda_p \geq 0,\ I,\ K} \quad & \sum_{p=0}^{|\mathcal{KS}_{I,K}|} \lambda_p V(z_p) \\ \text{s.t.} \quad & \sum_{p=0}^{|\mathcal{KS}_{I,K}|} \lambda_p = 1, \\ & \sum_{p=0}^{|\mathcal{KS}_{I,K}|} \lambda_p z_p = x, \quad z_p \in \mathcal{KS}_{I,K}, \end{aligned} \tag{13}$$

where $V(z_p)$ is the realized cost-to-go from state z_p during a past execution. At time step k of iteration $J+1$, we approximate the optimal control problem (10) by solving:

$$\begin{aligned} V(x_k^{J+1}) = & \\ \min_{u_{k|k},\ldots,u_{k+N-1|k},I,K} \quad & \sum_{t=k}^{k+N-1} h(x_{t|k}, u_{t|k}) + v(x_{k+N|k}) \\ \text{s.t.} \quad & x_{t+1|k} = f(x_{t|k}, u_{t|k}, t), \forall t \in [k, k+N-1], \\ & x_{t|k} \in \mathcal{X}, \; u_{t|k} \in \mathcal{U}, \quad \forall t \in [k, k+N-1], \\ & x_{k|k} = x_k^j, \\ & x_{k+N|k} \in \mathcal{CK}_{I,K} \cup \mathcal{P}, \end{aligned} \tag{14}$$

which searches for an input sequence over a horizon N that controls the system (1) to the state in a convex subtask safe state set or task target set $\mathcal{P}$ with the lowest cost-to-go (13). We use a receding horizon strategy:

$$u(x_k^j) = \pi^{\text{ILMPC}}(x_k^j) = u^\star_{k|k}. \tag{15}$$

At the first iteration of a new task, the ILMPC (14) requires non-empty sets $\mathcal{CK}_{\cdot,\cdot}$ containing at least one feasible

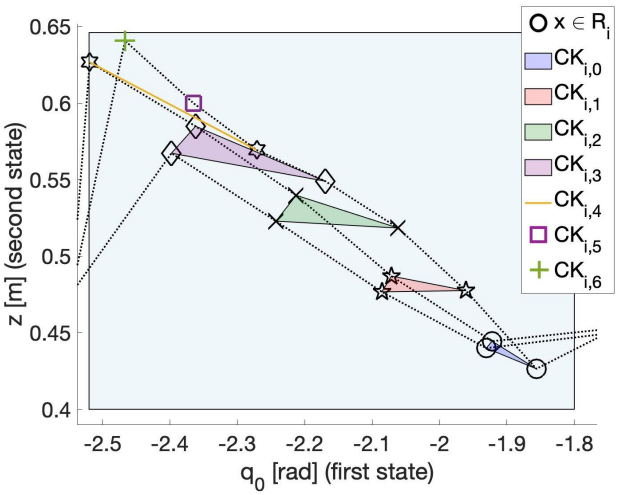

Fig. 1. Convex subtask safe sets contain states from which the transition set can be reached in a certain number of steps.

execution of the task. Next we present a computationally efficient approach for creating such sets using data from executions of a different, previous task. This new computationally efficient formulation of TDMPC for tasks with piecewise linear, convex modes is the main contribution of this work compared to Vallon and Borrelli (2019). We will require the following definition of controllability.

Definition: For a given target set $\mathcal{R} \subset \mathcal{X}$, the *N-step controllable set* $\mathcal{K}_N(\mathcal{R})$ of a system (1) subject to constraints (2) is defined recursively as:

$$\begin{aligned} \mathcal{K}_j(\mathcal{R}) = \text{Pre}(\mathcal{K}_{j-1}(\mathcal{R})) \cap \mathcal{X}, \; \mathcal{K}_0(\mathcal{R}) = \mathcal{R}, \\ j \in \{1, ..., N\}, \\ \text{Pre}(\mathcal{R}) = \{x : \exists u \in \mathcal{U} : A_k x + B_k u \in \mathcal{R}\}. \end{aligned}$$

For all states in the N-step controllable set to $\mathcal{R}$ there exists a feasible input sequence such that the system will be driven into $\mathcal{R}$ in N steps.

Proposition 1. In general, not all states belonging to the convex hull of stored subtask executions are controllable to the subtask transition set (6).

Proof. Proof in Appendix.

Proposition 2. All states in the time-indexed convex subtask safe sets (12) are controllable to the subtask transition set (6).

Proof. Proof follows from Thm. 1 in Sec. 3.2.

Propositions 1 and 2 motivate the approach proposed in this paper of storing task executions in time-indexed convex subtask safe sets (12).

3. TASK DECOMPOSITION FOR ILMPC

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

$$\mathcal{T}1 = \{\mathcal{S}_i\}_{i=1}^M, \quad \mathcal{T}2 = \{\mathcal{S}_{l_i}\}_{i=1}^M, \tag{16}$$

where the sequence $[l_1, l_2, ..., l_M]$ is a reordering of the sequence $[1, 2, ..., M]$. Assume non-empty subtask safe sets $\mathcal{KS}_{[1 \to M]}$ (11) containing task data from Task 1.

Our goal is to use stored subtask safe sets (11) from Task 1 in order to find convex subtask safe sets (12) for Task 2. These sets can then be used to initalize a controller for the new task. The key intuition of TDMPC is that all successful subtask executions (7a) from Task 1 are also successful subtask executions for Task 2, as this definition only depends on properties (7b-7c) of each individual subtask, not the subtask sequence. With this notion, Alg. 1 proceeds backwards through the new subtask sequence.

3.1 TDMPC Algorithm

The notation $(l_M, \cdot)$ or $(i, \cdot)$ indicates that the described action is undertaken for all appropriate second arguments.

Algorithm 1 TDMPC algorithm

input $\mathcal{KS}_{[1\to M]}$, $\mathcal{KU}_{[1\to M]}$, $[l_1, l_2, ..., l_M]$
do $\mathcal{CK}_{[1\to M]} = \text{convexify}(\mathcal{KS}_{[1\to M]})$ (12)
do $\mathcal{SG}_i = \text{guard set clustering}(\mathcal{KS}_{[1\to M]})$ (17)
initialize empty $\hat{\mathcal{KS}}$, $\hat{\mathcal{KU}}$, $\hat{\mathcal{CK}}$, $\hat{\mathcal{CU}}$
$\hat{\mathcal{CK}}_{l_M,\cdot} \leftarrow \mathcal{CK}_{l_M,\cdot}$, $\hat{\mathcal{CU}}_{l_M,\cdot} \leftarrow \mathcal{CU}_{l_M,\cdot}$
for $i \in [l_{M-1} : -1 : l_1]$ **do**
  $\hat{\mathcal{KS}}_{i,\cdot} \leftarrow \mathcal{KS}_{i,\cdot}$
  $\hat{\mathcal{KU}}_{i,\cdot} \leftarrow \mathcal{KU}_{i,\cdot}$
  initialize empty $\hat{\mathcal{KU}}_{i,0}$
  for $x \in \mathcal{SG}_i$ **do**
    check $(q^\star, u^\star) = \text{Ctrb}(x, \hat{\mathcal{CK}}_{i+1,\cdot})$ (18)
    if infeasible then
      $\hat{\mathcal{KS}}_{i,\cdot} \leftarrow \hat{\mathcal{KS}}_{i,\cdot} \backslash \text{trajectory}(x)$
      $\hat{\mathcal{KU}}_{i,\cdot} \leftarrow \hat{\mathcal{KU}}_{i,\cdot} \backslash \text{trajectory}(u^\star)$
    else
      $\hat{\mathcal{KU}}_{i,0} \leftarrow u^\star$
  $\hat{\mathcal{CK}}_{i,\cdot} \leftarrow \text{convexify}(\hat{\mathcal{KS}}_{i,\cdot})$ (12)
  $\hat{\mathcal{CU}}_{i,\cdot} \leftarrow \text{convexify}(\hat{\mathcal{KU}}_{i,\cdot})$ (12)
Return $\mathcal{CK}_{[l_1\to l_M]} \leftarrow \hat{\mathcal{CK}}$, $\mathcal{CU}_{[l_1\to l_M]} \leftarrow \hat{\mathcal{CU}}$

- Consider the last subtask, $\mathcal{S}_{l_M}$. By definition, for any state in $\mathcal{CK}_{l_M,\cdot}$ there exists a stored input sequence in $\mathcal{CU}_{l_M,\cdot}$ that can be applied such that the system evolves into $\mathcal{R}_{l_M}$. (Algorithm 1, Lines 2-5).
- Prune the convex safe sets of the preceding subtask $\mathcal{S}_{l_{M-1}}$ to contain only states which can also be controlled to $\mathcal{R}_{l_M}$. We verify this property only for states in the sampled guard set of $\mathcal{S}_{l_{M-1}}$, defined as:

$$\mathcal{SG}_{l_{M-1}} = \mathcal{KS}_{l_{M-1},0}. \tag{17}$$

  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 a past execution of $\mathcal{T}1$ (Algorithm 1, Line 10).
- Determine which points in $\mathcal{SG}_{l_{M-1}}$ are one-step controllable to $\mathcal{CK}_{l_M,k}$ for some time index k. This problem can be solved using a variety of numerical approaches. In the results presented in this paper, we check controllability for each k, and choose the input u to minimize the cost of the resulting state according to (13). Specifically, for each point $x \in \mathcal{SG}_{l_{M-1}}$ and index k, we solve $(q^\star, u^\star) = \text{Ctrb}(x, \hat{\mathcal{CK}}_{l_M,k})$, where:

$$\begin{aligned} u^\star = \arg\min_u\ & v(z) && (18)\\ \text{s.t. } & z = A_{l_{M-1}}x + B_{l_{M-1}}u \\ & u \in \mathcal{U}_{l_{M-1}} \\ & z \in \hat{\mathcal{CK}}_{l_M,k}, \\ q^\star = v(A_{l_{M-1}}&x + B_{l_{M-1}}u^\star). && (19) \end{aligned}$$

  If (18) is feasible, the previously stored $\mathcal{T}1$ cost-to-go (13) from the state x is replaced by $q^\star$, the cost to reach the goal set of $\mathcal{T}2$. (Algorithm 1, Line 11)
- For all states x in $\mathcal{SG}_{l_{M-1}}$ not controllable to any convex safe set in $\mathcal{S}_{l_M}$, we remove the stored subtask execution ending in x out of the set of subtask safe sets for $\mathcal{S}_{l_M}$. (Algorithm 1, Lines 12-16)
- After checking the entire sampled guard set, all remaining convex subtask safe sets for $\mathcal{S}_{l_{M-1}}$ are controllable to convex subtask safe sets in $\mathcal{S}_{l_M}$, and therefore also to $\mathcal{R}_{l_M}$. (Algorithm 1, Lines 17-19)

Alg. 1 iterates backwards through the remaining subtasks, verifying the controllability of points in sampled guard sets to a convex subtask safe set in the following subtask. The algorithm returns convex subtask safe sets for Task 2 that can be used to initialize an ILMPC (14 - 15) for Task 2.

TDMPC offers a computationally efficient, data-driven method of initializing an ILMPC for new tasks. In contrast to multi-step or set-based methods that are common for model-based or hybrid systems, Alg. 1 only solves controllability problems from discrete points in the sampled guard set to already verified feasible sets. Furthermore, TDMPC directly provides a robust policy for solving the task associated with the verified trajectories.

Note the reformulation of the stored Task 1 executions (8) into convex sets (12). We can thus replace the point-to-point controllability verification from Vallon and Borrelli (2019) with point-to-set controllability in Alg. 1. This allows for three major improvements to the procedure:

1. (18) is a convex optimization problem, which is, in general, much faster to solve than the non-convex point-to-point controllability.
2. By using the convex hull of stored states (12) as a target set in (18), rather than individual states, more points in the sampled guard sets (17) can potentially be demonstrated to lead to feasible Task 2 executions.
3. We increase the number of points for which we know a feasible Task 2 policy, since we implicitly consider all points in the time-indexed convex hulls of Task 1 trajectories (12), rather than only the Task 1 trajectories.

3.2 Feasibility

We prove the feasibility of ILMPC policies (15) initialized using Alg. 1.

Assumption 2: Task 1 and Task 2 are defined as in (16), with each subtask defined by linear dynamics (4) and convex constraints (5).

Theorem 1. Let Assumptions 1-2 hold. Assume Alg. 1 outputs non-empty sets $\mathcal{CK}^0_{[l_1\to l_M]}$ for Task 2. Then, if $x_0 \in \mathcal{CK}^0_{[l_1\to l_M]}$, the policy $\pi^{\text{ILMPC}}_{[l_1\to l_M]}$, as defined in (15), produces a feasible execution of Task 2.

Proof. We will use induction to prove the feasibility. First, we show that the ILMPC (14-15) is feasible at time step $k = 0$ of the j-th execution of Task 2. By assumption $\mathcal{CK}^0_{[l_1 \to l_M]}$ is not empty. From (12) we have that $\mathcal{CK}^0_{[l_1 \to l_M]} \subseteq \mathcal{CK}^{j-1}_{[l_1 \to l_M]}$ $\forall j \geq 1$, and consequently $\mathcal{CK}^{j-1}_{[l_1 \to l_M]}$ is not empty. Therefore, $x_0 \in \mathcal{CK}^0_{[l_1 \to l_M]} \subseteq \mathcal{CK}^{j-1}_{[l_1 \to l_M]}$, and there exist $I^\star$, $K^\star$, and multipliers $\lambda^\star_p$ such that

$$x_0 = \sum_{p=1}^{|\mathcal{CK}^{j-1}_{I^\star,K^\star}|} \lambda^\star_p x_p,\ x_p \in \mathcal{CK}^{j-1}_{I^\star,K^\star}.$$

We define

$$\bar{u} = \lambda^\star_p u_p \in \mathcal{U}_{I^\star} \tag{20}$$

where u_p is the input associated with the state $x_p \in \mathcal{CK}^{j-1}_{I^\star,K^\star}$ in a previous task execution of Task 1. Now, note that we have

$$\begin{aligned} \bar{x} = A_{I^\star} + B_{I^\star}\bar{u} &= \sum_{p=1}^{|\mathcal{CK}^{j-1}_{I^\star,K^\star-1}|} \lambda^\star_p x_p, \\ x_p &\in \begin{cases} \mathcal{CK}^{j-1}_{I^\star,K^\star-1}, & K^\star \geq 1; \\ \mathcal{CK}^{j-1}_{I^\star+1,K^{\star\star}} & K^\star = 0, \end{cases} \end{aligned} \tag{21}$$

for some $K^{\star\star}$. The second case ($K^\star = 0$) follows directly from Alg. 1. This procedure (20-21) can be repeated N times in order to find a feasible input sequence for the initial state x_0 satisfying (14). Therefore there exists a feasible solution to the ILMPC (14 - 15) at time step $k = 0$ of the j-th execution of Task 2.

Next, we show that the policy is recursively feasible. Assume that at time step k of the j-th iteration the ILMPC (14 - 15) is feasible, and let $\mathbf{x}^{\star,j}_{k:k+N|k}$ and $\mathbf{u}^{\star,j}_{k:k+N|k}$ be the optimal trajectory and input sequence according to (14). From (15), the realized state and input at time k of the j-th iteration are given by

$$x^j_k = x^{\star,j}_{k|k},\ u^j_k = u^{\star,j}_{k|k},$$

and the terminal constraint in (14) enforces $x^{\star,j}_{k+N|k} \in \mathcal{CK}^{j-1}_{I',K'}$ for some I', K', where $\mathcal{CK}^{j-1}_{I',K'}$ contains states from the previous $j-1$ trajectories. As in (20-21), define an input and corresponding state

$$\begin{aligned} u' &= \lambda^\star_p u_p \in \mathcal{U}_{I'} \\ x' &= A_{I'} x^{\star,j}_{k+N|k} + B_{I'} u' \in \mathcal{CK}^{j-1}_{I',K'-1}. \end{aligned}$$

We therefore have

$$x^j_{k+1} = x^{\star,j}_{k+1|k}.$$

It follows that at time step $k+1$ of the j-th Task 2 execution, the input sequence and related feasible state trajectory

$$\begin{aligned} &[u^{\star,j}_{k+1|k},\ u^{\star,j}_{k+2|k},\ ...,u^{\star,j}_{k+N-1|k}, u'] \\ &[x^{\star,j}_{k+1|k},\ x^{\star,j}_{k+2|k},\ ...,x^{\star,j}_{k+N-1|k}, x'] \end{aligned} \tag{22}$$

satisfy input and state constraints in (14). Therefore, (22) is a feasible solution for (14) at time step $k+1$.

We have shown that at the j-th iteration of Task 2, $\forall j \geq 1$, the ILMPC is feasible at time step $k = 0$ and that if the ILMPC is feasible at time step k, it must also be feasible at time step $k+1$. We can conclude by induction that (14 - 15) is feasible $\forall j \geq 1$ and $k \in \mathbb{Z}_{0+}$ when initialized with sets output by Alg. 1. ■

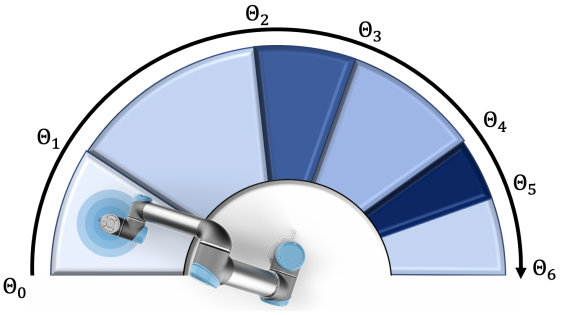

Fig. 2. Topview of the path planning task. Each subtask corresponds to a pair of upper and lower obstacles.

It follows from the same arguments (20-21) that the ILMPC policy (14-15) will eventually bring any $x_0 \in \mathcal{CK}^J$ to the task target set $\mathcal{R}_{l_M}$.

4. RESULTS: ROBOT PATH PLANNING

4.1 Task Formulation

We demonstrate the effectiveness of Alg. 1 in a robotic path planning example. Consider a UR5e [1] robotic arm tasked with maneuvering through six sets of obstacles modeled as extruded disks of varying heights above and below the robot. Each set of upper and lower obstacles leaves a workspace between the disks for the robot to move between. Here, each subtask $\mathcal{S}_i$ corresponds to the workspace between a pair of lower and upper obstacles. Different subtask orderings correspond to a rearranging of the obstacle locations, indicated by Θ_i in Fig. 2.

The UR5e has high end-effector reference tracking accuracy, allowing us to use a simplified end-effector model in place of a discretized second-order model as in Spong and Vidyasagar (1989). We solve the task in the reduced state space:

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

where q_{0_k} is the angle of the robot's base joint along the Θ direction and z_k is the height of the robot end-effector at time step k, calculated from the six joint angles via forward kinematics. $\dot{q}_{0_k}$ and $\dot{z}_k$ are the corresponding velocities. We control $\ddot{q}_{0_k}$ and $\ddot{z}_k$, the accelerations of q_0 and z, respectively. This reduced state space allows us to formulate the task as a concatenation of $M = 6$ subtasks with piecewise affine dynamics and convex constraints, according to (3).

Subtask Dynamics A_i, B_i We model the base-and-end-effector system as a quadruple integrator:

$$x_{k+1} = A_i x_k + B_i u_k \tag{24a}$$

$$A_i = \begin{bmatrix} 1 & dt & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & dt \\ 0 & 0 & 0 & 1 \end{bmatrix}, B_i = \begin{bmatrix} 0 & 0 \\ dt & 0 \\ 0 & 0 \\ 0 & dt \end{bmatrix},\ \forall i \in [1,6] \tag{24b}$$

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

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.

Subtask Constraints $\mathcal{X}_i$

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

where Θ_{i-1} and Θ_i mark the cumulative angle to the beginning and end of the i-th obstacle, as in Fig. 2. The robot end-effector is constrained to remain in the space between the upper and lower obstacles, bounded by $o_{\min,i}$ and $o_{\max,i}$. The base's rotational velocity $\dot{q}_{0k}$ and $\dot{z}_k$ are constrained to lie in the experimentally determined region of high end-effector tracking accuracy. Specifically, we take

$$\dot{z}_{\max,i} = C_1 \sin\left(\arccos\left(\frac{o_{\min,i}}{d_1}\right)\right),$$
$$\dot{z}_{\min,i} = -\dot{z}_{\max,i},$$

where the constants C_1 and d_1 depend on setup parameters and joint limits provided by the manufacturer.

Subtask Input Space $\mathcal{U}_i$

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

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

$$\ddot{z}_{\max,i} = C_2 \sin\left(\arccos\left(\frac{o_{\min,i}}{d_2}\right) + \frac{o_{\min,i}}{d_3}\right),$$
$$\ddot{z}_{\min,i} = -\ddot{z}_{\max,i},$$

where C_2, d_2 and d_3 depend on setup parameters and joint limits provided by the manufacturer.

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 $x^+ = A_i x + B_i u$ (4). The task target set is the end of the last mode:

$$\mathcal{R}_6 = \{x : q_0 = \Theta_6,\ o_{\min,6} \leq z \leq o_{\max,6}\}.$$

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

$$h(x_k, u_k) = \begin{cases} 0, & x_k \in \mathcal{R}_6 \\ 1, & \text{otherwise.} \end{cases}$$

4.2 Experimental Results

We evaluate the efficiency of Alg. 1 by comparing its run-time with the the run-time of the point-to-point controllability analysis for task decomposition introduced in Vallon and Borrelli (2019) for nonlinear systems.

First, an ILMPC (14)-(15) is used to complete five executions of five different training tasks, where each training task is a different reordering of the six obstacles. Each ILMPC is initialized with a suboptimal state and input trajectory that tracks the center-height of each subtask

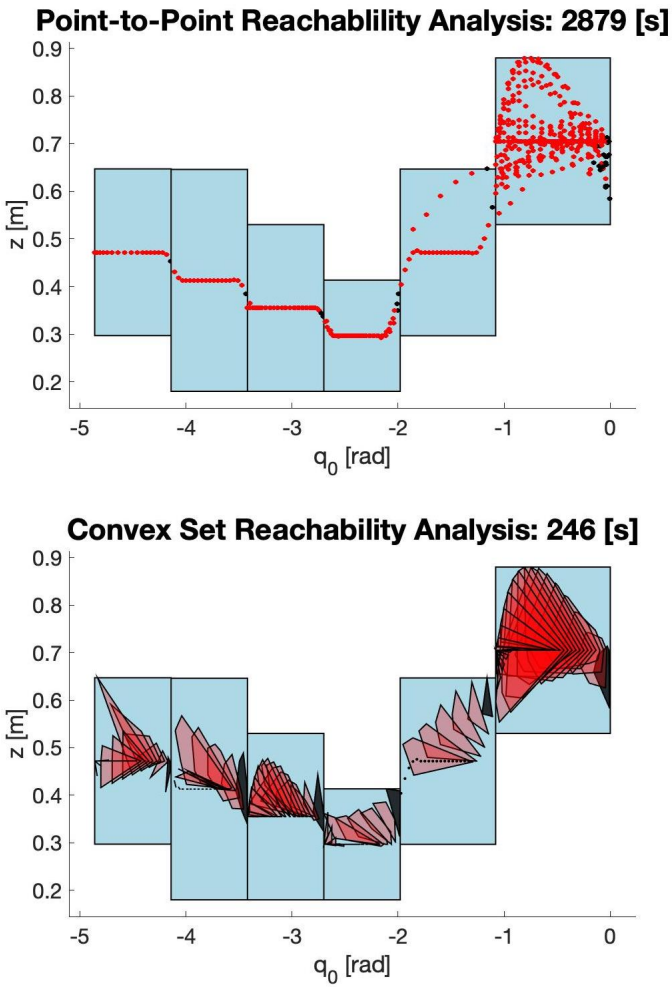

Fig. 3. Alg. 1 produces a significantly larger set of feasible states for $\mathcal{T}2$ in 10% of the time as the algorithm in Vallon and Borrelli (2019). The sampled guard sets for each subtask are plotted in black.

while the robot arm rotates at a low base velocity $\dot{q}_0$. In each task, the ILMPC tries to reach the target set as quickly as possible while avoiding the obstacles. The iterations are completed in simulation using the simplified model (24), and the corresponding subtask executions are saved as convex subtask safe sets (12). The new task $\mathcal{T}2$ is configured from another new reshuffling of the obstacles. Fig. 3 depicts the $\mathcal{T}2$ workspace in light blue, along with the $\mathcal{T}2$ safe sets returned by the two versions of the TDMPC algorithm. The top image plots in red the feasible safe states for $\mathcal{T}2$ output from the point-to-point controllability method from Vallon and Borrelli (2019). The bottom image shows in red the feasible safe sets output by the efficient point-to-convex-set controllability method from Alg. 1.

In the example shown, our improved method was an order of magnitude faster at finding safe states for $\mathcal{T}2$ than the point-to-point method, requiring 246 seconds of processing instead of 2879 seconds, using a 2017 Mac Book Pro with 2.8 GHz Quad-Core Intel Core i7. Indeed, the efficient reformulation of Alg. 1 for linear systems resulted in an average eleven fold speed-up for five different trials of the described setup and testing procedure. In Tab. 1, each trial corresponds to a newly shuffled $\mathcal{T}2$.

Table 1. Controllability Analysis Run-Time

Trial	Convex Set Analysis	Pointwise Analysis
1	246 s	2879 s
2	312 s	5561 s
3	219 s	1618 s
4	212 s	1810 s
5	264 s	2806 s

For each state in a subtask's sampled guard set, the point-to-point controllability method solves a mixed-integer program to try to find an input that controls the system to the last state of a subsequent subtask trajectory. Therefore the complexity of both controllability methods depend on the state dimension and number of trajectories through the subsequent subtask (as this provides an upper bound for the size of the subtask safe set). The efficiency improvement results from replacing the mixed-integer constraint in point-to-point controllability with a convex constraint of equal complexity, which is typically easier to compute.

Remark 2. The convex reachability safe sets in Fig. 3 demonstrate the implications of Prop. 1. Within each subtask, the time-indexed convex safe sets do not entirely contain the previous time step's convex safe set.

Remark 3. As is clear from Fig. 3, the convex set controllability analysis outputs a significantly larger set of feasible states for $\mathcal{T}2$ than the pointwise method. This results from two main phenomena. First, more states from the $\mathcal{T}1$ sampled guard sets remain in $\mathcal{T}2$ when using the convex set controllability than point-to-point controllability. This follows since the new controllability analysis (18) considers a larger one-step target set than the pointwise controllablity analysis (i.e. a convex hull of states in the next subtask rather than individual states). So more points in the sampled guard set can be shown to lead to feasible executions of $\mathcal{T}2$. Second, while the point-to-point controllability analysis only checks for $\mathcal{T}2$ feasibility of the actual subtask trajectories from $\mathcal{T}1$, the new convex set controllability analysis automatically also provides a policy for the convex subtask safe sets induced by the trajectories. All safe states found using the point-to-point controllability method are thus also found using the convex set controllability method. Accordingly, an ILMPC (14 - 15) initialized with safe sets returned from Alg. 1 will also lead to a faster first execution of $\mathcal{T}2$.

5. CONCLUSION

In this paper, an extension to the Task Decomposition for iterative learning Model Predictive Control (TDMPC) is presented. The algorithm uses stored state and input trajectories from executions of a task, and efficiently designs set-based policies for executing variations of that task. The algorithm is designed for linear time-varying systems with piecewise convex constraint sets, and is shown to significantly reduce the computational burden associated with the TDMPC algorithm. We prove that the resulting policies are guaranteed to be feasible for the new task. Finally, we evaluate the effectiveness of the proposed algorithm on a robotic path planning tasks, and demonstrate the reduced computational burden compared with TDMPC for nonlinear systems.

REFERENCES

Berenson, D., Abbeel, P., and Goldberg, K. (2012). A robot path planning framework that learns from experience. In *2012 IEEE International Conference on Robotics and Automation*, 3671–3678. IEEE.

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

Bristow, D.A., Tharayil, M., and Alleyne, A.G. (2006). A survey of iterative learning control. *IEEE Control Systems Magazine*, 26(3), 96–114.

Cobb, M.K., Barton, K., Fathy, H., and Vermillion, C. (2019). 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*, 1–13. doi: 10.1109/TCST.2019.2912345.

Dai, T. and Sznaier, M. (2018). A moments based approach to designing mimo data driven controllers for switched systems. In *2018 IEEE Conference on Decision and Control (CDC)*, 5652–5657.

Fitzgerald, T., Short, E., Goel, A., and Thomaz, A. (2019). Human-guided trajectory adaptation for tool transfer. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '19, 1350–1358.

Kabzan, J., Hewing, L., Liniger, A., and Zeilinger, M.N. (2019). Learning-based model predictive control for autonomous racing. *IEEE Robotics and Automation Letters*, 4(4), 3363–3370.

Lee, J.H. and Lee, K.S. (2007). Iterative learning control applied to batch processes: An overview. *Control Engineering Practice*, 15(10), 1306–1318.

Nguyen, Q., Da, X., Grizzle, J., and Sreenath, K. (2016). Dynamic walking on stepping stones with gait library and control barrier functions. *Arbor*, 1001, 48109.

Paraschos, A., Neumann, G., and Peters, J. (2013). A probabilistic approach to robot trajectory generation. In *2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, 477–483. IEEE.

Pereida, K., Helwa, M.K., and Schoellig, A.P. (2018). Data-efficient multirobot, multitask transfer learning for trajectory tracking. *IEEE Robotics and Automation Letters*, 3(2), 1260–1267.

Rosolia, U. and Borrelli, F. (2017). Learning model predictive control for iterative tasks: A computationally efficient approach for linear system. In *20th IFAC World Congress*. IFAC.

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

Spong, M. and Vidyasagar, M. (1989). *Robot Dynamics And Control*. Wiley.

Stolle, M. and Atkeson, C. (2010). Finding and transferring policies using stored behaviors. *Autonomous Robots*, 29(2), 169–200.

Vallon, C. and Borrelli, F. (2019). Task decomposition for iterative learning model predictive control. *[Submitted]*. URL `http://arxiv.org/abs/1903.07003`.

Yang, X., Agrawal, A., Sreenath, K., and Michael, N. (2019). Online adaptive teleoperation via motion primitives for mobile robots. *Autonomous Robots*, 43(6),

1357–1373.
Zhi, W., Lai, T., Ott, L., Francis, G., and Ramos, F. (2019). Octnet: Trajectory generation in new environments from past experiences.

Appendix A. PROOF OF PROPOSITION 1

We know that all states in a feasible subtask execution (7a) are controllable to a point in the subtask transition set (7b). A sufficient condition for all states in the convex hull of feasible subtask executions to also be controllable to the convex hull of corresponding points in the subtask transition set is for latter to be a control invariant set.

By definition of a feasible task execution (8) and the proof of Thm. 1, any point in the convex hull of stored states in the transition set is also controllable to the task's control invariant goal set. For linear time-invariant systems, states that are controllable to an invariant set are also invariant (Borrelli et al. (2017)). However, for linear time-varying systems these states are only stabilizable to the invariant. Therefore the convex hull of the points in the transition set is not inherently an invariant.

We recognize that the convex hull of states in the transition set *is* a control invariant set if its backward controllable sets grow to contain each other, i.e. if for all scalar values N, $\mathcal{K}_{N-1} \subseteq \mathcal{K}_N$. Unfortunately, this property does not generally hold for linear systems. Consider for example the double integrator system

$$x_{k+1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} x_k + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u_k, \tag{A.1}$$

subject to the state and input constraints

$$x_k \in \mathcal{X}, \quad u_k \in [-2, 2]. \tag{A.2}$$

Define a target set, $\mathcal{R}$, to be the convex hull of three points:

$$\mathcal{R} = \text{convhull}\left(\begin{bmatrix} 3 \\ 2 \end{bmatrix}, \begin{bmatrix} 2 \\ 2.5 \end{bmatrix}, \begin{bmatrix} 3 \\ 3 \end{bmatrix} \right). \tag{A.3}$$

The 1-step, 2-step, and 3-step controllable sets to $\mathcal{R}$ are plotted in Fig. A.1. We also plot $\mathcal{C}$, the convex hull of $\mathcal{R}$ and the controllable sets. It is clear from the plot that $\mathcal{R}$ is not an invariant set. This means it is possible that states in the convex hull of trajectories leading into $\mathcal{R}$ are not N-step controllable to $\mathcal{R}$ for any value of N. In Fig. A.1, this corresponds to states who are in $\mathcal{C}$, but not in any $\mathcal{K}_N(\mathcal{R})$.

Thus, we have shown that the convex hull of stored subtask executions is not generally controllable to the subtask transition set. ■

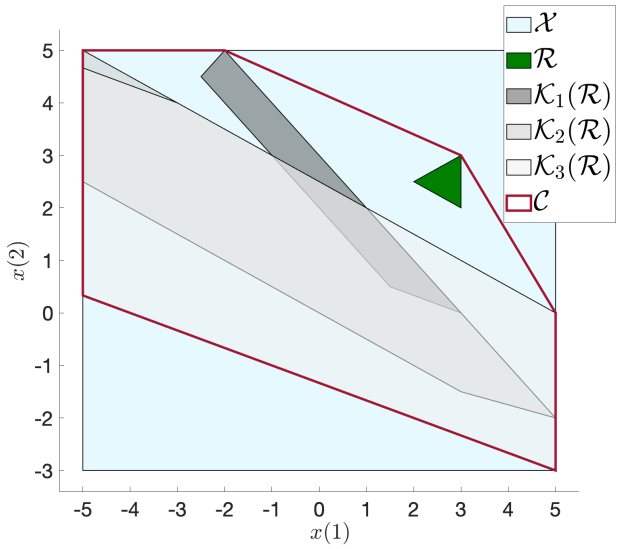

Fig. A.1. For the double integrator system (A.1 - A.2), the chosen target set (A.3) is not an invariant set, as the N-step controllable sets are not subsets of the $(N+1)$-step controllable sets.