Receding-Constraint Model Predictive Control using a Learned Approximate Control-Invariant Set

Gianni Lunardi[1], Asia La Rocca[1], Matteo Saveriano[1] and Andrea Del Prete[1]

***Abstract*— In recent years, advanced model-based and data-driven control methods are unlocking the potential of complex robotics systems, and we can expect this trend to continue at an exponential rate in the near future. However, ensuring safety with these advanced control methods remains a challenge. A well-known tool to make controllers (either Model Predictive Controllers or Reinforcement Learning policies) safe, is the so-called *control-invariant set* (a.k.a. safe set). Unfortunately, for nonlinear systems, such a set cannot be exactly computed in general. Numerical algorithms exist for computing approximate control-invariant sets, but classic theoretic control methods break down if the set is not exact. This paper presents our recent efforts to address this issue. We present a novel Model Predictive Control scheme that can guarantee recursive feasibility and/or safety under weaker assumptions than classic methods. In particular, recursive feasibility is guaranteed by making the safe-set constraint move backward over the horizon, and assuming that such set satisfies a condition that is weaker than control invariance. Safety is instead guaranteed under an even weaker assumption on the safe set, triggering a safe task-abortion strategy whenever a risk of constraint violation is detected. We evaluated our approach on a simulated robot manipulator, empirically demonstrating that it leads to less constraint violations than state-of-the-art approaches, while retaining reasonable performance in terms of tracking cost and number of completed tasks.**

I. Introduction

Ensuring safety is crucial in all robotics applications. However, this is more and more difficult with the recently increasing complexity of control methods and robotic platforms. Indeed, recent data-driven approaches, often relying on Reinforcement Learning (RL) algorithms, typically produce black-box policies that are inherently hard to certify as safe. Moreover, even model-based control methods for constrained nonlinear systems in practice struggle to guarantee safety, which consists in recursive constraint satisfaction (a.k.a. *recursive feasibility*). This is because the classic approach to guaranteeing safety, both for Model Predictive Control (MPC) and for Quadratic-Programming-based control methods, relies on the assumption of knowing a so-called *safe set* (a.k.a. control-invariant set) [1], [2], or a Control Barrier Function (CBF) [3], [4]. However, exactly computing safe sets (or CBFs) for nonlinear systems is not feasible in general. Therefore, practitioners must rely on numerical methods to compute approximate versions of such sets (or functions) [5]–[12]. Unfortunately, safety guarantees are lost if the used safe set is not exact.

In this paper, we present a novel MPC scheme that ensures: i) safety, assuming the safe set is a *conservative* approximation of a specific backward reachable set; ii) recursive feasibility, assuming the safe set is N-step control invariant, which is a weaker assumption than classic control invariance. We compared our approach with classic MPC schemes: the standard formulation (without terminal constraints but a longer horizon), and a formulation using the safe set to constrain the terminal state. Our method could successfully avoid constraint violation in more tests than the others, being able to trade off performance and safety depending on the conservativeness of the used safe set.

[1]The authors are with the Industrial Engineering Department, University of Trento, Via Sommarive 11, 38123, Trento, Italy. {name.surname}@unitn.it

This work has been partially supported by the PRIN project DOCEAT (CUP n. E63C22000410001) and the European Union under the NextGenerationEU project iNest (ECS 00000043).

II. Preliminaries

A. Notation

- $\mathbb{N}$ denotes the set of natural numbers;
- $\{x_i\}_0^N$ denotes a discrete-time trajectory given by the sequence $(x_0, \dots, x_N)$;
- $x_{i|k}$ denotes the state at time step $k+i$ predicted when solving the MPC problem at time step k;

B. Problem statement

Let us consider a discrete-time dynamical system with state and control constraints:

$$x_{i+1} = f(x_i, u_i), \qquad x \in \mathcal{X}, \qquad u \in \mathcal{U}. \tag{1}$$

Our goal is to design a control algorithm to ensure *safety* (i.e., constraint satisfaction), while preserving performance (i.e., cost minimization) as much as possible. Let us define $\mathcal{S}$ as the set containing all the equilibrium states of our system:

$$\mathcal{S} = \{x \in \mathcal{X} \mid \exists\, u \in \mathcal{U} : x = f(x, u)\}. \tag{2}$$

To achieve our goal, we rely on the *Infinite-Time Backward-Reachable Set* [1] of $\mathcal{S}$, which we denote as $\mathcal{V}$. Mathematically, it is defined as the subset of $\mathcal{X}$ starting from which it is possible to reach $\mathcal{S}$ in finite time:

$$\mathcal{V} \triangleq \{x_0 \in \mathcal{X} \mid \exists \{u_i\}_0^k, k \in \mathbb{N} : x_{k+1} \in \mathcal{S}, x_i \in \mathcal{X}, \\ u_i \in \mathcal{U}, \forall\, i = 0, \dots, k\}. \tag{3}$$

As all backward reachable sets of equilibrium states, the set $\mathcal{V}$ is a control-invariant set [1]. This means that, starting from inside $\mathcal{V}$, it is possible to remain inside $\mathcal{V}$ indefinitely. If we knew $\mathcal{V}$ we could use it to construct a safe controller. However, we cannot reasonably assume to know it in general, but we rely instead on a more realistic assumption.

Assumption 1. *We know a conservative approximation of the set $\mathcal{V}$:*

$$\hat{\mathcal{V}} \subseteq \mathcal{V} \tag{4}$$

Note that $\hat{\mathcal{V}}$ is not control invariant in general.

Assumption 2. *We know an upper bound on the number of time steps needed to safely drive the system to an equilibrium from a state in $\hat{\mathcal{V}}$, which we refer to as $\bar{N}$.*

As discussed in Section I, numerical methods exists to compute approximations of $\mathcal{V}$. Among the others, the method in [12] can be made conservative by an appropriate choice of a safety margin and it also produces an estimate of $\bar{N}$, satisfying Assumption 2. Therefore, we used [12] in our evaluation. Now we discuss different approaches to exploit $\hat{\mathcal{V}}$ in an MPC formulation to try to achieve safety.

C. Model Predictive Control and Recursive Feasibility

Let us consider the following MPC problem:

$$\begin{aligned}
\underset{\{x_i\}_0^N,\{u_i\}_0^{N-1}}{\text{minimize}} \quad & \sum_{i=0}^{N-1} \ell_i(x_i,u_i) + \ell_N(x_N) && \text{(5a)}\\
\text{subject to} \quad & x_0 = x_{init} && \text{(5b)}\\
& x_{i+1} = f(x_i,u_i) \qquad i = 0 \dots N-1 && \text{(5c)}\\
& x_i \in \mathcal{X}, u_i \in \mathcal{U} \qquad i = 0 \dots N-1 && \text{(5d)}\\
& x_N \in \mathcal{X}_N, && \text{(5e)}
\end{aligned}$$

where $\ell(\cdot)/\ell_N(\cdot)$ is the running/terminal cost, x_{init} is the current state, and $\mathcal{X}_N \subseteq \mathcal{X}$ is the terminal set [13].

Even though MPC is one of the most suited frameworks for controlling constrained systems, ensuring safety (i.e., constraint satisfaction) remains challenging when the dynamics or the constraints are nonlinear. The most common approach to ensuring safety is based on *recursive feasibility* (RF), which guarantees that, under the assumption of no disturbances/modeling errors, if an MPC problem is feasible at the first loop, it remains feasible forever.

RF is guaranteed if the MPC horizon N is *sufficiently* long (see Section 8.2 of [2]). However, in general we cannot know how long N should be. Moreover, even if N were known, it may be too long to result in acceptable computation times.

Alternatively, RF can be guaranteed by using the terminal set $\mathcal{X}_N$ to constrain the final state inside a *control-invariant* set (see Section II-D). While theoretically elegant, the practical issue with this approach is that control-invariant sets are extremely challenging (if not impossible) to compute for nonlinear systems/constraints. A special case of this approach is when an *equilibrium* state (or a set of equilibria) is used as terminal set. This solves the issue of computing control-invariant sets, but at the price of (potentially drastically) reducing the *basin of attraction* of the MPC.

Other approaches to RF exist that rely on the optimality properties of the solution and the stability of the closed loop (e.g., Section 8.3 of [2]). However, these approaches require controllability and other conditions on running and terminal costs. Therefore, they are not applicable to arbitrary cost formulations as the methods discussed in this paper.

D. Terminal Constraint

As discussed above, a common way to ensure recursive feasibility in MPC is to constrain the final state inside a control-invariant set, such as $\mathcal{V}$. Unfortunately, we do not know $\mathcal{V}$, but only $\hat{\mathcal{V}}$, which is not control invariant in general. Therefore, using $\hat{\mathcal{V}}$ as terminal set in our MPC does not ensure RF. This means that our MPC problem could become unfeasible, and at that point classic MPC theory does not tell us what to do. A common strategy to deal with unfeasibility is to relax the terminal constraint with a slack variable, which is heavily penalized in the cost function [14], [15]. In this way, when the terminal constraint cannot be satisfied, we can still get a solution that allows us to keep controlling the system, in the hope that eventually the terminal constraint be satisfied again. However, this approach does not ensure *safety*, nor RF, because the soft constraint allows the state to leave $\hat{\mathcal{V}}$, which eventually can lead to constraint violations.

III. Safe Model Predictive Control

This section describes our novel MPC scheme, which relies on two components: a safe task-abortion strategy (Section III-A, and a receding-constraint MPC formulation (Section III-B), which can be used together (Section III-C).

A. Safe Task Abortion

Our key idea to ensure safety relies on Assumption 1 and 2, and on the following two assumptions.

Assumption 3. *We have access to two computational units, which we refer to as unit A and unit B.*

Assumption 4. *We can solve the following OCP for any $x_{init} \in \hat{\mathcal{V}}$, in at most $N-1$ time steps:*

$$\begin{aligned}
\underset{\{x_i\}_0^{\bar{N}},\{u_i\}_0^{\bar{N}-1}}{\text{minimize}} \quad & \sum_{i=0}^{\bar{N}-1} \ell_i(x_i,u_i) + \ell_{\bar{N}}(x_{\bar{N}})\\
\text{subject to} \quad & \text{(5b), (5c), (5d)}\\
& x_{\bar{N}} = x_{\bar{N}-1},
\end{aligned} \tag{6}$$

The choice of the cost function is irrelevant, and can simply be used to help the solver to converge faster.

OCP (6) can be used to find a feasible trajectory to reach an equilibrium state from x_{init}. Now we can describe our strategy to safely abort the task in case we detect a risk of constraint violation. Let us assume that we are using a classic MPC formulation with terminal constraint $x_N \in \hat{\mathcal{V}}$, and that at the MPC loop k our problem becomes unfeasible. In this situation, we can follow these steps to safely abort the task:

1) unit A uses the MPC solution computed at loop $k-1$ to reach the terminal state $x_{N|k-1} \in \hat{\mathcal{V}}$;
2) in parallel, unit B solves OCP (6), using $x_{N|k-1}$ as initial state;
3) after reaching $x_{N|k-1}$, we follow the solution of OCP (6) to safely reach an equilibrium state.

This strategy allows us to reach a safe equilibrium state, where a stabilizing controller can be used to maintain the system still. Actually, we do not need to abort the task as

Algorithm 1 Terminal-Constraint MPC with Safe Abortion

Require: Number of time steps T, Initial state x_0, Initial guess $\{x_i^g\}_0^N, \{u_i^g\}_0^{N-1}$, OCP (5), Safe-abort OCP (6)

$fails \leftarrow 0$ ▷ Counter for failed OCP's
for $t = 0 \rightarrow T-1$ **do**
  $\{x_i^*\}_0^N, \{u_i^*\}_0^{N-1}, feas \leftarrow \text{OCP}(x_t, \{x_i^g\}_0^N, \{u_i^g\}_0^{N-1})$
  if $feas$ = True **then** ▷ If OCP's solution is feasible
    $fails \leftarrow 0$ ▷ Reset counter
  else
    if $fails = 0$ **then** ▷ Start solving (6) in Unit B
      SOLVESAFEABORTOCP(x_{N-1}^g)
    if $fails = N-1$ **then** ▷ Abort task
      return FOLLOWSAFEABORTTRAJECTORY()
    $fails \leftarrow fails + 1$ ▷ Increment counter
    $\{x_i^*\}_0^N, \{u_i^*\}_0^{N-1} \leftarrow \{x_i^g\}_0^N, \{u_i^g\}_0^{N-1}$ ▷ Copy last feasible solution
  $x_{t+1} \leftarrow f(x_t, u_0^*)$ ▷ Simulate system
  $\{x_i^g\}_0^{N-1}, \{u_i^g\}_0^{N-2} \leftarrow \{x_i^*\}_1^N, \{u_i^*\}_1^{N-1}$
  $x_N^g, u_{N-1}^g \leftarrow x_{N-1}^g, u_{N-2}^g$

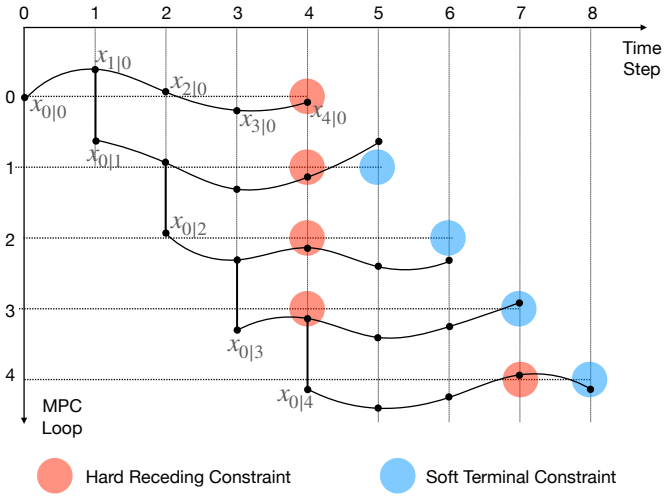

Fig. 1. Example of Receding-Constraint MPC with $N = 4$. After the MPC loop 3, the receding constraint slides forward because $x_{4|3} \in \hat{\mathcal{V}}$.

soon as one MPC problem becomes unfeasible. While we follow the last feasible solution, we can keep trying to solve OCP (5). This strategy is summarized in Alg. 1 and it can guarantee safety, as stated in the following Lemma.

Lemma 1. *Under Assumptions 1 to 4, the hard terminal-constraint MPC with safe task abortion described in Alg. 1 guarantees that constraints are never violated.*

Proof. This proof is straightforward. OCP (6) is always feasible because, by Assumption 1 and 2, from any state in $\hat{\mathcal{V}}$ we can reach an equilibrium in at most $\bar{N}$ time steps. Assumption 4 ensures that, by dedicating a computational unit to solving OCP (6), we get a solution before reaching the terminal state of the last feasible MPC problem, $x_{N|k-1}$. After reaching $x_{N|k-1}$, we follow the solution of OCP (6) to reach an equilibrium state, in which we can stay forever without violating the constraints. □

Our most critical assumption is probably Assumption 4, which relies on the MPC horizon N to be sufficiently long, and on $\bar{N}$ not to be too large, to allow for enough computation time to solve the OCP. This may be challenging because we can expect $\bar{N}$ to be rather large, since it must be sufficient to allow the system to reach an equilibrium from any state in $\hat{\mathcal{V}}$. At the same time, N cannot be set too large because it is proportional to the computation time of the MPC problem. However, learning-based warm-start techniques could be used to speed-up computation [16], [17].

1) Safe-Abort for Robot Manipulators: During our tests, we have noticed that the safe-abort OCP (6) was hard to solve for our numerical solver. Therefore, we suggest an alternative formulation, which is equivalent to (6) for the case of robot manipulators, but leads to less numerical issues with the solver. Given $x_{init} = (q_{init}, \dot{q}_{init}) \in \hat{\mathcal{V}}$, where q are the joint angles and $\dot{q}$ are the joint velocities, OCP (6) can be substituted by:

$$\begin{aligned} \underset{\{x_i\}_0^{\bar{N}}, \{u_i\}_0^{\bar{N}-1}}{\text{maximize}} \quad & d^\top \dot{q}_0 \\ \text{subject to} \quad & q_0 = q_{init} \\ & (I - dd^\top)\dot{q}_0 = 0 \\ & d^\top \dot{q}_0 \leq ||\dot{q}_{init}|| \\ & \text{(5c), (5d)}, x_{\bar{N}} = x_{\bar{N}-1}, \end{aligned} \tag{7}$$

where $d = \frac{\dot{q}_{init}}{||\dot{q}_{init}||}$ is the initial velocity direction. OCP (7) is inspired by the VBOC method [12]. Rather than fixing the initial state as in (6), we fix only the joint angles and the direction of the joint velocity vector, while maximizing the joint velocity norm. In this way, the problem is feasible for any $x_{init} \in \mathcal{X}$. In practice, our solver was always able to solve this formulation, even for cases where $x_{init} \notin \hat{\mathcal{V}}$, making the Safe Task Abortion more reliable.

B. Receding-Constraint MPC

Instead of relying exclusively on the final state to ensure safety, we could exploit the fact that, as long as at least one state $x_r \in \hat{\mathcal{V}}$ (with $1 \leq r \leq N$), we know that $x_1 \in \mathcal{V}$ because from x_1 we can reach x_r. This suggests that a less conservative constraint to include in our OCP would be:

$$(x_1 \in \hat{\mathcal{V}}) \vee (x_2 \in \hat{\mathcal{V}}) \vee \ldots \vee (x_N \in \hat{\mathcal{V}}) \tag{8}$$

Unfortunately, OR constraints are extremely challenging for numerical solvers. Even if this constraint cannot be used, we can find other ways to exploit this insight.

We suggest to adapt online the time step at which we constrain the state in $\hat{\mathcal{V}}$. For instance, if at the MPC loop $k-1$ we had $x_{r|k-1} \in \hat{\mathcal{V}}$, at the loop k we know that it is possible to have $x_{r-1|k} \in \hat{\mathcal{V}}$ (assuming no disturbances and modeling errors), therefore we can impose this constraint in a hard way. This is sufficient to ensure safety for r loops, during which this *receding constraint* would slide backward along the horizon. However, once the receding constraint reaches time

Algorithm 2 Receding-Constraint MPC with Task Abortion

Require: Number of time steps T, Initial state x_0, Initial guess $\{x_i^g\}_0^N, \{u_i^g\}_0^{N-1}$, OCP (9), `SafeAbortFlag`, Safe-abort OCP (6)

$r \leftarrow N$ ▷ Receding constraint index
for $t = 0 \rightarrow T-1$ **do**
  if `SafeAbortFlag` **and** $r = 0$ **then** ▷ Abort task
    return FOLLOWSAFEABORTTRAJECTORY()
  $\{x_i^*\}_0^N, \{u_i^*\}_0^{N-1} \leftarrow \text{OCP}(r, x_t, \{x_i^g\}_0^N, \{u_i^g\}_0^{N-1})$
  $r \leftarrow r - 1$ ▷ Recede constraint
  for $k = r+2 \rightarrow N$ **do** ▷ Search last state in $\hat{\mathcal{V}}$
    if $x_k^* \in \hat{\mathcal{V}}$ **then**
      $r \leftarrow k - 1$
  if `SafeAbortFlag` **and** $r = 0$ **then** ▷ Cannot recede
    SOLVESAFEABORTOCP(x_1^*) ▷ Solve (6) in Unit B
  $x_{t+1} \leftarrow f(x_t, u_0^*)$ ▷ Simulate dynamics
  $\{x_i^g\}_0^{N-1}, \{u_i^g\}_0^{N-2} \leftarrow \{x_i^*\}_1^N, \{u_i^*\}_1^{N-1}$
  $x_N^g, u_{N-1}^g \leftarrow x_{N-1}^g, u_{N-2}^g$

step 0, we can no longer rely on it to ensure safety. Therefore, we suggest to maintain also a soft constraint to encourage the terminal state to be in $\hat{\mathcal{V}}$. This MPC formulation can be stated as:

$$\begin{aligned} \underset{\{x_i\}_0^N, \{u_i\}_0^{N-1}, s}{\text{minimize}} \quad & \sum_{i=0}^{N-1} \ell_i(x_i, u_i) + \ell_N(x_N) + w_s||s||^2 \\ \text{subject to} \quad & \text{(5b), (5c), (5d)} \\ & x_r \in \hat{\mathcal{V}} \\ & x_N \in \hat{\mathcal{V}} \oplus s. \end{aligned} \tag{9}$$

After solving the MPC at loop $k-1$, we can check whether $x_{N|k-1} \in \hat{\mathcal{V}}$; if that is the case, at loop k we can move the receding constraint forward on $x_{N-1|k}$, which ensures safety for other $N-1$ loops. Actually, we can even check whether $x_i \in \hat{\mathcal{V}}$, for any $i > r$, and if that is the case we can set $r = i-1$ at the next loop. The resulting algorithm is summarized in Alg. 2 (with `SafeAbortFlag` set to false), and a simple example is depicted in Fig. 1.

To clarify the theoretical properties of this receding-constraint formulation, we first need to introduce the concept of N-step control invariant set.

Definition A set $\mathcal{A} \subseteq \mathcal{X}$ is *N-step control invariant* if, starting from any state in $\mathcal{A}$, it is possible either to remain in $\mathcal{A}$, or to leave $\mathcal{A}$ and come back to it within N time steps:

$$\begin{aligned} \forall x_0 \in \mathcal{A} : \exists \{u_i\}_0^{k-1},\ 1 \le k \le N, \\ x_k \in \mathcal{A},\ x_i \in \mathcal{X},\ u_i \in \mathcal{U},\ \forall i = 0, \dots, k-1 \end{aligned} \tag{10}$$

This is an extension of the well-known control invariance, with 1-step control invariance being equivalent to classic control invariance. Now we can state under which conditions the receding-constraint MPC is recursively feasible.

Theorem 1. *Assuming $\hat{\mathcal{V}}$ is N-step control invariant and the penalty on the soft terminal constraint w_s is sufficiently large, the Receding-Constraint MPC formulation described in Alg. 2 (with* `SafeAbortFlag` *set to false) is recursively feasible.*

Proof. Assume the receding-constraint formulation is feasible at the first MPC loop $k = 0$, which implies that $x_{N|0} \in \hat{\mathcal{V}}$. This guarantees recursive feasibility for N loops, during which the receding constraint can slide backward along the horizon. However, since $\hat{\mathcal{V}}$ is N-step control invariant by assumption, we know that in one of those N loops it will be possible to satisfy the soft terminal constraint $x_N \in \hat{\mathcal{V}}$. This is because at each loop k, the MPC solver tries to satisfy condition (10) for a fixed value of k. Since we know that (10) is feasible for some $k \in [1, N]$, we can infer that the soft terminal constraint $x_{N|k} \in \hat{\mathcal{V}}$ must be feasible for some MPC loop $k \in [1, N]$. Under the assumption that w_s is sufficiently large, we can infer that at that loop k, the soft terminal constraint will be exactly satisfied. When this happens, the hard receding constraint moves to time step $N-1$, ensuring recursive feasibility for another $N-1$ loops. At this point the same reasoning can be applied to ensure recursive feasibility indefinetely. □

This theorem highlights how the proposed receding-constraint MPC guarantees recursive feasibility even if the set $\hat{\mathcal{V}}$ is not control invariant. We rely indeed on a weaker condition, which is *N-step control invariance*. Our condition is weaker because any 1-step control invariant set is also N-step control invariant, for any $N > 1$, therefore our approach guarantees recursive feasibility for a larger class of sets, which contains the class of control-invariant sets. Unfortunately, computing exactly an N-step control invariant set is currently as hard as computing a standard control invariant set. However, in practice, it is more likely that a numerical method for approximating control invariant sets produces a set that is N-step control invariant, rather than control invariant. Therefore, as empirically shown in our results, our approach has a higher probability of being recursively feasible than a terminal-constraint MPC, even if in practice we cannot guarantee the assumptions of Theorem 1 to be satisfied.

C. Safe Task Abortion with Receding Constraint

Since in practice we cannot guarantee that $\hat{\mathcal{V}}$ be N-step control invariant, we cannot guarantee that the receding constraint formulation be recursively feasible. Therefore, we may need to use the task-abortion strategy when the receding constraint has reached time step 0. The problem is that at that point we have only one time step to solve OCP (6). In this paper, we assume that this computation time is enough, and we describe in Alg. 2 (with `SafeAbortFlag` set to true) the Receding-Constraint MPC with Task Abortion.

If one time step were not sufficient to solve (6), several solutions could be explored. We briefly discuss them in the following, but we leave their implementation for future work. A possible way to reduce computation time is to pre-compute a warm-start for OCP (6), before r reaches 0. While we do not know in which state the system will be at that time, we can use the trajectory predicted by the MPC as a guess. If this warm-start is not enough to solve OCP (6) in one time step, we could modify the receding-constraint formulation

to ensure that the pre-computed *safe-abort trajectory* starts exactly at the state of the system when the task abortion is initiated. To achieve this, we must modify the receding constraint from $x_{j|k} \in \hat{\mathcal{V}}$ to the more conservative $x_{j|k} = x_{j+1|k-1}$. In other words, we constrain the predicted state in $\mathcal{V}$ not to change across the MPC loops. This is bound to deteriorate performance, but it should still outperform the standard Terminal-Constraint MPC.

IV. Results

This section presents our results[1] comparing five MPC formulations:

- *Naive*: a classic formulation without terminal constraint, i.e., problem (5) with $\mathcal{X}_N = \mathcal{X}$.
- *Soft Terminal* (ST): it uses a soft terminal constraint set $\mathcal{X}_N = \hat{\mathcal{V}}$ with a penalty weight of 10^8.
- *Soft Terminal With Abort* (STWA): as the previous one, but it triggers the safe abort whenever $x_{N|k} \notin \hat{\mathcal{V}}$.
- *Hard Terminal With Abort* (HTWA): it uses a hard terminal constraint set $\mathcal{X}_N = \hat{\mathcal{V}}$, and it triggers the safe abort whenever the OCP is unfeasible (as in Alg. 1).
- *Receding*: the novel formulation (9) described by Alg. 2, using soft constraints for both $x_r \in \hat{\mathcal{V}}$ (penalty weight of 10^8) and $x_N \in \hat{\mathcal{V}}$ ($w_s = 10^5$).

For the simulations, we have considered a planar triple pendulum, thus $n_x = 6$, $n_u = 3$. We have used CasADi [18] for the symbolic computation of the dynamics, costs and constraints, and acados [19] to solve the OCPs and integrate the dynamics. The OCP is a tracking problem with respect to a static state, purposely chosen near the joint limits, to test the safety of the controllers:

$$x^{\text{ref}} = (q^{\max} - 0.05, \bar{q}, \bar{q}, 0, 0, 0), \tag{11}$$

with $\bar{q} = (q^{\max} + q^{\min})/2$. We have used as running cost a least-squares function, penalizing deviations from x^{ref} and control efforts:

$$\begin{aligned} l(x,u) &= ||x - x^{\text{ref}}||^2_Q + ||u||^2_R \\ Q &= \text{diag}([500, 10^{-4} I_5]), \quad R = 10^{-4} I_3, \end{aligned} \tag{12}$$

where I_k is the identity matrix with size k. Set membership to $\hat{\mathcal{V}}$ is verified with the constraint:

$$(1-\alpha)\phi(x) - ||\dot{q}|| \geq 0, \tag{13}$$

where $\phi(\cdot)$ is a Neural Network (NN) computing an upper bound on the joint velocity norm [12], and $\alpha \in [0, 1]$ is a safety margin that we introduced to ensure that $\hat{\mathcal{V}} \subseteq \mathcal{V}$.

We have run 100 simulations for each MPC formulation, starting from the same 100 random joint positions q_0 with $\dot{q}_0 = 0$. The time step of the MPCs was $dt = 5\,\text{ms}$. The horizon of *Naive* has been fixed to $N = 36$, so that each MPC iteration takes less than $4\,\text{ms}$ (leaving $1\,\text{ms}$ for further operations, to mimic the timing limitations of a real-time application). We used instead shorter horizons for the other approaches ($N = 35$ for the three terminal-constrained MPC's, and $N = 34$ for *Receding*), since their MPC iterations take more time due to the additional constraints.

[1]Our code is available at `https://github.com/idra-lab/safe-mpc`.

TABLE I

Number of times each controller completed the task, safely aborted it, or violated a constraint (with $\alpha = 2\%$).

MPC	Completed	Aborted	Failed
		(6) / (7)	(6) / (7)
Naive	68	-	32
ST	83	-	17
STWA	58	9 / 16	33 / 26
HTWA	60	10 / 17	30 / 23
Receding	72	11 / 20	17 / 8

TABLE II

Number of times each controller completed the task, safely aborted it, or violated a constraint (with $\alpha = 10\%$).

MPC	Completed	Aborted	Failed
		(6) / (7)	(6) / (7)
Naive	67	-	33
ST	80	-	20
STWA	56	20 / 36	24 / 8
HTWA	54	21 / 37	25 / 9
Receding	65	21 / 33	14 / 2

Table I reports the number of tasks completed, safely aborted, or failed by each controller, using a safety margin $\alpha = 2\%$. For the safe abort, we have tested both formulation (6) and (7). In terms of safety, *Naive* violated the constraints the most, while *Receding* violated them the least (when using (7) for safe abort). In terms of performance, ST completed more tasks than the others, but at the price of a higher number of failed tasks than *Receding*. STWA and HTWA performed strictly worse than ST, completing less tasks and failing more times. The lower number of completed tasks is explained by the trigger of the safe abort, while the relatively high number of failures could be explained by the small safety margin α, which is not enough to ensure $\hat{\mathcal{V}} \subseteq \mathcal{V}$.

Table II reports a similar comparison, but with a higher safety margin $\alpha = 10\%$. The number of completed tasks is slightly smaller for all approaches using $\hat{\mathcal{V}}$, but the number of failures is remarkably smaller for STWA, HTWA, and especially for *Receding*, which failed only 2 times. The number of failures remained large when using (6) for the safe abort, demonstrating the benefit of formulation (7).

Fig. 2 and 3 highlight the different risk-aversion levels of ST and STWA by showing the joint trajectories of two simulations with $\alpha = 10\%$. In both cases STWA aborted the task. ST instead completed the first task, while it failed the second one. ST is willing to take risks, which sometimes leads to completing the task (Fig. 2), but sometimes it leads to failure (Fig. 3). STWA is instead risk-averse, and it triggers a safe abort as soon as a risk of constraint violation is

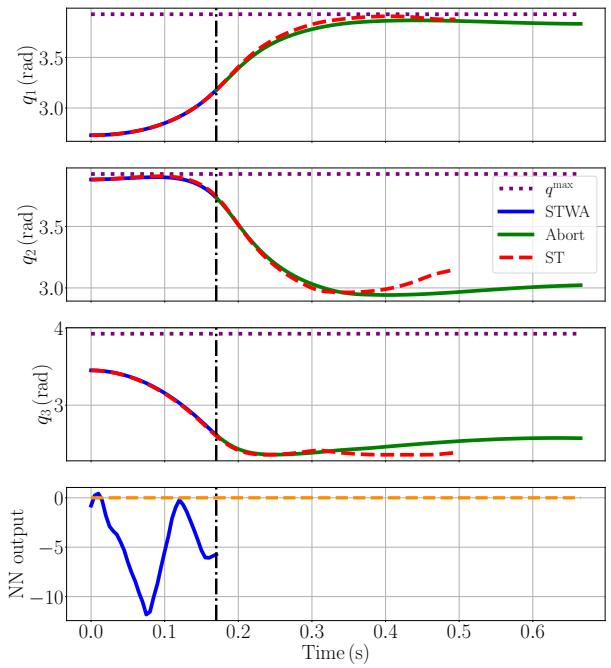

Fig. 2. Comparison between ST (task completed) and STWA (task aborted). The last plot shows the value of the terminal constraint (13). The vertical line highlights the start of the safe-abort trajectory.

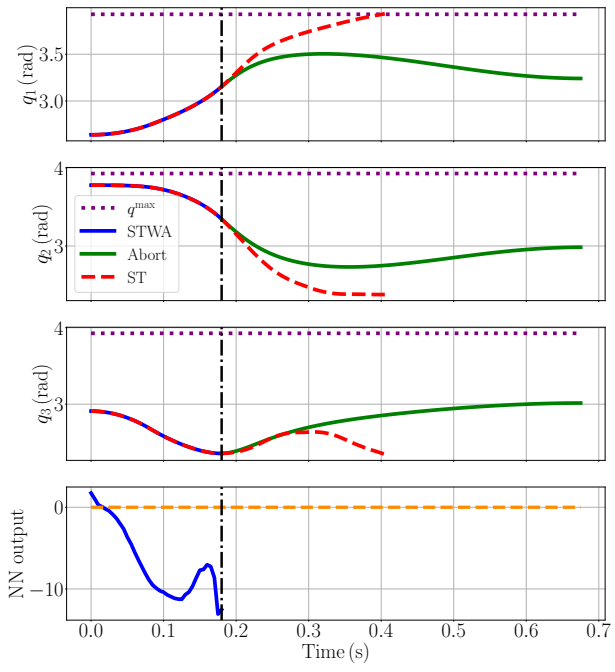

Fig. 3. Comparison between ST (task failed) and STWA (task aborted). The last plot shows the value of the terminal constraint (13). The vertical line highlights the start of the safe-abort trajectory.

TABLE III

MEAN TRACKING COST AND COMPUTATION TIMES FOR THE MPC REAL-TIME ITERATION (RTI) AND SAFE ABORT OCP.

MPC	COST ($\cdot 10^3$)	RTI (ms)	SAFE ABORT (s)	
			(6)	(7)
NAIVE	2.867	3.80	-	-
ST	2.868	3.74	-	-
STWA	3.012	4.13	0.17	2.34
HTWA	3.012	3.94	0.18	1.98
RECEDING	2.872	-	0.07	3.36

detected, which leads to less completed tasks, but also less failures.

In terms of cost, Table III shows that the average cost for the completed tasks (with $\alpha = 10\%$) is comparable for the different formulations, thus the tracking performance is not degraded by the extra constraints using $\hat{\mathcal{V}}$. The same table also reports the computation times. The 99-percentile for the real-time iteration scheme [20] is always below the time step duration (5 ms). We do not report the RTI computation times for *Receding* because of a technical issue. Indeed, the Python interface of Acados does not support time-varying constraints. Therefore our current implementation of *Receding* actually soft constrains the whole state trajectory in $\hat{\mathcal{V}}$, but then sets to zero the penalty weights for all time steps except for r and N, resulting in a much higher computation time than needed. The SAFE ABORT column reports the maximum computation times for the Task Abortion with the two methods. As previously stated, OCP (7) reports a higher number of successes (see Table I and II) at the cost of large computation times, while (6) reports good computation times (satisfying Assumption 4), but with a high number of failures.

V. CONCLUSIONS

We have presented a novel Receding-Constraint MPC formulation, which provides recursive feasibility guarantees under a weaker assumption on the used safe set with respect to classic approaches. Moreover, we have presented a task-abortion strategy that allows to reach an equilibrium state whenever a risk of constraint violation is detected. Our results on a 3-joint manipulator show the improved safety of the presented Receding-Constraint MPC with respect to other state-of-the-art methods.

Future research will focus on finding a safe-abort method that achieves high success rates as (7), but with reasonable computation times as (6). For this, we plan to extend the method in [12] to learn both the set $\hat{\mathcal{V}}$ and a policy to drive the state to an equilibrium. While this work focused on model-based control methods, our approach could be applied in the future to *safety filters* for making black-box RL policies safe.

REFERENCES

[1] F. Blanchini, "Set invariance in control," *Automatica*, vol. 35, pp. 1747–1767, 1999.

[2] L. Grüne, J. Pannek, L. Grüne, and J. Pannek, *Nonlinear model predictive control*. Springer, 2017.

[3] A. D. Ames, J. W. Grizzle, and P. Tabuada, "Control barrier function based quadratic programs with application to adaptive cruise control," in *IEEE Conference on Decision and Control*, 2014, pp. 6271–6278.

[4] Z. Wu, F. Albalawi, Z. Zhang, J. Zhang, H. Durand, and P. D. Christofides, "Control Lyapunov-Barrier function-based model predictive control of nonlinear systems," *Automatica*, vol. 109, p. 108508, 2019. [Online]. Available: https://doi.org/10.1016/j.automatica.2019.108508

[5] B. Djeridane and J. Lygeros, "Neural approximation of pde solutions: An application to reachability computations," in *IEEE Conference on Decision and Control*, 2006, pp. 3034–3039.

[6] P.-A. Coquelin, S. Martin, and R. Munos, "A dynamic programming approach to viability problems," in *IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, 2007, pp. 178–184.

[7] F. Jiang, G. Chou, M. Chen, and C. J. Tomlin, "Using neural networks to compute approximate and guaranteed feasible hamilton-jacobi-bellman pde solutions," 2016. [Online]. Available: https://www.arxiv.org/abs/1611.03158

[8] V. Rubies-Royo and C. Tomlin, "Recursive Regression with Neural Networks: Approximating the HJI PDE Solution," in *International Conference on Learning Representations*, 2017.

[9] K. C. Hsu, V. Rubies-Royo, C. J. Tomlin, and J. F. Fisac, "Safety and Liveness Guarantees through Reach-Avoid Reinforcement Learning," *Robotics: Science and Systems*, 2021.

[10] C. Dawson, S. Gao, and C. Fan, "Safe Control With Learned Certificates : A Survey of Neural Lyapunov , Barrier , and Contraction Methods for Robotics and Control," *IEEE Transactions on Robotics*, vol. 39, no. 3, pp. 1749–1767, 2023.

[11] Y. Zhou, D. Li, Y. Xi, and Y. Xu, "Data-driven approximation for feasible regions in nonlinear model predictive control," 2020. [Online]. Available: https://arxiv.org/abs/2012.03428

[12] A. La Rocca, M. Saveriano, and A. Del Prete, "VBOC: Learning the Viability Boundary of a Robot Manipulator using Optimal Control," *IEEE Robotics and Automation Letters*, 2023.

[13] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. Scokaert, "Constrained model predictive control: Stability and optimality," *Automatica*, vol. 36, no. 6, pp. 789–814, 2000.

[14] E. C. Kerrigan and J. M. Maciejowski, "Soft constraints and exact penalty functions in model predictive control," in *UKACC International Conference (Control)*, 2000.

[15] M. N. Zeilinger, M. Morari, and C. N. Jones, "Soft constrained model predictive control with robust stability guarantees," *IEEE Transactions on Automatic Control*, vol. 59, no. 5, pp. 1190–1202, 2014.

[16] N. Mansard, A. Del Prete, M. Geisert, S. Tonneau, and O. Stasse, "Using a Memory of Motion to Efficiently Warm-Start a Nonlinear Predictive Controller," in *IEEE International Conference on Robotics and Automation*, 2018, pp. 2986–2993.

[17] G. Grandesso, E. Alboni, G. P. Papini, P. M. Wensing, and A. Del Prete, "CACTO: Continuous Actor-Critic With Trajectory Optimization - Towards Global Optimality," *IEEE Robotics and Automation Letters*, vol. 8, no. 6, pp. 3318–3325, 2023.

[18] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.

[19] R. Verschueren, G. Frison, D. Kouzoupis, J. Frey, N. van Duijkeren, A. Zanelli, B. Novoselnik, T. Albin, R. Quirynen, and M. Diehl, "Acados: a modular open-source framework for fast embedded optimal control," *Mathematical Programming Computation*, vol. 14, pp. 147–183, 2019.

[20] M. Diehl, R. Findeisen, F. Allgöwer, H. G. Bock, and J. P. Schlöder, "Nominal stability of real-time iteration scheme for nonlinear model predictive control," *IEE Proceedings-Control Theory and Applications*, vol. 152, no. 3, pp. 296–308, 2005.