Safe Nonlinear Model Predictive Control using a Learned Approximate Control-Invariant Set

Asia La Rocca[1], Gianni Lunardi[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 *safe set* (a.k.a. control-invariant set). Unfortunately, for nonlinear systems, such a set cannot be exactly computed in general. Numerical algorithms exist for computing approximate safe sets, but classic theoretic control methods break down if the safe set is not exact. This extended abstract presents our recent efforts to address this issue. We present a novel Model Predictive Control scheme that can guarantee safety using a conservative approximation of a safe set. This method relies on a safe task-abortion strategy that drives the system to an equilibrium state when a risk of future constraint violation is detected. Moreover, it replaces the classic terminal constraint with a novel receding constraint, which leads to a higher number of completed tasks, while retaining safety guarantees. We evaluated our approach on a simulated robot manipulator, empirically demonstrating its superiority to two classic MPC schemes.**

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 guarantee 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) [**?**], or a Control Barrier Function (CBF) [1]. 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) [2]–[7]. Unfortunately, safety guarantees are lost if the used safe set is not exact.

In this abstract, we present a novel MPC scheme that ensures safety even when using an *approximate* safe set, as long as this set is *conservative* (i.e., it is a subset of a safe set). We compared our approach with two classic MPC schemes: one without terminal constraints (but with a longer horizon), and one using the approximate safe set to constrain the terminal state. Our method could successfully complete the task in more tests than the others, while also being able to safely abort the task in all the cases when the task could not be accomplished. While this work focuses on model-based control methods, our approach can be applied also to produce a so-called *safety filter*, to make any black-box RL policy safe.

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

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* of $\mathcal{S}$, which we denote $\mathcal{V}$. 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. 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.

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

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

$\hat{\mathcal{V}}$ is not control invariant in general. We also 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 above, numerical methods exists to compute approximations of $\mathcal{V}$ [8], which can be made conservative by an appropriate choice of a safety margin. 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 discrete-time definition of an MPC problem:

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

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

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 and/or the constraints are nonlinear. The most common approach to ensure safety is based on the concept of *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 will remain feasible forever.

RF is guaranteed if the MPC horizon N is *sufficiently* long (see Section 8.2 of [**?**]). 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 for details). While theoretically elegant, the practical issue with this approach is that safe sets are extremely challenging (if not impossible) to compute for arbitrary nonlinear systems/constraints.

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 [**?**]). However, these approaches rely on controllability assumptions and other conditions on running and terminal costs. Therefore, they are not applicable to arbitrary cost formulations as the methods discussed in this abstract.

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, constraining the final state of our MPC:

$$x_N \in \hat{\mathcal{V}} \tag{6}$$

does not ensure *recursive feasibility*. 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. 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 is satisfied again. However, this approach does not ensure *safety*, nor *recursive feasibility*, because the soft constraint allows the state to leave $\mathcal{V}$, which eventually can lead to constraint violations.

III. Safe Model Predictive Control

This section describes our novel Safe MPC scheme.

A. Safe Task Abortion

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

Assumption 2. *We have access to two computational units, which we refer to as unit A and unit B.*

Assumption 3. *We can solve the following OCP for any $x_{init} \in \hat{\mathcal{V}}$, in at most $N-1$ time steps:*

$$\begin{aligned} \underset{\{x\},\{u\}}{\text{minimize}} \quad & \sum_{i=0}^{\bar{N}-1} \ell_i(x_i, u_i) + \ell_{\bar{N}}(x_{\bar{N}}) \\ \text{subject to} \quad & x_0 = x_{init} \\ & x_{i+1} = f(x_i, u_i) \qquad i = 0 \dots \bar{N}-1 \\ & x_i \in \mathcal{X}, u_i \in \mathcal{U} \qquad i = 0 \dots \bar{N}-1 \\ & x_{\bar{N}} = x_{\bar{N}-1}, \end{aligned} \tag{7}$$

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

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 iteration 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 iteration $k-1$ to reach the terminal state $x_{N|k-1} \in \hat{\mathcal{V}}$;
2) in parallel, unit B solves OCP (7), using $x_{N|k-1}$ as initial state;
3) after reaching $x_{N|k-1}$, we follow the solution of OCP (7) to safely reach an equilibrium state.

This strategy allows us to reach a safe equilibrium state, where a basic stabilizing controller can be used to maintain the system still. This is possible thanks to the fact that, by Assumption 1, we know that OCP (7) is always feasible because from any state in $\hat{\mathcal{V}}$ we can reach an equilibrium state in at most $\bar{N}$ time steps. Assumption 3 is instead needed

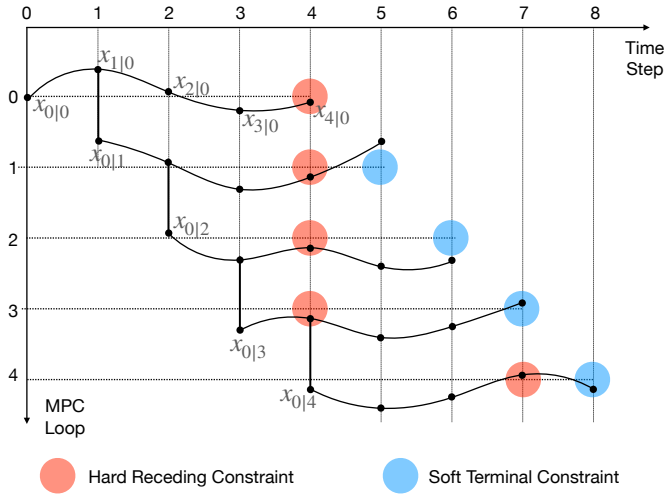

Fig. 1. Example of Receding-Constraint MPC with $N = 4$. After the MPC loop 3, the receding constraint slides forward because $x_{4|3} \in \hat{\mathcal{V}}$.

to make sure that we solve the OCP before reaching $x_{N|k}$. This is critical because it relies on the MPC horizon N to be sufficiently long to allow for enough computation time to solve the OCP. However, if this turns out to be challenging, learning-based warm-start techniques could be used to speed-up the computation of this OCP [9], [10].

B. Receding Constraint

Instead of relying exclusively on the final state to ensure safety, we could exploit the fact that, as long as at least one state $x_j \in \hat{\mathcal{V}}$ (with $1 \leq j \leq N$), we know that $x_1 \in \mathcal{V}$ because from x_1 we can reach x_j.

Based on 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 $x_{j|k-1} \in \hat{\mathcal{V}}$, at the loop k we know that it is possible to have $x_{j-1|k} \in \hat{\mathcal{V}}$ (assuming no disturbances and modeling errors), therefore we can impose this constraint in a hard way. This is sufficient to ensure safety for j loops, during which this *receding constraint* would slide backward along the horizon. However, once the receding constraint reaches time step 0, we can no longer rely on it to ensure safety. Therefore, we suggest to maintain also a soft constraint for the terminal state to be in $\hat{\mathcal{V}}$ and, after solving the MPC at loop $k-1$, to 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 would ensure safety for other $N-1$ loops. A simple example is depicted in Fig. 1.

This approach is better than the classic terminal-constraint approach because i) it ensures *safety* (if combined with the task abortion strategy, as described in the following subsection), and ii) it also ensures *recursive feasibility* for some MPC loops (i.e. j MPC loops, whenever a predicted state x_j is in $\hat{\mathcal{V}}$).

C. Safe Task Abortion with Receding Constraint

We discuss now how to combine the safe task abortion strategy with the receding constraint formulation. The issue is that the receding constraint formulation can become unfeasible when the receding constraint has reached time step 0. However, at that point, we have only one time step to solve OCP (7), which may not be enough.

A possible way to reduce computation time is to pre-compute a good warm-start for OCP (7), before the receding constraint reaches time step 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. For instance, suppose that we computed a trajectory with $x_{N-1|k} \in \hat{\mathcal{V}}$, but $x_{N|k} \notin \hat{\mathcal{V}}$. Then, we can start planning a *back-up trajectory* solving (7) with $x_{init} = x_{N-1|k}$; by Assumption 3, this computation terminates within $N-1$ time steps. At this point we have two possibilities. Case 1: during the next $N-1$ loops the MPC cannot find any state in $\hat{\mathcal{V}}$ after time step $k+N-1$, therefore we must start the task abortion procedure. In this case we can use the pre-computed *back-up trajectory* to warm-start OCP (7) from the current state x_{N-1+k}, which is probably near to $x_{N-1|k}$. Case 2: at loop r (with $k < r < k+N$), the MPC finds a trajectory with $x_{N|r} \in \hat{\mathcal{V}}$; then we can interrupt the current computation of OCP (7), and restart the whole procedure.

If this warm-start is not enough to solve OCP (7) in one time step, we could modify the receding-constraint formulation to ensure that the pre-computed *back-up 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 the performance, but it is still better than the standard terminal-constraint formulation.

IV. Results

A. Overview

In this section we compare three MPC formulations:

- *Naive MPC* is a classic formulation without any constraint besides the classic state and control sets;
- *Terminal MPC* is a classic formulation constraining the terminal state in $\hat{\mathcal{V}}$;
- *Receding MPC* is the novel formulation discussed in Section III-B, where we used soft constraints for both the receding constraint (penalty weight of 10^7) and the terminal constraint (penalty weight of 10^4).

We have considered a simulated planar double pendulum, thus $n_x = 4, n_u = 2$. We have used CasADi [11] for the symbolic computation of the dynamics, costs and constraints, and Acados [12] to solve the OCPs and integrate the dynamics. The OCP is formulated as a tracking problem with respect to a static configuration, purposely chosen to be near the joint limits, to check the capability of constraint satisfaction of the tested controllers. The used running cost is simply a least-squares function, penalizing deviations from the desired state and penalizing control efforts.

We have computed a conservative approximation of $\mathcal{V}$ using the method described in [8] and then introducing a

TABLE I

NUMBER OF TIMES EACH CONTROLLER COULD COMPLETE THE TASK, SAFELY ABORT IT, OR VIOLATED A CONSTRAINT.

MPC	COMPLETED	ABORTED	FAILED
NAIVE	32	0	68
TERMINAL	63	28	9
RECEDING	95	5	0

TABLE II

MEAN NUMBER OF SOLVER FAILURES, AND MEAN TRACKING COST.

MPC	SOLVER FAILURES	COST ($\cdot 10^4$)
NAIVE	22.7	6.343
TERMINAL	17.2	6.350
RECEDING	7.5	6.372

safety margin by reducing by 5% the maximum velocity computed by the neural network encoding the set.

We have carried out 100 simulations, starting from random joint positions q_0 and zero joint velocities, with time step $dt = 5\,\mathrm{ms}$. The horizon of the naive MPC has been fixed to $N = 32$, such that each MPC iteration can be computed within $4\,\mathrm{ms}$ (assuming $1\,\mathrm{ms}$ is needed for other operations). We have used a shorter horizon $N = 28$ for the other approaches, to account for additional computation time due to the extra constraints. For the safe abortion OCP we used a horizon $\bar{N} = 40$. We consider the task completed if the controller can run for 100 iterations (i.e., $500\,\mathrm{ms}$) without violating any constraint.

B. Discussion of the results

Table I shows how many times each approach i) completed the task successfully, or ii) safely aborted the task (see Section III-A), or iii) failed the task by violating a constraint. Overall, Receding MPC performed better than Terminal MPC, which performed better than Naive MPC. For the unaccomplished tasks, Terminal and Receding MPC could safely abort the task solving (7) in most cases. Remarkably, Receding MPC never violated any constraint. The reason why the safe task abortion was not always successful is likely that the solver could not find a solution, even if one existed.

We have evaluated the tracking performance through the total cost along the trajectory. Table II reports the mean total cost, considering only the completed tasks. The cost for Receding and Terminal MPC is slightly higher ($< 1\%$) than for Naive MPC, due to the additional constraints, which make the solver more conservative. SOLVER FAILURES refers to the mean number of times that the MPC solver failed to find a solution during a task, which is the highest for naive MPC and, to a lesser extend, for Terminal MPC.

TABLE III

COMPARISON BETWEEN NAIVE MPC AND THE OTHER FORMULATIONS IN TERMS OF NUMBER OF TIME STEPS BEFORE CONSTRAINT VIOLATION.

MPC	BETTER	EQUAL	WORSE
TERMINAL	38	26	36
RECEDING	67	29	4

Finally, Table III reports a direct comparison between each controller and Naive MPC. Each row specifies how many times, for a given task, an MPC formulation performed BETTER, EQUAL or WORSE than Naive MPC. A BETTER performance is a task where constraint violation occurred later than for Naive MPC (or never). Overall, Receding MPC outperformed the other approaches.

V. CONCLUSIONS

We presented an approach for safe nonlinear MPC, based on a conservative approximation of a control-invariant set. Our results show the benefits of the presented approach in terms of ability to avoid constraint violations in Monte Carlo simulations with a robot manipulator. Future work will explore scaling this approach to larger systems, and real-time implementation on hardware.

REFERENCES

[1] 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.

[2] 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.

[3] 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.

[4] 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

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.