NOS-NOC: A Software Package for Numerical Optimal Control of Nonsmooth Systems

Armin Nurkanović and Moritz Diehl

Abstract— **This letter introduces the open source software package for Nonsmooth Numerical Optimal Control (NOS-NOC). It is a modular tool based on CasADi [1], IPOPT [2] and MATLAB, for numerically solving Optimal Control Problems (OCP) with piecewise smooth systems (PSS). It relies on the recently introduced Finite Elements with Switch Detection [3] which enables high accuracy optimal control and simulation of PSS. The time-freezing reformulation [4], which transforms several classes of systems with state jumps into PSS is supported as well. This enables the treatment of a broad class of nonsmooth systems in a unified way. The algorithms and reformulations yield mathematical programs with complementarity constraints (MPCC). They can be solved with techniques of continuous optimization in a homotopy procedure, without the use of integer variables. The goal of the package is to automate all reformulations and to make nonsmooth optimal control problems practically solvable, without deep expert knowledge.**

I. Introduction

Nonsmooth and hybrid systems are a powerful tool to model complex physical and cyber-physical phenomena. Their theory is well developed and many good numerical simulation algorithms exist [5]. However, optimal control of nonsmooth systems is yet not wide spread, meanly due to the computational difficulty and lack of software. A notable exception are mixed integer optimization approaches [6]. However, they become intractable as soon as nonconvexities appear or exact junction times need to be computed. The open source software package `NOS-NOC` is designed to reduce this gap [7].

In this work we tackle a general class of piecewise smooth systems (PSS) of the form:

$$\dot{x} = f_i(x,u), \text{ if } x \in R_i \subset \mathbb{R}^{n_x},\ i \in \mathcal{I} := \{1,\dots,n_f\}, \quad (1)$$

where R_i are disjoint, nonempty, connected and open sets, $f_i(\cdot)$ are smooth functions on an open neighborhood of $\overline{R}_i$ and n_f is a positive integer. The event of x reaching some boundary ∂R_i is called a *switch*. The right hand side (r.h.s.) of (1) is in general discontinuous in x and u is an externally chosen control function. Several classes of systems with state jumps can be brought into the form of (1) via the time-freezing reformulation [4], [8], [9]. Thus, the focus on PSS enables a unified treatment of many different kinds of nonsmooth systems.

This research was supported by the DFG via Research Unit FOR 2401 and project 424107692 and by the EU via ELO-X 953348.

Armin Nurkanović is with the Department of Microsystems Engineering (IMTEK), University of Freiburg, Germany, Moritz Diehl is with the Department of Microsystems Engineering (IMTEK) and Department of Mathematics, University of Freiburg, Germany, {`armin.nurkanovic,moritz.diehl`}`@imtek.uni-freiburg.de`

One might wonder why not just to apply standard direct methods and existing software to a smoothed version of the r.h.s. of (1)? The necessity for tailored methods and software follows from two important results from the seminal paper of Stewart and Anitescu [10]. First, in standard direct approaches for (1), the numerical sensitivities are wrong no matter how small the integrator step-size is. This often yields artificial local minima and impairs the optimization progress [11]. Second, smoothing delivers right sensitivities only if the step-size shrinks faster than the smoothing parameter. Consequently, even for moderate accuracy many optimization variables are needed.

These two difficulties were overcome in a recently introduced discretization method that we call Finite Elements with Switch Detection (FESD) [3]. In this scheme the ODE (1) is transformed into a Dynamic Complementarity System (DCS). FESD relies on Implicit Runge-Kutta (IRK) discretizations of the DCS, but the integrator step-sizes are left as degrees of freedom as first proposed by [12]. Additional constraints ensure implicit and exact switch detection and eliminate spurious degrees of freedom. The discretization yields Mathematical Programs with Complementarity Constraints (MPCC). They are highly degenerate and nonsmooth Nonlinear Programs (NLP) [13], [14], but with suitable reformulations and homotopy procedures they can efficiently be solved with techniques for smooth NLP.

The MATLAB toolbox `NOS-NOC` [7] aims to automate the whole tool-chain and to make nonsmooth optimal control problems solvable for non-experts. In particular, it supports:

- automatic model reformulation of the PSS (1) into the computationally more suitable DCS.
- automatic formulation of the OCP, its discretization via FESD or standard IRK, several algorithms for solving the MPCC with a homotopy approach. An FESD-based switch detecting integrator is supported as well.
- time-freezing reformulation for systems with state jumps, reformulations to solve time-optimal control problems both for PSS and systems with state jumps.
- rapid prototyping with different formulations and algorithms for nonsmooth OCP

It builds on the open source software packages: CasADi [1] which is a symbolic framework for nonlinear optimization and the NLP solver IPOPT [2]. Having these packages as a back-end enables good computational performance, despite the fact that all user inputs are provided in MATLAB. All steps above can be performed in a couple of lines of code without needing a deep understanding of the numerical methods and implementation details.

Notation: The complementarity conditions for two vectors $a, b \in \mathbb{R}^n$ read as $0 \leq a \perp b \geq 0$, where $a \perp b$ means $a^\top b = 0$. The so-called C-functions $\Phi : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$ have the property $\Phi(a, b) = 0 \iff 0 \leq a \perp b \geq 0$, e.g., $\Phi(a, b) = \min(a, b)$. The concatenation of two column vectors $a \in \mathbb{R}^{n_a}$, $b \in \mathbb{R}^{n_b}$ is denoted by $(a, b) := [a^\top, b^\top]^\top$, the concatenation of several column vectors is defined in an analogous way. A column vector with all ones is denoted by $e = (1, 1, \dots, 1) \in \mathbb{R}^n$, its dimensions is clear from the context. The closure of a set C is denoted by $\overline{C}$, its boundary as ∂C. Given a matrix $M \in \mathbb{R}^{n \times m}$, its i-th row is denoted by $M_{i,\bullet}$ and its j-th column is denoted by $M_{\bullet,j}$.

Outline: Section II provides more details on the systems of interest. Sections III and IV describe many of the algorithmic features supported in NOS-NOC with a focus on FESD. In Section V we provide a breif tutorial for the use of NOS-NOC and Section VI outlines some future developments.

II. Preliminaries

In this section we briefly detail how to compactly represent the systems (1) and a how to transform them into a Dynamic Complementarity System (DCS) via Stewart's approach [15].

It is assumed that $\overline{\bigcup_{i \in \mathcal{I}} R_i} = \mathbb{R}^n$ and that $\mathbb{R}^n \setminus \bigcup_{i \in \mathcal{I}} R_i$ is a set of measure zero. Moreover, we assume that R_i are defined via the zero level sets of the components of the smooth function $c : \mathbb{R}^{n_x} \to \mathbb{R}^{n_c}$. We use a sign matrix $S \in \mathbb{R}^{n_f \times n_c}$ with non repeating rows for a compact representations as follows:

$$S = \begin{bmatrix} 1 & 1 & \dots & 1 & 1 \\ 1 & 1 & \dots & 1 & -1 \\ \vdots & \vdots & \dots & \vdots & \\ -1 & -1 & \dots & -1 & -1 \end{bmatrix}, \tag{2a}$$

$$R_i = \{x \in \mathbb{R}^{n_x} \mid \mathrm{diag}(S_{i,\bullet})c(x) > 0\}. \tag{2b}$$

The dynamics are not defined on ∂R_i and to have a meaningful notion of solution for the PSS (1) we use the Filippov convexification and define the following differential inclusion [16]:

$$\dot{x} \in F_{\mathrm{F}}(x, u) = \Big\{ F(x)\theta \mid \sum_{i \in \mathcal{I}} \theta_i = 1,\ \theta_i \geq 0,\ \theta_i = 0 \\ \text{if } x \notin \overline{R_i}, \forall i \in \mathcal{I} \Big\}, \tag{3}$$

where $\theta = (\theta_1, \dots, \theta_{n_f}) \in \mathbb{R}^{n_f}$ and $F(x) := [f_1(x), \dots, f_{n_f}(x)] \in \mathbb{R}^{n_x \times n_f}$. Note that in the interior of a set R_i we have $F_{\mathrm{F}}(x) = \{f_i(x)\}$ and on the boundary between some regions the resulting vector field is a convex combination of the neighboring vector fields. To have a computationally useful representation of the Filippov system (3) we transform it into a DCS via Stewart's reformulation [15]. In this reformulation, it is assumed that the sets R_i are represent via the *discriminant functions* $g_i(\cdot)$:

$$R_i = \{x \in \mathbb{R}^{n_x} \mid g_i(x) < \min_{j \in \mathcal{I}, j \neq i} g_j(x)\}. \tag{4}$$

Given the more intuitive representation via the sign matrix S in Eq. (2), it can be shown that the function $g : \mathbb{R}^{n_x} \to \mathbb{R}^{n_f}$ whose components are $g_i(x)$ can be found as [3]:

$$g(x) = -Sc(x). \tag{5}$$

With this representation, the convex multipliers in the r.h.s. of (3) can be found as a solution of a suitable Linear Program (LP) [15], and (3) is equivalent to

$$\dot{x} = F(x, u)\theta(x), \tag{6a}$$

$$\theta(x) \in \arg \min_{\tilde{\theta} \in \mathbb{R}^{n_f}} \quad g(x)^\top \tilde{\theta} \quad \text{s.t.} \quad e^\top \tilde{\theta} = 1,\ \tilde{\theta} \geq 0. \tag{6b}$$

We use a C-function $\Phi(\cdot, \cdot)$ for the complementarity conditions and write the KKT conditions of the LP as a nonsmooth equation

$$G_{\mathrm{LP}}(x, \theta, \lambda, \mu) := \begin{bmatrix} g(x) - \lambda - \mu e \\ 1 - e^\top \theta \\ \Phi(\theta, \lambda) \end{bmatrix} = 0. \tag{7}$$

where $\lambda \in \mathbb{R}^{n_f}_{\geq 0}$ and $\mu \in \mathbb{R}$ are the Lagrange multipliers associated to the constraints of the LP (6b). Note that $\mu = \min_{j \in \mathcal{I}} g_j(x)$. Finally, the Filippov system is equivalent to the following DCS (which can be interpreted as a nonsmooth differential algebraic equation):

$$\dot{x} = F(x, u)\theta, \quad 0 = G_{\mathrm{LP}}(x, \theta, \lambda, \mu). \tag{8}$$

A fundamental property of the multipliers $\lambda(\cdot)$ and $\mu(\cdot)$ is their continuity in time [3], whereas $\theta(\cdot)$ is in general a discontinuous function in time. This means, if there is an active-set change, which corresponds to a switch in the PSS, in some component of the complementarity pair $0 = \Phi(\theta(t_{\mathrm{s}}), \lambda(t_{\mathrm{s}}))$, then the corresponding component of $\lambda(t_{\mathrm{s}})$ must be zero. This is exploited for exact switch detection in the FESD scheme.

III. The FESD Discretization for a Single Control Interval

This section describes the discretization of single control interval in NOS-NOC via FESD. We start with a standard IRK scheme for the DCS (8). We subsequently introduce step-by-step the additional constraint which lead to the FESD scheme.

A. Standard Implicit Runge-Kutta Discretization

For ease of exposition we consider a single control interval $[0, T]$ with a given constant control input q (although more general forms are possible) and a given initial value $x_0 = s_0$. We divide the control interval into N_{fe} finite elements (i.e., integration intervals) $[t_n, t_{n+1}]$ via the grid points $0 = t_0 < t_1 < \ldots < t_{N_{\mathrm{fe}}} = T$. On each of these intervals we apply an n_{s}-stage IRK scheme, which is defined by its Butcher tableau entries $a_{i,j},\ b_i,\ c_i,\ i, j \in \{1, \dots, n_{\mathrm{s}}\}$ [17]. The fixed step-size reads as $h_n = t_{n+1} - t_n,\ n = 0, \dots, N_{\mathrm{fe}} - 1$. The approximation of the state at the grid points t_n is denoted by $x_n \approx x(t_n)$. The derivative of state at the stage points $t_n + c_i h_n,\ i = 1, \dots, n_{\mathrm{s}}$, for a single finite element are collected in the vector $V_n := (v_{n,1}, \dots, v_{n,n_{\mathrm{s}}}) \in$

$\mathbb{R}^{n_\mathrm{s} \cdot n_x}$. The stage values for the algebraic variables are collected in: $\Theta_n := (\theta_{n,1}, \ldots, \theta_{n,n_\mathrm{s}}) \in \mathbb{R}^{n_\mathrm{s} \cdot n_f}$, $\Lambda_n := (\lambda_{n,1}, \ldots, \lambda_{n,n_\mathrm{s}}) \in \mathbb{R}^{n_\mathrm{s} \cdot n_f}$ and $M_n := (\mu_{n,1}, \ldots, \mu_{n,n_\mathrm{s}}) \in \mathbb{R}^{n_\mathrm{s}}$.

Let x_n^{next} denote the value at the next time step t_{n+1}, which is obtained after a single IRK step. For the FESD scheme, next to the stage values of $\lambda_{n,i},\ \mu_{n,i}$ we need their values on t_n and t_{n+1} which are denoted by $\lambda_{n,0},\ \mu_{n,0}$ and $\lambda_{n,n_\mathrm{s}+1},\ \mu_{n,n_\mathrm{s}+1}$, respectively. Due to continuity, we impose that $\lambda_{n,n_\mathrm{s}+1} = \lambda_{n+1,0}$ and $\mu_{n,n_\mathrm{s}+1} = \mu_{n+1,0}$ and compute explicitly only the right boundary points of the finite elements in the sequel. The needed values can be computed from an LP solve for x_n^{next}:

$$0 = G_{\mathrm{LP}}(x_n^{\mathrm{next}}, \theta_{n,n_\mathrm{s}+1}, \lambda_{n,n_\mathrm{s}+1}, \mu_{n,n_\mathrm{s}+1}). \tag{9}$$

Now we can write the IRK equations for the DCS (8), including the additional Eq. (9) for the boundary value, in a compact *differential* form. We summarize all IRK equations of a single integration step in $G_{\mathrm{irk}}(x_n^{\mathrm{next}}, Z_n, q) = 0$, where $Z_n = (x_n, \Theta_n, \Lambda_n, M_n, V_n, h_n)$ collects all internal variables, and define

$$G_{\mathrm{irk}}(x_n^{\mathrm{next}}, Z_n, q) := \begin{bmatrix} v_{n,1} - F(x_n + h_n \sum_{j=1}^{n_\mathrm{s}} a_{1,j} v_{n,j}, q)\theta_{n,1} \\ \vdots \\ v_{n,n_\mathrm{s}} - F(x_n + h_n \sum_{j=1}^{n_\mathrm{s}} a_{n_\mathrm{s},j} v_{n,j}, q)\theta_{n,n_\mathrm{s}} \\ G_{\mathrm{LP}}(x_n + h_n \sum_{j=1}^{n_\mathrm{s}} a_{1,j} v_{n,j}, \theta_{n,1}, \lambda_{n,1}, \mu_{n,1}) \\ \vdots \\ G_{\mathrm{LP}}(x_n + h_n \sum_{j=1}^{n_\mathrm{s}} a_{n_\mathrm{s},j} v_{n,j}, \theta_{n,n_\mathrm{s}}, \lambda_{n,n_\mathrm{s}}, \mu_{n,n_\mathrm{s}}) \\ G_{\mathrm{LP}}(x_n^{\mathrm{next}}, \lambda_{n,n_\mathrm{s}+1}, \theta_{n,n_\mathrm{s}+1}, \mu_{n,n_\mathrm{s}+1}) \\ x_n^{\mathrm{next}} - x_n - h_n \sum_{i=1}^{n_\mathrm{s}} b_i v_{n,i} \end{bmatrix},$$

We remind the reader that some IRK schemes contain the right boundary points as a stage point (e.g., Radau and Lobbatto schemes [17]), i.e., $c_{n_\mathrm{s}} = 1$, thus we have $\lambda_{n,n_\mathrm{s}} = \lambda_{n,n_\mathrm{s}+1}\ \mu_{n,n_\mathrm{s}} = \mu_{n,n_\mathrm{s}+1}$ and we do not need the additional constraint (9).

To summarize all conditions for the standard IRK discretization, for a single control interval in a compact way, we introduce some new notation. The variables for all finite elements of a single control interval are collected in the following vectors $\mathbf{x} = (x_0, x_0^{\mathrm{next}}, \ldots, x_{N_{\mathrm{fe}}}) \in \mathbb{R}^{(2N_{\mathrm{fe}}+1)n_x}$, $\mathbf{V} = (V_0, \ldots, V_{N_{\mathrm{fe}}-1}) \in \mathbb{R}^{N_{\mathrm{fe}} n_\mathrm{s} n_x}$ and $\mathbf{h} := (h_0, \ldots, h_{N_{\mathrm{fe}}-1}) \in \mathbb{R}^{N_{\mathrm{fe}}}$. For the algebraic variables, we collect the stage values and the newly introduced boundary values into the vectors $\mathbf{\Theta} = (\Theta_0, \theta_{0,n_\mathrm{s}+1}, \ldots, \Theta_{N_{\mathrm{fe}}-1}) \in \mathbb{R}^{N_{\mathrm{fe}}(n_\mathrm{s}+1)n_f}$. The vectors $\mathbf{\Lambda} \in \mathbb{R}^{N_{\mathrm{fe}}(n_\mathrm{s}+1)n_f}$, $\mathbf{M} \in \mathbb{R}^{N_{\mathrm{fe}}(n_\mathrm{s}+1)}$ are defined accordingly. The vector $\mathbf{Z} = (\mathbf{x}, \mathbf{\Theta}, \mathbf{\Lambda}, \mathbf{M}, \mathbf{V}, \mathbf{h})$ collects all *internal* variables.

Finally, we can summarize all computations over a single control interval and interpret it as a discrete-time nonsmooth system:

$$s_1 = F_{\mathrm{std}}(\mathbf{x}),\ 0 = G_{\mathrm{std}}(s_0, \mathbf{Z}, q), \tag{10}$$

with $F_{\mathrm{std}}(\mathbf{x}) = x_{N_{\mathrm{fe}}}$ and

$$G_{\mathrm{std}}(s_0, \mathbf{Z}, q) := \begin{bmatrix} x_1 - x_0^{\mathrm{next}} \\ G_{\mathrm{irk}}(x_0^{\mathrm{next}}, Z_n, q) \\ \vdots \\ x_{N_{\mathrm{fe}}} - x_{N_{\mathrm{fe}}-1}^{\mathrm{next}} \\ G_{\mathrm{irk}}(x_{N_{\mathrm{fe}}-1}^{\mathrm{next}}, Z_{N_{\mathrm{fe}}-1}, q) \end{bmatrix}.$$

Note that we keep a dependency on h_n in (10), but h_n is implicitly given by the chosen discretization grid. This also means that for a standard IRK scheme for DCS, higher order accuracy can be achieved only if the grid points t_n coincide with all switching points, which is in practice impossible to achieve.

B. Cross-Complementarity

In FESD the step-sizes h_n are left as degrees of freedom such that the grid points t_n can coincide with the switching times. Consequently, the switches should not happen on the stages inside a finite element. To exploit the additional degrees of freedom and to achieve these two effects we introduce additional conditions to the IRK equations (10) called *cross complementaries*. A key assumption, of course, is that there are more grid points in the interior of the grid than switching points.

We exploit the fact that $\lambda(\cdot)$ and $\mu(\cdot)$ are continuous functions. To achieve implicit and exact switch detection at the boundaries of $[t_n, t_{n+1}]$ and to avoid switching inside an element we need the additional variables computed in (9). To achieve the effects described above, we introduce the cross complementarity conditions which read as [3]:

$$0 = G_{\mathrm{cross}}(\mathbf{\Theta}, \mathbf{\Lambda}) := \begin{bmatrix} \sum_{i=1}^{n_\mathrm{s}} \sum_{j=1, j\neq i}^{n_\mathrm{s}+1} \theta_{1,i}^\top \lambda_{1,j} \\ \vdots \\ \sum_{i=1}^{n_\mathrm{s}} \sum_{\substack{j=0,\\ j\neq i}}^{n_\mathrm{s}} \theta_{N_{\mathrm{fe}}-1,i}^\top \lambda_{N_{\mathrm{fe}}-1,j} \end{bmatrix}. \tag{11}$$

This additional constraint ensures two very important properties: (i) we have the same active-set in (10) in $\Phi(\theta_{n,m}, \lambda_{n,m})$ for all m and changes can happen only for different n, i.e., at grid points t_n, (ii) whenever the active-sets for two neighboring finite elements differ in the i-th and j-th components of $\Psi(\theta_{n,m}, \lambda_{n,m})$, then these two components of $\lambda_{n,n_\mathrm{s}+1}$ must be zero [3]. This will implicitly result in the constraint $0 = g_i(x_{n+1}) - g_j(x_{n+1})$ (which comes from (9) and the fact that $\mu_{n,n_\mathrm{s}+1} = \min_j g_j(x_{n+1})$). This defines the boundary between regions and R_i and R_j, cf. (4). Thus, it implicitly forces h_n to adapt for exact switch detection, cf. [3, Section 3].

C. Step-Equilibration

If no switches occur then also no active-set changes happen, hence the constraints (11) are trivially satisfied. Consequently, the step-size h_n can vary in a possibly undesired way and the optimizer can play with the discretization accuracy. To remove the spurious degrees of freedom we introduce an indicator function $\eta(\cdot)$ evaluated at the *inner* grid points $t_n,\ n = 1, \ldots, N_{\mathrm{fe}} - 1$ and its value at t_n is denoted by η_n. It has the following property: if a switch happens at t_n its value is zero, otherwise it is strictly positive. For brevity, we omit the details how a function $\eta(\cdot)$ is derived and refer to [3]. The discrete-time function $\eta(\cdot)$ depends on the values of Θ_n and Λ_n of neighboring finite elements and we define

$$\eta_n(\mathbf{\Theta}, \mathbf{\Lambda}) := \eta(\Theta_{n-1}, \Lambda_{n-1}, \Theta_n, \Lambda_n).$$

Thus, the constraints that remove possible spurious degrees of freedom in h_n read as:

$$0 = G_{\mathrm{eq}}(\mathbf{h}, \mathbf{\Theta}, \mathbf{\Lambda}) := \begin{bmatrix} (h_1 - h_0)\eta_1(\mathbf{\Theta}, \mathbf{\Lambda}) \\ \vdots \\ (h_{N_{\mathrm{fe}}-1} - h_{N_{\mathrm{fe}}-2})\eta_{N_{\mathrm{fe}}-1}(\mathbf{\Theta}, \mathbf{\Lambda}) \\ \sum_{n=0}^{N_{\mathrm{fe}}-1} h_n - T \end{bmatrix}. \quad (12)$$

We call the condition (12) step-equilibration. A consequence of (12) are locally equidistant state discretization grids between switching point, within a single control interval. Since this constraint can be quite nonlinear, NOS-NOC offers several reformulations and heuristics that help numerical convergence.

D. Finite Elements with Switch Detection

We can now summarize the developments of the last three subsections and state the FESD scheme, which makes exact switch detection possible. Similar to the standard IRK scheme (10), we summarize all computations over a single control interval and interpret it a s discrete-time nonsmooth system where internally exact switch detection is happening. The next step is computed by

$$s_1 = F_{\mathrm{fesd}}(\mathbf{x}),\ 0 = G_{\mathrm{fesd}}(\mathbf{x}, \mathbf{Z}, q), \quad (13)$$

and $F_{\mathrm{fesd}}(\mathbf{x}) = x_{N_{\mathrm{fe}}}$ renders the state transition map and the equation $0 = G_{\mathrm{fesd}}(\mathbf{x}, \mathbf{Z}, q)$ collects all other internal computations including all IRK steps within the regarded control interval:

$$G_{\mathrm{fesd}}(\mathbf{x}, \mathbf{Z}, q) := \begin{bmatrix} G_{\mathrm{std}}(\mathbf{x}, \mathbf{Z}, q) \\ G_{\mathrm{cross}}(\mathbf{\Theta}, \mathbf{\Lambda}) \\ G_{\mathrm{eq}}(\mathbf{h}, \mathbf{\Theta}, \mathbf{\Lambda}) \end{bmatrix}.$$

For a known control function q, the formulation (13) can be used as a integrator with exact switch detection for PSS (1). This feature is implemented in NOS-NOC via the function `integrator_fesd()`. It can automatically handle all kind of switching cases such as: crossing a discontinuity, sliding mode, leaving a sliding mode or spontaneous switches [16].

A forthcoming article on FESD [3] contains much more numerical examples and theoretical details including: a proof of local uniqueness of the solutions, proof of convergence of the scheme with the same accuracy as the underlying IRK scheme and convergence of the numerical sensitivities.

IV. Discretizing and Solving an Optimal Control Problem

A. Multiple Shooting-Type Discretization with FESD

One of the main goals of NOS-NOC is to numerically solve a discretized OCP. We consider $N_{\mathrm{stg}} \geq 1$ control intervals (usually equidistant) indexed by k, with piecewise constant controls collected in $\mathbf{q} = (q_0, \dots, q_{N_{\mathrm{stg}}-1}) \in \mathbb{R}^{N_{\mathrm{stg}} n_u}$. All internal variables of the previous subsections are additionally equipped with an index k. On every control interval k we apply a FESD discretization (13) with $N_{\mathrm{fe},k}$ internal finite elements. The state values at the control interval boundaries are collected in $\mathbf{s} = (s_0, \dots, s_{N_{\mathrm{stg}}}) \in \mathbb{R}^{(N_{\mathrm{stg}}+1)n_x}$. The following vector collect all internal variables of all FESD steps: $\mathcal{Z} = (\mathbf{Z}_0, \dots, \mathbf{Z}_{N_{\mathrm{stg}}-1})$. Finally the discretized OCP reads as:

$$\begin{aligned} \min_{\mathbf{s}, \mathbf{q}, \mathcal{Z}} \quad & \sum_{k=1}^{N_{\mathrm{stg}}-1} L(s_k, \mathbf{x}_k, q_k) + B(s_{N_{\mathrm{stg}}}) \\ \text{s.t.} \quad & s_0 = \bar{x}_0, \\ & s_{k+1} = F_{\mathrm{fesd}}(\mathbf{x}_k),\ k = 0, \dots, N_{\mathrm{stg}}-1, \\ & 0 = G_{\mathrm{fesd}}(\mathbf{x}_k, \mathbf{Z}_k, q_k),\ k = 0, \dots, N_{\mathrm{stg}}-1, \\ & 0 \leq G_{\mathrm{ineq}}(s_k, q_k),\ k = 0, \dots, N_{\mathrm{stg}}-1, \end{aligned}$$

where $L : \mathbb{R}^{n_x} \times \mathbb{R}^{(N_{\mathrm{fe}}+1)n_s n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}$ and $B : \mathbb{R}^{n_x} \to \mathbb{R}$ are the discrteized stage and terminal costs. The path constraint $G_{\mathrm{ineq}}(\cdot)$ in this formulation is evaluated only on the control discretization grid. However, NOS-NOC offers the possibility to evaluate this constraint on a finer grid, since all stage values $\mathbf{x}_k$ of the IRK steps are explicitly available in the OCP.

B. Reformulating and Solving MPCC

The discrete-time OCP from the last section obtained via direct transcription using FESD is an MPCC. It can be written more compactly as

$$\begin{aligned} \min_{w} \quad & f(w) && (14a) \\ \text{s.t.} \quad & 0 \leq h(w), && (14b) \\ & 0 \leq w_1 \perp w_2 \geq 0, && (14c) \end{aligned}$$

where $w = (w_0, w_1, w_2) \in \mathbb{R}^{n_w}$ is a given decomposition of the problem variables. MPCC are difficult nonsmooth NLP which violate e.g., the MFCQ at all feasible points [14]. Fortunately, they can often be solved efficiently via reformulations and homotopy approaches [13], [14], [18]. We briefly discuss the standard ways of solving MPCC that are implemented in NOS-NOC. They differ in how Eq. (14c) is handled. In all cases $w_1, w_2 \geq 0$ is kept unaltered and the bilinear constraint $w_1^\top w_2 = 0$ is treated differently.

In a homotopy procedure we solve a sequence of more regular, relaxed NLP related to (14) and parameterized by a homotopy parameter $\sigma_i \in \mathbb{R}_{\geq 0}$. Every new NLP is initialized with the solution of the previous one. In all approaches the homotopy parameter is updated via the rule: $\sigma_{i+1} = \kappa\sigma_i$, $\kappa \in (0, 1)$, $\sigma_0 > 0$, where i is the index of the NLP in the homotopy. In the limit as $\sigma_i \to 0$ (or often even for a finite i and σ_i) the solution of the relaxed NLP matches a solution of (14). `NOS-NOC` supports the following approaches:

Smoothing and Relaxation: In smoothing the bilinear term is replaced by the simpler constraint $w_1^\top w_2 = \sigma_i$ and in *relaxation* by $w_1^\top w_2 \leq \sigma_i$. A sequence of NLP for a decreasing σ_i is solved and under certain assumptions for $\sigma_i \to 0$ a solution of the initial MPCC (14) is obtained [13].

ℓ_1-Penalty: In this approach the bilinear constraint is discarded and the term $\frac{1}{\sigma_i} w_1^\top w_2$ is added to the objective, which is a penalized ℓ_1 norm of the complementarity residual. When the penalty $\frac{1}{\sigma_i}$ exceeds a certain (often finite) threshold we have $w_1^\top w_2 = 0$ and the solution of such an NLP is a solution to (14) [18].

Elastic Mode: In *elastic mode* (sometimes called ℓ_∞-approach) [14] a bounded scalar slack variable $\gamma \in [0, \bar{\gamma}]$ is introduced. The relaxed bilinear constraint reads as $w_1^\top w_2 \leq \gamma$ and we add to the objective $\frac{1}{\sigma_i}\gamma$. Variants with $w_1^\top w_2 = \gamma$ and $-\gamma \leq w_1^\top w_2 \leq \gamma$ are supported as well. Once the penalty $\frac{1}{\sigma_i}$ exceeds a certain (often finite) threshold we have $\gamma = 0$ and we recover a solution of (14) [14].

V. A NOS-NOC Tutorial

In this section we provide a short tutorial on the use of `NOS-NOC` for solving an OCP with a system with state jumps. We are interested in solving an OCP of the form of:

$$\min_{x(\cdot),u(\cdot)} \quad \int_0^T f_{\mathrm{q}}(x(\tau),u(\tau))\mathrm{d}\tau + f_{\mathrm{T}}(x(T)) \tag{15a}$$

$$\text{s.t.} \quad x_0 = s_0, \tag{15b}$$

$$\dot{x}(\tau) = f_i(x(\tau),u(\tau)), \text{ if } x \in R_i,\ i \in \mathcal{I}, \tau \in [0,T], \tag{15c}$$

$$0 \geq G_{\mathrm{ineq}}(x(\tau),u(\tau)),\ \tau \in [0,T], \tag{15d}$$

$$0 \geq G_{\mathrm{T}}(x(T)), \tag{15e}$$

where $f_{\mathrm{T}} : \mathbb{R}^{n_x} \to \mathbb{R}$ is the Mayer term and $f_{\mathrm{q}} : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}$ is the Lagrange objective term. The path and terminal constraints are collected in the functions $G_{\mathrm{ineq}} : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_{g1}}$ and $G_{\mathrm{T}} : \mathbb{R}^{n_x} \to \mathbb{R}^{n_{g2}}$, respectively. The user needs to provide the data of this OCP and it will be automatically reformulated, discretized and brought in the from described in Section IV. A user friendly interface is provided without the need for a deep understanding of the theoretical or implementation details. The user has access to all tuning parameters, intermediate results for all homotopy iterations and to all CasADi symbolic expressions and `Function` objects. This facilitates rapid prototyping and detailed analysis of solutions.

A. User Input

We consider an example of a 2D ball with elastic impacts. This system has state jumps and we use time-freezing [4] to reformulate it into a PSS of the form of (1). The first few lines of our MATLAB script read as:

```
1 import casadi.*
2 [settings] = default_settings_fesd();
3 settings.time_freezing = 1; settings.n_s = 3;
4 model.T = 5; model.N_stages = 15;
5 model.N_finite_elements = 3;
```

The function `default_settings_fesd()` provides a MATLAB `struct` with default values for all possible tuning parameters. In line 3 we indicate that time-freezing is used and set the number of IRK stages to $n_s = 3$. For the IRK scheme we keep the default choice of a Radau II-A scheme, hence we have an integration scheme with an accuracy order of 5 [17]. The next two lines define the time horizon T in the OCP, the number of control intervals N_{stg} and the number of finite elements N_{fe} per control interval, that is per default assumed to be equal for all stages. The MATLAB `struct` named `model` stores user input data which defines the OCP (15).

In our OCP example we consider a planar bouncing ball with elastic impacts. The system is reformulated into a PSS and the state vector is defined as $x = (q, v, t) \in \mathbb{R}^5$ with $q = (q_1, q_2) \in \mathbb{R}^2$ and $v = (v_1, v_2) \in \mathbb{R}^2$ being the ball's position and velocity, respectively and t is the clock state of time-freezing. The initial state is $x(0) = x_0 = (0, 0.5, 0, 0, 0)$ and the ball is controlled with some force $u(t) = (u_1(t), u_2(t)) \in \mathbb{R}^2$. After the time-freezing reformulation [4] the system has two modes of operation: (1) $\dot{x} = f_1(x,u) = (v, u_1, u_2 - g, 1)$ and $g = -9.81$, for $c(x) = q_2 > 0$; (2) $\dot{x} = f_2(x,u) = (0, v_2, 0, -\kappa q_2 - \zeta v_2, 0)$ with $\kappa = 10$ and $\zeta = 0.21198$, for $c(x) = q_2 < 0$. The first mode models the free flight of the ball when it is not in contact. The second mode is an auxliary ODE of the time-freezing PSS [4] whose trajectory endpoints satisfy the state jump law:

$$v_2(t^+) = -0.9 v_2(t^-), \quad \text{if } q_2(t) = 0 \text{ and } v_2(t) < 0. \tag{16}$$

Note that in the first mode $\dot{t} = 1$ and in the second mode we have $\dot{t} = 0$. Therefore, when $\dot{x} = f_2(x,u)$ is active, the evolution of the clock state t is frozen. By taking the pieces of the trajectory when t was evolving we recover the discontinuous solution of the original system which is modeled by $\dot{x} = f_1(x,u)$ and the point-wise law (16). However, in time-freezing the role of (16) was taken by $\dot{x} = f_2(x,u)$ and we can treat essentially a PSS. The next few lines of our script store the described PSS in `model`.

```
6    q = MX.sym('q',2); v = MX.sym('v',2);
       t = MX.sym('t');   model.x = [q;v;t];
7    model.x0 = [0;0.5;0;0;0];
8    model.c = q(2); model.S = [1; -1];
9    u = MX.sym('u',2); model.u = u;
10   f_1 = [v;u(1);u(2)-9.81;1];
11   f_2 = [0;v(2);0;-10*(q(2))-0.21198*v(2);0];
12   model.F = [f_1 f_2];
```

We use here the matrix $S = [1 \;\; -1]^\top$ for the compact definition of the regions R_i as in (2) and the matrix $F = [f_1 \; f_2]$ collects both modes of operation as in (3).

The goal is to reach $q_{\mathrm{f}} = (4, 0.25)$ with a minimal control effort and a minimal terminal velocity $v(T)$ with $T = 4$. The control force is bounded such that it is weaker than the gravitational force, i.e., $\|u\|_2^2 \leq R_u^2$ with $R_u = 7$. The next few lines summarize the stage cost, terminal cost, path and terminal constraint of our OCP of the form of (15).

```
13   model.f_q = u'*u; model.f_q_T = 10*v'*v;
14   model.g_ineq = u'*u-7^2;
15   model.g_terminal = q-[4;0.25];
```

This already completes the user input. Many more settings can be changed by the user, for example, one can choose between different MPCC reformulations via `mpcc_mode`, control the sparsity of the cross complementarities `cross_ complementarity_mode`, chose between different treatments of step equilibration via `step_equlibration_mode` and so on. A few more examples and a detailed user manual are available `NOS-NOC`'s repository [7].

B. *Solving an OCP with NOS-NOC*

The next few lines discretize and solve the OCP (15).

```
16 [solver,solver_initalization, model,settings]
       = create_nlp_fesd(model,settings);
17 [results,stats] = homotopy_solver(solver,
      model,settings,solver_initalization);
18 plot_result_ball(model,settings,results);
```

Line 16 automates all definitions and reformulations, and it has updated the `model` with all CasADi expressions for the DCS (8). Moreover, possible inconsistencies in the provided `settings` are refined. Finally, in line 17 we solve the discretized OCP with a homotopy as described in Section IV-B. The last line calls a simple function that generates the plots of Fig. 1. It depicts the optimal geometric trajectory depicted in the top plot and the recovered solution with state jumps is shown in the bottom left plot. The bottom right plot shows the optimal controls.

In total the solution trajectory has 10 switches for 5 state jumps whose number, order and timing were not known a priori. They were all exactly detected due to the FESD formulation. A high accuracy approximation of a discontinuous solution trajectory is obtained by solving 9 smooth NLP in a homotopy procedure. Internally, `NOS-NOC` took care of achieving an equidistant control grid and of all time transformations such that the desired terminal time T was achieved despite the time being frozen for state jumps.

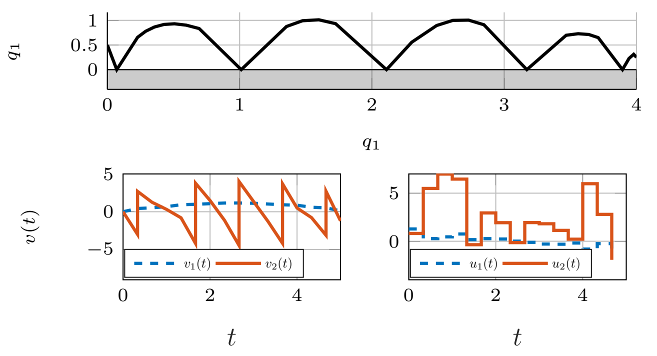

Fig. 1. The illustration of the optimal solution $q(t)$ is shown in the top plot. The bottom left plot shows the optimal velocities $v(t)$ as a function of the *physical time* t, where the state jumps are recovered. The bottom right plot shows the optimal controls $u(t)$.

VI. Conclusion and Outlook

In this letter we presented `NOS-NOC`, an open source software package for NonSmooth Numerical Optimal Control. With FESD and the time-freezing reformulation, it enables practical and high accuracy optimal control of several different classes of nonsmooth system in a unified way. The discretized OCP are solved with techniques solely from continuous optimization, without the need for any integer variables. All reformulations and details are hidden from the user so that an easy use without expert knowledge is possible.

In future work we aim to implement a python version of `NOS-NOC`. Moreover, further algorithmic developments in FESD, e.g., different reformulations of PSS into DCS, support for time-freezing for other classes of hybrid systems will be implemented as well.

References

[1] 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.

[2] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, 2006.

[3] A. Nurkanović, M. Sperl, S. Albrecht, and M. Diehl, "Finite Elements with Switch Detection for Direct Optimal Control of Nonsmooth Systems," *to be submitted*, 2022.

[4] A. Nurkanović, T. Sartor, S. Albrecht, and M. Diehl, "A Time-Freezing Approach for Numerical Optimal Control of Nonsmooth Differential Equations with State Jumps," *IEEE Control Systems Letters*, vol. 5, no. 2, pp. 439–444, 2021.

[5] B. Brogliato and A. Tanwani, "Dynamical systems coupled with monotone set-valued operators: Formalisms, applications, well-posedness, and stability," *SIAM Review*, vol. 62, no. 1, pp. 3–129, 2020.

[6] A. Bemporad and M. Morari, "Control of systems integrating logic, dynamics, and constraints," *Automatica*, vol. 35, no. 3, pp. 407–427, 1999.

[7] "NOS-NOC," https://github.com/nurkanovic/nosnoc, 2022.

[8] A. Nurkanović, S. Albrecht, B. Brogliato, and M. Diehl, "The Time-Freezing Reformulation for Numerical Optimal Control of Complementarity Lagrangian Systems with State Jumps," *arXiv preprint*, 2021.

[9] A. Nurkanović and M. Diehl, "Continuous optimization for control of hybrid systems with hysteresis via time-freezing," *Submitted to The IEEE Control Systems Letters (L-CSS)*, 2022.

[10] D. E. Stewart and M. Anitescu, "Optimal control of systems with discontinuous differential equations," *Numerische Mathematik*, vol. 114, no. 4, pp. 653–695, 2010.

[11] A. Nurkanović, S. Albrecht, and M. Diehl, "Limits of MPCC Formulations in Direct Optimal Control with Nonsmooth Differential Equations," in *2020 European Control Conference (ECC)*, 2020, pp. 2015–2020.

[12] B. T. Baumrucker and L. T. Biegler, "MPEC strategies for optimization of a class of hybrid dynamic systems," *Journal of Process Control*, vol. 19, no. 8, pp. 1248–1256, 2009.

[13] S. Scholtes, "Convergence properties of a regularization scheme for mathematical programs with complementarity constraints," *SIAM Journal on Optimization*, vol. 11, no. 4, pp. 918–936, 2001.

[14] M. Anitescu, P. Tseng, and S. J. Wright, "Elastic-mode algorithms for mathematical programs with equilibrium constraints: global convergence and stationarity properties," *Mathematical Programming*, vol. 110, no. 2, pp. 337–371, 2007.

[15] D. E. Stewart, "A high accuracy method for solving ODEs with discontinuous right-hand side," *Numerische Mathematik*, vol. 58, no. 1, pp. 299–328, 1990.

[16] A. F. Filippov, *Differential Equations with Discontinuous Righthand Sides*. Springer Science & Business Media, Series: Mathematics and its Applications (MASS), 2013, vol. 18.

[17] E. Hairer and G. Wanner, *Solving Ordinary Differential Equations II – Stiff and Differential-Algebraic Problems*, 2nd ed. Berlin Heidelberg: Springer, 1991.

[18] D. Ralph and S. J. Wright, "Some properties of regularization and penalization schemes for mpecs," *Optimization Methods and Software*, vol. 19, no. 5, pp. 527–556, 2004.