pNMPC - A Code Generation Software Tool for Implementation of Derivative Free Parameterized NMPC Scheme for Embedded Control Systems

Karthik Murali Madhavan Rathai, Mazen Alamir, Olivier Sename

pNMPC - A Code Generation Software Tool for Implementation of Derivative Free Parameterized NMPC Scheme for Embedded Control Systems

Karthik Murali Madhavan Rathai, Mazen Alamir and Olivier Sename

***Abstract*—In this paper, we propose a derivative free parameterized Nonlinear Model Predictive Control (pNMPC) code generation software (S/W) tool for embedded control systems. The proposed S/W tool serves two purposes a) Modeling and specification of the optimal control problem (OCP) and b) C code generation of a highly optimized, portable and efficient pNMPC controller suited for real time (RT) control of fast sampled systems. The underlying optimization module for the pNMPC controller is a SQP based black box optimization (BBO) solver which is specifically tailored to handle inequality constraints specified by the OCP. The S/W tool was completely programmed in C++ programming language and provides interface to MATLAB/Simulink environment with the help of MEX and C MEX-S function wrappers. Two benchmark problems were considered namely, Cart-pole stabilization problem and PVTOL probem and tests were conducted using the proposed S/W tool and also, compared against the ACADO toolkit to assess the performance, efficiency and evaluate the computation time and validate the proposed S/W in-spite of black box models. In total, the proposed S/W tool fares well and looks promising for practical RT embedded control.**

***Index Terms*—Nonlinear model predictive control, Derivative free optimization, Embedded control systems, Software engineering.**

I. Introduction

MODEL predictive control (MPC) is one of the most efficient and powerful control methodologies and over the last few years, MPC has become commonplace both in industry and academia due its performance and optimality. Initially, MPC (also known as dynamic matrix control) was restricted to chemical engineering with application for petrochemical industries, however comprehending it's potential and performance benefits, the method has gradually pervaded into other streams of engineering such as automotive, aerospace, biomedical, etc. Concomitantly, this has attracted several researchers from different engineering domains and this has positively ensued in multitude of its variants, methods, techniques and theory. Hitherto, some of the well-known extensions of the MPC [1] include (to name a few) explicit MPC [2] (EMPC), nonlinear MPC [3] (NMPC), stochastic MPC [4] (SMPC), tube based MPC [5] (TMPC), learning based MPC [6] (LMPC), economic MPC [7] (eMPC), adaptive MPC [8] (AMPC) etc. and more exciting extensions to be engendered in years to come and this trend seems to be growing unabatedly.

*This work was supported by the ITEA3 European project, 15016 EMPHYSIS (Embedded systems with physical models in the production code software). The authors are with Univ. Grenoble Alpes, CNRS, Grenoble INP, GIPSA-lab, 38000 Grenoble, France, {karthik.murali-madhavan-rathai, mazen.alamir, olivier.sename}@grenoble-inp.fr

The main reason for this popularity and rapid adoption stems from the idea of receding horizon control, the fact that an online optimization problem is solved at every sampling period to obtain the optimal control inputs for the current state of the system. This provides the necessary leeway for the control engineer either to statically or dynamically include the required objectives and system constraints into the optimization problem for control. Thus, the entire edifice of MPC controller reposes on the underpinning optimization problem. Typically, any state of the art optimization problem can be broadly classified into three types i) Zero order methods, ii) First order methods and iii) Second order methods. The order determines the type of information that the optimization solver can query to determine the solution, i.e. for zero order methods, only the function value can be obtained, for first order methods, the function value and it's first order derivatives (Jacobian) can be obtained and for second order methods, the function value, the first (Jacobian) and second order derivatives (Hessian) can be obtained or approximated. In optimization parlance, this is demarcated by classifying zero-order methods as derivative free methods or black box optimization (BBO) problem and the rest as derivative based methods. The focus of this paper circles around implementation of MPC controller using derivative free methods. Some of the compelling reasons for a need of a BBO based MPC are

- It is not uncommon in the real world industrial application, the plant model and parameters are secured due to intellectual property (IP) rights and perhaps this sensitive information is never divulged even to the internal engineers working with the system. Thus, the knowledge of the system is completely obscured from the control engineer and renders nearly impossible to design a MPC controller that predominantly relies upon the derivative information.
- In cases where the model of system exists only as computer codes or in binary format (executable files), it becomes very cumbersome to obtain accurate derivatives by numerical methods. To exacerbate, if the code involves several break, if-else or goto statements, it is highly impractical to obtain the derivatives. Case in point, this is common in the functional mock-up interface (FMI) standard [9], which is used for model exchange and co-simulation purposes. Also, this is the main framework of

the EMPHYSIS project [10].

- Even when the functional form is available to the control engineer, at times, computation of derivatives can be computationally too expensive and this could preclude from practical implementation of MPC controller for fast sampled systems. Also, in cases where the function is discontinuous and not differentiable, the derivative based methods can lead to undefined behavior.
- In today's world, with availability of deluge of data and increased computation power, data-driven modeling has challenged the pre-existing notions of first principles modeling and perhaps it wouldn't be too much of a stretch to consider the former superseding the latter in a few decades now. Machine learning models such as neural networks, Gaussian processes etc. provide excellent empirical approximations of the underlying dynamical systems and typically one can right away utilize these models for control design.

As per the cited reasons, it is certainly not an unreasonable requirement but, also, a matter of paramount importance to address the control problem by designing a derivative free MPC scheme. In this paper, we propose a BBO based parameterized NMPC (pNMPC) S/W tool to circumvent the aforementioned issues. The term parameterized refers to the parameterization of the control input [11]. The parameterization serves two purposes a) reduces the computational burden for the optimization solver and b) design of a parsimonious control input profile. It is also important to note that the potential of the pNMPC method relies upon the astuteness of the control engineer to model the input profile with efficient, effective and economical parameterization.

The key features and contributions of the pNMPC S/W are

1) The pNMPC S/W was completely programmed in C++ environment. The S/W by it's inbuilt design provides an OCP modeling framework, where the user can provide the objectives, constraints, differential equations, solver parameters, OCP parameters in a hassle-free way.
2) A one-stop control solution is provided to the end user for generation of portable, efficient, optimized and embeddable C code of the pNMPC controller. Extensions to MATLAB/Simulink environment are also provided.
3) The toolbox is independent of any external libraries such as BLAS, LAPACK, etc. and consumes less memory.
4) Additional to the inbuilt modeling framework, the S/W also provides the flexibility to include and call external functions (black box functions) which can be in the form of either source code or static/dynamic libraries.

It is also important to note that there are already several linear and nonlinear MPC off the shelf toolboxes (with code generation feature) for embedded systems such as ACADO [12], CasADi [13], VIATOC [14], FiOrdOs [15], CVXGEN [16], FORCES [17], DuQuad [18], GRAMPC [19], OSQP [20], SPLIT [21], MPT (for explicit MPC) [22]. A detailed exposition into several derivative based embedded optimal control methods is presented in [23]. Insofar, most of these toolboxes presumes that the knowledge of the model, objective and constraints are completely known and the functions involved are twice differentiable and the Jacobians/Hessians are either computable or available to the optimization solver. Thus, the MPC solver utilizes this information and exploits the structure of the system to compute the solution.

As of today, the only derivative free NMPC toolbox available on market is the PDF-MPC package [24] which uses MATLAB as the front end and MATLAB coder on top to deploy the code into embedded devices. The pNMPC code generation software tool presented in this paper is highly influenced from the aforementioned tool with improved support and features to cater the open source community and embedded control programmers. The pNMPC S/W is available in GitHub repository [25].

The paper is organized as follows : Section II discusses the pNMPC problem formulation. In Section III, the SQP-BBO optimization module for the pNMPC controller is discussed in detail. Section IV briefly discusses about the S/W syntax, structure and grammar and Section V discusses the code generation module in detail. In Section VI, simulation results are provided for two applications namely cart-pole swing up problem and PVTOL stabilization problem. Finally, the paper is concluded with future works and conclusions in Section VII.

II. pNMPC PROBLEM FORMULATION

The goal of the pNMPC toolbox is to solve the following OCP problem at every sampling period.

$$\begin{aligned} \min_{\mathbf{x}(.),\mathbf{p}(.)} \quad & \int_0^T l(\mathbf{x},\mathbf{u}(\mathbf{x},\mathbf{p},\boldsymbol{\kappa}),\boldsymbol{\kappa})\, dt + \psi(\mathbf{x}(T),\boldsymbol{\kappa}(T)) \\ \text{subject to} \quad & \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x},\mathbf{u}(\mathbf{x},\mathbf{p},\boldsymbol{\kappa}),\boldsymbol{\kappa},t),\ \forall t \in [0,T] \\ & \mathbf{u}(\mathbf{x},\mathbf{p},\boldsymbol{\kappa}) \in \mathcal{U},\ \mathbf{x} \in \mathcal{X},\ \mathbf{p} \in \mathcal{P}\ ,\forall t \in [0,T] \\ & \mathbf{x}(0) = \mathbf{x}_0,\ \mathbf{x}(T) \in \mathcal{X}_T \end{aligned} \tag{1}$$

where, $\mathbf{x} \in \mathbb{R}^{n_x}$, $\mathbf{u} \in \mathbb{R}^{n_u}$, $\mathbf{p} \in \mathbb{R}^{n_p}$ and $\boldsymbol{\kappa} \in \mathbb{R}^{n_\kappa}$ represents the state vector, input vector, input parameterization vector and external parameters (i.e. model parameters, set point for tracking or measured disturbances) vector respectively. The input map $\mathbf{u} : \mathbb{R}^{n_x} \times \mathbb{R}^{n_p} \times \mathbb{R}^{n_\kappa} \to \mathbb{R}^{n_u}$ maps the states, input parameterization vector and external parameters to actual input for the system. The sets $\mathcal{X}$, $\mathcal{U}$, $\mathcal{P}$ and $\mathcal{X}_T$ denote the state constraint, input constraint, input parameterization constraint and the terminal state constraint respectively. The OCP is subjected to a set of differential equations denoted with $\mathbf{f} : \mathbb{R}^{n_x} \times \mathbb{R}^{n_p} \times \mathbb{R}^{n_\kappa} \times \mathbb{R}_+ \to \mathbb{R}^{n_x}$ and the Lagrangian cost (stage cost) and Mayer's cost (terminal cost) are denoted with $l : \mathbb{R}^{n_x} \times \mathbb{R}^{n_p} \times \mathbb{R}^{n_\kappa} \to \mathbb{R}$ and $\psi : \mathbb{R}^{n_x} \times \mathbb{R}^{n_p} \times \mathbb{R}^{n_\kappa} \to \mathbb{R}$ respectively. It is important to note that the cost terms can be economic and no stipulations are enforced such as non-negativity or convexity. It is also important to note that handling the impact of the objectives or constraints design on the closed loop behavior is left to the user.

The pNMPC S/W transcribes the above OCP formulation (1) into a generic constrained optimization problem. The OCP transcription is implemented by discretizing the problem in time with a finite time step of Δt. Thus, the states of the system are eliminated from the problem by simulating the differential equation over time (direct single shooting method),

the integral objective is numerically approximated (Riemann sum) and the constraints are quantified at every discretized time step. Let J and h_i, $\forall i \in \{1 \dots n_c\}$ denote the transcribed objective and inequality constraints (n_c inequality constraints). The transcribed constrained optimization problem is described by the following form

$$\begin{aligned} &\min_{\mathbf{p}\in\mathbb{R}^{n_p}} \; J(\mathbf{p}) \\ &\text{s.t. } \; h_i(\mathbf{p}) \leq 0, \; \forall i \in \{1, \dots, n_c\} \end{aligned} \tag{2}$$

where $\mathbf{p} \in \mathbb{R}^{n_p}$ is the optimization vector or decision variables which is also the input parameterization vector. Let the solution of the above problem (2) be denoted with $\mathbf{p}^*$. Utilizing the optimized input parameterization vector, the optimal input $\mathbf{u}(\mathbf{x}^*(\tau), \mathbf{p}^*(\tau), \boldsymbol{\kappa}(\tau))$ is injected into the system over the time period $\tau \in [0, \Delta t]$. Henceforth, by marching forward in time and with receipt of new state vector, this process is repeated in receding horizon manner.

III. Optimization Module

In this section, the underlying BBO module for the proposed pNMPC controller is discussed in detail. The content and core results summarized in this section is based on the SQP-BBO method proposed in [24], [26], [27]. The method is based on the technique of sequentially approximating the cost and constraint functions by means of quadratic functions and at successive iterations the optimal solution is computed based on multiple trust region switch conditions. In the subsection III-B, uni-variate case of the optimization problem is discussed in detail and in subsection III-C, the method is extended for the multi-variate case.

A. Constraint reformulation (Scalarization)

Consider an optimization problem as defined below

$$\begin{aligned} &\min_{\mathbf{p}\in\mathbb{R}^{n_p}} \; J(\mathbf{p}) \\ &\text{s.t. } \; g(\mathbf{p}) \leq 0 \end{aligned} \tag{3}$$

where $\mathbf{p} \in \mathbb{R}^{n_p}$ is the vector of optimization variables and J and g represents the scalar cost and scalar constraint of the optimization problem respectively. In cases where there exists several inequality constraints, then all the constraints are scalarized by either of the two following forms.

Consider there exists n_c constraints acting upon the optimization problem, i.e.

$$h_i(\mathbf{p}) \leq 0, \; \forall i = \{1, \dots n_c\} \tag{4}$$

The two forms of constraint scalarization are

- **Form 1** - The scalar function g can be expressed as a sum over all the maximum of inequality violating constraints (if any) i.e.

$$g(\mathbf{p}) := \sum_{i=1}^{n_c} \max\{h_i(\mathbf{p}), 0\} \tag{5}$$

- **Form 2** - The scalar function g can be expressed as maximum over all the inequality constraints, i.e.

$$g(\mathbf{p}) := \max_{i\in\{1,2,\dots n_c\}} \{h_i(\mathbf{p}), 0\} \tag{6}$$

B. SQP based BBO (Uni-variate case)

Consider the optimization problem defined in (3) for an uni-variate case, then the optimization problem is defined with

$$\min_{p\in[p^{\min},\, p^{\max}]} \; J(p) \;\; \text{s.t. } \; g(p) \leq 0 \tag{7}$$

where the optimization variable $p \in \mathbb{R}$ belongs to a bounded interval $[p^{\min}, p^{\max}]$ with $p^{\max} \geq p^{\min}$. In order to define a local quadratic approximation of a function f (f is a generic representation of a function which can be either J or g) over an interval I, consider a variable $\alpha > 0$ with respect to a point p such that the interval I is defined with

$$I := [p - \alpha, p + \alpha] \cap [p^{\min}, p^{\max}] \tag{8}$$

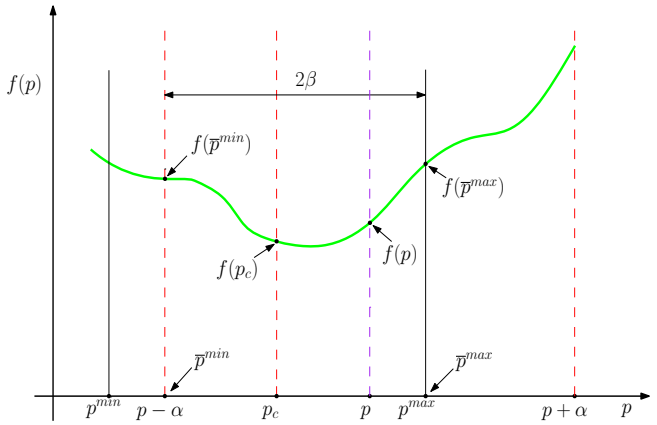

Fig. 1: Graphical interpretation of the terms used in SQP BBO method

Reformulating this interval with respect to the center of interval p_c between the extreme bounds $\overline{p}^{\min}$ and $\overline{p}^{\max}$ yields

$$I := [p_c - \beta, p_c + \beta] \tag{9}$$

where p_c, $\overline{p}^{\min}$, $\overline{p}^{\max}$ and β (semi-length of the interval I) are defined as follows

$$\begin{aligned} \overline{p}^{\min} &= \max\{p^{\min}, p - \alpha\} \\ \overline{p}^{\max} &= \min\{p^{\max}, p + \alpha\} \\ p_c &= \frac{1}{2}[\overline{p}^{\min} + \overline{p}^{\max}] \\ \beta &= \frac{1}{2}[\overline{p}^{\max} + \overline{p}^{\min}] \end{aligned} \tag{10}$$

Illustration Fig. 1 lucidly presents the defined reformulation. In order to fit a local quadratic function $q_f(p)$, three function points are considered $\{f(\overline{p}^{\min}), f(p_c), f(\overline{p}^{\max})\}$. The above points are equivalently represented with $\{f^-, f^0, f^+\}$ respectively. The locally approximated quadratic function $q_f(p)$ can be expressed using a parabolic parameteric form with

$$q_f(p) = a_f \left(\frac{p - p_c}{\beta}\right)^2 + b_f \left(\frac{p - p_c}{\beta}\right) + c_f \tag{11}$$

where a_f, b_f, c_f define the coefficients of the approximated quadratic function. As there exists three unknown coefficients and three sets of equations defined for the functions values $\{f^-, f^0, f^+\}$, the coefficients can be computed by solving the following linear algebra problem.

$$\begin{pmatrix} 1 & -1 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} a_f \\ b_f \\ c_f \end{pmatrix} = \begin{pmatrix} f^- \\ f^0 \\ f^+ \end{pmatrix} \tag{12}$$

The solution for the above problem (12) yields the results

$$\begin{aligned} a_f &= \frac{1}{2}[f^- + f^+] - f^0 \\ b_f &= \frac{1}{2}[f^+ - f^-] \\ c_f &= f^0 \end{aligned} \tag{13}$$

In order to compute the local minimizer for the approximated quadratic function (11), it is of high importance to consider the parameter a_f to determine the existence of minimizer, i.e. when $a_f \neq 0$, finite value for the minimizer exists and when $a_f = 0$, the solution can be either at $-\infty$ (when sign$(b_f) > 0$) or $+\infty$ (when sign$(b_f) < 0$). Thus, the solution $p_s^{(f)}$ is expressed with

$$p_s^{(f)} = \begin{cases} p_c - \dfrac{\beta b_f}{2a_f}, & \text{if } a_f \neq 0 \\ +\infty, & \text{if } a_f = 0 \text{ and } b_f \leq 0 \\ -\infty, & \text{if } a_f = 0 \text{ and } b_f > 0 \end{cases} \tag{14}$$

In order to confine the solution (14) obtained from the local quadratic approximation within the interval I defined in (9), the solution is projected onto the interval I and the projected solution is defined with

$$p_s^{(f,*)} := \min\Big\{\overline{p}^{\max}, \max\Big\{\overline{p}^{\min}, p_s^{(f)}\Big\}\Big\} \tag{15}$$

The optimal function value of the locally approximated quadratic function at the projected solution (15) is defined with

$$q_f^* = a_f \left(\frac{p_s^{(f,*)} - p_c}{\beta}\right)^2 + b_f \left(\frac{p_s^{(f,*)} - p_c}{\beta}\right) + c_f \tag{16}$$

Using the computed q_f^* value from (16) the extreme values of parabola $q_f(.)$ over the interval of interest I are obtained. These extreme values are defined with

$$\begin{aligned} q_f^{\min} &:= \min\Big\{f^-, f^+, q_f^*\Big\} \\ q_f^{\max} &:= \max\Big\{f^-, f^+, q_f^*\Big\} \end{aligned} \tag{17}$$

The values of p at these extreme values are denoted with $p_f^{\min}$ and $p_f^{\max}$ and are defined with

$$\begin{aligned} p_f^{\min} &:= \begin{cases} \overline{p}^{\min}, & \text{if } q_f^{\min} = f^- \\ \overline{p}^{\max}, & \text{if } q_f^{\min} = f^+ \\ p_s^{(f,*)}, & \text{if } q_f^{\min} = q_f^* \end{cases} \\ p_f^{\max} &:= \begin{cases} \overline{p}^{\min}, & \text{if } q_f^{\max} = f^- \\ \overline{p}^{\max}, & \text{if } q_f^{\max} = f^+ \\ p_s^{(f,*)}, & \text{if } q_f^{\max} = q_f^* \end{cases} \end{aligned} \tag{18}$$

When the generic function (f) is the inequality constraint function i.e. $f = g$, special cases arises for the local quadratic function $(q_g(p))$ approximation over the interval I. The possible cases are

1) $q_g(p)$ is non-negative for all $p \in I$ and this applies when $q_g^{\min} > 0$. In this case, g is non-negative in the interval I This means that the inequality constraint is violated over the entire interval I.
2) $q_g(p)$ is negative for all $p \in I$ and this applies when $q_g^{\max} \leq 0$. This means that the inequality constraint is admissible over the entire interval I.
3) $q_g(p)$ is negative on a strict subset of the interval I and this occurs when $q_g^{\min} \times q_g^{\max} < 0$.

In the last case, there could be either one or two values of p that belong to the interval I and this occurs only when $q_g(p) = 0$ and these value can be obtained by analyzing the discriminant $\Delta := b_g^2 - 4a_g c_g$ and when $\Delta \geq 0$ (non-negative) there is at least one real solution. Thus, the possible candidate solutions are

$$\begin{aligned} p_g^{(0,+)} &:= p_c + \beta \max\Big\{\frac{-b_g \pm \sqrt{\Delta}}{2a_g}\Big\} \\ p_g^{(0,-)} &:= p_c + \beta \min\Big\{\frac{-b_g \pm \sqrt{\Delta}}{2a_g}\Big\} \end{aligned} \tag{19}$$

Let $\mathcal{Z}_g^- \subset I$ denote the subset of values of p that belong to I where $q_g(.)$ is negative. The set $\mathcal{Z}_g$ is computed with

$$\mathcal{Z}_g^- := \begin{cases} [\overline{p}^{\min}, p_g^{(0,-)}], & \text{if } p_g^{(0,-)} < \overline{p}^{\min} \;\&\; g^- \leq 0 \\ [p_g^{(0,+)}, \overline{p}^{\max}], & \text{if } p_g^{(0,-)} < \overline{p}^{\min} \;\&\; g^+ \leq 0 \\ [\overline{p}^{\min}, p_g^{(0,-)}], & \text{if } p_g^{(0,+)} > \overline{p}^{\max} \;\&\; g^- \leq 0 \\ [p_g^{(0,-)}, \overline{p}^{\max}], & \text{if } p_g^{(0,+)} > \overline{p}^{\max} \;\&\; g^+ \leq 0 \\ [p_g^{(0,-)}, p_g^{(0,+)}], & \text{if } [p_g^{(0,-)}, p_g^{(0,+)}] \subset I \;\&\; a_g > 0 \\ \mathrm{A} \cup \mathrm{B}, & \text{if } [p_g^{(0,-)}, p_g^{(0,+)}] \subset I \;\&\; a_g < 0 \end{cases} \tag{20}$$

where, $\mathrm{A} = [\overline{p}^{\min}, p_g^{(0,-)}]$ and $\mathrm{B} = [p_g^{(0,+)}, \overline{p}^{\max}]$. It is important to note that the set $\mathcal{Z}_g$ is either an interval or a union of two intervals and this difference is demarcated with $n_g^z \in \{1, 2\}$ which corresponds to an interval $I_g^{(1)}$ or union of intervals $I_g^{(2)}$. All the terminologies used are summarized and tabulated in Table I.

TABLE I: Notation and meaning

Notation	Meaning
p_c	Center $I = [p - \alpha, p + \alpha] \cap [p^{\min}, p^{\max}]$
β	Semi-length of I
$q_f(.)$	Local quadratic approximation of f over I
a_f, b_f, c_f	Coefficients of parabola $q_f(.)$, $f \in \{J, g\}$
$p_s^{(f)}$	Position of singular point $q_f(.)$
$p_s^{(f,*)}$	Projection of $p_s^{(f)}$ over I
q_f^*	The value of parabola $q_f(.)$ at $p_s^{(f,*)}$
$q_f^{\min}$	Minimum value of $q_f(.)$ on I
$q_f^{\max}$	Maximum value of $q_f(.)$ on I
$p_f^{\min}$	Location of minimum value of $q_f(.)$ on I
$p_f^{\max}$	Location of maximum value of $q_f(.)$ on I
$p_g^{(0,+)}$, $p_g^{(0,-)}$	Solution of $q_g(p) = 0$
$\mathcal{Z}_g^-$	Subset of I where $q_g(.) \leq 0$
n_g^z	Number of intervals in $\mathcal{Z}_g^-$
$I_g^{(1)}$, $I_g^{(2)}$	Interval defining $\mathcal{Z}_g^-$

The BBO algorithm is completely premised upon the previously defined three cases. The algorithm for the next iteration $p^{(i+1)}$ and trust region update size $\alpha^{(i+1)}$ are given by the following steps

1) When $q_g^{\max} \leq 0$, then the whole interval I is the search space and the function J is minimized over the whole interval I. The candidate value of the update $p^{(i+1)}$ is given by

$$p^{\text{cand}} \leftarrow p_J^{\min} \tag{21}$$

The minimizer in the above equation (21) is obtained from equation (18) where the function f is replaced with J. This computation is based on the assumption that the quadratic approximation is appropriate. The logical condition to verify this assumption and also, to update the trust region size is given by $\mathcal{C} \leftarrow \mathcal{C}_1 \bigvee \mathcal{C}_2$ where,

$$\begin{aligned} \mathcal{C}_1 &\leftarrow \Big(J(p^{\text{cand}}) < J(p^{(i)})\Big) \bigwedge \Big(g(p^{\text{cand}}) \leq 0\Big) \\ \mathcal{C}_2 &\leftarrow \Big(J(p^{\text{cand}}) \leq J(p^{(i)})\Big) \bigwedge \Big(g(p^{\text{cand}}) < 0\Big) \end{aligned} \tag{22}$$

2) When $q_g^{\max} > 0$, which means that the constraints are strictly non-negative which tantamount to constraint violation and the priority ought to be given to minimization of the inequality constraint g. In this case, the candidate value of the update $p^{(i+1)}$ is given by

$$p^{\text{cand}} \leftarrow p_g^{\min} \tag{23}$$

and the trust region update condition is given with

$$\mathcal{C} \leftarrow \Big(g(p^{\text{cand}}) < g(p^{(i)})\Big) \tag{24}$$

3) When $q_g^{\max} \geq 0$ and $q_g^{\min} \leq 0$, which means that there exists a subset in I where there exists a solution. In this case, the integer n_g^z and the corresponding intervals $I_g^{(1)}$ and $I_g^{(2)}$ are computed. Utilizing these intervals, the potential candidate for $p^{(i+1)}$ update is computed by

$$p^{\text{cand}} \leftarrow \arg \min_{p \in \{p_J^{(\min,l)}\}_{l=1}^{n_g^z}} J(p) \tag{25}$$

where $p_J^{(\min,1)}$ and $p_J^{(\min,2)}$ are optimal solutions that minimize J over the intervals $I_g^{(1)}$ and $I_g^{(2)}$ respectively. The trust region update condition is given by

$$\mathcal{C} \leftarrow \begin{cases} \Big(g(p^{\text{cand}}) < g(p^{(i)})\Big), & \text{if } g(p^{(i)}) > 0 \\ \Big(J(p^{\text{cand}}) < J(p^{(i)})\Big) \bigwedge \Big(g(p^{\text{cand}})\Big) \leq 0, & \text{else} \end{cases} \tag{26}$$

The update of next iterate $p^{(i+1)}$ and $\alpha^{(i+1)}$ is implemented according to following rules.

- If $\mathcal{C}$ is true, then
  - $p^{(i+1)}$ is assigned to the computed candidate value p^{cand}.
  - The trust region parameter $\alpha^{(i)}$ is increased according to

$$\alpha^{(i+1)} \leftarrow \beta^+ \cdot \alpha^{(i)};\ \beta^+ > 1 \tag{27}$$

- Otherwise, the current value $p^{(i+1)} \leftarrow p^{(i)}$ is used and the trust region size is decreased according to

$$\alpha^{(i+1)} \leftarrow \max\Big\{\alpha^{\min}, \beta^- \cdot \alpha^{(i)}\Big\};\ 0 < \beta^- < 1 \tag{28}$$

Let N_{iter} represent the number of iterations the above algorithm is repeated, then the total number of function evaluations of the objective and constraint functions $(J,\ g)$ is

$$N_{eval} = 4N_{iter} + 1 \tag{29}$$

C. SQP based BBO (Multi-variate case)

The multi-variate case is an extension to the uni-variate case, where an uni-variate optimization problem is solved over each component of the decision variables while the rest of the values are maintained constant. Consider the decision variables to be $\mathbf{p} \in \mathbb{R}^{n_p}$ and let the list of decision variables be indexed with $l \in \{1, 2, \ldots, n_p\}$. Let $\eta \in \mathbb{R}$ denote the scalar variable over which the uni-variate optimization is performed. In notational form, this is expressed with $\mathbf{p}^{(\eta,l)}$ and an element in $\mathbb{R}^{n_p}$ is defined as

$$\mathbf{p}_j^{(\eta,l)} := \begin{cases} p_j & \text{if } j \neq l \\ \eta & \text{if } j = l \end{cases} \tag{30}$$

The formulation (30) is extended for $\forall j \in \{1, 2, \ldots, n_p\}$. The total number of loop count to visit all the n_p components for N_{iter} iterations and N_{eval} is given by

$$N_{loop} = \left\lceil \frac{N_{eval} - 1}{4n_p \times N_{iter}} \right\rceil \tag{31}$$

It is important to note that a feasible choice of pair (N_{eval}, N_{iter}) and must satisfy the inequality

$$N_{eval} \geq 4n_p N_{iter} + 1 \tag{32}$$

For convergence results for the SQP-BBO algorithm, refer [26][27].

IV. pNMPC S/W structure

In this section, the pNMPC S/W structure, syntax and features are discussed in a bird's eye view level. It is important to note that a complete coverage of all the features of the S/W is not feasible within the scope of this paper. The goal of this section is to point out the crucial elements of the S/W along with it's use case. The core components of the S/W can be broadly categorized into five parts which are a) Symbolic classes, b) OCP design classes, c) Real/Symbolic classes, d) Control parameterization classes and e) Code generation classes.

A. Symbolic classes

The symbolic classes of the pNMPC S/W are

- `HyperStates` - This serves as the base class for other derived symbolic classes. `HyperStates` can also be used as a substitution or an intermediate variable.
  Usage: `HyperStates H = 2*x1*k1+u1*x2+F;`.

- `States` - This symbolic class is used to define the states for the system.
  Usage: `States x1,x2;`.
- `Inputs` - This symbolic class is used to define the inputs for the system.
  Usage: `Inputs u1;`.
- `Params` - This symbolic class is used to define the parameters for the system.
  Usage: `Params k1;`.
- `External` - This symbolic class is used to define external variables which invokes function calls from source files or libraries. The below example links the symbolic object/variable to a function named `NeuralNet`.
  Usage: `External F = "NeuralNet";`.

B. OCP design classes

The OCP design classes of the pNMPC S/W are

- `ParameterizationMap` - This OCP class maps the input parameterization vector to the actual input for the system. The parameterization map must be defined within the begin and end guards. **Usage**:

```
BEGIN_PARAMETERIZATION_MAP
 ParameterizationMap u1 = p1*x1+p2;
 ParameterizationMap u2 = p3*x2+p4;
END_PARAMETERIZATION_MAP
```

  where `p1,p2,p3,p4` denote the input parameterized variables, `x1,x2` denote the states of the system and `u1,u2` denote the actual input to the system.

- `DiffEquation` - This OCP object is used to define the underlying differential equation of the system. The differential equations must be defined within the begin and end guards. **Usage**:

```
BEGIN_DIFFERENTIAL
 DiffEquation x1d = -x1*x2+u1;
 DiffEquation x2d = -x2+u2;
END_DIFFERENTIAL
```

- `ScalarConstraint` - This OCP class is used to define the scalar constraints for the system. The scalar constraint class can be classified into two types which are a) Regular constraints and b) Terminal constraints and these are defined within the respective begin and end guards. **Usage**:

```
BEGIN_CONSTRAINTS
 BEGIN_REGULAR_CONSTRAINTS
  ScalarConstraint GR1 = {-5<=x1<=5};
  ScalarConstraint GR2 = {-5<=x2<=5};
  ScalarConstraint GR3 = {-1<=u1<=1};
  ScalarConstraint GR4 = {-1<=u2<=1};
 END_REGULAR_CONSTRAINTS
 BEGIN_TERMINAL_CONSTRAINTS
  ScalarConstraint GT1 = {-1<=x1<=1};
  ScalarConstraint GT2 = {-1<=x2<=1};
  ScalarConstraint GT3 = {-1<=u1<=1};
  ScalarConstraint GT4 = {-1<=u2<=1};
 END_TERMINAL_CONSTRAINTS
END_CONSTRAINTS
```

- `ScalarObjective` - This OCP class is used to define the scalar objectives for the system. The scalar objective class can be classified into two types which are a) Lagrangian (stage cost) and b) Mayer (terminal cost) and these are defined within the respective begin and end guards. Multiple definition of objectives within each guard will be scalarized by addition. **Usage**:

```
BEGIN_OBJECTIVES
 BEGIN_LAGRANGIAN
  ScalarObjective L1 = x1*x1+x2*x2;
  ScalarObjective L2 = u1*u1+u2*u2;
 END_LAGRANGIAN
 BEGIN_MAYER
  ScalarObjective M1 = 5*x1*x1+5*x2*x2;
  ScalarObjective M2 = 2*u1*u1+2*u2*u2;
 END_MAYER
END_OBJECTIVES
```

C. Control parameterization classes

The pNMPC S/W has two control parameterization features namely a) Piecewise parameterization and b) Linear parameterization. The definitions for the classes are given below

- `ControlParamZ<Piecewise>` - This class parameterizes the input in a piecewise fashion over the prediction horizon. There are multiple constructor overloads for this class, however in the interest of brevity, only the simplest case is presented here. **Usage**:

```
ControlParamZ<Piecewise> {5,p1,-1,1};
```

  where, the first argument defines the number of piecewise parameterization placed equidistantly over the prediction horizon, the second argument `p1` represents the input object from the `Inputs` class, so the specified parameterization is latched to this input and the last two arguments represents the minimum and maximum bounds over the input `p1` respectively.

- `ControlParamZ<Linear>` - This class parameterizes the input in a linear fashion over the prediction horizon. The other specifications follow suit as the piecewise parameterization. **Usage**:

```
ControlParamZ<Linear> {2,p2,-1,1};
```

D. Real/Symbolic classes

The pNMPC software supports several functions and operation between real numbers and as well as symbolic

objects. The list of supported functions are {`sin`, `cos`, `tan`, `sinh`, `cosh`, `tanh`, `exp`, `log`, `abs`, `asin`, `acos`, `atan`, `asinh`, `acosh`, `atanh`, `minimum`, `maximum`, `sign`}. The list of supported numerical operations are {$+$, $-$, $*$, $\backslash$, $\wedge$, $\geq$, $\leq$, &&, ||}, where the last two are logical AND and OR operations, which is used to couple multiple constraints to one (Example - blocking constraints in state space region). The scalar real numbers are declared with `Real` keyword and real valued matrices are declared with `MATReal` keyword. Symbolic matrices are declared with `MATHyperStates` keyword. **Usage**:

```
MATHyperStates A(2,2); MATReal B(2,2);
// Populate symbolic matrix
A[0][0] = x1; A[0][1] = x1+sin(x2);
A[1][0] = u1; A[1][1] = u1*x2;
// Populate real matrix
B[0][0] = 2; B[0][1] = -1;
B[1][0] = 5; B[1][1] = 3;
// Symbolic Matrix - Real Matrix multiplication
MATHyperStates C = A*B;
```

There are several in-build matrix operations such as matrix-vector operations, matrix-matrix operations etc. packaged along with the S/W.

E. Code generation classes

The code generation classes are formed by a composition of three classes which are a) `INTEGRATOR`, b) `CONST_FORM` and c) `PNMPCGEN`. The definitions of the classes are given below.

- `INTEGRATOR` - This enum class specifies the integrator to be used to simulate the underlying differential equations. It is important to note that the S/W for now has support only for explicit solvers. **Usage**:

  ```
  INTEGRATOR iODE = INTEGRATOR::RK45;
  ```

- `CONST_FORM` - This enum class specifies the method for constraint scalarization either using Form 1 or Form 2 technique as mentioned in Section III-A. **Usage**:

  ```
  CONST_FORM cF = CONST_FORM::FORM_1;
  ```

- `PNMPCGEN` - This class is used to create the instance of the OCP with all the aforementioned classes and functions. The class follows singleton design pattern, thereby only one instance of the controller can be generated. The setting function calls are initial and final time of the OCP, step-size for the embedded integrator, SQP solver parameters, constraint form, integrator object and a Boolean flag to toggle between parameter data at current time step or over the prediction horizon. Finally, the method `genCCode()` is invoked to generate the respective C files. **Usage**:

  ```
  PNMPCGEN* pNMPC = PNMPCGEN::getSton();
  pNMPC->setInitialTime(0);
  pNMPC->setFinalTime(2);
  pNMPC->setStepSize(0.1);
  pNMPC->getSolver()->setNiter(4);
  pNMPC->setConstForm(cF);
  pNMPC->setIntegrator(iODE);
  // Generate C codes
  pNMPC->genCCode();
  ```

V. pNMPC CODE GENERATION MODULE

The pNMPC code generation module is not dissimilar to any compiler design paradigm [28] at the same time the steps involved are not as extensive as for compilation process of any programming language. The motivation for adopting a scheme as such are two folds which are:

1) The ingredients of the OCP problem (objectives, constraints, dynamics etc.) fed by the user are broken down into fundamental elements and then modeled in an appropriate way to suit the optimization module. Case in point, the inequality constraints ought to be aligned in a non-positive formulation as described in (3).
2) By breaking down the OCP into it's fundamental elements, code optimization can be performed efficiently which in turn benefits in reducing the memory footprint (space complexity) and burning less computer clock cycles (time complexity). This feature has high practical importance, especially for low-end embedded devices.

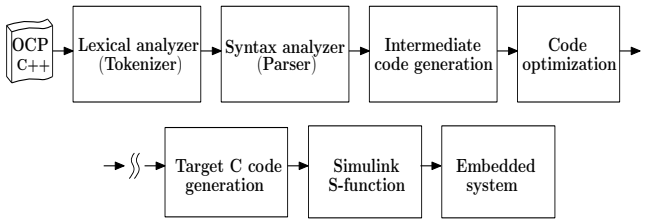

Fig. 2: pNMPC code generation process

The pNMPC code generation process is illustrated as a process flow diagram in Fig. 2. The key stages involved in the process are briefed below:

- **OCP design and specification**: The OCP's design and specification are programmed by the user using C++ as a front-end modeling language.
- **Lexical analyzer**: Lexical analyzer or tokenizer takes the user's OCP design and specification and breaks down the entries into separate characters or special tokens such as the states, inputs, parameters, math operations etc.
- **Syntax analyzer**: Syntax analyzer or parser scans through these tokens and contrives a relational tree or parse tree. Consider an example $y = x_1x_2 + sin(x_1x_2)$, then the computed parse tree for this relation is illustrated in Fig. 3.
- **Intermediate code generation**: In this stage, the parse tree is stored as Three-address code (3AC) and stored in

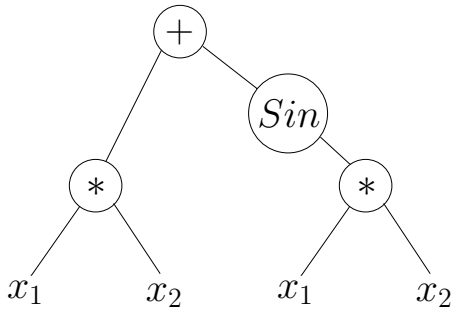

Fig. 3: Parse tree structure

a stack data structure. Additional variables x_3, x_4, x_5 and x_6 are induced in the due process. It is important to note that the code is unoptimized at this stage.

- **Code optimization**: The code optimization stage is a crucial stage of the code generation module. Typically, the code optimization module obliterates redundant relations, self negation operations, identity and inverse relations for addition, subtraction, multiplication and division operations such as (to list a few) $0 * \mathtt{x}$, $\mathtt{x} + 0$ etc. In the above example, it is clearly evident that the term $\mathtt{x1} * \mathtt{x2}$ is computed twice. After code optimization, this redundancy is removed and the stack is updated.
- **Target C code generation**: In this stage, the embedded C files are exported by using the file stream operations. Under circumstances of securing the generated source code, either a static or dynamic library can be created. However, this has to be done manually by the user.
- **Simulink S-function**: This stage is optional, however in today's world, embedded control has virtually become MATLAB/Simulink's fiefdom and it plays a dominant role in design, development and deployment of production code to embedded systems both in industry and academia. The pNMPC S/W provides the flexibility to provide Simulink compliant C codes and this can be included into Simulink by availing MEX wrappers in MATLAB or C-MEX S-functions features from Simulink.
- **Embedded system**: Finally, the generated code is deployed into the embedded system either through Simulink or manually by the user.

VI. Results and Simulations

The pNMPC S/W was tested for several examples and the simulation results looks promising and viable for RT implementation. In this section, in the interest of space, the results and simulation of two examples are described in detail which are a) Cart-pole swing up problem and b) PVTOL stabilization problem. The outputs were compared against ACADO toolkit [12] as a benchmark S/W to study the performance and computation time of the two examples. The examples were simulated in MATLAB/Simulink on a Intel Core i7, 16GB RAM PC. The pNMPC C++ codes for the respective examples are listed in Appendix A and B.

A. Cart-pole swing up problem

The task of the cart-pole swing up problem [29] is to stabilize a pole in an upright direction (typically starting from downward position) which is attached to a movable cart by means of a revolute joint. The control input to this system is the horizontal force applied to the cart and system is bounded by physical constraints which are the length of cart travel and the input force. The nonlinear state space equations of the system are given below

$$\begin{aligned}
\dot{x} &= v \\
\dot{v} &= \frac{-m_2 l sin(\theta)\dot{\theta}^2 + u + m_2 g cos(\theta) sin(\theta)}{m_1 + m_2(1 - cos^2(\theta))} \\
\dot{\theta} &= \omega \\
\dot{\omega} &= \frac{-m_2 l cos(\theta) sin(\theta)\dot{\theta}^2 + u cos(\theta) + (m_1 + m_2) g sin(\theta)}{l(m_1 + m_2(1 - cos^2(\theta)))}
\end{aligned} \tag{33}$$

where m_1, m_2, l, g represents the mass of the cart, mass of the pole, length of the pole and acceleration due to gravity respectively. The values for the parameters are listed in Appendix A. The state vector of the system are $\mathbf{x} = [x, v, \theta, \omega]$, which are the cart position, cart velocity, pole angle and pole angular velocity respectively and input to the system is u, which represents the force acting on the cart. The constraints acting on the system are

$$\begin{aligned}
-x_{max} \leq x \leq x_{max} \\
-u_{max} \leq u \leq u_{max}
\end{aligned} \tag{34}$$

where x_{max} and u_{max} represents the maximum bounds of the cart position and cart force respectively. The OCP for the system is given as

$$\begin{aligned}
\min_{\mathbf{x}(.),\,\mathbf{p}(.)} \quad & \mathbf{x}(t_f)^T Q_f \mathbf{x}(t_f) + \int_0^{t_f} (\mathbf{x}^T Q \mathbf{x} + u^T R u)\, dt \\
\text{subject to} \quad & (33),\ (34),\ \mathbf{x}(0) = \{[0, 0, \pi, 0], [0, 0, \frac{\pi}{2}, 0]\} \\
& u(\mathbf{p}, \mathbf{x}) = p_1 x + p_2 v + p_3 \theta + p_4 \omega + p_5
\end{aligned} \tag{35}$$

where t_f, Q_f, Q, R and T_s represents the look ahead period, quadratic terminal state cost, quadratic stage state cost, quadratic input cost and sampling period respectively. The input parameterization is an affine state feedback policy where the parameterization vector is $\mathbf{p} = [p_1, p_2, p_3, p_4, p_5]$ which is modeled a constant over the horizon. Once the OCP is solved for $\mathbf{p}^*$, the input $\mathbf{u}(\mathbf{p}^*, \mathbf{x}(0))$ is injected into the system over the period T_s and this process is repeated in a receding horizon fashion. To compare against the ACADO controller, an un-parameterized version of the NMPC problem (35) was implemented with the following settings - **Integrator** - 4th order Implicit Runge Kutta integrator, **QP solver** - qpOASES, **Hessian approximation** - Gauss-Newton, **Discretization** - Multiple shooting, **Discretization intervals** - 30 and for the rest, default parameters were utilized. The study was conducted in two parts with two initial conditions which are $\mathbf{x}(0) = [0, 0, \frac{\pi}{2}, 0]$ and $\mathbf{x}(0) = [0, 0, \pi, 0]$.

Fig. 4 illustrates the cart force, cart position and the pole angle of the system for pNMPC and ACADO controller for the first case respectively. The computation time of the pNMPC

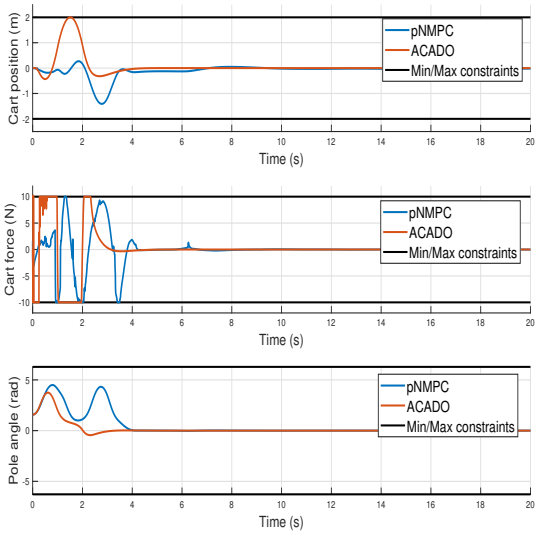

Fig. 4: Cart position, cart force and the pole angle of the system for initial condition $\mathbf{x}(0) = [0, 0, \frac{\pi}{2}, 0]$

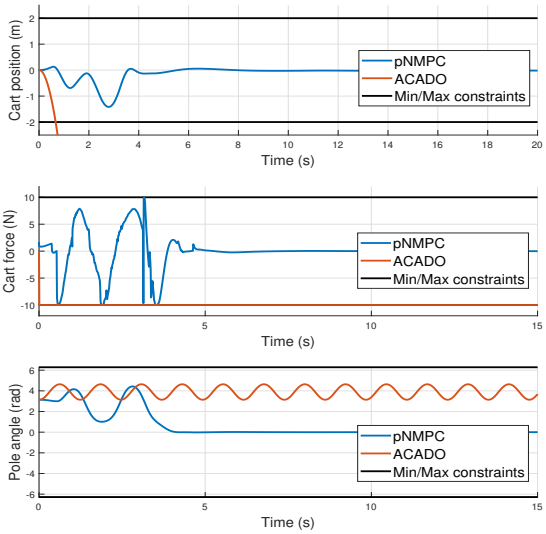

Fig. 6: Cart position, cart force and the pole angle of the system for initial condition $\mathbf{x}(0) = [0, 0, \pi, 0]$

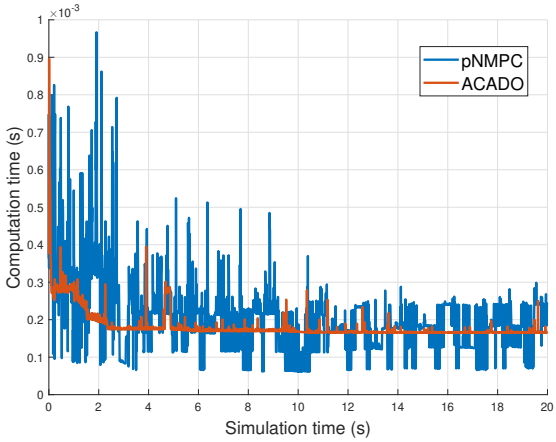

Fig. 5: Cart-pole system computation time

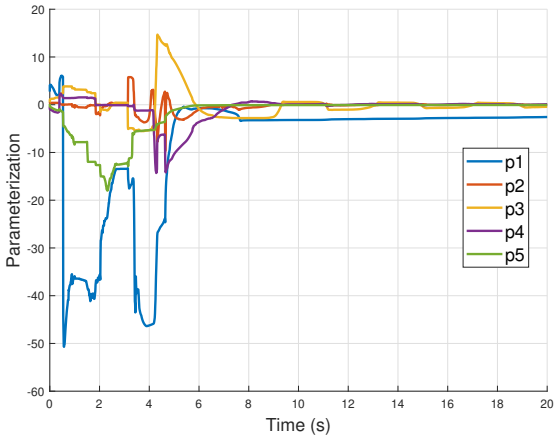

Fig. 7: Parameterization p_1, p_2, p_3, p_4, p_5

and ACADO controller for the first case is plotted in Fig. 5. The mean computation time of ACADO hovers around 185.57 ms and 205.05 μs for the pNMPC controller.

Fig. 6 illustrates the cart force, cart position and the pole angle of the system for pNMPC and ACADO controller for the second case respectively. From the plots, it is evident that the ACADO controller crashes, however, despite the numerical ill-conditioning of model, the pNMPC controller fares well with state feedback parameterization and at the same time, the system is stabilized. Fig. 7 illustrates the parameterization values for the second case.

B. PVTOL stabilization problem

The task of the PVTOL (planar vertical takeoff and landing aircraft) [30] stabilization problem is to regulate the states of the system to the origin given an initial perturbation. The nonlinear state space equations of the system are given below

$$\begin{aligned} \dot{y} &= v_y \\ \dot{v}_y &= \psi_y^{BB}(\mathbf{x}, \mathbf{u}, \mathbf{p}) \\ \dot{z} &= v_z \\ \dot{v}_z &= \psi_z^{BB}(\mathbf{x}, \mathbf{u}, \mathbf{p}) \\ \dot{\theta} &= \omega \\ \dot{\omega} &= u_2 \end{aligned} \tag{36}$$

where, the state vector is $\mathbf{x} = [y, v_y, z, v_z, \theta, \dot{\theta}]$ which represents the vertical position, vertical velocity, horizontal position, horizontal velocity, roll angle and roll rate respectively, the input vector is $\mathbf{u} = [u_1, u_2]$ which represents the lift acceleration and angular acceleration respectively and the parameter vector is $\mathbf{p} = [\sigma]$ which represents the coupling between roll and lift effects. The variables $\psi_y^{BB}(\mathbf{x}, \mathbf{u}, \mathbf{p}) = -u_1 sin(\theta) + \sigma u_2 cos(\theta)$

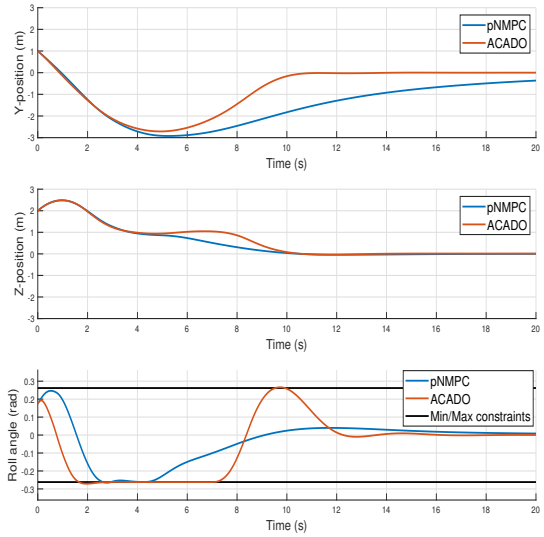

Fig. 8: PVTOL Y-position, Z-position, Roll angle (θ)

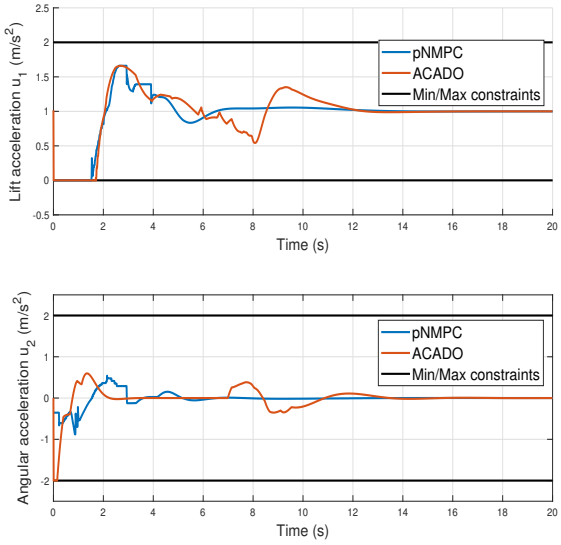

Fig. 9: PVTOL inputs (u_1, u_2)

and $\psi_z^{BB}(\mathbf{x}, \mathbf{u}, \mathbf{p}) = u_1 cos(\theta) + \sigma u_2 sin(\theta) - 1$ represents the dynamics of the system which are deliberately modeled as a black-box model and invoked using the function calls `"ModelY"` and `"ModelZ"` respectively. The input arguments for these functions are the state vector, input vector and parameter vector and the output is the respective dynamics of system. The source codes for these functions were compiled to a dynamic library file and linked during the compilation process of the pNMPC controller. This example serves as an use case of the pNMPC S/W, where external black box models can be linked with the S/W's inbuilt symbolic variables.

The constraints acting on the system are

$$\begin{aligned} 0 \leq u_1 &\leq u_1^{max} \\ -u_2^{max} \leq u_2 &\leq u_2^{max} \\ -\theta^{max} \leq \theta &\leq \theta^{max} \end{aligned} \tag{37}$$

where $u_1^{max}, u_2^{max}, \theta^{max}$ represents the maximum bounds over the inputs u_1 and u_2 and the roll angle state θ. The OCP for the stabilization problem is defined as

$$\begin{aligned} \min_{\mathbf{x}(.),\,\mathbf{p}(.)} \quad & \mathbf{x}(t_f)^T Q_f \mathbf{x}(t_f) + \int_0^{t_f} (\mathbf{x}^T Q \mathbf{x} + \mathbf{u}^T R \mathbf{u})\, dt \\ \text{subject to} \quad & (36),\ (37),\ \mathbf{x}(0) = [1, -1, 2, 1, \tfrac{-14\pi}{10}, -0.1] \\ & \mathbf{u}(\mathbf{p}) = \mathbf{p} \end{aligned} \tag{38}$$

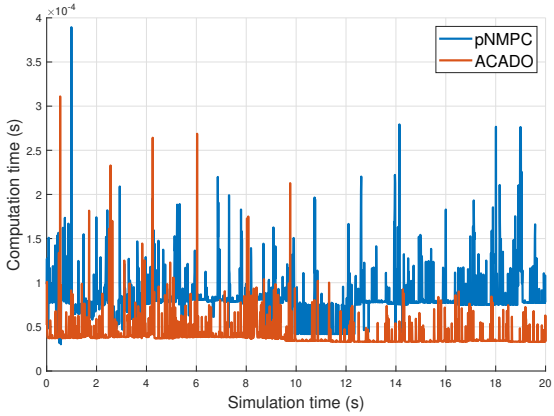

Fig. 10: PVTOL computation time

where t_f, Q, R and T_s represents the look ahead period, quadratic stage state cost, input cost and sampling period respectively. The input parameterization is $\mathbf{p} = [p_1, p_2]$ where each parameter has two control points placed equidistantly over the prediction horizon and follows a linear profile. Once the OCP is solved for $\mathbf{p}^*$, the input $u(\mathbf{p}^*)$ is injected into the system over the period T_s and this process is repeated in a receding horizon fashion. The following setting was chosen for the ACADO controller, **Integrator** - 4th order Runge Kutta integrator, **QP solver** - qpOASES, **Hessian approximation** - Gauss-Newton, **Discretization** - Multiple shooting, **Discretization intervals** - 5 and for the rest, default parameters were utilized. Also, for simulation of ACADO controller, the whole dynamics of the system was presumed to known already as ACADO toolkit has no feature to incorporate external function calls into it's symbolic framework.

Fig. 8 and Fig. 9 illustrates the Y-position, Z-position and roll angle (θ) and the inputs u_1 and u_2 of the system for the pNMPC and ACADO NMPC controller respectively. The computation time of the pNMPC and ACADO controller is plotted in Fig. 10. The mean computation time of ACADO hovers around 41.50 μs and 44.7 μs for the pNMPC controller. The comparison highlights the fact that in-spite of incorporating black box models, the performance and computation time is nearly on par with the ACADO toolkit.

VII. Future works and conclusions

The crux of this paper is to present a derivative free pNMPC code generation S/W and to validate its performance by means of simulation on multiple examples. From the simulation study conducted, it is certainly evident that the proposed S/W has applications for several engineering systems, where the model exists either as computer codes or as a data driven model. Despite the S/W is suffice for several real world applications, there are certain directions for improvement. The future line of work can be stratified into three parts a) Technical challenges, b) Hardware challenges and c) Software challenges. A detailed examination of the aforementioned division of work is expounded below

A. Technical challenges

1) Smart parameterization: As mentioned in the Section I and to re-emphasize the fact again that the potential of the method solely depends upon the parameterization technique adopted by the control engineer. However, in cases when this becomes intricate in nature, the onus of determining an optimal parameterization is substantially increased. In such a situation, it would be remiss of not utilizing tools from the machine learning community. Methods developed in reinforcement learning (RL) [31] zones in parallel to the proposed approach and by availing tools and methods from RL community a smart parameterization technique for pNMPC controller can be developed.

2) Implicit solvers: As of now, the proposed S/W provides only explicit ODE solvers for the pNMPC controller. However, in many applications where the system is intrinsically stiff in nature, one is obliged to use implicit solvers for numerical stability. The future version of the S/W would encompass support for several implicit solvers.

3) Equality constraints: In the future, the S/W would include support for equality constraints, which is important for control of periodic systems or to enforce time point constraints on the system.

4) PDEs, DAEs and Hybrid systems: In the future, the S/W would include support for control of partial differential equations (PDEs), differential algebraic equations (DAEs) and hybrid systems. The overarching goal is to widen the scope of the S/W and provide a one-stop shop pNMPC solution.

B. Hardware challenges

The SQP-BBO optimization module presented in Section III clearly provides room for implementation on multi-core processors. As for the multi-variate case of the SQP-BBO method, several threads can be spawned in parallel for every component of the parameterization vector. This would downsize the computation time for large parameterization vectors. In the future, the method would be extended to GPUs and CPU multi-core processors by using the CUDA/OpenMP frameworks.

C. S/W challenges

The current S/W provides a primitive interface to MATLAB/Simulink. In the future, it is planned to provide better interfaces to MATLAB/Simulink, Python and Julia to benefit all the embedded programmers across the board.

Appendix A
pNMPC OCP C++ code for cart-pole swing up problem

```
#include "pNMPC_headers.hpp"
#define PI 3.1416
using namespace pNMPC;
int main()
{
// States
States p, theta, pd, thetad;
// Inputs
Inputs p1, p2, p3, p4, p5;
// Constant parameters
Real m1 = 1, m2 = 0.1;
Real g = 9.81, l = 0.5;
// Bounds
Real pmax = 2, umax = 10;
Real thetamax = 2*PI;
// OCP data
MATHyperStates Xs(4,1);
Xs = {p,theta,pd,thetad};
MATReal Q = diag({5,10,1,1});
MATReal Qf = diag({10,20,1,1});
Real R = 0.1;

// Input parameterization
BEGIN_PARAMETERIZATION_MAP
 ParameterizationMap u = p1*p +
 p2*theta + p3*pd + p4*thetad + p5;
END_PARAMETERIZATION_MAP

// Differential equations
BEGIN_DIFFERENTIAL
 DiffEquation d1 = pd;
 DiffEquation d2 = thetad;
 DiffEquation d3 =
 (-l*m2*sin(theta)*((thetad)^2)
 +u+m2*g*cos(theta)*sin(theta))/
 (m1+m2*(1-(cos(theta))^2));
 DiffEquation d4 =
 (-l*m2*cos(theta)*sin(theta)
 *((thetad)^2)+ u*cos(theta)+
 (m1+m2)*g*sin(theta))/
 (l*m1+l*m2*(1-(cos(theta))^2));
END_DIFFERENTIAL

// Constraints
BEGIN_CONSTRAINTS
 BEGIN_REGULAR_CONSTRAINTS
  ScalarConstraint G1 =
  {-umax <= u <= umax};
```

```
  ScalarConstraint G2 =
  {-pmax <= p <= pmax};
  ScalarConstraint G3 =
  {-thetamax <= theta <= thetamax};
 END_REGULAR_CONSTRAINTS
END_CONSTRAINTS

// Lagrangian and Mayer Cost
BEGIN_OBJECTIVES
// Lagrangian Cost
 BEGIN_LAGRANGIAN
  ScalarObjective LC1 =
  {transpose(Xs)*Q*(Xs)};
  ScalarObjective LC2 =
  {R*((u)^2)};
 END_LAGRANGIAN
// Mayer Cost
 BEGIN_MAYER
  ScalarObjective MC1 =
  {transpose(Xs)*Qf*(Xs)};
 END_MAYER
END_OBJECTIVES

// Input parameterization
ControlParamZ<Linear>{1,p1};
ControlParamZ<Linear>{1,p2};
ControlParamZ<Linear>{1,p3};
ControlParamZ<Linear>{1,p4};
ControlParamZ<Linear>{1,p5};

// PNMPCGEN singleton object
PNMPCGEN* pNMPC = PNMPCGEN::getSton();
pNMPC->setInitialTime(0);
pNMPC->setFinalTime(1);
pNMPC->setStepSize(0.05);
pNMPC->getSolver()->setNiter(4);
pNMPC->setConstForm(CONST_FORM::FORM_1);
pNMPC->setIntegrator(INTEGRATOR::RK45);

// Generate C code
pNMPC->genCCode();

// Free allocated memory
pNMPC_free();
return 0;
}
```

Appendix B
pNMPC OCP C++ code for PVTOL stabilization problem

```
#include "pNMPC_headers.hpp"
using namespace pNMPC;
int main()
{
// States
States x1, x2, x3, x4, x5, x6;
// Inputs
Inputs p1, p2;
// External variables
External psi_y = "ModelY";
External psi_z = "ModelZ";
// Constant parameters
Real a_l = 0, a_u = 2;
Real alp_l = -2, alp_u = 2;
// OCP data
MATHyperStates Xs(4,1), Us(2,1);
Xs = {x1,x2,x3,x4,x5,x6};
MATReal Q = 5*eye(6);
MATReal Qf = 10*eye(6);
Real R = 0.01*eye(2);

// Input parameterization
BEGIN_PARAMETERIZATION_MAP
 ParameterizationMap a = p1;
 ParameterizationMap alp = p2;
END_PARAMETERIZATION_MAP
Us = {a,alp};

// Differential equations
BEGIN_DIFFERENTIAL
 DiffEquation d1 = x2;
 DiffEquation d2 = psi_y;
 DiffEquation d3 = x4;
 DiffEquation d4 = psi_z;
 DiffEquation d5 = x6;
 DiffEquation d6 = alp;
END_DIFFERENTIAL

// Constraints
BEGIN_CONSTRAINTS
 BEGIN_REGULAR_CONSTRAINTS
  ScalarConstraint G1 =
  {a_l <= a <= a_u};
  ScalarConstraint G2 =
  {alp_l <= alp <= alp_u};
  ScalarConstraint G3 =
  {-15*PI/180 <= x5 <= 15*PI/180};
 END_REGULAR_CONSTRAINTS
END_CONSTRAINTS

// Lagrangian and Mayer Cost
BEGIN_OBJECTIVES
// Lagrangian Cost
 BEGIN_LAGRANGIAN
  ScalarObjective LC1 =
  {transpose(Xs)*Q*(Xs)};
  ScalarObjective LC2 =
  {transpose(Us)*R*(Us)};
 END_LAGRANGIAN
// Mayer Cost
 BEGIN_MAYER
  ScalarObjective MC1 =
  {transpose(Xs)*Qf*(Xs)};
 END_MAYER
END_OBJECTIVES
```

```
// Input parameterization
ControlParamZ<Linear>{2,p1,a_l,a_u};
ControlParamZ<Linear>{2,p2,alp_l,alp_u};

// PNMPCGEN singleton object
PNMPCGEN* pNMPC = PNMPCGEN::getSton();
pNMPC->setInitialTime(0);
pNMPC->setFinalTime(2);
pNMPC->setStepSize(0.1);
pNMPC->getSolver()->setNiter(4);
pNMPC->setConstForm(CONST_FORM::FORM_1);
pNMPC->setIntegrator(INTEGRATOR::RK45);

// Generate C code
pNMPC->genCCode();

// Free allocated memory
pNMPC_free();
return 0;
}
```

REFERENCES

[1] S. V. Raković and W. S. Levine, *Handbook of model predictive control*. Springer, 2018.

[2] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos, "The explicit linear quadratic regulator for constrained systems," *Automatica*, vol. 38, no. 1, pp. 3–20, 2002.

[3] J. B. Rawlings, E. S. Meadows, and K. R. Muske, "Nonlinear model predictive control: A tutorial and survey," *IFAC Proceedings Volumes*, vol. 27, no. 2, pp. 185–197, 1994.

[4] A. Mesbah, "Stochastic model predictive control: An overview and perspectives for future research," *IEEE Control Systems Magazine*, vol. 36, no. 6, pp. 30–44, 2016.

[5] D. Q. Mayne, M. M. Seron, and S. Raković, "Robust model predictive control of constrained linear systems with bounded disturbances," *Automatica*, vol. 41, no. 2, pp. 219–224, 2005.

[6] A. Aswani, H. Gonzalez, S. S. Sastry, and C. Tomlin, "Provably safe and robust learning-based model predictive control," *Automatica*, vol. 49, no. 5, pp. 1216–1226, 2013.

[7] M. Ellis, H. Durand, and P. D. Christofides, "A tutorial review of economic model predictive control methods," *Journal of Process Control*, vol. 24, no. 8, pp. 1156–1178, 2014.

[8] M. Bujarbaruah, X. Zhang, U. Rosolia, and F. Borrelli, "Adaptive mpc for iterative tasks," in *2018 IEEE Conference on Decision and Control (CDC)*. IEEE, 2018, pp. 6322–6327.

[9] T. Blochwitz, M. Otter, M. Arnold, C. Bausch, H. Elmqvist, A. Junghanns, J. Mauß, M. Monteiro, T. Neidhold, D. Neumerkel *et al.*, "The functional mockup interface for tool independent exchange of simulation models," in *Proceedings of the 8th International Modelica Conference; March 20th-22nd; Technical Univeristy; Dresden; Germany*, no. 063. Linköping University Electronic Press, 2011, pp. 105–114.

[10] "EMPHYSIS: Embedded systems with physical models in the production code software," https://itea3.org/project/emphysis.html.

[11] M. Alamir, *Stabilization of nonlinear systems using receding-horizon control schemes: a parametrized approach for fast systems*. Springer, 2006, vol. 339.

[12] B. Houska, H. J. Ferreau, and M. Diehl, "An auto-generated real-time iteration algorithm for nonlinear mpc in the microsecond range," *Automatica*, vol. 47, no. 10, pp. 2279–2285, 2011.

[13] 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.

[14] J. Kalmari, J. Backman, and A. Visala, "A toolkit for nonlinear model predictive control using gradient projection and code generation," *Control Engineering Practice*, vol. 39, pp. 56–66, 2015.

[15] F. Ullmann, "Fiordos: A matlab toolbox for c-code generation for first order methods," *MS thesis*, 2011.

[16] J. Mattingley and S. Boyd, "Cvxgen: A code generator for embedded convex optimization," *Optimization and Engineering*, vol. 13, no. 1, pp. 1–27, 2012.

[17] A. Zanelli, A. Domahidi, J. Jerez, and M. Morari, "Forces nlp: an efficient implementation of interior-point methods for multistage nonlinear nonconvex programs," *International Journal of Control*, vol. 93, no. 1, pp. 13–29, 2020.

[18] I. Necoara and A. Patrascu, "Duquad: an inexact (augmented) dual first order algorithm for quadratic programming," *arXiv preprint arXiv:1504.05708*, 2015.

[19] T. Englert, A. Völz, F. Mesmer, S. Rhein, and K. Graichen, "A software framework for embedded nonlinear model predictive control using a gradient-based augmented lagrangian approach (grampc)," *Optimization and Engineering*, vol. 20, no. 3, pp. 769–809, 2019.

[20] G. Banjac, B. Stellato, N. Moehle, P. Goulart, A. Bemporad, and S. Boyd, "Embedded code generation using the OSQP solver," in *IEEE Conference on Decision and Control (CDC)*, 2017. [Online]. Available: https://doi.org/10.1109/CDC.2017.8263928

[21] H. A. Shukla, B. Khusainov, E. C. Kerrigan, and C. N. Jones, "Software and hardware code generation for predictive control using splitting methods," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 14 386–14 391, 2017.

[22] M. Kvasnica, P. Grieder, M. Baotić, and M. Morari, "Multi-parametric toolbox (mpt)," in *International Workshop on Hybrid Systems: Computation and Control*. Springer, 2004, pp. 448–462.

[23] H. J. Ferreau, S. Almér, R. Verschueren, M. Diehl, D. Frick, A. Domahidi, J. L. Jerez, G. Stathopoulos, and C. Jones, "Embedded optimization methods for industrial automatic control," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 13 194–13 209, 2017.

[24] M. Alamir, "The pdf-mpc package: A free-matlab-coder package for real-time nonlinear model predictive control," *arXiv preprint arXiv:1703.08255*, 2017.

[25] K. M. M. Rathai, "pNMPC: A code generation tool for implementation of pnmpc controller for embedded control systems," https://github.com/Kartz4code/pNMPC_CODEGEN, 2020, [Online; accessed 22-June-2020].

[26] M. Alamir, *A pragmatic story of model predictive control: self-contained algorithms and case-studies*. CreateSpace Independent Publishing Platform, 2013.

[27] ——, "A framework for real-time implementation of low-dimensional parameterized nmpc," *Automatica*, vol. 48, no. 1, pp. 198–204, 2012.

[28] A. V. Aho, R. Sethi, and J. D. Ullman, "Compilers, principles, techniques," *Addison wesley*, vol. 7, no. 8, p. 9, 1986.

[29] A. Mills, A. Wills, and B. Ninness, "Nonlinear model predictive control of an inverted pendulum," in *2009 American control conference*. IEEE, 2009, pp. 2335–2340.

[30] P. Martin, S. Devasia, and B. Paden, "A different look at output tracking: control of a vtol aircraft," *Automatica*, vol. 32, no. 1, pp. 101–107, 1996.

[31] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.