A Lyapunov-based Approach to Nonlinear Programming and Its Application to Nonlinear Model Predictive Torque Control

Kyunghwan Choi and Christoph M. Hackl, *Senior Member*

***Abstract*—A tuning-parameter-free and matrix-inversion-free solution of nonlinear programming (NLP) problems is proposed. The key idea is to design an update law based on Lyapunov analysis to satisfy the first-order necessary conditions for optimality. To this aim, first, the Lyapunov function is defined as the summation of the norms of these conditions. Then, the desired optimization variables and Lagrange multipliers, which minimize the Lyapunov function the most, are found analytically, thereby rapidly approaching the necessary conditions. The proposed method neither requires tuning parameters nor matrix inversions; thus, it can be implemented easily with less iterations and computational load than conventional methods, such as sequential quadratic programming (SQP) and augmented Lagrangian method (ALM). The effectiveness of the proposed method is applied to and validated by using it to solve a nonlinear model predictive torque control (NMPTC) problem in electrical drives. The results are compared with those of SQP and ALM.**

***Index Terms*—Lyapunov analysis, necessary conditions for optimality, matrix-inversion-free, nonlinear model predictive control (NMPC), nonlinear programming (NLP), tuning-parameter-free**

[CH: I have commented out the notation section; we must not exceed 8 pages! Or, is this wrong?]

I. Introduction

Nonlinear programming (NLP) has been used in various applications as a powerful tool for optimizing the performance of (dynamical) systems [?], [?]. However, solving NLP problems is still challenging because of the lack of a general analytical solution and the difficulty in designing effective numerical optimization processes [?].

Typical numerical approaches for NLP are sequential quadratic programming (SQP) and augmented Lagrangian method (ALM) [?]. SQP solves an NLP problem effectively when the NLP can be approximated as QPs and the iteration starts near the optimal solution. However, SQP is computationally expensive, especially for large-scale problems, because it involves the inversion of the Karush-Kuhn-Tucker (KKT) matrix [?]. In addition, two typical methods for SQP to handle inequality constraints, the interior point method and active-set method, may require a heavy computation at each iteration and numerous iterations, respectively, when the NLP includes many inequality constraints [?].

In contrast to SQP, ALM provides a simple but effective way to handle inequality constraints by adding penalty terms for the constraint violations to the objective function and by finding the penalty terms' weights (i.e., Lagrange multipliers) with a simple update law [?]. However, ALM uses multiple tuning parameters regarding handling the constraints, such as the barrier parameter and the constraint violation tolerance. Finding appropriate values for these parameters can be challenging and may require problem-specific tuning. In addition, ALM typically involves either the inversion of the Hessian or the approximation of the inverted Hessian, which is usually computationally demanding [?].

Besides SQP and ALM, most numerical approaches require (i) computationally demanding steps (such as the inversion of the KKT matrix or the Hessian or its approximation), (ii) tuning of multiple parameters (often problem-specific) and (iii) a large number of iterations until the optimization reaches a satisfactory solution. Therefore, this study presents a tuning-parameter-free and matrix-inversion-free numerical optimization method to solve NLPs. To this aim, a control perspective is adopted that interprets the numerical optimization process as a dynamical system such that typical control principles can be adopted to design an update law without introducing tuning parameters and utilizing matrix inversions. The key idea is to design an update law based on Lyapunov's second method to meet the two first-order necessary conditions for optimality [?]. The Lyapunov function is defined as the summation of the norms of these two conditions. The desired optimization variables and Lagrange multipliers (which minimize the Lyapunov function) are analytically found which allows and guarantees to approach the necessary conditions rapidly.

The proposed method is called Lyapunov-based Nonlinear Programming (LBNLP). It can be implemented easily and with less iterations and computational load than conventional methods, such as SQP and ALM, due to its tuning-parameter-freeness and matrix-inversion-freeness; thus, it is particularly interesting and effective for nonlinear model predictive control

This work was supported by Korea Research Institute for Defense Technology planning and advancement (KRIT) grant funded by Korea government DAPA (Defense Acquisition Program Administration) (No. KRIT-CT-22-087, Synthetic Battlefield Environment based Integrated Combat Training Platform).

Kyunghwan Choi is with the School of Mechanical Engineering, Gwangju Institute of Science and Technology, Gwangju 61005, South Korea (e-mail: khchoi@gist.ac.kr).

Christoph M. Hackl is with the Hochschule München (HM) University of Applied Sciences, Laboratory of Mechatronic and Renewable Energy Systems (LMRES), Munich, 80335, Germany (e-mail: christoph.hackl@hm.edu).

(NMPC), which requires an NLP problem to be solved in real time within a short control period. The proposed method also allows the violation of constraints during the iteration process as ALM does, which is another desirable feature for NMPC with many inequality constraints. The update law of the proposed method can be directly used as the control law for NMPC because each iteration is at least suboptimal and converges towards the local solution rapidly.

Previous studies, including [?], have already explored control perspectives on numerical optimization, introducing various update laws. Nevertheless, the majority of these update laws were designed to address unconstrained optimization or relatively straightforward constrained optimization scenarios, such as convex problems. To the authors' knowledge, using a Lyapunov-based approach to solve an NLP and to implement NMPC has not been investigated yet.

II. Preliminaries

This section provides preliminaries for this study: Section II-A defines a general formulation of NLPs, whereas Section II-B states the necessary conditions for optimality of NLPs. Section II-C describes SQP and ALM in more detail.

A. Nonlinear programming (NLP)

A general scalar formulation of NLP is given by

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) \tag{1a}$$

$$\text{subject to}$$

$$c_j^{\text{eq}}(\boldsymbol{x}) = 0,\ j \in \mathbb{E} \subset \mathbb{N}, \tag{1b}$$

$$c_i^{\text{in}}(\boldsymbol{x}) \leq 0,\ i \in \mathbb{I} \subset \mathbb{N}, \tag{1c}$$

where $f\colon \mathbb{R}^n \to \mathbb{R}$, $c_j^{\text{eq}}\colon \mathbb{R}^n \to \mathbb{R}$ and $c_i^{\text{in}}\colon \mathbb{R}^n \to \mathbb{R}$ are (at least) continuously differentiable and represent the objective function, the equality constraints for all $j \in \mathbb{E}$ (with dimension $n_{\text{eq}} := |\mathbb{E}|$) and the inequality constraints for all $i \in \mathbb{I}$ (with dimension $n_{\text{in}} := |\mathbb{I}|$), respectively. $\mathbb{I}$ and $\mathbb{E}$ are two finite sets of indices for the equality and inequality constraints, respectively. The vector $\boldsymbol{x} \in \mathbb{R}^n$ comprises the optimization variables. The goal is to find the optimal $\boldsymbol{x}^\star := \arg\min_{\boldsymbol{x}} f(\boldsymbol{x})$ subject to the constraints in (1). At least one of the functions in (1) must be nonlinear to obtain a nonlinear optimization problem.

B. Necessary Conditions for Optimality

The Lagrangian for the NLP in (1) is defined as

$$\begin{aligned} L(\boldsymbol{x}, \boldsymbol{\lambda}) &:= f(\boldsymbol{x}) + \sum_{j\in\mathbb{E}} \lambda_j^{\text{eq}} c_j^{\text{eq}}(\boldsymbol{x}) + \sum_{i\in\mathbb{I}} \lambda_i^{\text{in}} c_i^{\text{in}}(\boldsymbol{x}) \\ &= f(\boldsymbol{x}) + \boldsymbol{\lambda}^\top \boldsymbol{c}(\boldsymbol{x}) \end{aligned} \tag{2}$$

where λ_j^{eq} and λ_j^{in} are the Lagrange multipliers for the equality and inequality constraints, respectively. For compactness all multipliers and constraints are collected in the Lagrangian multiplier vector $\boldsymbol{\lambda} := (\lambda_1, \dots, \lambda_{n_\text{c}})^\top := (\lambda_1^{\text{eq}}, \dots, \lambda_1^{\text{in}}, \dots)^\top \in \mathbb{R}^{n_\text{c}}$ and the constraint vector $\boldsymbol{c} := (c_1, \dots, c_{n_\text{c}})^\top := (c_1^{\text{eq}}, \dots, c_1^{\text{in}}, \dots)^\top \in \mathbb{R}^{n_\text{c}}$ where

$$n_\text{c} := n_{\text{eq}} + n_{\text{in}}$$

is the overall dimension of the constraints. For later, the active set $\mathbb{A}(\boldsymbol{x})$ at any feasible $\boldsymbol{x}$ is introduced, which is the union of the set $\mathbb{E}$ and those indices of the active inequality constraints, i.e.

$$\mathbb{A}(\boldsymbol{x}) = \mathbb{E} \cup \left\{ a \in \mathbb{I} \mid c_a^{\text{in}}(\boldsymbol{x}) = 0 \right\}. \tag{3}$$

One constraint qualification for the necessary conditions is defined as follows:

Definition 1 (Linear independence constraint qualification (LICQ) [?]). *Given a feasible point $\boldsymbol{x}$ and the active set $\mathbb{A}(\boldsymbol{x})$, the linear independence constraint qualification (LICQ) holds if the set of active constraint gradients $\{\nabla_{\boldsymbol{x}} c_a(\boldsymbol{x}) \mid a \in \mathbb{A}(\boldsymbol{x})\}$ is linearly independent.*

The first-order necessary conditions for optimality, which provide the foundation for many numerical algorithms for NLP, are defined as follows.

Theorem 1 (First-Order Necessary Conditions [?]). *Suppose that $\boldsymbol{x}^\star$ is a local solution of (1) and that the LICQ holds at $\boldsymbol{x}^\star$. Then there is a Lagrange multiplier vector $\boldsymbol{\lambda}^\star := (\lambda_1^\star, \dots, \lambda_{n_\text{c}}^\star)^\top \in \mathbb{R}^{n_\text{c}}$, such that the following conditions are satisfied at $(\boldsymbol{x}^\star, \boldsymbol{\lambda}^\star)$:*

$$\nabla_x L(\boldsymbol{x}^\star, \boldsymbol{\lambda}^\star) = 0, \tag{4a}$$

$$c_j^{\text{eq}}(\boldsymbol{x}^\star) = 0, \textit{ for all } j \in \mathbb{E}, \tag{4b}$$

$$c_i^{\text{in}}(\boldsymbol{x}^\star) \leq 0, \textit{ for all } i \in \mathbb{I}, \tag{4c}$$

$$\lambda_i^{\text{in},\star} := \lambda_{i+n_{\text{eq}}}^\star \geq 0, \textit{ for all } i \in \mathbb{I}, \tag{4d}$$

$$\lambda_l^\star c_l(\boldsymbol{x}^\star) = 0, \textit{ for all } l \in \mathbb{E} \cup \mathbb{I}. \tag{4e}$$

The conditions (4) are known as the KKT conditions. Condition (4e) implies that the Lagrange multipliers corresponding to inactive inequality constraints are zero; thus, the first condition (4a) can be rewritten by omitting the terms for indices $l \notin \mathbb{A}(\boldsymbol{x}^\star)$ as follows

$$0 = \nabla_{\boldsymbol{x}} L(\boldsymbol{x}^\star, \boldsymbol{\lambda}^\star) = \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^\star) + \sum_{a\in\mathbb{A}(\boldsymbol{x}^\star)} \lambda_a^\star \nabla_{\boldsymbol{x}} c_a(\boldsymbol{x}^\star). \tag{5}$$

C. Typical Numerical Approaches to NLP

SQP and ALM are two typical numerical approaches which are described next to compare those to the proposed method.

1) SQP: SQP is an iterative method for NLP, solving a sequence of optimization subproblems, each of which is an approximation of the NLP as a quadratic programming (QP). Each subproblem is defined as follows [?]:

$$\begin{aligned} \min_{\Delta\boldsymbol{x}[k]} \ & f(\boldsymbol{x}[k]) + \nabla_{\boldsymbol{x}} f(\boldsymbol{x}[k])^\top \Delta\boldsymbol{x}[k] + \\ & + \tfrac{1}{2}(\Delta\boldsymbol{x}[k])^\top \boldsymbol{H}_L(\boldsymbol{x}[k], \boldsymbol{\lambda}[k]) \Delta\boldsymbol{x}[k] \end{aligned} \tag{6a}$$

$$\text{subject to}$$

$$\nabla_{\boldsymbol{x}} c_j(\boldsymbol{x}[k])^\top \Delta\boldsymbol{x}[k] + c_j(\boldsymbol{x}[k]) = 0,\ j \in \mathbb{E}, \tag{6b}$$

$$\nabla_{\boldsymbol{x}} c_i(\boldsymbol{x}[k])^\top \Delta\boldsymbol{x}[k] + c_i(\boldsymbol{x}[k]) \geq 0,\ i \in \mathbb{I}, \tag{6c}$$

where $\Delta\boldsymbol{x}[k] := \boldsymbol{x}[k+1] - \boldsymbol{x}[k]$ (with actual iteration step $k \in \mathbb{N}$) and $\boldsymbol{H}_L(\boldsymbol{x}, \boldsymbol{\lambda}) := \nabla_{\boldsymbol{x}}^2 L(\boldsymbol{x}, \boldsymbol{\lambda})$ denotes the Hessian of the Lagrangian with respect to $\boldsymbol{x}$. As the Hessian may neither be easy to compute nor is always positive definite within the admissible set, alternate choices for $\boldsymbol{H}_L$, such as full quasi-newton approximations and reduced-hessian approximations, can be used instead [?].

The solution of (6) can be obtained by solving

$$\underbrace{\begin{bmatrix} \boldsymbol{H}_L(\boldsymbol{x}[k],\boldsymbol{\lambda}[k]) & \boldsymbol{C}_{\mathbb{A}}(\boldsymbol{x}[k]) \\ \boldsymbol{C}_{\mathbb{A}}^\top(\boldsymbol{x}[k]) & 0 \end{bmatrix}}_{=:\boldsymbol{K}(\boldsymbol{x}[k],\boldsymbol{\lambda}[k])\in\mathbb{R}^{(n+m)\times(n+m)}} \begin{bmatrix} \Delta\boldsymbol{x}[k] \\ \Delta\boldsymbol{\lambda}[k] \end{bmatrix} = \begin{bmatrix} -\nabla_{\boldsymbol{x}} L(\boldsymbol{x}[k],\boldsymbol{\lambda}[k]) \\ -\boldsymbol{c}_{\mathbb{A}}(\boldsymbol{x}[k]) \end{bmatrix}, \tag{7}$$

for $\left(\Delta\boldsymbol{x}[k]^\top, \Delta\boldsymbol{\lambda}[k]^\top\right)^\top$ where $\Delta\boldsymbol{\lambda}[k] := \boldsymbol{\lambda}[k+1] - \boldsymbol{\lambda}[k]$, $\boldsymbol{C}_{\mathbb{A}}^\top(\boldsymbol{x}) := [\nabla_{\boldsymbol{x}} c_a(\boldsymbol{x})]_{a\in\mathbb{A}(\boldsymbol{x})}^\top \in \mathbb{R}^{m\times n}$, $\boldsymbol{c}_{\mathbb{A}}(\boldsymbol{x}) := [c_a(\boldsymbol{x})]_{a\in\mathbb{A}(\boldsymbol{x})} \in \mathbb{R}^m$ and $m \le n_{\mathrm{c}}$ is the number of active constraints. Each iteration k is well-defined when the non-singularity of the KKT matrix $\boldsymbol{K}(\boldsymbol{x}[k],\boldsymbol{\lambda}[k])$ holds, which is a consequence of LICQ and the positive-definiteness of the Hessian.

SQP solves the NLP effectively when each subproblem approximates the NLP reasonably well. However, solving (7) involves the inversion of KKT matrix, which is computationally expensive. In addition, SQP does not allow for the violation of constraints and thus may struggle to find a feasible solution when the NLP includes many inequality constraints.

2) ALM: ALM is an iterative method for NLP, which combines aspects of both Lagrange multipliers and penalty methods to handle constraints by defining the augmented Lagrangian function as follows [?]:

$$\begin{aligned} L_{\mathrm{ALM}}(\boldsymbol{x},\boldsymbol{\lambda};\mu) := f(\boldsymbol{x}) + \sum_{j\in\mathbb{E}} \lambda_j c_j(\boldsymbol{x}) + \tfrac{1}{2\mu}\sum_{j\in\mathbb{E}} c_j^2(\boldsymbol{x}) \\ + \sum_{i\in\mathbb{I}} \psi(c_{i+n_{\mathrm{eq}}}(\boldsymbol{x}), \lambda_{i+n_{\mathrm{eq}}};\mu), \end{aligned} \tag{8}$$

with barrier parameter $\mu > 0$ and function

$$\psi(t,\sigma;\mu) := \begin{cases} -\sigma t + t^2/(2\mu) & \text{if } t - \mu\sigma \le 0, \\ -\mu\sigma^2/2 & \text{otherwise}, \end{cases} \tag{9}$$

weighting the inequality constraints $c_{i+n_{\mathrm{eq}}}(\boldsymbol{x}) = c_i^{\mathrm{in}}(\boldsymbol{x})$ for all $i \in \mathbb{I}$. Note that the augmented Lagrangian L_{ALM} differs from the standard Lagrangian (2) due to the squared terms of the equality constraints $c_j(\boldsymbol{x}) = c_j^{\mathrm{eq}}(\boldsymbol{x})$ for all $j \in \mathbb{E}$ and the sum of $\psi(c_{i+n_{\mathrm{eq}}}(\boldsymbol{x}), \lambda_{i+n_{\mathrm{eq}}};\mu)$ for all $i \in \mathbb{I}$.

The vector $\boldsymbol{x}[k]$ is updated to minimize the augmented Lagrangian function L_{ALM} for given $\boldsymbol{\lambda}[k]$ and $\mu[k] > 0$ as follows

$$\min_{\boldsymbol{x}[k]} L_{\mathrm{ALM}}(\boldsymbol{x}[k],\boldsymbol{\lambda}[k];\mu[k]). \tag{10}$$

Methods of unconstrained optimization, such as Newton and quasi-Newton methods, are usually employed to solve this problem [?]. The estimated Lagrange multipliers $\boldsymbol{\lambda}[k]$ are updated based on the extent of constraint violation, i.e.

$$\lambda_j[k+1] := \lambda_j[k] + \tfrac{c_j(\boldsymbol{x}[k])}{\mu[k]}, \qquad \text{for all } j \in \mathbb{E}, \tag{11a}$$

$$\lambda_i[k+1] := \max\left(\lambda_i[k] + \tfrac{c_{i+n_{\mathrm{eq}}}(\boldsymbol{x}[k])}{\mu[k]}, 0\right), \text{ for all } i \in \mathbb{I}. \tag{11b}$$

The barrier parameter $\mu[k]$ is adjusted during the iterations to balance convergence and numerical stability.

This separate update of optimization variables and Lagrange multipliers can lead to more efficient and scalable solutions. Particularly, this approach allows for the violation of constraints during the iteration process and thus can handle infeasible starting points and inequality constraints. However, ALM uses multiple tuning parameters to handle the constraints, such as the barrier parameter and others used for practical implementation [?]. Finding appropriate values for these parameters can be challenging and may require problem-specific tuning. In addition, ALM typically involves either the inversion of the Hessian or the approximation of the inverted Hessian; which is (in general) computationally demanding.

III. Lyapunov-based Nonlinear Programming (LBNLP)

This section presents the novel Lyapunov-based approach for solving NLP. Section III-B presents an update law derived from Lyapunov analysis including a proof of convergence. Section III-C proposes an implementation algorithm for the proposed method and Section III-D describes beneficial properties of the proposed method in comparison to SQP and ALM.

A. (Re-)Introduction of crucial definitions and facts

For later, gradient

$$\underbrace{\boldsymbol{g}_L(\boldsymbol{x},\boldsymbol{\lambda})}_{\in\mathbb{R}^n} := \nabla_{\boldsymbol{x}} L(\boldsymbol{x},\boldsymbol{\lambda}) \overset{(2)}{=} \underbrace{\nabla_{\boldsymbol{x}} f(\boldsymbol{x})}_{=:\boldsymbol{g}_f(\boldsymbol{x})\in\mathbb{R}^n} + \underbrace{[\nabla_{\boldsymbol{x}} \boldsymbol{c_i}(\boldsymbol{x})^\top]}_{=:\boldsymbol{C}(\boldsymbol{x})\in\mathbb{R}^{n\times n_{\mathrm{c}}}} \boldsymbol{\lambda} \tag{12}$$

and Hessian

$$\begin{aligned} \underbrace{\boldsymbol{H}_L(\boldsymbol{x},\boldsymbol{\lambda})}_{\in\mathbb{R}^{n\times n}} &:= \nabla_{\boldsymbol{x}} \boldsymbol{g}_L(\boldsymbol{x},\boldsymbol{\lambda}) := \nabla_{\boldsymbol{x}}^2 L(\boldsymbol{x},\boldsymbol{\lambda}) \\ &\overset{(2)}{=} \underbrace{\nabla_{\boldsymbol{x}}^2 f(\boldsymbol{x})}_{=:\boldsymbol{H}_f(\boldsymbol{x})\in\mathbb{R}^{n\times n}} + \underbrace{\nabla_{\boldsymbol{x}} \boldsymbol{C}(\boldsymbol{x})\boldsymbol{\lambda}}_{\boldsymbol{H}_C(\boldsymbol{x},\boldsymbol{\lambda})\in\mathbb{R}^{n\times n}} \end{aligned} \tag{13}$$

of the Lagrangian L as in (2) with respect to $\boldsymbol{x}$ are required. Moreover, note that the following hold

$$\left.\begin{aligned} \nabla_{\boldsymbol{\lambda}} \boldsymbol{g}_L(\boldsymbol{x},\boldsymbol{\lambda}) &\overset{(12)}{=} \boldsymbol{C}(\boldsymbol{x}) \in \mathbb{R}^{n\times n_{\mathrm{c}}}, \\ \nabla_{\boldsymbol{\lambda}} L(\boldsymbol{x},\boldsymbol{\lambda}) &\overset{(2)}{=} \boldsymbol{c}(\boldsymbol{x})^\top \in \mathbb{R}^{1\times n_{\mathrm{c}}}, \text{ and} \\ \nabla_{\boldsymbol{\lambda}}^2 L(\boldsymbol{x},\boldsymbol{\lambda}) &\overset{(2)}{=} \boldsymbol{O}_{n_{\mathrm{c}}\times n_{\mathrm{c}}}. \end{aligned}\right\} \tag{14}$$

Furthermore, for $\boldsymbol{C} \in \mathbb{R}^{n\times n}$,

$$\|\boldsymbol{C} - \boldsymbol{I}_n\| = \max_i |\sigma_i(\boldsymbol{C}) - 1| \tag{15}$$

where $\sigma_i(\cdot)$ denotes the ith eigenvalue of its input matrix and $\sigma_1(\cdot) < \sigma_2(\cdot) < \cdots < \sigma_n(\cdot)$. For $x \in \mathbb{R}$ and $y \in \mathbb{R}$, the following holds

$$(x-y)^2 = x^2 - 2xy + y^2 \ge 0$$

which gives

$$xy \le \frac{1}{2}(x^2 + y^2). \tag{16}$$

Finally, the following Lyapunov function candidate

$$V := \tfrac{1}{2}\boldsymbol{g}_L(\boldsymbol{x},\boldsymbol{\lambda})^\top \boldsymbol{g}_L(\boldsymbol{x},\boldsymbol{\lambda}) + \tfrac{1}{2}\boldsymbol{c}(\boldsymbol{x})^\top \boldsymbol{c}(\boldsymbol{x}). \tag{17}$$

will play a crucial role with its time derivative

$$\begin{aligned}\tfrac{\mathrm{d}}{\mathrm{d}t}V &\overset{(13),(14)}{=} \boldsymbol{g}_L(\boldsymbol{x},\boldsymbol{\lambda})^\top\Big(\boldsymbol{H}_L(\boldsymbol{x},\boldsymbol{\lambda})\tfrac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{x}+\boldsymbol{C}(\boldsymbol{x})\tfrac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{\lambda}\Big)\\ &\quad +\boldsymbol{c}(\boldsymbol{x})^\top\boldsymbol{C}(\boldsymbol{x})^\top\tfrac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{x} \qquad (18)\\ &\approx \boldsymbol{g}_L(\boldsymbol{x},\boldsymbol{\lambda})^\top\Big(\boldsymbol{H}_L(\boldsymbol{x},\boldsymbol{\lambda})\tfrac{\Delta\boldsymbol{x}}{\Delta t}+\boldsymbol{C}(\boldsymbol{x})\tfrac{\Delta\boldsymbol{\lambda}}{\Delta t}\Big)\\ &\quad +\boldsymbol{c}(\boldsymbol{x})^\top\boldsymbol{C}(\boldsymbol{x})^\top\tfrac{\Delta\boldsymbol{x}}{\Delta t} \qquad (19)\\ &= \tfrac{1}{\Delta t}\begin{bmatrix}\boldsymbol{g}_L(\boldsymbol{x},\boldsymbol{\lambda})\\ \boldsymbol{c}(\boldsymbol{x})\end{bmatrix}^\top\boldsymbol{K}(\boldsymbol{x},\boldsymbol{\lambda})\begin{bmatrix}\Delta\boldsymbol{x}\\ \Delta\boldsymbol{\lambda}\end{bmatrix}, \quad (20)\end{aligned}$$

where, in the second step, the approximations $\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{x}\approx\frac{\Delta\boldsymbol{x}}{\Delta t}$ and $\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{\lambda}\approx\frac{\Delta\boldsymbol{\lambda}}{\Delta t}$ were used (with $\Delta t>0$) and $\boldsymbol{K}(\boldsymbol{x},\boldsymbol{\lambda})$ is as in (7). Clearly, it is well known from Lyapunov's second method [?], that if $\frac{\mathrm{d}}{\mathrm{d}t}V<0$ for all non-zero $(\boldsymbol{x},\boldsymbol{\lambda})$, the system is stable; which implies for this approach here that the iteration algorithm is not diverging.

B. Update law

Two update laws will be presented; one existing and one proposed. To do so, define (i) the vector of Lagrange multipliers of active constraints as $\boldsymbol{\lambda}_{\mathbb{A}}:=[\lambda_a]_{a\in\mathbb{A}}\in\mathbb{R}^m$ (with $m\le n_c$), (ii) the active constraint vector as $\boldsymbol{c}_{\mathbb{A}}(\boldsymbol{x}):=[c_a(\boldsymbol{x})]_{a\in\mathbb{A}}\in\mathbb{R}^m$ and (iii) its derivative with respect to $\boldsymbol{x}$ as $\boldsymbol{C}_{\mathbb{A}}(\boldsymbol{x}):=\nabla_{\boldsymbol{x}}\boldsymbol{c}_{\mathbb{A}}(\boldsymbol{x})\in\mathbb{R}^{n\times m}$. Hence, only active constraints are considered in the following which will allow to invoke the LICQ (see Defintion 1).

1) Update law 1 (Existing): The first update law is

$$\begin{bmatrix}\Delta\boldsymbol{x}\\ \Delta\boldsymbol{\lambda}_{\mathbb{A}}\end{bmatrix}:=-\alpha\boldsymbol{K}^\top(\boldsymbol{x},\boldsymbol{\lambda}_{\mathbb{A}})\begin{bmatrix}\boldsymbol{g}_L(\boldsymbol{x},\boldsymbol{\lambda}_{\mathbb{A}})\\ \boldsymbol{c}_{\mathbb{A}}(\boldsymbol{x},\boldsymbol{\lambda}_{\mathbb{A}})\end{bmatrix}, \qquad (21)$$

where $\alpha>0$ is the step length. Inserting (21) into the time derivative (20) of the Lyapunov function (17) yields[1]

$$\begin{aligned}\tfrac{\mathrm{d}}{\mathrm{d}t}V &\overset{(20)}{\approx} \tfrac{1}{\Delta t}\begin{bmatrix}\boldsymbol{g}_L\\ \boldsymbol{c}_{\mathbb{A}}\end{bmatrix}^\top\boldsymbol{K}\begin{bmatrix}\Delta\boldsymbol{x}\\ \Delta\boldsymbol{\lambda}_{\mathbb{A}}\end{bmatrix}\\ &\overset{(24)}{=} -\tfrac{\alpha}{\Delta t}\begin{bmatrix}\boldsymbol{g}_L\\ \boldsymbol{c}_{\mathbb{A}}\end{bmatrix}^\top\boldsymbol{K}\boldsymbol{K}^\top\begin{bmatrix}\boldsymbol{g}_L\\ \boldsymbol{c}_{\mathbb{A}}\end{bmatrix}\le 0. \qquad (22)\end{aligned}$$

This update law was introduced in [?] as the discrete-time Jacobian matrix transpose method (DJT).

2) Update law 2 (Proposed): Assume the inverse of $\boldsymbol{H}_L(\boldsymbol{x},\boldsymbol{\lambda}_{\mathbb{A}})$ exists but does not need to be known. Define

$$\mathbb{B}=\left\{\beta\in\mathbb{R}\;\middle|\;\left\|\beta\boldsymbol{H}_L^{-1}-\boldsymbol{I}_n\right\|\le 1,\beta\ge 0\right\}. \qquad (23)$$

For $\alpha>0$ and $\beta\in\mathbb{B}$, the second update law is given by

$$\begin{bmatrix}\Delta\boldsymbol{x}\\ \Delta\boldsymbol{\lambda}_{\mathbb{A}}\end{bmatrix}:=-\alpha\Big(\boldsymbol{K}^\top+\begin{bmatrix}\boldsymbol{O}_{n\times n}&\boldsymbol{O}_{n\times m}\\ \boldsymbol{O}_{m\times n}&-2\beta\boldsymbol{I}_m\end{bmatrix}\Big)\begin{bmatrix}\boldsymbol{g}_L\\ \boldsymbol{c}_{\mathbb{A}}\end{bmatrix} \qquad (24)$$

[1]The arguments $\boldsymbol{x}$ and $\boldsymbol{\lambda}$ are dropped to ease readability.

Evaluating the time derivative (20) of the Lyapunov function (17) for (24) yields

$$\begin{aligned}\tfrac{\mathrm{d}}{\mathrm{d}t}V &\overset{(20)}{\approx} \boldsymbol{g}_L^\top\Big(\boldsymbol{H}_L\tfrac{\Delta\boldsymbol{x}}{\Delta t}+\boldsymbol{C}_{\mathbb{A}}\tfrac{\Delta\boldsymbol{\lambda}}{\Delta t}\Big)+\boldsymbol{c}_{\mathbb{A}}^\top\boldsymbol{C}_{\mathbb{A}}^\top\tfrac{\Delta\boldsymbol{x}}{\Delta t}\\ &\overset{(24)}{=} -\tfrac{\alpha}{\Delta t}\boldsymbol{g}_L^\top\Big[\boldsymbol{H}_L\boldsymbol{H}_L^\top+\boldsymbol{C}_{\mathbb{A}}\boldsymbol{C}_{\mathbb{A}}^\top\Big]\boldsymbol{g}_L\\ &\quad -\tfrac{\alpha}{\Delta t}\boldsymbol{c}_{\mathbb{A}}^\top\boldsymbol{C}_{\mathbb{A}}^\top\boldsymbol{C}_{\mathbb{A}}\boldsymbol{c}_{\mathbb{A}}\\ &\quad +\tfrac{2\alpha}{\Delta t}\boldsymbol{g}_L^\top\boldsymbol{H}_L\big(\beta\boldsymbol{H}_L^{-1}-\boldsymbol{I}_n\big)\boldsymbol{C}_{\mathbb{A}}\boldsymbol{c}_{\mathbb{A}}, \qquad (25)\\ &\overset{(16)}{\le} -\tfrac{\alpha}{\Delta t}\boldsymbol{g}_L^\top\Big[\boldsymbol{C}_{\mathbb{A}}\boldsymbol{C}_{\mathbb{A}}^\top\Big]\boldsymbol{g}_L\le 0, \qquad (26)\end{aligned}$$

where (26) is derived invoking the following argument

$$\begin{aligned}&\boldsymbol{g}_L^\top\boldsymbol{H}_L\big(\beta\boldsymbol{H}_L^{-1}-\boldsymbol{I}_n\big)\boldsymbol{C}_{\mathbb{A}}\boldsymbol{c}_{\mathbb{A}}\\ &\quad\le\left\|\boldsymbol{g}_L^\top\boldsymbol{H}_L\right\|\left\|\beta\boldsymbol{H}_L^{-1}-\boldsymbol{I}_n\right\|\left\|\boldsymbol{C}_{\mathbb{A}}\boldsymbol{c}_{\mathbb{A}}\right\|\\ &\quad\overset{(23)}{\le}\left\|\boldsymbol{g}_L^\top\boldsymbol{H}_L\right\|\left\|\boldsymbol{C}_{\mathbb{A}}\boldsymbol{c}_{\mathbb{A}}\right\|\\ &\quad\overset{(16)}{\le}\frac{1}{2}\boldsymbol{g}_L^\top\Big[\boldsymbol{H}_L\boldsymbol{H}_L^\top\Big]\boldsymbol{g}_L+\frac{1}{2}\boldsymbol{c}_{\mathbb{A}}^\top\Big[\boldsymbol{C}_{\mathbb{A}}^\top\boldsymbol{C}_{\mathbb{A}}\Big]\boldsymbol{c}_{\mathbb{A}}.\end{aligned}$$

A possible selection of β depends on the eigenvalues of $\boldsymbol{H}_L$ according to (15). A suggestion is given as follows

$$\beta:=\begin{cases}\sigma_1(\boldsymbol{H}_L) & \text{if } \sigma_1(\boldsymbol{H}_L)\ge 0\\ |\sigma_n(\boldsymbol{H}_L)| & \text{if } \sigma_n(\boldsymbol{H}_L)\le 0\\ 0 & \text{else.}\end{cases} \qquad (27)$$

Please note that, for a proper choice of β, rough knowledge of $\boldsymbol{H}_L$ is required. [CH: Kyunghwan: Why do you not use "cases" and why are you introducing / typing so many { or } in the equations. There is a probably a reason for that. But I do not understand. Seems very complicated and tedious. Moreover, you can simply cite equations using "eqref{}". Is there a reason why you are doing it differently?]

3) Comparison of Update laws 1 and 2: $\Delta\boldsymbol{\lambda}_{\mathbb{A}}$ of update law 1 depends only on $\boldsymbol{g}_L$ not on $\boldsymbol{c}_{\mathbb{A}}$ (see the elements of matrix $\boldsymbol{K}$ defined in (7)). Thus, update law 1 may not effectively handle the constraints even though the Lyapunov analysis proved the convergence. Referring to ALM, where the update of $\boldsymbol{\lambda}_{\mathbb{A}}$ solely depends on $\boldsymbol{c}_{\mathbb{A}}$ (see (11)), update law 2 was derived to include an additional term of $2\alpha\beta\boldsymbol{c}_{\mathbb{A}}$ to allow for updating $\boldsymbol{\lambda}_{\mathbb{A}}$. Update law 1 is considered a special case of update law 2 with $\beta=0$.

The effectiveness of including the additional term can be shown by analyzing the time derivative (25) of the Lyapunov function for two extreme cases. When $\beta=0$ (i.e., update law 1), the upper bound of inequality (25) is derived as (26). When β is designed to make the last term in (25) negligibly small (i.e., ideal case of update 2), the upper bound of inequality (25) approximately becomes

$$\tfrac{\mathrm{d}}{\mathrm{d}t}V\overset{(25)}{\approx}-\tfrac{\alpha}{\Delta t}\boldsymbol{g}_L^\top\Big[\boldsymbol{H}_L\boldsymbol{H}_L^\top+\boldsymbol{C}_{\mathbb{A}}\boldsymbol{C}_{\mathbb{A}}^\top\Big]\boldsymbol{g}_L-\tfrac{\alpha}{\Delta t}\boldsymbol{c}_{\mathbb{A}}^\top\boldsymbol{C}_{\mathbb{A}}^\top\boldsymbol{C}_{\mathbb{A}}\boldsymbol{c}_{\mathbb{A}},$$

which shows a stronger convergence than update law 1 for both $\boldsymbol{g}_L$ and $\boldsymbol{c}_{\mathbb{A}}$. Furthermore, if $\boldsymbol{H}_L$ has full rank and the active set $\mathbb{A}(\boldsymbol{x})$ satisfies the LICQ (i.e., $\boldsymbol{H}_L\boldsymbol{H}_L^\top$ and $\boldsymbol{C}_{\mathbb{A}}^\top\boldsymbol{C}_{\mathbb{A}}$ are positive definite), then exponential decay is shown by

$$\begin{aligned}\tfrac{\mathrm{d}}{\mathrm{d}t}V &\le -\tfrac{\alpha}{\Delta t}\begin{bmatrix}\boldsymbol{g}_L\\ \boldsymbol{c}_{\mathbb{A}}\end{bmatrix}^\top\underbrace{\begin{bmatrix}\boldsymbol{H}_L\boldsymbol{H}_L^\top&\boldsymbol{O}\\ \boldsymbol{O}&\boldsymbol{C}_{\mathbb{A}}^\top\boldsymbol{C}_{\mathbb{A}}\end{bmatrix}}_{=:\boldsymbol{P}}\begin{bmatrix}\boldsymbol{g}_L\\ \boldsymbol{c}_{\mathbb{A}}\end{bmatrix},\\ &\le -\frac{2\alpha}{\Delta t}\lambda_{\min}(\boldsymbol{P})\,V,\end{aligned}$$

where $\boldsymbol{P}$ is positive definite. The exponential decay rate is $\frac{2\alpha}{\Delta t}\lambda_{\min}(\boldsymbol{P})$.

Because the update laws are implemented in discrete time, the step length α should be chosen to ensure their stability. The step length, proposed in Lemma 2.4 in [?], is adopted in this study as follows

$$\alpha[k+1] := ..., \tag{28}$$

[CH: Above, I would like to rewrite the following

$$\alpha[k] := \frac{\boldsymbol{r}[k]^\top \boldsymbol{\psi}(\boldsymbol{r}[k])}{\|\boldsymbol{\psi}(\boldsymbol{r}[k])\|^2}, \tag{29}$$

without $\boldsymbol{\psi}(\boldsymbol{r}[k]) := \begin{bmatrix} \Delta\boldsymbol{x}[k] & \Delta\boldsymbol{\lambda}_{\mathbb{A}}[k] \end{bmatrix}^\top / \alpha[k]$ (note / check $\boldsymbol{\psi}$ does not depend on $\boldsymbol{r}$ as follows now!) and $\boldsymbol{r}[k] := \begin{bmatrix} \boldsymbol{x}[k] & \boldsymbol{\lambda}_{\mathbb{A}}[k] \end{bmatrix}^\top$, where I think is something wrong! I think we can ease reading if we do not introduce too many new variables. So we should only use $\Delta\boldsymbol{\lambda}_{\mathbb{A}}[k]$, etc. all those are already introduced. Is ψ really depending on α? then there is no explicit solution as in (28). Please double-check reference again! might it be $\alpha[k-1]$ or the left-hand side above is $\alpha[k+1]$? That is what I have written. In general check arguments above!]

C. Algorithm

The optimization variables and Lagrange multipliers are updated as follows:

$$\boldsymbol{x}[k+1] \leftarrow \boldsymbol{x}[k] + \Delta\boldsymbol{x}[k], \tag{30a}$$
$$\boldsymbol{\lambda}_{\mathbb{A}}[k+1] \leftarrow \boldsymbol{\lambda}_{\mathbb{A}}[k] + \Delta\boldsymbol{\lambda}_{\mathbb{A}}[k]. \tag{30b}$$

Even though the Lyapunov function defined in (17) does not explicitly include three necessary conditions of (4c), (4d), and (4e) for the inequality constraints, these three conditions can be easily satisfied by adding the following actions at the end of each iteration:

- Restrain the Lagrange multipliers for the inequality constraints greater than or equal to 0.
- Remove the inequality constraints with $\lambda_i = 0$ from the active set $\mathbb{A}(\boldsymbol{x})$.
- Add the inequality constraints with $c_i(\boldsymbol{x}) < 0$ to the active set $\mathbb{A}(\boldsymbol{x})$.

The proposed method can be implemented by Algorithm 1.

D. Comparison of Properties of LBNLP, SQP, and ALM

Equations (24), (27), and (28) show that the proposed method is matrix-inversion-free and tuning-parameter-free. There is one more to note. The update law for $\boldsymbol{\lambda}_{\mathbb{A}}$, (24), is similar to that of ALM, (11), in that both include information on constraint violations (i.e., $\boldsymbol{c}_{\mathbb{A}}$) but differs in that (24) also includes information on $\boldsymbol{g}_L (= \nabla_{\boldsymbol{x}} L(\boldsymbol{x}, \boldsymbol{\lambda}))$. The inclusion of $\boldsymbol{g}_L$ allows the appropriate update of $\boldsymbol{\lambda}_{\mathbb{A}}$ without using tuning parameters as in ALM.

The properties of the proposed method are summarized in Table II as compared with SQP and ALM. Note that this comparison only considers the case for local convergence.

Algorithm 1: Proposed method for NLP

```
1  Compute a feasible starting point $(\boldsymbol{x}[0], \boldsymbol{\lambda}[0])$;
2  Set the initial active set $\mathbb{A}(\boldsymbol{x}[0])$;
3  [CH: here we should add also the update of $\alpha$!]
4  for $k = 0, 1, 2, \ldots$ do
5      Compute $\Delta\boldsymbol{x}[k]$ and $\Delta\boldsymbol{\lambda}_{\mathbb{A}}[k]$ using (24);
6      $\boldsymbol{x}[k+1] \leftarrow \boldsymbol{x}[k] + \Delta\boldsymbol{x}[k]$;
7      $\boldsymbol{\lambda}_{\mathbb{A}}[k+1] \leftarrow \boldsymbol{\lambda}_{\mathbb{A}}[k] + \Delta\boldsymbol{\lambda}_{\mathbb{A}}[k]$;
8      $\lambda_i[k+1] \leftarrow \max(\lambda_i[k+1], 0), i \in \mathbb{A}(x[k]) \cap \mathbb{E}$;
9      $\lambda_i[k+1] \leftarrow \lambda_i[k], i \notin \mathbb{A}(x[k])$;
10     $\mathbb{A}(\boldsymbol{x}[k+1]) \leftarrow \mathbb{A}(\boldsymbol{x}[k]) \backslash \{j\}$, for all $j \in \mathbb{A}(\boldsymbol{x}[k])$
         with $\lambda_j[k+1] = 0$;
11     $\mathbb{A}(\boldsymbol{x}[k+1]) \leftarrow \mathbb{A}(\boldsymbol{x}[k]) \cup \{j\}$, for all $j \notin \mathbb{A}(\boldsymbol{x}[k])$
         with $c_j(\boldsymbol{x}[k+1]) < 0$;
```

TABLE I
PROPERTY COMPARISON OF LBNLP, SQP, AND ALM.

	SQP	ALM	LBNLP
Matrix inversion(s)	Yes (e.g., KKT)	Yes (e.g., Hessian)	No
Tuning parameters	Not necessary	Necessary (μ and others)	No
Allowance of constraint violation	No	Yes	Yes
Update of $\boldsymbol{x}$ and λ	Simultaneous	Separate	Simultaneous

IV. APPLICATION OF LBNLP TO NMPTC

This section discusses the possible application of the proposed LBNLP to NMPC in general and NMPTC specifically: Section IV-A explains two different formulations of a general NMPC problem, whereas Section IV-B introduces NMPTC of electrical drives. Section IV-C provides validation results of the proposed method for the NMPTC problem and compares those to the results obtained by SQP and ALM.

A. Nonlinear model predictive control (NMPC)

1) Problem statement: The NMPC formulation is [CH: REF?]

$$\boldsymbol{u}^\star := \underset{\boldsymbol{u}[k+1],\cdots,\boldsymbol{u}[k+N]}{\arg\min} \|\boldsymbol{e}[k+N]\|^2_{\boldsymbol{F}} + \sum_{n=1}^{N-1} \|\boldsymbol{e}[k+n]\|^2_{\boldsymbol{Q}} + q\big(\boldsymbol{x}[k+n], \boldsymbol{u}[k+n]\big) \tag{31a}$$

subject to

$$\boldsymbol{x}[k+n+1] = \boldsymbol{f}\big(\boldsymbol{x}[k+n], \boldsymbol{u}[k+n]\big), \tag{31b}$$
$$\boldsymbol{y}[k+n] = \boldsymbol{g}\big(\boldsymbol{x}[k+n], \boldsymbol{u}[k+n]\big), \tag{31c}$$
$$\boldsymbol{h}(\boldsymbol{x}[k+n], \boldsymbol{u}[k+n]) \geq \boldsymbol{0}, \quad n = 1, \ldots, N \tag{31d}$$

where arguments k, $k+1$, ..., $k+N$ denote (succeeding) sampling instants until the prediction horizon $1 \leq N \in \mathbb{N}$; $\boldsymbol{x}$, $\boldsymbol{u}$, and $\boldsymbol{y}$ denote system state, control input, and output vector, respectively; $\boldsymbol{e} := \boldsymbol{y}_{\mathrm{ref}} - \boldsymbol{y}$ represents the tracking error (difference between output $\boldsymbol{y}$ and reference signal $\boldsymbol{y}_{\mathrm{ref}}$; $\boldsymbol{F}$ and $\boldsymbol{Q}$ are symmetric positive definite matrices (of appropriate size) weighting final and preceeding tracking errors; $q(\cdot,\cdot)$ represents the control effort function whose typical choice is

$\|\boldsymbol{u}\|^2_{\boldsymbol{R}}$ with symmetric positive definite matrix $\boldsymbol{R} = \boldsymbol{R}^\top > 0$; and $\boldsymbol{f}(\cdot,\cdot)$, $\boldsymbol{g}(\cdot,\cdot)$, and $\boldsymbol{h}(\cdot,\cdot)$ denote functions of the state, output, and (in)equality constraints, respectively.

This NMPC formulation can describe several optimal tracking control problems of various applications [CH: REF]. However, it requires an appropriate tuning of the matrices $\boldsymbol{F}$ and $\boldsymbol{Q}$, and function q to guarantee stability and convergence of the tracking error to zero due to the trade-off between the tracking error and control effort terms in the objective function [CH: REF?]. To avoid this tuning a reformulation of the problem statement is beneficial which is introduced next.

2) NMPC reformulation: The NMPC problem (31) can be reformulated by considering the (final) tracking error as equality constraint in contrast to considering it in the objective function (31a) which resolves the inevitable weighting trade-off between tracking error and control effort terms in (31a) [?]. The NMPC reformulation is given by

$$\boldsymbol{u}^\star := \underset{\boldsymbol{u}[k+1],\cdots,\boldsymbol{u}[k+N]}{\arg\min} \sum_{n=1}^{N-1} q\big(\boldsymbol{x}[k+n], \boldsymbol{u}[k+n]\big) \tag{32a}$$

$$\text{subject to}$$

$$\boldsymbol{x}[k+n+1] = \boldsymbol{f}\big(\boldsymbol{x}[k+n], \boldsymbol{u}[k+n]\big), \tag{32b}$$

$$\boldsymbol{y}[k+n] = \boldsymbol{g}\big(\boldsymbol{x}[k+n], \boldsymbol{u}[k+n]\big), \tag{32c}$$

$$\boldsymbol{h}(\boldsymbol{x}[k+n], \boldsymbol{u}[k+n]) \geq \boldsymbol{0},\ \ n = 1,\ldots,N \tag{32d}$$

$$\boldsymbol{e}[k+N] = \boldsymbol{0} \tag{32e}$$

There is no guarantee to satisfy the equality constraint in (36e) (i) if the reference signal $\boldsymbol{y}_{\text{ref}}$ cannot be attained within the prediction horizon no matter what control effort is admissible, or (ii) if a numerical optimization method for this problem does not allow violation of the equality constraint [CH: REF]. However, if a numerical optimization method allows for the violation of constraints like the proposed LBNLP method, this NMPC problem can (approximately) be solved even if the reference signal cannot be attained within the prediction horizon.

Note that SQP does not allow for the violation of constraints whereas ALM does. However, using the proposed LBNLP is even more advantageous than using ALM (recall Table II and see validation in the next Subection IV-B).

Remark 1. *In general, it is beneficial for the NMPC convergence if the reference signal is a smooth function as this helps to satisfy the (in)equality constraints at all sampling instants [CH: REF?]*

B. Nonlinear model predictive torque control (NMPTC)

The NMPTC problem presented in [?] is re-discussed to apply and validate the proposed LBNLP method. The NMPTC here is used to solve the Maximum-Torque per Current (MTPC) problem – a subproblem of optimal feedforward torque control (OFTC) [?] of permanent magnet synchronous machines (PMSMs) modelled by the nonlinear dynamics

$$\left.\begin{aligned} \boldsymbol{u}^{dq}_{\text{s}} &= \boldsymbol{R}^{dq}_{\text{s}}\boldsymbol{i}^{dq}_{\text{s}} + \omega_{\text{p}}\boldsymbol{J}\boldsymbol{\psi}^{dq}_{\text{s}} + \tfrac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{\psi}^{dq}_{\text{s}}, \\ \tfrac{\mathrm{d}}{\mathrm{d}t}\omega_{\text{m}} &= \tfrac{1}{\Theta_{\text{m}}}\Big(\tfrac{2}{3\kappa^2}n_{\text{p}}(\boldsymbol{i}^{dq}_{\text{s}})^\top\boldsymbol{J}\boldsymbol{\psi}^{dq}_{\text{s}} - m_{\text{l}}\Big) \\ \tfrac{\mathrm{d}}{\mathrm{d}t}\phi_{\text{m}} &= \omega_{\text{m}} \end{aligned}\right\} \tag{33}$$

with $\boldsymbol{J} := \left[\begin{smallmatrix}0 & -1\\ 1 & 0\end{smallmatrix}\right]$, stator voltages $\boldsymbol{u}^{dq}_{\text{s}} := (u^d_{\text{s}}, u^d_{\text{s}})^\top$, stator currents $\boldsymbol{i}^{dq}_{\text{s}} := (i^d_{\text{s}}, i^d_{\text{s}})^\top$, stator resistance matrix $\boldsymbol{R}^{dq}_{\text{s}} := \boldsymbol{R}^{dq}_{\text{s}}(\boldsymbol{i}^{dq}_{\text{s}}, \omega_{\text{p}}, \phi_{\text{p}}) = (\boldsymbol{R}^{dq}_{\text{s}})^\top > 0$, stator flux linkages $\boldsymbol{\psi}^{dq}_{\text{s}} := (\psi^d_{\text{s}}, \psi^d_{\text{s}})^\top := \boldsymbol{\psi}^{dq}_{\text{s}}(\boldsymbol{i}^{dq}_{\text{s}}, \omega_{\text{p}}, \phi_{\text{p}})$, load torque m_{l}, Clarke transformation factor $\kappa \in \{2/3, \sqrt{2/3}\}$ [?, Ch. 14], electrical angular velocity $\omega_{\text{p}} = n_{\text{p}}\omega_{\text{m}}$ (i.e., the product of pole pair number n_{p} and mechanical angular velocity ω_{m}) and machine torque $m_{\text{m}} := \frac{2}{3\kappa^2}n_{\text{p}}(\boldsymbol{i}^{dq}_{\text{s}})^\top\boldsymbol{J}\boldsymbol{\psi}^{dq}_{\text{s}}$. Note that stator resistance matrix $\boldsymbol{R}^{dq}_{\text{s}}(\boldsymbol{i}^{dq}_{\text{s}}, \omega_{\text{p}}, \phi_{\text{p}})$ and stator flux linkages $\boldsymbol{\psi}^{dq}_{\text{s}}(\boldsymbol{i}^{dq}_{\text{s}}, \omega_{\text{p}}, \phi_{\text{p}}$ may vary with current, frequency and angle, respectively [?]. For this paper, only the dependency on the currents $\boldsymbol{i}^{dq}_{\text{s}}$ is considered. Moreover, the machine torque m_{m} represents the average torque neglecting the dependencies on rotor position or iron losses [?]. Stator voltages and currents are limited to [?]

$$\|\boldsymbol{u}^{dq}_{\text{s}}\| \leq \widehat{u}_{\max} \quad \text{and} \quad \|\boldsymbol{i}^{dq}_{\text{s}}\| \leq \widehat{\imath}_{\max} \tag{34}$$

to protect the machine (isolation and thermal capacity). Rewriting (33) with current dynamics and discretizing[2] yields

$$\left.\begin{aligned} \boldsymbol{i}^{dq}_{\text{s}}[k+1] &= T_{\text{s}}\boldsymbol{L}^{dq}_{\text{s}}[k]^{-1}\Big(\boldsymbol{u}^{dq}_{\text{s}}[k] - \boldsymbol{R}^{dq}_{\text{s}}\boldsymbol{i}^{dq}_{\text{s}}[k] \\ &\quad -\omega_{\text{p}}[k]\boldsymbol{J}\boldsymbol{\psi}^{dq}_{\text{s}}[k]\Big) - \boldsymbol{i}^{dq}_{\text{s}}[k] =: \boldsymbol{f}(\boldsymbol{i}^{dq}_{\text{s}}[k], \boldsymbol{u}^{dq}_{\text{s}}[k]), \\ \omega_{\text{m}}[k+1] &= \tfrac{T_{\text{s}}}{\Theta_{\text{m}}}\Big(m_{\text{m}}[k] - m_{\text{l}}[k]\Big) - \omega_{\text{m}}[k] \\ \phi_{\text{m}}[k+1] &= T_{\text{s}}\omega_{\text{m}}[k] - \phi_{\text{m}}[k], \end{aligned}\right\} \tag{35}$$

where, to simplify notation, the following conventions were and will be applied for differential inductance matrix $\boldsymbol{L}^{dq}_{\text{s}}[k] := \boldsymbol{L}^{dq}_{\text{s}}(\boldsymbol{i}^{dq}_{\text{s}}[k]) = \boldsymbol{L}^{dq}_{\text{s}}(\boldsymbol{i}^{dq}_{\text{s}}[k])^\top > 0$, flux linkages $\boldsymbol{\psi}^{dq}_{\text{s}}[k] := \boldsymbol{\psi}^{dq}_{\text{s}}(\boldsymbol{i}^{dq}_{\text{s}}[k])$ and machine torque $m_{\text{m}}[k] := \frac{2}{3\kappa^2}n_{\text{p}}\boldsymbol{i}^{dq}_{\text{s}}[k]^\top\boldsymbol{J}\boldsymbol{\psi}^{dq}_{\text{s}}[k]$.

For MTPC (or MTPA[3]) operation, the minimization of Joule losses $(\boldsymbol{i}^{dq}_{\text{s}})^\top\boldsymbol{R}^{dq}_{\text{s}}\boldsymbol{i}^{dq}_{\text{s}}$ is crucial and usual a prediction horizon of $N = 1$ is sufficient [?], [?]. With these losses, the right-hand side $\boldsymbol{f}(\boldsymbol{i}^{dq}_{\text{s}}[k], \boldsymbol{u}^{dq}_{\text{s}}[k])$ of the current dynamics in (35) and the physical machine constraints in (34), the NLMPTC problem can directly be formulated:

$$\boldsymbol{u}^\star := \underset{\boldsymbol{u}[k+1]}{\arg\min}\, \underbrace{\boldsymbol{i}^{dq}_{\text{s}}[k+1]^\top\boldsymbol{R}^{dq}_{\text{s}}\boldsymbol{i}^{dq}_{\text{s}}[k+1]}_{=:q(\boldsymbol{x}[k+1],\boldsymbol{u}[k+1])} \tag{36a}$$

$$\text{subject to}$$

$$\boldsymbol{x}[k+1] := \boldsymbol{i}^{dq}_{\text{s}}[k+1] = \boldsymbol{f}\big(\boldsymbol{i}^{dq}_{\text{s}}[k], \boldsymbol{u}^{dq}_{\text{s}}[k]\big), \tag{36b}$$

$$y[k+1] := m_{\text{m}}[k+1] = \tfrac{2}{3\kappa^2}n_{\text{p}}\boldsymbol{i}^{dq}_{\text{s}}[k+1]^\top\boldsymbol{J}\boldsymbol{\psi}^{dq}_{\text{s}}[k+1], \tag{36c}$$

$$\boldsymbol{h}(\boldsymbol{x}[k+1], \boldsymbol{u}[k+1]) := \begin{pmatrix} \widehat{u}^2_{\max} - \|\boldsymbol{u}^{dq}_{\text{s}}[k+1]\|^2 \\ \widehat{\imath}^2_{\max} - \|\boldsymbol{i}^{dq}_{\text{s}}[k+1]\|^2 \end{pmatrix} \geq \boldsymbol{0}_2, \tag{36d}$$

$$e[k+1] := m_{\text{m,ref}}[k+1] - m_{\text{m}}[k+1] = 0 \tag{36e}$$

where $\boldsymbol{x}[k+1] = \boldsymbol{i}^{dq}_{\text{s}}[k+1]$ and $\boldsymbol{u}[k+1] = \boldsymbol{u}^{dq}_{\text{s}}[k+1]$ denote the stator current and voltage vectors at the next sampling instant, respectively. Clearly, NMPTC in (36) is a subproblem of the reformulated NMPC in (32).

[2]Invoking the Euler forward method leads to $\frac{\mathrm{d}}{\mathrm{d}t}x(t) \approx (x[k+1] - x[k])/T_{\text{s}}$ with sampled quantity $x[k] \approx x(kT_{\text{s}})$ at sampling instant $k \in \mathbb{N}$ and sampling period $T_{\text{s}} > 0$.

[3]Maximum Torque per Ampere.

TABLE II
IMPLEMENTATION PARAMETERS OF PMSM.

Parameter	Value
R_{s}	25 mΩ
L_{s}^{d}	0.45 mH
L_{s}^{q}	0.66 mH
ψ_{pm}	0.0563 Wb
n_{p}	8
$\widehat{u}_{\max}$	56.5 V
T_{s}	0.1 ms

C. Validation

For the validation of the proposed LBNLP, the NMPTC problem is implemented for an anisotropic PMSM with affine stator flux linkage

$$\boldsymbol{\psi}_{\mathrm{s}}^{dq} = \begin{bmatrix} L_{\mathrm{s}}^{d} & 0 \\ 0 & L_{\mathrm{s}}^{q} \end{bmatrix} \boldsymbol{i}_{\mathrm{s}}^{dq} + \begin{pmatrix} \psi_{\mathrm{pm}} \\ 0 \end{pmatrix},$$

where L_{s}^{d} and L_{s}^{q} denote the d- and q-axis inductances, respectively. ψ_{pm} denotes the flux linkage of the permanent magnets. The machine parameters are collected in Table II.

The NMPTC problem introduced in Section **??** was implemented and numerically solved in MATLAB 2022a by SQP, ALM, and the proposed LBNLP, respectively. The reference torque $m_{\mathrm{m,ref}}$ was given with 30 Nm. Two different electrical angular velocities ω_{p}, at 840 rad/s and 1090 rad/s, were simulated to examine the cases when the inequality (voltage) constraint in were inactive and active, respectively. [CH: SO we do MORE THAN MTPC??? also field weakening? Before I finalize this subsection, we need to have the updated figues ... see also comment below - very nice would be to have such comparative plots for all OFTC operation strategies like MTPC, MC, FW, MTPV!] [KC: The simulation results include MTPC and FW operations. I believe these two operations are good enough for this conference paper. Of course, we need to include MC and MTPV operations for a journal paper.] [KC: I just realized I used MTPA on the figures, not MTPC. I will revise this soon.]

ALM and the proposed method solved the problem in the formulation of (32), while SQP solved the problem in the formulation of (31) because SQP did not guarantee to handle the equality constraint (**??**). SQP was implemented by the 'fmincon' function available in MATLAB, which is a well-known NLP solver, with the option of 'sqp'. Three different values were used for [MS: R?] $\boldsymbol{H}_L$: 10^{-4}, 10^{-2}, and 10^{0}, to examine the trade-off between the tracking error terms and control effort term in the objective function. ALM was implemented by solving (10) using Newton's method with an iteration termination condition of $|\nabla_x L_{\mathrm{ALM}}| \leq 10^{-4}$. Three different values were used for μ: 10^{-2}, 10^{0}, and 10^{2}, to examine the sensitivity of solutions to this tuning parameter. The proposed method was implemented by Algorithm 1. Two different cases were examined when the for-loop in Algorithm 1 was repeated once and twice, respectively, for each time instant t.

The NMPTC results obtained using SQP with $w = 10^{-4}, 10^{-2}$, and 10^{0} are shown in Fig. 1. x_1 and x_2 on the horizontal and vertical axes represent the first and second elements of the state $\boldsymbol{x}$, respectively. The MTPA line, denoted by dashed lines in the figures, represents the points satisfying

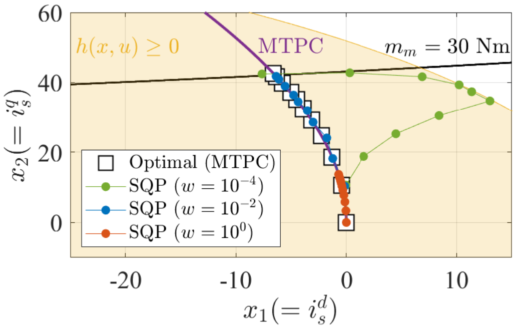
$$\min_{x} q(x,u) \tag{37a}$$

$$\text{subject to}$$

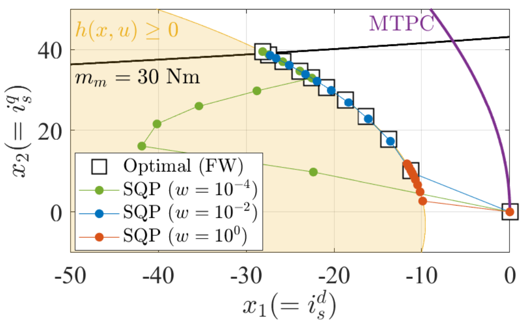
$$e = r - y = 0, \tag{37b}$$

meaning the target points of the state $\boldsymbol{x}$ when the inequality constraint $h(x,u) \geq 0$ is inactive. with $w = 10^{-4}$, the state trajectory, x_t, satisfied the tracking condition $e_{t+N} = 0$ well but quite deviated from the control-effort-minimizing points (i.e., MTPA line) due to the large portion of the tracking error terms (see Fig. 1). with $w = 10^{0}$, the state trajectory aligned with the MTPA line when the inequality constraint was inactive (see Fig. 1(a)) or moved close to the MTPA line even when the constraint was active (see Fig. 1(b)). However, the state trajectory hardly satisfied the tracking condition due to the large portion of the control effort term. The balance between the tracking error terms and control effort term was made with $w = 10^{-2}$.

The NMPTC results obtained using ALM with $\mu = 10^{-2}, 10^{0}$, and 10^{2} are shown in Fig. 2. with $\mu = 10^{-2}$, the state trajectory, x_t, was unstable until reaching the final point when the inequality constraint was inactive (see Fig. 2(a)). This was because the barrier parameter μ was too small so the Lagrange multiplier for the inequality constraint (**??**) was easily updated and considered active, which had to be inactive. with $w = 10^{2}$, by contrast, the state trajectory hardly satisfied

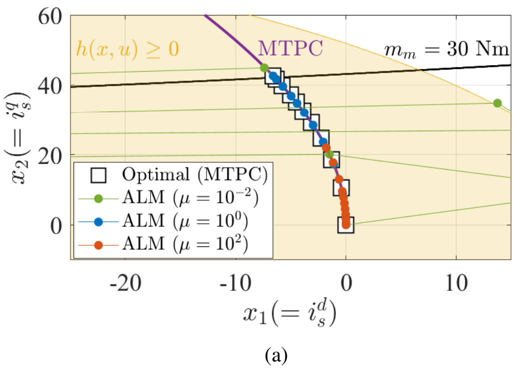


(a)



(b)

Fig. 1. NMPC results obtained using SQP with $w = 10^{-4}, 10^{-2}$, and 10^{0} for the conditions of (a) $w_r = 840$ rad/s and (b) $w_r = 1090$ rad/s.



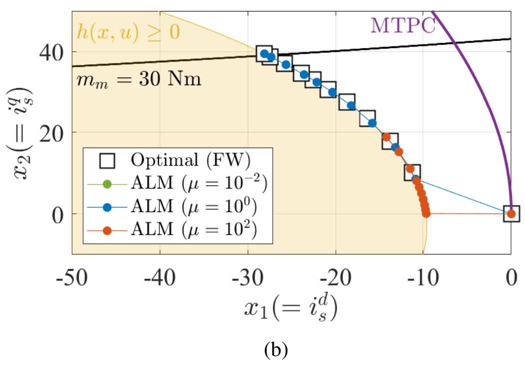


Fig. 2. NMPC results obtained using ALM with $\mu = 10^{-2}, 10^{0}$, and 10^{2} for the conditions of (a) $w_r = 840$ rad/s and (b) $w_r = 1090$ rad/s.

the tracking condition because the Lagrange multiplier for the equality constraint (**??**) was updated slowly due to the large value of μ. The Lagrange multipliers for the constraints were updated appropriately with $\mu = 10^0$, and thus a desirable state trajectory was obtained.

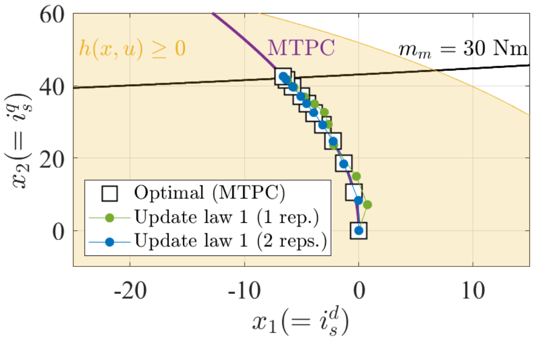
The NMPTC results obtained using update laws 1 and 2 (i.e., LBNLP) of repeating the for-loop once and twice are shown in Figs. 3 and 4, respectively. While update law 1 showed good convergence with only one iteration when the inequality constraint is not active, update law 1 failed to handle the active constraint with one iteration (see Fig. 3(b)). Update law 1 with two iterations converged to the target point at the end; however, it showed an infeasible transient behavior (i.e., the inequality constraint was violated) (see. Fig (3(b)). On the other hand, LBNLP provided an almost direct move to the target points even with one repetition of the for-loop (see Fig. 4). Just one more repetition of the for-loop guaranteed a very accurate move. This result is significant in that conventional methods like SQP and ALM provided satisfactory results only with appropriate values of tuning parameters, whereas LBNLP guaranteed a desirable result just by repeating the simple algorithm once or twice, not relying on tuning parameters.

[KC: Computation times will be replaced by a table or just text data, considering the page limit.] Computation times of representative results of each method are shown in Fig. **??**. The representative results were selected as Figs. **??**, **??**, and **??** for SQP, ALM, and the proposed method, respectively. The computation times of SQP were approximately one hundred times greater than those of ALM and the proposed method, which is probably due to the inversion of the KKT matrix for SQP and multiple iterations to elaborate solutions. The average computation time of ALM was approximately twice greater than that of the proposed method. This is because ALM used multiple iterations to guarantee the convergence of Newton's method. This result verifies that the proposed method, which is matrix-inversion-free and does not require large numbers of iterations, is computationally efficient.

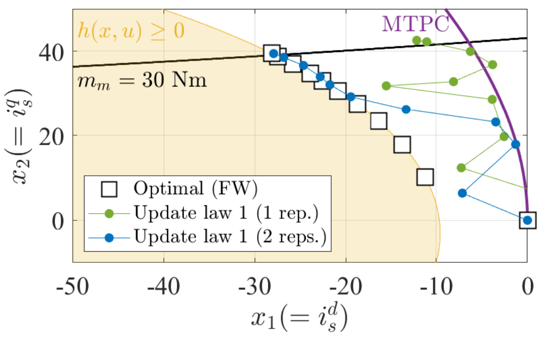
V. Conclusion

This study presented a novel numerical optimization method to solve NLP. This method was designed based on a Lyapunov approach to reach the necessary conditions for optimality. The advantage of using the Lyapunov approach was the update law was derived in a tuning-parameter-free and matrix-inversion-free manner; thus, the proposed method could be implemented easily with less iteration and computation time than conventional methods, such as SQP and ALM. The effectiveness of the proposed method was validated by using it to solve an NLP problem, which was an NMPC problem for optimal torque control of PMSMs, and comparing it with SQP and ALM.

The following issues need to be handled in a future study: [CH: In the figures, the correct operation strategy should be indicated as well; at the moment, the reader might think that MTPA should work but all methods do not always converge to the MTPA hyperbola (of course, this is due to the voltage constraint, Hence, we should show FW, MTPV, etc. all "correct" points on the operation strategy line.][KC: I will add such indications in the next update of the figures.] [CH: I would present for both angular velocities all results of SQP, ALM and LBNLP in ONE FIG. It would be nice if the simulations could show ALL OFTC operation strategies such as MPTC, MC, FW, MTPV - do you think this is feasible?]

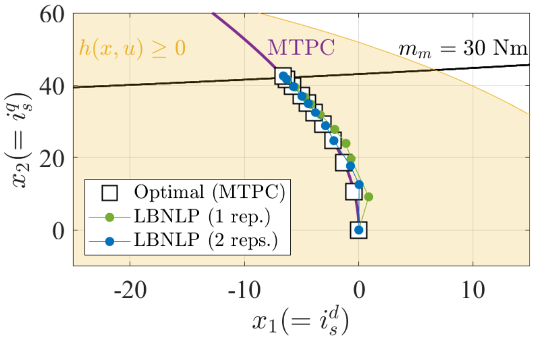


(a)

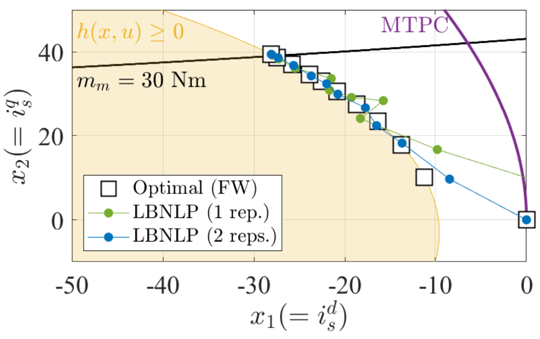


(b)

Fig. 3. NMPC results obtained using update law 1 of repeating the for-loop once and twice for each time instant t, for the conditions of (a) $w_r = 840$ rad/s and (b) $w_r = 1090$ rad/s.



(a)



(b)

Fig. 4. NMPC results obtained using LBNLP (update law 2) of repeating the for-loop once and twice for each time instant t, for the conditions of (a) $w_r = 840$ rad/s and (b) $w_r = 1090$ rad/s.