This paper has been accepted for publication at the
IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Las Vegas, 2020.

Online Weight-adaptive Nonlinear Model Predictive Control

Dimche Kostadinov and Davide Scaramuzza

***Abstract*— Nonlinear Model Predictive Control (NMPC) is a powerful and widely used technique for nonlinear dynamic process control under constraints. In NMPC, the state and control weights of the corresponding state and control costs are commonly selected based on human-expert knowledge, which usually reflects the acceptable stability in practice. Although broadly used, this approach might not be optimal for the execution of a trajectory with the lowest positional error and sufficiently "smooth" changes in the predicted controls. Furthermore, NMPC with an online weight update strategy for fast, agile, and precise unmanned aerial vehicle navigation, has not been studied extensively. To this end, we propose a novel control problem formulation that allows online updates of the state and control weights. As a solution, we present an algorithm that consists of two alternating stages: (i) state and command variable prediction and (ii) weights update. We present a numerical evaluation with a comparison and analysis of different trade-offs for the problem of quadrotor navigation. Our computer simulation results show improvements of up to 70% in the accuracy of the executed trajectory compared to the standard solution of NMPC with fixed weights.**

I. INTRODUCTION

In the past, diverse approaches have been used for robust, accurate and fast control [8], [3], [4] and [7]. Applied across a wide range of domains, from chemicals to aerospace industries, one of the most powerful and practically useful approaches is NMPC [1]. Its main advantage is that it allows making predictions about the immediate future point under constraints while considering all predicted future points over a given horizon. In recent years, several efficient solutions to NMPC and improvements have been proposed [11]. One of the most prominent methods for solving NMPC problem is the real-time iteration (RTI) scheme [6]. The advantage of RTI is that it allows "on-the-fly" prediction updates: as new estimates become available, an iterative solution using only a small number of iteration steps gives the new predictions.

In the NMPC problem, the state and control weights for the corresponding costs significantly impact the control performance. Usually, fixed weights are carefully selected based on human expert knowledge (platform and trajectory wise tuning) for unmanned aerial vehicle navigation. Under many scenarios, this strategy is preferable. Such weight selection takes into account stability-related properties and exploits the strengths of NMPC. However, whether this approach uses the advantages of the NMPC to the full extent remains an open question.

Dimche Kostadinov and Davide Scaramuzza are with the Robotics and Perception Group (http://rpg.ifi.uzh.ch) at the Dept. of Informatics and Neuroinformatics, University of Zurich and ETH Zurich, 8050 Zurich, Switzerland, e-mail: {dimche, sdavide}@ifi.uzh.ch

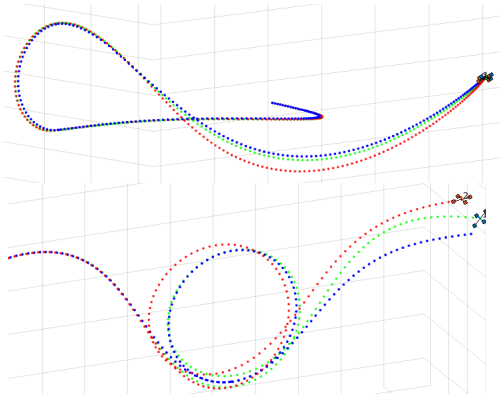

Fig. 1. **A)** The blue line represents the reference trajectory, **B)** The red line represents executed trajectory using the standard solution for NMPC with fixed weights, and **C)** The green line represents the executed trajectory using the algorithm that solves the proposed online weight-adaptive NMPC.

Additionally, weight tuning might not allow good generalization across a set[1] of diverse trajectories. On the other hand, online and data-adaptive strategies for updating the cost weights during trajectory execution were not studied extensively. In this line, besides the link between NMPC and online learning [16], other connections to known machine learning paradigms with emphasis on the weigh estimation remain not explored.

To utilize the full control potential of NMPC, we present online weight-adaptive approach. We introduce a novel control problem formulation, where, contrary to using predefined and fixed weights for the state cost, we include the weights of the state cost as a variable in our control problem. This allows us to propose an algorithm that can improve the state and control prediction by optimally updating the weights in an online fashion. Moreover, in our approach, we provide a generalization for a class of weight cost priors and function approximations (including but not limited to neural networks). Also, we give connections of our approach to online learning [13], reinforcement learning [14], and metric learning [9].

A. Contributions

In the following, we summarize our main contributions.

- We introduce a novel variant of the very well known NMPC control problem, where we address joint prediction of state and control variables and the estimate of the corresponding state and control weights.
- We propose a two-stage alternating RTI algorithm. It consists of (i) state and commands variable prediction

[1]In a sense that a single choice of weights might not allow tracking with the lowest positional error and "smooth" change over the predicted controls.

and (ii) optimal update of the control weights.
- We validate our approach by numerical simulation for the task of quadrotor navigation. We demonstrate that our approach reduces the error in the trajectory tracking while predicting controls that have "smooth" change over the executed trajectory. Compared to the solution of the commonly used NMPC, we show improvements of up to 70% in the accuracy of the executed trajectory.

B. Related Work

Adaptive Model Predictive Control (MPC) was studied in [2], [15], [17], [18] and [16]. In [2], the authors proposed an adaptive multi-variable zone controller and gave robustness guarantees for the controlled process. As an adaptive component, they added weighted slack coefficients to the nominal weight coefficients. In [15], the authors were focused on an analytical approach for tuning the control horizon. Their idea consists of computing the value for the optimal control horizon that ensures numerical stability. At the same time, their interest was on a wide set of linear controllable single-input single-output processes. In [18], the authors developed an adaptive cruise control system for vehicles. The authors utilized a hierarchical control architecture in which a low-level controller compensated for the nonlinear longitudinal vehicle dynamics. Their design enabled tracking of the desired acceleration. They solved a multi-objective control problem by a real-time weight tuning strategy by adjusting each objective's weight for different operating conditions. The authors in [17], proposed an adaptive stochastic model predictive control strategy. They considered multi-input multi-output systems that can be expressed by a finite impulse response model. In [16], a close connection between MPC and online learning was shown. The authors have proposed a new algorithm based on dynamic mirror descent. By doing so, they presented a general family of MPC algorithms that include many existing techniques as special instances. The same authors also provided different MPC perspectives and suggested principled schemes for the design of new MPC algorithms.

In contrast to the past work, as our main novelty, we consider cost-related weights as an additional variable in the NMPC problem. In this regard, our problem formulation extends the common NMPC problem. We propose a novel algorithm as a solution, while along the way, we also give connections to machine learning paradigms with a focus on metric learning.

C. Paper Organization

The rest of the paper is organized as follows. In Section II, we present the continuous control problem and its approximate discrete version. In Section III, we first present the approximate control problem formulation for our NMPC with online weight adaptation. Then, we propose our RTI solution in the form of a two-stage alternating algorithm and discuss the solution. We also give connections to known learning paradigms. While, we devote Section IV to numerical evaluation, and with Section V, we conclude the paper.

II. ROBOT CONTROL

In this section, we first present the continuous problem formulation for robot control and then give its discrete version.

We assume that the dynamics are described by a set of differential equations $f(\mathbf{x}, \mathbf{u})$, where $\mathbf{x} \in \Re^{M_x}$ and $\mathbf{u} : \Re^{\times M_u}$ denote state and control variables. Furthermore, we assume that an action objective is given, which defines the action cost $\mathcal{L}_a(\mathbf{x}, \mathbf{u}) : \Re^{M_x \times M_u} \rightarrow \Re^{+}$. The task of taking action can be expressed by the following optimization problem:

$$\begin{aligned} \{\hat{\mathbf{x}}, \hat{\mathbf{u}}\} = \arg\min_{\mathbf{x},\mathbf{u}} \int_{t_0}^{t_h} \mathcal{L}_a(\mathbf{x}, \mathbf{u}) dt, \\ \text{subject to } f(\mathbf{x}, \mathbf{u}) = 0, h(\mathbf{x}, \mathbf{u}) \leq 0, \end{aligned} \tag{1}$$

where $f(\mathbf{x}, \mathbf{u})$ and $h(\mathbf{x}, \mathbf{u})$ represent equality and inequality constraints that the solution should satisfy when we take action. In order to solve (1), first, a cost function for taking action is defined. Then (1) is discretized, transcribed, and linearized. In the following text, we go through these steps, which will allow us to present the discretized version of (1).

A. The Discrete Control Problem

As a common practice, the system dynamics (1) are discretized into N system points by using a time step δt over fixed time horizon t_N. This results in N state vectors $\mathbf{x}_k$ and N control vectors $\mathbf{u}_k$ for $k \in \{1, ..., N\}$. We assume that we have a reference state $\mathbf{x}_{r,k}$ and reference controls $\mathbf{u}_{r,k}$. While, we denote the state errors as $\Delta\mathbf{x}_k = \mathbf{x}_k - \mathbf{x}_{r,k}$ and the control errors as $\Delta\mathbf{u}_k = \mathbf{u}_k - \mathbf{u}_{r,k}$. We define our discrete objective as $\sum_{k=1}^{N} \Delta\mathbf{x}_k^T \mathbf{Q}_k \Delta\mathbf{x}_k + \sum_{k=1}^{N} \Delta\mathbf{u}_k^T \mathbf{R}_k \Delta\mathbf{u}_k$, where $\mathbf{Q}_k$ and $\mathbf{R}_k$ denote the state and control weight matrices.

In (1), the equality constraints represents the model for the robot dynamics $\frac{\partial \mathbf{x}_k}{\partial t} = \dot{\mathbf{x}}_k = f(\mathbf{x}_k, \mathbf{u}_k)$. While the inequality constraints $h(\mathbf{x}_k, \mathbf{u}_k)$ represent the physical limitations of the robot platform. As a common practice, problem (1) is transcribed using multiple shooting technique. Moreover, having the discrete dynamics and constraints over the coarse grid $[t_1, ..., t_N]$ for each time interval $[t_k, t_{k+1}]$, $\delta t = t_{k+1} - t_k$, a boundary value problem is solved, where additional continuity variables are imposed. In addition, an explicit integrator is applied to forward simulate the system dynamics along each interval $[t_k, t_{k+1}]$.

Under the above considerations, usually (1) is sequentially approximated by quadratic problems (QPs). The solution of the QPs are used as an gradient directions $\Delta\mathbf{x}_k$ and $\Delta\mathbf{u}_k$ in order to take steps that minimize the original continuous problem. Starting from the available guess (predictions) $\mathbf{x}_k^{pr}$ and $\mathbf{u}_k^{pr}$, the iterations are repeated by taking (not necessarily full) Newton steps in the form $\begin{bmatrix} \mathbf{x}^{pr} \\ \mathbf{u}^{pr} \end{bmatrix} = \begin{bmatrix} \mathbf{x}^{pr} \\ \mathbf{u}^{pr} \end{bmatrix} + \alpha \begin{bmatrix} \Delta\mathbf{x} \\ \Delta\mathbf{u} \end{bmatrix}$, where $\Delta\mathbf{x} = \begin{bmatrix} \Delta\mathbf{x}_1 \\ \vdots \\ \Delta\mathbf{x}_{N+1} \end{bmatrix}, \Delta\mathbf{u} = \begin{bmatrix} \Delta\mathbf{u}_1 \\ \vdots \\ \Delta\mathbf{u}_N \end{bmatrix}$ and α is the step size which guarantees that the update is in the decent direction. As an example, in the sequential QP approach, given a state estimate $\mathbf{x}_1^m$, a prediction about the state $\mathbf{x}_k^{pr}$ and the control $\mathbf{u}_k^{pr}$, the usual approximation of (1) with respect to $\Delta\mathbf{x}_k$ and

$\Delta\mathbf{u}_k$ is the following QP:

$$\begin{aligned}\left[\Delta\hat{\mathbf{x}},\Delta\hat{\mathbf{u}}\right]=&\arg\min_{\Delta\mathbf{x},\Delta\mathbf{u}}\sum_{k=1}^{N}\left(\left[\begin{smallmatrix}\Delta\mathbf{x}_k\\ \Delta\mathbf{u}_k\end{smallmatrix}\right]^T\mathbf{H}_k\left[\begin{smallmatrix}\Delta\mathbf{x}_k\\ \Delta\mathbf{u}_k\end{smallmatrix}\right]+\alpha\mathbf{w}_k^T\left[\begin{smallmatrix}\Delta\mathbf{x}_k\\ \Delta\mathbf{u}_k\end{smallmatrix}\right]\right),\\ &\text{subject to }\Delta\mathbf{x}_1=\mathbf{x}_1^m-\mathbf{x}_1^{pr},\\ &\qquad\Delta\mathbf{x}_{k+1}=\Delta\mathbf{r}_k+\left[\begin{smallmatrix}\mathbf{A}_k,\mathbf{0}\\ \mathbf{0},\mathbf{B}_k\end{smallmatrix}\right]\left[\begin{smallmatrix}\Delta\mathbf{x}_k\\ \Delta\mathbf{u}_k\end{smallmatrix}\right],\\ &\qquad\Delta\mathbf{h}_k+\left[\begin{smallmatrix}\mathbf{C}_k,\mathbf{0}\\ \mathbf{0},\mathbf{D}_k\end{smallmatrix}\right]\left[\begin{smallmatrix}\Delta\mathbf{x}_k\\ \Delta\mathbf{u}_k\end{smallmatrix}\right]\leq\mathbf{0},\end{aligned}\tag{2}$$

where $\Delta\mathbf{r}_k = f(\mathbf{x}_k^{pr},\mathbf{u}_k^{pr})-\mathbf{x}_{k+1}^{pr}$, $\Delta\mathbf{h}_k = h(\mathbf{x}_k^{pr},\mathbf{u}_k^{pr})$. While $\mathbf{A}_k=\frac{\partial f(\mathbf{x}_k,\mathbf{u}_k)}{\partial\mathbf{x}_k}|_{\{\mathbf{x}_k^{pr},\mathbf{u}_k^{pr}\}}$, $\mathbf{B}_k=\frac{\partial f(\mathbf{x}_k,\mathbf{u}_k)}{\partial\mathbf{u}_k}|_{\{\mathbf{x}_k^{pr},\mathbf{u}_k^{pr}\}}$, $\mathbf{C}_k=\frac{\partial h(\mathbf{x}_k,\mathbf{u}_k)}{\partial\mathbf{x}_k}|_{\{\mathbf{x}_k^{pr},\mathbf{u}_k^{pr}\}}$ and $\mathbf{D}_k=\frac{\partial h(\mathbf{x}_k,\mathbf{u}_k)}{\partial\mathbf{u}_k}|_{\{\mathbf{x}_k^{pr},\mathbf{u}_k^{pr}\}}$. Where $\frac{\partial f(\mathbf{x}_k,\mathbf{u}_k)}{\partial\mathbf{x}_k}|_{\{\mathbf{x}_k^{pr},\mathbf{u}_k^{pr}\}}$ denoted the partial derivative of $f(\mathbf{x}_k,\mathbf{u}_k)$ with respect to $\mathbf{x}_k$, which is evaluated at $\{\mathbf{x}_k^{pr},\mathbf{u}_k^{pr}\}$, $\mathbf{H}_k$ is the Hessian of the Lagrangian for (1) and $\mathbf{w}_k=\left[\begin{smallmatrix}\mathbf{Q}_k,\mathbf{0}\\ \mathbf{0},\mathbf{R}_k\end{smallmatrix}\right]\mathbf{l}_k=\left[\begin{smallmatrix}\mathbf{Q}_k,\mathbf{0}\\ \mathbf{0},\mathbf{R}_k\end{smallmatrix}\right]\left[\begin{smallmatrix}\mathbf{x}_k^{pr}-\mathbf{x}_{r,k}\\ \mathbf{u}_k^{pr}-\mathbf{u}_{r,k}\end{smallmatrix}\right]$. One popular approximation to the Hessian $\mathbf{H}_k$ is $\mathbf{H}_k=\left[\begin{smallmatrix}\mathbf{Q}_k,\mathbf{0}\\ \mathbf{0},\mathbf{R}_k\end{smallmatrix}\right]$ [12]. We note that in order to simplify the problem description (2), we omitted the cost related to the last state prediction, but nonetheless, we are taking it into account.

III. ONLINE WEIGHT-ADAPTIVE NMPC

In this section, we present the problem formulation for our online weight-adaptive NMPC. Then, we present our two-stage alternating algorithm. Afterward, we explain and discuss the related problems at each stage. Finally, we give connections to known learning principles.

A. Min-Max Approximate Control Problem

We propose to jointly (i) predict the control, and state variables and (ii) estimate the weight matrix. To that end, we present the following problem formulation:

$$\begin{aligned}\left[\Delta\hat{\mathbf{x}},\Delta\hat{\mathbf{u}},\hat{\mathbf{Q}}\right]=&\arg\max_{\mathbf{Q}}\Big\{-\lambda g(\mathbf{Q},\boldsymbol{\theta})+\\ &\min_{\Delta\mathbf{x},\Delta\mathbf{u}}\sum_{k=1}^{N}\left(\left[\begin{smallmatrix}\Delta\mathbf{x}_k\\ \Delta\mathbf{u}_k\end{smallmatrix}\right]+\alpha\mathbf{l}_k\right)^T\left[\begin{smallmatrix}\mathbf{Q}_k,\mathbf{0}\\ \mathbf{0},\mathbf{R}_k\end{smallmatrix}\right]\left[\begin{smallmatrix}\Delta\mathbf{x}_k\\ \Delta\mathbf{u}_k\end{smallmatrix}\right]\Big\},\\ &\text{subject to }\Delta\mathbf{x}_1=\mathbf{x}_1^m-\mathbf{x}_1^{pr},\\ &\qquad\Delta\mathbf{x}_{k+1}=\Delta\mathbf{r}_k+\left[\begin{smallmatrix}\mathbf{A}_k,\mathbf{0}\\ \mathbf{0},\mathbf{B}_k\end{smallmatrix}\right]\left[\begin{smallmatrix}\Delta\mathbf{x}_k\\ \Delta\mathbf{u}_k\end{smallmatrix}\right],\\ &\qquad\Delta\mathbf{h}_k+\left[\begin{smallmatrix}\mathbf{C}_k,\mathbf{0}\\ \mathbf{0},\mathbf{D}_k\end{smallmatrix}\right]\left[\begin{smallmatrix}\Delta\mathbf{x}_k\\ \Delta\mathbf{u}_k\end{smallmatrix}\right]\leq\mathbf{0},\end{aligned}\tag{3}$$

where $\Delta\mathbf{x}$ and $\Delta\mathbf{u}$ together with $\mathbf{Q}=[\mathbf{Q}_1,...,\mathbf{Q}_N]$ are problem variables that we would like to optimally estimate, while λ is the Lagrangian variable. In general, $g(\mathbf{Q},\boldsymbol{\theta})$ could be any weighs related cost function, and $\boldsymbol{\theta}$ is its corresponding parameter. In the simplest form, we consider diagonal $\mathbf{Q}_k$, and we define $g(\mathbf{Q},\boldsymbol{\theta})=(\mathbf{Q}\mathbf{1})^T(\mathbf{Q}\mathbf{1})$, where $\mathbf{1}$ is one vector.

Our proposed formulation is a min-max problem with quadratic and bilinear cost functions. If we fix the weights, the reduced problem over the remaining state and control variables is convex. On the other hand, if we fix the state and control variables then the reduced problem over the weight variables can be converted again into a convex problem.

B. The Algorithm

To solve (3), we propose an alternating algorithm, which consists of two stages. In stage one, we fix the weights and predict the state and control variables. In stage two, we fix the state and control variables and estimate the weights. In the following, we describe the corresponding problems for the two stages and discuss on their solution.

1) State and Control Prediction: Let the weights be fixed, then problem (3) reduces to the following QP:

$$\begin{aligned}\left[\Delta\hat{\mathbf{x}},\Delta\hat{\mathbf{u}}\right]=&\arg\min_{\Delta\mathbf{x},\Delta\mathbf{u}}\sum_{k=1}^{N}\left(\left[\begin{smallmatrix}\Delta\mathbf{x}_k\\ \Delta\mathbf{u}_k\end{smallmatrix}\right]+\alpha\mathbf{l}_k\right)^T\left[\begin{smallmatrix}\mathbf{Q}_k,\mathbf{0}\\ \mathbf{0},\mathbf{R}_k\end{smallmatrix}\right]\left[\begin{smallmatrix}\Delta\mathbf{x}_k\\ \Delta\mathbf{u}_k\end{smallmatrix}\right],\\ &\text{subject to }\Delta\mathbf{x}_1=\mathbf{x}_1^m-\mathbf{x}_1^{pr},\\ &\qquad\Delta\mathbf{x}_{k+1}=\mathbf{r}_k+\left[\begin{smallmatrix}\mathbf{A}_k\\ \mathbf{B}_k\end{smallmatrix}\right]\left[\begin{smallmatrix}\Delta\mathbf{x}_k\\ \Delta\mathbf{s}_k\end{smallmatrix}\right],\\ &\qquad\Delta\mathbf{h}_k+\left[\begin{smallmatrix}\mathbf{C}_k\\ \mathbf{D}_k\end{smallmatrix}\right]\left[\begin{smallmatrix}\Delta\mathbf{x}_k\\ \Delta\mathbf{u}_k\end{smallmatrix}\right]\leq\mathbf{0}.\end{aligned}\tag{4}$$

Since we have quadratic losses, and linear equality and inequality constraints, problem (4) represents a convex quadratic program with linear constraints. Problem (4) is well known and explored [5], while for a possible solver, we refer to [5], [12]. After (4) is solved, as a prediction for the state we use $\mathbf{x}^{pr}=\mathbf{x}^{pr}+\alpha\Delta\hat{\mathbf{x}}$, while as as a prediction for the control we use $\mathbf{u}^{pr}=\mathbf{u}^{pr}+\alpha\Delta\hat{\mathbf{u}}$.

2) Weights Update: This stage enables us to adapt our cost function for taking actions by adjusting and updating the weigh. In turn, this allows us to penalize future errors based on the error between the reference and the currently predicted state. To do so, we fix the control and state variables in (3) and let $\mathbf{q}=\mathbf{Q}_1\mathbf{1}=...=\mathbf{Q}_N\mathbf{1}$. In the simplest form, we define $g(\mathbf{Q})$ as $g(\mathbf{Q},\boldsymbol{\theta})=(\mathbf{Q}\mathbf{1})^T(\mathbf{Q}\mathbf{1})^2$. By denoting $\mathbf{v}_k=(\Delta\mathbf{x}_k+\alpha\mathbf{l}_k)\odot\Delta\mathbf{x}_k$, (3) reduces to the following quadratic problem:

$$\hat{\mathbf{q}}=\arg\min_{\mathbf{q}}\lambda\mathbf{q}^T\mathbf{q}-\sum_{k=1}^{N_s}\mathbf{v}_k^T\mathbf{q}.\tag{5}$$

where N_s, $N_s\leq N$, is the sub-horizon for the weight update. The main advantage of (5) is that it has a closed form solution, *i.e.*, $\hat{\mathbf{q}}=\frac{1}{2\lambda+\gamma}\left(\sum_{k=1}^{N_s}\mathbf{v}_k\right)$. Having the estimated $\hat{\mathbf{q}}$, we update $\mathbf{Q}$ as $\mathbf{Q}=diag(\hat{\mathbf{q}})$.

We point out that the vector $\frac{1}{2\lambda+\gamma}\sum_{k=1}^{N_s}\mathbf{v}_k$ is with non-negative values as long as $\sum_{k=1}^{N_s}\Delta\mathbf{x}_k\geq\sum_{k=1}^{N_s}\alpha(\mathbf{x}_k^{pr}-\mathbf{x}_{r,k})$. Therefore, under bounded variations $(\mathbf{x}_k^{pr}-\mathbf{x}_{r,k})$, we can ensure that our weight matrix $\mathbf{Q}$ is positive definite. While, under arbitrarily variations a non-negativity[3] constraint in (5) could be used to ensure that $\mathbf{Q}$ is positive definite.

C. Connection to Known Learning Paradigms

Our online weight update approach also represents a special case of metric learning [9], wherein our case, the loss $\mathcal{L}_a(\mathbf{x}_k,\mathbf{u}_k)$ represents the metric, which is included in

[2]It is worthwhile to mention that with $g(\mathbf{Q},\boldsymbol{\theta})$ we can consider a wide range of parametric function. Meaning that given data, we could also offline learn and estimate the parameters $\boldsymbol{\theta}$ in our function $g(\mathbf{Q},\boldsymbol{\theta})$.

[3]Note that if we include non-negativity constraint in (5), the closed from solution for $\mathbf{q}$ reads as $\hat{\mathbf{q}}=\max\left(\frac{1}{2\lambda+\gamma}\left(\sum_{k=1}^{N_s}\mathbf{v}_k\right),\mathbf{0}\right)$.

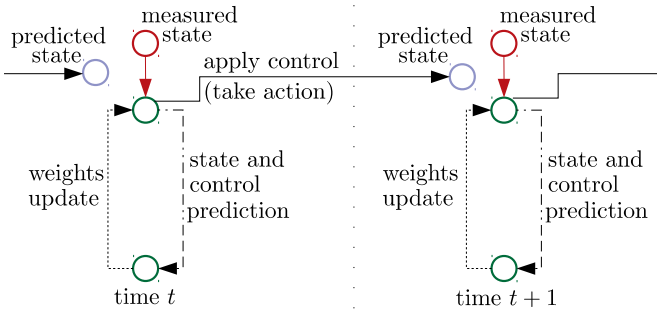

Fig. 2. Schematic diagram for the execution stages in our algorithm. In the first stage, we predict the state and control variables. In the second stage, we update the weights. After a number of alternating steps between the two stages, we take action by applying the first control prediction.

Algorithm 1 Online Weight-Adaptive NMPC

repeat
  Execute **Stage 1**
  Execute **Stage 2**
until *convergence*
Stage 1
  Input $\mathbf{x}_1^m, \mathbf{x}_r, \mathbf{x}^{pr}, \mathbf{u}_r, \mathbf{u}^{pr}$ and $\mathbf{Q}$
    $\left[\begin{smallmatrix}\Delta\mathbf{x}\\ \Delta\mathbf{u}\end{smallmatrix}\right] \leftarrow \text{updateDirection}(\mathbf{x}_1^m, \mathbf{x}_r, \mathbf{x}^{pr}, \mathbf{u}_r, \mathbf{u}^{pr}, \mathbf{Q})$
    $\left[\begin{smallmatrix}\mathbf{x}^{pr}\\ \mathbf{u}^{pr}\end{smallmatrix}\right] = \left[\begin{smallmatrix}\mathbf{x}^{pr}\\ \mathbf{u}^{pr}\end{smallmatrix}\right] + \alpha \left[\begin{smallmatrix}\Delta\mathbf{x}\\ \Delta\mathbf{u}\end{smallmatrix}\right]$
  Output $\mathbf{x}^{pr}$ and $\mathbf{u}^{pr}$
Stage 2
  Input $\mathbf{x}^{pr}, \mathbf{u}^{pr}, \mathbf{x}_r$ and $\mathbf{u}_r$
    $\mathbf{Q} \leftarrow \text{updateWeight}(\mathbf{x}_r, \mathbf{u}_r, \mathbf{x}^{pr}, \mathbf{u}^{pr})$
  Output $\mathbf{Q}$

(3). In the following, we give connections to metric learning, online learning, and reinforcement learning.

1) Metric Learning: The defined cost function for predicting the state is is the general Mahalanobis distance metric $(\mathbf{x}-\mathbf{x}_r)^T\mathbf{Q}(\mathbf{x}-\mathbf{x}_r)$. The online update of our distance metric under $\mathbf{Q}=\mathbf{L}^T\mathbf{L}$ with $\mathbf{L}$ is equivalent to learning a linear mapping (given by $\mathbf{x}\rightarrow\mathbf{L}\mathbf{x}$) that transforms the data in the space of $\mathbf{L}$. After projecting the data onto the new space through the linear map $\mathbf{L}$, the corresponding distance metric is the usual Euclidean distance.

2) Online Learning: Our algorithm can be viewed as an extension of the online learning approach to model predictive control [16]. Online learning concerns iterative interactions between a learner and an environment over several rounds N. At round (or time instance) t, the learner picks one out of the set of decisions. The environment then evaluates a loss function based on the learner's decision, and the learner suffers a cost. The learner's goal is to minimize the accumulated costs. As shown in [16], at time t (*i.e.*, round t), an MPC algorithm optimizes a controller (*i.e.*, the decision) over some cost function (*i.e.*, the per-round loss). In this regard, we highlight the following connection to online adaptation and learning. Our alternating algorithm, observes the cost of the initial controller and then improves the controller by updating the cost and the controller, and only then executes a control based on the improved controller.

TABLE I

THE TOTAL POSITION ERROR e AND THE CUMULATIVE TOTAL VARIATION TV OVER THE PREDICTED CONTROLS IN THE EXECUTION OF THE TRAJECTORIES UNDER VARYING λ.

		λ			
		0.01	0.67	1.67	3.00
		e[m] | TV	e[m] | TV	e[m] | TV	e[m] | TV
T_1	[7]	**0.44** | **1.57**	**0.66** | **1.29**	1.9 | 1.21	**0.87** | **1.19**
	N_2	0.76 | 2.25	0.95 | 1.58	1.55 | 1.56	1.93 | 1.58
	N_8	0.78 | 2.26	1.19 | 1.52	**1.36** | **1.43**	1.70 | 1.38
	N_{12}	0.79 | 1.89	1.39 | 1.48	1.58 | **1.43**	1.82 | 1.41
T_2	[7]	**0.37** | **0.72**	0.64 | **0.67**	1.16 | **0.64**	1.85 | **0.63**
	N_2	0.49 | 0.77	**0.58** | 0.75	**0.68** | 0.69	**0.80** | 0.67
	N_8	0.5 | 0.73	0.60 | 0.73	0.72 | 0.68	0.84 | 0.66
	N_{12}	0.52 | **0.72**	0.62 | 0.72	0.75 | 0.68	0.88 | 0.65
T_3	[7]	**0.34** | **0.03**	0.60 | **0.02**	2.11 | **0.02**	4.49 | **0.02**
	N_2	1.22 | 0.38	0.98 | 0.04	1.89 | 0.04	3.87 | 0.03
	N_8	0.94 | 0.20	0.73 | 0.04	1.82 | 0.03	3.81 | 0.03
	N_{12}	0.68 | 0.86	**0.57** | 0.04	**1.76** | 0.03	**3.72** | 0.03
T_4	[7]	**0.27** | **0.08**	4.63 | **0.05**	14.6 | **0.05**	28.7 | **0.04**
	N_2	0.38 | 0.21	2.56 | 0.12	**6.42** | 0.11	12.4 | 0.28
	N_8	0.29 | 0.11	2.43 | 0.08	6.33 | 0.12	12.2 | 0.19
	N_{12}	**0.27** | 0.46	**2.03** | 0.46	5.87 | 0.14	**11.9** | 0.31

3) Reinforcement Learning: Regarding the connection of our algorithm to the core principle behind reinforcement learning, we have the following. Upon observing a measurement, in the first stage of our algorithm, we generate state and control prediction. In light of reinforcement learning, we consider this as a sample from some underlining control policy. Afterward, in stage two, we update the weights. Thus our cost metric translates into updating the policy after observing the error between the predicted and reference state.

IV. NUMERICAL EVALUATION

In this section, we evaluate our approach. Our numerical experiments consider trajectory execution for a quadrotor. Therefore, in the following subsection, we first present the used dynamical model for quadrotor control. Afterward, we describe the experimental setup and discuss the results.

A. Used Model

In the following, we describe the used dynamical model.

1) Quadrotor Dynamics: Our state $\mathbf{x}$ and control $\mathbf{u}$ vectors of the quadrotor are defined as $\mathbf{x}=\left[\begin{smallmatrix}\mathbf{p}_{WB}\\ \mathbf{v}_{WB}\\ \mathbf{q}_{WB}\end{smallmatrix}\right]$ and $\mathbf{u}=[\,\overset{c}{\boldsymbol{\omega}_B}\,]$, where $\mathbf{p}_{WB}=[p_x,p_y,p_z]^T$ and $\mathbf{q}_{WB}=[q_w,q_x,q_y,q_z]^T$ denote the position and the orientation of the body frame B with respect to the world frame W, expressed in world frame, respectively. While $\mathbf{v}_{WB}=[v_x,v_y,v_z]^T$ denotes the linear velocity of the body, expressed in world frame, and $\boldsymbol{\omega}_B=[\omega_x,\omega_y,\omega_z]^T$ its angular velocity, expressed in the body frame. The vector $\mathbf{c}=[0,0,c]^T$ is the mass-normalized thrust vector, where $c=(f_1+f_2+f_3+f_4)/m$, f_i is the thrust produced by the i-th motor, and m is the mass of the vehicle. We define the dynamical model for the quadrotor as $\frac{\partial\mathbf{x}}{\partial t}=f(\mathbf{x},\mathbf{u})=\left[\begin{smallmatrix}\frac{\partial\mathbf{p}_{WB}}{\partial t}\\ \frac{\partial\mathbf{v}_{WB}}{\partial t}\\ \frac{\partial\mathbf{q}_{WB}}{\partial t}\end{smallmatrix}\right]=\left[\begin{smallmatrix}\mathbf{v}_{WB}\\ {}_W\mathbf{g}+\mathbf{q}_{WB}\odot\mathbf{c}\\ \frac{1}{2}\boldsymbol{\Lambda}(\boldsymbol{\omega}_B)\mathbf{q}_{WB}\end{smallmatrix}\right]$,

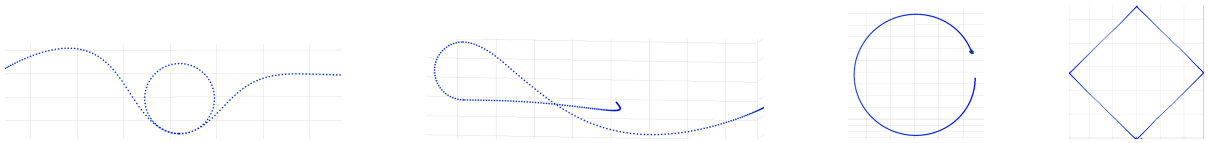

Fig. 3. Visualization of the used trajectories.

TABLE II

THE TOTAL POSITION ERROR e AND THE CUMULATIVE TOTAL VARIATION TV OVER THE PREDICTED CONTROLS IN THE EXECUTION OF THE TRAJECTORIES OVER VARYING HORIZON LENGTH N.

		N = 8	N = 14	N = 19	N = 24
		e[m] ǀ TV	e[m] ǀ TV	e[m] ǀ TV	e[m] ǀ TV
T_1	[7]	12.0 ǀ **1.24**	2.22 ǀ **1.38**	**0.98** ǀ **1.29**	**0.64** ǀ **1.34**
	N_8	**1.49** ǀ 1.38	**1.38** ǀ 1.49	1.13 ǀ 1.56	1.22 ǀ 1.55
	N_{14}	0 ǀ 0	1.61 ǀ 1.58	1.45 ǀ 1.53	1.22 ǀ 1.55
	N_{18}	0 ǀ 0	0 ǀ 0	1.28 ǀ 1.55	1.00 ǀ 1.56
T_2	[7]	3.03 ǀ 0.64	1.19 ǀ **0.66**	0.71 ǀ **0.67**	**0.47** ǀ **0.68**
	N_8	**0.97** ǀ 0.69	0.73 ǀ 0.71	**0.62** ǀ 0.72	0.53 ǀ 0.74
	N_{14}	0 ǀ 0	**0.75** ǀ 0.70	0.64 ǀ 0.72	0.55 ǀ 0.73
	N_{18}	0 ǀ 0	0 ǀ 0	0.65 ǀ 0.72	0.56 ǀ 0.73
T_3	[7]	12.75 ǀ **0.03**	2.73 ǀ **0.02**	0.72 ǀ **0.02**	**0.45** ǀ **0.02**
	N_8	**4.05** ǀ 0.05	1.79 ǀ 0.04	0.86 ǀ 0.04	0.52 ǀ 0.04
	N_{14}	0 ǀ 0	**1.17** ǀ 0.07	0.66 ǀ 0.04	0.50 ǀ 0.04
	N_{18}	0 ǀ 0	0 ǀ 0	**0.61** ǀ 0.92	0.51 ǀ 0.10
T_4	[7]	59.62 ǀ 0.06	15.7 ǀ **0.05**	5.73 ǀ **0.05**	2.07 ǀ **0.05**
	N_8	**24.69** ǀ 0.07	6.90 ǀ 0.08	2.94 ǀ 0.08	1.44 ǀ 0.08
	N_{14}	0 ǀ 0	**5.48** ǀ 0.08	2.27 ǀ 0.56	1.19 ǀ 0.42
	N_{18}	0 ǀ 0	0 ǀ 0	**1.95** ǀ 0.74	**1.15** ǀ 0.73

TABLE III

THE TOTAL POSITION ERROR e AND THE CUMULATIVE TOTAL VARIATION TV OVER THE PREDICTED CONTROLS IN THE EXECUTION OF THE TRAJECTORIES UNDER NOISE PERTURBATION WITH NOISE LEVEL σ.

		σ = 0.5	σ = 2.0	σ = 3.5	σ = 5.0
		$e_r[m]$	$e_r[m]$	$e_r[m]$	$e_r[m]$
T_1	[7]	**1.32**	**4.48**	8.04	12.32
	N_8	1.65	4.55	**7.58**	**11.58**
T_2	[7]	1.25	4.79	8.23	12.48
	N_8	**1.13**	**4.31**	**7.46**	**11.21**
T_3	[7]	1.77	5.38	9.04	12.87
	N_8	**1.44**	**4.87**	**8.95**	**12.57**
T_4	[7]	3.95	5.07	**8.72**	12.74
	N_8	**1.3**	**4.8**	8.85	**12.18**

where $\frac{\partial \mathbf{p}_{WB}}{\partial t}$, $\frac{\partial \mathbf{v}_{WB}}{\partial t}$ and $\frac{\partial \mathbf{q}_{WB}}{\partial t}$ are the time derivatives of the position, liner velocity and the quaternion, while ${}_W\mathbf{g} = [0, 0, -g]$ is the gravity vector, with $g = 9.81m/s$. The operator $\odot$ denotes the multiplication between a quaternion and a vector, $\mathbf{\Lambda}(\boldsymbol{\omega}_B)\mathbf{q}_{WB}$ denotes the time derivative of a quaternion $\mathbf{q}_{WB}$, while $\mathbf{\Lambda}(\boldsymbol{\omega}_B)$ is the the skew-symmetric matrix of the vector $\boldsymbol{\omega}_B$.

2) Quadrotor Physical Constraints: By the inequality constraint $h(\mathbf{x}, \mathbf{u})$, we model the physical limitations of the drone platform in order to attain feasible solutions. In our case it is the minimum and maximum thrust c_{min} and c_{max}, as well as minimum and maximum angular velocities $\boldsymbol{\omega}_{min}$ and $\boldsymbol{\omega}_{max}$, respectively, which we compactly express as $\mathbf{u}_{min} = \left[\begin{smallmatrix} c_{min} \\ \boldsymbol{\omega}_{min} \end{smallmatrix}\right]$ and $\mathbf{u}_{max} = \left[\begin{smallmatrix} c_{max} \\ \boldsymbol{\omega}_{max} \end{smallmatrix}\right]$.

B. Setup, Error Measures and Comparative Analysis

We generated four different trajectories, which were computed as proposed in [10]. As shown in Figure 3, the trajectories have different geometries. The first and the second trajectory are aggressive and are denoted as T_1 and T_2. The third trajectory T_3 is circle and the fourth trajectory T_4 is diamond.

We validated our approach under different setups. We experimented with different strength for the state costs in the control problem. As well as we experimented with different lengths of the fixed horizon and different lengths of the sub horizon, which were used to update the weight in the cost. In addition, we validated our approach under additive white Gaussian noise perturbation in the available state estimate. In summary, we present simulation results under:

(i) Different strength λ of the state cost,
(ii) Different length of the prediction horizon N and
(iii) Noise perturbation with different noise levels σ in the available (measured) state.

Over all trajectory points, we measure and report the total accumulation of error as $e = \sum_{i=1}^{L} d_i$, where $d_i = \| \left[\begin{smallmatrix} p_x \\ p_y \\ p_z \end{smallmatrix}\right] - \left[\begin{smallmatrix} p_{r,x} \\ p_{r,y} \\ p_{r,z} \end{smallmatrix}\right] \|_2$ represents the error between the simulated position $\left[\begin{smallmatrix} p_x \\ p_y \\ p_z \end{smallmatrix}\right]$ after applying the predicted control and the reference position $\left[\begin{smallmatrix} p_{r,x} \\ p_{r,y} \\ p_{r,z} \end{smallmatrix}\right]$. In addition, we also measure the total variation over commands, *i.e.*, TV $= \frac{1}{L}\sum_{i=1}^{L-1}\left(|T_i - T_{i+1}| + \sum_{j=1}^{3}|\omega_{j,i} - \omega_{j,i+1}|\right)$ and consider it as an indicator for "smooth" changes in the predicted commands during trajectory execution.

In the first two series of experiments for different strength λ of the state cost, and for different lengths of the prediction horizon N, we also experimented with different sub-horizon lengths N_s. Regarding the noise corruption protocol for the third set of experiments, we implement it as follows. We randomly selected the index $\tau \in \{1, ..., L\}$ for one point from the trajectory. Then, during execution, at the corresponding index τ, we corrupted the available state with Additive White Gaussian Noise (AWGN) $\boldsymbol{\nu}$ as follows $\mathbf{p}_\tau^{'} = \mathbf{p}_\tau + \sigma\boldsymbol{\nu}$, while we ensured that $\|\mathbf{p}_\tau\|_2 = \|\boldsymbol{\nu}\|_2$. In the same experiment, we compute average error as $e_r = \frac{1}{K}\sum_{k=1}^{K} e_k$ for $K = 15$

runs of this procedure. We compare our algorithm with the standard solution to NMPC with fixed weights, which we implemented using [7].

In the standard NMPC, the hole weight matrix $\mathbf{Q}$ has to be tuned. In contrast, in our approach the only tuning parameter is λ. During simulation, we also found out that $\hat{\mathbf{q}} = \exp\left(\frac{1}{2\lambda+\gamma}\left(\sum_{k=1}^{N_s}\mathbf{v}_k\right)\right)$, has better performance then $\hat{\mathbf{q}} = \frac{1}{2\lambda+\gamma}\left(\sum_{k=1}^{N_s}\mathbf{v}_k\right)$. Therefore, in all of the experiments, we update the weights as $\hat{\mathbf{q}} = \exp\left(\frac{1}{2\lambda+\gamma}\left(\sum_{k=1}^{N_s}\mathbf{v}_k\right)\right)$.

C. Results Discussion

In Tables I, II and III, we present the results of our computer simulation. We show the resulting trajectory tracking errors and total variation errors of our algorithm and the comparing method (the standard NMPC, with fixed weights).

As we can see in Table I, the total error in position is reduced as a result of a small increase in the total variation of the command prediction when compared to the common solution [7] of the NMPC. Moreover, as shown in Table I, for very small values of the λ parameter, the accuracy for the trajectory tracking of our algorithm is lower than the accuracy of the comparing algorithm [7]. However, we note that at such small values for λ, the changes in the predicted controls during the trajectory execution are not "smooth". In practice, very small λ might lead to potentially non-stable behavior. On the other hand for λ values above .5, *i.e.*, $\lambda > .5$, the changes in the predicted controls are more "smooth". At the same λ values, we report improved performance. Our approach achieves lower total error compared to the common solution of the standard NMPC. When the sub-horizon N_s[4] has larger length, the results error is lower, but the TV is also high.

Table II shows that the total error in position is consistently lower compared to [7] over different horizon lengths. It is interesting to highlight that even for low horizon length like $N = 8$, the algorithm achieves high tracking accuracy. While, both of the comparing algorithms have the lowest errors for horizons between 16 and 24.

In Table III, we can see that for a noise level in the range of $.5$ to 5, the average positional error e_r of the our algorithm is lower compared to the same error e_r for the standard NMPC [7].

As a summary, the simulation results demonstrated that by our approach, which is without manual weight tuning, we could archive accurate and stable quadrotor navigation. The execution of smooth as well as fast and rapidly changing reference trajectories benefits from online weigh adaptation. The results also show that we can have relatively good tracking performance even under a small horizon length. It is essential to point out that not all online weight update configurations are useful. In other words, not all algorithm setups (for different λ and N_s) provide improved accuracy with "smooth" changes in the predicted controls over the executed trajectories in the simulation.

[4]Our results are computed for sub-horizon length N_s smaller then the horizon length N, $N_s \leq N$.

V. CONCLUSIONS

In this paper, we presented a novel control problem formulation for NMPC with an online update of the cost weights. As a solution, we proposed a two-stage alternating algorithm. It consists of: (i) state and commands variable prediction and (ii) optimal weights update. Our evaluation by computer simulation demonstrated not only high accuracy for trajectory tracking but also robustness to noise perturbation. Comparing the solution of our approach to the solution of the common NMPC with fixed weights, we demonstrated lower tracking error for the used reference trajectories. Our next steps are to test the performance on a real drone platform.

REFERENCES

[1] F. Allgöwer and A. Zheng. Nonlinear model predictive control. *Progress in Systems and Control Theory*, 2000.

[2] A. An, X. Hao, Q. Wang, and C. Ren. Adaptive weights robust predictive zone control and its application for distillation column control. In *Proc. International Conference on Future Information Technology and Management Engineering*, pages 471–475, 2009.

[3] R. Bellman. Some applications of the theory of dynamic programming - a review. *Operations Research*, 2(3):275–288, 1954.

[4] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos. The explicit linear quadratic regulator for constrained systems. *Journal of Automatica*, 39(10):1845–1846, 2003.

[5] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[6] MM. Diehl, R. Findeisen, F. Allgöwer, H.G. Bock, and JP. Schlöder. Nominal stability of real-time iteration scheme for nonlinear model predictive control. *Proc. Control Theory and Applications*, 152, 2012.

[7] B. Houska, H.J. Ferreau, and M. Diehl. Acado toolkit – an open source framework for automatic control and dynamic optimization. *Optimal Control Applications and Methods*, 32:289–312, 2011.

[8] A. Karl Johan and T. Hägglund. *PID Controllers: Theory, Design, and Tuning*. The Instrumentation, Systems and Automation Society, 1995.

[9] B. Kulis. Metric learning: A survey. *Foundations and Trends in Machine Learning*, 5(4):287–364, 2013.

[10] D. Mellinger and V. Kumar. Minimum snap trajectory generation and control for quadrotors. In *Proc. IEEE International Conference on Robotics and Automation*, pages 2520–2525, 2011.

[11] S. Joe Qin and Thomas A. Badgwell. An overview of nonlinear model predictive control applications. In *Nonlinear Model Predictive Control*, pages 369–392, 2000.

[12] G. Sebastien, G. Mario, Z. Mario, Z. Rien, Q. Rien, and Q. Show. From linear to nonlinear mpc: bridging the gap via the real-time iteration. *International Journal of Control*, 2016.

[13] S. Shalev-Shwartz and S. Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, USA, 2014.

[14] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 2018.

[15] M. Turki, N. Langlois, and A. Yassine. An Analytical Tuning of MPC Control Horizon Using the Hessian Condition Number. *Proc. International Workshop on Advanced Control and Diagnosis*, 2017.

[16] N. Wagener, C.-An Cheng, J. Sacks, and B. Boots. An online learning approach to model predictive control. *Proc. Conference on Robot Learning*, 2019.

[17] M. Bujarbaruah X. Zhang and F. Borrelli. Adaptive mpc with chance constraints for fir systems. *Proc. Annual American Control Conference*, pages 2312–2317, 2018.

[18] R. C. Zhao, P. K. Wong, Z. C. Xie, and J. Zhao. Real-time weighted multi-objective model predictive controller for adaptive cruise control systems. *International Journal of Automotive Technology*, 18:1976–3832, 2017.