Linearized Gaussian Processes for Fast Data-driven Model Predictive Control

Truong X. Nghiem
School of Informatics, Computing, and Cyber Systems
Northern Arizona University
truong.nghiem@nau.edu

Abstract—Data-driven Model Predictive Control (MPC), where the system model is learned from data with machine learning, has recently gained increasing interests in the control community. Gaussian Processes (GP), as a type of statistical models, are particularly attractive due to their modeling flexibility and their ability to provide probabilistic estimates of prediction uncertainty. GP-based MPC has been developed and applied, however the optimization problem is typically non-convex and highly demanding, and scales poorly with model size. This causes unsatisfactory solving performance, even with state-of-the-art solvers, and makes the approach less suitable for real-time control. We develop a method based on a new concept, called linearized Gaussian Process, and Sequential Convex Programming, that can significantly improve the solving performance of GP-based MPC. Our method is not only faster but also much more scalable and predictable than other commonly used methods, as it is much less influenced by the model size. The efficiency and advantages of the algorithm are demonstrated clearly in a numerical example.

I. INTRODUCTION

Model predictive control (MPC) is an advanced control approach that utilizes a mathematical model of the controlled process to predict its future responses over a finite time horizon, then minimizes a cost function to obtain optimized control inputs subject to input and state constraints. This procedure is repeated at every time step, resulting in an optimal control strategy. MPC is attractive due to its ability to effectively and intuitively handle complex dynamics and system constraints [1]. Stochastic MPC (SMPC) can efficiently handle stochastic uncertainties in optimal control problems [2]. For these reasons, MPC and its variants have been applied widely in many practical control problems [3].

A major caveat of MPC is that it requires a reasonably accurate mathematical model of the system because its performance is highly dependent on the accuracy of the model-based predictions. Traditionally, these models are developed using first principles based on physics. However, for complex systems where knowledge of the system dynamics is often incomplete or hard to obtain fully, the required effort for such model development and engineering would pose significant difficulty and limit the use of physics-based models for MPC [4]. To overcome this challenge, data-driven modeling for MPC has recently been studied. This approach uses machine learning techniques to learn a system model from data, possibly with certain prior knowledge about the system, and uses that model in place of the traditional mathematical model in MPC. Such data-driven MPC approach has become increasingly appealing, especially for complex and large-scale systems, as a result of recent advancements in machine learning, optimization, and computation.

Gaussian Processes (GPs) – a type of statistical machine learning models – have been used for modeling dynamical systems and for MPC [5]. GPs are highly flexible and able to capture complex behaviors with fewer parameters than other machine learning techniques, hence they generally work well with small data sets [6]. More importantly, a GP provides an estimate of uncertainty or doubt in the predictions through the predictive variance, which can be used to assess or guarantee the performance of a learning-based system. Because of these advantages, GPs have been employed successfully in MPC frameworks and applications [4], [5], [7], [8]. In this paper, we will consider the GP-based MPC approach for data-driven predictive control, which we will call GP-MPC henceforth.

Computational complexity is a major challenge of GP-MPC. GP computations generally scale cubically with the training data size [6]. When the training data size increases, GP-MPC quickly becomes very expensive to solve. A nonlinear program (NLP) solver is usually used to solve GP-MPC, for instance by Interior-Point methods (IPM), Sequential Quadratic Programming (SQP) methods, and active-set methods [5]. However, the computational performance of these methods for GP-MPC is usually unsatisfactory [7].

Our goal is to make solving GP-MPC fast, scalable, and more predictable compared to other NLP methods. In particular, we aim to make solving GP-MPC less influenced by the training data size. We propose a new concept called linearized Gaussian Process (linGP), which is a (valid) Gaussian Process of the linearization of the latent function about a given input. This is not the same as linearizing the GP mean and variance because a linGP is still a GP and therefore it has mean, variance, and moments of any order. We then develop a Sequential Convex Programming (SCP) algorithm based on linGP to solve a general class of GP-MPC problems very efficiently and, most importantly, much more independently of the training data size compared to other methods. The proposed SCP algorithm is inspired by the successive convexification approach in [9] to solve non-convex optimal control problems and similar in concept to successive linearization methods to solve nonlinear MPC (such as in [10]). However, our approach is different from other attempts to accelerate GP-MPC and machine-learning-based MPC in general, for example [8], [10], in that we approximate not the output equation of the machine learning model but the underlying latent process of the model.

The concept of linGP is introduced in Section II, followed by the considered class of GP-MPC problems in Section III. We present the linGP-based SCP algorithm in Section IV. Finally, the efficacy of our method is demonstrated in a
II. LINEARIZED GAUSSIAN PROCESSES (LINGP)

This section introduces the concept of a linearized Gaussian Process (linGP) and develops its mathematical derivation. We will begin with a brief introduction to GPs and the GP-related notations used in this paper.

A. Gaussian Processes: Basics and Notations

Given $N$ noisy observations $y^{(i)}$ of an underlying function $f : \mathbb{R}^n \mapsto \mathbb{R}$ through a Gaussian noise model: $y^{(i)} = f(x^{(i)}) + \epsilon^{(i)},$ with inputs $x^{(i)} \in \mathbb{R}^n, \epsilon^{(i)} \sim \mathcal{N}(0, \sigma^2_{\epsilon}),$ and $i = 1, \ldots, N.$ A GP of $f,$ considered as a distribution over functions [6], is essentially a probability distribution on the observations of all possible realizations of $f.$ The prior $P(f)$ represents the initial belief about this distribution.

Let $\mathcal{D} = (X, Y)$ be the finite set of observation data of $f,$ where $X = [x^{(1)}, \ldots, x^{(N)}] \in \mathbb{R}^{n \times N}$ collects the regression vectors and $Y = [y^{(1)}, \ldots, y^{(N)}] \in \mathbb{R}^N$ contains the corresponding observed outputs. By conditioning the GP on $\mathcal{D},$ the posterior $P(f|\mathcal{D})$ is the updated distribution after seeing the observation data, which allows probabilistic inference at a new input.

The GP assigns a joint Gaussian distribution to any finite subset of random variables $\{x^{(1)}, \ldots, x^{(M)}\}$ corresponding to inputs $\{x^{(1)}, \ldots, x^{(M)}\}.$ It is fully specified by its mean function $m(x; \theta) = \mathbb{E}[f(x)]$ and covariance function $k(x, x'; \theta) = \mathbb{E}[(f(x) - m(x))(f(x') - m(x'))],$ parameterized by the hyperparameters $\theta.$ The mean function is often set to zero without loss of generality.

The output $f_x$ of the GP corresponding to an input $x$, is a random variable $f_x \sim \mathcal{N}(\mu_x, \sigma_x^2),$ where

$$\mu_x = m_x + K_x K^{-1}(X, \mu(X))$$

$$\sigma^2_x = K_x - K_x K^{-1} K_x$$

in which $K_x = [k(x, x^{(1)}), \ldots, k(x, x^{(N)})] \in \mathbb{R}^N,$ $K_{xx} = k(x, x) \in \mathbb{R},$ and $K \in \mathbb{R}^{N \times N}$ is the covariance matrix with $K_{ij} = k(x^{(i)}, x^{(j)}).$ Note that $\mu_x$ and $\sigma^2_x$ are nonlinear in $x$ and their computations scale cubically with $N.$

In theory, the hyperparameters $\theta$ are also random variables, whose posterior distributions are obtained by conditioning them on $\mathcal{D}$ using the Bayes’ theorem. In practice, however, $\theta$ are often obtained by maximizing the likelihood: $\arg\max_{\theta} \Pr(Y | X, \theta).$ For an in-depth treatment of GPs, interested readers are referred to [6].

B. Linearized Gaussian Processes

Consider a GP of a nonlinear function $f(x) : \mathbb{R}^n \mapsto \mathbb{R},$ with covariance function $k(x, x') : \mathbb{R}^n \times \mathbb{R}^n \mapsto \mathbb{R}.$ We will denote the GP of $f$ by $\mathcal{G}_f,$ whose predictive mean and variance at an input $x \in \mathbb{R}^n$ are given by (1) and are denoted by $\mu_{f|x}$ and $\sigma^2_{f|x},$ respectively. For notational simplicity we will take the mean function $m(x)$ to be zero. If the GP has a non-zero mean function, it is equivalent to the sum of $m(x)$ and a zero-mean GP, hence our results can be extended to this case.

The GP inference at any input $x,$ given by (1), is non-convex due to the covariance function $k(\cdot, \cdot).$ This makes any optimization involving GP regressions non-convex and generally difficult to solve. However, in many of these problems, non-convexity only comes from the GP regressions involved. One typical approach to overcome this issue is to approximate the mean and variance by linear or quadratic equations using Taylor’s series expansions. For instance, this approach is employed by SQP solvers. However, this approach often does not work well (see the full paper [12]). In this paper, we take a different approach, described below.

Given an input $x,$ we assume that $f$ is differentiable at $x.$ We also assume that the covariance function $k(\cdot, \cdot)$ is differentiable, which is typically the case [11]. Recall that the GP of $f$ is essentially a (posterior) distribution over the space of realizations of $f.$ We can then linearize the latent function $f$ around $x$ as $f(x + \Delta x) \approx f_x(\Delta x) = f(x) + \nabla f(x) \Delta x,$ for $\Delta x \in \mathbb{R}^n.$ Define $g = \nabla f$ as the gradient of $f.$ The above linearization can be rewritten as, with $\dot{x} = [1, \Delta x]^T,$

$$\dot{f}_x(\Delta x) = \begin{bmatrix} 1 \\ \Delta x \end{bmatrix}^T \begin{bmatrix} f(x) \\ g(x) \end{bmatrix} = \dot{x}^T \dot{f}(x).$$

Here, $\dot{f}(x) = [f(x), g(x)^T]^T$ is a random vector which, through the inner product with the vector $\dot{x},$ results in the random variable $\dot{f}_x(\Delta x)$ of the linearized process. In other words, we approximate the original GP of $f$ around $x$ by a stochastic process of linearized function $\dot{f}_x$ of $f,$ which we will call a linearized Gaussian Process or linGP. The linGP of the GP $\mathcal{G}_f$ at input $x$ is denoted by $\tilde{\mathcal{G}}_{f|x}.$

As differentiation is a linear operator, $f$ is a (multivariate) Gaussian Process derived from the original GP. Therefore, $\tilde{\mathcal{G}}_{f|x}$ has a distribution derived from $P(f|X,Y)$ over the linear approximations of the latent function $f$ near $x.$ To characterize this linGP, we need to characterize the GP of $f.$

Let $K^{(1,0)}(0, \delta) = (\nabla f, k)$ be the $n \times n$ matrix such that $K^{(1,1)}(x, x') = \frac{\partial^2}{\partial x \partial x'} k(x, x').$ Following [11], we can write the joint distribution of the observation data, the process $f,$ and the gradient process $g$ as

$$\begin{bmatrix} f \\ g \end{bmatrix} \sim \mathcal{N} \left( \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \mathbb{K}_{K^{(0,0)}(X,X) K^{(0,1)}(X,X) + K^{(1,0)}(X,X) K^{(1,1)}(X,X)} \right) \right).$$

Here, we use the convention that when a function is applied with $X,$ it is broadcast along the corresponding dimensions. For example, $K(x, X)$ is a $N \times N$ matrix with $K_{i,j}(X, X) = k(x^{(i)}, x^{(j)});$ similarly, the $i$th row of $K^{(0,1)}(X, x)$ is $K^{(0,1)}(x^{(i)}, x).$ The posterior distribution of $f$ conditioned on the observation data $(X, Y)$ is then

$$f|X, Y, x \sim \mathcal{N}\left( \hat{m}_x, \hat{V}_x \right)$$

$$\hat{m}_x = \left[ k(x, X) K^{(0,1)}(X, X) + \sigma^2_{\epsilon x} \right]^{-1} \left( X, \mu(X) \right)$$

$$\hat{V}_x = \left[ \left[ k(x, X) K^{(0,1)}(X, X) + \sigma^2_{\epsilon x} \right]^{-1} \left( X, \mu(X) \right) \right] K^{(1,0)}(X, X) \left[ k(x, X) K^{(0,1)}(X, X) + \sigma^2_{\epsilon x} \right]^{-1} \left( X, \mu(X) \right)$$

where $I$ denotes an identity matrix of appropriate dimensions. The linearized process in (2) can then be written as follows.

$$\dot{f}_x(\Delta x) \sim \mathcal{N}(\dot{m}_x(\Delta x), \dot{\sigma}^2_{f|x}(\Delta x))$$

$$\dot{m}_x(\Delta x) = \dot{m}_x^T \dot{x}, \quad \dot{\sigma}^2_{f|x}(\Delta x) = \dot{x}^T \dot{V}_x \dot{x}.$$
The mean of the original GP at \((x + \Delta x)\) is approximated by a linear function \(\tilde{\mu}_f|_{x}(\cdot)\) of \(\hat{x}\), while its variance is approximated by a quadratic function \(\tilde{\sigma}_f^2|_{x}(\cdot)\) of \(\hat{x}\). Because \(\hat{f}\) is a valid GP, \(\hat{V}_x\) is positive semi-definite (PSD) and, therefore, it is guaranteed that the variance of \(f_x(\Delta x)\) is non-negative. A key advantage of the linGP is that its mean and variance are respectively linear and convex quadratic functions.

**Remark 1:** The mean and variance of the predictive output of the GP can be approximated directly by Taylor’s series expansions, as commonly employed by SQP-based methods. A key distinction between such an approximation approach (1b) is approximated to the second order by a Taylor’s series. Taylor’s series approximation. As explained in Section IV-C of the full paper [12], this key advantage of the linGP approach could lead to a substantial performance improvement of our algorithm compared to other methods.

### III. GP-based Model Predictive Control

#### A. Gaussian Processes for Dynamical Systems

GPs can be used for modeling nonlinear dynamical systems [5]. This can be achieved by feeding autoregressive, or time-delayed, input and output signals back to the model as regressors. Specifically, it is common to model a discrete-time dynamical system by an autoregressive nonlinear function

\[
\tilde{y}_k = \tilde{f}(y_{k-1}, \ldots, y_{k-l}, u_{k-m}, \ldots, u_k).
\]

(7)

Here, \(k\) denotes the time step, \(u \in \mathbb{R}^n\) the control input, \(y \in \mathbb{R}\) the output, and \(l\) and \(m\) are respectively the lags for autoregressive outputs and control inputs. The regressor vector includes not only the current control input \(u_k\) but also the past values of \(y\) and \(u\). The nonlinear dynamical function \(\tilde{f}\) can be learned by a GP \(\tilde{G}_f\), trained from the system’s data in the same way as any other GPs. MPC formulations that use GP models of the involved dynamical systems will be called GP-MPCs in this paper. We assume that a single GP is used in a GP-MPC, however our results readily extend to the case when multiple GPs are involved.

Let \(H\) be the MPC horizon. Due to the predictive nature of MPC and the probabilistic nature of GPs, it is necessary to simulate the autoregressive GP model \(\tilde{G}_f\) over \(H\) future steps with propagation of the output uncertainty. There exist several methods for approximating uncertainty propagation in GPs [5]. However, it was shown in [7] that the zero-variance method, which replaces the autoregressive outputs with their corresponding expected values and therefore does not propagate uncertainty, could achieve sufficient prediction accuracy. Its computational simplicity is attractive, especially in GP-MPC where the GP must be simulated for many time steps. In this paper, we will assume the zero-variance method for predicting future GP outputs in MPC. Let us define the state vector \(x\) such that, at time step \(k\),

\[
x^T_k = [y_{k-1}, y_{k-l+1}, \ldots, y_{k-1}, u^T_{k-m}, u^T_{k-m+1}, \ldots, u^T_k]
\]

(8)

where \(y_i\) is the expected value of \(y_j\). Note that all \(\bar{y}_k\) for \(k < t\), where \(t\) is the current time, can be observed. The output of \(\tilde{G}_f\) at time \(k \geq t\) is then \(y_k \sim N(\bar{y}_k, \sigma^2_f, k)\) where \(\bar{y}_k = \mu_f(x_k, u_k)\), \(\sigma^2_f, k = \sigma^2_f(x_k, u_k)\).

#### B. Model Predictive Control with Gaussian Process Models

We consider a general GP-MPC formulation described below. Let \(t\) be the current time step. Define the collections \(\mathcal{Y}_t = \{\bar{y}_k | k \in \mathcal{I}_t\}\), \(\mathcal{S}_{y,t} = \{\sigma_{y,k} | k \in \mathcal{I}_t\}\), and \(\mathcal{U}_t = \{u_k | k \in \mathcal{I}_t\}\), where \(\mathcal{I}_t = \{t, \ldots, t + H - 1\}\) contains the indices of all time steps in the current MPC horizon. We also introduce additional variables \(z \in \mathbb{R}^n\), for example to model non-GP dynamics in the system, and define their collection \(\mathcal{Z}_t = \{z_k | k \in \mathcal{I}_t\}\). The GP-MPC reads

\[
\text{minimize}_{\mathcal{U}_t, \mathcal{Z}_t} \ J(\mathcal{Y}_t, \mathcal{S}_{y,t}, \mathcal{U}_t, \mathcal{Z}_t)
\]

(10a)

subject to

\[
y_k = \mu_f(x_k, u_k), \quad \sigma^2_{y, k} = \sigma^2_f(x_k, u_k)
\]

(10b)

\[
z_k \in Z_t, \quad u_k \in U_t
\]

(10c)

\[
y_i(\mathcal{Y}_t, \mathcal{S}_{y,t}, \mathcal{U}_t, \mathcal{Z}_t) \leq 0, \quad i = 1, \ldots, n_{\text{ineq}}
\]

(10d)

\[
h_i(\mathcal{Y}_t, \mathcal{S}_{y,t}, \mathcal{U}_t, \mathcal{Z}_t) = 0, \quad i = 1, \ldots, n_{\text{ineq}}
\]

(10e)

in which \(k\) ranges from \(t\) to \(t + H - 1\) and

- \(J(\cdot)\) is the objective function;
- Constraints (10b) represent the GP dynamics;
- Sets \(Z\) and \(U\) in (10c) are convex constraint sets of the variables \(z\) and \(u\), respectively;
- (10d) and (10e) are inequality and equality constraints.

We make the following assumption about the GP-MPC.

**Assumption 1:** Suppose that, for every \(k\), \(\bar{y}_k\) is affine and \(\sigma^2_{y,k}\) is convex quadratic in the control variables \(U_t\). Then \(\tilde{f}\) is convex, each \(g_i\) is convex, and each \(h_i\) is affine in the optimization variables \(U_t\) and \(Z_t\).

This assumption holds in many applications of GP-MPC because the non-convexity of (10) usually comes solely from the GP dynamics [4], [5], [7]. It also usually holds in stochastic GP-MPC with chance constraints approximated by convex deterministic constraints (see, e.g., [4], [7]).

### IV. LINGP-based Model Predictive Control

In this section, an algorithm based on linGP is developed to solve the GP-MPC problem (10) with significantly improved performance compared to general NLP solvers. More importantly, its complexity is much less influenced by the size of the GP model (i.e., the size of the training data set).

#### A. Local GP-MPC problem with linGP

Suppose that nominal feasible control inputs \(u^*_k \in U\), for \(k \in \mathcal{I}_t\), are given. One can then simulate the GP model \(\tilde{G}_f\) to obtain the nominal outputs \(\bar{y}_k^*\) and, consequently, the nominal state vectors \(x^*_k\) defined in (8). Define \(u_k = u^*_k + \Delta u_k\) and \(x_k = x^*_k + \Delta x_k\), for all \(k \in \mathcal{I}_t\). All \(\Delta u_k\) are
collected in $\Delta U_t = \{\Delta u_k | k \in I_t \}$. With a slight abuse of notations, we will write $U_t = U^*_t + \Delta U_t$, where $U^*_t$ collects all nominal control inputs. The GP-MPC (10) can then be reformulated equivalently by replacing $x_k$ with $x^*_k + \Delta x_k$, $u_k$ with $u^*_k + \Delta u_k$, and $U_t$ with $U^*_t + \Delta U_t$. This equivalent formulation can be approximated locally around the nominal values by replacing the GP model $G_f$ with its linGP $\hat{G}_f(z^*_t, u^*_t)$ at each $k$. Specifically, $\hat{y}_k$ and $\sigma^2_{y,k}$ are replaced respectively by $\hat{y}_k$ and $\hat{\sigma}^2_{y,k}$ of the linGP at nominal inputs $x^*_k$ and $u^*_k$. It follows from the derivation of the linGP, particularly Eqs. (6), that $\hat{y}_k$ is linear and $\hat{\sigma}^2_{y,k}$ is convex quadratic in $\Delta x_k$ and $\Delta u_k$: $\hat{y}_k = \hat{m}_k^T z^*_k u^*_k \xi_k$ and $\hat{\sigma}^2_{y,k} = \xi_k^T V_k z^*_k, u^*_k \xi_k$, where $\xi_k = [1, \Delta x^*_k, \Delta u^*_k]^T$. The vector $\hat{m}$ and the PSD matrix $\hat{V}$ are defined in (4). Let $\Delta \hat{y}_k = \hat{y}_k - y^*_k$. Then $\Delta x_k$ is derived from $\Delta \hat{y}_k$ and $\Delta u_k$ for $i < k$, see (8), as
\[
\Delta x^*_k = [\Delta \hat{y}_{k-1}, \Delta \hat{y}_{k-2}, \ldots, \Delta \hat{y}_{k-1}, \Delta u^*_k, \Delta u^*_{k-1}, \ldots, \Delta u^*_{k-i}, \Delta u^*_{k-i+1}, \ldots, \Delta u^*_{k-j}, \ldots, \Delta u^*_t].
\] As the linGP model, hence the approximate problem, is only accurate in a neighborhood of the nominal inputs $x^*_k$ and $u^*_k$, we introduce a trust region of size $\rho \geq 0$. Specifically, we impose the bounds $\|\Delta x_k\| \leq \rho$ and $\|\Delta u_k\| \leq \rho$, where $\|\cdot\|$ is an appropriate norm. Typically, box constraints, i.e., $\|\cdot\|_{\infty}$, are used. The trust region size $\rho$ is adapted by the sequential convex programming algorithm presented in Section IV-B.

Putting everything together, the complete local problem, which we will call linGP-MPC, is given in (12) below.

minimize $\Delta u_t, z_t, J(\hat{Y}_t, \hat{S}_{y,t}, U^*_t + \Delta U_t, Z_t)$
\[\text{subject to}
\]
\[
\hat{y}_k = \hat{m}_k^T z^*_k u^*_k \xi_k, \quad \hat{\sigma}^2_{y,k} = \xi_k^T V_k z^*_k, u^*_k \xi_k
\]
\[
\hat{y}_k = \hat{m}_k^T z^*_k u^*_k \xi_k, \quad \hat{\sigma}^2_{y,k} = \xi_k^T V_k z^*_k, u^*_k \xi_k
\]
\[
\|\Delta u_k\| \leq \rho, \quad \|\Delta \hat{y}_k\| \leq \rho
\]
\[
z_k \in Z, \quad u^*_k + \Delta u_k \in U
\]
\[
g_i(\hat{Y}_t, \hat{S}_{y,t}, U^*_t + \Delta U_t, Z_t) \leq 0, \quad i = 1, \ldots, n_{ineq}
\]
\[
h_i(\hat{Y}_t, \hat{S}_{y,t}, U^*_t + \Delta U_t, Z_t) = 0, \quad i = 1, \ldots, n_{eq}
\]
where $k \in I_t$, $\hat{Y}_t$ and $\hat{S}_{y,t}$ respectively collect $\hat{y}_k$ and $\hat{\sigma}^2_{y,k}$.

By Assumption 1, (12) is convex. By sequentially linearizing the GP with linGP, solving (12), and updating the nominal solution and trust region appropriately, the non-convex GP-MPC can be solved. The next subsection will present such a sequential convex programming (SCP) algorithm.

### B. Sequential convex programming for linGP-MPC

An issue that can arise in the SCP approach is that the local linGP-MPC (12) can be infeasible even though (10) is feasible. The nominal values may be infeasible for constraints (12f) and (12g) and, with $\rho$ small enough, (12e) is infeasible. This issue often occurs during the early SCP iterations. To address this issue, we will use exact penalty formulations of MPC. A penalized cost function is defined as $\phi(\hat{Y}_t, \hat{S}_{y,t}, U^*_t, Z_t) = J(\hat{Y}_t, \hat{S}_{y,t}, U^*_t, Z_t) + \lambda \sum_{i=1}^{n_{ineq}} \max(0, g_i(\hat{Y}_t, \hat{S}_{y,t}, U^*_t, Z_t)) + \sum_{i=1}^{n_{eq}} |h_i(\hat{Y}_t, \hat{S}_{y,t}, U^*_t, Z_t)|$ where $\lambda > 0$ is the penalty weight. If $\lambda$ is sufficiently large, the original GP-MPC can be solved by minimizing $\phi$ subject to the constraints (10b) to (10c). The exact penalty formulation of linGP-MPC is

\[
\text{minimize } \Delta u_t, z_t, \quad \phi(\hat{Y}_t, \hat{S}_{y,t}, U^*_t + \Delta U_t, Z_t)
\]
subject to constraints (12b) to (12e) in which $\phi$ is evaluated at the approximate linGP outputs.

We now present the linGP-SCP algorithm, inspired by the successive convexification algorithm in [9]. At iteration $j$, let $u^*_k(j)$ and $z^*_k(j)$ be the current solution, $\hat{y}^*_k(j)$ and $x^*_k(j)$ the corresponding outputs and states, $\rho(j)$ the current trust region size, and $\phi(j)$ the current exact penalized cost. We obtain the linGP model $\hat{G}_f(z^*_k, u^*_k(j))$ for each $k$. The convex problem (13) is then solved, resulting in the approximate solution $\hat{Y}_t, \hat{S}_{y,t}, U_t = U^*_t + \Delta U_t, Z_t$. By simulating $G_f$ with the inputs $U_t$, we obtain the exact output mean $\hat{y}_t$ and variance $\hat{S}_{y,t}$. To judge the algorithm progress, or the quality of the linGP approximation used, we compare the actual cost reduction $\delta(j) = \phi(j) - \phi(\hat{Y}_t, \hat{S}_{y,t}, U^*_t, Z_t)$ to the predicted cost reduction $\hat{\delta}(j) = \phi(j) - \phi(\hat{Y}_t, \hat{S}_{y,t}, U^*_t, Z_t)$. If $|\hat{\delta}(j)| < \epsilon$, for a predefined tolerance $\epsilon > 0$, the solution is considered converged and the algorithm is terminated. Otherwise, the ratio $r(j) = \hat{\delta}(j)/|\hat{\delta}(j)|$ is inspected. Given three predefined thresholds $0 < r_0 < r_1 < r_2 < 1$, there are four possibilities:

1. $r(j) < r_0$: the approximation is too inaccurate, hence the solution is rejected and the trust region size $\rho(j)$ is contracted by a predefined factor $\beta_{\text{fail}} < 1$;
2. $r_0 \leq r(j) < r_1$: the approximation is deemed inaccurate but acceptable, hence the solution is accepted but $\rho(j)$ is still contracted by $\beta_{\text{fail}}$;
3. $r_1 \leq r(j) < r_2$: the approximation is sufficiently accurate, hence the solution is accepted and $\rho(j)$ is retained;
4. $r(j) \geq r_2$: the approximation is deemed highly accurate or even conservative, hence the solution is accepted and $\rho(j)$ is enlarged by a predefined factor $\beta_{\text{success}} > 1$.

The iteration is repeated until convergence or until a maximum number of iterations $j_{\max} > 0$ is reached. The above linGP-SCP algorithm is detailed in Algorithm 1.

### C. Complexity of linGP-SCP algorithm

The linGP-SCP algorithm 1 consists of a main loop and the convex local linGP-MPC subproblem. Because a general MPC formulation is assumed, resulting in a general convex subproblem, it is not possible to determine the specific complexity of the subproblem. We instead briefly discuss the complexity of the algorithm with respect to the size of the GP (i.e., the size $N$ of its training data). For a more detailed discussion, readers are referred to the full paper [12].

An important feature of the linGP-SCP algorithm is that the subproblem’s complexity is independent of the size of the GP model. This is obvious from inspecting the linGP-MPC subproblem (12), where the parameters $\hat{m}$ and $\hat{V}$ do not depend on $N$. As a consequence, the inner iterations of the linGP-SCP algorithm, which solve the convex subproblem in each outer iteration of the main loop, do not depend on $N$.

The main loop involves the simulation and linearization of the GP model, which scale cubically with $N$. For a sufficiently large $N$, it is the largest factor determining the complexity and the solving time of the GP-MPC problem. Therefore, we
Algorithm 1 linGP-SCP Algorithm

Require: \(U(0), Z(0), t, \rho(0) > 0, \lambda > 0, 0 < r_0 < r_1 < r_2 < 1, \beta_{\text{fail}} < 1, \beta_{\text{succ}} > 1\), \(0 < r_0 < r_1 < r_2 < 1, \beta_{\text{fail}} < 1, \beta_{\text{succ}} > 1\), other two. Observe that linGP-SCP’s solving time was highly consistent with low variance and few small outliers.

1: Simulate \(\tilde{Y}_t\) with \(U(0), Z(0), t, \rho(0) > 0, \lambda > 0, 0 < r_0 < r_1 < r_2 < 1, \beta_{\text{fail}} < 1, \beta_{\text{succ}} > 1\), other two. Observe that linGP-SCP’s solving time was highly consistent with low variance and few small outliers.

2: \(\phi(0) \leftarrow \phi(\tilde{Y}_t, \tilde{S}_{y,t}^0, \tilde{U}_t, \tilde{Z}_t)\)

3: for \(j = 0, \ldots, j_{\text{max}} - 1\) do

4: Compute \(\tilde{m}_x(j, \tilde{u}_j)\) and \(\tilde{V}_k \in \mathbb{Z}_t\) by (4)

5: Solve problem (13) to get \(\tilde{Y}_t, \tilde{S}_{y,t}, \tilde{U}_t, \tilde{Z}_t\)

6: Simulate \(U_{t+1}\) with \(\tilde{U}_t, \tilde{S}_{y,t}, \tilde{U}_t, \tilde{Z}_t\)

7: \(\tilde{\delta}(j) \leftarrow \phi(S_{y,t}^{j+1}, t, \tilde{U}_t, \tilde{Z}_t)\)

8: \(\tilde{\delta}(j) \leftarrow \phi(S_{y,t}^{j+1}, t, \tilde{U}_t, \tilde{Z}_t)\)

9: if \(|\tilde{\delta}(j)| < \epsilon\) then stop and return \(U_t^*, Z_t^*(j)\)

10: if \(r(j) < r_0\) then

11: Keep current solution: \(U_t(j+1) \leftarrow U_t(j), Z_t(j+1) \leftarrow Z_t(j), \tilde{S}_{y,t}^{j+1} \leftarrow \tilde{S}_{y,t}^j, \tilde{U}_t \leftarrow \tilde{U}_t, \tilde{Z}_t \leftarrow \tilde{Z}_t\)

12: \(\rho_{\text{fail}}(j) \leftarrow \beta_{\text{fail}}\)

13: else

14: Accept solution: \(U_t(j+1) \leftarrow U_t(j), Z_t(j+1) \leftarrow \tilde{Z}_t, \tilde{S}_{y,t}^{j+1} \leftarrow \tilde{S}_{y,t}^j, \tilde{U}_t \leftarrow \tilde{U}_t, \tilde{Z}_t \leftarrow \tilde{Z}_t\)

15: if \(r(j) \leq r_1\) then \(\beta_{\text{succ}}(j) \leftarrow \beta_{\text{succ}}(j)\)

16: else if \(r(j) < r_0\) then \(\beta_{\text{fail}}(j) \leftarrow \beta_{\text{fail}}(j)\)

17: else \(\beta_{\text{fail}}(j) \leftarrow \beta_{\text{succ}}(j)\)

18: return \(U_t^{{j_{\text{max}}}}, Z_t^{{j_{\text{max}}}}\)

The GP-MPC problem is formulated as follows [5]

\[
\begin{align*}
\text{minimize} & \quad \sum_{k=t-H}^{t-1} (Q(y_k - r_t)^2 + R(u_k - u_{k-1})^2) \\
n & \quad 1 \leq u_k \leq 1, -0.5 \leq u_{k+1} - u_k \leq 0.5, \quad \forall k. 
\end{align*}
\]

subject to

\[
\begin{align*}
y_k = \mu f(y_{k-1}, u_k), & \quad \sigma_{g,k}^2 = \sigma_{f,k}^2(y_{k-1}, u_k) \quad (16a) \\
1 \leq u_k - 1, & \quad -0.5 \leq u_{k+1} - u_k \leq 0.5 \quad (16b) \\
-1 \leq u_k - 1, & \quad -0.5 \leq u_{k+1} - u_k \leq 0.5 \quad (16c) \\
& \quad y_k - u_k - 2 \sigma_{y,k} \leq 1.2 \quad (16d) \\
& \quad y_{k+1} - u_k - 2 \sigma_{y,k} \leq 1.2 \quad (16e) \\
& \quad \delta \leq y_{k+1} - u_k - 2 \sigma_{y,k} \leq 1.2 \quad (16f)
\end{align*}
\]

The constraints (16c) to (16e) are deterministic approximations of stochastic constraints that, with high probability, bound the process state. The MPC parameters are \(H = 12, Q = 10, R = 0.1, \delta = 0.075\). The closed-loop control system was simulated for 100 steps with the reference signal: \(r_k = -0.5\) for \(k \in [0, 50]\) and \(r_k = -0.2\) for \(k \in [51, 100]\). All simulations were run in Matlab R2017b on an iMac with Intel Core i7 4.2 GHz processor and 16 Gb RAM.

A. Solving the GP-MPC

To compare the performance of the linGP-SCP algorithm with state-of-the-art solvers, we solved the GP-MPC problem (16), for all 15 GP models, with different solvers:

- Ipopt: (16) was formulated in CasADi [13] and solved by Ipopt [14] with the optimized linear solver MA57. All Jacobians and Hessians were calculated automatically by CasADi in C code, well-known to be very fast.

- Knitro: similar to the previous, however the commercial nonlinear solver Knitro was used. Knitro is commonly regarded as one of the best nonlinear solvers.

- linGP-SCP: the linGP-SCP algorithm was prototyped purely in Matlab, where the subproblem was modeled by Yalmip [15] and solved by Gurobi.

The solvers were configured to similar accuracy so that their solutions were always within \(1 \times 10^{-6}\) from each other.

Remark 2: We also used the commercial solver fmincon of Matlab with its SQP and active-set methods. However, its performance was unpredictable and often worse than other solvers. Therefore we will not report its results here.

Remark 3: Unlike the highly optimized solvers Ipopt and Knitro, implemented in C/C++ and/or Fortran, the linGP-SCP prototype in Matlab is not an optimized implementation. In addition, the Yalmip toolbox is known to be slower than the C-based CasADi library. Therefore, there is significant room to improve the performance of the linGP-SCP implementation.

B. Simulation results and discussions

In all cases, the GP-MPC controller was able to track the given reference. As our main interest is in the solving performance, we omit the tracking control results and only report the performance results. Fig. 1 is a box plot of the solving time of each MPC step by Knitro and linGP-SCP, for all 15 GP models. We do not report the timing results of Ipopt in Fig. 1 because they were significantly worse than the other two. Observe that linGP-SCP’s solving time was highly consistent with low variance and few small outliers.
Most importantly, while the solving time of Knitro grew quickly with $N$, the solving time of linGP-SCP grew much slower. Fig. 2 shows the relative ratio of the median solving time of Ipopt and Knitro to that of linGP-SCP. It can be seen that Ipopt and Knitro became increasingly slower than linGP-SCP when $N$ increased. These results confirm the complexity analysis in Section IV-C that, compared to other methods, the complexity of linGP-SCP is less affected by the GP model size and therefore grows slowly with $N$.

In Table I, the mean $\mu_N$ and standard deviation $\sigma_N$ of the execution time in seconds per MPC step for several model sizes $N$ are listed. These numbers confirm the performance advantage of linGP-SCP. We can also observe that the solving time of the linGP-MPC subproblem (the “linGP-inner” row in the table) was constant regardless of the GP model size, as stated in the complexity analysis in Section IV-C.

### TABLE I

SOLVING TIME STATISTICS FOR SEVERAL GP MODEL SIZES.

|         | $\mu_{500}$ | $\sigma_{500}$ | $\mu_{900}$ | $\sigma_{900}$ | $\mu_{1500}$ | $\sigma_{1500}$ |
|---------|--------------|----------------|-------------|----------------|--------------|-----------------|
| linGP   | 0.096        | 0.0011         | 0.134       | 0.0013         | 0.209        | 0.0042          |
| linGP-inner | 0.024       | 0.0002         | 0.024       | 0.0002         | 0.024        | 0.0003          |
| Knitro  | 0.246        | 0.0081         | 0.896       | 0.0208         | 2.740        | 0.0390          |
| Ipopt   | 0.331        | 0.0385         | 1.29        | 0.1700         | 4.010        | 0.4680          |

VI. CONCLUSION

We developed a fast and scalable approach to solve a general class of GP-based MPC problems. The approach is based on the concept of linearized Gaussian Process (linGP), proposed in this paper, and Sequential Convex Programming. Our approach not only solves GP-MPC faster than other NLP methods but also is much less influenced by the GP training data size – a key factor affecting the computational complexity of GP-MPC. Therefore, our algorithm is more scalable and predictable. We are developing a more efficient implementation of the method, studying the convergence properties of the linGP-SCP algorithm, and investigating different applications of the linGP concept other than MPC.

REFERENCES

[1] J. M. Maciejowski, *Predictive control: with constraints*. Pearson, 2002.
[2] A. T. Schwarm and M. Nikolaou, “Chance-constrained model predictive control,” *AIChE Journal*, vol. 45, no. 8, pp. 1743–1752, Aug. 1999.
[3] D. Q. Mayne, “Model predictive control: Recent developments and future promise,” *Automatica*, vol. 50, no. 12, pp. 2967–2986, 2014.
[4] A. Jain, T. X. Nghiem, M. Morari, and R. Mangharam, “Learning and Control using Gaussian Processes: Towards bridging machine learning and controls for physical systems,” in *Proc. of the International Conference on Cyber-Physical Systems (ICCPS)*. ACM/IEEE, 2018.
[5] J. Kocijan, *Modelling and control of dynamic systems using Gaussian process models*. Springer, 2016.
[6] C. E. Rasmussen and C. K. Williams, *Gaussian processes for machine learning*. MIT press Cambridge, 2006, vol. 1.
[7] T. X. Nghiem and C. N. Jones, “Data-driven demand response modeling and control of buildings with gaussian processes,” in *Proceedings of American Control Conference (ACC)*, 2017.
[8] G. Cao, E. M.-K. Lai, and F. Alam, “Gaussian process model predictive control of unknown non-linear systems,” *IET Control Theory & Applications*, vol. 11, no. 5, pp. 703–713, Mar. 2017.
[9] Y. Mao, M. Szmuk, and B. Aşkınmeşe, “Successive convexification of non-convex optimal control problems and its convergence properties,” in *IEEE Conference on Decision and Control (CDC)*, 2016.
[10] M. Lawrynczuk, *Computationally Efficient Model Predictive Control Algorithms: A Neural Network Approach*, ser. Studies in Systems, Decision and Control. Springer International Publishing, 2014.
[11] E. Solak, R. Murray-smith, W. E. Leithead, D. J. Leith, and C. E. Rasmussen, “Derivative Observations in Gaussian Process Models of Dynamic Systems,” in *Advances in Neural Information Processing Systems 15*. MIT Press, 2003, pp. 1057–1064.
[12] T. X. Nghiem, “Linearized Gaussian Processes for Fast Data-driven Model Predictive Control,” *arXiv:1812.10579 [cs]*, Dec. 2018.
[13] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, “CasADi – A software framework for nonlinear optimization and optimal control,” *Mathematical Programming Computation*, 2018.
[14] A. Wächter and L. Biegler, “Ipopt-an interior point optimizer,” 2009.
[15] J. Löfberg, “Yalmip : A toolbox for modeling and optimization in *Mathematical Programming Computation*, 2018.