IEEE Robotics and Automation Letters (RAL) paper presented at the
2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

Gaussian Process Online Learning with a Sparse Data Stream

Jehyun Park and Jongeun Choi, *Member, IEEE*

Abstract— **Gaussian processes (GPs) have been exploited for various applications even including online learning. To learn time-varying hyperparameters from an information-limited sparse data stream, we consider the infinite-horizon Gaussian process (IHGP) with a low computational complexity for online learning. For example, the IHGP framework could provide efficient GP online learning with a data stream from mobile devices. However, we show that the originally proposed IHGP has difficulty in learning time-varying hyperparameters online from the sparse data stream. In this paper, we show how to extend the IHGP in order to learn time-varying hyperparameters using a sparse and non-stationary data stream. Firstly, we show that our solution approach offers the exact gradient as the solution of a Lyapunov equation. Next, we present that our approach with the exact gradient and a constraint for sparse sampling achieves better performance with a sparse data stream while still keeping the computational complexity low. Finally, to demonstrate the effectiveness, we present the benchmark comparison results. We also consider a slow vision system that needs to be combined with other fast sensory units for feedback control in the field of autonomous driving. In particular, we apply our approach to vehicle lateral position error estimation together with a deep learning model for autonomous driving using non-stationary lateral position error signals in a model-free and data-driven fashion.**

I. INTRODUCTION

GAUSSIAN processes (GPs, [1]) have been extensively exploited for various applications that require prediction and its predicted confidence region in a nonparametric and data-driven fashion [2]–[8]. GP regression has been applied to many practical applications including embedded systems such as mobile sensor networks and self-driving cars [2], [9]–[11]. Practical problems often require high quality online adaptive learning from a sparse data stream [12]. In the case of data sensor fusion for autonomous mobile vehicles and robots, the sampling rate of the global positioning system (GPS) is generally much slower than that of the inertial measurement unit (IMU). Hence, there are many techniques to solve such a multirate situation formulated as a multirate sensor fusion problem [13]. Recently, there has been much attention on high performing modern machine vision such as deep learning using vision data to obtain vehicle state information (e.g., lane tracking problems) [14]–[16]. Industrial researchers are highly motivated to adopt low-cost embedded systems with sparse data streams [17]. Machine vision (e.g., deep learning models with multiple layers) generally slows down the data stream, which makes it difficult for the GP regression to adaptively learn the time-varying hyperparameters from such an information-limited data stream.

This work was supported in part by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (2018R1A4A1025986) and in part by the Mid-career Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (NRF-2018R1A2B6008063).

Jehyun Park and Jongeun Choi are with the School of Mechanical Engineering, Yonsei University, 50 Yonsei Ro, Seodaemun Gu, Seoul 03722, South Korea jhyunpark@yonsei.ac.kr; jongeunchoi@yonsei.ac.kr. Jongeun Choi is the corresponding author jongeunchoi@yonsei.ac.kr.

In what follows, we introduce efforts to design efficient GP regression for online learning. The naive implementation of GPs requires a computational complexity of $\mathcal{O}(n_t^3)$, where n_t is the number of training samples. Since the growing complexity (as n_t increases) is an impediment to a practical application (e.g., online learning using the non-stationary data stream), many researchers adopt to use approximation methods for computationally efficient GP regression [9], [18], [19]. These approximation methods are largely classified into two groups, viz., sparse methods and local methods. Sparse methods [20] approximate the posterior using $n_r \ll n_t$ number of samples (i.e., inducing points) that well represent the data [21]. In this case, the computational complexity is reduced to $\mathcal{O}(n_r^2 n_t)$. These methods are appropriate for modeling smoothly-varying functions with high correlations. On the other hand, local methods [22], [23] are appropriate for modeling highly varying functions with low correlations only using local data. These methods divide the data into K sets of size n_t/n_r, then each local GP performs in each division. In this case, the computational complexity reduced to $\mathcal{O}(n_t^3/K^2)$ [21]. A promising local approximation method is Infinite-Horizon GP approximation (IHGP, [9]). Instead of construction of the kernel matrix, the IHGP leverages Markov structure of the process and represents the model as a linear Gaussian state space model to solve using Kalman filtering [24]. The Kalman filter requires a computational complexity of $\mathcal{O}(M^3N)$, where M is the dimension of the state space and N is the number of the data points. The IHGP further reduces the GPs' computation complexity to $\mathcal{O}(M^2N)$ by simplifying its filter recursion using a stationary Kalman filter gain with a steady state assumption at each iteration [9]. From this reduction in the computational complexity, the IHGP is then capable of online learning of time-varying hyperparameters [9]. The IHGP proposed in [9] shows an example of online adaptive IHGP running in real-time on an iPhone. The IHGP estimates hyperparameters adaptively by maximizing the marginal likelihood using its numerically found gradient. However, the originally proposed IHGP fails to yield accurate prediction and its confidence region when applied to a sparse online data stream, i.e., an information-limited situation as shown in Fig. 1. The prediction (in solid line) and its confidence region (in grey) of the adaptive IHGP largely differ from the base-

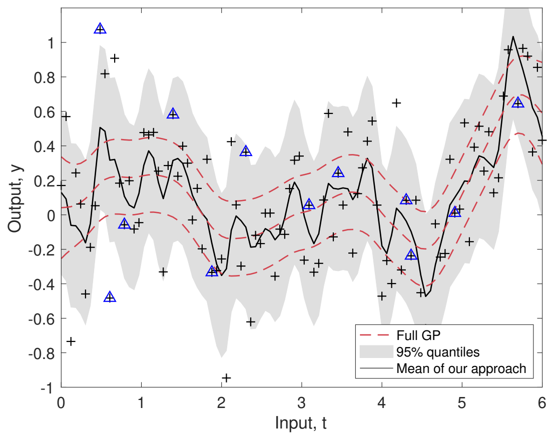


Fig. 1: A poor convergent behavior of online GP learning using a sparse data stream (only with 20% of the total data, marked with $\triangle$) by the originally proposed IHGP [9] as compared to that of the full GP using total data samples marked with $+$.

line predictive references (in dotted lines) produced by the standard GP regression denoted by Full GP (Fig. 1). In this paper, we address the aforementioned problem. Therefore, the objective of the paper is to develop a GP online algorithm for adaptively learning time-varying hyperparameters in order to compute posteriori predictive mean and prediction error variance functions for a given sparse data stream.

The contributions of our paper are as follows. Firstly, we consider a challenging problem of designing computationally efficient, adaptive, and online learning with sparse time-series data for GP regression. In particular, we consider a non-stationary process for the time-series signal with an application to vehicle lateral position error estimation for autonomous driving. Our solution approach builds on the IHGP [9] that has many advantages such as a low computational complexity and online adaptive learning of hyperparameters for non-stationary data streams. However, the IHGP shows poor convergent behaviors for sparse data streams (Fig. 1). Therefore, in this paper, we focus on the extension of the IHGP in order for time-varying hyperparameter learning to deal with sparse and non-stationary data streams. In particular, we provide the exact gradient to learn the time-varying hyperparameters to compute the gradient. Our approach requires the partial derivative of the solution of the discrete time algebraic Riccati equation (DARE), with respect to hyperparameters, which is shown to be the solution to a specific discrete Lyapunov equation (see Lemma 2.1). In addition, we set a lower bound on the length scale hyperparameter during the optimization process to overcome the sparse data sampling in time. Finally, we provide two examples to evaluate the efficacy of the proposed methodology. One example is a GP regression on simulated benchmark data taken from [9]. The other example is the online adaptive learning of hyperparameters for a non-stationary time series of vehicle error state estimates, which is generated by a deep learning model. Simulation results successfully demonstrate the effectiveness of our approach.

This paper is organized as follows. In Section II, we present our approach with technical details along with a background. Section III describes the comparative simulation study setups. Our successful simulation results are presented in Section IV. Finally, Section V provides a conclusion along with our future work.

II. Methods

A. Stochastic differential equation interpretation of a GP

Let $x(t)$ be a realization of a non-stationary GP on the time input t. A GP is completely defined by a mean function $\mu(t)$ and a covariance function $\mathcal{K}(t,t')$, i.e., $x(t) \sim \mathcal{GP}(\mu(t), \mathcal{K}(t,t'))$. GP regression is to infer the posterior distribution $p(x(t)|y_o) = \mathcal{N}(x(t)|\hat{\mu}(t|y_o), \hat{\mathcal{K}}(t,t'|y_o))$, where y_o is the measurement, for computing prediction and its predicted confidence region. In order to represent a GP in the form of stochastic differential equation (SDE), we use a half-integer Matérn covariance function that can be converted to an exact SDE representation [25], [26]. The Matérn covariance function is given as

$$\mathcal{K}_{Mat.}(t,t') = \sigma^2 \left(1 + \frac{\sqrt{3}|t-t'|}{\ell}\right) \exp\left(-\frac{\sqrt{3}|t-t'|}{\ell}\right),$$

where σ and ℓ are magnitude and length scale hyperparameters that control the correlation scale and variability of the process [26].

By using the interpretation of a GP as an SDE, we present the non-stationary GP in terms of the corresponding linear time-varying (LTV) SDE as follows [25]–[30].

$$\begin{aligned} \frac{dx(t)}{dt} &= F(t)x(t) + u(t) + B(t)\xi(t), \\ y_i &= C(t_i)x(t_i) + \epsilon_i, \end{aligned} \tag{1}$$

where $x(t) = (x_1(t), x_2(t), \cdots, x_M(t))^T \in \mathbb{R}^M$ is the state vector that consists of M stochastic processes, and $u(t)$ is a known control input. The process noise $\xi(t)$ is given by

$$\xi(t) \sim \mathcal{GP}(0, Q_c\delta(t-t')),$$

a zero-mean, multi-dimensional Gaussian process with the power-spectral density matrix $Q_c \in \mathbb{R}^{S\times S}$ and $\delta(\cdot)$ is the Dirac delta function. The measurement noise ϵ_i is given by

$$\epsilon_i \sim \mathcal{N}(0, \sigma_n^2 I),$$

and is realized by an independent and identically distributed (i.i.d.), zero-mean, multi-dimensional Gaussian distribution. $F(t) \in \mathbb{R}^{M\times M}$, $B(t) \in \mathbb{R}^{M\times S}$, and $C(t) \in \mathbb{R}^{1\times M}$ are the system, noise effect, and measurement matrices, respectively. In our case, state space model matrices are as follows [25].

$$F = \begin{bmatrix} 0 & 1 \\ -\frac{3}{\ell^2} & -\frac{2\sqrt{3}}{\ell} \end{bmatrix}, B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, C = \begin{bmatrix} 1 & 0 \end{bmatrix}, Q_c = 12\sqrt{3}\frac{\sigma^2}{\ell^3}.$$

The initial state is given by the prior with the initial state covariance matrix P_0, i.e., $x_0 \sim \mathcal{N}(0, P_0)$. The white Gaussian process (i.e., process noise) term $\xi(t)$ may be interpreted as a derivative of the Wiener process $dw(t)$, where $dw(t) = \xi(t)dt$ is an increment of the Wiener process $w(t)$ at time point t, i.e., $dw(t) := w(t + dt) - w(t)$. By multiplying the SDE (1) by dt, (1) can be more formally represented in terms of the increment of the Wiener process $dw(t)$ as

$$dx = (F(t)x(t) + u(t))dt + B(t)dw,\; y_i = C(t_i)x(t_i) + \epsilon_i.$$

The solution of the LTV-SDE (1) is

$$x(t) = \Phi(t, t_0)x(t_0) + \int_{t_0}^{t} \Phi(t, \tau)\{u(\tau) + B(\tau)w(\tau)\}d\tau,$$

where $\Phi(t, \tau)$ is the state transition matrix from τ to t [30]. Then the mean and covariance functions of the corresponding GP to the LTV-SDE (1) can be obtained by taking the first and second moment of $x(t)$ as follows.

$$\mu(t) = \mathbb{E}[x(t)] = \Phi(t, t_0)\mu_0 + \int_{t_0}^{t} \Phi(t, \tau)u(\tau)d\tau,$$

$$\begin{aligned}\mathcal{K}(t, t') =& \mathbb{E}\big[(x(t) - \mu(t))(x(t') - \mu(t'))^T\big]\\ =& \Phi(t, t_0)\mathcal{K}_0\Phi(t', t_0)^T\\ &+ \int_{t_0}^{\min(t,t')} \Phi(t, \tau)B(\tau)Q_cB(\tau)^T\Phi(t', \tau)^T d\tau,\end{aligned}$$

where μ_0 and $\mathcal{K}_0$ are the initial state mean and covariance matrix at t_0, respectively. For a practical application, a micro-controller needs to deal with discrete-time inputs and sampled measurements. For discrete state values $x_k = x(t_k)$, the state space model is now given by

$$\begin{aligned}x_k &\sim \mathcal{N}(x_k|A_{k-1}x_{k-1}, Q_{k-1}),\\ y_k &\sim \mathcal{N}(y_k|C_kx_k, \sigma_n^2 I),\end{aligned} \tag{2}$$

where A_k is the discrete-time state transition matrix between time-instance t_k and t_{k+1}. Q_k is the covariance matrix of the process noise. The corresponding discrete-time system matrices A_k and Q_k are then obtained by the SDE system parameters from (1)

$$\begin{aligned}A_k &= \Phi(t_{k+1}, t_k),\\ Q_k &= \int_{t_k}^{t_{k+1}} \Phi(t_{k+1}, \tau)B(\tau)Q_cB(\tau)^T\Phi(t_{k+1}, \tau)^T d\tau,\end{aligned} \tag{3}$$

where $\Phi(t_{k+1}, t_k)$ is a transition matrix.

B. Steady-state Kalman filter

We can explicitly represent the Bayesian filtering using the Kalman filter, which is the recursive solution to a general linear Gaussian model in (2). The Kalman filter equations can be represented as prediction and update steps as follows [24].

Prediction step:

$$\begin{aligned}m_k^p &= A_{k-1}m_{k-1}^f,\\ P_k^p &= A_{k-1}P_{k-1}^f A_{k-1}^T + Q_{k-1},\end{aligned}$$

Update step:

$$\begin{aligned}S_k &= C_kP_k^pC_k^T + R,\\ K_k &= P_k^pC_k^TS_k^{-1},\\ m_k^f &= m_k^p + K_k[y_k - C_km_k^p],\\ P_k^f &= P_k^p - K_kS_kK_k^T,\end{aligned}$$

where m_k^p and P_k^p are predictive state mean and covariance, respectively, and m_k^f and P_k^f are filter state mean and covariance, respectively. S_k and K_k are temporary variables to calculate the mean and covariance, and R is equal to the measurement noise power $\sigma_n^2 I$. If the measurements are given by a scalar, then S_k is a scalar, and therefore matrix inversion is not necessary.

Now, let us assume a case of linear time-invariant (LTI) SDE without control inputs, such that $F(t) = F, B(t) = B, C(t) = C$, and $u(t) = 0$. Since the model is stationary, the stationary state is distributed by $x_\infty \sim \mathcal{N}(0, P_\infty)$ and the stationary covariance of $x(t)$ is the solution of the Lyapunov equation, $\dot{P}_\infty = FP_\infty + P_\infty F^T + BQ_cB^T = 0$.

In this case, the system matrices in (3) are then shown as

$$\begin{aligned}A_k &= \exp(F\triangle t_k),\\ Q_k &= \int_0^{\triangle t_k} \exp(F(\triangle t_k - \tau))BQ_cB^T\exp(F(\triangle t_k - \tau))^T d\tau,\end{aligned} \tag{4}$$

where $\triangle t_k := t_{k+1} - t_k$.

Given that P_∞ exists and is known, the process noise covariance in (4) can be computed efficiently as follows [31].

$$Q_k = P_\infty - A_kP_\infty A_k^T. \tag{5}$$

In steady-state Kalman filtering, we assume equidistant observations in time ($A_k := A$ and $Q_k := Q$) as $t \to \infty$ [9], [32]. Then the filter gain converges to the stationary Kalman filter gain $K = PC^T(CPC^T + R)^{-1}$, where P is the unique solution of the following DARE [32].

$$P = A^TPA - (A^TPC^T)(CPC^T + R)^{-1}CPA + Q. \tag{6}$$

The filter conditional mean (or GP prediction) and the filter state covariance of the Kalman filter recursion with the stationary Kalman filter gain are given as follows.

$$\begin{aligned}m_k^f &= (A - KCA)m_{k-1}^f + Ky_k,\\ P^f &= P - KCP,\end{aligned} \tag{7}$$

for all $k = 1, 2, \cdots, N$. The recursion in (7) has a computational complexity with one $M \times M$ matrix-vector multiplication that results in $\mathcal{O}(M^2N)$ while the Kalman filter has a computational complexity of $\mathcal{O}(M^3N)$, where M is the dimension of the state space and N is the number of the data points.

C. Gradient descent algorithm for online learning

Our approach adaptively optimizes the hyperparameters θ using the following gradient descent algorithm with the exact gradient.

$$\theta_j = \theta_{j-1} + \eta\nabla\log p(y_j|\theta_{j-1}), \tag{8}$$

where η is a learning-rate. The optimization process for our benchmark results uses a nonlinear programming solver with gradient information (fmincon) in the MATLAB optimization toolbox. In our case, θ is consist of σ_n, σ and ℓ, which represent the measurement noise variance, signal variance, and length scale, respectively. In order to calculate the exact gradient of the log marginal likelihood function, we first define the log marginal likelihood in terms of $v_k := y_k - CAm_k^f$ and $s_k := CPC^T + \sigma_n^2$ as follows [9].

$$\begin{aligned}\log p(y) =& \sum_{k=1}^{N} \frac{1}{2}(\log 2\pi s_k + v_k^2/s_k) \\ =& \frac{N}{2}\log 2\pi + \sum_{k=1}^{N}\frac{1}{2}\log s_k + \sum_{k=1}^{N}\frac{v_k^2}{2s_k} \\ =& \frac{N}{2}\log 2\pi + \sum_{k=1}^{N}\frac{1}{2}\log(CPC^T + \sigma_n^2) \\ &+ \sum_{k=1}^{N}\frac{(y_k - CAm_k^f)^2}{2(CPC^T + \sigma_n^2)}.\end{aligned}$$

Using this log marginal likelihood, we show how to calculate the exact gradient of the likelihood for online learning in (8). In order to explain our method, we first introduce the following lemma.

Lemma 2.1: Let us define the notation $(\cdot)' := \frac{\partial \cdot}{\partial \theta}$. Given the discrete algebraic Riccati equation (6), the partial derivative of the solution of the DARE with respect to θ (i.e., $P' := \frac{\partial P}{\partial \theta}$) can be exactly computed by solving the discrete Lyapunov equation

$$\bar{A}^T P' \bar{A} - P' + \bar{Q} = 0, \tag{9}$$

where $\bar{A} := A - C^T S^{-1} CPA$, and $\bar{Q} := A'^T PA + A^T PA'^T - A'^T PC^T S^{-1} CPA - A^T PC^T S^{-1} CPA' + A^T PC^T S^{-1} R' S^{-1} CPA + Q'$.

Proof: $P' = \frac{\partial P}{\partial \theta}$ is obtained by differentiating the DARE (6) with respect to θ as follows.

$$\begin{aligned}&\left[A^T PA - P - (A^T PC^T)S^{-1}CPA + Q\right]' \\ =&A'^T PA + A^T P' A + A^T PA'^T - P' - A'^T PC^T S^{-1} CPA \\ &- A^T P' C^T S^{-1} CPA + A^T PC^T S^{-1} S' S^{-1} CPA \\ &- A^T PC^T S^{-1} CP' A - A^T PC^T S^{-1} CPA' + Q' = 0,\end{aligned} \tag{10}$$

where S is $CPC^T + R$.

If we substitute $\bar{A}$ and $\bar{Q}$ with $A - C^T S^{-1} CPA$, and $A'^T PA + A^T PA'^T - A'^T PC^T S^{-1} CPA - A^T PC^T S^{-1} CPA' + A^T PC^T S^{-1} R' S^{-1} CPA + Q'$, respectively, (10) becomes the discrete Lyapunov equation (9) that can be solved for P'. Therefore, P' can be exactly computed by the solution of the discrete Lyapunov equation (9) in Lemma 2.1. ■

Now, we show how to obtain the exact gradient of the log marginal likelihood.

Proposition 2.2: Given a log marginal likelihood as $\log p(y) = \sum_{k=1}^{N} \frac{1}{2}(\log 2\pi s_k + v_k^2/s_k)$, where $v_k = y_k - CAm_k^f$ and $s_k = CPC^T + \sigma_n^2$, the exact gradient of the log marginal likelihood is calculated as in lines from 13 to 20 of Algorithm 2.

Proof: The gradient of the log marginal likelihood is given by

$$\begin{aligned}\nabla \log p(y) =& \sum_{k=1}^{N} \frac{1}{2(CPC^T + \sigma_n^2)} \frac{\partial(CPC^T + \sigma_n^2)}{\partial \theta} \\ &- \sum_{k=1}^{N} \frac{(y_k - CAm_k^f)}{(CPC^T + \sigma_n^2)} \frac{\partial(CAm_k^f)}{\partial \theta} \\ &- \sum_{k=1}^{N} \frac{(y_k - CAm_k^f)^2}{2(CPC^T + \sigma_n^2)^2} \frac{\partial(CPC^T + \sigma_n^2)}{\partial \theta}.\end{aligned} \tag{11}$$

From (11), it is straightforward to see how P' that is obtained by solving the Lyapunov equation (9), is used to compute the exact gradient as in Algorithm 2. ■

Note that we set a lower bound on the length scale hyperparameter ℓ during the optimization process (8) to overcome the sparse data sampling.

Remark 2.3: Our method to compute the exact gradient differs from the previous work based on the numerically found gradient using the DARE solver [9]. Our exact solution also saves additional computational times. The computational times of our approach and the previous IHGP (measured using Matlab programming) are 0.0031s and 0.0042s, respectively.

Algorithm 1 Online time-varying hyperparameter learning using sparse data.

```
Input: $\theta = (\sigma_n^2, \sigma^2, \ell)_{k-1}, y_k$
 1: $w$ = window size
 2: $Y_k = \{y_k\} \cup Y_{k-1} \setminus \{y_{k-w}\}$
 3: model := Matern covariance functions to state space
 4: while Until $\theta$ converges do
 5:     $(F, Q_c, C, P_\infty) = \text{model}(\theta)$
 6:     $A = \exp(Fdt)$,
 7:     $Q = P_\infty - AP_\infty A^T$,
 8:     $R = \theta(1)$,
 9:     $P = \text{dare}(A, C^T, Q, R)$
10:     Obtain $m$ by applying Kalman smoother
11:     [$L$, grad]=Algorithm 2 $(A, C, P, Y_k, \theta, m)$
12:     $\theta = \theta + \eta\text{grad}$ (8)
13: end while
```

Algorithm 1 presents the IHGP algorithm [9] performed on sparsely windowed observations Y_k, assuming the locally steady-state at time t_k for online time-varying hyperparameter learning. The reason for using the windowed observations Y_k is to consider the situation of estimating the lateral position error while the vehicle is driving (Section III-B). To minimize the the memory size of the electronic control unit (ECU) of the vehicle, we maintain the size of the window by continuously adding the latest observation and removing the oldest observation in a sparse sampling rate. Note that to reconstruct the posteriori predictive mean function and its prediction error variance function, i.e., interpolation using sparse data streams, we applied Kalman smoothing [24] in Algorithm 1.

Our modified algorithm to compute the exact gradient is given by Algorithm 2, which is nested in Algorithm 1 in order to estimate the time-varying hyperparameters at time t_k by maximizing log likelihood function using (8) in Algorithm 1.

Algorithm 2 Evaluating the log marginal likelihood and its gradient.

```
    Input: $A, C, P, Y, \theta, m$
 1: $\hat{s} = CPC^T + R$
 2: $K = PC^T / \hat{s}$
 3: $dR = [1 \quad 0 \quad 0]^T$
 4: j = the number of hyperparameters
 5: for $i = 1 \to j$ do
 6:     obtain $dA, dQ$ (see Appendix in Section VI)
 7:     $\bar{A} = A - C^T \hat{s}^{-1} CPA$
 8:     $\bar{Q}_i = dA_i^T PA + A^T P dA_i^T - dA_i^T PC^T \hat{s}^{-1} CPA - A^T PC^T \hat{s}^{-1} CP dA_i + A^T PC^T \hat{s}^{-1} dR_i \hat{s}^{-1} CPA + dQ_i$
 9:     $P_i' = \text{dlyap}(\bar{A}^T, \bar{Q}_i)$
10:     $dS_i = CP_i' C^T + dR_i$
11:     $dK_i = P_i' C^T / \hat{s} - PC^T dS_i / \hat{s}^2$
12: end for
13: $L = \frac{N}{2} \log(2\pi) + \frac{N}{2} \log(\hat{s})$
14: grad $= \frac{N}{2} dS / \hat{s}$
15: for all $y_i \in Y$ do
16:     $v = y_i - CAm$
17:     $L = L + 0.5 v^2 / \hat{s}$
18:     $dv = -m^T C dA - CAdm$
19:     grad = grad $+ v dv / \hat{s} - 0.5 v^2 dS / \hat{s}^2$
20:     $dm = AKCAdm + dK y_i$
21: end for
22: return $L$, grad
```

Algorithm 2 summarizes the optimization steps by our approach over a sparse data stream. First, by using the state space model, solve the DARE for the state covariance, and compute the stationary gain. We then calculate the exact gradient by solving the discrete Lyapunov equation (9). Previously, the IHGP method [9] tries to reformulate (10) as another DARE for solving P' using a MATLAB function (e.g., dare in MATLAB Control System Toolbox, from MATLAB, MathWorks, Natick, MA, USA) in $\mathcal{O}(M^3)$. Since our approach replaces this part with another MATLAB function (e.g., dlyap in MATLAB Control System Toolbox) that solves discrete Lyapunov equation in $\mathcal{O}(M^3)$ for the exact P', we maintain the same computational complexity as the IHGP approach.

Note that similar approaches to obtain P' exactly using a Lyapunov equation are reported for the optimization of the sampling points to perform robotic environmental monitoring [33], and for the minimization of the gap between two controllers to solve an inverse optimal control problem [34].

III. Comparison Studies

We compare the results of our approach against those of the IHGP [9] over sparse online data streams. All experiments run in Mathworks MATLAB (R2019a) on a workstation equipped with 3.2GHz Intel Core i7 and 16Gb RAM.

A. Simulated benchmark data

We provide a GP regression example with $N = 100$ observations simulated from $y_k = \text{sinc}(6 - x_k) + \epsilon_k$, where $\epsilon_k \sim \mathcal{N}(0, 0.1)$. This example is taken from [9]. In order to make a sparse data stream, we randomly select 20 out of 100 samples (i.e., 20%) to obtain the gradient of the marginal likelihood while optimizing the hyperparameters.

B. Lateral position error estimation

Now we consider a practical problem for another comparison study. We perform online GP learning and prediction on a non-stationary vehicle state variable. In particular, we consider the lateral position error of the vehicle, the variable of which can be used for lane keeping feedback control. We define the lateral position error as the distance from the road center to the vehicle perpendicularly. We collected the data by using AirSim car simulator (Microsoft Corporation, Redmond, WA USA) based on Unreal Engine 4 (Epic Games, Inc. Cary, NC, USA) as shown in Fig. 3. We depict the trajectory of the vehicle in Fig. 7. The red line is the road center and the blue boxes are the collection of the poses of the vehicle over time. To be more practical, we simulate a situation where deep learning-based machine vision produces the non-stationary vehicle state variable at each sampling time. Our online GP learning is then applied to produce the smooth prediction and its predicted confidence region based on the sparse output sequences from machine vision in a data-driven way, i.e., without any vehicle model such as kinematics or dynamics. In this example, similar to the previous example, we randomly select 19 out of 50 samples (i.e., 38%) to make a sparse data stream to obtain the gradient of the marginal likelihood while optimizing the hyperparameters. The deep neural network takes a series of images with the resolution of 400×300 pixels (Fig. 2) as the input and produces one estimated vehicle state variable, i.e., the lateral position error with respect to the road center. The deep neural network consists of a convolutional neural network (CNN) and a long short-term memory (LSTM) network. In what follows we explain both networks in details.

1) CNN: We use the pre-trained convolutional layers of Google's InceptionV3 model to extract features from the vision data [35]. At every time step, the CNN takes an image and generates a feature vector that contains information of the image. Then we stack the sequential feature vectors of the last five time steps and transfer them to the LSTM. The reason for using a sequence of images as the input is to filter the output to improve estimation performance via noise reduction [36].

2) LSTM network: In our deep neural network, two LSTM layers follow the convolutional layers. The LSTM network is well suited for the prediction from sequential data. An LSTM layer has a memory cell, an input gate, an output gate and a forget gate. These gates regulate whether the information of the input needs to be remembered or not. The conceptual

Fig. 2: An example of a sequence of images that the deep neural network takes as the input to produce the non-stationary vehicle state variable.

Fig. 3: The simulation with a realistic visual environment for collecting vehicle state variable (lateral position error with respect to the road center).

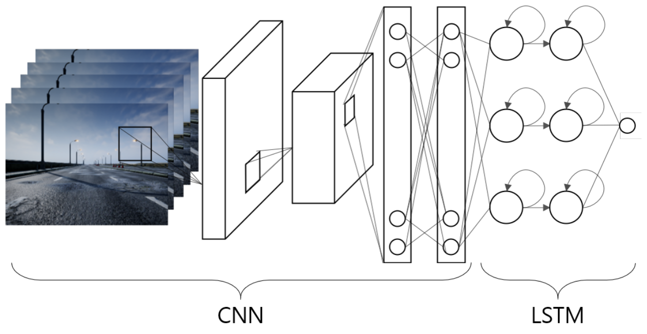


Fig. 4: The structure of the CNN-LSTM network.

model structure of the deep neural network is illustrated in Fig. 4.

IV. Comparison study results

A. Simulated benchmark data

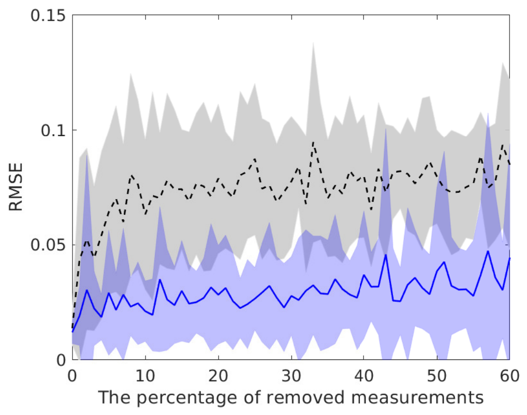
Figures 6 and 1 show the simulation results of our approach and the original IHGP [9], respectively. We use the full GP implementation with the total data points (marked with $+$ in Fig. 6) as a reference (Full GP) to compare the performance of our approach and the original IHGP. For further performance analysis, we first define the Root Mean Square Error (RMSE) with respect to the results of Full GP regression, where the error is the difference between the prediction with a sparse data stream and Full GP prediction. Our approach (Fig. 6) uses the solution of the discrete Lyapunov equation while the original IHGP (Fig. 1) uses the forward recursion to exactly compute the gradient of the log likelihood. Both methods perform well with a sufficient number of data samples. However, our approach (Fig. 6) outperforms the original IHGP (Fig. 1) over a sparse data stream (marked with $\triangle$ in Fig. 6). In particular, our approach successfully produces prediction (in solid line) and its predicted confidence region (in grey) that match well with those of Full GP regression (in dotted lines) (Fig. 6) even with a sparse data stream. On the other hand, the prediction (in solid line) and its confidence region (in grey) by the original IHGP are based on wrongfully estimated hyperparameters online (i.e., too small bandwidths over time) and so they significantly differ from those of Full GP regression (in dotted lines) as shown in Fig. 1. In this particular realization, the RMSE values of our approach (Fig. 6) and IHGP (Fig. 1) are 0.0009 and 0.0172, respectively.

B. Monte Carlo simulations

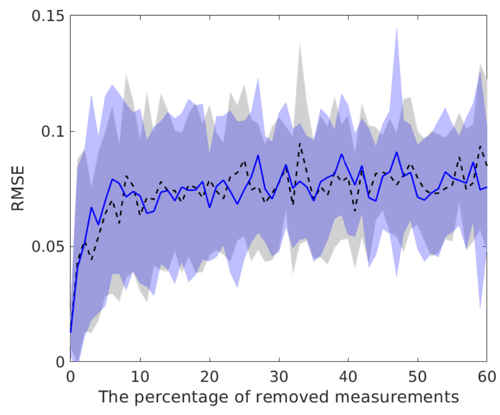
In order to evaluate the averaged performance of our approach against IHGP, we repeatedly realized the data streams with random measurement noises 30 times under different sparse rates. In particular, for each simulated data stream, we vary the percentage of removed measurements from 0% to 60% to compare the averaged RMSE values of the two approaches (over random realizations). If we remove 0% of the data, the RMSE values of both approaches converge to 0. However, as the amount of sparsity increases (i.e., the percentage of removed measurements increases), the RMSE value increases. In that case, our approach shows generally much lower averaged RMSE values than the original IHGP for overall ranges of sparsity (Fig. 5a). On average over sparsity, the RMS reduction of our achieved averaged RMSE values as compared to that of IHGP is 43%. Our approach performs better due to the lower bound on the length scale hyperparameter ℓ during the optimization process (Fig. 5a) as compared to the unconstrained optimization (Fig. 5b).

C. Lateral position error estimation

The trajectory of the collected (ground truth) vehicle poses is shown in blue rectangles during the test as shown in Fig. 7. For each time step, the optimization process is performed for a window of size 50 and it takes 0.026 seconds, which allows the practical implementation of our approach. The results of prediction and its predicted confidence region on the lateral position variables by our approach and by the original IHGP are shown in Fig. 8a and Fig. 8b, respectively. The prediction (in solid line) and its confidence region (in grey) by the original IHGP do not track the references (dotted lines) of Full GP regression well while our approach's prediction



(a) RMSE values with a constrained optimization.



(b) RMSE values with an unconstrained optimization.

Fig. 5: RMSE values of our approach and IHGP from Monte Carlo benchmark simulations by varying the sparsity from the removal percentage of 0% to 60%. Black dotted lines are predictions using the original IHGP, and blue solid lines are predictions (a) with and (b) without a lower bound on the length scale hyperparameter during the optimization process. Blue and grey regions are confidence regions of each prediction.

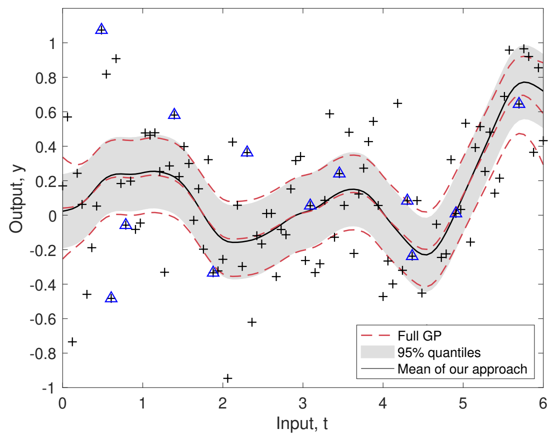


Fig. 6: Simulation result of a GP regression on sparse data using the modified IHGP.

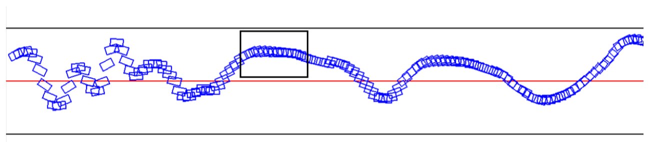
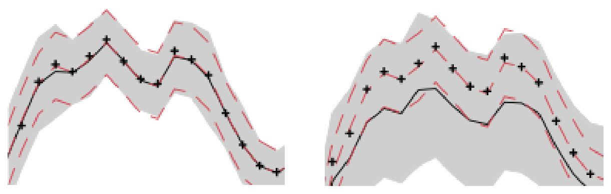


Fig. 7: The trajectory of the collected (ground truth) vehicle poses in blue rectangles during the test. The road center is shown as a red solid line.


(a) The prediction results by our approach.

(b) The prediction result by the IHGP.

Fig. 8: The prediction (in solid line) and its confidence region (in grey) of the box in Fig. 7 by our approach and the IHGP.

tracks the reference prediction well as shown in Fig. 8a. The RMSE values of our approach and IHGP during the entire trajectory given in Fig. 7 are 0.0066, and 0.0436, respectively. In addition, we collected data 10 times using AirSim car simulator. For 10 simulated data streams, the averaged RMSE values of our approach and IHGP are 0.008 and 0.0182, respectively. The performance gain by using our approach is not significant in an RMSE sense due to a small level of the randomness in our deep neural network output. However, from the results of Section IV-B, we expect that our approach is effective when it is applied to cheap sensor units with high measurement noise levels.

V. Conclusion

In this paper, we successfully extend the IHGP in order to learn the time-varying hyperparameters using sparse and non-stationary data streams. Our approach calculates the exact gradient of the marginal likelihood for GP online learning using the solution of the discrete Lyapunov equation. Our approach's maximization with the exact gradient and a lower bound improves the performance against the previous IHGP with sparse data streams. Our approach still maintains the overall computational complexity with $\mathcal{O}(m^2 n)$ for GP online learning for possible use in embedded systems. We have successfully demonstrated the usefulness of our approach under sparse non-stationary data streams 1) in a simulated benchmark problem and 2) in another practical problem of estimating vehicle lateral position error using the outputs of a CNN-LSTM network in a non-parametric fashion (without any vehicle models). Future work is to apply our approach to build algorithms to compute prediction and its confidence region using different machine vision applications in embedded systems for robotics and self-driving cars.

VI. Appendix

In this section, we show how to compute dA and dQ in Algorithm 2. Given $F = \begin{bmatrix} 0 & 1 \\ -\frac{3}{\ell^2} & -\frac{2\sqrt{3}}{\ell} \end{bmatrix} \in \mathbb{R}^{2\times 2}$, Two components of $dF \in \mathbb{R}^{2\times 2\times 2}$, viz., $dF(:,:,1) = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$, $dF(:,:,2) = \begin{bmatrix} 0 & 0 \\ \frac{6}{\ell^3} & \frac{2\sqrt{3}}{\ell^2} \end{bmatrix}$ are obtained by differentiating F with respect to σ and ℓ, respectively. $dA \in \mathbb{R}^{2\times 2\times 2}$ is then calculated as follows.

$dA(:,:,i) = \exp(dF(:,:,i)\cdot dt),\ i \in \{1,2\}$.

Given $P_\infty = \begin{bmatrix} \sigma & 0 \\ 0 & \frac{3\sigma}{\ell^2} \end{bmatrix} \in \mathbb{R}^{2\times 2}$, $dP_\infty \in \mathbb{R}^{2\times 2\times 2}$ is obtained by differentiating P_∞ with respect to σ and ℓ as follows. $dP_\infty(:,:,1) = \begin{bmatrix} 1 & 0 \\ 0 & \frac{3}{\ell^2} \end{bmatrix}$, $dP_\infty(:,:,2) = \begin{bmatrix} 0 & 0 \\ 0 & -\frac{6\sigma}{\ell^3} \end{bmatrix}$.

$dQ \in \mathbb{R}^{2\times 2\times 2}$ is then calculated as follows.

$dQ(:,:,i) = dP_\infty(:,:,i) - dA(:,:,i)P_\infty A$
$- A^T dP_\infty(:,:,i)A - A^T P_\infty dA(:,:,i),\ i \in \{1,2\}$.

References

[1] C. Rasmussen and C. Williams, "Gaussian processes for machine learning the MIT press," 2006.

[2] Y. Xu, J. Choi, S. Dass, and T. Maiti, *Bayesian prediction and adaptive sampling algorithms for mobile sensor networks: Online environmental field reconstruction in space and time*. Springer, 2015.

[3] M. Jadaliha, Y. Xu, J. Choi, N. S. Johnson, and W. Li, "Gaussian process regression for sensor networks under localization uncertainty," *IEEE Transactions on Signal Processing*, vol. 61, no. 2, pp. 223–237, 2012.

[4] G. Camps-Valls, J. Verrelst, J. Munoz-Mari, V. Laparra, F. Mateo-Jimenez, and J. Gomez-Dans, "A survey on Gaussian processes for earth-observation data analysis: A comprehensive investigation," *IEEE Geoscience and Remote Sensing Magazine*, vol. 4, no. 2, pp. 58–78, 2016.

[5] Y. Xu and J. Choi, "Adaptive sampling for learning Gaussian processes using mobile sensor networks," *Sensors*, vol. 11, no. 3, pp. 3051–3066, 2011.

[6] R. C. Grande, G. Chowdhary, and J. P. How, "Nonparametric adaptive control using Gaussian processes with online hyperparameter estimation," in *52nd IEEE Conference on Decision and Control*. IEEE, 2013, pp. 861–867.

[7] M. Blum and M. A. Riedmiller, "Optimization of Gaussian process hyperparameters using Rprop." in *ESANN*. Citeseer, 2013, pp. 339–344.

[8] K. Liu, M. L. Yi Li, Xiaosong Hu, and D. Widanlage, "Gaussian process regression with automatic relevance determination kernel for calendar aging prediction of lithium-ion batteries," *IEEE Transactions on industrial informatics*, 2019.

[9] A. Solin, J. Hensman, and R. E. Turner, "Infinite-horizon Gaussian processes," in *Advances in Neural Information Processing Systems*, 2018, pp. 3486–3495.

[10] S. A. Goli, B. H. Far, and A. O. Fapojuwo, "Vehicle trajectory prediction with Gaussian process regression in connected vehicle environment ⋆," in *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2018, pp. 550–555.

[11] J. Huang, Y. Cao, C. Xiong, and H.-T. Zhang, "An echo state Gaussian process-based nonlinear model predictive control for pneumatic muscle actuators," *IEEE Transactions on Automation Science and Engineering*, 2018.

[12] J. C. Shiyi Yang, Soo Jeon, "Multi-fidelity sampling for fast Bayesian shape estimation with tactile exploration," *IEEE Transactions on industrial informatics*, 2019.

[13] Y. Cong, L. Zhou, Z. Song, and Z. Ge, "Multirate dynamic process monitoring based on multirate linear Gaussian state-space model," *IEEE Transactions on Automation Science and Engineering*, 2019.

[14] N. Gupta, S. Djeziri, and E. R. Corral-Soto, "Vehicle vision system with adaptive lane marker detection," Apr. 17 2018, uS Patent 9,946,940.

[15] Y. Wang, Y. Liu, H. Fujimoto, and Y. Hori, "Vision-based lateral state estimation for integrated control of automated vehicles considering multirate and unevenly delayed measurements," *IEEE/ASME Transactions on Mechatronics*, vol. 23, no. 6, pp. 2619–2627, 2018.

[16] Z. Chen and X. Huang, "End-to-end learning for lane keeping of self-driving cars," in *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2017, pp. 1856–1860.

[17] S. Lucia, D. Navarro, Ó. Lucía, P. Zometa, and R. Findeisen, "Optimized FPGA implementation of model predictive control for embedded systems using high-level synthesis tool," *IEEE transactions on industrial informatics*, vol. 14, no. 1, pp. 137–145, 2017.

[18] Y. Xu, J. Choi, and S. Oh, "Mobile sensor network navigation using Gaussian processes with truncated observations," *IEEE Transactions on Robotics*, vol. 27, no. 6, pp. 1118–1131, 2011.

[19] Y. Xu, J. Choi, S. Dass, and T. Maiti, "Sequential Bayesian prediction and adaptive sampling algorithms for mobile sensor networks," *IEEE Transactions on Automatic Control*, vol. 57, no. 8, pp. 2078–2084, 2011.

[20] E. Snelson and Z. Ghahramani, "Sparse Gaussian processes using pseudo-inputs," in *Advances in neural information processing systems*, 2006, pp. 1257–1264.

[21] M. M. Zhang, B. Dumitrascu, S. A. Williamson, and B. E. Engelhardt, "Sequential Gaussian processes for online learning of nonstationary functions," *arXiv preprint arXiv:1905.10003*, 2019.

[22] M. P. Deisenroth and J. W. Ng, "Distributed Gaussian processes," *arXiv preprint arXiv:1502.02843*, 2015.

[23] J. W. Ng and M. P. Deisenroth, "Hierarchical mixture-of-experts model for large-scale Gaussian process regression," *arXiv preprint arXiv:1412.3078*, 2014.

[24] S. Särkkä, *Bayesian filtering and smoothing*. Cambridge University Press, 2013, vol. 3.

[25] A. Solin *et al.*, "Stochastic differential equation methods for spatio-temporal Gaussian process regression," 2016.

[26] J. Hartikainen and S. Särkkä, "Kalman filtering and smoothing solutions to temporal gaussian process regression models," in *2010 IEEE international workshop on machine learning for signal processing*. IEEE, 2010, pp. 379–384.

[27] S. Särkkä and A. Solin, *Applied stochastic differential equations*. Cambridge University Press, 2019, vol. 10.

[28] J. Choi and D. Milutinović, "Tips on stochastic optimal feedback control and Bayesian spatiotemporal models: Applications to robotics," *Journal of Dynamic Systems, Measurement, and Control*, vol. 137, no. 3, p. 030801, 2015.

[29] L. Petrović, I. Marković, and M. Seder, "Multi-agent Gaussian process motion planning via probabilistic inference," *IFAC-PapersOnLine*, vol. 51, no. 22, pp. 160–165, 2018.

[30] T. D. Barfoot, C. H. Tong, and S. Särkkä, "Batch continuous-time trajectory estimation as exactly sparse Gaussian process regression." in *Robotics: Science and Systems*. Citeseer, 2014.

[31] E. Davison and F. Man, "The numerical solution of A'Q+ QA=-C," *IEEE Transactions on Automatic Control*, vol. 13, no. 4, pp. 448–449, 1968.

[32] F. Gustafsson, *Adaptive filtering and change detection*. Citeseer, 2000, vol. 1.

[33] M. Jadaliha and J. Choi, "Environmental monitoring using autonomous aquatic robots: Sampling algorithms and experiments," *IEEE Transactions on Control Systems Technology*, vol. 21, no. 3, pp. 899–905, 2012.

[34] M. C. Priess, R. Conway, J. Choi, J. M. Popovich, and C. Radcliffe, "Solutions to the inverse LQR problem with application to biological systems analysis," *IEEE Transactions on control systems technology*, vol. 23, no. 2, pp. 770–777, 2014.

[35] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[36] C. Jiang, S. Chen, Y. Chen, B. Zhang, Z. Feng, H. Zhou, and Y. Bo, "A MEMS IMU De-Noising method using long short term memory recurrent neural networks (LSTM-RNN)," *Sensors*, vol. 18, no. 10, p. 3470, 2018.