Robotics and Autonomous Systems 87 (2017) 120–132

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot

Incremental sparse GP regression for continuous-time trajectory estimation and mapping

Xinyan Yan [a,*], Vadim Indelman [b], Byron Boots [a]

[a] Institute for Robotics and Intelligent Machines, College of Computing, Georgia Institute of Technology, Atlanta, GA 30332, USA
[b] Faculty of Aerospace Engineering, Technion - Israel Institute of Technology, Haifa 32000, Israel

HIGHLIGHTS

- An incremental sparse GP regression algorithm for STEAM problems is proposed.
- The benefits of GP-based approaches and incremental smoothing are combined.
- The approach elegantly handles asynchronous and sparse measurements.
- Results indicate significant speed-up in performance with little loss in accuracy.

ARTICLE INFO

Article history:
Received 26 January 2016
Accepted 4 October 2016
Available online 14 October 2016

Keywords:
State estimation
Localization
Continuous time
SLAM
Gaussian process regression

ABSTRACT

Recent work on simultaneous trajectory estimation and mapping (STEAM) for mobile robots has used Gaussian processes (GPs) to efficiently represent the robot's trajectory through its environment. GPs have several advantages over discrete-time trajectory representations: they can represent a continuous-time trajectory, elegantly handle asynchronous and sparse measurements, and allow the robot to query the trajectory to recover its estimated position at any time of interest. A major drawback of the GP approach to STEAM is that it is formulated as a *batch* trajectory estimation problem. In this paper we provide the critical extensions necessary to transform the existing GP-based batch algorithm for STEAM into an extremely efficient incremental algorithm. In particular, we are able to vastly speed up the solution time through efficient variable reordering and incremental sparse updates, which we believe will greatly increase the practicality of Gaussian process methods for robot mapping and localization. Finally, we demonstrate the approach and its advantages on both synthetic and real datasets.

1. Introduction & related work

Simultaneously recovering the location of a robot and a map of its environment from sensor readings is a fundamental challenge in robotics [1–3]. Well-known approaches to this problem, such as square root smoothing and mapping (SAM) [4], have focused on regression-based methods that exploit the sparse structure of the problem to efficiently compute a solution. The main weakness of the original SAM algorithm was that it was a *batch* method: all of the data must be collected before a solution can be found. For a robot traversing an environment, the inability to update an estimate of its trajectory online is a significant drawback. In response to this weakness, Kaess et al. [5] developed a critical extension to the batch SAM algorithm, iSAM, that overcomes this problem by *incrementally* computing a solution. The main drawback of iSAM, was that the approach required costly periodic batch steps for variable reordering to maintain sparsity and relinearization. This approach was extended in iSAM 2.0 [6], which employs an efficient data structure called the *Bayes tree* [7] to perform incremental variable reordering and just-in-time relinearization, thereby eliminating the bottleneck caused by batch variable reordering and relinearization. The iSAM 2.0 algorithm and its extensions are widely considered to be state-of-the-art in robot trajectory estimation and mapping.

The majority of previous approaches to trajectory estimation and mapping, including the smoothing-based SAM family of algorithms, have formulated the problem in discrete time [1–4,6,8,9]. However, discrete-time representations are restrictive: they are not easily extended to trajectories with irregularly spaced waypoints or asynchronously sampled measurements. A continuous-time formulation of the SAM problem where measurements constrain the trajectory at any point in time, would elegantly contend with these difficulties. Viewed from this perspective, the robot trajectory is a *function* $\boldsymbol{x}(t)$, that maps any time t to a robot

* Corresponding author.
E-mail address: xinyan.yan@cc.gatech.edu (X. Yan).

http://dx.doi.org/10.1016/j.robot.2016.10.004
0921-8890/

state. The problem of estimating this function along with landmark locations has been dubbed *simultaneous trajectory estimation and mapping* (STEAM) [10].

Tong et al. [11] proposed a Gaussian process (GP) regression approach to solving the STEAM problem. While their approach was able to accurately model and interpolate asynchronous data to recover a trajectory and landmark estimate, it suffered from significant computational challenges: naive Gaussian process approaches to regression have notoriously high space and time complexity. Additionally, Tong et al.'s approach is a *batch* method, so updating the solution necessitates saving all of the data and completely re-solving the problem. In order to combat the computational burden, Tong et al.'s approach was extended in Barfoot et al. [10] to take advantage of the sparse structure inherent in the STEAM problem. The resulting algorithm significantly speeds up solution time and can be viewed as a continuous-time analog of Dellaert's original square-root SAM algorithm [4]. Unfortunately, like SAM, Barfoot et al.'s GP-based algorithm remains a batch algorithm, which is a disadvantage for robots that need to continually update the estimate of their trajectory and environment.

In this work, we provide the critical extensions necessary to transform the existing Gaussian process-based approach to solving the STEAM problem into an extremely efficient incremental approach. Our algorithm elegantly combines the benefits of Gaussian processes and iSAM 2.0. Like the GP regression approaches to STEAM, our approach can model continuous trajectories, handle asynchronous measurements, and naturally interpolate states to speed up computation and reduce storage requirements, and, like iSAM 2.0, our approach uses a Bayes tree to efficiently calculate a *maximum a posteriori* (MAP) estimate of the GP trajectory while performing incremental factorization, variable reordering, and just-in-time relinearization. The result is an online GP-based solution to the STEAM problem that remains computationally efficient while scaling up to large datasets.

The present paper is an extension of the work presented in [12,13]. As a further contribution, in this manuscript we elaborate more on variable re-ordering that is key to making both batch and incremental GP-regression computationally more efficient, and we study the performance of the proposed approach in an additional real-world dataset. Furthermore, we release an efficient implementation of the approach developed herein as open source code.[1]

2. Batch trajectory estimation & mapping as Gaussian process regression

We begin by describing how the simultaneous trajectory estimation and mapping (STEAM) problem can be formulated in terms of Gaussian process regression. Following Tong et al. [11] and Barfoot et al. [10], we represent robot trajectories as functions of time t sampled from a Gaussian process:

$$\boldsymbol{x}(t) \sim \mathcal{GP}(\boldsymbol{\mu}(t), \mathcal{K}(t, t')), \qquad t_0 < t, t'. \tag{1}$$

Here, $\boldsymbol{x}(t)$ is the continuous-time trajectory of the robot through state-space, represented by a Gaussian process with mean $\boldsymbol{\mu}(t)$ and covariance $\mathcal{K}(t, t')$ functions.

We next define a finite set of measurements:

$$\boldsymbol{y}_i = \boldsymbol{h}_i(\boldsymbol{\theta}_i) + \boldsymbol{n}_i, \quad \boldsymbol{n}_i \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{R}_i), \quad i = 1, 2, \ldots, N. \tag{2}$$

The measurement $\boldsymbol{y}_i$ can be any linear or nonlinear functions of a set of related variables $\boldsymbol{\theta}_i$ plus some Gaussian noise $\boldsymbol{n}_i$. The related variables for a range measurement are the robot state at the corresponding measurement time $\boldsymbol{x}(t_i)$ and the associated landmark location $\boldsymbol{\ell}_j$. We assume the total number of measurements is N, and the number of trajectory states at measurement times is M.

Based on the definition of Gaussian processes, any finite collection of robot states has a joint Gaussian distribution [14]. So the robot states at measurement times are normally distributed with mean $\boldsymbol{\mu}$ and covariance $\mathcal{K}$.

$$\begin{aligned} &\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \mathcal{K}), \quad \boldsymbol{x} = [\boldsymbol{x}_1^\mathsf{T} \quad \ldots \quad \boldsymbol{x}_M^\mathsf{T}]^\mathsf{T}, \quad \boldsymbol{x}_i = \boldsymbol{x}(t_i) \\ &\boldsymbol{\mu} = [\boldsymbol{\mu}_1^\mathsf{T} \quad \ldots \quad \boldsymbol{\mu}_M^\mathsf{T}]^\mathsf{T}, \quad \boldsymbol{\mu}_i = \boldsymbol{\mu}(t_i), \quad \mathcal{K}_{ij} = \mathcal{K}(t_i, t_j). \end{aligned} \tag{3}$$

Note that any point along the continuous-time trajectory can be estimated from the Gaussian process model. Therefore, the trajectory does not need to be discretized and robot trajectory states do not need to be evenly spaced in time, which is an advantage of the Gaussian process approach over discrete-time approaches (e.g. Dellaert's square-root SAM [4]).

The landmarks $\boldsymbol{\ell}$ which represent the map are assumed to conform to a joint Gaussian distribution with mean $\boldsymbol{d}$ and covariance $\boldsymbol{W}$ (Eq. (4)). The prior distribution of the combined state $\boldsymbol{\theta}$ that consists of robot trajectory states at measurement times and landmarks is, therefore, a joint Gaussian distribution (Eq. (5)).

$$\boldsymbol{\ell} \sim \mathcal{N}(\boldsymbol{d}, \boldsymbol{W}), \quad \boldsymbol{\ell} = [\boldsymbol{\ell}_1^\mathsf{T} \quad \boldsymbol{\ell}_2^\mathsf{T} \quad \ldots \quad \boldsymbol{\ell}_O^\mathsf{T}]^\mathsf{T} \tag{4}$$

$$\boldsymbol{\theta} \sim \mathcal{N}(\boldsymbol{\eta}, \mathcal{P}), \quad \boldsymbol{\eta} = [\boldsymbol{\mu}^\mathsf{T} \quad \boldsymbol{d}^\mathsf{T}]^\mathsf{T}, \quad \mathcal{P} = \begin{bmatrix} \mathcal{K} & \\ & \boldsymbol{W} \end{bmatrix}. \tag{5}$$

To solve the STEAM problem, given the prior distribution of the combined state and the likelihood of measurements, we compute the *maximum a posteriori* (MAP) estimate of the combined state *conditioned* on measurements via Bayes' rule:

$$\begin{aligned} \boldsymbol{\theta}^* &\triangleq \boldsymbol{\theta}_{MAP} = \operatorname*{argmax}_{\boldsymbol{\theta}} p(\boldsymbol{\theta}|\boldsymbol{y}) = \operatorname*{argmax}_{\boldsymbol{\theta}} \frac{p(\boldsymbol{\theta})p(\boldsymbol{y}|\boldsymbol{\theta})}{p(\boldsymbol{y})} \\ &= \operatorname*{argmax}_{\boldsymbol{\theta}} p(\boldsymbol{\theta})p(\boldsymbol{y}|\boldsymbol{\theta}) = \operatorname*{argmin}_{\boldsymbol{\theta}} \left(-\log p(\boldsymbol{\theta}) - \log p(\boldsymbol{y}|\boldsymbol{\theta})\right) \\ &= \operatorname*{argmin}_{\boldsymbol{\theta}} \left(\|\boldsymbol{\theta} - \boldsymbol{\eta}\|_{\mathcal{P}}^2 + \|\boldsymbol{h}(\boldsymbol{\theta}) - \boldsymbol{y}\|_{\boldsymbol{R}}^2\right) \end{aligned} \tag{6}$$

where the norms are Mahalanobis norms defined as: $\|\boldsymbol{e}\|_{\Sigma}^2 = \boldsymbol{e}^\mathsf{T}\Sigma^{-1}\boldsymbol{e}$, and $\boldsymbol{h}(\boldsymbol{\theta})$ and $\boldsymbol{R}$ are the mean and covariance of the measurements collected, respectively:

$$\boldsymbol{h}(\boldsymbol{\theta}) = [\boldsymbol{h}_1(\boldsymbol{\theta}_1)^\mathsf{T} \quad \boldsymbol{h}_2(\boldsymbol{\theta}_2)^\mathsf{T} \quad \ldots \quad \boldsymbol{h}_N(\boldsymbol{\theta}_N)^\mathsf{T}]^\mathsf{T} \tag{7}$$

$$\boldsymbol{R} = \operatorname{diag}(\boldsymbol{R}_1, \boldsymbol{R}_2, \ldots, \boldsymbol{R}_N). \tag{8}$$

Because both covariance matrices $\mathcal{P}$ and $\boldsymbol{R}$ are positive definite, the objective in Eq. (6) corresponds to a least squares problem. Consequently, if some of the measurement functions $\boldsymbol{h}_i(\cdot)$ are nonlinear, this becomes a nonlinear least squares problem, in which case iterative methods including Gauss–Newton and Levenberg–Marquardt [15] can be utilized; in each iteration, an optimal update is computed given a linearized problem at the current estimate. A linearization of a measurement function at current state estimate $\bar{\boldsymbol{\theta}}_i$ can be accomplished by a first-order Taylor expansion:

$$\boldsymbol{h}_i\left(\bar{\boldsymbol{\theta}}_i + \delta\boldsymbol{\theta}_i\right) \approx \boldsymbol{h}_i(\bar{\boldsymbol{\theta}}_i) + \left.\frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{\theta}_i}\right|_{\bar{\boldsymbol{\theta}}_i} \delta\boldsymbol{\theta}_i. \tag{9}$$

Combining Eq. (9) with Eq. (6), the optimal increment $\delta\boldsymbol{\theta}^*$ at the current combined state estimate $\bar{\boldsymbol{\theta}}$ is

$$\delta\boldsymbol{\theta}^* = \operatorname*{argmin}_{\delta\boldsymbol{\theta}} \left(\|\bar{\boldsymbol{\theta}} + \delta\boldsymbol{\theta} - \boldsymbol{\eta}\|_{\mathcal{P}}^2 + \|\boldsymbol{h}(\bar{\boldsymbol{\theta}}) + \boldsymbol{H}\delta\boldsymbol{\theta} - \boldsymbol{y}\|_{\boldsymbol{R}}^2\right) \tag{10}$$

$$\boldsymbol{H} = \operatorname{diag}(\boldsymbol{H}_1, \boldsymbol{H}_2, \ldots, \boldsymbol{H}_N), \qquad \boldsymbol{H}_i = \left.\frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{\theta}_i}\right|_{\bar{\boldsymbol{\theta}}_i} \tag{11}$$

where $\boldsymbol{H}$ is the measurement Jacobian matrix. To solve the linear least squares problem in Eq. (10), we take the derivative with

[1] Please check out the code at https://github.com/XinyanGT/online-gpslam-code.

respect to $\delta\boldsymbol{\theta}$, and set it to zero, which gives us $\delta\boldsymbol{\theta}^*$ embedded in a set of linear equations

$$\underbrace{(\boldsymbol{\mathcal{P}}^{-1}+\boldsymbol{H}^{\mathsf{T}}\boldsymbol{R}^{-1}\boldsymbol{H})}_{\boldsymbol{\mathcal{I}}}\delta\boldsymbol{\theta}^* = \underbrace{\boldsymbol{\mathcal{P}}^{-1}(\boldsymbol{\eta}-\bar{\boldsymbol{\theta}})+\boldsymbol{H}^{\mathsf{T}}\boldsymbol{R}^{-1}(\boldsymbol{y}-\bar{\boldsymbol{h}})}_{\boldsymbol{b}} \tag{12}$$

with covariance

$$\mathrm{cov}(\delta\boldsymbol{\theta}^*, \delta\boldsymbol{\theta}^*) = \boldsymbol{\mathcal{I}}^{-1}. \tag{13}$$

The positive definite matrix $\boldsymbol{\mathcal{P}}^{-1} + \boldsymbol{H}^{\mathsf{T}}\boldsymbol{R}^{-1}\boldsymbol{H}$ is the *a posteriori* information matrix, which we label $\boldsymbol{\mathcal{I}}$. To solve this set of linear equations for $\delta\boldsymbol{\theta}^*$, we do not actually have to calculate the inverse $\boldsymbol{\mathcal{I}}^{-1}$. Instead, factorization-based methods can provide a fast, numerically stable solution. For example, $\delta\boldsymbol{\theta}^*$ can be found by first performing a Cholesky factorization $\boldsymbol{\mathcal{L}}\boldsymbol{\mathcal{L}}^{\mathsf{T}} = \boldsymbol{\mathcal{I}}$, and then solving $\boldsymbol{\mathcal{L}}\boldsymbol{d} = \boldsymbol{b}$ and $\boldsymbol{\mathcal{L}}^{\mathsf{T}}\delta\boldsymbol{\theta}^* = \boldsymbol{d}$ by back substitution. At each iteration we perform a *batch* state estimation update $\bar{\boldsymbol{\theta}} \leftarrow \bar{\boldsymbol{\theta}} + \delta\boldsymbol{\theta}^*$ and repeat the process until convergence.

If $\boldsymbol{\mathcal{I}}$ is dense, the time complexity of a Cholesky factorization and back substitution are $O(n^3)$ and $O(n^2)$ respectively, where $\boldsymbol{\mathcal{I}} \in \mathbb{R}^{n\times n}$ [16]. However, if $\boldsymbol{\mathcal{I}}$ has sparse structure, then the solution can be found much faster. For example, for a narrowly banded matrix, the computation time is $O(n)$ instead of $O(n^3)$ [16]. Fortunately, we can guarantee sparsity for the STEAM problem (see Section 2.2 below).

2.1. State interpolation

An advantage of the Gaussian process representation of the robot trajectory is that any trajectory state can be interpolated from other states by computing the posterior mean [11]:

$$\bar{\boldsymbol{x}}(t) = \boldsymbol{\mu}(t) + \boldsymbol{\mathcal{K}}(t)\boldsymbol{\mathcal{K}}^{-1}(\bar{\boldsymbol{x}} - \boldsymbol{\mu}), \tag{14}$$

with

$$\bar{\boldsymbol{x}} = [\bar{\boldsymbol{x}}_1^{\mathsf{T}} \quad \dots \quad \bar{\boldsymbol{x}}_M^{\mathsf{T}}]^{\mathsf{T}} \quad \text{and}$$

$$\boldsymbol{\mathcal{K}}(t) = [\boldsymbol{\mathcal{K}}(t, t_1) \quad \dots \quad \boldsymbol{\mathcal{K}}(t, t_M)]. \tag{15}$$

By utilizing interpolation, we can reduce the number of robot trajectory states that we need to estimate in the optimization procedure [11]. For simplicity, assume $\boldsymbol{\theta}_i$, the set of the related variables of the ith measurement according to the model (Eq. (2)), is $\boldsymbol{x}(\tau)$. Then, after interpolation, Eq. (9) becomes:

$$\begin{aligned}\boldsymbol{h}_i\left(\bar{\boldsymbol{\theta}}_i + \delta\boldsymbol{\theta}_i\right) &= \boldsymbol{h}_i\left(\bar{\boldsymbol{x}}(\tau) + \delta\boldsymbol{x}(\tau)\right)\\ &\approx \boldsymbol{h}_i(\bar{\boldsymbol{x}}(\tau)) + \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{x}(\tau)} \cdot \frac{\partial \boldsymbol{x}(\tau)}{\partial \boldsymbol{x}}\bigg|_{\bar{\boldsymbol{x}}} \delta\boldsymbol{x}\\ &= \boldsymbol{h}_i\left(\boldsymbol{\mu}(\tau)+\boldsymbol{\mathcal{K}}(\tau)\boldsymbol{\mathcal{K}}^{-1}(\bar{\boldsymbol{x}}-\boldsymbol{\mu})\right)+\boldsymbol{H}_i\boldsymbol{\mathcal{K}}(\tau)\boldsymbol{\mathcal{K}}^{-1}\delta\boldsymbol{x}.\end{aligned} \tag{16}$$

By employing Eq. (16) during optimization, we can make use of measurement i without explicitly estimating the trajectory states that it relates to. We exploit this advantage to greatly speed up the solution to the STEAM problem in practice (Section 5).

2.2. Sparse Gaussian process regression

The efficiency of the Gaussian process Gauss–Newton algorithm presented in Section 2 is heavily dependent on the choice of kernel. It is well-known that if the information matrix $\boldsymbol{\mathcal{I}}$ is sparse, then it is possible to very efficiently compute the solution to Eq. (12) [4]. Barfoot et al. suggest a kernel matrix with a sparse inverse that is well-suited to the simultaneous trajectory estimation and mapping problem [10]. In particular, Barfoot et al. show that $\boldsymbol{\mathcal{K}}^{-1}$ is exactly block-tridiagonal when the GP is assumed to be generated by linear, time-varying (LTV) stochastic differential equation (SDE) which we describe here:

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{A}(t)\boldsymbol{x}(t) + \boldsymbol{v}(t) + \boldsymbol{F}(t)\boldsymbol{w}(t), \tag{17}$$

$$\boldsymbol{w}(t) \sim \mathcal{GP}(\boldsymbol{0},\ \boldsymbol{Q}_c\delta(t-t')) \quad t_0 < t, t' \tag{18}$$

where $\boldsymbol{x}(t)$ is trajectory, $\boldsymbol{v}(t)$ is known exogenous input, $\boldsymbol{w}(t)$ is process noise, and $\boldsymbol{F}(t)$ is time-varying system matrix. The process noise $\boldsymbol{w}(t)$ is modeled by a Gaussian process, and $\delta(\cdot)$ is the *Dirac delta function*. (See [10] for details). We consider a specific case of this model in the experimental results in Section 5.1. Because the mean function $\boldsymbol{\mu}(t)$ is an integral of the known exogenous input $\boldsymbol{v}(t)$, the assumption of zero $\boldsymbol{v}(t)$ leads to Gaussian process with zero mean $\boldsymbol{\mu}(t)$.

Assuming the GP is generated by Eq. (17), the measurements are landmark and odometry measurements, and the variables are ordered in XL ordering,[2] the sparse information matrix becomes

$$\boldsymbol{\mathcal{I}} = \begin{bmatrix}\boldsymbol{\mathcal{I}}_{xx} & \boldsymbol{\mathcal{I}}_{x\ell}\\ \boldsymbol{\mathcal{I}}_{x\ell}^{\mathsf{T}} & \boldsymbol{\mathcal{I}}_{\ell\ell}\end{bmatrix} \tag{19}$$

where $\boldsymbol{\mathcal{I}}_{xx}$ is block-tridiagonal and $\boldsymbol{\mathcal{I}}_{\ell\ell}$ is block-diagonal. $\boldsymbol{\mathcal{I}}_{x\ell}$'s density depends on the frequency of landmark measurements, and how they are taken. See Fig. 1(a) for an example.

When the GP is generated by LTV SDE, $\boldsymbol{\mathcal{K}}(\tau)\boldsymbol{\mathcal{K}}^{-1}$ in Eq. (14) has a specific sparsity pattern—only two column blocks that correspond to trajectory states at t_{i-1} and t_i are nonzero ($t_{i-1} < \tau < t_i$) [10]:

$$\boldsymbol{\mathcal{K}}(\tau)\boldsymbol{\mathcal{K}}^{-1} = \begin{bmatrix}0 & \dots & 0 & \boldsymbol{\Lambda}(\tau) & \boldsymbol{\Psi}(\tau) & 0 & \dots & 0\end{bmatrix} \tag{20}$$

$$\begin{aligned}\boldsymbol{\Lambda}(\tau) &= \boldsymbol{\Phi}(\tau, t_{i-1}) - \boldsymbol{Q}_\tau \boldsymbol{\Phi}(t_i, \tau)^{\mathsf{T}}\boldsymbol{Q}_i^{-1}\boldsymbol{\Phi}(t_i, t_{i-1})\\ \boldsymbol{\Psi}(\tau) &= \boldsymbol{Q}_\tau \boldsymbol{\Phi}(t_i, \tau)^{\mathsf{T}}\boldsymbol{Q}_i^{-1}\end{aligned}$$

$\boldsymbol{\Phi}(\tau, s)$ is the state transition matrix from s to τ. $\boldsymbol{Q}_\tau$ is the integral of $\boldsymbol{Q}_c$, the covariance of the process noise $\boldsymbol{w}(t)$ (Eq. (17)):

$$\boldsymbol{Q}_\tau = \int_{t_{i-1}}^{\tau} \boldsymbol{\Phi}(\tau, s)\boldsymbol{F}(s)\boldsymbol{Q}_c\boldsymbol{F}(s)^{\mathsf{T}}\boldsymbol{\Phi}(\tau, s)^{\mathsf{T}}ds. \tag{21}$$

And $\boldsymbol{Q}_i$ is the integral from t_{i-1} to t.

Consequently, based on Eqs. (14) and (20), $\bar{\boldsymbol{x}}(\tau)$ is an affine function of only two nearby states $\bar{\boldsymbol{x}}_{i-1}$ and $\bar{\boldsymbol{x}}_i$ (the current estimate of the states at t_{i-1} and t_i, $t_{i-1} < \tau < t_i$):

$$\bar{\boldsymbol{x}}(\tau) = \boldsymbol{\mu}(\tau) + \begin{bmatrix}\boldsymbol{\Lambda}(\tau) & \boldsymbol{\Psi}(\tau)\end{bmatrix}\left(\begin{bmatrix}\bar{\boldsymbol{x}}_{i-1}\\ \bar{\boldsymbol{x}}_i\end{bmatrix} - \begin{bmatrix}\boldsymbol{\mu}_{i-1}\\ \boldsymbol{\mu}_i\end{bmatrix}\right). \tag{22}$$

Thus, it only takes $O(1)$ time to query any $\bar{\boldsymbol{x}}(\tau)$ using Eq. (22). Moreover, because interpolation of a state is only determined by the two nearby states, measurement interpolation in Eq. (16) can be simplified to:

$$\begin{aligned}\boldsymbol{h}_k\left(\bar{\boldsymbol{\theta}}_k + \delta\boldsymbol{\theta}_k\right) &= \boldsymbol{h}_k\left(\bar{\boldsymbol{x}}(\tau) + \delta\boldsymbol{x}(\tau)\right)\\ &\approx \boldsymbol{h}_k(\bar{\boldsymbol{x}}(\tau)) + \frac{\partial \boldsymbol{h}_k}{\partial \boldsymbol{x}(\tau)} \cdot \frac{\partial \boldsymbol{x}(\tau)}{\partial \boldsymbol{x}}\bigg|_{\bar{\boldsymbol{x}}} \delta\boldsymbol{x}\\ &= \boldsymbol{h}_k(\bar{\boldsymbol{x}}(\tau))+\boldsymbol{H}_k\begin{bmatrix}\boldsymbol{\Lambda}(\tau) & \boldsymbol{\Psi}(\tau)\end{bmatrix}\begin{bmatrix}\delta\boldsymbol{x}_{i-1}\\ \delta\boldsymbol{x}_i\end{bmatrix}\end{aligned} \tag{23}$$

with $\bar{\boldsymbol{x}}(\tau)$ defined in Eq. (22).

3. Batch GP-regression with variable reordering

Previous work on batch continuous-time trajectory estimation as sparse Gaussian process regression [10,11] assumes that the information matrix $\boldsymbol{\mathcal{I}}$ is sparse (Eq. (19)) and applies standard block

[2] XL ordering is an ordering where process variables come before landmarks variables.

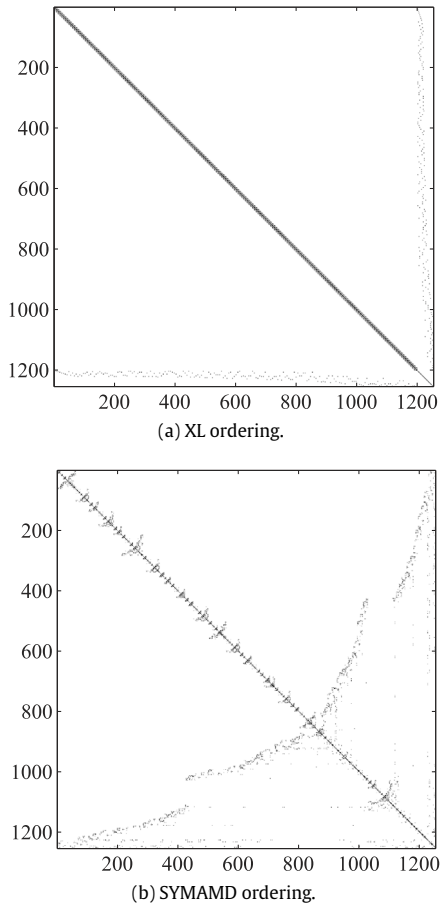


Fig. 1. Sparse information matrices. The information matrix $\mathcal{I}$ with XL ordering (a), and SYMAMD ordering (b). Both sparse matrices have the same number of non-zero elements, yet the second matrix can be factored much more efficiently due to the heuristic ordering of the matrix columns. (See Table 1.) For illustration, only 200 trajectory states are shown here.

elimination to factor and solve Eq. (12). Despite the sparsity of $\mathcal{I}$, for large numbers of landmarks this process can be very inefficient. Inspired by square root SAM [4], which uses variable reordering for efficient Cholesky factorization in a discrete-time context, we show that factorization-time can be dramatically improved by matrix column reordering in the sparse Gaussian process context as well.

It is reasonable to base our approach on SAM because the information matrix and factor graph of the sparse GP [10] has structure similar to the SAM formulations of the problem [4,5], and the intuitions from previous discrete-time approaches apply here. If the Cholesky decompositions are performed naively, fill-in can occur, where entries that are zero in the information matrix become non-zero in the Cholesky factor. This occurs because the Cholesky factor of a sparse matrix is guaranteed to be sparse for some variable orderings, but not all variable orderings [17]. Therefore, we want to find a good variable ordering so that the Cholesky factor is sparse.

Although finding the optimal ordering for a symmetric positive definite matrix is NP-complete [18], good heuristics do exist. One such heuristic is Symmetric Approximate Minimum Degree Permutation (SYMAMD), which is a variant of Column Approximate Minimum Degree Ordering (COLAMD) [19] on a positive definite matrix [19]. Specifically, matrix $\boldsymbol{A}$ is constructed such that the sparsity pattern of $\boldsymbol{A}^{\mathsf{T}}\boldsymbol{A}$ is the same as the information matrix $\mathcal{I}$. Then COLAMD is applied to $\boldsymbol{A}$. SYMAMD produces a reasonable

Table 1
Cost of Cholesky factorization with different ordering methods including ordering time.

	XL	SYMAMD	Block SYMAMD
nnz[a]	1817k (100%)	192k (10.6%)	176k (9.7%)
Time (s)	0.9677 (100%)	0.0274 (2.83%)	0.0175 (1.81%)

[a] The number of non-zero elements.

ordering for the Cholesky factorization in practice, but, in general, does not guarantee the sparsity of the resultant Cholesky factor. Alternative methods, for example, (1) utilizing ordering algorithm designed for symmetric matrices, e.g. AMDBAR [20], or (2) directly providing COLAMD $\boldsymbol{A}$ based on the definition of $\mathcal{I}$ in Eq. (12)), are worthy of further investigation. The computation complexity of COLAMD on $\boldsymbol{A}$ is the same as the sparse matrix multiplication of $\boldsymbol{A}$ and the $\mathcal{U}$ in $\boldsymbol{A}$'s LU factorization. In practice, it is much faster than the factorization.

To demonstrate the benefits of variable reordering, we constructed a synthetic example and compared different approaches. The example, which is explained in detail in Section 5.1, consists of 1500 time steps with trajectory states, $\boldsymbol{x}_i = [\boldsymbol{p}_i \quad \dot{\boldsymbol{p}}_i]^{\mathsf{T}}$, $\boldsymbol{p}_i = [x_i \quad y_i \quad \theta_i]^{\mathsf{T}}$, and with odometry and range measurements. The total number of landmarks is 298. The structure of the information matrix $\mathcal{I}$ and Cholesky factor $\mathcal{L}$, with and without variable reordering, are compared in Figs. 1 and 2. Although variable reordering does not change the sparsity of the information matrix $\mathcal{I}$ (Fig. 1), it dramatically increases the sparsity of the Cholesky factor $\mathcal{L}$ (Fig. 2). Table 1 demonstrates this clear benefit of reordering. The Cholesky factor after SYMAMD ordering contains 10.6% non-zeros of XL ordering, and takes 2.83% of the time, which are significant improvements in both time and space complexity.

We also experimented with block SYMAMD [4], which exploits domain knowledge to group together variables belonging to a particular trajectory state $\boldsymbol{x}(t_i)$ or landmark location $\boldsymbol{\ell}_j$ before performing SYMAMD. Empirically we found that this further improves performance.

It is straightforward to incorporate variable reordering methods like SYMAMD and block SYMAMD into the batch GP-Regression algorithm from Section 2: given a new batch of data, directly update the sparse information matrix $\mathcal{I}$, reorder the variables with (block) SYMAMD, and then recompute the Cholesky factor $\mathcal{L}$ on the way to solving for $\delta\boldsymbol{\theta}$ in Eq. (12).

In most STEAM problems, we are interested in estimating the robot's trajectory *as it traverses the environment*. In Alg. 1, we accomplish this by repeatedly executing the batch algorithm with variable reordering. Although this approach seems like it should be very costly, with variable reordering it is actually quite efficient. Building and factoring the sparse information matrix is much faster than the linearization step required for a single iteration of the Gauss–Newton algorithm. Since the computational bottleneck is not the Cholesky decomposition, but rather the relinearization of the measurement model, we suggest only periodic Gauss–Newton iterations.

4. The Bayes tree data structure for fast incremental updates to sparse Gaussian process regression

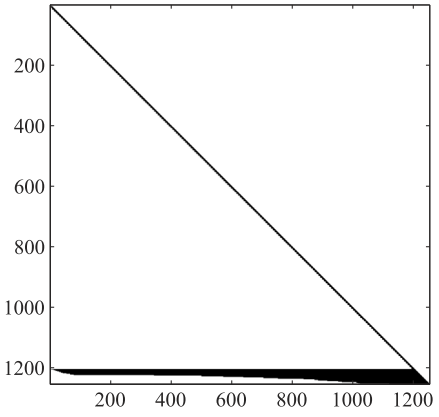
Despite the efficiency of periodic batch updates, Alg. 1 is still repeatedly executing a batch algorithm that requires reordering and refactoring $\mathcal{I}$, and periodically relinearizing the measurement function for all of the estimated states each time new data is collected. Here we provide the extensions necessary to avoid these costly steps and turn the naive batch algorithm into an efficient, truly incremental, algorithm. The key idea is to perform just-in-time relinearization and to efficiently *update* an existing sparse factorization instead of re-calculating one from scratch.

Algorithm 1 Periodic Batch Sparse GP Regression

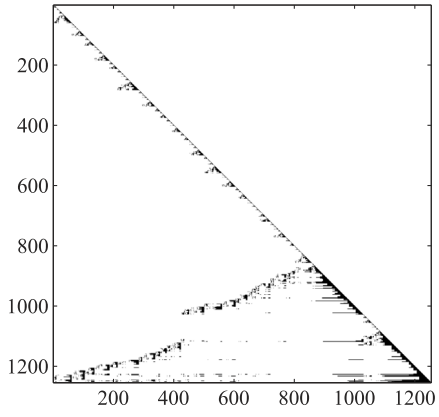
while collecting data **do**

1. Get measurement results belonging to this period, $\boldsymbol{y} \leftarrow [\boldsymbol{y}, \boldsymbol{y}_{new}]^\intercal$
2. Initial guess for the newly encountered states, $\bar{\boldsymbol{\theta}} \leftarrow [\bar{\boldsymbol{\theta}}, \bar{\boldsymbol{\theta}}_{new}]^\intercal$
3. Build the measurement Jacobian $\boldsymbol{H}$, and then $\boldsymbol{\mathcal{I}}$ and $\boldsymbol{b}$ required in Eq. (12)
4. Find an ordering p for $\boldsymbol{\mathcal{I}}$, and reorder $\boldsymbol{\mathcal{I}}_p \overset{p}{\leftarrow} \boldsymbol{\mathcal{I}}$, $\boldsymbol{b}_p \overset{p}{\leftarrow} \boldsymbol{b}$
5. Solve $\boldsymbol{\mathcal{I}}_p \delta\boldsymbol{\theta}_p^* = \boldsymbol{b}_p$ using Cholesky factorization
6. Recover the solution $\delta\boldsymbol{\theta}^* \overset{r}{\leftarrow} \delta\boldsymbol{\theta}_p^*$ by inverse ordering $r = p^{-1}$
7. Update estimate $\bar{\boldsymbol{\theta}} \leftarrow \bar{\boldsymbol{\theta}} + \delta\boldsymbol{\theta}^*$

end while



(a) XL ordering.



(b) SYMAMD ordering.

Fig. 2. The Cholesky factors $\boldsymbol{\mathcal{L}}$ of $\boldsymbol{\mathcal{I}}$. In (a), $\boldsymbol{\mathcal{L}}$ is computed with XL ordering, which exhibits fill-in. In (b), $\boldsymbol{\mathcal{L}}$ is computed with SYMAMD ordering, which is more sparse. For illustration, only 200 states are shown here.

4.1. The Bayes tree data structure

We base our approach on iSAM 2.0 proposed by Kaess et al. [6], which was designed to efficiently solve a nonlinear estimation problem in an incremental and real-time manner by directly operating on the factor graph representation of the SAM problem. The core technology behind iSAM 2.0 is the *Bayes tree* data structure which allows for incremental variable reordering and fluid relinearization [7]. As demonstrated by Kaess et al. [6], the Bayes tree provides a dramatic speedup compared to the periodic batch method, while only incurring a negligible loss in accuracy. We apply the same data structure to sparse Gaussian process regression in the context of the STEAM problem, thereby eliminating the need for periodic batch computation.

The Bayes tree data structure captures the formal equivalence between the sparse QR factorization in linear algebra and the inference in graphical models, translating *abstract updates* to a matrix factorization into *intuitive edits* to a graph. Here we give a brief introduction of Bayes trees (see [7] for details), and how they help solve the sparse Gaussian process regression incrementally.

A Bayes tree is constructed from a Bayes net, which is itself constructed from a factor graph. A factor graph is a bipartite graph $G = (\boldsymbol{\theta}, \mathcal{F}, \mathcal{E})$, representing the factorization of a function (Eq. (24)). $\boldsymbol{\theta} = \{\theta_1, \ldots, \theta_m\}$ are *variables*, $\mathcal{F} = \{f_1, \ldots, f_n\}$ are *factors* (functions of variables), and $\mathcal{E}$ are the *edges* that connect these two types of nodes. $e_{ij} \in \mathcal{E}$ if and only if $\theta_j \in \boldsymbol{\theta}_i$ and $f_i(\cdot)$ is a function of $\boldsymbol{\theta}_i$.

$$f(\boldsymbol{\theta}) = \prod_i f_i(\boldsymbol{\theta}_i). \tag{24}$$

In the context of localization and mapping, a factor graph encodes the complex probability estimation problem in a graphical model. It represents the *joint density* of the variables consisting of both trajectory and mapping, and factors correspond to the soft constraints imposed by the measurements and priors. If we assume that the priors are Gaussian, measurements have Gaussian noise, and measurement functions are linear or linearized, as in Section 2, the joint density becomes a product of Gaussian distributions:

$$f(\boldsymbol{\theta}) \propto \exp\left\{-\frac{1}{2}\sum \|\boldsymbol{A}_i\boldsymbol{\theta}_i - \boldsymbol{b}_i\|_2^2\right\} = \exp\left\{-\frac{1}{2}\|\boldsymbol{A}\boldsymbol{\theta} - \boldsymbol{b}\|_2^2\right\}. \tag{25}$$

Here $\boldsymbol{A}_i$ and $\boldsymbol{b}_i$ are derived from factor $f_i(\cdot)$. $\boldsymbol{A}$ is a square-root information matrix, with $\boldsymbol{\mathcal{I}} = \boldsymbol{A}^\intercal\boldsymbol{A}$ [4], so the QR factor $\boldsymbol{R}$ of $\boldsymbol{A}$ is equal to the transpose of the Cholesky factor $\boldsymbol{\mathcal{L}}$ of $\boldsymbol{\mathcal{I}}$. Maximizing the joint density is equivalent to the least-square problem in Eq. (10).

A Gaussian process generated from linear, time-varying (LTV) stochastic differential equations (SDE), as discussed in Section 2.2, has a block-tridiagonal inverse kernel matrix $\boldsymbol{\mathcal{K}}^{-1}$ [10]. In other words, the resulting Gaussian process prior factors only connect consecutive pairs of states. This leads to a sparse Factor graph (Fig. 3). The Gaussian process prior $f_{\mathcal{GP}}(\cdot)$ between $\boldsymbol{x}_{i-1}$ and $\boldsymbol{x}_i$ is defined as:

$$f_{\mathcal{GP}}(\boldsymbol{x}_{i-1}, \boldsymbol{x}_i) \propto \exp\left\{-\frac{1}{2}\|\boldsymbol{\Phi}(t_i, t_{i-1})\boldsymbol{x}_{i-1} + \boldsymbol{v}_i - \boldsymbol{x}_i\|_{\boldsymbol{Q}_i}^2\right\} \tag{26}$$

where $\boldsymbol{\Phi}(t_i, t_{i-1})$ is the state transition matrix, $\boldsymbol{Q}_i$ is the integral of the covariance of the process noise (Eq. (21)), and $\boldsymbol{v}_i$ is the integral of the exogenous input $\boldsymbol{v}(t)$ (Eq. (17)):

$$\boldsymbol{v}_i = \int_{t_{i-1}}^{t_i} \boldsymbol{\Phi}(t_i, s)\boldsymbol{v}(s)ds. \tag{27}$$

An illustrative sparse factor graph example including the GP factors is presented in Fig. 3(a). Note that although the Gaussian process representation of the trajectory is continuous in time, to impose this prior knowledge only $M - 1$ factors connecting adjacent states are required, where M is the total number of states [10].

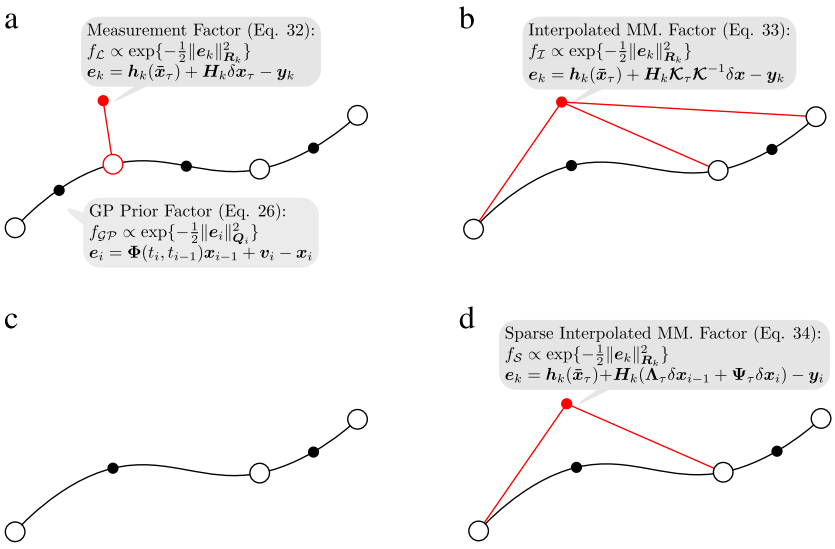


Fig. 3. The effect of interpolation and without interpolation. (a) Measurements are fully utilized. (b) Missing state, using interpolated measurements. (c) Missing state, ignore measurements. (d) Missing state, using sparse interpolated measurements.

The key of just-in-time relinearization and fluid variable reordering is to identify the portion of a factor graph impacted by a new or modified factor. When adding a new measurement, a prior for a new variable, or relinearizing a previous measurement, a factor graph will change accordingly. However, most modifications only have *local* effects. For example, in a pose graph optimization setting [21], due to the fact that relative pose measurements are measurements on the two most recently accessed variables, they only affect the top of the Bayes tree, leaving branches of the tree downstream untouched [6]. Therefore, the time complexity of adding each measurement factor is $O(1)$, although the effect of loop constraints is dependent on the ordering.

Exploiting this observation is the foundation for efficient incremental updates. Identifying the impacted portion is difficult to achieve directly from a factor graph, but in a Bayes tree, it can be efficiently identified as directly affected nodes and their ascendants in the tree. To obtain a Bayes tree from a factor graph, the factor graph is first converted to a Bayes net through the iterative elimination algorithm related to Gaussian elimination. In each step, one variable θ_i is eliminated from the joint density $f(\theta_i, \boldsymbol{s}_i)$ and removed from the factor graph, resulting in a new conditional $P(\theta_i|\boldsymbol{s}_i)$ and a new factor $f(\boldsymbol{s}_i)$, satisfying $f(\theta_i, \boldsymbol{s}_i) = P(\theta_i|\boldsymbol{s}_i)f(\boldsymbol{s}_i)$. The joint density $f(\theta_i, \boldsymbol{s}_i)$ is the product of the factors adjacent to θ_i, and $\boldsymbol{s}_i$ is the set of variables that are connected to these factors, excluding θ_i. The new conditional is added to the Bayes net, and the new factor is added back to the factor graph.

The unnormalized joint density $f(\theta_i, \boldsymbol{s}_i)$ is Gaussian, due to Eq. (25):

$$f(\theta_i, \boldsymbol{s}_i) \propto \exp\left\{-\frac{1}{2}\|\boldsymbol{a}\theta_i + \boldsymbol{A}_s\boldsymbol{s}_i - \boldsymbol{b}_i\|_2^2\right\} \tag{28}$$

where $\boldsymbol{a}$, $\boldsymbol{A}_s$ and $\boldsymbol{b}_i$ correspond to the factors that are currently adjacent to θ_i. These factors can be the factors included in the original factor graph, or the factors induced by the elimination process. The conditional $P(\theta_i|\boldsymbol{s}_i)$ is obtained by evaluating Eq. (28) with a given $\boldsymbol{s}_i$:

$$P(\theta_i|\boldsymbol{s}_i) \propto \exp\left\{-\frac{1}{2}(\theta_i + \boldsymbol{r}^\intercal\boldsymbol{s}_i - d)^2\right\} \tag{29}$$

where $\boldsymbol{r} = (\boldsymbol{a}^\dagger\boldsymbol{A}_s)^\intercal$, $d = \boldsymbol{a}^\dagger\boldsymbol{b}_i$, and $\boldsymbol{a}^\dagger = (\boldsymbol{a}^\intercal\boldsymbol{a})^{-1}\boldsymbol{a}^\intercal$. $f(\boldsymbol{s}_i)$ can be further computed by substituting $\theta_i = d - \boldsymbol{r}^\intercal\boldsymbol{s}_i$ into Eq. (28). This elimination step is equivalent to one step of Gram–Schmidt. Thus the new conditional $P(\theta_i|\boldsymbol{s}_i)$ specifies one row in the $\boldsymbol{R}$ factor of the QR factorization of $\boldsymbol{A}$. The sequence of the variables to be eliminated is selected to reduce fill-in in $\boldsymbol{R}$, just as in the case of matrix column reordering. Analogous to the variable reordering in Section 3 for reducing fill-in in $\mathcal{L}$, the sequence of the variables to be eliminated here influences fill-in in $\boldsymbol{R}$. The joint density $f(\boldsymbol{\theta})$ represented by the Bayes net is maximized by assigning $d - \boldsymbol{r}^\intercal\boldsymbol{s}_i$ to θ_i, due to Eq. (29), starting from the variable that is eliminated last. This procedure is equivalent to the back-substitution in linear algebra. The Bayes net is further transformed into a directed tree graphical model—the Bayes tree. To accomplish this, conditionals belonging to a clique in the Bayes net are grouped together into one node in the Bayes tree. This is performed in reverse elimination order.

When a factor is modified or added to the Bayes tree, the impacted portion of the Bayes tree is re-interpreted as a factor graph, the change is incorporated to the graph, and the graph is eliminated with a new ordering. During elimination, information only flows upward in the Bayes tree, from leaves to the root, so only the ascendants of the nodes that contain the variables involved in the factor are influenced. Additional details of the Bayes tree can be found in Kaess et al. [7]. In particular, a simple diagrammatic treatment can be found in Fig. 6 of [7].

In summary, the Bayes tree can be used to perform fast incremental updates to the Gaussian process representation of the continuous-time trajectory. As we demonstrate in the experimental results, this can greatly increase the efficiency of Barfoot et al.'s batch sparse GP algorithm when the trajectory and map need to be updated online.

Despite the interpretation of the trajectory as a Gaussian process, the approach described above is algorithmically identical to iSAM2.0 when the states associated with each measurement are explicitly estimated. In Section 4.2 below, we extend our incremental algorithm to use Gaussian process interpolation within the Bayes tree. By interpolating missing states, we can handle asynchronous measurements and even remove states in order to speed computation. In Sections 5.1 and 5.2 we show that this results in a significant speedup over iSAM2.0.

4.2. Faster updates through interpolation

To handle asynchronous measurements or to further reduce computation time, we take advantage of Gaussian process state interpolation, described in Section 2.1, within our incremental

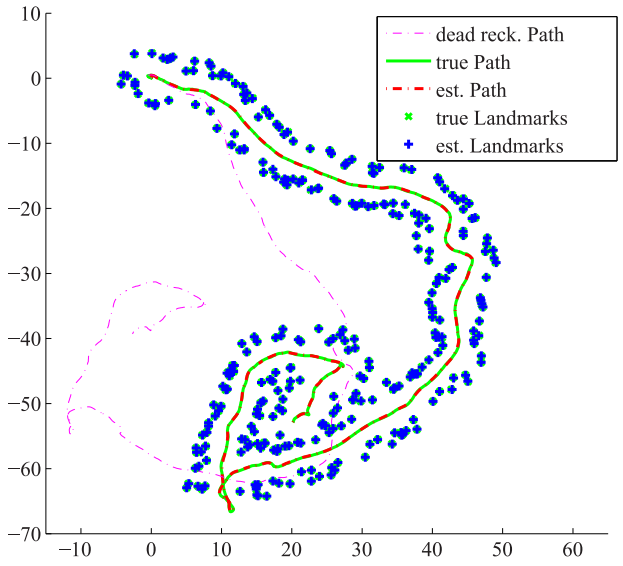


Fig. 4. Synthetic dataset: Ground truth, dead reckoning path, and the estimates are shown. State and landmark estimates obtained from BTGP approach are very close to ground truth.

algorithm. This allows us to reduce the total number of estimated states, while still using all of the measurements, including those that involve interpolated states. By only estimating a small fraction of the states along the trajectory, we realize a large speedup relative to a naive application of the Bayes tree (see Section 5). This is an advantage of continuous-time GP-based methods compared to discrete-time methods like iSAM 2.0.

To use Gaussian process interpolation within our incremental algorithms, we add a new type of factors that correspond to missing states (states to be interpolated). We start by observing that, from Eq. (2), the *measurement* factor $f_{\mathcal{M}}(\cdot)$ derived from the measurement $h_k(\cdot)$ is:

$$f_{\mathcal{M}}(\boldsymbol{\theta}_k) \propto \exp\left\{-\frac{1}{2}\|\boldsymbol{h}_k(\bar{\boldsymbol{\theta}}_k + \delta\boldsymbol{\theta}_k) - \boldsymbol{y}_k\|^2_{\boldsymbol{R}_k}\right\}. \tag{30}$$

Without loss of generality, we assume that $\boldsymbol{x}(\tau)$ is the set of variables related to the measurement and the factor, with $t_{i-1} < \tau < t_i$, so $f^m(\cdot)$ is a unitary factor of $\boldsymbol{x}(\tau)$

$$f_{\mathcal{M}}(\boldsymbol{x}(\tau)) \propto \exp\left\{-\frac{1}{2}\|\boldsymbol{h}_k\left(\bar{\boldsymbol{x}}(\tau) + \delta\boldsymbol{x}(\tau)\right) - \boldsymbol{y}_k\|^2_{\boldsymbol{R}_k}\right\}. \tag{31}$$

After linearizing the factor through Eq. (9), we arrive at:

$$f_{\mathcal{L}}(\boldsymbol{x}(\tau)) \propto \exp\left\{-\frac{1}{2}\|\boldsymbol{h}_k\left(\bar{\boldsymbol{x}}(\tau)\right) + \boldsymbol{H}_k\delta\boldsymbol{x}(\tau) - \boldsymbol{y}_k\|^2_{\boldsymbol{R}_k}\right\}. \tag{32}$$

If $\boldsymbol{x}(\tau)$ is *missing*, then this factor cannot be added to the factor graph directly, because a missing state implies that it should not be estimated explicitly. Instead of creating a new state, we interpolate the state and utilize the linearized measurement function after *interpolation* (Eq. (16)) :

$$f_{\mathcal{I}}(\boldsymbol{x}) \propto \exp\left\{-\frac{1}{2}\|\boldsymbol{h}_k\left(\bar{\boldsymbol{x}}(\tau)\right) + \boldsymbol{H}_k\mathcal{K}(\tau)\mathcal{K}^{-1}\delta\boldsymbol{x} - \boldsymbol{y}_k\|^2_{\boldsymbol{R}_k}\right\}. \tag{33}$$

We apply the interpolation equations for the *sparse* GP (Eqs. (20) and (22)), so that the factor becomes a function of the two nearby

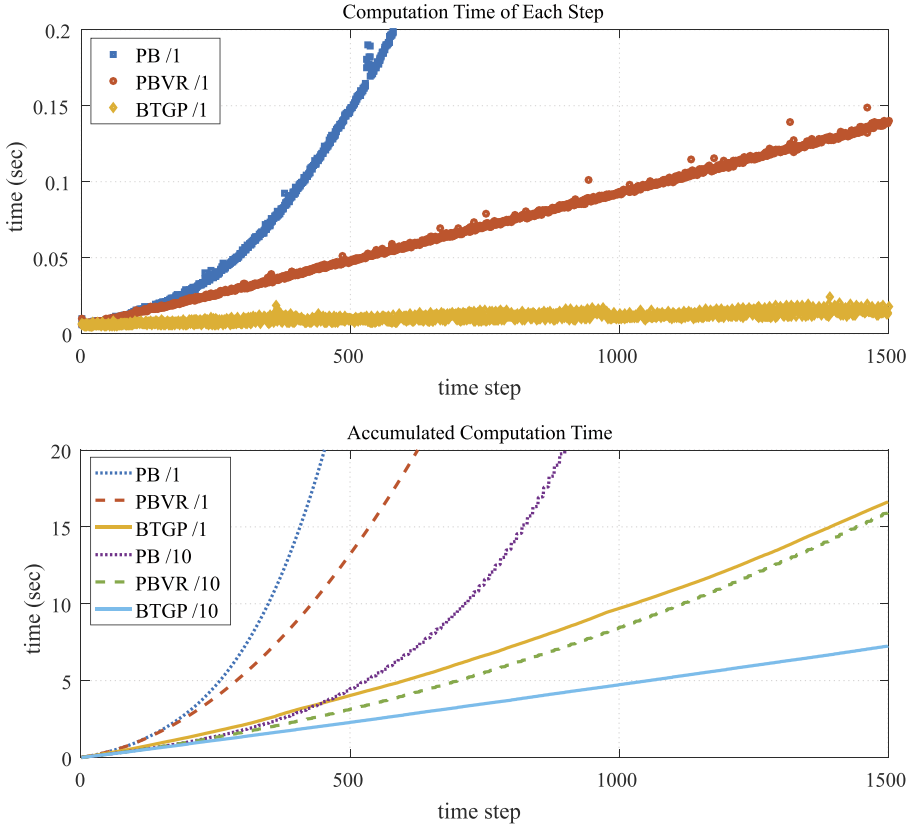


Fig. 5. Synthetic dataset: Comparison of the computation time of three approaches PB, PBVR, and BTGP. The modifiers /1 and /10 indicate frequency of estimate updates—the number of range measurements between updates. For example: *BTGP/1* updates the estimate after 1 new range measurement using BTGP. Likewise *BTGP/10* updates the estimate after 10 new range measurements using BTGP. Due to the large number of landmarks, 298, variable reordering dramatically improves the performance.

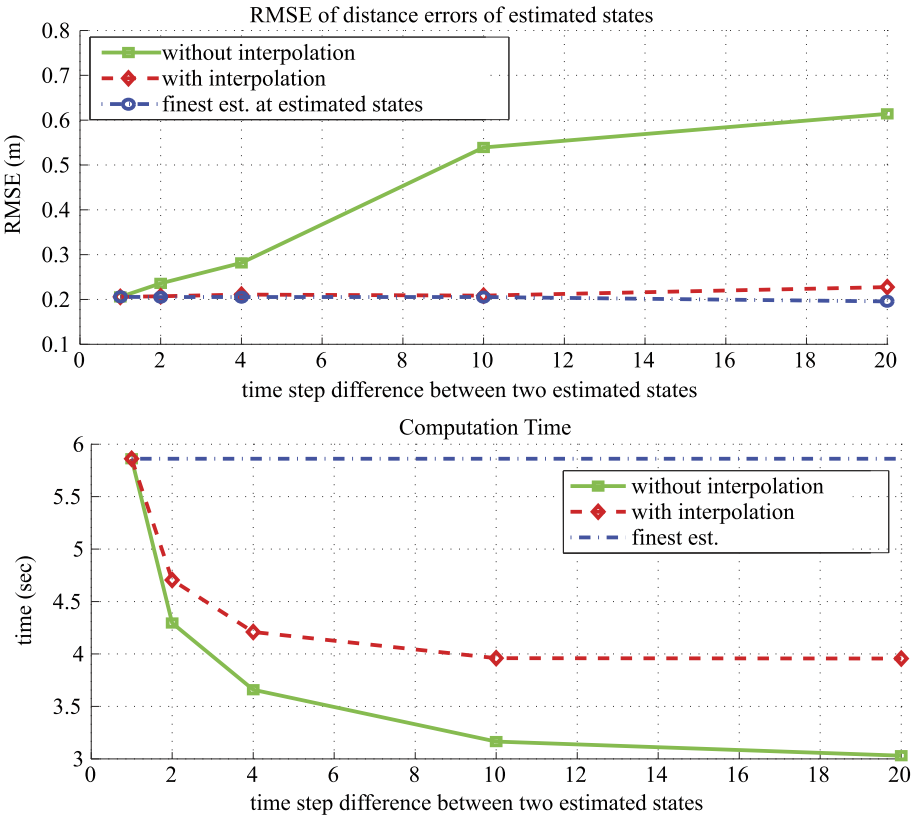


Fig. 6. Synthetic dataset: Trade-off between computation time and accuracy if BTGP makes use of interpolation. The y-axis measures the RMSE of distance errors of the estimated trajectory states and total computation time with increasing amounts of interpolation. The x-axis measures the time step difference between two estimated (non-interpolated) states. The results indicate that interpolating ∼90% of the states (i.e. estimating only ∼10% of the states) while running BTGP can result in a 33% reduction in computation time over iSAM 2.0 without sacrificing accuracy.

states (in contrast to the missing state):

$$f_S(\boldsymbol{x}_{i-1}, \boldsymbol{x}_i) \propto \exp\left\{-\frac{1}{2}\left\|\boldsymbol{h}_k(\bar{\boldsymbol{x}}(\tau))+\boldsymbol{H}_k\Big(\boldsymbol{\Lambda}(\tau)\delta\boldsymbol{x}_{i-1}+\boldsymbol{\Psi}(\tau)\delta\boldsymbol{x}_i\Big)-\boldsymbol{y}_k\right\|^2_{\boldsymbol{R}_k}\right\} \quad (34)$$

where $\bar{\boldsymbol{x}}(\boldsymbol{\tau})$ is specified in Eq. (22). The effect of interpolation and without interpolation on the factor graph is presented in Fig. 3.

A factor graph augmented with the factors associated with measurements at missing states has several advantages: (1) We can avoid estimating a missing state at time t explicitly, but still make use of a measurement at time t. This allows our algorithm to naturally handle asynchronous measurements. (2) We can also reduce the size of the Bayes tree and the associated matrices by skipping states, which results in a reduction of computation time. Empirically, we show in Sections 5.1 and 5.2 that skipping large numbers of states can reduce computation time by almost 70% with only a small reduction in accuracy. For example, in Section 5.2, we show that for the Autonomous Lawnmower dataset, interpolating 4 missing states instead of directly estimating them reduces computation time by 68% with a 20% increase in an already small RMSE.

Interpolation enables extreme flexibility in how measurements can be incorporated into the trajectory estimation problem. When a new measurement arrives, it can be thrown away, it can be directly incorporated as a new state, or a factor can be added corresponding to a missing/interpolated state. Even after the decision has been made, it can be changed via edits to the Bayes tree. The incurred expense is dependent on the tree structure. Different strategies can be applied based on the current uncertainty in estimation (Eq. (13)), computing resources available, and required estimation accuracy. Research on designing strategies that balance the trade-off for specific applications are left for future work.

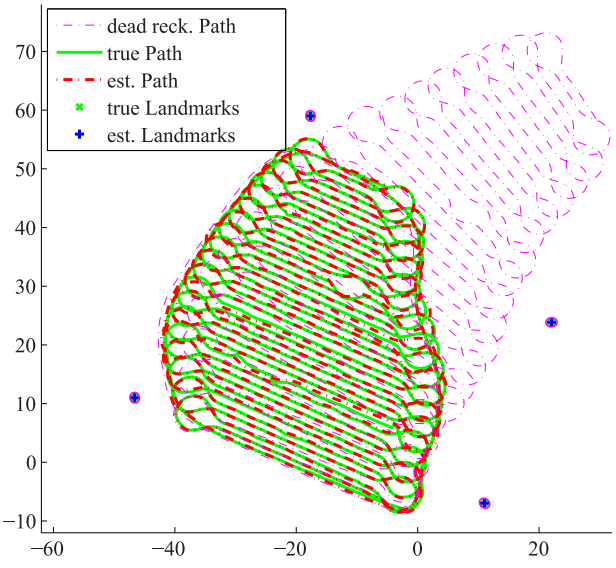


Fig. 7. The Autonomous Lawnmower dataset: Ground truth, dead reckoning path and estimates are shown. The range measurements are sparse, noisy, and asynchronous. Ground truth and the estimates of path and landmarks obtained from BTGP are very close.

The full incremental algorithm is described in Algorithm 2. In particular, when a measurement related to a missing state is received, the variables necessary to interpolate the state are

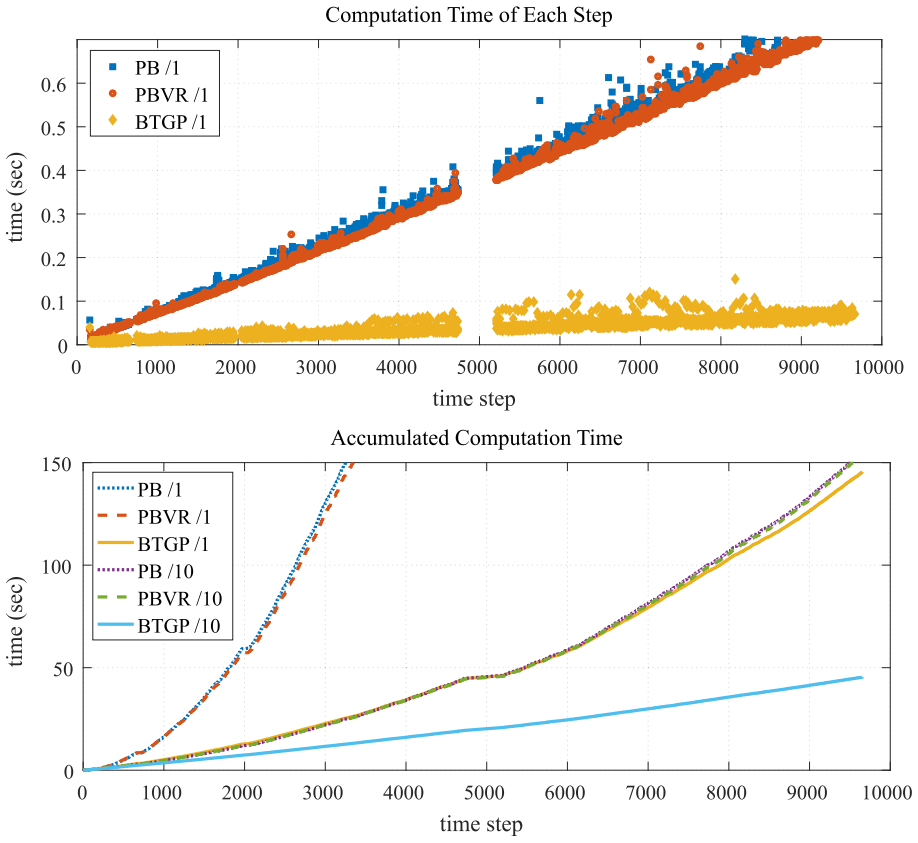


Fig. 8. Autonomous Lawnmower dataset: Comparison of the computation time of PB, PBVR, and BTGP. As in Fig. 5, /1 and /10 are modifiers—the number of range measurements between updates, and no interpolation is used by BTGP. The 'gap' in the upper graph is due to a long stretch around timestep 5000 with no range measurements. Due to the low number of landmarks, variable reordering does not help The incremental BTGP approach dramatically reduces computation time.

Algorithm 2 Incremental Sparse GP Regression visa the Bayes tree with Gaussian Process Priors (BTGP)

Set the sets of *affected* variables, variables involved in *new factors*, and *relinearized* variables to empty sets, $\boldsymbol{\theta}_{aff} = \boldsymbol{\theta}_{nf} = \boldsymbol{\theta}_{rl} = \varnothing$.
while collecting data **do**

1. Collect measurements, store as new factors. Set $\boldsymbol{\theta}_{nf}$ to the set of variables involved in the *new factors*. If $\boldsymbol{x}(\tau) \in \boldsymbol{\theta}_{nf}$ is a missing state, replace it by nearby states (Eq. (22)); If $\boldsymbol{x}(\tau) \in \boldsymbol{\theta}_{nf}$ is a new state to estimate, a GP prior (Eq. (26)) is stored, and $\boldsymbol{\theta}_{nf} = \boldsymbol{\theta}_{nf} \cup \boldsymbol{x}(\tau)$.
2. For all $\theta_i \in \boldsymbol{\theta}_{aff} = \boldsymbol{\theta}_{rl} \cup \boldsymbol{\theta}_{nf}$, remove the corresponding cliques and ascendants up to the root of the Bayes tree.
3. Relinearize the factors required to create the removed part using interpolation if missing states are involved (Eq. (23)).
4. Add the cached marginal factors from the orphaned sub-trees of the removed cliques and create a factor graph.
5. Eliminate the factor graph by a new variable ordering, create a Bayes tree, and attach back orphaned sub-trees.
6. Partially update estimate from the root to leaves, and stop when updates to variables are below a threshold.
7. Collect variables, for which the difference between the current estimate and the previous linearization point is above a threshold, into $\boldsymbol{\theta}_{rl}$.

end while

identified. Since the sparse GP has a LTV SDE prior, each interpolated state is only a function of two nearby states (see Eq. (22)). These nearby states are therefore included into the set of variables $\boldsymbol{\theta}_{nf}$ related to the new factor (step 1). In the case that the GP relies on a different kernel matrix, the corresponding states used for interpolation can be determined from Eq. (14). Linearization of factors that involve missing states (step 3) is performed by incorporating state interpolation via Eq. (16).

5. Experimental results

We evaluate the performance of our incremental sparse GP regression algorithm for solving the STEAM problem on synthetic and real-data experiments and compare our approach to the state-of-the-art. In particular, we evaluate how variable reordering can dramatically speed up the batch solution to the sparse GP regression problem, and how, by utilizing the Bayes tree and interpolation for incremental updates, our algorithm can yield even greater gains in the online trajectory estimation scenario. We compare:

- **PB**: Periodic batch (described in Section 2). This is the state-of-the-art algorithm presented in Barfoot et al. [10] (XL variable ordering), which is periodically executed as data is received.
- **PBVR**: Periodic batch with variable reordering (described in Section 3). Variable reordering is applied to achieve efficient matrix factorization.
- **BTGP**: The proposed approach—Bayes tree with Gaussian process prior factors (described in Section 4).

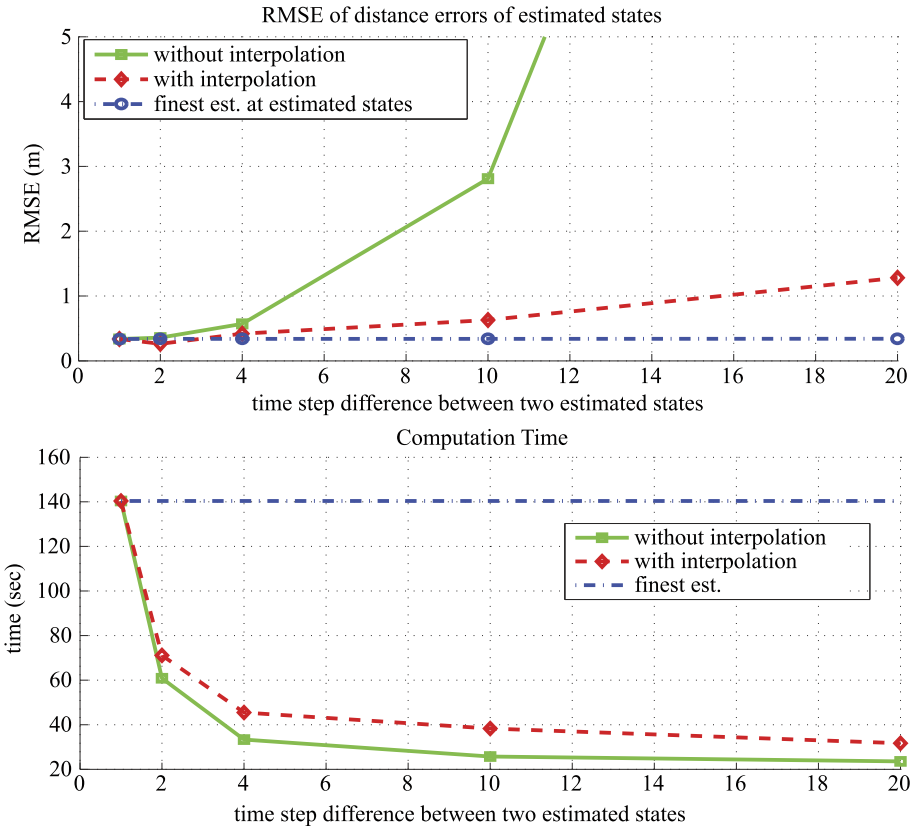


Fig. 9. Autonomous Lawnmower dataset: Trade-off between computation time and accuracy if BTGP makes use of interpolation. The *y*-axis measures the RMSE of distance errors and total computation time with increasing amounts of interpolation. The *x*-axis measures the time step difference between two estimated (non-interpolated) states. The results indicate that interpolating ∼80% of the states within BTGP results in only an 8 cm increase in RSME while reducing the overall computation time by 68% over iSAM 2.0.

If the GP is only used to estimate the state at measurement times, the proposed approach offers little beyond a reinterpretation of the standard discrete-time iSAM 2.0 algorithm. Therefore, we also compare our GP-based algorithm, which leverages interpolation, to the standard Bayes tree approach used in iSAM 2.0. We show that by interpolating large fractions of the trajectory during optimization, the GP allows us to realize significant performance gains over iSAM 2.0 with minimal loss in accuracy. For these experiments we compare:

- **without interpolation**: BTGP without interpolation at a series of lower temporal resolutions. The lower the resolution, the fewer the states to be estimated. Without interpolation BTGP is algorithmically identical to iSAM 2.0 with coarse discretization of the trajectory. Measurements between two estimated states are simply ignored.
- **with interpolation**: BTGP with interpolation at a series of lower resolutions. In contrast to the above case, measurements between estimated states are fully utilized by interpolating missing states at measurement times (described in Section 4.2).
- **finest estimate**: The baseline. BTGP at the finest resolution, estimating all states at measurement times. When measurements are synchronous with evenly-spaced waypoints and no interpolation is used, BTGP is identical to iSAM 2.0 applied to the full dataset with all measurements.

All algorithms are implemented with the same C++ library, GTSAM 3.2,[3] to make the comparison fair and meaningful.

[3] https://collab.cc.gatech.edu/borg/gtsam/.

Table 2
Summary of the experimental datasets.

	# time steps	# odo. m.	# landmark m.	# landmarks	Travel dist. (km)
Synthetic	1500	1500	1500	298	0.2
Auto. Mower	9658	9658	3529	4	1.9
Victoria Park	6969	6969	3640	151	3.5

Evaluation is performed on three datasets summarized in Table 2. We first evaluate performance in a synthetic dataset (Section 5.1), analyzing estimation errors with respect to ground truth data. Results using real-world datasets are then presented in Sections 5.2 and 5.3.

5.1. Synthetic SLAM exploration task

This dataset consists of an exploration task with 1500 time steps. Each time step contains a trajectory state $\boldsymbol{x}_i = [\boldsymbol{p}_i^\intercal \quad \dot{\boldsymbol{p}}_i^\intercal]^\intercal$, $\boldsymbol{p}_i = [x_i \quad y_i \quad \theta_i]^\intercal$, an odometry measurement, and a range measurement related to a nearby landmark. The total number of landmarks is 298. The trajectory is randomly sampled from a Gaussian process generated from white noise acceleration $\ddot{\boldsymbol{p}}(t) = \boldsymbol{w}(t)$, i.e. constant velocity, and with zero mean.

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{F}\boldsymbol{w}(t) \tag{35}$$

$$\boldsymbol{x}(t) = \begin{bmatrix} \boldsymbol{p}(t) \\ \dot{\boldsymbol{p}}(t) \end{bmatrix}, \quad \boldsymbol{p}(t) = \begin{bmatrix} x(t) \\ y(t) \\ \theta(t) \end{bmatrix}, \quad \boldsymbol{A} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{I} \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix},$$

$$\boldsymbol{F} = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{I} \end{bmatrix}, \quad \boldsymbol{w}(t) \sim \mathcal{GP}(\boldsymbol{0}, \boldsymbol{Q}_c \delta(t - t')). \tag{36}$$

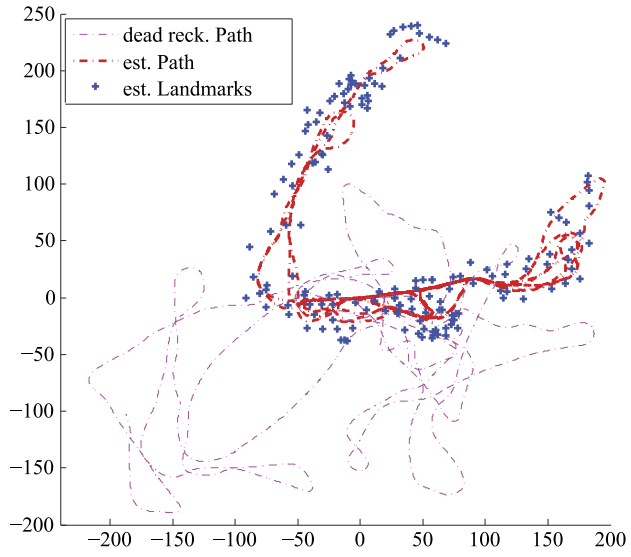


Fig. 10. Victoria Park dataset: Dead reckoning and estimated path obtained from BTGP approach.

Note that velocity $\dot{\boldsymbol{p}}(t)$ must to be included in trajectory state to represent the motion in LTV SDE form [10]. This Gaussian process representation of trajectory is also applied the other two datasets. The odometry and range measurements with Gaussian noise are specified in Eqs. (37) and (38) respectively.

$$\boldsymbol{y}_{\mathcal{O}}(\boldsymbol{p}_i) = \begin{bmatrix} \dot{x}_i \cos\theta_i + \dot{y}_i \sin\theta_i \\ \dot{\theta}_i \end{bmatrix} + \boldsymbol{n}_{\mathcal{O}} \tag{37}$$

$$y_{\mathcal{R}}(\boldsymbol{p}_i, \boldsymbol{\ell}_j) = \left\| \begin{bmatrix} x_i & y_i \end{bmatrix}^{\mathrm{T}} - \boldsymbol{\ell}_j \right\|_2 + n_{\mathcal{R}} \tag{38}$$

where odometry measurements $\boldsymbol{y}_{\mathcal{O}}(\cdot)$ consists of the robot-oriented velocity and heading angle velocity with Gaussian noise, and range measurements $y_{\mathcal{R}}(\cdot)$ is the distance between the robot and a specific landmark $\boldsymbol{\ell}_j$ at t_i with Gaussian noise. The estimation results are shown in Fig. 4).

We compare the computation time of the three approaches (PB, PBVR and BTGP) in Fig. 5. The incremental Gaussian process regression (BTGP) offers significant improvements in computation time compared to the batch approaches (PBVR and PB). In Fig. 6, we demonstrate that BTGP can further increase speed over a naive application of the Bayes tree (e.g. iSAM 2.0) without sacrificing much accuracy by leveraging interpolation. To illustrate the trade-off between the accuracy and time efficiency due to interpolation, we plot RMSE of distance errors and the total computation time by varying the time step difference (the rate of interpolation) between estimated states. To further speed up our method while maintaining accuracy, it may be possible to dynamically specify the number of interpolated states between two estimated states, fully exploiting the flexibility provided by interpolation. This is left for future work.

5.2. The Autonomous Lawnmower

The second experiment evaluates our approach on real data from a freely available range-only SLAM dataset collected from an autonomous lawn-mowing robot [22]. The "Plaza" dataset consists of odometer data and range data to stationary landmarks collected via time-of-flight radio nodes. (Additional details on the experimental setup can be found in [22].) Ground truth paths are

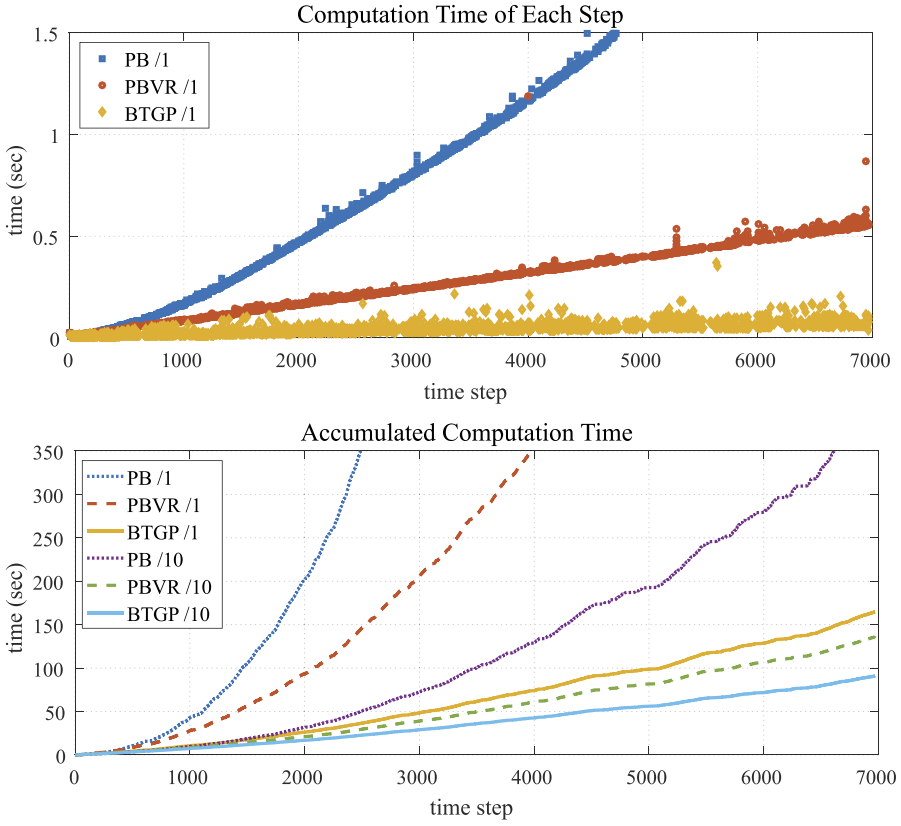


Fig. 11. Victoria Park dataset: Comparison of the computation time of three approaches PB, PBVR, and BTGP. As in Figs. 5 and 8, the modifiers /1 and /10 indicate frequency of state updates. Since many landmarks are involved, PBVR dramatically improves performance, compared to PB. The incremental BTGP algorithm improves performance even further. Unlike in previous datasets, we did not evaluate the trade-off between interpolation and accuracy for Victoria Park, since we do not have access to ground truth and cannot evaluate the effect on accuracy. However, like previous datasets, interpolation can greatly increase the speed of BTGP.

computed from GPS readings and have 2 cm accuracy according to [22]. The environment, including the locations of the landmarks and the ground truth paths, are shown in Fig. 7. The robot traveled 1.9 km, occupied 9658 poses, and received 3529 range measurements, while following a typical path generated during mowing. The dataset has sparse range measurements, but contains odometry measurements at each time step. The results of incremental BTGP are shown in Fig. 7 and demonstrate that we are able to estimate the robot's trajectory and map with a very high degree of accuracy.

As in Section 5.1, performance of three approaches—periodic batch relinearization (PB), periodic batch relinearization with variable reordering (PBVR) and incremental Bayes tree (BTGP) are compared in Fig. 8. In this dataset, the number of landmarks is 4, which is extremely small relative to the number of trajectory states, so there is no performance gain from reordering. However, the Bayes tree-based approach dramatically outperforms the other two approaches. As the problem size increases, there is negligible increase in computation time, even for close to 10,000 trajectory states.

In Fig. 9, the results of interpolation at different levels of resolutions are presented, which indicate a significant reduction in computation time can be achieved with minor sacrifice in accuracy.

5.3. Victoria park

The third experiment evaluates our approach on the Victoria Park dataset [23], which consists of range-bearing measurements to landmarks, and speed and steering odometry measurements (see Fig. 10). The data was collected from a vehicle equipped with a laser sensor driving through the Sydney's Victoria Park. The environment contains a high number of trees as landmarks. The vehicle traveled$\sim$3.5 km in 26 min. After repeated measurements, taken when the vehicle is stationary, are dropped, the dataset consists of 6969 time steps and 3640 range-bearing measurements relative to 151 landmarks. The bearing measurement is specified in Eq. (39), as the relative angle from vehicle heading to the landmark direction with Gaussian noise $n_{\mathcal{B}}$:

$$y_{\mathcal{B}}(\boldsymbol{p}_i, \boldsymbol{\ell}_j) = \operatorname{atan2}\left(y_{\ell j} - y_i,\ x_{\ell j} - x_i\right) - \theta_i + n_{\mathcal{B}} \tag{39}$$

where $\boldsymbol{\ell}_j = [x_{\ell j} \quad y_{\ell j}]^{\mathrm{T}}$ is the location of landmark j, and $\boldsymbol{p}_i$ is defined the same as in Section 5.1. The results, shown in Fig. 11, further demonstrate the advantages of BTGP. Variable reordering drastically reduces computation time when used within batch optimization (PBVR), and even further in the incremental algorithm (BTGP).

6. Conclusion

We have introduced an incremental sparse Gaussian process regression algorithm for computing the solution to the continuous-time simultaneous trajectory estimation and mapping (STEAM) problem. The proposed algorithm elegantly combines the benefits of Gaussian process-based approaches to STEAM while simultaneously employing state-of-the-art innovations from incremental discrete-time algorithms for smoothing and mapping. Our empirical results show that by parameterizing trajectories with a small number of states and utilizing Gaussian process interpolation, our algorithm can realize large gains in speed over iSAM 2.0 with very little loss in accuracy (e.g. reducing computation time by 68% while increasing RMSE by only 25% (8 cm) on the Autonomous Lawnmower Dataset) .

References

[1] Sebastian Thrun, Wolfram Burgard, Dieter Fox, Probabilistic Robotics (Intelligent Robotics and Autonomous Agents), The MIT Press, ISBN: 0262201623, 2005.

[2] Hugh Durrant-Whyte, Tim Bailey, Simultaneous localisation and mapping (SLAM): Part I the essential algorithms, IEEE Robot. Autom. Mag. 2 (2006) 2006.

[3] T. Bailey, H. Durrant-Whyte, Simultaneous localisation and mapping (SLAM): Part II state of the art, Robot. Autom. Mag. 13 (3) (2006) 108–117.

[4] Frank Dellaert, Michael Kaess, Square root SAM: Simultaneous localization and mapping via square root information smoothing, Int. J. Robot. Res. 25 (2006) 2006.

[5] M. Kaess, A. Ranganathan, F. Dellaert, iSAM: Incremental Smoothing and Mapping, IEEE Trans. Robot. (ISSN: 1552-3098) 24 (6) (2008) 1365–1378. http://dx.doi.org/10.1109/TRO.2008.2006706.

[6] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J.J. Leonard, F. Dellaert, iSAM2: Incremental Smoothing and Mapping Using the Bayes Tree, Int. Robot. Res. 31 (2) (2012) 217–236.

[7] Michael Kaess, Viorela Ila, Richard Roberts, Frank Dellaert, The Bayes tree: An algorithmic foundation for probabilistic robot mapping, in: Algorithmic Foundations of Robotics IX, Springer, 2011, pp. 157–173.

[8] Michael Montemerlo, Sebastian Thrun, Daphne Koller, Ben Wegbreit, FastSLAM: A factored solution to the simultaneous localization and mapping problem, in: Proceedings of the AAAI National Conference on Artificial Intelligence, AAAI, 2002, pp. 593–598.

[9] Byron Boots, Geoffrey J. Gordon, A spectral learning approach to range-only SLAM, in: Proceedings of the 30th International Conference on Machine Learning, ICML, 2013.

[10] Tim Barfoot, Chi Hay Tong, Simo Sarkka, Batch continuous-time trajectory estimation as exactly sparse Gaussian process regression, in: Proceedings of Robotics: Science and Systems, Berkeley, USA, July 2014, 2014.

[11] Chi Hay Tong, Paul Furgale, Timothy D Barfoot, Gaussian process Gauss–Newton for non-parametric simultaneous localization and mapping, Int. J. Robot. Res. 32 (5) (2013) 507–525.

[12] X. Yan, V. Indelman, B. Boots, Incremental sparse GP regression for continuous-time trajectory estimation and mapping, in: NIPS Workshop on Autonomously Learning Robots, December, 2014.

[13] X. Yan, V. Indelman, B. Boots, Incremental sparse GP regression for continuous-time trajectory estimation and mapping, in: International Symposium on Robotics Research, ISRR, September, 2015.

[14] C.E. Rasmussen, C.K.I. Williams, Gaussian Processes for Machine Learning, 2006.

[15] J.E. Dennis, Jr., Robert B. Schnabel, Numerical Methods for Unconstrained Optimization and Nonlinear Equations (Classics in Applied Mathematics, 16), Soc for Industrial & Applied Math, ISBN: 0898713641, 1996.

[16] Gene H. Golub, Charles F. Van Loan, Matrix Computations, third ed., Johns Hopkins University Press, Baltimore, MD, USA, ISBN: 0-8018-5414-8, 1996.

[17] A. Ranganathan, Ming-Hsuan Yang, J. Ho, Online sparse Gaussian process regression and its applications, IEEE Trans. Image Process. (ISSN: 1057-7149) 20 (2) (2011) 391–404. http://dx.doi.org/10.1109/TIP.2010.2066984.

[18] M. Yannakakis, Computing the minimum fill-in is NP-complete, SIAM J. Algebr. Discrete Methods 2 (1) (1981) 77–79. http://dx.doi.org/10.1137/0602010.

[19] Timothy A. Davis, John R. Gilbert, Stefan I. Larimore, Esmond G. Ng, Algorithm 836: COLAMD, a column approximate minimum degree ordering algorithm, ACM Trans. Math. Softw. (ISSN: 0098-3500) 30 (3) (2004) 377–380.

[20] Patrick R Amestoy, Timothy A Davis, Iain S Duff, Algorithm 837: AMD, an approximate minimum degree ordering algorithm, ACM Trans. Math. Softw. 30 (3) (2004) 381–388.

[21] Feng Lu, Evangelos Milios, Globally consistent range scan alignment for environment mapping, Auton. Robots 4 (4) (1997) 333–349.

[22] Joseph Djugash, Geolocation with range: robustness, efficiency and scalability (Ph.D. thesis), Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, 2010.

[23] J.E. Guivant, E.M. Nebot, Optimization of the simultaneous localization and map-building algorithm for real-time implementation, IEEE Trans. Robot. Autom. (ISSN: 1042-296X) 17 (3) (2001) 242–257. http://dx.doi.org/10.1109/70.938382.

Xinyan Yan is a first-year Ph.D. student in Robotics in the School of Interactive Computing at Georgia Institute of Technology. He obtained his M.S. degree in Computer Science with concentration in Computational Perception and Robotics in 2014 from Georgia Institute of Technology. He also holds a B.S. degree in Information Security Engineering, awarded by Shanghai Jiao Tong University. His research interests include machine learning and motion planning.

Vadim Indelman obtained his Ph.D. degree in Aerospace Engineering at the Technion — Israel Institute of Technology in 2011. He also holds B.A. and B.Sc. degrees in Computer Science and Aerospace Engineering, respectively, both awarded by the Technion in 2002. Between 2012 and 2014, Dr. Indelman was a postdoctoral fellow in the Institute of Robotics and Intelligent Machines (IRIM) at the Georgia Institute of Technology. In July 2014 he assumed his present position of an Assistant Professor in Aerospace Engineering at the Technion. Dr. Indelman is also a member of the Technion Autonomous Systems Program (TASP). His research interests include autonomous navigation and mapping, distributed perception and information fusion, belief-space planning and active sensing, vision-aided navigation (VAN) and simultaneous localization and mapping (SLAM).

Byron Boots is an Assistant Professor in the School of Interactive Computing and the College of Computing at the Georgia Institute of Technology. He directs the Georgia Tech Robot Learning Lab, which is part of the Institute for Robotics and Intelligent Machines (IRIM). From 2012 to 2014, Dr. Boots was a postdoctoral fellow in the department of Computer Science and Engineering at the University of Washington. He received his Ph.D. in Machine Learning from Carnegie Mellon University in 2012. His research interests include statistical machine learning, artificial intelligence, and robotics.