Incremental Sparse GP Regression for Continuous-time Trajectory Estimation & Mapping

Xinyan Yan, Vadim Indelman and Byron Boots

Abstract Recent work on simultaneous trajectory estimation and mapping (STEAM) for mobile robots has found success by representing the trajectory as a Gaussian process. Gaussian processes 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 this approach is that STEAM is formulated as a *batch* estimation problem. In this paper we provide the critical extensions necessary to transform the existing batch algorithm 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 [14, 7, 1]. 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 up-

Xinyan Yan and Byron Boots
College of Computing, Georgia Institute of Technology, Atlanta, GA, US
e-mail: {xinyan.yan, bboots}@cc.gatech.edu

Vadim Indelman
Department of Aerospace Engineering, Technion - Israel Institute of Technology, Haifa, Israel
e-mail: vadim.indelman@technion.ac.il

date an estimate of its trajectory online is a significant drawback. In response to this weakness, Kaess et al. [9] 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 [10], which employs an efficient data structure called the *Bayes tree* [11] 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 [12, 14, 7, 1, 4, 10, 3]. 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* $\mathbf{x}(t)$, that maps any time t to a robot state. The problem of estimating this function along with landmark locations has been dubbed *simultaneous trajectory estimation and mapping* (STEAM) [2].

Tong et al. [15] 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 resolving the problem. In order to combat the computational burden, Tong et al.'s approach was extended in Barfoot et al. [2] 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 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.

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. [15] and Barfoot et al. [2], we represent robot trajectories as functions of time t sampled from a Gaussian process:

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

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

We next define a finite set of measurements:

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

The measurement $\mathbf{y}_i$ can be any linear or nonlinear functions of a set of related variables $\boldsymbol{\theta}_i$ plus some Gaussian noise $\mathbf{n}_i$. The related variables for a range measurement are the robot state at the corresponding measurement time $\mathbf{x}(t_i)$ 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 [13]. So the robot states at measurement times are normally distributed with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\mathcal{K}}$.

$$\begin{aligned} &\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\mathcal{K}}), \quad \mathbf{x} = [\,\mathbf{x}(t_1)^\intercal \ldots \mathbf{x}(t_M)^\intercal\,]^\intercal \\ &\boldsymbol{\mu} = [\,\boldsymbol{\mu}(t_1)^\intercal \ldots \boldsymbol{\mu}(t_M)^\intercal\,]^\intercal, \quad \boldsymbol{\mathcal{K}}_{ij} = \boldsymbol{\mathcal{K}}(t_i, t_j) \end{aligned} \tag{3}$$

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 $\mathbf{d}$ and covariance $\mathbf{W}$ (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).

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

$$\boldsymbol{\theta} \sim \mathcal{N}(\boldsymbol{\eta}, \boldsymbol{\mathcal{P}}), \quad \boldsymbol{\eta} = [\,\boldsymbol{\mu}^\intercal \; \mathbf{d}^\intercal\,]^\intercal, \quad \boldsymbol{\mathcal{P}} = \text{diag}(\boldsymbol{\mathcal{K}}, \mathbf{W}) \tag{5}$$

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:

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

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

$$\mathbf{h}(\boldsymbol{\theta}) = [\,\mathbf{h}_1(\boldsymbol{\theta}_1)^\intercal\ \mathbf{h}_2(\boldsymbol{\theta}_2)^\intercal\ \ldots\ \mathbf{h}_N(\boldsymbol{\theta}_N)^\intercal\,]^\intercal,\ \ \mathbf{R} = \operatorname{diag}(\mathbf{R}_1, \mathbf{R}_2, \ldots, \mathbf{R}_N) \tag{7}$$

Because both covariance matrices $\boldsymbol{\mathcal{P}}$ and $\mathbf{R}$ are positive definite, the objective in Eq. 6 corresponds to a least squares problem. Consequently, if some of the measurement functions $\mathbf{h}_i(\cdot)$ are nonlinear, this becomes a nonlinear least squares problem, in which case iterative methods including Gauss-Newton and Levenberg-Marquardt [5] 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:

$$\mathbf{h}_i\left(\bar{\boldsymbol{\theta}}_i + \delta\boldsymbol{\theta}_i\right) \approx \mathbf{h}_i(\bar{\boldsymbol{\theta}}_i) + \left.\frac{\partial \mathbf{h}_i}{\partial \boldsymbol{\theta}_i}\right|_{\bar{\boldsymbol{\theta}}_i} \delta\boldsymbol{\theta}_i \tag{8}$$

Combining Eq. 8 with Eq. 6, the optimal increment $\delta\boldsymbol{\theta}^*$ is:

$$\delta\boldsymbol{\theta}^* = \underset{\delta\boldsymbol{\theta}}{\operatorname{argmin}}\ \left(\|\bar{\boldsymbol{\theta}}+\delta\boldsymbol{\theta}-\boldsymbol{\eta}\|^2_{\boldsymbol{\mathcal{P}}} + \|\mathbf{h}(\bar{\boldsymbol{\theta}})+\mathbf{H}\delta\boldsymbol{\theta}-\mathbf{y}\|^2_{\mathbf{R}}\right) \tag{9}$$

$$\mathbf{H} = \operatorname{diag}(\mathbf{H}_1, \mathbf{H}_2, \ldots, \mathbf{H}_N), \qquad \mathbf{H}_i = \left.\frac{\partial \mathbf{h}_i}{\partial \boldsymbol{\theta}_i}\right|_{\bar{\boldsymbol{\theta}}_i} \tag{10}$$

where $\mathbf{H}$ is the measurement Jacobian matrix. To solve the linear least squares problem in Eq. 9, we take the derivative with respect to $\delta\boldsymbol{\theta}$, and set it to zero, which gives us $\delta\boldsymbol{\theta}^*$ embedded in a set of linear equations

$$\underbrace{(\boldsymbol{\mathcal{P}}^{-1}+\mathbf{H}^\intercal\mathbf{R}^{-1}\mathbf{H})}_{\boldsymbol{\mathcal{I}}}\,\delta\boldsymbol{\theta}^* = \underbrace{\boldsymbol{\mathcal{P}}^{-1}(\boldsymbol{\eta}-\bar{\boldsymbol{\theta}})+\mathbf{H}^\intercal\mathbf{R}^{-1}(\mathbf{y}-\bar{\mathbf{h}})}_{\mathbf{b}} \tag{11}$$

with covariance $\operatorname{cov}(\delta\boldsymbol{\theta}^*, \delta\boldsymbol{\theta}^*) = \boldsymbol{\mathcal{I}}^{-1}$.

The positive definite matrix $\boldsymbol{\mathcal{I}}$ is the *a posteriori* information matrix. To solve the linear equations for $\delta\boldsymbol{\theta}^*$, factorization-based methods can provide a fast, numerically stable solution. For example, $\delta\boldsymbol{\theta}^*$ can be found by first performing a Cholesky factorization $\boldsymbol{\mathcal{L}}\boldsymbol{\mathcal{L}}^\intercal = \boldsymbol{\mathcal{I}}$, and then solving 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 $\boldsymbol{\mathcal{I}} \in \mathbb{R}^{n\times n}$ [8]. 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 $O(n^3)$ [8]. Fortunately, we can guarantee sparsity for the STEAM problem (see Section 2.2).

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 [15]:

$$\bar{\mathbf{x}}(t) = \boldsymbol{\mu}(t) + \boldsymbol{\mathcal{K}}(t)\boldsymbol{\mathcal{K}}^{-1}(\bar{\mathbf{x}} - \boldsymbol{\mu}) \tag{12}$$

$$\bar{\mathbf{x}} = [\,\bar{\mathbf{x}}(t_1)^\intercal \ldots \bar{\mathbf{x}}(t_M)^\intercal\,]^\intercal, \quad \boldsymbol{\mathcal{K}}(t) = [\,\boldsymbol{\mathcal{K}}(t,t_1) \ldots \boldsymbol{\mathcal{K}}(t,t_M)\,]$$

By utilizing interpolation, we can reduce the number of robot trajectory states that we need to estimate in the optimization procedure [15]. For simplicity, assume $\boldsymbol{\theta}_i$, the set of the related variables of the ith measurement according to the model (Eq. 2), is $\mathbf{x}(\tau)$. Then, after interpolation, Eq. 8 becomes:

$$\begin{aligned}\mathbf{h}_i\left(\bar{\boldsymbol{\theta}}_i + \delta\boldsymbol{\theta}_i\right) &= \mathbf{h}_i\left(\bar{\mathbf{x}}(\tau) + \delta\mathbf{x}(\tau)\right) \\ &\approx \mathbf{h}_i(\bar{\mathbf{x}}(\tau)) + \frac{\partial \mathbf{h}_i}{\partial \mathbf{x}(\tau)} \cdot \left.\frac{\partial \mathbf{x}(\tau)}{\partial \mathbf{x}}\right|_{\bar{\mathbf{x}}} \delta\mathbf{x} \\ &= \mathbf{h}_i\left(\boldsymbol{\mu}(\tau) + \boldsymbol{\mathcal{K}}(\tau)\boldsymbol{\mathcal{K}}^{-1}(\bar{\mathbf{x}} - \boldsymbol{\mu})\right) + \mathbf{H}_i\boldsymbol{\mathcal{K}}(\tau)\boldsymbol{\mathcal{K}}^{-1}\delta\mathbf{x} \end{aligned} \tag{13}$$

By employing Eq. 13 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 4).

2.2 Sparse Gaussian Process Regression

The efficiency of the Gaussian process Gauss-Newton algorithm presented in Section 2 is 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. 11 [4]. Barfoot et al. suggest a kernel matrix with a sparse inverse that is well-suited to the simultaneous trajectory estimation and mapping problem [2]. 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:

$$\begin{aligned}\dot{\mathbf{x}}(t) &= \mathbf{A}(t)\mathbf{x}(t) + \mathbf{v}(t) + \mathbf{F}(t)\mathbf{w}(t), \\ \mathbf{w}(t) &\sim \mathcal{GP}(\mathbf{0},\, \mathbf{Q}_c\delta(t - t')) \quad t_0 < t, t'\end{aligned} \tag{14}$$

where $\mathbf{x}(t)$ is trajectory, $\mathbf{v}(t)$ is known exogenous input, $\mathbf{w}(t)$ is process noise, and $\mathbf{F}(t)$ is time-varying system matrix. The process noise $\mathbf{w}(t)$ is modeled by a Gaussian process, and $\delta(\cdot)$ is the *Dirac delta function*. (See [2] for details). We consider a specific case of this model in the experimental results in Section 4.1.

Because the mean function $\boldsymbol{\mu}(t)$ is an integral of the known exogenous input $\mathbf{v}(t)$, the assumption of zero $\mathbf{v}(t)$ leads to Gaussian process with zero mean $\boldsymbol{\mu}(t)$.

Assuming the GP is generated by Eq. 14, the measurements are landmark and odometry measurements, and the variables are ordered in XL ordering[1], the sparse information matrix becomes

$$\boldsymbol{\mathcal{I}} = \begin{bmatrix} \boldsymbol{\mathcal{I}}_{xx} & \boldsymbol{\mathcal{I}}_{x\ell} \\ \boldsymbol{\mathcal{I}}_{x\ell}^{\mathsf{T}} & \boldsymbol{\mathcal{I}}_{\ell\ell} \end{bmatrix} \tag{15}$$

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.

When the GP is generated by LTV SDE, $\boldsymbol{\mathcal{K}}(\tau)\boldsymbol{\mathcal{K}}^{-1}$ in Eq. 12 has a specific sparsity pattern — only two column blocks that correspond to trajectory states at t_{i-1} and t_i are nonzero ($t_{i-1} < \tau < t_i$) [2]:

$$\boldsymbol{\mathcal{K}}(\tau)\boldsymbol{\mathcal{K}}^{-1} = \begin{bmatrix} 0 \ldots 0 \; \boldsymbol{\Lambda}(\tau) \; \boldsymbol{\Psi}(\tau) \; 0 \ldots 0 \end{bmatrix} \tag{16}$$

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

$\boldsymbol{\Phi}(\tau, s)$ is the state transition matrix from s to τ. $\mathbf{Q}_\tau$ is the integral of $\mathbf{Q}_c$, the covariance of the process noise $\mathbf{w}(t)$ (Eq. 14):

$$\mathbf{Q}_\tau = \int_{t_{i-1}}^{\tau} \boldsymbol{\Phi}(\tau, s) \mathbf{F}(s) \mathbf{Q}_c \mathbf{F}(s)^{\mathsf{T}} \boldsymbol{\Phi}(\tau, s)^{\mathsf{T}} ds \tag{17}$$

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

Consequently, based on Eq. 12 and Eq. 16, $\bar{\mathbf{x}}(\tau)$ is an affine function of only two nearby states $\bar{\mathbf{x}}(t_{i-1})$ and $\bar{\mathbf{x}}(t_i)$ (the current estimate of the states at t_{i-1} and t_i):

$$\bar{\mathbf{x}}(\tau) = \boldsymbol{\mu}(\tau) + \begin{bmatrix} \boldsymbol{\Lambda}(\tau) & \boldsymbol{\Psi}(\tau) \end{bmatrix} \left(\begin{bmatrix} \bar{\mathbf{x}}(t_{i-1}) \\ \bar{\mathbf{x}}(t_i) \end{bmatrix} - \begin{bmatrix} \boldsymbol{\mu}(t_{i-1}) \\ \boldsymbol{\mu}(t_i) \end{bmatrix} \right), \; t_{i-1} < \tau < t_i \tag{18}$$

Thus, it only takes $O(1)$ time to query any $\bar{\mathbf{x}}(\tau)$ using Eq. 18. Moreover, because interpolation of a state is only determined by the two nearby states, measurement interpolation in Eq. 13 can be simplified to:

$$\begin{aligned} \mathbf{h}_k\left(\bar{\boldsymbol{\theta}}_k + \delta\boldsymbol{\theta}_k\right) &= \mathbf{h}_k\left(\bar{\mathbf{x}}(\tau) + \delta\mathbf{x}(\tau)\right) \\ &\approx \mathbf{h}_k(\bar{\mathbf{x}}(\tau)) + \frac{\partial \mathbf{h}_k}{\partial \mathbf{x}(\tau)} \cdot \left. \frac{\partial \mathbf{x}(\tau)}{\partial \mathbf{x}} \right|_{\bar{\mathbf{x}}} \delta\mathbf{x} \\ &= \mathbf{h}_k(\bar{\mathbf{x}}(\tau)) + \mathbf{H}_k \begin{bmatrix} \boldsymbol{\Lambda}(\tau) \, \boldsymbol{\Psi}(\tau) \end{bmatrix} \begin{bmatrix} \delta\mathbf{x}(t_{i-1}) \\ \delta\mathbf{x}(t_i) \end{bmatrix} \end{aligned} \tag{19}$$

with $\bar{\mathbf{x}}(\tau)$ defined in Eq. 18.

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

3 The Bayes Tree Data structure for Fast Incremental Updates to Sparse Gaussian Process Regression

Previous work on batch continuous-time trajectory estimation as sparse Gaussian process regression [15, 2] assumes that the information matrix $\mathcal{I}$ is sparse (Eq. 15) and applies standard block elimination to factor and solve Eq. 11. But for large numbers of landmarks, this process is very inefficient. In square root SAM [4], matrix column reordering has been applied for efficient Cholesky factorization in a discrete-time context. Similarly, naive periodic variable reordering can be employed here to solve the STEAM problem. (See [16] for details).

However, despite the efficiency of periodic batch updates, it 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.

3.1 The Bayes Tree Data Structure

We base our approach on iSAM 2.0 proposed by Kaess et al. [10], 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 [11]. 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 [11] for details), and how they help solve the sparse Gaussian process regression incrementally.

A Bayes tree is constructed from a Bayes net, which is further 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. 20). $\boldsymbol{\theta} = \{\theta_1, \ldots, \theta_m\}$ 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 $\boldsymbol{\theta}_i$.

$$f(\boldsymbol{\theta}) = \prod_i f_i(\boldsymbol{\theta}_i) \tag{20}$$

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:

$$f(\boldsymbol{\theta}) \propto \exp\left\{-\frac{1}{2}\sum \|\mathbf{A}_i\boldsymbol{\theta}_i - \mathbf{b}_i\|_2^2\right\} = \exp\{-\frac{1}{2}\|\mathbf{A}\boldsymbol{\theta} - \mathbf{b}\|_2^2\} \tag{21}$$

Here $\mathbf{A}_i$ and $\mathbf{b}_i$ are derived from factor $f_i(\cdot)$. $\mathbf{A}$ is a square-root information matrix, with $\boldsymbol{\mathcal{I}} = \mathbf{A}^\intercal\mathbf{A}$ [4], so the QR factor $\mathbf{R}$ of $\mathbf{A}$ 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. 9.

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 $\boldsymbol{\mathcal{K}}^{-1}$ and can be represented by a *sparse* factor graph [2]. In this case, the factors derived from the Gaussian process prior are (suppose $f_j(\cdot)$ is the GP factor between $\mathbf{x}(t_{i-1})$ and $\mathbf{x}(t_i)$):

$$f_j(\boldsymbol{\theta}_j) = f_j(\mathbf{x}(t_{i-1}), \mathbf{x}(t_i)) \propto \exp\{-\frac{1}{2}\left\|\boldsymbol{\Phi}(t_i, t_{i-1})\mathbf{x}(t_{i-1}) + \mathbf{v}_i - \mathbf{x}(t_i)\right\|_{\mathbf{Q}_i}^2\} \tag{22}$$

where $\boldsymbol{\Phi}(t_i, t_{i-1})$ is the state transition matrix, $\mathbf{Q}_i$ is the integral of the covariance of the process noise (Eq. 17), and $\mathbf{v}_i$ is the integral of the exogenous input $\mathbf{v}(t)$ (Eq. 14):

$$\mathbf{v}_i = \int_{t_{i-1}}^{t_i} \boldsymbol{\Phi}(t_i, s)\mathbf{v}(s)ds \tag{23}$$

An illustrative sparse factor graph example including the GP factors is presented in Fig. 1(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 [2].

The key of just-in-time relinearization and fluid variable reordering is to identify the portion of a graph impacted by a new or modified factor, which is difficult to achieve directly from a factor graph. So 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 $f(\theta_i, \mathbf{s}_i)$ and removed from the factor graph, resulting in a new conditional $P(\theta_i|\mathbf{s}_i)$ and a new factor $f(\mathbf{s}_i)$, satisfying $f(\theta_i, \mathbf{s}_i) = P(\theta_i|\mathbf{s}_i)f(\mathbf{s}_i)$. The joint density $f(\theta_i, \mathbf{s}_i)$ is the product of the factors adjacent to θ_i, and $\mathbf{s}_i$ 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 $f(\theta_i, \mathbf{s}_i)$ is Gaussian, due to Eq. 21:

$$f(\theta_i, \mathbf{s}_i) \propto \exp\{-\frac{1}{2}\|\mathbf{a}\theta_i + \mathbf{A}_s\mathbf{s}_i - \mathbf{b}_i\|_2^2\} \tag{24}$$

where $\mathbf{a}$, $\mathbf{A}_s$ and $\mathbf{b}_i$ 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 $P(\theta_i|\mathbf{s}_i)$ is obtained by evaluating Eq.24 with a given $\mathbf{s}_i$:

$$P(\theta_i|\mathbf{s}_i) \propto \exp\{-\frac{1}{2}(\theta_i + \mathbf{r}^\intercal\mathbf{s}_i - d)^2\} \tag{25}$$

where $\mathbf{r} = (\mathbf{a}^\dagger\mathbf{A}_s)^\intercal$, $d = \mathbf{a}^\dagger\mathbf{b}_i$, and $\mathbf{a}^\dagger = (\mathbf{a}^\intercal\mathbf{a})^{-1}\mathbf{a}^\intercal$. $f(\mathbf{s}_i)$ can be further computed by substituting $\theta_i = d - \mathbf{r}^\intercal\mathbf{s}_i$ into Eq. 24. This elimination step is equivalent to one step of Gram-Schmidt. Thus the new conditional $P(\theta_i|\mathbf{s}_i)$ specifies one row in the $\mathbf{R}$ factor of the QR factorization of $\mathbf{A}$. The sequence of the variables to be eliminated is selected to reduce fill-in in $\mathbf{R}$, just as in the case of matrix column reordering. The joint density $f(\boldsymbol{\theta})$ represented by the Bayes net is maximized by assigning $d - \mathbf{r}^\intercal\mathbf{s}_i$ to θ_i, due to Eq. 25, 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, by grouping together conditionals belonging to a clique in the Bayes net 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 impacted.

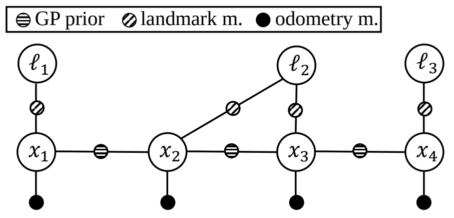

Fig. 1 **(a)** A simple factor graph that includes landmark measurements, odometry measurements, and Gaussian process priors. The odometry measurements are unitary, when they measure the instant velocity in the robot state.

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 3.2 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 Section 4.1 and 4.2 we show that this results in a significant speedup over iSAM2.0.

3.2 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 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 4). 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 factor $f_j(\cdot)$ derived from the measurement $h_k(\cdot)$ is:

$$f_j(\boldsymbol{\theta}_j) \propto \exp\{-\frac{1}{2}\|\mathbf{h}_k(\boldsymbol{\theta}_k + \delta\boldsymbol{\theta}_k) - \mathbf{y}_k\|^2_{\mathbf{R}_k}\} \tag{26}$$

where $\boldsymbol{\theta}_j$ (the variables adjacent to factor $f_j(\cdot)$), and $\boldsymbol{\theta}_k$ (the variables related to measurement $h_k(\cdot)$), are the same set of variables.

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

$$f_j(\boldsymbol{\theta}_j) \propto \exp\{-\frac{1}{2}\|\mathbf{h}_k\left(\bar{\mathbf{x}}(\tau) + \delta\mathbf{x}(\tau)\right) - \mathbf{y}_k\|^2_{\mathbf{R}_k}\},\ \ \boldsymbol{\theta}_j \triangleq \delta\mathbf{x}(\tau) \tag{27}$$

If $\mathbf{x}(\tau)$ is *missing*, then this factor can not 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 directly, we interpolate the state and utilize the linearized measurement function after interpolation (Eq. 13):

$$f_j(\boldsymbol{\theta}_j) \propto \exp\{-\frac{1}{2}\|\mathbf{h}_k\left(\bar{\mathbf{x}}(\tau)\right) + \mathbf{H}_k\boldsymbol{\mathcal{K}}(\tau)\boldsymbol{\mathcal{K}}^{-1}\delta\mathbf{x} - \mathbf{y}_k\|^2_{\mathbf{R}_k}\},\ \boldsymbol{\theta}_j \triangleq \delta\mathbf{x} \tag{28}$$

We apply the interpolation equations for the sparse GP (Eq. 16 and Eq. 18), so that the factor becomes a function of the two nearby states (in contrast to the missing state):

$$f_j(\boldsymbol{\theta}_j) \propto \exp\{-\frac{1}{2}\|\mathbf{h}_k(\bar{\mathbf{x}}(\tau)) + \mathbf{H}_k\left[\boldsymbol{\Lambda}(\tau)\,\boldsymbol{\Psi}(\tau)\right]\boldsymbol{\theta}_j - \mathbf{y}_k\|^2_{\mathbf{R}_k}\}, \boldsymbol{\theta}_j \triangleq \begin{bmatrix}\delta\mathbf{x}(t_{i-1})\\ \delta\mathbf{x}(t_i)\end{bmatrix} \tag{29}$$

where $\bar{\mathbf{x}}$ is specified in Eq. 18.

A factor graph augmented with the factors associated with measurements at missing states has several advantages. 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. 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. Importantly, incorporating GP state interpolation and regression (Sections 2.1 and 2.2) within Bayes tree closely follows MAP inference. In particular, we show in Section 4.1, and Section 4.2 that skipping large numbers of states can reduce computation time by almost 70% with only a small reduction in accuracy. The full incremental algorithm is described in Algorithm 1.

Algorithm 1 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 $\mathbf{x}(\tau) \in \boldsymbol{\theta}_{nf}$ is a missing state, replace it by nearby states (Eq. 18); If $\mathbf{x}(\tau) \in \boldsymbol{\theta}_{nf}$ is a new state to estimate, a GP prior (Eq. 22) is stored, and $\boldsymbol{\theta}_{nf} := \boldsymbol{\theta}_{nf} \cup \mathbf{x}_{i-1}$.
    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 when missing states are involved (Eq. 29).
    4. Add the cached marginal factors from the orphaned sub-trees of the removed cliques.
    5. Eliminate the graph by a new ordering into a Bayes tree, attach back orphaned sub-trees.
    6. Partially update estimate from the root, stop when updates 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
```

4 Experimental Results

We evaluate the performance of our incremental sparse GP regression algorithm to 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. [2] (XL variable ordering), which is periodically executed as data is received.
- **PBVR**: Periodic batch with variable reordering [16]. Variable reordering is applied to achieve efficient matrix factorization.

- **BTGP**: The proposed approach - Bayes tree with Gaussian process prior factors (described in Section 3).

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 3.2).
- **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,[2] to make the comparison fair and meaningful. Evaluation is performed on two datasets summarized in Table 1. We first evaluate performance in a synthetic dataset (Section 4.1), analyzing estimation errors with respect to ground truth data. Results using a real-world dataset are then presented in Section 4.2.

Table 1 Summary of the experimental datasets

	# time steps	# odo. m.	# landmark m.	# landmarks	travel dist.(km)
Synthetic	1,500	1,500	1,500	298	0.2
Auto. Mower	9,658	9,658	3,529	4	1.9

4.1 Synthetic SLAM Exploration Task

This dataset consists of an exploration task with 1,500 time steps. Each time step contains a trajectory state $\mathbf{x}(t_i) = [\,\mathbf{p}(t_i)^\intercal\ \dot{\mathbf{p}}(t_i)^\intercal\,]^\intercal$, $\mathbf{p}(t_i) = [\,x(t_i)\ y(t_i)\ \theta(t_i)\,]^\intercal$,

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

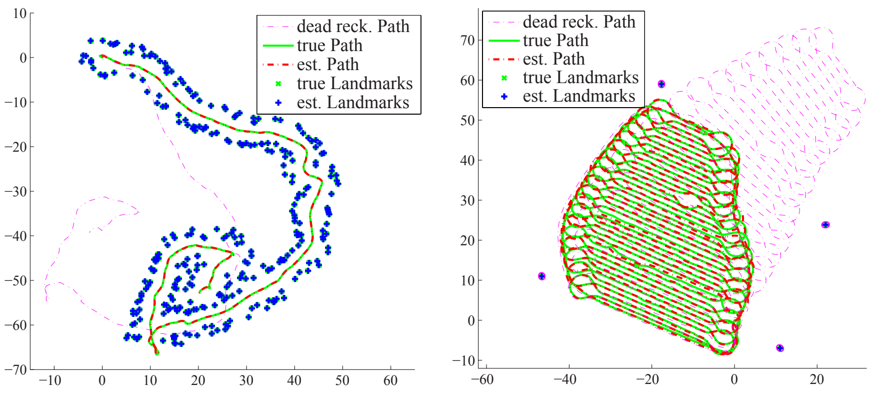

Fig. 2 (left) Synthetic dataset: Ground truth, dead reckoning path, and the estimates are shown. State and landmark estimates obtained from BTGP approach are very close to ground truth. (right) The Autonomous Lawnmower dataset: Ground truth, dead reckoning path and estimates are shown. The range measurements are sparse, noisy, and asynchronous. Ground truth and the estimates of path and landmarks obtained from BTGP are very close.

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 $\ddot{\mathbf{p}}(t) = \mathbf{w}(t)$, i.e. constant velocity, and with zero mean.

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{F}\mathbf{w}(t) \tag{30}$$

$$\mathbf{x}(t) = \left[\mathbf{p}(t)^\intercal \ \dot{\mathbf{p}}(t)^\intercal\right]^\intercal, \quad \mathbf{p}(t) = \left[x(t) \ y(t) \ \theta(t)\right]^\intercal, \quad \mathbf{A} = \begin{bmatrix} \mathbf{0} & \mathbf{I} \\ \mathbf{0} & \mathbf{0} \end{bmatrix},$$
$$\mathbf{F} = \left[\mathbf{0} \ \mathbf{I}\right]^\intercal, \quad \mathbf{w}(t) \sim \mathcal{GP}(\mathbf{0}, \mathbf{Q}_c \delta(t - t')) \tag{31}$$

Note that velocity $\dot{\mathbf{p}}(t)$ must to be included in trajectory state to represent the motion in LTV SDE form [2]. This Gaussian process representation of trajectory is also applied to the real dataset. The odometry and range measurements with Gaussian noise are specified as:

$$\mathbf{y}_{io} = \begin{bmatrix} \cos\theta(t_i) \cdot \dot{x}(t_i) + \sin\theta(t_i) \cdot \dot{y}(t_i) \\ \dot{\theta}(t_i) \end{bmatrix} + \mathbf{n}_o, \ \ y_{ir} = \left\| \left[x(t_i) \ y(t_i)\right]^\intercal - \boldsymbol{\ell}_j \right\|_2 + n_r \tag{32}$$

where $\mathbf{y}_{io}$ consists of the robot-oriented velocity and heading angle velocity with Gaussian noise, and y_{ir} is the distance between the robot and a specific landmark $\boldsymbol{\ell}_j$ at t_i with Gaussian noise.

We compare the computation time of the three approaches (PB, PBVR and BTGP) in Fig. 3. The incremental Gaussian process regression (BTGP) offers significant improvements in computation time compared to the batch approaches (PBVR and PB).

In Fig. 3, we also 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.

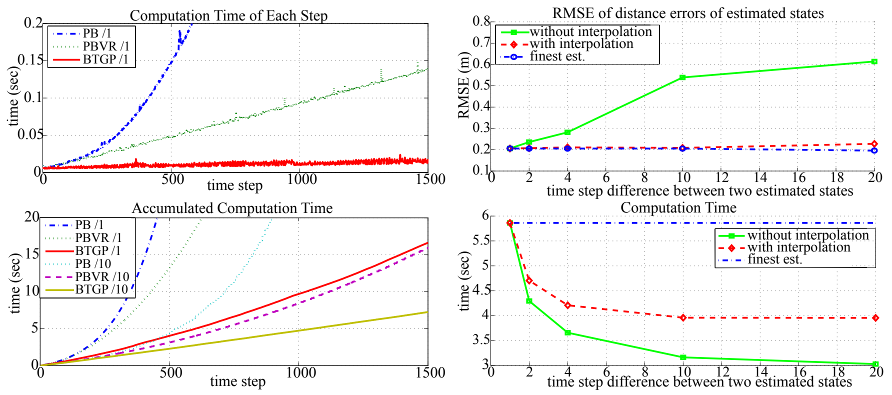

Fig. 3 Synthetic dataset: (left) Comparison of the computation time of three approaches PB, PBVR, and BTGP. The modifiers /1 and /10 indicate frequency of estimate updates — the number of range measurements between updates. Due to the large number of landmarks, 298, variable reordering dramatically improves the performance. (right) 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 $\sim$ 90% of the states (i.e. estimating only $\sim$ 10% of the states) while running BTGP can result in a 33% reduction in computation time over iSAM 2.0 without sacrificing accuracy.

4.2 *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 [6]. 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 [6].) Ground truth paths are computed from GPS readings and have 2cm accuracy according to [6]. The environment, including the locations of the landmarks and the ground truth paths, are shown in Fig. 2. The robot travelled 1.9km, occupied 9,658 poses, and received 3,529 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. 2 and demonstrate that we are able to estimate the robot's trajectory and map with a very high degree of accuracy.

As in Section 4.1, performance of three approaches – PB, PBVR, and BTGP are compared in Fig. 4. 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. 4, 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.

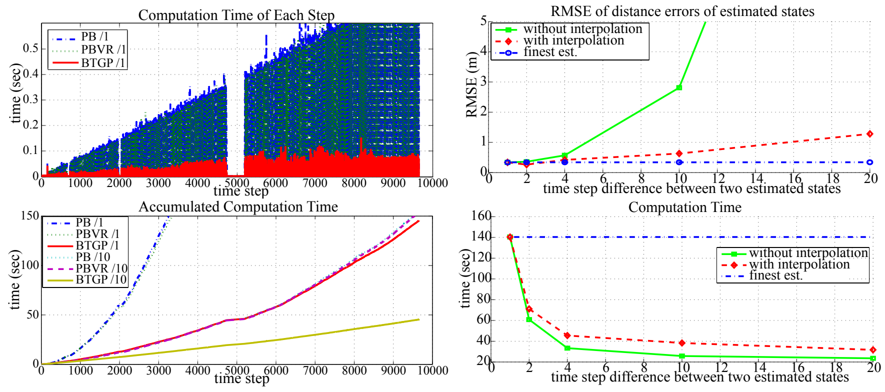

Fig. 4 Autonomous Lawnmower dataset: (left) Comparison of the computation time of PB, PBVR, and BTGP. As in Fig. 3, /1 and /10 are modifiers — the number of range measurement between updates, and no interpolation is used by BTGP. The 'gap' in the upper graph is due to a long stretch around timestep 5000 with no range measurements . Due to the low number of landmarks, variable reordering does not help The incremental BTGP approach dramatically reduces computation time. (right) 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 $\sim 80\%$ of the states within BTGP results in only an 8cm increase in RSME while reducing the overall computation time by 68% over iSAM 2.0.

5 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 8cm on the Autonomous Lawnmower Dataset) .

References

1. T. Bailey and H. Durrant-Whyte. Simultaneous localisation and mapping (SLAM): Part II state of the art. *Robotics and Automation Magazine*, 13(3):108–117, 2006.
2. Tim Barfoot, Chi Hay Tong, and Simo Sarkka. Batch continuous-time trajectory estimation as exactly sparse gaussian process regression. In *Proceedings of Robotics: Science and Systems*, Berkeley, USA, July 2014.
3. Byron Boots and Geoffrey J. Gordon. A spectral learning approach to range-only SLAM. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, 2013.
4. Frank Dellaert and Michael Kaess. Square root sam: Simultaneous localization and mapping via square root information smoothing. *International Journal of Robotics Reasearch*, 25:2006, 2006.
5. J. E. Dennis, Jr. and Robert B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations (Classics in Applied Mathematics, 16)*. Soc for Industrial & Applied Math, 1996. ISBN 0898713641.
6. Joseph Djugash. *Geolocation with Range: Robustness, Efficiency and Scalability*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, November 2010.
7. Hugh Durrant-Whyte and Tim Bailey. Simultaneous localisation and mapping (slam): Part i the essential algorithms. *IEEE ROBOTICS AND AUTOMATION MAGAZINE*, 2:2006, 2006.
8. Gene H. Golub and Charles F. Van Loan. *Matrix Computations (3rd Ed.)*. Johns Hopkins University Press, Baltimore, MD, USA, 1996. ISBN 0-8018-5414-8.
9. M. Kaess, A. Ranganathan, and F. Dellaert. isam: Incremental smoothing and mapping. *Robotics, IEEE Transactions on*, 24(6):1365–1378, Dec 2008. ISSN 1552-3098. doi: 10.1109/TRO.2008.2006706.
10. M. Kaess, H. Johannsson, R. Roberts, V. Ila, J.J. Leonard, and F. Dellaert. iSAM2: Incremental smoothing and mapping using the Bayes tree. *Intl. J. of Robotics Research, IJRR*, 31(2):217–236, Feb 2012.
11. Michael Kaess, Viorela Ila, Richard Roberts, and Frank Dellaert. The bayes tree: An algorithmic foundation for probabilistic robot mapping. In *Algorithmic Foundations of Robotics IX*, pages 157–173. Springer, 2011.
12. Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *In Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 593–598. AAAI, 2002.
13. C.E. Rasmussen and C.K.I. Williams. *Gaussian Processes for Machine Learning*. 2006.
14. Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005. ISBN 0262201623.
15. Chi Hay Tong, Paul Furgale, and Timothy D Barfoot. Gaussian process gauss–newton for non-parametric simultaneous localization and mapping. *The International Journal of Robotics Research*, 32(5):507–525, 2013.
16. Yan Xinyan, Vadim Indelman, and Byron Boots. Incremental sparse gp regression for continuous-time trajectory estimation & mapping. *arXiv preprint arXiv:arXiv:1504.02696*, 2015.