Batch Nonlinear Continuous-Time Trajectory Estimation as Exactly Sparse Gaussian Process Regression

Sean Anderson · Timothy D. Barfoot · Chi Hay Tong · Simo Särkkä

Abstract In this paper, we revisit batch state estimation through the lens of *Gaussian process* (GP) regression. We consider continuous-discrete estimation problems wherein a trajectory is viewed as a one-dimensional GP, with time as the independent variable. Our continuous-time prior can be defined by any nonlinear, time-varying stochastic differential equation driven by white noise; this allows the possibility of smoothing our trajectory estimates using a variety of vehicle dynamics models (e.g., 'constant-velocity'). We show that this class of prior results in an inverse kernel matrix (i.e., covariance matrix between all pairs of measurement times) that is exactly sparse (block-tridiagonal) and that this can be exploited to carry out GP regression (and interpolation) very efficiently. When the prior is based on a linear, time-varying stochastic differential equation and the measurement model is also linear, this GP approach is equivalent to classical, discrete-time smoothing (at the measurement times); when a nonlinearity is present, we iterate over the whole trajectory to maximize accuracy. We test the approach experimentally on a simultaneous trajectory estimation and mapping problem using a mobile robot dataset.

1 Introduction

Probabilistic state estimation has been a core topic in mobile robotics since the 1980s (Durrant-Whyte, 1988; Smith and Cheeseman, 1986; Smith et al., 1990), often as part of the *simultaneous localization and mapping* (SLAM) problem (Bailey and Durrant-Whyte, 2006; Durrant-Whyte and Bailey, 2006). Early work in estimation theory focused on recursive (as opposed to batch) formulations (Kalman, 1960), and this was mirrored in the formulation of SLAM as a filtering problem (Smith et al., 1990). However, despite the fact that continuous-time estimation techniques have been available since the 1960s (Jazwinski, 1970; Kalman and Bucy, 1961), trajectory estimation for mobile robots has been formulated almost exclusively in discrete time.

Lu and Milios (1997) showed how to formulate SLAM as a batch estimation problem incorporating both odometry measurements (to smooth solutions) as well as landmark measurements. This can be viewed as a generalization of *bundle adjustment* (Brown, 1958; Sibley et al., 2010), which did not incorporate odometry. Today, batch approaches in mobile robotics are commonplace (e.g., GraphSLAM by Thrun and Montemerlo (2006)). Kaess et al. (2008) show how batch solutions can be efficiently updated as new measurements are gathered and Strasdat et al. (2010) show that batch methods are able to achieve higher accuracy than their filtering counterparts, for the same computational cost. Most of these results are formulated in discrete time.

Discrete-time representations of robot trajectories are sufficient in many situations, but they do not work well when estimating motion from certain types of sensors (e.g., rolling-shutter cameras and scanning laser-rangefinders) and

S. Anderson
Autonomous Space Robotics Lab, University of Toronto Institute for Aerospace Studies, Canada
E-mail: sean.anderson@mail.utoronto.ca

T. D. Barfoot
Autonomous Space Robotics Lab, University of Toronto Institute for Aerospace Studies, Canada
E-mail: tim.barfoot@utoronto.ca

C. H. Tong
Mobile Robotics Group, University of Oxford, United Kingdom
E-mail: chi@robots.ox.ac.uk

S. Särkkä
Department of Biomedical Engineering and Computational Science, Aalto University, Finland
E-mail: simo.sarkka@aalto.fi

Fig. 1 To carry out batch trajectory estimation, we use GP regression with a smooth, continuous-time prior and discrete-time measurements. This allows us to query the trajectory at any time of interest, τ.

sensor combinations (e.g., high datarate, asynchronous). In these cases, a smooth, continuous-time representation of the trajectory is more suitable. For example, in the case of estimating motion from a scanning-while-moving sensor, a discrete-time approach (with no motion prior) can fail to find a unique solution; something is needed to tie together the observations acquired at many unique timestamps. Additional sensors (e.g., odometry or inertial measurements) could be introduced to serve in this role, but this may not always be possible. In these cases, a motion prior can be used instead (or as well), which is most naturally expressed in continuous time.

One approach to continuous-time trajectory representation is to use interpolation (e.g., linear, spline) directly between nearby discrete poses (Bibby and Reid, 2010; Bosse and Zlot, 2009; Dong and Barfoot, 2012; Furgale et al., 2012; Hedborg et al., 2012; Lovegrove et al., 2013). Instead, we choose to represent the trajectory nonparametrically as a one-dimensional Gaussian process (GP) (Rasmussen and Williams, 2006), with time as the independent variable (see Figure 1). Tong et al. (2012, 2013) show that querying the state of the robot at a time of interest can be viewed as a nonlinear, GP regression problem. While their approach is very general, allowing a variety of GP priors over robot trajectories, it is also quite expensive due to the need to invert a large, dense kernel matrix.

While GPs have been used in robotic state estimation to accomplish dimensionality reduction (Ferris et al., 2006, 2007; Lawrence, 2003) and to represent the measurement and motion models (Deisenroth et al., 2012; Ko and Fox, 2009, 2011), these uses are quite different than representing the latent robot trajectory as a GP (Tong et al., 2012, 2013).

In this paper, we consider a particular class of GPs, generated by nonlinear, time-varying (NTV) stochastic differential equations (SDE) driven by white noise. We first show that GPs based on linear, time-varying (LTV) SDEs have an inverse kernel matrix that is *exactly sparse* (block-tridiagonal) and can be derived in closed form; an approximation for GPs based on a NTV SDE is then shown that

results in the same sparsity properties as the linear case. Concentrating on this class of covariance functions results in only a minor loss of generality, because many commonly used covariance functions such the Matérn class and the squared exponential covariance function can be exactly or approximately represented as linear SDEs (Hartikainen and Särkkä, 2010; Särkkä et al., 2013; Solin and Särkkä, 2014). We provide an example of this relationship at the end of this paper. The resulting sparsity allows the approach of Tong et al. (2012, 2013) to be implemented very efficiently. The intuition behind why this is possible is that the state we are estimating is *Markovian* for this class of GPs, which implies that the corresponding precision matrices are sparse (Lindgren et al., 2011).

This sparsity property has been exploited in estimation theory to allow recursive methods (both filtering and smoothing) since the 1960s (Kalman, 1960; Kalman and Bucy, 1961). Jumarie (1990) offers an interesting discussion on nonlinear, continuous-time filtering (using both a nonlinear dynamical plant and nonlinear observations). The tracking literature, in particular, has made heavy use of motion priors (in both continuous and discrete time) and has exploited the Markov property for efficient solutions (Maybeck, 1979). For the nonlinear, discrete-time case, Bell (1994) shows that Kalman filtering and smoothing with iterated relinearization is equivalent to Gauss-Newton on the full-state trajectory. It is of no surprise that in vision and robotics, discrete-time batch methods commonly exploit this sparsity property as well (Triggs et al., 2000). In this paper, we make the (retrospectively obvious) observation that this sparsity can also be exploited in a batch, continuous-time context. The result is that we derive a principled method to construct trajectory-smoothing terms for batch optimization (or factors in a factor-graph representation) based on a class of useful motion models; this paves the way to incorporate vehicle dynamics models, including exogenous inputs, to help with trajectory estimation.

Therefore, our main contribution is to emphasize the strong connection between classical estimation theory and machine learning via GP regression. We use the fact that the inverse kernel matrix is sparse for a class of useful GP priors (Lindgren et al., 2011; Särkkä et al., 2013) in a new way to efficiently implement nonlinear, GP regression for batch, continuous-time trajectory estimation. We also show that this naturally leads to a subtle generalization of SLAM that we call *simultaneous trajectory estimation and mapping* (STEAM), with the difference being that chains of discrete *poses* are replaced with *Markovian trajectories* in order to incorporate continuous-time motion priors in an efficient way. Finally, by using this GP paradigm, we are able to exploit the classic GP interpolation approach to query the trajectory at any time of interest in an efficient manner.

This ability to query the trajectory at any time of interest in a principled way could be useful in a variety of situations. For example, Newman et al. (2009) mapped a large urban area using a combination of stereo vision and laser rangefinders; the motion was estimated using the camera and the laser data were subsequently placed into a three-dimensional map based on this estimated motion. Our method could provide a seamless means to (i) estimate the camera trajectory and then (ii) query this trajectory at every laser acquisition time.

This paper is a significant extension of our recent conference paper (Barfoot et al., 2014). We build upon the exactly-sparse GP-regression approach that used *linear*, time-varying SDEs, and show how to use GPs based on *nonlinear*, time-varying SDEs, while maintaining the same level of sparsity as the linear case. The algorithmic differences are discussed in detail and results are provided using comparable linear and nonlinear priors. Furthermore, this paper shows how the block-tridiagonal sparsity of the kernel matrix can be exploited to improve the computational performance of hyperparameter training. Finally, discussion is provided on using the GP interpolation equation for further state reduction at the cost of accuracy.

The paper is organized as follows. Section 2 summarizes the general approach to batch state estimation via GP regression. Section 3 describes the particular class of GPs we use and elaborates on our main result concerning sparsity. Section 4 demonstrates this main result on a mobile robot example using a 'constant-velocity' prior and compares the computational cost to methods that do not exploit the sparsity. Section 5 provides some discussion and Section 6 concludes the paper.

2 Gaussian Process Regression

We take a *Gaussian-process-regression* approach to state estimation. This allows us to (i) represent trajectories in continuous time (and therefore query the solution at any time of interest), and (ii) optimize our solution by iterating over the entire trajectory (recursive methods typically iterate at a single timestep).

We will consider systems with a continuous-time, GP process model and a discrete-time, nonlinear measurement model:

$$\mathbf{x}(t) \sim \mathcal{GP}(\check{\mathbf{x}}(t), \check{\mathbf{P}}(t, t')), \qquad t_0 < t, t' \tag{1}$$

$$\mathbf{y}_n = \mathbf{g}(\mathbf{x}(t_n)) + \mathbf{n}_n, \qquad t_1 < \cdots < t_N, \tag{2}$$

where $\mathbf{x}(t)$ is the state, $\check{\mathbf{x}}(t)$ is the mean function, $\check{\mathbf{P}}(t, t')$ is the covariance function, $\mathbf{y}_n$ are measurements, $\mathbf{n}_n \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_n)$ is Gaussian measurement noise, $\mathbf{g}(\cdot)$ is a nonlinear measurement model, and $t_1 < \ldots < t_N$ is a sequence of measurement times. For the moment, we do not consider

the STEAM problem (i.e., the state does not include landmarks), but we will return to this case in our example later.

We follow the approach of Tong et al. (2013) to set up our batch, GP state estimation problem. We will first assume that we want to query the state at the measurement times, and will return to querying at other times later on. We start with an initial guess, $\mathbf{x}_{\mathrm{op}}$, for the trajectory that will be improved iteratively. At each iteration, we solve for the optimal perturbation, $\delta\mathbf{x}^\star$, to our guess using GP regression, with our measurement model linearized about the current best guess.

The joint likelihood between the state and the measurements (both at the measurement times) is

$$p\left(\begin{bmatrix}\mathbf{x}\\\mathbf{y}\end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix}\check{\mathbf{x}}\\\mathbf{g} + \mathbf{G}(\check{\mathbf{x}} - \mathbf{x}_{\mathrm{op}})\end{bmatrix}, \begin{bmatrix}\check{\mathbf{P}} & \check{\mathbf{P}}\mathbf{G}^T\\\mathbf{G}\check{\mathbf{P}} & \mathbf{G}\check{\mathbf{P}}\mathbf{G}^T + \mathbf{R}\end{bmatrix}\right), \tag{3}$$

where

$$\mathbf{x} = \begin{bmatrix}\mathbf{x}(t_0)\\\vdots\\\mathbf{x}(t_N)\end{bmatrix}, \quad \mathbf{x}_{\mathrm{op}} = \begin{bmatrix}\mathbf{x}_{\mathrm{op}}(t_0)\\\vdots\\\mathbf{x}_{\mathrm{op}}(t_N)\end{bmatrix}, \quad \check{\mathbf{x}} = \begin{bmatrix}\check{\mathbf{x}}(t_0)\\\vdots\\\check{\mathbf{x}}(t_N)\end{bmatrix},$$

$$\mathbf{y} = \begin{bmatrix}\mathbf{y}_1\\\vdots\\\mathbf{y}_N\end{bmatrix}, \quad \mathbf{g} = \begin{bmatrix}\mathbf{g}(\mathbf{x}_{\mathrm{op}}(t_1))\\\vdots\\\mathbf{g}(\mathbf{x}_{\mathrm{op}}(t_N))\end{bmatrix}, \quad \mathbf{G} = \left.\frac{\partial\mathbf{g}}{\partial\mathbf{x}}\right|_{\mathbf{x}_{\mathrm{op}}},$$

$$\mathbf{R} = \mathrm{diag}\left(\mathbf{R}_1, \ldots, \mathbf{R}_N\right), \quad \check{\mathbf{P}} = \left[\check{\mathbf{P}}(t_i, t_j)\right]_{ij}.$$

Note, the measurement model is linearized about our best guess so far. We then have that the Gaussian posterior is

$$p(\mathbf{x}|\mathbf{y}) =$$
$$\mathcal{N}\left(\underbrace{\check{\mathbf{x}} + \check{\mathbf{P}}\mathbf{G}^T\left(\mathbf{G}\check{\mathbf{P}}\mathbf{G}^T + \mathbf{R}\right)^{-1}\left(\mathbf{y} - \mathbf{g} - \mathbf{G}\left(\check{\mathbf{x}} - \mathbf{x}_{\mathrm{op}}\right)\right)}_{\hat{\mathbf{x}}, \text{ the posterior mean}},\right.$$
$$\left.\underbrace{\check{\mathbf{P}} - \check{\mathbf{P}}\mathbf{G}^T\left(\mathbf{G}\check{\mathbf{P}}\mathbf{G}^T + \mathbf{R}\right)^{-1}\mathbf{G}\check{\mathbf{P}}}_{\hat{\mathbf{P}}, \text{ the posterior covariance}}\right). \tag{4}$$

Letting $\delta\mathbf{x}^\star = \hat{\mathbf{x}} - \mathbf{x}_{\mathrm{op}}$, and rearranging the posterior mean expression using the Sherman-Morrison-Woodbury identity, we have

$$\left(\check{\mathbf{P}}^{-1} + \mathbf{G}^T\mathbf{R}^{-1}\mathbf{G}\right)\delta\mathbf{x}^\star = \check{\mathbf{P}}^{-1}(\check{\mathbf{x}} - \mathbf{x}_{\mathrm{op}}) + \mathbf{G}^T\mathbf{R}^{-1}(\mathbf{y} - \mathbf{g}), \tag{5}$$

which is a linear system for $\delta\mathbf{x}^\star$ and can be viewed as the solution to the associated *maximum a posteriori* (MAP) problem. We know that the $\mathbf{G}^T\mathbf{R}^{-1}\mathbf{G}$ term in (5) is block-diagonal (assuming each measurement depends on the state at a single time), but in general $\check{\mathbf{P}}^{-1}$ could be dense, depending on the choice of GP prior. At each iteration, we solve for $\delta\mathbf{x}^\star$ and then update the guess according to $\mathbf{x}_{\mathrm{op}} \leftarrow \mathbf{x}_{\mathrm{op}} + \delta\mathbf{x}^\star$; upon convergence we set $\hat{\mathbf{x}} = \mathbf{x}_{\mathrm{op}}$. This is effectively Gauss-Newton optimization over the whole trajectory.

We may want to also query the state at some other time(s) of interest (in addition to the measurement times). Though we could jointly estimate the trajectory at the measurement and query times, a better idea is to use GP interpolation after the solution at the measurement times converges (Rasmussen and Williams, 2006; Tong et al., 2013) (see Section 3.1.3 for more details). GP interpolation automatically picks the correct interpolation scheme for a given prior; it arrives at the same answer as the joint approach (in the linear case), but at lower computational cost.

In general, this GP approach has complexity $O(N^3 + N^2 J)$, where N is the number of measurement times and J is the number of query times (the initial solve is $O(N^3)$ and the queries are $O(N^2 J)$). This is quite expensive, and therefore we will seek to improve the cost by exploiting the structure of the matrices involved under a particular class of GP priors.

3 A Class of Exactly Sparse GP Priors

3.1 Linear, Time-Varying Stochastic Differential Equations

We now show that the inverse kernel matrix is exactly sparse for a particular class of useful GP priors. We consider GPs generated by linear, time-varying (LTV) stochastic differential equations (SDE) of the form

$$\dot{\mathbf{x}}(t) = \mathbf{F}(t)\mathbf{x}(t) + \mathbf{v}(t) + \mathbf{L}(t)\mathbf{w}(t), \tag{6}$$

where $\mathbf{x}(t)$ is the state, $\mathbf{v}(t)$ is a (known) exogenous input, $\mathbf{w}(t)$ is white process noise, and $\mathbf{F}(t)$, $\mathbf{L}(t)$ are time-varying system matrices. The process noise is given by

$$\mathbf{w}(t) \sim \mathcal{GP}(\mathbf{0}, \mathbf{Q}_C\,\delta(t - t')), \tag{7}$$

a (stationary) zero-mean *Gaussian process* (GP) with (symmetric, positive-definite) *power-spectral density matrix*, $\mathbf{Q}_C$, and $\delta(\cdot)$ is the *Dirac delta function*.

The general solution to this LTV SDE (Maybeck, 1979; Stengel, 1994) is

$$\mathbf{x}(t) = \mathbf{\Phi}(t, t_0)\mathbf{x}(t_0) + \int_{t_0}^{t} \mathbf{\Phi}(t, s)\left(\mathbf{v}(s) + \mathbf{L}(s)\mathbf{w}(s)\right)\,ds, \tag{8}$$

where $\mathbf{\Phi}(t, s)$ is known as the *transition matrix*. From this model, we seek the mean and covariance functions for $\mathbf{x}(t)$.

3.1.1 Mean Function

For the mean function, we take the expected value of (8):

$$\check{\mathbf{x}}(t) = E[\mathbf{x}(t)] = \mathbf{\Phi}(t, t_0)\check{\mathbf{x}}_0 + \int_{t_0}^{t} \mathbf{\Phi}(t, s)\mathbf{v}(s)\,ds, \tag{9}$$

where $\check{\mathbf{x}}_0 = \check{\mathbf{x}}(t_0)$ is the initial value of the mean. If we now have a sequence of measurement times, $t_0 < t_1 < t_2 < \cdots < t_N$, then we can write the mean at these times in *lifted form* as

$$\check{\mathbf{x}} = \mathbf{F}\mathbf{v}, \tag{10}$$

where

$$\check{\mathbf{x}} = \begin{bmatrix} \check{\mathbf{x}}(t_0) \\ \check{\mathbf{x}}(t_1) \\ \vdots \\ \check{\mathbf{x}}(t_N) \end{bmatrix}, \quad \mathbf{v} = \begin{bmatrix} \check{\mathbf{x}}_0 \\ \mathbf{v}_1 \\ \vdots \\ \mathbf{v}_N \end{bmatrix}, \quad \mathbf{v}_n = \int_{t_{n-1}}^{t_n} \mathbf{\Phi}(t_n, s)\mathbf{v}(s)\,ds,$$

$$\mathbf{F} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{\Phi}(t_1, t_0) & \mathbf{1} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{\Phi}(t_2, t_0) & \mathbf{\Phi}(t_2, t_1) & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \mathbf{0} & \mathbf{0} \\ \mathbf{\Phi}(t_{N-1}, t_0) & \mathbf{\Phi}(t_{N-1}, t_1) & \cdots & \mathbf{1} & \mathbf{0} \\ \mathbf{\Phi}(t_N, t_0) & \mathbf{\Phi}(t_N, t_1) & \cdots & \mathbf{\Phi}(t_N, t_{N-1}) & \mathbf{1} \end{bmatrix}. \tag{11}$$

Note that $\mathbf{F}$, the *lifted transition matrix*, is lower-triangular. We arrive at this form by simply splitting up (9) into a sum of integrals between each pair of measurement times.

3.1.2 Covariance Function

For the covariance function, we take the second moment of (8) to arrive at

$$\begin{aligned} \check{\mathbf{P}}(t, t') &= E\left[(\mathbf{x}(t) - \check{\mathbf{x}}(t))(\mathbf{x}(t') - \check{\mathbf{x}}(t'))^T\right] \\ &= \mathbf{\Phi}(t, t_0)\check{\mathbf{P}}_0\mathbf{\Phi}(t', t_0)^T \\ &\quad + \int_{t_0}^{\min(t, t')} \mathbf{\Phi}(t, s)\mathbf{L}(s)\mathbf{Q}_C\mathbf{L}(s)^T\mathbf{\Phi}(t', s)^T\,ds, \end{aligned} \tag{12}$$

where $\check{\mathbf{P}}_0$ is the initial covariance at t_0 and we have assumed $E[\mathbf{x}(t_0)\mathbf{w}(t)^T] = \mathbf{0}$. Using a sequence of measurement times, $t_0 < t_1 < t_2 < \cdots < t_N$, we can write the covariance between two times as

$$\check{\mathbf{P}}(t_i, t_j) = \begin{cases} \mathbf{\Phi}(t_i, t_j)\left(\sum_{r=0}^{j} \mathbf{\Phi}(t_j, t_r)\mathbf{Q}_r\mathbf{\Phi}(t_j, t_r)^T\right) & t_j < t_i \\ \sum_{r=0}^{i} \mathbf{\Phi}(t_i, t_r)\mathbf{Q}_r\mathbf{\Phi}(t_i, t_r)^T & t_i = t_j \\ \left(\sum_{r=0}^{i} \mathbf{\Phi}(t_i, t_r)\mathbf{Q}_r\mathbf{\Phi}(t_i, t_r)^T\right)\mathbf{\Phi}(t_j, t_i)^T & t_i < t_j \end{cases} \tag{13}$$

where

$$\mathbf{Q}_n = \int_{t_{n-1}}^{t_n} \mathbf{\Phi}(t_n, s)\mathbf{L}(s)\mathbf{Q}_C\mathbf{L}(s)^T\mathbf{\Phi}(t_n, s)^T\,ds, \tag{14}$$

for $n = 1\ldots N$ and $\mathbf{Q}_0 = \check{\mathbf{P}}_0$ (to keep the notation simple). Given this preparation, we are now ready to state the main sparsity result that we will exploit in the rest of the paper.

Lemma 1 *Let $t_0 < t_1 < t_2 < \cdots < t_N$ be a monotonically increasing sequence of time values. Using* (13)*, we define the* $(N+1) \times (N+1)$ *kernel matrix (i.e., the prior covariance matrix between all pairs of times),* $\check{\mathbf{P}} = \left[\check{\mathbf{P}}(t_i, t_j) \right]_{ij}$. *Then, we can factor* $\check{\mathbf{P}}$ *according to a lower-diagonal-upper decomposition,*

$$\check{\mathbf{P}} = \mathbf{F}\mathbf{Q}\mathbf{F}^T, \tag{15}$$

where $\mathbf{F}$ *is the lower-triangular matrix given in* (3.1.1) *and* $\mathbf{Q} = \mathrm{diag}\left(\check{\mathbf{P}}_0, \mathbf{Q}_1, \ldots, \mathbf{Q}_N\right)$ *with* $\mathbf{Q}_n$ *given in* (14).

Proof Straightforward to verify by substitution. □

Theorem 1 *The inverse of the kernel matrix constructed in Lemma 1,* $\check{\mathbf{P}}^{-1}$, *is exactly sparse (block-tridiagonal).*

Proof The decomposition of $\check{\mathbf{P}}$ in Lemma 1 provides

$$\check{\mathbf{P}}^{-1} = (\mathbf{F}\mathbf{Q}\mathbf{F}^T)^{-1} = \mathbf{F}^{-T}\mathbf{Q}^{-1}\mathbf{F}^{-1}. \tag{16}$$

where the inverse of the lifted transition matrix is

$$\mathbf{F}^{-1} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ -\boldsymbol{\Phi}(t_1, t_0) & \mathbf{1} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & -\boldsymbol{\Phi}(t_2, t_1) & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \ddots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \cdots & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \cdots & -\boldsymbol{\Phi}(t_N, t_{N-1}) & \mathbf{1} \end{bmatrix}, \tag{17}$$

and $\mathbf{Q}^{-1}$ is block-diagonal. The block-tridiagonal property of $\check{\mathbf{P}}^{-1}$ follows by substitution and multiplication. □

While the block-tridiagonal property stated in Theorem 1 has been exploited in vision and robotics for a long time (Lu and Milios, 1997; Triggs et al., 2000; Thrun and Montemerlo, 2006), the usual route to this point is to begin by converting the continuous-time motion model to discrete time and then to directly formulate a maximum a posteriori optimization problem; this bypasses writing out the full expression for $\check{\mathbf{P}}$ and jumps to an expression for $\check{\mathbf{P}}^{-1}$. However, we require expressions for both $\check{\mathbf{P}}$ and $\check{\mathbf{P}}^{-1}$ to carry out our GP reinterpretation and facilitate querying the trajectory at an arbitrary time (through interpolation). That said, it is also worth noting we have not needed to convert the motion model to discrete time and have made no approximations thus far.

Given the above results, the prior over the state (at the measurement times) can be written as

$$\mathbf{x} \sim \mathcal{N}\left(\check{\mathbf{x}}, \check{\mathbf{P}}\right) = \mathcal{N}\left(\mathbf{F}\mathbf{v}, \mathbf{F}\mathbf{Q}\mathbf{F}^T\right). \tag{18}$$

Fig. 2 Although we began with a continuous-time prior to smooth our trajectory, the class of exactly sparse GPs results in only $N+1$ smoothing terms, J_i, in the associated optimization problem, the solution of which is (19). We can depict these graphically as factors (black dots) in a factor-graph representation of the prior (Dellaert and Kaess, 2006). The triangles are *trajectory states*, the nature of which depends on the choice of prior.

More importantly, using the result of Theorem 1 in (5) gives

$$\overbrace{\left(\mathbf{F}^{-T}\mathbf{Q}^{-1}\mathbf{F}^{-1} + \mathbf{G}^T\mathbf{R}^{-1}\mathbf{G}\right)}^{\text{block-tridiagonal}} \delta\mathbf{x}^\star$$
$$= \mathbf{F}^{-T}\mathbf{Q}^{-1}(\mathbf{v} - \mathbf{F}^{-1}\mathbf{x}_{\mathrm{op}}) + \mathbf{G}^T\mathbf{R}^{-1}(\mathbf{y} - \mathbf{g}). \tag{19}$$

which can be solved in $O(N)$ time (at each iteration), using a sparse solver (e.g., sparse Cholesky decomposition then forward-backward passes). In fact, in the case of a linear measurement model, one such solver is the classical, forward-backward smoother (i.e., Kalman or Rauch–Tung–Striebel smoother) (Bell, 1994). Put another way, the forward-backward smoother is possible *because* of the sparse structure of (19). For nonlinear measurement models, our scheme iterates over the whole trajectory; it is therefore related to, but not the same as, the 'extended' version of the forward-backward smoother (Särkkä, 2013; Särkkä and Sarmavuori, 2013; Särkkä et al., 2013).

Perhaps the most interesting outcome of Theorem 1 is that, although we are using a continuous-time prior to smooth our trajectory, at implementation we require only $N + 1$ smoothing terms in the associated MAP optimization problem: N between consecutive pairs of measurement times plus 1 at the initial time (unless we are also estimating a map). As mentioned before, this is the same form that we would have arrived at had we started by converting our motion model to discrete time at the beginning (Lu and Milios, 1997; Triggs et al., 2000; Thrun and Montemerlo, 2006). This equivalence has been noticed before for recursive solutions to estimation problems with a continuous-time state and discrete-time measurements (Särkkä, 2006), but not in the batch scenario. Figure 2 depicts the $N + 1$ smoothing terms in a factor-graph representation of the prior (Dellaert and Kaess, 2006; Kaess et al., 2012).

However, while the form of the smoothing terms/factors is similar to the original discrete-time form introduced by Lu and Milios (1997), our approach provides a principled method for their construction, starting from the continuous-time motion model. Critically, we stress that the state being estimated must be *Markovian* in order to obtain the desirable

sparse structure. In the experiment section, we will investigate a common GP prior, namely the 'constant-velocity' or white-noise-on-acceleration model: $\ddot{\mathbf{p}}(t) = \mathbf{w}(t)$, where $\mathbf{p}(t)$ represents position. For this choice of model, $\mathbf{p}(t)$ is not Markovian, but

$$\mathbf{x}(t) = \begin{bmatrix} \mathbf{p}(t) \\ \dot{\mathbf{p}}(t) \end{bmatrix}, \tag{20}$$

is. This implies that, if we want to use the 'constant-velocity' prior and enjoy the sparse structure without approximation, we must estimate a stacked state with both position and velocity. Marginalizing out the velocity variables fills in the inverse kernel matrix, thereby destroying the sparsity.

If all we cared about was estimating the value of the state at the measurement times, our GP paradigm arguably offers little beyond a reinterpretation of the usual discrete-time approach to batch estimation. However, by taking the time to set up the problem in this manner, we can now query the trajectory at *any* time of interest using the classic interpolation scheme that is inherent to GP regression (Tong et al., 2013).

3.1.3 Querying the Trajectory

As discussed in Section 2, after we solve for the trajectory at the measurement times, we may want to query it at other times of interest. This operation also benefits greatly from the sparse structure. To keep things simple, we consider a single query time, $t_n \leq \tau < t_{n+1}$ (see Figure 1). The standard linear GP interpolation formulas (Rasmussen and Williams, 2006; Tong et al., 2013) are

$$\hat{\mathbf{x}}(\tau) = \check{\mathbf{x}}(\tau) + \check{\mathbf{P}}(\tau)\check{\mathbf{P}}^{-1}(\hat{\mathbf{x}} - \check{\mathbf{x}}), \tag{21a}$$

$$\hat{\mathbf{P}}(\tau, \tau) = \check{\mathbf{P}}(\tau, \tau) + \check{\mathbf{P}}(\tau)\check{\mathbf{P}}^{-1}\left(\hat{\mathbf{P}} - \check{\mathbf{P}}\right)\check{\mathbf{P}}^{-T}\check{\mathbf{P}}(\tau)^T, \tag{21b}$$

where $\check{\mathbf{P}}(\tau) = \begin{bmatrix} \check{\mathbf{P}}(\tau, t_0) \cdots \check{\mathbf{P}}(\tau, t_N) \end{bmatrix}$. Note, we write (21b) in a less common, but equivalent, form, as we intend to exploit the sparsity of the product $\check{\mathbf{P}}(\tau)\check{\mathbf{P}}^{-1}$.

For the mean function at the query time, we simply have

$$\check{\mathbf{x}}(\tau) = \mathbf{\Phi}(\tau, t_n)\check{\mathbf{x}}_n + \int_{t_n}^{\tau} \mathbf{\Phi}(\tau, s)\mathbf{v}(s)\, ds, \tag{22}$$

which can be evaluated in $O(1)$ time. For the covariance function at the query time, we have

$$\check{\mathbf{P}}(\tau, \tau) = \mathbf{\Phi}(\tau, t_n)\check{\mathbf{P}}(t_n, t_n)\mathbf{\Phi}(\tau, t_n)^T + \\ \int_{t_n}^{\tau} \mathbf{\Phi}(\tau, s)\mathbf{L}(s)\mathbf{Q}_C\mathbf{L}(s)^T\mathbf{\Phi}(\tau, s)^T\, ds, \tag{23}$$

which is also $O(1)$ to evaluate.

The computational savings come from the sparsity of the product $\check{\mathbf{P}}(\tau)\check{\mathbf{P}}^{-1}$, which represents the burden of the cost in

the interpolation formula. After some effort, it turns out we can write $\check{\mathbf{P}}(\tau)$ as

$$\check{\mathbf{P}}(\tau) = \mathbf{V}(\tau)\mathbf{F}^T, \tag{24}$$

where $\mathbf{F}$ was defined before,

$$\mathbf{V}(\tau) = \Big[\mathbf{\Phi}(\tau, t_n)\mathbf{\Phi}(t_n, t_0)\check{\mathbf{P}}_0 \;\; \mathbf{\Phi}(\tau, t_n)\mathbf{\Phi}(t_n, t_1)\mathbf{Q}_1 \cdots \\ \cdots \mathbf{\Phi}(\tau, t_n)\mathbf{\Phi}(t_n, t_{n-1})\mathbf{Q}_{n-1} \;\; \mathbf{\Phi}(\tau, t_n)\mathbf{Q}_n \cdots \\ \cdots \mathbf{Q}_\tau\mathbf{\Phi}(t_{n+1}, \tau)^T \;\; \mathbf{0} \cdots \mathbf{0} \Big], \tag{25}$$

and

$$\mathbf{Q}_\tau = \int_{t_n}^{\tau} \mathbf{\Phi}(\tau, s)\mathbf{L}(s)\mathbf{Q}_C\mathbf{L}(s)^T\mathbf{\Phi}(\tau, s)^T\, ds. \tag{26}$$

Returning to the desired product, we have

$$\check{\mathbf{P}}(\tau)\check{\mathbf{P}}^{-1} = \mathbf{V}(\tau)\underbrace{\mathbf{F}^T\mathbf{F}^{-T}}_{\mathbf{1}}\mathbf{Q}^{-1}\mathbf{F}^{-1} = \mathbf{V}(\tau)\mathbf{Q}^{-1}\mathbf{F}^{-1}. \tag{27}$$

Since $\mathbf{Q}^{-1}$ is block-diagonal, and $\mathbf{F}^{-1}$ has only the main diagonal and the one below it non-zero, we can evaluate the product very efficiently. Note, there are exactly two non-zero block-columns:

$$\check{\mathbf{P}}(\tau)\check{\mathbf{P}}^{-1} = \begin{bmatrix} \mathbf{0} & \cdots & \mathbf{0} & \underbrace{\mathbf{\Lambda}(\tau)}_{\substack{\text{block} \\ \text{col. } n}} & \underbrace{\mathbf{\Psi}(\tau)}_{\substack{\text{block} \\ \text{col.} n+1}} & \mathbf{0} & \cdots & \mathbf{0} \end{bmatrix}, \tag{28}$$

where

$$\mathbf{\Lambda}(\tau) = \mathbf{\Phi}(\tau, t_n) - \mathbf{\Psi}(\tau)\mathbf{\Phi}(t_{n+1}, t_n), \tag{29a}$$

$$\mathbf{\Psi}(\tau) = \mathbf{Q}_\tau\mathbf{\Phi}(t_{n+1}, \tau)^T\mathbf{Q}_{n+1}^{-1}. \tag{29b}$$

Inserting this into (21), we have

$$\hat{\mathbf{x}}(\tau) = \check{\mathbf{x}}(\tau) + \begin{bmatrix} \mathbf{\Lambda}(\tau) & \mathbf{\Psi}(\tau) \end{bmatrix} \begin{bmatrix} \hat{\mathbf{x}}_n - \check{\mathbf{x}}_n \\ \hat{\mathbf{x}}_{n+1} - \check{\mathbf{x}}_{n+1} \end{bmatrix}, \tag{30a}$$

$$\hat{\mathbf{P}}(\tau, \tau) = \check{\mathbf{P}}(\tau, \tau) + \begin{bmatrix} \mathbf{\Lambda}(\tau) & \mathbf{\Psi}(\tau) \end{bmatrix} \left(\begin{bmatrix} \hat{\mathbf{P}}_{n,n} & \hat{\mathbf{P}}_{n,n+1} \\ \hat{\mathbf{P}}_{n+1,n} & \hat{\mathbf{P}}_{n+1,n+1} \end{bmatrix} \right. \\ \left. - \begin{bmatrix} \check{\mathbf{P}}_{n,n} & \check{\mathbf{P}}_{n,n+1} \\ \check{\mathbf{P}}_{n+1,n} & \check{\mathbf{P}}_{n+1,n+1} \end{bmatrix} \right) \begin{bmatrix} \mathbf{\Lambda}(\tau)^T \\ \mathbf{\Psi}(\tau)^T \end{bmatrix}, \tag{30b}$$

which is a linear combination of just the terms from t_n and t_{n+1}. If the query time is beyond the last measurement time, $t_N < \tau$, the expression will involve only the term at t_N and represents extrapolation/prediction rather than interpolation/smoothing. In summary, to query the trajectory at a single time of interest is $O(1)$ complexity.

3.1.4 Interpolating Measurement Times

Thus far, we have shown that by storing the *Markovian*, $\mathbf{x}(t_n)$, state at every measurement time, $t_n, n = 1 \ldots N$, we are able to perform Gaussian-process regression in $O(N)$ time and then query for other times of interest $\mathbf{x}(\tau)$, all without approximation. Given the ability to interpolate, storing the state at every measurement time may be excessive, especially in a scenario where the measurement rate is high in comparison to the smoothness of the robot kinematics (e.g. a 1000 Hz IMU or individually timestamped lidar measurements, mounted on a slow indoor platform).

Tong et al. (2013) discuss a scheme to remove some of the measurement times from the initial solve, which further reduces computational cost with some loss of accuracy. By estimating $\mathbf{x}$ at only some *keytimes*, $t_k, k = 0 \ldots K$, which may or may not align with some subset of the measurement times, the interpolation equation can be used to modify our measurement model as follows,

$$
\begin{aligned}
\mathbf{y}_n &= \mathbf{g}\left(\mathbf{x}_{\mathrm{op}}(t_n)\right) \\
&= \mathbf{g}\left(\check{\mathbf{x}}(t_n) + \begin{bmatrix} \mathbf{\Lambda}(t_n) & \mathbf{\Psi}(t_n) \end{bmatrix} \begin{bmatrix} \mathbf{x}_{\mathrm{op},k} - \check{\mathbf{x}}_k \\ \mathbf{x}_{\mathrm{op},k+1} - \check{\mathbf{x}}_{k+1} \end{bmatrix}\right),
\end{aligned} \quad (31)
$$

where $t_k \leq t_n < t_{k+1}$. The matrices $\mathbf{\Lambda}(t_n)$ and $\mathbf{\Psi}(t_n)$ are constructed according to (28), where the measurement time, t_n, is now the query time and the bounding times (previously t_n and t_{n+1}) become the keytimes t_k and t_{k+1}. The effect on $\mathbf{G}$ is that each block row now has two adjacent non-zero block columns, rather than one. This causes the structure of $\mathbf{G}^T \mathbf{R}^{-1} \mathbf{G}$ to change from being block-diagonal to block-tridiagonal; the structure of $\mathbf{G}^T \mathbf{R}^{-1} \mathbf{G}$ is of particular importance because it directly affects the complexity of solving (19). Fortunately, the complexity is unaffected because our prior term is also block-tridiagonal.

An important intuition is that the interpolated state at some measurement time, t_n, depends only on the state estimates, $\mathbf{x}_{\mathrm{op},k}$ and $\mathbf{x}_{\mathrm{op},k+1}$, and the prior terms, $\check{\mathbf{x}}(t)$ and $\check{\mathbf{P}}(t, t')$. Therefore, the effect of estimating $\mathbf{x}$ at some reduced number of times is that we obtain a smoothed solution; any details provided by high frequency measurements between two times of interest, t_k and t_{k+1}, are lost. Although this detail information is smoothed over, there are obvious computational savings in having a smaller state and a subtle benefit regarding the prevention of overfitting measurements. With respect to fitting, the glaring issue with this scheme is that there is not a principled method to determine an appropriate spacing for the *keytimes*, t_k, such that we could guarantee a bound on the loss of accuracy. Learning from the parametric continuous-time estimation schemes, which suffer from a similar issue, it is reasonable to start with some uniform spacing and add additional *keytimes* based on the results of a normalized-innovation-squared test

between each pair of *keytimes* (Oth et al., 2013). Some experimentation is necessary to better understand which approach to state discretization is best in which situation.

3.2 Nonlinear, Time-Varying Stochastic Differential Equations

In reality, most systems are inherently nonlinear and cannot be accurately described by a LTV SDE in the form of (6). Moving forward, we show how our results concerning sparsity can be applied to nonlinear, time-varying (NTV) stochastic differential equations (SDE) of the form

$$
\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), \mathbf{w}(t)), \quad (32)
$$

where $\mathbf{f}(\cdot)$ is a nonlinear function, $\mathbf{x}(t)$ is the state, $\mathbf{u}(t)$ is a (known) exogenous input, and $\mathbf{w}(t)$ is white process noise. To perform GP regression with a nonlinear process model, we begin by linearizing the SDE about a continuous-time operating point $\mathbf{x}_{\mathrm{op}}(t)$,

$$
\begin{aligned}
\dot{\mathbf{x}}(t) &= \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), \mathbf{w}(t)) \\
&\approx \mathbf{f}(\mathbf{x}_{\mathrm{op}}(t), \mathbf{u}(t), \mathbf{0}) + \mathbf{F}(t)(\mathbf{x}(t) - \mathbf{x}_{\mathrm{op}}(t)) + \mathbf{L}(t)\mathbf{w}(t).
\end{aligned} \quad (33)
$$

where

$$
\mathbf{F}(t) = \left. \frac{\partial \mathbf{f}}{\partial \mathbf{x}} \right|_{\mathbf{x}_{\mathrm{op}}(t), \mathbf{u}(t), \mathbf{0}}, \quad \mathbf{L}(t) = \left. \frac{\partial \mathbf{f}}{\partial \mathbf{w}} \right|_{\mathbf{x}_{\mathrm{op}}(t), \mathbf{u}(t), \mathbf{0}}. \quad (34)
$$

Setting

$$
\mathbf{v}(t) = \mathbf{f}(\mathbf{x}_{\mathrm{op}}(t), \mathbf{u}(t), \mathbf{0}) - \mathbf{F}(t)\mathbf{x}_{\mathrm{op}}(t) \quad (35)
$$

lets us rewrite (33) in the familiar LTV SDE form,

$$
\dot{\mathbf{x}}(t) \approx \mathbf{F}(t)\mathbf{x}(t) + \mathbf{v}(t) + \mathbf{L}(t)\mathbf{w}(t), \quad (36)
$$

where $\mathbf{F}(t)$, $\mathbf{v}(t)$, and $\mathbf{L}(t)$ are known functions of time, since $\mathbf{x}_{\mathrm{op}}(t)$ is known. Setting the operating point, $\mathbf{x}_{\mathrm{op}}(t)$, to our best guess of the underlying trajectory at each iteration of GP regression, we note a similarity in nature to the discrete-time, recursive method of Bell (1994); in essence, our approach offers a continuous-discrete version of the Gauss-Newton estimator, using the Gaussian-process-regressor type of approximate bridging between the measurement times.

3.2.1 Mean and Covariance Function

Although the equations for calculating the mean, $\check{\mathbf{x}}$, and covariance, $\check{\mathbf{P}}$, remain the same as in (9) and (12), there are a few algorithmic differences and issues that arise due to the new dependence on the continuous-time operating point, $\mathbf{x}_{\mathrm{op}}(t)$. An obvious difference in contrast to the GP regression using a LTV SDE is that we now must recalculate $\check{\mathbf{x}}$

and $\check{\mathbf{P}}$ at each iteration of the optimization (since the linearization point is updated). The main algorithmic issue that presents itself is that the calculation of $\check{\mathbf{x}}$ and $\check{\mathbf{P}}$ require integrations involving $\mathbf{F}(t)$, $\mathbf{v}(t)$, and $\mathbf{L}(t)$, which in turn require the evaluation of $\mathbf{x}_{\text{op}}(t)$ over the time period $t \in [t_0, t_N]$. Since an estimate of the posterior mean, $\mathbf{x}_{\text{op}}$, is only stored at times of interest, we must make use of the efficient (for our particular choice of process model) GP interpolation equation derived in Section 3.1.3,

$$\mathbf{x}_{\text{op}}(\tau) = \check{\mathbf{x}}(\tau) + \check{\mathbf{P}}(\tau)\check{\mathbf{P}}^{-1}(\mathbf{x}_{\text{op}} - \check{\mathbf{x}}). \tag{37}$$

The problem is that the above interpolation depends again on $\check{\mathbf{x}}$ and $\check{\mathbf{P}}$, which are the prior variables for which we want to solve. To rectify this circular dependence, we take advantage of the iterative nature of GP regression and choose to evaluate (37) using the values of $\check{\mathbf{x}}(\tau)$, $\check{\mathbf{x}}$, and $\check{\mathbf{P}}(\tau)\check{\mathbf{P}}^{-1}$ from the previous iteration.

Another issue with nonlinear process models is that identifying an analytical expression for the state transition matrix, $\mathbf{\Phi}(t, s)$ (which is dependent on the form of $\mathbf{F}(t)$), can be very challenging. Fortunately, the transition matrix can also be calculated numerically via the integration of the normalized fundamental matrix, $\mathbf{\Upsilon}(t)$, where

$$\dot{\mathbf{\Upsilon}}(t) = \mathbf{F}(t)\mathbf{\Upsilon}(t), \quad \mathbf{\Upsilon}(0) = \mathbf{1}. \tag{38}$$

Storing $\mathbf{\Upsilon}(t)$ at times of interest, the transition matrix can then be computed using

$$\mathbf{\Phi}(t, s) = \mathbf{\Upsilon}(t)\mathbf{\Upsilon}(s)^{-1}. \tag{39}$$

In contrast to the LTV SDE system, using a nonlinear process model causes additional computational costs (primarily due to numerical integrations), but complexity remains linear in the length of the trajectory (at each iteration), and therefore continues to be computationally tractable.

3.2.2 Querying the Trajectory

In this section, we discuss the algorithmic details of the GP interpolation procedure as it pertains to a NTV SDE process model. Recall the standard linear GP interpolation formulas presented in (21), for a single query time, $t_n \leq \tau < t_{n+1}$:

$$\hat{\mathbf{x}}(\tau) = \check{\mathbf{x}}(\tau) + \check{\mathbf{P}}(\tau)\check{\mathbf{P}}^{-1}(\hat{\mathbf{x}} - \check{\mathbf{x}}),$$
$$\hat{\mathbf{P}}(\tau, \tau) = \check{\mathbf{P}}(\tau, \tau) + \check{\mathbf{P}}(\tau)\check{\mathbf{P}}^{-1}\left(\hat{\mathbf{P}} - \check{\mathbf{P}}\right)\check{\mathbf{P}}^{-T}\check{\mathbf{P}}(\tau)^T.$$

The final iteration of GP regression (where $\hat{\mathbf{x}} = \mathbf{x}_{\text{op}}$), provides values for $\check{\mathbf{x}}$, $\check{\mathbf{P}}$, $\hat{\mathbf{x}}$, and $\hat{\mathbf{P}}$; however, obtaining values for $\check{\mathbf{x}}(\tau)$, $\check{\mathbf{P}}(\tau)\check{\mathbf{P}}^{-1}$ (recall sparse structure in (28)), and $\hat{\mathbf{P}}(\tau, \tau)$ is not trivial. The suggestion made in Section 3.2.1 was to use values from the previous (or in this case, final) iteration. Thus far, the method of storing these continuous-time functions has been left ambiguous.

The naive way to store $\check{\mathbf{x}}(\tau)$, $\check{\mathbf{P}}(\tau)\check{\mathbf{P}}^{-1}$, and $\check{\mathbf{P}}(\tau, \tau)$ is to keep the values at all numerical integration timesteps; the memory requirement of this is proportional to the length of the trajectory. During the optimization procedure this naive method may in fact be preferable as it reduces computation time in lieu of additional storage (which is fairly cheap using current technology). For long-term storage, a method that uses a smaller memory footprint (at the cost of additional computation) may be desirable. The remainder of this section will focus on identifying the minimal storage requirements, such that queries remain $O(1)$ complexity.

We begin by examining the mean function; substituting (22) and (35), into the GP interpolation formula above:

$$\hat{\mathbf{x}}(\tau) = \mathbf{\Phi}(\tau, t_n)\check{\mathbf{x}}_n + \int_{t_n}^{\tau} \mathbf{\Phi}(\tau, s)\big(\mathbf{f}(\hat{\mathbf{x}}(s), \mathbf{u}(s), \mathbf{0}) \\ - \mathbf{F}(s)\hat{\mathbf{x}}(s)\big)\, ds + \check{\mathbf{P}}(\tau)\check{\mathbf{P}}^{-1}(\hat{\mathbf{x}} - \check{\mathbf{x}}). \tag{41}$$

It is straightforward to see how $\hat{\mathbf{x}}(\tau)$ can be simultaneously numerically integrated with the normalized fundamental matrix from (38), as long as we can evaluate the term $\check{\mathbf{P}}(\tau)\check{\mathbf{P}}^{-1}$. Although we chose to examine the mean function, a similar conclusion can be drawn by examining the covariance function in (23). Recalling the sparse structure of $\check{\mathbf{P}}(\tau)\check{\mathbf{P}}^{-1}$ (see (28)) for an LTV SDE process model, where

$$\mathbf{\Lambda}(\tau) = \mathbf{\Phi}(\tau, t_n) - \mathbf{\Psi}(\tau)\mathbf{\Phi}(t_{n+1}, t_n),$$
$$\mathbf{\Psi}(\tau) = \mathbf{Q}_\tau \mathbf{\Phi}(t_{n+1}, \tau)^T \mathbf{Q}_{n+1}^{-1},$$

we are able to draw two conclusions. First, in the case that an analytical expression for $\mathbf{\Phi}(t, s)$ is unavailable, we must store $\mathbf{\Upsilon}(t_n)$ at the times of interest $t_n, n = 1 \dots N$, since any numerical integration will involve 'future' values of $\mathbf{\Upsilon}(t)$ (via the evaluation of $\mathbf{\Phi}(t_{n+1}, t_n)$, $\mathbf{\Phi}(t_{n+1}, \tau)$, and $\mathbf{Q}_{n+1}^{-1}$). Second, in the case that $\mathbf{L}(t)$ is a time-varying matrix, we must store $\mathbf{Q}_n^{-1}, n = 1 \dots N$, since the evaluation of $\mathbf{Q}_n^{-1}$, requires $\mathbf{L}(s)$ (evaluated at $\hat{\mathbf{x}}(s)$) over the time period $[t_{n-1}, t_n]$. The memory requirements of this alternative are still proportional to the length of the trajectory, but are greatly reduced in contrast to the naive method; the added cost is that any new query requires numerical integration from the nearest time t_n to the query time τ (which is of order $O(1)$).

3.3 Training the Hyperparameters

As with any GP regression, we have *hyperparameters* associated with our covariance function, namely $\mathbf{Q}_C$, which affect the smoothness and length scale of the class of functions we are considering as motion priors. The covariances of the measurement noises can also be unknown or uncertain. The standard approach to selecting these parameters is to

use a training dataset (with ground-truth), and perform optimization using the log marginal likelihood (log-evidence) or its approximation as the objective function (Rasmussen and Williams, 2006). We begin with the marginal likelihood equation,

$$\log p(\mathbf{y}|\mathbf{Q}_C) = -\frac{1}{2}(\mathbf{y} - \check{\mathbf{x}})^T \mathbf{P}_w^{-1}(\mathbf{y} - \check{\mathbf{x}}) \tag{42}$$
$$- \frac{1}{2}\log|\mathbf{P}_w| - \frac{n}{2}\log 2\pi,$$

$$\mathbf{P}_w = \check{\mathbf{P}}(\mathbf{Q}_C) + \sigma_w^2 \mathbf{1}, \tag{43}$$

where $\mathbf{y}$ is a stacked vector of state observations (ground-truth measurements) with additive noise $\mathcal{N}(\mathbf{0}, \sigma_w^2 \mathbf{1})$, $\check{\mathbf{x}}$ is a stacked vector of the mean equation evaluated at the observation times, $t_w, w = 1 \ldots W$, and $\check{\mathbf{P}}$ is the covariance matrix associated with $\check{\mathbf{x}}$ and generated using the hyperparameters $\mathbf{Q}_C$. Taking partial derivatives of the marginal likelihood with respect to the hyperparameters, we get

$$\frac{\partial}{\partial \mathbf{Q}_{C_{ij}}} \log p(\mathbf{y}|\mathbf{Q}_C) = \frac{1}{2}(\mathbf{y} - \check{\mathbf{x}})^T \mathbf{P}_w^{-1} \frac{\partial \mathbf{P}_w}{\partial \mathbf{Q}_{C_{ij}}} \mathbf{P}_w^{-1}(\mathbf{y} - \check{\mathbf{x}})$$
$$- \frac{1}{2}\operatorname{tr}\left(\mathbf{P}_w^{-1} \frac{\partial \mathbf{P}_w}{\partial \mathbf{Q}_{C_{ij}}}\right), \tag{44}$$

where we have used that $\frac{\partial \mathbf{P}_w^{-1}}{\partial \mathbf{Q}_{C_{ij}}} = -\mathbf{P}_w^{-1} \frac{\partial \mathbf{P}_w}{\partial \mathbf{Q}_{C_{ij}}} \mathbf{P}_w^{-1}$.

The typical complexity of hyperparameter training is bottlenecked at $O(W^3)$ due to the inversion of $\mathbf{P}_w$ (which is typically dense). Given $\mathbf{P}_w^{-1}$, the complexity is then typically bottlenecked at $O(W^2)$ due to the calculation of $\frac{\partial \mathbf{P}_w}{\partial \mathbf{Q}_{C_{ij}}}$. Fortunately, in the present case, the computation of the log marginal likelihood can also be done efficiently due to the sparseness of the inverse kernel matrix, $\check{\mathbf{P}}^{-1}$. Despite the addition of observation noise, $\sigma_w^2 \mathbf{1}$, causing $\mathbf{P}_w^{-1}$ to be a dense matrix, we are still able to take advantage of our sparsity using the Sherman-Morrison-Woodbury identity:

$$\mathbf{P}_w^{-1} = \check{\mathbf{P}}^{-1} - \check{\mathbf{P}}^{-1}\left(\check{\mathbf{P}}^{-1} + \frac{1}{\sigma_w^2}\mathbf{1}\right)^{-1}\check{\mathbf{P}}^{-1} = \mathbf{F}^{-T}\mathbf{Q}_w^{-1}\mathbf{F}^{-1}, \tag{45}$$

where we define

$$\mathbf{Q}_w^{-1} = \left(\mathbf{Q}^{-1} - \mathbf{Q}^{-1}\mathbf{F}^{-1}\left(\check{\mathbf{P}}^{-1} + \frac{1}{\sigma_w^2}\mathbf{1}\right)^{-1}\mathbf{F}^{-T}\mathbf{Q}^{-1}\right). \tag{46}$$

Although computing $\mathbf{P}_w^{-1}$ explicitly is of order $O(W^2)$, we note that the product with a vector, $\mathbf{P}_w^{-1}\mathbf{v}$, can be computed in $O(W)$ time; this is easily observable given that $\mathbf{Q}^{-1}$ is block-diagonal, $\mathbf{F}^{-1}$ is lower block-bidiagonal, and the product $(\check{\mathbf{P}}^{-1} + \frac{1}{\sigma_w^2}\mathbf{1})^{-1}\mathbf{v}$ can be computed in $O(W)$ time using Cholesky decomposition, because $(\check{\mathbf{P}}^{-1} + \frac{1}{\sigma_w^2}\mathbf{1})$ is

block-tridiagonal. While the two terms in (44) can be combined into a single trace function, it is simpler to study their computational complexity separately. Starting with the first term, we have

$$\frac{1}{2}(\mathbf{y} - \check{\mathbf{x}})^T \mathbf{P}_w^{-1} \frac{\partial \mathbf{P}_w}{\partial \mathbf{Q}_{C_{ij}}} \mathbf{P}_w^{-1}(\mathbf{y} - \check{\mathbf{x}})$$
$$= \frac{1}{2}(\mathbf{y} - \check{\mathbf{x}})^T \mathbf{F}^{-T}\mathbf{Q}_w^{-1}\frac{\partial \mathbf{Q}}{\partial \mathbf{Q}_{C_{ij}}}\mathbf{Q}_w^{-1}\mathbf{F}^{-1}(\mathbf{y} - \check{\mathbf{x}}), \tag{47}$$

where

$$\frac{\partial \mathbf{Q}}{\partial \mathbf{Q}_{C_{ij}}} = \operatorname{diag}\left(\mathbf{0}, \frac{\partial \mathbf{Q}_1}{\partial \mathbf{Q}_{C_{ij}}}, \ldots, \frac{\partial \mathbf{Q}_N}{\partial \mathbf{Q}_{C_{ij}}}\right), \tag{48}$$

$$\frac{\partial \mathbf{Q}_n}{\partial \mathbf{Q}_{C_{ij}}} = \int_{t_{n-1}}^{t_n} \mathbf{\Phi}(t_n, s)\mathbf{L}(s)\frac{\partial \mathbf{Q}_C}{\partial \mathbf{Q}_{C_{ij}}}\mathbf{L}(s)^T\mathbf{\Phi}(t_n, s)^T \, ds,$$

$$= \int_{t_{n-1}}^{t_n} \mathbf{\Phi}(t_n, s)\mathbf{L}(s)\mathbf{1}_{i,j}\mathbf{L}(s)^T\mathbf{\Phi}(t_n, s)^T \, ds, \tag{49}$$

and $\mathbf{1}_{i,j}$ denotes a projection matrix with a 1 at the ith row and j^{th} column. Taking advantage of the sparse matrices and previously mentioned fast matrix-vector products, it is clear that (47) can be computed in $O(W)$ time (in contrast to the typical $O(W^3)$ time).

Taking a look at the second term, we have that

$$\frac{1}{2}\operatorname{tr}\left(\mathbf{P}_w^{-1}\frac{\partial \mathbf{P}_w}{\partial \mathbf{Q}_{C_{ij}}}\right)$$
$$= \frac{1}{2}\operatorname{tr}\left(\left(\mathbf{F}^{-T}\mathbf{Q}_w^{-1}\mathbf{F}^{-1}\right)\left(\mathbf{F}\frac{\partial \mathbf{Q}}{\partial \mathbf{Q}_{C_{ij}}}\mathbf{F}^T\right)\right)$$
$$= \frac{1}{2}\operatorname{tr}\left(\frac{\partial \mathbf{Q}}{\partial \mathbf{Q}_{C_{ij}}}\mathbf{Q}_w^{-1}\right), \tag{50}$$

which can only be computed in $O(W^2)$, due to the form of $\mathbf{Q}_w^{-1}$. In general, the total complexity of training for this sparse class of prior is then bottlenecked at $O(W^2)$. A complexity of $O(W)$ can only be achieved by ignoring the additive measurement noise, $\sigma_w^2 \mathbf{1}$; revisiting the second term of (44), and setting $\mathbf{P}_w = \check{\mathbf{P}}$, we find that

$$\frac{1}{2}\operatorname{tr}\left(\check{\mathbf{P}}^{-1}\frac{\partial \check{\mathbf{P}}}{\partial \mathbf{Q}_{C_{ij}}}\right) = \frac{1}{2}\operatorname{tr}\left(\frac{\partial \mathbf{Q}}{\partial \mathbf{Q}_{C_{ij}}}\mathbf{Q}^{-1}\right), \tag{51}$$

which can be computed in time $O(W)$. The effect of ignoring the measurement noise, $\sigma_w^2 \mathbf{1}$, is that the trained hyperparameters will result in an underconfident prior; the degree of this underconfidence depends on the magnitude of the noise we are choosing to ignore. If accurate ground-truth measurements are available and the size of the training dataset is very large, this approximation may be beneficial.

3.4 Complexity

We conclude this section with a brief discussion of the time complexity of the overall algorithm when exploiting the sparse structure. If we have N measurement times and want to query the trajectory at J additional times of interest, the complexity of the resulting algorithm using GP regression with *any* linear (or nonlinear), time-varying process model driven by white noise will be $O(N + J)$. This is broken into the two major steps as follows. The initial solution to find $\mathbf{x}_{\text{op}}$ (at the measurement times) can be done in $O(N)$ time (per iteration) owing to the block-tridiagonal structure discussed earlier. Then, the cost of the queries at J other times of interest is $O(J)$ since each individual query is $O(1)$. Clearly, $O(N+J)$ is a big improvement over the $O(N^3 + N^2 J)$ cost when we did not exploit the sparse structure of the problem.

4 Mobile Robot Example

4.1 Linear Constant-Velocity GP Prior

We will demonstrate the advantages of the sparse structure through an example employing the 'constant-velocity' prior, $\ddot{\mathbf{p}}(t) = \mathbf{w}(t)$. This can be expressed as a linear, time-invariant SDE of the form in (6) with

$$\mathbf{x}(t) = \begin{bmatrix} \mathbf{p}(t) \\ \dot{\mathbf{p}}(t) \end{bmatrix}, \ \mathbf{F}(t) = \begin{bmatrix} \mathbf{0} & \mathbf{1} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}, \ \mathbf{v}(t) = \mathbf{0}, \ \mathbf{L}(t) = \begin{bmatrix} \mathbf{0} \\ \mathbf{1} \end{bmatrix}, \tag{52}$$

where $\mathbf{p}(t) = \begin{bmatrix} x(t) & y(t) & \theta(t) \end{bmatrix}^T$ is the pose and $\dot{\mathbf{p}}(t)$ is the pose rate. In this case, the transition function is

$$\boldsymbol{\Phi}(t, s) = \begin{bmatrix} \mathbf{1} & (t - s)\mathbf{1} \\ \mathbf{0} & \mathbf{1} \end{bmatrix}, \tag{53}$$

which can be used to construct $\mathbf{F}$ (or $\mathbf{F}^{-1}$ directly). As we will be doing a STEAM example, we will constrain the first trajectory state to be $\mathbf{x}(t_0) = \mathbf{0}$ and so will have no need for $\check{\mathbf{x}}_0$ and $\check{\mathbf{P}}_0$. For $n = 1 \dots N$, we have $\mathbf{v}_n = \mathbf{0}$ and

$$\mathbf{Q}_n = \begin{bmatrix} \frac{1}{3}\Delta t_n^3 \mathbf{Q}_C & \frac{1}{2}\Delta t_n^2 \mathbf{Q}_C \\ \frac{1}{2}\Delta t_n^2 \mathbf{Q}_C & \Delta t_n \mathbf{Q}_C \end{bmatrix}, \tag{54}$$

with $\Delta t_n = t_n - t_{n-1}$. The inverse blocks are

$$\mathbf{Q}_n^{-1} = \begin{bmatrix} 12\Delta t_n^{-3}\mathbf{Q}_C^{-1} & -6\Delta t_n^{-2}\mathbf{Q}_C^{-1} \\ -6\Delta t_n^{-2}\mathbf{Q}_C^{-1} & 4\Delta t_n^{-1}\mathbf{Q}_C^{-1} \end{bmatrix}, \tag{55}$$

so we can build $\mathbf{Q}^{-1}$ directly. We now have everything we need to represent the prior: $\mathbf{F}^{-1}$, $\mathbf{Q}^{-1}$, and $\mathbf{v} = \mathbf{0}$, which can be used to construct $\check{\mathbf{P}}^{-1}$.

We will also augment the trajectory with a set of L landmarks, $\boldsymbol{\ell}$, into a combined state, $\mathbf{z}$, in order to consider the STEAM problem:

$$\mathbf{z} = \begin{bmatrix} \mathbf{x} \\ \boldsymbol{\ell} \end{bmatrix}, \quad \boldsymbol{\ell} = \begin{bmatrix} \boldsymbol{\ell}_1 \\ \vdots \\ \boldsymbol{\ell}_L \end{bmatrix}, \quad \boldsymbol{\ell}_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix}. \tag{56}$$

While others have folded velocity estimation into discrete-time, filter-based SLAM (Davison et al., 2007) and even discrete-time, batch SLAM (Grau and Pansiot, 2012), we are actually proposing something more general than this: the choice of prior tells us what to use for the trajectory states. And, although we solve for the state at a discrete number of measurement times, our setup is based on an underlying continuous-time prior, meaning that we can query it at any time of interest in a principled way.

4.2 Nonlinear Constant-Velocity GP Prior

Since the main source of acceleration in this example is robot-oriented (the actuated wheels), we investigate the use of an alternate 'constant-velocity' prior, with the white noise affecting acceleration in the robot body frame, $\dot{\boldsymbol{\nu}}(t) = \mathbf{w}(t)$. This *nonlinear* prior can be written as,

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), \mathbf{w}(t)) \\ &= \begin{bmatrix} \mathbf{0} & \mathbf{R}_{IB}(\theta(t)) \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \mathbf{x}(t) + \mathbf{u}(t) + \begin{bmatrix} \mathbf{0} \\ \mathbf{1} \end{bmatrix} \mathbf{w}(t), \end{aligned} \tag{57}$$

where

$$\mathbf{R}_{IB}(\theta(t)) = \begin{bmatrix} \cos\theta(t) & -\sin\theta(t) & 0 \\ \sin\theta(t) & \cos\theta(t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{58}$$

is a rotation matrix between the inertial and robot body frame, and the *Markovian* state is $\mathbf{x}(t) = \begin{bmatrix} \mathbf{p}(t)^T & \boldsymbol{\nu}(t)^T \end{bmatrix}^T$, where $\boldsymbol{\nu}(t) = \begin{bmatrix} v(t) & u(t) & \omega(t) \end{bmatrix}^T = \mathbf{R}_{IB}(\theta(t))^T \dot{\mathbf{p}}(t)$ is the robot-oriented velocity (with longitudinal, latitudinal, and rotational components, respectively).

Linearizing about an arbitrary operating point, $\mathbf{x}_{\text{op}}(t)$, the components $\mathbf{F}(t)$, $\mathbf{v}(t)$ and $\mathbf{L}(t)$ from (36) are straightforward to derive. Similarly, to the linear prior example described above, we will define the exogenous input $\mathbf{u}(t) = \mathbf{0}$; however, we note that $\mathbf{v}(t) \neq \mathbf{0}$ (see (35)). Since expressions for $\boldsymbol{\Phi}(t, s)$ and $\mathbf{Q}_n$ are not obvious, we will rely on numerical integration for their evaluation.

4.3 Measurement Models

We will use two types of measurements: range/bearing to landmarks (using a laser rangefinder) and wheel odometry

$$J_{\text{rb},ij} = \frac{1}{2}\mathbf{e}_{\text{rb},ij}^T \mathbf{Q}_{\text{rb},ij}^{-1}\mathbf{e}_{\text{rb},ij}$$
$$\mathbf{e}_{\text{rb},ij} = \mathbf{y}_{ij} - \mathbf{g}_{\text{rb}}(\mathbf{x}(t_i),\boldsymbol{\ell}_j)$$

$$J_{\text{wo},i} = \frac{1}{2}\mathbf{e}_{\text{wo},i}^T \mathbf{Q}_{\text{wo},i}^{-1}\mathbf{e}_{\text{wo},i}$$
$$\mathbf{e}_{\text{wo},i} = \mathbf{y}_i - \mathbf{g}_{\text{wo}}(\mathbf{x}(t_i))$$

Fig. 3 Factor-graph representation of our STEAM problem. There are factors (black dots) for (i) the prior (binary), (ii) the landmark measurements (binary), and (iii) the wheel odometry measurements (unary). Triangles are *trajectory states* (position and velocity, for this prior); the first trajectory state is locked. Hollow circles are landmarks.

(in the form of robot-oriented velocity). The range/bearing measurement model takes the form

$$\mathbf{y}_{ni} = \mathbf{g}_{\text{rb}}(\mathbf{x}(t_n),\boldsymbol{\ell}_i) + \mathbf{n}_{ni}$$
$$= \begin{bmatrix} \sqrt{(x_i - x(t_n))^2 + (y_i - y(t_n))^2} \\ \text{atan2}(y_i - y(t_n), x_i - x(t_n)) \end{bmatrix} + \mathbf{n}_{ni}. \quad (59)$$

The wheel odometry measurement model gives the longitudinal and rotational speeds of the robot, taking the form

$$\mathbf{y}_n = \mathbf{g}_{\text{wo}}(\mathbf{x}(t_n)) + \mathbf{n}_n = \begin{bmatrix} \cos\theta(t_n) & \sin\theta(t_n) & 0 \\ 0 & 0 & 1 \end{bmatrix}\dot{\mathbf{p}}(t_n) + \mathbf{n}_n, \quad (60)$$

for the LTI SDE prior, and

$$\mathbf{y}_n = \mathbf{g}_{\text{wo}}(\mathbf{x}(t_n)) + \mathbf{n}_n = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}\boldsymbol{\nu}(t_n) + \mathbf{n}_n, \quad (61)$$

for the NTV SDE prior. Note that in both cases the velocity information is extracted easily from the state since we are estimating it directly. The Jacobians with respect to the state, $\mathbf{x}(t)$, are straightforward to derive.

4.4 Exploiting Sparsity

Figure 3 shows an illustration of the STEAM problem we are considering. In terms of linear algebra, at each iteration we need to solve a linear system of the form

$$\underbrace{\begin{bmatrix} \mathbf{W}_{xx} & \mathbf{W}_{\ell x}^T \\ \mathbf{W}_{\ell x} & \mathbf{W}_{\ell\ell} \end{bmatrix}}_{\mathbf{W}}\underbrace{\begin{bmatrix} \delta\mathbf{x}^\star \\ \delta\boldsymbol{\ell}^\star \end{bmatrix}}_{\delta\mathbf{z}^\star} = \underbrace{\begin{bmatrix} \mathbf{b}_x \\ \mathbf{b}_\ell \end{bmatrix}}_{\mathbf{b}}, \quad (62)$$

which retains exploitable structure despite introducing landmarks to the state (Brown, 1958). In particular, $\mathbf{W}_{xx}$ is block-tridiagonal (due to our GP prior) and $\mathbf{W}_{\ell\ell}$ is block-diagonal; the sparsity of the off-diagonal block, $\mathbf{W}_{\ell x}$, depends on the specific landmark observations. We reiterate

the fact that if we marginalize out the $\dot{\mathbf{p}}(t_n)$ (or $\boldsymbol{\nu}(t_n)$) variables and keep only the $\mathbf{p}(t_n)$ variables to represent the trajectory, the $\mathbf{W}_{xx}$ block becomes dense (for these priors); this is precisely the approach of Tong et al. (2013).

To solve (62) efficiently, we can begin by either exploiting the sparsity of $\mathbf{W}_{xx}$ or of $\mathbf{W}_{\ell\ell}$. Since each trajectory variable represents a unique measurement time (range/bearing or odometry), there are potentially a lot more trajectory variables than landmark variables, $L \ll M$, so we will exploit $\mathbf{W}_{xx}$.

We use a sparse (lower-upper) Cholesky decomposition:

$$\underbrace{\begin{bmatrix} \mathbf{V}_{xx} & \mathbf{0} \\ \mathbf{V}_{\ell x} & \mathbf{V}_{\ell\ell} \end{bmatrix}}_{\mathbf{V}}\underbrace{\begin{bmatrix} \mathbf{V}_{xx}^T & \mathbf{V}_{\ell x}^T \\ \mathbf{0} & \mathbf{V}_{\ell\ell}^T \end{bmatrix}}_{\mathbf{V}^T} = \underbrace{\begin{bmatrix} \mathbf{W}_{xx} & \mathbf{W}_{\ell x}^T \\ \mathbf{W}_{\ell x} & \mathbf{W}_{\ell\ell} \end{bmatrix}}_{\mathbf{W}} \quad (63)$$

We first decompose $\mathbf{V}_{xx}\mathbf{V}_{xx}^T = \mathbf{W}_{xx}$, which can be done in $O(N)$ time owing to the block-tridiagonal sparsity. The resulting $\mathbf{V}_{xx}$ will have only the main block-diagonal and the one below it non-zero. We can then solve $\mathbf{V}_{\ell x}\mathbf{V}_{xx}^T = \mathbf{W}_{\ell x}$ for $\mathbf{V}_{\ell x}$ in $O(LN)$ time. Finally, we decompose $\mathbf{V}_{\ell\ell}\mathbf{V}_{\ell\ell}^T = \mathbf{W}_{\ell\ell} - \mathbf{V}_{\ell x}\mathbf{V}_{\ell x}^T$, which we can do in $O(L^3 + L^2N)$ time. This completes the decomposition in $O(L^3 + L^2N)$ time. We then perform the standard forward-backward passes, ensuring to exploit the sparsity: first solve $\mathbf{V}\mathbf{d} = \mathbf{b}$ for $\mathbf{d}$, then $\mathbf{V}^T\delta\mathbf{z}^\star = \mathbf{d}$ for $\delta\mathbf{z}^\star$, both in $O(L^2 + LN)$ time. Note, this approach does not marginalize out any variables during the solve, as this can ruin the sparsity (i.e., we avoid inverting $\mathbf{W}_{xx}$). The whole solve is $O(L^3 + L^2N)$.

At each iteration, we update the state, $\bar{\mathbf{z}} \leftarrow \bar{\mathbf{z}} + \delta\mathbf{z}^\star$, and iterate to convergence. Finally, we query the trajectory at J other times of interest using the GP interpolation discussed earlier. The whole procedure is then $O(L^3 + L^2N + J)$, including the extra queries.

Due to the addition of landmarks, the cost of a STEAM problem must be either be of order $O(L^3 + L^2N + J)$ or $O(N^3 + N^2L + J)$, depending on the way we choose to exploit $\mathbf{W}_{xx}$ or $\mathbf{W}_{\ell\ell}$. The state reduction scheme presented in Section 3.1.4 becomes very attractive when both the number of measurements and landmarks are very high; estimating the state at K *keytimes*, $K < N$, and exploiting $\mathbf{W}_{\ell\ell}$, the procedure becomes order $O(K^3 + K^2L + N + J)$, which is the same complexity as a traditional discrete-time SLAM problem (with the addition of the J query times).

4.5 Experiment

For experimental validation, we employed the same mobile robot dataset as used by Tong et al. (2013). This dataset consists of a mobile robot equipped with a laser rangefinder driving in an indoor, planar environment amongst a forest of 17 plastic-tube landmarks. The odometry and landmark measurements are provided at a rate of 1Hz, and additional

(a) *GP-Traj-Sparse-LTI* trajectory sample.

(b) *GP-Traj-Sparse-NTV* trajectory sample.

Fig. 4 The smooth and continuous trajectories and 3σ covariance envelope estimates produced by the *GP-Traj-Sparse* estimators (both linear and nonlinear) over a short segment of the dataset.

trajectory queries are computed at a rate of 10Hz after estimator convergence. Ground-truth for the robot trajectory and landmark positions is provided by a Vicon motion capture system.

We implemented three estimators for comparison. The first was the algorithm described by Tong et al. (2013), *GP-Pose-Dense*, the second was a naive version of our estimator, *GP-Traj-Dense*, based on the LTI SDE prior described in Section 4.1, but did not exploit sparsity, and the third was a full implementations of our estimator, *GP-Traj-Sparse*, that exploited the sparsity structure as described in this paper. The final estimator has two variants, *GP-Traj-Sparse-LTI* and *GP-Traj-Sparse-NTV*, based on the LTI SDE and NTV SDE priors described in Sections 4.1 and 4.2, respectively; as the first two estimators, *GP-Pose-Dense* and *GP-Traj-Dense*, are only of interest with regard to computational performance, they each implement only the LTI 'constant-velocity' prior.

For this experiment, we obtained $\mathbf{Q}_C$ for both the LTI and NTV priors by modelling it as a diagonal matrix and taking the data-driven training approach using log marginal likelihood (with ground-truth measurements) described in Section 3.3.

Though the focus of the exactly sparse Gaussian process priors is to demonstrate the significant reductions in computational cost, we provide Figure 4 to illustrate the smooth trajectory estimates we obtained from the continuous-time formulation. While the algorithms differed in their number of degrees of freedom and types of their estimated states, their overall accuracies were similar for this dataset. The *GP-Traj-Sparse-NTV* algorithm differed slightly from the others; qualitatively, we found that the trajectory estimated by the *GP-Traj-Sparse-NTV* algorithm more accurately matched the local shape of the ground-truth on many occasions, such as the ones highlighted by the insets in Fig-

ure 4. Also, it is clear from the plotted 3σ covariance envelope that the estimate from the *GP-Traj-Sparse-NTV* algorithm tends to be more uncertain.

4.5.1 Computational Cost

To evaluate the computational savings of exploiting an exactly sparse GP prior, we implemented all algorithms in Matlab with a 2.4GHz i7 processor and 8GB of 1333MHz DDR3 RAM and timed the computation for segments of the dataset of varying lengths. These results are shown in Figure 5, where we provide the computation time for the individual operations that benefit most from the sparse structure, as well as the overall processing time.

We see that the *GP-Traj-Dense* algorithm is much slower than the original *GP-Pose-Dense* algorithm of Tong et al. (2013). This is because we have reintroduced the velocity part of the state, thereby doubling the number of variables associated with the trajectory. However, once we start exploiting the sparsity with the *GP-Traj-Sparse* methods, the increase in number of variables pays off.

For the *GP-Traj-Sparse* methods, we see in Figure 5(a) that the kernel matrix construction was linear in the number of estimated states. This can be attributed to the fact that we constructed the sparse $\check{\mathbf{P}}^{-1}$ directly. As predicted, the optimization time per iteration was also linear in Figure 5(b), and the interpolation time per additional query was constant regardless of state size in Figure 5(c). Finally, Figure 5(d) shows that the total compute time was also linear.

The additional cost of the *GP-Traj-Sparse-NTV* algorithm over the *GP-Traj-Sparse-LTI* algorithm in kernel construction time is due to the linearization and numerical integration of the prior mean and covariance. The optimization time of the *GP-Traj-Sparse-NTV* algorithm is also affected because the kernel matrix must be reconstructed from a new

(a) Kernel matrix construction time.

(b) Optimization time per iteration.

(c) Interpolation time per additional query time.

(d) Total computation time.

Fig. 5 Plots comparing the compute time (as a function of trajectory length) for the *GP-Pose-Dense* algorithm described by Tong et al. (2013) and three versions of our approach: *GP-Traj-Dense* (does not exploit sparsity) and the two *GP-Traj-Sparse* variants (exploit sparsity). The plots confirm the predicted computational complexities of the various methods; notably, the *GP-Traj-Sparse* estimators have linear cost in trajectory length. Please note the change from a linear to a log scale in the upper part of each plot.

linearization of the prior during every optimization iteration. The *GP-Traj-Sparse-NTV* algorithm also incurs some numerical integration cost in interpolation time, but it is constant and very small.

We also note that the number of iterations for optimization convergence varied for each algorithm. In particular, we found that the *GP-Traj-Sparse* implementations converged in fewer iterations than the other implementations due to the fact that we constructed the inverse kernel matrix directly, which resulted in greater numerical stability. The *GP-Traj-Sparse* approaches clearly outperform the other algorithms in terms of computational cost.

4.5.2 Increasing Nonlinearity

In a problem with fairly accurate and high-rate measurements, both the *GP-Traj-Sparse-LTI* and *GP-Traj-Sparse-NTV* estimators provide similar accuracy. In order to expose the benefit of a nonlinear prior based on the expected motion of the vehicle, we increase the nonlinearity of the problem by reducing measurement frequency.

The result of varying range measurement frequency, with and without the use of odometry measurements, is shown in Figure 6. In general, it is clear that as the interval between range measurements is increased, the *GP-Traj-Sparse-NTV* estimator is able to produce a more accurate estimate of the continuous-time pose (translation and rotation) than the *GP-Traj-Sparse-LTI* estimator.

In the case that the 1 Hz odometry measurements are available, as seen in Figure 6(a), the difference in the rotation estimates is small, because the *GP-Traj-Sparse-LTI* estimator has a good amount of information about its heading;

however, in the case that the odometry measurements are unavailable, as seen in Figure 6(b), the advantage of the nonlinear prior implemented by the *GP-Traj-Sparse-NTV* estimator is prominent with respect to the rotational estimate.

In order to gain some qualitative intuition about how the continuous-time pose estimates are affected by the reduction of measurements, Figure 7 shows the trajectories for the same small subsection as presented in Figure 4; the estimates used an interval between range measurements of 7 seconds and are shown with and without the use of odometry measurements. In both plots, it is clear that the *GP-Traj-Sparse-NTV* estimator matches the ground-truth more closely, as previously indicated by the error plots in Figure 6.

5 Discussion and Future Work

It is worth elaborating on a few issues. The main reason that the $\mathbf{W}_{xx}$ block is sparse in our approach, as compared to Tong et al. (2013), is that we reintroduced velocity variables that had effectively been marginalized out. This idea of reintroducing variables to regain exact sparsity has been used before by Eustice et al. (2006) in the delayed state filter and by Walter et al. (2007) in the extended information filter. This is a good lesson to heed: the underlying structure of a problem may be exactly sparse, but by marginalizing out variables it appears dense. For us this means we need to use a Markovian trajectory state that is appropriate to our prior.

In much of mobile robotics, odometry measurements are treated more like inputs to the mean of the prior than pure measurements. We believe this is a confusing thing to do as it conflates two sources of uncertainty: the prior over trajectories and the odometry measurement noise. In our frame-

(a) RMS errors using odometry measurements.

(b) RMS errors without using odometry measurements.

Fig. 6 Plots comparing use of the *GP-Traj-Sparse-LTI* and *-NTV* algorithms for an increasingly nonlinear problem; the dataset was made more nonlinear by varying the interval between available range measurements. The results in (a) used the odometry measurements available at 1 Hz, while (b) was made to be even more nonlinear by excluding the use of odometry measurements. The plots show that for a small interval between range measurements, both estimators perform similarly; as the interval was increased, the estimate provided by the nonlinear prior is consistently better in both translation and angular error. Notably, when odometry is available, both estimators are able to achieve a similar orientation performance; the negative effect of removing odometry measurements is more prominent on the linear prior estimator.

(a) With odometry measurements.

(b) Without odometry measurements.

Fig. 7 Plots showing the *GP-Traj-Sparse-LTI* and *-NTV* estimates for the same small trajectory subsection as Figure 4, with an interval between range measurements of 7 seconds. Results in (a) used odometry measurements, while (b) did not.

work, we have deliberately separated these two functions and believe this is easier to work with and understand. We can see these two functions directly in Figure 3, where the prior is made up of binary factors joining consecutive trajectory states, and odometry measurements are unary factors attached to some of the trajectory states (we could have used binary odometry factors but chose to set things up this way due to the fact that we were explicitly estimating velocity).

While our analysis appears to be restricted to a small class of covariance functions, we have only framed our discussions in the context of robotics. Recent developments

from machine learning (Hartikainen and Särkkä, 2010) and signal processing (Särkkä et al., 2013) have shown that it is possible to generate other well-known covariance functions using a LTV SDE (some exactly and some approximately). This means they can be used with our framework. One example is the Matérn covariance family (Rasmussen and Williams, 2006),

$$\check{\mathbf{P}}_{\mathrm{m}}(t,t') = \sigma^2 \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu}}{\ell} |t-t'| \right)^{\nu} K_{\nu} \left(\frac{\sqrt{2\nu}}{\ell} |t-t'| \right) \mathbf{1}$$

$$(64)$$

where σ, ν, $\ell > 0$ are magnitude, smoothness, and length-scale parameters, $\Gamma(\cdot)$ is the gamma function, and $K_\nu(\cdot)$ is the modified Bessel function. For example, if we let

$$\mathbf{x}(t) = \begin{bmatrix} \mathbf{p}(t) \\ \dot{\mathbf{p}}(t) \end{bmatrix}, \tag{65}$$

with $\nu = p + \frac{1}{2}$ with $p = 1$ and use the following SDE:

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} \mathbf{0} & \mathbf{1} \\ -\lambda^2 \mathbf{1} & -2\lambda \mathbf{1} \end{bmatrix} \mathbf{x}(t) + \begin{bmatrix} \mathbf{0} \\ \mathbf{1} \end{bmatrix} \mathbf{w}(t), \tag{66}$$

where $\lambda = \sqrt{2\nu}/\ell$ and $\mathbf{w}(t) \sim \mathcal{GP}\left(\mathbf{0}, \mathbf{Q}_C\, \delta(t - t')\right)$ (our usual white noise) with power spectral density matrix,

$$\mathbf{Q}_C = \frac{2\sigma^2 \pi^{\frac{1}{2}} \lambda^{2p+1} \Gamma(p+1)}{\Gamma(p + \frac{1}{2})} \mathbf{1}, \tag{67}$$

then we have that $\mathbf{p}(t)$ is distributed according to the Matérn covariance family: $\mathbf{p}(t) \sim \mathcal{GP}(\mathbf{0}, \check{\mathbf{P}}_{\mathrm{m}}(t, t'))$ with $p = 1$. Another way to look at this is that passing white noise through LTV SDEs produces particular coloured-noise priors (i.e., not flat across all frequencies).

In terms of future work, we plan to incorporate the dynamics (i.e., kinematics plus Newtonian mechanics) of a robot platform into the GP priors; real sensors do not move arbitrarily through the world as they are usually attached to massive robots and this serves to constrain the motion. Another idea is to incorporate *latent force models* into our GP priors (e.g., see Alvarez et al. (2009) or Hartikainen et al. (2012)). We also plan to look further at the sparsity of STEAM and integrate our work with modern solvers to tackle large-scale problems; this should allow us to exploit more than just the primary sparsity of the problem and do so in an online manner.

6 Conclusion

We have considered continuous-discrete estimation problems where a trajectory is viewed as a one-dimensional *Gaussian process* (GP), with time as the independent variable and measurements acquired at discrete times. Querying the trajectory can be viewed as nonlinear, GP regression. Our main contribution in this paper is to show that this querying can be accomplished very efficiently. To do this, we exploited the Markov property of our GP priors (generated by nonlinear, time-varying stochastic differential equations driven by white noise) to construct an inverse kernel matrix that is sparse. This makes it fast to solve for the state at the measurement times (as is commonly done in vision and robotics) but also at any other time(s) of interest through GP interpolation. Other implications of this sparsity were discussed with respect to hyperparameter training, and including measurements at query times. We also considered

a slight generalization of the SLAM problem, *simultaneous trajectory estimation and mapping* (STEAM), which makes use of a continuous-time trajectory prior and allows us to query the state at any time of interest in an efficient manner. We hope this paper serves to deepen the connection between classical state estimation theory and recent machine learning methods by viewing batch estimation through the lens of Gaussian process regression.

References

Alvarez, M., Luengo, D., and Lawrence, N. (2009). Latent force models. In *Proceedings of the Int. Conf. on Artificial Intelligence and Statistics (AISTATS)*.

Bailey, T. and Durrant-Whyte, H. (2006). SLAM: Part II State of the art. *IEEE RAM*, 13(3):108–117.

Barfoot, T. D., Tong, C. H., and Särkkä, S. (2014). Batch continuous-time trajectory estimation as exactly sparse gaussian process regression. In *Proceedings of Robotics: Science and Systems (RSS)*, Berkeley, USA.

Bell, B. M. (1994). The iterated kalman smoother as a Gauss-Newton method. *SIAM Journal on Optimization*, 4(3):626–636.

Bibby, C. and Reid, I. D. (2010). A hybrid SLAM representation for dynamic marine environments. In *Proc. ICRA*.

Bosse, M. and Zlot, R. (2009). Continuous 3D scan-matching with a spinning 2D laser. In *Proc. ICRA*.

Brown, D. C. (1958). A solution to the general problem of multiple station analytical stereotriangulation. RCA-MTP data reduction tech. report no. 43, Patrick Airforce Base.

Davison, A. J., Reid, I. D., Molton, N. D., and Stasse, O. (2007). MonoSLAM: Real-time single camera SLAM. *IEEE T. PAMI*, 29(6):1052–1067.

Deisenroth, M. P., Turner, R., Huber, M., Hanebeck, U. D., and Rasmussen, C. E. (2012). Robust filtering and smoothing with Gaussian processes. *IEEE T. Automatic Control*, 57:1865–1871.

Dellaert, F. and Kaess, M. (2006). Square Root SAM: Simultaneous localization and mapping via square root information smoothing. *IJRR*, 25(12):1181–1204.

Dong, H. J. and Barfoot, T. D. (2012). Lighting-invariant visual odometry using lidar intensity imagery and pose interpolation. In *Proc. Field and Service Robotics*.

Durrant-Whyte, H. and Bailey, T. (2006). SLAM: Part I Essential algorithms. *IEEE RAM*, 11(3):99–110.

Durrant-Whyte, H. F. (1988). Uncertain geometry in robotics. *IEEE Journal of Robotics and Automation*, 4(1):23–31.

Eustice, R. M., Singh, H., and Leonard, J. J. (2006). Exactly sparse delayed-state filters for view-based SLAM. *IEEE TRO*, 22(6):1100–1114.

Ferris, B., Fox, D., and Lawrence, N. (2007). Wifi-SLAM using Gaussian process latent variable models. In *Proc. IJCAI*.

Ferris, B., Hähnel, D., and Fox, D. (2006). Gaussian processes for signal strength-based localization. In *Proc. RSS*.

Furgale, P. T., Barfoot, T. D., and Sibley, G. (2012). Continuous-time batch estimation using temporal basis functions. In *Proc. ICRA*.

Grau, O. and Pansiot, J. (2012). Motion and velocity estimation of rolling shutter cameras. In *Proceedings of the 9th European Conference on Visual Media Production*, pages 94–98.

Hartikainen, J. and Särkkä, S. (2010). Kalman filtering and smoothing solutions to temporal Gaussian process regression models. In *Proc. of the IEEE Int. Work. on Machine Learning for Signal Processing*.

Hartikainen, J., Seppänen, M., and Särkkä, S. (2012). State-space inference for non-linear latent force models with application to satellite orbit prediction. In *Proc. ICML*.

Hedborg, J., Forssén, P., Felsberg, M., and Ringaby, E. (2012). Rolling shutter bundle adjustment. In *Proc. CVPR*.

Jazwinski, A. H. (1970). *Stochastic Processes and Filtering Theory*. Academic, New York.

Jumarie, G. (1990). Nonlinear filtering. A weighted mean squares approach and a Bayesian one via the maximum entropy principle. *Signal Processing*, 21(4):323–338.

Kaess, M., Johannsson, H., Roberts, R., Ila, V., Leonard, J. J., and Dellaert, F. (2012). iSAM2: Incremental smoothing and mapping using the Bayes tree. *IJRR*, 31(2):217–236.

Kaess, M., Ranganathan, A., and Dellaert, R. (2008). iSAM: Incremental smoothing and mapping. *IEEE TRO*, 24(6):1365–1378.

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45.

Kalman, R. E. and Bucy, R. S. (1961). New results in linear filtering and prediction theory. *Transactions of the ASME–Journal of Basic Engineering*, 83(3):95–108.

Ko, J. and Fox, D. (2009). GP-BayesFilters: Bayesian filtering using Gaussian process prediction and observation models. *Autonomous Robots*, 27(1):75–90.

Ko, J. and Fox, D. (2011). Learning GP-BayesFilters via Gaussian process latent variable models. *Auton. Robots*, 30(1):3–23.

Lawrence, N. (2003). Gaussian process latent variable models for visualization of high dimensional data. In *Proc. NIPS*.

Lindgren, F., Rue, H., and Lindström, J. (2011). An explicit link between Gaussian fields and Gaussian Markov random fields: the stochastic partial differential equation approach. *J. of the Royal Stat. Society: Series B*, 73(4):423–498.

Lovegrove, S., Patron-Perez, A., and Sibley, G. (2013). Spline fusion: A continuous-time representation for visual-inertial fusion with application to rolling shutter cameras. In *Proc. BMVC*.

Lu, F. and Milios, E. (1997). Globally consistent range scan alignment for environment mapping. *Auton. Robots*, 4(4):333–349.

Maybeck, P. S. (1979). *Stochastic Models, Estimation, and Control*, volume 141 of *Mathematics in Science and Engineering*. Academic Press Inc.

Newman, P., Sibley, G., Smith, M., Cummins, M., Harrison, A., Mei, C., Posner, I., Shade, R., Schroeter, D., Murphy, L., Churchill, W., Cole, D., and Reid, I. (2009). Navigating, recognising and describing urban spaces with vision and laser. *IJRR*, 28(11-12):1406–1433.

Oth, L., Furgale, P. T., Kneip, L., and Siegwart, R. (2013). Rolling shutter camera calibration. In *Proc. of The IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, Portland, USA.

Rasmussen, C. E. and Williams, C. K. I. (2006). *Gaussian Processes for Machine Learning*. MIT Press, Cambridge, MA.

Särkkä, S. (2006). *Recursive Bayesian Inference on Stochastic Differential Equations*. PhD thesis, Helsinki Uni. of Technology.

Särkkä, S. (2013). *Bayesian Filtering and Smoothing*. Cambridge University Press.

Särkkä, S. and Sarmavuori, J. (2013). Gaussian filtering and smoothing for continuous-discrete dynamic systems. *Signal Processing*, 93(2):500–510.

Särkkä, S., Solin, A., and Hartikainen, J. (2013). Spatiotemporal learning via infinite-dimensional Bayesian filtering and smoothing: A look at Gaussian process regression through Kalman filtering. *IEEE Signal Processing Magazine*, 30(4):51–61.

Sibley, G., Matthies, L., and Sukhatme, G. (2010). Sliding window filter with application to planetary landing. *Journal of Field Robotics*, 27(5):587–608.

Smith, R. C. and Cheeseman, P. (1986). On the representation and estimation of spatial uncertainty. *IJRR*, 5(4):56–68.

Smith, R. C., Self, M., and Cheeseman, P. (1990). Estimating uncertain spatial relationships in robotics. In Cox, I. J. and Wilfong, G. T., editors, *Autonomous Robot Vehicles*, pages 167–193. Springer Verlag, New York.

Solin, A. and Särkkä, S. (2014). Explicit link between periodic covariance functions and state space models. In *Proceedings of the Int. Conf. on Artificial Intelligence and Statistics (AISTATS)*.

Stengel, R. F. (1994). *Optimal Control and Estimation*. Dover Publications Inc.

Strasdat, H., Montiel, J. M. M., and Davison, A. J. (2010). Real-time monocular SLAM: Why filter? In *Proc. ICRA*.

Thrun, S. and Montemerlo, M. (2006). The graph SLAM algorithm with applications to large-scale mapping of urban structures. *IJRR*, 25(5-6):403–429.

Tong, C. H., Furgale, P., and Barfoot, T. D. (2012). Gaussian process Gauss-Newton: Non-parametric state estimation. In *Proc. of the 9th Conf. on Computer and Robot Vision*, pages 206–213.

Tong, C. H., Furgale, P. T., and Barfoot, T. D. (2013). Gaussian process Gauss-Newton for non-parametric simultaneous localization and mapping. *IJRR*, 32(5):507–525.

Triggs, B., McLauchlan, P., Hartley, R., and Fitzgibbon, A. (2000). Bundle adjustment — A modern synthesis. In Triggs, B., Zisserman, A., and Szeliski, R., editors, *Vision Algorithms: Theory and Practice*, volume 1883 of *Lecture Notes in Computer Science*, pages 298–372. Springer Berlin Heidelberg.

Walter, M. R., Eustice, R. M., and Leonard, J. J. (2007). Exactly sparse extended information filters for feature-based SLAM. *IJRR*, 26(4):335–359.