Autonomous Robots
https://doi.org/10.1007/s10514-018-9770-1

STEAP: simultaneous trajectory estimation and planning

Mustafa Mukadam[1] · **Jing Dong**[1] · **Frank Dellaert**[1] · **Byron Boots**[1]

Received: 1 December 2017 / Accepted: 22 May 2018

Abstract
We present a unified probabilistic framework for simultaneous trajectory estimation and planning. Estimation and planning problems are usually considered separately, however, within our framework we show that solving them simultaneously can be more accurate and efficient. The key idea is to compute the full continuous-time trajectory from start to goal at each time-step. While the robot traverses the trajectory, the history portion of the trajectory signifies the solution to the estimation problem, and the future portion of the trajectory signifies a solution to the planning problem. Building on recent probabilistic inference approaches to continuous-time localization and mapping and continuous-time motion planning, we solve the joint problem by iteratively recomputing the *maximum a posteriori* trajectory conditioned on all available sensor data and cost information. Our approach can contend with high-degree-of-freedom trajectory spaces, uncertainty due to limited sensing capabilities, model inaccuracy, the stochastic effect of executing actions, and can find a solution in real-time. We evaluate our framework empirically in both simulation and on a mobile manipulator.

Keywords Estimation · Motion planning · Replanning · Trajectory optimization · Probabilistic inference · Factor graphs · Gaussian processes

1 Introduction

Trajectory estimation and planning are both important capabilities for autonomous robot navigation. Trajectory estimation is fundamentally backward-looking: the robot estimates a trajectory of previous states that are consistent with a history of noisy and incomplete sensor data. Conversely, planning is fundamentally forward looking: starting from an estimate of its current state, the robot optimizes a trajectory of future states to minimize a cost function and achieve a feasible solution.

In this work, we provide a unified approach to trajectory estimation and planning. Our key insight is that both these problems are inherently variants of trajectory optimization and can therefore be combined to remove the redundancy present in a traditional two step process. The idea is to compute the complete continuous-time trajectory from start to goal at each time-step, such that given the current time-step, the solutions to the estimation problem (history of the trajectory) and the planning problem (future of the trajectory) automatically fall out. Additionally, performing this joint optimization allows information to flow between estimation and planning resulting in mutual benefits. This problem can be quite difficult to solve; the robot must contend with a potentially high-degree-of-freedom (DOF) trajectory space, uncertainty due to limited sensing capabilities, model inaccuracy, and the stochastic effect of executing actions. For the solution to be practical, it must be generated in (faster than) real-time.

We propose a solution to the problem of simultaneous trajectory estimation and planning (STEAP) by viewing trajectory optimization as probabilistic inference and building on recent approaches to continuous-time localization

Mustafa Mukadam and Jing Dong contributed equally to this article.

This is one of several papers published in *Autonomous Robots* comprising the "Special Issue on Robotics Science and Systems".

✉ Mustafa Mukadam
mmukadam3@gatech.edu

Jing Dong
jdong@gatech.edu

Frank Dellaert
frank@cc.gatech.edu

Byron Boots
bboots@cc.gatech.edu

[1] Institute for Robotics and Intelligent Machines, Georgia Institute of Technology, Atlanta, GA, USA

Published online: 24 July 2018

and mapping (Anderson et al. 2015; Yan et al. 2017) and continuous-time motion planning (Dong et al. 2016; Mukadam et al. 2017c). We represent the continuous-time trajectory as a function mapping time to robot states and model the trajectory distribution as a Gaussian process (GP) (Barfoot et al. 2014; Mukadam et al. 2016). At each time-step, we incrementally (re)estimate the entire continuous-time trajectory, as new sensor data or cost information is encountered, by iteratively recomputing the *maximum a posteriori* (MAP) trajectory conditioned on all the available sensor data and cost information. In general, sensor data can include various measurements from proprioception (encoders, inertial measurement unit (IMU), etc) or perception for external information (cameras, LIDARs, etc). On the other hand, cost information can include full or partial information know before (start, goal, map of the environment, etc) or constraints and information encountered during execution (changing goal, orientation constraints on the end effector for a portion of the trajectory when for example a cup with liqiud is held, etc).

We formulate the STEAP problem on a single probabilistic graphical model and seek the MAP function with incremental inference (Kaess et al. 2012). This allows us to exploit the underlying sparsity of the problem and avoid resolving it from scratch as new information is encountered. In our approach the trajectory is only updated where required, dramatically reducing the overall computational burden and enabling a faster-than-real-time solution. We also provide theoretical insight on the connections between our approach and various methods in mapping, estimation, and planning in the context of solving them as inference on graphical models. To better accommodate mobile manipulation problems, we build on recent work by Anderson and Barfoot (2015) on continuous-time trajectory estimation on SE(3) and extend Dong et al. (2016) to plan trajectories on Lie groups. We implement our framework for solving STEAP and perform several experiments to evaluate our approach in simulation and on the Vector mobile manipulator (Fig. 1), and show that our framework is able to incrementally integrate real-world sensor data and directly update its trajectory estimate and motion plan in real-time. This paper is an extended and revised version of our conference paper (Mukadam et al. 2017b). In particular,

- We summarize the incremental inference via Bayes trees approach (Kaess et al. 2012) in Sect. 6, specifically in the context of solving the STEAP problem.
- We provide a detailed explanation on formulating sparse GPs on Lie groups (Dong et al. 2017a) in Sect. 7.
- We present updated experiments on a harder dataset for the planar robot and new experiments with a 18-DOF PR2 robot in simulation.

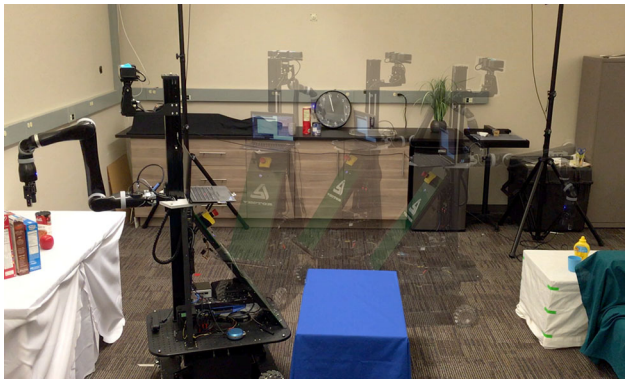


Fig. 1 The Vector mobile manipulator, with an omni-drive base and a 6-DOF Kinova JACO2 arm, is solving the STEAP problem. The task involves picking up an object from the white table on the right and dropping it off on the white table on the left. The semi-transparent robots show the trajectory taken, while the solid robot is the goal configuration

- We provide further insight in to our approach by adding a discussion on limitations and future work in Sect. 10.

2 Related work

By viewing trajectory estimation and motion planning as inference, we are able to borrow and combine tools from different areas of robotics. The Simultaneous Localization and Mapping (SLAM) community has focused on efficient optimization algorithms for many years. One of the more successful approaches is the Smoothing and Mapping (SAM) family of algorithms (Dellaert and Kaess 2006) that formulates SLAM as inference on a factor graph (Kschischang et al. 2001) and exploits the sparsity of the underlying large-scale linear systems to perform inference efficiently. Given new sensor data, incremental Smoothing and Mapping (iSAM) (Kaess et al. 2008, 2012) exploits the structure of the problem to efficiently update the solution rather than resolving the entire problem from scratch. Recently, Tong et al. (2013) introduced a continuous-time formulation of the SAM problem, in which the robot trajectory is a function that maps any time to a robot state. The problem of estimating this function along with landmark locations has been dubbed simultaneous trajectory estimation and mapping (STEAM). This approach was further extended in Barfoot et al. (2014) to take advantage of the sparse structure inherent in the STEAM problem, in Yan et al. (2017) to efficiently and incrementally update the solution, and in Dong et al. (2017b) to 4D mapping problems. The resulting algorithms speed up solution time and can be viewed as continuous-time analogs of the original square-root SAM algorithm in Dellaert and Kaess (2006) and the iSAM2 algorithm in Kaess et al. (2012).

While probabilistic inference is frequently used as a foundation for state estimation and localization, it is only recently that these techniques have been used for planning. The duality between linear estimation and control has long been established (Kalman et al. 1960), but solutions to estimation and control problems have, for the most part, evolved independently within their own subfields. In the last decade this has begun to change. The optimization-inference duality has been shown to extend to planning and optimal control (Todorov 2008) with some early work in this direction looking at solving Markov decision processes (MDP) (Attias 2003). Several researchers have recently proposed a probabilistic inference perspective on planning and control problems, leveraging expectation maximization (Toussaint and Storkey 2006; Levine and Koltun 2013), expectation propagation (Toussaint 2009), KL-minimization (Rawlik et al. 2012), and efficient inference on factor graphs (Dong et al. 2016; Mukadam et al. 2017c, a; Huang et al. 2017; Rana et al. 2017). Interestingly, the incremental inference technique (Kaess et al. 2011) used in Dong et al. (2016) to solve replanning problems is the same as originally used in Kaess et al. (2012) to solve SLAM problems. We exploit this idea to solve our more general class of simultaneous trajectory estimation and planning problems.

Efficient replanning algorithms for navigation are an active area of research (Koenig and Likhachev 2005; Ferguson et al. 2006), but most previous work is difficult to extend to real, high-dimensional systems, is computationally expensive, or does not incorporate uncertainty in the robot's state estimate. Recent work in simultaneous localization and planning (SLAP) attempts to unify robot localization and planning, with early work using HMMs (Penny 2014), more recent approaches designed for dynamic environments (Agha-mohammadi et al. 2015; Rafieisakhaei et al. 2016), and new approaches (Ta et al. 2014) that combine state estimation and model predictive control (MPC) (Camacho and Alba 2013). Unfortunately, these approaches are too computationally expensive due to the MPC style re-evaluation of the new plan, which is compounded with high DOF systems in cluttered environments. In this work, we tackle the simultaneous trajectory estimation and planning (STEAP) problem within a unified probabilistic inference framework. The STEAP problem can be considered as a generalization of the SLAP problem in that the goal of STEAP is to compute the full continuous-time trajectory conditioned on observations and costs in both the past and the future. By contrast, SLAP only computes the current state estimate and the new plan.

3 Background: trajectory optimization as probabilistic inference

Following previous work on both STEAM problems (Barfoot et al. 2014; Yan et al. 2017) and Gaussian process motion planning (Dong et al. 2016; Mukadam et al. 2017c), we view the problem of estimating or optimizing continuous-time trajectories as probabilistic inference. We represent the trajectory as a continuous-valued function mapping time t to robot states $\boldsymbol{\theta}(t)$. The goal is to find the *maximum a posteriori* (MAP) continuous-time trajectory given a prior distribution on the space of trajectories and a likelihood function.

3.1 Trajectory prior

A prior distribution over trajectories can be defined as a vector-valued Gaussian process $\boldsymbol{\theta}(t) \sim GP(\boldsymbol{\mu}(t), \mathbf{K}(t, t'))$, where $\boldsymbol{\mu}(t)$ is a vector-valued mean function and $\mathbf{K}(t, t')$ is a matrix-valued covariance function. For any collection of times $\boldsymbol{t} = \{t_0, \ldots, t_N\}$, $\boldsymbol{\theta}$ has a joint Gaussian distribution

$$\boldsymbol{\theta} \doteq \begin{bmatrix} \boldsymbol{\theta}_0 & \ldots & \boldsymbol{\theta}_N \end{bmatrix}^\top \sim N(\boldsymbol{\mu}, \mathbf{K}) \tag{1}$$

with mean vector $\boldsymbol{\mu}$ and covariance kernel $\mathbf{K}$ defined as

$$\boldsymbol{\mu} \doteq \begin{bmatrix} \boldsymbol{\mu}(t_0) & \ldots & \boldsymbol{\mu}(t_N) \end{bmatrix}^\top, \ \mathbf{K} \doteq [\mathbf{K}(t_i, t_j)]\Big|_{ij, 0 \leq i, j \leq N}. \tag{2}$$

The prior distribution is then defined by the GP mean $\boldsymbol{\mu}$ and covariance $\mathbf{K}$

$$p(\boldsymbol{\theta}) \propto \exp\left\{ -\frac{1}{2} \parallel \boldsymbol{\theta} - \boldsymbol{\mu} \parallel^2_{\mathbf{K}} \right\}. \tag{3}$$

The prior encodes information about the system that is known *a priori*. For example, in robotic state estimation problems, a structured GP prior may encourage trajectories to follow known system dynamics, e.g. the robot velocity changes smoothly (Barfoot et al. 2014; Tong et al. 2013). In motion planning, the prior is selected to encourage higher-order derivatives of the system configuration to be minimized (Dong et al. 2016; Mukadam et al. 2017c). The prior we use in our implementation is detailed in Sect. 5.1.1.

3.2 Likelihood function

The likelihood function encodes information about a particular problem instance. For example, in STEAM problems, the likelihood function encourages posterior trajectories to be consistent with proprioceptive or landmark observations (Barfoot et al. 2014), while in motion planning problems the likelihood function encourages posterior trajectories to be collision-free (Dong et al. 2016).

Let $\mathbf{e}$ be a collection of random binary events. Examples of events include collision, receiving a sensor measurement, or reaching a goal. The likelihood function is the conditional distribution $l(\boldsymbol{\theta}; \mathbf{e}) = p(\mathbf{e}|\boldsymbol{\theta})$, which specifies the probability of events $\mathbf{e}$ given a trajectory $\boldsymbol{\theta}$. We define the likelihood as a distribution in the exponential family

$$l(\boldsymbol{\theta}; \mathbf{e}) \propto \exp\left\{ -\frac{1}{2} \parallel \boldsymbol{h}(\boldsymbol{\theta}, \mathbf{e}) \parallel^2_{\boldsymbol{\Sigma}} \right\} \tag{4}$$

where $\boldsymbol{h}(\boldsymbol{\theta}, \mathbf{e})$ can be any vector-valued cost function with covariance matrix $\boldsymbol{\Sigma}$. The specific likelihood used in our implementation is detailed in Sect. 5.1.

3.3 Computing the MAP trajectory

The posterior distribution of the trajectory given the events can be written in terms of the prior and the likelihood using the Bayes rule

$$p(\boldsymbol{\theta}|\mathbf{e}) \propto p(\boldsymbol{\theta}) p(\mathbf{e}|\boldsymbol{\theta}). \tag{5}$$

Then, we can compute the *maximum a posteriori* (MAP) trajectory

$$\begin{aligned} \boldsymbol{\theta}^* &= \text{argmax}_{\boldsymbol{\theta}} \left\{ p(\boldsymbol{\theta}|\mathbf{e}) \right\} = \text{argmax}_{\boldsymbol{\theta}} \left\{ p(\boldsymbol{\theta}) p(\mathbf{e}|\boldsymbol{\theta}) \right\} && (6) \\ &= \text{argmin}_{\boldsymbol{\theta}} \left\{ -\log\left(p(\boldsymbol{\theta}) p(\mathbf{e}|\boldsymbol{\theta}) \right) \right\} && (7) \\ &= \text{argmin}_{\boldsymbol{\theta}} \left\{ \frac{1}{2} \parallel \boldsymbol{\theta} - \boldsymbol{\mu} \parallel^2_{\mathbf{K}} + \frac{1}{2} \parallel \boldsymbol{h}(\boldsymbol{\theta}, \mathbf{e}) \parallel^2_{\boldsymbol{\Sigma}} \right\} && (8) \end{aligned}$$

where Eq. (8) follows from Eq. (3) and Eq. (4). The MAP estimation problem can therefore be reduced to a nonlinear least squares problem and can be solved with tools like Gauss-Newton or Levenberg-Marquardt.

When solving estimation and planning simultaneously, we encounter new measurements and/or cost information during online execution, thus changing the likelihood. A naïve approach to contending with this new information would be to resolve the problem in Eq. (8) repeatedly. However, this is very inefficient and computationally expensive for an online setting. In the following sections, we formulate the inference problem on graphical models that allow us to exploit the underlying sparsity of the problem (Sects. 4–5), and then we use incremental inference techniques that allow us to iteratively update the solution only where needed resulting in a computationally efficient approach (Sect. 6).

4 Mapping, estimation, and planning with factor graphs

The MAP trajectory computation in Sect. 3.3 can be executed efficiently by exploiting known structure in the problem. This is accomplished by representing the posterior distribution as a *factor graph*. With a factor graph (Kschischang et al. 2001) any distribution can be *factored* into a product of functions that is organized as a bipartite graph $G = \{\Theta, F, E\}$. This graph consists of variable nodes $\Theta \doteq \{\boldsymbol{\theta}_0, \ldots, \boldsymbol{\theta}_N\}$, factor nodes $F \doteq \{f_0, \ldots, f_M\}$, and edges E that connect the two types of nodes.

In our case, the variables are a set of instantaneous robot states along the trajectory, and the factors are conditional probability distributions on variable subsets Θ_i of Θ. Therefore, we can write the posterior distribution as a product of the factors

$$p(\boldsymbol{\theta}|\mathbf{e}) \propto \prod_{i=0}^{M} f_i(\Theta_i). \tag{9}$$

The precision matrix of this distribution also encodes the connectivity in the graph. Consequently, a sparse factor graph structure yields a sparse precision matrix, which can be exploited to make the computation in Eq. (8) efficient (Dellaert and Kaess 2006).

We can further write the posterior distribution as a product of prior factors and likelihood factors,

$$p(\boldsymbol{\theta}|\mathbf{e}) \propto p(\boldsymbol{\theta}) p(\mathbf{e}|\boldsymbol{\theta}) \propto f^{prior}(\Theta) f^{like}(\Theta). \tag{10}$$

In the remainder of this section, we will use this general formulation to illustrate relationships between prior work in mapping, estimation, and planning. Then, we will extend this idea and connect it with our proposed work in the next section.

SAM We begin with the smoothing and mapping (SAM) (Dellaert and Kaess 2006) problem, an early work that uses factor graphs to address the state estimation and mapping problem in robotics. The goal is to estimate the full posterior trajectory in the past given all measurements. The factor graph used in SAM is

$$p(\boldsymbol{\theta}_{est}|\mathbf{e}) \propto f^{prior} f^{meas}, \tag{11}$$

where $\boldsymbol{\theta}_{est}$ signifies the history portion of the trajectory to be estimated, $f^{prior} = f^{prior}(\boldsymbol{\theta}_0)$ is the prior on the first state, and f^{meas} is the likelihood of all sensor measurements, which itself factors as

$$f^{meas} = \prod_i f_i^{meas}(\Theta_i). \tag{12}$$

Unary measurement factors can refer to odometer, GPS or IMU measurements, while higher order factors on a subset of states (Θ_i) often represent landmark observations.

STEAM Like SAM, simultaneous trajectory estimation and mapping (STEAM) (Barfoot et al. 2014; Anderson and Bar-

foot 2015) addresses trajectory estimation problems. The key difference is that in STEAM, the trajectory is no longer treated as a discrete sequence of states Θ, but rather a continuous-time trajectory sampled from a GP. The prior is a joint distribution on the full trajectory $f^{prior} = f^{gp}(\Theta)$, yielding a factor graph

$$p(\boldsymbol{\theta}_{est}|\mathbf{e}) \propto f^{gp} f^{meas}. \tag{13}$$

GPMP2 GPMP2 (Dong et al. 2016) is a probabilistic inference framework for solving planning problems. It utilizes the GP trajectory representation from STEAM to find collision free future trajectories that satisfy the GP prior. Unlike SAM and STEAM, however, the likelihood is not based on sensor measurements, but rather the likelihood of a trajectory being free from collision with obstacles. The collision factor is defined as

$$f^{obs} = \prod_i f_i^{obs}(\boldsymbol{\theta}_i). \tag{14}$$

A fixed start and goal state (this can also be an end effector goal in workspace) is also required in planning problem, and therefore can be incorporated in to the likelihood. So factors to fix start and goal configurations are also employed

$$f^{fix} = f^{start}(\boldsymbol{\theta}_0) f^{goal}(\boldsymbol{\theta}_N). \tag{15}$$

The full factor graph of GPMP2 is, therefore

$$p(\boldsymbol{\theta}_{plan}|\mathbf{e}) \propto f^{gp} f^{obs} f^{fix} \tag{16}$$

where $\boldsymbol{\theta}_{plan}$ signifies the future portion of the trajectory to be planned.

SLAP In real robotics applications, it is frequently the case that both estimation and planning problems must be solved. One approach to tackling this problem is SLAP (Penny 2014; Agha-mohammadi et al. 2015). Although previous work in this area does not employ factor graphs, we reformulate SLAP using them here to illustrate its relation to other problems. SLAP can be viewed as splitting the inference problem into two factor graphs, an estimation graph and a planning graph, defined by

$$p(\boldsymbol{\theta}_{est}|\mathbf{e}) \propto f^{prior} f^{meas}, \tag{17}$$

$$p(\boldsymbol{\theta}_{plan}|\mathbf{e}) \propto f^{prior} f^{curr} f^{obs} f^{goal}. \tag{18}$$

If a continuous-time trajectory representation like GPs are employed, we can replace f^{prior} with f^{gp}. SLAP solves the estimation graph first to find an estimate of the current state $f^{curr}(\boldsymbol{\theta}_{curr})$ and then uses it to initialize and solve the planning problem.

We summarize the factorization of these various problems in Table 1, with their factor graphs shown in Fig. 2.

Table 1 Summary of related problems

Method	Problem solved	Factorization
SAM	Estimation + mapping	$f^{prior} f^{meas}$
STEAM	Estimation + mapping	$f^{gp} f^{meas}$
GPMP2	Planning	$f^{gp} f^{obs} f^{fix}$
SLAP	Estimation + planning	Estimation: $f^{prior} f^{meas}$ Planning: $f^{prior} f^{obs} f^{fix}$
STEAP	Estimation + planning	$f^{gp} f^{meas} f^{obs} f^{fix}$

5 Simultaneous trajectory estimation and planning

In this paper, we present simultaneous trajectory estimation and planning (STEAP), where the task is to perform inference on the entire factor graph from start to goal at once, in contrast to SLAP, which would solve the estimation and planning graph sequentially. We optimize the full trajectory $\boldsymbol{\theta} = \boldsymbol{\theta}_{est} \cup \boldsymbol{\theta}_{plan}$ represented by the GP prior in Sect. 3.1 given all sensor data and cost information collected in to a single likelihood. Compared to prior work discussed in the previous section, the likelihood here is interpreted more broadly to represent events than happen in the past and in the future, all together. The STEAP factor graph is defined as,

$$p(\boldsymbol{\theta}|\mathbf{e}) \propto f^{gp} f^{meas} f^{obs} f^{fix}. \tag{19}$$

We define these factors in Sect. 5.1. Given the current timestep, the solutions to the estimation and planning problems automatically fall out. During online execution as new measurement or cost information is encountered, the likelihood, and by extension the factor graph, can be updated appropriately. Performing inference on the new graph then provides the updated estimation and replanning solutions. We explain this procedure with a simple toy example in Sect. 5.2. STEAP has the following major advantages:

(i) Optimization of a single graph allows information to flow between the two sub-graphs of estimation and planning, which is not possible with SLAP. This increases performance in both estimation and planning, and provides mutual benefit. The collision-free likelihood of both the past and the future part of the graph encourages the estimated past trajectory to remain in areas without obstacles, since a successfully traversed trajectory would not have passed through obstacles. This helps contend with noisy (or drops in) raw measurements and reduces trajectory estimation errors. Similarly, the trajectory estimation information corrects the estimate of the current robot position, providing feedback for the planned future trajectory.

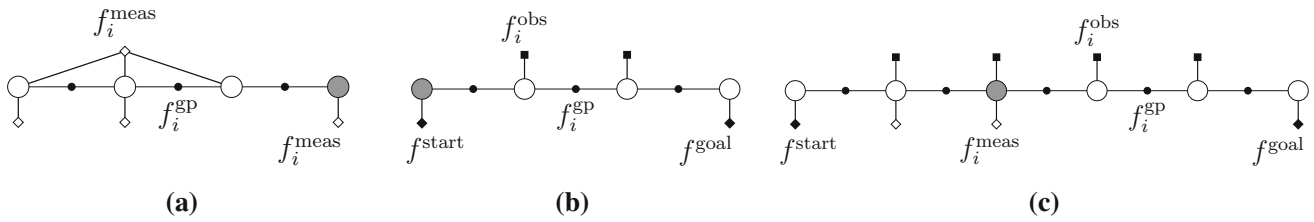


Fig. 2 Example factor graph representation of **a** STEAM, **b** GPMP2, and **c** STEAP. Gray node shows current time-step (Color figure online)

(ii) The number of variables in a STEAP factor graph does not change much during execution i.e. only a few factors are added in each step. This allows for very efficient incremental inference using the Bayes tree data structure (Kaess et al. 2011). Updating the solution with Bayes trees only requires a small fraction of the runtime compared to reoptimizing the full graph from scratch. Additional discussion of incremental inference using the Bayes tree is in Sect. 6.

5.1 STEAP factor definitions

5.1.1 The Gaussian process prior factor

A Gaussian RBF kernel defines a prior distribution of trajectories with no pairwise independences. In other words, all states are connected to a single GP prior factor, $f^{gp} = f^{gp}(\Theta)$. This prior cannot be factored, and destroys the problem's sparsity, making inference computationally expensive. However, in the context of STEAM problems, Barfoot et al. (2014) showed that certain types of GP priors generated by linear time varying (LTV) stochastic differential equations (SDEs), are sufficient to model Markovian robot trajectories. These priors are highly structured, and factor according to

$$f^{gp} = \prod_i f_i^{gp}(\boldsymbol{\theta}_i, \boldsymbol{\theta}_{i+1}) \tag{20}$$

where any GP prior factor connects to only its two neighboring states, forming a (Gauss–Markov) chain. This is shown in Fig. 2a where states (white circle) form a chain by connecting to GP prior factors (black circle). In GPMP2 (Dong et al. 2016), the GP prior on trajectories is generated by a LTV-SDE defined on a vector space (Fig. 2b). GP priors have also been formulated with non-linear SDEs (Anderson et al. 2015) and on the SE(3) Lie group (Anderson and Barfoot 2015).

If the robot configuration is in vector space $\mathbb{R}^n$, similar to GPMP2, STEAP can use the GP prior defined in (Barfoot et al. 2014). But we develop STEAP for mobile manipulators that have their configuration space defined by a Lie group product $\boldsymbol{\theta}_i \in \text{SE}(2) \times \mathbb{R}^n$ where n is the degree-of-freedom of the arm and the SE(2) Lie group defines a planar translation and rotation (yaw) for the mobile base. We employ a constant velocity i.e. noise-on-acceleration model to define a non-linear SDE that generates our GP prior. See Sect. 7 for details about the GP prior.

5.1.2 Obstacle factor

All obstacle factors are constructed similar to GPMP2 (Dong et al. 2016) except that they are defined for the Lie group configuration space. The obstacle factors evaluate collision cost using a hinge loss function and a signed distance field of the environment. See (Dong et al. 2016) for details.

5.1.3 Start and goal factor

These are multivariate Gaussian factors

$$f^{start}(\boldsymbol{\theta}_0) = \exp\left\{ -\frac{1}{2} \parallel \boldsymbol{\theta}_0 - \boldsymbol{\theta}_{start} \parallel^2_{\Sigma_{fix}} \right\} \tag{21}$$

$$f^{goal}(\boldsymbol{\theta}_N) = \exp\left\{ -\frac{1}{2} \parallel \boldsymbol{\theta}_N - \boldsymbol{\theta}_{goal} \parallel^2_{\Sigma_{fix}} \right\} \tag{22}$$

with the mean as the start or goal and a small covariance Σ_{fix}, and are used to tie down the trajectory at the start and goal locations. When the trajectory has finished execution, the goal factor is replaced with the pose measurement factor so that the final posterior update gives the final trajectory estimate.

5.1.4 Measurement factor

There are many types of sensors that provide different measurements, and thus many types of measurement factors have been proposed by the SLAM community (Dellaert and Kaess 2006). For example, measurements from an inertial measurement unit (IMU) can be incorporated into the factor graph with pre-integrated IMU factors (Forster et al. 2015), and visual landmark measurements from a camera can be incorporated with Schur complement factors (Carlone et al. 2014).

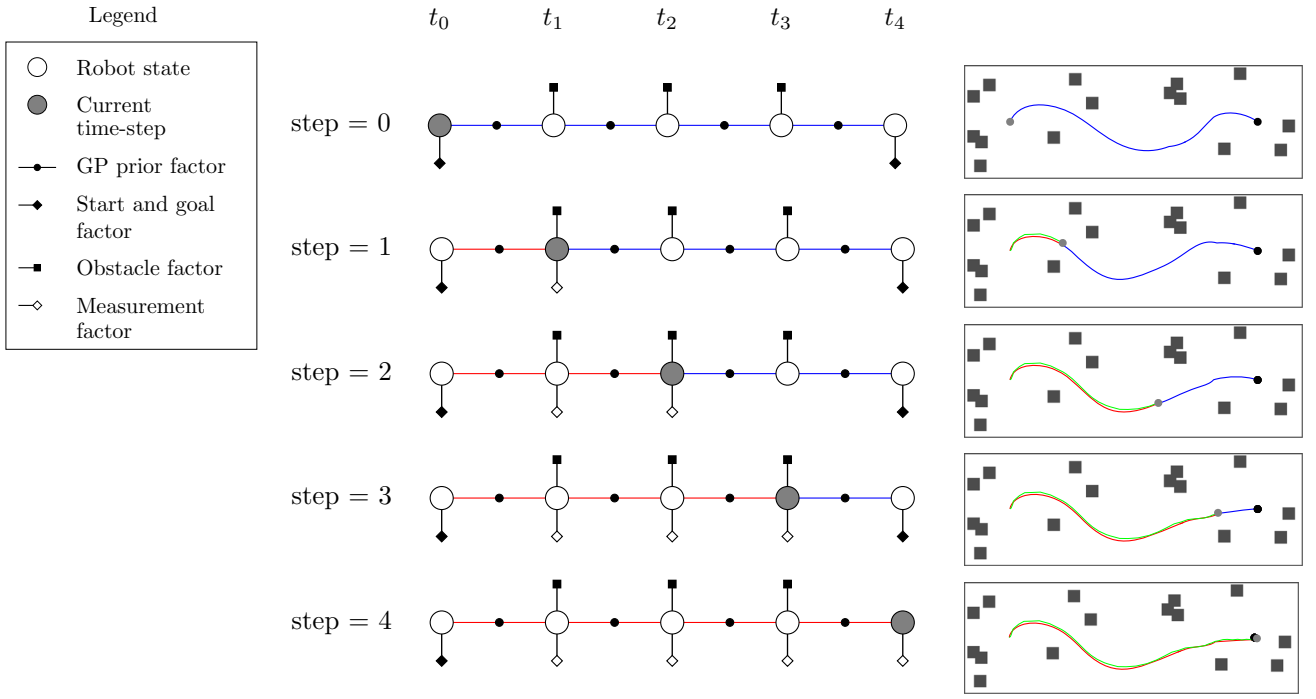


Fig. 3 A simple example illustrates STEAP using a robot (gray) that navigates to the goal (black circle) while avoiding obstacles. At each step the right side shows the environment with ground-truth (green), estimated (red), and replanned (blue) trajectories. The left side shows the corresponding factor graph. See text for details (Color figure online)

For the sake of simplicity, in the remainder of this paper we use a multivariate Gaussian measurement factor for the state measurement

$$f_i^{meas}(\boldsymbol{\theta}_i) = \exp\left\{ -\frac{1}{2} \parallel \boldsymbol{\theta}_i - \boldsymbol{\mu}_i^{meas} \parallel^2_{\Sigma_{meas}} \right\} \tag{23}$$

where the measurement queried from sensors has mean $\boldsymbol{\mu}_i^{meas}$ with covariance Σ_{meas}. The multivariate Gaussian is a typical noise assumption for many sensors. For example, GPS can provide coordinate and velocity measurements in Cartesian space with covariance (Leandro et al. 2005), and 2D/3D laser scanners can provide a raw localization with covariance from a 2D/3D point-cloud with the ICP algorithm (Censi 2007). Note that, although we only use multivariate Gaussian measurement factors in our examples and evaluations, the STEAP framework is general and can incorporate any type of measurement factors.

5.2 A STEAP example

We use an example, illustrated in Fig. 3, to describe how STEAP works using Algorithm 1 that is complemented by the block diagram in Fig. 8. Note that the small size of the graph in Fig. 3 is just for illustration, in practice our approach can handle much larger graphs (see Sect. 9 for the graph sizes used in our experiments). In this example, a robot with stochastic dynamics starts at time-step t_0 and needs to reach the goal at time-step t_4 while avoiding any obstacles.

First, we construct a factor graph that will reflect the prior distribution. A small, sparse set of robot states are connected via GP factors that collectively form the prior distribution of a continuous-time trajectory. Then, we add a start and a goal factor with a small covariance (to tie the trajectory down at the start and goal) and obstacle factors. In practice, there are also multiple binary obstacle factors present between any two states (omitted here for clarity) that use GP interpolation to project the cost between any two states back on to those states and allow the trajectory to stay sparse but still reason about obstacles between the sparse states (see Dong et al. 2016 for details). The start, goal and obstacle factors together form the likelihood. We can find the mode of the posterior shown in blue at the top level of Fig. 3, which is inherently a special case of our approach providing the solution to the GPMP2 motion planning problem, since no measurement factors are present and there is no state estimation yet at this step.

Next, the planned solution between t_0 and t_1 is upsampled to a desired resolution with GP interpolation, checked for safety, and then executed on the robot. The ground-truth trajectory is illustrated in green. Since the system is stochastic, execution is noisy. We make an observation to generate a measurement factor and insert it into the graph at t_1. This new factor is combined with the old likelihood to produce the

Algorithm 1 STEAP

$\boldsymbol{\theta}$: trajectory, f: factors, T: Bayes tree

$\boldsymbol{\theta}_{init}$ = initializeTrajectory()
FG = createFactorGraph(f^{gp}, f^{obs}, f^{fix})
$\boldsymbol{\theta}$ = inference(FG, $\boldsymbol{\theta}_{init}$)
T = createBayesTree(FG, $\boldsymbol{\theta}$)
for $i = 0$ **to** $N - 1$ **do**
  $\theta_{i:i+1}$ = interpolateGP($\boldsymbol{\theta}$, i, $i+1$, resolution)
  if collisionFree($\theta_{i:i+1}$) **then**
    execute($\theta_{i:i+1}$)
    f_{i+1}^{meas} = localize()
    $\boldsymbol{\theta}$, T = incrementalInference($\boldsymbol{\theta}$, f_{i+1}^{meas}, T)
  else
    return failure
  end if
end for
return success

updated likelihood. Using the Bayes tree to efficiently organize computation, we generate a new MAP solution. Note that, in this case, the factor graph is changed by adding only one measurement factor, so the incremental inference performed using the Bayes tree will be very fast. The red portion of the trajectory is an estimate of the trajectory traversed by the robot until current time-step t_1 and the blue portion of the trajectory is the replanned solution to the goal. This whole process is then repeated (steps are shown from top to bottom in Fig. 3) until the robot reaches the goal at t_4. At t_4 again we have a special case of our approach that provides a solution to the trajectory estimation problem (STEAM), but with extra obstacle factors.

6 Incremental inference with the Bayes tree data structure

In Sect. 3.3, we discussed how to solve the MAP inference problem as non-linear least squares optimization. But one significant drawback of using non-linear optimization like Gauss-Newton or Levenberg-Marquardt to solve inference on factor graphs is that they are iterative methods, and with every iteration the problem must be completely linearized and resolved (the cost on every factor will be evaluated and every variable is updated), even if the factor graph is mostly unchanged.

To reduce these redundant calculations, Kaess et al. (2012) proposed efficient incremental updates of non-linear least square problems with the *Bayes tree* data structure. When re-solving a graph with only minor changes (in variables or factors), only the parts of the Bayes tree associated with the changes will be updated, leaving most of the Bayes tree unchanged. By updating the solution in this *incremental* manner, the efficiency of inference is significantly improved.

Since only a very small portion of the STEAP factor graph changes (few variables with new measurement factors) at each time-step, we convert the factor graph into a Bayes tree, and update the tree incrementally. By utilizing this efficient incremental inference technique we get a significant performance boost, and easily achieve real-time performance, as illustrated in our experiments.

In this section, we first give a brief overview of the Bayes tree and its relation to factor graphs, discuss how to perform incremental inference on Bayes tree, and then give a detailed example to show how to use a Bayes tree to perform incremental inference for a STEAP problem. Readers are encouraged to refer to the iSAM2 paper (Kaess et al. 2012) for a more detailed and general explanation.

6.1 Building a Bayes tree from a factor graph

A Bayes tree is a directed tree-structured graphical model which is derived from a Bayes net that has very close relation to junction tree. Both the Bayes tree's and junction tree's nodes are *cliques* of a Bayes net, but the Bayes tree is *directed* to reflect the conditional relations in factored probability density.

Before we officially define the Bayes tree, we first introduce the *variable elimination* algorithm (Dellaert and Kaess 2006), which converts a factor graph into a *Bayes net*. For a given factor graph, we first choose an ordering of variables. Although any variable ordering works for the explanation here, different orderings generate Bayes nets with different numbers of edges, and, in general, a smaller number of edges is better for reducing computation. Choosing the optimal ordering is a NP-hard problem. To contend with this problem, several approximation heuristics have been proposed. We use COLAMD (Davis et al. 2004) to estimate a close-to-optimal ordering.

Given a factor graph and a variable ordering, we eliminate each variable by Algorithm 2 and factorize the probability density over all variables to

$$p(\theta) = \prod_j p(\theta_j | S_j), \tag{24}$$

where $S_j \subset \theta$ is the separator of θ_j. Note that the factorized probability density in Eq. (24) meets the conditional dependencies of a Bayes net, so, by elimination, we convert a factor graph to a Bayes net. Fig. 4 shows an example of running elimination Algorithm 2 on a GPMP2 planning factor graph with reverse variable ordering, which is actually the optimal variable ordering in this case.

For a Bayes net generated by Algorithm 2, we extract all cliques C_k from the Bayes net and build a Bayes tree by defining each node by one clique C_k. For each node of the Bayes tree, we further define the conditional density $p(F_k | S_k)$, where S_k is the *separator variables* S_k, by $S_k = C_k \cap \Pi_k$ intersecting between C_k and C_k's parent node

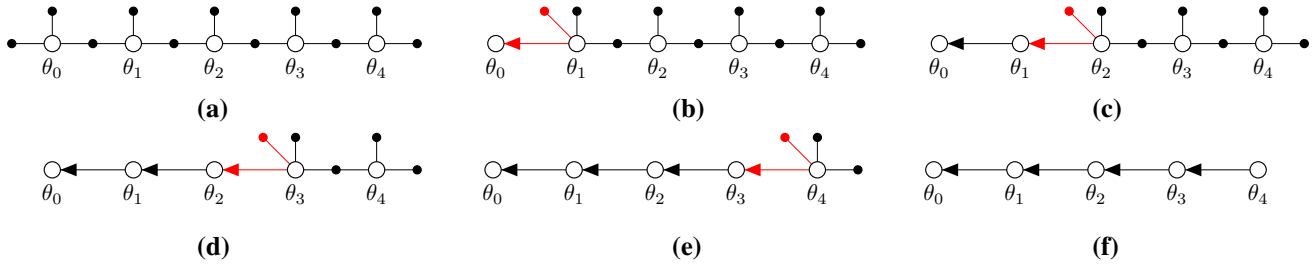


Fig. 4 Example of applying variable elimination on a planning factor graph. Red arrows/factors indicate the parts that change in Bayes net/factor graph respectively at step 5 of Algorithm 2. **a** Factor graph. **b** Eliminating θ_0. **c** Eliminating θ_1. **d** Eliminating θ_2. **e** Eliminating θ_3. **f** Eliminating θ_4 and final Bayes net (Color figure online)

Algorithm 2 Variable elimination of factor graph

1: **for all** θ_j, in ordering, **do**
2: Remove all factors f_i connected to θ_j from factor graph, define $S_j = \{\text{all variables involved in all } f_i\}$.
3: $f_j(\theta_j, S_j) = \prod_i f_i(\theta)$.
4: Factorize $f_j(\theta_j, S_j) = p(\theta_j|S_j) f_{new}(S_j)$.
5: Add $p(\theta_j|S_j)$ in Bayes net, add $f_{new}(S_j)$ back in factor graph.
6: **end for**

Algorithm 3 Creating a Bayes tree from a Bayes net

1: **for all** $p(\theta_j|S_j)$, in *reverse* ordering, **do**
2: **if** $S_j = \emptyset$ **then**
3: Start a tree with a root clique C_r with $p_{C_r} = p(\theta_j)$, $F_r = \{\theta_j\}$.
4: **else**
5: Find the parent clique C_p that contains the first eliminated variable in S_j is in F_p.
6: **if** $F_p \cap S_p \subseteq S_j$ **then**
7: Add $p(\theta_j|S_j)$ in p_{C_p}, add θ_j in F_p, and add S_j in S_p.
8: **else**
9: Add a new clique C' in tree with $p_{C'} = p(\theta_j|S_j)$, $F' = \{\theta_j\}$, $S' = S_j$ as a child node of C_p.
10: **end if**
11: **end if**
12: **end for**

Π_k, and the *frontal variables* F_k, by $F_k = C_k \backslash S_k$. The clique is written as $C_k = F_k : S_k$. The probability density of a Bayes tree is defined by the joint density of all nodes

$$p(\theta) = \prod_k p_{C_k}(F_k|S_k). \tag{25}$$

The algorithm to convert a Bayes net to a Bayes tree is summarized in Algorithm 3, and the Bayes tree example of the toy planning factor graph from Fig. 4 is illustrated in Fig. 5. Here, the example is straightforward: the Bayes net has 4 cliques, $\{\theta_0, \theta_1\}$, $\{\theta_1, \theta_2\}$, $\{\theta_2, \theta_3\}$ and $\{\theta_3, \theta_4\}$, so the Bayes tree has 4 nodes.

6.2 Incremental inference on Bayes tree

Given a Bayes tree generated from a factor graph, we can efficiently add new factors and perform incremental inference. One of the most important properties of the Bayes tree is that

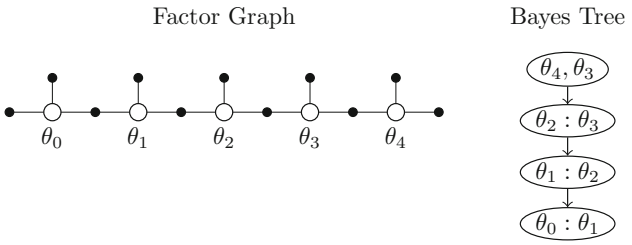


Fig. 5 Example of a Bayes Tree of a planning factor graph

if a factor, involving a variable θ_i, is added in the Bayes tree, only the cliques between the cliques containing θ_i and the root of the tree (assume the root of the tree is at the top) will be affected. The sub-tree below the cliques containing θ_i will remain unchanged during incremental inference. This means that by updating the Bayes tree a large part of the computation that is redundant and involves only untouched cliques, can be prevented compared to solving a full new non-linear least squares optimization problem.

The procedure to update a Bayes tree with new factors is stated in Algorithm 4. Given a set of new factors, we first find all the cliques which contain the variables involved in those new factors, and reinterpret the sub-tree as a factor graph. After adding new factors in the factor graph, we perform elimination on the factor graph to get the corresponding Bayes net, and further convert the Bayes net in to a Bayes tree. Finally, we attach the untouched sub-tree to the updated sub-tree, to get the final updated tree. The procedure is further explained by a toy example in Fig. 6. In this example we add a unary factor to a planning factor graph at θ_2. As explained in the previous sub-section, we have the Bayes net of the factor graph in Fig. 6b, where the dashed boxes mark the affected part of the tree. We see that the clique $\theta_0 : \theta_1$ remains untouched during the whole update procedure.

6.3 Using the Bayes tree in STEAP

We are now ready to discuss how the Bayes tree can be used to speed up STEAP. At the beginning of the STEAP algorithm, the factor graph is the same as the planning factor

Algorithm 4 Incremental inference on a Bayes tree

Require: Bayes tree T, add factors F.
1: Remove top of T, reinterpret as factor graph.
2: Add F to factor graph.
3: Eliminate factor graph to Bayes net, then to Bayes tree.
4: Attach unchanged sub-tree to updated sub-tree.
5: **return** Updated Bayes tree T^*.

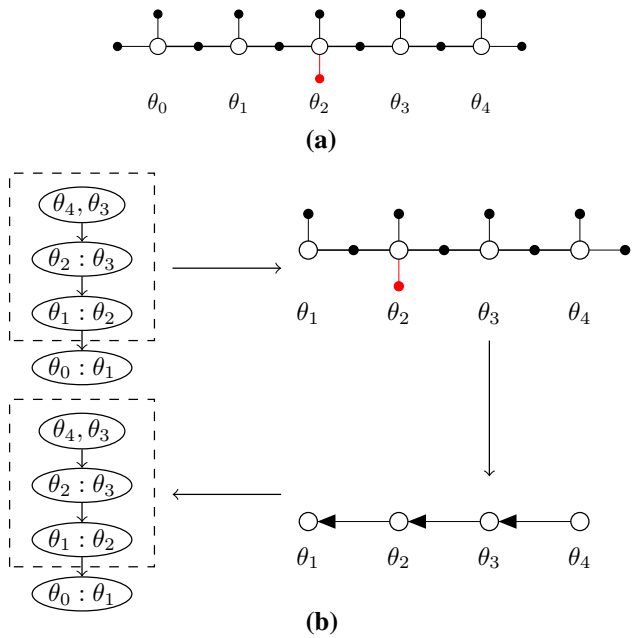


Fig. 6 An example of how to add factors and perform incremental inference on a Bayes tree. **a** The inference problem illustrated as a factor graph. The added factor is displayed in red. **b** Steps to add the red factor into Bayes tree (Color figure online)

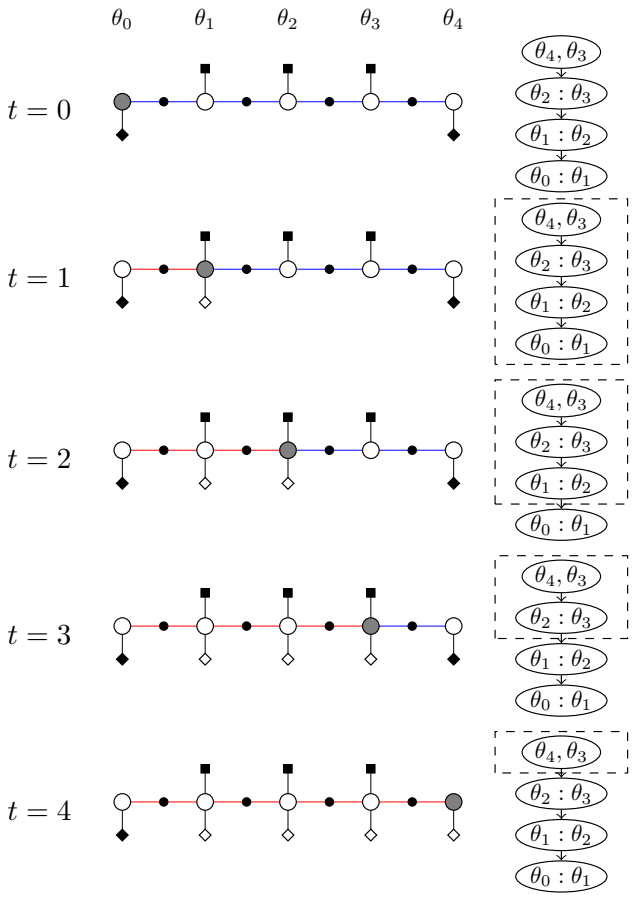


Fig. 7 The Bayes tree of the STEAP example (red and blue are the estimation and planning parts of the graph respectively) from Sect. 5.2. At $t = 0$ the Bayes tree is built from the factor graph shown on the left, and from $t = 1$ to $t = 4$ the incremental inference is performed on the tree, with affected sub-trees marked by dashed boxes (Color figure online)

graph in GPMP2 (Dong et al. 2016). A Bayes tree is constructed from the factor graph using the approach discussed in Sect. 6.1. After construction of the Bayes tree, the factor graph is no longer needed or maintained, since all remaining steps are performed directly on the Bayes tree. When the Bayes tree needs to be updated with new measurement factors, we update the tree and perform incremental inference using the approach described in Sect. 6.2. An example is shown in Fig. 7 that corresponds to the STEAP example in Sect. 5.2.

As mentioned in Sect. 5, one of the advantages of STEAP, compared to other trajectory estimation and planning approaches, is that it can use Bayes trees to gain a significant speed up while performing incremental inference. In the example in Fig. 7, we see that for each step of STEAP, only part of the Bayes tree is updated leaving the remaining tree unchanged. Therefore, incremental inference is more efficient while maintaining similar accuracy compared to batch inference, which needs to perform all the steps (linearization, solving linear systems, etc.) on the full graph. We also gain additional efficiency improvement from the iSAM2 (Kaess et al. 2012) implementation of the Bayes tree (which we use) that avoids excessive linearization operations by fluid relinearization. We will see in the experimental section how the Bayes tree and iSAM2 improve the efficiency of STEAP compared to other approaches.

7 GP priors

7.1 GP priors on vector space

GP priors on vector-valued system states $\boldsymbol{\theta}(t)$ can be generated by linear time-varying stochastic differential equations (LTV-SDEs) (Barfoot et al. 2014)

$$\dot{\boldsymbol{\theta}}(t) = \mathbf{A}(t)\boldsymbol{\theta}(t) + \mathbf{u}(t) + \mathbf{F}(t)\mathbf{w}(t), \tag{26}$$

where $\mathbf{u}(t)$ is the known system control input, $\mathbf{w}(t)$ is white process noise, and both $\mathbf{A}(t)$ and $\mathbf{F}(t)$ are time-varying sys-

tem matrices. The white process noise is represented by

$$\mathbf{w}(t) \sim GP(\mathbf{0}, \mathbf{Q}_C \delta(t - t')), \tag{27}$$

where $\mathbf{Q}_C$ is the power-spectral density matrix, which is a hyperparameter (Barfoot et al. 2014), and $\delta(t - t')$ is the Dirac delta function. The mean and covariance of the GP is computed by taking the first and second order moments of the solution to Eq. (26)

$$\boldsymbol{\mu}(t) = \boldsymbol{\Phi}(t, t_0)\boldsymbol{\mu}_0 + \int_{t_0}^{t} \boldsymbol{\Phi}(t, s)\mathbf{u}(s)ds$$

$$\mathbf{K}(t, t') = \boldsymbol{\Phi}(t, t_0)\mathbf{K}_0\boldsymbol{\Phi}(t', t_0)^\top + \tag{28}$$

$$\int_{t_0}^{\min(t,t')} \boldsymbol{\Phi}(t, s)\mathbf{F}(s)\mathbf{Q}_C\mathbf{F}(s)^\top\boldsymbol{\Phi}(t', s)^\top ds \tag{29}$$

where $\boldsymbol{\mu}_0$ is the initial mean value of first state, $\mathbf{K}_0$ is the covariance of first state, and $\boldsymbol{\Phi}(t, s)$ is transition matrix.

7.1.1 Constant velocity GP prior

The *constant-velocity* GP prior is generated by a LTV-SDE with a noise-on-acceleration model

$$\ddot{\mathbf{p}}(t) = \mathbf{w}(t), \tag{30}$$

where $\mathbf{p}(t)$ is the N-dimensional vector-valued position (or pose) variable of trajectory, if the system has N degrees of freedom. To convert this prior into the LTV-SDE form of Eq. (26), a Markov system state variable is defined

$$\boldsymbol{\theta}(t) \doteq \begin{bmatrix} \mathbf{p}(t) \\ \dot{\mathbf{p}}(t) \end{bmatrix}, \tag{31}$$

The prior in Eq. (30) then can easily be converted into a LTV-SDE in Eq. (26) by defining

$$\mathbf{A}(t) = \begin{bmatrix} \mathbf{0} & \mathbf{1} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}, \quad \mathbf{u}(t) = \mathbf{0}, \quad \mathbf{F}(t) = \begin{bmatrix} \mathbf{0} \\ \mathbf{1} \end{bmatrix}. \tag{32}$$

7.1.2 GP prior factor

Given the LTV-SDE formulation defined in Eq. (32), we define the GP factor between any two states at t_i and t_{i+1} by (Barfoot et al. 2014)

$$f_i^{gp}(\boldsymbol{\theta}_i, \boldsymbol{\theta}_{i+1}) = \exp\left\{ -\frac{1}{2}\|\boldsymbol{\Phi}(t_{i+1}, t_i)\boldsymbol{\theta}_i - \boldsymbol{\theta}_{i+1}\|^2_{\mathbf{Q}_{i,i+1}} \right\} \tag{33}$$

where

$$\boldsymbol{\Phi}(t, s) = \begin{bmatrix} \mathbf{1} & (t-s)\mathbf{1} \\ \mathbf{0} & \mathbf{1} \end{bmatrix}, \mathbf{Q}_{i,i+1} = \begin{bmatrix} \frac{1}{3}\Delta t_i^3\mathbf{Q}_C & \frac{1}{2}\Delta t_i^2\mathbf{Q}_C \\ \frac{1}{2}\Delta t_i^2\mathbf{Q}_C & \Delta t_i\mathbf{Q}_C \end{bmatrix}. \tag{34}$$

7.1.3 Constant time GP interpolation

The posterior mean of the trajectory at *any* time τ can be approximated by Laplace's method and expressed in terms of the current trajectory $\boldsymbol{\theta}$ at time points $\boldsymbol{t}$ (Barfoot et al. 2014):

$$\boldsymbol{\theta}(\tau) = \boldsymbol{\mu}(\tau) + \mathbf{K}(\tau, \boldsymbol{t})\mathbf{K}^{-1}(\boldsymbol{\theta} - \boldsymbol{\mu}). \tag{35}$$

Although the interpolation in Eq. (35) naïvely requires $O(N)$ operations, with structured kernels $\boldsymbol{\theta}(\tau)$ can be computed in as fast as $O(1)$ (Barfoot et al. 2014; Dong et al. 2016).

7.2 GP prior on Lie groups

The sparse GP prior defined in GPMP2 (Dong et al. 2016) works well for robot manipulators since their configurations can be defined using a *vector space*. But not all robots' configuration spaces can be well represented with a vector space. For example, the orientations of rigid body in 3D space cannot be represented by vectors without singularity (Euler angle) or extra degrees (quaternion). *Lie groups* offer more general robot configuration space representations. For example, SE(2) can represent position and orientation for a planar base of a mobile manipulator, and SE(3) can represent position and orientation for an aerial vehicle. Prior work (Anderson and Barfoot 2015) proposed the sparse GP prior on SE(3) which is useful for trajectory estimation. In this section we extend the sparse GP priors on Lie groups (Chirikjian 2011) that will be useful for more general configuration representations. This section is an overview of GP priors on Lie groups, a more detailed discussion can be found in our technical report (Dong et al. 2017a).

A N-dimensional matrix Lie group G is a sub-group of the general linear group and defines a smooth differentiable manifold whose local tangent space is described by its associated *Lie algebra* $\mathfrak{g}$ (Chirikjian 2011). For example, the Lie algebra of SE(2) is defined by a skew symmetric matrix. We can switch between them using the *exponential map* $\exp : \mathfrak{g} \to G$ and the *logarithm map* $\log : G \to \mathfrak{g}$ and to convert elements in local coordinate of G to Lie algebra and vice versa we can use the *hat operator* $\wedge : \mathbb{R}^N \to \mathfrak{g}$ and the *vee operator* $\vee : \mathfrak{g} \to \mathbb{R}^N$ respectively (Chirikjian 2011).

7.2.1 Constant velocity GP prior

We define a constant velocity GP prior here to match the vector space prior in GPMP2 (Dong et al. 2016), but just like the vector space, other priors can also be applied here. Let $T \in G$ represent a state in G, such that $T(t)$ defines a continuous-time trajectory in G. To generate the GP we need to first construct a stochastic differential equation (SDE)

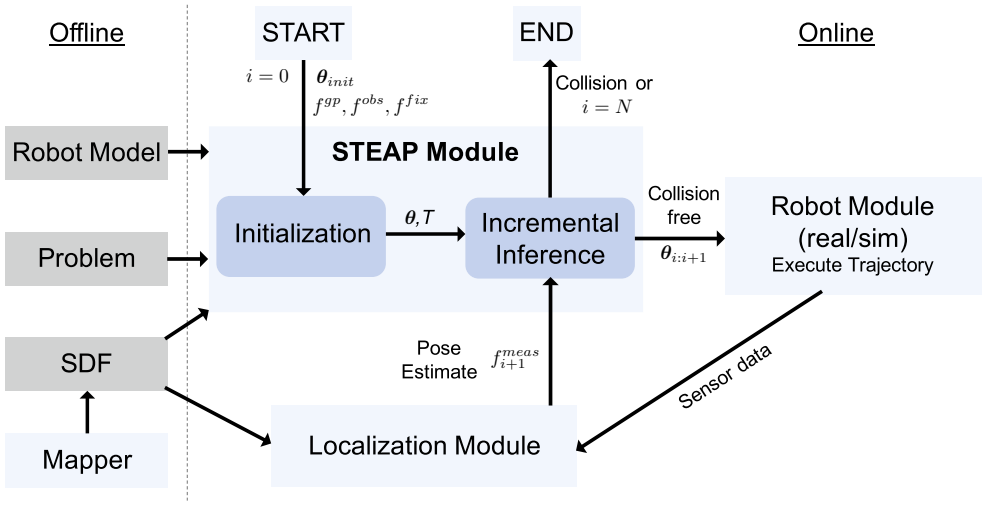


Fig. 8 Block diagram of our framework showing all the components and how they interact. Blue boxes are modules and gray boxes are data. Sensor measurements flow from the `Robot Module` to the `Localization Module`. This block diagram is also complemented by Algorithm 1 (Color figure online)

with a Markovian state (Barfoot et al. 2014). Let that state be $\boldsymbol{\theta}(t) \doteq \{T(t), \boldsymbol{\varpi}(t)\}$, with the SDE as a double integrator noise-on-acceleration model that will yield a constant velocity prior,

$$\dot{\boldsymbol{\varpi}}(t) = \mathbf{w}(t), \quad \mathbf{w}(t) \sim GP\left(\mathbf{0}, \mathbf{Q}_C \delta(t - t')\right) \tag{36}$$

where $\boldsymbol{\varpi}(t)$ is the 'body-frame velocity' variable defined by

$$\boldsymbol{\varpi}(t) \doteq (T(t)^{-1}\dot{T}(t))^{\vee}. \tag{37}$$

and $\mathbf{w}(t)$ is a white noise, zero mean Gaussian process and power-spectral density matrix $\mathbf{Q}_C$ (Barfoot et al. 2014). Since $\forall\, T \in G,\ T^{-1}\dot{T} \in \mathfrak{g}$ (Chirikjian 2011), we can apply the $\vee$ operator on $T(t)^{-1}\dot{T}(t)$. However, unlike GPMP2 (Dong et al. 2016), this SDE is non-linear. Fortunately through linearization we can convert it to the linear time-varying SDE (LTV-SDE) that GPMP2 uses.

7.2.2 Local linearization

With linearization on the Lie group around any T_i, we can define both a *local* GP and a LTV-SDE on the linear tangent space to leverage the constant-velocity GP prior. A local GP at any time t, $t_i \leq t \leq t_{i+1}$ will be

$$T(t) = T_i \exp(\boldsymbol{\xi}_i(t)^{\wedge}), \quad \boldsymbol{\xi}_i(t) \sim N(\mathbf{0}, \mathbf{K}(t_i, t)) \tag{38}$$

where the *local* pose variable $\boldsymbol{\xi}_i(t) \in \mathbb{R}^N$ around T_i is

$$\boldsymbol{\xi}_i(t) \doteq \log(T_i^{-1}T(t))^{\vee}. \tag{39}$$

Then the local LTV-SDE is defined using the local pose similar to Eq. (36) as a double integrator noise-on-acceleration model to give a constant velocity prior

$$\ddot{\boldsymbol{\xi}}_i(t) = \mathbf{w}(t), \quad \mathbf{w}(t) \sim GP(\mathbf{0}, \mathbf{Q}_C \delta(t - t')) \tag{40}$$

with the local Markovian state for the LTV-SDE as

$$\boldsymbol{\gamma}_i(t) \doteq \{\boldsymbol{\xi}_i(t), \dot{\boldsymbol{\xi}}_i(t)\}. \tag{41}$$

To show the equivalence between the original nonlinear SDE and the local approximation we use the identity (Chirikjian 2011)

$$T(t)^{-1}\dot{T}(t) = \left(\mathbf{J}_r(\boldsymbol{\xi}_i(t))\dot{\boldsymbol{\xi}}_i(t)\right)^{\wedge} \tag{42}$$

where $\mathbf{J}_r$ is the *right Jacobian* of G. Following from Eq. (37) we get

$$\dot{\boldsymbol{\xi}}_i(t) = \mathbf{J}_r(\boldsymbol{\xi}_i(t))^{-1}\boldsymbol{\varpi}(t) \tag{43}$$

If the time interval between any t_i and t_{i+1} is small then the approximation

$$\dot{\boldsymbol{\xi}}_i(t) \approx \boldsymbol{\varpi}(t) \tag{44}$$

is good. Then linear SDE Eq. (40) is a good approximation of non-linear SDF Eq. (36), so we linearize the SDE and we can apply LTV-SDE GP prior by (Barfoot et al. 2014).

7.2.3 GP prior factor

We define the GP factor between any two states at t_i and t_{i+1} with their local pose using the linearized SDE formulation

on the Lie group manifold

$$f_i^{gp}(\boldsymbol{\theta}_i, \boldsymbol{\theta}_{i+1}) = \exp\left\{ -\frac{1}{2}\|\boldsymbol{\Phi}(t_{i+1}, t_i)\boldsymbol{\gamma}_i - \boldsymbol{\gamma}_{i+1}\|^2_{\mathbf{Q}_{i,i+1}} \right\} \quad (45)$$

where the logarithm map provide transformation from SE(2) to $\mathbb{R}^3$ and vice versa with the exponential map, and

$$\boldsymbol{\Phi}(t,s) = \begin{bmatrix} \mathbf{1} & (t-s)\mathbf{1} \\ \mathbf{0} & \mathbf{1} \end{bmatrix}, \mathbf{Q}_{i,i+1} = \begin{bmatrix} \frac{1}{3}\Delta t_i^3 \mathbf{Q}_C & \frac{1}{2}\Delta t_i^2 \mathbf{Q}_C \\ \frac{1}{2}\Delta t_i^2 \mathbf{Q}_C & \Delta t_i \mathbf{Q}_C \end{bmatrix}. \quad (46)$$

We use this GP prior in our current implementation.

8 Implementation details

We implement STEAP within the PIPER (Mukadam 2017) package using ROS (Quigley et al. 2009) and GPMP2 (Dong et al. 2016) and have open-sourced the code. Fig. 8 shows a block diagram of the framework. The offline phase assimilates (i) robot-specific information including model and physical parameters, (ii) problem definitions and optimization parameters, and (iii) a pre-generated signed distance field (SDF) of the environment, which is assumed to be static. In the online phase, this information is passed to our central module, `STEAP Module`, that solves STEAP problems and communicates with the `Robot Module` (simulated or physical) with sensors, and the `Localization Module` that takes raw sensor measurements and outputs a noisy pose estimate for the robot that can be interpreted by the `STEAP Module`.

Note that in our framework, the `Localization Module` is free to be any source of raw or processed sensor information, as long as suitable factors are defined to fuse sensor information in the factor graph, e.g. GPS, LIDAR, monocular, or stereo camera data. In our implementation we use a depth image-based localization algorithm, detailed in Sect. 8.4.

8.1 STEAP module

This module uses the robot and problem configuration. In the first step, the module initializes the problem by constructing an initial factor graph, performing inference to get the first planned solution, and constructing a Bayes tree for future iterations. During every other step, the module performs incremental inference by updating the Bayes tree directly. At every step, the replanned solution is upsampled and checked for safety, and is sent to the `Robot Module`. When the `Localization Module` returns a current pose measurement, new sensor measurement factors are added to the Bayes tree and the updated posterior is evaluated. This procedure repeats until the full trajectory completes execution or exits due to collision failure. Given the generic implementation of this module, our framework can be used on any simulated or real robot as long as the robot information is provided in the offline phase.

8.2 Robot module

This module consists of the robot API and controllers that can understand and execute the trajectory passed to it by the `STEAP Module` for either a real robot or an interface with Gazebo for a simulated robot. Sensors on the robot pass information to the `Localization Module`. Note that in our simulator, adjustable Gaussian noise is mixed in both system dynamics (more precisely vehicle velocities) and sensor measurements (more precisely depth measurements of depth camera simulator), to simulate the real-world stochasticity.

8.3 Mapper

Obstacle factors require a signed distance field (SDF) to calculate obstacle cost. Although we can calculate SDFs from CAD models in simulation, we will not have these models available in most real-world environments. Therefore, we build SDFs from sensor data. We use depth scans from depth sensors (a simulated depth camera in simulation and a PrimeSense, shown in Fig. 9, in real-world experiments), and an occupancy grid mapping approach (Thrun et al. 2005, Ch.9) to generate the SDF. The space is first discretized into small cells, and a probability of occupancy p_o is assigned to each cell, with initial $p_o = 0.5$ (since we have no information). All p_os are updated by sensor measurements, and after all sensor data is received, we assume cells with $p_o \geq 0.5$ are occupied, and cells with $p_o < 0.5$ are unoccupied. Note that we assume cells with $p_o = 0.5$, which indicates no depth measurement available, are occupied, since it is safer to assume that locations never observed are occupied by obstacles. True camera poses are provided by a motion capture system during the mapping process. After occupancy grid mapping, we calculate the signed distance field by the efficient distance transformation (Felzenszwalb and Huttenlocher 2012). We use CUDA (Nickolls et al. 2008) to implement occupancy grid mapping and distance transformation, allowing us to achieve real-time performance on scenes roughly of size 8 m × 6 m × 1.5 m with 3 cm resolution. A map constructed with this approach is shown in Fig. 9.

8.4 Localization module

The Localization module reads raw sensor data from the `Robot Module`, calculates a pose estimate of the robot, and provides this information to the `STEAP Module`. Here it's free to choose any Localization algorithm. We choose

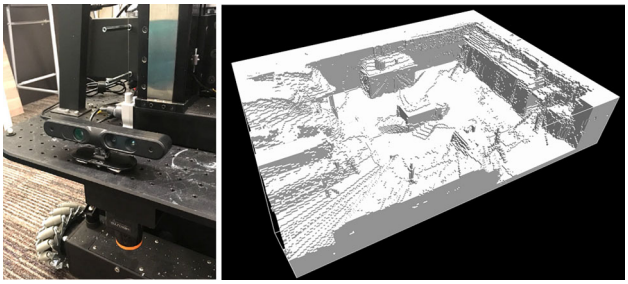


Fig. 9 Left: the PrimeSense depth camera mounted on the robot base. Right: one 8 m×6 m×1.5m occupancy grid map built by the mapper module

an ICP-style iterative approach, which similar to tracking in KinectFusion (Izadi et al. 2011) but operates on the full SDF generated by the `Mapper` rather than the truncated SDF (TSDF) in KinectFusion. We use CUDA to implement and parallelize the tracking module to achieve real-time performance. Although additional sensor data like RGB images, odometry, and laser scans are available to the robot, we only use depth images from PrimeSense in our experiments.

8.5 Computational complexity

Since our approach can involve performing inference on arbitrary factor graphs and incremental inference on their corresponding Bayes trees, the computational complexity depends on the problem and the structure of the graph or tree and how the tree changes over time. Considering the case of the experiments in this paper where the graphs are chain-like and measurement factors are unary, the complexity of the batch step is $O(TD^2)$, where T is the number of time-steps (or states) and D is the dimension of the system state. The time complexity $O(TD^2)$ comes from solving the block-tridiagonal linear system with block size $D \times D$.

In the online step, the time complexity of the incremental update is $O(VD^2)$, where V is the total number of variables (or states) in all the affected cliques of the Bayes tree starting from the root of the tree. In the beginning since all states in the Bayes tree are affected, V is the same as T and the time complexity is equal to the batch step complexity. When the robot is moving along the trajectory, V is much smaller, as shown in Fig. 7.

The time complexity for STEAP discussed above only applies to cases of chain-like graphs (as shown in Fig. 3) that have a start-to-goal variable ordering. In the presence of non-unary measurement factors (for example, higher-order measurement factors like landmark observations) that connect to states across greater time-intervals, or in the case of loop-closures, the time complexity will be greater. In general, the time complexity increases as the sparsity in the graph decreases, and is also dependent on the variable ordering selected by the COLAMD heuristics.

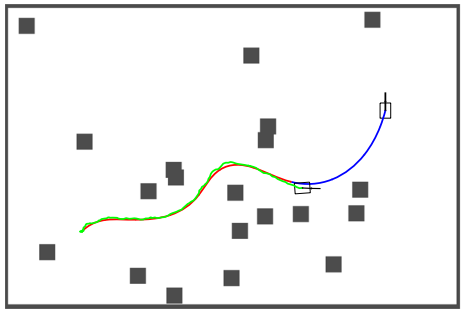


Fig. 10 STEAP result on an example from the simulation benchmark with a planar 2-link mobile arm. Ground-truth (green), estimated past (red) and replanned future (blue) trajectories are shown with the current robot pose between the red and blue trajectories and the goal at the end of the blue trajectory. Best viewed in color (Color figure online)

9 Evaluation

We evaluate our framework with simulation and real-world experiments.[1] The simulation benchmark is performed on two datasets: a planar 2-link mobile arm in several randomly generated 2D environments as shown in Fig. 10, and a 18-DOF PR2 robot in a simulated indoor environment as shown in Fig. 11. The real-world experiments are performed on a mobile manipulator, with an omni-drive base and a 6-DOF Kinova JACO2 arm, in an indoor environment as shown in Fig. 1.

In our experiments we compare our proposed approach, simultaneous trajectory estimation and planning (`STEAP`), against an open loop execution (`OL`). In `OL` the initial inference solves the GPMP2 planning problem and the planned trajectory is executed without any estimation or replanning until the final time-step or a collision. In our simulation experiments, we also compared against simultaneous localization and planning (`SLAP`) that uses the current measurement factor in the graph, but updates only a truncated version of the graph associated with the future states to replan. This graph also uses the GP prior factors from `STEAP`.

9.1 Benchmark with a planar 2-link mobile arm

For this benchmark we use a simulated 2-link planar mobile arm with base of size 1 m × 0.7 m and link length 0.6 m in an environment of size 30 m × 20 m. The robot is equipped with a simulated 2D laser scanner for localization using ICP. The environment is populated with 20 randomly generated obstacles of size 1 m × 1 m. The graph consists of 30 states from start to goal with 5 interpolated binary obstacle factors between any two states. We compare `OL`, `SLAP` and `STEAP` across different amounts of robot dynamics noise

[1] A video of experiments is available at https://youtu.be/lyayNKV1eAQ.

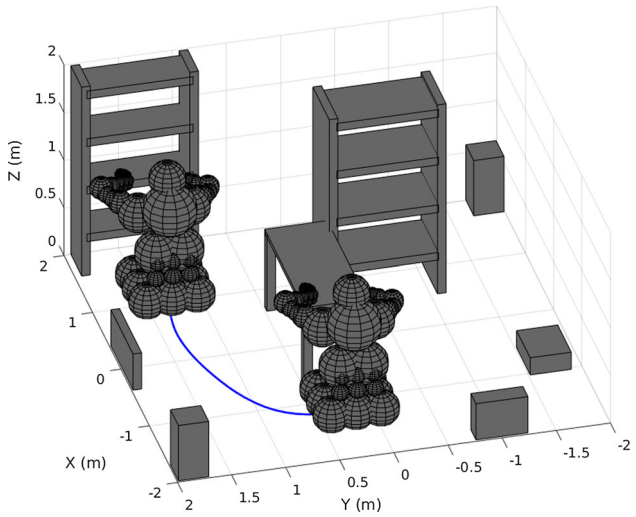


Fig. 11 Workspace setup for PR2 full-body problems. An example problem with start and end robot state, and planned trajectory of the PR2 base is shown. As done in GPMP2, the PR2 body is approximated with a set of spheres for collision checking

(n_{dyn}), implemented as uniform bounded additive noise (m/s) to the robot velocity, and camera measurement noise (n_{cam}), implemented as additive Gaussian noise (m^{-1}) when receiving depth information from the camera on the robot. Each setting is run with 40 distinct seeds (each seed yields a new environment) to account for stochasticity, which are kept the same across all three scenarios. In each trial we record if the trajectory successfully finishes without collision (success), the distance from the goal (goal error) at the end of execution, and L_2 norm of the ground-truth trajectory with the estimated trajectory (estimation error).

The results for this benchmark are summarized in Table 2A–E. The goal and estimation error are aggregates of runs where success is true (the robot reached the goal without colliding with any obstacles). As expected, OL exhibits a low success rate that drops further with an increase in n_{dyn}. SLAP has lower success rates compared to OL; a possible reason is the low precision trajectory estimation from SLAP confuses the robot. Comparatively STEAP has a higher success rate than SLAP and OL, and also follows the decreasing trend with increasing n_{dyn}.

The goal error in STEAP is much lower than SLAP and OL, with the help from high precision trajectory estimation. The trajectory estimation error is significantly smaller with STEAP compared to SLAP and the difference between them increases with increasing n_{cam}. This can be attributed to simultaneously solving the trajectory estimation and planning problems; the motion plan helps in providing a better estimate of the robot's trajectory and in turn a better estimate of the trajectory helps in generating a better motion plan.

9.2 Benchmark with 18-DOF full-body PR2

We additionally evaluated our approach on a simulation dataset in 3D, with a 18-DOF PR2 robot in an indoor setting. The PR2 has two 7-DOF arms (without grippers), a 3-DOF mobile base, 1-DOF linear actuator that moves the torso in the vertical direction, and a 2D laser scanner on the robot base (10 cm above the ground level) used for localization. The environment consists of two cabinets, a desk and several random obstacles so that the laser scanner receives non-trivial readings when the robot is facing outwards. The benchmark consists of a total of 6 tasks, each with a start and a goal state, and the robot is tasked with reaching the goal states from the start states. The environment and an example problem from the benchmark is shown in Fig. 11. We compare OL, SLAP and STEAP across different n_{dyn} and n_{cam} settings, with a graph of 40 states from start to goal. Each setting is run for 120 trials (6 tasks with 20 distinct seeds) and similar metrics as the 2D benchmarks are recorded.

The results for this benchmark are summarized in Table 3A–F, which support the finding in the previous 2D benchmark. We see that STEAP has higher success rates compared to SLAP and OL. This is a result of the lower estimation errors (compared to SLAP) and better quality plans achieved through joint inference. STEAP has a significantly lower estimation errors with larger n_{cam}. The goal error in SLAP and STEAP are smaller than OL since both are feedback approaches that can reduce the error when approaching the goal. We also observe that STEAP has a smaller performance drops caused by a high value of both n_{dyn} and n_{cam}, than SLAP and OL. We also report timing results for this benchmark. On average STEAP takes about 17ms per time-step, which is significantly faster than SLAP with an average of about 130 ms per time-step.

9.3 Experiments with a real robot

Real-world experiments were performed in an 8 m × 6 m indoor environment. Various obstacles (desks, sofas and small objects like boxes and cans) are placed in the environment, to simulate domestic scenes. During the experiments, ground-truth robot trajectories are recorded by an Optitrack motion capture system. A photo of the robot traversing the environment and a map of the environment can be found in Figs. 1 and 9, respectively. Our implementation runs on a desktop computer equipped with Intel 4.0 GHz quad-core CPU, 32 GB memory and one NVIDIA Titan X GPU. Robot sensor data is streamed to the desktop over WiFi, and STEAP commands are streamed back to robot after processing.

We design 2 problems for performance evaluation. In each problem the robot begins from a start configuration and is tasked with reaching the goal configuration. For both problems the graph consists of 50 states from start to goal

Table 2 A: Success rate on planar 2-link mobile arm benchmark, B: Goal translational error (in m), C: Goal rotational error (in rad), D: Estimation translational error (in m), E: Estimation rotational error (in rad)

	$\mathbf{n_{cam}}$		0.02		0.1	
		OL	SLAP	STEAP	SLAP	STEAP
(A)						
$\mathbf{n_{dyn}}$	0.1	0.5250	0.1250	0.5750	0.0750	0.7000
	0.2	0.3250	0.1250	0.6000	0.1000	0.4750
	0.5	0.2500	0.0750	0.3750	0.0500	0.4500
(B)						
$\mathbf{n_{dyn}}$	0.1	0.7463	0.7556	0.3413	0.7067	0.4377
	0.2	0.9715	0.8012	0.4888	1.0793	0.5162
	0.5	1.4179	1.2110	0.6872	1.1295	0.7345
(C)						
$\mathbf{n_{dyn}}$	0.1	0.0805	0.1978	0.0269	0.0497	0.0379
	0.2	0.0952	0.2344	0.0433	0.0847	0.0401
	0.5	0.0952	0.2344	0.0433	0.0847	0.0401

	$\mathbf{n_{cam}}$	0.02		0.1	
		SLAP	STEAP	SLAP	STEAP
(D)					
$\mathbf{n_{dyn}}$	0.1	0.3662	0.1598	0.9217	0.2213
	0.2	0.3644	0.2183	0.8885	0.2242
	0.5	0.3937	0.3065	1.0125	0.3309
(E)					
$\mathbf{n_{dyn}}$	0.1	0.0470	0.0183	0.1287	0.0266
	0.2	0.0480	0.0280	0.1132	0.0297
	0.5	0.0527	0.0450	0.1265	0.0468

with 2 interpolated binary obstacle factors between any two states. Figure 12 shows a screenshot when the robot is running STEAP for problem 1. To evaluate the performance of our STEAP implementation, we performed 10 runs for each problem, in which 5 runs switch the start and goal configurations. We record the planned, estimated and ground-truth trajectories and calculate the same performance criteria as in simulation: success rate, final goal error, and trajectory estimation error.

Table 4 reports performance in these real-world experiments. We first run one-time batch planning by GPMP2 and use an open loop controller to follow the planned trajectory. Since the control command execution on the omnidirectional wheels is noisy, the robot base cannot follow the planned trajectory well, so every run ends with a collision. With the state estimation and replanning provided by STEAP, the robot can follow planned trajectories better, and compensate for perturbations. With STEAP the robot can achieve a 95% overall success rate for the given tasks, with a final translation error of about 14.2 cm in problem 1, and 5.19 cm in problem 2. This goal error is due to the finite-horizon trajectory setup that we use. Since if the robot overshoots when near the end of the trajectory, it may not have enough time steps left to recover. The goal error can be reduced with a receding-horizon formulation of our problem.

In addition to improving planning results, STEAP helps with trajectory estimation. We show the raw localization error in Table 4. Due to the noisy depth measurements, the localization module provides poor estimates of the robot pose. Sometimes the localization module additionally fails due to the scene being out of sensor range (for example when the robot is too close to obstacles). With STEAP, we can reduce the estimation error by about 50–60% as shown in Table 4. In the video of experiments,[1] one can see that although the raw localization positions have significant jumps between each measurement, the estimation results in STEAP are stabilized given previous sensor information and the planned trajectory.

To evaluate the efficiency of our implementation, we time the localization and STEAP modules separately. Timing results show that, in real-world experiments, localization and STEAP modules have average runtimes of 19.3 and 76.0 ms respectively, and maximum runtimes of 30.3 and 149 ms respectively, indicating that our localization implementation

Table 3 A: Success rate on 18-DOF full-body PR2 benchmark, B: Goal translational error (in m), C: Goal rotational error (in rad), D: Estimation translational error (in m), E: Estimation rotational error (in rad), F: Average runtime (in s)

	$\mathbf{n_{cam}}$		0.1		0.5	
		OL	SLAP	STEAP	SLAP	STEAP
(A)						
$\mathbf{n_{dyn}}$	0.02	0.3833	0.7500	0.8333	0.5417	0.6917
	0.1	0.0750	0.3083	0.8583	0.0667	0.6250
	0.5	0.0083	0.0500	0.3500	0.0083	0.3083
(B)						
$\mathbf{n_{dyn}}$	0.02	0.2032	0.0798	0.0426	0.0970	0.0943
	0.1	0.4197	0.1108	0.0440	0.1647	0.1024
	0.5	2.8040	0.1357	0.0936	0.4100	0.1238
(C)						
$\mathbf{n_{dyn}}$	0.02	0.1078	0.0396	0.0219	0.0587	0.0543
	0.1	0.3206	0.0690	0.0261	0.0944	0.0644
	0.5	0.5170	0.0803	0.0473	0.2780	0.0784

	$\mathbf{n_{cam}}$	0.1		0.5	
		SLAP	STEAP	SLAP	STEAP
(D)					
$\mathbf{n_{dyn}}$	0.02	0.0807	0.0287	0.1487	0.0739
	0.1	0.1777	0.0307	0.1749	0.0728
	0.5	0.2100	0.0595	0.2550	0.0858
(E)					
$\mathbf{n_{dyn}}$	0.02	0.0415	0.0168	0.0689	0.0423
	0.1	0.0906	0.0185	0.0790	0.0459
	0.5	0.1007	0.0308	0.1310	0.0475
(F)					
$\mathbf{n_{dyn}}$	0.02	0.1181	0.0141	0.1273	0.0188
	0.1	0.1377	0.0149	0.1353	0.0193
	0.5	0.1474	0.0185	0.1438	0.0195

can easily process the depth image stream at 30 Hz, and run STEAP at $\sim$ 10 Hz.

10 Limitations and future work

The primary limitation of our current work is that it is only applicable in known, static environments. We use an existing map of the environment and precompute a signed distance field for collision checking. In dynamic environments, the map and the signed distance field would need to be constantly updated, which can be a major computational bottle neck, especially in large environments. Using techniques like incremental mapping (Yan et al. 2017), incremental signed distance fields (Oleynikova et al. 2017), and dynamic tracking (Schmidt et al. 2014), we can extend our method to perform SLAM and planning simultaneously online and handle dynamic environments.

Our current implementation is limited to holonomic systems, like the omni-directional mobile base we use in our experiments. It does not support nonholonomic and inequality constraints. But, they can be incorporated as soft constraints with small covariances, for example, the configuration and velocity limit factors (Mukadam et al. 2017c). A sequential quadratic programming type procedure would have to be set up to handle hard constraints.

Our approach is a local trajectory optimization method and is, therefore, prone to local minima. In the context of estimation, trajectory optimization rarely suffers from bad local minimas, since obtaining reasonable initial values is not hard during estimation (e.g. from odometry). In the context of planning, trajectory optimization suffers from bad local minimas (ones that are in collision) primarily in extremely cluttered or maze type environments due to the nature of the optimization methods used. Readers are encouraged to refer to Dong et al. (2016) and Mukadam et al. (2017c) for some

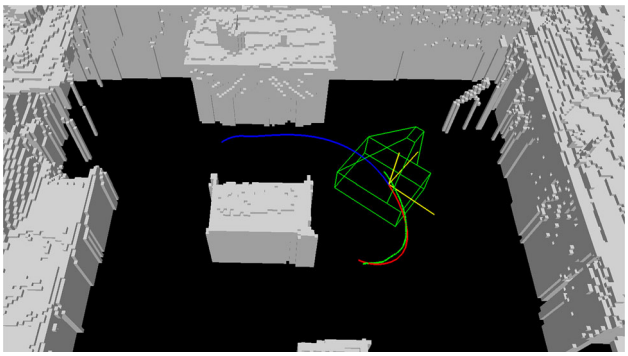


Fig. 12 Visualization of STEAP results on one run of problem 1. The green line is the ground-truth trajectory as determined by the motion capture system, and green robot outline shows the current pose of the Vector robot. The blue line is the planned trajectory and the red line is the estimated trajectory. The yellow axis is current raw pose estimate. The ground plane is cut for visibility. Best viewed in color (Color figure online)

Table 4 Real-world experimental results

	Problem 1	Problem 2
`OL` success rate	0/10	0/10
`STEAP` success rate	9/10	10/10
Goal translation error (cm)	14.20	5.19
Localization error (cm)	7.07	6.45
Trajectory estimation error (cm)	3.48	2.53

quantitative evaluations on several commonly used trajectory optimization techniques. In case of batch optimization i.e. the first inference step, several ideas like random initializations and graph-based initializations (Huang et al. 2017) exist to improve results. However, there are no known methods to address this problem during incremental inference using the Bayes tree, except by re-solving a new batch optimization problem, which will be computationally expensive. Developing new incremental algorithms that are better able to contend with local minima is an interesting research direction.

Our approach is capable of fusing information from multiple sensors in an asynchronous fashion using GP interpolation (Yan et al. 2017), we are interested in exploring this capability more fully. Finally, as discussed in the results section, a receding horizon formulation of STEAP would help reduce goal-errors and better support navigation with exploration. We leave this as future work.

11 Conclusion

We formulate the problem of simultaneous trajectory estimation and planning (STEAP) as probabilistic inference. By representing the prior distribution of a continuous-time trajectory and likelihood function of costs and observations with factor graphs, we can efficiently perform inference to compute the posterior distribution of the trajectory. We solve STEAP in an online setting to simultaneously estimate and smooth the trajectory history as well as replan for the future trajectory as new information is encountered. This is made possible by performing efficient incremental inference to update the previous solution. We conducted experiments in simulation and on a real mobile manipulator and showed that our framework is able to perform in real-time and robustly handle the stochasticity associated with execution. Our results demonstrate that this framework is suitable for online applications with high-degree-of-freedom systems in known, static real-world environments.

Acknowledgements This work was partially supported by NSF NRI award 1637908 and National Institute of Food and Agriculture, USDA, award 2014-67021-22556. The authors thank Muhammad Asif Rana, David Kent, Vivian Chu, and Sonia Chernova for access to and help with the Vector robot.

Funding NSF NRI Award 1637908 and National Institute of Food and Agriculture, USDA, Award 2014-67021-22556.

Compliance with ethical standards

Conflict of interest Author Dellaert is affiliated with Facebook, and author Boots is affiliated with Google.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

Informed consent No informed consent was required.

References

Agha-mohammadi, A. A., Agarwal, S., Chakravorty, S., & Amato, N. M. (2015). *Simultaneous localization and planning for physical mobile robots via enabling dynamic replanning in belief space*. arXiv:1510.07380

Anderson, S., & Barfoot, T. D. (2015) Full STEAM ahead: Exactly sparse Gaussian process regression for batch continuous-time trajectory estimation on SE(3). In *IEEE/RSJ international conference on intelligent robots and systems (IROS)* (pp 157–164). IEEE.

Anderson, S., Barfoot, T. D., Tong, C. H., & Särkkä, S. (2015). Batch nonlinear continuous-time trajectory estimation as exactly sparse gaussian process regression. *Autonomous Robots*, *39*(3), 221–238.

Attias, H. (2003). Planning by probabilistic inference. In *International conference on artificial intelligence and statistics (AISTATS)*.

Barfoot, T., Tong, C. H., & Sarkka, S. (2014). Batch continuous-time trajectory estimation as exactly sparse Gaussian process regression. In *Science and Systems (RSS): Robotics*.

Camacho, E. F., & Alba, C. B. (2013). *Model predictive control*. Berlin: Springer.

Carlone, L., Kira, Z., Beall, C., Indelman, V., & Dellaert, F. (2014). Eliminating conditionally independent sets in factor graphs: A unifying perspective based on smart factors. In *IEEE international conference on robotics and automation (ICRA)* (pp. 4290–4297). IEEE.

Censi, A. (2007). An accurate closed-form estimate of ICP's covariance. In *IEEE International Conference on Robotics and Automation (ICRA)* (pp. 3167–3172). IEEE.

Chirikjian, G. S. (2011). *Stochastic models, information theory, and Lie groups, volume 2: Analytic methods and modern applications* (Vol. 2). Berlin: Springer.

Davis, T. A., Gilbert, J. R., Larimore, S. I., & Ng, E. G. (2004). A column approximate minimum degree ordering algorithm. *ACM Transactions on Mathematical Software (TOMS)*, *30*(3), 353–376.

Dellaert, F., & Kaess, M. (2006). Square root SAM: Simultaneous localization and mapping via square root information smoothing. *International Journal of Robotics Research*, *25*(12), 1181–1203.

Dong, J., Mukadam, M., Dellaert, F., & Boots, B. (2016). Motion planning as probabilistic inference using Gaussian processes and factor graphs. In *Robotics: Science and Systems (RSS)*

Dong, J., Boots, B., & Dellaert, F. (2017a). *Sparse Gaussian processes for continuous-time trajectory estimation on matrix Lie groups*. arXiv:1705.06020

Dong, J., Burnhan, J., Boots, B., Rains, G., & Dellaert, F. (2017b). 4D crop monitoring: Spatio-temporal reconstruction for agriculture. In *IEEE International Conference on Robotics and Automation (ICRA)*

Felzenszwalb, P., & Huttenlocher, D. (2012). *Distance transforms of sampled functions* (p. 19). Technical Report

Ferguson, D., Kalra, N., Stentz, A. (2006) Replanning with RRTs. In *IEEE International Conference on Robotics and Automation (ICRA)* (pp. 1243–1248). IEEE.

Forster, C., Carlone, L., Dellaert, F., & Scaramuzza, D. (2015) IMU preintegration on manifold for efficient visual-inertial maximum-a-posteriori estimation. In *Robotics: Science and Systems (RSS)*

Huang, E., Mukadam, M., Liu, Z., & Boots, B. (2017). Motion planning with graph-based trajectories and Gaussian process inference. In *IEEE international conference on robotics and automation (ICRA)*

Izadi, S., Kim, D., Hilliges, O., Molyneaux, D., Newcombe, R., Kohli, P., Shotton, J., Hodges, S., Freeman, D., & Davison, A., et al. (2011). KinectFusion: Real-time 3D reconstruction and interaction using a moving depth camera. In *Proceedings of ACM symposium on user interface software and technology (UIST)* (pp. 559–568)

Kaess, M., Ranganathan, A., & Dellaert, F. (2008). iSAM: Incremental smoothing and mapping. *IEEE Transaction on Robotics*, *24*(6), 1365–1378.

Kaess, M., Ila, V., Roberts, R., Dellaert, F. (2011). The Bayes tree: An algorithmic foundation for probabilistic robot mapping. In *Algorithmic Foundations of Robotics* (pp. 157–173). Berlin: Springer.

Kaess, M., Johannsson, H., Roberts, R., Ila, V., Leonard, J. J., & Dellaert, F. (2012). iSAM2: Incremental smoothing and mapping using the Bayes tree. *International Journal of Robotics Research*, *31*(2), 216–235.

Kalman, R. E., et al. (1960). A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, *82*(1), 35–45.

Koenig, S., & Likhachev, M. (2005). Fast replanning for navigation in unknown terrain. *IEEE Transaction on Robotics*, *21*(3), 354–363.

Kschischang, F. R., Frey, B. J., & Loeliger, H. A. (2001). Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, *47*(2), 498–519.

Leandro, R., Santos, M., & Cove, K. (2005). An empirical approach for the estimation of GPS covariance matrix of observations. In: *ION 61st annual meeting The MITRE corporation & draper laboratory* (pp. 27–29).

Levine, S., & Koltun, V. (2013) Variational policy search via trajectory optimization. In *Advances in neural information processing systems* (pp. 207–215)

Mukadam, M. (2017). *PIPER (Online)*. https://github.com/gtrll/piper

Mukadam, M., Yan, X., & Boots, B. (2016). Gaussian process motion planning. In *IEEE international conference on robotics and automation (ICRA)* (pp. 9–15).

Mukadam, M., Cheng, C. A., Yan, X., Boots, B. (2017a). Approximately optimal continuous-time motion planning and control via probabilistic inference. In *IEEE international conference on robotics and automation (ICRA)*

Mukadam, M., Dong, J., Dellaert, F., Boots, B. (2017b). Simultaneous trajectory estimation and planning via probabilistic inference. In *Robotics: Science and Systems (RSS)*.

Mukadam, M., Dong, J., Yan, X., Dellaert, F., & Boots, B. (2017c). *Continuous-time Gaussian process motion planning via probabilistic inference*. arXiv preprint arXiv:1707.07383

Nickolls, J., Buck, I., Garland, M., & Skadron, K. (2008). Scalable parallel programming with CUDA. *Queue*, *6*(2), 40–53. https://doi.org/10.1145/1365490.1365500.

Oleynikova, H., Taylor, Z., Fehr, M., Siegwart, R., & Nieto, J. (2017). *Voxblox: Incremental 3D euclidean signed distance fields for on-board mav planning*. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Penny, W. (2014). Simultaneous localisation and planning. In *4th International workshop on cognitive information processing (CIP)* (pp. 1–6). IEEE.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A. Y. (2009). ROS: an open-source Robot Operating System. In *ICRA workshop on open source software, Kobe* (Vol. 3, p. 5)

Rafieisakhaei, M., Chakravorty, S., Kumar, P. (2016). *Non-Gaussian slap: Simultaneous localization and planning under non-Gaussian uncertainty in static and dynamic environments*. arXiv:1605.01776

Rana, M. A., Mukadam, M., Ahmadzadeh, S. R., Chernova, S., Boots, B. (2017). Towards robust skill generalization: Unifying learning from demonstration and motion planning. In *Proceedings of the 1st Annual Conference on Robot Learning, PMLR* (Vol. 78, pp. 109–118).

Rawlik, K., Toussaint, M., & Vijayakumar, S. (2012). On stochastic optimal control and reinforcement learning by approximate inference. In *Science and Systems (RSS): Robotics*.

Schmidt, T., Newcombe, R. A., & Fox, D. (2014). DART: Dense articulated real-time tracking. In *Robotics: Science and Systems (RSS)*

Ta, D. N., Kobilarov, M., Dellaert, F. (2014). A factor graph approach to estimation and model predictive control on unmanned aerial vehicles. In International conference on unmanned aircraft systems (ICUAS) (pp. 181–188). IEEE.

Thrun, S., Burgard, W., & Fox, D. (2005). *Probabilistic robotics*. Cambridge: MIT press.

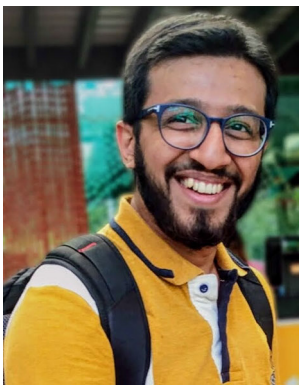
Todorov, E. (2008). General duality between optimal control and estimation. In *IEEE Conference on Decision and Control* (pp. 4286–4292). IEEE.

Tong, C. H., Furgale, P., & Barfoot, T. D. (2013). Gaussian process Gauss–Newton for non-parametric simultaneous localization and mapping. *International Journal of Robotics Research*, *32*(5), 507–525.

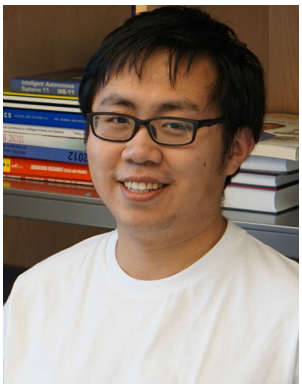
Toussaint, M. (2009). Robot trajectory optimization using approximate inference. In *International Conference on Machine Learning (ICML)* (pp 1049–1056). ACM.

Toussaint, M., Storkey, A. (2006). Probabilistic inference for solving discrete and continuous state Markov decision processes. In *International Conference on Machine Learning (ICML)* (pp 945–952). ACM.

Yan, X., Indelman, V., & Boots, B. (2017). Incremental sparse GP regression for continuous-time trajectory estimation and mapping. *Robotics and Autonomous Systems*, *87*, 120–132.



Mustafa Mukadam is currently a Ph.D. student in Robotics at Georgia Institute of Technology, working with Byron Boots in the Institute for Robotics and Intelligent Machines. He received his M.S. degree in Aerospace Engineering from University of Illinois at Urbana-Champaign. His research focuses on motion planning, estimation, and learning from demonstration, often employing probabilistic and machine learning tools, for autonomous navigation and mobile manipulation.



Jing Dong is a Ph.D. student in Computer Science at Georgia Institute of Technology. Prior to joining Georgia Tech, he got his bachelor degree in Engineering Mechanics and Aerospace Engineering from Tsinghua University, Beijing, China. His current research interests cover various topics in robotics and computer vision, including but not limited to Simultaneous Localization and Mapping (SLAM), 3D reconstruction, visual feature learning and matching, and real-time motion planning.

Frank Dellaert is a Professor in the School of Interactive Computing at Georgia Tech. He obtained his Ph.D. from Carnegie Mellon University in 2001. He has applied Markov chain Monte Carlo sampling methodologies in a variety of novel settings, including the Monte Carlo localization method for estimating and tracking the pose of robots, now a standard and popular tool in mobile robotics. His current research interests lie in the overlap of Robotics and Computer vision, and he is particularly interested in graphical model techniques to solve large-scale problems in mapping and 3D reconstruction. The GTSAM toolbox which embodies many of the ideas his group has worked on in the past few years is available as open source, and uses factor graphs as the unifying graphical model language to tie together many optimization problems in computer vision, robotics, and even discrete optimization.

Byron Boots is an Assistant Professor in the College of Computing at Georgia Tech. He directs the Georgia Tech Robot Learning Lab, affiliated with the Center for Machine Learning and the Institute for Robotics and Intelligent Machines. Byron's research focuses on development of theory and systems that tightly integrate perception, learning, and control. He received his Ph.D. in Machine Learning from Carnegie Mellon University and was a postdoctoral researcher in Computer Science and Engineering at the University of Washington. His research has won several awards including Best Paper at ICML in 2010.