# Simultaneous Trajectory Estimation and Planning via Probabilistic Inference

Mustafa Mukadam, Jing Dong, Frank Dellaert, and Byron Boots
Institute for Robotics & Intelligent Machines, Georgia Institute of Technology, Atlanta, GA, USA
{mmukadam3,jdong}@gatech.edu, {frank,bboots}@cc.gatech.edu

***Abstract*—We provide a unified probabilistic framework for trajectory estimation and planning. The key idea is to view these two problems, usually considered separately, as a single problem. At each time-step the robot is tasked with finding the complete continuous-time trajectory from start to goal. This can be quite difficult; 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, and the robot must find the solution in (faster than) real time. To overcome these challenges, we build 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. Finally, we evaluate our framework empirically in both simulation and on a mobile manipulator.**

## I. Introduction & Related Work

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 paper, we provide a unified approach to trajectory estimation and planning. The key idea is to view these two problems, usually considered separately, as a single problem. At each time-step the robot is tasked with finding the complete continuous-time trajectory from start to goal. This can be quite difficult; 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, and the robot must find the solution in (faster than) real time.

To overcome these challenges, we build 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* (MAP) trajectory conditioned on all available sensor data and cost information. In brief, we represent continuous-time trajectories as samples from a Gaussian process (GP) [5, 22] and then formulate the estimation and planning problem on a single probabilistic graphical model. We perform inference on the graph using numerical tools that solve a nonlinear least squares optimization problem, and generate fast, iterative solutions by exploiting the structure of the problem.

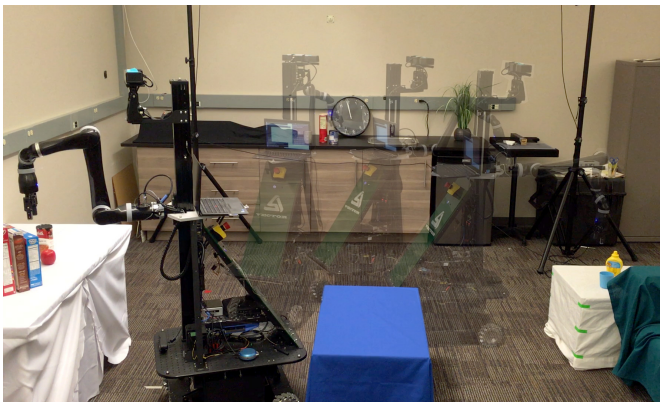


Fig. 1: The Vector mobile manipulator, with an omni-drive base and a 6-DOF Kinova JACO2 arm, is solving the STEAP problem. The task involves picking up an object from the white table on the right and dropping it off on the white table on the left, while avoiding the blue table. The semi-transparent robots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

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 [6] that formulates SLAM as inference in a factor graph [19] and exploits the sparsity of the underlying large-scale linear systems to perform inference efficiently. Given new sensor data, incremental Smoothing and Mapping (iSAM) [14, 16] exploits the structure of the problem to efficiently update the solution rather than resolving the entire problem from scratch. Recently, Tong et al. [32] 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). Tong et al.'s approach was extended in Barfoot et al. [5] to take advantage of the sparse structure inherent in the STEAM problem, in Yan et. al [35] to efficiently incrementally update the solution, and in Dong et al. [9] to 4D mapping problems. The resulting algorithms speed up solution time and can be viewed as continuous-time

analogs of Dellaert's original square-root SAM algorithm [6] and Kaess et al.'s iSAM2 algorithm [16].

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 [17], but the solution 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 [31] with some early work in this direction looking at solving Markov decision processes (MDP) [4]. Several researchers have recently proposed a probabilistic inference perspective on planning and control problems, leveraging expectation maximization [34, 20], expectation propagation [33], KL-minimization [28], and efficient inference in factor graphs [7, 12, 23]. Interestingly, the incremental inference technique [15] used in [7] to solve replanning problems is the same as originally used in [16] 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 [18, 11] 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 [25], more recent approaches designed for dynamic environments [1, 27], and new approaches that combine state estimation and model predictive control [29].

In this work, we tackle the simultaneous trajectory estimation and planning (STEAP) problem within a unified probabilistic inference framework. The STEAP problem is 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 an updated plan. Following Yan et al. [35] and Dong et al. [7], we represent the trajectory as a continuous-valued function mapping time $t$ to robot states $\boldsymbol{\theta}(t)$, and seek the MAP function with an incremental solver [16]. This allows us to avoid resolving the STEAP problem from scratch as new observations are encountered and only update the trajectory where required, dramatically reducing the overall computational burden of our approach and enabling a faster-than-realtime solution. To better accommodate mobile manipulation problems, we build on Barfoot et al.'s recent work on continuous-time trajectory estimation on SE(3) [2] and extend Dong et al. [7] to plan trajectories on Lie groups. Finally, we implement our probabilistic inference framework for solving the STEAP problem 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.

## II. Background: Trajectory Optimization as Probabilistic Inference

Following previous work on both STEAM problems [5, 35] and Gaussian process motion planning [7, 22], 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.

### A. The Trajectory Prior

A prior distribution over trajectories can be defined as a vector-valued Gaussian process $\boldsymbol{\theta}(t) \sim \mathcal{GP}(\boldsymbol{\mu}(t), \boldsymbol{\mathcal{K}}(t,t'))$, where $\boldsymbol{\mu}(t)$ is a vector-valued mean function and $\boldsymbol{\mathcal{K}}(t,t')$ 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

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

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

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

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

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

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. that the robot velocity changes smoothly [5, 32]. In motion planning, the prior is selected to encourage higher-order derivatives of the system configuration to be minimized [7]. The prior we use in our implementation is detailed in Sec. III-C1.

### B. The 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 [5], while in motion planning problems the likelihood function encourages posterior trajectories to be collision-free [7].

Let $\mathbf{e}$ be a collection of random binary events. Examples of events include collision, receiving a sensor reading, 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 an event or a measurement $\mathbf{e}$ 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 Section III-C.

### C. Computing the MAP Trajectory

Given Eqs. (3)-(4), the goal is to compute the *maximum a posteriori* (MAP) trajectory

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \{p(\boldsymbol{\theta})p(\mathbf{e}|\boldsymbol{\theta})\} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \{-\log\left(p(\boldsymbol{\theta})p(\mathbf{e}|\boldsymbol{\theta})\right)\} \quad (5)$$

$$= \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \left\{ \frac{1}{2} \parallel \boldsymbol{\theta} - \boldsymbol{\mu} \parallel^2_{\boldsymbol{\mathcal{K}}} + \frac{1}{2} \parallel \boldsymbol{h}(\boldsymbol{\theta}, \mathbf{e}) \parallel^2_{\boldsymbol{\Sigma}} \right\} \quad (6)$$

where Eq. (6) follows from Eq. (3) and Eq. (4). The MAP estimation problem can therefore be reduced to a nonlinear least squares problem and solved with tools like Gauss-Newton or Levenberg-Marquardt.

## III. Simultaneous Trajectory Estimation and Planning with Factor Graphs

The MAP trajectory computation in Section II-C can be executed efficiently by exploiting known structure in the problem. In particular, the prior and the likelihood functions can be *factored* into a product of functions that is organized as a bipartite *factor graph* $G = \{\boldsymbol{\Theta}, \mathcal{F}, \mathcal{E}\}$,

$$p(\boldsymbol{\theta})p(\mathbf{e}|\boldsymbol{\theta}) \propto \prod_i f_i(\boldsymbol{\Theta}_i). \quad (7)$$

The variables $\boldsymbol{\Theta} \doteq \{\boldsymbol{\theta}_0, \ldots, \boldsymbol{\theta}_N\}$ are a set of instantaneous robot states along the trajectory, the *factors* $\mathcal{F} \doteq \{f_0, \ldots, f_M\}$ are functions on variable subsets $\boldsymbol{\Theta}_i$ of $\boldsymbol{\Theta}$, and $\mathcal{E}$ are edges connected to the two types of nodes. Thus, we can write the posterior distribution as a product of the factors that collectively represent the prior and the likelihood

$$p(\boldsymbol{\theta}|\mathbf{e}) \propto f^{prior}(\boldsymbol{\Theta}) f^{like}(\boldsymbol{\Theta}). \quad (8)$$

In the remainder of this section we describe the factor graph formulation of and relationship between Smoothing and Mapping (SAM) [6], Simultaneous Trajectory Estimation and Mapping (STEAM) [5, 2], Gaussian Process Motion Planning 2 (GPMP2) [7], Simultaneous Localization and Planning (SLAP) [25, 1], and our proposed method, Simultaneous Trajectory Estimation and Planning (STEAP). We then provide details on the factors used in the STEAP problem, discuss incremental inference and summarize the STEAP approach with a simple toy example.

### A. Factorization in Related Problems

We begin with the smoothing and mapping (SAM) [6] problem, an early work that uses factor graphs to address state estimation problems in robotics. The goal is to estimate the full posterior trajectory given measurements. The factor graph used in SAM is

$$p(\boldsymbol{\theta}_{est}|\mathbf{e}) \propto f^{prior} f^{meas}, \quad (9)$$

where $f^{prior}$ is the prior on the first state $f^{prior} = f^{prior}(\boldsymbol{\theta}_0)$, and $f^{meas}$ is the likelihood of all sensor measurements, which itself factors as

$$f^{meas} = \prod_i f_i^{meas}(\boldsymbol{\Theta}_i). \quad (10)$$

TABLE I: 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 : $f^{prior} f^{meas}$<br>Planning : $f^{prior} f^{obs} f^{fix}$ |
| STEAP | Estimation + Planning | $f^{gp} f^{meas} f^{obs} f^{fix}$ |

Like SAM, Simultaneous trajectory estimation and mapping (STEAM) [5, 2] addresses trajectory estimation problems. The key difference is that in STEAM, the trajectory is no longer treated as a discrete sequence of states $\boldsymbol{\Theta}$, but rather a continuous-time trajectory sampled from a GP. The prior is a joint distribution on the full trajectory $f^{prior} = f^{gp}(\boldsymbol{\Theta})$, yielding a factorization

$$p(\boldsymbol{\theta}_{est}|\mathbf{e}) \propto f^{gp} f^{meas}. \quad (11)$$

GPMP2 [7] is a probabilistic inference framework for solving planning problems that utilizes the GP trajectory representation from STEAM. The goal is to find collision-free future trajectories that satisfy the GP prior. 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

$$f^{obs} = \prod_i f_i^{obs}(\boldsymbol{\theta}_i). \quad (12)$$

A fixed start and goal state is also required in planning problems, and therefore incorporated in to the likelihood. So factors to fix start and goal configurations are also employed

$$f^{fix} = f^{start}(\boldsymbol{\theta}_0) f^{goal}(\boldsymbol{\theta}_N). \quad (13)$$

The full factor graph of GPMP2 is, therefore

$$p(\boldsymbol{\theta}_{plan}|\mathbf{e}) \propto f^{gp} f^{obs} f^{fix}. \quad (14)$$

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 [25, 1]. Although previous work in this area does not employ factor graphs, we explain 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

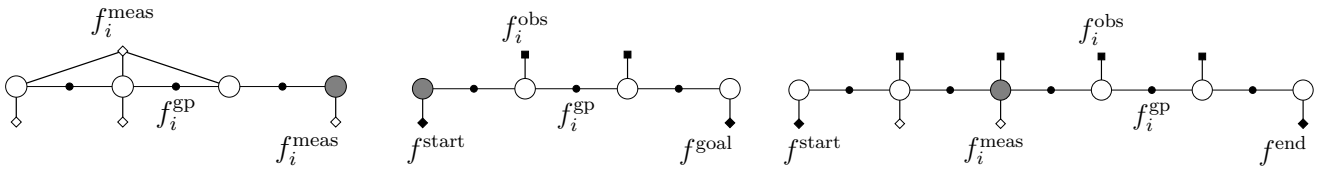
$$p(\boldsymbol{\theta}_{est}|\mathbf{e}) \propto f^{prior} f^{meas}, \quad (15)$$

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

SLAP solves the estimation graph first to generate an estimate of the current state and then uses the MAP estimate of the current state to initialize and solve the planning problem.

### B. Simultaneous Trajectory Estimation and Planning (STEAP)

We now formally define the STEAP problem: STEAP solves state estimation and planning problems simultaneously like SLAP, but instead of splitting the factor graph in to two pieces,




(a) (b) (c)

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

STEAP perform inference on the entire factor graph at once. The STEAP factor graph is defined as,

$$p(\boldsymbol{\theta}|\mathbf{e}) \propto f^{gp} f^{meas} f^{obs} f^{fix}. \tag{17}$$

By solving a STEAP problem, we solve for the full trajectory $\boldsymbol{\theta} = \boldsymbol{\theta}_{est} \cup \boldsymbol{\theta}_{plan}$. There are two major advantages of using STEAP as compared to SLAP:

(i) Optimization of a single graph allows information flow between the two sub-graphs of estimation and planning, which is not possible with SLAP. This increases performance in both estimation and planning. For example, 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) localization and reduces estimation errors. Similarly, the trajectory estimation information corrects the estimate of the current robot position, providing feedback for the planned future trajectory.

(ii) The number of variables in a STEAP factor graph do not change much during execution i.e. only a few measurement factors are added in each step. This allows for very efficient incremental inference using the Bayes tree algorithm [16]. Recomputing 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 Sec. III-D. Factorization of the various problems is summarized in Table. I, and their factor graphs are shown in Fig. 2.

### C. STEAP Factor Definitions

*1) The Gaussian process prior factor:* A Gaussian RBF kernel defines a prior distribution of trajectories with no pair-wise independences. In other words, all states are connected to a single GP prior factor, $f^{gp} = f^{gp}(\boldsymbol{\Theta})$. This prior cannot be factored, and destroys the problem's sparsity, making inference computationally expensive. However, Barfoot et al. [5] 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

$$f^{gp} = \prod_i f_i^{gp}(\boldsymbol{\theta}_i, \boldsymbol{\theta}_{i+1}) \tag{18}$$

where any GP prior factor connects to only its two neighboring states, forming a (Gauss-Markov) chain.

In GPMP2 [7], the GP prior on trajectories is generated by a LTV-SDE defined on a vector space. This is shown in Fig. 2 (b) where states (white circle) form a chain by connecting to GP prior factors (black circle). A similar representation was used earlier to define GP prior on STEAM problems [5] (Fig. 2 (c)). GP priors have also been formulated with non-linear SDEs [3] and on the SE(3) Lie group [2].

If the robot configuration is in vector space $\mathbb{R}^n$, like GPMP2, STEAP can use the GP prior defined in [5]. But we develop STEAP for mobile manipulators that have their configuration space defined by a Lie group product $\mathbf{x} \in \text{SE(2)}\times\mathbb{R}^n$ where $n$ is the degree of freedom of the arm and the SE(2) Lie group defines a planar translation ($x$ and $y$) 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 [8] for details about the GP prior that we use for Lie groups.

*2) Obstacle factors:* All obstacle factors are constructed similarly to GPMP2 [7] 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 [7] for details.

*3) Start and goal factor:* These are multivariate Gaussian factors

$$f^{start}(\boldsymbol{\theta}_0) = \exp\left\{ -\frac{1}{2} \parallel \boldsymbol{\theta}_0 - \boldsymbol{\theta}_{start} \parallel^2_{\Sigma_{fix}} \right\} \tag{19}$$

$$f^{goal}(\boldsymbol{\theta}_N) = \exp\left\{ -\frac{1}{2} \parallel \boldsymbol{\theta}_N - \boldsymbol{\theta}_{goal} \parallel^2_{\Sigma_{fix}} \right\} \tag{20}$$

with the mean as the start or goal and a small covariance $\Sigma_{fix}$, 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.

*4) Measurement factor:* Since there are many types of sensors that provide different measurements, there are many types of measurement factors [6]. We use a multivariate Gaussian measurement factor for the current state measurement

$$f_i^{meas}(\boldsymbol{\theta}_i) = \exp\left\{ -\frac{1}{2} \parallel \boldsymbol{\theta}_i - \boldsymbol{\mu}_i^{meas} \parallel^2_{\Sigma_{meas}} \right\} \tag{21}$$

where the measurement queried from sensors has mean $\boldsymbol{\mu}_i^{meas}$ with covariance $\Sigma_{meas}$.

### D. Incremental Inference

In Sec. II-C we discussed how to solve the MAP inference problem as non-linear least squares optimization. But one

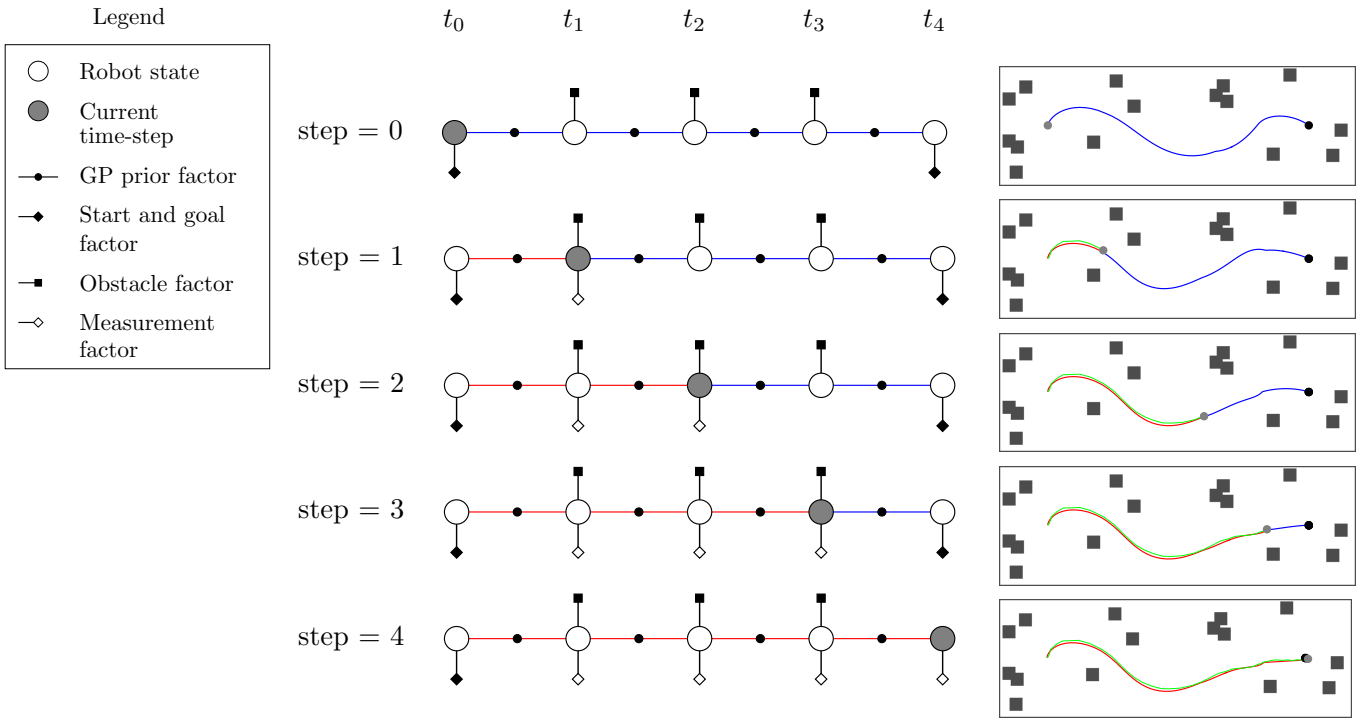



Fig. 3: A simple example illustrates STEAP using a robot (gray) that navigates to the goal (black circle) while avoiding obstacles. At each step the right side shows the environment with ground-truth (green), estimated (red), and replanned (blue) trajectories. The left side shows the corresponding factor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

significant drawback of using non-linear least squares is that with every update the problem must be completely 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. proposed efficient updates with the Bayes tree data structure [16]. When re-solving a graph with only minor changes (in variables or factors), only the parts of the solution associated with the changes will be updated, leaving most of the solution 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 added as 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.

### *E. A STEAP Example*

We use an example, illustrated in Fig. 3, to describe how STEAP works using Algorithm 1. In this example, a robot with stochastic dynamics starts at time $t_0$ and needs to reach goal at time $t_4$ 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. The current time-step is shown in gray. Note that the small size of the graph is just for illustration, in practice our approach can handle very large graphs (see Section V for the graph sizes used in our experiments). 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 [7] for details). The start, goal and obstacle factors together form the likelihood. Next, 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 factor 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 is then executed on the robot. The ground-truth trajectory is shown in green. Since the system is stochastic, execution is noisy. We make an observation to get a measurement factor and insert it into the graph at $t_1$. This new factor is combined with the old likelihood to produce the 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

**Algorithm 1** STEAP

Initialize $\boldsymbol{\theta}$
$\mathcal{FG}$ = updateFactorGraph($f^{gp}$, $f^{obs}$, $f^{fix}$)
$\boldsymbol{\theta}$ = incrementalInference($\mathcal{FG}$, $\boldsymbol{\theta}$)
**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()
    $\mathcal{FG}$ = updateFactorGraph($\mathcal{FG}$, $f_{i+1}^{meas}$)
    $\boldsymbol{\theta}$ = incrementalInference($\mathcal{FG}$, $\boldsymbol{\theta}$)
  **else**
    **return** failure
  **end if**
**end for**
**return** success

robot until time $t_1$ 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.

## IV. Implementation Details

We implement STEAP within the PIPER [21] package using ROS [26] and GPMP2 [7] and have open-sourced the code. Fig. 4 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 Section IV-C.

### A. The STEAP Module

This module constructs the factor graph for the problem using the robot and problem configuration and iteratively performs inference with the changing factor graph. At any step the replanned solution is upsampled and checked for safety, and is sent to `Robot Module`. When `Localization Module` returns a current pose measurement, new measurement factors are added to the graph and the updated posterior is evaluated. This procedure repeats until the full trajectory completes

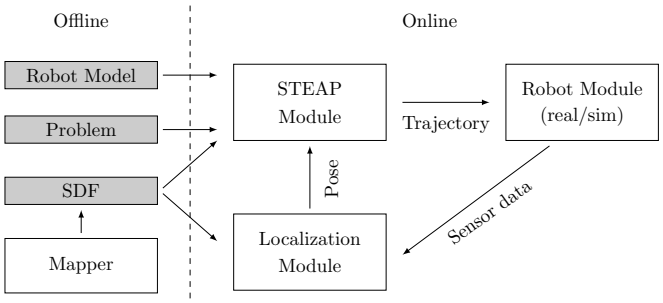



Fig. 4: Block diagram of our framework showing all the components and how they interact. White boxes are modules and gray boxes are data. Sensor measurements flow from `Robot Module` to `Localization Module`.

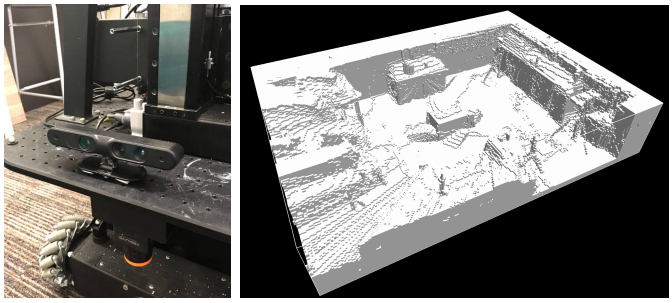


Fig. 5: Left: the PrimeSense depth camera mounted on the robot base. Right: one 8m×6m×1.5m occupancy grid map built by the mapper module.

execution. Given the generic implementation of this module, our framework can be used for any simulated or real robot as long as the robot information is provided in the offline phase.

### B. The Robot Module

This module consists of the robot API and controllers that can understand and execute the trajectory passed by `STEAP Module` for 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 simulation adjustable random noise is mixed in both system dynamics and sensor measurements, to simulate the real-world stochasticity.

### C. The Localization Module

The Localization module reads raw sensor data from `Robot Module`, calculates a pose estimate of the robot, and provides this information to `STEAP Module`. We can use any SLAM pipeline as a Localization module, and we choose an ICP-style iterative approach, similar to the tracking in KinectFusion [13], to achieve real-time performance on the entire system. Tracking is executed on the full SDF generated by `Mapper`. We use CUDA [24] 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 in our experiments.

### D. The Mapper

Obstacle factors require signed distances to calculate obstacle cost. Although we can calculate a SDFs from CAD models

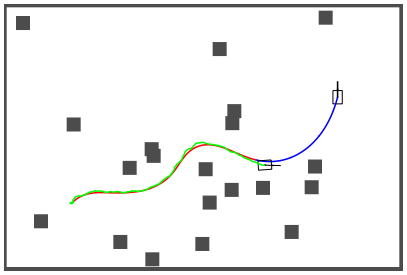


Fig. 6: STEAP result on an example from the simulation benchmark with a planar 2-link mobile arm. Ground-truth (green), estimated past (red) and replanned future (blue) trajectories are shown with the current robot pose between the red and blue trajectories and the goal at the end of the blue trajectory. Best viewed in color.

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. 5, in real-world experiments), and an occupancy grid mapping approach [30, Ch.9] to generate the signed distance field. 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 $p_o$s 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. After the occupancy grid mapping, we calculate the signed distance field by efficient distance transformation [10]. We use CUDA to implement occupancy grid mapping and distance transformation, allowing us to achieve real-time performance on scenes roughly of size 8m×6m×1.5m with 3cm resolution. A map constructed with this approach is shown in Fig. 5.

## V. Evaluation

We conduct experiments[1] and demonstrate our framework on a 2-link planar mobile arm in simulation, shown in Fig. 6 and a mobile manipulator, show in Fig. 1 consisting of an omni-drive base and a 6-DOF Kinova JACO2 arm.

### A. Benchmark With a 2-link Planar Mobile Arm

We use a simulated benchmark to compare three scenarios. First, an open loop execution (OL) that, after the initial inference, executes the planned trajectory without any estimation or replanning. Second, 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, and, finally, our proposed simultaneous trajectory estimation and planning framework (STEAP) that performs inference on the full factor graph, updating the complete MAP estimate of the trajectory with every new incoming measurement factor.

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

Table II.A Success rate.

| | $\mathbf{n_{cam}}$ | | **0.0001** | | **0.0005** | | **0.001** | |
|---|---|---|---|---|---|---|---|---|
| | | OL | SLAP | STEAP | SLAP | STEAP | SLAP | STEAP |
| $\mathbf{n_{dyn}}$ | **0.05** | 0.4 | 1 | 0.975 | 0.975 | 1 | 1 | 0.925 |
| | **0.1** | 0.275 | 1 | 0.925 | 1 | 1 | 0.975 | 0.95 |
| | **0.5** | 0.1 | 0.875 | 0.775 | 0.875 | 0.875 | 0.85 | 0.85 |

Table II.B Goal error.

| | $\mathbf{n_{cam}}$ | | **0.0001** | | **0.0005** | | **0.001** | |
|---|---|---|---|---|---|---|---|---|
| | | OL | SLAP | STEAP | SLAP | STEAP | SLAP | STEAP |
| $\mathbf{n_{dyn}}$ | **0.05** | 3.01 | 0.526 | 0.436 | 0.239 | 0.422 | 0.219 | 0.269 |
| | **0.1** | 2.9 | 0.646 | 0.236 | 0.322 | 0.39 | 0.338 | 0.296 |
| | **0.5** | 3.71 | 0.655 | 0.533 | 0.38 | 0.535 | 0.399 | 0.428 |

Table II.C Estimation error.

| | $\mathbf{n_{cam}}$ | **0.0001** | | **0.0005** | | **0.001** | |
|---|---|---|---|---|---|---|---|
| | | SLAP | STEAP | SLAP | STEAP | SLAP | STEAP |
| $\mathbf{n_{dyn}}$ | **0.05** | 2.46 | 1.41 | 2.68 | 1.3 | 2.76 | 1.51 |
| | **0.1** | 2.52 | 1.25 | 2.36 | 1.21 | 2.67 | 0.686 |
| | **0.5** | 1.92 | 0.943 | 1.77 | 1.18 | 2.51 | 0.64 |

For this benchmark we use a simulated 2-link planar mobile arm with base of size $1m \times 0.7m$ and link length $0.6m$ in an environment of size $30m \times 20m$. The environment is populated with 20 randomly generated obstacles of size $1m \times 1m$. 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 ($n_{dyn}$), implemented as additive noise to the robot velocity, and camera noise ($n_{cam}$), implemented as additive noise 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 II.A–II.C. The goal and estimation error are aggregates of runs where success is true (the robot reached the goal without colliding with an obstacle). As expected, OL performs poorly, exhibiting an extremely low success rate that drops further with an increase in $n_{dyn}$. For trials where the robot is able to successfully execute the full trajectory, the executed trajectory exhibits large goal error that also increases with $n_{dyn}$. Comparatively SLAP and STEAP have a much higher success rate and follow the decreasing trend with increasing $n_{dyn}$. The goal error in STEAP and SLAP are comparable and much lower than OL. The estimation error is, on average, 2 times smaller with STEAP compared to SLAP and the difference between them increases with increasing $n_{cam}$. We believe that this is due to simultaneously solving both the trajectory estimation and planning problems: the motion plan can help to provide a better estimate of the robot's trajectory and the estimate of the trajectory can help to generate a better motion plan.

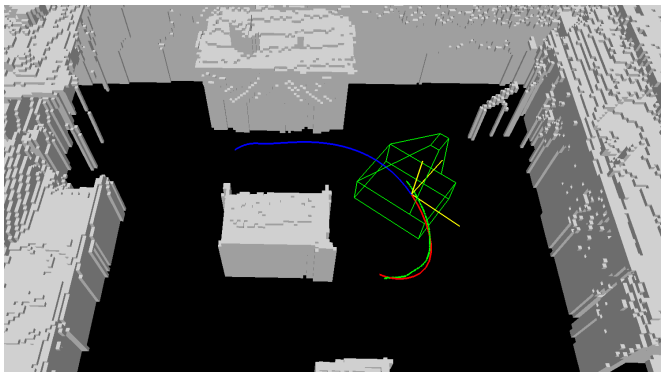


Fig. 7: Visualization of STEAP results on one run of problem 1. The green line is the ground-truth trajectory as determined by the motion capture system, and green robot outline shows the current pose of the Vector robot. The blue line is the planned trajectory and the red line is the estimated trajectory. The yellow axis is current raw pose estimate. The ground plane is cut for visibility. Best viewed in color.

TABLE III: 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 |

### *B. Experiments with a Real Robot*

Real-world experiments are performed in an $8m \times 6m$ 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 Fig. 1 and Fig. 5, respectively. Our implementation runs on a desktop computer equipped with Intel 4.0GHz quad-core CPU, 32GB 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 starts from the start configuration, and is tasked with driving towards the goal configuration. For both problems the graph consists of 50 states from start to goal with 2 interpolated binary obstacle factors between any two states. Fig. 7 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. III shows the 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 omni-directional 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 fix drifting. With STEAP the robot can achieve a 95% overall success rate for the given tasks, with final translation error of about 14.2cm in problem 1, and 5.19cm in problem 2. This goal error is due to the finite horizon trajectory set up 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. III. 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 seen in Table. III. In the experiments video,[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 19.3ms and 76.0ms respectively, and maximum runtimes of 30.3ms and 149ms respectively, indicating that our localization implementation can easily process the depth image stream at 30Hz, and run STEAP at $\sim$ 10Hz.

## VI. 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 historical trajectory as well as replan for the future trajectory as new information is encountered. This is made possible by 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 handle stochasticity associated with execution or unmodeled behaviors. Our results demonstrate that this framework is suitable for online applications with high-degree-of-freedom systems in known, static real-world environments.


## Acknowledgments

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 their help with the Vector robot and providing access to it.

## REFERENCES


[1] Ali-akbar Agha-mohammadi, Saurav Agarwal, Suman Chakravorty, and Nancy M Amato. Simultaneous localization and planning for physical mobile robots via enabling dynamic replanning in belief space. *arXiv:1510.07380*, 2015.

[2] Sean Anderson and Timothy D Barfoot. Full STEAM ahead: Exactly sparse Gaussian process regression for batch continuous-time trajectory estimation on SE (3). In *Intelligent Robots and Systems, IEEE/RSJ International Conference on (IROS)*, pages 157–164. IEEE, 2015.

[3] Sean Anderson, Timothy D Barfoot, Chi Hay Tong, and Simo Särkkä. Batch nonlinear continuous-time trajectory estimation as exactly sparse Gaussian process regression. *Autonomous Robots*, 39(3):221–238, 2015.

[4] Hagai Attias. Planning by probabilistic inference. In *AISTATS*, 2003.

[5] Tim Barfoot, Chi Hay Tong, and Simo Sarkka. Batch continuous-time trajectory estimation as exactly sparse Gaussian process regression. *Proceedings of Robotics: Science and Systems (RSS)*, 2014.

[6] Frank Dellaert and Michael Kaess. Square root SAM: Simultaneous localization and mapping via square root information smoothing. *The International Journal of Robotics Research*, 25 (12):1181–1203, 2006.

[7] Jing Dong, Mustafa Mukadam, Frank Dellaert, and Byron Boots. Motion planning as probabilistic inference using Gaussian processes and factor graphs. In *Proceedings of Robotics: Science and Systems (RSS-2016)*, 2016.

[8] Jing Dong, Byron Boots, and Frank Dellaert. Sparse Gaussian processes for continuous-time trajectory estimation on matrix lie groups. *arXiv:1705.06020*, 2017.

[9] Jing Dong, John Burnhan, Byron Boots, Glen Rains, and Frank Dellaert. 4d crop monitoring: Spatio-temporal reconstruction for agriculture. In *Proceedings of the 2017 IEEE Conference on Robotics and Automation (ICRA)*, 2017.

[10] Pedro Felzenszwalb and Daniel Huttenlocher. Distance transforms of sampled functions. Technical Report 19, 2012.

[11] Dave Ferguson, Nidhi Kalra, and Anthony Stentz. Replanning with RRTs. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 1243–1248. IEEE, 2006.

[12] Eric Huang, Mustafa Mukadam, Zhen Liu, and Byron Boots. Motion planning with graph-based trajectories and Gaussian process inference. In *Proceedings of the 2017 IEEE Conference on Robotics and Automation (ICRA)*, 2017.

[13] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, et al. Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 559–568, 2011.

[14] Michael Kaess, Ananth Ranganathan, and Frank Dellaert. iSAM: Incremental smoothing and mapping. *Robotics, IEEE Transactions on*, 24(6):1365–1378, 2008.

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

[16] Michael Kaess, Hordur Johannsson, Richard Roberts, Viorela Ila, John J Leonard, and Frank Dellaert. iSAM2: Incremental smoothing and mapping using the Bayes tree. *The International Journal of Robotics Research*, page 0278364911430419, 2011.

[17] Rudolph Emil Kalman et al. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1): 35–45, 1960.

[18] Sven Koenig and Maxim Likhachev. Fast replanning for navigation in unknown terrain. *IEEE Transactions on Robotics*, 21(3):354–363, 2005.

[19] Frank R Kschischang, Brendan J Frey, and Hans-Andrea Loeliger. Factor graphs and the sum-product algorithm. *Information Theory, IEEE Transactions on*, 47(2):498–519, 2001.

[20] Sergey Levine and Vladlen Koltun. Variational policy search via trajectory optimization. In *Advances in Neural Information Processing Systems*, pages 207–215, 2013.

[21] Mustafa Mukadam. PIPER. *[Online] Available*, 2017. URL https://github.com/gtrll/piper.

[22] Mustafa Mukadam, Xinyan Yan, and Byron Boots. Gaussian process motion planning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9–15, May 2016. doi: 10.1109/ICRA.2016.7487091.

[23] Mustafa Mukadam, Ching-An Cheng, Xinyan Yan, and Byron Boots. Approximately optimal continuous-time motion planning and control via probabilistic inference. In *Proceedings of the 2017 IEEE Conference on Robotics and Automation (ICRA)*, 2017.

[24] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with cuda. *Queue*, 6(2):40–53, March 2008. ISSN 1542-7730. doi: 10.1145/1365490.1365500. URL http://doi.acm.org/10.1145/1365490.1365500.

[25] Will Penny. Simultaneous localisation and planning. In *Cognitive Information Processing (CIP), 2014 4th International Workshop on*, pages 1–6. IEEE, 2014.

[26] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. ROS: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[27] Mohammadhussein Rafieisakhaei, Suman Chakravorty, and PR Kumar. Non-Gaussian slap: Simultaneous localization and planning under non-Gaussian uncertainty in static and dynamic environments. *arXiv:1605.01776*, 2016.

[28] Konrad Rawlik, Marc Toussaint, and Sethu Vijayakumar. On stochastic optimal control and reinforcement learning by approximate inference. *Proceedings of Robotics: Science and Systems VIII*, 2012.

[29] Duy-Nguyen Ta, Marin Kobilarov, and Frank Dellaert. A factor graph approach to estimation and model predictive control on unmanned aerial vehicles. In *Unmanned Aircraft Systems (ICUAS), 2014 International Conference on*, pages 181–188. IEEE, 2014.

[30] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

[31] Emanuel Todorov. General duality between optimal control and estimation. In *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*, pages 4286–4292. IEEE, 2008.

[32] Chi Hay Tong, Paul Furgale, and Timothy D Barfoot. Gaussian process Gauss-Newton for non-parametric simultaneous localization and mapping. *The International Journal of Robotics Research*, 32(5):507–525, 2013.

[33] Marc Toussaint. Robot trajectory optimization using approximate inference. In *Proceedings of the 26th annual international conference on machine learning*, pages 1049–1056. ACM, 2009.

[34] Marc Toussaint and Amos Storkey. Probabilistic inference for solving discrete and continuous state Markov decision processes. In *Proceedings of the 23rd international conference on Machine learning*, pages 945–952. ACM, 2006.

[35] Xinyan Yan, Vadim Indelman, and Byron Boots. Incremental sparse GP regression for continuous-time trajectory estimation and mapping. In *Robotics and Autonomous Systems*, volume 87, pages 120–132, 2017.