STEAP: Towards Online Estimation and Replanning

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

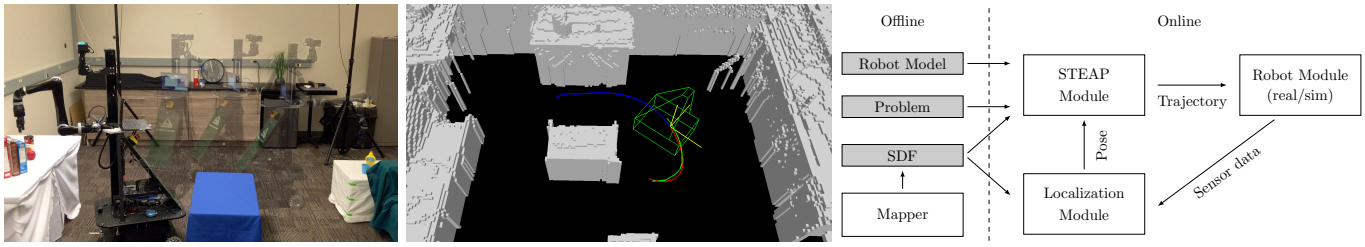

Fig. 1: *Left*: The Vector mobile manipulator is solving the STEAP problem, where the semi-transparent robots show the trajectory taken, while the solid robot is the goal configuration. *Middle*: Green line is the ground-truth trajectory, green robot outline shows the current pose, blue line is the planned trajectory, red line is the estimated trajectory, and yellow axis is the current raw pose estimate. *Right*: Block diagram of our framework showing all the components and how they interact.

***Abstract*—In this work, we present simultaneous trajectory estimation and planning (STEAP) - a unified approach to solving continuous-time trajectory estimation and planning problems. Although, these problems are usually considered separately, within our framework we show how estimation and planning can benefit from each other and remove redundancy during computation. Each time-step the robot is tasked with finding the full continuous-time trajectory from start to goal, such that the history of the trajectory signifies the solution to the estimation problem, while the future of the trajectory signifies a solution to the planning problem. Building on recent work we employ incremental inference on probabilistic graphical models to solve this problem, and provide an approach that can contend with high-degree-of-freedom (DOF) trajectory space, 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 approach empirically 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 finds a trajectory of future states to minimize a cost (for example path length or energy used) and achieve a feasible (collision-free) solution.

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 [4] that formulates SLAM as inference in a factor graph [15] and exploit the sparsity of the underlying large-scale linear systems to perform inference efficiently. Given new sensor data, incremental Smoothing and Mapping (iSAM) [11, 13] exploits the structure of the problem to efficiently update the solution rather than resolving the entire problem from scratch. Recently, Tong et al. [25] 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. This was extended in Barfoot et al. [3] to take advantage of the sparse structure inherent in the problem, in Yan et. al [28] to efficiently incrementally update the solution, and in Dong et al. [7] to 4D mapping problems.

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. Several researchers have recently proposed a probabilistic inference perspective on planning and control problems, leveraging expectation maximization [27, 16], expectation propagation [26], KL-minimization [23], and efficient inference in factor graphs [5, 9, 19]. Interestingly, the incremental inference technique [12] used in [5] to solve replanning problems is the same as originally used in [13] to solve SLAM problems. We exploit this idea to solve our more general class of simultaneous trajectory estimation and planning problems.

Efficient replanning for navigation is an active area of research [14, 8] but most previous work are difficult to extend to real-world, high-dimensional systems, are computationally expensive, or do not incorporate uncertainty in the robot's state estimate. Recent work in Simultaneous Localization and Planning (SLAP) attempt to unify localization and planning, with early work using HMMs [20], more recent approaches designed for dynamic environments [1, 22], and new approaches that combine state estimation and model predictive control [24].

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. We represent the trajectory as a continuous-valued function mapping time t to robot state $\boldsymbol{\theta}(t)$, and seek the solution with an incremental solver [13]. This allows us to avoid re-solving 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. [5] to plan trajectories on Lie groups [6]. 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. Trajectory Optimization as Inference

Following previous work on estimation and mapping problems [3, 28] and Gaussian process motion planning [5, 18], 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 state $\boldsymbol{\theta}(t)$. The goal is to find the *maximum a posteriori* (MAP) continuous-time trajectory given a prior distribution $p(\boldsymbol{\theta})$ on the space of trajectories and a likelihood $p(\mathbf{e}|\boldsymbol{\theta})$,

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \{ p(\boldsymbol{\theta}) p(\mathbf{e}|\boldsymbol{\theta}) \} \tag{1}$$

A. The Trajectory Prior

We represent the prior distribution of continuous-time trajectories as a vector-valued Gaussian process (GP) [3, 18]. For any collection of times $\boldsymbol{t} = \{t_0, \ldots, t_N\}$, the prior distribution is defined by the GP mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\mathcal{K}}$:

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

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 [3, 25]. In motion planning, the prior is selected to encourage higher-order derivatives of the system configuration to be minimized [5]. 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 [6] for details about the GP prior that we use for Lie groups.

B. The Likelihood Function

The likelihood function encodes information about a particular problem instance. For example, in estimation problems, the likelihood function encourages posterior trajectories to be consistent with proprioceptive or landmark observations [3], while in motion planning problems the likelihood function encourages posterior trajectories to be collision-free [5].

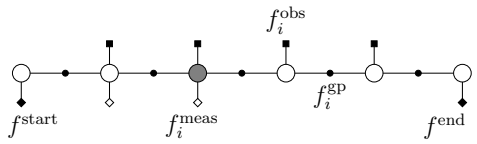

Fig. 2: Example factor graph representation of STEAP. Gray node shows current time t_c at time-step c.

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

$$p(\mathbf{e}|\boldsymbol{\theta}) \propto \exp \left\{ -\frac{1}{2} \parallel \boldsymbol{h}(\boldsymbol{\theta}, \mathbf{e}) \parallel^2_{\boldsymbol{\Sigma}} \right\} \tag{3}$$

where $\boldsymbol{h}(\boldsymbol{\theta}, \mathbf{e})$ can be any vector-valued cost function with covariance matrix $\boldsymbol{\Sigma}$.

III. Online STEAP with Factor Graphs

The MAP trajectory computation in Eq. 1 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* [15]. Thus, we can write the posterior distribution as a product of the factors that collectively represent the prior and the likelihood

$$p(\boldsymbol{\theta}) p(\mathbf{e}|\boldsymbol{\theta}) \propto f^{prior} f^{like}. \tag{4}$$

STEAP solves state estimation and planning problems simultaneously, and performs inference on the entire factor graph at once. An example STEAP factor graph at time t_c is shown in Fig. 2 and is defined by Eq. 4 where,

$$f^{prior} = f^{gp} = \prod_{i=0}^{N-1} f_i^{gp}(\boldsymbol{\theta}_i, \boldsymbol{\theta}_{i+1}), \quad f^{like} = f^{meas} f^{obs} f^{fix},$$
$$f^{meas} = \prod_{i=1}^{c} f_i^{meas}, \quad f^{obs} = \prod_{i=1}^{N-1} f_i^{obs}(\boldsymbol{\theta}_i),$$
$$f^{fix} = f^{start}(\boldsymbol{\theta}_0) f^{end}(\boldsymbol{\theta}_N)$$

There are two major advantages of using STEAP: (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 or factor in a

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

STEAP factor graph do not change much during execution, i.e. only a few measurement factors are added at each time-step. This allows for very efficient incremental inference using the Bayes tree algorithm [13]. Recomputing the solution with Bayes trees only requires a small fraction of the run-time compared to re-optimizing the full graph from scratch. By utilizing this efficient incremental inference technique we get a significant performance boost, and easily achieve real-time performance, as illustrated in our experiments.

The STEAP procedure is summarized in Algorithm 1. We implement STEAP within the PIPER [17] package using ROS [21] and GPMP2 [5] and have open-sourced the code. Fig. 1 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 produced by the `Mapper` and assumed to be static, and is used for collision checking. In the online phase, this information is passed to our central module, `STEAP Module`, that solves STEAP problems and passes optimized trajectories to be executed to 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 (in our implementation we use a depth image-based localization algorithm, similar to the tracking in KinectFusion [10]) that can be interpreted by the `STEAP Module`.

IV. Evaluation

We conduct experiments[1] and demonstrate our framework on a mobile manipulator, shown in Fig. 1, consisting of an omni-drive base and a 6-DOF Kinova JACO2 arm. Real-world experiments are performed in a $8m \times 6m$ 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 a motion capture system. Fig. 1 shows the robot traversing within a map of the environment.

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

TABLE I: 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
Raw localization error (cm)	7.07	6.45
Trajectory estimation error (cm)	3.48	2.53

Our implementation runs on a computer equipped with Intel 4.0GHz quad-core CPU, 32GB memory and one NVIDIA Titan X GPU. Robot sensor data is streamed to the computer over WiFi, and STEAP commands are streamed back to robot.

We design 2 problems for performance evaluation. In each problem the robot starts from a start configuration, and is tasked with driving towards a goal configuration. For both problems the graph consists of 50 states from start to goal with 2 interpolated binary obstacle factors [5] between any two states. 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 success rate, final goal error, raw localization error and trajectory estimation error.

Table. I shows the performance in these real-world experiments. We first run one-time batch planning by GPMP2 and use an open loop controller (OL) to follow the planned trajectory. Since the control command execution on is noisy, the robot base cannot follow the planned trajectory well and every run ends with a collision. With 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. I. 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. I. 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 implementation can easily process the depth image stream at $\sim$ 30Hz, and run STEAP at $\sim$ 10Hz.

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] 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.

[4] 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.

[5] 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.

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

[7] 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.

[8] 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.

[9] 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.

[10] 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.

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

[12] 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.

[13] 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.

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

[15] 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.

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

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

[18] 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.

[19] 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.

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

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

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

[27] 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.

[28] 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.