Motion Planning as Probabilistic Inference using Gaussian Processes and Factor Graphs

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

***Abstract*—With the increased use of high degree-of-freedom robots that must perform tasks in real-time, there is a need for fast algorithms for motion planning. In this work, we view motion planning from a probabilistic perspective. We consider smooth continuous-time trajectories as samples from a Gaussian process (GP) and formulate the planning problem as probabilistic inference. We use factor graphs and numerical optimization to perform inference quickly, and we show how GP interpolation can further increase the speed of the algorithm. Our framework also allows us to incrementally update the solution of the planning problem to contend with changing conditions. We benchmark our algorithm against several recent trajectory optimization algorithms on planning problems in multiple environments. Our evaluation reveals that our approach is several times faster than previous algorithms while retaining robustness. Finally, we demonstrate the incremental version of our algorithm on replanning problems, and show that it often can find successful solutions in a fraction of the time required to replan from scratch.**

I. Introduction & Related Work

Motion planning is a fundamental tool in robotics. The goal is to find a trajectory through the robot's configuration space that is both *feasible* and *optimal*. In this work, feasible trajectories bring the robot from a start to goal configuration while remaining collision-free and obeying the robot's physical limitations. Optimality is related to trajectory smoothness, in other words trajectories that minimize dynamical criteria like velocity or acceleration are considered to be more optimal than trajectories that do not [12, 26, 30]. Previous work has focused on both sampling-based algorithms and trajectory optimization approaches to motion planning.

Sampling-based algorithms such as Probabilistic Road Maps [13] and Rapidly exploring Random Trees [16, 17] can generate feasible trajectories in complex high-dimensional configuration spaces. However, due to the nature of random sampling, trajectories generated by sampling-based algorithms often lack quality and may result in jerky and redundant motions. The problem is further compounded when the configuration space is sparsely populated with obstacles or the problem has tight constraints on navigation.

Trajectory optimization attempts to find smooth, collision-free trajectories that optimize a cost function. For example, CHOMP and related methods [2, 8, 24, 30] optimize a cost functional using covariant gradient descent, while STOMP [12] optimizes non-differentiable constraints by stochastic sampling of noisy trajectories. One of the drawbacks of these approaches is that the trajectory is finely discretized in practice so that the optimizers can reason about obstacles in the environment and guarantee smoothness in the solution. Unfortunately, this results in high computational cost. TrajOpt [25, 26] attempts to overcome this problem and speed up solution time by formulating trajectory optimization as sequential quadratic programming and performing continuous-time collision checking. This allows TrajOpt to represent trajectories using fewer states in theory, but a densely parameterized trajectory is still required for reasoning about obstacles in more complex environments and when smoothness in the output trajectory is required during execution.

Continuous-time trajectory representations can overcome the computational cost incurred by using large number of states. B-Splines [6] and kernel methods [19] have been used to represent trajectories with fewer states. The Gaussian Process Motion Planner (GPMP) [20] parameterizes trajectories only with a few states and then uses Gaussian process (GP) interpolation to query the trajectory at any time of interest.

In this work we view motion planning as trajectory optimization from a probabilistic inference perspective [27, 28]. We represent continuous-time trajectories as samples from a GP [20] and then formulate the planning problem via a probabilistic graphical model. We perform inference on the graph using numerical tools that solve a nonlinear least squares optimization problem, and generate fast solutions by exploiting the sparsity of the underlying linear system.

By viewing motion planning as inference, we are able to borrow tools from other areas of robotics. The Simultaneous Localization and Mapping (SLAM) community has been focusing on efficient optimization algorithms for many years, and one of the more successful approaches is the Smoothing and Mapping (SAM) family of algorithms [4]. By formulating SLAM as inference in a factor graph [15], and exploiting the sparsity of underlying large-scale linear systems, SAM can perform efficient inference. We adopt these methods into our motion planning algorithm, and realize a significant increase in speed over previous methods.

Replanning problems, which entail making modifications to the trajectory such as changing goal locations, are common tasks in real-world motion planning applications. Early replanning work includes D* [14] and Anytime A* [18], but these algorithms are formulated for discrete state-spaces with limited dimensionality. Recently, ITOMP [21] was able to accomplish fluent replanning with a scheduler to enforce hard timing deadlines, but no optimal or even feasible solution is

guaranteed. Parallel computing with GPUs is suggested by Park et al. [22] to significantly speed up replanning, but this approach does little to reduce the actual computational cost.

We provide a method for replanning inspired by work in the SLAM community. Incremental Smoothing and Mapping (iSAM) [9, 11] performs inference on factor graphs incrementally and in real-time. By using an incremental solver [11], we avoid re-solving the entire planning problem from scratch and only update the trajectory where required. This dramatically reduces the overall computational costs of our approach enabling fast replanning.

II. Continuous-time Trajectory Optimization with Gaussian Processes

Following the conventions of Toussaint et al. [27, 28], we view the trajectory optimization problem as probabilistic inference. In this framework, the goal is to find the *maximum a posteriori* (MAP) trajectory given a prior distribution on the space of trajectories encouraging *smoothness* and a likelihood function that encourages the trajectory to be *collision-free*.

We represent the trajectory as a continuous-valued function mapping time t to robot states $\boldsymbol{\theta}(t)$. We find the MAP function $\boldsymbol{\theta}^*$ by probabilistic inference. This involves several steps, which we preview here before describing in detail below.

First, we place a Gaussian process prior encouraging smoothness directly on the function space of trajectories (Section II-A). Next, we specify a likelihood function that encourages collision-free trajectories (Section II-B). The MAP estimate is then calculated from the posterior distribution (Section II-C). Finally, the trajectory can be queried at *any* time of interest by computing the mean trajectory under the Laplace approximation of the posterior distribution (Section II-D).

A. The Gaussian Process Prior

A vector-valued Gaussian process (GP) is a collection of random variables, any finite number of which have a joint Gaussian distribution. We define a prior distribution over trajectories $\boldsymbol{\theta}(t) \sim \mathcal{GP}(\boldsymbol{\mu}(t), \boldsymbol{\mathcal{K}}(t,t'))$, where $\boldsymbol{\mu}(t)$ is a vector-valued mean function and $\boldsymbol{\mathcal{K}}(t,t')$ is a matrix-valued covariance function. Using the GP framework we can say that for any collection of times $\boldsymbol{t} = \{t_0, \ldots, t_N\}$, $\boldsymbol{\theta}$ has a joint Gaussian distribution:

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

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

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

The smoothness and generalization properties of a GP are encoded by the kernel $\boldsymbol{\mathcal{K}}$. Similar to previous work on Gaussian process motion planning [20], we employ a structured covariance function generated by a linear time varying stochastic differential equation (LTV-SDE) [1]:

$$\dot{\boldsymbol{\theta}}(t) = \mathbf{A}(t)\boldsymbol{\theta}(t) + \mathbf{u}(t) + \mathbf{F}(t)\mathbf{w}(t), \tag{3}$$

where $\mathbf{u}(t)$ is the known system control input, $\mathbf{A}(t)$ and $\mathbf{F}(t)$ are time varying matrices of the system, and $\mathbf{w}(t)$ is the white process noise which is represented by

$$\mathbf{w}(t) \sim \mathcal{GP}(\mathbf{0}, \mathbf{Q}_C \delta(t - t')), \tag{4}$$

where $\mathbf{Q}_C$ is the power-spectral density matrix and $\delta(t-t')$ is the Dirac delta function. The mean and covariance of the GP are generated from the solution of the LTV-SDE in Eq. (3):

$$\boldsymbol{\mu}(t) = \boldsymbol{\Phi}(t, t_0)\boldsymbol{\mu}_0 + \int_{t_0}^{t} \boldsymbol{\Phi}(t, s)\mathbf{u}(s)ds, \tag{5}$$

$$\boldsymbol{\mathcal{K}}(t,t') = \boldsymbol{\Phi}(t,t_0)\boldsymbol{\mathcal{K}}_0\boldsymbol{\Phi}(t',t_0)^\top + \int_{t_0}^{\min(t,t')} \boldsymbol{\Phi}(t,s)\mathbf{F}(s)\mathbf{Q}_C\mathbf{F}(s)^\top\boldsymbol{\Phi}(t',s)^\top ds, \tag{6}$$

where $\boldsymbol{\mu}_0$ is the initial mean value of the first state, $\boldsymbol{\mathcal{K}}_0$ is the covariance of the first state, and $\boldsymbol{\Phi}(t,s)$ is the state transition matrix [1].

The GP prior distribution is then defined in terms of its mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\mathcal{K}}$:

$$P(\boldsymbol{\theta}) \propto \exp\left\{ -\frac{1}{2} \parallel \boldsymbol{\theta} - \boldsymbol{\mu} \parallel^2_{\boldsymbol{\mathcal{K}}} \right\}. \tag{7}$$

The benefit of this prior is that the inverse kernel matrix $\boldsymbol{\mathcal{K}}^{-1}$ is proved to be exactly sparse (block-tridiagonal), due to the Markov property of the LTV-SDE [1]. As we will see in Section III-A, this kernel allows for fast, structure-exploiting inference.

B. The Likelihood Function

The likelihood function is given by the conditional distribution $L_{obs}(\boldsymbol{\theta}_i | c_i = 0) = P(c_i = 0 | \boldsymbol{\theta}_i)$, which specifies the probability of being clear of collisions, given the current configuration $\boldsymbol{\theta}_i$. We define the likelihood as a distribution in the exponential family

$$L_{obs}(\boldsymbol{\theta}_i | c_i = 0) \propto \exp\left\{ -\frac{1}{2} \parallel \mathbf{h}(\boldsymbol{\theta}_i) \parallel^2_{\boldsymbol{\Sigma}_{obs_i}} \right\}, \tag{8}$$

where $\mathbf{h}(\boldsymbol{\theta}_i)$ is vector-valued *obstacle cost function* and $\boldsymbol{\Sigma}^{-1}_{obs_i}$ is a hyperparmeter of the distribution The specific likelihood and obstacle cost function used in our implementation are detailed in Section IV-B.

C. Computing the MAP Trajectory

Given a Gaussian process prior and the exponential family likelihood function, the goal is to compute the MAP posterior trajectory

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \left\{ P(\boldsymbol{\theta}) \prod_i P(c_i | \boldsymbol{\theta}_i) \right\}, \tag{9}$$

$$= \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \left\{ -\log\left(P(\boldsymbol{\theta}) \prod_i P(c_i | \boldsymbol{\theta}_i) \right) \right\}, \tag{10}$$

$$= \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \left\{ \frac{1}{2} \parallel \boldsymbol{\theta} - \boldsymbol{\mu} \parallel^2_{\boldsymbol{\mathcal{K}}} + \frac{1}{2} \parallel \boldsymbol{h}(\boldsymbol{\theta}) \parallel^2_{\boldsymbol{\Sigma}_{obs}} \right\}, \tag{11}$$

where $\boldsymbol{h}(\boldsymbol{\theta}) \doteq [\mathbf{h}(\boldsymbol{\theta}_0) \ldots \mathbf{h}(\boldsymbol{\theta}_N)]^\top$ and Eq. (11) follows from Eq. (7) and Eq. (8).

Eq. (11) illustrates the duality between probabilistic inference and optimization, two different perspectives on motion planning problems. The terms in Eq. (11) can also be viewed as information, or 'cost' to be reduced. The MAP estimation problem can therefore be reduced to a (possibly nonlinear) least squares problem, which has been well studied and for which many numerical tools are available.

Iterative approaches, like Gauss-Newton or Levenberg-Marquardt repeatedly resolve a linearized approximation of Eq. (11) until convergence. Linearizing $\mathbf{h}(\boldsymbol{\theta}_i)$

$$\mathbf{h}(\boldsymbol{\theta}_i) \approx \mathbf{h}_i(\overline{\boldsymbol{\theta}_i}) + \mathbf{H}_i \delta\boldsymbol{\theta}_i, \mathbf{H}_i \doteq \frac{\partial \mathbf{h}_i}{\partial \boldsymbol{\theta}_i}\Big|_{\overline{\boldsymbol{\theta}}_i(t_i)}, \tag{12}$$

where $\mathbf{H}_i$ is the Jacobian matrix of $\mathbf{h}(\boldsymbol{\theta}_i)$, we convert the nonlinear least square problem to a linear problem around linearization point $\overline{\boldsymbol{\theta}}$

$$\delta\boldsymbol{\theta}^* = \underset{\delta\boldsymbol{\theta}}{\operatorname{argmin}} \left\{ \frac{1}{2} \parallel \overline{\boldsymbol{\theta}} + \delta\boldsymbol{\theta} - \boldsymbol{\mu} \parallel^2_{\boldsymbol{\mathcal{K}}} + \frac{1}{2} \parallel \boldsymbol{h}(\overline{\boldsymbol{\theta}}) + \mathbf{H}\delta\boldsymbol{\theta} \parallel^2_{\boldsymbol{\Sigma}_{obs}} \right\}, \tag{13}$$

where $\mathbf{H} \doteq \operatorname{diag}(\mathbf{H}_0, \ldots, \mathbf{H}_N)$. The optimal perturbation $\delta\boldsymbol{\theta}^*$ is given by solving the following linear system

$$(\boldsymbol{\mathcal{K}}^{-1} + \mathbf{H}^\top \boldsymbol{\Sigma}_{obs}^{-1} \mathbf{H}) \delta\boldsymbol{\theta}^* = \boldsymbol{\mathcal{K}}^{-1}(\boldsymbol{\mu} - \overline{\boldsymbol{\theta}}) - \mathbf{H}^\top \boldsymbol{\Sigma}_{obs}^{-1} \boldsymbol{h}(\overline{\boldsymbol{\theta}}). \tag{14}$$

Once the linear system is solved, iteration $\overline{\boldsymbol{\theta}} \leftarrow \overline{\boldsymbol{\theta}} + \delta\boldsymbol{\theta}^*$ is applied until convergence criteria are met.

D. Fast Gaussian Process Interpolation

Following [1, 20, 29], 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 $\overline{\boldsymbol{\theta}}$ at time points $\boldsymbol{t}$ [23]:

$$\overline{\boldsymbol{\theta}}(\tau) = \boldsymbol{\mathcal{K}}(\tau, \boldsymbol{t}) \boldsymbol{\mathcal{K}}^{-1} \overline{\boldsymbol{\theta}}. \tag{15}$$

Although the interpolation in Eq. (15) naïvely requires $O(N)$ operations, $\overline{\boldsymbol{\theta}}(\tau)$ can be computed in $O(1)$ by leveraging the structure of the sparse GP prior introduced in Section II-A [1]. Since the LTV-SDE is Markovian, $\overline{\boldsymbol{\theta}}(\tau)$ at $\tau, t_i < \tau < t_{i+1}$, is a linear combination of *only* the adjacent function values $\overline{\boldsymbol{\theta}}_i$ and $\overline{\boldsymbol{\theta}}_{i+1}$ and is efficiently computed by

$$\overline{\boldsymbol{\theta}}(\tau) = \boldsymbol{\mu}(\tau) + \boldsymbol{\Lambda}(\tau)(\overline{\boldsymbol{\theta}}_i - \boldsymbol{\mu}_i) + \boldsymbol{\Psi}(\tau)(\overline{\boldsymbol{\theta}}_{i+1} - \boldsymbol{\mu}_{i+1}) \tag{16}$$

$$\boldsymbol{\Lambda}(\tau) = \boldsymbol{\Phi}(\tau, t_i) - \mathbf{Q}_\tau \boldsymbol{\Phi}(\tau, t_i)^\top \mathbf{Q}_{i+1}^{-1} \boldsymbol{\Phi}(t_{i+1}, t_i) \tag{17}$$

$$\boldsymbol{\Psi}(\tau) = \mathbf{Q}_\tau \boldsymbol{\Phi}(\tau, t_i)^\top \mathbf{Q}_{i+1}^{-1} \tag{18}$$

where

$$\mathbf{Q}_i = \int_{t_{i-1}}^{t_i} \boldsymbol{\Phi}(t_i, s) \mathbf{F}(s) \mathbf{Q}_c \mathbf{F}(s)^\top \boldsymbol{\Phi}(t_i, s)^\top ds. \tag{19}$$

In Section III-B, we will describe how GP interpolation can be used *during* trajectory optimization to dramatically speed up inference.

III. Fast Inference and Incremental updates with Factor Graphs

We now describe how factor graphs and efficient trajectory interpolation can be used to perform fast inference, and incremental updates to solve problems like rapid replanning.

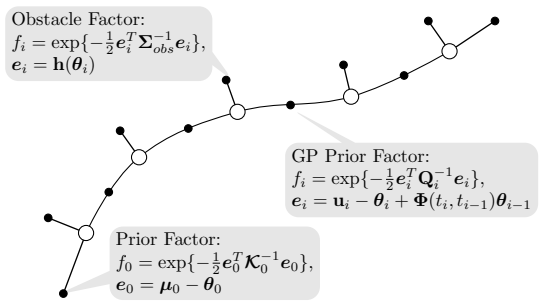


Fig. 1: A factor graph of an example trajectory optimization problem showing optimized states (white circles) and three kinds of factors (black dots), namely prior factors on start and goal states, obstacle cost factors on each state, and GP prior factors that connect consecutive states.

A. Factor Graphs: Faster Inference by Exploiting Sparsity

If the linear system in Eq. (14) is sparse, then $\delta\boldsymbol{\theta}^*$ can be solved efficiently by exploiting the sparse Cholesky decomposition followed by forward-backward passes [7]. Fortunately, this is the case: $\mathbf{H}$ is block-diagonal, the block-diagonal property of $\mathbf{H}^\top \boldsymbol{\Sigma}_{obs}^{-1} \mathbf{H}$ is trivial, and we have selected a Gaussian process prior with a block tridiagonal $\boldsymbol{\mathcal{K}}^{-1}$ (Section II-A).

By maintaining a sparse linear system in Eq. (14), MAP trajectory optimization can be equivalently viewed as the problem of inference on a *factor graph* [15]. A factor graph $G = \{\boldsymbol{\Theta}, \mathcal{F}, \mathcal{E}\}$ is a bipartite graph, which represents the factorization of a function (see Eq. (20)). $\boldsymbol{\Theta} \doteq \{\boldsymbol{\theta}_0, \ldots, \boldsymbol{\theta}_N\}$ is set of variables, $\mathcal{F} \doteq \{f_0, \ldots, f_M\}$ is a set of factors, where f_i are functions on variable subsets $\boldsymbol{\Theta}_i$, and $\mathcal{E}$ are edges connected to the two type of nodes.

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

In this work the factors represent the prior and likelihood functions. GP prior factors are defined

$$f_{gp}(\boldsymbol{\theta}_{i-1}, \boldsymbol{\theta}_i) \doteq \exp\left\{ -\frac{1}{2} \parallel \mathbf{u}_i - \boldsymbol{\theta}_i + \boldsymbol{\Phi}(t_i, t_{i-1})\boldsymbol{\theta}_{i-1} \parallel^2_{\mathbf{Q}_i} \right\}, \tag{21}$$

with $\mathbf{u}_i = \int_{t_{i-1}}^{t_i} \boldsymbol{\Phi}(t_i, s)\mathbf{u}(s)ds$ (see [29] for details). The obstacle factor is defined as $f_{obs}(\boldsymbol{\theta}_i) = L_{obs}(\boldsymbol{\theta}_i | c_i = 0)$.

An example factor graph is illustrated in Fig. 1. Performing inference on the factor graph is equivalent to solving the MAP estimation problem in Eq. (9). This equivalence has been exploited for years in the SLAM community [4], resulting in a range of tools that can be used to efficiently perform inference [9, 10, 11, 29]. This insight is critical in Section III-C where we describe how to incrementally update the MAP trajectory given new conditions.

B. Even Faster Inference through Trajectory Interpolation

In practice trajectory optimizers often represent trajectories at a set of discrete time points [12, 21, 24, 25]. Usually a densely-sampled set of time points is required to reason about collisions, forcing optimizers to deal with a large number of parameters at increased computational cost.

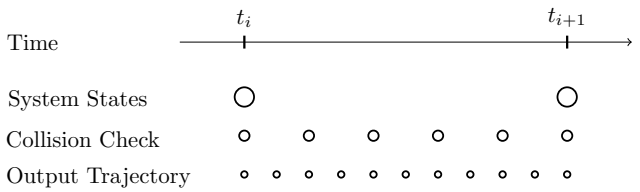


Fig. 2: An example that shows the trajectory at different resolutions. System states parameterize the trajectory, collision checking is performed at a higher resolution during optimization and the output trajectory can be up-sampled further for execution.

One of the primary benefits of using Gaussian processes in motion planning is that the trajectories are represented as functions that can be queried at any time of interest through Gaussian process interpolation (Section II-D) [20]. The key insight is that a smooth trajectory can be represented by a very small number of states, but rich obstacle information between states can still be considered during optimization by dense interpolation between the states. This is illustrated in Fig. 2 for a section of the trajectory between any two time points.

To reason about collisions between states during optimization, a new version of an obstacle factor is implemented. Unlike the obstacle factor in Fig. 1, which is *unary* and only calculates collision cost at the connected state, the new obstacle factor is *binary* and is connected to its two neighboring states $\boldsymbol{\theta}_i$ and $\boldsymbol{\theta}_{i+1}$. For any time τ, $t_i < \tau < t_{i+1}$, the state $\boldsymbol{\theta}(\tau)$ is interpolated by Eq. (16), and the obstacle cost is calculated as $\mathbf{h}(\boldsymbol{\theta}(\tau))$. The cost at the interpolated state is then incorporated into a binary factor

$$f_{obs}(\boldsymbol{\theta}_i, \boldsymbol{\theta}_{i+1}) \doteq \exp\Big\{ -\frac{1}{2} \parallel \mathbf{h}\Big(\boldsymbol{\mu}(\tau) + \boldsymbol{\Lambda}(\tau)(\boldsymbol{\theta}_i - \boldsymbol{\mu}_i) + \boldsymbol{\Psi}(\tau)(\boldsymbol{\theta}_{i+1} - \boldsymbol{\mu}_{i+1})\Big) \parallel^2_{\boldsymbol{\Sigma}_{obs}} \Big\}. \quad (22)$$

In summary, the interpolated obstacle factor incorporates the obstacle information at the time τ in the factor graph and then updates the discrete states $\boldsymbol{\theta}_i$ and $\boldsymbol{\theta}_{i+1}$.

Once the optimization has converged, a trajectory for execution can be generated by densely interpolating the trajectory as shown in Fig. 2.

C. Incremental Inference

In addition to motion planning, we consider the *replanning* problem: given a solved motion planning problem and new conditions, solve the new problem. Replanning problems are commonly encountered in the real world, when, for example: (i) the goal position for the end-effector has been moved or (ii) the robot receives updated information about its current configuration. Since the replanning is performed during the robot's operation, possibly in dynamic environments, real-time replanning is critical to ensuring safety.

The naïve way to solve the problem is to literally replan by running trajectory optimization from scratch. However, if the majority of the problem is left unchanged, re-solving the entire problem duplicates work. We suggest an incremental approach to *update* the current solution given new information. To accomplish this task, we again turn to the SLAM community which has provided very efficient tools for performing incremental inference on factor graphs [9, 11, 29].

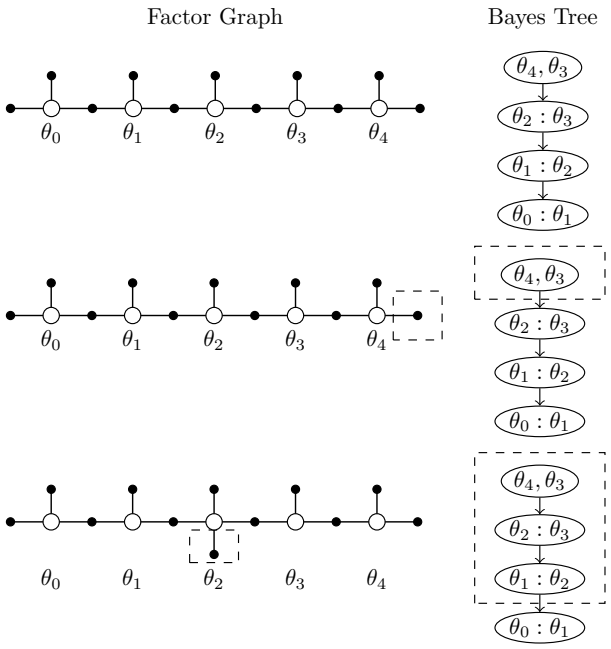


Fig. 3: Examples of replanning with the Bayes Tree. Dashed boxes in the middle and lower examples indicate the parts of factor graphs and Bayes Trees which are affected and changed while performing replanning.

In particular, we use the *Bayes Tree* [10, 11] to perform incremental inference on the factor graph [29]. The Bayes Tree is a data structure similar to a clique tree but directed. The upper part of Fig. 3 shows a factor graph of a simple motion planning problem: given start and end states, and obstacle cost factors, solve the MAP inference problem to find the optimal trajectory. The Bayes Trees in Fig. 3 are generated by the corresponding factor graphs and the elimination order from first to last state. If any factor is added or removed in the graph, only parts of the Bayes Tree are updated based on where the factor is added or removed. For details see [10, 11].

Two replanning examples are shown in Fig. 3. The middle example shows replanning when the goal state is changed. When the Bayes Tree is updated with the new goal only the root node of the tree is changed. The lower example shows a replanning problem given an observation of the current actuator configuration (e.g. from perception during execution) added in the middle of the trajectory at $\boldsymbol{\theta}_2$. When the Bayes Tree is updated, only the part of the tree corresponding to the remaining trajectory is changed.

In our implementation, we use the iSAM2 incremental solver [11] augmented by GP factors for interpolation [29] to solve replanning problems: first, we collect the additional information. Second, we form factors that need to be added to

or replaced in the factor graph. Finally, we run Algorithm 1 to update the Bayes Tree inside iSAM2, to get a new optimal solution.

Algorithm 1 Replanning using iSAM2

Input: new factors f_{new}, replaced factors $f_{replace}$
Output: updated optimal trajectory $\boldsymbol{\theta}^*$

Initialization :
add factors $f_{add} = \emptyset$, remove factors $f_{remove} = \emptyset$
iSAM2 update :
$f_{add} = f_{new}$
if ($f_{replace} \neq \emptyset$) **then**
  $f_{add} = f_{add} + f_{replace}$
  $f_{remove} = \textit{findOldFactors}(f_{replace})$
end if
iSAM2.*updateBayesTree*(f_{add}, f_{remove})
return iSAM2.*getCurrentEstimation*()

IV. Implementation Details

Here we provide technical details about our practical implementation for solving motion planning problems.

A. The Constant Velocity GP Prior

In our implementation, we use the 'constant velocity' GP prior [1], which applies the white-noise-on-acceleration model

$$\ddot{\boldsymbol{\xi}} = \mathbf{w}(t), \tag{23}$$

where $\boldsymbol{\xi} \in \mathbb{R}^D$ is the system configuration in a D-dimensional vector space (e.g. number of joints of a robot arm). For the model in Eq. (3), $\boldsymbol{\xi}$ is not a valid Markovian state, but

$$\boldsymbol{\theta}(t) = \begin{bmatrix} \boldsymbol{\xi}(t) \\ \dot{\boldsymbol{\xi}}(t) \end{bmatrix} \tag{24}$$

is Markovian in the LVT-SDE model in Eq. (3). We express the constant velocity prior through the SDE in Eq. (3) with

$$\mathbf{A}(t) = \begin{bmatrix} \mathbf{0} & \mathbf{1} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}, \mathbf{u}(t) = \mathbf{0}, \mathbf{F}(t) = \begin{bmatrix} \mathbf{0} \\ \mathbf{1} \end{bmatrix}. \tag{25}$$

In this case, given $\Delta t_i = t_i - t_{i-1}$, we have

$$\boldsymbol{\Phi}(t,s) = \begin{bmatrix} \mathbf{1} & (t-s)\mathbf{1} \\ \mathbf{0} & \mathbf{1} \end{bmatrix}, \mathbf{Q}_i = \begin{bmatrix} \frac{1}{3}\Delta t_i^3 \mathbf{Q}_C & \frac{1}{2}\Delta t_i^2 \mathbf{Q}_C \\ \frac{1}{2}\Delta t_i^2 \mathbf{Q}_C & \Delta t_i \mathbf{Q}_C \end{bmatrix}. \tag{26}$$

The 'constant velocity' GP prior is centered around a zero acceleration trajectory, so applying such prior will minimize actuator acceleration in configuration space, and gives the physical meaning of *smoothness* in our approach.

B. The Likelihood

For the likelihood in Eq. (8) we first define the hinge loss[1]

$$\mathbf{c}(z) = \begin{cases} -d(z) + \epsilon & \text{if } d(z) \leqslant \epsilon \\ 0 & \text{if } d(z) > \epsilon \end{cases} \tag{27}$$

[1]The hinge loss is not differentiable at $d(z) = \epsilon$, so in our implementation we set $\mathbf{c}'(z) = -0.5$ when $d(z) = \epsilon$.

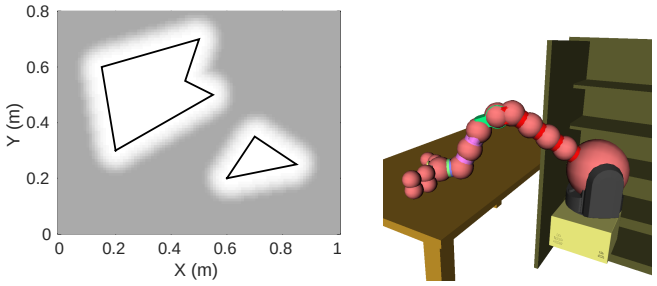


Fig. 4: The left subfigure shows the likelihood, L_{obs} in a 2D space with two obstacles and $\epsilon = 0.1$m. Obstacles are marked by black lines and darker area has higher likelihood for no-collision. The right subfigure shows the WAM arm represented with spheres (pink) used for collision checking.

where $d(z)$ is the *signed distance* from any point z in the workspace to the closest obstacle surface, and ϵ is a 'safety distance' indicating the boundary of the 'danger area' near obstacle surfaces. The signed distance $d(z)$ is calculated from a *signed distance field* (SDF) precomputed before optimization [30].

To perform fast collision checking for an arbitrary shape of the robot's physical body, we represent the robot with a set of spheres as in Zucker et al. [30] (shown in Fig. 4). In this way, we convert the problem of finding the minimum signed distance from the robot surface to any obstacles, to the problem of finding the signed distance of sphere centers, minus the sphere radius to any obstacles. The obstacle cost function for each configuration $\boldsymbol{\theta}_i$ is then completed by computing the signed distances for each sphere s_j ($j = 1, \ldots, M$) and collecting them into a single vector,

$$\mathbf{h}(\boldsymbol{\theta}_i) = [\mathbf{c}(\mathbf{x}(\boldsymbol{\theta}_i, s_j))]\Big|_{1 \leq j \leq M} \tag{28}$$

where $\mathbf{x}$ is the forward kinematics that maps any configuration $\boldsymbol{\theta}_i$ to the workspace. The remaining parameter $\boldsymbol{\Sigma}_{obs_i}$ needed to fully implement the likelihood in Eq. (8) is defined as, $\boldsymbol{\Sigma}_{obs_i} = \sigma_{obs}\mathbf{I}$.

The example in Fig. 4 visualizes the conditional probabilities of being free from collision given the likelihood defined by the obstacle cost function in Eq. (28). The darker region shows a free configuration space where the likelihood of no-collision is high. The small area beyond the boundary of the obstacles is lighter implying 'safety distance' defined by ϵ.

C. Motion constraints

Motion constraints exist in real-world planning problems and should be considered during trajectory optimization. Examples include the constrained start and goal states as well as constraints on any other states along the trajectory. These constraints are handled in the inference framework by treating them as prior knowledge on the trajectory states with very small uncertainties.

Additional equality constraints, such as end-effector rotation constraints (e.g. holding a cup filled with water upright)

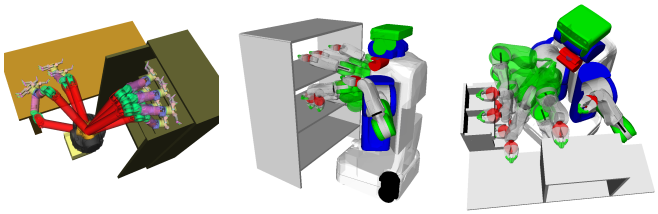
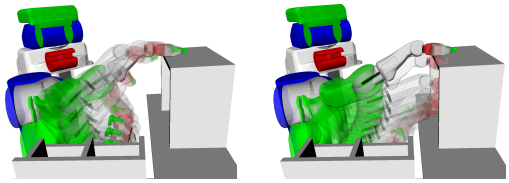


Fig. 5: Environments used for evaluation with robot start and goal configurations showing the WAM dataset (left), and PR2 dataset in *bookshelves* (center) and *industrial* scenes (right).



(a) $\sigma_{obs} = 0.005$ (b) $\sigma_{obs} = 0.05$

Fig. 6: Left subfigure shows successful trajectory with a good selection of σ_{obs}; right subfigure shows failure when σ_{obs} is too large.

written as $\mathbf{f}(\boldsymbol{\theta}_c) = \mathbf{0}$, where $\boldsymbol{\theta}_c$ is the set of states involved, can be incorporated into a likelihood,

$$L_{constraint}(\boldsymbol{\theta}) \propto \exp\left\{ -\frac{1}{2} \parallel \mathbf{f}(\boldsymbol{\theta}_c) \parallel^2_{\boldsymbol{\Sigma}_c} \right\}, \qquad (29)$$

where, $\boldsymbol{\Sigma}_c = \sigma_c \mathbf{I}$, σ_c is an arbitrary variance for this constraint, indicating how 'tight' the constraint is.

To prevent joint-limit violations, we detect the violations at each iteration and clamp the maximum joint value at that time-stamp by adding an equality constraint factor, and then continuing the optimization. The effect of this approach is similar to the process of preventing joint-limit violations in CHOMP [30].

V. Evaluation

A. Experimental Setup

We used the GTSAM [3] C++ library to implement our algorithm and OpenRAVE [5] for simulations. Two versions of our algorithm were implemented, a batch version (**GPMP2**) and an incremental version (**iGPMP2**). GPMP2 uses the Levenberg-Marquardt algorithm to solve the nonlinear least squares optimization problem, with initial $\lambda = 0.01$, and the optimization is stopped if a maximum of 100 iterations is reached, or the relative decrease in error is smaller than 10^{-4}. iGPMP2 uses the iSAM2 [11] incremental optimizer with default settings.

We conducted our experiments on two datasets that have a number of different start and goal configurations. The start and goal are specified by the Gaussian prior factor (with a very small covariance) as a soft constraint. Although the solution is theoretically inexact at start and goal, the error is trivial in our experiments and can be ignored. We: (1) used the 7-DOF WAM arm dataset used in GPMP [20], (Fig. 5 (left)) consisting of 24 unique planning problems; and (2) used the PR2's 7-DOF right arm dataset used in TrajOpt [25, 26] consisting of a total of 90 unique planning problems in *bookshelves* (Fig. 5 (center)) and *industrial* (Fig. 5 (right)) scenes. Finally, we also validated successful trajectories on a real 7-DOF WAM arm, setup in an environment identical to the simulation.[2]

B. The Batch Planner

1) Setup: We benchmarked our algorithm, GPMP2 when using interpolation (**GPMP2_inter**) during optimization against itself without using interpolation (**GPMP2_no-inter**) and other trajectory optimization algorithms - TrajOpt [25, 26], GPMP [20], CHOMP [24, 30] and STOMP [12] on both datasets. All benchmarks were run on a single thread of a 3.4GHz Intel Core i7 CPU.

All algorithms were initialized by a constant-velocity straight line trajectory in configuration space, except GPMP, which is required to be initialized by an acceleration-smooth straight line [20]. For the WAM dataset, all initialized trajectories for all algorithms were parameterized by 101 equidistant states. Since GPMP2_inter and GPMP use interpolation we initialized them with 11 equidistant states such that 9 points are interpolated between any two states (101 states effectively). Since trajectory tasks are shorter in the PR2 dataset, we used 51 states for all algorithms and 11 states with 4 interpolation points (51 states effectively) for GPMP2_inter and GPMP. GPMP, CHOMP and STOMP were allowed to optimize until a maximum of 250 iterations, or if a collision free trajectory is found (collision checking is started after optimizing for at least 10 iterations).

To keep comparisons fair we also compared against TrajOpt using only 11 states (TrajOpt-11) in both datasets since it uses continuos-time collision checking and can usually find a successful trajectory with fewer states. Although TrajOpt is faster when using fewer states, post-processing on the resulting trajectory is needed to make it executable and keep it smooth. It is interesting to note that since the continuous time-collision checking is performed only linearly, after the trajectory is post-processed it cannot offer any collision free guarantees. Our approach (and GPMP) avoid this problem when using fewer states by up-sampling the trajectory with GP interpolation and checking for collision at the interpolated points. This up-sampled trajectory remains smooth and can be used directly during execution.

2) Parameters: Parameters of our approaches include 'safety distance', ϵ and 'obstacle cost weight', σ_{obs}. Generally ϵ is selected to be about double the minimum distance to any obstacle allowed in the scene (in the benchmark we choose $\epsilon = 0.2$m for the WAM dataset and $\epsilon = 0.08$m for the PR2 dataset). σ_{obs} acts like a weight term that balances smoothness and collision-free requirements on the optimized trajectory and is set based on the trajectory requirements of the application. Larger σ_{obs} puts more weight on smoothness versus obstacle avoidance and vice versa. Fig. 6 shows an example of an optimized trajectory for PR2 with different settings of σ_{obs}. In

[2]A video of experiments is available at https://youtu.be/mVA8qhGf7So.

Table I.A Results for 24 planning problems on the 7-DOF WAM arm.

	GPMP2_inter	GPMP2_no-inter	TrajOpt-101	TrajOpt-11	GPMP	CHOMP	STOMP
Success Rate (%)	**95.8**	91.7	91.7	20.8	**95.8**	75.0	40.0
Average Time to Success (s)	0.068	0.120	0.323	**0.027**	0.590	1.337	6.038
Maximum Time to Success (s)	0.112	0.217	0.548	**0.033**	1.322	6.768	22.971

Table I.B Results for 90 planning problems on PR2's 7-DOF right arm.

	GPMP2_inter	GPMP2_no-inter	TrajOpt-51	TrajOpt-11	GPMP	CHOMP	STOMP
Success Rate (%)	**94.4**	88.9	93.3	88.9	47.8	80.0	52.4
Average Time to Success (s)	**0.033**	0.053	0.860	0.168	1.134	4.999	7.586
Maximum Time to Success (s)	**0.083**	0.120	4.827	0.455	9.516	44.623	95.679

our experiments we found that the range $[0.001, 0.02]$ works well for σ_{obs} and larger robot arms should use larger σ_{obs} (in the benchmark we choose $\sigma_{obs} = 0.02$m for the WAM dataset and $\sigma_{obs} = 0.005$ for the PR2 dataset).

3) Analysis: The benchmark results for the WAM dataset are summarized in Table I.A[3] and results for the PR2 dataset are summarized in Table I.B[4]. Average time to success and maximum time to success include only successful runs.

Evaluating motion planning algorithms is a difficult task. The algorithms here use different techniques to formulate and solve the motion planning problem, and exhibit performance that depends on initial conditions as well as a range of parameter settings that can change based on the nature of the planning problem. Therefore, in our experiments we have tuned each algorithm to the settings close to default ones that worked best for each dataset. However, we still observe that TrajOpt-11 performs poorly on the WAM dataset (possibly due to using too few states on the trajectory) while GPMP performs poorly on the PR2 dataset (possibly due to the different initialization of the trajectory, and also the start and end configurations in the dataset being very close to the obstacles).

From the results in Table I.A and I.B we see that our algorithms perform consistently well compared to other algorithms on these datasets. Using interpolation during optimization (GPMP2_inter) achieves $30 - 50\%$ speedup of average and maximum runtimes when compared to not using interpolation (GPMP2_no-inter). On the WAM dataset TrajOpt-11 has the lowest runtime but is able to solve only 20% of the problems, while GPMP2_inter and GPMP2_no-inter have the second and third lowest runtimes. On the PR2 dataset, GPMP2_inter has the lowest runtimes and GPMP2_no-inter comes in second. In all our experiments the relative decrease in error condition is always satisfied before the maximum iteration cap is reached, possibly due to the quadratic convergence rates of our algorithms. Therefore, all the failure cases were due to infeasible local minimas. Solutions like random restarts may resolve this issue, but are not currently implemented.

[3]Parameters for benchmark on the WAM dataset: For our algorithms, $Q_c = 1$. For GPMP, $Q_c = 100$. For GPMP and CHOMP, $\lambda = 0.005$, $\epsilon = 0.2$, $\eta = 1$. For STOMP, $k = 5$. For TrajOpt, coeffs $= 20$, dist_pen $= 0.05$.

[4]Parameters for benchmark on the PR2 dataset: For GPMP and CHOMP, $\epsilon = 0.05$. All remaining parameters are the same from the WAM dataset.

C. The Incremental Planner

We evaluate our incremental motion planner (iGPMP2) by benchmarking it against our batch planner (GPMP2) on replanning problems from each scene of the WAM and PR2 datasets. This replanning problem entails planning a trajectory from a start configuration to an originally assigned goal configuration. Then, at the middle time-step the trajectory is assigned a new goal configuration. This requires two changes to the factor graph: a new goal factor at the end of the trajectory to ensure that the trajectory reaches the location in configuration where the goal is changed, and a fixed state factor at the middle time step to enforce constraint of current state. GPMP2 reinitializes the trajectory as a constant-velocity straight line from the middle state to the new goal however, iGPMP2 can use the solution to the old goal and the updated Bayes Tree as the initialization to incrementally update the trajectory thus finding the solution much faster. This claim is corroborated by our experiments.

32 replanning problems are prepared for the WAM dataset and 28 replanning problems are prepared for the PR2 dataset (18 in *bookshelves* and 10 in *industrial*). GP interpolation is used and all parameters are the same as the batch benchmarks. The benchmark results are shown in Table II.A and Table II.B. We see from the results that iGPMP2 provides a significant speed-up, suffering only a small loss in the success rate, compared to GPMP2 that re-plans the trajectory from scratch.

Two possible reasons why iGPMP2's success rate suffers as compared to the GPMP2's are: (1) iGPMP2 uses the original trajectory as initialization, which may be a poor choice if the goal has moved significantly; and (2) GPMP2 uses Levenberg-Marquardt for optimization, that provides appropriate step damping helping to improve the results, but iGPMP2 does not use similar step damping in its implementation.

Examples of successfully replanned trajectories generated using iGPMP2 are shown in Fig. 7. Since configuration and velocity constraints are added at the robot states, the replanner makes a smooth transition between original trajectories and replanned trajectories, which is critical if the trajectory is being executed on a real robot.

VI. Discussion

A. Comparisons with Related Work

Although our work is inspired by the probabilistic view of motion planning from [28], instead of using message passing,

Table II.A Results for 32 replanning problems on WAM.

	iGPMP2	GPMP2
Success Rate (%)	90.6	**100.0**
Average Time to Success (ms)	**2.38**	30.21
Maximum Time to Success (ms)	**3.92**	46.60

Table II.B Results for 28 replanning problems on PR2.

	iGPMP2	GPMP2
Success Rate (%)	75.0	**96.4**
Average Time to Success (ms)	**4.27**	26.70
Maximum Time to Success (ms)	**6.67**	58.84

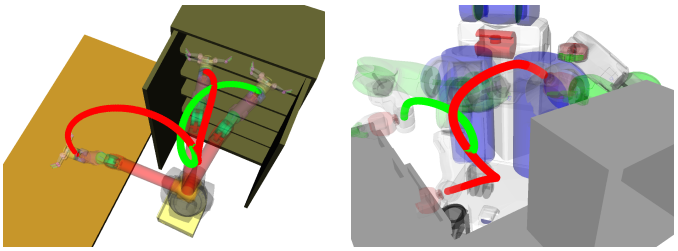


Fig. 7: Example iGPMP2 results on the WAM and PR2 *industrial*. Red lines show originally planned end-effector trajectories, and green lines show replanned end-effector trajectories. Best viewed in color.

we convert the inference problem to a nonlinear least squares problem. This allows us to use solvers that exploit the sparse structure of our problem, leading to much faster algorithms.

Both GPMP [20] and our algorithms use sparse GP's to represent the continuous-time trajectory, use GP interpolation to reduce the system states to be optimized, and use signed distance fields for collision checking. However, our approach differs from GPMP in that we fully embrace the probabilistic view of motion planning, and we formulate the problem as nonlinear least squares. Our algorithm uses optimization methods with quadratic convergence rates, allowing much faster solutions compared to GPMP (and the implementations of CHOMP and STOMP) which use a gradient decent-based technique with only linear convergence rate.

TrajOpt [25, 26] formulates the motion planning problem as constrained optimization, which allows the use of hard constraints on obstacles but also makes the optimization problem much more difficult and slower to solve. Benchmark results in Section V-B show that our approach is faster than TrajOpt even when it uses a small number of states to represent the trajectory. TrajOpt performs continuous-time collision checking and can, therefore, solve problems only with a few states in theory. However, the trajectory does not have a continuous-time representation and therefore must perform collision checking by approximating the convex-hull of obstacles and a straight line between states. This may not work in practice since a trajectory with few states would need to be post-processed to make it executable. Furthermore, depending on the post-processing method, collision-free guarantees may not exist for the final trajectory. Representing trajectories in continuous-time with GPs and using GP interpolation to up-sample them, allows our algorithms to circumvent this problem.

Finally, our framework allows us to solve replanning problems very quickly, something that none of the above trajectory optimization approaches can provide. Solving these types of problems fast is very useful in real-time real-world applications. We are able to achieve this by formulating the motion planning problem as probabilistic inference on a factor graph and is one of the major contributions of this paper.

B. Limitations & Future Work

A drawback of iterative methods for solving nonlinear least square problems is that they offer no global optimality guarantees. However, given that our objective is to satisfy *smoothness* and to be *collision-free*, a globally optimal solution is not strictly necessary. Many of the prior approaches to motion planning face similar issues of getting stuck in local minima. Several solutions [30] have been discussed in prior work including random restarts. Since our approach is fast for a single run, random restarts are a promising solution to overcome the local minima problem.

The main drawback of our proposed approach is that it is limited in its ability to handle motion constraints like nonlinear inequality constraints. Sequential quadratic programming (SQP) can be used to solve problems with such constraints, and has been used before in motion planning [25, 26]. We believe that SQP can be integrated into our trajectory optimizer, although this remains future work.

VII. Conclusions

We formulate motion planning as probabilistic inference using Gaussian processes to reason about continuous-time trajectories. Exploiting the sparse structure of our formulation, we perform fast inference on factor graphs as nonlinear least squares optimization. Using GP interpolation we can query the trajectory at any time of interest such that the initial trajectory can be parameterized by only a few states and then up-sampled during optimization to check for and propagate collision cost information to the states being optimized.

We benchmark our algorithm against several state-of-the-art trajectory optimization algorithms on 7-DOF arm planning problems on two datasets in three distinct environments and show that our approach is consistently faster, often several times faster, than its nearest competitors.

Finally, using the Bayes Tree data structure and iSAM2 optimizer we are able to incrementally solve replanning problems in a few milliseconds, which is something unique to our motion planning algorithm and highly useful for planning in real-time real-world applications.

Acknowledgments

This work is partially supported by National Institute of Food and Agriculture, U.S. Department of Agriculture, under award number 2014-67021-22556, and NSF CRII award number 1464219.

REFERENCES

[1] 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, Berkeley, USA*, 2014.

[2] Arunkumar Byravan, Byron Boots, Siddhartha S Srinivasa, and Dieter Fox. Space-time functional gradient optimization for motion planning. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 6499–6506. IEEE, 2014.

[3] Frank Dellaert. Factor graphs and GTSAM: a hands-on introduction. Technical report, Georgia Tech Technical Report, GT-RIM-CP&R-2012-002, 2012.

[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] Rosen Diankov. *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, Robotics Institute, August 2010.

[6] M. Elbanhawi, M. Simic, and R. Jazar. Randomized bidirectional B-Spline parameterization motion planning. *Intelligent Transportation Systems, IEEE Transactions on*, PP(99):1–1, 2015.

[7] Gene H Golub and Charles F Van Loan. *Matrix computations*, volume 3. JHU Press, 2012.

[8] Keliang He, Elizabeth Martin, and Matt Zucker. Multigrid CHOMP with local smoothing. In *Proc. of 13th IEEE-RAS Int. Conference on Humanoid Robots (Humanoids)*, 2013.

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

[10] 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.

[11] 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.

[12] Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, Peter Pastor, and Stefan Schaal. STOMP: Stochastic trajectory optimization for motion planning. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 4569–4574. IEEE, 2011.

[13] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, 12(4):566–580, 1996.

[14] Sven Koenig, Craig Tovey, and Yuri Smirnov. Performance bounds for planning in unknown terrain. *Artificial Intelligence*, 147(1):253–279, 2003.

[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] James J Kuffner and Steven M LaValle. RRT-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 2, pages 995–1001. IEEE, 2000.

[17] Steven M LaValle and James J Kuffner. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400, 2001.

[18] Maxim Likhachev, David I Ferguson, Geoffrey J Gordon, Anthony Stentz, and Sebastian Thrun. Anytime dynamic A*: An anytime, replanning algorithm. In *ICAPS*, pages 262–271, 2005.

[19] Zita Marinho, Anca Dragan, Arun Byravan, Byron Boots, Siddhartha Srinivasa, and Geoffrey J. Gordon. Functional gradient motion planning in reproducing kernel Hilbert space. *CoRR*, abs/1601.03648, 2016. URL http://arxiv.org/abs/1601.03648.

[20] Mustafa Mukadam, Xinyan Yan, and Byron Boots. Gaussian process motion planning. In *Proceedings of the 2016 IEEE Conference on Robotics and Automation (ICRA-2016)*, 2016.

[21] Chonhyon Park, Jia Pan, and Dinesh Manocha. ITOMP: Incremental trajectory optimization for real-time replanning in dynamic environments. In *ICAPS*, 2012.

[22] Chonhyon Park, Jia Pan, and Dinesh Manocha. Real-time optimization-based planning in dynamic environments using GPUs. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 4090–4097. IEEE, 2013.

[23] Carl Edward Rasmussen. *Gaussian processes for machine learning*. Citeseer, 2006.

[24] Nathan Ratliff, Matthew Zucker, J Andrew Bagnell, and Siddhartha Srinivasa. CHOMP: Gradient optimization techniques for efficient motion planning. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 489–494. IEEE, 2009.

[25] John Schulman, Jonathan Ho, Alex Lee, Ibrahim Awwal, Henry Bradlow, and Pieter Abbeel. Finding locally optimal, collision-free trajectories with sequential convex optimization. In *Robotics: Science and Systems*, volume 9, pages 1–10. Citeseer, 2013.

[26] John Schulman, Yan Duan, Jonathan Ho, Alex Lee, Ibrahim Awwal, Henry Bradlow, Jia Pan, Sachin Patil, Ken Goldberg, and Pieter Abbeel. Motion planning with sequential convex optimization and convex collision checking. *The International Journal of Robotics Research*, 33(9):1251–1270, 2014.

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

[28] Marc Toussaint and Christian Goerick. A Bayesian view on motor control and planning. In *From Motor Learning to Interaction Learning in Robots*, pages 227–252. Springer, 2010.

[29] Xinyan Yan, Vadim Indelman, and Byron Boots. Incremental sparse GP regression for continuous-time trajectory estimation & mapping. In *Proceedings of the International Symposium on Robotics Research (ISRR-2015)*, 2015.

[30] Matt Zucker, Nathan Ratliff, Anca D Dragan, Mihail Pivtoraiko, Matthew Klingensmith, Christopher M Dellin, J Andrew Bagnell, and Siddhartha S Srinivasa. CHOMP: Covariant Hamiltonian optimization for motion planning. *The International Journal of Robotics Research*, 32(9-10):1164–1193, 2013.