Continuous-time Gaussian process motion planning via probabilistic inference

Mustafa Mukadam*, Jing Dong*, Xinyan Yan, Frank Dellaert and Byron Boots

Abstract
We introduce a novel formulation of motion planning, for continuous-time trajectories, as probabilistic inference. We first show how smooth continuous-time trajectories can be represented by a small number of states using sparse Gaussian process (GP) models. We next develop an efficient gradient-based optimization algorithm that exploits this sparsity and Gaussian process interpolation. We call this algorithm the Gaussian Process Motion Planner (GPMP). We then detail how motion planning problems can be formulated as probabilistic inference on a factor graph. This forms the basis for GPMP2, a very efficient algorithm that combines GP representations of trajectories with fast, structure-exploiting inference via numerical optimization. Finally, we extend GPMP2 to an incremental algorithm, iGPMP2, that can efficiently replan when conditions change. We benchmark our algorithms against several sampling-based and trajectory optimization-based motion planning algorithms on planning problems in multiple environments. Our evaluation reveals that GPMP2 is several times faster than previous algorithms while retaining robustness. We also benchmark iGPMP2 on replanning problems, and show that it can find successful solutions in a fraction of the time required by GPMP2 to replan from scratch.

Keywords
Motion planning, Gaussian processes, probabilistic inference, factor graphs, trajectory optimization

1. Introduction

Motion planning is a key tool in robotics, used to find trajectories of robot states that achieve a desired task. While searching for a solution, motion planners evaluate trajectories based on two criteria: feasibility and optimality. The exact notion of feasibility and optimality can vary depending on the system, tasks, and other problem-specific requirements. In general, feasibility evaluates a trajectory based on whether or not it respects the robot or task-specific constraints such as avoiding obstacles, while reaching the desired goal. In other words, feasibility is often binary: a trajectory is feasible or it is not. In contrast with feasibility, optimality often evaluates the quality of trajectories without reference to task-specific constraints. For example, optimality may refer to the smoothness of a trajectory and encourage the motion planner to minimize dynamical criteria like velocity or acceleration. A variety of motion planning algorithms have been proposed to find trajectories that are both feasible and optimal. These approaches can be roughly divided into two categories: sampling-based algorithms and trajectory optimization algorithms.

Sampling-based algorithms (Kavraki et al. 1996; Kuffner and LaValle 2000; LaValle 2006) can effectively find feasible trajectories for high dimensional systems but the trajectories often exhibit jerky and redundant motion and therefore require post processing to address optimality. Although optimal planners (Karaman and Frazzoli 2010) have been proposed, they are computationally inefficient on high-dimensional problems with challenging constraints.

Trajectory optimization algorithms (Ratliff et al. 2009; Zucker et al. 2013; Kalakrishnan et al. 2011; He et al. 2013; Byravan et al. 2014; Marinho et al. 2016) minimize an objective function that encourages trajectories to be both feasible and optimal. A drawback of these approaches is that, in practice, a fine discretization of the trajectory is necessary to integrate cost information when reasoning about thin obstacles and tight constraints. Additionally,
trajectory optimization is locally optimal, and may need to be rerun with different initial conditions to find a feasible solution, which can incur high computational cost. A solution to this latter problem is to initialize trajectory optimization with the solution discovered by a sampling-based algorithm. TrajOpt (Schulman et al. 2013, 2014) attempts to avoid finely discretized trajectories by formulating trajectory optimization as sequential quadratic programming. It achieves reduced computational costs by parameterizing the trajectory with a small number of states and employing continuous-time collision checking. However, due to the discrete-time representation of the trajectory, a sparse solution may need post-processing for execution and may not remain collision-free. In other words, a fine discretization may still be necessary on problems in complex environments.

A continuous-time trajectory representation can avoid some of these challenges to yield a more efficient approach (Elbanhawi et al. 2015; Marinho et al. 2016). In this work, we adopt a continuous-time representation of trajectories; specifically, we view trajectories as functions that map time to robot state. We assume these functions are sampled from a Gaussian process (GP) (Rasmussen 2006). We will show that GPs can inherently provide a notion of trajectory optimality through a prior. Efficient structure-exploiting GP Regression (GPR) can be used to query the trajectory at any time of interest in $O(1)$. Using this representation, we develop a gradient-based optimization algorithm called GPMP (Gaussian Process Motion Planner) that can efficiently overcome the large computational costs of fine discretization while still maintaining smoothness in the result.

Through the GP formulation, we can view motion planning as probabilistic inference (Toussaint 2009; Toussaint and Goerick 2010). Similar to how the notion of trajectory optimality is captured by a prior on trajectories, the notion of feasibility can also be viewed probabilistically as well and encoded in a likelihood function. Bayesian inference can then be used to compute a solution to our motion planning problem efficiently through the use of factor graphs (Kschischang et al. 2001). The duality between inference and optimization allows us to perform efficient inference on factor graphs by solving sparse least squares problems, thereby exploiting the structure of the underlying system. Similar techniques have been used to solve large-scale Simultaneous Localization and Mapping (SLAM) problems (Dellaert and Kaess 2006). With this key insight we can use preexisting efficient optimization tools developed by the SLAM community, and use them in the context of motion planning. This results in the GPMP2 algorithm, which is more efficient than previous motion planning algorithms.

Another advantage of GPMP2 is that we can easily extend the algorithm using techniques designed for incremental inference on factor graphs developed in the context of SLAM. For example, Incremental Smoothing and Mapping (iSAM) (Kaess et al. 2008, 2011b) can be adapted to efficiently solve replanning problems.

In this paper, we provide a revised and extended version of our previous work (Mukadam et al. 2016; Dong et al. 2016), give more theoretical insight, and proof for the sparsity of the linear system in GPMP2. We also conduct benchmarks on larger datasets than before and compare GPMP and GPMP2 against popular and leading trajectory optimization based (Zucker et al. 2013; Schulman et al. 2014) as well as sampling-based motion planning (Kuffner and LaValle 2000; Sucan and Kavraki 2009; Sucan et al. 2012) algorithms in multiple environments with multiple systems on standard reaching tasks (Figure 1). Our results show GPMP2 to be several times faster than the state-of-the-art with higher success rates. We also benchmark GPMP2 against our incremental planner, iGPMP2, on replanning tasks and show that iGPMP2 can incrementally solve replanning problems an order of magnitude faster than GPMP2 solving from scratch.

2. Related work

Most motion planning algorithms are categorized as either sampling-based or trajectory optimization-based...
algorithms. Sampling-based planners such as probabilistic roadmaps (PRMs) (Kavraki et al. 1996) construct a dense graph from random samples in obstacle-free areas of the robot’s configuration space. PRMs can be used for multiple queries by finding the shortest path between a start and goal configuration in the graph. Rapidly exploring random trees (RRTs) (Kuffner and LaValle 2000) find trajectories by incrementally building space-filling trees through directed sampling. RRTs are very good at finding feasible solutions in highly constrained problems and high-dimensional search spaces. Both PRMs and RRTs offer probabilistic completeness, ensuring that, given enough time, a feasible trajectory can be found, if one exists. Despite guarantees, sampling-based algorithms may be difficult to use in real-time applications due to computational challenges. Often computation is wasted exploring regions that may not lead to a solution, leading to recent work in informed techniques (Gammell et al. 2015) that bias the sampling approach to make them more tractable.

In contrast with sampling-based planners, trajectory optimization starts with an initial, possibly infeasible, trajectory and then optimizes the trajectory by minimizing a cost function. Covariant Hamiltonian Optimization for Motion Planning (CHOMP) and related methods (Ratliff et al. 2009; Zucker et al. 2013; He et al. 2013; Byravan et al. 2014; Marinho et al. 2016) optimize a cost functional using covariant gradient descent, while Stochastic Trajectory Optimization for Motion Planning (STOMP) (Kalakrishnan et al. 2011) optimizes non-differentiable costs by stochastic sampling of noisy trajectories. TrajOpt (Schulman et al. 2013, 2014) solves a sequential quadratic program and performs convex continuous-time collision checking. Trajectory optimization methods are very fast, however, unlike sampling-based planners, they methods will only find a locally optimal solution. The computational bottleneck results from evaluating costs on a fine discretization of the trajectory or, in difficult problems, repeatedly changing the initial conditions until a feasible trajectory is discovered.

Continuous-time trajectory representations can overcome the computational cost incurred by finely discretizing the trajectory. Linear interpolation (Bosse and Zlot 2009; Li et al. 2013; Dong and Barfoot 2014), splines (Bibby and Reid 2010; Anderson and Barfoot 2013; Furgale et al. 2013; Leutenegger et al. 2015; Patron-Perez et al. 2015; Furgale et al. 2015), and hierarchical wavelets (Anderson et al. 2014) have been used to represent trajectories in filtering and state estimation. Recently, B-Splines (Elbanhawi et al. 2015) and kernel methods (Marinho et al. 2016) have similarly been used to represent trajectories with fewer states in motion planning problems.

In this work we use Gaussian process (GPs) (Rasmussen 2006) to parametrize and reason about continuous-time trajectories. GPs have been used for function approximation in supervised learning (Vijayakumar et al. 2005; Kersting et al. 2007), inverse dynamics modeling (Nguyen-Tuong et al. 2008; Sturm et al. 2009), reinforcement learning (Deisenroth and Rasmussen 2011), path prediction (Tay and Laugier 2008), simultaneous localization and mapping (Barfoot et al. 2014; Yan et al. 2017), state estimation (Ko and Fox 2009; Tong et al. 2012), and controls (Theodorou et al. 2010), but to our knowledge GPs have not been used in motion planning.

We also consider motion planning from the perspective of probabilistic inference. Early work by Attias (2003) uses inference to solve Markov decision processes. More recently, solutions to planning and control problems have used probabilistic tools such as expectation propagation (Toussaint 2009), expectation maximization (Toussaint and Storkey 2006; Levine and Koltun 2013), and KL-minimization (Rawlik et al. 2012). We exploit the duality between inference and optimization to perform inference on factor graphs by solving nonlinear least square problems. While this is an established and efficient approach (Dellaert and Kaess 2006) to solving large scale SLAM problems, we introduce this technique in the context of motion planning. Incremental inference can also be performed efficiently on factor graphs (Kaess et al. 2008, 2011b), a fact we take advantage of to solve replanning problems.

Replanning involves adapting a previously solved solution to changing conditions. Early replanning work like D’ (Koenig et al. 2003) and Anytime A’ (Likhachev et al. 2005) need a finely discretized state space and therefore do not scale well with high dimensional problems. Recent trajectory optimization algorithms inspired from CHOMP (Ratliff et al. 2009) like incremental trajectory optimization for motion planning (ITOMP) (Park et al. 2012) can fluently replan using a scheduler that enforces timing restrictions but the solution cannot guarantee feasibility. GPUs have been suggested as a way to increase the speed of replanning (Park et al. 2013), with some success. Our algorithm is inspired from the incremental approach to SLAM problems (Kaess et al. 2011b) that can efficiently update factor graphs to generate new solutions without performing redundant calculations. During planning, we use this method to update the trajectory only where necessary, thus reducing computational costs and making fast replanning possible.

3. Motion planning as trajectory optimization

The goal of motion planning via trajectory optimization is to find trajectories θ(t) that satisfy constraints and minimize costs (Zucker et al. 2013; Kalakrishnan et al. 2011; Schulman et al. 2014). Motion planning can therefore
be formalized as
\[
\begin{align*}
\text{minimize} & \quad \mathcal{F}[\theta(t)] \quad \text{(1)} \\
\text{subject to} & \quad G_i[\theta(t)] \leq 0, \quad i = 1, \ldots, m_{ineq} \\
& \quad H_i[\theta(t)] = 0, \quad i = 1, \ldots, m_{eq}
\end{align*}
\]
where the trajectory $\theta(t)$ is a continuous-time function, mapping time $t$ to robot states, which are generally configurations (and possibly higher-order derivatives). $\mathcal{F}[\theta(t)]$ is an objective or cost functional that evaluates the quality of a trajectory and usually encodes smoothness that minimizes higher-order derivatives of the robot states (for example, velocity or acceleration) and collision costs that enforces the trajectory to be collision-free. $G_i[\theta(t)]$ are inequality constraint functionals such as joint angle limits, and $H_i[\theta(t)]$ are task-dependent equality constraints, such as the desired start and end configurations and velocities, or the desired end-effector orientation (for example, holding a cup filled with water upright). The number of inequality or equality constraints may be zero, depending on the specific problem. Based on the optimization technique used to solve Eq. (1), collision cost may also appear as an obstacle avoidance inequality constraint (Schulman et al. 2014). In practice, most existing trajectory optimization algorithms work with a fine discretization of the trajectory, which can be used to reason about thin obstacles or tight navigation constraints, but can incur a large computational cost.

4. Gaussian processes for continuous-time trajectories

A vector-valued Gaussian process (GP) (Rasmussen 2006) provides a principled way to reason about continuous-time trajectories, where the trajectories are viewed as functions that map time to state. In this section, we describe how GPs can be used to encode a prior on trajectories such that map time to state. In this section, we describe how trajectories, where the trajectories are viewed as functions provides a principled way to reason about continuous-time trajectories that will be useful in optimization techniques used to solve Eq. (1), collision cost may also appear as an obstacle avoidance inequality constraint (Schulman et al. 2014). In practice, most existing trajectory optimization algorithms work with a fine discretization of the trajectory, which can be used to reason about thin obstacles or tight navigation constraints, but can incur a large computational cost.

4.2 A Gauss-Markov model

Similar to previous work (Sarkka et al. 2013; Barfoot et al. 2014), we use a structured kernel generated by a linear time-varying stochastic differential equation (LTV-SDE)
\[
\dot{\theta}(t) = A(t)\theta(t) + u(t) + F(t)w(t),
\]
where $u(t)$ is the known system control input, $A(t)$ and $F(t)$ are time-varying matrices of the system, and $w(t)$ is generated by a white noise process. The white noise process is itself a zero-mean GP
\[
w(t) \sim GP(0, QC\delta(t - t')).
\]
$QC$ is the power-spectral density matrix and $\delta(t - t')$ is the Dirac delta function. The solution to the initial value problem of this LTV-SDE is
\[
\theta(t) = \Phi(t, t_0)\theta_0 + \int_{t_0}^{t} \Phi(t, s)(u(s) + F(s)w(s)) ds,
\]
where $\Phi(t, s)$ is the state transition matrix, which transfers state from time $s$ to time $t$. The mean and covariance

Figure 2. An example GP prior for trajectories. The dashed line is the mean trajectory $\mu(t)$ and the shaded area indicates the covariance. The 5 solid lines are sample trajectories $\theta(t)$ from the GP prior.

\[\theta_0 \quad \theta_N\]

\[\text{with the mean vector } \mu \text{ and covariance kernel } \mathcal{K} \text{ defined as }\]
\[
\mu = [\mu(t_0) \ldots \mu(t_N)]^\top, \quad \mathcal{K} = [\mathcal{K}(t_i, t_j)]_{i,j=0,1}^{N}.
\]
\[\theta_i \in \mathbb{R}^D \text{ are support states that parameterize the continuous-time trajectory, where } D \text{ is the dimensionality of state.}\]
functions of the GP defined by this LTV-SDE are calculated by taking the first and second moments respectively on Eq. (7).

\[
\bar{\mu}(t) = \Phi(t, t_0)\mu_0 + \int_{t_0}^{t} \Phi(t, s)u(s)\,ds, \quad (8)
\]

\[
\bar{\mathcal{K}}(t, t') = \Phi(t, t_0)\mathcal{K}_0\Phi(t', t_0)^\top + \int_{t_0}^{\min(t,t')} \Phi(t, s)F(s)Q_cF(s)^\top\Phi(t', s)^\top\,ds. \quad (9)
\]

\(\mu_0\) and \(\mathcal{K}_0\) are the initial mean and covariance of the start state respectively.

The desired prior of trajectories between a given start state \(\theta_0\) and goal state \(\theta_N\) for a finite set of support states, as described in Section 4.1, can be found by conditioning this GP with a fictitious observation on the goal state with mean \(\mu_N\) and covariance \(\mathcal{K}_N\). Specifically

\[
\mu = \bar{\mu} + \bar{\mathcal{K}}(t_N, t)^\top(\bar{\mathcal{K}}(t_N, t_N) + \mathcal{K}_N)^{-1}(\theta_N - \mu_N) \quad (10)
\]

\[
\mathcal{K} = \bar{\mathcal{K}} - \bar{\mathcal{K}}(t_N, t)^\top(\bar{\mathcal{K}}(t_N, t_N) + \mathcal{K}_N)^{-1}\bar{\mathcal{K}}(t, t), \quad (11)
\]

where \(\bar{\mathcal{K}}(t_N, t) = [\bar{\mathcal{K}}(t_N, t_0) \ldots \bar{\mathcal{K}}(t_N, t_N)]\) (see Appendix A for proof).

This particular construction of the prior leads to a Gauss-Markov model that generates a GP with an exactly sparse tridiagonal precision matrix (inverse kernel) that can be factored as:

\[
\mathcal{K}^{-1} = B^\top Q^{-1} B \quad (12)
\]

with,

\[
B = \begin{bmatrix}
I & 0 & \ldots & 0 & 0 \\
-\Phi(t_1, t_0) & I & \ldots & 0 & 0 \\
0 & -\Phi(t_2, t_1) & \ddots & \vdots & \vdots \\
\vdots & \ddots & \ddots & I & 0 \\
0 & 0 & \ldots & -\Phi(t_N, t_{N-1}) & I \\
0 & 0 & \ldots & 0 & I
\end{bmatrix}
\]

\[
Q^{-1} = \text{diag}(\mathcal{K}_0^{-1}, Q_{0,1}, \ldots, Q_{N-1,N}, \mathcal{K}_N^{-1}), \quad (14)
\]

\[
Q_{a,b} = \int_{t_a}^{t_b} \Phi(b, s)F(s)Q_cF(s)^\top\Phi(b, s)^\top\,ds \quad (15)
\]

(see Appendix A for proof). This sparse structure is useful for fast GP interpolation (Section 4.3) and efficient optimization (Section 5 and 6).

An interesting observation here is that this choice of kernel can be viewed as a generalization of CHOMP (Zucker et al. 2013). For instance, if the identity and zero blocks in the precision matrix are scalars, the state transition matrix \(\Phi\) is a unit scalar, and \(Q^{-1}\) is an identity matrix, \(\mathcal{K}^{-1}\) reduces to the matrix \(A\) formed by finite differencing in CHOMP. In this context, it means that CHOMP considers a trajectory of positions in configuration space, that is generated by a deterministic differential equation (since \(Q^{-1}\) is identity).

The linear model in Eq. (5) is sufficient to model kinematics for the robot manipulators considered in the scope of this work, however our framework can be extended to consider non-linear models following Anderson et al. (2015).

### 4.3 Gaussian process interpolation

One of the primary benefits of using Gaussian processes in motion planning is that although continuous-time trajectories are represented as functions parameterized by only a sparse set of support states, they can be queried at any time of interest through Gaussian process interpolation. The reduced parameterization makes each iteration of the optimization efficient. Given the choice of the structured prior from the previous subsection, rich collision costs between the support states can be evaluated by performing dense GP interpolation between the support states quickly and efficiently. This cost can then be used to
update the support states in a meaningful manner, reducing the computational effort. A much denser resolution of interpolation (Figure 4) can also be useful in practice to feed the trajectory to a controller on a real robot.

The process of updating a trajectory with GP interpolation is explained through an example illustrated in Figure 3. At each iteration of optimization, the trajectory with a sparse set of support states can be densely interpolated with a large number of states, and the collision cost can be evaluated on all the states (both support and interpolated). Next, collision costs at the interpolated states are propagated and accumulated to the nearby support states (the exact process to do this is explained in Section 5.3 and 6.2). Finally, the trajectory is updated by only updating the support states given the accumulated cost information.

Following Sarkka et al. (2013); Barfoot et al. (2014); Yan et al. (2017), we show how we exploit the structured prior to perform fast GP interpolation. The posterior mean of the trajectory at any time $\tau$ can be found in terms of the current trajectory $\theta$ at time points $t$ (Rasmussen 2006) by conditioning on the support states that parameterize trajectory:

$$θ(τ) = \tilde{μ}(τ) + \tilde{K}(τ, t)\tilde{K}^{-1}(1) (θ - \tilde{μ})$$

i.e. performing Gaussian process regression. Although the interpolation in Eq. (16) naively requires $O(N)$ operations, $θ(τ)$ can be computed in $O(1)$ by leveraging the structure of the sparse GP prior generated by the Gauss-Markov model introduced in Section 4. This implies that $θ(τ)$ at $τ, t_i < τ < t_i+1$ can be expressed as a linear combination of only the adjacent function values $θ_i$ and $θ_{i+1}$ and is efficiently computed as

$$θ(τ) = \tilde{μ}(τ) + Λ(τ)(θ_i - \tilde{μ}_i) + Ψ(τ)(θ_{i+1} - \tilde{μ}_{i+1})$$

where

$$Λ(τ) = Φ(τ, t_i) - Ψ(τ)Φ(t_{i+1}, t_i)$$
$$Ψ(τ) = Q_{i+1, τ}Φ(t_{i+1}, τ)^TQ_{i+1, τ}^{-1}$$

is derived by substituting

$$\tilde{K}(τ)\tilde{K}^{-1} = [0 \ldots 0 Λ(τ) Ψ(τ) 0 \ldots 0]$$

in Eq. (16) with only the $(i)^{th}$ and $(i + 1)^{th}$ block columns being non-zero.

This provides an elegant way to do fast GP interpolation on the trajectory that exploits the structure of the problem. In Section 5.3 and 6.2 we show how this is utilized to perform efficient optimization.

### 5. Gaussian process motion planning

We now describe the Gaussian Process Motion Planner (GPMP), which combines the Gaussian process representation with a gradient descent-based optimization algorithm for motion planning.

#### 5.1 Cost functionals

Following the problem definition in Eq. (1) we design the objective functional as

$$F[θ(t)] = F_{obs}[θ(t)] + λF_{gp}[θ(t)]$$

where $F_{gp}$ is the GP prior cost functional (the negative natural logarithm of prior distribution) from Eq. (4)

$$F_{gp}[θ(t)] = F_{gp}[θ] = \frac{1}{2}\|θ - μ\|^2_{K}$$

penalizing the deviation of the parameterized trajectory from the prior mean, $F_{obs}$ is the obstacle cost functional that penalizes collision with obstacles and $λ$ is the trade-off between the two functionals.

As discussed in Section 4.2 the GP smoothness prior can be considered a generalization to the one used in practical applications of CHOMP constructed through finite dynamics. In contrast to CHOMP, we also consider our trajectory to be augmented by velocities and acceleration. This allows us to keep the state Markovian in the prior model (Section 4.2), is useful in computation of the obstacle cost gradient (Section 5.2), and also allows us to stretch or squeeze the trajectory in space while keeping the states on the trajectory temporally equidistant (Byravan et al. 2014).

The obstacle cost functional $F_{obs}$ is also similar to the one used in CHOMP (Zucker et al. 2013). This functional computes the arc-length parameterized line integral of the workspace obstacle cost of each body point as it passes through the workspace, and integrates over all body points:

$$F_{obs}[θ(t)] = \int_{t_0}^{t_N} \int_B c(x)\|\dot{x}\|\,du\,dt$$

where $c(\cdot): \mathbb{R}^3 \to \mathbb{R}$ is the workspace cost function that penalizes the set of points $B \subset \mathbb{R}^3$ on the robot body when they are in or around an obstacle, and $x$ is the forward kinematics that maps robot configuration to workspace (see Zucker et al. (2013) for details).
In practice, the cost functional can be approximately evaluated on the discrete support state parameterization of the trajectory \( \mathcal{F}_{\text{obs}}[\theta(t)] = \mathcal{F}_{\text{obs}}[\Theta] \), the obstacle cost is calculated using a precomputed signed distance field (see Section 8.1), and the inner integral is replaced with a summation over a finite number of body points that well approximate the robot’s physical body.

### 5.2 Optimization

We adopt an iterative, gradient-based approach to minimize the non-convex objective functional in Eq. (18). In each iteration, we form an approximation to the cost functional via a Taylor series expansion around the current parameterized trajectory \( \theta \):

\[
\mathcal{F}[\theta + \delta \theta] \approx \mathcal{F}[\theta] + \nabla \mathcal{F}[\theta] \delta \theta \tag{21}
\]

We next minimize the approximate cost while constraining the trajectory to be close to the previous one. Then the optimal perturbation \( \delta \theta^* \) to the trajectory is:

\[
\delta \theta^* = \arg\min_{\delta \theta} \left\{ \mathcal{F}[\theta] + \nabla \mathcal{F}[\theta] \delta \theta + \frac{\eta}{2} \| \delta \theta \|^2_{\mathcal{K}} \right\} \tag{22}
\]

where \( \eta \) is the regularization constant. Differentiating the right-hand side and setting the result to zero we obtain the update rule for each iteration:

\[
\nabla \mathcal{F}[\theta] + \eta \mathcal{K}^{-1} \delta \theta^* = 0 \quad \Rightarrow \quad \delta \theta^* = -\frac{1}{\eta} \mathcal{K} \nabla \mathcal{F}[\theta] \]

\[
\theta \leftarrow \theta + \delta \theta^* = \theta - \frac{1}{\eta} \mathcal{K} \nabla \mathcal{F}[\theta] \tag{23}
\]

To compute the update rule we need to find the gradient of the cost functional at the current trajectory

\[
\nabla \mathcal{F}[\theta] = \nabla \mathcal{F}_{\text{obs}}[\theta] + \lambda \nabla \mathcal{F}_{\text{gp}}[\theta], \tag{24}
\]

which requires computing the gradients of the GP and obstacle cost functional. The gradient of the GP prior cost can be computed by taking the derivative of Eq. (19) with respect to the current trajectory

\[
\mathcal{F}_{\text{gp}}[\theta] = \frac{1}{2}(\theta - \mu)^\top \mathcal{K}^{-1}(\theta - \mu)
\]

\[
\nabla \mathcal{F}_{\text{gp}}[\theta] = \mathcal{K}^{-1}(\theta - \mu) \tag{25}
\]

The gradient of the obstacle cost functional can be computed from the Euler-Lagrange equation (Courant and Hilbert 1966) in which a functional of the form \( \mathcal{F}[\theta(t)] = \int v(\theta(t)) \, dt \) yields a gradient

\[
\nabla \mathcal{F}[\theta(t)] = \frac{\partial v}{\partial \theta(t)} - \frac{d}{dt} \frac{\partial v}{\partial \dot{\theta}(t)} \tag{26}
\]

Applying Eq. (26) to find the gradient of Eq. (20) in the workspace and then mapping it back to the configuration space via the kinematic Jacobian \( J \), and following the proof by Quinlan (1994), we compute the gradient with respect to configuration position, velocity, and acceleration at any time point \( t_i \) as

\[
\nabla \mathcal{F}_{\text{obs}}[\theta_i] = \left[ \int_B J^\top \| \dot{x} \| \left[ (I - \hat{x} \hat{x}^\top) \nabla_c - c \kappa \right] \, du \right] \int_B J^\top c \dot{x} \, du \begin{bmatrix} 0 \end{bmatrix} \tag{27}
\]

where \( \kappa = ||\dot{x}||^{-2}(I - \hat{x} \hat{x}^\top)\hat{x} \) is the curvature vector along the workspace trajectory traced by a body point, \( \dot{x}, \ddot{x} \) are the velocity and acceleration respectively, of that body point determined by forward kinematics and the Hessian, and \( \hat{x} = \dot{x}/||\dot{x}|| \) is the normalized velocity vector. Due to the augmented state, the velocity and acceleration can be obtained through the Jacobian and Hessian directly from the state. This is in contrast to CHOMP, which approximates the velocity and acceleration through finite differencing. The gradients at each time point are stacked together into a single vector \( g = \nabla \mathcal{F}_{\text{obs}}[\theta] \). We plug the cost gradients back into the update rule in Eq. (23) to get the update

\[
\theta \leftarrow \theta - \frac{1}{\eta} \mathcal{K} \left( \lambda \mathcal{K}^{-1}(\theta - \mu) + g \right) \tag{28}
\]

This update rule can be interpreted as a generalization of the update rule for CHOMP with an augmented trajectory and a generalized prior.

### 5.3 Compact trajectory representations and faster updates via GP interpolation

In this section, we show that the finite number of states used to parameterize smooth trajectories can be very sparse in practice. Through GP interpolation, we can up-sample the trajectory to any desired resolution, calculate costs and gradients at this resolution, and then project the gradients back to just the sparse set of support states. To interpolate \( n_{ip} \) states between two support states at \( t_i \) and \( t_{i+1} \), we define two aggregated matrices using Eq. (17),

\[
\mathbf{A}_i = \begin{bmatrix} \mathbf{A}_{i,1}^\top & \ldots & \mathbf{A}_{i,j}^\top & \ldots & \mathbf{A}_{i,n_{ip}}^\top \end{bmatrix}^\top
\]

\[
\mathbf{\Psi}_i = \begin{bmatrix} \mathbf{\Psi}_{i,1}^\top & \ldots & \mathbf{\Psi}_{i,j}^\top & \ldots & \mathbf{\Psi}_{i,n_{ip}}^\top \end{bmatrix}^\top
\]

If we want to up-sample a sparse trajectory \( \Theta \) by interpolating \( n_{ip} \) states between every support state, we can quickly compute the new trajectory \( \Theta_{up} \) as

\[
\Theta_{up} = \mathbf{M}(\Theta - \mu) + \mu_{up} \tag{29}
\]
where $\mu_{up}$ corresponds to the prior mean with respect to the up sampled trajectory, and

$$M = \begin{bmatrix}
1 & 0 & 0 & \ldots & \ldots & \ldots & 0 & 0 \\
A_0 & \Psi_0 & 0 & \ldots & \ldots & \ldots & 0 & 0 \\
0 & 1 & 0 & \ldots & \ldots & \ldots & 0 & 0 \\
0 & A_1 & \Psi_1 & \ldots & \ldots & \ldots & 0 & 0 \\
\vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\
0 & 0 & \ldots & 1 & 0 & \ldots & 0 & 0 \\
0 & 0 & \ldots & A_1 & \Psi_1 & \ldots & 0 & 0 \\
0 & 0 & \ldots & 0 & 1 & \ldots & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & \ldots & \ldots & \ldots & \ldots & A_{N-1} & \Psi_{N-1} \\
0 & 0 & \ldots & \ldots & \ldots & \ldots & 0 & 1
\end{bmatrix} \quad (30)$$

is a tall matrix that up-samples a sparse trajectory $\theta$ with only $N + 1$ support states to trajectory $\theta_{up}$ with $(N + 1) + N \times n_{up}$ states. The fast, high-temporal-resolution interpolation is also useful in practice if we want to feed the planned trajectory into a controller.

The efficient update rule is defined analogous to Eq. (28) except on a sparse parametrization of the trajectory

$$\theta \leftarrow \theta - \frac{1}{\eta} \mathcal{K}\left(\lambda\mathcal{K}^{-1}(\theta - \mu) + M^Tg_{up}\right) \quad (31)$$

where the obstacle gradient over the sparse trajectory is found by chain rule using Eq. (29) and the obstacle gradient, $g_{up}$ over the up-sampled trajectory. In other words, the above equation calculates the obstacle gradient for all states (interpolated and support) and then projects them back onto just the support states using $M^T$. Cost information between support states is still utilized to perform the optimization, however only a sparse parameterization is necessary making the remainder of the update more efficient.

GPMP demonstrates how a continuous-time representation of the trajectory using GPs can generalize CHOMP and improve performance through sparse parameterization. However, the gradient-based optimization scheme has two drawbacks: first convergence is slow due to the large number of iterations required to get a feasible solution; and, second, the gradients can be costly to calculate (See Figure 12). We improve upon GPMP and address these concerns in the next section.

6. Motion planning as probabilistic inference

To fully evoke the power of GPs, we view motion planning as probabilistic inference. A similar view has been explored before by Toussaint et al. (Toussaint 2009; Toussaint and Goerick 2010). Unlike this previous work, which uses message passing to perform inference, we exploit the duality between inference and optimization and borrow ideas from the SLAM community for a more efficient approach. In particular, we use tools from the Smoothing and Mapping (SAM) framework (Dellaert and Kaess 2006) that performs inference on factor graphs by solving a nonlinear least squares problem (Kschischang et al. 2001). This approach exploits the sparsity of the underlying problem to obtain quadratic convergence.

The probabilistic inference view of motion planning provides several advantages:

1. The duality between inference and least squares optimization allows us to perform inference very efficiently, so motion planning is extremely fast.
2. Inference tools from other areas of robotics, like the incremental algorithms based on the Bayes tree data structure (Kaess et al. 2011b), can be exploited and used in the context of planning. These tools can help speed up replanning.
3. Inference can provide a deeper understanding of the connections between different areas of robotics, such as planning and control (Mukadam et al. 2017a), estimation and planning (Mukadam et al. 2017b), and learning from demonstration and planning (Rana et al. 2017).

In this section we first develop the GPMP2 algorithm, which is more efficient compared to GPMP. In Section 7, we show how Bayes trees can be used to develop a more efficient algorithm for replanning. Finally, we discuss theoretical connections to other areas in Section 10.

6.1 Maximum a posteriori inference

To formulate this problem as inference, we seek to find a trajectory parameterized by $\theta$ given desired events $e$. For example, binary events $e_i$ at $t_i$ might signify that the trajectory is collision-free if all $e_i = 0$ (i.e. $e = 0$) and in collision if any $e_i = 1$. In general, the motion planning problem can be formulated with any set of desired events, but we will primarily focus on the collision-free events in this paper.

The posterior density of $\theta$ given $e$ can be computed by Bayes rule from a prior and likelihood

$$p(\theta|e) = p(\theta)p(e|\theta)/p(e) \quad (32)$$

$$\propto p(\theta)p(e|\theta), \quad (33)$$

where $p(\theta)$ is the prior on $\theta$ that encourages smooth trajectories, and $p(e|\theta)$ is the likelihood which specifies that collision-free trajectories are more likely to be successful.

The optimal trajectory $\theta$ is found by the maximum a posteriori (MAP) estimator, which chooses the trajectory

---

Submitted to International Journal of Robotics Research (IJRR) as manuscript #IJR-17-2909.
that maximizes the posterior \( p(\theta|e) \)

\[
\theta^* = \arg\max_\theta p(\theta|e) = \arg\max_\theta p(\theta)l(\theta; e),
\]

(35)

where \( l(\theta; e) \) is the likelihood of states \( \theta \) given events \( e \) on the whole trajectory

\[
l(\theta; e) \propto p(e|\theta).
\]

(36)

We use the same GP prior as in Section 4

\[
p(\theta) \propto \exp \left\{ -\frac{1}{2} \| \theta - \mu \|^2_\Sigma \right\}.
\]

(37)

The collision free likelihood is defined as a distribution in the exponential family

\[
l(\theta; e) = \exp \left\{ -\frac{1}{2} \| h(\theta) - y \|^2_{\Sigma_{obs}} \right\}
\]

(38)

where \( h(\theta) \) is a vector-valued obstacle cost for the trajectory, and \( \Sigma_{obs} \) is a diagonal matrix and the hyperparameter of the distribution. The specific obstacle cost used in our implementation is defined in Section 6.2.

### 6.2 Factor graph formulation

Given the Markovian structure of the trajectory and sparsity of inverse kernel matrix, the posterior distribution can be further factored such that MAP inference can be equivalently viewed as performing inference on a factor graph (Kschischang et al. 2001).

A factor graph \( G = \{\Theta, F, E\} \) is a bipartite graph, which represents a factored function, where \( \Theta = \{\theta_0, \ldots, \theta_N\} \) are a set of variable nodes, \( F = \{f_0, \ldots, f_M\} \) are a set of factor nodes, and \( E \) are edges connecting the two type of nodes.

In our problems, the factorization of the posterior distribution can be written as

\[
p(\theta|e) \propto \prod_{m=1}^M f_m(\Theta_m),
\]

(39)

where \( f_m \) are factors on variable subsets \( \Theta_m \).

Given the tridiagonal inverse kernel matrix defined by Eq. (12)-(14), we factor the prior

\[
p(\theta) \propto f_0^p(\theta_0)f_N^p(\theta_N) \prod_{i=0}^{N-1} f_i^{gp}(\theta_i, \theta_{i+1}),
\]

(40)

where \( f_0^p(\theta_0) \) and \( f_N^p(\theta_N) \) define the prior distributions on start and end states respectively

\[
f_i^{gp}(\theta_i) = \exp \left\{ -\frac{1}{2} \| \theta_i - \mu_i \|^2_{\Sigma_i} \right\}, i = 0 \text{ or } N
\]

(41)

\[
f_i^{obs} = \exp \left\{ -\frac{1}{2} \| \theta_i - \mu_i \|^2_{\Sigma_{obs}} \right\}, i = 0 \text{ or } N
\]

(42)

where \( \Sigma_0 \) and \( \Sigma_N \) are covariance matrices on start and end states respectively, and \( \mu_0 \) and \( \mu_N \) are prior (known) start and end states respectively. The GP prior factor is

where \( K_0 \) and \( K_N \) are covariance matrices on start and end states, respectively, and \( \mu_0 \) and \( \mu_N \) are prior (known) start and end states. The GP prior factor is

\[
f_i^{obs}(\theta_i) = \exp \left\{ -\frac{1}{2} \| \theta_i - \mu_i \|^2_{\Sigma_{obs}} \right\}, i = 0 \text{ or } N
\]

(43)

(44)

where \( \Sigma_{obs} \) is a M-dimensional vector-valued obstacle cost function for a single state, and \( \Sigma_{obs} \) is a \( M \times M \) hyperparameter matrix.

---

Prepared using sagej.cls

Submitted to International Journal of Robotics Research (IJRR) as manuscript #IJR-17-2909.
The interpolated obstacle factor describes the obstacle cost at \( \tau_j \), which is not on any support state and needs be interpolated from the support states. Since the Gauss-Markov model we choose enables fast interpolation from adjacent states, we can interpolate a state at any \( \tau_j \) from \( \theta_i \) and \( \theta_{i+1} \) by Eq. (17), which satisfies \( t_i < \tau_j < t_{i+1} \). This allows us to derive a binary interpolated obstacle factor that relates the cost at an interpolated point to the adjacent two trajectory states

\[
\begin{align*}
J^{intp}_{\tau_j}(\theta_i, \theta_{i+1}) &= \exp \left\{ -\frac{1}{2} \left\| \mathbf{h}(\theta(\tau_j)) \right\|_\Sigma^2 \right\} \\
&= \exp \left\{ -\frac{1}{2} \left\| \mathbf{h}^{intp}(\theta_i, \theta_{i+1}) \right\|^2_\Sigma \right\}.
\end{align*}
\]  

In other words, \( \theta(\tau_j) \) is a function of \( \theta_i \) and \( \theta_{i+1} \) (see Eq. (17)). Just like in GPMP, here too the interpolated obstacle factor incorporates the obstacle information at all \( \tau \) in the factor graph and is utilized to meaningfully update the sparse set of support states.

An example factor graph that combines all of the factors described above is illustrated in Figure 5. Note that if there are enough support states to densely cover the trajectory, interpolated obstacle factors are not needed. But to fully utilize the power of the continuous-time trajectory representation and to maximize performance, the use of sparse support states along with interpolated obstacle factor is encouraged.

Given the factorized obstacle likelihood in Eq. (43)-(45), we can retrieve the vector-valued obstacle cost function of the trajectory defined in Eq. (38) by simply stacking all the vector-valued obstacle cost functions on all regular and interpolated states into a single vector

\[
\mathbf{h}(\theta) = [\mathbf{h}(\theta_0); \mathbf{h}^{intp}_{r_1}(\theta_0, \theta_1); \ldots; \mathbf{h}^{intp}_{r_{n_{ip}}}(\theta_0, \theta_1); \\
\mathbf{h}(\theta_1); \mathbf{h}^{intp}_{r_1}(\theta_1, \theta_2); \ldots; \mathbf{h}^{intp}_{r_{n_{ip}}}(\theta_1, \theta_2); \\
\ldots \\
\mathbf{h}(\theta_{N-1}); \mathbf{h}^{intp}_{r_1}(\theta_{N-1}, \theta_N); \ldots; \mathbf{h}^{intp}_{r_{n_{ip}}}(\theta_{N-1}, \theta_N); \\
\mathbf{h}(\theta_N)],
\]

where all \( \mathbf{h} \) are obstacle cost functions from regular obstacle factors defined in Eq. (44), and all \( \mathbf{h}^{intp} \) are obstacle cost functions from interpolated obstacle factors defined in Eq. (45). Since there are a total of \( N + 1 \) regular obstacle factors on support states, and \( n_{ip} \) interpolated factors between each support state pair, the total dimensionality of \( \mathbf{h}(\theta) \) is \( M \times (N + 1 + N \times n_{ip}) \). The hyperparameter matrix \( \Sigma_{obs} \) in Eq. (38) is then defined by

\[
\Sigma_{obs} = \begin{bmatrix}
\sigma_{obs} & \cdots \\
\vdots & \ddots \\
\end{bmatrix},
\]

which has size \( M \times (N + 1 + N \times n_{ip}) \) by \( M \times (N + 1 + N \times n_{ip}) \).

In our framework, the obstacle cost function \( \mathbf{h} \) can be any nonlinear function, and the construction of \( M \), and \( \sigma_{obs} \) are flexible as long as \( \left( \theta; \mathbf{e} \right) \) gives the collision-free likelihood. Effectively \( \mathbf{h}(\theta_i) \) should have a larger value when a robot collides with obstacles at \( \theta_i \), and a smaller value when the robot is collision-free. Our implementation of \( \mathbf{h} \), definition of \( M \), and guideline for the hyperparameter \( \sigma_{obs} \) is discussed in Section 8.2.2.

### 6.3 Computing the MAP trajectory

To solve the MAP inference problem in Eq. (35), we first illustrate the duality between inference and optimization by performing minimization on the negative log of the posterior distribution

\[
\theta^* = \arg\max_\theta p(\theta | (\theta; \mathbf{e})
\]

\[
= \arg\min_\theta \left\{ -\log p(\theta | (\theta; \mathbf{e})) \right\}
\]

\[
= \arg\min_\theta \left\{ \frac{1}{2} \left\| \theta - \mu \right\|_K + \frac{1}{2} \left\| \mathbf{h}(\theta) \right\|^2_{\Sigma_{obs}} \right\}
\]  

where Eq. (48) follows from Eq. (37) and Eq. (38). This duality connects the two different perspectives on motion planning problems such that the terms in Eq. (48) can be viewed as ‘cost’ to be minimized, or information to be maximized. The apparent construction of the posterior now becomes clear as we have a nonlinear least squares optimization 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 quadratic approximation of Eq. (48) until convergence.

Linearizing the nonlinear obstacle cost function around the current trajectory \( \theta \)

\[
\mathbf{h}(\theta + d\theta) \approx \mathbf{h}(\theta) + \mathbf{H} d\theta
\]

\[
\mathbf{H} = \frac{\partial \mathbf{h}}{\partial \theta}|_{\theta}
\]

where \( \mathbf{H} \) is the Jacobian matrix of \( \mathbf{h}(\theta) \), we convert Eq. (48) to a linear least squares problem

\[
\delta\theta^* = \arg\min_\delta\theta \left\{ \frac{1}{2} \left\| \theta - \mu \right\|_K^2 + \frac{1}{2} \left\| \mathbf{h}(\theta) + \mathbf{H}\delta\theta \right\|_{\Sigma_{obs}}^2 \right\}
\]  

Prepared using sagej.cls

Submitted to International Journal of Robotics Research (IJRR) as manuscript #IJR-17-2909.
The optimal perturbation $\delta \theta^*$ results from solving the following linear system

$$\left( \mathbf{K}^{-1} + \mathbf{H}^\top \mathbf{\Sigma}_{\text{obs}}^{-1} \mathbf{H} \right) \delta \theta^* = \mathbf{K}^{-1} (\theta - \mu) - \mathbf{H}^\top \mathbf{\Sigma}_{\text{obs}}^{-1} h(\theta) \tag{52}$$

Once the linear system is solved, the iteration

$$\theta \leftarrow \theta + \delta \theta^* \tag{53}$$

is applied until convergence criteria are met. Eq. (53) serves as the update rule for GPMP2.

If the linear system in Eq. (52) is sparse, then $\delta \theta^*$ can be solved efficiently by exploiting the sparse Cholesky decomposition followed by forward-backward passes (Golub and Van Loan 2012). Fortunately, this is the case: we have selected a Gaussian process prior with a block tridiagonal precision matrix $\mathbf{K}^{-1}$ (Section 4.2) and $\mathbf{H}^\top \mathbf{\Sigma}_{\text{obs}}^{-1} \mathbf{H}$ is also block tridiagonal (see proof in Appendix B). The structure exploiting iteration combined with the quadratic convergence rate of nonlinear least squares optimization method we employ (Gauss-Newton or Levenberg-Marquardt) makes GPMP2 more efficient and faster compared to GPMP.

7. **Incremental inference for fast replanning**

We have described how formulating motion planning problem as probabilistic inference on factor graphs results in fast planning through least squares optimization. In this section, we show that this perspective also gives us the flexibility to use other inference and optimization tools on factor graphs. In particular, we describe how factor graphs can be used to perform *incremental* updates to solve replanning problems efficiently.

The replanning problem can be defined as: given a solved motion planning problem, resolve the problem with partially changed conditions. Replanning problems are commonly encountered in the real world, when, for example: (i) the goal position for the end-effector has changed during middle of the execution; (ii) the robot receives updated estimation about its current state; or (iii) new information about the environment is available.

Since replanning is performed online, possibly in dynamic environments, fast real-time replanning is critical to ensuring safety.

A naïve way to solve this problem is to literally replan by re-optimizing from scratch. However, this is potentially too slow for real-time settings. Furthermore, if the majority of the problem is left unchanged, resolving the entire problem duplicates work and should be avoided to improve efficiency.

Here we adopt an incremental approach to updating the current solution given new or updated information. We use the Bayes Tree (Kaess et al. 2011a,b) data structure to perform incremental inference on factor graphs.* The Bayes Tree is similar to a clique tree but directed. Figure 6 shows a factor graph of a simple motion planning problem: given start and end states, and all factors, the goal is to solve the MAP inference problem to find the optimal trajectory. The Bayes Trees in Figure 6 are generated by their corresponding factor graphs with the elimination order being from the first to the 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 Kaess et al. (2011a,b).

Two replanning examples are shown in Figure 7. The first example shows replanning when the goal configuration changes causing an update to the prior factor on the goal state. When the Bayes Tree is updated with the new goal, only the root node of the tree is changed. The second example shows a replanning problem, given an observation of the current configuration (e.g. from perception during execution) that is added as a prior factor at $\theta_2$ where the

---

*Given that the trajectories are represented by GPs, the incremental updates of the factor graphs can also be viewed as incremental GP regression (Yan et al. 2017).
estimation was taken. When the Bayes Tree is updated, the parts of the tree that change correspond to the parts of the trajectory that get updated.

In our implementation, we use the iSAM2 incremental solver (Kaess et al. 2011b) within the GPMP2 framework to solve the replanning problem. We call this incremental variant of GPMP2, iGPMP2. A replanning scenario typically has the following steps. First, the original batch problem is solved with GPMP2. Then, we collect the additional information to form factors that need to be added or replaced within the factor graph. Finally, we run Algorithm 1 to update the Bayes Tree inside iSAM2, to get a newly updated optimal solution.

8. Implementation details

GPMP is implemented on top of the CHOMP (Zucker et al. 2013) code since it uses an identical framework, albeit with several augmentations. To implement GPMP2 and iGPMP2 algorithms, we used the GTSAM (Dellaert 2012) library. Our implementation is available as a single open source C++ library, gpmp2. We have also released a ROS interface as part of the PIPER (Mukadam 2017) package. In this section we describe the implementation details of our algorithms.

8.1 GPMP

8.1.1 GP prior: GPMP employs a constant-acceleration (i.e. jerk-minimizing) prior to generate a trajectory with a Markovian state comprising of configuration position, velocity and acceleration, by following the LTV-SDE in Eq. (5) with parameters

\[
\begin{align*}
A(t) &= \begin{bmatrix} 0 & I & 0 \\ 0 & 0 & I \\ 0 & 0 & 0 \end{bmatrix}, \quad u(t) = 0, \quad F(t) = \begin{bmatrix} 0 \\ 0 \\ I \end{bmatrix}
\end{align*}
\] (54)

and given \( \Delta t_i = t_{i+1} - t_i \),

\[
\Phi(t, s) = \begin{bmatrix} I & (t-s)I & \frac{1}{2}(t-s)^2I \\ 0 & I & (t-s)I \\ 0 & 0 & I \end{bmatrix}
\] (55)

\[
Q_{i,i+1} = \begin{bmatrix} \frac{1}{6} \Delta t_i^3 Q_C & \frac{1}{2} \Delta t_i^3 Q_C & \frac{1}{2} \Delta t_i^3 Q_C \\ \frac{1}{6} \Delta t_i^3 Q_C & \frac{1}{2} \Delta t_i^3 Q_C & \frac{1}{2} \Delta t_i^3 Q_C \\ \frac{1}{6} \Delta t_i^3 Q_C & \frac{1}{2} \Delta t_i^3 Q_C & \frac{1}{2} \Delta t_i^3 Q_C \end{bmatrix}
\] (56)

This prior is centered around a zero jerk trajectory and encourages smoothness by attempting to minimize jerk during optimization.

8.1.2 Obstacle avoidance and constraints: To quickly calculate the collision cost for an arbitrary shape of the robot’s physical body, GPMP represents the robot with a set of spheres, as in Zucker et al. (2013) (shown in Figure 8). This leads to a more tractable approximation to finding the signed distance from the robot surface to obstacles. GPMP uses the same obstacle cost function as CHOMP (see Eq. (20)) where the cost is summed over the sphere set on the robot body calculated using a precomputed signed distance field (SDF). Constraints are also handled in the same manner as CHOMP. Joint limits are enforced by smoothly projecting joint violations using the technique similar to projecting the obstacle gradient in Eq. (31). Along each point on the up-sampled trajectory the violations are calculated via \( L_1 \) projections to bring inside the limits (see Zucker et al. (2013) for details). Then they are collected into a violation trajectory, \( \theta_{up} \), to be projected:

\[
\theta = \theta + KM^{-1} \theta_{up}^\top.
\] (57)

8.2 GPMP2 and iGPMP2

GPMP2 uses the Levenberg-Marquardt algorithm to solve the nonlinear least squares optimization problem, with the

\[
\begin{align*}
A(t) &= \begin{bmatrix} 0 & I & 0 \\ 0 & 0 & I \\ 0 & 0 & 0 \end{bmatrix}, \quad u(t) = 0, \quad F(t) = \begin{bmatrix} 0 \\ 0 \\ I \end{bmatrix}
\end{align*}
\] (54)

\[\Phi(t, s) = \begin{bmatrix} I & (t-s)I & \frac{1}{2}(t-s)^2I \\ 0 & I & (t-s)I \\ 0 & 0 & I \end{bmatrix}\]

\[Q_{i,i+1} = \begin{bmatrix} \frac{1}{6} \Delta t_i^3 Q_C & \frac{1}{2} \Delta t_i^3 Q_C & \frac{1}{2} \Delta t_i^3 Q_C \\ \frac{1}{6} \Delta t_i^3 Q_C & \frac{1}{2} \Delta t_i^3 Q_C & \frac{1}{2} \Delta t_i^3 Q_C \\ \frac{1}{6} \Delta t_i^3 Q_C & \frac{1}{2} \Delta t_i^3 Q_C & \frac{1}{2} \Delta t_i^3 Q_C \end{bmatrix}\]

\[\theta = \theta + KM^{-1} \theta_{up}^\top.
\] (57)

\[\begin{align*}
A(t) &= \begin{bmatrix} 0 & I & 0 \\ 0 & 0 & I \\ 0 & 0 & 0 \end{bmatrix}, \quad u(t) = 0, \quad F(t) = \begin{bmatrix} 0 \\ 0 \\ I \end{bmatrix}
\end{align*}
\] (54)

\[\Phi(t, s) = \begin{bmatrix} I & (t-s)I & \frac{1}{2}(t-s)^2I \\ 0 & I & (t-s)I \\ 0 & 0 & I \end{bmatrix}\]

\[Q_{i,i+1} = \begin{bmatrix} \frac{1}{6} \Delta t_i^3 Q_C & \frac{1}{2} \Delta t_i^3 Q_C & \frac{1}{2} \Delta t_i^3 Q_C \\ \frac{1}{6} \Delta t_i^3 Q_C & \frac{1}{2} \Delta t_i^3 Q_C & \frac{1}{2} \Delta t_i^3 Q_C \\ \frac{1}{6} \Delta t_i^3 Q_C & \frac{1}{2} \Delta t_i^3 Q_C & \frac{1}{2} \Delta t_i^3 Q_C \end{bmatrix}\]

\[\theta = \theta + KM^{-1} \theta_{up}^\top.
\] (57)

\[\begin{align*}
A(t) &= \begin{bmatrix} 0 & I & 0 \\ 0 & 0 & I \\ 0 & 0 & 0 \end{bmatrix}, \quad u(t) = 0, \quad F(t) = \begin{bmatrix} 0 \\ 0 \\ I \end{bmatrix}
\end{align*}
\] (54)

\[\Phi(t, s) = \begin{bmatrix} I & (t-s)I & \frac{1}{2}(t-s)^2I \\ 0 & I & (t-s)I \\ 0 & 0 & I \end{bmatrix}\]

\[Q_{i,i+1} = \begin{bmatrix} \frac{1}{6} \Delta t_i^3 Q_C & \frac{1}{2} \Delta t_i^3 Q_C & \frac{1}{2} \Delta t_i^3 Q_C \\ \frac{1}{6} \Delta t_i^3 Q_C & \frac{1}{2} \Delta t_i^3 Q_C & \frac{1}{2} \Delta t_i^3 Q_C \\ \frac{1}{6} \Delta t_i^3 Q_C & \frac{1}{2} \Delta t_i^3 Q_C & \frac{1}{2} \Delta t_i^3 Q_C \end{bmatrix}\]

\[\theta = \theta + KM^{-1} \theta_{up}^\top.
\] (57)
initial damping parameter set as 0.01. The optimization is stopped if a maximum of 100 iterations is reached, or if the relative decrease in error is smaller than $10^{-4}$. iGPMP2 uses the iSAM2 (Kaess et al. 2011b) incremental optimizer with default settings.

8.2.1 GP prior: We use a constant-velocity prior in GPMP2 with the Markovian state comprising of configuration position and velocity. Note that, unlike GPMP, we did not include acceleration since it was not needed for any gradients and an acceleration-minimizing prior for optimization was sufficient for the tasks we consider in this work. Ideally a jerk-minimizing trajectory would be beneficial to use on faster moving systems like quadrotors. GPMP2 scales only cubically in computation with the size of the state. So even if the same prior as GPMP was used, GPMP2 would still be faster given its quadratic convergence rate.

The trajectory is similarly generated by following the LTV-SDE in Eq. (5) with

$$A(t) = \begin{bmatrix} 0 & I \\ 0 & 0 \end{bmatrix}, \quad u(t) = 0, \quad F(t) = \begin{bmatrix} 0 \\ I \end{bmatrix}$$

(58)

and given $\Delta t_i = t_{i+1} - t_i$,

$$\Phi(t, s) = \begin{bmatrix} 1 & (t - s)I \\ 0 & I \end{bmatrix}, \quad Q_{i,i+1} = \begin{bmatrix} \frac{1}{2} \Delta t_i^2 \sigma C & \frac{1}{2} \Delta t_i \sigma C \\ \frac{1}{2} \Delta t_i \sigma C & \frac{1}{2} \Delta t_i^2 \sigma C \end{bmatrix}$$

(59)

Analogously this prior is centered around a zero-acceleration trajectory.

8.2.2 Collision-free likelihood: Similar to GPMP and CHOMP, the robot body is represented by a set of spheres as shown in Figure 8, and the obstacle cost function for any configuration $\theta_i$ is then completed by computing the hinge loss for each sphere $S_j$ ($j = 1, \ldots, M$) and collecting them into a single vector,

$$h(\theta_i) = \left[ c(\mathbf{d}(\mathbf{x}(\theta_i, S_j))) \right]_{1 \leq j \leq M}$$

(60)

where $\mathbf{x}$ is the forward kinematics, $\mathbf{d}$ is the signed distance function, $c$ is the hinge loss function, and $M$ is the number of spheres that represent the robot model.

Forward kinematics $\mathbf{x}(\theta_i, S_j)$ maps any configuration $\theta_i$ to the 3D workspace, to find the center position of any sphere $S_j$. Given a sphere and its center position, we calculate $\mathbf{d}(x)$, the signed distance from the sphere at $x$ to the closest obstacle surface in the workspace. The sphere shape makes the surface-to-surface distance easy to calculate, since it is equal to the distance from sphere center to closest obstacle surface minus the sphere radius. Using a precomputed signed distance field (SDF), stored in a voxel grid with a desired resolution, the signed distance of any position in 3D space is queried by trilinear interpolation on the voxel grid. The hinge loss function\(^1\) is defined as

$$c(d) = \begin{cases} -d + \epsilon & \text{if } d \leq \epsilon \\ 0 & \text{if } d > \epsilon \end{cases}$$

(61)

where $d$ is the signed distance, and $\epsilon$ is a ‘safety distance’ indicating the boundary of the ‘danger area’ near obstacle surfaces. By adding a non-zero obstacle cost, even if the robot is not in collision but rather too close to the obstacles, $\epsilon$ enables the robot to stay a minimum distance away from obstacles. The remaining parameter $\sigma_{obs}$ needed to fully implement the likelihood in Eq. (44) and Eq. (45) is defined by an isotropic diagonal matrix

$$\Sigma_{obs} = \sigma_{obs}^2 I,$$

(62)

where $\sigma_{obs}$ is the ‘obstacle cost weight’ parameter.

Figure 9 visualizes a 2D example of the collision-free likelihood defined by the obstacle cost function in Eq. (60). 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 marginals’ defined by $\epsilon$.

Note that the obstacle cost function used here is different from the one used in GPMP and CHOMP, where $c$ is instead a smooth function (necessary for gradient calculation) and is multiplied with the norm of the workspace velocity (see Eq. (20)). This arc-length parameterization helps in making the trajectory avoid obstacles rather than speeding through them, while minimizing cost. The GP prior we use for GPMP2 helps us achieve the same purpose, by incorporating cost on large accelerations. The choice of cost function in Eq. (61) serves as a good approximation for the tasks we consider and is also less computationally expensive.

\(^1\)The hinge loss function is not differentiable at $d = \epsilon$, so in our implementation we set $dc(d)/dd = -0.5$ when $d = \epsilon$.  

Figure 9. The likelihood function $h$ in a 2D space with two obstacles and $\epsilon = 0.1m$. Obstacles are marked by black lines and darker area has higher likelihood for no-collision.
against trajectory optimizations algorithms -

We benchmarked our algorithms, GPMP

9.1.1 Setup: We benchmarked our algorithms, GPMP

9.1.2 Parameters: For both GPMP and GPMP2, Qc

with interpolation (GPMP2-intp) during optimization and without interpolation (GPMP2-

9.2.3 Motion constraints: Motion constraints exist in

A video of experiments is available at https://youtu.be/
mVA8qhGf7So.
Table 1.A Results for 24 planning problems on the 7-DOF WAM arm.

|                  | GPMP-2-intp | GPMP-2-no-intp | TrajOpt-101 | TrajOpt-11 | GPMP | CHOMP | RRT-Connect | LBKPIECE |
|------------------|------------|----------------|-------------|-------------|------|-------|-------------|----------|
| Success (%)      | 91.7       | 100.0          | 91.7        | 20.8        | 95.8 | 75    | 91.7        | 62.5     |
| Avg. Time (s)    | 0.121      | 0.384          | 0.313       | **0.027**   | 0.3  | 0.695 | 1.87        | 6.89     |
| Max Time (s)     | 0.367      | 0.587          | 0.443       | **0.033**   | 0.554| 2.868 | 5.18        | 9.97     |

Table 1.B Results for 198 planning problems on PR2's 7-DOF right arm.

|                  | GPMP2-intp | GPMP2-no-intp | TrajOpt-61 | TrajOpt-11 | GPMP | CHOMP | RRT-Connect | LBKPIECE |
|------------------|------------|----------------|-------------|-------------|------|-------|-------------|----------|
| Success (%)      | 79.3       | 78.8           | 68.7        | 77.8        | 36.9 | 59.1  | **82.3**    | 33.8     |
| Avg. Time (s)    | **0.11**   | 0.196          | 0.958       | 0.191       | 1.7  | 2.38  | 3.02        | 7.12     |
| Max Time (s)     | **0.476**  | 0.581          | 4.39        | 0.803       | 9.08 | 9.81  | 9.33        | 9.95     |

**Figure 11.** Left subfigure shows successful trajectory with a good selection of \( \sigma_{obs} \); right subfigure shows failure when \( \sigma_{obs} \) is too large.

Finding a good selection of \( \sigma_{obs} \) and \( \epsilon \) is preferable in problems with more difficult navigation constraints. However, a very high value might result in noisy trajectories since the weight on the smoothness cost becomes relatively low. A reverse effect will be seen with a smaller value. This parameter can be set based on the problem and the prior model used (for example, constant velocity or constant acceleration). In our benchmarks for GPMP we set \( Q_C = 100 \) for the WAM dataset and \( Q_C = 50 \) for the PR2 dataset and for GPMP2 we set \( Q_C = 1 \) for both datasets.

Another common parameter, ‘safety distance,’ \( \epsilon \) is selected to be about double the minimum distance to any obstacle allowed in the scene and should be adjusted based on the robot, environment, and the obstacle cost function used. In our benchmarks we set \( \epsilon = 0.2m \) for both GPMP and GPMP2 for the WAM dataset, and \( \epsilon = 0.05m \) for GPMP and \( \epsilon = 0.08m \) for GPMP2 for the PR2 dataset.

For GPMP2 the ‘obstacle cost weight’ \( \sigma_{obs} \) acts like a weight term that balances smoothness and collision-free requirements on the optimized trajectory and is set based on the application. Smaller \( \sigma_{obs} \) puts more weight on obstacle avoidance and vice versa. Figure 11 shows an example of an optimized trajectory for PR2 with different settings of \( \sigma_{obs} \). In our experiments we found that the range \([0.001, 0.02]\) works well for \( \sigma_{obs} \) and larger robot arms should use larger \( \sigma_{obs} \). In the benchmarks we set \( \sigma_{obs} = 0.02m \) for the WAM dataset and \( \sigma_{obs} = 0.005 \) for the PR2 dataset.

9.1.3 Analysis: The benchmark results for the WAM dataset are summarized in Table 1.A and for the PR2 dataset are summarized in Table 1.B. Average time and maximum time include only successful runs.

Evaluating motion planning algorithms is a challenging 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 1.A and 1.B we see that GPMP2 perform consistently well compared to other algorithms on these datasets. Using interpolation during optimization (GPMP2-intp) achieves \( 30 - 50\% \) speedup of average and maximum runtime when compared to not using interpolation (GPMP2-no-intp). On the WAM dataset TrajOpt-11 has the lowest runtime but is able to solve only \( 20\% \) of the problems, while GPMP2-intp has the second lowest runtime with a much higher success rate. GPMP2-no-intp has the highest success rate. On the relatively harder PR2 dataset, GPMP2-intp has the lowest runtime and is twice as fast with a slightly higher success rate compared to other algorithms.
Table 2. Average number of optimization iterations on successful runs.

|          | CHOMP | GPMP-no-intp | GPMP-intp | GPMP2-no-intp | GPMP2-intp |
|----------|-------|--------------|-----------|---------------|------------|
| WAM      | 26.4  | 11.5         | 12.0      | 23.6          | 13.0       |
| PR2      | 46.6  | 32.2         | 19.1      | 26.4          | 24.4       |

Figure 12. Breakdown of average timing per task per iteration on all problems in the WAM dataset is shown for CHOMP, GPMP-no-intp, GPMP-intp, GPMP2-no-intp and GPMP2-intp.

to TrajOpt-11. GPMP2-intp has the second highest success rate and is slightly behind RRT-Connect but is 30 times faster. The timing for RRT-Connect would further increase if a post processing or smoothing step was applied.

As seen from the max run times, GPMP2 always converges well before the maximum time limit and all the failure cases are due to infeasible local minima. Solutions like, random restarts (that are commonly employed) or GPMP-GRAPH (Huang et al. 2017), an extension to our approach that uses graph-based trajectories, can help contend with this issue.

To understand how the GP representation and the inference framework result in performance boost we compare timing breakdowns during any iteration for CHOMP, GPMP and GPMP2. Figure 12 shows the breakdown of average timing per task per iteration on the WAM dataset where the solution update portion (dark blue) incorporates the optimization costs. Table 2 shows average number of optimization iterations for successful runs in both the WAM and the PR2 datasets. We see that compared to CHOMP, GPMP is more expensive per iteration primarily from the computation of the Hessian, that is needed to find the acceleration in workspace (CHOMP approximates the acceleration with finite differencing). However, due to the GP representation and gradients on the augmented trajectory, GPMP is able to take larger update steps and hence converge faster with fewer iterations. GPMP2 on the other hand takes advantage of quadratic convergence while also benefiting from the GP representation and the inference framework. GP interpolation further reduces the runtime per iteration, especially for GPMP2. The dashed bars in Figure 12 represent computational costs due collision checking during optimization at a finer resolution, on top of the computational cost incurred to evaluate gradient information. This was necessary to determine convergence, since the CHOMP solution can jump in and out of feasibility between iterations (Zucker et al. 2013). GPMP also incurs this cost since it too exhibits this behavior due to its similar construction. Note that the total computational time in Table 1.A reflects the total iteration time as shown in Figure 12 plus time before and after the iterations including setup and communication time.

9.2 Incremental planning benchmark

We evaluate our incremental motion planner iGPMP2 by benchmarking it against GPMP2 on replanning problems with the WAM and PR2 datasets.

For each problem in this benchmark, we have a planned trajectory from a start configuration to an originally assigned goal configuration. Then, at the middle time-step of the trajectory a new goal configuration is assigned. The replanning problem entails finding a trajectory to the newly assigned goal. 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 new location in configuration, and a fixed state factor at the middle time step to enforce constraint of current state.

A total of 72 and 54 replanning problems are prepared for the WAM and the PR2 datasets, respectively. GP interpolation is used and all parameters are the same as the batch benchmarks. The benchmark results are shown in Table 3.A and Table 3.B. We see from the results that iGPMP2 provides an order of magnitude speed-up, suffering a small loss in the success rate compared to GPMP2.

GPMP2 reinitializes the trajectory as a constant-velocity straight line from the middle state to the new goal and replans from scratch. However, iGPMP2 can use the solution to the old goal and the updated Bayes Tree as
that for the remaining 27 problems where \( \| \theta_o - \theta_r \|_2 \geq 2.0 \), iGPMP2 only has 51.9% success rate.

Examples of successfully replanned trajectories generated using iGPMP2 are shown in Figure 13. The use of the fixed state factor at the middle time step helps make a smooth transition between original trajectories and replanned trajectories, which is critical if the trajectory is being executed on a real robot.

10. Discussion

10.1 Comparisons with related work

GPMP can be viewed as a generalization on CHOMP where the trajectory is a sample from a GP and is augmented with velocities and accelerations. Both GPMP and GPMP2 use the GP representation for a continuous-time trajectory, GP interpolation, and signed distance fields for collision checking. However, with GPMP2 we fully embrace the probabilistic view of motion planning. In contrast to similar views on motion planning (Toussaint 2009; Toussaint and Storkey 2006) that use message passing, we instead solve the inference problem as nonlinear least squares. This allows us to use solvers with quadratic convergence rates that exploit the sparse structure of our problem, leading to a much faster algorithm compared to GPMP (and CHOMP) that only has linear convergence and is encumbered by the slow gradient computation. The update step in GPMP2 involves only linearization and the Cholesky decomposition to solve the linear system.

TrajOpt (Schulman et al. 2013, 2014) 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, as a consequence, slower to solve. Benchmark results in Section 9.1 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 with only 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.

Unlike sampling based methods, our algorithms do not guarantee probabilistic completeness. However, from the benchmarks we see that GPMP2 is efficient at finding locally optimal trajectories that are feasible from naïve
straight line initialization that may be in collision. We note
that trajectory optimization is prone to local minima and
this strategy may not work on harder planning problems like
mazes where sampling based methods excel. Recent work
however, has begun to push the boundaries in trajectory
optimization based planning. GPMP-GRAPH (Huang et al.
2017), an extension of our work, employs graph-based
trajectories to explore exponential number of initializations
simultaneously rather than trying them one at a time.
Results show that it can quickly find feasible solutions
even in mazes. Depending on the problem and time
budget, multiple random initializations can also be a viable
approach (since GPMP2 is fast), or GPMP2 can also be
used on top of a path returned from a sampling based
method to generate a time parameterized trajectory that is
smooth.

Finally, our framework allows us to solve replanning
problems very quickly, something that none of the above
trajectory optimization approaches can provide. We are
able to achieve this through incremental inference on a
factor graph. On simpler replanning problems like changing
goals, multi-query planners like PRM (Kavraki et al. 1996)
can be useful but are time consuming since a large initial
exploration of the space is necessary to build the first graph,
a majority of which may not be needed. Solving these
types of problems fast is very useful in real-time real-world
applications.

10.2 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. Random restarts is a commonly used
method to combat this, however our approach allows for
a more principled way (Huang et al. 2017) in which this
problem can be tackled.

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 (Schulman et al. 2013, 2014). We believe that SQP
can be integrated into our trajectory optimizer, although this
remains future work.

11. Conclusion
We use Gaussian processes to reason about continuous-
time trajectories in the context of motion planning as
trajectory 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
support states. The up-sampled trajectory is used during
optimization to propagate the cost information back to
the support states such that only they are updated. By
formulating motion planning as probabilistic inference on
factor graphs we also perform fast structure exploiting
nonlinear least square optimization.

We benchmark our algorithms against several state-of-the-art trajectory optimization and sampling based
algorithms on 7-DOF arm planning problems on two
datasets in multiple environments and show that our
approach, GPMP2 is consistently faster, often several times
faster, than its nearest competitors.

Finally, by performing incremental inference on factor
graphs we solve replanning problems with iGPMP2
incrementally in an order of magnitude faster than resolving
from scratch with GPMP2. This property is unique to our
motion planning algorithm and highly useful for planning
in real-time real-world applications.

Acknowledgements
This work was partially supported by National Institute of Food
and Agriculture, U.S. Department of Agriculture, under award
number 2014-67021-22556, NSF NRI award number 1637758,
and NSF CRII award number 1464219.

References
Anderson S and Barfoot TD (2013) Towards relative continuous-
time SLAM. In: IEEE Intl. Conf. on Robotics and Automation
(ICRA). IEEE.
Anderson S, Barfoot TD, Tong CH and Särkkä S (2015) Batch
nonlinear continuous-time trajectory estimation as exactly
sparse gaussian process regression. Autonomous Robots
39(3): 221–238.
Anderson S, Dellaert F and Barfoot T (2014) A hierarchical
wavelet decomposition for continuous-time SLAM. In: IEEE
Intl. Conf. on Robotics and Automation (ICRA).
Attias H (2003) Planning by probabilistic inference. In: AISTATS.
Barfoot T, Tong CH and Särkkä S (2014) Batch continuous-
time trajectory estimation as exactly sparse Gaussian process
regression. Proceedings of Robotics: Science and Systems, Berkeley, USA.
Bibby C and Reid I (2010) A hybrid SLAM representation for
dynamic marine environments. In: IEEE Intl. Conf. on
Robotics and Automation (ICRA). IEEE, pp. 257–264.
Bosse M and Zlot R (2009) Continuous 3D scan-matching with
a spinning 2D laser. In: IEEE Intl. Conf. on Robotics and
Automation (ICRA). IEEE, pp. 4312–4319.
Byravan A, Boots B, Srinivasa SS and Fox D (2014) Space-time
functional gradient optimization for motion planning. In:
Marinho Z, Dragan A, Byravan A, Boots B, Gordon GJ and Srinivasa S (2016) Functional gradient motion planning in reproducing kernel Hilbert spaces. In: *Proceedings of Robotics: Science and Systems (RSS)*.

Mukadam M (2017) PIPER. [Online] Available at https://github.com/gtrll/piper.

Mukadam M, Cheng CA, Yan X and Boots B (2017a) Approximately optimal continuous-time motion planning and control via probabilistic inference. In: *Proceedings of the 2017 IEEE Conference on Robotics and Automation (ICRA)*.

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

Mukadam M, Yan X and Boots B (2016) Gaussian process motion planning. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. pp. 9–15.

Nguyen-Tuong D, Peters J, Seeger M and Schölkopf B (2008) Learning inverse dynamics: a comparison. In: *European Symposium on Artificial Neural Networks, EPFL-CONF-175477*.

Park C, Pan J and Manoche D (2012) ITOMP: Incremental trajectory optimization for real-time replanning in dynamic environments. In: *ICAPS*.

Park C, Pan J and Manoche D (2013) Real-time optimization-based planning in dynamic environments using GPUs. In: *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, pp. 4090–4097.

Patron-Perez A, Lovegrove S and Sibley G (2015) A spline-based trajectory representation for sensor fusion and rolling shutter cameras. *International Journal of Computer Vision* 113(3): 208–219.

Quinlan S (1994) Real-time modification of collision-free paths. PhD Thesis, Stanford University.

Rana MA, Mukadam M, Ahmadzadeh SR, Chernova S and Boots B (2017) Skill generalization via inference-based planning. In: *RSS Workshop on Mathematical Models, Algorithms, and Human-Robot Interaction*.

Rasmussen CE (2006) Gaussian processes for machine learning. Citeseer.

Ratliff N, Zucker M, Bagnell JA and Srinivasa S (2009) CHOMP: Gradient optimization techniques for efficient motion planning. In: *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. IEEE, pp. 489–494.

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

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

Schulman J, Duan Y, Ho J, Lee A, Awwal I, Bradlow H, Pan J, Patil S, Goldberg K and Abbeel P (2014) Motion planning with sequential convex optimization and convex collision checking. *The International Journal of Robotics Research* 33(9): 1251–1270.

Schulman J, Ho J, Lee A, Awwal I, Bradlow H and Abbeel P (2013) Finding locally optimal, collision-free trajectories with sequential convex optimization. In: *Robotics: Science and Systems*, volume 9. Citeseer, pp. 1–10.

Sturm J, Plagemann C and Burgard W (2009) Body schema learning for robotic manipulators from visual self-perception. *Journal of Physiology-Paris* 103(3): 220–231.

Şucan IA and Kavraki LE (2009) Kinodynamic motion planning by interior-exterior cell exploration. In: *Algorithmic Foundation of Robotics VIII*. Springer, pp. 449–464.

Şucan IA, Moll M and Kavraki LE (2012) The open motion planning library. *IEEE Robotics & Automation Magazine* 19(4): 72–82.

Tay MKC and Laugier C (2008) Modelling smooth paths using Gaussian processes. In: *Field and Service Robotics*. Springer, pp. 381–390.

Theodorou E, Tassa Y and Todorov E (2010) Stochastic differential dynamic programming. In: *American Control Conference (ACC), 2010*. IEEE, pp. 1125–1132.

Tong CH, Furgale P and Barfoot TD (2012) Gaussian process Gauss-Newton: Non-parametric state estimation. In: *Computer and Robot Vision (CRV), 2012 Ninth Conference on*. IEEE, pp. 206–213.

Toussaint M (2009) Robot trajectory optimization using approximate inference. In: *Proceedings of the 26th annual international conference on machine learning*. ACM, pp. 1049–1056.

Toussaint M and Goerick C (2010) A Bayesian view on motor control and planning. In: *From Motor Learning to Interaction Learning in Robots*. Springer, pp. 227–252.

Toussaint M and Storkey A (2006) Probabilistic inference for solving discrete and continuous state Markov decision processes. In: *Proceedings of the 23rd international conference on Machine learning*. ACM, pp. 945–952.

Vijayakumar S, D’souza A and Schaal S (2005) Incremental online learning in high dimensions. *Neural computation* 17(12): 2602–2634.

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

Zucker M, Ratliff N, Dragan AD, Pivtoraiko M, Klingensmith M, Dellin CM, Bagnell JA and Srinivasa SS (2013) CHOMP: Covariant Hamiltonian optimization for motion planning. *The International Journal of Robotics Research* 32(9-10): 1164–1193.
Appendix A: The trajectory prior

First, we review conditioning a distribution of state $\theta$ on observations $Y$ in general (for a full treatment see (Rasmussen 2006)). Let the observation be given by the following linear equation

$$ Y = C\theta + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \bar{K}_y). $$

(64)

We can write their joint distribution as

$$ \mathcal{N}\left( \begin{bmatrix} \bar{\mu} \\ C\bar{\mu} \end{bmatrix}, \begin{bmatrix} \bar{K} & \bar{K}C^T \\ C\bar{K} & C\bar{K}C^T + \bar{K}_y \end{bmatrix} \right). $$

(65)

The distribution of the state conditioned on the observations is then $\mathcal{N}(\mu, K)$ where

$$ \mu = \bar{\mu} + \bar{K}C^T(\bar{K}C^T + \bar{K}_y)^{-1}(Y - C\bar{\mu}) $$

(66)

$$ K = \bar{K} - \bar{K}C^T(\bar{K}C^T + \bar{K}_y)^{-1}\bar{K} $$

(67)

Now, we are interested in conditioning just on the goal state $\theta_N$ with mean $\mu_N$ and covariance $K_N$. Therefore in the above equations we use $C = [0 \ldots 0 \ I]$ and $\bar{K}_y = K_N$ to get

$$ \mu = \bar{\mu} + \bar{K}\theta_N^T(\bar{K}\theta_N + \bar{K}_y)^{-1}(\theta_N - \mu_N) $$

(68)

$$ K = \bar{K} - \bar{K}\theta_N^T(\bar{K}\theta_N + \bar{K}_y)^{-1}\bar{K}\theta_N $$

(69)

where $\bar{K}\theta_N = [\bar{K}(t_N, t_0) \ldots \bar{K}(t_N, t_n)]$. Using the Woodbury matrix identity we can write Eq. (67) as

$$ K = (\bar{K}^{-1} + C^T\bar{K}_y^{-1}C)^{-1} $$

(70)

and substituting $C$ and $\bar{K}_y$ as before for conditioning on the goal we get

$$ K = \left( \bar{K}^{-1} + [0 \ldots 0 \ I]^T\bar{K}_N^{-1}[0 \ldots 0 \ I] \right)^{-1}. $$

(71)

From (Barfoot et al. 2014) we know that the precision matrix of the distribution obtained from the LTV-SDE in Eq. (5) can be decomposed as $K^{-1} = A^{-1}Q^{-1}A^{-1}$. Therefore,

$$ K^{-1} = \begin{bmatrix} A^{-1} & 0 & \cdots & 0 & I \\ 0 & Q & \cdots & 0 & I \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & A^{-1} & I \\ 0 & 0 & \cdots & 0 & I \end{bmatrix} $$

(72)

$$ B = \begin{bmatrix} I & 0 & \cdots & 0 & 0 \\ \Phi(t_1, t_0) & I & \cdots & 0 & 0 \\ 0 & \Phi(t_2, t_1) & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & I & 0 \\ 0 & 0 & \cdots & -\Phi(t_N, t_{N-1}) & I \\ 0 & 0 & \cdots & 0 & 1 \end{bmatrix} $$

(73)

where

$$ Q^{-1} = \text{diag}(K_{0,0}^{-1}, Q_{0,1}^{-1}, \ldots, Q_{N-1, N}^{-1}), $$

(75)

$$ Q_{a,b} = \int_{t_a}^{t_b} \Phi(b, s)F(s)Q(s)\Phi^T(b, s)\, ds $$

(76)

Appendix B: Sparsity of the likelihood in GPMP2

In Eq. (52) we argue that matrix $K^{-1} + H^T\Sigma_{\text{obs}}^{-1}H$ is sparse. In Section 4.2, we proved the block-tridiagonal property of $K^{-1}$. In this section we prove that $H^T\Sigma_{\text{obs}}^{-1}H$ is also block-tridiagonal.

Given the isotropic definition of $\Sigma_{\text{obs}}$ in Eq. (47) and Eq. (62)

$$ H^T\Sigma_{\text{obs}}^{-1}H = \sigma_{\text{obs}}^{-2}H^TH. $$

(77)

Given the definition of $h(\theta)$ in Eq. (46), the size of $H$ is $M \times (N + 1 + N \times n_ip)$ by $(N + 1) \times D$, therefore $H^TH$ has size $(N + 1) \times D$. For simplicity, we partition $H$ and $H^TH$ by forming blocks corresponding to the system DOF $D$, and dimensionality $M$ of $h$, and work with these block matrices in the remaining section. So $H$ and $H^TH$ have blockwise size $N + 1 + N \times n_ip$ by $N + 1$ and $N + 1$ by $N + 1$ respectively. We define $A(i, j)$ to be the block element at row $i$ and column $j$ of $A$.

Given the definition of $h(\theta)$ in Eq. (46), each element of $H$ is defined by

$$ H(i, j) = \frac{\partial h(\theta_s)}{\partial \theta_j} $$

(78)

for rows contain regular obstacle factors, where $s_i$ is the support state index connects the regular obstacle factor of row $i$, or

$$ H(i, j) = \frac{\partial h^{\text{intp}}(\theta_{s_i}, \theta_{s_{i+1}})}{\partial \theta_j} $$

(79)

for rows contain interpolated obstacle factors, where $s_i$ is the before support state index of interpolated obstacle factor of row $i$. Since $h(\theta_{s_i})$ is only a function of $\theta_{s_i}$, and $h^{\text{intp}}(\theta_{s_i}, \theta_{s_{i+1}})$ is only function of $\theta_{s_i}$ and $\theta_{s_{i+1}}$, they have zero partial derivatives with respect to any other states in $\theta$, so for any block element in $H$

$$ H(i, j) = 0, \text{ if } j \neq s_i \text{ or } s_i + 1. $$

(80)

For each block element in $H^TH$

$$ H^TH(i, j) = \sum_{k=1}^{N+1+N \times n_ip} H^T(i, k)H(k, j) $$

(81)

$$ = \sum_{k=1}^{N+1+N \times n_ip} H(k, i)^T H(k, j). $$

(82)
For each $k$, non-zero $H(k, i)^\top H(k, j)$ is possible when the following condition is satisfied,

$$\{i = s_k \text{ or } s_k + 1\} \text{ and } \{j = s_k \text{ or } s_k + 1\}. \quad (83)$$

So for non-zero $H^\top H(i, j)$

$$|i - j| \leq 1, \quad (84)$$

since if $i$ and $j$ has difference larger than 1, Eq. (83) is unsatisfied on every $k$, so $H^\top H(i, j)$ will be zero based on Eq. (82). Given we know that $H^\top H$ is block tridiagonal, and Eq. (77), we have proved that $H^\top \Sigma^{-1}_{obs} H$ is also block tridiagonal.