

# Continuous-time Gaussian process motion planning via probabilistic inference

The International Journal of  
Robotics Research  
2018, Vol. 37(11) 1319–1340  
© The Author(s) 2018  
Article reuse guidelines:  
sagepub.com/journals-permissions  
DOI: 10.1177/0278364918790369  
journals.sagepub.com/home/ijr  


Mustafa Mukadam<sup>\*</sup>, Jing Dong<sup>\*</sup>, Xinyan Yan<sup>ID</sup>, 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 GP 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 (Byravan et al., 2014; He et al., 2013; Kalakrishnan et al., 2011; Marinho et al., 2016; Ratliff et al., 2009; Zucker et al., 2013) 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. In addition, trajectory optimization is locally optimal, and

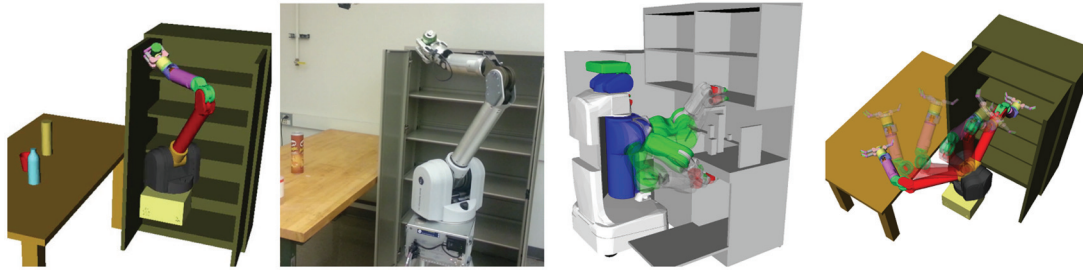
---

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

<sup>\*</sup>These authors contributed equally.

## Corresponding author:

Mustafa Mukadam, Institute for Robotics and Intelligent Machines, Georgia Institute of Technology, Atlanta, GA 30332, USA.  
Email: mmukadam3@gatech.edu



**Fig. 1.** Optimized trajectory found by GPMP2 is used to place a soda can on a shelf in simulation (left) and with a real WAM arm (middle left). Examples of successful trajectories generated by GPMP2 are shown in the countertop (middle right) and lab (right) environments with the PR2 and WAM robots, respectively.

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., 2014, 2013) 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, owing 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 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 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., 2011b, 2008) can be adapted to efficiently solve replanning problems.

In this paper, we provide a revised and extended version of our previous work (Dong et al., 2016; Mukadam et al., 2016), give more theoretical insight, and provide a proof for the sparsity of the linear system in GPMP2. We also conduct additional benchmarks on larger datasets and compare GPMP and GPMP2 against leading trajectory optimization-based motion planning algorithms (Schulman et al., 2014; Zucker et al., 2013) as well as sampling-based motion planning algorithms (Kuffner and LaValle, 2000; Şucan and Kavraki, 2009; Sucan et al., 2012) in multiple 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 algorithms 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 owing to computational challenges. Often computation is wasted exploring regions that may not lead to a solution. Recent work in informed techniques (Gammell et al., 2015) combatted this problem by biasing the sampling approach to make search more tractable.

In contrast to 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 (Byravan et al., 2014; He et al., 2013; Marinho et al., 2016; Ratliff et al., 2009; Zucker et al., 2013) 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., 2014, 2013) solves a sequential quadratic program and performs convex continuous-time collision checking. In contrast to sampling-based planners, trajectory optimization methods are very fast, but only find 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; Dong and Barfoot, 2014; Li et al., 2013), splines (Anderson and Barfoot, 2013; Bibby and Reid, 2010; Furgale et al., 2013, 2015; Leutenegger et al., 2015; Patron-Perez et al., 2015), and hierarchical wavelets (Anderson et al., 2014) have been used to represent trajectories in filtering and state estimation. B-Splines (Elbanhawi et al., 2015) have similarly been used to represent trajectories in motion planning problems. Compared with parametric representations such as splines and wavelets, GPs provide a natural notion of uncertainty on top of allowing a sparse parameterization of the continuous-time trajectory. A critical distinction in motion planning problems is that even with a sparse parameterization, the collision cost has to be evaluated at a finer resolution. Therefore, if the interpolation procedure for a chosen continuous-time representation is computationally expensive, the resulting speedup obtained from a sparse representation is negligible and may result in an overall slower algorithm. Recent work by Marinho et al. (2016) works to optimize trajectories in reproducing kernel Hilbert space (RKHS) with radial basis function (RBF) kernels, but

ignores the cost between sparse waypoints. Even without interpolation, these dense kernels result in relatively computationally expensive updates. In this work, we use structured GPs that allow us to exploit the underlying sparsity in the problem to perform efficient inference. We are able to use fast GPR to interpolate the trajectory and evaluate obstacle cost on a finer resolution, while the trajectory can be parameterized by a small number of support states. We also show in this work that the probabilistic representation naturally allows us to represent the motion planning problem with a factor graph and the GP directly corresponds to the system dynamics or motion model, thus giving it a physical meaning.

GPs have been used for function approximation in supervised learning (Kersting et al., 2007; Vijayakumar et al., 2005), inverse dynamics modeling (Nguyen-Tuong et al., 2008; Sturm et al., 2009), reinforcement learning (Deisenroth and Rasmussen, 2011), path prediction (Tay and Laugier, 2008), SLAM (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 (Levine and Koltun, 2013; Toussaint and Storkey, 2006), and Kullback–Leibler (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., 2011b, 2008), a fact we take advantage of to solve replanning problems.

Replanning involves adapting a previously solved solution to changing conditions. Early replanning work such as  $D^*$  (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) such as 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. Graphical processing units (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,  $\theta(t) : t \rightarrow \mathbb{R}^D$ , where  $D$  is dimensionality of the state, that satisfy constraints and minimize cost (Kalakrishnan et al., 2011; Schulman et al., 2014; Zucker et al., 2013). Motion planning can therefore be formalized as

$$\begin{aligned} & \text{minimize} && \mathcal{F}[\theta(t)] \\ & \text{subject to} && \mathcal{G}_i[\theta(t)] \leq 0, \quad i = 1, \dots, m_{\text{ineq}} \\ & && \mathcal{H}_i[\theta(t)] = 0, \quad i = 1, \dots, m_{\text{eq}} \end{aligned} \quad (1)$$

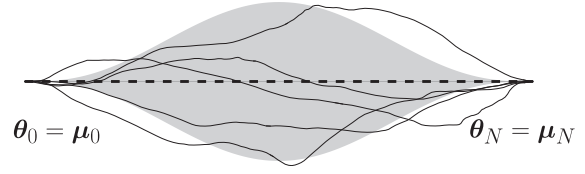
where the trajectory  $\theta(t)$  is a continuous-time function, mapping time  $t$  to robot states, which are generally configurations (and possibly higher-order derivatives).  $\mathcal{F}[\theta(t)]$  is an objective or cost functional that evaluates the quality of a trajectory and usually encodes *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*. Here  $\mathcal{G}_i[\theta(t)]$  are inequality constraint functionals such as joint angle limits, and  $\mathcal{H}_i[\theta(t)]$  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 Equation (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. GPs for continuous-time trajectories

A vector-valued 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 optimality properties such as smoothness are naturally encouraged (Section 4.1). We also consider a class of structured priors for trajectories that will be useful in efficient optimization (Section 4.2), and we provide details about how fast GP interpolation can be used to query the trajectory at any time of interest (Section 4.3).

#### 4.1. The GP prior

We consider continuous-time trajectories as samples from a vector-valued GP,  $\theta(t) \sim \mathcal{GP}(\mu(t), \mathcal{K}(t, t'))$ , where  $\mu(t)$  is a



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

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

$$\theta = [\theta_0 \quad \dots \quad \theta_N]^T \sim \mathcal{N}(\mu, \mathcal{K}) \quad (2)$$

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

$$\mu = [\mu_0 \quad \dots \quad \mu_N]^T, \quad \mathcal{K} = [\mathcal{K}(t_i, t_j)]_{ij, 0 \leq i, j \leq N} \quad (3)$$

We use bold  $\theta$  to denote the matrix formed by vectors  $\theta_i \in \mathbb{R}^D$ , which are *support states* that parameterize the continuous-time trajectory  $\theta(t)$ . Similar notation is used for  $\mu$ .

The GP defines a prior on the space of trajectories:

$$p(\theta) \propto \exp \left\{ -\frac{1}{2} \|\theta - \mu\|_{\mathcal{K}}^2 \right\} \quad (4)$$

where  $\|\theta - \mu\|_{\mathcal{K}}^2 \doteq (\theta - \mu)^T \mathcal{K}^{-1} (\theta - \mu)$  is the Mahalanobis distance. Figure 2 shows an example GP prior for trajectories. Intuitively this prior encourages *smoothness* encoded by the kernel  $\mathcal{K}$  and directly applies on the function space of trajectories. The negative log of this distribution serves as the prior cost functional in the objective (see Section 5.1) and penalizes the deviation of the trajectory from the mean defined by the prior.

#### 4.2. A Gauss–Markov model

Similar to previous work (Barfoot et al., 2014; Sarkka et al., 2013), we use a structured kernel generated by a linear time-varying stochastic differential equation (LTV-SDE)

$$\dot{\theta}(t) = \mathbf{A}(t)\theta(t) + \mathbf{u}(t) + \mathbf{F}(t)\mathbf{w}(t) \quad (5)$$

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 generated by a white noise process. The white noise process is itself a zero-mean GP

$$\mathbf{w}(t) \sim \mathcal{GP}(\mathbf{0}, \mathbf{Q}_C \delta(t - t')) \quad (6)$$

Here  $\mathbf{Q}_C$  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

$$\boldsymbol{\theta}(t) = \boldsymbol{\Phi}(t, t_0)\boldsymbol{\theta}_0 + \int_{t_0}^t \boldsymbol{\Phi}(t, s)(\mathbf{u}(s) + \mathbf{F}(s)\mathbf{w}(s))ds \quad (7)$$

where  $\boldsymbol{\Phi}(t, s)$  is the state transition matrix, which transfers state from time  $s$  to time  $t$ . The mean and covariance functions of the GP defined by this LTV-SDE are calculated by taking the first and second moments respectively on Equation (7),

$$\tilde{\boldsymbol{\mu}}(t) = \boldsymbol{\Phi}(t, t_0)\boldsymbol{\mu}_0 + \int_{t_0}^t \boldsymbol{\Phi}(t, s)\mathbf{u}(s) ds \quad (8)$$

$$\begin{aligned} \tilde{\mathcal{K}}(t, t') &= \boldsymbol{\Phi}(t, t_0)\mathcal{K}_0\boldsymbol{\Phi}(t', t_0)^T \\ &+ \int_{t_0}^{\min(t, t')} \boldsymbol{\Phi}(t, s)\mathbf{F}(s)\mathbf{Q}_C\mathbf{F}(s)^T\boldsymbol{\Phi}(t', s)^T ds \end{aligned} \quad (9)$$

where  $\boldsymbol{\mu}_0$  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  $\boldsymbol{\theta}_0$  and goal state  $\boldsymbol{\theta}_N$  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  $\boldsymbol{\mu}_N$  and covariance  $\mathcal{K}_N$ . Specifically,

$$\boldsymbol{\mu} = \tilde{\boldsymbol{\mu}} + \tilde{\mathcal{K}}(t_N, \mathbf{t})^T (\tilde{\mathcal{K}}(t_N, t_N) + \mathcal{K}_N)^{-1} (\boldsymbol{\theta}_N - \boldsymbol{\mu}_N) \quad (10)$$

$$\mathcal{K} = \tilde{\mathcal{K}} - \tilde{\mathcal{K}}(t_N, \mathbf{t})^T (\tilde{\mathcal{K}}(t_N, t_N) + \mathcal{K}_N)^{-1} \tilde{\mathcal{K}}(t_N, \mathbf{t}) \quad (11)$$

where  $\tilde{\mathcal{K}}(t_N, \mathbf{t}) = [\tilde{\mathcal{K}}(t_N, t_0) \dots \tilde{\mathcal{K}}(t_N, t_N)]$  (see Appendix A for the 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

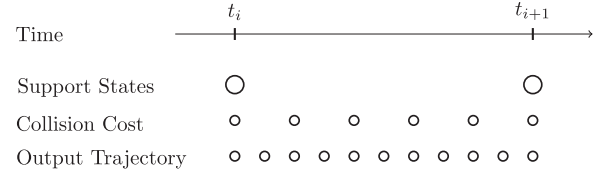
$$\mathcal{K}^{-1} = \mathbf{B}^T \mathbf{Q}^{-1} \mathbf{B} \quad (12)$$

with

$$\mathbf{B} = \begin{bmatrix} \mathbf{I} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ -\boldsymbol{\Phi}(t_1, t_0) & \mathbf{I} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & -\boldsymbol{\Phi}(t_2, t_1) & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \dots & -\boldsymbol{\Phi}(t_N, t_{N-1}) & \mathbf{I} \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{I} \end{bmatrix} \quad (13)$$

which has a band diagonal structure and  $\mathbf{Q}^{-1}$  is block diagonal such that

$$\mathbf{Q}^{-1} = \text{diag}(\mathcal{K}_0^{-1}, \mathbf{Q}_{0,1}^{-1}, \dots, \mathbf{Q}_{N-1,N}^{-1}, \mathcal{K}_N^{-1}) \quad (14)$$



**Fig. 3.** An example that shows the trajectory at different resolutions. Support states parameterize the trajectory, collision cost checking is performed at a higher resolution during optimization and the output trajectory can be up-sampled further for execution.

$$\mathbf{Q}_{a,b} = \int_{t_a}^{t_b} \boldsymbol{\Phi}(b, s)\mathbf{F}(s)\mathbf{Q}_C\mathbf{F}(s)^T\boldsymbol{\Phi}(b, s)^T ds \quad (15)$$

(see Appendix A for the proof). This sparse structure is useful for fast GP interpolation (Section 4.3) and efficient optimization (Sections 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  $\boldsymbol{\Phi}$  is a unit scalar, and  $\mathbf{Q}^{-1}$  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  $\mathbf{Q}^{-1}$  is identity).

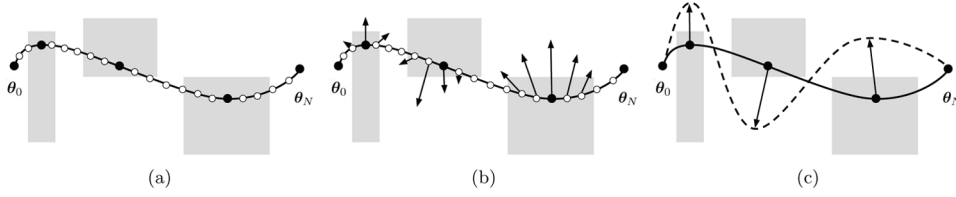
The linear model in Equation (5) is sufficient to model kinematics for the robot manipulators considered in the scope of this work, however our framework can be extended to consider nonlinear models following Anderson et al. (2015).

#### 4.3. GP interpolation

One of the benefits of GPs is that they can be parameterized by only a sparse set of *support states*, but the trajectory can be queried at *any* time of interest through GP interpolation. The reduced parameterization makes each iteration of trajectory 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 3) 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 4. 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 Sections 5.3 and





**Fig. 4.** An example showing how GP interpolation is used during optimization. In (a), the current iteration of the trajectory (black curve) is parameterized by a sparse set of support states (black circles). GPR is used to densely up-sample the trajectory with interpolated states (white circles). Then, in (b), cost is evaluated on all states and their gradients are illustrated by the arrows. Finally, in (c), the cost and gradient information is propagated to just the support states illustrated by the larger arrows such that only the support states are updated that parameterize the new trajectory (dotted black curve).

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), and Yan et al. (2017), we show how to 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:

$$\theta(\tau) = \tilde{\mu}(\tau) + \tilde{\mathcal{K}}(\tau, t) \tilde{\mathcal{K}}^{-1}(\theta - \tilde{\mu}) \quad (16)$$

i.e. performing GPR. Although the interpolation in Equation (16) naïvely requires  $O(N)$  operations,  $\theta(\tau)$  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  $\theta(\tau)$  at  $\tau, t_i < \tau < t_{i+1}$  can be expressed as a linear combination of *only* the adjacent function values  $\theta_i$  and  $\theta_{i+1}$  and is efficiently computed as

$$\theta(\tau) = \tilde{\mu}(\tau) + \Lambda(\tau)(\theta_i - \tilde{\mu}_i) + \Psi(\tau)(\theta_{i+1} - \tilde{\mu}_{i+1}) \quad (17)$$

where

$$\Lambda(\tau) = \Phi(\tau, t_i) - \Psi(\tau)\Phi(t_{i+1}, t_i)$$

$$\Psi(\tau) = \mathbf{Q}_{i,\tau} \Phi(t_{i+1}, \tau)^T \mathbf{Q}_{i,i+1}^{-1}$$

is derived by substituting

$$\tilde{\mathcal{K}}(\tau) \tilde{\mathcal{K}}^{-1} = [\mathbf{0} \dots \mathbf{0} \Lambda(\tau) \Psi(\tau) \mathbf{0} \dots \mathbf{0}]$$

in Equation (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 Sections 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 GP representation with a gradient descent-based optimization algorithm for motion planning.

### 5.1. Cost functionals

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

$$\mathcal{F}[\theta(t)] = \mathcal{F}_{obs}[\theta(t)] + \lambda \mathcal{F}_{gp}[\theta(t)] \quad (18)$$

where  $\mathcal{F}_{gp}$  is the GP prior cost functional (the negative natural logarithm of prior distribution) from Equation (4)

$$\mathcal{F}_{gp}[\theta(t)] = \mathcal{F}_{gp}[\theta] = \frac{1}{2} \|\theta - \mu\|_{\mathcal{K}}^2 \quad (19)$$

penalizing the deviation of the parameterized trajectory from the prior mean,  $\mathcal{F}_{obs}$  is the obstacle cost functional that penalizes collision with obstacles, and  $\lambda$  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  $\mathcal{F}_{obs}$  is also similar to that 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:

$$\mathcal{F}_{obs}[\theta(t)] = \int_{t_0}^{t_N} \int_{\mathcal{B}} c(x) \|\dot{x}\| du dt \quad (20)$$

where  $c(\cdot) : \mathbb{R}^3 \rightarrow \mathbb{R}$  is the workspace cost function that penalizes the set of points  $\mathcal{B} \subset \mathbb{R}^3$  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, i.e.  $\mathcal{F}_{obs}[\theta(t)] = \mathcal{F}_{obs}[\theta]$ , the obstacle cost is

calculated using a precomputed signed distance field (SDF; 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 Equation (18). In each iteration, we form an approximation to the cost functional via a Taylor series expansion around the current parameterized trajectory  $\theta$ :

$$\mathcal{F}[\theta + \delta\theta] \approx \mathcal{F}[\theta] + \bar{\nabla}\mathcal{F}[\theta]\delta\theta \quad (21)$$

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

$$\delta\theta^* = \underset{\delta\theta}{\operatorname{argmin}} \left\{ \mathcal{F}[\theta] + \bar{\nabla}\mathcal{F}[\theta]\delta\theta + \frac{\eta}{2} \|\delta\theta\|_{\mathcal{K}}^2 \right\} \quad (22)$$

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:

$$\begin{aligned} \bar{\nabla}\mathcal{F}[\theta] + \eta\mathcal{K}^{-1}\delta\theta^* &= 0 \quad \Rightarrow \quad \delta\theta^* = -\frac{1}{\eta}\mathcal{K}\bar{\nabla}\mathcal{F}[\theta] \\ \theta &\leftarrow \theta + \delta\theta^* = \theta - \frac{1}{\eta}\mathcal{K}\bar{\nabla}\mathcal{F}[\theta] \end{aligned} \quad (23)$$

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

$$\bar{\nabla}\mathcal{F}[\theta] = \bar{\nabla}\mathcal{F}_{obs}[\theta] + \lambda\bar{\nabla}\mathcal{F}_{gp}[\theta] \quad (24)$$

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 Equation (19) with respect to the current trajectory

$$\begin{aligned} \mathcal{F}_{gp}[\theta] &= \frac{1}{2}(\theta - \mu)^T \mathcal{K}^{-1}(\theta - \mu) \\ \bar{\nabla}\mathcal{F}_{gp}[\theta] &= \mathcal{K}^{-1}(\theta - \mu) \end{aligned} \quad (25)$$

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

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

Applying Equation (26) to find the gradient of Equation (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

$$\bar{\nabla}\mathcal{F}_{obs}[\theta_i] = \begin{bmatrix} \int_B J^T \|\dot{x}\| [(I - \hat{x}\hat{x}^T)\nabla c - c\kappa] du \\ \int_B J^T c \hat{x} du \\ 0 \end{bmatrix} \quad (27)$$

where  $\kappa = \|\dot{x}\|^{-2}(I - \hat{x}\hat{x}^T)\ddot{x}$  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. Owing 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  $\mathbf{g} = \bar{\nabla}\mathcal{F}_{obs}[\theta]$ . We plug the cost gradients back into the update rule in Equation (23) to obtain the update

$$\theta \leftarrow \theta - \frac{1}{\eta}\mathcal{K}(\lambda\mathcal{K}^{-1}(\theta - \mu) + \mathbf{g}) \quad (28)$$

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 Equation (17),

$$\begin{aligned} \Lambda_i &= \begin{bmatrix} \Lambda_{i,1}^T & \dots & \Lambda_{i,j}^T & \dots & \Lambda_{i,n_{ip}}^T \end{bmatrix}^T \\ \Psi_i &= \begin{bmatrix} \Psi_{i,1}^T & \dots & \Psi_{i,j}^T & \dots & \Psi_{i,n_{ip}}^T \end{bmatrix}^T \end{aligned}$$

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

$$\theta_{up} = \mathbf{M}(\theta - \mu) + \mu_{up} \quad (29)$$

where  $\mu_{up}$  corresponds to the prior mean with respect to the up sampled trajectory, and

$$\mathbf{M} = \begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} & \dots & \dots & \dots & \mathbf{0} & \mathbf{0} \\ \Lambda_0 & \Psi_0 & \mathbf{0} & \dots & \dots & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} & \dots & \dots & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \Lambda_1 & \Psi_1 & \dots & \dots & \dots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \ddots & & & & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{I} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \dots & \Lambda_i & \Psi_i & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{I} & \dots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & & & \ddots & & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \dots & \dots & \dots & \Lambda_{N-1} & \Psi_{N-1} \\ \mathbf{0} & \mathbf{0} & \dots & \dots & \dots & \dots & \mathbf{0} & \mathbf{I} \end{bmatrix} \quad (30)$$

is a tall matrix that up-samples a sparse trajectory  $\theta$  with only  $N+1$  support states to trajectory  $\theta_{up}$  with  $(N+1) + N \times n_{ip}$  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 Equation (28) except on a sparse parametrization of the trajectory

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

where the obstacle gradient over the sparse trajectory is found by chain rule using Equation (29) and the obstacle gradient,  $\mathbf{g}_{up}$ , 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  $\mathbf{M}^T$ . 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 obtain a feasible solution; and, second, the gradients can be costly to calculate (See Figure 13). 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 (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, such as 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; Mukadam-AURO-18), and learning from demonstration and planning (Rana et al., 2017; Rana-IROS-18).

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  $\mathbf{e}$ . For example, binary events  $e_i$  at  $t_i$  might signify that the trajectory is collision-free if all  $e_i = 0$  (i.e.  $\mathbf{e} = \mathbf{0}$ ) 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  $\mathbf{e}$  can be computed by Bayes rule from a prior and likelihood

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

$$\propto p(\theta)p(\mathbf{e}|\theta) \quad (33)$$

where  $p(\theta)$  is the prior on  $\theta$  that encourages smooth trajectories, and  $p(\mathbf{e}|\theta)$  is the likelihood that the trajectory  $\theta$  is collision-free. The optimal trajectory  $\theta$  is found by the *maximum a posteriori* (MAP) estimator, which chooses the trajectory that maximizes the posterior  $p(\theta|\mathbf{e})$

$$\theta^* = \underset{\theta}{\operatorname{argmax}} p(\theta|\mathbf{e}) \quad (34)$$

$$= \underset{\theta}{\operatorname{argmax}} p(\theta)l(\theta;\mathbf{e}) \quad (35)$$

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

$$l(\theta;\mathbf{e}) \propto p(\mathbf{e}|\theta) \quad (36)$$



We use the same GP prior as in Section 4

$$p(\boldsymbol{\theta}) \propto \exp \left\{ -\frac{1}{2} \|\boldsymbol{\theta} - \boldsymbol{\mu}\|_{\boldsymbol{\kappa}}^2 \right\} \quad (37)$$

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

$$l(\boldsymbol{\theta}; \mathbf{e}) = \exp \left\{ -\frac{1}{2} \|\mathbf{h}(\boldsymbol{\theta})\|_{\boldsymbol{\Sigma}_{obs}}^2 \right\} \quad (38)$$

where  $\mathbf{h}(\boldsymbol{\theta})$  is a vector-valued *obstacle cost* for the trajectory, and  $\boldsymbol{\Sigma}_{obs}$  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  $G = \{\Theta, \mathcal{F}, \mathcal{E}\}$  is a bipartite graph, which represents a factored function, where  $\Theta = \{\boldsymbol{\theta}_0, \dots, \boldsymbol{\theta}_N\}$  are a set of variable nodes,  $\mathcal{F} = \{f_0, \dots, f_M\}$  are a set of factor nodes, and  $\mathcal{E}$  are edges connecting the two type of nodes.

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

$$p(\boldsymbol{\theta}|\mathbf{e}) \propto \prod_{m=1}^M f_m(\Theta_m), \quad (39)$$

where  $f_m$  are factors on variable subsets  $\Theta_m$ .

Given the tridiagonal inverse kernel matrix defined by Equations (12)–(14), we factor the prior

$$p(\boldsymbol{\theta}) \propto f_0^p(\boldsymbol{\theta}_0) f_N^p(\boldsymbol{\theta}_N) \prod_{i=0}^{N-1} f_i^{gp}(\boldsymbol{\theta}_i, \boldsymbol{\theta}_{i+1}) \quad (40)$$

where  $f_0^p(\boldsymbol{\theta}_0)$  and  $f_N^p(\boldsymbol{\theta}_N)$  define the prior distributions on start and goal states, respectively,

$$f_i^p(\boldsymbol{\theta}_i) = \exp \left\{ -\frac{1}{2} \|\boldsymbol{\theta}_i - \boldsymbol{\mu}_i\|_{\boldsymbol{\kappa}_i}^2 \right\}, \quad i = 0 \text{ or } N \quad (41)$$

where  $\boldsymbol{\kappa}_0$  and  $\boldsymbol{\kappa}_N$  are covariance matrices on start and goal states, respectively, and  $\boldsymbol{\mu}_0$  and  $\boldsymbol{\mu}_N$  are prior (known) start and goal states, respectively. The GP prior factor is

$$f_i^{gp}(\boldsymbol{\theta}_i, \boldsymbol{\theta}_{i+1}) = \exp \left\{ -\frac{1}{2} \|\boldsymbol{\Phi}(t_{i+1}, t_i) \boldsymbol{\theta}_i - \boldsymbol{\theta}_{i+1} + \mathbf{u}_{i,i+1}\|_{\mathbf{Q}_{i,i+1}}^2 \right\} \quad (42)$$

where  $\mathbf{u}_{a,b} = \int_{t_a}^{t_b} \boldsymbol{\Phi}(b, s) \mathbf{u}(s) ds$ ,  $\boldsymbol{\Phi}(t_{i+1}, t_i)$  is the state transition matrix, and  $\mathbf{Q}_{i,i+1}$  is defined by Equation (14) (see Yan et al., 2017 for details).

To factor the collision-free likelihood  $l(\boldsymbol{\theta}; \mathbf{e})$ , we define two types of obstacle cost factors: regular obstacle factors  $f_i^{obs}$  and interpolated obstacle factors  $f_{\tau_j}^{intp}$ . The  $l(\boldsymbol{\theta}; \mathbf{e})$  is the product of all obstacle factors

$$l(\boldsymbol{\theta}; \mathbf{e}) = \prod_{i=0}^N \left\{ f_i^{obs}(\boldsymbol{\theta}_i) \prod_{j=1}^{n_{ip}} f_{\tau_j}^{intp}(\boldsymbol{\theta}_i, \boldsymbol{\theta}_{i+1}) \right\} \quad (43)$$

where  $n_{ip}$  is the number of *interpolated* states defined between each nearby support state pair  $\boldsymbol{\theta}_i$  and  $\boldsymbol{\theta}_{i+1}$ , and  $\tau_j$  is the time to perform interpolation which satisfies  $t_i < \tau_j < t_{i+1}$ .

The regular obstacle factor describes the obstacle cost on a single state variable and is a *unary* factor defined as

$$f_i^{obs}(\boldsymbol{\theta}_i) = \exp \left\{ -\frac{1}{2} \|\mathbf{h}(\boldsymbol{\theta}_i)\|_{\boldsymbol{\sigma}_{obs}}^2 \right\} \quad (44)$$

where  $\mathbf{h}(\boldsymbol{\theta}_i)$  is an  $M$ -dimensional vector-valued obstacle cost function for a single state, and  $\boldsymbol{\sigma}_{obs}$  is a  $M \times M$  hyperparameter matrix.

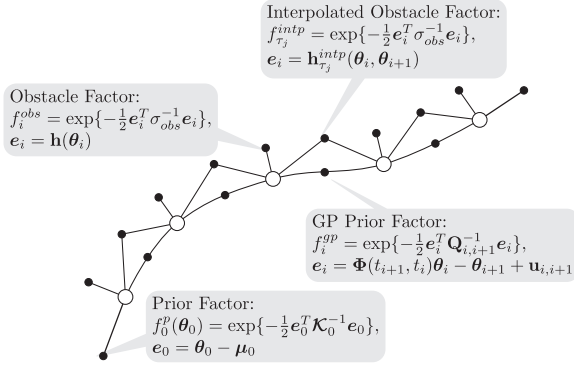
The interpolated obstacle factor describes the obstacle cost at  $\tau_j$ , which is not on any support state and needs be interpolated from the support states. Since the Gauss-Markov model we choose enables fast interpolation from adjacent states, we can interpolate a state at any  $\tau_j$  from  $\boldsymbol{\theta}_i$  and  $\boldsymbol{\theta}_{i+1}$  by Equation (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

$$\begin{aligned} f_{\tau_j}^{intp}(\boldsymbol{\theta}_i, \boldsymbol{\theta}_{i+1}) &= \exp \left\{ -\frac{1}{2} \|\mathbf{h}(\boldsymbol{\theta}(\tau_j))\|_{\boldsymbol{\sigma}_{obs}}^2 \right\} \\ &= \exp \left\{ -\frac{1}{2} \|\mathbf{h}_{\tau_j}^{intp}(\boldsymbol{\theta}_i, \boldsymbol{\theta}_{i+1})\|_{\boldsymbol{\sigma}_{obs}}^2 \right\}. \end{aligned} \quad (45)$$

In other words,  $\boldsymbol{\theta}(\tau_j)$  is a function of  $\boldsymbol{\theta}_i$  and  $\boldsymbol{\theta}_{i+1}$  (see Equation (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 Equation (43)–(45), we can retrieve the vector-valued obstacle cost function of the trajectory defined in Equation (38) by simply stacking all the vector-valued obstacle cost functions on all regular and interpolated states into a single vector



**Fig. 5.** A factor graph of an example trajectory optimization problem showing support states (white circles) and four kinds of factors (black dots), namely prior factors on start and goal states, GP prior factors that connect consecutive support states, obstacle factors on each state, and interpolated obstacle factors between consecutive support states (only one shown here for clarity, any number of them may be present in practice).

$$\begin{aligned} \mathbf{h}(\boldsymbol{\theta}) = & \left[ \mathbf{h}(\boldsymbol{\theta}_0); \mathbf{h}_{\tau_1}^{interp}(\boldsymbol{\theta}_0, \boldsymbol{\theta}_1); \dots; \mathbf{h}_{\tau_{n_{ip}}}^{interp}(\boldsymbol{\theta}_0, \boldsymbol{\theta}_1); \right. \\ & \mathbf{h}(\boldsymbol{\theta}_1); \mathbf{h}_{\tau_1}^{interp}(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2); \dots; \mathbf{h}_{\tau_{n_{ip}}}^{interp}(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2); \\ & \dots \\ & \mathbf{h}(\boldsymbol{\theta}_{N-1}); \mathbf{h}_{\tau_1}^{interp}(\boldsymbol{\theta}_{N-1}, \boldsymbol{\theta}_N); \dots; \mathbf{h}_{\tau_{n_{ip}}}^{interp}(\boldsymbol{\theta}_{N-1}, \boldsymbol{\theta}_N); \\ & \left. \mathbf{h}(\boldsymbol{\theta}_N) \right] \end{aligned} \quad (46)$$

where all  $\mathbf{h}$  are obstacle cost functions from regular obstacle factors defined in Equation (44), and all  $\mathbf{h}^{interp}$  are obstacle cost functions from interpolated obstacle factors defined in Equation (45). As 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  $\mathbf{h}(\boldsymbol{\theta})$  is  $M \times (N + 1 + N \times n_{ip})$ . The hyperparameter matrix  $\boldsymbol{\Sigma}_{obs}$  in Equation (38) is then defined by

$$\boldsymbol{\Sigma}_{obs} = \begin{bmatrix} \boldsymbol{\sigma}_{obs} & & \\ & \ddots & \\ & & \boldsymbol{\sigma}_{obs} \end{bmatrix} \quad (47)$$

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  $\mathbf{h}$ ,  $M$ , and  $\boldsymbol{\sigma}_{obs}$  are flexible as long as  $l(\boldsymbol{\theta}; \mathbf{e})$  gives the collision-free likelihood. Effectively  $\mathbf{h}(\boldsymbol{\theta}_i)$  should have a larger value when a robot collides with obstacles at  $\boldsymbol{\theta}_i$ , and a smaller value when the robot is collision-free. Our implementation of  $\mathbf{h}$ , definition of  $M$ , and guideline for the hyperparameter  $\boldsymbol{\sigma}_{obs}$  is discussed in Section 8.2.2.

### 6.3. Computing the MAP trajectory

To solve the MAP inference problem in Equation (35), we first illustrate the duality between inference and

optimization by performing minimization on the negative log of the posterior distribution

$$\begin{aligned} \boldsymbol{\theta}^* &= \underset{\boldsymbol{\theta}}{\operatorname{argmax}} p(\boldsymbol{\theta}) l(\boldsymbol{\theta}; \mathbf{e}) \\ &= \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \{-\log(p(\boldsymbol{\theta}) l(\boldsymbol{\theta}; \mathbf{e}))\} \\ &= \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \left\{ \frac{1}{2} \|\boldsymbol{\theta} - \boldsymbol{\mu}\|_{\mathcal{K}}^2 + \frac{1}{2} \|\mathbf{h}(\boldsymbol{\theta})\|_{\boldsymbol{\Sigma}_{obs}}^2 \right\} \end{aligned} \quad (48)$$

where Equation (48) follows from Equations (37) and (38). This duality connects the two different perspectives on motion planning problems such that the terms in Equation (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, such as Gauss–Newton or Levenberg–Marquardt, repeatedly resolve a quadratic approximation of Equation (48) until convergence.

Linearizing the nonlinear obstacle cost function around the current trajectory  $\boldsymbol{\theta}$

$$\mathbf{h}(\boldsymbol{\theta} + d\boldsymbol{\theta}) \approx \mathbf{h}(\boldsymbol{\theta}) + \mathbf{H}d\boldsymbol{\theta} \quad (49)$$

$$\mathbf{H} = \frac{d\mathbf{h}}{d\boldsymbol{\theta}} \Big|_{\boldsymbol{\theta}} \quad (50)$$

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

$$\delta\boldsymbol{\theta}^* = \underset{\delta\boldsymbol{\theta}}{\operatorname{argmin}} \left\{ \frac{1}{2} \|\boldsymbol{\theta} + \delta\boldsymbol{\theta} - \boldsymbol{\mu}\|_{\mathcal{K}}^2 + \frac{1}{2} \|\mathbf{h}(\boldsymbol{\theta}) + \mathbf{H}\delta\boldsymbol{\theta}\|_{\boldsymbol{\Sigma}_{obs}}^2 \right\}. \quad (51)$$

The optimal perturbation  $\delta\boldsymbol{\theta}^*$  results from solving the following linear system

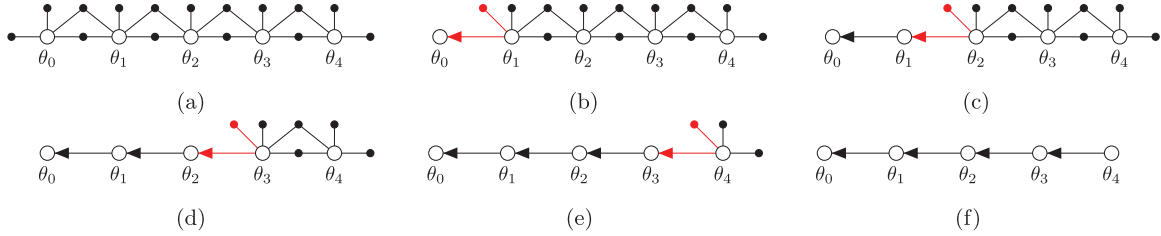
$$(\mathcal{K}^{-1} + \mathbf{H}^T \boldsymbol{\Sigma}_{obs}^{-1} \mathbf{H}) \delta\boldsymbol{\theta}^* = -\mathcal{K}^{-1}(\boldsymbol{\theta} - \boldsymbol{\mu}) - \mathbf{H}^T \boldsymbol{\Sigma}_{obs}^{-1} \mathbf{h}(\boldsymbol{\theta}) \quad (52)$$

Once the linear system is solved, the iteration

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \delta\boldsymbol{\theta}^* \quad (53)$$

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

If the linear system in Equation (52) is sparse, then  $\delta\boldsymbol{\theta}^*$  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 GP prior with a block tridiagonal precision matrix  $\mathcal{K}^{-1}$  (Section 4.2) and  $\mathbf{H}^T \boldsymbol{\Sigma}_{obs}^{-1} \mathbf{H}$  is also block tridiagonal (see the 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.



**Fig. 6.** Example of applying variable elimination on a planning factor graph: (a) factor graph; (b) eliminating  $\theta_0$ ; (c) eliminating  $\theta_1$ ; (d) eliminating  $\theta_2$ ; (e) eliminating  $\theta_3$ ; (f) eliminating  $\theta_4$  and final Bayes net. Red arrows/factors indicate the parts that change in Bayes net/factor graph, respectively.

## 7. Incremental inference for fast replanning

We have described how to formulate a motion planning problem as probabilistic inference on factor graphs, resulting 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.<sup>1</sup> We first give a brief overview of the Bayes tree and its relation to factor graphs, and then we give a more detailed example to show how to use a Bayes tree to perform incremental inference for a replanning problem.

### 7.1. Bayes tree as incremental inference

Before introducing the Bayes tree, we first briefly review the *variable elimination* process in smoothing and mapping (Dellaert and Kaess, 2006), which converts a factor graph into a *Bayes net*. More details about the elimination algorithm can be found in Dellaert and Kaess (2006) and Kaess et al. (2011a). Given a factor graph over variables  $\Theta$ , and a

variable elimination ordering, we would like to convert the probabilistic density over all variables  $p(\Theta)$  to the form

$$p(\Theta) = \prod_j p(\theta_j | S_j) \quad (54)$$

by Algorithm 1, where  $S_j \subset \Theta$  is the separator of  $\theta_j$ . Note that different variable elimination orderings will generate different Bayes nets. Therefore, we want to select an optimal ordering to produce a Bayes net which is as sparse as possible. Given the chain-like factor graph structure in planning problems, as shown in Figure 5, we can simply select the ordering from start state  $\theta_0$  to goal state  $\theta_N$ , which gives us a chain-like Bayes net structure (which is equivalent to  $S_j = \{\theta_{j+1}\}$ ) and the density can be factorized as

$$p(\theta) = p(\theta_N) \prod_{j=0}^{N-1} p(\theta_j | \theta_{j+1}) \quad (55)$$

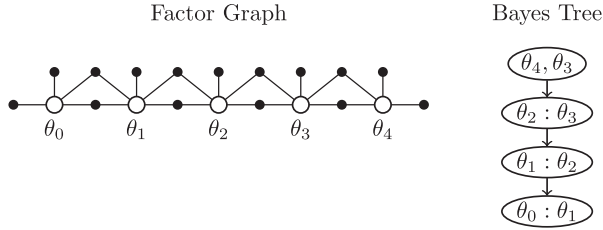
An example of applying variable elimination on a planning factor graph of five states is shown in Figure 6.

**Algorithm 1:** Eliminate variable  $\theta_j$  from factor graph

1. Remove all factors  $f_i$  connected to  $\theta_j$  from factor graph, define  $S_j = \{\text{all variables involved in all } f_i\}$
3.  $f_j(\theta_j, S_j) = \prod_i f_i(\Theta_i)$
4. Factorize  $f_j(\theta_j, S_j) = p(\theta_j | S_j) f_{\text{new}}(S_j)$
5. Add  $p(\theta_j | S_j)$  in Bayes net, and add  $f_{\text{new}}(S_j)$  back in factor graph.

A Bayes tree is a tree-structured graphical model that is derived from a Bayes net. The Bayes net generated from the variable elimination algorithm is proved to be chordal (Kaess et al., 2011b), so we can always produce junction trees (Cowell et al., 2006) from such Bayes nets. A Bayes tree is similar to a junction tree but is directed, which represent the conditional relations in factored probabilistic density.

For a Bayes net that results from Algorithm 1, we extract all cliques  $C_k$  and build the Bayes tree by defining all node as cliques  $C_k$ . For each node of Bayes tree, we define a conditional density  $p(F_k | S_k)$ , where  $S_k$  is the *separator* as intersection  $S_k = C_k \cap \Pi_k$  between  $C_k$  and  $C_k$ 's parent  $\Pi_k$ .



**Fig. 7.** Example of a Bayes tree with its corresponding factor graph.

The *frontal variable* is defined by  $F_k = C_k \setminus S_k$ . We can also write the clique as  $C_k = F_k : S_k$ . The joint density of the Bayes tree is defined by

$$p(\Theta) = \prod_k p_{C_k}(F_k | S_k) \quad (56)$$

For a detailed algorithm to convert an arbitrary chordal Bayes net into a Bayes tree please refer to Kaess et al. (2011b), but since our planning Bayes net has a simple chain structure, as shown in Equation (55) and Figure 6, all the cliques are  $C_k = \{\theta_k, \theta_{k+1}\}$  and the joint density is

$$p(\theta) = p_{C_{N-1}}(\theta_N, \theta_{N-1}) \prod_{k=0}^{N-2} p_{C_k}(\theta_k | \theta_{k+1}) \quad (57)$$

and the corresponding Bayes tree is shown in Figure 7.

Since a Bayes tree has nice tree structure, the inference is performed from root to leaves. If variable  $\theta_i$  is affected by new factors, only cliques contain  $\theta_i$  or cliques between cliques contain  $\theta_i$  to root are affected, and need to be re-factorized, remaining cliques can be left unchanged. This makes performing incremental inference with minimal computation possible, since we only have to re-factorize parts of cliques that have changed. The closer the affected cliques are to the root (in planning cases the affected state  $\theta_i$  is closer to the goal state  $\theta_N$ ), the less computation is needed for incremental inference. Readers are encouraged to refer to Kaess et al. (2011b) for more details about how incremental inference is performed with a Bayes tree.

## 7.2. Replanning using Bayes tree

Two replanning examples with Bayes tree are shown in Figure 8. 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 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.

## Algorithm 2: Replanning using iSAM2

---

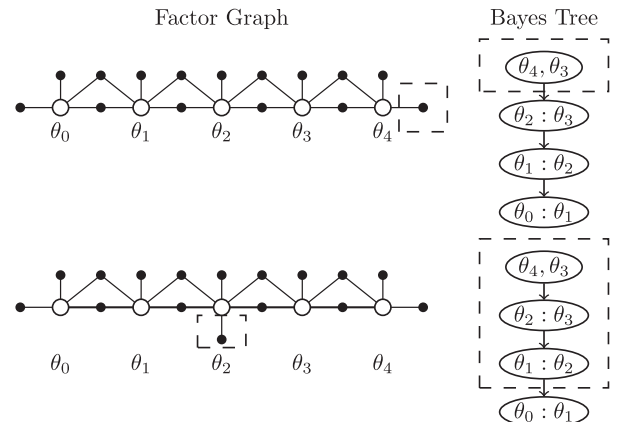
**Input:** new factors  $f_{new}$ , replaced factors  $f_{replace}$   
**Output:** updated optimal trajectory  $\theta^*$   
*Initialization :*  
 add factors  $f_{add} = \emptyset$ , remove factors  $f_{remove} = \emptyset$   
*iSAM2 update:*  
 $f_{add} = f_{new}$   
**if** ( $f_{replace} \neq \emptyset$ ) **then**  
      $f_{add} = f_{add} + f_{replace}$   
      $f_{remove} = \text{findOldFactors}(f_{replace})$   
**end**  
 iSAM2.updateBayesTree( $f_{add}$ ,  $f_{remove}$ )  
**return** iSAM2.getCurrentEstimation()  


---

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 2 to update the Bayes tree inside iSAM2, to obtain 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, `gmp2`.<sup>2</sup> 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.



**Fig. 8.** Replanning examples using Bayes trees. Dashed boxes indicate parts of the factor graphs and Bayes trees that are affected and changed while performing replanning.

### 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 the configuration position, velocity, and acceleration, by following the LTV-SDE in Equation (5) with parameters

$$\mathbf{A}(t) = \begin{bmatrix} \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix}, \quad \mathbf{u}(t) = \mathbf{0}, \quad \mathbf{F}(t) = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{I} \end{bmatrix} \quad (58)$$

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

$$\Phi(t, s) = \begin{bmatrix} \mathbf{I} & (t-s)\mathbf{I} & \frac{1}{2}(t-s)^2\mathbf{I} \\ \mathbf{0} & \mathbf{I} & (t-s)\mathbf{I} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix} \quad (59)$$

$$\mathbf{Q}_{i,i+1} = \begin{bmatrix} \frac{1}{2}\Delta t_i^5 \mathbf{Q}_C & \frac{1}{8}\Delta t_i^4 \mathbf{Q}_C & \frac{1}{6}\Delta t_i^3 \mathbf{Q}_C \\ \frac{1}{8}\Delta t_i^4 \mathbf{Q}_C & \frac{1}{3}\Delta t_i^3 \mathbf{Q}_C & \frac{1}{2}\Delta t_i^2 \mathbf{Q}_C \\ \frac{1}{6}\Delta t_i^3 \mathbf{Q}_C & \frac{1}{2}\Delta t_i^2 \mathbf{Q}_C & \Delta t_i \mathbf{Q}_C \end{bmatrix} \quad (60)$$

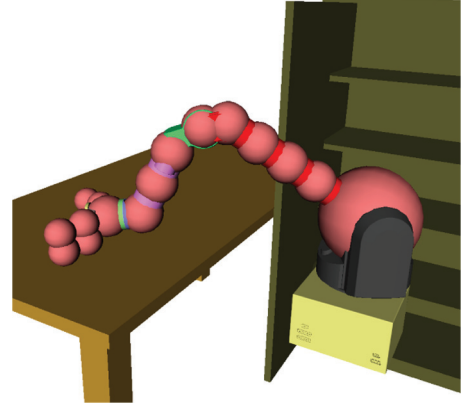
This prior is centered around a zero jerk trajectory and encourages smoothness by attempting to minimize jerk during optimization.

**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 9). 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 Equation (20)) where the cost is summed over the sphere set on the robot body calculated using a precomputed 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 Equation (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,  $\theta_{up}^v$  to be projected:

$$\theta = \theta + \mathcal{K}\mathbf{M}^T \theta_{up}^v \quad (61)$$

### 8.2. GPMP2 and iGPMP2

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



**Fig. 9.** WAM arm represented by multiple spheres (pink), which are used during collision cost calculation.

**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 such as quadrotors. GPMP2 scales only quadratically 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 Equation (5) with

$$\mathbf{A}(t) = \begin{bmatrix} \mathbf{0} & \mathbf{I} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}, \quad \mathbf{u}(t) = \mathbf{0}, \quad \mathbf{F}(t) = \begin{bmatrix} \mathbf{0} \\ \mathbf{I} \end{bmatrix} \quad (62)$$

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

$$\Phi(t, s) = \begin{bmatrix} \mathbf{I} & (t-s)\mathbf{I} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}, \quad \mathbf{Q}_{i,i+1} = \begin{bmatrix} \frac{1}{2}\Delta t_i^3 \mathbf{Q}_C & \frac{1}{2}\Delta t_i^2 \mathbf{Q}_C \\ \frac{1}{2}\Delta t_i^2 \mathbf{Q}_C & \Delta t_i \mathbf{Q}_C \end{bmatrix} \quad (63)$$

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 9, and the obstacle cost function for any configuration  $\theta_i$  is then completed by computing the hinge loss for each sphere  $S_j$  ( $j = 1, \dots, M$ ) and collecting them into a single vector,

$$\mathbf{h}(\theta_i) = [\mathbf{c}(\mathbf{d}(\mathbf{x}(\theta_i, S_j)))]_{1 \leq j \leq M} \quad (64)$$

where  $\mathbf{x}$  is the forward kinematics,  $\mathbf{d}$  is the signed distance function,  $\mathbf{c}$  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 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<sup>3</sup> is defined as

$$\mathbf{c}(d) = \begin{cases} -d + \epsilon & \text{if } d \geq \epsilon \\ 0 & \text{if } d < \epsilon \end{cases} \quad (65)$$

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 Equations (44) and (45) is defined by an isotropic diagonal matrix

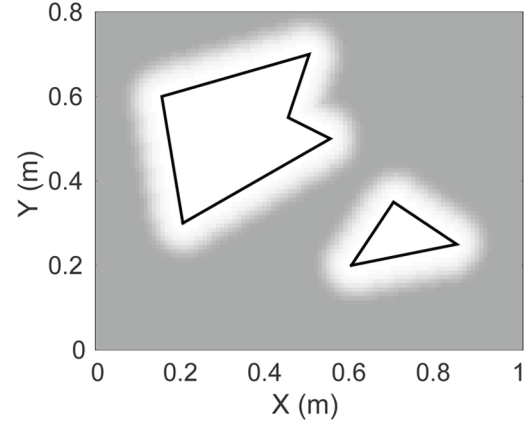
$$\Sigma_{obs} = \sigma_{obs}^2 \mathbf{I} \quad (66)$$

where  $\sigma_{obs}$  is the “obstacle cost weight” parameter.

Figure 10 visualizes a 2D example of the collision-free likelihood defined by the obstacle cost function in Equation (64). 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 that used in GPMP and CHOMP, where  $\mathbf{c}$  is instead a smooth function (necessary for gradient calculation) and is multiplied with the norm of the workspace velocity (see Equation (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 Equation (65) serves as a good approximation for the tasks we consider and is also less computationally expensive.

**8.2.3. 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. Since we are solving unconstrained least square problems, there is no way to enforce direct equality or inequality constraints during inference. In our framework, these constraints are instead handled in a “soft” way, by treating them as prior knowledge on the trajectory states with very small uncertainties. Although the



**Fig. 10.** The likelihood function  $\mathbf{h}$  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.

constraints are not exact, this has not been an issue in practice in any of our evaluations.

Additional *equality* motion constraints, such as end-effector rotation constraints (e.g. holding a cup filled with water upright) written as  $\mathbf{f}(\theta_c) = \mathbf{0}$ , where  $\theta_c$  is the set of states involved, can be incorporated into a likelihood,

$$L_{constraint}(\theta) \propto \exp \left\{ -\frac{1}{2} \|\mathbf{f}(\theta_c)\|_{\Sigma_c}^2 \right\} \quad (67)$$

where  $\Sigma_c = \sigma_c^2 \mathbf{I}$ ,  $\sigma_c$  is an arbitrary variance for this constraint, indicating how “tight” the constraint is.

To prevent joint-limit (and velocity-limit) violations, we add *inequality* soft constraint factors to the factor graph. Similar to obstacle factors, the inequality motion constraint factor uses a hinge loss function to enforce soft constraints at both the maximum  $\theta_{\max}^d$  and the minimum  $\theta_{\min}^d$  values, with some given safety margin  $\epsilon$  on each dimension  $d = \{1, \dots, D\}$

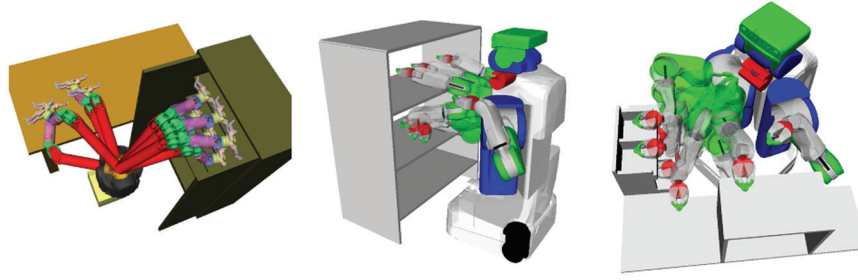
$$\mathbf{c}(\theta_i^d) = \begin{cases} -\theta_i^d + \theta_{\min}^d - \epsilon & \text{if } \theta_i^d < \theta_{\min}^d + \epsilon \\ 0 & \text{if } \theta_{\min}^d + \epsilon \leq \theta_i^d \leq \theta_{\max}^d - \epsilon \\ \theta_i^d - \theta_{\max}^d + \epsilon & \text{if } \theta_i^d > \theta_{\max}^d - \epsilon \end{cases} \quad (68)$$

This factor has a vector valued cost function  $\mathbf{f}(\theta_i) = [\mathbf{c}(\theta_i^d)]_{1 \leq d \leq D}$  and the same likelihood as the equality constraint factor in Equation (67). At the final iteration we also detect limit violations and clamp to the maximum or minimum values.

## 9. Evaluation

We conducted our experiments<sup>4</sup> on two datasets with different start and goal configurations. We used: (1) the seven-degree-of-freedom (7-DOF) WAM arm dataset (Mukadam et al., 2016) consisting of 24 unique planning problems in the *lab* environment; and (2) the PR2’s 7-DOF right arm





**Fig. 11.** Environments used for evaluation with robot start and goal configurations showing the WAM dataset (left), and a subset of the PR2 dataset (*bookshelves* (center) and *industrial* (right)).

dataset (Schulman et al., 2014) consisting of a total of 198 unique planning problems in four different environments (Figure 11). Finally, we validated successful trajectories on a real 7-DOF WAM arm in an environment identical to the simulation (Figure 1).

### 9.1. Batch planning benchmark

**9.1.1. Setup.** We benchmarked our algorithms, GPMP and GPMP2, both with interpolation (GPMP2-intp) during optimization and without interpolation (GPMP2-no-intp) against trajectory optimization algorithms, TrajOpt (Schulman et al., 2014) and CHOMP (Zucker et al., 2013), and against sampling based algorithms, RRTConnect (Kuffner and LaValle, 2000) and LBKPIECE (Şucan and Kavraki, 2009) available within the OMPL implementation (Şucan et al., 2012). All benchmarks were run on a single thread of a 3.4 GHz Intel Core i7 CPU.

For trajectory optimizers, GPMP2, TrajOpt and CHOMP were initialized by a constant-velocity straight line trajectory in configuration space and GPMP was initialized by an acceleration-smooth straight line. For the WAM dataset all initialized trajectories were parameterized by 101 temporally equidistant states. GPMP2-intp and GPMP use interpolation so we initialized them with 11 support states and  $n_{ip} = 9$  (101 states effectively). Since trajectory tasks are shorter in the PR2 dataset, we used 61 temporally equidistant states to initialize the trajectories and for GPMP2-intp and GPMP we used 11 support states and  $n_{ip} = 5$  (61 states effectively).

To keep comparisons fair we also compared against TrajOpt using only 11 states (TrajOpt-11) in both datasets since it uses continuous-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 may not offer any collision-free guarantees. GPMP and GPMP2 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.

For sampling-based planners no post-processing or smoothing step was applied and they were used with default settings.

All algorithms were allowed to run for a maximum of 10 seconds on any problem and marked successful if a feasible solution is found in that time. GPMP, CHOMP, RRT-Connect, and LBKPIECE are stopped if a collision-free trajectory is found before the max time (for GPMP and CHOMP collision checking is started after optimizing for at least 10 iterations). GPMP2 and TrajOpt are stopped when convergence is reached before the max time (we observed this was always the case) and feasibility is evaluated post-optimization.

**9.1.2. Parameters.** For both GPMP and GPMP2,  $Q_C$  controls the uncertainty in the prior distribution. A higher value means the trajectories will have a lower cost on deviating from the mean and the distribution covers a wider area of the configuration space. Thus, a higher value 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  $\epsilon = 0.2$  m for both GPMP and GPMP2 for the WAM dataset, and  $\epsilon = 0.05$  m for GPMP and  $\epsilon = 0.08$  m for GPMP2 for the PR2 dataset.

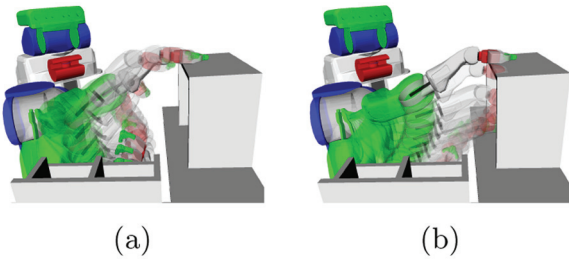
For GPMP2 the “obstacle cost weight”  $\sigma_{obs}$  acts like a weight term that balances smoothness and collision-free requirements on the optimized trajectory and is set based on the application. Smaller  $\sigma_{obs}$  puts more weight on

**Table 1.** Results for 24 planning problems on the 7-DOF WAM arm.

	GPMP2-intp	GPMP2-no-intp	TrajOpt-101	TrajOpt-11	GPMP	CHOMP	RRT-Connect	LBKPIECE
Success (%)	91.7	<b>100.0</b>	91.7	20.8	95.8	75	91.7	62.5
Average time (s)	0.121	0.384	0.313	<b>0.027</b>	0.3	0.695	1.87	6.89
Maximum time (s)	0.367	0.587	0.443	<b>0.033</b>	0.554	2.868	5.18	9.97

**Table 2.** 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	<b>82.3</b>	33.8
Average time (s)	<b>0.11</b>	0.196	0.958	0.191	1.7	2.38	3.02	7.12
Maximum time (s)	<b>0.476</b>	0.581	4.39	0.803	9.08	9.81	9.33	9.95

**Fig. 12.** (a) Successful trajectory with a good selection of  $\sigma_{obs}$ ,  $\sigma_{obs} = 0.005$ . (b) Failure, where the trajectory collides with the top part of the shelf, when  $\sigma_{obs}$  is too large,  $\sigma_{obs} = 0.05$ .

obstacle avoidance and vice versa. Figure 12 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<sup>5</sup> and for the PR2 dataset are summarized in Table 2.<sup>6</sup> 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).

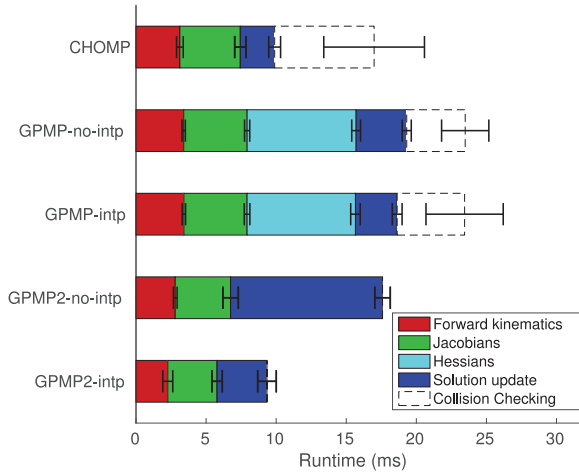
From the results in Table 1 and 2 we see that GPMP2 perform consistently well compared with other algorithms on these datasets. Using interpolation during optimization (GPMP2-intp) achieves 30–50% speedup of average and maximum runtime when compared with not using interpolation (GPMP2-no-intp). On the 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 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 maximum run times, GPMP2 always converges well before the maximum time limit and all the failure cases are due to infeasible local minima. Solutions such as 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 13 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 3 shows the average number of optimization iterations for successful runs in both the WAM and the PR2 datasets. We see that compared with 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, owing to the GP representation and gradients on the augmented

**Table 3.** 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

**Fig. 13.** 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.

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 13 represent computational costs due to 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 owing to its similar construction. Note that the total computational time in Table 1 reflects the total iteration time as shown in Figure 13 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

**Table 4.** Results for 72 replanning problems on WAM.

	iGPMP2	GPMP2
Success (%)	<b>100.0</b>	<b>100.0</b>
Average time (ms)	<b>8.07</b>	65.68
Maximum time (ms)	<b>12.65</b>	148.31

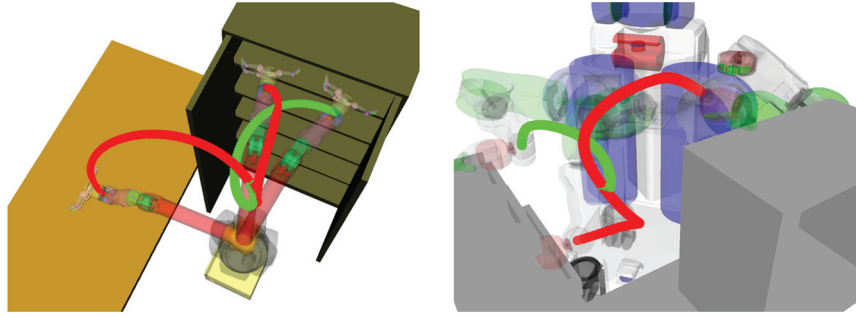
**Table 5.** Results for 54 replanning problems on PR2.

	iGPMP2	GPMP2
Success (%)	66.7	<b>88.9</b>
Average time (ms)	<b>6.17</b>	27.30
Maximum time (ms)	<b>7.37</b>	87.95

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 Tables 4 and 5. We see from the results that iGPMP2 provides an order of magnitude speed-up, but suffers loss in 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 the initialization to incrementally update the trajectory, thus finding the solution much faster. There are three possible explanations why iGPMP2's success rate suffers as compared with the GPMP2's. First, iGPMP2 uses the original trajectory as initialization, which may be a poor choice if the goal has moved significantly. Second, in iSAM2 not every factor is relinearized and updated in Bayes tree for efficiency, which may lead to a poor linear approximation. Finally, 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 current implementation. Since relinearizing factors or reinitializing variables will update the corresponding cliques of the Bayes tree and break its incremental nature, this results in runtime similar to batch optimization, and should not be



**Fig. 14.** 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.

done frequently. A good heuristic is to only perform relinearization/reinitialization when a planning failure is detected. We leave the task of designing a better solution to overcome this issue as future work.

To maximize performance and overcome the deficiencies of iGPMP2, the rule of thumb when using iGPMP2 for replanning is to keep the difference between replanning problems and existing solutions to a minimum. This will lead to better initialization and reduced effect of linearization errors, and thus will improve iGPMP2's success rate. We verify this with the PR2 benchmark, where a smaller distance between original goal configuration and new goal configuration means a smaller difference between the replanning problem and the existing solution. We use  $L2$  distance  $\|\theta_o - \theta_r\|_2$  to quantify the distance between the original goal  $\theta_o$  and the new goal  $\theta_r$ . From the PR2 benchmark, 27 problems have  $\|\theta_o - \theta_r\|_2 < 2.0$ , where iGPMP2 has 81.5% success rate. On the other hand, we see 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 14. 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 SDFs 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 with 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., 2014, 2013) 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 such as changing goals, multi-query planners such as 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 and 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 such as 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., 2014, 2013). We believe that SQP can be integrated into our trajectory optimizer, although this remains for future work.

## 11. Conclusion

We have used GPs 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.

### Funding

This work was partially supported by National Institute of Food and Agriculture, US Department of Agriculture (award number 2014-67021-22556), NSF NRI (award number 1637758), and NSF CRII (award number 1464219).

### ORCID iD

Xinyan Yan  <https://orcid.org/0000-0001-5265-5170>

### Notes

1. Given that the trajectories are represented by GPs, the incremental updates of the factor graphs can also be viewed as incremental GPR (Yan et al., 2017).
2. Available at <https://github.com/gtrl/gpmp2>
3. The hinge loss function is not differentiable at  $d = \epsilon$ , so in our implementation we set  $dc(d)/dd = -0.5$  when  $d = \epsilon$ .
4. A video of the experiments is available at <https://youtu.be/mVA8qhGf7So>.
5. Parameters for benchmark on the WAM dataset: for GPMP and CHOMP,  $\lambda = 0.005$ ,  $\eta = 1$ . For CHOMP,  $\epsilon = 0.2$ . For TrajOpt,  $\text{coeffs} = 20$ ,  $\text{dist\_pen} = 0.05$ .
6. Parameters for benchmark on the PR2 dataset: for CHOMP,  $\epsilon = 0.05$ . All remaining parameters are the same as for the WAM dataset.

### References

- Alonso P-P, Steven L and Gabe S (2015) A spline-based trajectory representation for sensor fusion and rolling shutter cameras. *International Journal of Computer Vision*, 113(3): 208–219.
- Arunkumar B, Byron B, Siddhartha SS, et al. (2014) Space-time functional gradient optimization for motion planning. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 6499–6506. IEEE.
- Carl ER (2006) *Gaussian processes for machine learning*. Cite-seer, 2006.
- Charles B and Ian R (2010) A hybrid SLAM representation for dynamic marine environments. In: *IEEE Conference on Robotics and Automation (ICRA)*, 2010, pp. 257–264. IEEE.
- Chi HT, Paul F and Timothy DB (2012) Gaussian process Gauss-Newton: Non-parametric state estimation. In: *2012 Ninth Conference on Computer and Robot Vision (CRV)*, pp. 206–213. IEEE.
- Chonhyon P, Jia P and Dinesh M (2012) ITOMP: Incremental trajectory optimization for realtime replanning in dynamic environments. In: *ICAPS*, 2012.
- Chonhyon P, Jia P and Dinesh M (2013) Real-time optimization-based planning in dynamic environments using GPUs. In: *2013 IEEE International Conference on Robotics and Automation (ICRA)*, 2013, pp. 4090–4097. IEEE.

- Duy N-T, Jan P, Matthias S, et al. Learning inverse dynamics: a comparison. In: *European Symposium on Artificial Neural Networks*, number EPFL-CONF-175477, 2008.
- Elbanhawi M, Simic M and Jazar R (2015) Randomized bidirectional B-Spline parameterization motion planning. *IEEE Transactions on Intelligent Transportation Systems* 99: 1–1.
- Eric H, Mustafa M, Zhen L, et al. (2017) Motion planning with graph-based trajectories and Gaussian process inference. In: *Proceedings of the 2017 IEEE Conference on Robotics and Automation (ICRA)*, 2017.
- Evangelos T, Yuval T and Emo T (2010) Stochastic differential dynamic programming. In: *American Control Conference (ACC)*, 2010, pp. 1125–1132. IEEE.
- Frank D (2012) *Factor graphs and GTSAM: A hands-on introduction*. Georgia Tech Technical Report, GT-RIM-CP&R-2012-002.
- Frank D and Michael K (2006) Square root SAM: Simultaneous localization and mapping via square root information smoothing. *The International Journal of Robotics Research* 25(12): 1181–1203.
- Frank RK, Brendan JF and Hans-Andrea L (2001) Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory* 47(2): 498–519.
- Gene HG and Charles FVL (2012) *Matrix computations* (vol 3). JHU Press.
- Hagai A (2003) Planning by probabilistic inference. In: *Proceedings of the 9th International Workshop on Artificial Intelligence and Statistics*. Society for Artificial Intelligence and Statistics.
- Hang D and Timothy DB (2014) Lighting-invariant visual odometry using lidar intensity imagery and pose interpolation. In: *Field and Service Robotics (FSR)*, pp. 327–342. Springer.
- Ioan AS, Mark M and Lydia EK (2012) The open motion planning library. *IEEE Robotics & Automation Magazine* 19(4): 72–82.
- Ioan A Şucan and Lydia EK (2009) Kinodynamic motion planning by interior-exterior cell exploration. In: *Algorithmic Foundations of Robotics VIII*, pp. 449–464. Springer.
- Jürgen S, Christian P and Wolfram B (2009) Body schema learning for robotic manipulators from visual self-perception. *Journal of Physiology-Paris* 103(3): 220–231.
- James JK and Steven ML (2000) RRT-connect: An efficient approach to single-query path planning. In *Proceedings of the ICRA'00 IEEE International Conference on Robotics and Automation*, 2000, (Vol. 2, pp. 995–1001). IEEE.
- Jing D, Mustafa M, Frank D, et al. Motion planning as probabilistic inference using Gaussian processes and factor graphs. In: *Proceedings of Robotics: Science and Systems (RSS)*, 2016.
- John S, Jonathan H, Alex L, et al. (2013) Finding locally optimal, collision-free trajectories with sequential convex optimization. In: *Robotics: Science and Systems* (Vol. 9, pp. 1–10). Citeseer.
- John S, Yan D, Jonathan H, et al. (2014) Motion planning with sequential convex optimization and convex collision checking. *The International Journal of Robotics Research* 33(9): 1251–1270.
- Jonathan DG, Siddhartha SS and Timothy DB (2015) Batch informed trees (bit\*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 3067–3074. IEEE.
- Jonathan K and Dieter F (2009) GP-BayesFilters: Bayesian filtering using Gaussian process prediction and observation models. *Autonomous Robots* 27(1): 75–90.
- Keliang H, Elizabeth M and Matt Z (2013) Multigrid CHOMP with local smoothing. In: *Proceedings of 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, 2013.
- Konrad R, Marc T and Sethu V (2012) On stochastic optimal control and reinforcement learning by approximate inference. *Proceedings of Robotics: Science and Systems*.
- Kristian K, Christian P, Patrick P, et al. (2007) Most likely heteroscedastic Gaussian process regression. In: *Proceedings of the 24th International conference on Machine Learning*, pp. 393–400. ACM.
- Lydia EK, Petr S, Latombe J-C, et al. (1996) Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation* 12(4): 566–580.
- Marc D and Carl ER (2011) PILCO: A model-based and data-efficient approach to policy search. In: *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pp. 465–472.
- Marc T (2009) Robot trajectory optimization using approximate inference. In: *Proceedings of the 26th annual international conference on machine learning*, pp. 1049–1056. ACM.
- Marc T and Amos S (2006) Probabilistic inference for solving discrete and continuous state Markov decision processes. In: *Proceedings of the 23rd International Conference on Machine Learning*, pp. 945–952. ACM.
- Marc T and Christian G (2010) A Bayesian view on motor control and planning. In: *From Motor Learning to Interaction Learning in Robots*, pp. 227–252. Springer.
- Matt Z, Nathan R, Anca DD, et al. (2013) CHOMP: Covariant Hamiltonian optimization for motion planning. *The International Journal of Robotics Research* 32(9-10): 1164–1193.
- Maxim L, David IF, Geoffrey JG, et al. (2005) Anytime dynamic A\*: An anytime, replanning algorithm. In: *ICAPS*, pp. 262–271.
- Meng KCT and Christian L (2008) Modelling smooth paths using Gaussian processes. In: *Field and Service Robotics*, pp. 381–390. Springer.
- Michael B and Robert Z (2009) Continuous 3D scan-matching with a spinning 2D laser. In *IEEE Conference on Robotics and Automation (ICRA)*, 2009, pp. 4312–4319. IEEE.
- Michael K, Ananth R and Frank D (2008) iSAM: Incremental smoothing and mapping. *IEEE Transactions on Robotics* 24(6): 1365–1378.
- Michael K, Hordur J, Richard R, et al. (2011) iSAM2: Incremental smoothing and mapping using the Bayes tree. *The International Journal of Robotics Research*, page 0278364911430419.
- Michael K, Violela I, Richard R, et al. (2011) The Bayes tree: An algorithmic foundation for probabilistic robot mapping. In: *Algorithmic Foundations of Robotics IX*, pp. 157–173. Springer.
- Mingyang L, Byung HK and Anastasios IM (2013) Real-time motion tracking on a cellphone using inertial sensing and a rolling-shutter camera. In: *IEEE International Conference on Robotics and Automation (ICRA)*, IEEE.
- Mrinal K, Sachin C, Evangelos T, et al. (2011) STOMP: Stochastic trajectory optimization for motion planning. In: *2011 IEEE International Conference on Robotics and Automation (ICRA)*, 2011, pp. 4569–4574. IEEE.
- Muhammad AR, Mustafa M, Seyed RA, et al. (2017) Towards robust skill generalization: Unifying learning from demonstration and motion planning. In: *Proceedings of the 1st Annual*



- Conference on Robot Learning, 13–15 November 2017, (Vol. 78, pp. 109–118). PMLR.
- Mustafa M (2017) *PIPER* [Online] Available at <https://github.com/gtrll/piper>.
- Mustafa M, Ching-An C, Xinyan Y, et al. (2017) Approximately optimal continuous-time motion planning and control via probabilistic inference. In: *Proceedings of the 2017 IEEE Conference on Robotics and Automation (ICRA)*, 2017.
- Mustafa M, Jing D, Frank D, et al. (2017) Simultaneous trajectory estimation and planning via probabilistic inference. In: *Proceedings of Robotics: Science and Systems (RSS)*, 2017.
- Mustafa M, Xinyan Y and Byron B (2016) Gaussian process motion planning. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*, May 2016, pp. 9–15.
- Nathan R, Matthew Z, Bagnell JA, et al. (2009) CHOMP: Gradient optimization techniques for efficient motion planning. In *IEEE International Conference on Robotics and Automation, 2009. ICRA'09*. pp. 489–494. IEEE.
- Paul F, Chi HT, Timothy DB, et al. (2015) Continuous-time batch trajectory estimation using temporal basis functions. *The International Journal of Robotics Research* 34(14): 1688–1710.
- Paul F, Joern R and Roland S (2013) Unified temporal and spatial calibration for multisensory systems. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE.
- Richard C and David H (1966) *Methods of mathematical physics*, Volume 1. CUP Archive.
- Robert GC, Philip D, Steffen LL, et al. (2006) *Probabilistic networks and expert systems: Exact computational methods for Bayesian networks*. Springer Science & Business Media.
- Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 3067–3074. IEEE.
- Sean A and Timothy DB (2013) Towards relative continuous-time SLAM. In: *IEEE Conference on Robotics and Automation (ICRA)*. IEEE.
- Sean A, Frank D and Timothy B (2014) A hierarchical wavelet decomposition for continuous-time SLAM. In: *IEEE Conference on Robotics and Automation (ICRA)*, Hong Kong, China, 31 May–7 June 2014. IEEE.
- Sean A, Timothy DB, Chi HT, et al. (2015) Batch nonlinear continuous-time trajectory estimation as exactly sparse gaussian process regression. *Autonomous Robots* 39(3): 221–238.
- Sean Q (1994) *Real-time Modification of Collision-free Paths*. PhD Thesis, Stanford University.
- Sergey L and Vladlen K (2013) Variational policy search via trajectory optimization. In: *Advances in Neural Information Processing Systems*, pp. 207–215.
- Sertac K and Emilio F (2010) Incremental sampling-based algorithms for optimal motion planning. *Robotics Science and Systems VI*, 104.
- Sethu V, Aaron D and Stefan S (2005) Incremental online learning in high dimensions. *Neural Computation* 17(12): 2602–2634.
- Simo S, Arno S and Jouni H (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.
- Stefan L, Simon L, Michael B, et al. Keyframe-based visual-inertial odometry using nonlinear optimization. *International Journal of Robotics Research*, 34(3): 314–334.
- Steven ML (2006) *Planning Algorithms*. Cambridge University Press.
- Sven K, Craig T and Yuri S (2003) Performance bounds for planning in unknown terrain. *Artificial Intelligence* 147(1): 253–279.
- Tim B, Chi HT and Simo S (2014) Batch continuous-time trajectory estimation as exactly sparse Gaussian process regression. In: *Proceedings of Robotics: Science and Systems*, Berkeley, USA, 2014.
- Xinyan Y, Vadim I and Byron B (2017) Incremental sparse GP regression for continuous-time trajectory estimation and mapping. *Robotics and Autonomous Systems* 87: 120–132.
- Zita M, Anca D, Arunkumar B, et al. (2016) Functional gradient motion planning in reproducing kernel Hilbert spaces. In: *Proceedings of Robotics: Science and Systems (RSS)*, 2016.

## 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

$$Y = C\theta + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \tilde{\mathcal{K}}_y) \quad (69)$$

We can write their joint distribution as

$$\mathcal{N}\left(\begin{bmatrix} \tilde{\mu} \\ C\tilde{\mu} \end{bmatrix}, \begin{bmatrix} \tilde{\mathcal{K}} & \tilde{\mathcal{K}}C^T \\ C\tilde{\mathcal{K}} & C\tilde{\mathcal{K}}C^T + \tilde{\mathcal{K}}_y \end{bmatrix}\right) \quad (70)$$

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

$$\mu = \tilde{\mu} + \tilde{\mathcal{K}}C^T(C\tilde{\mathcal{K}}C^T + \tilde{\mathcal{K}}_y)^{-1}(Y - C\tilde{\mu}) \quad (71)$$

$$\mathcal{K} = \tilde{\mathcal{K}} - \tilde{\mathcal{K}}C^T(C\tilde{\mathcal{K}}C^T + \tilde{\mathcal{K}}_y)^{-1}C\tilde{\mathcal{K}} \quad (72)$$

Now, we are interested in conditioning just on the goal state  $\theta_N$  with mean  $\mu_N$  and covariance  $\mathcal{K}_N$ . Therefore, in the above equations we use  $C = [0 \ \dots \ 0 \ \mathbf{I}]$  and  $\tilde{\mathcal{K}}_y = \mathcal{K}_N$  to obtain

$$\mu = \tilde{\mu} + \tilde{\mathcal{K}}(t_N, \mathbf{t})^T(\tilde{\mathcal{K}}(t_N, t_N) + \mathcal{K}_N)^{-1}(\theta_N - \mu_N) \quad (73)$$

$$\mathcal{K} = \tilde{\mathcal{K}} - \tilde{\mathcal{K}}(t_N, \mathbf{t})^T(\tilde{\mathcal{K}}(t_N, t_N) + \mathcal{K}_N)^{-1}\tilde{\mathcal{K}}(t_N, \mathbf{t}) \quad (74)$$

where  $\tilde{\mathcal{K}}(t_N, t) = [\tilde{\mathcal{K}}(t_N, t_0) \ \dots \ \tilde{\mathcal{K}}(t_N, t_N)]$ .

Using the Woodbury matrix identity we can write Equation (72) as

$$\mathcal{K} = (\tilde{\mathcal{K}}^{-1} + C^T\tilde{\mathcal{K}}_y^{-1}C)^{-1} \quad (75)$$

and substituting  $C$  and  $\tilde{\mathcal{K}}_y$  as before for conditioning on the goal we obtain

$$\mathcal{K} = \left(\tilde{\mathcal{K}}^{-1} + \begin{bmatrix} 0 & \dots & 0 & \mathbf{I} \end{bmatrix}^T \mathcal{K}_N^{-1} \begin{bmatrix} 0 & \dots & 0 & \mathbf{I} \end{bmatrix}\right)^{-1} \quad (76)$$

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

$$\mathcal{K}^{-1} = \begin{bmatrix} \tilde{\mathbf{A}}^{-1} & & \\ \mathbf{0} & \dots & \mathbf{0} & \mathbf{I} \end{bmatrix}^T \begin{bmatrix} \tilde{\mathbf{Q}}^{-1} & \\ & \mathcal{K}_N^{-1} \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{A}}^{-1} & & \\ \mathbf{0} & \dots & \mathbf{0} & \mathbf{I} \end{bmatrix} \quad (77)$$

$$= \mathbf{B}^T \mathbf{Q}^{-1} \mathbf{B} \quad (78)$$

where

$$\mathbf{B} = \begin{bmatrix} \mathbf{I} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ -\Phi(t_1, t_0) & \mathbf{I} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & -\Phi(t_2, t_1) & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \dots & -\Phi(t_N, t_{N-1}) & \mathbf{I} \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{I} \end{bmatrix} \quad (79)$$

and

$$\mathbf{Q}^{-1} = \text{diag}(\mathcal{K}_0^{-1}, \mathbf{Q}_{0,1}^{-1}, \dots, \mathbf{Q}_{N-1,N}^{-1}, \mathcal{K}_N^{-1}), \quad (80)$$

$$\mathbf{Q}_{a,b} = \int_{t_a}^{t_b} \Phi(b, s) \mathbf{F}(s) \mathbf{Q}_c \mathbf{F}(s)^T \Phi(b, s)^T ds \quad (81)$$

## Appendix B. Sparsity of the likelihood in GMP2

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

Given the isotropic definition of  $\Sigma_{obs}$  in Equations (47) and (66)

$$\mathbf{H}^T \Sigma_{obs}^{-1} \mathbf{H} = \sigma_{obs}^{-2} \mathbf{H}^T \mathbf{H} \quad (82)$$

Given the definition of  $\mathbf{h}(\theta)$  in Equation (46), the size of  $\mathbf{H}$  is  $M \times (N+1 + N \times n_{ip})$  by  $(N+1) \times D$ , therefore  $\mathbf{H}^T \mathbf{H}$  has size  $(N+1) \times D$  by  $(N+1) \times D$ .

For simplicity, we partition  $\mathbf{H}$  and  $\mathbf{H}^T \mathbf{H}$  by forming blocks corresponding to the system DOF  $D$ , and dimensionality  $M$  of  $\mathbf{h}$ , and work with these block matrices in the

remaining section. Thus,  $\mathbf{H}$  and  $\mathbf{H}^T \mathbf{H}$  have block-wise size  $N+1 + N \times n_{ip}$  by  $N+1$  and  $N+1$  by  $N+1$ , respectively. We define  $\mathbf{A}(i, j)$  to be the block element at row  $i$  and column  $j$  of  $\mathbf{A}$ .

Given the definition of  $\mathbf{h}(\theta)$  in Equation (46), each element of  $\mathbf{H}$  is defined by

$$\mathbf{H}(i, j) = \frac{\partial \mathbf{h}(\theta_{s_i})}{\partial \theta_j} \Big|_{\theta} \quad (83)$$

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

$$\mathbf{H}(i, j) = \frac{\partial \mathbf{h}^{interp}(\theta_{s_i}, \theta_{s_i+1})}{\partial \theta_j} \Big|_{\theta} \quad (84)$$

for rows containing interpolated obstacle factors, where  $s_i$  is the before support state index of the interpolated obstacle factor of row  $i$ . Since  $\mathbf{h}(\theta_{s_i})$  is only a function of  $\theta_{s_i}$ , and  $\mathbf{h}^{interp}(\theta_{s_i}, \theta_{s_i+1})$  is only a 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  $\mathbf{H}$

$$\mathbf{H}(i, j) = \mathbf{0}, \text{ if } j \neq s_i \text{ or } s_i + 1 \quad (85)$$

For each block element in  $\mathbf{H}^T \mathbf{H}$

$$\mathbf{H}^T \mathbf{H}(i, j) = \sum_{k=1}^{N+1+N \times n_{ip}} \mathbf{H}^T(i, k) \mathbf{H}(k, j) \quad (86)$$

$$= \sum_{k=1}^{N+1+N \times n_{ip}} \mathbf{H}(k, i)^T \mathbf{H}(k, j) \quad (87)$$

For each  $k$ , non-zero  $\mathbf{H}(k, i)^T \mathbf{H}(k, j)$  is possible when the following condition is satisfied,

$$\{i = s_k \text{ or } s_k + 1\} \text{ and } \{j = s_k \text{ or } s_k + 1\} \quad (88)$$

Thus, for non-zero  $\mathbf{H}^T \mathbf{H}(i, j)$

$$|i - j| \leq 1 \quad (89)$$

since if the difference between  $i$  and  $j$  is larger than 1, Equation (88) is unsatisfied on every  $k$ , so  $\mathbf{H}^T \mathbf{H}(i, j)$  will be zero based on Equation (87). Given we know that  $\mathbf{H}^T \mathbf{H}$  is block tridiagonal, and following Equation (82), we have proved that  $\mathbf{H}^T \Sigma_{obs}^{-1} \mathbf{H}$  is also block tridiagonal.