Gaussian Process Motion Planning

Mustafa Mukadam, Xinyan Yan, and Byron Boots

***Abstract*— Motion planning is a fundamental tool in robotics, used to generate collision-free, smooth, trajectories, while satisfying task-dependent constraints. In this paper, we present a novel approach to motion planning using Gaussian processes. In contrast to most existing trajectory optimization algorithms, which rely on a discrete state parameterization in practice, we represent the continuous-time trajectory as a sample from a Gaussian process (GP) generated by a linear time-varying stochastic differential equation. We then provide a gradient-based optimization technique that optimizes continuous-time trajectories with respect to a cost functional. By exploiting GP interpolation, we develop the Gaussian Process Motion Planner (GPMP), that finds optimal trajectories parameterized by a small number of states. We benchmark our algorithm against recent trajectory optimization algorithms by solving 7-DOF robotic arm planning problems in simulation and validate our approach on a real 7-DOF WAM arm.**

I. Introduction & Related Work

Motion planning is a fundamental skill for robots that aspire to move through an environment without collision or manipulate objects to achieve some goal. We consider motion planning from a trajectory optimization perspective, where we seek to find a trajectory that minimizes a given cost function while satisfying any given constraints.

A significant amount of previous work has focused on trajectory optimization and related problems. Khatib [1] pioneered the idea of potential field where the goal position is an attractive pole and the obstacles form repulsive fields. Various extensions have been proposed to address problems like local minimum [2], and ways of modeling the free space [3]. Covariant Hamiltonian Optimization for Motion Planning (CHOMP) [4], [5] utilizes covariant gradient descent to minimize obstacle and smoothness cost functionals, and a precomputed signed distance field for fast collision checking. STOMP [6] is a stochastic trajectory optimization method that can work with non-differentiable constraints by sampling a series of noisy trajectories. An important drawback of CHOMP and STOMP is that a large number of trajectory states are needed to reason about small obstacles or find feasible solutions when there are many constraints. Multigrid CHOMP [7] attempts to reduce the computation time of CHOMP when using a large number of states by starting with a low-resolution trajectory and gradually increasing the resolution at each iteration. Finally, TrajOpt [8] formulates motion planning as sequential quadratic programming. Swept volumes are considered to ensure continuous-time safety, enabling TrajOpt to solve more problems with fewer states.

In this paper, we present a novel Gaussian process (GP) approach to motion planning. Although, Gaussian processes [9] have been used for function approximation in many areas of robotics including supervised learning [10], [11], inverse dynamics modeling [12], [13], reinforcement learning [14], path prediction [15], simultaneous localization and mapping [16], [17], state estimation [18]–[20], and controls [21], to our knowledge they have not been used for motion planning before.

GPs provide a very natural way to model motion planning problems with several advantages over previous approaches. Vector-valued GPs provide a principled way to represent continuous-time trajectories as *functions* that map time to robot states. Smooth trajectories can be represented compactly with only a small number of states, and Gaussian process regression can be used to query the state of the robot at any time of interest. Using this insight, we develop the Gaussian Process Motion Planner (GPMP), a new motion planning algorithm that optimizes trajectories parameterized by a small number of states, using Gaussian process interpolation and gradient-based optimization. We evaluate our algorithm both in simulation and on a 7-DOF Barrett WAM arm, and we benchmark our approach against several recent trajectory optimization algorithms.

II. A Gaussian Process Model for Continuous-time Trajectories

Motion planning traditionally involves finding a smooth, collision-free trajectory from a start state to a goal state. Similar to previous approaches [4]–[8], we treat motion planning as an optimization problem and search for a trajectory that minimizes a predefined cost function (Section III). In contrast to these previous approaches, which consider finely discretized discrete time trajectories in practice, we consider *continuous-time* trajectories such that the state at time t is

$$\xi(t) = \{\xi^d(t)\}_{d=1}^{D} \quad \xi^d(t) = \{\xi^{dr}(t)\}_{r=1}^{R} \tag{1}$$

where D is the dimension of the configuration space (number of joints), and R is the number of variables in each configuration dimension. Here we choose $R = 3$, specifying joint *positions, velocities, and accelerations*, ensuring that our state is Markovian. Using the Markov property of the state in the motion prior, defined in Eq. 6 below, allows us to build an exactly sparse inverse kernel (precision matrix) [16], [22] useful for efficient optimization and fast GP interpolation (Section II-B).

Mustafa Mukadam, Xinyan Yan, and Byron Boots are affiliated with the Institute for Robotics and Intelligent Machines, Georgia Institute of Technology, Atlanta, GA 30332, USA. mmukadam3@gatech.edu {xinyan.yan,bboots}@cc.gatech.edu. This work was funded in part by the National Science Foundation, grant number IIS-1464219.

Continuous-time trajectories are assumed to be sampled from a *vector-valued* Gaussian process (GP):

$$\xi(t) \sim \mathcal{GP}(\mu(t), \mathcal{K}(t,t')) \quad t_0 < t, t' < t_{N+1} \tag{2}$$

where $\mu(t)$ is a vector-valued mean function whose components are the functions $\{\mu^{dr}(t)\}_{d=1,r=1}^{D,R}$ and the entries $\mathcal{K}(t,t')_{dr,d'r'}$ in the matrix-valued covariance function $\mathcal{K}(t,t')$ corresponding to the covariances between $\xi^{dr}(t)$ and $\xi^{d'r'}(t')$. Based on the definition of vector-valued Gaussian processes [23], any collection of N states on the trajectory has a joint Gaussian distribution,

$$\begin{aligned} \xi \sim \mathcal{N}(\mu, \mathcal{K}), \quad \xi = \xi_{1:N}, \quad \mu = \mu_{1:N}, \\ \mathcal{K}_{ij} = \mathcal{K}(t_i, t_j), \quad t_1 \leq t_i, t_j \leq t_N \end{aligned} \tag{3}$$

The start and goal states, ξ_0 and ξ_{N+1}, are excluded because they are fixed. ξ_i denotes the state at time t_i.

Reasoning about trajectories with GPs has several advantages. The Gaussian process imposes a prior distribution on the space of trajectories. Given a fixed set of time-parameterized states, we can condition the GP on those states to compute the posterior mean at any time of interest:

$$\bar{\xi}(\tau) = \mu(\tau) + \mathcal{K}(\tau)\mathcal{K}^{-1}(\xi - \mu) \tag{4}$$

$$\mathcal{K}(\tau) = [\ \mathcal{K}(\tau, t_1) \quad \dots \quad \mathcal{K}(\tau, t_N)] \tag{5}$$

A consequence of GP interpolation is that the trajectory does not need to be discretized at a high resolution, and trajectory states do not need to be evenly spaced in time. Additionally, the posterior mean is the *maximum a posteriori* interpolation of the trajectory, and high-frequency interpolation can be used to generate an executable trajectory or reason about the cost over the entire trajectory instead of just at the states.

A. Gaussian Processes Generated by Linear Time-Varying Stochastic Differential Equations

The choice of prior is an important design consideration when working with Gaussian processes. In this work, we consider the Gaussian processes generated by the linear time varying (LTV) stochastic differential equations (SDEs) [16]:

$$\begin{aligned} &\xi'(t) = A(t)\xi(t) + F(t)w(t) \\ &w(t) \sim \mathcal{GP}(0,\, Q_c\delta(t-t')), \quad t_0 < t, t' < t_{N+1} \end{aligned} \tag{6}$$

where $A(t)$ are $F(t)$ are time-varying system matrices, $\delta(\cdot)$ is the Dirac delta function, $w(t)$ is the process noise modeled by a Gaussian process with zero mean, and Q_c is a power-spectral density matrix, a hyperparameter that can be set using data collected on the system [16].

Given the *fixed start state* ξ_0 as the initial condition, and $\Phi(t,s)$, the state transition matrix that propagates the state from time s to t, the solution to the SDE is:

$$\widetilde{\xi}(t) = \Phi(t,t_0)\xi_0 + \int_{t_0}^{t} \Phi(t,s)F(s)w(s)ds \tag{7}$$

$\widetilde{\xi}(t)$ corresponds to a Gaussian process. The mean function and the covariance function can be obtained from the first and second moments:

$$\widetilde{\mu}(t) = \Phi(t,t_0)\xi_0 \tag{8}$$

$$\begin{aligned} \widetilde{\mathcal{K}}(t,t') &= E[(\widetilde{\xi}(t) - \widetilde{\mu}(t))(\widetilde{\xi}(t') - \widetilde{\mu}(t'))^T] \\ &= \int_{t_0}^{\min(t,t')} \Phi(t,s)F(s)Q_cF(s)^T\Phi(t',s)^T ds \end{aligned} \tag{9}$$

After considering the end state ξ_{N+1} as a noise-free "measurement", and conditioning $\widetilde{\xi}(t)$ on this measurement, we derive the Gaussian process for the trajectories with *fixed* start and end states. The mean and covariance functions are:

$$\begin{aligned} \mu(t) &= \widetilde{\mu}_t + \widetilde{\mathcal{K}}_{N+1,t}\widetilde{\mathcal{K}}_{N+1,N+1}^{-1}(\xi_{N+1} - \widetilde{\mu}_{N+1}) \\ &= \Phi_{t,0}\xi_0 + W_t\Phi_{N+1,t}^{\intercal}W_{N+1}^{-1}(\xi_{N+1} - \Phi_{N+1,0}\xi_0) \end{aligned} \tag{10}$$

$$\begin{aligned} \mathcal{K}(t,t') &= \widetilde{\mathcal{K}}_{t,t'} - \widetilde{\mathcal{K}}_{N+1,t}^{\intercal}\widetilde{\mathcal{K}}_{N+1,N+1}^{-1}\widetilde{\mathcal{K}}_{N+1,t'}, \quad t < t' \\ &= W_t\Phi_{t',t}^{\intercal} - W_t\Phi_{N+1,t}^{\intercal}W_{N+1}^{-1}\Phi_{N+1,t'}W_{t'}, \quad t < t' \end{aligned} \tag{11}$$

where

$$W_t = \int_{t_0}^{t} \Phi(t,s)F(s)Q_cF(s)^T\Phi(t',s)^T ds \tag{12}$$

The inverse kernel matrix $\mathcal{K}^{-1}$ for the N states is exactly sparse (block-tridiagonal), and can be factored as:

$$\mathcal{K}^{-1} = B^T Q^{-1} B \tag{13}$$

where

$$B = \begin{bmatrix} 1 & 0 & \dots & 0 \\ -\Phi(t_1,t_0) & 1 & \dots & 0 \\ 0 & -\Phi(t_2,t_1) & \ddots & \vdots \\ \vdots & \vdots & \ddots & 1 \\ 0 & 0 & \dots & -\Phi(t_N,t_{N-1}) \end{bmatrix} \tag{14}$$

and

$$Q = \text{diag}(Q_1, ..., Q_{N+1}),$$

$$Q_i = \int_{t_{i-1}}^{t_i} \Phi(t_i,s)F(s)Q_cF(s)^T\Phi(t_i,s)^T ds$$

Here Q^{-1} is block diagonal, and B is band diagonal.

The inverse kernel matrix, $\mathcal{K}^{-1}$, encodes constraints on consecutive states imposed by the state transition matrix, $\Phi(t,s)$. If state $\xi(t)$ includes joint angles, velocities, and accelerations, then the constraints are on all of these variables. If we take this state transition matrix and Q^{-1} to be unit scalars, then $\mathcal{K}^{-1}$ reduces to the matrix A formed by finite differencing in CHOMP [4].

B. Fast Gaussian Process Interpolation

An advantage of Gaussian process trajectory estimation is that any point on a trajectory can be interpolated from other points by computing the posterior mean (Eq. 4). In the case of a GP generated from the LTV-SDE in Sec. II-A, interpolation is very fast, involving only two nearby states and is performed in $O(1)$ time [16]:

$$\bar{\xi}(\tau) = \mu(\tau) + [\ \Lambda(\tau)\ \Psi(\tau)\ ]\left(\begin{bmatrix} \xi_i \\ \xi_{i+1} \end{bmatrix} - \begin{bmatrix} \mu_i \\ \mu_{i+1} \end{bmatrix}\right) \tag{15}$$

where $t_i < \tau < t_{i+1},\ i = 0, ..., N$. This is because only the $(i)^{th}$ and $(i+1)^{th}$ block columns of $\mathcal{K}(\tau)\mathcal{K}^{-1}$ are non-zero:

$$\mathcal{K}(\tau)\mathcal{K}^{-1} = [\ 0\ \dots\ 0\ \Lambda(\tau)\ \Psi(\tau)\ 0\ \dots\ 0\ ] \tag{16}$$

where $\Lambda(\tau) = \Phi(\tau, t_i) - \Psi(\tau)\Phi(t_{i+1}, t_i)$, and $\Psi(\tau) = Q_\tau \Phi(\tau, t_i)^T Q_{i+1}^{-1}$. We make use of this fast interpolation during optimization (Section III-D).

III. Continuous-Time Motion Planning with Gaussian Processes

Motion planning can be viewed as the following problem:

$$\begin{aligned} &\text{minimize} \quad \mathcal{U}[\xi] \\ &\text{subject to} \quad f_i[\xi] \leq 0,\ i = 1, ..., m \end{aligned} \tag{17}$$

where the trajectory ξ is a continuous-time function with start and end states fixed, $\mathcal{U}[\xi]$ is an objective or cost functional that evaluates the quality of a trajectory, and $f_i[\xi]$ are constraint functionals such as joint limits and task-dependent constraints. For simplicity of notation, we use ξ to represent either the continuous-time function or the N states that parameterize the function (Eq. 3). Its meaning should be clear from context. Functionals take their function arguments in *brackets*, for example $\mathcal{U}[\xi]$.

A. Cost Functionals

The usual goal of trajectory optimization is to find trajectories that are smooth and collision free. Therefore, we define the objective functional as a combination of two objectives: a prior functional, $\mathcal{F}_{gp}$, determined by the Gaussian process prior that penalizes the displacement of the trajectory from the prior mean to keep the trajectory smooth, and an obstacle cost functional, $\mathcal{F}_{obs}$, that penalizes collision with obstacles. Thus the objective functional is

$$\mathcal{U}[\xi] = \mathcal{F}_{obs}[\xi] + \lambda \mathcal{F}_{gp}[\xi] \tag{18}$$

where the tradeoff between the two functionals is controlled by λ. This objective functional looks similar to the one used in CHOMP [4], however, in our case the trajectory also contains velocities and accelerations. Another important distinction is that the smoothness cost in CHOMP is calculated from finite dynamics while here it is generated from the GP prior.

The prior functional $\mathcal{F}_{gp}$ is induced by the GP formulation in Eq. 2:

$$\mathcal{F}_{gp}[\xi] = \frac{1}{2}\|\xi - \mu\|_{\mathcal{K}}^2 \tag{19}$$

where the norm is the Mahalanobis norm: $\|e\|_\Sigma^2 = e^\intercal \Sigma^{-1} e$.

We use an obstacle cost functional $\mathcal{F}_{obs}$ similar to the one used in CHOMP [4], [5], that 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:

$$\begin{aligned} \mathcal{F}_{obs}[\xi] &= \int_{\mathcal{B}} \int_{t_0}^{t_{N+1}} c(x)||x'||\ dt\ du \\ &= \int_{t_0}^{t_{N+1}} \int_{\mathcal{B}} c(x)||x'||\ du\ dt \end{aligned} \tag{20}$$

where $\mathcal{B} \subset \mathbb{R}^3$ is the set of points on the robot body, $x(\cdot) : \mathcal{C} \times \mathcal{B} \to \mathbb{R}^3$ is the forward kinematics that maps robot configuration q to workspace, and $c(\cdot) : \mathbb{R}^3 \to \mathbb{R}$ is the workspace cost function that penalizes the body points when they are in or around an obstacle. In practice, this is calculated using a precomputed signed distance field. x' is the velocity of a body point in workspace. Multiplying the norm of the velocity to the cost in the line integral above gives an arc-length parameterization that is invariant to re-timing the trajectory. Intuitively, this encourages trajectories to minimize cost by circumventing obstacles rather than attempting to pass through them very quickly.

B. Optimization

Optimizing the objective functional $\mathcal{U}[\xi]$ in Eq. 18 can be quite difficult since the cost $\mathcal{F}_{obs}$ is non-convex. We adopt an iterative, gradient-based approach to minimize $\mathcal{U}[\xi]$. At each iteration we form an approximation to the cost functional via a Taylor series expansion around the current trajectory $\bar{\xi}$:

$$\mathcal{U}[\xi] \approx \mathcal{U}[\bar{\xi}] + \bar{\nabla}\mathcal{U}[\bar{\xi}](\xi - \bar{\xi}) \tag{21}$$

We then minimize the approximate cost while constraining the trajectory to be close to the previous one. The optimal perturbation $\delta\xi^*$ to the trajectory is:

$$\begin{aligned} \delta\xi^* &= \underset{\delta\xi}{\operatorname{argmin}} \left\{ \mathcal{U}[\bar{\xi}] + \bar{\nabla}\mathcal{U}[\bar{\xi}](\xi - \bar{\xi}) + \frac{\eta}{2}\|\xi - \bar{\xi}\|_{\mathcal{K}}^2 \right\} \\ &= \underset{\delta\xi}{\operatorname{argmin}} \left\{ \mathcal{U}[\bar{\xi}] + \bar{\nabla}\mathcal{U}[\bar{\xi}]\delta\xi + \frac{\eta}{2}\|\delta\xi\|_{\mathcal{K}}^2 \right\} \end{aligned} \tag{22}$$

where η is the regularization constant. Eq. 22 can be solved by differentiating the right-hand side and setting the result to zero:

$$\bar{\nabla}\mathcal{U}[\bar{\xi}] + \eta\mathcal{K}^{-1}\delta\xi^* = 0 \quad \Longrightarrow \quad \delta\xi^* = -\frac{1}{\eta}\mathcal{K}\bar{\nabla}\mathcal{U}[\bar{\xi}] \tag{23}$$

At each iteration, the update rule is therefore

$$\bar{\xi} \leftarrow \bar{\xi} + \delta\xi^* = \bar{\xi} - \frac{1}{\eta}\mathcal{K}\bar{\nabla}\mathcal{U}[\bar{\xi}] \tag{24}$$

To complete the update, we need to compute the gradient of the cost functional $\bar{\nabla}\mathcal{U}[\xi]$ at the current trajectory $\bar{\xi}$, which requires finding $\bar{\nabla}\mathcal{F}_{gp}$ and $\bar{\nabla}\mathcal{F}_{obs}$:

$$\bar{\nabla}\mathcal{U}[\xi] = \bar{\nabla}\mathcal{F}_{obs} + \lambda\bar{\nabla}\mathcal{F}_{gp} \tag{25}$$

We discuss the two component gradients below.

1) Gradient of the Prior Cost: The cost functional $\mathcal{F}_{gp}$ can be viewed as penalizing trajectories that deviate from a prescribed motion model. An example is given in Eq. 37 in the Experimental Results (Section IV). To compute the gradient of the prior cost, we first expand Eq. 19:

$$\begin{aligned} \mathcal{F}_{gp}[\xi] &= \frac{1}{2}(\xi^T - \mu^T)\mathcal{K}^{-1}(\xi - \mu) \\ &= \frac{1}{2}\xi^T\mathcal{K}^{-1}\xi - \mu^T\mathcal{K}^{-1}\xi + \frac{1}{2}\mu^T\mathcal{K}^{-1}\mu \end{aligned} \tag{26}$$

and then take the derivative with respect to the trajectory ξ

$$\bar{\nabla}\mathcal{F}_{gp}[\xi] = \mathcal{K}^{-1}\xi - \mathcal{K}^{-1}\mu = \mathcal{K}^{-1}(\xi - \mu) \tag{27}$$

2) Gradient of Obstacle Cost: The functional gradient of the obstacle cost $\mathcal{F}_{obs}$ in Eq. 20 can be computed from the Euler-Lagrange equation [24] in which a functional of the form $\mathcal{F}[\xi] = \int v(\xi)dt$ yields a gradient

$$\bar{\nabla}\mathcal{F}[\xi] = \frac{\partial v}{\partial \xi} - \frac{d}{dt}\frac{\partial v}{\partial \xi'} \tag{28}$$

Applying Eq. 28 to find the gradient of Eq. 20 in the workspace and then mapping it back to the configuration space via the kinematic Jacobian [25], and following the proof in [26], we compute the gradient with respect to position, velocity, and acceleration as

$$\bar{\nabla}\mathcal{F}_{obs}[\xi] = \begin{bmatrix} \int_{\mathcal{B}} J^T ||x'|| \big[(I - \hat{x}'\hat{x}'^T)\nabla c - c\kappa\big] du \\ \int_{\mathcal{B}} J^T c\, \hat{x}' du \\ 0 \end{bmatrix} \tag{29}$$

where $\kappa = ||x'||^{-2}(I - \hat{x}'\hat{x}'^T)x''$ is the curvature vector along workspace trajectory traced by a body point, x', x'' are the velocity and acceleration of that body point determined by forward kinematics and the Hessian, $\hat{x}' = x'/||x'||$ is the normalized velocity vector, and J is the kinematic Jacobian of that body point. CHOMP uses finite differencing to calculate x' and x'', but with our representation of the state (Eq. 1), x' and x'' can be obtained directly from the Jacobian and Hessian respectively, since velocities and accelerations in configuration space are a part of the state.

C. Update Rule

In order to compute the gradient of prior and obstacle functionals, we parameterize the continuous-time trajectory by a small, N number of states, $\xi = \xi_{1:N}$. If we consider GPs generated by LTV-SDEs, as in Sec. II-A, then the GP prior cost $\mathcal{F}_{gp}(\xi)$ can be computed from the observed states based on the expression of the inverse kernel matrix in Eq. 13. Using B in Eq. 14, and $\mu(t)$ in Eq. 10:

$$\begin{aligned} \mathcal{F}_{gp}(\xi) &= \frac{1}{2}\|\xi - \mu\|^2_{\mathcal{K}} = \frac{1}{2}\|B(\xi - \mu)\|^2_{Q} \\ &= \frac{1}{2}\sum_{i=1}^{N} \|\Phi(t_i, t_{i-1})\xi_{i-1} - \xi_i\|^2_{Q_i} \end{aligned} \tag{30}$$

$$\bar{\nabla}\mathcal{F}_{gp}(\xi) = \mathcal{K}^{-1}(\xi - \mu) = B^T Q^{-1} B(\xi - \mu) \tag{31}$$

The gradient of the obstacle cost is calculated at the N states. In practice $\bar{\nabla}F_{obs}(\xi)$ is further approximated by summing up the gradients with regard to a finite number of body points of the robot $\mathcal{B}' \subset \mathcal{B}$. So the obstacle gradient becomes:

$$\bar{\nabla}_{\xi_i}\mathcal{F}_{obs}(\xi) = \begin{bmatrix} \sum'_{\mathcal{B}} J_i^T ||x'_i|| \big[(I - \hat{x}'_i\hat{x}'^T_i)\nabla c_i - c_i\kappa_i\big] du \\ \sum'_{\mathcal{B}} J_i^T c_i\, \hat{x}'_i du \\ 0 \end{bmatrix} \tag{32}$$

where i denotes the value of the symbol at time t_i.

The update rule for states can be summarized from Eqs. 24, 25, and 27 as:

$$\bar{\xi} \leftarrow \bar{\xi} - \frac{1}{\eta}\mathcal{K}\Big[\lambda\mathcal{K}^{-1}(\bar{\xi} - \mu) + g\Big] \tag{33}$$

where g is a vector of obstacle gradients at each state (Eq.32). Note that this update rule can be interpreted as a generalization of the update rule for CHOMP, but with a trajectory of states that have been augmented to include velocities and accelerations. Thus, we will refer to this algorithm as Augmented CHOMP (AugCHOMP). In the next sub-section we fully leverage the GP prior and GP interpolation machinery to generalize the motion planning algorithm further.

D. Compact Trajectory Representations and Faster Updates via Gaussian Process Interpolation

A key benefit of Gaussian process motion planning over fixed, discrete state parameterizations of trajectories, is that we can use the GP to query the trajectory function at *any* time of interest (see Section II-B). To interpolate p points between two states i and $i+1$, we define two aggregated matrices:

$$\begin{aligned} \Lambda_i &= \begin{bmatrix} \Lambda^T_{i,1} & \dots & \Lambda^T_{i,j} & \dots & \Lambda^T_{i,p} \end{bmatrix}^T \\ \Psi_i &= \begin{bmatrix} \Psi^T_{i,1} & \dots & \Psi^T_{i,j} & \dots & \Psi^T_{i,p} \end{bmatrix}^T \end{aligned} \tag{34}$$

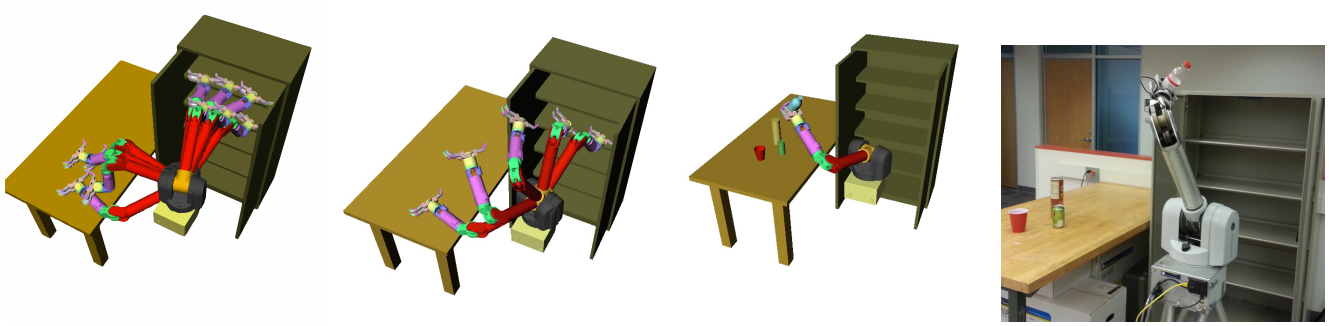
For example, if we want to upsample a trajectory by interpolating p points between each two trajectory states, we can quickly compute the new trajectory as

$$\bar{\xi} = M(\bar{\xi}_w - \mu_w) + \mu \tag{35}$$

where, $M = \begin{bmatrix} \Psi_0 & 0 & \dots & \dots & \dots & \dots & 0 & 0 \\ I & 0 & \dots & \dots & \dots & \dots & 0 & 0 \\ \Lambda_1 & \Psi_1 & \dots & \dots & \dots & \dots & 0 & 0 \\ 0 & I & \dots & \dots & \dots & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & & & & \vdots & \vdots \\ 0 & 0 & \dots & I & 0 & \dots & 0 & 0 \\ 0 & 0 & \dots & \Lambda_i & \Psi_i & \dots & 0 & 0 \\ 0 & 0 & \dots & 0 & I & \dots & 0 & 0 \\ \vdots & \vdots & & & & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & \dots & \dots & \dots & 0 & I \\ 0 & 0 & \dots & \dots & \dots & \dots & 0 & \Lambda_N \end{bmatrix}$

$\bar{\xi}_w$ is the trajectory with a sparse set of states. Fast, high temporal-resolution interpolation is useful in practice if we want to feed the planned trajectory into a controller, or if we want to carefully check for collisions between states during optimization.

GP interpolation additionally allows us to reduce the number of states needed to represent a high-fidelity continuous-time trajectory, as compared with the discrete-state formulation of CHOMP, resulting in a speedup during optimization with virtually no effect on the final plan. The idea is to modify Eq. 33 to update only a sparse set of states $\bar{\xi}_w$, but sweep through the full trajectory to compute the obstacle cost by interpolating a large number of intermediate points. The obstacle gradient is calculated for *all* points (interpolated



(a) (b) (c) (d)

Fig. 1: (a) Environment with 4 start robot arm configurations (on the table) and 6 end robot arm configurations (on the cabinet shelves) used to generate a set of 24 unique planning problems. These planning problems entail displacing objects from the table to the cabinet shelves. The plans are generated in OpenRAVE and executed on the real robot. (b) Example of a successful plan generated by GPMP on one of the planning problems. This plan is used to move a milk bottle from the table to a shelf in the cabinet in (c) simulation and (d) on the real WAM robot.

points and states) and then projected back onto just the states using the matrix M^T (see Eq. 36 below). The update rule for this approach is derived from Eq. 33 as

$$\begin{aligned} & \bar{\xi} \leftarrow \bar{\xi} - \frac{1}{\eta}\mathcal{K}\left[\lambda\mathcal{K}^{-1}(\bar{\xi}-\mu)+g\right] \\ \implies & M\bar{\xi}_w \leftarrow M\bar{\xi}_w - \\ & \quad \frac{1}{\eta}M\mathcal{K}_w M^T\left[\lambda M^{-T}\mathcal{K}_w^{-1}M^{-1}M(\bar{\xi}_w-\mu_w)+g\right] \\ \implies & \quad \bar{\xi}_w \leftarrow \bar{\xi}_w - \frac{1}{\eta}\mathcal{K}_w\left[\lambda\mathcal{K}_w^{-1}(\bar{\xi}_w-\mu_w)+M^T g\right] \end{aligned} \tag{36}$$

where $\mathcal{K}_w$ is the kernel and μ_w is the mean of the trajectory $\bar{\xi}_w$ with sparse set of states. We refer to this algorithm as the Gaussian Process Motion Planner (GPMP).

IV. Experimental Results

We benchmarked GPMP against CHOMP, AugCHOMP, STOMP, and TrajOpt by optimizing trajectories in OpenRAVE [27], a standard simulator used in motion planning [4], [5], [8], [28]–[31], and validated the resulting trajectories on a real robot arm. Our experimental setup consisted of a table and a cabinet and a 7-DOF Barrett WAM arm which was initialized in several distinct configurations. Figure 1 (a) shows the start and end configurations of 24 unique planning problems on which all the algorithms are tested.

For GPMP and AugCHOMP we employed a constant-acceleration prior, i.e. white-noise-on-jerk model, $q'''(t) = w(t)$. (Recall that $w(t)$ is the process noise, see Eq. 6). For simplicity we assumed independence between the degrees of freedom (joints) of the robot. Following the form of LTV-SDE in Eq. 6 we specified the transition function of any joint $d \in D$,

$$\Phi^d(t,s) = \begin{bmatrix} 1 & (t-s) & \frac{1}{2}(t-s)^2 \\ 0 & 1 & (t-s) \\ 0 & 0 & 1 \end{bmatrix} \tag{37}$$

With the independence assumption, it follows that $\Phi(t,s) = \text{diag}(\Phi^1(t,s),\ldots,\Phi^D(t,s))$, and for all the states at t_i, $i = 1\ldots N+1$ we compute

$$\begin{aligned} Q_i^d &= \begin{bmatrix} \frac{1}{2}\Delta t_i^5 Q_c & \frac{1}{8}\Delta t_i^4 Q_c & \frac{1}{6}\Delta t_i^3 Q_c \\ \frac{1}{8}\Delta t_i^4 Q_c & \frac{1}{3}\Delta t_i^3 Q_c & \frac{1}{2}\Delta t_i^2 Q_c \\ \frac{1}{6}\Delta t_i^3 Q_c & \frac{1}{2}\Delta t_i^2 Q_c & \Delta t_i Q_c \end{bmatrix}, \\ (Q_i^d)^{-1} &= \begin{bmatrix} 720\Delta t_i^{-5}Q_c^{-1} & -360\Delta t_i^{-4}Q_c^{-1} & 60\Delta t_i^{-3}Q_c^{-1} \\ -360\Delta t_i^{-4}Q_c^{-1} & 192\Delta t_i^{-3}Q_c^{-1} & -36\Delta t_i^{-2}Q_c^{-1} \\ 60\Delta t_i^{-3}Q_c^{-1} & -36\Delta t_i^{-2}Q_c^{-1} & 9\Delta t_i^{-1}Q_c^{-1} \end{bmatrix} \end{aligned} \tag{38}$$

where $\Delta t_i = t_i - t_{i-1}$. Similarly we compute $Q_i = \text{diag}(Q_i^1,\ldots,Q_i^D)$ and $Q_i^{-1} = \text{diag}((Q_i^1)^{-1},\ldots,(Q_i^D)^{-1})$. The matrix Q^{-1} (Eq. 13) can be built directly and efficiently from the Q_i^{-1}s.

Both GPMP and AugCHOMP handle constraints (Eq 17) in the same way as CHOMP. For AugCHOMP joint limits are obeyed by finding a violation trajectory ξ^v, calculated by taking each point on the trajectory in violation and bringing it within joint limits via a L_1 projection. It is then scaled by the kernel such that it cancels out the largest violation in the trajectory (see [5] for details),[1]

$$\bar{\xi} = \bar{\xi} + \mathcal{K}\xi^v \tag{39}$$

Joint limits for GPMP are obeyed using a technique similar to Eq. 36 applied to Eq. 39,

$$\bar{\xi}_w = \bar{\xi}_w + \mathcal{K}_w M^T \xi^v \tag{40}$$

For all algorithms except GPMP and AugCHOMP, trajectories were initialized as a straight line in configuration space. For GPMP and AugCHOMP we found that initializing the trajectory as a acceleration-smooth line yielded lower prior cost at the start. The initialized trajectories were parameterized by 103 equidistant states. For GPMP 18 states were used with $p = 5$ (103 states effectively), where p is a parameter that represents the number of points interpolated between any two states during each iteration.

[1]For simplicity only positions in configuration space are considered while calculating ξ^v.

TABLE I: Results for 24 planning problems on the 7-DOF WAM arm. See text for details.

	GPMP	AugCHOMP	CHOMP	STOMP	TrajOpt
Problems Solved	24/24	24/24	18/24	10/24	23/24
Average Time to Success (s)	0.503	0.658	0.866	5.656	0.3737
Average Iterations to Success	22.125	22.958	38.378	82.837	6.087

As observed in [5] the CHOMP trajectory can oscillate between feasible and infeasible during optimization. This behavior is observed in AugCHOMP and GPMP as well. Therefore, with the exception of TrajOpt, all algorithms were allowed to optimize for at least 10 iterations before checking for feasibility. Once a feasible trajectory was discovered, the solution was returned. We capped the optimization for these algorithms at 250 iterations, after which, if a feasible solution was not found, a failed trajectory was returned and the problem was marked unsolved by the particular algorithm.

The results on the 24 arm planning problems for all the algorithms are summarized in Table I.[2] Average time to success was computed from successful runs, unsuccessful runs were excluded. Since STOMP is stochastic in nature, we ran the problem set on STOMP 5 times. The results for STOMP reflect the aggregate of the 5 runs.

V. Discussion

From the results in Table I we see that GPMP compares favorably with recent trajectory optimization algorithms on the benchmark. GPMP is able to solve all the 24 problems in a reasonable amount of time, while solving for trajectories in a state space 3 times the size of the state for all the other algorithms (with the exception of AugCHOMP). GPMP provides a speedup over AugCHOMP by utilizing GP interpolation during optimization, such that each iteration is faster without any significant loss in the quality of the trajectory at the end of optimization. This is illustrated in a comparison on an example optimized trajectory obtained from GPMP and AugCHOMP on the same problem (Figure 2).

GPMP and AugCHOMP are able to converge to better solution trajectories (solve more problems) and do so faster when compared to CHOMP. One advantage of these algorithms is that the trajectory is augmented with velocities and accelerations. In contrast to CHOMP, where velocities and accelerations are computed from finite differencing, velocities and accelerations in GPMP and AugCHOMP can be used directly during optimization. This affects the calculation of the objective cost and its gradient, resulting in better gradient steps and convergence in fewer iterations.

Benchmark results show that TrajOpt is faster than our approach and fails on only one of the problems. By formulating the optimization problem as sequential quadratic programming (SQP), TrajOpt achieves faster convergence with fewer iterations. However, GPMP offers several advantages over TrajOpt: the continuous representation allows

[2]Parameters for the benchmark were set as follows: For GPMP and AugCHOMP, $Q_c = 100$. For GPMP, AugCHOMP and CHOMP, $\lambda = 0.005, \eta = 1$. For STOMP, $k = 5$. For TrajOpt, coeffs $= 20$, dist_pen $= 0.05$.

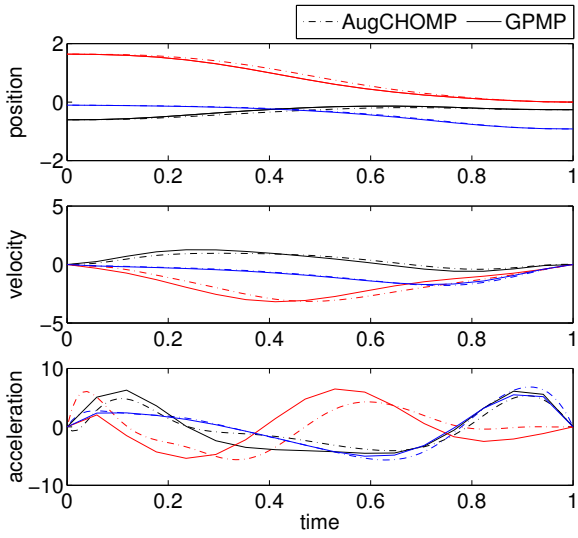


Fig. 2: Comparison of an example optimized trajectory (position, velocity and acceleration in configuration space) obtained using GPMP (18 states) and AugCHOMP (103 states). The black, blue, and red lines, correspond to 3 of the 7 degrees of freedom of the arm.

GPMP to use only a few states to parameterize the trajectory, ensuring smoothness. GP interpolation can be used to propagate obstacle cost from interpolated points to the states during optimization, and also enables up-sampling the output trajectory (ensuring smoothness and providing collision-free guarantee) such that it is executable on a real system.

VI. Conclusions

We have presented a novel approach to motion planning using Gaussian processes for reasoning about continuous-time trajectories by optimizing a small number of states. We considered trajectories consisting of joint positions, velocities, and accelerations sampled from a GP generated from a LTV-SDE, and we provided a gradient-based optimization technique for solving motion planning problems. The Gaussian process machinery enabled us to query the trajectory at any time point of interest, which allowed us to generate executable trajectories or reason about the cost of the entire trajectory instead of just at the states. We benchmarked our algorithm against various recent trajectory optimization algorithms on a set of 7-DOF robotic arm planning problems, and we validated our algorithms by running them on a 7-DOF Barrett WAM arm. Our empirical results show GPMP to be competitive or superior to competing algorithms with respect to speed and number of problems solved.

REFERENCES

[1] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *The international journal of robotics research*, vol. 5, no. 1, pp. 90–98, 1986.

[2] M. Mabrouk and C. McInnes, "Solving the potential field local minimum problem using internal agent states," *Robotics and Autonomous Systems*, vol. 56, no. 12, pp. 1050–1060, 2008.

[3] J.-H. Chuang and N. Ahuja, "An analytically tractable potential field model of free space and its application in obstacle avoidance," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 28, no. 5, pp. 729–736, 1998.

[4] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "CHOMP: Gradient optimization techniques for efficient motion planning," in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. IEEE, 2009, pp. 489–494.

[5] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa, "CHOMP: Covariant Hamiltonian optimization for motion planning," *The International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1164–1193, 2013.

[6] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "STOMP: Stochastic trajectory optimization for motion planning," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 4569–4574.

[7] K. He, E. Martin, and M. Zucker, "Multigrid CHOMP with local smoothing," in *Proc. of 13th IEEE-RAS Int. Conference on Humanoid Robots (Humanoids)*, 2013.

[8] J. Schulman, J. Ho, A. Lee, I. Awwal, H. Bradlow, and P. Abbeel, "Finding locally optimal, collision-free trajectories with sequential convex optimization." in *Robotics: Science and Systems*, vol. 9, no. 1. Citeseer, 2013, pp. 1–10.

[9] C. E. Rasmussen, *Gaussian processes for machine learning*. Citeseer, 2006.

[10] S. Vijayakumar, A. D'souza, and S. Schaal, "Incremental online learning in high dimensions," *Neural computation*, vol. 17, no. 12, pp. 2602–2634, 2005.

[11] K. Kersting, C. Plagemann, P. Pfaff, and W. Burgard, "Most likely heteroscedastic Gaussian process regression," in *Proceedings of the 24th international conference on Machine learning*. ACM, 2007, pp. 393–400.

[12] D. Nguyen-Tuong, J. Peters, M. Seeger, and B. Schölkopf, "Learning inverse dynamics: a comparison," in *European Symposium on Artificial Neural Networks*, no. EPFL-CONF-175477, 2008.

[13] J. Sturm, C. Plagemann, and W. Burgard, "Body schema learning for robotic manipulators from visual self-perception," *Journal of Physiology-Paris*, vol. 103, no. 3, pp. 220–231, 2009.

[14] M. Deisenroth and C. E. Rasmussen, "PILCO: A model-based and data-efficient approach to policy search," in *Proceedings of the 28th International Conference on machine learning (ICML-11)*, 2011, pp. 465–472.

[15] M. K. C. Tay and C. Laugier, "Modelling smooth paths using Gaussian processes," in *Field and Service Robotics*. Springer, 2008, pp. 381–390.

[16] T. Barfoot, C. H. Tong, and S. Sarkka, "Batch continuous-time trajectory estimation as exactly sparse Gaussian process regression," *Proceedings of Robotics: Science and Systems, Berkeley, USA*, 2014.

[17] X. Yan, V. Indelman, and B. Boots, "Incremental sparse GP regression for continuous-time trajectory estimation & mapping," in *Proceedings of the International Symposium on Robotics Research (ISRR-2015)*, 2015.

[18] J. Ko and D. Fox, "GP-BayesFilters: Bayesian filtering using Gaussian process prediction and observation models," *Autonomous Robots*, vol. 27, no. 1, pp. 75–90, 2009.

[19] J. Ko, D. J. Klein, D. Fox, and D. Haehnel, "GP-UKF: Unscented Kalman filters with Gaussian process prediction and observation models," in *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*. IEEE, 2007, pp. 1901–1907.

[20] C. H. Tong, P. Furgale, and T. D. Barfoot, "Gaussian process Gauss-Newton: Non-parametric state estimation," in *Computer and Robot Vision (CRV), 2012 Ninth Conference on*. IEEE, 2012, pp. 206–213.

[21] E. Theodorou, Y. Tassa, and E. Todorov, "Stochastic differential dynamic programming," in *American Control Conference (ACC), 2010*. IEEE, 2010, pp. 1125–1132.

[22] M. R. Walter, R. M. Eustice, and J. J. Leonard, "Exactly sparse extended information filters for feature-based SLAM," *The International Journal of Robotics Research*, vol. 26, no. 4, pp. 335–359, 2007.

[23] M. A. Alvarez, L. Rosasco, and N. D. Lawrence, "Kernels for vector-valued functions: A review," *arXiv preprint arXiv:1106.6251*, 2011.

[24] R. Courant and D. Hilbert, *Methods of mathematical physics*. CUP Archive, 1966, vol. 1.

[25] N. Ratliff, "Analytical dynamics and contact analysis," *Available: http://ipvs.informatik.uni-stuttgart.de*.

[26] S. Quinlan, "Real-time modification of collision-free paths," Ph.D. dissertation, Stanford University, 1994.

[27] R. Diankov and J. Kuffner, "Openrave: A planning architecture for autonomous robotics," *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-08-34*, vol. 79, 2008.

[28] A. Byravan, B. Boots, S. S. Srinivasa, and D. Fox, "Space-time functional gradient optimization for motion planning," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 6499–6506.

[29] M. R. Dogar and S. S. Srinivasa, "Push-grasping with dexterous hands: Mechanics and a method," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*. IEEE, 2010, pp. 2123–2130.

[30] A. D. Dragan, N. D. Ratliff, and S. S. Srinivasa, "Manipulation planning with goal sets using constrained trajectory optimization," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 4582–4588.

[31] L. Y. Chang, S. S. Srinivasa, and N. S. Pollard, "Planning pre-grasp manipulation for transport tasks," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE, 2010, pp. 2697–2704.