Mixtures of Gaussian Processes for Robot Motion Planning using Stochastic Trajectory Optimization

Luka Petrović[1], Ivan Marković[1], Ivan Petrović[1]

***Abstract*—Robot motion planing methods based on trajectory optimization can efficiently generate feasible and optimal trajectories by minimizing a suitable cost function, even in high-dimensional spaces. However, the main drawback of these methods lies in their proneness to infeasible local minima, especially in complex environments. To mitigate this issue, we propose a novel motion planning method that represents trajectories as samples from a mixture of continuous-time Gaussian processes (MGP) and employs stochastic optimization in order to update the MGP parameters in a cost-minimizing manner. The contributions of the proposed trajectory optimization method arise from the introduced mixture representation and stochastic gradient estimation, dominantly enabling better exploration of the trajectory space and including non-differentiable optimizing costs. We evaluated the proposed method in multiple simulation benchmarks featuring 7 DOF robot arms and a 10 DOF mobile manipulator. We also conducted a real-world experiment with a 14 DOF dual arm robot. The experimental results demonstrated that the proposed method achieves higher success rate than several state of the art methods, while the advantages stemming from MGPs and stochastic optimization, like trajectory smoothness, support of non-differentiable cost functions, multiple trajectory solutions, and the ability to tackle high-dimensional planning problems, are inherently kept.**

I. Introduction

Motion planning is a crucial robotics task necessary for efficient and safe robot motion and interaction in real world environments. The goal of motion planning is to produce feasible robot trajectories in configuration space that reach the goal optimally according to a given criterion. Feasibility of a trajectory often refers to collision avoidance and constraints assurance, such as robot joint limits, while optimality often corresponds to trajectory smoothness, minimization of acceleration, executed path length, or a similar qualitative metric. The rapid growth in complexity of both robots and operating environments has accentuated the need for efficient high-dimensional motion planning methods. Take for an example a mobile manipulator working with objects in an unstructured environment or a personal household robot operating in a complex and dynamic environment. Such high degree-of-freedom (DOF) systems require reliable motion planning to perform assigned tasks, while satisfying various constraints and moving through an environment without collisions. Computational efficiency is also particularly important, since trajectory replanning might be needed in constantly changing and dynamic environments. The two most common approaches to motion planning in high-dimensional spaces include sampling-based approaches and trajectory optimization.

The sampling-based approaches [1], [2], [3] build upon the notion of connecting randomly sampled points from the free configuration space – they check the drawn samples for collisions in the environment to figure out if a sample configuration is a part of the free configuration space or not. Sampling-based approaches often provide formal guarantees on finding the solution and most of them exhibit probabilistic completeness, i.e. the probability of generating a solution approaches one as more time is spent [3]. The most prominent sampling-based approaches are probabilistic roadmaps (PRM) [1] and rapidly exploring random trees (RRT) [4], [5]. Both demonstrated the ability to find solutions in configuration spaces with many DOFs and kinodynamic constraints [6], [7]. Even though sampling-based methods do not explicitly optimize a cost function, authors in [3] proposed adaptations of RRT and PRM that were proven to be asymptotically optimal. Nevertheless, many sampling-based methods are computationally inefficient for challenging high-dimensional problems, especially if the environment is dynamic and replanning is needed. They may suffer from the curse of dimensionality and frequently spend sizable computational effort for sampling parts of the configuration space that might be irrelevant for the task. Furthermore, sampling-based approaches rarely take into account trajectory smoothness and consequently the obtained paths necessitate post-processing to avoid jerky or redundant motion. To mitigate the mentioned deficiencies, methods combining potential fields with sampling based planning [8], [9] have been introduced. However, they are unable to optimize secondary costs (e.g. motor torque minimization) and for high-dimensional spaces they either rely on human demonstrations or still require significant computational effort.

The trajectory optimization approaches encode the trajectory as a sequence of states and optimize an arbitrary state-dependent objective function that is in most cases a sum of a smoothness cost and a collision-avoidance cost. While computationally efficient, they lack formal guarantees and sometimes converge to infeasible local minima. They can be divided in two categories: gradient-based approaches and stochastic optimization approaches.

The most prominent gradient-based trajectory optimization approaches are the Covariant Hamiltonian optimization for motion planning (CHOMP) [10], [11], Trajectory Optimization for Motion Planning (TrajOpt) [12], [13], and Gaussian process motion planning (GPMP) [14] and its extensions [15], [16], [17]. GPMP parameterizes the robot trajectory with

[1] Authors are with the University of Zagreb Faculty of Electrical Engineering and Computing, Laboratory for Autonomous Systems and Mobile Robotics, Croatia. E-mail: {name.surname@fer.hr}

This research has been supported by the European Regional Development Fund under the grant KK.01.1.1.01.0009 (DATACROSS).

a few support states and exploits efficient continuous-time Gaussian process (GP) interpolation to query the trajectory at any time of interest, circumventing the CHOMP's need for a large number of states or TrajOpt's need for trajectory post-processing. The GPMP2 algorithm [15], [17] casts the motion planning problem as probabilistic inference, leveraging a sparse GP prior and sparsity-exploiting optimization tools to achieve notably fast performance. While efficient, gradient-based trajectory optimization methods often perform poorly in complex environments as they are unable to avoid local minima due to limited exploration ability and are unable to optimize costs that are non-differentiable.

Stochastic trajectory optimization methods combine sampling and trajectory optimization to overcome the local minima problem present in gradient-based optimization. Contrary to sampling-based approaches, which sample in the configuration space, stochastic trajectory optimization methods sample in the trajectory space, which may be more sample-efficient. Stochastic trajectory optimization for motion planning (STOMP) [18] explores the space around an initial trajectory by taking a batch of noisy trajectories, evaluating their cost, and calculating a stochastic gradient estimate to update the initial trajectory in a cost-minimizing manner. Stochastic multimodal trajectory optimization algorithm (SMTO) [19] utilized variational Bayesian expectation maximization to successfully find multiple solution trajectories, but it is orders of magnitude computationally slower than STOMP. A particle swarm filter method for trajectory optimization [20] displayed being less prone to local minima than gradient-based methods, but it did not showcase its performance in highly cluttered environments. Another group of methods uses cross-entropy optimization [21], [22], [23]. In [21], authors rely on drawing trajectories from the set of feasible paths. The proposed framework generates smooth trajectories; however, sampling whole collision-free trajectories can become infeasible in complex environments. In [22], [23], we represented trajectories as samples from a heteroscedastic GP and the method displayed good performance in cluttered environments emanating from the underlying cross-entropy optimization.

In this paper, we propose a novel motion planning method that leverages advantages of both the trajectory optimization and sampling-based approaches, while retaining computational efficiency. Opposed to representing trajectories as samples from a heteroscedastic GP in [23], the proposed method represents trajectories as samples from a mixture of continuous-time GPs, which better covers the configuration space with regards to criteria such as smoothness and torques while performing similarly in collision avoidance. Furthermore, while the method in [23] employs a cross-entropy-based optimization, the proposed method utilizes an update rule similar to gradient descent but employing a stochastic gradient estimation technique. Due to this update rule, it is significantly more sample efficient than the method in [23] and thus requires less computational resources. It is also able to optimize multiple different optimization criteria, while the method in [23] is not well-suited for this task due to its sample inefficiency.

The main features of the proposed method are: (i) continuous-time GPs as mixture components lead to solution trajectories that can be queried at any time of interest and do not require post-processing, (ii) the proposed MGP initializations cover large parts of trajectory space alleviating the local minima problem while retaining smoothness, (iii) the proposed stochastic gradient estimation allows for optimizing costs that are non-differentiable, and (iv) the MGP representation allows finding multiple solution trajectories to a given problem. These features bring advantages to our method and make it unique since none of the other existing trajectory optimization methods possess all of them. We evaluated our method in multiple simulation benchmarks featuring 7 DOF robot arms and a 10DOF mobile manipulator. We also conducted a real-world experiment with a 14DOF dual-arm robot. The results demonstrated that the proposed method achieves a higher success rate than several state-of-the-art methods, while the advantages stemming from MGPs and stochastic optimization, like trajectory smoothness, support of non-differentiable costs, multiple trajectory solutions, and the ability to tackle high-dimensional planning problems, are inherently kept.

II. Theoretical background

In this section, we present the continuous-time GP suitable for motion planning and the general framework for formulating trajectory optimization problem as probabilistic inference. We closely follow developments from [17], as our method inherits the benefits of the proposed representation. For a more in-depth treatment, we refer the reader to [24], [25].

A. Robot trajectories as Gaussian Processes

We consider the robot's trajectory as a sample from a continuous-time GP [26]

$$\boldsymbol{\theta}(t) \sim \mathcal{GP}(\boldsymbol{\mu}(t), \boldsymbol{\mathcal{K}}(t,t')) \tag{1}$$

that is parameterized with N *support states* at discrete time instants, $\boldsymbol{\theta}_i \in \mathrm{R}^D$, $i \in N$, where D is the state dimensionality. This implies that, for any collection of times $\boldsymbol{t} = \{t_0, \ldots, t_N\}$, $\boldsymbol{\mu}(t)$ is a vector-valued mean function and $\boldsymbol{\mathcal{K}}(t,t'))$ is a matrix-valued covariance function [17], defined as

$$\boldsymbol{\mu} = [\boldsymbol{\mu}(t_0) \ldots \boldsymbol{\mu}(t_N)]^T,\ \boldsymbol{\mathcal{K}} = [\boldsymbol{\mathcal{K}}(t_i,t_j)]|_{i,j,0 \le i,j \le N}. \tag{2}$$

A vector-valued GP provides a theoretically grounded continuous-time trajectory representation, where trajectories are deemed as functions that map time to robot state. A structured kernel that belongs to a special class of GP priors is generated by a linear time-varying stochastic differential equation (LTV-SDE)

$$\dot{\boldsymbol{\theta}}(t) = \boldsymbol{F}(t)\boldsymbol{\theta}(t) + \boldsymbol{v}(t) + \boldsymbol{L}(t)\boldsymbol{w}(t), \tag{3}$$

where $\boldsymbol{F}$ and $\boldsymbol{L}$ are system matrices and $\boldsymbol{v}$ is a known exogenous input. The white noise process $\boldsymbol{w}(t)$ is itself a GP with zero mean value $\boldsymbol{w}(t) \sim \mathcal{GP}(\boldsymbol{0}, \boldsymbol{Q}_c(t)\delta(t-t'))$, where $\boldsymbol{Q}_c(t)$ is a positive semi-definite power-spectral density matrix. A similar dynamical system has been extensively utilized to represent trajectory distributions in estimation [27], [28], calibration [29] and motion planning [17], [23].

A paramount advantage of representing continuous-time trajectories in motion planning with GPs generated by the

LTV-SDE in (3) is the opportunity to query the planned trajectory state $\boldsymbol{\theta}(\tau)$ at any time of interest τ [27]. The exactly sparse block tridiagonal precision matrix supports computationally efficient, structure-exploiting GP interpolation with $\mathcal{O}(1)$ complexity. As shown in [15], state $\boldsymbol{\theta}(\tau)$ at $\tau \in [t_i, t_{i+1}]$ is a function only of its neighboring states. Efficient GP interpolation enables keeping a relatively small number of *support states* even in cluttered environments with small obstacles, which reduces the computational burden compared to the discrete-time representation. It can also be exploited for providing a dense trajectory on the output, which can be directly executed without the need for post-processing.

We represent robot dynamics with the double integrator linear system with white noise injected in the acceleration. The trajectory generated by (3) is defined by the Markovian $\boldsymbol{\theta}(t)$ consisting of position and velocity in the configuration space with the following system matrices

$$\boldsymbol{F}(t) = \begin{bmatrix} \mathbf{0} & \mathbf{I} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}, \boldsymbol{L}(t) = \begin{bmatrix} \mathbf{0} \\ \mathbf{I} \end{bmatrix}, \tag{4}$$

providing a constant velocity GP. By applying this motion model, the joint acceleration is minimized, minimizing the energy consumption in the process and fostering a physical meaning of smoothness [17].

B. Trajectory optimization as probabilistic inference

To formulate motion planning as probabilistic inference, we cast the motion planning problem as searching for the *maximum a posteriori* (MAP) continuous-time trajectory starting from a prior distribution on the space of trajectories and an arbitrary likelihood function. More formally, we look for the posterior density of $\boldsymbol{\theta}$ given a collection of events $\boldsymbol{e}$ which can be calculated via Bayes' rule

$$p(\boldsymbol{\theta}|\boldsymbol{e}) = \frac{p(\boldsymbol{\theta})p(\boldsymbol{e}|\boldsymbol{\theta})}{p(\boldsymbol{e})} \propto p(\boldsymbol{\theta})p(\boldsymbol{e}|\boldsymbol{\theta}), \tag{5}$$

where $p(\boldsymbol{\theta})$ represents the prior on trajectory $\boldsymbol{\theta}$ and $p(\boldsymbol{e}|\boldsymbol{\theta})$ is the likelihood of an event $\boldsymbol{e}$ given trajectory $\boldsymbol{\theta}$.

In previous works [15], [30], a prior distribution on the space of trajectories, whose role is to encourage the smoothness of the trajectory, is a GP defined by the mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\mathcal{K}}$

$$p(\boldsymbol{\theta}) \propto \exp\left\{-\frac{1}{2}\|\boldsymbol{\theta} - \boldsymbol{\mu}\|^2_{\boldsymbol{\mathcal{K}}}\right\}, \tag{6}$$

while the likelihood function, whose role is to encourage posterior trajectories that are collision-free, is commonly defined as a distribution in the exponential family

$$L(\boldsymbol{\theta}; \mathbf{e}) \propto \exp\left\{-\frac{1}{2}\|\boldsymbol{h}(\boldsymbol{\theta}, \mathbf{e})\|^2_{\boldsymbol{\Sigma}}\right\}, \tag{7}$$

where $\boldsymbol{h}(\boldsymbol{\theta}, \mathbf{e})$ is an arbitrary vector-valued cost function with covariance matrix $\boldsymbol{\Sigma}$.

The optimal trajectory $\boldsymbol{\theta}^*$ is then found by minimizing the negative logarithm of the posterior $p(\boldsymbol{\theta}|\boldsymbol{e})$ as follows [17]

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \left\{\text{-log}\left(p(\boldsymbol{\theta})p(\boldsymbol{e}|\boldsymbol{\theta})\right)\right\} \tag{8}$$

$$= \arg\min_{\boldsymbol{\theta}} \left\{-\frac{1}{2}\|\boldsymbol{\theta} - \boldsymbol{\mu}\|^2_{\boldsymbol{\mathcal{K}}} - \frac{1}{2}\|\boldsymbol{h}(\boldsymbol{\theta}, \mathbf{e})\|^2_{\boldsymbol{\Sigma}}\right\}. \tag{9}$$

If a prior on the space of trajectories is given as a GP, the negative logarithm reduces the MAP problem to nonlinear least squares that can be solved with iterative optimization techniques such as Gauss-Newton or Levenberg-Marquardt [15].

III. Proposed method for trajectory optimization

In this section we derive the proposed method for trajectory optimization, which we dubbed *Mixture of Gaussian Processes for Trajectory Optimization* (MGPTO). First we introduce robot trajectory representations as mixtures of Gaussian processes, then we formulate the trajectory optimization problem with an MGP trajectory prior. Afterwards, we present the procedures for stochastic gradient estimation and for weights update of the MGP. At the end, we present the MGPTO method in the form of an algorithm.

A. Robot trajectories as mixtures of Gaussian processes

A single GP trajectory representation allows searching for a collision-free trajectory in the neighborhood of the mean trajectory, which is often initialized as a constant-velocity straight line, but it does not allow thorough exploration of the environment, which is needed for local minima avoidance. Therefore we consider a continuous-time trajectory as a sample from an MGP, allowing for broader initialization and thus better coverage of the environment. The MGP model is defined as a linear superposition of multiple GP components

$$p(\boldsymbol{\theta}(t)) = \sum_{m=1}^{M} \pi_m \mathcal{GP}(\boldsymbol{\mu}_m(t), \boldsymbol{\mathcal{K}}_m(t, t')), \tag{10}$$

where π_m are the mixing coefficients such that $0 \leq \pi_m \leq 1$, $\sum_{m=1}^{M} \pi_m = 1$. Since our main idea is to represent trajectories as samples from an MGP and employ optimization in order to update the MGP parameters in a cost-minimizing manner, we will need to formulate the MGP in terms of discrete *latent* variables to make this problem soluble. Our development is similar to the Gaussian mixture model widely employed in machine learning [31], where the latent variable, which is usually unknown, represents which Gaussian component generated which observation. In our case the problem is a bit different, since we know in advance which MGP component generated which trajectory.

First, we introduce the latent variable $\boldsymbol{z}$ that is an M-dimensional discrete random variable

$$p(\boldsymbol{z}) \sim \text{Categorical}(\boldsymbol{\pi}), \tag{11}$$

where the values of $\boldsymbol{z}$ have a 1-of-M encoding, meaning that only a particular vector element z_m is equal to 1, while all other vector elements are equal to 0. A certain vector element z_m being equal to 1 refers to the m-th mixture component. The joint distribution $p(\boldsymbol{\theta}, \boldsymbol{z})$ can be defined in terms of the marginal and conditional distribution

$$p(\boldsymbol{\theta}, \boldsymbol{z}) = p(\boldsymbol{z})p(\boldsymbol{\theta}|\boldsymbol{z}). \tag{12}$$

The marginal distribution $p(\boldsymbol{z})$ is specified in terms of the aforementioned mixing coefficients $p(z_m = 1) = \pi_m$, while the conditional distribution, given a particular value for $\boldsymbol{z}$, is a GP, $p(\boldsymbol{\theta}|z_m = 1) = \mathcal{GP}(\boldsymbol{\mu}_m(t), \boldsymbol{\mathcal{K}}_m(t,t'))$. Given that, the marginal $p(\boldsymbol{\theta})$ can then be computed by summing over all possible states of $\boldsymbol{z}$

$$p(\boldsymbol{\theta}) = \sum_{\boldsymbol{z}} p(\boldsymbol{z}) p(\boldsymbol{\theta}|\boldsymbol{z}) = \sum_{m=1}^{M} \pi_m \mathcal{GP}(\boldsymbol{\mu}_m(t), \boldsymbol{\mathcal{K}}_m(t,t')). \tag{13}$$

Thus the marginal distribution of $\boldsymbol{\theta}$ is an MGP of the form (10). We have therefore found an equivalent formulation of the MGP involving an explicit latent variable. This formulation implies that every trajectory $\boldsymbol{\theta}^k$ has a corresponding latent variable $\boldsymbol{z}^k$ that determines from which GP component the pertaining trajectory was generated.

1) Drawing trajectory samples: To draw a sample from an MGP involving a latent variable, we first sample the latent variable $\boldsymbol{z}$ with the roulette wheel selection where a given mixture weight π_m is the probability of $z_m = 1$. A sample trajectory can then be generated from the mixture component corresponding to the selected latent variable z_m with

$$\boldsymbol{\theta} = \boldsymbol{\mu}_m + \boldsymbol{A}_m \boldsymbol{Z}, \tag{14}$$

where $\boldsymbol{A}_m$ is a lower triangular matrix obtained by Cholesky decomposition of the covariance matrix, $\boldsymbol{\mathcal{K}}_m = \boldsymbol{A}_m \boldsymbol{A}_m^T$, and $\boldsymbol{Z}$ is a vector of N standard normal variables $\boldsymbol{Z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.

2) Initialization of the mixture components: With the GP trajectory representation, selecting the initial mean is relatively straightforward since a constant-velocity straight line minimizes the joint accelerations and thus provides the notion of smoothness. However, with the MGP trajectory representation we have to select an initial mean for multiple GPs. We introduced the MGP representation in order to achieve better exploration and improve success rate in comparison to the state-of-the-art methods in finding feasible trajectories in complex environments. To accomplish this goal, we have to somewhat sacrifice smoothness since searching for trajectories that are far from the constant-velocity straight line in the configuration space requires larger joint accelerations. We propose an initialization technique of the MGP mixture components that tries to cover distant portions of the configuration space while retaining relatively small joint accelerations.

An initial mean μ_m is generated similarly to sampling a random trajectory with (14) but with a deterministic $\boldsymbol{Z}_m$

$$\boldsymbol{\mu}_m = \boldsymbol{\mu}_0 + \boldsymbol{A}_m \boldsymbol{Z}_m, \tag{15}$$

where $\boldsymbol{\mu}_0$ is a constant-velocity straight line in the configuration space. The elements of $\boldsymbol{Z}_m$ corresponding to the velocity of the m-th robot joint are set to either 1 or -1 during the first half of robot trajectory, and to the opposite value during the second half of trajectory (so that the total integral of injected noise is 0), while all other elements are set to 0. By exerting acceleration only on a single robot joint at a time, the initialized trajectories remain smooth, while still exhibiting significant deviation in the task space in different directions, allowing for better exploration of the environment. An example of the proposed initialization and its comparison to a single GP is shown in Fig. 1, both for an omnidirectional planar robot platform and a 7 DOF robot arm.

B. Probabilistic inference with an MGP prior

We formulate trajectory optimization as probabilistic inference as in Section II-B, but instead of a GP prior we use the prior distribution given as an MGP with its mixing coefficients π_m, means $\boldsymbol{\mu}_m$, and covariances $\boldsymbol{\mathcal{K}}_m$. The conditional distribution $p(\boldsymbol{\theta}|\boldsymbol{z})$, given a particular value for $\boldsymbol{z}$, is the following mixture component

$$p(\boldsymbol{\theta}|z_m = 1) \propto \exp\{-\frac{1}{2}\|\boldsymbol{\theta} - \boldsymbol{\mu}_m\|^2_{\boldsymbol{\mathcal{K}}_m}\}, \tag{16}$$

which corresponds to the GP prior in (6), similarly to the case of [15]. Then, the marginal distribution $p(\boldsymbol{\theta})$ that represents an MGP prior can then be written as

$$p(\boldsymbol{\theta}) \propto \sum_{m=1}^{M} \pi_m \exp\{-\frac{1}{2}\|\boldsymbol{\theta} - \boldsymbol{\mu}_m\|^2_{\boldsymbol{\mathcal{K}}_m}\}. \tag{17}$$

The optimal trajectory $\boldsymbol{\theta}^*$ is found by minimizing the negative logarithm of the posterior distribution $p(\boldsymbol{\theta}|\boldsymbol{e})$ given in (5)

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \Big\{ \text{-log} \left(p(\boldsymbol{\theta}) p(\boldsymbol{e}|\boldsymbol{\theta}) \right) \Big\} \tag{18}$$

$$= \arg\min_{\boldsymbol{\theta}} \Big\{ \text{-log} \sum_{m=1}^{M} \pi_m \exp\{-\frac{1}{2}\|\boldsymbol{\theta} - \boldsymbol{\mu}_m\|^2_{\boldsymbol{\mathcal{K}}_m} - \frac{1}{2}\|\boldsymbol{h}(\boldsymbol{\theta}, \mathrm{e})\|^2_{\boldsymbol{\Sigma}}\} \Big\}. \tag{19}$$

While the negative logarithm reduced the MAP problem to nonlinear least squares when a GP prior was used, in our case when the prior is an MGP, the negative logarithm does not simplify the problem due to the summation operator inside the logarithm. In the following we elaborate on how to solve this issue.

C. Stochastic gradient estimation

Suppose that we know the value of latent variable $\boldsymbol{z}$, i.e. which element $z_m = 1$, and that it remains constant. Then the value of m-th mixing coefficient becomes $p(z_m = 1) = \pi_m = 1$, while all other mixing coefficients are zero. In this case, the summation operator within the optimization problem in (19) disappears, and the problem becomes equivalent to (9)

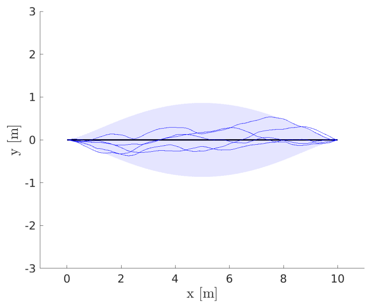
$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \{\frac{1}{2}\|\boldsymbol{\theta} - \boldsymbol{\mu}_m\|^2_{\boldsymbol{\mathcal{K}}_m} + \frac{1}{2}\|\boldsymbol{h}(\boldsymbol{\theta}, \mathrm{e})\|^2_{\boldsymbol{\Sigma}}\}. \tag{20}$$

In order to allow optimizing trajectory costs that are non-differentiable, we employ derivative-free stochastic optimization, instead of the gradient descent optimization proposed in [15]. The introduced stochasticity could additionally enhance exploration and help alleviate the local minima problem present in gradient-based methods. Given that, first we convert the deterministic problem in (20) to a stochastic one by applying the expectation operator

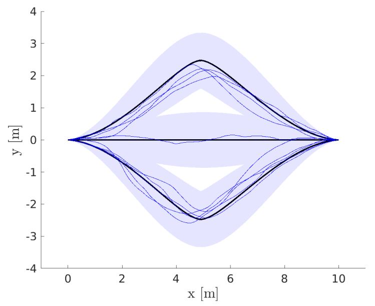
$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \left\{ \mathbb{E}\left[\frac{1}{2}\|\boldsymbol{\theta} - \boldsymbol{\mu}_m\|^2_{\boldsymbol{\mathcal{K}}_m} + \frac{1}{2}\|\boldsymbol{h}(\boldsymbol{\theta}, \mathrm{e})\|^2_{\boldsymbol{\Sigma}} \right] \right\}. \tag{21}$$

Next we take the gradient with respect to $\boldsymbol{\theta}$ to minimize the pertaining expectation

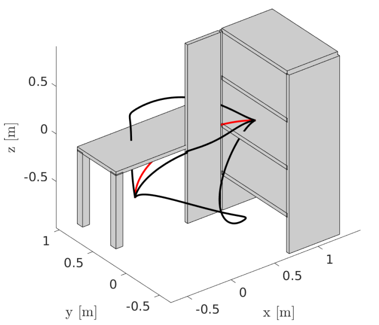
$$\nabla_{\boldsymbol{\theta}} \mathbb{E}\left[(\boldsymbol{\theta} - \boldsymbol{\mu}_m)^T \boldsymbol{\mathcal{K}}_m^{-1} (\boldsymbol{\theta} - \boldsymbol{\mu}_m) + \boldsymbol{h}^T(\boldsymbol{\theta}, \mathrm{e}) \boldsymbol{\Sigma}^{-1} \boldsymbol{h}(\boldsymbol{\theta}, \mathrm{e}) \right] = 0 \tag{22}$$



(a) 2D GP with a straight-line mean and its sample trajectories



(b) 2D MGP with three mixture components and its sample trajectories



(c) 7D MGP with four mixture components. Red trajectory is a straight line in the configuration space, while one of initial trajectories (black) is almost collision-free.

Fig. 1: Comparison of GP and MGP initializations for a planar omnidirectional robot and a 7 DOF robot arm.

leading to

$$\mathbb{E}[\boldsymbol{\theta}] = \boldsymbol{\mu}_m - \boldsymbol{\mathcal{K}}_m \underbrace{\mathbb{E}\left[\nabla_{\boldsymbol{\theta}}\left(\boldsymbol{h}^{\boldsymbol{T}}(\boldsymbol{\theta},\mathbf{e})\boldsymbol{\Sigma}^{-1}\boldsymbol{h}(\boldsymbol{\theta},\mathbf{e})\right)\right]}_{\delta\hat{\boldsymbol{\theta}}_m,\text{ the gradient estimate}}. \quad (23)$$

To further simplify the gradient estimation problem, we assume that the cost $\boldsymbol{h}(\boldsymbol{\theta}, \mathbf{e})$ of each configuration $\boldsymbol{\theta}_i$ depends only on that configuration, meaning that future or past costs do not impact the current cost. This implies that the covariance matrix $\boldsymbol{\Sigma}$ is isotropic, i.e. $\boldsymbol{\Sigma} = \sigma \boldsymbol{I}$, leading to

$$\delta\hat{\boldsymbol{\theta}}_m = \mathbb{E}\left[\nabla_{\boldsymbol{\theta}}\left(\textstyle\sum_{i=1}^{N} \sigma h^2(\boldsymbol{\theta}_i)\right)\right], \quad (24)$$

where σ is a scalar parameter that can be utilized for cost weighting. In this paper, we set $\sigma = 1$ for simplicity.

Existing trajectory optimization methods [10], [15] achieved computational efficiency by employing an iterative gradient descent update using an analytical gradient requiring a smooth differentiable optimization criteria. On the other hand, STOMP [18] proposed a gradient estimation update that circumvents the analytical requirement and can optimize non-differentiable costs. Inspired by STOMP and work in the path integral reinforcement learning [32], we propose an estimated gradient formulation as follows

$$\delta\hat{\boldsymbol{\theta}}_m = \sum_{k=1}^{K} z_m^k \boldsymbol{P}(\tilde{\boldsymbol{\theta}}^k)(\tilde{\boldsymbol{\theta}}^k - \boldsymbol{\mu}_m) \quad (25)$$

with probability metric

$$\boldsymbol{P}(\tilde{\boldsymbol{\theta}}^k) = \exp(-\frac{1}{\lambda}\boldsymbol{f}(\tilde{\boldsymbol{\theta}}^k)), \quad (26)$$

where $\tilde{\boldsymbol{\theta}}^k$ is a single sampled trajectory, z_m^k is the corresponding latent variable, K is the total number of sampled trajectories, and $\boldsymbol{f}(\tilde{\boldsymbol{\theta}}^k)$ corresponds to the state-dependent cost function, $\boldsymbol{f}(\tilde{\boldsymbol{\theta}}^k) = \sum_{i=1}^{N} \sigma h^2(\tilde{\boldsymbol{\theta}}_i^k)$. The parameter λ determines the sensitivity of the exponentiated cost. The variable z_m^k in (25) ensures that only samples from the m-th mixture component partake in gradient estimation for that component.

In other words, to estimate the gradient of the trajectory cost, first we draw K sample trajectories $\tilde{\boldsymbol{\theta}}^k$ and then evaluate the pertaining trajectory cost $\boldsymbol{f}(\tilde{\boldsymbol{\theta}}^k)$ for each sampled trajectory. Afterwards, the probability metric in (26), normalized to unity similarly as in [18], is calculated for each trajectory cost and finally the gradient is computed following (25). Intuitively, trajectories with lower cost are desirable, as they are farther from the obstacles. If a trajectory cost of a certain sampled trajectory $\tilde{\boldsymbol{\theta}}^k$ is particularly small compared to other sampled trajectories, the probability $\boldsymbol{P}(\tilde{\boldsymbol{\theta}}^k)$ of that trajectory is very high. Then the trajectory cost gradient $\delta\hat{\boldsymbol{\theta}}_m$ points in the direction of that trajectory sample. The trajectory mean $\boldsymbol{\mu}_m$ updated with the pertaining gradient *moves* toward the part of trajectory space with smaller trajectory costs, i.e. with desirable properties. The proposed gradient estimation in (25) and update rule in (23) represent the generalization of STOMP for a continuous-time GP. Unlike the discrete-time trajectory representation utilized in STOMP, the continuous-time GP representation includes velocities which can be useful for optimizing costs that can explicitly reason about time [33]. It also supports the aforementioned efficient interpolation that can be exploited for querying the trajectory at any time of interest, resulting in dense collision checking and bypassing the need for post-processing.

D. Updating the MGP weights during optimization

The assumption that the variable $\boldsymbol{z}$ is constant and known allows us to carry out the same optimization technique when sampling from the whole MGP, and not just from a single mixture component. Unlike the previous section, where we had K trajectories from a single component, now we have K trajectories sampled from the whole mixture. Since the information about the trajectory origin, i.e. the corresponding latent variable z_m^k in known and kept, we can utilize trajectories sampled from the m-th mixture component to estimate the gradient and update the mean for that component. We do this for each of M mixture components, following the gradient estimation procedure (25). Essentially, the latent variable formulation enabled us to split the challenging optimization problem in (19) into M simpler ones that can be solved using the update rule (23).

However, we do not want to solve those M simpler problems independently. Since each mixture component covers a different part of trajectory space, some will have smaller cost, and therefore, higher chance of converging to a feasible solution trajectory. We want to explore around initializations that are closer to finding a feasible solution and neglect mixture components with higher trajectory cost. In other words, we also want to update the mixture weights between iterations to ensure that trajectories with smaller costs have higher mixture weights, so that computational time is not spent exploring around trajectories that have high costs, i.e. that are prone to collision. Therefore, we propose to update the mixture weights by utilizing a probabilistic metric similar to (26)

$$\pi_m = \exp(-\frac{1}{\lambda}\boldsymbol{f}(\boldsymbol{\mu}_m)). \tag{27}$$

Given that, after updating each of the M means $\boldsymbol{\mu}_m$, we evaluate their costs $\boldsymbol{f}(\boldsymbol{\mu}_m)$ and update their weights π_m. In the subsequent iterations the components with higher mixture weights get more trajectory samples and get more explored.

Algorithm 1 MGPTO

```
Input: Start and goal states $\boldsymbol{\theta}_0$, $\boldsymbol{\theta}_N$, a state-dependent cost
    function $f(\tilde{\boldsymbol{\theta}}^k(t))$
Precompute: Initial means $\boldsymbol{\mu}_m$ and covariances $\boldsymbol{\mathcal{K}_m}$
 1: while convergence criterion not satisfied do
 2:     for k = $1 \dots K$ do
 3:         Draw a sample trajectory
 4:         $\tilde{\boldsymbol{\theta}}^k(t) \sim \sum_{m=1}^{M} \pi_m \mathcal{GP}(\boldsymbol{\mu}(t), \boldsymbol{\mathcal{K}}(t,t'))$
 5:         Evaluate trajectory cost $f(\tilde{\boldsymbol{\theta}}^k(t))$
 6:         Evaluate probability $\boldsymbol{P}(\tilde{\boldsymbol{\theta}}^k) = \exp(-\frac{1}{\lambda}\boldsymbol{f}(\tilde{\boldsymbol{\theta}}^k))$
 7:     end for
 8:     for m = $1 \dots M$ do
 9:         Normalize probability $\boldsymbol{P}_m(\tilde{\boldsymbol{\theta}}^k) = \frac{z_m^k \boldsymbol{P}(\tilde{\boldsymbol{\theta}}^k)}{\sum_{k=1}^{K} z_m^k \boldsymbol{P}(\tilde{\boldsymbol{\theta}}^k)}$
10:         Estimate gradient
11:         $\delta\hat{\boldsymbol{\theta}}_m = \sum_{k=1}^{K} z_m^k \boldsymbol{P}_m(\tilde{\boldsymbol{\theta}}^k)(\tilde{\boldsymbol{\theta}}^k - \boldsymbol{\mu}_m)$
12:         Compute new mean $\boldsymbol{\mu}_m = \boldsymbol{\mu}_m + \boldsymbol{\mathcal{K}}_m \delta\hat{\boldsymbol{\theta}}_m$
13:         Evaluate cost of new mean $\boldsymbol{f}(\boldsymbol{\mu}_m)$
14:     end for
15:     Compute mixture weights $\pi_m = \frac{\exp(-\frac{1}{\lambda}\boldsymbol{f}(\boldsymbol{\mu}_m))}{\sum_{m=1}^{M}\exp(-\frac{1}{\lambda}\boldsymbol{f}(\boldsymbol{\mu}_m))}$
16: end while
```

E. *MGPTO algorithm*

In this section we present the MGPTO method in the form of an algorithm, summarized in Algorithm 1, which uses results from previous subsections. As inputs, the algorithm requires a state-dependent cost function $f(\tilde{\boldsymbol{\theta}}^k)$ to be minimized as well as the start and goal states $\boldsymbol{\theta_0}$, $\boldsymbol{\theta}_N$ of the trajectory. A state-dependent cost function $f(\tilde{\boldsymbol{\theta}}^k)$ can be non-differentiable and pertain to any type of trajectory constraints or quality metrics. The mixture component means $\boldsymbol{\mu}_m$ are initialized using the technique proposed in Section III-A. To initialize the mixture component covariances $\boldsymbol{\mathcal{K}}_m$, first the integral of the LTV-SDE in (3) is calculated at *support states* and the corresponding covariance matrix is written in the *lifted* form [27]. The underlying GP is then conditioned on the goal state to generate initializations of the covariance matrices $\boldsymbol{\mathcal{K}}_m$ that are suitable for motion planning [17], [23]. After initialization, the main loop of the algorithm is iterated until a given convergence criterion is satisfied. The main loop consists of two *for* loops. In the first *for* loop, the K trajectory samples $\tilde{\boldsymbol{\theta}}^k$ with the corresponding latent variables z_m^k are drawn with (14). For each sampled trajectory, its respective state-dependent cost is evaluated and the probability metric $\boldsymbol{P}(\tilde{\boldsymbol{\theta}}^k)$ is calculated following (26). In the second *for* loop, the gradient is estimated for each of the M mixture components using (25) and each of the M means are updated with

$$\boldsymbol{\mu}_m = \boldsymbol{\mu}_m + \boldsymbol{\mathcal{K}}_m \delta\hat{\boldsymbol{\theta}}_m. \tag{28}$$

After updating, the state-dependent cost of each new mean $\boldsymbol{f}(\boldsymbol{\mu}_m)$ is calculated, which is used at the end to compute the new mixture weights following (27). Note that we also normalize (27) to unity for correct probability representation.

IV. Implementation details of the proposed method

In this section we discuss the implementation details necessary for better comprehension and full reproducibility of the proposed MGPTO method.

A. *Convergence criteria*

In Algorithm 1, the optimization runs until a convergence criterion is satisfied. There are several different convergence criteria that we implement for different purposes. For example, when searching for a single collision-free trajectory, we terminate the optimization when one of the means $\boldsymbol{\mu}_m$ of mixture components becomes collision free, i.e. its collisions cost becomes 0, and that mean is outputted as the solution trajectory. Alternatively, our approach also allows to search for multiple different solution trajectories, i.e., when a collision-free mean $\boldsymbol{\mu}_m$ is found, the pertaining m-th mixture component is pruned from the mixture and the mixture weights π_m for the remaining components are recalculated. The proposed convergence criterion can thus be utilized to find multiple modes of the cost function corresponding to multiple solution trajectories, which has not been extensively studied in the literature. Multiple different solutions allow selecting a single solution based on how the trajectory may appear, which homotopy class it should belong or according to secondary criteria, e.g. the distance from singular configurations or energy consumption. Selecting one of the solutions obtained by MGPTO circumvents the need to replan with different initialization or cost function which might be the case with methods that produce a single solution. Furthermore, if changes in the environment that block a certain path occur during planning time, having alternative solutions that avoid the blocked pathway bypasses the need for replanning. Multiple solutions may also be of use in task-level motion planning [19]. In any case, we can terminate the optimization if too much computational effort was spent by setting a t_{max} parameter for maximum allowed processing time. When optimizing secondary costs (e.g. motor torques), the algorithm stops when the trajectory cost converges, i.e. when it does not decrease for several iterations.

B. Collision avoidance

Similarly as in [15], [23], we deem the robot body as a set of spheres. For a given robot configuration $\boldsymbol{\theta}_i$, we calculate the hinge loss for each sphere representing the robot's body and sum it into a single scalar, resulting in the following obstacle cost function

$$h(\boldsymbol{\theta}_i) = \sum_{j=1}^{S} c\big[d\big(\boldsymbol{x}(\boldsymbol{\theta}_i, S_j), \mathcal{O}\big)\big] \tag{29}$$

where $\boldsymbol{x}$ is the forward kinematics of a robot, $\mathcal{O}$ is the set of obstacles in the environment, d is the Euclidean signed distance function, c is the hinge loss function and S is the number of spheres that represent the robot model. Representing a robot as a set of spheres enables easy computation of the signed distance by subtracting the sphere radius from the calculated distance between a sphere center and its closest obstacle. By precomputing the Euclidean signed distance for the task space represented with a voxel grid of desired resolution we generate the Euclidean signed distance field (SDF), which is used for fast evaluation of the obstacle cost function. When accurate geometry modeling is required, robot representations based on meshes and convex primitives might be better suited, although at higher computation costs. In such cases, any other collision checking tool, e.g. Flexible Collision Library (FCL) [34], can be coupled with our algorithm.

C. Torque optimization

Given a robot dynamics model, the feed-forward torque, needed at each joint to track the desired trajectory, can be computed using inverse dynamics algorithms [35]. The motor torques at any time instant t_i are function of the robot joint states and the pertaining derivatives

$$\boldsymbol{\tau}(t_i) = f(\boldsymbol{\theta}_i, \dot{\boldsymbol{\theta}}_i, \ddot{\boldsymbol{\theta}}_i). \tag{30}$$

While the joint states and joint velocities at time instant t_i are a part of each sampled trajectory due to the underlying constant-velocity model, joint accelerations are calculated by forward finite differentiation.

We construct the cost function that is utilized to minimize the robot motor torques similarly as in [18], by adding the magnitude of torque for each of the J robot joints

$$h(\boldsymbol{\theta}_i) = \sum_{j=1}^{J} \big|\boldsymbol{\tau}_j(t_i)\big| \,. \tag{31}$$

Obtaining torque derivations can be challenging and the absolute value operator makes the cost function in (31) non-differentiable at 0. This is an example of a state-dependent cost that the proposed stochastic optimization method can optimize, unlike the gradient-based methods. Even if we define a differentiable cost function, analytical derivatives of the inverse dynamics function (30) are often unavailable in practice, incurring additional computational burden for gradient-based methods.

D. Joint limits

MGPTO method deals with joint limits by clamping the sampled trajectories $\tilde{\boldsymbol{\theta}}^k$ at the limit values, similarly as in [18]. The updated means $\boldsymbol{\mu}_m$ at each iteration respect the joint limits, since the sampled trajectories $\tilde{\boldsymbol{\theta}}^k$ stay within the limits and the stochastic gradient update is essentially a convex combination of sampled trajectories. Moreover, since the estimated gradient $\delta\hat{\boldsymbol{\theta}}_m$ is smoothed through the covariance matrix $\boldsymbol{\mathcal{K}}_m$ with the underlying constant-velocity model, the updated means $\boldsymbol{\mu}_m$ smoothly touch the joint limit, thus avoiding jerky motion. Given that, MGPTO handles the joint limits during optimization, unlike GPMP2 and CHOMP that handle joint violations as a post-processing step by smoothly projecting joint violations on the output trajectory, which can sometimes lead to the post-processed trajectories that are in collision. Moreover, another advantage of the MGP representation is that it includes velocities as part of the state-space, unlike the discrete representation in STOMP, meaning that MGPTO can also enforce joint angular velocity limits in the same manner, which STOMP is currently unable to do.

V. Experimental results

In this section we present the results of experimental validation of the proposed method and its comparison to the state-of-the-art trajectory optimization techniques GPMP2 [15], STOMP [18] and the method proposed in [23]. We conducted five experiments in different motion planning scenarios:

- *box obstacle benchmark*,
- *robot arm in a complex environment benchmark*,
- *torque optimization benchmark*,
- *mobile manipulator scenario*,
- *dual-arm robot real-world experiment*.

The aim of the *box obstacle benchmark* was to show the benefits of the proposed mixture representation, which allows for determining multiple modes of the cost function unlike the state-of-the-art methods relying on a single straight-line initialization. On this experiment we also conducted extensive analysis of MGPTO hyperparameters to determine their optimal values for achieving the best performance. The aim of the *robot arm in a complex environment benchmark* was to attest the benefits of the MGP prior in aiding better exploration and avoiding local minima to find feasible solutions to the planning problem. The *torque optimization benchmark* aimed to show the ability of MGPTO to optimize a cost function consisting of multiple cost functionals which may be non-differentiable. State-of-the-art gradient-based methods (TrajOpt, CHOMP, GPMP2) are not able to optimize such costs. The goal of the *mobile manipulator scenario* experiment with a 10 DOF mobile manipulator was to showcase the ability of MGPTO to successfully find solution trajectories for high DOF robots in cluttered environments. And finally, the *dual-arm robot real-world experiment* aimed to verify the effectiveness of MGPTO for high DOF motion planning in a real-world environment.

For GPMP2, we used its open-source C++ library [15], [36] and the respective MATLAB toolbox based on the GTSAM library [37], while we opted for our own implementation of STOMP and the method in [23]. All experiments were performed on a single core of a 2.8-GHz Intel Core i7-7700HQ processor with 16 GB of RAM. In our evaluation, every mixture component was initialized with the same covariance matrix $\mathcal{K}_m$, instead of having M different ones. We also used the aforementioned efficient GP interpolation for dense collision checking, similarly as in [15]. Since the *support states* of sampled trajectories are equidistant in time and the covariance matrix $\mathcal{K}_m$ is constant and time-invariant for every mixture component, we precomputed the employed interpolation matrices [15], instead of computing them every time interpolation was needed. This provided a significant boost to the computational efficiency of MGPTO.

A. Box obstacle benchmark

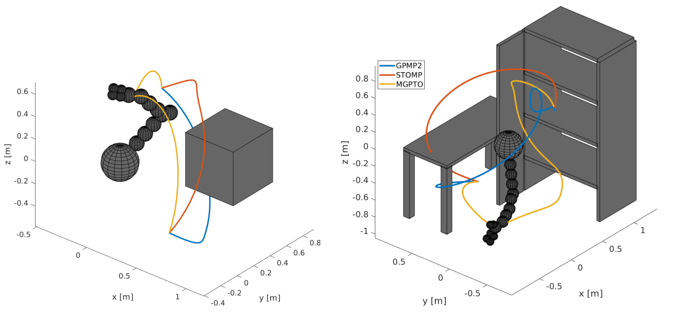
The first benchmark consisted of a simple environment with a single floating box obstacle, as shown in Figure 2a. We conducted 100 unique planning experiments, all with different start and goal states with start points being on the left-hand side of the box and end points being on its right-hand side. To make this experiment non-trivial, for every planning problem we ensured that the robot arm is relatively close to the box obstacle at start and end points and that the initial constant-velocity straight line passes through the box obstacle. While this problem is relatively easy to solve for state-of-the-art methods, benchmarking the algorithm in this environment has several advantages. First, it is easily reproducible due to the simplicity of the environment and thus allows for a fair comparison. A box in the robot's workspace can be seen as a generalization of any obstacle. For this reason we conducted a comprehensive hyperparameter analysis of MGPTO on this benchmark as its simplicity allows to gain intuitive insight about the impact of various hyperparameters. Furthermore, there are multiple solutions to the problem that are obvious to a human – a robot arm can move over, under or behind the box. The proposed setup can thus be utilized to demonstrate the ability of MGPTO to find multiple modes of the cost function corresponding to multiple solution trajectories.

In this experiment we measured the success rate, average algorithm execution time, and the average number of different solution trajectories for MGPTO and compare it to GPMP2, STOMP and the method in [23]. The average execution time for MGPTO corresponds to the time to find the first feasible solution. Due to the simplicity of the experiment, we set the maximum execution time for every method to $t_{max} = 1\,\mathrm{s}$ after which the optimization is terminated. We evaluated the performance of MGPTO with regards to its three hyperparameters: (i) M, the number of mixture components, required for mixture initialization in (10) and further mixture updates in (27), (ii) K, the number of trajectories sampled at each iteration, required for gradient estimation in (25), (iii) Q_c, the power spectral density matrix parameter of the underlying white noise process which determines how noisy the sampled trajectories are, required in (3). We tested twelve different parameter sets, evaluating the impact of each parameter with four different values, *ceteris paribus*. For MGPTO, GPMP2 and the method in [23], we set the total trajectory time, i.e. the timespan in which the robot moves from start to goal state, to $t_{total} = 10\,\mathrm{s}$ and we parameterized trajectories with 10 *support states* with 10 interpolation states in-between for which the trajectory cost was evaluated, resulting in total of 101 states. Using less support states with more interpolated states in-between leads to the optimal performance of GPMP2 [17] and this also applies to MGPTO since we utilize a similar underlying GP framework. We parameterized the STOMP trajectory with $N = 100$ discrete states. For GPMP2, the parameters $\epsilon = 0.2$ and $\Sigma = 0.2$ were chosen for which we empirically determined that they lead to the best results in this experiment. For STOMP, the parameter that determines the number of noisy trajectories generated at each iteration was set to $K = 10$, which was demonstrated to achieve good performance [18]. For the method in [23], we empirically determined its parameter $K = 200$ that led to the best results in this experiment. In all of our experiments, we set the parameter λ that determines the sensitivity of the cost in (26) as $\lambda = 0.1$, which was also chosen in [18] as the optimal value.

The results of this experiment are shown in Table I. MGPTO was able to solve every planning problem with multiple combinations of parameters. Out of four tested values for parameter determining the number of mixture components, $M = 15$ demonstrated the best performance. More different mixture initializations lead to faster exploration of the environment and to a larger number of different solutions found. Note that the maximum number of different mixture initializations in this case was exactly 15 since our initialization technique proposed in Section III-A exerts acceleration only on a single robot joint at a time to keep the trajectory smooth. Since there are 7 robot joints, there is a maximum of 15 initializations; one constant-velocity straight-line and two initializations for every robot joint exerting either positive or negative acceleration during the first half of robot trajectory. The optimal value of parameter K was found in the range from 30 to 70. If there are multiple mixture initializations, then the number of trajectories sampled at each iteration K smaller than 30 does not allow for exploration around every initialization. If K is too large then much of the computation time is spent on evaluating the cost for large number of trajectories which slows down the convergence. The optimal value of parameter Q_c was found in the range from 1 to 5. For small values of Q_c, MGPTO does not explore much around initial trajectories and sometimes fails to find collision-free trajectories. For values of Q_c larger than 5 there is too much noise in the sampled trajectories which sometimes do not even resemble the initial trajectory around which they were sampled. In such cases MGPTO becomes reliant on *luck* which means that it sometimes fails to find collision-free trajectories and that two different runs of the method result in potentially vastly different solutions. MGPTO greatly outperformed STOMP which often failed to find a

TABLE I: Performance of MGPTO with various combination of hyperparameters and comparison with trajectory optimization methods GPMP2, STOMP and the method in [23] in the box obstacle scenario.

	MGPTO												GPMP2	STOMP	Method in [23]
M	3	5	10	15	15	15	15	15	15	15	15	15			
K	50	50	50	50	30	70	100	200	50	50	50	50			
Q_c	1	1	1	1	1	1	1	1	0.1	2	5	10			
Success rate (%)	42	90	90	**100**	**100**	**100**	97	80	87	**100**	**100**	98	**100**	47	**100**
Time to first solution (ms)	433	452	452	177	203	312	339	720	290	238	304	330	**94**	469	252
Average number of solutions	0.6	1.52	2.73	**6.33**	5.9	4.4	3.7	1.83	2.96	5.61	4.82	4.55	1	1	1



(a) An example of three different solution trajectories in the *box obstacle benchmark*

(b) An example where MGPTO finds a collision-free trajectory while GPMP2 and STOMP fail in the *robot arm in a complex scenario benchmark*

Fig. 2: A simulated WAM arm in two scenarios: (a) an environment featuring a single box obstacle (first benchmark), (b) an environment featuring a table and a drawer (second benchmark). Plotted lines depict the end effector trajectories.

collision-free solution. STOMP was particularly troubled when the start or end point was initially close to the box obstacle. GPMP2 solved every planning problem and it was almost twice as fast as MGPTO, while the method in [23] demonstrated similar performance to MGPTO in this benchmark. However, the advantage of MGPTO stems from the fact that the mixture representation allowed finding multiple solutions, while GPMP2 and the method in [23] went above the box obstacle in most cases. Figure 2a presents an example where MGPTO finds three qualitatively different solutions.

B. Robot arm in a complex environment benchmark

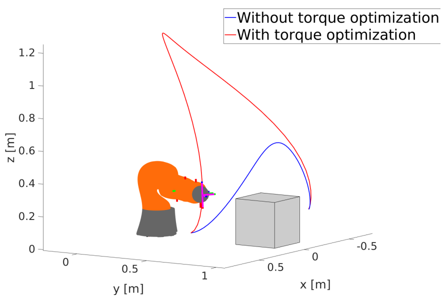
The robot arm planning benchmark consisted of a simulated WAM robotic arm in an environment featuring a table and a drawer cabinet, as shown in Figure 2b. We conducted 100 unique planning experiments, all with different start and goal states with start states being near the table and end states being inside the cabinet. This is a typical scenario for benchmarking trajectory optimization methods, featuring a realistic environment with challenging local minima.

In this benchmark we compared MGPTO with GPMP2, STOMP, the method in [23] and GPMP2 with random restarts (GPMP2-RR) that is a commonly used approach to alleviate the local minima problems in gradient-based trajectory optimization [10]. Following our hyperparameter analysis in Section V-A, for MGPTO we chose the number of mixture components $M = 15$ and the number of sampled trajectories in each iteration $K = 50$. The experimental setup in this benchmark is more complex than the box scenario so larger Q_c is warranted for better exploration and being less prone to the local minima. We empirically determined that $Q_c = 2$ leads to the best performance. As in our previous experiments, we set the total trajectory time $t_{\text{total}} = 10\,\text{s}$. The trajectories for MGPTO, GPMP2 and the method in [23] were parameterized with $N = 10$ equidistant *support states* and 10 interpolation steps in-between. For GPMP2, the parameters $\epsilon = 0.2$ and $\Sigma = 0.2$ were chosen for which we empirically determined that they lead to the best results on this experiment. In GPMP2-RR, the optimizer is first initialized with a straight-line and then, on failure, reinitialized with a random trajectory. We allowed for a maximum of 20 restarts. For the method in [23], we empirically determined its parameter $K = 400$ that led to the best results in this experiment. For STOMP, the parameter K was set to $K = 10$, which was demonstrated to achieve good performance [18]. Due to stochastic nature of results, we repeated every planning experiment with MGPTO, GPMP2-RR, the method in [23] and STOMP 10 times to correctly assess the success rate and computation time. We set the maximum execution time for every method to $t_{max} = 3\,\text{s}$ after which the optimization is terminated.

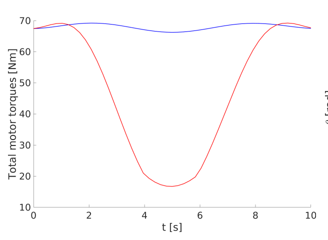
TABLE II: Comparison of MGPTO with GPMP2, STOMP and the method in [23] in robot arm planning benchmark

	MGPTO	GPMP2	GPMP2-RR	STOMP	Method in [23]
Success rate (%)	98	51	86	35	**100**
Execution time (ms)	792	**82**	460	1844	1471

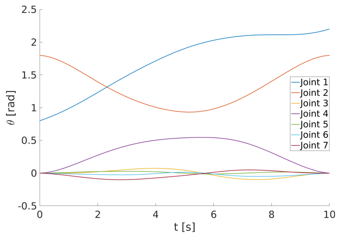
The results of the experiment are shown in Table II. MGPTO outperformed GPMP2 and STOMP regarding the success rate. STOMP achieved both the worst success rate and execution time due to its inability to explore. While the baseline GPMP2 often fails, the stochasticity introduced by random restarts helps in achieving higher success rates. GPMP2 is an order of magnitude faster than MGPTO and STOMP. This is expected since it is a classical gradient-based trajectory optimization method. The method in [23] is well-suited for this type of problem and was able to solve every problem instance but required significantly more computation time than MGPTO. Note that MGPTO was actually able to solve every problem in the dataset on separate occasions, but it did not solve every problem every time and thus the reported success rate is not 100%. However, the sheer ability to solve every problem on this dataset is an indication of benefits of the MGP prior in aiding better exploration of the environment. It also implies that, on failure, MGPTO can be executed again to retry finding a solution trajectory. An example of a case where our method finds a solution, while GPMP2 and STOMP fail is depicted in Figure 2b. The trajectory obtained with method in [23] was



(a) An example of obtained trajectories



(b) Sum of absolute motor torques through time



(c) Joint angles through time

Fig. 3: Comparison of MGPTO with and without torque optimization in an environment featuring a single box obstacle (*torque optimization benchmark*).

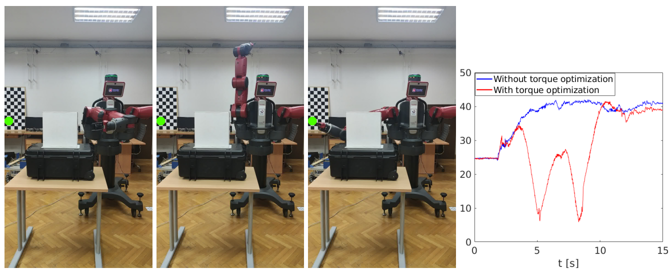


Fig. 4: Baxter's right arm configurations during execution of a trajectory obtained with MGPTO and the sum of absolute motor torques. The green dot depicts end-effector's goal pose.

qualitatively similar to the solution found by MGPTO and thus we omit it for clarity.

C. Torque optimization benchmark

The torque optimization benchmark featured a simulated 7 DOF Kuka LBR iiwa robotic arm in an environment featuring a single box (Figure 3a), similar to the setup in Section V-A. We conducted 100 unique planning experiments with different start and goal states where the robot arm starts from the left side of the box and finishes on the right side. Besides collision avoidance, the optimized trajectory cost $\boldsymbol{f}(\tilde{\boldsymbol{\theta}}^k)$, required in (26), included the criteria for torque minimization, described in (31).

In this experiment we tested MGPTO with and without torque optimization cost in order to demonstrate its effectiveness in minimizing secondary costs that are non-differentiable. We compared the results with STOMP since it is, to the best of authors' knowledge, the only state-of-the-art trajectory optimization method that demonstrated the ability to minimize motor torques. As in our previous experiments, we set the total trajectory time $t_{\text{total}} = 10\,\text{s}$ and we parameterized the trajectory with $N = 10$ equidistant *support states* with 10 interpolation steps in-between. Guided by our hyperparameter analysis in Section V-A, we chose the number of mixture components $M = 15$, the number of sampled trajectories in each iteration $K = 50$ and the covariance matrix parameter of the white noise $Q_c = 1$.

TABLE III: Comparison of MGPTO with STOMP in torque optimization benchmark

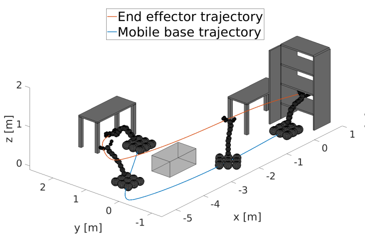
	MGPTO	STOMP	No torque cost
Avg. sum of torques (Nm)	**42.34**	44.02	60.24
Execution time (s)	5.66	11.98	**0.44**

The average sum of absolute motor torques and the average execution times of MGPTO and STOMP in this experiment are shown in Table III. The MGPTO with torque optimization improved the average sum of absolute motor torques during the trajectory around 30% when compared to MGPTO without torque optimization, and around 4% when compared to STOMP. Furthermore, the runtime of MGPTO compared to STOMP was significantly shorter which stems from the fact that some of the mixture components in MGPTO initially cover parts of the configuration space with lower torques, while STOMP needs more time to explore. MGPTO thus converges faster, attesting the benefits of using mixtures instead of a single straight line initialization. An example of the obtained trajectories in simulation is shown in Figure 3a, while the corresponding sum of absolute robot motor torques is shown in Figure 3b. When the torque cost was optimized, the resulting trajectory moved the robot arm upwards resulting in configurations that required lower gravity compensation torques. Since the arm movements are relatively slow, most of the motor torques are used for gravity compensation and thus torques near the start and end states remain the same with and without optimization. We omit the trajectory obtained with STOMP from the illustration since it was qualitatively very similar to the trajectory obtained with MGPTO. Joint angles through time for the torque-optimized trajectory, depicted in Figure 3c, demonstrate that our trajectories retain smoothness even when multiple different optimization criteria are included in the cost function.

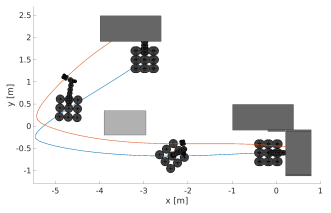
We also conducted a real-world experiment featuring the 7 DOF right arm of the Rethink Robotics Baxter robot. The resulting trajectory and the corresponding sum of absolute motor torques through time are shown in Figure 4 and the accompanying video. The results of the experiment were qualitatively similar to those obtained in simulation, thus verifying MGPTO's performance in real-world scenarios.

D. Mobile manipulator experiment

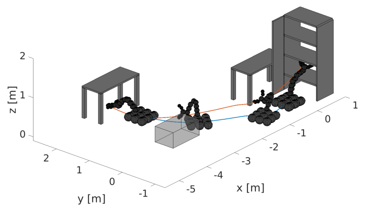
We conducted the mobile manipulator trajectory planning experiment to demonstrate the ability of MGPTO to find collision-free trajectories for high DOF robots in complex environments. The simulated mobile manipulator consisted of a simulated WAM robotic arm mounted on top of an omnidirectional platform in an environment featuring two



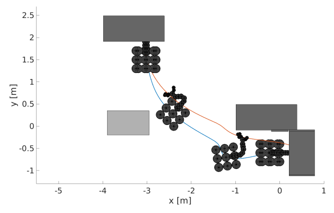
(a) 3d view of the first trajectory



(b) 2d view of the first trajectory



(c) 3d view of the second trajectory



(d) 2d view of the second trajectory

Fig. 5: An example of two trajectories obtained with MGPTO on a simulated mobile manipulator in a static environment.

tables, a drawer cabinet and a generic static obstacle (see Figure 5). The system has 10 DOF corresponding to mobile platform's 3 DOF pose and robot arm's 7 joint angles. We planned the trajectory of the whole system, simultaneously planning the motion of the platform and the robotic arm.

We chose the number of mixture components $M = 5$. This choice of M leads to initial mixture component trajectories differing in trajectories of the mobile platform, while the initial trajectories of the robot arm are locally identical. Using more mixture components, which would change initial trajectories of the attached robot arm, would not lead to noticeably better exploration since the movement of the mobile platform has more impact on the global pose of the whole mobile manipulator than the movement of the attached robotic arm. The number of sampled trajectories in each iteration was set to $K = 30$, while the covariance matrix parameter of the white noise was set to $Q_c = 1$. For both parameters we empirically determined that they lead to satisfactory performance with regards to computation time. As in our previous experiments, we set the total trajectory time $t_{\text{total}} = 10\,\text{s}$. The trajectory was parameterized with $N = 20$ equidistant *support states* and 10 interpolation steps in-between. The maximum execution time was set to $t_{max} = 5\,\text{s}$. To assess the required computation time, we repeated this experiment 30 times. Every take resulted in successfully finding at least one collision-free trajectory and the average algorithm time to first solution was 841 ms, while the worst case computation time was 988 ms. In most cases, MGPTO was able to find two distinct trajectories belonging to different homotopy classes. The resulting trajectories are shown in Figure 5. The trajectory in Figures 5a, 5b was found every time, while the trajectory in Figures 5c, 5d was found in 23 out of 30 experiment runs. The average computation time to find the second solution trajectory was 1801 ms.

E. Dual-arm robot real-world experiment

We conducted the dual-arm robot real-world experiment to demonstrate the ability of MGPTO to find collision-free trajectories for high DOF robots in real environments. We planned the motion of the Rethink Robotics Baxter robot that features two 7 DOF robotic arms giving it a total of 14 DOF. Our laboratory environment consisted of a table and a drawer, resembling the simulated environment used in Section V-B. We planned the trajectory of the whole system, simultaneously planning the motion of both arms. The left arm was tasked with moving from the top shelf of the drawer to the left side of the robot, while the right arm simultaneously moved from the table towards the middle shelf of the drawer.

As in our previous experiments, we set the total trajectory time $t_{\text{total}} = 10\,\text{s}$. The trajectories were parameterized with $N = 10$ equidistant *support states* and 10 interpolation steps in-between. The computation time to find the solution trajectory was 2404 ms with parameters $M = 15$, $K = 50$ and $Q_c = 1$. The results of the experiment are shown in Figure 6 and the accompanying video.

F. Discussion

In the *box obstacle benchmark*, we demonstrated the ability of MGPTO to find multiple solutions to the planning problem. This feature is unique to MGPTO as none of the state-of-the-art trajectory optimization methods are able to do so. The *robot arm in a complex environment benchmark* showed the improvement of MGPTO over prior techniques, namely STOMP and GPMP2, in finding a collision-free trajectory for a 7 DOF manipulator in cluttered environment. The *torque optimization benchmark* showcased the ability of MGPTO to optimize the sum of absolute robot motor torques during the trajectory and demonstrated improvement in comparison to STOMP. State-of-the-art gradient based methods, such as GPMP2, are unable to optimize a cost function consisting of multiple cost functionals which may be non-differentiable. The qualitative mobile manipulator experiment and the real-world dual-arm robot experiment showed the ability of MGPTO to successfully find solution trajectories for high DOF robots in cluttered environments.

One may argue that MGPTO should be significantly slower than STOMP due to the fact that MGPTO conducts M trajectory updates at each iteration. However, the computational burden stemming from sampling K trajectories at each iteration is much higher than the gradient estimation procedure in (25) and the mean update rule in (23). Thus, if we sample the same number of trajectories K with MGPTO and STOMP, a single MGPTO iteration will be just slightly slower than a single STOMP iteration. On the other hand, some of our mixture component initializations cover parts of trajectory space with smaller cost already in the beginning and MGPTO will therefore often converge in much less iterations, being more sample efficient and faster in practical applications, as shown in the experiments.

When compared to GPMP2, MGPTO showed better exploration of the environment, although being slower. MGPTO successfully found solutions to motion planning problems that

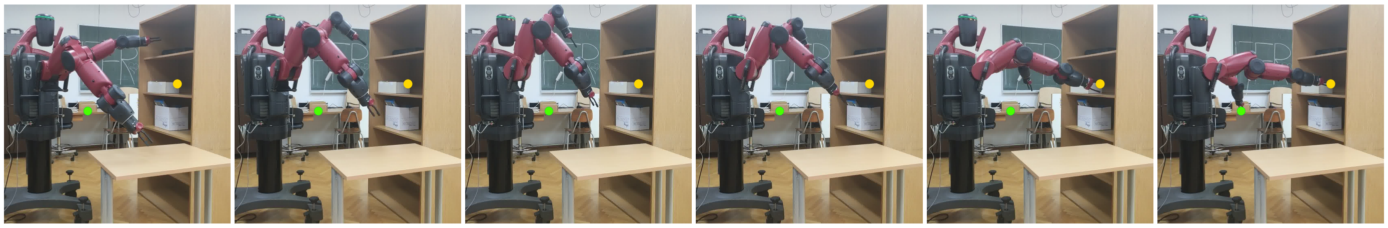


Fig. 6: Baxter's dual-arm configurations during execution of a trajectory obtained with MGPTO. The left arm was tasked with moving from the top shelf of the drawer to the left side of the robot (green dot), while the right arm simultaneously moved from the table towards the middle shelf of the drawer (yellow dot).

GPMP2 was not able to solve. GPMP2 is more appropriate for environments with low complexity, such as the box obstacle scenario, due to its computational efficiency. However, MGPTO outperforms it in complex motion planning problems due to its high capacity for exploration, as well as the ability to explicitly handle joint limits and non-differentiable costs pertaining to arbitrary qualitative metrics or constraints.

Compared to [23], MGPTO achieved similar exploration of the environment while being less computationally demanding. The method in [23] focuses solely on collision avoidance, while MGPTO possesses unique features such as explicit handling of joint position and velocity limits while retaining smoothness, ability to find multiple solutions and the ability to optimize secondary criteria which makes it more suitable for general motion planning problems.

VI. Conclusion

In this paper, we proposed a novel motion planning algorithm dubbed MGPTO that represents trajectories as samples from a mixture of continuous-time GPs and employs stochastic optimization in order to update the MGP parameters in a cost-minimizing manner. We evaluated MGPTO in multiple simulation benchmarks featuring 7 DOF robot arms and a 10 DOF mobile manipulator. We also conducted a real-world experiment with a 14 DOF dual arm robot. The experimental results demonstrated that the proposed method achieves higher success rate than several state-of-the-art methods, while the advantages stemming from MGPs and stochastic optimization, like trajectory smoothness, support of non-differentiable costs, multiple trajectory solutions, and the ability to tackle high-dimensional problems, are inherently kept.

An interesting avenue for future work would be coupling MGPTO with algorithms for learning from demonstration. The initial parameters of the proposed MGP representation could be learned from human demonstrations on a given task [38] and the proposed stochastic trajectory optimization would enable generalized skill reproduction. Another potential direction lies in adapting MGPTO for dynamic environments, especially since the ability to find multiple solution trajectories can be exploited when a particular trajectory becomes infeasible due to changes in the environment.

References

[1] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[2] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[3] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[4] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *IEEE International Conference on Robotics and Automation (ICRA)*, vol. 2, 2000, pp. 995–1001.

[5] S. M. LaValle, J. J. Kuffner, B. Donald, *et al.*, "Rapidly-exploring random trees: Progress and prospects," *Algorithmic and computational robotics: new directions*, vol. 5, pp. 293–308, 2001.

[6] R. Kindel, D. Hsu, J.-C. Latombe, and S. Rock, "Kinodynamic motion planning amidst moving obstacles," in *IEEE International Conference on Robotics and Automation (ICRA)*, vol. 1, 2000, pp. 537–543.

[7] S. M. LaValle and J. J. Kuffner Jr, "Randomized kinodynamic planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[8] Z. Tahir, A. H. Qureshi, Y. Ayaz, and R. Nawaz, "Potentially guided bidirectionalized rrt* for fast optimal path planning in cluttered environments," *Robotics and Autonomous Systems*, vol. 108, pp. 13–27, 2018.

[9] N. García, J. Rosell, and R. Suárez, "Motion planning by demonstration with human-likeness evaluation for dual-arm robots," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 49, no. 11, pp. 2298–2307, 2017.

[10] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "Chomp: Gradient optimization techniques for efficient motion planning," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2009, pp. 489–494.

[11] 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.

[12] J. Schulman, J. Ho, A. X. 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, 2013, pp. 1–10.

[13] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel, "Motion planning with sequential convex optimization and convex collision checking," *The International Journal of Robotics Research*, vol. 33, no. 9, pp. 1251–1270, 2014.

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

[15] J. Dong, M. Mukadam, F. Dellaert, and B. Boots, "Motion planning as probabilistic inference using gaussian processes and factor graphs." in *Robotics: Science and Systems*, vol. 12, no. 4, 2016.

[16] E. Huang, M. Mukadam, Z. Liu, and B. Boots, "Motion planning with graph-based trajectories and gaussian process inference," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 5591–5598.

[17] M. Mukadam, J. Dong, X. Yan, F. Dellaert, and B. Boots, "Continuous-time gaussian process motion planning via probabilistic inference," *The International Journal of Robotics Research*, vol. 37, no. 11, pp. 1319–1340, 2018.

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

[19] T. Osa, "Multimodal trajectory optimization for motion planning," *The International Journal of Robotics Research*, vol. 39, no. 8, pp. 983–1001, 2020.

[20] J.-J. Kim and J.-J. Lee, “Trajectory optimization with particle swarm optimization for manipulator motion planning,” *IEEE Transactions on Industrial Informatics*, vol. 11, no. 3, pp. 620–631, 2015.

[21] M. Kobilarov, “Cross-entropy randomized motion planning,” in *Robotics: Science and Systems*, vol. 7, 2012, pp. 153–160.

[22] L. Petrović, J. Peršić, M. Seder, and I. Marković, “Stochastic optimization for trajectory planning with heteroscedastic gaussian processes,” in *European Conference on Mobile Robots (ECMR)*. IEEE, 2019, pp. 1–6.

[23] ——, “Cross-entropy based stochastic optimization of robot trajectories using heteroscedastic continuous-time gaussian processes,” *Robotics and Autonomous Systems*, p. 103618, 2020.

[24] T. D. Barfoot, *State Estimation for Robotics*. Cambridge University Press, 2017.

[25] P. S. Maybeck, *Stochastic models, estimation, and control*. Academic press, 1982.

[26] C. E. Rasmussen, “Gaussian processes in machine learning,” in *Summer School on Machine Learning*. Springer, 2003, pp. 63–71.

[27] T. D. Barfoot, C. H. Tong, and S. Särkkä, “Batch continuous-time trajectory estimation as exactly sparse gaussian process regression.” in *Robotics: Science and Systems*, vol. 10, 2014, pp. 1–10.

[28] S. Anderson, T. D. Barfoot, C. H. Tong, and S. Särkkä, “Batch nonlinear continuous-time trajectory estimation as exactly sparse gaussian process regression,” *Autonomous Robots*, vol. 39, no. 3, pp. 221–238, 2015.

[29] J. Peršić, L. Petrović, I. Marković, and I. Petrović, “Spatiotemporal multisensor calibration via gaussian processes moving target tracking,” *IEEE Transactions on Robotics*, vol. 37, no. 5, pp. 1401–1415, 2021.

[30] M. Mukadam, J. Dong, F. Dellaert, and B. Boots, “Steap: simultaneous trajectory estimation and planning,” *Autonomous Robots*, vol. 43, no. 2, pp. 415–434, 2019.

[31] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[32] E. Theodorou, J. Buchli, and S. Schaal, “A generalized path integral control approach to reinforcement learning,” *The Journal of Machine Learning Research*, vol. 11, pp. 3137–3181, 2010.

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

[34] J. Pan, S. Chitta, and D. Manocha, “Fcl: A general purpose library for collision and proximity queries,” in *2012 IEEE International Conference on Robotics and Automation*. IEEE, 2012, pp. 3859–3866.

[35] R. Featherstone, *Rigid body dynamics algorithms*. Springer, 2014.

[36] J. Dong, B. Boots, and F. Dellaert, “Sparse gaussian processes for continuous-time trajectory estimation on matrix lie groups,” *arXiv preprint arXiv:1705.06020*, 2017.

[37] F. Dellaert, “Factor graphs and gtsam: A hands-on introduction,” Georgia Institute of Technology, Tech. Rep., 2012.

[38] S. Cho and S. Jo, “Incremental online learning of robot behaviors from selected multiple kinesthetic teaching trials,” *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 43, no. 3, pp. 730–740, 2012.