

# Memory of Motion for Warm-starting Trajectory Optimization

Teguh Santoso Lembono<sup>1</sup>, Antonio Paolillo<sup>2</sup>, Emmanuel Pignat<sup>1</sup>, and Sylvain Calinon<sup>1</sup>

**Abstract**—Trajectory optimization for motion planning requires good initial guesses to obtain good performance. In our proposed approach, we build a memory of motion based on a database of robot paths to provide good initial guesses. The memory of motion relies on function approximators and dimensionality reduction techniques to learn the mapping between the tasks and the robot paths. Three function approximators are compared:  $k$ -Nearest Neighbor, Gaussian Process Regression, and Bayesian Gaussian Mixture Regression. In addition, we show that the memory can be used as a metric to choose between several possible goals, and using an ensemble method to combine different function approximators results in a significantly improved warm-starting performance. We demonstrate the proposed approach with motion planning examples on the dual-arm robot PR2 and the humanoid robot Atlas.

**Index Terms**—Learning and Adaptive Systems; Motion and Path Planning

## I. INTRODUCTION

MOTION planning for robots with high Degree-of-Freedoms (DoFs) presents many challenges, especially in the presence of constraints such as obstacle avoidance, joint limits, etc. To handle the high-dimensionality and the various constraints, many works [1] [2] [3] focus on *trajectory optimization* methods that attempt to find a locally optimal solution. In this approach, the motion planning problem is formulated as an optimization problem

$$\min_{\mathbf{q}_{0:T}} \ell(\mathbf{q}_{0:T}), \quad \text{s.t.} \quad \mathbf{g}(\mathbf{q}_{0:T}) \leq \mathbf{0}, \quad \mathbf{h}(\mathbf{q}_{0:T}) = \mathbf{0}, \quad (1)$$

where  $\mathbf{q}_{0:T}$  denotes the robot's configurations from time step  $t = 0$  to  $t = T$ ;  $\ell(\cdot)$ ,  $\mathbf{g}(\cdot)$  and  $\mathbf{h}(\cdot)$  are the cost, the inequality and the equality constraints. The solution of (1) is the *path*  $\mathbf{y} = \mathbf{q}_{0:T}^* = (\mathbf{q}_0^{*\top}, \dots, \mathbf{q}_T^{*\top})^\top \in \mathbb{R}^{DT}$ , with  $D$  the dimension of  $\mathbf{q}_t$ . When the path is parameterized by time, it is called *trajectory*.

As an example, consider the planning problem depicted in Fig. 1, where the PR2 robot has to move its base around an object or to perform a dual-arm motion to pick items from

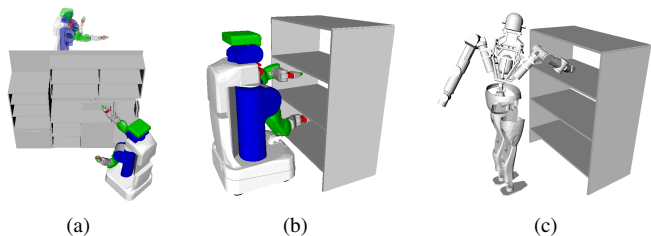


Figure 1. Examples of motion planning problems: (a) moving the PR2 base between two points while avoiding an obstacle, (b) dual arm motion of PR2 to pick items from a shelf to another, and (c) whole body motion of Atlas.

the shelves. If the task  $\mathbf{x} = (\mathbf{q}_{\text{init}}^\top, \mathbf{q}_{\text{goal}}^\top)^\top$  is to move from an initial configuration  $\mathbf{q}_{\text{init}}$  to a goal configuration  $\mathbf{q}_{\text{goal}}$  while minimizing the total joint velocity, the optimization problem can be written as

$$\min_{\mathbf{q}_{0:T}} \sum_{t=0}^{T-1} \|\mathbf{q}_{t+1} - \mathbf{q}_t\|^2, \quad \text{s.t.} \quad \mathbf{q}_0 = \mathbf{q}_{\text{init}}, \quad \mathbf{q}_T = \mathbf{q}_{\text{goal}}. \quad (2)$$

Other constraints can also be added, e.g. to avoid collisions, to comply with joint limits, etc.

Such optimization problems are in general non-convex, especially due to the collision constraints, which makes finding the global optimum very difficult. Trajectory optimization methods such as TrajOpt [1], CHOMP [2], or STOMP [3] solve the non-convex problem by iteratively optimizing around the current solution. While such approach is very popular and yields good practical results, the convergence and the quality of the solution are very sensitive to the choice of the initial guess. If it is far from the optimal solution, the method can get stuck at a poor local optimum.

To overcome this problem, our approach builds a *memory of motion* that learns how to provide good initializations (i.e., a *warm-start*) to the solver based on previously solved problems. Functionally, the memory of motion is expected to learn the mapping  $\mathbf{f} : \mathbf{x} \rightarrow \mathbf{y}$  that maps each task  $\mathbf{x}$  to the robot path  $\mathbf{y}$ . Such mapping can be highly nonlinear and *multimodal* (i.e., one task  $\mathbf{x}$  can be associated to several robot paths  $\mathbf{y}$ ), and the dimension of  $\mathbf{y}$  is typically very high. Our proposed method relies on machine learning techniques such as function approximation and dimensionality reduction to learn this mapping effectively. We use the term *memory of motion* to include both the database of motions and the algorithms to query the warm-starts from the database.

We point out that while other techniques such as sampling-based motion planners can also be used to warm-start the solver (e.g. in [4]), such methods typically require a consid-

Manuscript received: September, 10, 2019; Revised December, 31, 2019; Accepted February, 03, 2020.

This paper was recommended for publication by Editor Dongheui Lee upon evaluation of the Associate Editor and Reviewers' comments. This work was supported by the European Commission's Horizon 2020 Programme (MEMMO project, <http://www.memmo-project.eu/>, grant 780684).

<sup>1</sup>T. S. Lembono, E. Pignat, and S. Calinon are with Idiap Research Institute, Switzerland and EPFL, Switzerland (email: teguh.lembono@idiap.ch; emmanuel.pignat@idiap.ch; sylvain.calinon@idiap.ch)

<sup>2</sup>A. Paolillo is with Idiap Research Institute, Switzerland (email: antonio.paolillo@idiap.ch)

Digital Object Identifier (DOI): 10.1109/LRA.2020.2972893

©2020 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

erable computation time (i.e. in the order of seconds) that is comparable to the solver’s convergence time itself, given the very high dimensional problems considered here. In contrast, querying the memory of motion can be done very fast, in the order of milliseconds. Additionally, our proposed method produces initial guesses that are close to the optimal solutions, reducing the convergence time.

The contribution of this paper is the following. First, we propose the use of function approximation methods to learn the mapping  $f(\mathbf{x})$ . We consider three methods:  $k$ -Nearest Neighbor ( $k$ -NN), Gaussian Process Regressor (GPR) and Bayesian Gaussian Mixture Regression (BGMR), and discuss their different characteristics on various planning problems. We show in particular that BGMR handles multimodal output very well. Furthermore, we show that the memory of motion can be also be used as a metric for choosing optimally between several possible goals. Finally, we demonstrate that using an ensemble of function approximators to provide warm-starts boosts the success rate significantly.

The paper is organized as follows. In Section II we discuss the related work that use the concept of memory of motion for various problems. Section III explains the methods for constructing and using the memory of motion. The experimental results are presented and discussed in Section IV and V. Finally, Section VI concludes the paper.

## II. RELATED WORK

The idea of using a memory of motion to warm-start an optimization solver has previously been explored in the context of optimal control and motion planning. In [5] a trajectory library is constructed to learn a control policy. A set of trajectories are planned offline using  $A^*$  algorithm and stored as library, then  $k$ -NN is used online to determine the action to perform at each state. In [6], they use similar approach to predict an initial guess for balancing a two-link robot, which is then optimized by Differential Dynamic Programming. An iterative method to build a memory of motion to initialize an optimal control solver is proposed in [7]. They use neural networks to approximate the mapping from the task descriptors (initial and goal states) to the state and control trajectories. Another neural network is trained to approximate the value function, which is then used as a metric to determine how close two states are dynamically. In [8] GPR is used to predict new trajectories based on the library of demonstrated movements encoded as Dynamic Movement Primitives (DMP) [9]. GPR is used to map the task descriptors to the DMP parameters.

In robot motion planning, Probabilistic Roadmap (PRM) [10] can be seen as the construction of memory of motion by precomputing the graph connecting robot configurations. Some works exploit Rapidly-exploring Random Trees (RRT) [11], another popular sampling-based method. For example, in [12] an offline computation is used to speed-up the online planning in the form of an additional bias in sampling the configurations. In [13], an *Experience Graph* is built from previously planned solutions. During the online planning, the search is biased towards this graph. The *Lightning* framework is proposed in [14] to plan paths in

high-dimensional spaces by learning from experience. The path library is constructed incrementally. Given the current path library and a task to be executed, the algorithm runs two versions of the planner online, one that plans from scratch and the other one initialized by the library. In [15] a high-dimensional (791) task descriptor is constructed, and the metric between the task descriptors is refined to minimize the necessary refinement of the initial trajectory using  $L_1$  norm, resulting in a sparse metric and hence sparse descriptors. In [4], a subindexing is used to reduce the amount of memory storage and to use the subtrajectories of the original solutions. In robot locomotion [16], a mapping from the task space to the optimal trajectory for cyclic walking is learned using various machine learning algorithms, but the prediction is not re-optimized online. In [17], the initial trajectories for real-time catching are predicted using  $k$ -NN, Support Vector Regression, and GPR.

As compared to the above works, our proposed method has the following differences: (i) none of the above methods attempt to handle the multimodal output cases. We show that BGMR can handle well such cases, (ii) we show that the memory of motion can be used as a metric for choosing optimally between several possible goals, and (iii) we show that using an ensemble of methods to provide the warm-start outperforms the individual methods significantly.

## III. METHOD

Section III-A discusses the main idea of building the memory of motion using function approximation and dimensionality reduction techniques to learn the mapping between the task and the associated robot path. Section III-B explains how the memory of motion can be used as a metric for choosing between different goals. Finally, Section III-C explains how the warm-starting performance can be improved significantly using an ensemble method.

### A. Building a Memory of Motion

To learn the mapping  $f : \mathbf{x} \rightarrow \mathbf{y}$ , we firstly generate a set of tasks  $\{\mathbf{x}\}$  and the corresponding robot paths  $\{\mathbf{y}\}$ . This is done by sampling  $\mathbf{x}$  from a uniform distribution covering the space of possible tasks and run the trajectory optimizer to obtain the robot paths  $\mathbf{y}$  until we obtain  $N$  samples  $(\mathbf{x}, \mathbf{y})$ . Let  $\mathbf{X} = (\mathbf{x}_0, \dots, \mathbf{x}_{N-1})$  and  $\mathbf{Y} = (\mathbf{y}_0, \dots, \mathbf{y}_{N-1})$ . The mapping  $f$  can be learned by training function approximators using the database  $\{\mathbf{X}, \mathbf{Y}\}$ . In this paper we consider three function approximators:  $k$ -NN, GPR, and BGMR.

1) *k-Nearest Neighbor (k-NN)*:  $k$ -NN is a very simple non-parametric method. Given a task  $\mathbf{x}^*$ , the algorithm finds  $K$  samples  $\{\mathbf{x}_k, \mathbf{y}_k\}_{k=1}^K$  in the database  $\{\mathbf{X}, \mathbf{Y}\}$  where  $\{\mathbf{x}_k\}_{k=1}^K$  are the  $K$ -nearest to  $\mathbf{x}^*$  according to a chosen metric (in this paper the Euclidean metric is used). It then predicts the corresponding robot path  $\mathbf{y}^*$  by taking the average  $\mathbf{y}^* = \frac{1}{K} \sum_{k=1}^K \mathbf{y}_k$ . The method is very simple to implement and it works well if there is a sufficiently dense dataset, but it suffers from the curse of dimensionality; as the dimension of  $\mathbf{x}$  increases, the number of data that needs to be stored increases exponentially. This method is mainly considered as the baseline against the next two methods.

2) *Gaussian Process Regressor (GPR)*: Like  $k$ -NN, GPR [18] is a non-parametric method which improves its accuracy as the number of data increases. While having higher computational complexity as compared to  $k$ -NN, GPR tends to better interpolate, resulting in higher approximation accuracy. Given the database  $\{\mathbf{X}, \mathbf{Y}\}$ , GPR assigns a Gaussian prior to the joint probability of  $\mathbf{Y}$ , i.e.,  $p(\mathbf{Y}|\mathbf{X}) = \mathcal{N}(\boldsymbol{\mu}(\mathbf{X}), \mathbf{K}(\mathbf{X}, \mathbf{X}))$ .  $\boldsymbol{\mu}(\mathbf{X})$  is the mean function and  $\mathbf{K}(\mathbf{X}, \mathbf{X})$  is the covariance matrix constructed with elements  $\mathbf{K}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ , where  $k(\mathbf{x}_i, \mathbf{x}_j)$  is the kernel function that measures the similarity between the inputs  $\mathbf{x}_i$  and  $\mathbf{x}_j$ . In this paper we use Radial Basis Function (RBF) as the kernel function, and the mean function  $\boldsymbol{\mu}(\mathbf{X})$  is set to be zero as usually done in GPR.

To predict the output  $\mathbf{y}^*$  given a new input  $\mathbf{x}^*$ , GPR constructs the joint probability distribution of the training data and the prediction, and then conditions on the training data to obtain the predictive distribution of the output,  $p(\mathbf{y}^*|\mathbf{x}^*) \sim \mathcal{N}(\mathbf{m}, \boldsymbol{\Sigma})$ , where  $\mathbf{m}$  is the posterior mean computed as

$$\mathbf{m} = \mathbf{K}(\mathbf{x}^*, \mathbf{X}) \mathbf{K}^{-1}(\mathbf{X}, \mathbf{X}) \mathbf{Y}(\mathbf{X}), \quad (3)$$

and  $\boldsymbol{\Sigma}$  is the posterior covariance which provides a measure of uncertainty on the output. In this work we simply use the posterior mean  $\mathbf{m}$  as the output, i.e.,  $\mathbf{y}^* = \mathbf{m}$ .

While having good approximation accuracy, one major limitation with GPR is that it does not scale well with very large datasets. There are variants of GPR that attempt to overcome this problem, e.g., sparse GPR [19] or using Stochastic Variational Inference (SVI) [20]. More details on GPR can be found in [18] and [21].

3) *Bayesian Gaussian Mixture Regression (BGMR)*: When using RBF as the covariance function, GPR assumes that the mapping from  $\mathbf{x}$  to  $\mathbf{y}$  is smooth and continuous. When this assumption is met, it performs very well, but otherwise it will yield poor results. For example, when there is discontinuity in the mapping or there are multimodal outputs, GPR tends to average the solutions from both sides of the discontinuity or from both modes. This characteristic is also shared by many other function approximators. To handle discontinuity and multimodality problems, using local models is one of the possible solutions. Each local model can be fit to each side of the discontinuity or to each mode.

Gaussian Mixture Regression (GMR) is an example of such local models approaches [22]. It can be seen as a probabilistic mixture of linear regressions. Given the database  $\{\mathbf{X}, \mathbf{Y}\}$  it can be used to construct the joint probability of  $(\mathbf{x}, \mathbf{y})$  as a mixture of Gaussians

$$p(\mathbf{x}, \mathbf{y}) = \sum_{k=1}^K \pi_k \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k), \quad (4)$$

where  $\pi_k$ ,  $\boldsymbol{\mu}_k$ , and  $\boldsymbol{\Sigma}_k$  are the  $k$ -th component's mixing coefficient, mean, and covariance, respectively. Given a query  $\mathbf{x}^*$ , the conditional probability of the output  $\mathbf{y}^*$  is also a mixture of Gaussians.

In GMR, the parameters  $\pi_k$ ,  $\boldsymbol{\mu}_k$  and  $\boldsymbol{\Sigma}_k$  are determined from the data by Expectation-Maximization method, while the number of Gaussians  $K$  is usually determined by the

user. Bayesian GMR (BGMR) [23] is a Bayesian extension of GMR that allows us to estimate the posterior distribution of the mixture parameters (instead of relying on a single point estimate as in GMR). The number of components  $K$  can also be automatically determined from the data. As a Bayesian model, BGMR gives priors to the parameters  $\pi_k$ ,  $\boldsymbol{\mu}_k$  and  $\boldsymbol{\Sigma}_k$ , and computes the posterior distribution of those parameters given the data. In high dimensional problems, the prior reduces the overfitting that commonly occurs with GMR. The prediction  $\mathbf{y}^*$ , given the input  $\mathbf{x}^*$ , is then computed by marginalizing over the posterior distribution and conditioning on  $\mathbf{x}^*$ . The resulting predictive distribution of  $\mathbf{y}$  is a mixture of t-distributions,

$$p(\mathbf{y}|\mathbf{x}^*, \mathbf{X}, \mathbf{Y}) = \sum_{k=1}^K p(k|\mathbf{x}^*, \mathbf{X}, \mathbf{Y}) p(\mathbf{y}|k, \mathbf{x}^*, \mathbf{X}, \mathbf{Y}), \quad (5)$$

where  $p(k|\mathbf{x}^*, \mathbf{X}, \mathbf{Y})$  is the probability of  $\mathbf{x}^*$  belonging to the  $k$ -th component of the mixture, and  $p(\mathbf{y}|k, \mathbf{x}^*, \mathbf{X}, \mathbf{Y})$  is a multivariate t-distribution, the mean of which is linear in  $\mathbf{x}^*$ . We can interpret (5) as  $K$  probabilistic linear regression models, each of which has the probability of  $p(k|\mathbf{x}^*, \mathbf{X}, \mathbf{Y})$ . More details about BGMR can be found in [23].

To obtain a point-prediction  $\mathbf{y}^*$  from (5), there are several approaches. One of the most used is to take the mean of the predictive distribution in (5) using moment matching. While this approach can provide smooth estimates (as required in many applications), the same problems as in GPR will appear in the case of discontinuity and multimodality; taking average in those cases will give us poor results. Instead, we propose to take, as the point prediction, the mean<sup>1</sup> of the component in (5) having the highest probability, which approximately corresponds to the mode of the multimodal distribution. Alternatively, we can also use the mean of each t-distributions as separate predictions, which gives us several possible solutions. In some cases (e.g., when we would like to retrieve all possible solutions) this approach can be very useful, as will be presented in Section IV-A.

4) *Dimensionality reduction*: In our problem, the path  $\mathbf{y} \in \mathbb{R}^{DT}$  is a vector consisting of the sequence of configurations with dimension  $D$  during  $T$  time steps, which can be very high. This motivates us to use dimensionality reduction techniques to reduce the dimension of  $\mathbf{y}$ . For example, when  $T$  is large and the time interval is small, RBF can be used to represent the evolution of each variable as weights of the basis functions. Techniques such as Principal Component Analysis (PCA), Independent Component Analysis, Factor Analysis, and Variational Autoencoder [21] [24] can also be used. The mapping to be learned then becomes the mapping from  $\mathbf{x}$  to  $\hat{\mathbf{y}}$ , where  $\hat{\mathbf{y}}$  is the projection of  $\mathbf{y}$  to the lower dimensional subspace. The advantage is that the memory required to store the data is reduced significantly, while the approximation performance is maintained or even improved because the important correlations between the variables are preserved. In this work, since the number of time steps is not large, we use PCA to reduce the dimension of  $\mathbf{y}$ .

<sup>1</sup>As in Gaussian distribution, the mean of a multivariate t-distribution is also its mode.

**Algorithm 1** Building a Memory of Motion

---

**INPUT:** number of samples  $N$   
**OUTPUT:** the database  $\{\mathbf{X}, \mathbf{Y}\}$  and the function approximator  $\mathbf{f}$

- 1:  $\mathbf{X} = [], \mathbf{Y} = []$
- 2: **for**  $i = 1, 2, \dots, N$  **do**
- 3:   sample a random task  $\mathbf{x}_i$
- 4:   compute the initial guess  $\tilde{\mathbf{y}}_i$  to achieve  $\mathbf{x}_i$  by straight-line motion
- 5:   solve  $\mathbf{x}_i$  using TrajOpt warm-started by  $\tilde{\mathbf{y}}_i$ , to obtain the path  $\mathbf{y}_i$
- 6:   **if**  $\mathbf{y}_i$  is valid **then**
- 7:     add  $(\mathbf{x}_i, \mathbf{y}_i)$  to  $\{\mathbf{X}, \mathbf{Y}\}$
- 8:   **end if**
- 9: **end for**
- 10: apply PCA to  $\mathbf{Y}$  to obtain  $\hat{\mathbf{Y}}$  (*Optional*)
- 11: train the function approximator  $\mathbf{f}$  on  $\{\mathbf{X}, \mathbf{Y}\}$    (*or on  $\{\mathbf{X}, \hat{\mathbf{Y}}\}$  if PCA is used*)

---

**Algorithm 2** Using the Memory as a Metric

---

**INPUT:** A list of goals  $\{\mathbf{x}_j\}_{j=1}^M$ , a function approximator  $\mathbf{f}$   
**OUTPUT:** The optimal goal  $\mathbf{x}^*$  and the corresponding path  $\mathbf{y}^*$

- 1: **for**  $j = 1, 2, \dots, M$  **do**
- 2:   compute the initial guess  $\tilde{\mathbf{y}}_j = \mathbf{f}(\mathbf{x}_j)$
- 3:   compute the cost  $\ell(\tilde{\mathbf{y}}_j)$
- 4: **end for**
- 5:  $j^* \leftarrow \arg \min_j \ell(\tilde{\mathbf{y}}_j)$
- 6:  $\mathbf{x}^* \leftarrow \mathbf{x}_{j^*}$
- 7:  $\tilde{\mathbf{y}}^* \leftarrow \tilde{\mathbf{y}}_{j^*}$
- 8: solve  $\mathbf{x}^*$  using TrajOpt warm-started by  $\tilde{\mathbf{y}}^*$ , to obtain the path  $\mathbf{y}^*$

---

*B. Using the Memory as a Metric*

In some planning problems, there can be several alternative goals to be achieved. For example, in robot drilling task [25], the orientation around the drilling axis is free (the number of possible goals is infinite). A naive way is to choose one of the goals randomly, plan the motion, and if it fails then select another goal. While this is simple to implement, it does not make use of the benefit of having multiple goals. Another method is to plan the paths to each goal and select the one having the smallest cost, but this is computationally expensive. It will be useful, therefore, to have a metric that measures the cost to a given goal. Our idea is to use the memory of motion as the metric.

In Section III-A, function approximators were trained to predict an initial guess to achieve a task  $\mathbf{x}$ . The possible goals can then be formulated as multiple tasks  $\{\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_{M-1}\}$ . For each task  $\mathbf{x}_i$ , the function approximator predicts the initial guess  $\tilde{\mathbf{y}}_i$  corresponding to the task, and the cost  $\ell(\tilde{\mathbf{y}}_i)$  can be computed. The initial guess  $\tilde{\mathbf{y}}_i^*$  and the corresponding task  $\mathbf{x}_i^*$  with the lowest cost is then taken as the chosen goal to be given to the trajectory optimizer. Since the cost computation (the total discrete velocity in (2)) can be done quickly relative to optimization time, this approach can yield significant improvements to the trajectory optimizer performance.

*C. Using Ensemble Method to Provide Warm-Start*

In machine learning, methods such as AdaBoost [26] and Random Forests [27] have shown that using an ensemble of methods often yields improved performances as compared to

**Algorithm 3** Ensemble Method

---

**INPUT:** Task  $\mathbf{x}^*$ , a list of function approximators  $\{\mathbf{f}_j\}_{j=1}^M$   
**OUTPUT:** The path  $\mathbf{y}^*$  that accomplishes the task  $\mathbf{x}^*$

- 1: **for all**  $j = 1, 2, \dots, M$  **do in parallel**
- 2:   compute the initial guess  $\tilde{\mathbf{y}}_j = \mathbf{f}_j(\mathbf{x}^*)$
- 3:   solve  $\mathbf{x}^*$  using TrajOpt warm-started by  $\tilde{\mathbf{y}}_j$ , to obtain the path  $\mathbf{y}_j$
- 4:   **if**  $\mathbf{y}_j$  is valid **then**
- 5:      $\mathbf{y}^* = \mathbf{y}_j$
- 6:     Terminate the parallel execution
- 7:   **end if**
- 8: **end for**

---

choosing a single method. We propose to use an ensemble method where we run multiple trajectory optimizations in parallel, each one warm-started by one of the function approximators in Section III-A, and once one of them finds a successful path the others are terminated. Since each function approximator has different learning characteristics, combining them in this way can significantly improve the motion planning performance. The method in Section III-B can also be used as one of the ensemble's component.

## IV. EXPERIMENTS

To evaluate the proposed method, we consider several examples of motion planning for PR2 and Atlas robots. TrajOpt [1] is used as the trajectory optimizer to be warm-started. The output is the robot path that accomplishes the given task. In this paper we only work with robot path as the output, but the method can also be applied to robot trajectory.

We consider 5 motion planning cases presented in ascending order of complexity. Each case is chosen to demonstrate certain characteristics of the proposed method. For each case, we follow the following procedures. First we generate the dataset by randomly sampling  $N_{\text{train}}$  tasks from a uniform distribution and run TrajOpt to find the paths achieving the tasks. The number of time steps  $T$  is set to 30, except for Atlas ( $T = 15$ ). In all cases, the cost is defined as the discrete velocity of the states, as defined in (2). The number of  $N_{\text{train}}$  is different for each case, depending on the complexity of the task. The function approximators are then trained with or without PCA using the dataset. We heuristically set 50 components for the PCA; for the  $k$ -NN, we use  $K = 1$ .

To validate the performance, we sample  $N_{\text{test}}$  random tasks and use the various methods to warm-start TrajOpt. The solutions are compared in terms of *convergence time*, *success rate* and *cost*. The planning is considered successful if the solution is feasible. The comparison results are presented in the Tables I-VI. The values are averaged over  $N_{\text{test}}$  tasks, and the standard deviation is also given for the convergence time and the cost. In the presented results, we use the label 'STD' to refer to the solution obtained by warm-starting the solver with a straight-line path (via waypoint, if any), and the names of the function approximators for the rest. The subscript 'PCA' is added when PCA is used. The query time for predicting the warm-starts by each method is negligible w.r.t. the convergence time, i.e. less than 5 ms for most methods, except for BGMR without PCA (around 20ms), so they are not included in the

Table I  
BASE MOTION PLANNING, ONE WAYPOINT.

Method	Success (%)	Conv. time (s)	Cost (rad/s)
STD	80.0	0.55±0.29	1.37±0.37
$k$ -NN	93.0	0.35±0.20	1.45±0.47
GPR	96.0	0.37±0.15	<b>1.32±0.36</b>
BGMR	<b>97.0</b>	<b>0.32±0.14</b>	1.34±0.35

Table II  
BASE MOTION PLANNING, TWO WAYPOINT.

Method	Success (%)	Conv. time (s)	Cost (rad/s)
STD	79.0	0.53±0.23	1.43±0.37
$k$ -NN	<b>95.0</b>	0.32±0.16	1.53±0.62
GPR	00.0	-	-
BGMR	94.0	<b>0.31±0.15</b>	<b>1.33±0.40</b>

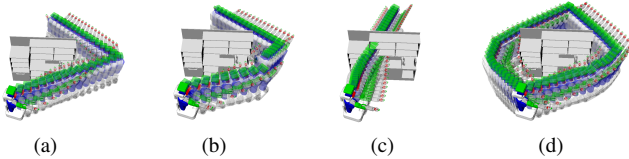


Figure 2. Motion planning for the PR2 mobile base. Warm-start produced by (a) straight-line with waypoint, (b)  $k$ -NN, (c) GPR and (d) BGMR.

comparison. The codes to run the experiments are provided in [https://github.com/teguhSL/memmo\\_for\\_trajopt\\_codes](https://github.com/teguhSL/memmo_for_trajopt_codes), and the videos are submitted as supplementary file.

### A. Base motion planning

The task is to plan the motion for the PR2 mobile base from a random pose in front of the kitchen to another random pose behind the kitchen (Fig. 1a). In this case, the state  $\mathbf{q}$  is the 3 DoF planar pose of the base. The task descriptor is then  $\mathbf{x} = (\mathbf{q}_{\text{init}}^{\top}, \mathbf{q}_{\text{goal}}^{\top})^{\top}$ . The database is constructed with  $N_{\text{train}} = 200$  samples and the evaluation is performed with  $N_{\text{test}} = 100$ . Although this is an easy problem, TrajOpt actually finds it difficult to solve without a proper initialization. For example, initializing TrajOpt with a straight-line interpolation from  $\mathbf{q}_{\text{init}}$  to  $\mathbf{q}_{\text{goal}}$  never manages to find a feasible solution because it results in a path that moves the robot through the kitchen while colliding, and the solver get stuck in poor local optima due to the conflicting gradients. To obtain better initialization for building the database, we initialize TrajOpt with two manually chosen waypoints on the left and on the right of the kitchen ( $\mathbf{q}_{\text{left}}$  and  $\mathbf{q}_{\text{right}}$ , respectively).

We consider two cases of building the database: in the first one, we only use  $\mathbf{q}_{\text{right}}$  as waypoint, while in the second we use both  $\mathbf{q}_{\text{left}}$  and  $\mathbf{q}_{\text{right}}$ . We initialize TrajOpt with the straight-line motion from  $\mathbf{q}_{\text{init}}$  to the waypoint and from the waypoint to  $\mathbf{q}_{\text{goal}}$ . With this setting we build the database, train the function approximators, and obtain the results as shown in Table I and II.

In the first case, the mapping from  $\mathbf{x}$  to  $\mathbf{y}$  is unimodal because all movements go through the right. Table I shows that the performance of  $k$ -NN, GPR and BGMR are quite

Table III  
PLANNING FROM FIXED  $\mathbf{q}_{\text{INIT}}$  TO RANDOM  $\mathbf{q}_{\text{GOAL}}$ .

Method	Success (%)	Conv. time (s)	Cost (rad/s)
STD	80.0	0.77±0.37	1.83±0.61
$k$ -NN	91.2	<b>0.58±0.29</b>	1.93±0.69
GPR	92.4	0.65±0.25	1.84±0.57
GPR <sub>PCA</sub>	<b>92.8</b>	0.66±0.26	<b>1.83±0.57</b>
BGMR	88.8	0.64±0.26	1.85±0.56
BGMR <sub>PCA</sub>	92.0	0.67±0.26	1.84±0.58

Table IV  
PLANNING FROM RANDOM  $\mathbf{q}_{\text{INIT}}$  TO  $\mathbf{q}_{\text{GOAL}}$ .

Method	Success (%)	Conv. time (s)	Cost (rad/s)
STD	75.2	0.82±0.43	<b>1.31±0.74</b>
$k$ -NN	65.6	1.16±0.58	1.55±0.88
GPR	85.6	0.85±0.39	1.32±0.74
GPR <sub>PCA</sub>	88.0	<b>0.81±0.36</b>	1.33±0.73
BGMR	84.0	0.81±0.40	1.34±0.76
BGMR <sub>PCA</sub>	78.3	0.88±0.42	1.39±0.78
Waypoints	94.0	1.52±0.67	1.83±1.34
Ensemble	<b>97.2</b>	1.06±0.41	1.42±0.82

similar. In the second case, however, the output is multimodal because the database contains two possible ways (modes) to accomplish the same task. This affects GPR significantly (see Table II), as GPR averages both modes and outputs a path that goes through the kitchen, while  $k$ -NN and BGMR are not affected.  $k$ -NN does not average the modes because we use  $K = 1$ , while BGMR overcomes the multimodality by constructing local models for each mode automatically.

Fig. 2 shows the examples of warm-starts produced by each method in the second case. As expected, GPR provides a warm-start that goes through the kitchen (hence 0 success rate). With BGMR, if we retrieve the components with the two highest probability, both possible solutions are obtained.

### B. Planning from a fixed initial configuration to a random goal configuration

Here  $\mathbf{q}$  consists of 14 joint angles of the two 7 DoFs arms of PR2. The task  $\mathbf{x}$  is to move from a fixed  $\mathbf{q}_{\text{init}}$  to a random goal configuration  $\mathbf{q}_{\text{goal}}$  (i.e.  $\mathbf{x} = \mathbf{q}_{\text{goal}}$ ). The database is constructed with  $N_{\text{train}} = 500$ , and the evaluation results with  $N_{\text{test}} = 250$  are presented in Table III.

Since each PR2 arm is redundant, the path from  $\mathbf{q}_{\text{init}}$  to  $\mathbf{q}_{\text{goal}}$  can be multimodal, which may pose a problem for GPR. However, Table III shows that GPR and BGMR perform similarly. This is due to the fact that although redundant robots can achieve a goal configuration in many different ways, planning using optimization here results in similar motions for similar goal configurations. The use of PCA does not improve the performance significantly, but it still helps to reduce the size of the data. In this case, for each path it reduces the number of variables from  $30 \times 14$  ( $D \times T$ ) to 50 (number of PCA components), more than 8 times reduction while maintaining the performance.

Table V  
PLANNING FROM FIXED  $\mathbf{q}_{\text{INIT}}$  TO RANDOM CARTESIAN GOAL.

Method	Success (%)	Conv. time (s)	Cost (rad/s)
STD	65.2	1.10±0.62	1.86±0.86
$k$ -NN	73.6	1.28±0.96	1.84±0.81
GPY	66.4	1.81±0.96	1.87±0.87
GPY <sub>PCA</sub>	66.8	1.68±0.98	1.78±0.83
BGMR	74.4	1.37±0.82	1.82±0.86
BGMR <sub>PCA</sub>	77.2	1.33±0.75	1.84±0.80
METRIC GPR <sub>PCA</sub>	86.8	<b>0.70±0.30</b>	<b>1.49±0.56</b>
Ensemble	<b>98.0</b>	1.50±0.60	1.60±0.68

Table VI  
PLANNING THE MOTION OF ATLAS FROM FIXED  $\mathbf{q}_{\text{INIT}}$  TO RANDOM CARTESIAN GOAL.

Method	Success (%)	Conv. time (s)	Cost (rad/s)
STD	50.8	6.31±3.90	0.12±0.07
$k$ -NN	58.8	1.48±1.39	0.11±0.06
GPY	54.4	<b>1.29±1.09</b>	<b>0.10±0.05</b>
GPY <sub>PCA</sub>	60.0	1.54±1.46	0.11±0.05
BGMR	56.4	1.32±1.57	<b>0.10±0.05</b>
BGMR <sub>PCA</sub>	58.0	1.36±1.16	0.11±0.06
Ensemble	<b>71.2</b>	1.46±1.40	0.12±0.06

### C. Planning from a random initial configuration to a random goal configuration

To proceed with a more complex case, the task here is to plan a path from a random initial configuration  $\mathbf{q}_{\text{init}}$  to a random target  $\mathbf{q}_{\text{goal}}$ . The task  $\mathbf{x}$  consists of the initial and goal configurations,  $\mathbf{x} = (\mathbf{q}_{\text{init}}^\top, \mathbf{q}_{\text{goal}}^\top)^\top$ . The database is constructed with  $N_{\text{train}} = 500$  and evaluated with  $N_{\text{test}} = 250$ . The result is presented in Table IV.

$k$ -NN performs poorly here, similar to STD, due to the dimension of the input space  $\mathbf{x}$  that is much larger as compared to Section IV-B. To achieve good performance,  $k$ -NN requires a much denser dataset. GPR outperforms BGMR by a wide margin.

The last row of Table IV shows the result of the ensemble method described in Section III-C. Given an input  $\mathbf{x}^*$ , the method uses all function approximators to provide different warm-starts, each of which is used to initialize an instance of TrajOpt in parallel. Once a valid solution is obtained, the other instances of TrajOpt are terminated. This method results in a huge boost of the success rate, with comparable convergence time and cost to the other methods. As comparison, we also include here the standard multiple initializations suggested by TrajOpt (labeled as ‘waypoints’). Each initialization is created by interpolating through a waypoint that is manually defined. While the success rate is high, the convergence time and the cost increase significantly. On the contrary, each initialization in the ensemble method has a good probability of being close to the optimal solution, resulting in lower cost and convergence time.

### D. Planning to Cartesian goals from a fixed initial configuration

In Section IV-B and IV-C, we use TrajOpt to plan to goals in configuration space. In practical situations, however, the task is often to reach a certain Cartesian pose using the end-effector (e.g., to pick an object on the shelf), instead of

planning to a specific joint configuration. One way to solve this problem is to first compute a configuration that achieves the Cartesian pose using an inverse kinematic solver and plan to this configuration, but it does not make use of the flexibility inherent in the task. TrajOpt has an option to plan directly to a Cartesian goal, but it typically requires longer convergence time and lower success rate than planning to a joint configuration goal.

We present two approaches to use the memory of motion in this problem. In the first approach, we rely on the similar procedure as in previous cases: we formulate the task as  $\mathbf{x} = (\mathbf{p}_{\text{left}}^\top, \mathbf{p}_{\text{right}}^\top)^\top$  where  $\mathbf{p}_{\text{left}}$  and  $\mathbf{p}_{\text{right}}$  are the Cartesian positions of the right and left hand of PR2. The database is then constructed with  $N_{\text{train}} = 1000$  and the function approximators are trained. In this approach, TrajOpt plans to a Cartesian goal directly. The second approach relies on the fact that a Cartesian goal corresponds to multiple goals in configuration space. In Section IV-B we have already constructed several function approximators that can predict an initial guess  $\tilde{\mathbf{y}} = \tilde{\mathbf{q}}_{0:T}$ , given a goal  $\mathbf{q}_{\text{goal}}$  in configuration space. The second approach uses one of them as a *metric* (Sect. III-B) to choose between the different goals in configuration space. First, given a Cartesian goal  $\mathbf{x}$ , we run an inverse kinematic solver to find  $M = 5$  joint configurations that satisfy this pose. For each joint configuration, we use the function approximator to predict the initial guess of the robot path to reach that configuration, and we compute the cost of that path. Finally, the goal configuration and the path with the lowest cost are chosen, and TrajOpt is run to reach this goal configuration with the given path as the warm-start. Note that in this second approach, TrajOpt plans to a joint configuration instead of a Cartesian goal. For this approach we choose the method GPR<sub>PCA</sub> from the Section IV-B, and use the term ‘METRIC GPR<sub>PCA</sub>’ to differentiate from the first approach (denoted in standard notation).

We present the results in Table V with  $N_{\text{test}} = 250$ . Among the methods using the first approach, we note that BGMR yields better result than GPR because the mapping from the Cartesian goal  $\mathbf{x}$  to the robot path  $\mathbf{y}$  here is multimodal, as planning to a Cartesian pose has more redundancy as compared to planning to a joint configuration. This again demonstrates that BGMR handles multimodal output better than GPR. However, the second approach METRIC GPR<sub>PCA</sub> outperforms even BGMR. The improvement over the first approach is very significant in all three criteria. This demonstrates that using the memory as a metric to choose the optimal goal results in large improvements. We point out that the additional computational time required to find  $M = 5$  IK solutions and the corresponding warm-starts is only around 0.1 s, which is negligible compared to the convergence time. Finally, we use the ensemble method that uses all function approximators in parallel, including METRIC GPR<sub>PCA</sub>. This boosts the success rate to 98%.

### E. Planning whole-body motion for an Atlas robot

Finally, we also applied our method for planning the motion of the 34-DoFs Atlas robot (28-DoFs joints and 6-DoFs

root pose). We consider the same task as in Section IV-D, i.e. planning from a fixed initial configuration to a random Cartesian pose, in this case chosen to be the location of Atlas' right hand. The task  $\mathbf{x} = (p_x, p_y, p_z)^\top$  corresponds to the target position of Atlas' right hand, while the orientation is not constrained. The feet location are fixed, while the Zero Moment Point (ZMP) is constrained to be between the two feet location. We use here the first approach as explained in Section IV-D, i.e. treating it as a regression problem where the input  $\mathbf{x}$  is the Cartesian goal and the output  $\mathbf{y}$  is the trajectory, and use the various function approximators to predict the initial guesses. The database is constructed with  $N_{\text{train}} = 1000$  and the evaluation is performed with  $N_{\text{test}} = 250$ . The results are presented in Table VI.

$k$ -NN performs quite well, as the input size of  $\mathbf{x}$  is small (the position of the hand is constrained to be inside the shelf). Unlike in Section IV-D, the performance of GPR and BGMR are quite similar, although the goals are also in the Cartesian space. This is due to the difference in the implementation; in Section IV-D, given a Cartesian goal, we use an inverse kinematic solver to calculate the joint configuration that satisfies this goal, and calculate the initial guess as straight-line interpolation from the fixed initial configuration to the goal configuration. This initial guess is used when building the database. Due to the redundancy of the PR2 dual arm, similar Cartesian goals can correspond to very different joint configurations, resulting in the multimodality of the solutions in the database. In this Atlas experiment, however, we do not provide initial guesses to TrajOpt when building the database, so TrajOpt always tries to solve the problem with zero initialization. This results in more uniform solutions, and hence GPR can still perform quite well. Finally, using the ensemble method again shows superior results, giving us an increase of the success rate by more than 10%.

Planning for such high DoFs problem with many constraints (feet location, ZMP constraint, kinematic constraint) requires quite a lot of computational time ( $\sim 6.3$  s in average without warm-start). Using the memory of motion in this complex task further exemplify the benefit of the approach, as our method speeds up the computational time significantly by more than four times faster. We note that the tasks are sampled randomly, and there is no guarantee that the task is indeed feasible. This explains why even the best method (i.e. the ensemble method) only achieves  $\sim 70\%$  success rates.

## V. DISCUSSIONS

### A. Choice of function approximators

In Section IV, we have compared the performance of  $k$ -NN, GPR and BGMR over different tasks, and shown that they have different characteristics. When the dataset is quite dense or the input space is small,  $k$ -NN usually manages to obtain good performance (as shown in Section IV-A, IV-B, and IV-D), while for larger input space (Section IV-D) it does not yield good results. GPR performs the best when the output is unimodal (Section IV-B and IV-C), while for multimodal output BGMR has a better performance than GPR (Section IV-D). This comparison can guide us to select the best method

for each task. However, it may not be obvious whether a given task (and its solution) is unimodal or multimodal (e.g. compare Section IV-D and IV-E). A better way is to combine the different methods via an ensemble method, as we have shown in this paper.

### B. Data requirement

In Fig. 3, we plot the performance of various methods against the number of training samples, with STD given as the baseline. We choose the task in Section IV-C, since it has the largest input space among the other tasks. It is interesting that when the training size is small, GPR performs quite well, while  $k$ -NN and BGMR are even worse than STD. As training size increases,  $k$ -NN and BGMR start to approach the performance of GPR. On the contrary, the performance of the ensemble method is quite stable even when the training size is small. As the training size grows, its convergence time decreases, while the success rate is already high even when the training size is small.

### C. Ensemble method

Using an ensemble method for motion planning has been explored in [28], which uses an ensemble of motion planners. While such approach also manages to boost the performance successfully, it is not easy to design and set up several motion planners for a given task. On the contrary, many function approximators are available and can be used easily, since our problem is formulated as a standard regression problem. We only need to configure one motion planner (in this work, TrajOpt, but other optimization frameworks can also be used) for a given task, unlike in [28]. Another benefit of our ensemble method is that each of the ensemble's component starts from an initial guess that has good probability of being close to the optimal solution. This reduces the average computational time, as we have shown by comparing it against the multiple waypoints initialization in Table IV.

### D. Dynamic environment

In this work we assume that the environment is static, so that the trajectories previously planned remain valid. When the environment changes, a new memory of motion has to be built. For the simple example in Section IV-A, building the memory takes only  $\sim 3$  minutes of computational time, but complex example such as Section IV-E takes  $\sim 3$  hours. While parallelization can be used to speed up the building process, more effective strategies would be interesting to explore. In [29], an efficient way of updating a dynamic roadmap when the environment changes is presented. Such method can possibly be used to modify the existing memory of motion, so that we do not have re-build from scratch but only modify those affected. Alternatively, when the environment largely remain the same but a few obstacles are moving (as in many real tasks), we can include these obstacles' locations as additional inputs to the regression problem, at the expense of larger input size. We will explore these ideas in our future work.

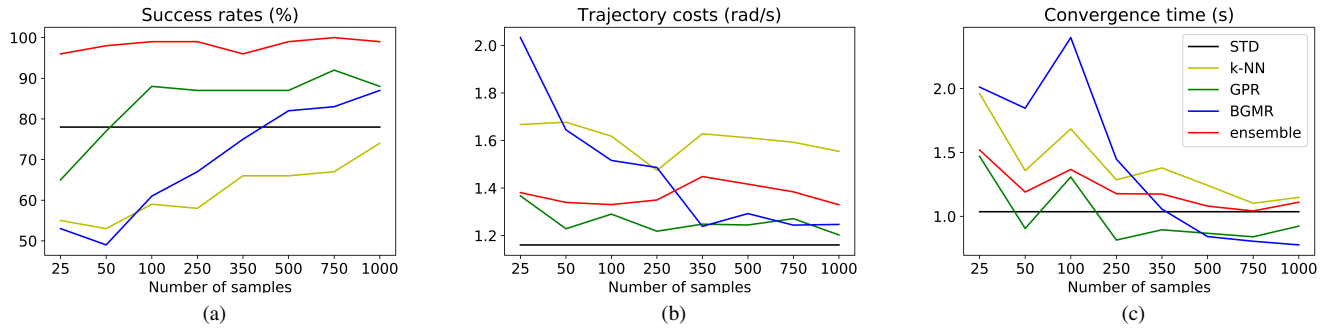


Figure 3. Performance comparison against the number of training samples  $N_{\text{train}}$

## VI. CONCLUSION

We have presented an approach to build a memory of motion to warm-start trajectory optimization solver, and demonstrate through experiments with PR2 and Atlas robots that the warm-start can improve the solver’s performance. Function approximators and dimensionality reduction are used to learn the mapping between the task descriptor and the corresponding robot path. Three function approximators are considered:  $k$ -NN as baseline, GPR, and BGMR, and their different characteristics have been discussed. The use of PCA also improves the solution, although not very significantly, while reducing the memory storage. We have also shown that we can use the memory of motion as a metric to choose optimally between several alternative goals, and this results in a significantly improved performance for the case of Cartesian goal planning. Finally, the different function approximators can be combined as an ensemble method, which boosts the success rate significantly.

## REFERENCES

- [1] 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 *Proc. R:SS*, vol. 9, no. 1, 2013, pp. 1–10.
- [2] 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,” *Intl Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1164–1193, 2013.
- [3] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, “Stomp: Stochastic trajectory optimization for motion planning,” in *Proc. IEEE ICRA*, 2011, pp. 4569–4574.
- [4] W. Merkt, V. Ivan, and S. Vijayakumar, “Leveraging precomputation with problem encoding for warm-starting trajectory optimization in complex environments,” in *Proc. IEEE/RSJ IROS*, 2018, pp. 5877–5884.
- [5] M. Stolle and C. G. Atkeson, “Policies based on trajectory libraries,” in *Proc. IEEE ICRA*, 2006, pp. 3344–3349.
- [6] C. Liu and C. G. Atkeson, “Standing balance control using a trajectory library,” in *Proc. IEEE/RSJ IROS*, 2009, pp. 3031–3036.
- [7] N. Mansard, A. Del Prete, M. Geisert, S. Tonneau, and O. Stasse, “Using a memory of motion to efficiently warm-start a nonlinear predictive controller,” in *Proc. IEEE ICRA*, 2018, pp. 2986–2993.
- [8] D. Forte, A. Gams, J. Morimoto, and A. Ude, “On-line motion synthesis and adaptation using a trajectory database,” *Robotics and Autonomous Systems*, vol. 60, no. 10, pp. 1327–1339, 2012.
- [9] A. J. Ijspeert, J. Nakanishi, and S. Schaal, “Learning rhythmic movements by demonstration using nonlinear oscillators,” in *Proc. IEEE/RSJ IROS*, 2002, pp. 958–963.
- [10] L. E. Kavraki, P. Svestka, J. . Latombe, and M. H. Overmars, “Probabilistic roadmaps for robot path planning,” *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.
- [11] S. M. LaValle, “Rapidly-exploring random trees: A new tool for path planning,” Iowa State University, Computer Science Department, Tech. Rep. TR 98-11, 1998.
- [12] S. R. Martin, S. E. Wright, and J. W. Sheppard, “Offline and online evolutionary bi-directional rrt algorithms for efficient re-planning in dynamic environments,” in *Proc. IEEE CASE*, 2007, pp. 1131–1136.
- [13] M. Phillips, B. J. Cohen, S. Chitta, and M. Likhachev, “E-graphs: Bootstrapping planning with experience graphs,” in *Proc. R:SS*, vol. 5, no. 1, 2012, p. 110.
- [14] D. Berenson, P. Abbeel, and K. Goldberg, “A robot path planning framework that learns from experience,” in *Proc. IEEE ICRA*, 2012, pp. 3671–3678.
- [15] N. Jetchev and M. Toussaint, “Trajectory prediction: learning to map situations to robot trajectories,” in *Proc. ICML*, 2009, pp. 449–456.
- [16] A. Werner, D. Trautmann, D. Lee, and R. Lampariello, “Generalization of optimal motion trajectories for bipedal walking,” in *Proc. IEEE/RSJ IROS*, 2015, pp. 1571–1577.
- [17] R. Lampariello, D. Nguyen-Tuong, C. Castellini, G. Hirzinger, and J. Peters, “Trajectory planning for optimal robot catching in real-time,” in *Proc. IEEE ICRA*, 2011, pp. 3719–3726.
- [18] C. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*. Cambridge, MA, USA: MIT Press, 2006.
- [19] J. Quiñero-Candela and C. E. Rasmussen, “A unifying view of sparse approximate gaussian process regression,” *Journal of Machine Learning Research*, vol. 6, no. Dec, pp. 1939–1959, 2005.
- [20] J. Hensman, N. Fusi, and N. D. Lawrence, “Gaussian processes for big data,” in *Conference on Uncertainty in Artificial Intelligence*, 2013, pp. 282–290.
- [21] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.
- [22] S. Calinon, “A tutorial on task-parameterized movement learning and retrieval,” *Intelligent Service Robotics*, vol. 9, no. 1, pp. 1–29, 2016.
- [23] E. Pignat and S. Calinon, “Bayesian Gaussian mixture model for robotic policy imitation,” *IEEE Robotics and Automation Letters (RA-L)*, pp. 1–7, 2019.
- [24] C. Doersch, “Tutorial on variational autoencoders,” *arXiv preprint arXiv:1606.05908*, 2016.
- [25] F. Suárez-Ruiz, T. S. Lembono, and Q.-C. Pham, “Robotsp—a fast solution to the robotic task sequencing problem,” in *Proc. IEEE ICRA*, 2018, pp. 1611–1616.
- [26] Y. Freund and R. E. Schapire, “A decision-theoretic generalization of on-line learning and an application to boosting,” *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, 1997.
- [27] L. Breiman, “Random forests,” *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.
- [28] S. Choudhury, S. Arora, and S. Scherer, “The planner ensemble: Motion planning by executing diverse algorithms,” in *Proc. IEEE ICRA*, 2015, pp. 2389–2395.
- [29] Y. Yiming, W. Merkt, V. Ivan, L. Zhibin, and S. Vijayakumar, “Hdrm: A resolution complete dynamic roadmap for real-time motion planning in complex scenes,” *IEEE RA-L*, vol. 3, no. 1, 2017.