electronics

MDPI

Article

Voronoi Tessellation for Efficient Sampling in Gaussian Process-Based Robotic Motion Planning

Jee-Yong Park [1], Hoosang Lee [1], Changhyeon Kim [1] and Jeha Ryu [2,*]

[1] School of Integrated Technology, Gwangju Institute of Science and Technology, Gwangju 61005, Republic of Korea; jy.park21@gm.gist.ac.kr (J.-Y.P.); hoosang223@gm.gist.ac.kr (H.L.); changhyeon@gm.gist.ac.kr (C.K.)

[2] School of Integrated Technology, AI Graduate School, Gwangju Institute of Science and Technology, Gwangju 61005, Republic of Korea

* Correspondence: ryu@gist.ac.kr; Tel.: +82-62-715-2389

Abstract: On-line motion planning in dynamically changing environments poses a significant challenge in the design of autonomous robotic system. Conventional methods often require intricate design choices, while modern deep reinforcement learning (DRL) approaches demand vast amounts of robot motion data. Gaussian process (GP) regression-based imitation learning approaches address such issues by harnessing the GP's data-efficient learning capabilities to infer generalized policies from a limited number of demonstrations, which can intuitively be generated by human operators. GP-based methods, however, are limited in data scalability as computation becomes cubically expensive as the amount of learned data increases. This issue is addressed by proposing Voronoi tessellation sampling, a novel data sampling strategy for learning GP-based robotic motion planning, where spatial correlation between input features and the output of the trajectory prediction model is exploited to select the data to be learned that are informative yet learnable by the model. Where the baseline is set by an imitation learning framework that uses GP regression to infer trajectories that learns policies optimized via a stochastic, reward-based optimization algorithm, experimental results demonstrate that the proposed method can learn optimal policies spanning over all of feature space using fewer data compared to the baseline method.

Keywords: path planning; imitation learning; reinforcement learning; Gaussian process regression

check for updates

Citation: Park, J.-Y.; Lee, H.; Kim, C.; Ryu, J. Voronoi Tessellation for Efficient Sampling in Gaussian Process-Based Robotic Motion Planning. *Electronics* **2023**, *12*, 4122. https://doi.org/10.3390/electronics12194122

Academic Editors: Somaiyeh MahmoudZadeh, Adham Atyabi and Mohd Nadhir Ab Wahab

Received: 14 August 2023
Revised: 20 September 2023
Accepted: 26 September 2023
Published: 2 October 2023

1. Introduction

On-line robotic trajectory planning, especially under dynamically changing environment settings, is a challenging core problem in the field of robotic autonomy. Many conventional robotic systems have been used in static environment scenarios such as automated factory production lines in which the off-line preplanning strategies for its paths are feasible and adequate [1]. With recent advances in the domain of AI and reinforcement learning, however, efforts are being made to address scenarios where such assumptions do not hold.

Most conventional non-data-driven solutions for dynamic adaptive motion planning such as [2] require considerable effort in elaborate tuning and design tailored to the system. On the other hand, many state-of-the-art systems employ deep reinforcement learning (DRL) approaches, harnessing the generalization capability of neural network models to learn the particular characteristics of the problem and the system at hand and in many cases creating end-to-end solutions [3]. DRL approaches, however, require an extensive amount of training data which, depending on the problem, may not be feasible.

In such cases, Gaussian process (GP) regression is often explored as an alternative to DRL as it shows powerful generalization capabilities given the small amount of data. This data efficiency is especially well exploited in imitation learning scenarios, where the model must infer generalized policies from a limited number of demonstrations. However, where GP-based approaches excel in data efficiency, they are limited in data scalability, as the

required matrix inversion computation becomes highly expensive as the amount of data increases [4]. This issue extends to limitation in terms of scalability in the dimensionality of the problem, as higher dimensionality of feature space warrants a larger amount of data needed to cover the feature space. This study thus aims to address the data scalability issue of GP-based dynamic motion planning.

1.1. Literature Review

The most widely used conventional motion planners generally fall into the category of either search-based methods or optimization-based methods [5]. Graph-based search methods [6,7] quantize the configuration space in graph nodes and iteratively search for the best path in the graph, while sampling-based search methods [8,9] iteratively create tree-like graphs from given starting points such as initial and global configurations to form a global path. While widely used for ease of implementation, these methods can become computationally expensive; are only asymptotically optimal, meaning that the paths are initially largely suboptimal; and show jerky behaviors due to the discretized nature of the methods. Local optimization in path planning has become popular for dynamic local path planning as the high computational costs of the optimization have become feasible. Timed elastic bands [10] is a popular method as it incorporates the kinodynamic constraints of the robot into the optimization. Such methods, however, require a separate global path planner to be implemented in a hierarchical framework, and tuning the optimization parameters adds to the complexity of the planner design.

In this regard, GP-based motion planners [11–14] provide a powerful solution in that they can generalize smooth, optimal motion policies given the limited amount of initial data, which can be generated intuitively by human operators. These, however, suffer from the aforementioned limitation in terms of scalability. Such a limitation in GP-based approaches can be addressed by either improving the efficiency of the computation algorithm or employing sampling strategies that obtain only the most necessary training data. There exist several works [4,15,16] that attempt to address the data scalability issue from a computational efficiency perspective.

To the authors' knowledge, however, there has been limited research investigating data sampling methods for enhanced data efficiency and the exploration–exploitation trade-off in the GP-based imitation learning of robotic motion planning. As Gaussian process regression learning offers higher data efficiency compared to neural-network-based models, most conventional works do not focus on data sampling strategies for training their models. Refs. [12,13], for example, which both propose GP-based probabilistic trajectory planning frameworks for dynamic adaptation, both employ i.i.d. uniform random sampling or grid sampling approaches to obtain training data.

Especially in cases such as [13] where the GP regression model learns optimized motion policies and thus needs to infer a reasonably optimal output for the optimization algorithm to converge to a learnable outcome, it is important to ensure that the model learns data that are learnable yet informative. One way to do so is by employing an active learning strategy, in which the data to be learned are determined by querying the model for uncertainty w.r.t. unseen data. Again, there are few works on employing active learning in GP regression models for robotic motion planning, but [17,18] use the learning strategy to approximate reward functions in RL quickly and in a data-efficient manner.

Voronoi tessellation, on the other hand, is classically employed in robotic path planning as a method to partition the configuration space and generate graphs of possible paths. These include methods that use generalized Voronoi diagrams [19] to generate collision-free paths and those that employ Voronoi tessellation as a means to discretize the configuration space to apply graph search path planning [5,20]. Whereas such an application of Voronoi tessellation in robotic path planning has been extensively studied, there have been few efforts in its application in the learning of motion policies.

1.2. Motivation

While these methods do not extend the concept of Voronoi tessellation to sampling training features for the learning of motion planning, it can be reasoned that where these methods employ Voronoi tessellation to interpolate points between configuration space via-points and obstacles, it can also be used to interpolate points between features in higher-dimensional feature space where features represent the configuration space coordinates of via-points. Here, GP regression itself can also be interpreted as an interpolation machine that learns the correlation of input features and the outputs to find "midpoints" between learned outputs given an unseen set of features. As such, where the initial data features are conditioned to have an even spread over the feature space, Voronoi tessellation can be used to acquire features that are adequately conditioned in between the known features, such that the model can learn an even spread of data.

Hence, this nature of Voronoi tessellation as a spatial interpolation method is utilized to propose a sampling strategy that selects only the most necessary data for the GP regression model to learn, i.e., data that contribute to a relatively even distribution of learned features along the feature space.

1.3. Contribution

This study proposes a novel training scheme to address the data scalability issue of a GP regression model in dynamic robotic motion planning. This continuous-space data sampling approach uses Voronoi tessellation to leverage the spatial correlation between the input and the output of the model and balance exploration and exploitation. As baseline, Ref. [13] provides a simple yet realistically applicable baseline experiment for a motion policy learning framework, where the model uses GP regression to infer motion policies for unseen environment configurations and consequently learns optimized versions of the inferred policies. Comparative experimental evaluation is performed to show that policies derived with the proposed method can indeed show similar optimality with fewer learned data, thereby limiting the amount of computation required for real-time inference.

2. Preliminaries

2.1. Probabilistic Motion Primitives

Probabilistic motion primitive (ProMP) is a Bayesian model that represents time-varying variance over a set of trajectories, represented as a joint probability distribution of Gaussian basis functions that span over degrees of freedom (DOFs) of whatever space the motion is represented in [21]. Alternatively, it is an approximation of a set of trajectories as a weighted sum of Gaussian basis functions evenly spaced along the time axis [13]. The encoding of the probability distribution over trajectory $\boldsymbol{\tau}$ given weight parameter vector $\boldsymbol{w}$ is

$$\boldsymbol{y}_t = \begin{pmatrix} q_t \\ \dot{q}_t \end{pmatrix} = \boldsymbol{\Phi}_t \boldsymbol{w} + \boldsymbol{\epsilon}_y \tag{1}$$

$$p(\boldsymbol{\tau}|\boldsymbol{w}) = \prod_t \mathcal{N}(\boldsymbol{y}_t|\boldsymbol{\Phi}_t \boldsymbol{w}, \boldsymbol{\Sigma}_y) \tag{2}$$

where $\boldsymbol{\Phi}_t$ denotes the basis function matrix, q_t the positional configuration of agent, n the number of basis functions, and $\boldsymbol{\epsilon}_y$ the Gaussian noise of $\mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma}_y)$, defined by arbitrary $\boldsymbol{\Sigma}_y$. Here, a single trajectory sample $\boldsymbol{y}_t$ is the aforementioned sum of the weighted Gaussian basis functions with added noise, which translates to the joint Gaussian distribution conditioned on the trajectory weights $\boldsymbol{w}$.

Learning a ProMP from demonstration of motion is achieved by fitting the distribution of $\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{w}}, \boldsymbol{\Sigma}_{\boldsymbol{w}})$ via a maximum-likelihood estimation (MLE) for stroke-based motions. This, given a set of N demonstrated trajectories, yields

$$\boldsymbol{\mu}_{\boldsymbol{w}} = \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{w}_i \tag{3}$$

$$\boldsymbol{\Sigma_w} = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{w}_i - \boldsymbol{\mu_w})(\boldsymbol{w}_i - \boldsymbol{\mu_w})^T \tag{4}$$

For periodic motions, ProMP fitting is performed through expectation maximization (EM); however, this will not be discussed as it is beyond the scope of this study.

ProMP being a probability distribution of trajectories allows for the easy implementation of Gaussian processes computation, as the posterior probability distribution of trajectory given environment configuration can directly be computed [13].

2.2. PRO-GP

The learning algorithm proposed in [13] (dubbed PRO-GP hereafter) is primarily a semi-supervised imitation learning framework. It consists of a GP regression model that infers trajectory distributions encoded as ProMPs given a feature vector that encodes environment configuration, and an optimization algorithm dubbed as PRO (Pearson-correlation-based relevance weighted policy optimization) which optimizes trajectories sampled from the trajectory distribution inferred by the GP regression model.

Demonstration trajectories are provided for a number of different environment configurations, and the sets of trajectories that correspond to the same environment configurations are fitted and parameterized as ProMP as means $(\mu_{w_{1,m}}, \cdots, \mu_{w_{n,m}})$ and variances $(\sigma^2_{w_{1,m}}, \cdots, \sigma^2_{w_{n,m}})$ where n is the maximum index of ProMP weights (or Gaussian basis functions) and m denotes the index of the environment configuration. GPs (prior to the distribution of the model) are defined as one each weight, where the mean functions are defined by the average of the means and variances of ProMPs over all environment configurations for each weight; μ_{w_n} and $\sigma^2_{w_n}$. The kernel matrix $\boldsymbol{K}$ is a covariance function that describes the relationship between different environment configurations that the model has "seen," where each element in row i, column j, is an exponential kernel $k(\boldsymbol{e}_i, \boldsymbol{e}_j)$;

$$\mu_{w_n} = \frac{1}{M}\sum_{m=1}^{M}\mu_{w_{n,m}}, \qquad \sigma^2_{w_n} = \frac{1}{M}\sum_{m=1}^{M}\sigma^2_{w_{n,m}} \tag{5}$$

$$k(\boldsymbol{e}_i, \boldsymbol{e}_j) = \exp\Big(-\alpha(\boldsymbol{e}_i - \boldsymbol{e}_j)^T(\boldsymbol{e}_i - \boldsymbol{e}_j)\Big), \qquad \alpha > 0 \tag{6}$$

where M is the maximum index of demonstration environment configurations. Here, the kernel parameter α dictates how important the correlation between the learned feature and the unseen feature is. As the value of α becomes larger, learned features that are far away from (i.e., dissimilar to) the queried unseen feature are regarded with less weight in inferring the output.

Hereon, the algorithm repeats an unsupervised learning loop for a set number of iterations. First, a new, unseen environment feature $\mathbf{e}_x$ for which to learn a new trajectory distribution is sampled. GP regression is performed to obtain posterior distribution $p(w_{n,x}|w_{x,1:M}) = \mathcal{N}(\mu_{w_{n,x}}, \sigma^2_{w_{n,x}})$, where

$$\mu_{w_{n,x}} = \mu_{w_n} + \boldsymbol{K}_{x,1:M}(\boldsymbol{K}_{1:M,1:M} + \boldsymbol{\Sigma}_{w_n})^{-1}(\boldsymbol{\mu}_{w_{n,1:M}} - \boldsymbol{\mu}_{w_n}) \tag{7}$$

$$\sigma^2_{w_{n,x}} = \boldsymbol{K}_{x,x} + \sigma^2_{w_n} - \boldsymbol{K}_{x,1:M}(\boldsymbol{K}_{1:M,1:M} + \boldsymbol{\Sigma_{w_n}})^{-1}\boldsymbol{K}_{1:M,x} \tag{8}$$

$$\boldsymbol{K}_x = \begin{pmatrix} \boldsymbol{K}_{x,x} & \boldsymbol{K}_{x,1:M} \\ \boldsymbol{K}_{1:M,x} & \boldsymbol{K}_{1:M,1:M} \end{pmatrix} \tag{9}$$

where $\boldsymbol{\Sigma}_{w_n} = diag(\sigma^2_{w_{n,1}}, \cdots, \sigma^2_{w_{n,M}}, \boldsymbol{\mu}_{w_{n,1:M}} = (\mu_{w_{n,1}}, \cdots, \mu_{w_{n,M}})^T, \boldsymbol{\mu}_{w_n} = (\mu_{w_n}, \cdots, \mu_{w_n})^T$ is a vector with μ_{w_n} repeated M times, and $\boldsymbol{K}_x$ is the newly inferred kernel matrix of the GPs.

The obtained trajectory distribution is then optimized with PRO so as to ensure that the model learns optimal policies. PRO uses a Pearson correlation coefficient to learn relevance functions, which represent how each weights are linearly correlated to a given

objective function. Relevance functions are then used to construct a Gaussian distribution from which individual trajectories (ProMP weights) are sampled. The sampled trajectories are optimized using reward-weighted regression (RWR) [22] until the reward function (defined based on objective functions) converges.

3. Methods

The experiment, illustrated in Figure 1, follows the experiment detailed in Section 5.2 of Ref. [13] (the codes can be accessed at M. Ewerton's Github repository. URL: https://github.com/marcoewerton/avoid_moving_walls (accessed on 19 February 2022)). As shown in Figure 1, it involves a 2 DOF point particle agent navigating through a 2D task space, going past a via-point (x_c, y_c) at a window through a wall obstacle and terminating at the given goal position (x_g, y_g). The starting position is fixed at $(0, 5)$, and the 4D environment vector $\boldsymbol{e} = (x_c, y_c, x_g, y_g)^T$ is defined in the range of $2 \leq x_c \leq 8$, $1 \leq y_c \leq 9$, $x_c + 1.5 \leq x_g \leq 10$, $0 \leq y_g \leq 10$.

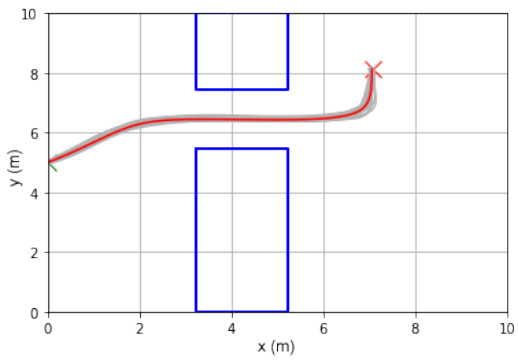

Figure 1. Experiment visualization. Point particle agent begins at a fixed initial position (green cross), navigates through a window through a moving wall obstacle to terminate at a moving goal position (red cross).

Training is run for 945 iterations of an unsupervised learning loop, starting with 5 manually-generated demonstration trajectories for each of the 30 initial environment configurations. The kernel parameter as shown in Equation (6) was defined to be $\alpha = 10^{-3}$ as chosen by authors of [13].

PRO uses 5 objective functions. Three of these are via-point objectives: initial position error to start configuration, minimum signed Euclidean distance to obstacle, and terminal position error to goal configuration. Two trajectory smoothing objectives are added in this experiment to be able to compare the optimality of the inferred policies. These are the average absolute value of jerk as the jerk objective function and the signed difference between the inferred trajectory and corresponding reference trajectory as the trajectory length objective function. As shown in Figure 2, the reference trajectory (green) is automatically synthesized as three line segments between the initial position, the midpoints of the beginning and ending of the window in the wall, and the goal position.

Here, the reward function for PRO is formulated as $R_o = \exp\big(-\beta(o + \beta_l l + \beta_j j)\big)$, where $o, \beta_l, l, \beta_j, j$ denote the via-point objective, trajectory length objective weight, trajectory length objective function, jerk objective weight, and jerk objective function, respectively. The objective weights are determined to be $\beta_l = 2, \beta_j = 1000$ by empirically testing over values of [0.1, 0.5, 1, 2, 5, 10] for β_l and $[1 \times 10^3, 5 \times 10^3, 1 \times 10^4, 2 \times 10^4, 5 \times 10^4]$ for β_j, except for $\beta = 20$, which is the value as set in the authors' codes.

Additionally, the convergence and trajectory optimality criteria were relaxed in order to mitigate the additional objectives making successful convergence more difficult. A fault in the algorithm for optimization loop termination was also identified and fixed, where originally PRO would continue the optimization loop if the optimality criteria were not met even after convergence was achieved. This, with some optimization in the codes and removing redundancies, reduced the training time from 24 h or more to 6–12 h.

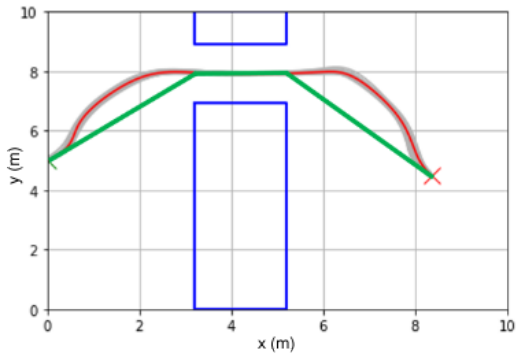

Figure 2. Inferred mean trajectory (red), inferred trajectory variance (gray) and automatically synthesized reference trajectory (green) for trajectory length objective computation in PRO.

3.1. Baseline

The baseline approach as implemented in [13] uses the random uniform sampling (RUS) of unseen environment features (training data).

Initial environment features for demonstration are also sampled via random uniform sampling, for which there are 5 demonstrations given for each of the 30 environment configurations. In total, 945 iterations of the self-improvement loop are set to run after which training terminates.

3.2. Voronoi Tessellation Sampling

The PRO-GP training scheme assumes that PRO can adequately optimize trajectories inferred by the GP regression model, but this is not necessarily always true. If the GP regression model cannot infer trajectories that are already adequate to a certain degree, PRO fails to converge to a learnable outcome, and nothing is learned in the iteration. This is especially so when te query feature is sampled such that it is far removed from the distribution of learned data and the model subsequently infers a highly suboptimal trajectory, which is likely during early iterations of training.

The baseline method of random uniform sampling depends on pure luck to find an environment feature to learn, which does not guarantee that the sampled data will be informative for the model. The GP regression model also requires learning an even distribution of data over the feature space to be able to yield adequate generalization performance, and random uniform sampling lacks in this regard as well.

In other words, the baseline method can often fail to find data that are similar enough to the learned data for the model to be able to learn, are dissimilar enough for them to be informative for the model, and ensure that the distribution of the learned data will converge towards an even conditioning over feature space. This poor conditioning of training data is likely to result in requiring a large amount of data before the model is able to reliably generalize over all feature space.

As such, an alternative data sampling strategy is warranted such that the exploration–exploitation trade–off is implemented to ensure that the model learns an evenly conditioned distribution of data, spanning the entire feature space, which in turn will potentially result in better inferred policies with fewer learned data. It can thus be reasoned that applying such a method will allow for the training of fewer data to yield better results.

GP regression in essence can be interpreted as a statistical interpolation method between learned features and output variables. Especially in the case of this experiment, there exists an intuitively observable spatial correlation between the input features (environment configurations) and inferred outputs (trajectory distributions) as input features dictate where in the configuration space the trajectory needs to navigate. Hence, without trying to decide the extent to which exploration should be allowed starting from a cluster of initial data, it would be possible to efficiently learn to cover the feature space by interpolating between the learned data to obtain training data features if the initial demonstration data

are conditioned to be even along the feature space and cover edge cases of the feature space, where Voronoi vertices can be a good representation of such interpolation, as shown in Figure 3.

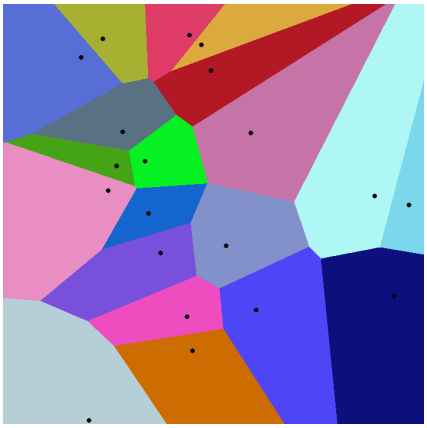

Figure 3. Voronoi tessellation forms vertices that can be viewed as interpolations between the seed points. Image courtesy of Wikimedia Commons.

Hence, in this experiment, a new set of demonstrations is created in 30 environment configurations sampled with Latin hypercube sampling (LHS) which enables the set of features to better represent uniform distribution than individually, independently sampled features. Using these "learned" environment features as seed points, Voronoi tessellation is performed where the obtained Voronoi vertices are the new set of environment features to be trained on. To identify the effect of initial data sampling on the end result, 5 different sets of initial demonstration data are created via random uniform sampling as per baseline, with which models are trained using Voronoi tessellation sampling (VTS) and their performances are compared to that of the model trained with initial data conditioned with LHS. The generalized algorithmic process of VTS is briefed in Algorithm 1.

Algorithm 1 Voronoi tessellation sampling

Require: set of learned feature vectors D
Ensure: set of feature vectors to be learned V
  Generate grid of 945 points over feature space G
  for each point g in G **do**
    Evaluate density of learned data distribution $d_{k=5}$ around g
  end for
  Add 20 grid points with lowest $d_{k=5}$ to D to obtain seed set S
  Perform Voronoi tessellation with S as seeds to obtain set of Voronoi vertices V
  for all vertices v in V **do**
    Evaluate density of learned data $d_{k=5}$ around v
    Evaluate closest distance to learned data d_{min} around v
    if $d_{min} < 1$ and $1.5 < d_{k=5} < 2.5$ **then**
      Delete v from V
    end if
  end for
  return V

Here, two additional measures are placed to ensure effective learning—data pruning and bias injection. Sampling by interpolating alone does not ensure that the sampled features will yield inferred policies that are optimal enough for PRO to successfully converge and yet be sufficiently different from learned features that it is informative to the model. To do so, vertices (sampled features) that already have enough learned data in their vicinities are pruned before training. Here, an average Euclidean distance of 5 nearest

neighboring learned data is used to measure the density of learned data around a sampled feature. Sampled features that are not within the defined feature space must also be discarded.

Furthermore, it must be noted that the interpolation-based sampling method tends to predominantly sample from within the cluster of learned data, and exploration towards the outside of the cluster is minimal if not unseen. Bias injection is thus performed when sampling, simply by adding points at relatively underexplored regions of feature space into the set of seed points.

This process is demonstrated in Figure 4, where the data pruning and bias injection encourage gradual exploration towards the outside of the cluster of learned data and contribute to efficiently learning larger areas of the feature space. To identify the effect of data pruning and bias injection measures, a model is trained without the said procedures to be compared with control.

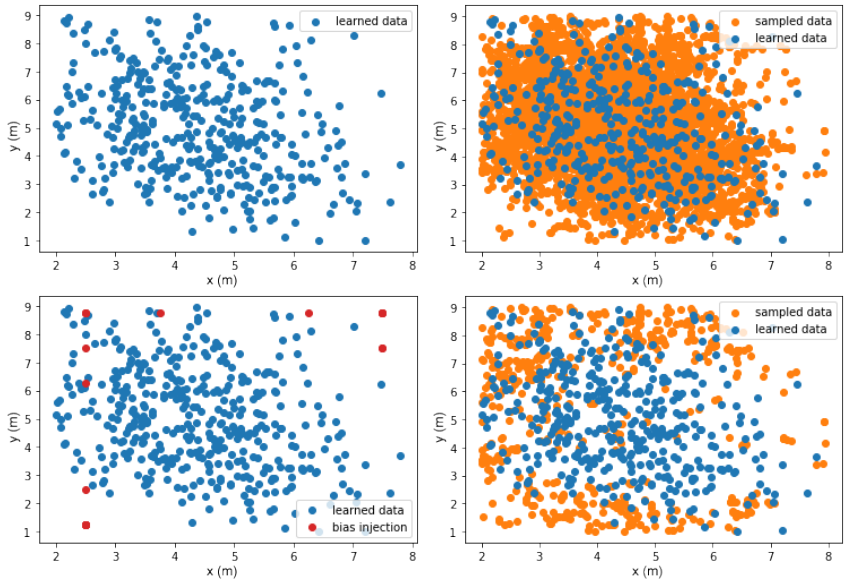

Figure 4. Demonstration of data pruning and bias injection in effect. Given a set of learned data, likely to be in a cluster (**top left**), simple Voronoi tessellation (**top right**) will likely result in sampling vertices predominantly within the cluster and often too many. By injecting grid points with low data density into the set of seeds (**bottom left**) and pruning sampled data according to data density, the resulting samples (**bottom right**) are "pushed" outwards of the cluster of learned data.

For this experiment, 20 grid points with lowest data density are used as injected bias seeds, and sampled features are kept to be trained on only if $d_{min} \leq 1$ and $1.5 \leq d_{K=5} < 2.5$, where d_{min} is Euclidean distance to the closest neighboring learned data, and $d_{K=5}$ is the 5-NN average Euclidean distance from learned data. The number of grid points to add to the seed set and the data pruning threshold were determined empirically by inspecting 6 rounds of Voronoi tessellation sampling on the initial data without training the GP model, over the range of [5, 10, 20, 40] for the number of grid points added, [0.5, 1, 1.2, 1.5] for maximum d_{min} threshold, [1, 1.2, 1.5] for minimum $d_{K=5}$ threshold, and [2, 2.2, 2.5, 3] for $d_{K=5}$ maximum threshold.

3.3. Evaluation Metrics

The outcome of each training strategy is evaluated through the optimality of the inferred policy. To measure how well the model's trajectory inference performs, rewards (in the form of the five aforementioned objective functions, i.e., initial position error to start configuration, minimum signed Euclidean distance to obstacle, terminal position error to goal configuration, average absolute jerk, and signed trajectory length difference with synthesized reference) are evaluated for trajectories inferred in 945 environments

generated from a grid of the feature space. The mean, standard deviation, minimum, and maximum of this metric will provide insight into the model's overall performance in any given environment configuration.

4. Results

As discussed in the above section, all trained models' trajectory inferences were evaluated for five performance metrics with respect to a grid of 945 features generated across the feature space. The metrics are initial position error to start configuration, minimum signed Euclidean distance to obstacle, terminal position error to goal configuration, average absolute jerk, and signed trajectory length difference with synthesized reference. Only the mean trajectories of each inferred trajectory distribution are considered for this evaluation.

Results are shown in Table 1. Each approach is denoted by the training data sampling method used and any changes to the training setup described in Section 3. RUS and VTS denote random uniform sampling and Voronoi tessellation sampling, respectively. Models with suffix "-full" are trained with no changes made to the described methods. VTS-NBI was trained without bias injection and data pruning measures to identify the effectiveness of these measures in the design of the VTS algorithm. Early-training cut-off was performed on "-full" models at 300th the data point to obtain RUS-300 and VTS-300 so as to compare performances with limited amounts of data. Finally, to verify the importance of initial data conditioning, VTS-RD shows aggregate statistics for five VTS-300 models trained with five sets of initial data, where the environment configurations are random uniform sampled. Here, the baseline RUS-full learned 909 data out of 945 iterations, VTS-full learned a mere 367, and VTS-NBI 513.

Table 1. Trained models' trajectory inference performance statistics, where RUS and VTS denote the training data sampling method used, the suffix "-full" denotes no change to the training procedure as detailed in Section 3, "-NBI" training without bias injection and data pruning, "-300" training cut-off at the 300th data point learned, and "-RD" the use of random-uniform-sampled initial demonstration data.

Metric	Method	Mean	STD	Max	Min
Start point error	RUS-full	1.964×10^{-2}	1.359×10^{-2}	1.290×10^{-1}	8.852×10^{-4}
	VTS-full	3.556×10^{-2}	3.884×10^{-2}	3.090×10^{-1}	2.385×10^{-4}
	VTS-NBI	7.334×10^{-2}	9.271×10^{-2}	6.994×10^{-1}	5.778×10^{-4}
	RUS-300	2.625×10^{-2}	1.892×10^{-2}	1.349×10^{-1}	2.628×10^{-4}
	VTS-300	4.362×10^{-2}	4.599×10^{-2}	3.479×10^{-1}	6.045×10^{-4}
	VTS-RD	4.034×10^{-2}	1.113×10^{-1}	8.120×10^{-1}	1.213×10^{-4}
End point error	RUS-full	2.939×10^{-2}	2.472×10^{-2}	2.100×10^{-1}	3.301×10^{-4}
	VTS-full	3.760×10^{-2}	3.939×10^{-2}	2.850×10^{-1}	5.287×10^{-4}
	VTS-NBI	6.709×10^{-2}	6.941×10^{-2}	4.995×10^{-1}	1.092×10^{-3}
	RUS-300	3.753×10^{-2}	3.469×10^{-2}	2.882×10^{-1}	1.804×10^{-3}
	VTS-300	3.077×10^{-2}	2.655×10^{-2}	2.372×10^{-1}	7.816×10^{-4}
	VTS-RD	3.231×10^{-2}	6.521×10^{-2}	3.099×10^{-1}	5.655×10^{-4}
Obstacle distance	RUS-full	8.380×10^{-1}	1.150×10^{-1}	9.934×10^{-1}	9.369×10^{-3}
	VTS-full	8.310×10^{-1}	1.130×10^{-1}	9.950×10^{-1}	2.800×10^{-1}
	VTS-NBI	8.080×10^{-1}	1.407×10^{-1}	9.881×10^{-1}	2.503×10^{-2}
	RUS-300	8.178×10^{-1}	1.453×10^{-1}	9.944×10^{-1}	-8.114×10^{-1}
	VTS-300	8.337×10^{-1}	9.992×10^{-2}	9.953×10^{-1}	2.485×10^{-1}
	VTS-RD	8.193×10^{-1}	3.255×10^{-1}	9.947×10^{-1}	-8.109×10^{-1}
Trajectory length	RUS-full	1.637×10^{-2}	7.853×10^{-2}	6.790×10^{-1}	-2.880×10^{-1}
	VTS-full	2.900×10^{-2}	9.305×10^{-2}	9.930×10^{-1}	-2.866×10^{-1}
	VTS-NBI	1.458×10^{-1}	3.606×10^{-1}	3.765	-4.026×10^{-1}
	RUS-300	2.625×10^{-2}	1.136×10^{-1}	1.072	-3.684×10^{-1}
	VTS-300	7.282×10^{-2}	1.994×10^{-1}	2.326	-1.790×10^{-1}
	VTS-RD	2.666×10^{-2}	2.585×10^{-1}	1.594	-4.969×10^{-1}

Table 1. *Cont.*

Metric	Method	Mean	STD	Max	Min
Average jerk	RUS-full	5.192×10^{-5}	1.672×10^{-5}	1.378×10^{-4}	1.970×10^{-5}
	VTS-full	5.534×10^{-5}	2.155×10^{-5}	2.176×10^{-4}	2.129×10^{-5}
	VTS-NBI	6.709×10^{-5}	3.099×10^{-5}	2.630×10^{-4}	2.103×10^{-5}
	RUS-300	5.556×10^{-5}	1.900×10^{-5}	1.544×10^{-4}	2.309×10^{-5}
	VTS-300	5.721×10^{-5}	2.451×10^{-5}	2.353×10^{-4}	2.114×10^{-5}
	VTS-RD	5.576×10^{-5}	4.879×10^{-5}	2.313×10^{-4}	2.002×10^{-5}

Demonstration for Real-World Application

The problem used for this experiment is, while simple, easily expandable to applications in a real-world domain. For instance, a scenario may be assumed in which a mobile robot needs to navigate through multiple dynamic obstacles (e.g., people) where an opening through the obstacles is defined as the window in the wall obstacle as shown in Figure 1. Scenarios involving collaborative robotics are also possible, in which a robot is required to track the user's hand, come into contact to pick up an object, and continue to the goal position. The problem setting can also easily be altered to a dynamic obstacle avoidance problem where a collaborative robot needs to keep away from the reach of the user's hand in a shared workspace.

To demonstrate, the learned trajectory inference model is directly mapped to a UR-10 6 DOF collaborative robot, for which a 2D workspace is defined as the black tabletop in Figure 5. With the initial and goal configurations fixed on the either side of the workspace and 2 ArUco AR markers tracking the 3D coordinate of the tabletop (for reference) and the white marker, the robot is to navigate its manipulator across the workspace, passing through the white marker that is moved around by a human user. Figure 5 shows the robot changing its reference trajectory as the via-point marker is dynamically moved around in the workspace. Figure 6 suggests a simple way in which this demonstration can be implemented as a motion planning solution for shared autonomy, where obstacle (e.g., hand of a human co-operator) positions are detected via computer vision, around which a collision boundary with a set radius is generated. The robotic agent only needs to reposition the via-point (window through the wall in the original problem) outside the collision boundary to avoid a collision with the dynamic obstacle.

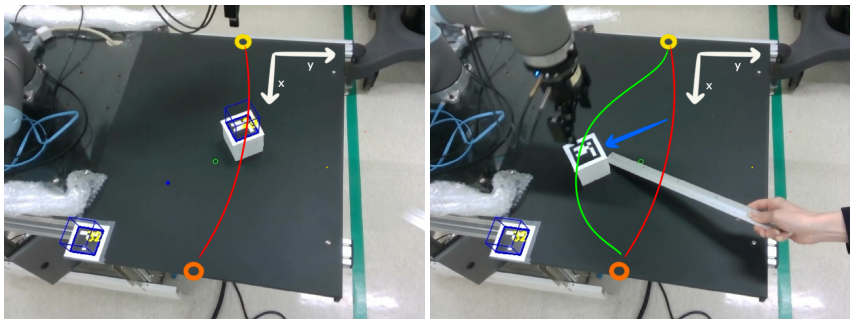

Figure 5. UR-10 robot manipulator follows initial trajectory (red) with fixed initial (yellow) and terminal (orange) configurations, until the via-point marker is moved and it follows a new trajectory (green) inferred in real time that corresponds to the changed environment.

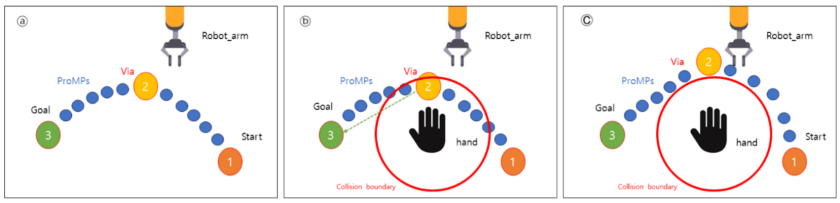

Figure 6. A suggestion for real-world application for the shown demonstration. (**a**) Initial motion planning (blue) of the robot arm manipulator navigating from the initial configuration (orange) through a via point (yellow) to a goal configuration (green); (**b**) collision boundary generated around detected obstacle (hand); (**c**) re-planning with re-positioned via-point outside of collision boundary.

5. Discussion

Results in Table 1 are displayed in Figures 7–9 for ease of visual analyses. As can be seen in Figure 7, the proposed method (VTS-full) only manages to learn approximately 40% of the data learned by the baseline, but it achieves comparable performance. It can also be seen how the VTS model performs far worse without bias injection and data pruning measures. The reason can be identified in Figure 10, which shows on the task space which environmental configuration features occurred most frequently as training data. Here, the learned data of VTS-NBI have clustered towards the center while VTS-full has learned to cover the feature space evenly. Where the bias injection and data pruning measures encourage exploration, the lack thereof resulted in the VTS-NBI model not being able to learn in certain regions of the feature space, thus leading to suboptimal policies.

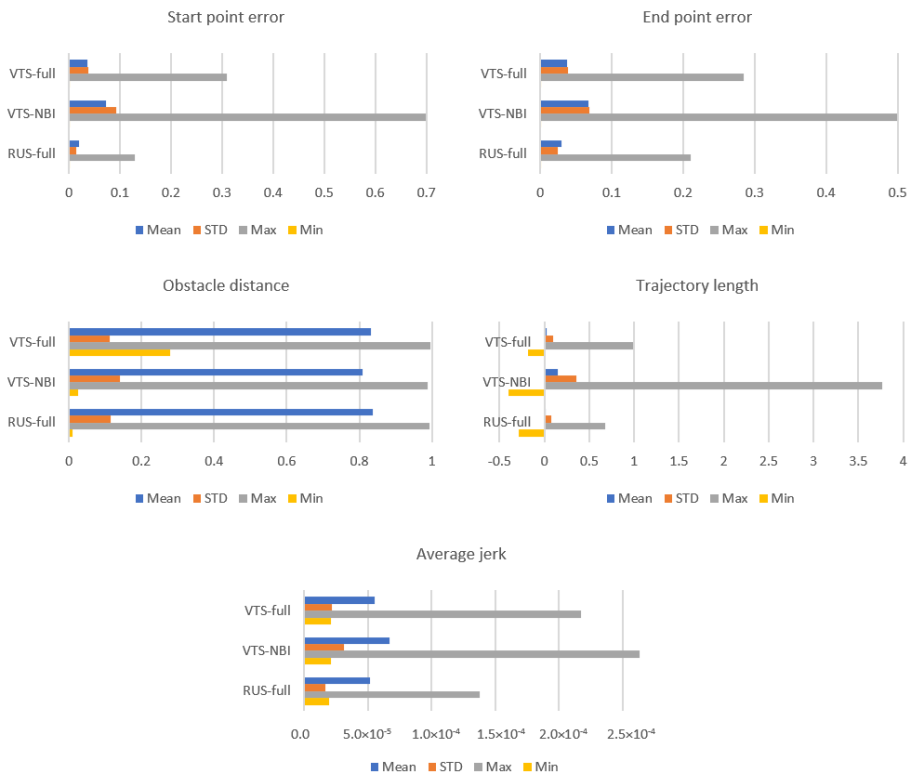

Figure 7. Trained models' trajectory inference statistics for fully trained models and VTS without bias injection and data pruning (VTS-NBI).

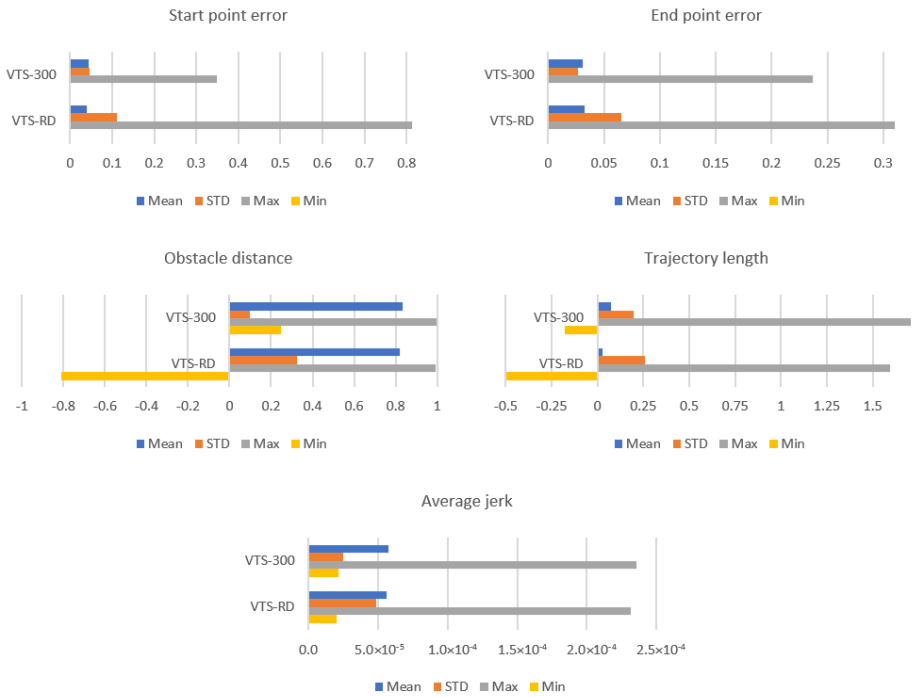

Figure 8. Trained models' trajectory inference statistics for VTS models with initial data conditioned with LHS (VTS-300) and RUS (VTS-RD). Training cut-off at 300th learned data point for both.

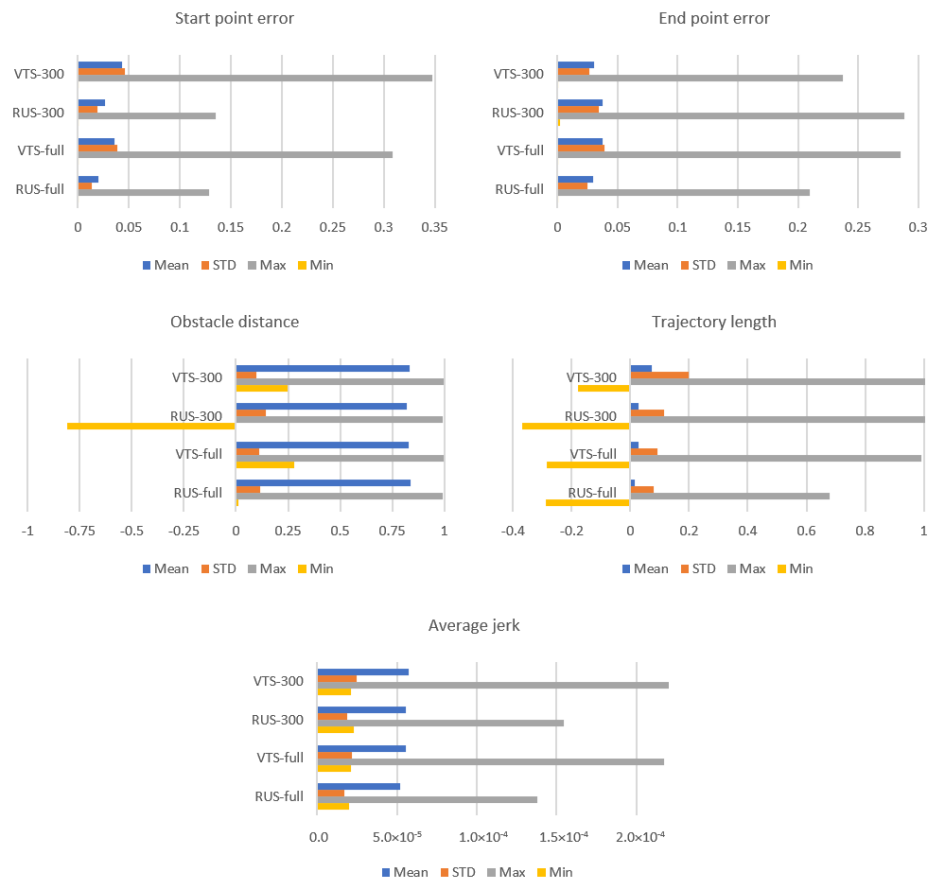

Figure 9. Trained models' trajectory inference statistics for models with training cut-off at 300th learned data point.

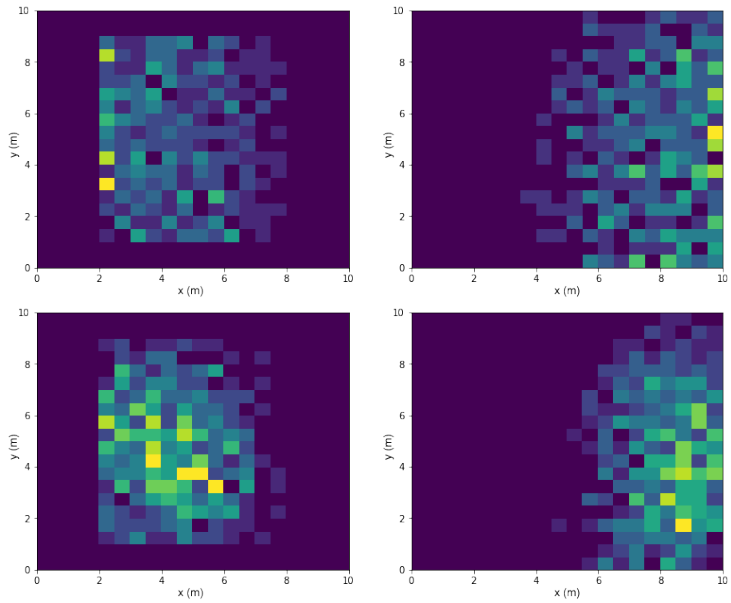

Figure 10. Heatmaps of learned data distribution for VTS-full (**above**) and VTS without bias injection and data pruning (VTS-NBI, **below**) for via-point configurations (**left**) and goal configurations (**right**). Purple denotes values close to 0 and yellow the maximum.

Conditioning the spread of the initial data features also proved to be important. Latin hypercube sampling was chosen to represent an "even" spread of initial data features in the feature space in the proposed method, and Figure 8 shows the standard proposed model (VTS-300) outperforming its counterpart (VTS-RD) with randomly selected initial data at nearly every performance metric. In particular, note the minimum obstacle distance metric, where a negative value indicates collision, rendering VTS-RD unusable in certain regions of the feature space. Note, on the other hand, in Figures 7–9, negative minimum values for the trajectory length objective function indicate no notable significance as outliers or undesirable behaviors, as the negative values only signify that the inferred mean trajectories are shorter than the corresponding synthesized three-segment reference trajectories shown in Figure 2.

For both baseline and VTS models, early training cut-off was performed at the 300th data point learned to verify if VTS achieves a good exploration–exploitation trade-off with a small amount of data by comparing the performance of the baseline model trained with only 300 data points. For most metrics, VTS-300 achieves similar performances to VTS-full, which is worse than that of the baseline, but within a tolerable margin. Note, however, the negative minimum obstacle distance for RUS-300 in Figure 9. This being the signed minimum distance to the obstacle, negative values mean that some inferred trajectories have collided into the obstacle, i.e., the model has failed to learn enough of the feature space to be able to reliably output an adequate, if not optimal, policy throughout all ranges of environment configurations possible.

This, when compared to the performance of VTS-300, highlights the ability of the VTS method to evenly sample across the data space while ensuring that PRO is able to optimize learnable trajectories, and it can be reasoned that by employing VTS, the GP regression model learns to reliably generalize over all feature space using fewer learned data, allowing for less computation in policy inference. In real applications, a training cut-off could be implemented, conditioned on the learning rate (amount of data learned per iteration) and "evenness" of data distribution to automatically harness this efficiency and allow for training with fewer iterations.

Furthermore, as RUS relies on luck and a large number of iterations to reach optimal learning, it is unlikely that the optimality it learned shown in the results will extend to problems of higher DOFs as it becomes exponentially more difficult to randomly find a

learnable and informative feature point. On the other hand, as VTS leverages geometric interpolation to find training data, it will likely retain the ability to reach even distribution of data along feature space.

6. Conclusions

In this study, a novel data sampling strategy for learning GP regression-based dynamic robot motion planning was proposed. This method exploits the nature of GP regression's generalization as an interpolation technique. It allows for simple and data-efficient implementation of continuous-space data sampling for GP regression models, addressing the data scalability issue of GP regression approaches by choosing the right data to learn, and in theory is scalable to problems with higher dimensionality. Experiments show that this method can achieve similar or better policies over all feature space with fewer data compared to baseline.

While the proposed method is conceptually simple, the learning hyperparameters can be somewhat difficult to tune. Especially in the data pruning stage of VTS, there is no methodical way to determine optimal thresholds for data density measures. It is also difficult to conduct preliminary experiments with less iterations to gain insight, as the data distribution will wildly vary depending on the training outcome. These hyperparameters are, however, vital to optimal and efficient training as the restriction in these thresholds result in the algorithm prioritizing the learning of data that are more different than the learned data, thus acting as exploration–exploitation trade-off parameters. This added complexity can prove to be problematic in application to more complex scenarios.

In the future, the scalability of VTS is to be scrutinized further by exploring problem settings with higher DOFs. It would also be beneficial to explore other exploration strategies used in RL, as well as using data density metrics other than K-NN average Euclidean distance (e.g., kernel density functions).

Author Contributions: Conceptualization, J.-Y.P. and J.R.; methodology, J.-Y.P., H.L. and C.K.; software, J.-Y.P., H.L. and C.K.; validation, J.-Y.P.; formal analysis, J.-Y.P.; investigation, J.-Y.P., H.L. and C.K.; resources, J.R.; data curation, J.-Y.P.; writing—original draft preparation, J.-Y.P.; writing—review and editing, J.-Y.P., H.L., C.K. and J.R.; visualization, J.-Y.P. and C.K.; supervision, J.R.; project administration, J.R.; funding acquisition, J.R. All authors have read and agreed to the published version of the manuscript.

Funding: This work was partly supported by 'Development of a remote dismantling training system with force-torque responding virtual nuclear power plant' (Project Number: 20201510300280) funded by Korea Institute of Energy Technology Evaluation and Planning, and by Institute of Information & Communications Technology Planning and Evaluation (IITP) grant funded by Korea Ministry of Science and ICT (MSIT) (No. 2019-0-01842, Artificial Intelligence Graduate School Program (GIST)).

Institutional Review Board Statement: Not applicable.

Data Availability Statement: The data presented in this study are available on request from the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript:

DOF	Degrees of freedom
GP	Gaussian process
LHS	Latin hypercube sampling
PRO-GP	Pearson-correlation-based relevance weighted policy optimization
ProMP	Probabilistic motion primitives

RL	Reinforcement learning
RUS	Random uniform sampling
VTS	Voronoi tessellation sampling

References

1. Heimann, O.; Guhl, J. Industrial Robot Programming Methods: A Scoping Review. In Proceedings of the 2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Vienna, Austria, 8–11 September 2020. [CrossRef]
2. Sozzi, A.; Bonfè, M.; Farsoni, S.; Rossi, G.D.; Muradore, R. Dynamic Motion Planning for Autonomous Assistive Surgical Robots. *Electronics* **2019**, *8*, 957. [CrossRef]
3. Mir, I.; Gul, F.; Mir, S.; Khan, M.A.; Saeed, N.; Abualigah, L.; Abuhaija, B.; Gandomi, A.H. A Survey of Trajectory Planning Techniques for Autonomous Systems. *Electronics* **2022**, *11*, 2801. [CrossRef]
4. Banerjee, A.; Dunson, D.B.; Tokdar, S.T. Efficient Gaussian process regression for large datasets. *Biometrika* **2012**, *100*, 75–89. [CrossRef] [PubMed]
5. Sánchez-Ibáñez, J.R.; del Pulgar, C.J.P.; García-Cerezo, A. Path Planning for Autonomous Mobile Robots: A Review. *Sensors* **2021**, *21*, 7898. [CrossRef] [PubMed]
6. Dijkstra, E.W. A note on two problems in connexion with graphs. *Numer. Math.* **1959**, *1*, 269–271. [CrossRef]
7. Hart, P.; Nilsson, N.; Raphael, B. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Trans. Syst. Sci. Cybern.* **1968**, *4*, 100–107. [CrossRef]
8. LaValle, S.M.; Kuffner, J.J. Randomized Kinodynamic Planning. *Int. J. Robot. Res.* **2001**, *20*, 378–400. [CrossRef]
9. Kavraki, L.; Svestka, P.; Latombe, J.C.; Overmars, M. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. Autom.* **1996**, *12*, 566–580. [CrossRef]
10. Rösmann, C.; Feiten, W.; Wösch, T.; Hoffmann, F.; Bertram, T. Trajectory modification considering dynamic constraints of autonomous robots. In Proceedings of the 7th German Conference on Robotics (ROBOTIK 2012), Munich, Germany, 21–22 May 2012; pp. 1–6.
11. Mukadam, M.; Dong, J.; Yan, X.; Dellaert, F.; Boots, B. Continuous-time Gaussian process motion planning via probabilistic inference. *Int. J. Robot. Res.* **2018**, *37*, 1319–1340. [CrossRef]
12. Frank, F.; Paraschos, A.; van der Smagt, P.; Cseke, B. Constrained Probabilistic Movement Primitives for Robot Trajectory Adaptation. *IEEE Trans. Robot.* **2022**, *38*, 2276–2294. [CrossRef]
13. Ewerton, M.; Arenz, O.; Maeda, G.; Koert, D.; Kolev, Z.; Takahashi, M.; Peters, J. Learning Trajectory Distributions for Assisted Teleoperation and Path Planning. *Front. Robot. AI* **2019**, *6*, 89. [CrossRef]
14. Schneider, M.; Ertel, W. Robot Learning by Demonstration with local Gaussian process regression. In Proceedings of the 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, Taipei, Taiwan, 18–22 October 2010. [CrossRef]
15. Wang, K.; Pleiss, G.; Gardner, J.; Tyree, S.; Weinberger, K.Q.; Wilson, A.G. Exact Gaussian Processes on a Million Data Points. In Proceedings of the Advances in Neural Information Processing Systems, Vancouver, BC, Canada, 8–14 December 2019; Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2019; Volume 32.
16. Hensman, J.; Fusi, N.; Lawrence, N.D. Gaussian Processes for Big Data. In Proceedings of the 29th Conference on Uncertainty in Artificial Intelligence (UAI'13), Arlington, VA, USA, 11–15 July 2013; pp. 282–290.
17. Daniel, C.; Kroemer, O.; Viering, M.; Metz, J.; Peters, J. Active reward learning with a novel acquisition function. *Auton. Robot.* **2015**, *39*, 389–405. [CrossRef]
18. Bıyık, E.; Huynh, N.; Kochenderfer, M.J.; Sadigh, D. Active Preference-Based Gaussian Process Regression for Reward Learning. *arXiv* **2020**, arXiv:2005.02575. https://doi.org/10.48550/ARXIV.2005.02575.
19. Huang, S.K.; Wang, W.J.; Sun, C.H. A Path Planning Strategy for Multi-Robot Moving with Path-Priority Order Based on a Generalized Voronoi Diagram. *Appl. Sci.* **2021**, *11*, 9650. [CrossRef]
20. Bhattacharya, P.; Gavrilova, M. Roadmap-Based Path Planning - Using the Voronoi Diagram for a Clearance-Based Shortest Path. *IEEE Robot. Autom. Mag.* **2008**, *15*, 58–66. [CrossRef]
21. Paraschos, A.; Daniel, C.; Peters, J.; Neumann, G. Using probabilistic movement primitives in robotics. *Auton. Robot.* **2017**, *42*, 529–551. [CrossRef]
22. Peters, J.; Schaal, S. Reinforcement learning by reward-weighted regression for operational space control. In Proceedings of the 24th International Conference on Machine Learning, Corvallis, OR, USA, 20–24 June 2007. [CrossRef]