Gaussian Random Paths

Sungjoon Choi, Kyungjae Lee, and Songhwai Oh

Abstract— **In this paper, we propose Gaussian random paths by defining a probability distribution over continuous paths interpolating a finite set of anchoring points using Gaussian process regression. The mean path of Gaussian random paths can effectually replace a parametric curve model, e.g., a spline model, and is notably effective when the data points are scarce. Furthermore, a Gaussian random path planner is proposed by fully utilizing the generative property of Gaussian random paths, from which a diverse set of paths can be sampled to steer a robot to a goal position. The Gaussian random path planner can be used in a number of applications, including local path planning and target tracking. We have conducted an extensive set of simulations and experiments, showing that the proposed planner outperforms look-ahead planners which use a pre-defined subset of egocentric trajectories in terms of collision rates and trajectory lengths.**

I. Introduction

With recent advances in computing, hardware, and sensing technologies, robot applications have spread out from structured laboratories to complex and cluttered real-world environments. In particular, one representative example is the Google driverless car. In order for a robot or an autonomous vehicle to navigate safely in a real-world environment, real-time path planning based on sensed data is an essential requirement. In the past, path planning is applied to a static environment where a complete configuration map is given *a priori* and considerable accomplishments have been made in this regard. However, the necessity of robotic applications to handle scenarios, in which a full prior map is inaccessible, e.g., disaster relief or military operations in cluttered environments, leads to an emphasis on making a decision based on a current sensory input in real-time, which we refer to as a local planning.

While designing a local planner has been hampered by real-time and kinodynamic constraints, a path set method has effectively been deployed in a number of applications, including unmanned ground and aerial vehicles. In fact, a majority of the finalists in the 2007 DARPA Urban Challenge (DUC) used the path set method or its variants [1]–[6]. A path set consists of a fixed set of control sequences where the resulting responses (trajectories) are pre-computed using a high-fidelity vehicle model [7]. In this context, Erickson et. al. [8] focused on the problem of selecting an effective subset of paths by proposing the notion of *survivability*. A path set method in a dynamic environment was discussed in [7].

S. Choi, K. Lee, and S. Oh are with the Department of Electrical and Computer Engineering and ASRI, Seoul National University, Seoul 151-744, Korea (e-mail: {sungjoon.choi, kyungjae.lee, songhwai.oh}@cpslab.snu.ac.kr).

Whereas a local planner based on the path set method, which will be referred to as a look-ahead planner (LAP), has been successfully used in many applications, it has a major drawback in that it cannot incorporate a target location into its planning framework. An LAP usually works in a two-step process. First, given a path set, it prunes infeasible paths which are obstructed by obstacles. Then it selects the optimal path with the minimum cost where the cost reflects the objective of planning. As a path set is often pre-computed beforehand due to the real-time constraint, the information about the target location is reflected only through the cost function.

In this paper, we focus on the problem of sampling diverse trajectories that passes certain anchoring points (waypoints) and propose Gaussian random paths by defining a probability distribution over continuous paths (or functions) using Gaussian process regression. The mean path of Gaussian random paths can effectively replace parametric curve models and is shown to be remarkably effective with fewer anchoring points. Furthermore, using the fact that sampled paths from Gaussian random paths connect certain waypoints, e.g., the current position of a robot and its goal position, a Gaussian random path planner is proposed and applied to a local path planning problem and a short-range target tracking problem. We also demonstrate in simulation and experiments that the proposed method outperforms look-ahead planners with a fixed set of paths in terms of collision rates and trajectory lengths.

The remainder of this paper is organized as follows. In Section II, Gaussian process regression and model selection algorithms are discussed. The proposed Gaussian random path is defined in Section III along with a rule of thumb for selecting hyperparameters and an ϵ run-up method for path planning. In Section IV, two applications using Gaussian random paths are illustrated: local path planner and short-range target tracking. In Section V, we conducted local path planning experiments using a Pioneer 3DX mobile robot with two Kinect cameras.

II. Preliminaries

A. Gaussian Process Regression

A Gaussian process defines a distribution over functions, i.e., infinite dimensional vectors, and is completely specified by its mean function $m(\mathbf{x})$ and covariance function $k(\mathbf{x}, \mathbf{x}')$ [9], i.e., a Gaussian process $f(\mathbf{x})$ can be represented as:

$$f(\mathbf{x}) \sim GP(m(\mathbf{x}),\ k(\mathbf{x}, \mathbf{x}')). \tag{1}$$

Suppose that $\mathbf{x} \in \mathbb{R}^d$ is an input and $y \in \mathbb{R}$ is an output, such that $y = f(\mathbf{x}) + w$, where w is a white Gaussian noise. Given

N observed data $D = \{(\mathbf{x}_i, y_i)|\ i = 1, \cdots, N\}$, an output $y_\star \in \mathbb{R}$ for a new input vector $x_\star \in \mathbb{R}^d$ can be predicted by Gaussian process regression (GPR) [9].

Assuming that $y_i = f(\mathbf{x}_i) + w_i$ and $w_i \overset{i.i.d.}{\sim} \mathcal{N}(0, \sigma_w^2)$, the covariance between y_i and y_j can be computed as

$$\mathbf{cov}(y_i,\, y_j) = k(\mathbf{x}_i,\, \mathbf{x}_j) + \sigma_w^2 \delta_{ij}. \tag{2}$$

We can represent the covariance in the following matrix form

$$\mathbf{cov}(\mathbf{y}) = k(\mathbf{X},\, \mathbf{X}) + \sigma_w^2 I, \tag{3}$$

where $\mathbf{y} = [y_1 \dots y_N]^T$, $\mathbf{X} = [\mathbf{x}_1^T \dots \mathbf{x}_N^T]^T$, and $k(\mathbf{X}, \mathbf{X})$ is the covariance matrix computed from N data points.

Let $D = \{(\mathbf{x}_i, y_i) \mid i = 1, \dots, N\}$ be a set of input-output pairs. The conditional distribution of $y_\star$ at a new input $\mathbf{x}_\star$ given data becomes

$$y_\star | D \sim \mathcal{N}(\mu_\star(\mathbf{x}_\star|D),\ \sigma_\star^2(x_\star|D)), \tag{4}$$

where

$$\mu_\star(\mathbf{x}_\star|D) = k(\mathbf{x}_\star, \mathbf{X})^T (k(\mathbf{X}, \mathbf{X}) + \sigma_w^2 I)^{-1} \mathbf{y}, \tag{5}$$

and

$$\sigma_\star^2(\mathbf{x}_\star|D) = k_\star - k(\mathbf{x}_\star, \mathbf{X})^T (k(\mathbf{X}, \mathbf{X}) + \sigma_w^2 I)^{-1} k(\mathbf{x}_\star, \mathbf{X}), \tag{6}$$

where $k_\star = k(\mathbf{x}_\star, \mathbf{x}_\star)$ and $k(\mathbf{x}_\star, \mathbf{X}) \in \mathbb{R}^N$ is a covariance vector between the test point $\mathbf{x}_\star$ and data points $\mathbf{X}$.

One of the widely used kernel functions is a squared exponential (SE) kernel function:

$$k_{SE}(x, x') = g_{SE}^2 \exp\left(\frac{-||x - x'||^2}{2 l_{SE}^2}\right), \tag{7}$$

where g_{SE}^2 and l_{SE}^2 are hyperparameters of a SE kernel function, called gain and length parameters, respectively. Particularly, the length parameter l_{SE}^2 determines how far each training data affects the resulting regression.

B. Model Selection in Gaussian Process Regression

A kernel function indicates our basic assumption about the phenomena we aim to model, and thus deploying a proper kernel function is often the key to successful Gaussian process regression. Recently, Duvenaud [10] proposed an automatic model selection method for searching an appropriate kernel function. In this subsection, we focus on estimating hyperparameters of a fixed kernel function and briefly introduce two model selection algorithms, maximum likelihood estimation and leave-one-out cross validation.

Maximum likelihood estimation (MLE) maximizes the following log marginal likelihood given N training data $(\mathbf{X},\, \mathbf{y})$, where $\mathbf{X}$ is a set of input locations and $\mathbf{y}$ is a concatenated output vector indicating a set of corresponding outputs,

$$\log p(\mathbf{y}|\mathbf{X}, \theta) = -\frac{1}{2} \mathbf{y}^T \mathbf{K}_\mathbf{X}^{-1} \mathbf{y} - \frac{1}{2} \log |\mathbf{K}_\mathbf{X}| - \frac{N}{2} \log 2\pi. \tag{8}$$

Here, $\mathbf{K}_\mathbf{X} = \mathbf{K}(\mathbf{X}, \mathbf{X}) + \sigma_w^2 I \in \mathbb{R}^{N \times N}$ is a kernel matrix of N training inputs, where σ_w^2 is a measurement error variance and θ is a set of hyperparameters for the kernel function. The optimization is usually done by a gradient ascent method since (8) is differentiable with respect to θ.

A leave-one-out cross validation (LOO-CV) method maximizes the following leave-one-out log predictive probability:

$$\begin{aligned} L_{LOO}(\mathbf{X}, \mathbf{y}, \theta) &= \sum_{i=1}^{N} \log(y_i|\mathbf{X}, \mathbf{y}_{-i}, \theta) \\ &= \sum_{i=1}^{N} -\frac{1}{2} \log \sigma_i^2 - \frac{(y_i - \mu_i)^2}{2\sigma_i^2} - \frac{1}{2} \log 2\pi, \end{aligned} \tag{9}$$

where $\mathbf{y}_{-i}$ indicates a set of outputs, excluding the ith element, $\mu_i = y_i - [\mathbf{K}_\mathbf{X}^{-1}\mathbf{y}]_i / [\mathbf{K}_\mathbf{X}^{-1}]_{ii}$, and $\sigma_i^2 = 1/[\mathbf{K}_\mathbf{X}^{-1}]_{ii}$. Similar to MLE, a gradient ascent method can be used to find hyperparameters.

Note that while (8) is the posterior probability of the training data given the assumptions of the model, (9) is the predictive probability which is less influenced by the model assumption [9]. Thus, the LOO-CV method is more likely to be robust against model mis-specification [11] and we used LOO-CV to estimate hyperparameters of a given kernel function.

III. Gaussian Random Paths

In order to describe Gaussian random paths, we first define a path and anchoring points.

Definition 1: A path $\mathbf{p}$ is a vector-valued function where the input space is a time interval $\mathcal{I} \subseteq \mathbb{R}$ and the output space is a set of locations:

$$\mathbf{p} : \mathcal{I} \to \mathbb{R}^d,$$

where d is the dimension of the space.

Definition 2: Anchoring points $D_a = (\mathbf{s}_a, \mathbf{x}_a)$ are a set, which consists of M time indices $\mathbf{s}_a = \{s_i | i = 1, 2, ..., M\}$ and locations $\mathbf{x}_a = \{\mathbf{x}_i | i = 1, 2, ..., M\}$, such that a path must pass through.

In order to sample paths, we treat a path defined in Definition 1 as a random process and exploit a Gaussian process to define a probability distribution over continuous paths. A Gaussian process is a generalization of the Gaussian probability distribution. Whereas a probability distribution describes random variables which are scalar or vectors, a stochastic process governs the properties of functions. While dealing with infinite dimensional objects is computationally infeasible in most cases, the marginalization property of the Gaussian distribution makes it tractable [9]. This property is particularly important in modeling paths as it indicates a path modeled by a Gaussian process is free from quantization issues.

Proposition 1: A path modeled by a Gaussian process is quantization free.

Unfortunately, a Gaussian process itself cannot model a path which passes through certain anchoring points and, to achieve this, we revisit Gaussian process regression (GPR) in Section II-A. GPR computes a Gaussian posterior distribution using (5) and (6) and we can use this GPR framework to model a path with anchoring points, where the training

data corresponds to the anchoring points. Now, we are ready to define Gaussian random paths, which define a probability distribution over paths interpolating a set of anchoring points.

Definition 3: Given a kernel function $k(t, t')$ and a set of M anchoring points $D_a = (\mathbf{s}_a, \mathbf{x}_a) = \{(s_i, x_i) \,|\, i = 1, 2, \dots, M\}$, a Gaussian random path $\mathcal{P}$ with a sequence of T test time indices $\mathbf{t}_{test} = \{t_i \,|\, i = 1, 2, \dots, T\}$ is specified with a mean path $\mu_{\mathcal{P}}$ and a covariance matrix $K_{\mathcal{P}}$.

$$\mathcal{P} \sim \mathcal{N}(\mu_{\mathcal{P}},\, K_{\mathcal{P}}), \tag{10}$$

where

$$\mu_{\mathcal{P}} = \mathbf{k}(\mathbf{t}_{test},\, \mathbf{s}_a)^T (\mathbf{K}_a + \sigma_w^2 I)^{-1} \mathbf{x}_a, \tag{11}$$

$$\mathcal{K}_{\mathcal{P}} = \mathbf{K}_{test} - \mathbf{k}(\mathbf{t}_{test},\, \mathbf{s}_a)^T (\mathbf{K}_a + \sigma_w^2 I)^{-1} \mathbf{k}(\mathbf{t}_{test},\, \mathbf{s}_a), \tag{12}$$

$\mathbf{k}(\mathbf{t}_{test}, \mathbf{s}_a) \in \mathbb{R}^{T \times M}$ is a kernel matrix of test time indices and anchoring time indices, $\mathbf{K}_a = \mathbf{K}(\mathbf{s}_a, \mathbf{s}_a) \in \mathbb{R}^{M \times M}$ is a kernel matrix of anchoring time indices, and $\mathbf{K}_{test} = \mathbf{K}(\mathbf{t}_{test}, \mathbf{t}_{test}) \in \mathbb{R}^{T \times T}$ is a kernel matrix of test time indices. Note that this is an one-dimensional path case and can easily be extended to a d-dimensional path by assuming independence between each dimension.

Remark 1: Once a set of anchoring points and test time indices are fixed, (10) becomes a multivariate Gaussian distribution with mean (11) and variance (12). Thus, generating paths interpolating a set of anchoring points is equivalent to sampling from a Gaussian distribution.

The proposed Gaussian random path (GRP) differs from usual Gaussian process regression in that, whereas the main purpose of regression is to predict an output $y_\star$ of an unseen input $\mathbf{x}_\star$ using (5), the GRP focuses on defining a distribution over a sequence of time indices in the interval $\mathcal{I}$ and sampling paths interpolating a set of anchoring points from the distribution.

Another useful property of Gaussian random paths is that paths sampled from the GRP distribution are C^n continuous, if $2n$ moments of the scale parameter exist [12]. While computing the nth momentum of the scale parameter can be demanding, it is known that a sample path with a squared exponential and rational quadratic kernel function are infinitely differentiable. This property is particularly important in path planning since we often require smooth paths satisfying certain constraints.

Proposition 2: Paths sampled from the GRP distribution with a squared exponential kernel function are C^∞ continuous.

The proposed Gaussian random paths can be effectively used in sampling-based trajectory optimization algorithms, e.g., STOMP [13]. While STOMP requires an additional smoothing step as it utilizes noisy sampled trajectories, paths sampled from the Gaussian random path planner in Section IV are already smooth and thus an additional step is not required. Moreover, since the GRP distribution can incorporate any arbitrary anchoring points into account, it can be applied to the problem of optimizing a trajectory over waypoints.

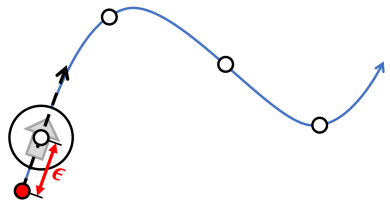


Fig. 1: An ϵ run-up method applied to GRPs with three anchoring points. An added extra anchoring point is shown in a red disk.

A. ϵ Run-Up Method

When it comes to applying GRPs to a unicycle model, sampled paths may not directly be used by a low level controller due to its nonholonomic constraints. In other words, the resulting path should be *trackable* with respect to the specific dynamic model.

Definition 4: A path $\mathbf{p}(t) = [\mathbf{p}_x(t)\ \mathbf{p}_y(t)]^T$ is *trackable* if the gradient of a path $[\frac{d\mathbf{p}_x(t)}{dt}\ \frac{d\mathbf{p}_y(t)}{dt}]^T$ at t_0 equals to the heading vector $[v \cdot \cos(\theta_0)\ v \cdot \sin(\theta_0)]^T$ of a robot, where t_0 is the time index of a robot at its origin, v is the directional velocity, and $\theta_0 \in [0, 2\pi)$ is the heading of a robot.

This problem can be handled by an ϵ run-up method. Suppose that a position, a heading, and a directional velocity of a robot are $\mathbf{x}_0 = [x_0\ y_0]^T$, $\theta_0 \in [0, 2\pi)$, and v, respectively. Then the ϵ run-up method adds an additional anchoring point $[t_{ru}\ \mathbf{x}_{ru}]^T$ to the original set of anchoring points as shown in Figure 1, where

$$t_{ru} = t_0 - \epsilon / v \tag{13}$$

and

$$\mathbf{x}_{ru} = [x_0 - \epsilon \cdot \cos(\theta_0)\ y_0 - \epsilon \cdot \sin(\theta_0)]^T. \tag{14}$$

If a sampled path is C^1 continuous, then the path becomes *trackable* as ϵ goes to zero.

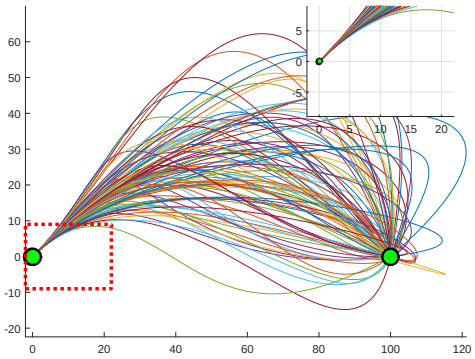
Proposition 3: Paths sampled from the GRP distribution with a squared exponential kernel function with the ϵ run-up method are *trackable* as ϵ goes to zero.

Proof: Let $\mathbf{x}_0$ and θ_0 be the position and heading of a robot, respectively, and $\mathbf{p}$ be a sampled path from the GRP distribution with a squared exponential kernel function given an ϵ run-up anchoring point. Then $[\frac{\epsilon \cdot \cos(\theta_0)}{\epsilon / v}\ \frac{\epsilon \cdot \sin(\theta_0)}{\epsilon / v}]^T$ corresponds to the left-derivative of $\mathbf{p}$ at t_0. The path $\mathbf{p}$ must be infinitely differentiable at $\mathbf{x}_0$ from Proposition 2 meaning that the left derivative at $\mathbf{x}_0$ equals to the right derivative, which completes the proof. ■

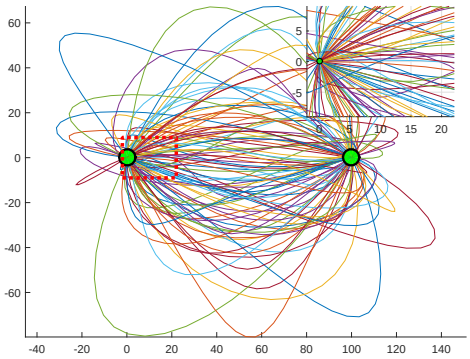
Figure 2 shows sampled paths with and without the ϵ run-up method. One can clearly see that the paths sampled with the ϵ run-up method are more suitable for control than the paths sampled without the ϵ run-up method.

B. Rule of Thumb in Model Selection

As shown in Section II, hyperparameters of a kernel function play an important role in using a Gaussian process. These hyperparameters can be estimated using methods described in Section II, if a sufficient number of training



(a)



(b)

Fig. 2: Paths sampled from the GRP distribution (a) with ϵ run-up (b) without ϵ, where the heading of a robot is $45°$.

data are provided. However, this is not the case for GRPs since anchoring points are the training data and the number of anchoring points is usually small.

In this section, we give a rule of thumb for selecting hyperparameters, g_{SE}^2 and l_{SE}^2, of the squared exponential (SE) kernel function (7) based on the maximum distance between given anchoring points.

Let d_{max} be the maximum distance between two consecutive anchoring points. Then a rule of thumb for selecting the scale and length parameters of the SE kernel in (7) is

$$\bar{g}_{SE}^2 = d_{max}^2 \tag{15}$$

$$\bar{l}_{SE}^2 = d_{max}^2. \tag{16}$$

Figure 3 illustrates the effects of the length parameter to the sampled paths passing four points, $(1,1)$, $(1,4)$, $(4,4)$, and $(4,1)$, where the mean path is shown in a green dotted line. One can clearly see that the resulting paths get more diverse and complex as the length parameter, l_{SE}^2, gets smaller. This is closely related to the fact that the mean number of level-zero upcrossings N_u for a Gaussian process with a squared exponential kernel function in 1D is

$$\mathbb{E}[N_u] = \frac{1}{2\pi}\sqrt{\frac{-k_I''(0)}{k_I(0)}} = \frac{1}{2\pi l_{SE}},$$

where $k_I(t) = g_{SE}^2 \exp\left(-\frac{t^2}{2l_{SE}^2}\right)$.

C. Comparison With Parametric Curve Fitting Models

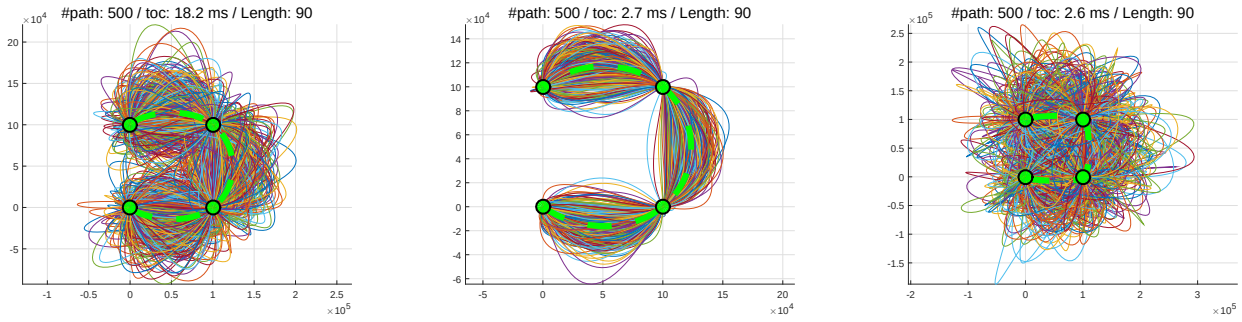
An objective of a parametric curve fitting model is to find an underlying parametric curve, e.g., a polynomial function, that best expresses given data points. A spline model is a set of piecewise polynomial functions and is widely used to interpolate data points. These models are fully specified by a set of polynomial coefficients and the required number of parameters is proportional to the number of piecewise functions and the degree of a polynomial. For example, a cubic spline function with n splines is defined by $6n$ parameters and, thus, requires $6n$ data points.

Figure 4(a) shows mean fitting errors of the mean path (11) of the proposed Gaussian random path and cubic spline models with 5 and 20 splines. A reference path $\mathbf{p}_{ref}$ and an interval $\mathcal{I}$ are given by $\mathbf{p}_{ref}(t) = [\sin(t/2) \;\; t(t-10)(t-20)/500]^T$ and $\mathcal{I} = (10, 30]$, respectively. The fitting error is measured by the root mean square, i.e., $\sqrt{\frac{1}{N}\sum_{i=1}^{N}(\mathbf{p}_{ref}(t_i) - \mathbf{p}_{fit}(t_i))^2}$, where $N = 500$, $\mathbf{p}_{fit}(t)$ indicates the fitted path which can either be from the GRP or cubic spline models, and each test time index t_i is uniformly placed in $\mathcal{I}$ while varying the number of anchoring points from 5 to 200. In each scenario, each time index s_j of an anchoring point is randomly sampled from $\mathcal{I}$ and the corresponding location is computed from $\mathbf{p}_{ref}(t)$ with a white Gaussian noise, i.e., $\mathbf{x}_j = \mathbf{p}_{ref}(s_j) + w_j$ where $w_j \sim \mathcal{N}(0, 10^{-4})$ and 10 independent trials are performed to compute the average. The hyperparameters of the SE kernel function is estimated from leave-one-out cross validation described in Section II-B.

The effect of the number of splines is well illustrated in Figure 4(a). While fitting with a small number of splines (5 splines) shows relatively better performance when the number of data points is low, the performance improvement is negligible as more data points are given indicating a poor asymptotic property. On the contrary, fitting with a larger number of splines (20 splines) shows a poor performance when the number of data points is low. However, it outperforms the case with a small number of splines as the number of data points increases.

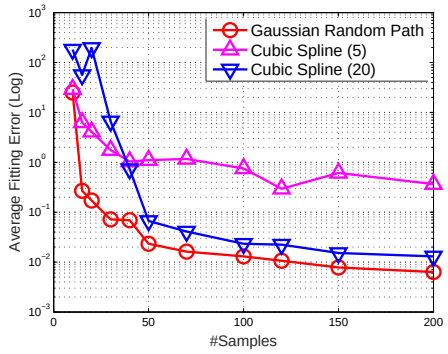
On the contrary, the proposed GRP is a nonparametric generative model which is less influenced by the number of anchoring points. Moreover, when it comes to generating diverse paths connecting given anchoring points, parametric models can hardly be used. In other words, whereas parameters of a spline model are deterministically defined once anchoring points are given, the proposed GRP defines a probability distribution over the interpolating paths and, thus, we can easily sample paths connecting anchoring points.

Figure 4(b) shows the fitted curves using the mean path of GRPs and the cubic spline models with 5 and 20 splines where 10 data points are given. One can clearly see that an overfitting occurred in the spline model with 20 splines mainly due to the fact that it tries to estimate 120 parameters from 10 training data. Note that it is well known that Gaussian process regression with a polynomial kernel function is equivalent to solving a spline smoothing problem [9].

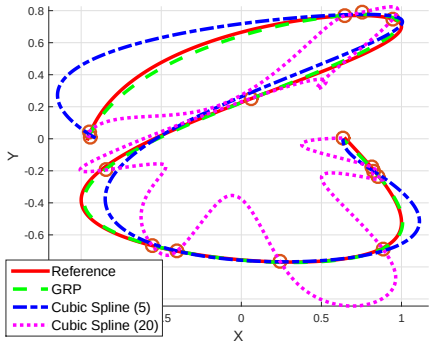


(a) (b) (c)

Fig. 3: Sampled paths with (a) rule-of-thumb hyperparameters, $(\bar{g}_{SE}^2, \bar{l}_{SE}^2)$, (b) twice the length parameter, $(\bar{g}_{SE}^2, 2 \cdot \bar{l}_{SE}^2)$, and (c) a half of the length parameter, $(\bar{g}_{SE}^2, 0.5 \cdot \bar{l}_{SE}^2)$, where $\bar{g}_{SE}^2$ and $\bar{l}_{SE}^2$ are selected according to (15) and (16).



(a)



(b)

Fig. 4: (a) The fitting errors (in a log scale) of the GRP and spline models with 5 and 20 splines as functions of the number of samples. (b) A snapshot of curves fitted with the mean path of the GRP (green dotted line) and cubic spline functions with 5 (blue dashed line) and 20 (magenta dashed line) splines. When the number of anchoring points is low, the spline model with 20 splines tends to overfit.

IV. Applications

A. Local Path Planner

In many cases, a mobile robot is ordered to accomplish assigned tasks by navigating through given waypoints. For instance, in the DARPA Urban Challenge, the objective of a path planner was to find a collision-free local path connecting given waypoints using sensory measurements [1].

The proposed Gaussian random path can be effectively applied to a local planning problem. Given the current position of a robot and the goal position, we first sample a diverse set of trackable paths from the GRP distribution with the ϵ run-up method from Section III-A, where ϵ is set to 100mm. The size of ϵ is adjusted to be sufficiently small compared to the size of the operation region which is 10m $\times$ 10m. Then an occupancy grid map constructed from the current range sensor measurements is used to examine the sampled paths and the algorithm selects the shortest collision free path with respect to the occupancy grid map. However, this procedure requires heavy computational loads as it has to check each position of a sampled path whether it is obstructed or not.

To handle this issue, we utilized Graphics processing units (GPUs) to alleviate the computational load by assigning a GPU thread to each point in a path. In particular, we used an NVIDIA GTX870m graphics card with 1344 CUDA cores and 6GB of memory and achieved more than 500 times speedup, which requires less than 10ms to examine 500 paths. An overall path planning algorithm is summarized in Algorithm 1. For a low-level controller, we used a pure-pursuit algorithm [14]. The whole control process including generating 500 paths and examination took about 10ms in MATLAB.

Algorithm 1 Gaussian Random Path Planner (GRPP)

1: **Input** An occupancy grid map M, a robot position $\mathbf{x}_0$, a directional velocity v, and a goal position $\mathbf{x}_g$
2: **Output** A Gaussian random path $\mathbf{p}_{opt}$
3: Sample a set $\mathcal{P}$ of paths from the GRP distribution with anchoring points $D_a = (\mathbf{s}_a, \mathbf{x}_a)$ where $\mathbf{s}_a = \{t_{ru}, t_0, t_g\}$, $\mathbf{x}_a = \{\mathbf{x}_{ru}, \mathbf{x}_0, \mathbf{x}_g\}$, $t_0 = 0$, and $t_g = ||\mathbf{x}_g - \mathbf{x}_0||/v$.
4: Find the shortest collision-free path $\mathbf{p}_{opt}$ among $\mathcal{P}$ with respect to M.

To validate the performance of the proposed method, a Gaussian random path planner (GRPP), we compared our

	GRPP	LAP (2s)	LAP (4s)	LAP (6s)	LAP (8s)	LAP (10s)
Sampling Time (ms)	0.82 (0.037)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
Examine Time (ms)	9.84 (0.13)	10.08 (0.11)	10.06 (0.12)	10.04 (0.12)	10.01 (0.09)	10.01 (0.13)
Total Time (ms)	10.66 (0.14)	10.08 (0.11)	10.06 (0.12)	10.04 (0.12)	10.01 (0.09)	10.01 (0.13)

TABLE I: Computation times of tested path planners: GRPP with 500 paths and LAPs with $1,000$ paths. (The number inside parentheses is the standard deviation.)

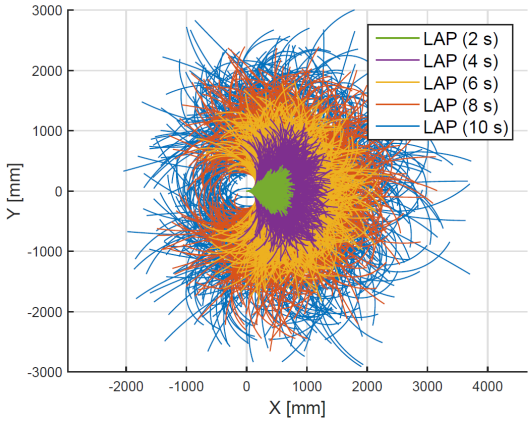


Fig. 5: $1,000$ pre-chosen paths for each look-ahead planner (LAP) with different look-ahead times shown in different colors.

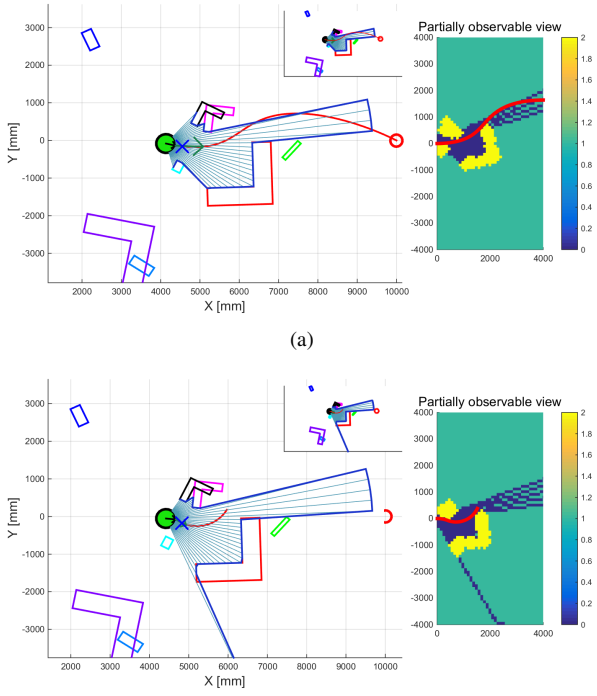


Fig. 6: Snapshots of the local planning scenario with (a) the proposed Gaussian random path planner (GRPP) and (b) the look-ahead planner (LAP) with 8 sec look-ahead time.

method with a look-ahead planner (LAP) [1]–[6], which utilizes a path set method where the look-ahead time of a path set differs from 2 to 10 seconds. For fair comparison, $1,000$ paths are pre-computed offline in each path set as this setting makes the computing time of each path planner similar as shown in Table I. The pre-computed paths are shown in Figure 5. Note that it took more than one second to generate $1,000$ paths, making it infeasible to generate a path set in real-time. On the other hand, it takes about 10ms to sample 500 paths and find the shortest feasible path using the proposed GRPP, making it a suitable choice for real-time local path planning.

We assume that a robot can only obtain information form its egocentric view with a 120° field of view to make a realistic setup in that a decision is made solely based on the occupancy grid map generated from a partially observable view (an example is shown at the right side of Figure 6). For each run, the goal is to reach the target position located at $(10,000, 0)$ and the number of obstacles varies from 2 to 12 and they are randomly placed. For each setting, 50 independent runs are performed and collision rates as well as the total moved distance is computed. Excluding the path generation step, the LAP works identical to the GRPP. Figure 6 shows snapshots of the local path planning scenarios using the GRPP and the LAP.

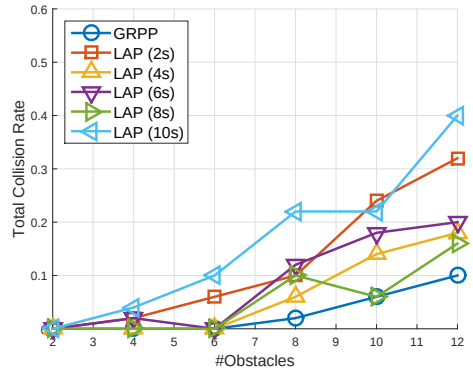
The quantitative simulation results are shown in Figure 7. In Figure 7(a), the total collision rate is presented. In most cases, the GRPP outperforms the LAPs and this is mainly due to the fact that whereas the GRPP considers a full path connecting the current robot position to the target position, the LAP only considers its look-ahead period. The total moved distance is shown in Figure 7(b) and we can see that the GRPP achieves shorter moving distances.

Figure 7(c) illustrates an immanent drawback of the LAP in that it is sensitive to the look-ahead period. The LAP with a shorter look-ahead period makes a myopic control leading to a higher direct collision rate. In contrast, as the look-ahead period gets longer, the probability of entering an inevitable collision state, where all paths are obstructed by obstacles, increases. This phenomenon is shown in Figure 7(c).

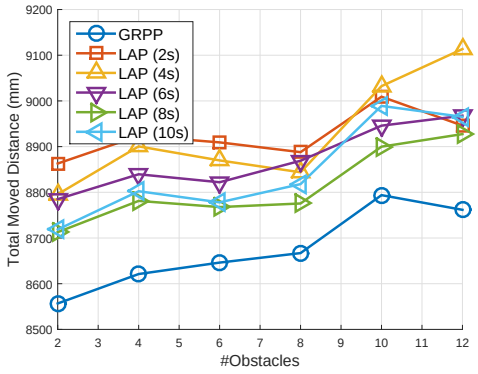
B. Short Range Target Tracking

Target tracking can be formulated as a nonlinear constrained optimization of finding the optimal path while guaranteeing feasibility, i.e., not obstructed by obstacles. One of the main difficulties is the real-time constraint as the start and goal locations of the path planning are constantly varying.

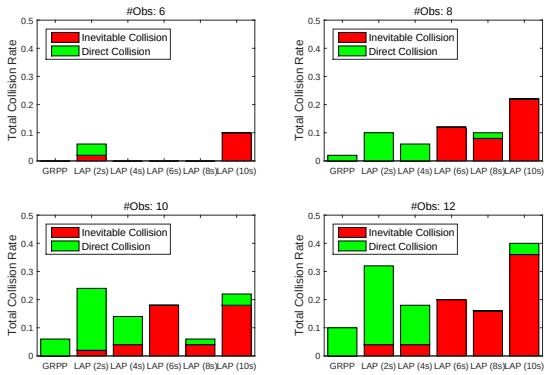
The Gaussian random path planner (GRPP) in Algorithm 1 can be applied to implement a target tracking algorithm. The main difference between the target tracking problem and local path planning is whether the target is moving



(a)



(b)

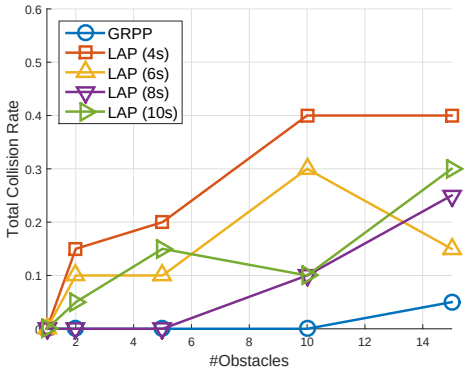


(c)

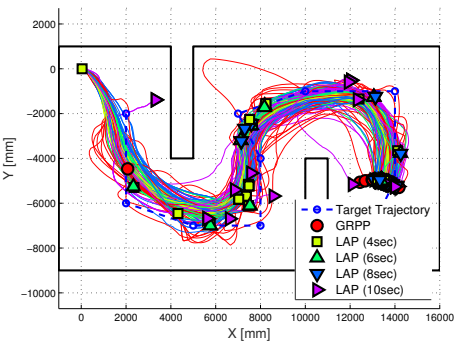
Fig. 7: Local path planning simulation results of the proposed GRPP and look-ahead planners (LAPs) with different prediction periods (2, 4, 6, 8, and 10 seconds): (a) total collision rates, (b) total moved distances, and (c) inevitable collision and direct collision rates.

or not. While this is problematic in a single query path planning algorithm, e.g., RRT or RRT*, or ones that require pre-computation stages, e.g., a probabilistic roadmap, the proposed GRPP can be directly used in a tracking scenario.

We used the same configuration as the local path planning scenario in that the robot can only use the information acquired from its egocentric view with 120° field of view.



(a)



(b)

Fig. 8: Target tracking simulation results. (a) Total collision rates. (b) Target trajectories are shown in blue dashed lines moving from left to right. Moving paths and final positions of tested path planners are shown in different colors and shapes, respectively. While the paths of the GRPP (red lines) make more detour than those of the LAPs, it managed to reach the final target location in most cases.

The testing map and target trajectory are shown in Figure 8(b) with black lines and blue dashed lines, respectively. To validate the performance of the path planners, we vary the number of obstacles from 2 to 14 and measure collision rates from 20 independent runs. The obstacles are randomly deployed inside the testing map.

The simulation results are depicted in Figure 8. Total collision rates of tested path planners are shown in Figure 8(a) and it shows that the GRPP outperforms LAPs. Figure 8(b) shows paths of tested path planners and their final locations in different colors and shapes. Obstacle configurations are not shown in the figure as they are randomly deployed in each scenario. While trajectories of the GRPP (red lines) are more complicated than those of the look-ahead planners, we can see that it successfully manage to navigate through obstacles in most of the cases as the majority of last positions of GRPP reach the final target position.

V. Local Path Planning Experiments

We conducted local path planning experiments using a Pioneer 3DX mobile robot with two mounted Microsoft

		Convex wall	Concave wall
GRP	collision rate	0%	0%
	moving distance	4477.8 (182.0)	4537.4 (233.6)
LAP (8s)	collision rate	0%	21.4%
	moving distance	4583.6 (440.7)	4317.9 (170.0)

TABLE II: Average collision rates and moving distances (mm) of GRPP and LAP in two wall configurations. (The number inside parentheses is the standard deviation.)

Kienct cameras. The objective is to navigate safely to the goal position 4m away in two different wall configurations as shown in Figure 9(a) and Figure 9(b), which we will refer to as a convex wall and a concave wall, respectively. While path planning in a convex wall is relatively easy, the horseshoe-shaped wall configuration in a concave wall makes navigation hard as a robot is more likely to enter an inevitable collision state.

We compared the proposed Gaussian random path planner (GRPP) with the look-ahead planner (LAP) with 8 seconds look-ahead time as it showed the best performance among other LAPs as shown in Section IV. For each scenario, 10 independent experiments are performed and the quantitative results are shown in Table II. Trajectories of the robot at each scenario are shown in Figure 9(c).

In the easier case (convex wall), both the GRPP and the LAP successfully control the robot to the goal position and the average moved distance of the GRPP is slightly shorter than that of the LAP. On the other hand, for the concave wall case, the LAP shows a collision rate of 21%, whereas the GRPP achieves collision-free navigation in all cases.

VI. Conclusion

In this paper, we have proposed a rapid path sampling method, called Gaussian random paths, to generate a diverse set of paths passing through anchoring points using Gaussian process regression. The proposed algorithm can effectively replace parametric curve fitting models and it is shown to be more effective especially when data points are scarce. Furthermore, the Gaussian random path planner is proposed by exploiting the efficiency of Gaussian random paths. By fully utilizing GPUs, the overall process of generating 500 random paths and finding the optimal feasible path takes about 10ms in MATLAB. In both simulations and experiments, the proposed GPRR outperforms the LAP in all cases in terms of collision rates and trajectory lengths.

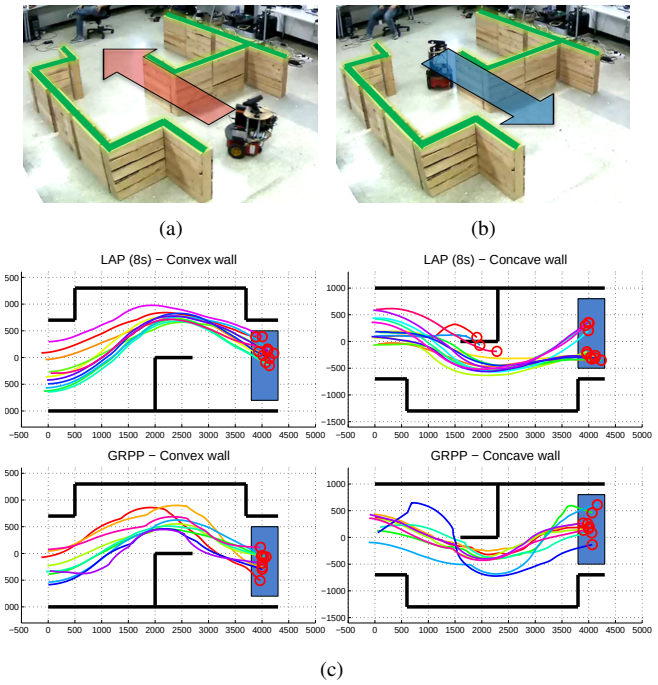


Fig. 9: (a) Convex wall. (b) Concave wall. (c) Trajectories of a Pioneer 3DX mobile robot using the GRPP and LAP with 8 seconds look-ahead. The goal is to safely navigate through walls and reach the blue shaded area. The wall configuration is not available to both algorithms.

References

[1] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann *et al.*, "Stanley: The robot that won the darpa grand challenge," *Journal of field Robotics*, vol. 23, no. 9, pp. 661–692, 2006.

[2] F. Von Hundelshausen, M. Himmelsbach, F. Hecker, A. Mueller, and H.-J. Wuensche, "Driving with tentacles: Integral structures for sensing and motion," *Journal of Field Robotics*, vol. 25, no. 9, pp. 640–673, 2008.

[3] J. Bohren, T. Foote, J. Keller, A. Kushleyev, D. Lee, A. Stewart, P. Vernaza, J. Derenick, J. Spletzer, and B. Satterfield, "Little ben: the ben franklin racing team's entry in the 2007 darpa urban challenge," *Journal of Field Robotics*, vol. 25, no. 9, pp. 598–614, 2008.

[4] D. Ferguson, T. M. Howard, and M. Likhachev, "Motion planning in urban environments," *Journal of Field Robotics*, vol. 25, no. 11-12, pp. 939–960, 2008.

[5] A. Bacha, C. Bauman, R. Faruque, M. Fleming, C. Terwelp, C. Reinholtz, D. Hong, A. Wicks, T. Alberi, D. Anderson *et al.*, "Odin: Team victortango's entry in the darpa urban challenge," *Journal of Field Robotics*, vol. 25, no. 8, pp. 467–492, 2008.

[6] F. W. Rauskolb, K. Berger, C. Lipski, M. Magnor, K. Cornelsen, J. Effertz, T. Form, F. Graefe, S. Ohl, W. Schumacher *et al.*, "Caroline: An autonomously driving vehicle for urban environments," *Journal of Field Robotics*, vol. 25, no. 9, pp. 674–724, 2008.

[7] R. A. Knepper and M. T. Mason, "Path diversity is only part of the problem," in *Proc. of the International Conference of Robotics and Automation (ICRA)*. IEEE, 2009.

[8] L. H. Erickson and S. M. LaValle, "Survivability: Measuring and ensuring path diversity," in *Proc. of the International Conference of Robotics and Automation (ICRA)*. IEEE, 2009.

[9] C. Rasmussen and C. Williams, *Gaussian processes for machine learning*. MIT press Cambridge, MA, 2006, vol. 1.

[10] D. Duvenaud, "Automatic model construction with Gaussian processes," Ph.D. dissertation, Computational and Biological Learning Laboratory, University of Cambridge, 2014.

[11] G. Wahba, *Spline models for observational data*. CBMS-NSF Regional Conference Series in Applied Mathematics, 1990, vol. 59.

[12] C. J. Paciorek, "Nonstationary Gaussian processes for regression and spatial modelling," Ph.D. dissertation, Carnegie Mellon University, 2003.

[13] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "STOMP: Stochastic trajectory optimization for motion planning," in *Proc. of the International Conference of Robotics and Automation (ICRA)*. IEEE, 2011.

[14] O. Amidi and C. E. Thorpe, "Integrated mobile robot control," Robotics Institute, Carnegie Mellon University, Tech. Rep. Robotics Institute, Carnegie Mellon University, 1990.