CollisionGP: Gaussian Process-Based Collision Checking for Robot Motion Planning

Javier Muñoz, Peter Lehner, Luis E. Moreno, Alin Albu-Schäffer, and Máximo A. Roa

***Abstract*—Collision checking is the primitive operation of motion planning that consumes most time. Machine learning algorithms have proven to accelerate collision checking. We propose CollisionGP, a Gaussian process-based algorithm for modeling a robot's configuration space and query collision checks. CollisionGP introduces a Pòlya-Gamma auxiliary variable for each data point in the training set to allow classification inference to be done exactly with a closed-form expression. Gaussian processes provide a distribution as the output, obtaining a mean and variance for the collision check. The obtained variance is processed to reduce false negatives (FN). We demonstrate that CollisionGP can use GPU acceleration to process collision checks for thousands of configurations much faster than traditional collision detection libraries. Furthermore, we obtain better accuracy, TPR and TNR results than state-of-the-art learning-based algorithms using less support points, thus making our proposed method more sparse.**

***Index Terms*—Collision avoidance, Gaussian processes, machine learning, motion planning.**

I. Introduction

PATH planning is the task of creating a path to move a robot from a start configuration to a target configuration while avoiding self-collisions and collisions with the environment. Most path planning solutions are calculated in the configuration space (C-space) of the robot rather than in the Cartesian space of the environment. In the C-space, each dimension represents a degree of freedom (DoF) of the robot [1]. As the number of DoFs of the robot increases, the dimensionality of the C-space grows, thus increasing the complexity of the path planning problem. To plan collision-free paths, path planning algorithms rely on collision checkers. The algorithms determine if a specific configuration belongs to the free configuration space C_{free} or to the configuration space blocked by obstacles C_{obs}. The queries to the collision checker are the most computationally expensive primitive operation of path planning algorithms, accounting for about 90% of the computation time [2].

The configuration space can be modeled by standard machine learning techniques, allowing the model to receive queries from the path planning algorithm. However, such a model is rendered invalid when any aspect of the environment changes. Computationally expensive models with millions of weights, such as Artificial Neural Networks (ANNs), cannot be updated by an algorithm fast enough to be useful in changing environments. In this letter, we introduce a novel approach to collision detection using Gaussian processes (GPs), aimed to building a lighter and computationally less expensive model that reduces the computation time spent on collision checking during path planning applications.

We propose CollisionGP, a lightweight GP model based on Pòlya-Gamma auxiliary variables for binary classification [3]. This is the first application of GPs to the problem of checking collisions in the configuration space of a robot, to the best of our knowledge. Other methods, however, use GPs for motion planning [4], [5], [6]. The main advantages of GPs is that they are easy and fast to train since their number of hyperparameters is significantly smaller than other methods such as ANNs, and they provide a full probabilistic distribution as the output of the model. This allows us not only to determine if a query point belongs to C_{free} or C_{obs}, but to measure the uncertainty of the model about that prediction. We use this uncertainty measure to weight the decision boundary towards eliminating false negatives, which would lead to a collision of the robot with an obstacle or with itself. This is extremely important in critical applications such as chemical waste handling or surgical robotics.

We choose the rational quadratic kernel with automatic relevance determination (ARD) as the GP kernel, which improves the model accuracy by ruling out the DoFs that do not directly influence the collisions. To obtain a lightweight model, we use a set of inducing points to perform the predictions on the path planning algorithm queries. We demonstrate that few inducing points are needed to obtain trustable and accurate C-space maps. Moreover, we show that an algorithm can update the model fast enough to adapt to changing environments while keeping the number of inducing points constant.

II. Related Work

Machine learning techniques have been previously employed to model the C-space of a robot and accelerate collision checking in path planning.

Manuscript received 30 January 2023; accepted 8 May 2023. Date of publication 29 May 2023; date of current version 2 June 2023. This letter was recommended for publication by Associate Editor P. Gao and Editor A. Bera upon evaluation of the reviewers' comments. This work was supported in part by the DLR Internal Project Factory of the Future Extended, in part by Helmholtz Association, sustainability challenge iFOODis KA-HSC-06_iFOODis and Project COVIPA KA1-Co-02, in part by the Bavarian Ministry of Economic Affairs, Regional Development and Energy, under Project OPERA DIK-2107-0005//DIK0374/02, and in part by European Commission: Innovation and Networks Executive Agency (INEA), through the European H2020 LABYRINTH Project, under Grant Agreement H2020-MG-2019-TwoStages-861696. *(Corresponding author: Javier Muñoz.)*

Javier Muñoz and Luis E. Moreno are with the Department of Automation and Systems Engineering, Universidad Carlos III de Madrid, 28911 Leganés, Spain (e-mail: jamunozm@ing.uc3m.es; moreno@ing.uc3m.es).

Peter Lehner, Alin Albu-Schäffer, and Máximo A. Roa are with the Institute of Robotics and Mechatronics, German Aerospace Center (DLR), 82234 Weßling, Germany (e-mail: peter.lehner@dlr.de; Alin.Albu-Schaeffer@dlr.de; maximo.roa@dlr.de).

Digital Object Identifier 10.1109/LRA.2023.3280820

Pan et al. [7] use support vector machines (SVMs) to generate an accurate decision boundary between C_{free} and C_{obs} for two objects, and present an active learning strategy to improve this boundary iteratively. In our case, the computations of CollisionGP can be processed online and the algorihtm does not need a new model for every pair of objects.

Arslan et al. [8] use a Bayesian classifier to compute two approximate probability density functions to determine C_{free} and C_{obs} based on the available data, and then use the Bayesian rule to predict if a query point belongs to C_{free} or C_{obs}. CollisionGP has the advantage over naive Bayes classifiers of being able to provide a predictive variance for the model outputs.

Danielczuk et al. [9] use a neural network that learns collisions from scenes, represented as point clouds, and is able to predict collisions for 6 DoF object poses within the scene. However, they need 1 million scene/object point cloud pairs and more than 2 billion collision queries to train the network and they need a camera with depth perception to perform collision checks. They show the applicability of the method with a robot performing tabletop rearrangement tasks. Kew et al. [10] present ClearanceNet and CN-RRT, a neural network collision checking heuristic and a planning algorithm, respectively. CN-RRT uses the capability of ClearanceNet to process batches of thousands of collision checks to efficiently and quickly plan paths. Qureshi et al. [11] use Motion Planning Networks, a neural network that encodes the workspace given a point cloud and computes collision free paths for the robot. The training is done via expert demonstrations, using 4000 demonstrations for 100 different workspaces. The neural network is able to generate multiple collision free paths for a single combination of start and goal configurations in a finite time due to its stochastic behavior. Ichter et al. [12] use learned critical probabilistic roadmaps to plan robot motions. For this purpose, they train a neural network with previously created probabilistic roadmaps with critical zones calculated via betweenness centrality [13]. The main advantage of CollisionGP over neural network approaches is that our method relies on a limited set of data points and can be updated with dynamic obstacles, while neural networks require tens of thousands of data points and are not as flexible. Also, CollisionGP does not require a lot of data and parameter tuning for training, not having to choose the architecture or needing to optimize the hundreds to millions weights of the network. Moreover, CollisionGP can provide full probabilistic distributions as model predictions.

Pan et al. [14] store prior query samples and use k-NN (k-nearest neighbor) search to find prior query samples that are close to the new query configuration. Then, they estimate the probability that the new query configuration belongs to C_{free} or C_{obs} based on the found query samples. CollisionGP has the advantage of being scalable, since all collision checks are not stored in hash tables that grow larger with the dataset size, and it is also able to provide predictive variances.

Han et al. [15] train a different classifier for each DoF of the robot, obtaining better results than SVMs and KNNs trained on all DoFs at the same time. CollisionGP improves on this approach by using automatic relevance determination to determine the relevance of each dimension, instead of user inputs, and can predict variances for the model outputs.

Das et al. [16] propose Fastron, a learning-based algorithm that uses a support vector machine to classify points belonging to C_{free} and C_{obs}. They demonstrate that they can train the model and query collision checks an order of magnitude faster than state-of-the-art collision libraries, making it capable of adapting to dynamic environments. They demonstrate the applicability to robots by performing pick and place and surgery tasks. CollisionGP improves on this approach by using automatic relevance determination for the model inputs, which makes it able to maintain its accuracy with any robot, and by being able to provide uncertainty values for the model outputs.

III. Methodology

This section describes the components of the CollisionGP algorithm. We begin by describing the binary classification problem applied to GPs. Then, we provide an overview of the kernel selection for the task. Next, we go into the details of generating the dataset and creating and training the GP model. In the next subsection we show how the proposed algorithm takes the generated output mean and variance to make predictions on the query points. Then, we highlight the capability of the model to take advantage of CUDA and use batches of data as inputs instead of single query points, which can parallelize the computations and provide faster predictions as evaluated in a comparison with the traditional FCL method. Finally, we showcase the ability of the model to quickly adapt to changes in the environment while keeping the same model size by sampling around the current inducing points locations.

A. Pòlya-Gamma Auxiliary Variables: Binary Classification

Gaussian processes in the context of classification do not support exact inference with a closed form expression. One possible solution is to introduce additional latent variables that restore conjugacy. We introduce a Pòlya-Gamma auxiliary variable for each data point in the training dataset. The Pòlya-Gamma [3] distribution (PG) is a univariate distribution with support on the positive real line. In our context PG is interesting because if ω_i is distributed according to $\mathrm{PG}(1,0)$ then the logistic likelihood $\sigma(\cdot)$ for data point (x_i, y_i) can be represented as:

$$\begin{aligned}\sigma(y_i f_i) &= \frac{1}{1+\exp(-y_i f_i)} \\ &= \tfrac{1}{2}\mathbb{E}_{\omega_i \sim \mathrm{PG}(1,0)}\left[\exp\left(\tfrac{1}{2}y_i f_i - \tfrac{\omega_i}{2} f_i^2\right)\right]\end{aligned}$$

where $y_i \in \{-1, 1\}$ is the binary label of data point i, and f_i is the Gaussian Process prior evaluated at input x_i, in this case $[q_0, q_1, \ldots, q_j]$ where q_j is the position of joint j of the robot. The crucial point is that f_i appears quadratically in the exponential within the expectation. In other words, conditioned on ω_i, we can integrate out f_i exactly, just as if we were doing regression with a Gaussian likelihood.

Wenzel et al. [17] demonstrate that this binary classification method is up to two orders of magnitude faster than the state-of-the-art, while being competitive in terms of prediction performance, and is capable of working with datasets with tens of millions of datapoints.

Inference in exact GP models typically has a time complexity of $O(n^3)$, with n the number of data points in the model, and $O(n)$ and $O(n^2)$ for computing the mean and variance respectively. This makes exact GP models unfeasible when dealing with large datasets, although the development of libraries like GPyTorch [18] allow using GPU acceleration, which makes them faster to train and make predictions. Hensman et al. [19] propose an approach to reduce the complexity to $O(m^3)$, where m is the number of inducing points.

To train the GP model, we need to optimize both the kernel hyperparameters and the inducing points parameters, i.e. their locations, the variational covariance matrix, and a variational mean vector that controls the inducing points. These last two parameters will be optimized using natural gradient updates, since they are invariant to reparametrization of the variational family and provide second-order optimization updates. The kernel hyperparameters and the inducing points locations are optimized using the Adam optimizer. The loss function used for the optimizer is the variational evidence lower bound (ELBO), which is commonly used to optimize variational GPs. Equation (1) shows the variational ELBO loss function [19]:

$$\mathcal{L}_{\text{ELBO}} \approx \sum_{i=1}^{N} \mathbb{E}_{q(f_i)} \left[\log p(y_i \mid f_i)\right] - \text{KL}\left[q(\mathbf{u}) \| p(\mathbf{u})\right] \quad (1)$$

where N is the number of datapoints, $q(\mathbf{u})$ is the variational distribution for the inducing function values, $q(f_i)$ is the marginal of $p(f_i|\mathbf{u}, \mathbf{x}_i)q(\mathbf{u})$, KL is the Kullback-Leibler divergence [20], and $p(\mathbf{u})$ is the prior distribution for the inducing function values.

B. Kernel Selection

The kernel function $k(\mathbf{x_1}, \mathbf{x_2})$ compares a configuration $\mathbf{x_1}$ to $\mathbf{x_2}$ by mapping to some feature space and taking an inner product. This function should provide a large value for similar configurations (meaning they are strongly correlated) and a small value for dissimilar configurations. The kernel selected for this application is the RQ (rational quadratic) kernel, which can be seen as a scale mixture (an infinite sum) of RBF (radial basis function) kernels with different characteristic lengthscales. The RQ kernel is given by:

$$k_{\text{RQ}}(\mathbf{x_1}, \mathbf{x_2}) = \left(1 + \frac{1}{2\alpha}(\mathbf{x_1} - \mathbf{x_2})^\top \Theta^{-2}(\mathbf{x_1} - \mathbf{x_2})\right)^{-\alpha} \quad (2)$$

where Θ is a lengthscale parameter, and α is the rational quadratic relative weighting parameter. In this particular case, we train a different lengthscale for each degree of freedom of the robotic arm, since the movement in some of these degrees of freedom clearly has more influence in collisions than others.

A typical choice for GP models is to add a vector of automatic relevance determination (ARD) [21] hyperparameters Θ,

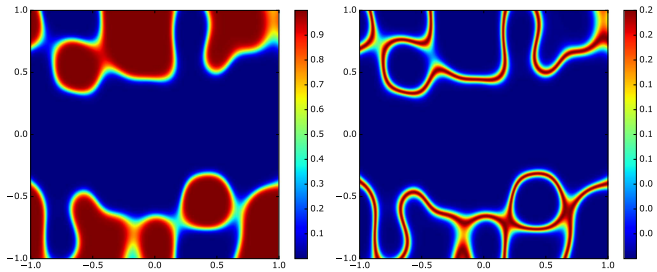
$$k_{\text{RQ}}(\mathbf{x_1}, \mathbf{x_2}) = \left(1 + \frac{1}{2\alpha}(\mathbf{x_1} - \mathbf{x_2})^\top diag(\Theta)^{-2}(\mathbf{x_1} - \mathbf{x_2})\right)^{-\alpha} \quad (3)$$

where $diag(\Theta)$ is a diagonal matrix with d entries Θ along the diagonal, with d the number of dimensions of the input data. Intuitively, if a particular Θ_i has a large value, the kernel becomes independent of the i-th input, effectively removing it automatically. Hence, irrelevant dimensions are discarded.

C. Dataset Generation and Model Creation

To create the dataset, a uniformly random set of N unlabeled configurations X are generated, and the true labels Y are provided by the FCL collision detection library [22]. To deal with the joint limits of the robot, we map the bounded d-dimensional joint space to a d-dimensional input space $[-1, 1]^d$. To map the joint values q_{joint} to the input space q_{input}, we use:

$$q_{input} = (2q_{joint} - q_u - q_l) \oslash (q_u - q_l) \quad (4)$$



(a) (b)

Fig. 1. Mean and variance obtained when performing collision checks on all available points for the first two DoF of the KUKA iiwa robot. (a) Mean values obtained when making predictions for all points. (b) Variance values obtained when making predictions for all points.

where q_l and q_u are vectors containing the lower and upper joint limits, and $\oslash$ performs element-wise division.

Once the dataset is ready, the PG likelihood and the GP model are initialized and their hyperparameters tuned as explained in Section III-A. Then, collision checks are performed with the updated model to evaluate its accuracy.

D. Making Predictions

To make predictions with the GP model for a test point x_*, we use:

$$p(f_* \mid y) = \int p(f_* \mid \mathbf{u}) q(\mathbf{u}) d\mathbf{u} = \mathcal{N}(f^* \mid \mu_*, \sigma_*^2) \quad (5)$$

where the prediction mean is $\mu_* = K_{*m}K_{mm}^{-1}\boldsymbol{\mu}$ and the variance $\sigma_*^2 = K_{**} + K_{*m}K_{mm}^{-1}(\sum K_{mm}^{-1} - I)K_{*m}$. $\mathbf{u}$ is the set of inducing points of size m, the matrix K_{*m} denotes the kernel matrix between the test point and $\mathbf{u}$, and K_{**} the kernel value of the test point. Fig. 1 provides an example of collision checks for the first two DoF of a KUKA iiwa manipulator, using 128 inducing points. The obtained mean values are in the interval [0,1], since they are normalized with a Bernoulli distribution. The points predicted as free points in space have mean values closer to zero, while the points predicted as obstacles have mean values closer to one. The variance reflects the obstacle borders, as the variance values are close to zero in the free space and inside of the obstacle, but reach their maximum values in the borders between C_{free} and C_{obs}.

When the algorithm decides based solely on the mean to determine if a point belongs to C_{free} or C_{obs}, taking all points with a mean equal or greater than 0.5 as obstacles and the points with a mean lower than 0.5 as free space, we risk obtaining false negatives, i.e. points classified as free space when they actually belong to an obstacle, which would lead to collisions when executing a motion with the robot. To prevent false negatives, we take advantage of the fact that GPs provide the variance σ in addition to the mean μ, and propose a new decision value to classify the points, γ, as

$$\gamma = \mu + \beta * \sigma \quad (6)$$

where β is the variable than modifies the influence of the variance σ in the decision.

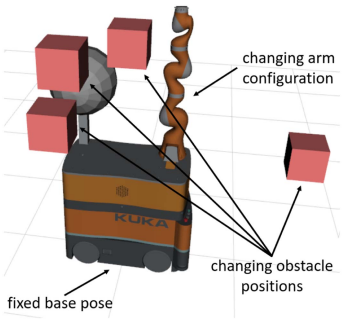


Fig. 2. RViz model of the robot used for the experiments. It consists of a KUKA KMR mobile base with a KUKA iiwa robot arm. Red boxes represent obstacles in the environment.

E. Updating Model in Dynamic Environments

To avoid retraining the whole model when the environment changes, the algorithm updates the model based on the inducing points locations. The already placed inducing points and a determined number of points A sampled uniformly around the workspace are checked and labeled with the FCL method. The algorithm updates the model with the information from the new dataset. This uniform distribution of A over the workspace allows to scan the environment for possible new obstacles and to grasp a general vision of all possible obstacle modifications or displacements.

As the inducing points are already well placed in the training phase, it would be a waste to delete them completely for updating purposes, since we assume slight changes in the environment. It would be a tradeoff between accuracy and updating time, as the updating time in case of deleting all inducing points would be the same as the initial training time.

IV. Experiments

In this section we evaluate CollisionGP against Fastron [16], a state-of-the-art machine learning method for collision checking, for 2, 4 and 7 actuated DoFs of a 7 DoF robot in 15 different environments with 4 obstacles. An example of test environment is shown in Fig. 2. We compare the number of inducing points, query times, training times and updating times for both static and dynamic environments with a dataset of $N = 10000$ training configurations and $A = 1000$. The Fastron and CollisionGP CPU tests run on an Intel Xeon E5-1620 v3 @3.5 GHz processor, and the CollisionGP GPU tests run on an Nvidia RTX 3060 graphics card.

A. Effect of ARD on the Kernel Lenghtscales

To study the effect of the application of ARD to this particular robot, we observe the lenghtscale values of the RQ kernel for all DoFs of the robot. Table I shows that the first two DoFs have the highest influence in the collisions, while the DoFs at the end of the robot have lower influence. A lower lengthscale translates into a higher significance in the GP model. The seventh DoF of the robot is the rotation of the end effector, and since there is no tool attached, this joint has no influence on the collisions.

TABLE I
Lengthscale Obtained for Each Dimension for Each Robot DoF

		Lengthscale of DoF						
		1	2	3	4	5	6	7
No. DoF	2	0.38	0.44	-	-	-	-	-
	3	0.23	0.30	15.76	-	-	-	-
	4	0.30	0.41	0.89	0.95	-	-	-
	5	0.28	0.38	0.67	0.89	6.43	-	-
	6	0.26	0.35	0.65	0.81	5.25	5.26	-
	7	0.26	0.35	0.60	0.85	7.66	7.55	7.61

B. Selection of β

To select β for this particular case, we evaluate the two, four and seven DoF models for different values β to see how the accuracy, TPR (True Positive Rate) and TNR (True Negative Rate) evolve. Fig. 3 shows the evolution of the TPR and TNR as β rises. Note that for values of $\beta > 0.5$ the TPR is above 0.95 while the TNR holds at above 0.85, effectively reducing the number false negatives to an acceptable threshold. In the following experiments we select $\beta = 0.5$ for all DoFs.

C. Batch Computations

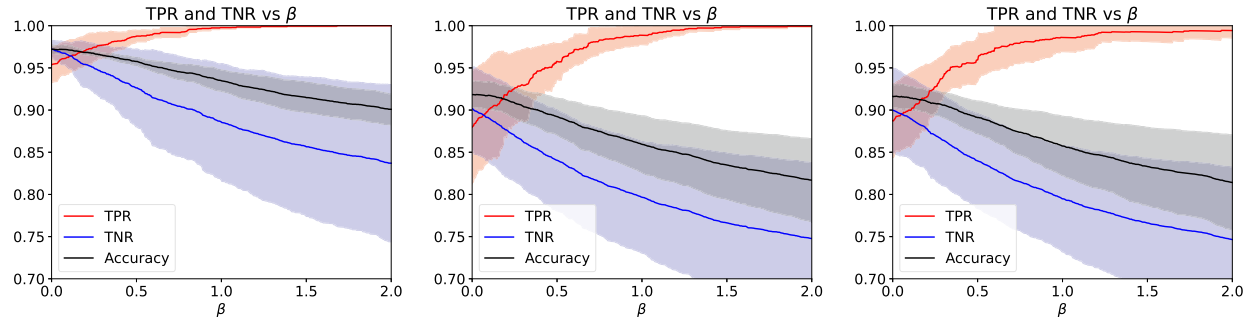
PyTorch allows to process the data in batches, which makes predictions faster since the software parallelizes the necessary calculations. Fig. 4 shows the necessary time to compute a certain number of collision checks at once for the FCL method and for CollisionGP using the CPU and the GPU with CUDA.

Fig. 4 shows the mean and standard deviation of the time needed to compute collisions for the GP models based on the number of configurations and inducing points. The CPU implementation shows that the prediction times grow exponentially as the number of configurations increase, while for the GPU implementation they increase linearly. Note that the GPU implementation is the fastest and most efficient method, and that the computation times for CollisionGP exhibit a narrow confidence interval, while the FCL method has a wider confidence interval.

D. Selecting the Number of Inducing Points

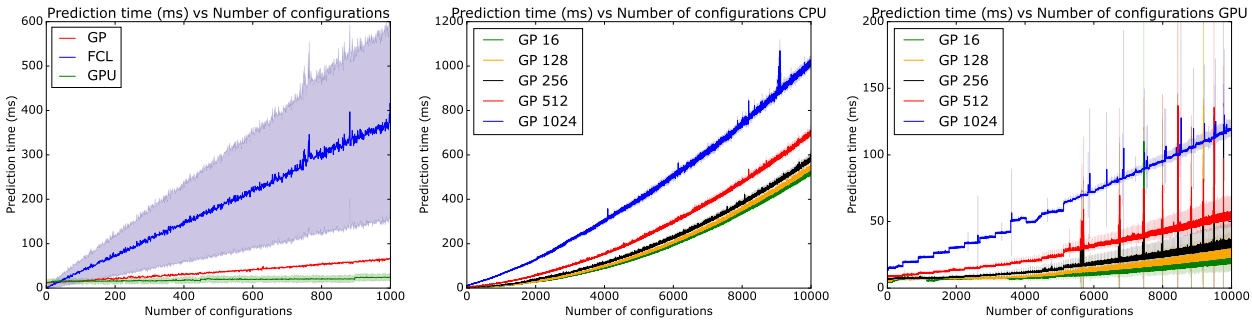
To select the number of inducing points for each DoF case, we train the models for different sets of inducing points to evaluate the model performance as the number of inducing points increases. Fig. 5 shows the accuracy of the models when actuating two, four and seven DoFs of the robot depending on the number of inducing points selected and using $\beta = 0.5$. The accuracy gradually increases as the number of inducing points increases until the value reaches a limit, which is the maximum possible accuracy that can be achieved with CollisionGP for the selected β.

To illustrate the performance of both Fastron and CollisionGP in the C-space, we provide an example with the first two DoFs of the robot manipulator, since a visualization in two dimensions allows for a qualitative comparison. Fig. 6 shows the obtained C-space for Fastron and CollisionGP, and the mean and variance maps obtained for the CollisionGP model with 128 inducing points when actuating the first two DoFs of the robot. The C-spaces obtained with both CollisionGP and Fastron are very similar to the ground truth, although CollisionGP uses less inducing points and also provides a measure of uncertainty, which results in the highlighting of obstacle borders. As most manipulators



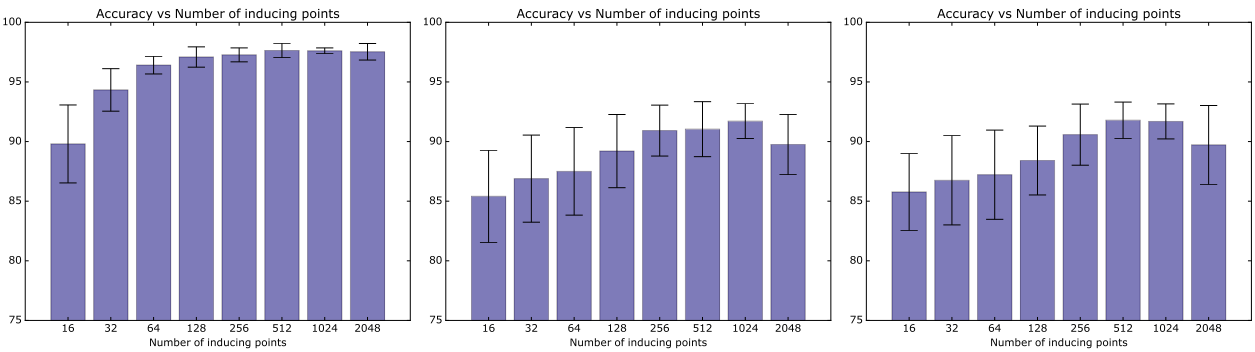
(a) Two DoF (b) Four DoF. (c) Seven DoF.

Fig. 3. TPR and TNR of CollisionGP for values of β for the seven DoF robot in five different scenarios with four static obstacles.



(a) FCL (b) CPU (c) GPU.

Fig. 4. Prediction time vs number of configurations for different numbers of inducing points for the seven DoF case. The step increases and the spikes in the GPU implementation could be related to the VRAM allocation of the GPU managed by the CUDA implementation, as they are produced in the same number of configurations for all inducing points.



(a) Two DoF (b) Four DoF (c) Seven DoF

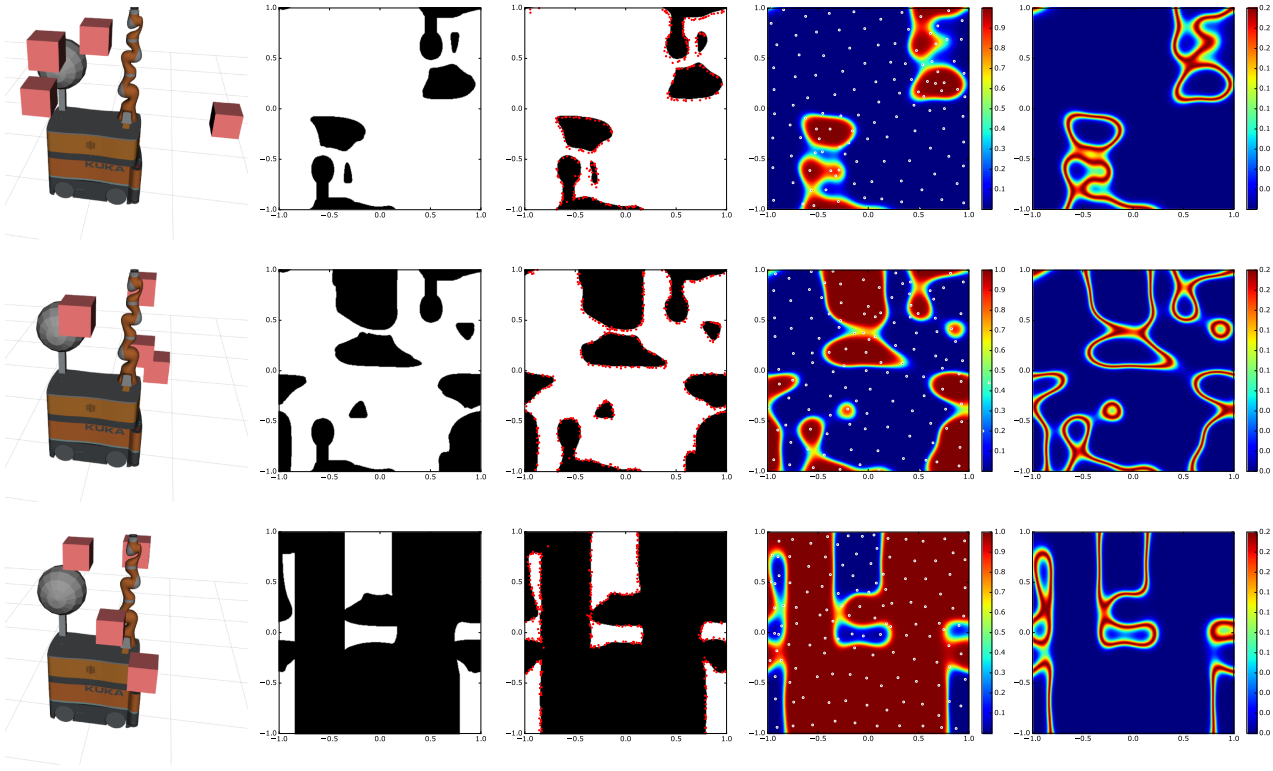
Fig. 5. Accuracy of the CollisionGP model for $\beta = 0.5$ with different numbers of inducing points for the seven DoF robot in fifteen different scenarios with four static obstacles.

today contain revolute joints, the C-space is smooth even for non-smooth obstacles and can be approximated well by the GP.

E. Accuracy, TPR and TNR

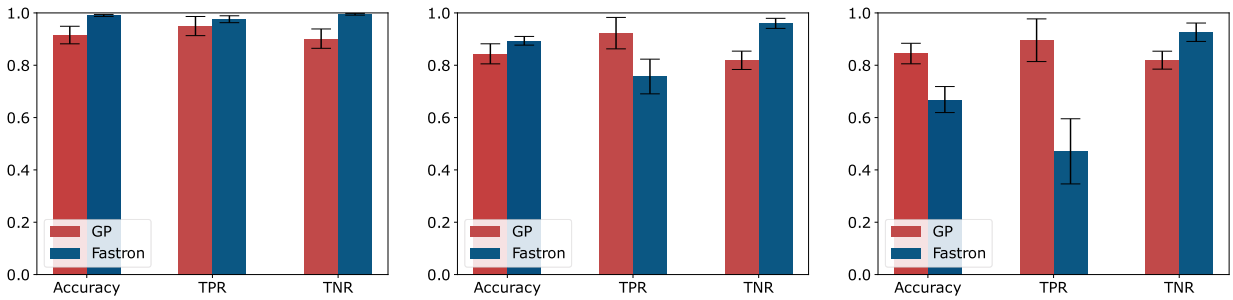
Fig. 7 shows accuracy, TPR and TNR comparisons between CollisionGP and Fastron in the fifteen simulated environments. Three of these environments are shown in Fig. 6. Both methods achieve similar results in the two DoF case, but CollisionGP mantains the accuracy, TPR and TNR for the four and seven DoF cases, while the values for Fastron decrease as the number of DoFs increases.

Table II shows query time, training time and number of inducing points comparisons for CollisionGP and Fastron. CollisionGP is the sparsest method and has the fastest query times of both methods for all DoFs, while Fastron achieves the fastest training times. The GPU implementation is significantly faster than the CPU implementation. Query times for CollisionGP are



(a) Scenario (b) FCL (c) Fastron (d) GP mean (e) GP variance

Fig. 6. Example environments and C-space approximations using Fastron and the proposed GP classification method. Column (d) shows the inducing points of the GP.



(a) Two DoF (b) Four DoF. (c) Seven DoF.

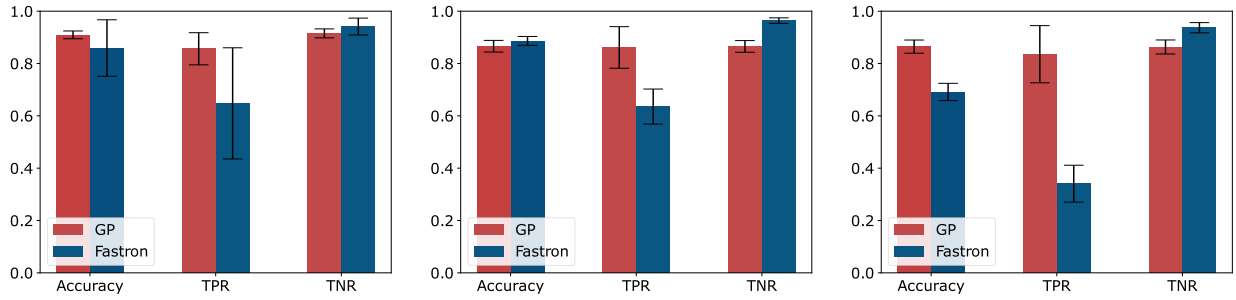
Fig. 7. Accuracy, TPR and TNR for Fastron and CollisionGP in fifteen different scenarios with four static obstacles.

considered when predicting 10000 configurations. Notice that Fastron is compiled in C++, while CollisionGP is executed in Python, which can affect the query and training times. We expect CollisionGP to be even faster, when programmed and compiled in C++ as well. In terms of computational cost, both methods have a complexity of $O(m^3)$. We evaluated the signifance of the results with the two-sample T-test for the seven DoF comparative measures. The resulting p-values are all below 10^{-7}, proving high statistical significance.

F. Accuracy, TPR and TNR in Dynamic Environments

Fig. 8 shows accuracy, TPR and TNR comparisons between CollisionGP and Fastron after updating the models in dynamic environments, where all boxes are moved up to 10 cm away from their initial positions after the initial training has finished and the first static GP model is created. We assume the obstacles stay in a fixed position while resampling the environment. As in the static environment test, CollisionGP achieves more consistent and higher scores than Fastron, for every evaluated number of DoFs. The scores for the seven DoF case are significantly lower when evaluating Fastron.

Table III shows query time, training time and number of inducing points comparisons for CollisionGP and Fastron after updating the models in a dynamic environment. Since we sample $m + A$ new configurations, the sampling times for Fastron are higher since m is higher. However, the model update is faster for the Fastron method. When taking into account the sampling



(a) Two DoF (b) Four DoF. (c) Seven DoF.

Fig. 8. Accuracy, TPR and TNR for Fastron and CollisionGP in fifteen different scenarios with four dynamic obstacles.

TABLE II
PERFORMANCE OF COLLISIONGP AND FASTRON IN FIFTEEN DIFFERENT SCENARIOS WITH FOUR STATIC OBSTACLES, WITH GROUND TRUTH COLLISION DETECTION COMPUTED USING FCL

	GP (GPU)	GP (CPU)	Fastron
Two DoF			
m	**128**	**128**	298.64 ± 49.6
Query time (μs)	$\mathbf{2.150 \pm 0.76}$	61.89 ± 1.01	76.62 ± 3.59
Training time (s)	0.446 ± 0.07	0.496 ± 0.04	$\mathbf{0.140 \pm 0.07}$
Accuracy	91.59 ± 3.57	91.59 ± 3.57	$\mathbf{98.97 \pm 0.35}$
Four DoF			
m	**256**	**256**	1575.60 ± 312.59
Query time (μs)	$\mathbf{2.803 \pm 0.65}$	65.23 ± 1.05	80.87 ± 1.66
Training time (s)	0.467 ± 0.07	0.727 ± 0.05	$\mathbf{0.112 \pm 0.02}$
Accuracy	84.36 ± 3.83	84.36 ± 3.83	$\mathbf{89.35 \pm 1.66}$
Seven DoF			
m	**512**	**512**	2917.80 ± 563.34
Query time (μs)	$\mathbf{4.724 \pm 0.32}$	75.91 ± 1.22	91.02 ± 4.44
Training time (s)	0.684 ± 0.07	1.988 ± 0.02	$\mathbf{0.254 \pm 0.05}$
Accuracy	$\mathbf{84.47 \pm 3.92}$	$\mathbf{84.47 \pm 3.92}$	66.88 ± 4.97

The bold entities indicate the best obtained values in each row.

TABLE III
PERFORMANCE OF COLLISIONGP VS. FASTRON IN FIFTEEN DIFFERENT SCENARIOS WITH FOUR DYNAMIC OBSTACLES

	GP (GPU)	GP (CPU)	Fastron
Two DoF			
m	**128**	**128**	145.1 ± 37.5
Sampling time (s)	$\mathbf{0.78 \pm 0.37}$	$\mathbf{0.78 \pm 0.37}$	1.02 ± 0.51
Updating time (s)	0.011 ± 0.008	0.18 ± 0.006	$\mathbf{0.003 \pm 0.003}$
Total (s)	$\mathbf{0.79 \pm 0.37}$	0.97 ± 0.37	1.02 ± 0.51
Accuracy	$\mathbf{90.95 \pm 1.48}$	$\mathbf{90.95 \pm 1.48}$	85.92 ± 10.79
Four DoF			
m	**256**	**256**	1316.2 ± 160.77
Sampling time (s)	$\mathbf{0.64 \pm 0.35}$	$\mathbf{0.64 \pm 0.35}$	2.32 ± 0.97
Updating time (s)	0.017 ± 0.02	0.23 ± 0.02	$\mathbf{0.009 \pm 0.003}$
Total (s)	$\mathbf{0.657 \pm 0.37}$	0.87 ± 0.37	2.33 ± 0.97
Accuracy	86.61 ± 2.22	86.61 ± 2.22	$\mathbf{88.65 \pm 1.68}$
Seven DoF			
m	**512**	**512**	2732.0 ± 319.64
Sampling time (s)	$\mathbf{0.43 \pm 0.22}$	$\mathbf{0.43 \pm 0.22}$	3.49 ± 1.78
Updating time (s)	0.033 ± 0.02	0.38 ± 0.01	$\mathbf{0.03 \pm 0.007}$
Total (s)	$\mathbf{0.46 \pm 0.24}$	0.82 ± 0.23	3.52 ± 1.79
Accuracy	$\mathbf{86.45 \pm 2.52}$	$\mathbf{86.45 \pm 2.52}$	69.13 ± 3.29

The bold entities indicate the best obtained values in each row.

time plus the updating time, CollisionGP is able to obtain an updated model up to 7 times faster in the case of the seven DoF experiment. We evaluated the signifance of the results with the two-sample T-test for the seven DoF comparative measures. The resulting p-values are all below 10^{-8}, proving high statistical significance.

V. DISCUSSION

The data obtained from the experiments suggests that GPs are able to provide an effective probabilistic approach to configuration space modeling and collision checking. The experiments show that the proposed method CollisionGP achieves greater accuracy, TPR and TNR values than the state-of-the-art algorithm Fastron, due to ARD and the ability to use the provided variance to determine which areas are more dangerous for possible collisions. CollisionGP is also more sparse, resulting in faster update times and faster prediction times, as shown in Tables II and III, where the query times for CollisionGP are up to 38 times faster than Fastron in the two DoF case, and the updating time for CollisionGP is 7 times faster than Fastron for seven DoF. However, training times are slower, as CollisionGP optimizes the hyperparameters of the kernel and the inducing points for a fixed number of iterations, while Fastron uses a custom training loop with a termination condition to update a matrix of weights. Another limitation of CollisionGP is the slow prediction time when making a collision query of a single configuration. We recommend to use batches and predict more than 10000 configurations at once, taking advantage of the array algebra of PyTorch. The prediction times can be further reduced by using a GPU to perform the necessary computations. We could also perform the calculations for the mean of the GP only, reducing the computational complexity from $O(m^3)$ to $O(m)$.

Nonetheless, we believe that the variance gives us valuable information about the configuration space that other methods cannot provide. FCL has a high standard deviation in Fig. 4 because the method employs a broad phase and a narrow phase, and configurations close to obstacles require more computation time. Based on the results of our previous publication [23], we expect a decrease in computation time by a factor of two to ten when replacing FCL with CollisionGP in a realistic motion planning problem: One query requires 5000 to 10000 collision checks. If the planning algorithm queries collision checks in large batches, computation time will decrease significantly, as shown for example in [24].

When directly comparing Fastron and CollisionGP in Fig. 6, the obtained C-spaces show that Fastron accurately models the sharp corners and edges while CollisionGP does not. However, we believe that CollisionGP achieves a good enough representation while being more sparse. Furthermore, it provides a probabilistic approach to model the C-space, with the variance being higher around the obstacle borders. High variance provides crucial information for the application and raises

awareness to regions where the computed motion should be investigated with a convential collision checker. CollisionGP could be implemented with deep kernel methods trained in different scenarios to improve the accuracy. However, while deep kernel methods have several advantages over standard Gaussian processes, such as improved model flexibility and ability to handle larger datasets, they also have some disadvantages compared to standard Gaussian processes, such as computational complexity, interpretability, overfitting, data requirements and hyperparameter and architecture tuning.

While learning-based algorithms cannot replace conventional collision checking algorithms in all applications, we suggest to use the technology in combination with collaborative robots. As the manipulators guarantee safe interaction, collision checking and motion planning algorithms need to minimize the chance of a collision and not necessarily eliminate it completely. This allows the robot to leverage the computational advantage of CollisionGP, while avoiding collisions in most motions. However, one disadvantage of CollisionGP compared to conventional collision checking algorithms is that it does not provide information about which link is colliding.

The TNR, TPR and accuracy for Fastron decreases as the number of DoFs increases (Fig. 7). This can be due to the fact that CollisionGP optimizes a separate kernel lengthscale for each DoF, while Fastron considers all DoFs to have the same weight and sets a fixed value of α. One advantage of keeping a fixed value of α is that the training times are shorter.

VI. Conclusion

In this work, we proposed and validated the CollisionGP algorithm, which models the configuration space for collision detection. CollisionGP allows collision detections to be performed an order of magnitude faster than traditional collision checking algorithms such as FCL when using batches. We also showed that CollisionGP is more sparse than Fastron, a state-of-the-art machine learning algorithm for collision checking, achieving faster predictions and model updating times. Furthermore, to the best of our knowledege CollisionGP is the first collision checking algorithm that uses GPs, providing not only a mean estimate, but also the predictive variance, which gives a measure of uncertainty about the prediction and is useful to eliminate false negatives. This is specially important for critical applications where the final path could be checked with FCL to ensure that no collisions are produced.

Using GPs with PG auxiliary variables for probabilistic collision checking opens some interesting lines of future work in robotics. One possible future application of CollisionGP is for robots with many DoF, such as humanoid robots. The large C-spaces require exploration of many states and thus fast primitive operations, such as collision checking. Another future work is the implementation of CollisionGP with different batch path planners in real environments to test its efficiency. The code for CollisionGP can be found in https://github.com/jmunozmendi/CollisionGP.

References

[1] H. Choset, *Principles of Robot Motion: Theory, Algorithms, and Implementations*, Cambridge, MA, USA: MIT Press, 2005.

[2] M. Elbanhawi and M. Simic, "Sampling-based robot motion planning: A review," *IEEE Access*, vol. 2, pp. 56–77, 2014.

[3] N. G. Polson, J. G. Scott, and J. Windle, "Bayesian inference for logistic models using Pólya–Gamma latent variables," *J. Amer. Stat. Assoc.*, vol. 108, no. 504, pp. 1339–1349, 2013.

[4] J. Dong, M. Mukadam, F. Dellaert, and B. Boots, "Motion planning as probabilistic inference using Gaussian processes and factor graphs," *Robot.: Sci. Syst.*, vol. 12, no. 4, 2016.

[5] M. Mukadam, X. Yan, and B. Boots, "Gaussian process motion planning," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2016, pp. 9–15.

[6] G. Chou, H. Wang, and D. Berenson, "Gaussian process constraint learning for scalable chance-constrained motion planning from demonstrations," *IEEE Robot. Automat. Lett.*, vol. 7, no. 2, pp. 3827–3834, Apr. 2022.

[7] J. Pan and D. Manocha, "Efficient configuration space construction and optimization for motion planning," *Engineering*, vol. 1, pp. 46–57, 2015.

[8] O. Arslan and P. Tsiotras, "Machine learning guided exploration for sampling-based motion planning algorithms," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2015, pp. 2646–2652.

[9] M. Danielczuk, A. Mousavian, C. Eppner, and D. Fox, "Object rearrangement using learned implicit collision functions," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2021, pp. 6010–6017.

[10] J. Chase Kew, B. Ichter, M. Bandari, T. W. E. Lee, and A. Faust, "Neural collision clearance estimator for batched motion planning," in *Proc. Int. Workshop Algorithmic Found. Robot.*, 2020, pp. 73–89.

[11] A. H. Qureshi, A. Simeonov, M. J. Bency, and M. C. Yip, "Motion planning networks," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2019, pp. 2118–2124.

[12] B. Ichter, E. Schmerling, T. W. E. Lee, and A. Faust, "Learned critical probabilistic roadmaps for robotic motion planning," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2020, pp. 9535–9541.

[13] L. C. Freeman, "A set of measures of centrality based on betweenness," *Sociometry*, vol. 40, pp. 35–41, 1977.

[14] J. Pan and D. Manocha, "Fast probabilistic collision checking for sampling-based motion planning using locality-sensitive hashing," *Int. J. Robot. Res.*, vol. 35, no. 12, pp. 1477–1496, 2016.

[15] Y. Han, W. Zhao, J. Pan, and Y.-J. Liu, "Configuration space decomposition for learning-based collision checking in high-DOF robots," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2020, pp. 5678–5684.

[16] N. Das and M. Yip, "Learning-based proxy collision detection for robot motion planning applications," *IEEE Trans. Robot.*, vol. 36, no. 4, pp. 1096–1114, Aug. 2020.

[17] F. Wenzel, T. Galy-Fajou, C. Donner, M. Kloft, and M. Opper, "Efficient Gaussian process classification using Pòlya-Gamma data augmentation," in *Proc. AAAI Conf. Artif. Intell.*, 2019, vol. 33, pp. 5417–5424.

[18] J. Gardner, G. Pleiss, K. Weinberger, D. Bindel, and A. Wilson, "GPyTorch: Blackbox matrix-matrix Gaussian process inference with GPU acceleration," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, 2018.

[19] J. Hensman, A. Matthews, and Z. Ghahramani, "Scalable variational Gaussian process classification," in *Proc. Artif. Intell. Statist.*, 2015, pp. 351–360.

[20] S. Kullback and R. A. Leibler, "On information and sufficiency," *Ann. Math. Statist.*, vol. 22, no. 1, pp. 79–86, 1951.

[21] D. J. MacKay, "Probable networks and plausible predictions-a review of practical Bayesian methods for supervised neural networks," *Netw.: Comput. Neural Syst.*, vol. 6, no. 3, 1995, Art. no. 469.

[22] J. Pan, S. Chitta, and D. Manocha, "FCL: A general purpose library for collision and proximity queries," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2012, pp. 3859–3866.

[23] P. Lehner, M. A. Roa, and A. Albu-Schäffer, "Kinematic transfer learning of sampling distributions for manipulator motion planning," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2022, pp. 7211–7217.

[24] J. Bialkowski, S. Karaman, and E. Frazzoli, "Massively parallelizing the RRT and the RRT," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2011, pp. 3513–3518.