Fastron: An Online Learning-Based Model and Active Learning Strategy for Proxy Collision Detection

Nikhil Das, Naman Gupta, Michael Yip
Electrical and Computer Engineering
University of California, San Diego

Abstract: We introduce the Fastron, a configuration space (C-space) model to be used as a proxy to kinematic-based collision detection. The Fastron allows iterative updates to account for a changing environment through a combination of a novel formulation of the kernel perceptron learning algorithm and an active learning strategy. Our simulations on a 7 degree-of-freedom arm indicate that proxy collision checks may be performed at least 2 times faster than an efficient polyhedral collision checker and at least 8 times faster than an efficient high-precision collision checker. The Fastron model provides conservative collision status predictions by padding C-space obstacles, and proxy collision checking time does not scale poorly as the number of workspace obstacles increases. All results were achieved without GPU acceleration or parallel computing.

Keywords: configuration space, collision detection

1 Introduction

Configuration space (C-space) is a space that completely defines every kinematic configuration of the robot [1]. Robot configurations that are not in collision with workspace obstacles comprise the C_{free} regions of C-space, and the C_{obs} regions denote configurations in which the robot is in collision with a workspace obstacle. Checking for collisions is often a computational burden for robots working in environments with obstacles, but is a necessity for processes in which the robot must interact with or navigate through its environment, such as with Rapidly-Exploring Random Trees (RRTs) [2], a sampling-based motion planning algorithm.

A difficulty in working with C-space is that obstacle geometries generally do not trivially map from the workspace to C-space [1, 3]. Sampling-based motion planners instead spend a large majority of their computation time on performing collision checks [4] to infer C-space obstacles. In the case of workspaces with moving obstacles, C_{obs} changes non-trivially, which makes maintenance of an updated map in C-space for collision detection a bottleneck in performance. Specialized hardware such as FPGAs [5] accelerates the collision detection step, but algorithmic solutions may reduce the overall computation, which in turn may further improve hardware-based solutions.

1.1 Contributions

Realizing the high cost involved in kinematic-based collision detections (KCDs), we seek to decrease the computational cost of collision checking by learning a proxy collision detector that efficiently learns and maintains C-space representations that change over time. In this paper, we present the Fastron algorithm, a fast technique to generate and update an approximate C-space representation for proxy collision checking.

The purpose of these efforts is to reduce the computation required for collision checking for processes that suffer from a large number of KCDs so that more resources may be dedicated toward other computationally-intensive tasks, including further sampling for fine motion planning or model updates for reinforcement learning algorithms. Integrating the Fastron into motion planning algorithms is an obvious utilization, yet other highly-iterative applications that rely on collision detection

1st Conference on Robot Learning (CoRL 2017), Mountain View, United States.

could benefit from the Fastron, such as reward evaluation for reinforcement learning for simulated robot manipulation tasks and approximate object interactions in physics or CAD simulations.

A learning-based approach to modeling C-space is advantageous because a lightweight model and intelligent information gathering may be used in lieu of dense representation and sampling of a typically large-dimensional space. The Fastron is based on a modification of the kernel perceptron learning algorithm and uses a novel active learning strategy to reduce the total number of KCDs in favor of faster, proxy collision checks. Active learning algorithms select which samples to query so as to potentially reduce the number of queries to an oracle (who provides true labels at a higher cost) to perform during training or model updates [6, 7]. In the case of C-space estimation, active learning is useful when selecting on which samples accurate yet costly KCDs should be performed. The Fastron algorithm updates iteratively using periodic snapshots of obstacles' shapes and locations in the reachable workspace. Prior knowledge of all potential obstacle geometry models and trajectories is not required.

The novel contributions of this paper are:

1. a simple yet efficient method to learn and represent C-space obstacles using a kernel perceptron decision boundary
2. a modified kernel perceptron that allows both addition and removal of support points, and
3. an active learning strategy to efficiently search for collision status changes in a changing environment, where there is limited computation time between control cycles.

1.2 Related Work

As with our Fastron method, previous works have utilized machine learning-based models to approximate C_{free} and C_{obs} based on sampled configurations and use active learning strategies to guide the search for new information to update or refine the models. The following are contributions toward representing C-space environments with learning-based models.

Pan et al. [6] use an incremental support vector machine (SVM) to learn an accurate representation of C-space between two objects in an offline step. Their active learning strategy exploits the structure of the SVM-based hyperplane to add new points in order to construct a near-perfect representation of C-space obstacles. A new classifier must be precomputed for each pair of objects, thereby increasing the training time and proxy collision detection time. Additionally, since the models are learned in an offline stage, the geometry models of all workspace obstacles must be known a priori, which is not always a luxury. Online implementations would fare poorly when new obstacles are introduced into the workspace since this would require learning a completely new SVM model, which is unsuitable for real-time applications.

Huh and Lee [8] use Gaussian mixture models (GMM) to represent C_{free} and C_{obs}, from which proxy collision detection is performed by assigning a query configuration the same label as the closest Gaussian. Their iterative GMM technique allows the model to update when workspace obstacles move to intersect a planned trajectory. A limitation of the GMM approach is the model may not fit irregularly-shaped C_{obs} regions effectively as GMMs use a limited number of Gaussians. Addition-

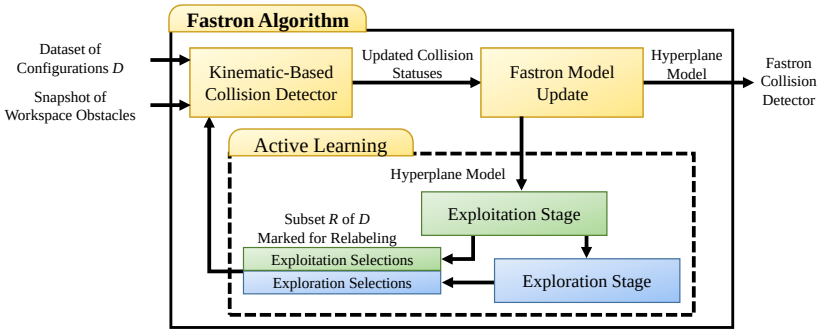


Figure 1: Pipeline of Fastron algorithm for generating and updating the C-space model used for fast collision checking.

ally, the underlying generative models are updated to fit new information, which consequently does not guarantee the resulting discriminative classifier immediately fits the new information.

Burns and Brock [9] use a k-nearest neighbors (k-NN) model for C-space, which is not intended to be used in the case of moving workspace obstacles. Pan and Manocha [10] also use a k-NN model, accelerated by locality-sensitive hashing. Their method significantly reduces the time required for collision checking for sampling-based motion planners by building a database to use for k-NN queries. Though not implemented in a changing environment, they propose their method can extend to a changing environment by gridding the workspace and only performing collision checks on configurations associated with dynamic cells.

2 Methods

In this section, we provide a detailed description of the Fastron algorithm. The steps of the algorithm are summarized in the block diagram in Fig. 1. The algorithm cycles through two steps: updating the collision boundary model (2.1) and active learning to search for collision status changes (2.2).

2.1 Modeling C-Space Using Perceptron

We require (and the Fastron offers) a model that

1. is fast to train,
2. is fast in classifying query configurations,
3. adequately fits training data,
4. attempts to reduce mistakes where C_{obs} configurations are classified as C_{free},
5. has an easily exploitable structure to facilitate the search for collision status changes, and
6. can efficiently account for collision status changes without retraining from scratch.

The batch kernel perceptron algorithm, which identifies a set of support points defining a separating hyperplane between two classes, satisfies the first three requirements and thus serves as the base model for the algorithm. We modify the kernel perceptron to satisfy the remaining requirements. This section describes the original batch kernel perceptron algorithm and our modifications. Pseudocode for the modified perceptron is shown in Algorithm 1.

Algorithm 1: Fastron Model Updating

Input: Weight vector α; hypothesis vector F; Gram matrix G for a dataset $\mathcal{D}$; true labels y for $\mathcal{D}$; conditional bias parameter r^+; maximum number of updates $maxUpdates$

Output: Updated weight vector α; updated hypothesis vector F

for $iter = 1$ *to* $maxUpdates$ **do**
  `// Remove redundant support points`
  while $\exists$ *i s.t.* $y_i(F_i - \alpha_i) > 0$ *and* $\alpha_i \neq 0$ **do**
    $j \leftarrow \text{argmax}_i \, y_i(F_i - \alpha_i)$
    $F_i \leftarrow F_i - G_{ij}\alpha_j \, \forall i$
    $\alpha_j \leftarrow 0$
  `// Margin-based prioritization`
  if $y_i F_i > 0 \, \forall i$ **then**
    return α, F
  else
    $j \leftarrow \text{argmin}_i \, y_i F_i$
  `// One-step weight correction with conditional biasing`
  if $y_j > 0$ **then**
    $\Delta\alpha \leftarrow r^+ y_j - F_j$
  else
    $\Delta\alpha \leftarrow y_j - F_j$
  $\alpha_j \leftarrow \alpha_j + \Delta\alpha$
  $F_i \leftarrow F_i + G_{ij}\Delta\alpha \, \forall i$
return α, F

2.1.1 Training and Classification with Original Kernel Perceptron

The original batch kernel perceptron algorithm trains a model that may be used to classify a query point into one of two classes. During training, the model updates when it encounters a training point that it would misclassify. Given a training dataset $\mathcal{D}$ of N labeled samples, the kernel perceptron algorithm learns a hypothesis $f(x)$, which has the form $\sum_i \alpha_i K(x_i, x)$, where $\alpha \in \mathbb{R}^N$ is a sparse weight vector, $K(\cdot, \cdot)$ is the Gaussian kernel function, and x_i is a sample in $\mathcal{D}$ with a known label $y_i \in [-1, +1]$. The Gaussian kernel is defined as $K(x_i, x_j) = exp(-\gamma \|x_i - x_j\|^2)$, where γ is a parameter specifying the narrowness of the Gaussian. The goal of the perceptron algorithm is to define α_i such that the margin $y_i f(x_i)$ for each training point x_i is positive. The original algorithm learns α_i by shuffling $\mathcal{D}$ and computing $y_i f(x_i)$ for each x_i. Whenever $y_i f(x_i) \leq 0$, y_i is added to α_i. This shuffle and update procedure is repeated until all training points have a positive margin or an epoch limit has been reached.

The hypothesis at each sample can be written in vector form as $F = G\alpha$, where the i^{th} element of F is $f(x_i)$ and G is the kernel Gram matrix for the N datapoints. To avoid redundant matrix-vector multiplications, we can store F and add (or subtract) the i^{th} column of G whenever we increment (or decrement) α_i. The update rule for the original kernel perceptron may thus be written as

$$\alpha_i \leftarrow \alpha_i + y_i \tag{1}$$

$$F \leftarrow F + y_i G_{*i} \tag{2}$$

where G_{*i} is the i^{th} column of G.

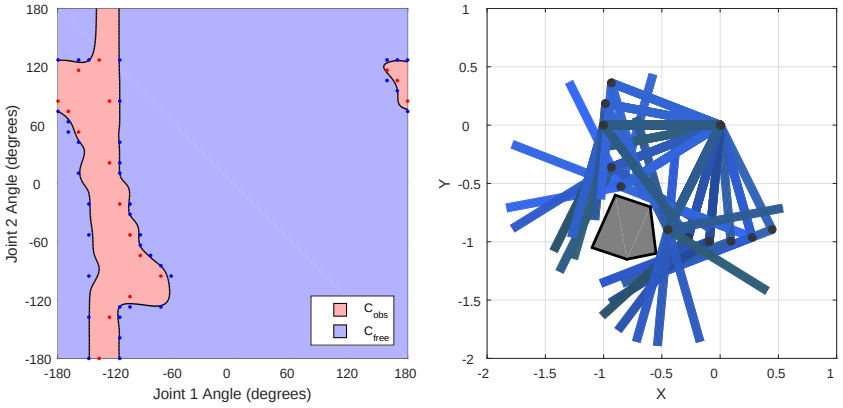
The support set S is the set of points in $\mathcal{D}$ with a nonzero weight in α. The support points that comprise S may be used to classify a query configuration x as $\hat{y}(x) = sgn\left(\sum_{i:x_i \in S} \alpha_i K(x_i, x)\right)$. We may use this classification as a proxy collision check where $\hat{y} = \pm 1$ represents an in-collision or a collision-free status, respectively.

2.1.2 One-Step Weight Correction and Conditional Biasing

The original kernel perceptron algorithm increases the weight of a misclassified point x_i by y_i, but x_i will still be incorrectly classified if the magnitude of the margin $\|y_i f(x_i)\|$ prior to update is greater than 1. The appropriate value to assign to weight α_i to ensure x_i is correctly classified may be easily realized based on the requirement that the resulting margin must be positive. It is evident that for x_i to be classified correctly, α_i must equal $ry_i - \sum_{j \neq i} \alpha_j K(x_j, x_i)$, where $r > 0$. We can avoid computing the summation in the second term by noting the change in α_i after the update is $\Delta\alpha_i = ry_i - \sum_j \alpha_j K(x_j, x_i) = ry_i - f(x_i)$. Thus, the update rule for our modified perceptron is:

$$\alpha_i \leftarrow \alpha_i + \Delta\alpha_i \tag{3}$$

$$F \leftarrow F + \Delta\alpha_i G_{*i} \tag{4}$$



(a) (b)

Figure 2: (a) Decision boundary (black curve) and support points (red and blue points) learned by our modified kernel perceptron. (b) Workspace representations of C_{free} support points from our modified kernel perceptron (blue 2 DOF manipulators) and a workspace obstacle (gray polygon).

Algorithm 2: Fastron Active Learning Strategy

Input: KCD allowance A; exploitation proportion p; support set S; dataset $\mathcal{D}$; Gram matrix G; maximum number of nearest non-support points k_{NS}

Output: Set of points $R \subset \mathcal{D}$ to be relabeled with oracle collision detection function

```
// Exploitation Stage
if |S| ≤ A then
    R ← S
    for k = 1 to k_NS do
        if |R| < pA then
            R ← R ∪ knnsearch(D\S, S, k)
else
    R ← sample(S, A)
// Exploration Stage
R ← R ∪ sample(D\R, A − |R|)
return R
```

The advantage of this modification is the misclassified point x_i is guaranteed to be modeled correctly after the update, which generally reduces the total number of update iterations. To increase the safety of the hyperplane, we conditionally set r depending on the label of the support point we are adding to S. More explicitly, we define a conditional bias parameter $r^+ > 1$, and we set $r = r^+$ when $y_i > 0$ and $r = 1$ when $y_i < 0$. When r^+ is greater than 1, C_{obs} configurations have a larger influence on the update to the hyperplane compared to C_{free} configurations which slightly pads the C-space obstacles, thereby potentially reducing the false negatives (misclassification of a C_{obs} configuration as C_{free}) when compared to the original perceptron algorithm.

2.1.3 Margin-Based Prioritization

The magnitude of a point's margin indicates how confidently the point is assigned to its predicted label. By updating the weight associated with the most negative margin, the most erroneous point is forced to be correctly classified using the one-step weight adjustment described above. Thus, rather than shuffling the data and running through $\mathcal{D}$ in a random order, we choose to update α_i where $i = \text{argmin}_j \, y_j f(x_j)$. The advantage of margin-based prioritization is that the support points end up closer to the decision boundary, granting the ability to exploit the structure of the model when searching for collision status changes near the boundary.

2.1.4 Redundant Support Point Removal

A support point should be removed from S (but remain in $\mathcal{D}$) when it is redundant. Redundant support points are those that will be correctly classified even if their corresponding α value is 0, i.e., $\{x_i | x_i \in S \wedge y_i(F_i - \alpha_i) > 0\}$. Support points are removed in decreasing order of positive resultant margin by setting the weight to 0 and updating F accordingly. The removal step is complete once $y_i(f(x_i) - \alpha_i) < 0 \, \forall i$, i.e., removing another support point will cause it to be misclassified.

Redundant support point removal is useful when the collision status of the points in $\mathcal{D}$ change in response to moving obstacles, causing the updated decision boundary to shift away from previous support points. Redundant support point removal ensures that the support points are as close as possible to the hyperplane. With the original perceptron algorithm which does not include redundant support point removal, it is possible that eventually $S = \mathcal{D}$, which slows classification by forfeiting the sparsity of the model.

2.2 Active Learning for Efficient Relabeling

In response to a changing environment, the collision statuses of the points in $\mathcal{D}$ must be updated before updating the hyperplane model. To know with absolute certainty which points have switched labels, KCD must be performed on each point in $\mathcal{D}$, which is clearly a time-consuming and potentially unnecessary process. Instead, the Fastron selects a subset R of $\mathcal{D}$ to relabel, where the size of R is set by a user-defined allowance A for the total number of KCDs to perform per model update.

Points are added to R using a two-stage active learning strategy. A common active learning strategy is to balance exploitation of the current model and exploration of the entire space, which is the

technique the SVM C-space approach uses [6]. The Fastron adopts a similar active learning strategy, but in the interest of efficient model updating, the Fastron relabels preexisting samples in $\mathcal{D}$ rather than generating entirely new samples. This allows the Fastron to take advantage of distances stored in the Gram matrix rather than reevaluating the kernel values.

Our strategy selects at least pA points in the exploitation stage, where p is a user-defined proportion of the allowance dedicated for exploitation. The remainder of the allowance is exhausted in the exploration stage. The following subsections describe how the two stages create the subset R. Pseudocode is provided in Algorithm 2, and an example subset R is shown in Fig. 3.

2.2.1 Exploitation Stage

Assuming that movements of the workspace obstacles cause small perturbations of the corresponding C-space obstacles, the Fastron first checks for status changes near the boundary of the C-space obstacles. This is accomplished by exploiting the structure of the perceptron model, which typically has its support points near the decision boundary when using our modified perceptron. At the beginning of each model update, R is initialized to the empty set. All current support points are then included in R. In the case that including all support points will exceed the allowance A, A support points are randomly chosen to be included in R. After adding support points to R, if $|R|$ is less than pA, each support point's i^{th}-nearest non-support point is iteratively included until either the resulting $|R|$ is greater than or equal to pA or $k_{NS}|S|$ non-support points have been included in R, where k_{NS} is a user-defined amount.

Distance information between points is conveniently available in Gram matrix G, and since the values of G do not change throughout the lifetime of the Fastron algorithm, costly conventional k-NN searches or more efficient approximations are not necessary. Line 5 in Algorithm 2 assumes the k-NN search utilizes G.

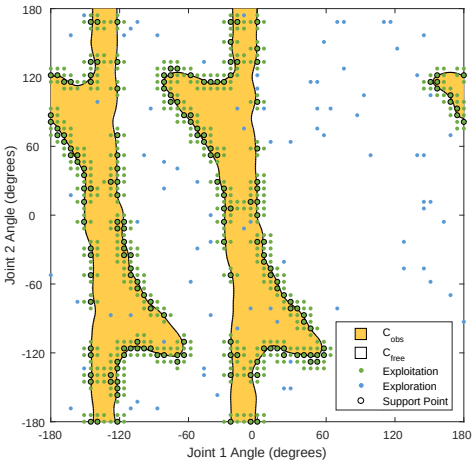


Figure 3: Example subset R selected by the active learning strategy for relabeling via KCD.

2.2.2 Exploration Stage

If the collision check allowance is not yet exhausted, the remainder of the allowance is utilized by randomly selecting $A - |R|$ configurations from $\mathcal{D} \setminus R$. The purpose of this random exploration step is to search for new or drastically different C-space obstacles, such as when a new object enters the reachable workspace or an existing object moves quickly.

3 Experimental Results

3.1 Experiments on 2 DOF Manipulator

We perform preliminary experiments on a 2 DOF manipulator to easily visualize both the workspace and C-space. We create random convex polygonal workspace obstacles, and use the Gilbert-Johnson-Keerthi (GJK) algorithm [11] for KCDs. We perform all 2 DOF simulations in MATLAB without the use of GPU acceleration or parallel computing to demonstrate its native speed.

We compare the collision detection time of Fastron-based collision detections (FCDs) and KCDs under increasingly difficult conditions (increasing number of workspace obstacles). We use $N = 625$, kernel width $\gamma = 10$, and conditional bias parameter $r^+ = 100$ for our Fastron model parameters, where γ and r^+ were selected via cross-validation. In the interest of generating a safe model, recall (true positive rate, or percentage of C_{obs} configurations correctly classified) is our primary metric for performance. High values of recall indicate that the model rarely considers C_{obs} configurations to be in C_{free}. Table 1 demonstrates the performance of FCD for various numbers of workspace

	Number of Obstacles				
	1	2	3	4	5
FCD Recall (%)	98.3	98.3	98.5	98.9	98.9
FCD FPR (%)	3.6	6.7	11.5	13.9	16.0
FCD Time (μs)	33.8	37.9	39.2	39.6	40.5
Ratio of KCD to FCD Time	**4.9**	**7.5**	**9.4**	**11.1**	**12.0**

Table 1: Recall, false positive rate, and collision check time of FCDs for 2 DOF manipulator with various number of obstacles. KCD timings scale poorly with obstacle number, while FCDs do not.

	$A = 0.1N$			$A = 0.3N$			$A = 0.5N$		
	Recall	FPR	Time	Recall	FPR	Time	Recall	FPR	Time
$N = 100$	75.0	6.5	1.5	84.2	7.5	3.4	85.5	7.8	4.7
$N = 400$	94.6	2.7	5.6	95.4	3.3	12.6	93.9	2.9	16.6
$N = 625$	91.0	2.0	8.3	95.4	2.2	18.8	95.7	2.0	26.2
$N = 900$	94.5	1.6	13.2	96.5	1.6	26.4	93.8	1.4	36.3
$N = 1225$	95.6	1.3	17.0	95.9	1.4	37.5	95.6	1.0	48.4

Table 2: Recall (%), false positive rate (%), and model update time (ms) for various dataset sizes N and exploitation stage proportions p for 2 DOF manipulator in a changing environment.

obstacles. Recall remains high (over 98%) as the number of obstacles increases. Table 1 also includes false positive rate (FPR) to demonstrate the effect of padding due to conditional biasing in a more crowded workspace. FPR increases along with the number of obstacles because the Fastron has a bias toward labeling configurations as C_{obs} in regions of uncertainty, namely near the decision boundary. The speed improvement of FCD over KCD drastically increases as the number of obstacles increases, showing FCDs are more resilient to obstacle count than KCDs.

We evaluate the performance of the Fastron in an environment with a moving randomly-generated polygon under various dataset sizes N and relabeling allowances A, with $\gamma = 10$, exploitation proportion $p = 0.8$, and a maximum nearest non-support point number $k_{NS} = 4$. We tabulate the average recall, FPR, and update time (model updating and active learning) over 10 second trials in Table 2. Compared to the static case shown in Table 1, recall is lower in the moving obstacle case possibly because all collision status changes may not have been detected. However, recall is still large (over 90%) for N larger than or equal to 400. Update time worsens as p increases because more KCDs are required. FPR decreases for increasing N because when there are more points distributed in C-space, there is a decreased requirement for the Fastron to be conservative by padding C-space obstacles in regions of uncertainty.

3.2 Experiments on 7 DOF Manipulator

We apply the Fastron algorithm to a simulated 7 DOF PR2 arm in a C++ ROS environment with shape primitives as workspace obstacles. KCD is performed using either the Flexible Collision Library (FCL) [12] collision checker or GJK in the Bullet physics library. In the FCL cases, the actual PR2 arm mesh is used. While FCL may be used for high-precision collision checking, it is a popular collision checking framework and starts with a broad phase collision check which makes many collision checks fast. In the GJK cases, we simplify the arm as a set of oriented bounding

		Number of Obstacles		
		1	2	3
FCL	FCD Recall (%)	92.8	95.3	98.1
	FCD FPR (%)	14.3	22.9	30.9
	FCD Time (μs)	4.1	4.0	4.2
	Ratio of KCD to FCD Time	**8.1**	**9.4**	**10.3**
GJK	FCD Recall (%)	91.6	94.0	96.0
	FCD FPR (%)	7.2	11.1	32.6
	FCD Time (μs)	3.6	4.0	4.6
	Ratio of KCD to FCD Time	**2.0**	**2.7**	**2.9**

Table 3: Recall, false positive rate, and collision check time of FCDs for 7 DOF manipulator with various number of obstacles. KCD timings scale poorly with obstacle number, while FCDs do not.

		$A = 0.1N$			$A = 0.3N$			$A = 0.5N$		
		Recall	FPR	Time	Recall	FPR	Time	Recall	FPR	Time
FCL	$N = 1000$	98.9	36.0	2.7	98.9	38.2	2.9	98.8	37.5	3.1
	$N = 4000$	96.1	18.4	29.1	95.7	17.3	31.7	94.7	15.6	32.8
	$N = 8000$	90.2	8.5	116.5	90.2	8.2	130.6	87.8	6.7	138.1
GJK	$N = 1000$	95.2	16.7	2.2	94.5	14.9	2.3	94.1	14.5	2.4
	$N = 4000$	93.4	9.5	29.3	92.0	7.6	30.8	91.0	7.0	31.8
	$N = 8000$	93.1	7.7	123.3	91.7	5.9	131.5	90.6	5.1	138.4

Table 4: Recall (%), false positive rate (%), and model update time (ms) for various dataset sizes N and exploitation stage proportions p for 7 DOF PR2 manipulator.

boxes to provide an instance of a high-speed but low-fidelity collision checking framework. We do not rely on GPU acceleration or parallelization to speed up any part of the algorithm. In all following simulations, we use a fixed value of kernel width $\gamma = 10$ and conditional bias parameter $r^+ = 2$.

With a dataset of size $N = 4000$, the recall is sufficiently large (over 90%) with both FCL and GJK KCDs as shown in Table 3. FPR increases as the number of obstacles increases because the C-space is high dimensional so the spacing of the 4000 points cause the Fastron to pad the C-space obstacles more. The speed improvement of the FCD over KCD increases as the number of obstacles increases.

We evaluate the performance of the model in changing environments under various dataset sizes N and relabeling allowances A by considering the average recall, false positive rate, and update time with exploitation proportion $p = 0.5$ and a maximum nearest non-support point number $k_{NS} = 4$. Table 4 shows that update time increases with A because active learning involves KCDs. Recall decreases as N increases but is generally above 90%, and false positive rate improves as N increases.

We demonstrate one use case of the Fastron algorithm by implementing a standard RRT [2] using FCDs and KCDs for collision checks, henceforth referred to as FCD-RRT and KCD-RRT, respectively. We choose the standard RRT due to its simplicity, yet we note that dynamic RRTs and other variants designed for moving obstacles will see similar benefits from the Fastron. We repeatedly compute an RRT from scratch over the course of a 10 second trial, translating the workspace obstacle between each RRT plan to simulate a changing environment. The obstacle is randomly placed such that the arm cannot take a straight approach to the goal configuration. We use $N = 4000$ and $A = 0.3N$ for the RRT experiments.

When using FCL for KCDs, the average time spent in the collision checking stage of the FCD-RRTs is 108 ms, while 399 ms is required for the KCD-RRT's collision checking stage. When using GJK for KCDs, the collision checking stage takes 104 ms for FCD-RRTs and 164 ms for KCD-RRTs. Model updating and active learning (which together take around 30 ms) are included when timing the FCD-RRTs' collision checking stages. As the collision checking stage is 3.7 times faster in the high-precision FCL case and 1.6 times faster in the low-fidelity GJK case, the Fastron demonstrates the collision check bottleneck sampling-based motion planners face may be lessened, especially if KCDs needed for information gathering are parallelized.

4 Concluding Remarks

We present the Fastron algorithm as a method to quickly represent and update a learning-based C-space model to be used for fast, proxy collision detection. We note that the Fastron complements, but not entirely supplants, kinematic-based collision checks because KCDs still serve as an oracle for acquiring information about the changing environment. The advantage of utilizing a learning-based model to represent C-space is a dense representation is not required. Instead, only a few support points represent the decision boundary between C_{free} and C_{obs}, whose structure may be exploited to reduce costly query evaluations of the oracle KCD function.

A limitation of the Fastron method is FPR increases as the obstacle number increases due to the C-space obstacle padding. Another limitation is the necessity to store the dataset $\mathcal{D}$ and the Gram matrix G to speed up model updates. In future work, we will determine a method to incorporate resampling (rather than relabeling) to increase model precision and a method to provide a confidence score on the classification output to facilitate active learning by guiding the information search toward regions of low confidence.

References

[1] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun. Principles of Robot Motion: Theory, Algorithms, and Implementations, 2005.

[2] S. M. Lavalle. Rapidly-Exploring Random Trees: A New Tool for Path Planning. Technical report, 1998.

[3] Y. K. Hwang and N. Ahuja. Gross motion planninga survey. *ACM Computing Surveys (CSUR)*, 24(3):219–291, 1992.

[4] M. Elbanhawi and M. Simic. Sampling-Based Robot Motion Planning: A Review. *IEEE Access*, 2:56–77, 2014. ISSN 2169-3536. doi:10.1109/ACCESS.2014.2302442.

[5] S. Murray, W. Floyd-Jones, Y. Qi, D. Sorin, and G. Konidaris. Robot Motion Planning on a Chip. In *Proceedings of Robotics: Science and Systems*, AnnArbor, Michigan, June 2016. doi:10.15607/RSS.2016.XII.004.

[6] J. Pan, X. Zhang, and D. Manocha. Efficient penetration depth approximation using active learning. *ACM Trans. Graph.*, 32, 2013.

[7] G. Schohn and D. Cohn. Less is more: Active learning with support vector machines. In *ICML*, pages 839–846. Citeseer, 2000.

[8] J. Huh and D. D. Lee. Learning high-dimensional Mixture Models for fast collision detection in Rapidly-Exploring Random Trees. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 63–69, May 2016. doi:10.1109/ICRA.2016.7487116.

[9] B. Burns and O. Brock. Toward Optimal Configuration Space Sampling. In *Proceedings of Robotics: Science and Systems*, Cambridge, USA, June 2005. doi:10.15607/RSS.2005.I.015.

[10] J. Pan and D. Manocha. Fast probabilistic collision checking for sampling-based motion planning using locality-sensitive hashing. *The International Journal of Robotics Research*, 35(12): 1477–1496, 2016. doi:10.1177/0278364916640908. URL `http://dx.doi.org/10.1177/0278364916640908`.

[11] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal on Robotics and Automation*, 4(2):193–203, Apr 1988. ISSN 0882-4967. doi:10.1109/56.2083.

[12] J. Pan, S. Chitta, and D. Manocha. FCL: A general purpose library for collision and proximity queries. In *2012 IEEE International Conference on Robotics and Automation*, pages 3859–3866, May 2012. doi:10.1109/ICRA.2012.6225337.