Fastron: A Learning-Based Configuration Space Model for Rapid Collision Detection for Gross Motion Planning in Changing Environments

Nikhil Das, Naman Gupta, Michael Yip
University of California, San Diego

I. Introduction

Collision detection is a necessary but costly step for sampling-based motion planners, such as Rapidly-Exploring Random Trees [7]. Motion planning is typically performed in configuration space (C-space)[2]. 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. The goal of motion planning in C-space is to find a path that lies entirely in C_{free}. A difficulty in C-space planning is that obstacle geometries generally do not trivially map from the workspace to C-space [2, 6]. Sampling-based motion planners instead infer or estimate C-space obstacles through collision checks on random configurations.

While sampling-based methods eventually generate a feasible motion plan, sampling-based motion planners spend a large majority of their computation time on performing collision checks [3]. In the case of workspaces with moving obstacles, C_{obs} changes non-trivially, making maintenance of an updated map in C-space for collision detection a bottleneck in performance.

A. Contributions

Robot manipulation requires both gross and fine motion planning. Realizing the high cost involved in kinematic-based collision detections (KCDs) for motion planning, we present a fast technique to update an approximate C-space representation to be used as a proxy collision detector for gross motion planning in this paper. The purpose of this effort is to reduce the computation required for gross motion planning such that more resources may be dedicated toward sampling or data collection required for fine motion planning.

The novel contributions of this work are:

1) a simple yet efficient method to sparsely represent C-space obstacles using a kernel perceptron hyperplane
2) a modified kernel perceptron that allows addition and removal of support points, and
3) an active learning strategy to update the model in response to a changing environment.

B. Related Work

Previous works have utilized machine learning models to approximate C_{free} and C_{obs}, such as with Gaussian mixture models [5], k-Nearest Neighbor (kNN) models, [1, 8] and incremental support vector machines (SVMs) [10]. Active learning strategies are employed to guide the search for new information to update these models. Such strategies potentially reduce the number of query evaluations during model updates [10, 11].

Similar to the SVM method, the Fastron uses a hyperplane model to represent C-space. However, the Fastron uses active learning to directly modify a single hyperplane model to accommodate for changing environments in real-time rather than to increase precision during an offline computation phase.

II. Methods

The block diagram in Fig. 1 summarizes the steps of the Fastron algorithm. The algorithm cycles through two steps: updating the collision boundary model (II-A) and active learning to search for collision status changes (II-B). These two steps are summarized in Algorithms 1 and 2, respectively.

A. Collision Boundary Model Updating

Given a labeled set $\mathcal{D}$ of N robot configurations, the kernel perceptron algorithm can predict the collision status $\hat{y}(x)$ of a configuration x using $\hat{y}(x) = sgn\left(\sum_{i:x_i \in S} \alpha_i K(x_i, x)\right)$, where $\alpha \in \mathbb{R}^N$ is a sparse weight vector, $K(\cdot,\cdot)$ is a kernel function, x_i is a sample in $\mathcal{D}$ with collision status label $y_i \in [-1, +1]$, and $S \in \mathcal{D}$ is the set of support points for which α_i is nonzero.

The Fastron model maintains three structures: a hypothesis vector $F \in \mathbb{R}^N$, Gram matrix $G \in \mathbb{R}^{N \times N}$, and α. Element

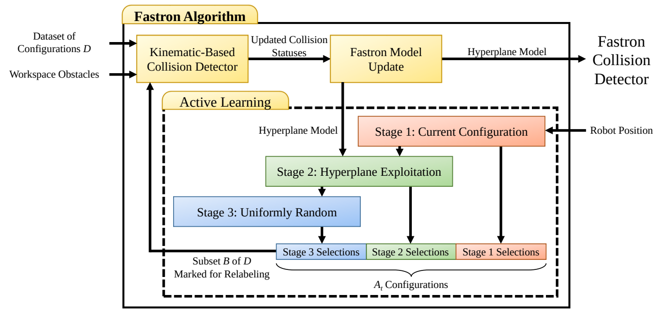
Fastron Algorithm
Dataset of Configurations D
Workspace Obstacles
Kinematic-Based Collision Detector
Updated Collision Statuses
Fastron Model Update
Hyperplane Model
Fastron Collision Detector
Active Learning
Hyperplane Model
Stage 1: Current Configuration
Robot Position
Stage 2: Hyperplane Exploitation
Stage 3: Uniformly Random
Stage 3 Selections
Stage 2 Selections
Stage 1 Selections
Subset B of D Marked for Relabeling
A_t Configurations

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

Algorithm 1: Fastron Model Updating

```
Input: Weight vector $\alpha$; hypothesis vector $F$; Gram matrix $G$; true
       labels $y$ for dataset $\mathcal{D}$; max number of updates $maxUpdates$
Output: Updated $\alpha$; updated $F$
1  for $iter = 1$ to $maxUpdates$ do
       // Remove redundant support points
2      while $\exists\, i$ s.t. $y_i(F_i - \alpha_i) > 0$ and $\alpha_i \neq 0$ do
3          $j \leftarrow \text{argmax}_i\, y_i(F_i - \alpha_i)$
4          $F_i \leftarrow F_i - G_{ij}\alpha_j \,\forall i$
5          $\alpha_j \leftarrow 0$
       // Margin-based prioritization
6      if $y_i F_i > 0 \;\forall i$ then
7          return $\alpha, F$
8      else
9          $j \leftarrow \text{argmin}_i\, y_i F_i$
       // Weight correction
10     $\Delta\alpha \leftarrow y_j - F_j$
11     $\alpha_j \leftarrow \alpha_j + \Delta\alpha$
12     $F_i \leftarrow F_i + G_{ij}\Delta\alpha \,\forall i$
13 return $\alpha, F$
```

Algorithm 2: Fastron Active Learning Strategy

```
Input: Total, stage 1, and stage 2 KCD allowances $A_t$, $A_1$, and $A_2$;
       support set $S$; dataset $\mathcal{D}$; Gram matrix $G$ for $\mathcal{D}$; max number
       of nearest non-support points $k_{NS}$; current configuration $q_c$
Output: Set of points $B \subset \mathcal{D}$ to be relabeled
   // Stage 1
1  $B \leftarrow knnsearch(\mathcal{D}, q_c, A_1)$
   // Stage 2
2  if $|B \cup S| \leq A_t - A_1$ then
3      $B \leftarrow B \cup S$
4      for $k = 1$ to $k_{NS}$ do
5          if $|B| < A_1 + A_2$ then
6              $B \leftarrow B \cup knnsearch(\mathcal{D}\backslash S, S, k)$
7  else
8      $B \leftarrow B \cup sample(S, A_t - A_1)$
   // Stage 3
9  $B \leftarrow B \cup sample(\mathcal{D}\backslash B, A_t - |B|)$
10 return $B$
```

G_{ij} is determined for each configuration pair in $\mathcal{D}$ using a Gaussian kernel. F stores the matrix-vector product of G and α and is useful to avoid repeated matrix-vector product evaluations. Configurations with the most negative value of $y_i F_i$ are most erroneously classified by the model, and thus their α_i is selected to be updated. The algorithm terminates when $y_i F_i > 0\,\forall i$.

To encourage a sparse model, redundant support points are removed from S by setting their α to 0. Redundant support points are those that will be correctly classified even if their corresponding α value is 0, i.e., $\{x | x \in S \wedge y(F - \alpha) > 0\}$. Pseudocode is provided in Algorithm 1.

B. Active Learning Strategy

Rather than performing collision checks on each configuration in $\mathcal{D}$ to check for collision status changes, a subset B of $\mathcal{D}$ is selected to relabel, where $|B|$ is set by an allowance A_t on the number of KCDs to perform per model update. A_1 KCDs are dedicated to search for status changes near the

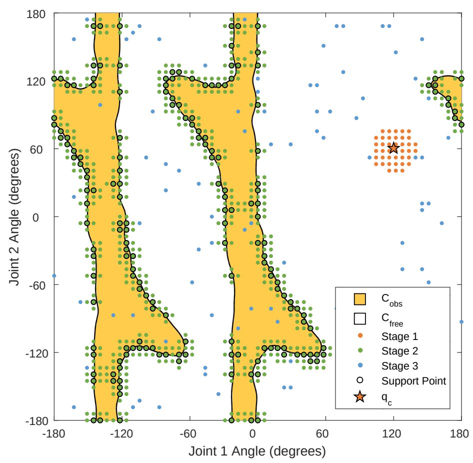

Fig. 2: Example set of samples selected for relabeling via KCD by the active learning strategy.

current robot configuration, A_2 KCDs are dedicated to search near the hyperplane model to search for changes, and the remainder of A_t is exhausted by randomly selecting points in $\mathcal{D}$ to check. The Gram matrix may be used as a lookup table for distances during the near-hyperplane kNN search. Pseudocode is provided in Algorithm 2, and an example set of points selected by the strategy is shown in Fig. 2.

III. Results and Concluding Remarks

For a 2 DOF manipulator with revolute joints in MATLAB, we randomly generate and place a polygonal obstacle in a 2D environment and use the Gilbert-Johnson-Keerthi (GJK) algorithm [4] for KCD. Using $N = 625$, each FCD takes 34μs with 97% accuracy and 98% recall, while each KCD takes 164μs. FCD time does not increase significantly when there are more workspace obstacles. With 3 workspace obstacles, FCD time is 39μs while KCD time is 369μs, with 93% accuracy and 98% recall. In a changing environment with a single moving obstacle, the accuracy was 98% and recall was 95%, with an average model update time of 13.5 ms with $N = 625$ and $A_t = 0.2N$.

For a 7 DOF PR2 manipulator in ROS, we place a polyhedral obstacle in a 3D environment, and use the Flexible Collision Library [9] for KCD. Using $N = 4000$, each FCD takes 12μs, while each KCD takes 23μs. In a changing environment with a single moving obstacle, the accuracy was 85% and recall was 88%, with an average model update time of 35 ms with $N = 4000$ and $A_t = 0.5N$.

These preliminary results suggest the Fastron algorithm is a promising method for quickly creating a proxy collision detection model for gross motion planning in changing environments. As all results were achieved without parallel computing or GPU acceleration, collision checks and model updates may be even faster when utilized. Future work includes exploring methods to improve the precision of the model to potentially adapt this algorithm for fine motion planning and determining a method to provide a confidence score on the classification output.

References

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

[2] Howie Choset, Kevin M Lynch, Seth Hutchinson, G Kantor, Wolfram Burgard, Lydia E Kavraki, and Sebastian Thrun. Principles of Robot Motion: Theory, Algorithms, and Implementations, 2005.

[3] 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.

[4] 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.

[5] 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.

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

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

[8] 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.

[9] 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.

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

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