Autonomous Navigation in Unknown Environments using Sparse Kernel-based Occupancy Mapping

Thai Duong Nikhil Das Michael Yip Nikolay Atanasov

***Abstract*— This paper focuses on real-time occupancy mapping and collision checking onboard an autonomous robot navigating in an unknown environment. We propose a new map representation, in which occupied and free space are separated by the decision boundary of a kernel perceptron classifier. We develop an online training algorithm that maintains a very sparse set of support vectors to represent obstacle boundaries in configuration space. We also derive conditions that allow complete (without sampling) collision-checking for piecewise-linear and piecewise-polynomial robot trajectories. We demonstrate the effectiveness of our mapping and collision checking algorithms for autonomous navigation of an Ackermann-drive robot in unknown environments.**

I. Introduction

Autonomous navigation in robotics involves localization, mapping, motion planning, and control in a partially known environment perceived through streaming data from onboard sensors [1], [2]. In this paper, we focus on the mapping problem and, specifically, on enabling large-scale, yet compact, representations and efficient collision checking to support autonomous navigation. Existing work uses a variety of map representations based on voxels [3], [4], [5], [6], surfels [7], geometric primitives [8], objects [9], etc. We propose a novel mapping method that uses a kernel perceptron model to represent the occupied and free space of the environment. The model uses a set of support vectors to represent obstacle boundaries in configuration space. The complexity of this representation scales with the complexity of the obstacle boundaries rather than the environment size. We develop an online training algorithm to update the support vectors incrementally as new depth observations of the local surroundings are provided by the robot's sensors. To enable motion planning in the new occupancy representation, we develop an efficient collision checking algorithm for piecewise-linear and piecewise-polynomial trajectories in configuration space.

Related Work. Occupancy grid mapping is a commonly used approach for modeling the free and occupied space of an environment. The space is discretized into a collection of cells, whose occupancy probabilities are estimated online using the robot's sensory data. While early work [3] assumes that the cells are independent, Gaussian process (GP) occupancy mapping [10], [11], [12] uses a kernel function to capture the correlation among grid cells and predict the occupancy of unobserved cells. Online training of a Gaussian process model, however, does not scale well as its computational complexity grows cubically with the number of data points. Ramos et al. [13] improve on this by projecting the data points into Hilbert space and training a logistic regression model. Lopez and How [14] propose an efficient determinstic alternative, which builds a k-d tree from point clouds and queries the nearest obstacles for collision checking. Using spatial partitioning similar to a k-d tree, octree-based maps [4], [15] offer efficient map storage by performing octree compression. Meanwhile, AtomMap [16] stores a collection of spheres in a k-d tree as a way to avoid grid cell discretization of the map.

Navigation in an unknown environment, requires the safety of potential robot trajectory to be evaluated through a huge amount of collision checks with respect to the map representation [17], [18], [19]. Many works rely on sampling-based collision checking, simplifying the safety verification of continuous-time trajectories by evaluating only a finite set of samples along the trajectory [20], [18]. This may be undesirable in safety critical applications. Bialkowski et al. [17] propose an efficient collision checking method using safety certificates with respect to the nearest obstacles. Using a different perspective, learning-based collision checking methods [21], [22], [23], [24] sample data from the environment and train machine learning models to approximate the obstacle boundaries. Pan et al. [22] propose an incremental support vector machine model for pairs of obstacles but train the models offline. Closely related to our work, Das et al. [21], [25] develop an online training algorithm, called Fastron, to train a kernel perceptron collision classifier. To handle dynamic environments, Fastron actively resamples the environment and updates the model globally. Geometry-based collision checking methods, such as the Flexible Collision Library (FCL) [26], are also related but rely on mesh representations of the environment which may be inefficient to generate from local observations.

Inspired by GP mapping techniques, we utilize a radial basis function (RBF) kernel to capture occupancy correlations but focus on a compact representation of obstacle boundaries using kernel perceptron. Furthermore, motivated by the safety certificates in [17], we derive our own safety guarantees for efficient collision checking algorithms.

Contributions. This paper introduces a sparse kernel-based mapping method that:

- represents continuous-space occupancy using a sparse set of support vectors stored in an R^*-tree data structure, scaling efficiently with the complexity of obstacle boundaries (Sec. IV),

We gratefully acknowledge support from ARL DCIST CRA W911NF-17-2-0181 and ONR SAI N00014-18-1-2828.

The authors are with the Department of Electrical and Computer Engineering, University of California, San Diego, La Jolla, CA 92093 USA {tduong, nrdas, yip, natanasov}@ucsd.edu

- allows online map updates from streaming partial observations using our proposed incremental kernel perceptron training built on the Fastron model (Sec. IV),
- provides efficient and complete (without sampling) collision checking for piecewise-linear and piecewise-polynomial trajectories with safety guarantees based on nearest support vectors (Sec. V and VI).

II. Problem Formulation

Consider a spherical robot with center $\mathbf{s}$ in a *workspace* $\mathcal{S} := [0,1]^d$ and radius $r \in \mathbb{R}_{>0}$ navigating in an unknown environment. Let $\mathcal{S}_{obs}$ and $\mathcal{S}_{free}$ be the obstacle space and free space in $\mathcal{S}$, respectively. In *configuration space* (C-space) $\mathcal{C}$, the robot body becomes a point $\mathbf{s}$, while the obstacle space and free space are transformed as $\mathcal{C}_{obs} = \cup_{\mathbf{x}\in\mathcal{S}_{obs}}\mathcal{B}(\mathbf{x}, r)$, where $\mathcal{B}(\mathbf{x}, r) = \{x' \in \mathcal{S} : \|x - x'\|_2 \leq r\}$, and $\mathcal{C}_{free} = \mathcal{S} \setminus \mathcal{C}_{obs}$. Assume that the robot position $\mathbf{s}_{t_k} \in \mathcal{C}$ at time t_k is known or provided by a localization algorithm. Let $\mathbf{s}_{t_{k+1}} = f(\mathbf{s}_{t_k}, \mathbf{a}_k)$ characterize the robot dynamics for an action $\mathbf{a}_k \in \mathcal{A}$. Applying $\mathbf{a}_k$ at $\mathbf{s}_{t_k}$ also incurs a motion cost $c(\mathbf{s}_{t_k}, \mathbf{a}_k)$ (e.g., distance or energy). The robot is equipped with a depth sensor that provides distance measurements $\mathbf{z}_{t_k}$ to the obstacle space $\mathcal{S}_{obs}$ within its field of view. Our objective is to construct an occupancy map $\hat{m}_{t_k} : \mathcal{C} \to \{-1, 1\}$ of the C-space based on accumulated observations $\mathbf{z}_{t_{0:k}}$, where "-1" and "1" mean "free" and "occupied", respectively. Assuming unobserved regions are free, we rely on $\hat{m}_{t_k}$ to plan a robot trajectory to a goal region $\mathcal{C}_{goal} \subset \mathcal{C}_{free}$. As the robot is navigating, new sensor data is used to update the map and recompute the motion plan. In this online setting, the map update, $\hat{m}_{t_{k+1}} = g(\hat{m}_{t_k}, \mathbf{z}_{t_k})$, is a function of the previous estimate $\hat{m}_{t_k}$ and the new data $\mathbf{z}_{t_k}$.

Problem 1. Given a start state $\mathbf{s}_0 \in \mathcal{C}_{free}$ and a goal region $\mathcal{C}_{goal} \subset \mathcal{C}_{free}$, find a sequence of actions that leads the robot to $\mathcal{C}_{goal}$ safely, while minimizing the motion cost:

$$\min_{N,\mathbf{a}_0,\ldots,\mathbf{a}_N} \sum_{k=0}^{N-1} c(\mathbf{s}_{t_k}, \mathbf{a}_k) \tag{1}$$
$$\text{s.t.} \quad \mathbf{s}_{t_{k+1}} = f(\mathbf{s}_{t_k}, \mathbf{a}_k), \hat{m}_{t_{k+1}} = g(\hat{m}_{t_k}, \mathbf{z}_{t_k}),$$
$$\mathbf{s}_{t_N} \in \mathcal{C}_{goal},\ \hat{m}_{t_k}(\mathbf{s}_{t_k}) = -1,\ k = 0, \ldots, N.$$

III. Preliminaries

In this section, we provide a summary on kernel perceptron and Fastron which is useful for our derivations in the next sections. The *kernel perceptron* model is used to classify a set of N labeled data points. For $l = 1, \ldots, N$, a data point $\mathbf{x}_l$ with label $y_l \in \{-1, 1\}$ is assigned a weight $\alpha_l \in \mathbb{R}$. Training determines a set of M^+ positive support vectors and their weights $\Lambda^+ = \{(\mathbf{x}_i, \alpha_i)\}$ and a set of M^- negative support vectors and their weights $\Lambda^- = \{(\mathbf{x}_j^-, \alpha_j^-)\}$. The decision boundary is represented by a score function,

$$F(\mathbf{x}) = \sum_{i=1}^{M^+} \alpha_i^+ k(\mathbf{x}_i^+, \mathbf{x}) - \sum_{j=1}^{M^-} \alpha_j^- k(\mathbf{x}_j^-, \mathbf{x}), \tag{2}$$

where $k(\cdot, \cdot)$ is a kernel function and $\alpha_j^-, \alpha_i^+ > 0$. The sign of $F(\mathbf{x})$ is used to predict the class of a test point $\mathbf{x}$.

Algorithm 1 Incremental Fastron Training with Local Data

Input: Sets $\Lambda^+ = \{(\mathbf{x}_i^+, \alpha_i^+)\}$ and $\Lambda^- = \{(\mathbf{x}_j^-, \alpha_j^-)\}$ of M^+ positive and M^- negative support vectors stored in an R^*-tree; Local dataset $\mathcal{D} = \{(\mathbf{x}_l, y_l)\}$ generated from location $\mathbf{s}_t$; $\xi^+, \xi^- > 0$; N_{max}
Output: Updated Λ^+, Λ^-.

Get K^+, K^- nearest negative and positive support vectors.
for $(\mathbf{x}_l, y_l)$ in $\mathcal{D}$ **do**
  Calculate $F_l = \sum_{i=1}^{K^+} \alpha_i^+ k(\mathbf{x}_i^+, \mathbf{x}_l) - \sum_{j=1}^{K^-} \alpha_j^- k(\mathbf{x}_j^-, \mathbf{x}_l)$
for $t = 1$ to N_{max} **do**
  if $y_l F_l > 0 \quad \forall l$ **then return** Λ^+, Λ^- ▷ Margin-based priotization
  $m = \text{argmin}_l y_l F_l$
  if $y_m > 0$ **then** $\Delta\alpha = \xi^+ y_m - F_m$ ▷ One-step weight correction
  else $\Delta\alpha = \xi^- y_m - F_m$
  if $\exists(\mathbf{x}_m, \alpha_m^+) \in \Lambda^+$ **then** $\alpha_m^+ \mathrel{+}= \Delta\alpha$, $F_l \mathrel{+}= k(\mathbf{x}_l, \mathbf{x}_m)\Delta\alpha, \forall l$
  else if $\exists(\mathbf{x}_m, \alpha_m^-) \in \Lambda^-$ **then** $\alpha_m^- \mathrel{-}= \Delta\alpha$, $F_l \mathrel{-}= k(\mathbf{x}_l, \mathbf{x}_m)\Delta\alpha, \forall l$
  else if $y_m > 0$ **then** $\alpha_m^+ = \Delta\alpha$, $\Lambda^+ = \Lambda^+ \cup \{(\mathbf{x}_m, \alpha_m^+)\}$
  else $\alpha_m^- = -\Delta\alpha$, $\Lambda^- = \Lambda^- \cup \{(\mathbf{x}_m, \alpha_m^-)\}$
  for $(\mathbf{x}_l, y_l) \in \mathcal{D}$ **do** ▷ Remove redundant support vectors
    if $\exists(\mathbf{x}_l, \alpha_l^+) \in \Lambda^+$ and $y_l(F_l - \alpha_l^+) > 0$ **then**
      $\Lambda^+ = \Lambda^+ \setminus \{(\mathbf{x}_l, \alpha_l^+)\}$, $F_n \mathrel{-}= k(\mathbf{x}_l, \mathbf{x}_n)\alpha_l^+, \forall(\mathbf{x}_n, \cdot) \in \mathcal{D}$
    if $\exists(\mathbf{x}_l, \alpha_l^-) \in \Lambda^-$ and $y_l(F_l + \alpha_l^-) > 0$ **then**
      $\Lambda^- = \Lambda^- \setminus \{(\mathbf{x}_l, \alpha_l^-)\}$, $F_n \mathrel{+}= k(\mathbf{x}_l, \mathbf{x}_n)\alpha_l^-, \forall(\mathbf{x}_n, \cdot) \in \mathcal{D}$
return Λ^+, Λ^-

Fastron [21], [25] is an efficient training algorithm for the kernel perceptron model. It prioritizes updating misclassified points based on their margins instead of random selection as in the original kernel perceptron training. Our previous work [21], [25] shows that if $\alpha_l = \xi y_l - (\sum_{i\neq l} \alpha_i^+ k(\mathbf{x}_i^+, \mathbf{x}_l) - \sum_{j\neq l} \alpha_j^- k(\mathbf{x}_j^-, \mathbf{x}_l))$ for some $\xi > 0$, then $\mathbf{x}_l$ is correctly classified with label y_l. Based on this fact, Fastron utilizes one-step weight correction $\Delta\alpha = \xi y_l - (\sum \alpha_i^+ k(\mathbf{x}_i^+, \mathbf{x}_l) - \sum \alpha_j^- k(\mathbf{x}_j^-, \mathbf{x}_l))$ where $\xi = \xi^+$ if $y_l = 1$ and $\xi = \xi^-$ if $y_l = -1$.

IV. Sparse Kernel-based Occupancy Mapping

We propose a new version of the Fastron algorithm, utilizing only local streaming data, to achieve real-time sparse kernel-based occupancy mapping. For online learning, Fastron resamples a *global* training dataset around the current support vectors and updates the support vectors. In our setting, only *local* data in the robot's vicinity is available from the onboard sensors. We propose an incremental version of Fastron in Alg. 1 such that: 1) training is performed with *local* data $\mathcal{D}_t$ generated from a depth measurement $\mathbf{z}_t$ at a time t and 2) the support vectors are stored in an R^*-tree data structure, enabling efficient score function (2) computations.

Data Generation. Fig. 1a illustrates a lidar scan $\mathbf{z}_t$ obtained by the robot at time t. In configuration space, each laser ray end point corresponds to a ball-shaped obstacle, while the robot body becomes a point as shown in Fig. 1b. To generate local training data $\mathcal{D}_t$, the occupied and free C-space areas observed by the lidar are sampled (e.g., on a regular grid). As shown in Fig. 1c, this generates a set $\bar{\mathcal{D}}_t$ of points with label "1" (occupied) in the ball-shaped occupied areas and with label "-1" (free), outside. As unobserved areas are assumed free, neighboring points to the occupied samples in $\bar{\mathcal{D}}_t$ that are not already in $\bar{\mathcal{D}}_t$ or in the support vectors are added to an augmented set $\tilde{\mathcal{D}}_t$ with label "-1". Adding the neighboring points is enough to maintain good decision boundaries for the obstacles. The augmented dataset $\tilde{\mathcal{D}}_t$ is

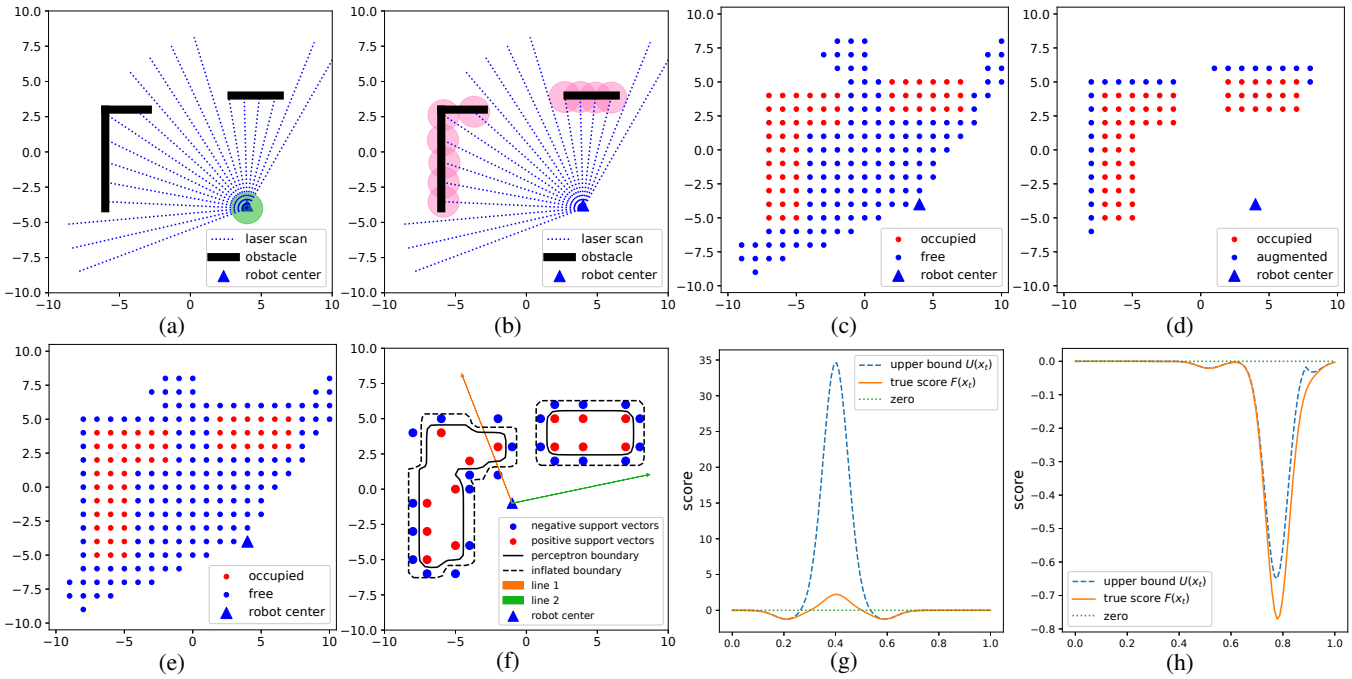

Fig. 1: Example of our mapping method: (a) scan in work space; (b) scan in C-space; (c) samples from scan; (d) augmented free points (e) training data $\mathcal{D}$ from one lidar scan; (f) the exact decision boundary generated by the score $F(\mathbf{x})$ and the inflated boundary generated by the upper bound $U(\mathbf{x})$; $F(\mathbf{x})$ and $U(\mathbf{x})$ along two rays: (g) one that enters the occupied space and (h) one that remains obstacle-free.

illustrated in Fig. 1d assuming the set of support vectors is empty. The local data $\mathcal{D}_t = \bar{\mathcal{D}}_t \cup \tilde{\mathcal{D}}_t$ (Fig. 1e) is used in our Incremental Fastron Algorithm to update the support vectors (Fig. 1f). Storing the sets of support vectors Λ^+, Λ^- over time requires significantly less memory than storing the training data $\cup_t \mathcal{D}_t$. The occupancy of a query point $\mathbf{x}$ can be estimated from the support vectors by evaluating the score function $F(\mathbf{x})$ in Eq. (2). Specifically, $\hat{m}_t(\mathbf{x}) = -1$ if $F(\mathbf{x}) < 0$ and $\hat{m}_t(\mathbf{x}) = 1$ if $F(\mathbf{x}) \geq 0$. Fig. 1f illustrates the boundaries generated by Alg. 1.

Score Approximation. As the robot explores the environment, the number of support vectors required to represent the obstacle boundaries increases. Since the score function (2) depends on all support vectors, the time to train the kernel perceptron model online would increase as well. We propose an approximation to the score function $F(\mathbf{x})$ under the assumption that the kernel function $k(\mathbf{x}_i, \mathbf{x}_j)$ is *radial* (depends only on $\|\mathbf{x}_i - \mathbf{x}_j\|$) and *monotone* (its value decreases as $\|\mathbf{x}_i - \mathbf{x}_j\|$ increases). To keep the presentation specific, we make the following assumption in the remainder of the paper.

Assumption. $k(\mathbf{x}_i, \mathbf{x}_j) := \eta \exp\left(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2\right)$

The kernel parameters $\eta, \gamma \in \mathbb{R}_{>0}$ can be optimized offline via automatic relevance determination [27] using training data from known occupancy maps. The assumption implies that the value of $F(\mathbf{x})$ is not affected significantly by support vectors far from $\mathbf{x}$. We introduce an R^*-tree data structure constructed from the sets of positive and negative support vectors Λ^+, Λ^- to allow efficient K nearest-neighbor lookup. The K nearest support vectors, consisting of K^+ and K^- positive and negative support vectors, are used to approximate the score $F(\mathbf{x})$ (Lines 1-3 in Alg. 1).

V. Collision Checking with Kernel-based Maps

A map representation is useful for navigation only if it allows checking a potential robot trajectory $\mathbf{s}(t)$ over time t for collisions. We derive conditions for complete (without sampling) collision-checking of continuous-time trajectories $\mathbf{s}(t)$ in our sparse kernel-based occupancy map representation. Checking that a curve $\mathbf{s}(t)$ is collision-free is equivalent to verifying that $F(\mathbf{s}(t)) < 0$, $\forall t \geq 0$. It is not possible to express this condition for t explicitly due to the nonlinearity of F. Instead, in Prop. 1, we show that an accurate upper bound $U(\mathbf{s}(t))$ of the score $F(\mathbf{s}(t))$ exists and can be used to evaluate the condition $U(\mathbf{s}(t)) < 0$ explicitly in terms of t.

Proposition 1. *For any* $(\mathbf{x}_j^-, \alpha_j^-) \in \Lambda^-$, *the score* $F(\mathbf{x})$ *is bounded above by* $U(\mathbf{x}) = k(\mathbf{x}, \mathbf{x}_*^+) \sum_{i=1}^{M^+} \alpha_i^+ - k(\mathbf{x}, \mathbf{x}_j^-)\alpha_j^-$ *where* $\mathbf{x}_*^+$ *is the closest positive support vector to* $\mathbf{x}$.

Proof. The proposition holds because $k(\mathbf{x}, \mathbf{x}_i^+) \leq k(\mathbf{x}, \mathbf{x}_*^+)$, $\forall \mathbf{x}_i^+$ and $\sum_{j=1}^{M^-} \alpha_j^- k(\mathbf{x}, \mathbf{x}_j^-) \geq \alpha_j^- k(\mathbf{x}, \mathbf{x}_j^-)$, $\forall \mathbf{x}_j^-$. □

Fig. 1f, 1g, 1h illustrate the exact decision boundary $F(\mathbf{x}) = 0$ and the accuracy of the upper bound $U(\mathbf{s}(t))$ along two lines $\mathbf{s}(t)$ in C-space. The upper bound $U(\mathbf{s}(t))$ is loose in the occupied space but remains close to the score $F(\mathbf{s}(t))$ in the free space since the RBF kernel $k(\mathbf{x}, \mathbf{x}')$ is negligible away from the obstacle's boundary. As a result, the boundary $U(\mathbf{x}) = 0$ remains close to the true decision boundary. The upper bound provides a conservative but fairly accurate "inflated boundary", allowing efficient collision checking for line segments and polynomial curves as shown next.

A. Collision Checking for Line Segments

Suppose that the robot's path is described by a ray $\mathbf{s}(t) = \mathbf{s}_0 + t\mathbf{v}$, $t \geq 0$ such that $\mathbf{s}_0$ is obstacle-free, i.e., $U(\mathbf{s}_0) < 0$,

and $\mathbf{v}$ is constant. To check if $\mathbf{s}(t)$ collides with the inflated boundary, we find the first time t_u such that $U(\mathbf{s}(t_u)) \geq 0$. This means that $\mathbf{s}(t)$ is collision-free for $t \in [0, t_u)$.

Proposition 2. *Consider a ray $\mathbf{s}(t) = \mathbf{s}_0 + t\mathbf{v}$, $t \geq 0$ with $U(\mathbf{s}_0) < 0$. Let $\mathbf{x}_i^+$ and $\mathbf{x}_j^-$ be arbitrary positive and negative support vectors. Then, any point $\mathbf{s}(t)$ is free as long as:*

$$t < t_u := \min_{i \in \{1,\dots,M^+\}} \rho(\mathbf{s}_0, \mathbf{x}_i^+, \mathbf{x}_j^-) \tag{3}$$

where $\beta = \frac{1}{\gamma}\left(\log(\alpha_j^-) - \log(\sum_{i=1}^{M^+} \alpha_i^+)\right)$ *and*

$$\rho(\mathbf{s}_0, \mathbf{x}_i^+, \mathbf{x}_j^-) = \begin{cases} +\infty, & \text{if } \mathbf{v}^T(\mathbf{x}_i^+ - \mathbf{x}_j^-) \leq 0 \\ \frac{\beta - \|\mathbf{s}_0 - \mathbf{x}_j^-\|^2 - \|\mathbf{s}_0 + \mathbf{x}_i^+\|^2}{2\mathbf{v}^T(\mathbf{x}_j^- - \mathbf{x}_i^+)}, & \text{if } \mathbf{v}^T(\mathbf{x}_i^+ - \mathbf{x}_j^-) > 0 \end{cases}.$$

Proof. From Prop. 1, a point $\mathbf{s}(t)$ is free if $U(\mathbf{s}(t)) < 0$ or

$$t < \rho(\mathbf{s}_0, \mathbf{x}_*^+, \mathbf{x}_j^-) \tag{4}$$

Since $\mathbf{x}_*^+$ varies with t but belongs to a finite set, $U(\mathbf{s}(t)) < 0$ if we take the minimum of $\rho(\mathbf{s}_0, \mathbf{x}_i^+, \mathbf{x}_j^-)$ over all $\mathbf{x}_i^+$. □

Prop. 1 and 2 hold for any negative support vector $\mathbf{x}_j^-$. Since $\mathbf{x}_j^-$ belongs to a finite set, we can take the best bound on t over the set of negative support vectors.

Corollary 1. *Consider a ray $\mathbf{s}(t) = \mathbf{s}_0 + t\mathbf{v}$, $t \geq 0$ with $U(\mathbf{s}_0) < 0$. Let $\mathbf{x}_i^+$ and $\mathbf{x}_j^-$ be arbitrary positive and negative support vectors, respectively. A point $\mathbf{s}(t)$ is free as long as:*

$$t < t_u^* := \min_{i \in \{1,\dots,M^+\}} \max_{j \in \{1,\dots,M^-\}} \rho(\mathbf{s}_0, \mathbf{x}_i^+, \mathbf{x}_j^-). \tag{5}$$

The computational complexities of calculating t_u and t_u^* are $O(M)$ and $O(M^2)$, respectively, where $M = M^+ + M^-$. Note that since often the robot's movement is limited to the neighborhood of its current position, t_u can reasonably approximate t_u^* if $\mathbf{x}_j^-$ is chosen as the negative support vector, closest to $\mathbf{s}_0$. Calculation of t_u in Eq. (3) is efficient in the sense that it has the same complexity as checking a point for collision ($O(M)$), yet it can evaluate the collision status for an entire line segment for $t \in [0, t_u)$ without sampling.

Collision checking becomes slower when the number of support vectors M increases. We improve this further by using K^+ and K^- nearest positive and negative support vectors instead of M^+ and M^-, respectively. Assuming K^+ and K^- are constant, the computational complexities of calculating t_u and t_u^* reduce to $O(\log M)$ which is the complexity of an R^*-tree lookup.

In path planning, one often performs collision checking for a line segment $(\mathbf{s}_A, \mathbf{s}_B)$. All points on the segment can be expressed as $\mathbf{s}(t_A) = \mathbf{s}_A + t_A\mathbf{v}_A$, $\mathbf{v}_A = \mathbf{s}_B - \mathbf{s}_A$, $0 \leq t_A \leq 1$. Using the upper bound t_{uA} on t_A provided by Eq. (3) or Eq. (5), we find the free region on $(\mathbf{s}_A, \mathbf{s}_B)$ starting from $\mathbf{s}_A$. Similarly, we calculate t_{uB} which specifies the free region from $\mathbf{s}_B$. If $t_{uA} + t_{uB} > 1$, the entire line segment is free; otherwise the segment is considered colliding. The proposed approach is summarized in Alg. 2 and illustrated in Fig. 2.

B. Collision Checking for Polynomial Curves

In the previous section, $\mathbf{v}$ was a constant velocity representing the direction of motion of the robot. In general, the velocity might be time varying leading to trajectories described by polynomial curves [28]. We extend the collision checking algorithm by finding an Euclidean ball $\mathcal{B}(\mathbf{s}_0, r)$ around $\mathbf{s}_0$ whose interior is free of obstacles.

Algorithm 2 Line segment collision check

Input: Line segment $(\mathbf{s}_A, \mathbf{s}_B)$; support vectors $\Lambda^+ = \{(\mathbf{x}_i^+, \alpha_i^+)\}$ and $\Lambda^- = \{(\mathbf{x}_j^-, \alpha_j^-)\}$

1: $\mathbf{v}_A = \mathbf{s}_B - \mathbf{s}_A$, $\mathbf{v}_B = \mathbf{s}_A - \mathbf{s}_B$
2: Calculate t_{uA} and t_{uB} using Eq. (3) or Eq. (5)
3: **if** $t_{uA} + t_{uB} > 1$ **then return** True (Free)
4: **else return** False (Colliding)

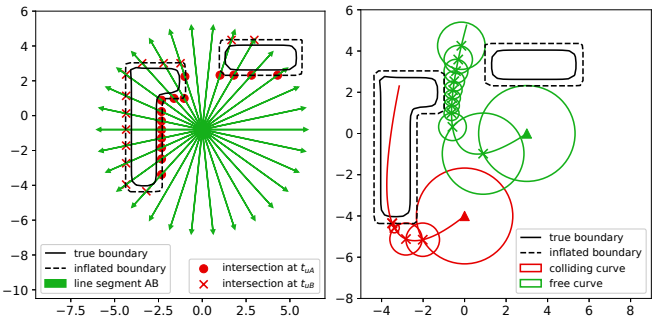

Fig. 2: Collision checking for line segments (left), with bounds t_{uA} and t_{uB} obtained from Eq. (5), and for second-order polynomial curves (right) using Euclidean balls.

Algorithm 3 Polynomial curve collision check

Input: Polynomial curve $\mathbf{s}(t)$, $t \in [0, t_f]$; threshold ε; support vectors $\Lambda^+ = \{(\mathbf{x}_i^+, \alpha_i^+)\}$ and $\Lambda^- = \{(\mathbf{x}_j^-, \alpha_j^-)\}$, $k = 0$, $t_0 = 0$

while True **do**
  Calculate r_k using Eq. (6) or Eq. (7).
  if $r_k < \varepsilon$ **then return** False (Colliding)
  Solve $\|\mathbf{s}(t) - \mathbf{s}(t_k)\| = r_k$ for $t_{k+1} \geq t_k$
  if $t_{k+1} \geq t_f$ **then return** True (Free)

Corollary 2. *Let $\mathbf{s}_0 \in \mathcal{C}$ be such that $U(\mathbf{s}_0) < 0$ and let $\mathbf{x}_i^+$ and $\mathbf{x}_j^-$ be arbitrary positive and negative support vectors. Then, every point inside the Euclidean balls $\mathcal{B}(\mathbf{s}_0, r_u) \subseteq \mathcal{B}(\mathbf{s}_0, r_u^*)$ is free for:*

$$r_u := \min_{i \in \{1,\dots,M^+\}} \bar{\rho}(\mathbf{s}_0, \mathbf{x}_i^+, \mathbf{x}_j^-) \tag{6}$$

$$r_u^* := \min_{i \in \{1,\dots,M^+\}} \max_{j \in \{1,\dots,M^-\}} \bar{\rho}(\mathbf{s}_0, \mathbf{x}_i^+, \mathbf{x}_j^-) \tag{7}$$

where $\bar{\rho}(\mathbf{s}_0, \mathbf{x}_i^+, \mathbf{x}_j^-) = \frac{\beta - \|\mathbf{s}_0 - \mathbf{x}_j^-\|^2 + \|\mathbf{s}_0 - \mathbf{x}_i^+\|^2}{2\|\mathbf{x}_j^- - \mathbf{x}_i^+\|}$ *and* $\beta = \frac{1}{\gamma}\left(\log(\alpha_j^-) - \log(\sum_{i=1}^{M^+} \alpha_i^+)\right)$.

Proof. Directly follows from Prop. 2 by using the Cauchy-Schwarz inequality to bound $\mathbf{v}^T(\mathbf{x}_j^- - \mathbf{x}_*^+) \leq \|(\mathbf{x}_j^- - \mathbf{x}_*^+)\|$ for any unit vector $\mathbf{v}$ (i.e. $\|\mathbf{v}\| = 1$). □

Consider a polynomial $\mathbf{s}(t) = \mathbf{s}_0 + \mathbf{a}_1 t + \mathbf{a}_2 t^2 + \ldots + \mathbf{a}_d t^d$, $t \in [0, t_f]$ from $\mathbf{s}_0$ to $\mathbf{s}_f := \mathbf{s}(t_f)$. Collorary 2 shows that all points inside $\mathcal{B}(\mathbf{s}_0, r)$ are free for $r = r_u$ or r_u^*. If we can find the smallest positive t_1 such that $\|\mathbf{s}(t_1) - \mathbf{s}_0\| = r$, then all points on the curve $\mathbf{s}(t)$ for $t \in [0, t_1)$ are free. This is equivalent to finding the smallest non-negative root of a $2d$-order polynomial. Note that, if $d \leq 2$, there is a closed-form solution for t_1. For higher order polynomials, one can use a root-finding algorithm to obtain t_1 numerically. We perform collision checking by iteratively covering the curve by Euclidean balls. If the radius of any ball is smaller than

a threshold ε, the curve is considered colliding. Otherwise, the curve is considered free. The collision checking process for d-order polynomial curves is shown in Alg. 3 and Fig. 2.

VI. Autonomous Navigation

Finally, we present a complete online mapping and navigation approach that solves Problem 1. Given the kernel-based map $\hat{m}_{t_k}$ proposed in Sec. IV, a motion planning algorithm such as A^* [29] may be used with our collision-checking algorithms to generate a path that solves the autonomous navigation problem [28]. The robot follows the path for some time and updates the map estimate $\hat{m}_{t_{k+1}}$ with new observations. Using the updated map, the robot re-plans the path and follows the new path instead. This is repeated until the goal is reached or a time limit is exceeded (Alg. 4). Note that as we use an inflated boundary for collision checking, our planning algorithm is not complete.

We consider robots with two different motion models. In simulation, we use a first-order fully actuated robot, $\dot{\mathbf{s}} = \mathbf{v}$, with piecewise-constant velocity $\mathbf{v}(t) \equiv \mathbf{v}_k \in \mathcal{V}$ for $t \in [t_k,\ t_{k+1})$, leading to piecewise-linear trajectories:

$$\mathbf{s}(t) = \mathbf{s}_k + (t - t_k)\mathbf{v}_k, \qquad t \in [t_k,\ t_{k+1}), \tag{8}$$

where $\mathbf{s}_k := \mathbf{s}(t_k)$. In real experiments, we consider a ground wheeled Ackermann-drive robot:

$$\dot{\mathbf{s}} = v \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix}, \qquad \dot{\theta} = \frac{v}{\ell} \tan\phi, \tag{9}$$

where $\mathbf{s} \in \mathbb{R}^2$ is the position, $\theta \in \mathbb{R}$ is the orientation, $v \in \mathbb{R}$ is the linear velocity, $\phi \in \mathbb{R}$ is the steering angle, and ℓ is the distance between the front and back wheels. The nonlinear car dynamics can be transformed into a 2nd-order fully actuated system $\ddot{\mathbf{s}} = \mathbf{a}$ via feedback linearization [30], [31]. Using piecewise-constant acceleration $\mathbf{a}(t) \equiv \mathbf{a}_k \in \mathcal{A}$ for $t \in [t_k,\ t_{k+1})$ leads to piecewise-2nd-order-polynomial trajectories $\mathbf{s}(t) = \mathbf{s}_k + (t - t_k)v_k \begin{bmatrix} \cos(\theta_k) \\ \sin(\theta_k) \end{bmatrix} + \frac{(t-t_k)^2}{2}\mathbf{a}_k$. where $\mathbf{s}_k := \mathbf{s}(t_k)$, $\theta_k := \theta(t_k)$, $v_k := v(t_k)$. To apply A^* to these models, we restrict the input sets $\mathcal{V}$ and $\mathcal{A}$ to be finite.

VII. Experimental Results

This section presents an evaluation of Alg. 4 using a fully actuated robot (8) in simulation and a car-like robot (Fig. 4) with Ackermann-drive dynamics (9) in real experiments. We use an RBF kernel with parameters $\eta = 1$, $\gamma = 2.5$ and an R^*-tree approximation of the score $F(\mathbf{x})$ with $K^+ + K^- = 200$ nearest support vectors around the robot location $\mathbf{s}_k$ for map updating. The online training data (Sec. IV) were generated from a grid with resolution $0.25m$. Timing results are reported from an Intel i7 2.2 GHz CPU with 16GB RAM.

A. Simulations

The accuracy and memory consumption of our sparse kernel-based map was compared with OctoMap [4] in a warehouse environment shown in Fig. 3. As the ground-truth map represents the work space instead of C-space, a point robot ($r = 0$) was used for an accurate comparison. Lidar scan measurements were simulated along a robot trajectory shown in Fig. 3 and used to build our map and OctoMap simultaneously. OctoMap's resolution was also set to $0.25m$ to match that of grid used to sample our training data from. Furthermore, since our map does not provide occupancy probability, OctoMap's binary map was used as the baseline.

Algorithm 4 Autonomous Mapping and Navigation with a Sparse Kernel-based Occupancy Map

Input: Initial position $\mathbf{s}_0 \in \mathcal{C}_{free}$; goal region $\mathcal{C}_{goal}$; time limit T; prior support vectors Λ_0^+ and Λ_0^-, $t = 0$

1: **while** $\mathbf{s}_t \notin \mathcal{C}_{goal}$ and $t < T$ **do**
2: $\quad\mathbf{z}_t \leftarrow$ New Depth Sensor Observation
3: $\quad\mathcal{D} \leftarrow$ Training Data Generation$(\mathbf{z}_t, \Lambda_t^+, \Lambda_t^-)$ ▷ Sec. IV
4: $\quad\Lambda_{t+1}^+, \Lambda_{t+1}^- \leftarrow$ Incremental Fastron$(\Lambda_t^+, \Lambda_t^-, \mathcal{D}, \mathbf{s}_t)$ ▷ Alg. 1
5: $\quad$Path Planning$(\Lambda_{t+1}^+, \Lambda_{t+1}^-, \mathbf{s}_t, \mathcal{C}_{\text{goal}})$ ▷ Replan via A^* (Alg. 2&3)
6: $\quad\mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t)$ ▷ Move to the first position along the path

	KM	KM-SA	IM	IM-SA	OM
Accuracy	98.5%	98.5%	83.8%	83.2%	96.1%
Recall	97.4%	97.3%	99.0%	98.9%	96.8%
Vectors/Nodes	1947	2721	1947	2721	12372
Storage	15.6kB	21.7kB	15.6kB	21.7kB	24.7kB

TABLE I: Comparison among our kernel-based map (KM), KM map with score approximation (KM-SA), our inflated map (IM), IM map with score approximation (IM-SA) and OctoMap (OM) [4].

Table I compares the accuracy and the memory consumption of OctoMap's binary map versus our kernel-based map and inflated map (using the upper bound in Prop. 1) with and without score approximation (Sec. IV). The kernel-based and inflated maps (with score approximation) are shown in Fig. 3. The kernel-based maps and OctoMap's binary map lead to similar accuracy of $\sim 96 - 98\%$ compared to the ground truth map. OctoMap required a compressed octree with 12372 non-leaf nodes with 2 bytes per node, leading to a memory requirement of $\sim 24.7kB$. As the memory consumption depends on the computer architecture and how the information on the support vectors is compressed, we provide only a rough estimate to show that our map's memory requirements are at least comparable to those of OctoMap. We stored an integer representing a support vector's location on the underlying grid and a float representing its weight. This requires 8 bytes on a 32-bit architecture per support vector. Our maps contained 1947 and 2721 support vectors with and without score approximation, leading to memory requirements of $15.6kB$ and $21.7kB$, respectively. The recall (true positive rate) reported in Table I demonstrates the safety guarantee provided by our inflated map as $\sim 99\%$ of the occupied cells are correctly classified.

We also compared the average collision checking time over 1000000 random line segments using our complete method (Alg. 2 with Eq. (5) and $K^+ = K^- = 10$ for score approximation) and sampling-based methods with different sampling resolutions using the ground truth map. Fig. 6 shows that the time for sampling-based collision checking increased as the line length increased or the sampling resolution decreased. Meanwhile, our method's time was stable at $\sim 15\mu s$ suggesting its suitability for real-time applications.

B. Real Robot Experiments

Real experiments were carried out on an $1/10$th scale Racecar robot (Fig. 4) equipped with a Hokuyo UST-10LX

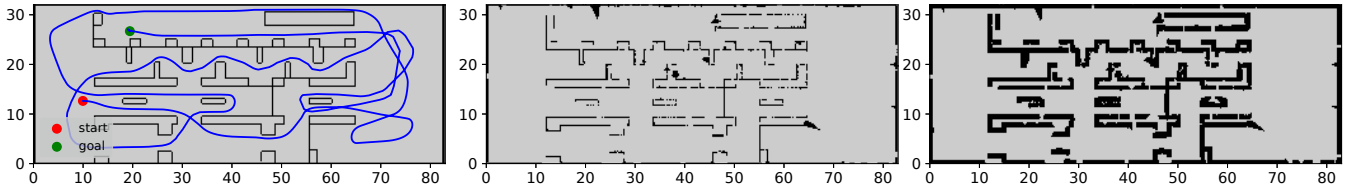

Fig. 3: Ground truth map and robot trajectory (left) used to generate simulated lidar scans, the occupancy map generated from our sparse kernel-based representation (middle) and the inflated map from the upper bound $U(\mathbf{x})$ proposed in Prop. 1 (right).

Fig. 4: Third person views of the autonomous Racecar robot navigating in an unknown environment with moving obstacles.

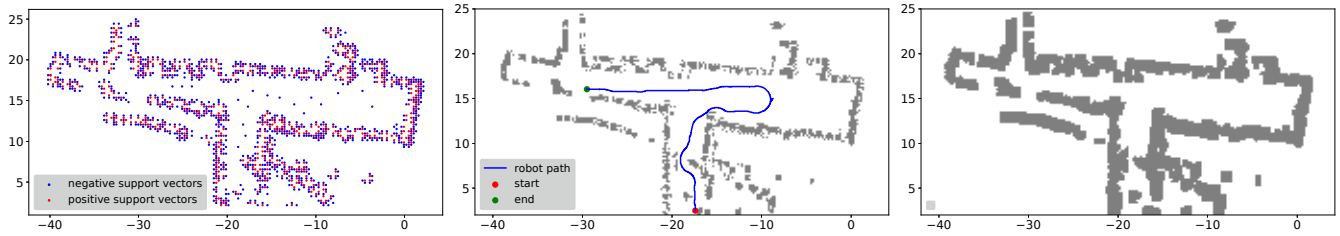

Fig. 5: Final 1762 support vectors (left), kernel-based map (middle), and inflated map (right) obtained from the real experiments.

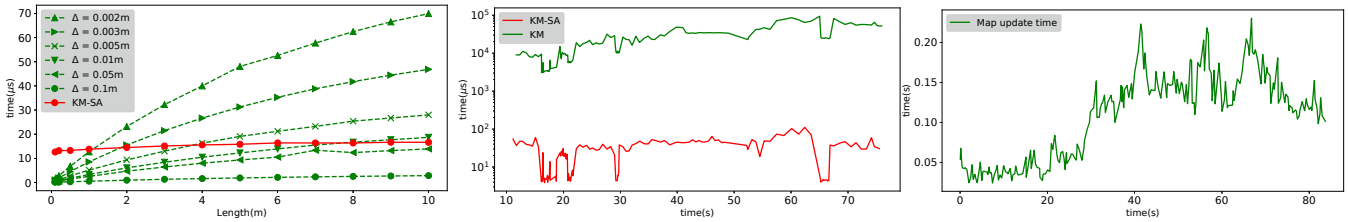

Fig. 6: Checking line segments in simulation (left): comparison between our method with score approximation (KM-SA) and a sampling-based method with different sampling resolution Δ. Planning time per motion primitive in the real experiment (middle): comparison between our method with (KM) and without score approximation (KM-SA). Map update time for the real experiment (right).

Lidar and Nvidia TX2 computer. The robot body was modeled by a ball of radius $r = 0.25m$. Second-order polynomial motion primitives were generated with time discretization of $\tau = 1$s as described in Sec. VI. The motion cost was defined as $c(\mathbf{s}, \mathbf{a}) := (\|\mathbf{a}\|^2 + 2)\tau$ to encourage both smooth and fast motion [28]. Alg. 3 with Eq. (7), $\varepsilon = 0.2$, and score approximation with $K^+ = K^- = 2$ was used for collision checking. The trajectory generated by an A^* motion planner was tracked using a closed-loop controller [32]. The robot navigated in an unknown hallway with moving obstacles to destinations randomly chosen by a human operator. Fig. 5 shows the support vectors, the kernel-based map, and the inflated map with score approximation for the experiment.

We observed that kernel-based mapping is susceptible to noise since the support vectors are quickly updated with newly observed data, even though it is noisy and affected by localization errors. This is caused by the kernel perceptron model not maintaining occupancy probabilities. Future work will focus on sparse generative models for occupancy maps.

The map update time taken by Alg. 1 from one lidar scan and the A^* replanning time per motion primitive with and without score approximation are shown in Fig. 6. Map updates implemented in Python took 0.11s on average, and increased as the car moved into unseen regions. To evaluate collision checking time, the A^* replanning time was normalized by the number of motion primitives being checked to account for differences in planning to nearby and far goals. Without score approximation, the planning time per motion primitive was in the order of milliseconds and increased over time as more support vectors were added. With score approximation, it was stable at $\sim 40\mu s$ illustrating the benefits of our R^*-tree data structure.

VIII. Conclusion

This paper proposes a sparse kernel-based mapping method for a robot navigating in unknown environments. The method offers efficient map storage that scales with obstacle complexity rather than environment size. We developed efficient and complete collision checking for linear and polynomial trajectories in this new map representation. The experiments show the potential of our approach to provide compressed, yet accurate, occupancy representations of large environments. The mapping and collision checking algorithms offer a promising avenue for safe, real-time, long-term autonomous navigation in unpredictable and rapidly changing environments. Future work will explore simultaneous localization and mapping, sparse probablistic models for mapping, active exploration and map uncertainty reduction.

REFERENCES

[1] J. Guzzi, A. Giusti, L. M. Gambardella, G. Theraulaz, and G. A. D. Caro, "Human-friendly Robot Navigation in Dynamic Environments," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2013, pp. 423–430.

[2] L. Janson, T. Hu, and M. Pavone, "Safe Motion Planning in Unknown Environments: Optimality Benchmarks and Tractable Policies," in *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, June 2018.

[3] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. MIT press, 2005.

[4] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, 2013.

[5] "IEEE standard for robot map data representation for navigation," *1873-2015 IEEE Standard for Robot Map Data Representation for Navigation*, pp. 1–54, Oct 2015.

[6] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, and J. Nieto, "Voxblox: Incremental 3D Euclidean Signed Distance Fields for On-Board MAV Planning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[7] J. Behley and C. Stachniss, "Efficient Surfel-Based SLAM using 3D Laser Range Data in Urban Environments," in *Robotics: Science and Systems (RSS)*, 2018.

[8] M. Kaess, "Simultaneous localization and mapping with infinite planes," in *IEEE Int. Conference on Robotics and Automation (ICRA)*, 2015, pp. 4605–4611.

[9] S. Bowman, N. Atanasov, K. Daniilidis, and G. Pappas, "Probabilistic data association for semantic slam," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2017.

[10] S. T. O'Callaghan and F. T. Ramos, "Gaussian process occupancy maps," *The International Journal of Robotics Research (IJRR)*, vol. 31, no. 1, pp. 42–62, 2012.

[11] J. Wang and B. Englot, "Fast, accurate gaussian process occupancy maps via test-data octrees and nested bayesian fusion," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 1003–1010.

[12] M. G. Jadidi, J. V. Miro, and G. Dissanayake, "Warped gaussian processes occupancy mapping with uncertain inputs," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 680–687, 2017.

[13] F. Ramos and L. Ott, "Hilbert maps: Scalable continuous occupancy mapping with stochastic gradient descent," *The International Journal of Robotics Research*, vol. 35, no. 14, pp. 1717–1730, 2016.

[14] B. T. Lopez and J. P. How, "Aggressive 3-d collision avoidance for high-speed navigation," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 5759–5765.

[15] J. Chen and S. Shen, "Improving octree-based occupancy maps using environment sparsity with application to aerial robot navigation," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 3656–3663.

[16] D. Fridovich-Keil, E. Nelson, and A. Zakhor, "Atommap: A probabilistic amorphous 3d map representation for robotics and surface reconstruction," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 3110–3117.

[17] J. Bialkowski, M. Otte, S. Karaman, and E. Frazzoli, "Efficient collision checking in sampling-based motion planning via safety certificates," *The International Journal of Robotics Research*, vol. 35, no. 7, pp. 767–796, 2016.

[18] J. Luo and K. Hauser, "An empirical study of optimal motion planning," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2014, pp. 1761–1768.

[19] K. Hauser, "Lazy collision checking in asymptotically-optimal motion planning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

[20] E. G. Tsardoulias, A. Iliakopoulou, A. Kargakos, and L. Petrou, "A review of global path planning methods for occupancy grid maps regardless of obstacle density," *Journal of Intelligent & Robotic Systems*, vol. 84, no. 1, pp. 829–858, 2016.

[21] N. Das, N. Gupta, and M. Yip, "Fastron: An Online Learning-Based Model and Active Learning Strategy for Proxy Collision Detection," in *Conference on Robot Learning (CoRL)*, 2017, pp. 496–504.

[22] J. Pan and D. Manocha, "Efficient configuration space construction and optimization for motion planning," *Engineering*, vol. 1, no. 1, pp. 046–057, 2015.

[23] J. Huh and D. D. Lee, "Learning high-dimensional Mixture Models for fast collision detection in Rapidly-Exploring Random Trees," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 63–69.

[24] O. Arslan and P. Tsiotras, "Machine learning guided exploration for sampling-based motion planning algorithms," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2015.

[25] N. Das and M. Yip, "Learning-based proxy collision detection for robot motion planning applications," *arXiv preprint arXiv:1902.08164*, 2019.

[26] J. Pan, S. Chitta, and D. Manocha, "Fcl: A general purpose library for collision and proximity queries," in *IEEE International Conference on Robotics and Automation*. IEEE, 2012, pp. 3859–3866.

[27] R. M. Neal, *Bayesian learning for neural networks*. Springer Science & Business Media, 2012, vol. 118.

[28] S. Liu, N. Atanasov, K. Mohta, and V. Kumar, "Search-based motion planning for quadrotors using linear quadratic minimum time control," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 2872–2879.

[29] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, 2009.

[30] A. De Luca, G. Oriolo, and M. Vendittelli, "Stabilization of the unicycle via dynamic feedback linearization," *IFAC Proceedings Volumes*, vol. 33, no. 27, pp. 687–692, 2000.

[31] J. Franch and J. Rodriguez-Fortun, "Control and trajectory generation of an ackerman vehicle by dynamic linearization," in *European Control Conference (ECC)*, 2009, pp. 4937–4942.

[32] O. Arslan and D. E. Koditschek, "Exact robot navigation using power diagrams," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 1–8.