Robotics: Science and Systems 2024
Delft, Netherlands, July 15-July 19, 2024

Safe Planning for Articulated Robots Using Reachability-based Obstacle Avoidance With Spheres

Jonathan Michaux[1], Adam Li[1], Qingyi Chen[1], Che Chen[1], Bohao Zhang[1], and Ram Vasudevan[1]

***Abstract*—Generating safe motion plans in real-time is necessary for the wide-scale deployment of robots in unstructured and human-centric environments. These motion plans must be safe to ensure humans are not harmed and nearby objects are not damaged. However, they must also be generated in real-time to ensure the robot can quickly adapt to changes in the environment. Many trajectory optimization methods introduce heuristics that trade-off safety and real-time performance, which can lead to potentially unsafe plans. This paper addresses this challenge by proposing Safe Planning for Articulated Robots Using Reachability-based Obstacle Avoidance With Spheres (SPARROWS). SPARROWS is a receding-horizon trajectory planner that utilizes the combination of a novel reachable set representation and an exact signed distance function to generate provably-safe motion plans. At runtime, SPARROWS uses parameterized trajectories to compute reachable sets composed entirely of spheres that overapproximate the swept volume of the robot's motion. SPARROWS then performs trajectory optimization to select a safe trajectory that is guaranteed to be collision-free. We demonstrate that SPARROWS' novel reachable set is significantly less conservative than previous approaches. We also demonstrate that SPARROWS outperforms a variety of state-of-the-art methods in solving challenging motion planning tasks in cluttered environments. Code, data, and video demonstrations can be found at https://roahmlab.github.io/sparrows/.**

I. Introduction

Robot manipulators have the potential to make large, positive impacts on society and improve human lives. These potential impacts range from replacing humans performing dangerous and difficult tasks in manufacturing and construction to assisting humans with more delicate tasks such as surgery or in-home care. In each of these settings, the robot must remain safe at all times to prevent harming nearby humans, colliding with obstacles, or damaging high-value objects. It is also essential that the robot generate motion plans in real-time so as to perform any given task efficiently or to quickly adjust their behavior to react to changes in the environment.

Modern model-based motion planning frameworks typically consist of a high-level planner, a mid-level trajectory planner, and a low-level tracking controller. The high-level planner generates a path consisting of a sequence of discrete waypoints between the robot's start and goal configurations. The mid-level trajectory planner calculates velocities and accelerations at specific time intervals that move the robot from one waypoint to the next. The low-level tracking controller generates control inputs that attempt to minimize deviations from the desired trajectory. For example, one may use a high-level sampling-based planner such as a Probabilistic Road Map [1] to generate discrete waypoints for a trajectory optimization algorithm such as CHOMP [2] or TrajOpt [3], and track the resulting trajectories with an appropriately designed low-level inverse dynamics controller [4]. While variations of this framework have been demonstrated to work on various robotic platforms, there are several limitations that prevent wide-scale deployment in the real-world. For instance, this approach can become computationally demanding as the complexity of the robot or environment increases, making it less practical for real-time applications. Many algorithms also introduce heuristics such as reducing the number of collision checks to achieve real-time performance at the expense of robot safety. Furthermore, it is often assumed that the robot's dynamics are fully known, while in reality there can be considerable uncertainty. Unfortunately, both of these notions increase the potential for the robot to collide with obstacles.

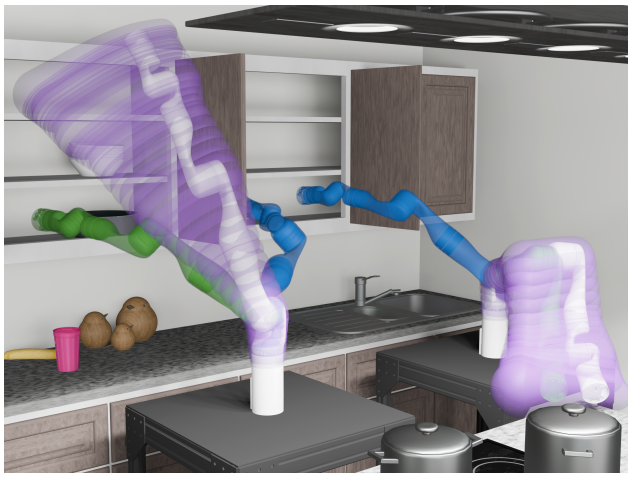

Fig. 1: This paper presents SPARROWS, a method that is capable of generating safe motion plans in dense and cluttered environments for single- and multi-arm robots. Here, both arms have start and goal configurations shown in blue and green, respectively. Prior to planning, SPARROWS is given full access to polytope overapproximations of obstacles to compute an exact signed distance function of the scene geometry. At runtime, SPARROWS combines the signed distance function with the novel Spherical Forward Occupancy as obstacle-avoidance constraints for SPARROWS to generate safe trajectories between the start and goal in a receding horizon manner. Each trajectory is selected by solving a nonlinear optimization problem such that an overapproximation (purple) of the swept volume of its entire motion remains collision-free. Note in this figure, a small, thin barrier is placed to prevent unwanted collisions between the two arms. A video demonstration can be found at https://roahmlab.github.io/sparrows/.

Reachability-based Trajectory Design (RTD) [5] is a recent example of a hierarchical planning framework that uses a mid-

This work is supported by the National Science Foundation Career Award 1751093 and by the Air Force Office of Scientific Research under award 23-S15. JM is supported by the University of Michigan Rackham Merit Fellowship.

[1]Robotics Institute, University of Michigan, Ann Arbor, MI ⟨jmichaux, adamli, chenqy, cctom, jimzhang, ramv⟩@umich.edu.

level trajectory planner to enforce safety. RTD combines zonotope arithmetic [6] with classical recursive robotics algorithms [7] to iteratively compose reachable sets for high dimensional systems such as robot manipulators [8]. During planning, reachable sets are constructed at runtime and used to enforce obstacle avoidance in continuous-time. In addition to generating collision-free trajectories, RTD has been demonstrated to select trajectories that are guaranteed to be dynamically feasible even in the presence of uncertainty [9]. Notably, RTD is able to accomplish provably-safe planning in real-time. However, as we show here, the RTD reachable set formulation for manipulators [8]–[10] tends to be overly conservative. This work proposes a novel reachable set formulation composed entirely of spheres that is tighter than that of RTD. We demonstrate that this novel reachable set formulation enables planning in extremely cluttered environments, whereas RTD fails to find feasible paths, largely due to the size of its reachable sets.

A. Related Work

The purpose of a safe motion planning algorithm is to ensure that a robot can move from one configuration to another while avoiding collisions with all obstacles in its environment for all time. Ideally, one would ensure that the entire swept volume [11]–[15] of a moving robot does not intersect with any objects nor enter any unwanted regions of its workspace. Although swept volume computation has long been used for collision detection during motion planning [16], computing the exact swept volumes of articulated robots such as serial manipulators is analytically intractable [17]. Instead, many algorithms rely on approximations using CAD models [18]–[20], occupancy grids, and convex polyhedra. However, these methods often suffer from high computational costs and are generally not suitable when generating complex robot motions and are therefore most useful when applied while motion planning offline [21]. Furthermore, approximate swept volume algorithms may be overly conservative [20], [22] and thus limit the efficiency of finding a feasible path in cluttered environments. To address some of these limitations, [23] used a precomputed swept volume to perform a parallelized collision check at run-time. Unfortunately, these precomputed swept volumes need to be recomputed whenever the robot grasps an object.

An alternative to computing swept volumes for obstacle avoidance is to model the robot, or the environment, with simple geometric primitives such as spheres [24], [25], ellipsoids [26], capsules [27], [28], or convex polygons and perform collision-checking along a given trajectory at discrete time instances. This is common for state-of-the-art trajectory optimization-based approaches such as CHOMP [2], TrajOpt [3], MPOT [29], and cuRobo [30]. CHOMP represents the robot as a collection of discrete spheres and maintains a safety margin with the environment by utilizing a signed distance field. TrajOpt represents the robot and obstacles as the support mapping of convex shapes. Then, the distance and penetration depth between two convex shapes is computed by the Gilbert-Johnson-Keerthi [31] and Expanding Polytope Algorithms [32], respectively. MPOT represents the robot geometry and obstacles as spheres and implements a collision cost using an occupancy map [29]. cuRobo represents the robot as a collection of spheres and solves multiple trajectory optimization problems that avoid colliding with the environment in parallel in GPU. Although these approaches have been demonstrated to solve challenging motion planning tasks in real-time, the resulting trajectories cannot be considered safe as collision avoidance is only enforced as a soft penalty in the cost function.

Reachability-based Trajectory Design (RTD) [5] is a recent approach to real-time motion planning that generates provably safe trajectories in a receding-horizon fashion. At runtime, RTD uses (polynomial) zonotopes [33] to construct reachable sets that overapproximate all possible robot positions corresponding to a pre-specified continuum of parameterized trajectories. RTD then solves a nonlinear optimization problem to select a feasible trajectory such that the swept volume corresponding to that motion is guaranteed to be collision-free. If a feasible trajectory is not found, RTD executes a braking maneuver that brings the robot safely to a stop. Importantly, the reachable sets are constructed such that obstacle-avoidance constraints are satisfied in continuous-time. However, while extensions of RTD have demonstrated real-time, certifiably-safe motion planning for robotic arms [8]–[10] with seven degrees of freedom, RTD's reachable sets tend to be overly conservative as we demonstrate in this paper.

B. Contributions

To address the limitations of existing approaches, this paper proposes Safe Planning for Articulated Robots Using Reachability-based Obstacle Avoidance With Spheres (SPARROWS). The proposed method combines reachability analysis with sphere-based collision primitives and an exact signed distance function to enable real-time motion planning that is certifiably-safe, yet less conservative than previous methods. This paper's contributions are three-fold:

I. A novel reachable set representation composed of overlapping spheres, called the Spherical Forward Occupancy ($\mathcal{SFO}$), that overapproximates the robot's reachable set and is differentiable;
II. An algorithm that computes the exact signed distance between a point and a three dimensional zonotope;
III. A demonstration that SPARROWS outperforms similar state-of-the-art methods on a set of challenging motion planning tasks.

The remainder of this manuscript is organized as follows: Section II summarizes the set representations used throughout the paper and gives a brief overview of signed distance functions; Section III describes how the robot arm and environment are modeled; Section IV describes the formulation of the safe motion planning problem using the Spherical Forward Occupancy and an exact signed distance function; Section V summarizes the evaluation of the proposed method on a variety of different example problems.

TABLE I: Summary of polynomial zonotope operations.

Operation	Computation
$\mathbf{P}_1 \oplus \mathbf{P}_2$ (Minkowski Sum) [9, eq. (10)]	Exact
$\mathbf{P}_1\mathbf{P}_2$ (PZ Multiplication) [9, eq. (11)]	Exact
$\texttt{slice}(\mathbf{P}, x_j, \sigma)$ (4)	Exact
$\texttt{sup}(\mathbf{P})$ (5) and $\texttt{inf}(\mathbf{P})$(6)	Overapproximative

II. PRELIMINARIES

This section establishes the notation used throughout the paper and describes the set-based representations and operations used throughout this document.

A. Notation

Sets and subspaces are typeset using capital letters. Subscripts are primarily used as an index or to describe a particular coordinate of a vector. Let $\mathbb{R}$ and $\mathbb{N}$ denote the spaces of real numbers and natural numbers, respectively. The Minkowski sum between two sets $\mathcal{A}$ and $\mathcal{A}'$ is $\mathcal{A} \oplus \mathcal{A}' = \{a + a' \mid a \in \mathcal{A},\ a' \in \mathcal{A}'\}$. Let $\hat{e}_l \in \mathbb{R}^n$ denote the l^{th} unit vector in the standard orthogonal basis. Given vectors α, let $[\alpha]_i$ denote the i-th element of α. Given $\alpha \in \mathbb{R}^n$ and $\varepsilon > 0$, let $B(\alpha, \varepsilon)$ denote the n-dimensional closed ball with center α and radius ε under the Euclidean norm. Given vectors $\alpha, \beta \in \mathbb{R}^n$, let $\alpha \odot \beta$ denote element-wise multiplication. Given a set $\Omega \subset \mathbb{R}^{n_d}$, let $\partial\Omega \subset \mathbb{R}^{n_d}$ be its boundary, $\Omega^c \subset \mathbb{R}^{n_d}$ denote its complement, and $co(\Omega)$ denote its convex hull.

B. Polynomial Zonotopes

We present a brief overview of the defintions and operations on polynomial zonotopes (PZs) that are used throughout the remainder of this document. A thorough introduction to polynomial zonotopes is available in [33].

Given a *center* $c \in \mathbb{R}^n$, $n_g \in \mathbb{N}$ *dependent generators* $g_i \in \mathbb{R}^n$, $n_h \in \mathbb{N}$ *independent generators* $h_j \in \mathbb{R}^n$, and *exponents* $\alpha_i \in \mathbb{N}_i^{n_g}$ for $i \in \{0, \ldots, n_g\}$, a *polynomial zonotope* is a set:

$$\mathbf{P} = \Big\{ z \in \mathbb{R}^n \mid z = c + \sum_{i=1}^{n_g} g_i x^{\alpha_i} + \sum_{j=1}^{n_h} h_j y_j, x \in [-1,1]^{n_g}, \quad (1)$$

$$y_j \in [-1,1] \Big\}. \quad (2)$$

For each $i \in \{1, \ldots, n_g\}$, we refer to x^{α_i} as a *monomial*. We refer to $x \in [-1,1]^{n_g}$ and $y_j \in [-1,1]$ as *indeterminates* for the dependent and independent generators, respectively. Note that one can define a matrix polynomial zonotope by replacing the vectors above with matrices of compatible dimensions.

Throughout this document, we exclusively use bold symbols to denote polynomial zonotopes. We also introduce the shorthand $\mathbf{P} = \mathcal{PZ}(c, g_i, \alpha_i, x, h_j, y_j)$ when we need to emphasize the generators and exponents of a polynomial zonotope. Note that zonotopes are a special subclass of polynomial zonotopes whose dependent generators are all zero [33].

C. Polynomial Zonotope Operations

There exists a variety of operations that one can perform on polynomial zonotopes such as minksowski sums, multiplying two or more polynomial zonotopes, and generating upper and lower bounds. As illustrated in Tab. I, the output of these operations is a polynomial zonotope that either exactly represents or over approximates the result of the operation on each individual element of the polynomial zonotope inputs. For the interested reader, the operations given in Tab. I are rigorously defined in [9].

1) Interval Conversion: Intervals can be represented as polynomial zonotopes. Let $[z] = [\underline{z}, \overline{z}] \subset \mathbb{R}^n$, then $[z]$ can be converted to a polynomial zonotope $\mathbf{z}$ using

$$\mathbf{z} = \frac{\overline{z} + \underline{z}}{2} + \sum_{i=1}^{n} \frac{\overline{z}_i - \underline{z}_i}{2} x_i, \quad (3)$$

where $x \in [-1,1]^n$ is the indeterminate vector.

2) Slicing a PZ: One particularly useful property of polynomial zonotopes is the ability to obtain various subsets by plugging in different values of known indeterminates. To this end, we introduce the "slicing" operation whereby a new subset is obtained from a polynomial zonotope $\mathbf{P} = \mathcal{PZ}(c, g_i, \alpha_i, x, h_j, y_j)$ is obtained by evaluating one or more indeterminates. Thus, given the j^{th} indeterminate x_j and a value $\sigma \in [-1,1]$, slicing yields a subset of $\mathbf{P}$ by plugging σ into the specified element x_j:

$$\texttt{slice}(\mathbf{P}, x_j, \sigma) \subset \mathbf{P} = \left\{ z \in \mathbf{P} \mid z = \sum_{i=0}^{n_g} g_i x^{\alpha_i}, x_j = \sigma \right\}. \quad (4)$$

3) Bounding a PZ: In later sections, we also require operations to bound the elements of a polynomial zonotope. We define the `sup` and `inf` operations, which return these upper and lower bounds, by setting the monomials $x^{\alpha_i} = 1$ and adding or subtracting all of the generators to the center, respectively. For $\mathbf{P} \subseteq \mathbb{R}^n$, these operations return

$$\texttt{sup}(\mathbf{P}) = g_0 + \sum_{i=1}^{n_g} |g_i| + \sum_{j=1}^{n_h} |h_j|, \quad (5)$$

$$\texttt{inf}(\mathbf{P}) = g_0 - \sum_{i=1}^{n_g} |g_i| - \sum_{j=1}^{n_h} |h_j|. \quad (6)$$

Note that these operations efficiently generate upper and lower bounds on the values of a polynomial zonotope through overapproximation.

D. Distance Functions and Projections onto Sets

The method we propose utilizes a signed distance function to compute the distance between the robot and obstacles:

Definition 1. *Given a subset Ω of $\mathbb{R}^{n_d}$, the* signed distance function *s_d between a point is a map $s_d : \mathbb{R}^{n_d} \to \mathbb{R}$ defined as*

$$s_d(x; \Omega) = \begin{cases} d(x; \partial\Omega) & \text{if } x \in \Omega^c \\ -d(x; \partial\Omega) & \text{if } x \in \Omega, \end{cases} \quad (7)$$

where $d(x; \Omega)$ is the distance function *associated with Ω and defined by:*

$$d(x; \Omega) = \min_{y \in \partial\Omega} \|x - y\|. \quad (8)$$

The exact signed distance function algorithm developed in Sec. IV-C utilizes Euclidean projections onto sets.

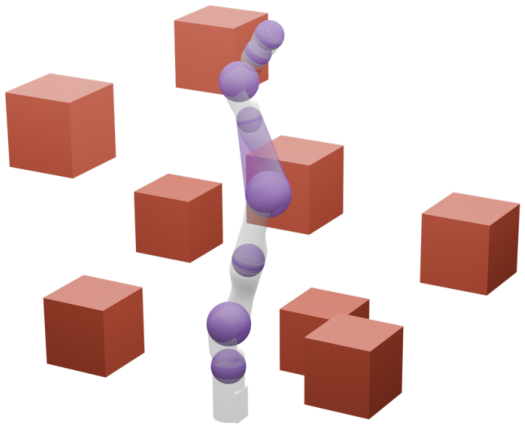

Fig. 2: A visualization of the robot arm and its environment. The obstacles are shown in red and the robot is shown in grey (translucent). The volume of each joint is overapproximated by a sphere, shown in purple, in the workspace. Each link volume is overapproximated by a tapered capsule formed by the convex hull shown in light purple of two consecutive joint spheres (Assum. 5).

Definition 2. *For any $x \in \mathbb{R}^{n_d}$, the* Euclidean projection ϕ *from c to $\Omega \subset \mathbb{R}^{n_d}$ is defined as*

$$\phi(x;\Omega) = \{\omega \in \Omega \mid \|x-\omega\| = d(x;\Omega)\}. \tag{9}$$

III. Arm and Environment Modeling

This section summarizes the environment, obstacles, and robot models used throughout the remainder of the paper.

A. Environment

The arm must avoid obstacles in the environment while performing safe motion planning. We begin by stating an assumption about the environment and its obstacles:

Assumption 3. *The robot and obstacles are all located in a fixed inertial world reference frame, which we denote $W \subset \mathbb{R}^3$. We assume the origin of W is known and that each obstacle is represented relative to the origin of W. At any time, the number of obstacles $n \in \mathbb{N}$ in the world is finite. Let $\mathcal{O} = \{O_1, O_2, \ldots, O_n\}$ be the set of all obstacles. Each obstacle is convex, bounded, and static with respect to time. For each obstacle, we have access to a zonotope that overapproximates the obstacle's volume. See Fig. 2.*

Note that any convex, bounded object can always be over-approximated by a zonotope [6]. In addition, any non-convex bounded obstacle can be outerapproximated by its convex hull. For the remainder of this document, safety is checked using the zonotope overapproximation of each obstacle.

B. Arm Kinematics

This work considers an n_q degree of freedom serial robotic manipulator with configuration space Q. Given a compact time interval $T \subset \mathbb{R}$, we define a trajectory for the configuration as $q : T \to Q \subset \mathbb{R}^{n_q}$ and a trajectory for the velocity as $\dot{q} : T \to \mathbb{R}^{n_q}$. We make the following assumptions about the structure of the robot model:

Assumption 4. *The robot operates in a three dimensional workspace, denoted $W_s \subset \mathbb{R}^3$, such that $W_s \subset W$. There exists a reference frame called the base frame, which we denote the 0^{th} frame, that indicates the origin of the robot's kinematic chain. The transformation between the robot's base frame and the origin of the world frame is known. The robot is fully actuated and composed of only revolute joints, where the j^{th} joint actuates the robot's j^{th} link. The robot's j^{th} joint has position and velocity limits given by $q_j(t) \in [q^-_{j,\text{lim}}, q^+_{j,\text{lim}}]$ and $\dot{q}_j(t) \in [\dot{q}^-_{j,\text{lim}}, \dot{q}^+_{j,\text{lim}}]$ for all $t \in T$, respectively. The robot's input is given by $u : T \to \mathbb{R}^{n_q}$.*

Recall from Assum. 4 that the transformation from *world* from to the the robot's 0^{th}, or *base*, frame is known. We further assume that the j^{th} reference frame $\{\hat{x}_j, \hat{y}_j, \hat{z}_j\}$ is attached to the robot's j^{th} revolute joint, and that $\hat{z}_j = [0,0,1]^\top$ corresponds to the j^{th} joint's axis of rotation.

Then the forward kinematics $\text{FK}_j : Q \to \mathbb{R}^{4\times 4}$ maps a time-dependent configuration, $q(t)$, to the pose of the j^{th} joint in the world frame:

$$\text{FK}_j(q(t)) = \prod_{l=1}^{j} H_l^{l-1}(q_l(t)) = \begin{bmatrix} R_j(q(t)) & p_j(q(t)) \\ 0 & 1 \end{bmatrix}, \tag{10}$$

where

$$R_j(q(t)) := R_j^0(q(t)) = \prod_{l=1}^{j} R_l^{l-1}(q_l(t)) \tag{11}$$

and

$$p_j(q(t)) = \sum_{l=1}^{j} R_l(q(t)) p_l^{l-1}. \tag{12}$$

Note that we use the revolute joint portion of Assum. 4 to simplify the description of forward kinematics; however, these assumptions and the methods described in the following sections can be readily be extended to other joint types such as prismatic or spherical joints in a straightforward fashion. Note that the lack of input constraints means that one could apply an inverse dynamics controller [4] to track any trajectory of the robot perfectly. It is also possible to deal with the tracking error due to uncertainty in the robot's dynamics [9]. However, this is outside of the scope of the current work and as a result, we focus exclusively on modeling the kinematic behavior of the manipulator.

C. Arm Occupancy

Next, we use the arm's kinematics to define the *forward occupancy* which is the volume occupied by the arm in the workspace. Let $L_j \subset \mathbb{R}^3$ denote the volume occupied by the j^{th} link with respect to the j^{th} reference frame. Then the forward occupancy of link j is the map $\text{FO}_j : Q \to \mathcal{P}(W_s)$ defined as

$$\text{FO}_j(q(t)) = p_j(q(t)) \oplus R_j(q(t)) L_j, \tag{13}$$

where $p_j(q(t))$ the position of joint j and $R_j(q(t))L_j$ is the rotated volume of link j. The volume occupied by the entire

Algorithm 1: $\{\mathrm{FK}_\mathbf{j}(\mathbf{q}(\mathbf{T_i};\mathbf{K})) : j \in N_q\} = \mathtt{PZFK}(\mathbf{q}(\mathbf{T_i};\mathbf{K}))$

$\mathbf{p_0}(\mathbf{q}(\mathbf{T_i};\mathbf{K})) \leftarrow 0$
$\mathbf{R_0}(\mathbf{q}(\mathbf{T_i};\mathbf{K})) \leftarrow I_{3\times 3}$
for $j = 1 : n_q$
  $\mathbf{R_j^{j-1}}(\mathbf{q_j}(\mathbf{T_i};\mathbf{K})), \mathbf{p_j}(\mathbf{q}(\mathbf{T_i};\mathbf{K})) \leftarrow \mathbf{H_j^{j-1}}(\mathbf{q_j}(\mathbf{T_i};\mathbf{K}))$
  $\mathbf{p_j}(\mathbf{q}(\mathbf{T_i};\mathbf{K})) \leftarrow \mathbf{p_{j-1}}(\mathbf{q}(\mathbf{T_i};\mathbf{K})) \oplus \mathbf{R_{j-1}}(\mathbf{q}(\mathbf{T_i};\mathbf{K}))\mathbf{p_j^{j-1}}$
  $\mathbf{R_j}(\mathbf{q}(\mathbf{T_i};\mathbf{K})) \leftarrow \mathbf{R_{j-1}}(\mathbf{q}(\mathbf{T_i};\mathbf{K}))\mathbf{R_j^{j-1}}(\mathbf{q_j}(\mathbf{T_i};\mathbf{K}))$
  $\mathrm{FK}_\mathbf{j}(\mathbf{q}(\mathbf{T_i};\mathbf{K})) \leftarrow \{\mathbf{R_j}(\mathbf{q}(\mathbf{T_i};\mathbf{K})), \mathbf{p_j}(\mathbf{q}(\mathbf{T_i};\mathbf{K}))\}$
end for

arm in the workspace is given by the function $\mathrm{FO} : Q \to W_s$ that is defined as

$$\mathrm{FO}(q(t)) = \bigcup_{j=1}^{n_q} \mathrm{FO}_j(q(t)) \subset W_s. \tag{14}$$

Note that any of the robot's link volumes L_j may be arbitrarily complex. We therefore introduce an assumption to simplify the construction of the reachable set in Sec. IV-B:

Assumption 5. *For every $j \in \{1,\dots,n_q\}$ there exists a sphere of radius r_j with center $p_j(q(t))$ that overapproximates the volume occupied by the j^{th} joint in W_s. Furthermore, the link volume L_j is a subset of the tapered capsule formed by the convex hull of the spheres overapproximating the j^{th} and $j+1^{th}$ joints. See Fig. 2.*

Following Assum. 5, we define the sphere $S_j(q(t))$ overapproximating the j^{th} joint as

$$S_j(q(t)) = B(p_j(q(t)), r_j) \tag{15}$$

and the tapered capsule $TC_j(q(t))$ overapproximating the j^{th} link as

$$TC_j(q(t)) = co\Big(S_j(q(t)) \cup S_{j+1}(q(t))\Big). \tag{16}$$

Then, (13) and (14) can then be overapproximated by

$$\begin{aligned}\mathrm{FO}_j(q(t)) &\subset co\Big(S_j(q(t)) \cup S_{j+1}(q(t))\Big)\\ &= TC_j(q(t)),\end{aligned} \tag{17}$$

and

$$\mathrm{FO}(q(t)) \subset \bigcup_{j=1}^{n_q} TC_j(q(t)) \subset W_s, \tag{18}$$

respectively.

For convenience, we use the notation $\mathrm{FO}(q(T))$ to denote the forward occupancy over an entire time interval T. Note that we say that the arm is *in collision* with an obstacle if $\mathrm{FO}_j(q(t)) \cap O_\ell \neq \emptyset$ for any $j \in N_q$ or $\ell \in N_\mathscr{O}$ where $N_\mathscr{O} = \{1,\dots,n\}$.

D. Trajectory Design

SPARROWS computes safe trajectories by solving an optimization program over parameterized trajectories at each planning iteration in a receding-horizon manner. Offline, we pre-specify a continuum of trajectories over a compact set $K \subset \mathbb{R}^{n_k}$, $n_k \in \mathbb{N}$. Then each trajectory, defined over a compact time interval T, is uniquely determined by a *trajectory parameter* $k \in K$. Note that we design K for safe manipulator motion planning, but K may also be designed for other robot morphologies or to accommodate additional tasks [8], [34]–[36] as long as it satisfies the following properties:

Definition 6 (Trajectory Parameters)**.** *For each $k \in K$, a* parameterized trajectory *is an analytic function $q(\cdot\,;k) : T \to Q$ that satisfies the following properties:*

I. The parameterized trajectory starts at a specified initial condition $(q_0, \dot{q}_0)$, so that $q(0;k) = q_0$, and $\dot{q}(0;k) = \dot{q}_0$.
II. Each parameterized trajectory brakes to a stop such that $\dot{q}(t_f;k) = 0$.

SPARROWS performs real-time receding horizon planning by executing the desired trajectory computed at the previous planning iteration while constructing the next desired trajectory for the subsequent time interval. Therefore, the first property ensures each parameterized trajectory is generated online and begins from the appropriate future initial condition of the robot. The second property ensures that a braking maneuver is always available to bring the robot safely to a stop.

IV. Planning Algorithm Formulation

The goal of this work is to construct collision free trajectories in a receding-horizon fashion by solving the following nonlinear optimization problem at each planning iteration:

$$\min_{k\in K} \quad \mathtt{cost}(k) \tag{19}$$
$$q_j(t;k) \in [q^-_{j,\lim}, q^+_{j,\lim}] \qquad \forall t \in T, j \in N_q \tag{20}$$
$$\dot{q}_j(t;k) \in [\dot{q}^-_{j,\lim}, \dot{q}^+_{j,\lim}] \qquad \forall t \in T, j \in N_q \tag{21}$$
$$\mathrm{FO}_j(q(t;k)) \bigcap \mathscr{O} = \emptyset \qquad \forall t \in T, j \in N_q \tag{22}$$

The cost function (19) specifies a user- or task-defined objective, while each constraint guarantees the safety of any feasible parameterized trajectory. The first two constraints ensure that the trajectory does not violate the robot's joint position (20) and velocity limits (21). The last constraint (22) ensures that the robot does not collide with any obstacles in the environment.

Implementing a real-time algorithm to solve this optimization problem is challenging for two reasons. First, obstacle-avoidance constraints are typically non-convex [37]. Second, each constraint must be satisfied for all time t in a continuous, and uncountable, time interval T. To address these challenges, this section develops SPARROWS a novel, real-time trajectory planning method that combines reachability analysis with differentiable collision primitives. We first discuss how to overapproximate a family of parameterized trajectories that the robot follows. Next, we discuss how to overapproximate (18) using the Spherical Forward Occupancy ($\mathcal{SFO}$). Then, we discuss how to compute the signed distances between the forward occupancy and obstacles in the robot's environment. Finally, in Sec. IV-E we discuss how to combine the $\mathcal{SFO}$ and signed distance function to generate obstacle-avoidance

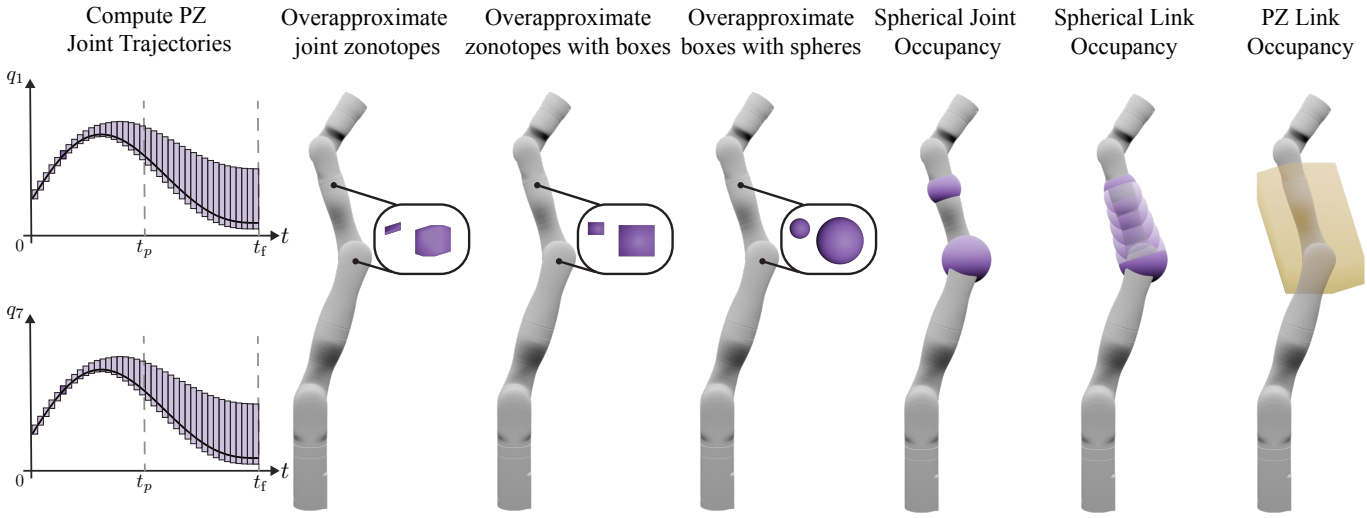

Fig. 3: A visualization of the Spherical Forward Occupancy construction for a robotic arm in 3D. (First column) The planning time horizon is partitioned into a finite set of polynomial zonotopes (Sec. IV-A1) before computing the parameterized trajectory polynomial zonotopes shown in purple (Sec. IV-A2). The time horizon consists of a planning phase $[0,t_p)$ and a braking phase $[t_p,t_f]$. A single desired trajectory for each joint is shown in black. (Second column) The polynomial zonotope forward kinematics algorithm (Sec. IV-A3) computes an overapproximation of the joint positions for each time interval. (Third column) The joint position zonotopes are first overapproximated by axis-aligned boxes and (Fourth column) subsequently overapproximated by the smallest sphere that circumscribes the box (Sec. IV-B1). For the sake of clarity, the insets in columns 2 through 4 depict the polynomial zonotope joint positions, overapproximating boxes, and the circumscribed spheres as larger. (Fifth column) The radii of the circumscribed spheres are then added to the radii of the nominal joint spheres (Assum. 5) to produce the Spherical Joint Occupancy (Lem. 9). (Sixth column) Finally, the Spherical Forward Occupancy is computed for each link corresponding to the desired trajectory. (Seventh column) Note that SPARROWS generates link occupancies that are less conservative than previous reachability methods [8], [9].

constraints for a real-time algorithm that can be used for safe online motion planning.

A. Polynomial Zonotope Trajectories and Forward Kinematics

This subsection summarizes the relevant results from [9] that are used to overapproximate parameterized trajectories (Def. 6) using polynomial zonotopes.

1) Time Horizon and Trajectory Parameter PZs: We first describe how to create polynomial zonotopes representing the planning time horizon T. We choose a timestep Δt so that $n_t := \frac{T}{\Delta t} \in \mathbb{N}$ divides the compact time horizon $T \subset \mathbb{R}$ into n_t time subintervals denoted by $N_t := \{1,\dots,n_t\}$. We then represent the i^{th} time subinterval, corresponding to $t \in [(i-1)\Delta t, i\Delta t]$, as a polynomial zonotope $\mathbf{T_i}$, where

$$\mathbf{T_i} = \left\{ t \in T \mid t = \tfrac{(i-1)+i}{2}\Delta t + \tfrac{1}{2}\Delta t x_{t_i},\ x_{t_i} \in [-1,1] \right\} \tag{23}$$

with indeterminate $x_{t_i} \in [-1,1]$. We also use polynomial zonotopes to represent the set of trajectory parameters K. For each $j \in \{1,\dots,n_q\}$, K_j is the compact one-dimensional interval $K_j = [-1,1]$ such that K is given by the Cartesian product $K = \bigtimes_{i=1}^{n_q} K_i$. Following (3), we represent the interval K_j as a polynomial zonotope $\mathbf{K_j} = x_{k_j}$ where $x_{k_j} \in [-1,1]$ is an indeterminate.

2) Parameterized Trajectories: Using the time partition $\mathbf{T_i}$ and trajectory parameter $\mathbf{K_j}$ polynomial zonotopes described above, we create polynomial zonotopes $\mathbf{q_j}(\mathbf{T_i};\mathbf{K})$ and $\dot{\mathbf{q}}_\mathbf{j}(\mathbf{T_i};\mathbf{K})$ that overapproximate the parameterized position (Fig. 3, col. 1) and velocity trajectories defined in Def. 6 for all t in the i^{th} time subinterval. This is accomplished by plugging $\mathbf{T_i}$ and $\mathbf{K}$ into the formulas for $q_j(t;k)$ and $\dot{q}_j(t;k)$. For convenience, we restate [9, Lemma 14] which proves that sliced position and velocity polynomial zonotopes are overapproximative:

Lemma 7 (Parameterized Trajectory PZs)**.** *The parameterized trajectory polynomial zonotopes $\mathbf{q_j}(\mathbf{T_i};\mathbf{K})$ are overapproximative, i.e., for each $j \in N_q$ and $k \in \mathbf{K_j}$,*

$$q_j(t;k) \in \mathbf{q_j}(\mathbf{T_i};k) \quad \forall t \in \mathbf{T_i} \tag{24}$$

One can similarly define $\dot{\mathbf{q}}_\mathbf{j}(\mathbf{T_i};\mathbf{K})$ that are also overapproximative.

3) PZ Forward Kinematics: We begin by representing the robot's forward kinematics (10) using polynomial zonotope overapproximations of the joint position trajectories. Using $\mathbf{q}(\mathbf{T_i};\mathbf{K})$, we compute $\mathbf{p_j}(\mathbf{q}(\mathbf{T_i};\mathbf{K}))$ and $\mathbf{R_j}(\mathbf{q}(\mathbf{T_i};\mathbf{K}))$, which represent overapproximations of the position and orientation of the j^{th} frame with respect to frame $(j-1)$ at the i^{th} time step. The polynomial zonotope forward kinematics can be computed (Alg. 1) in a similar fashion to the classical formulation (10). For convenience, we restate [9, Lemma 17] that proves that this is overapproximative:

Lemma 8 (PZ Forward Kinematics)**.** *Let the polynomial zonotope forward kinematics for the j^{th} frame at the i^{th} time*

step be defined as

$$\mathbf{FK_j}(\mathbf{q}(\mathbf{T_i};\mathbf{K})) = \begin{bmatrix} \mathbf{R_j}(\mathbf{q}(\mathbf{T_i};\mathbf{K})) & \mathbf{p_j}(\mathbf{q}(\mathbf{T_i};\mathbf{K})) \\ \mathbf{0} & 1 \end{bmatrix}, \tag{25}$$

where

$$\mathbf{R_j}(\mathbf{q}(\mathbf{T_i};\mathbf{K})) = \prod_{l=1}^{j} \mathbf{R_l^{l-1}}(\mathbf{q_l}(\mathbf{T_i};\mathbf{K})), \tag{26}$$

$$\mathbf{p_j}(\mathbf{q}(\mathbf{T_i};\mathbf{K})) = \sum_{l=1}^{j} \mathbf{R_l}(\mathbf{q}(\mathbf{T_i};\mathbf{K}))\mathbf{p_l^{l-1}}, \tag{27}$$

then for each $j \in N_q$, $k \in \mathbf{K}$, $\mathrm{FK}_j(q(t;k)) \in \mathbf{FK_j}(\mathbf{q}(\mathbf{T_i};k))$ *for all* $t \in \mathbf{T_i}$.

B. Spherical Forward Occupancy Construction

This subsection describes how to use the parameterized trajectory polynomial zonotopes discussed in Section IV-A to construct the Spherical Forward Occupancy introduced in Section IV-B2.

1) Spherical Joint Occupancy: We next introduce the Spherical Joint Occupancy to compute an overapproximation of the spherical volume occupied by each arm joint in the workspace. In particular, we show how to construct a sphere that overapproximates the volume occupied by the j^{th} joint over the i^{th} time interval $\mathbf{T_i}$ given a parameterized trajectory polynomial zonotope $\mathbf{q}(\mathbf{T_i};\mathbf{K})$. First, we use the polynomial zonotope forward kinematics to compute $\mathbf{p_j}(\mathbf{q}(\mathbf{T_i};\mathbf{K}))$, which overapproximates the position of the j^{th} joint frame (Fig. 3 col. 2). Next, we decompose $\mathbf{p_j}(\mathbf{q}(\mathbf{T_i};\mathbf{K}))$ into

$$\mathbf{p_j}(\mathbf{q}(\mathbf{T_i};\mathbf{K})) = \mathbf{C_j}(\mathbf{T_i};\mathbf{K}) \oplus \mathbf{u_j}(\mathbf{T_i}), \tag{28}$$

where $\mathbf{C_j}(\mathbf{T_i};\mathbf{K})$ is a polynomial zonotope that only contains generators that are dependent on $\mathbf{K}$, and $\mathbf{u_j}(\mathbf{T_i})$ is a zonotope that is independent of $\mathbf{K}$. We then bound $\mathbf{u_j}(\mathbf{T_i})$ with an axis-aligned cube (Fig. 3, col. 3) that is subsequently bounded by a sphere (Fig. 3, col. 4) of radius $u_{j,i}$. Finally, the Spherical Joint Occupancy is defined as

$$S_j(\mathbf{q}(\mathbf{T_i};\mathbf{K})) = \mathbf{C_j}(\mathbf{T_i};\mathbf{K}) \oplus B(\mathbf{0}, r_j + u_{j,i}), \tag{29}$$

where $\mathbf{C_j}(\mathbf{T_i};\mathbf{K})$ is a polynomial zonotope representation of the sphere center, r_j is the radius nominal joint sphere, and $u_{j,i}$ is the sphere radius that represents additional uncertainty in the position of the joint. Because $\mathbf{C_j}(\mathbf{T_i};\mathbf{K})$ is only dependent on $\mathbf{K}$, slicing a particular trajectory parameter $k \in K$ yields a single point for the sphere center $\mathbf{C_j}(\mathbf{T_i};k)$. We state this result in the following lemma whose proof is in Appendix A-A:

Lemma 9 (Spherical Joint Occupancy)**.** *Let the* Spherical Joint Occupancy *for the* j^{th} *frame at the* i^{th} *time step be defined as*

$$S_j(\mathbf{q}(\mathbf{T_i};\mathbf{K})) = \mathbf{C_j}(\mathbf{T_i};\mathbf{K}) \oplus B(\mathbf{0}, r_j + u_{j,i}), \tag{30}$$

where $\mathbf{C_j}(\mathbf{T_i};\mathbf{K})$ *is the center of* $\mathbf{p_j}(\mathbf{q}(\mathbf{T_i};\mathbf{K}))$, r_j *is the radius of* $S_j(q(t;k))$, *and* $u_{j,i}$ *is the radius of a sphere that bounds the independent generators of* $\mathbf{p_j}(\mathbf{q}(\mathbf{T_i};\mathbf{K}))$. *Then,*

$$S_j(q(t;k))) \subset S_j(\mathbf{q}(\mathbf{T_i};k)) \tag{31}$$

for each $j \in N_q$, $k \in \mathbf{K}$, $S_j(q(t;k)) \in S_j(\mathbf{q}(\mathbf{T_i};k))$ *for all* $t \in \mathbf{T_i}$.

2) Spherical Forward Occupancy: Previous work [9] used (13) along with Lem. 8 to overapproximate the forward occupancy of the j^{th} link over the i^{th} time step using polynomial zonotopes:

$$\mathbf{FO_j}(\mathbf{q}(\mathbf{T_i};\mathbf{K})) = \mathbf{p_j}(\mathbf{q}(\mathbf{T_i};\mathbf{K})) \oplus \mathbf{R_j}(\mathbf{q}(\mathbf{T_i};\mathbf{K}))\mathbf{L_j}. \tag{32}$$

Our key insight is replacing (13) with an alternate representation based on a tapered capsule (17). Recall from Assum. 5 that the j^{th} link can be outer-approximated by the tapered capsule formed by the two spheres encompassing the closest joints. Sec. IV-B1 described how to overapproximate each joint volume over a continuous time interval $\mathbf{T_i}$. Therefore, by Assum. 5, the volume occupied by j^{th} link over the $\mathbf{T_i}$ is overapproximated by the tapered capsule formed by $S_j(\mathbf{q}(\mathbf{T_i};k))$ and $S_{j+1}(\mathbf{q}(\mathbf{T_i};k))$ for a given trajectory parameter $k \in K$:

$$TC_j(\mathbf{q}(\mathbf{T_i};k)) = co(S_j(\mathbf{q}(\mathbf{T_i};k)) \cup S_{j+1}(\mathbf{q}(\mathbf{T_i};k))) \quad \forall t \in \mathbf{T_i}. \tag{33}$$

However, tapered capsules are not ideal for performing collision checking with arbitrary, zonotope-based obstacles. Instead, we introduce the Spherical Forward Occupancy ($\mathcal{SFO}$) to overapproximate $TC_j(\mathbf{q}(\mathbf{T_i};k))$ with $n_s \in \mathbb{N}$ spheres (Fig. 3, col. 6):

Theorem 10 (Spherical Forward Occupancy)**.** *Let* $n_s \in \mathbb{N}$ *be given and let the* Spherical Forward Occupancy *for the* j^{th} *link at the* i^{th} *time step be defined as*

$$\mathcal{SFO}_j(\mathbf{q}(\mathbf{T_i};k)) = \left\{ \bar{S}_{j,i,m}(\mathbf{q}(\mathbf{T_i};k)) : m \in N_s \right\}, \tag{34}$$

where $N_s = \{1, \cdots, n_s\}$ *and*

$$\bar{S}_{j,i,m}(\mathbf{q}(\mathbf{T_i};k)) = B\left(\bar{c}_{j,i,m}(k), \bar{r}_{j,i,m}(k)\right), \tag{35}$$

is the m^{th} *link sphere with with center* $\bar{c}_{j,i,m}(k)$ *and radius* $\bar{r}_{j,i,m}(k)$. *Then*

$$TC_j(\mathbf{q}(\mathbf{T_i};k)) \subset \mathcal{SFO}_j(\mathbf{q}(\mathbf{T_i};k)) \tag{36}$$

for each $j \in N_q$, $k \in \mathbf{K}$, *and* $t \in \mathbf{T_i}$.

Theorem 10 yields a tighter reachable set (Fig. 3, col. 6) compared to previous methods (Fig. 3, col. 7) while remaining overapproximative. Sec. IV-E describes how the Spherical Forward Occupancy is used to compute collision-avoidance constraints.

C. Exact Signed Distance Computation

The Spherical Forward Occupancy constructed in IV-B2 is composed entirely of spheres, where each sphere can be considered a point with a margin of safety corresponding to its radius. Therefore, utilizing spheres within obstacle-avoidance constraints requires only point-to-obstacle distance calculations. To make use of this sphere-based representation for trajectory planning, this subsection presents Alg. 2 for computing the exact signed distance between a point and a zonotope in 3D based on the following lemma whose proof can be found in the appendix:

Algorithm 2: $\texttt{SDF}(\mathcal{O}, c)$

$A, b, E \leftarrow \mathcal{O}$
$d_{\text{face}} \leftarrow \max(Ac - b)$
if $d_{\text{face}} \leq 0$ // check negative distances to faces
  Return d_{face}
else // check positive distances to faces
  for $i_A = 1 : m_A$
    $d_{\text{face}} \leftarrow (Ac-b)_{i_A}$
    $p_{\text{face}} \leftarrow c - d_{\text{face}} \cdot A_{i_A}^T$
    if $d_{\text{face}} \geq 0$ **and** $\max(Ap_{\text{face}} - b) \leq 0$
      Return d_{face}
    end if
  end for
end if
$d_{\text{edge}} \leftarrow \inf$
for $i_E = 1 : |E|$ // check distances to edges
  $d_{\text{edge}} \leftarrow \min(d(c, E_{i_E}), d_{\text{edge}})$
end for
Return d_{edge}

Lemma 11. *Given an obstacle zonotope $\mathcal{O} \in \mathbb{R}^3$, consider its polytope representation with normalized rows (A, b) and edges E computed from its vertex representation. Then for any point $c \in \mathbb{R}^3$,*

$$s_d(c; \mathcal{O}) = \begin{cases} \max(Ac - b) & \text{if } c \in \mathcal{O} \\ (Ac-b)_{i_A} & \text{if } c \notin \mathcal{O}, \max(Ap_{face} - b) \leq 0 \\ \min\limits_{E_{i_E} \in E} d(c; E_{i_E}) & \text{otherwise} \end{cases}, \tag{37}$$

i_A indexes the rows of $(Ac-b)$, i_E indexes the edges of E, and $p_{face} = c - (Ac-b)_{i_A} \cdot A_{i_A}^T$ is the projection of c onto the $i_A{}^{th}$ hyperplane of $\mathcal{O}$ whose normal vector is given by $A_{i_A}^T$.

D. Trajectory Optimization

We combine parameterized polynomial zonotope trajectories (Sec. IV-A) with the Spherical Forward Occupancy (Sec. IV-B2) and an exact signed distance function (Sec. IV-C) to formulate the following trajectory optimization problem:

$$\min_{k \in K} \quad \texttt{cost}(k) \qquad \texttt{(Opt)} \tag{38}$$
$$\mathbf{q_j}(\mathbf{T_i}; k) \subseteq [q_{j,\text{lim}}^-, q_{j,\text{lim}}^+] \qquad \forall (i,j) \in N_t \times N_q \tag{39}$$
$$\dot{\mathbf{q}}_\mathbf{j}(\mathbf{T_i}; k) \subseteq [\dot{q}_{j,\text{lim}}^-, \dot{q}_{j,\text{lim}}^+] \qquad \forall (i,j) \in N_t \times N_q \tag{40}$$
$$\texttt{SDF}(\mathcal{O}_n, \bar{c}_{j,i,m}(k)) > \bar{r}_{j,i,m}(k) \qquad \forall (i,j,m,n) \in N_t \times N_q \times N_s \times N_\mathcal{O}, \tag{41}$$

where $i \in N_t$ indexes the time intervals $\mathbf{T_i}$, $j \in N_q$ indexes the j^{th} robot joint, $m \in N_s$ indexes the spheres in each link forward occupancy, and $n \in N_\mathcal{O}$ indexes the number of obstacles in the environment.

E. Safe Motion Planning

SPARROWS's planning algorithm is summarized in Alg. 3. The polytope representation (A, b) and edges E for computing the signed distance (Alg. 2) to each obstacle $\mathcal{O}_n \in \mathscr{O}$ are computed offline before planning. Every planning iteration is given t_p seconds to find a safe trajectory. The $\mathcal{SJO}$ (Lem. 9) is computed at the start of every planning iteration outside of the optimization problem and the $\mathcal{SFO}$ (Thm. 10) is computed repeatedly while numerically solving the optimization problem. If a solution to (Opt) is not found, the output of Alg. 3 is set to NaN and the robot executes a braking maneuver (Line 11) using the trajectory parameter found in the previous planning iteration. To facilitate real-time motion planning, analytical constraint gradients for (39)–(41) are provided to aid computing the numerical solution of (Opt).

Algorithm 3: SPARROWS Online Planning

1: **Require:** $t_p > 0$, $N_t, N_s \in \mathbb{N}$, $\mathscr{O}$, SDF, and $\texttt{cost} : K \to \mathbb{R}$.
2: **Initialize:** $i = 0$, $t_i = 0$, and $\{k_j^*\} = \texttt{Opt}(q_{\text{start}}, 0, 0, \mathscr{O}, \texttt{cost}, t_p, N_t, N_s, \texttt{SDF}, \mathcal{SFO})$
3: **If** $k_j^* = \texttt{NaN}$, **then** break
4: **Loop:**
5: // Line 6 executes simultaneously with Lines 7 – 9 //
6: // Use Lem. 9, Thm. 10, and Alg. 2//
  Execute $q_d(t; k_i^*)$ on robot for $t \in [t_i, t_i + t_p]$
7: $\{k_{i+1}^*\} = \texttt{Opt}(q_d(t_p; k_j^*), \dot{q}_d^j(t_p; k_j^*), \ddot{q}_d^j(t_p; k_j^*), \mathscr{O}, \texttt{cost}, t_p, N_t, N_s, \texttt{SDF}, \mathcal{SFO})$
8: **If** $k_{i+1}^* = \texttt{NaN}$, **then** break
9: **Else** $t_{i+1} \leftarrow t_i + t_p$ and $i \leftarrow i + 1$
10: **End**
11: **Execute** $q_d(t; k_i^*)$ on robot for $t \in [t_i + t_p, t_i + t_f]$

V. Demonstrations

We demonstrate SPARROWS in simulation using the Kinova Gen3 7 DOF manipulator. Code, data, and video demonstrations can be found at https://roahmlab.github.io/sparrows/.

A. Implementation Details

Experiments for SPARROWS, ARMTD [8], and MPOT [29] were conducted on a computer with an Intel Core i7-8700K CPU @ 3.70GHz CPU and two NVIDIA RTX A6000 GPUs. Baseline experiments for CHOMP [2] and TrajOpt [3] were conducted on a computer with an Intel Core i9-12900H @ 4.90GHz CPU. Polynomial zonotopes and their arithmetic are implemented in PyTorch [38] based on the CORA toolbox [39].

1) Simulation and Simulation Environments: We use a URDF of the Kinova Gen3 7-DOF serial manipulator [40] from [41] and place its base at the origin of each simulation environment. Collision geometry for the robot is provided as a mesh, which we utilize with the trimesh library [42] to check for collisions with obstacles. Note that we do not check self-collisions of the robot. For simplicity, all obstacles are static, axis-aligned cubes parameterized by their centers and edge lengths. We assume that the start and goal configurations of the robot are collision-free. Although we do not assume that a feasible path exists in each environment, most scenes were determined to be feasible by running an RRT [43] for several minutes.

2) Signed Distance Computation: Recall from Assum. 3 that each obstacle is represented by a zonotope Z. The edges E and polytope representation (A,b) of Z are computed offline before planning begins. PyTorch is used to build the computation graph for calculating the signed distances in Alg. 2. The analytical gradients of the output of Alg. 2 are derived and implemented to speed up the optimization problem (Opt).

3) Desired Trajectory: We parameterize the trajectories with a piece-wise linear velocity such that the *trajectory parameter* $k = (k_1, \cdots, k_{n_q}) \in \mathbb{R}^{n_q}$ is a constant acceleration over the planning horizon $[0, t_p)$. The trajectories are also designed to contain a constant braking acceleration applied over $[t_p, t_f]$ that brings the robot safely to a stop. Thus, given an initial position q_0 and velocity $\dot{q}_0$, the parameterized trajectory is given by:

$$q(t;k) = \begin{cases} q_0 + \dot{q}_0 t + \frac{1}{2} k t^2, & t \in [0, t_p) \\ q_0 + \dot{q}_0 t_p + \frac{1}{2} k t_p^2 + & \\ (\dot{q}_0 t_p + k t_p) \frac{(2t_f - t_p - t)(t - t_p)}{2(t_f - t_p)}, & t \in [t_p, t_f]. \end{cases} \tag{42}$$

4) Optimization Problem Implementation: SPARROWS and ARMTD utilize IPOPT [44] for online planning by solving (Opt). The constraints for (Opt), which are constructed from the Spherical Joint Occupancy (Lem. 9) and Spherical Forward Occupancy (Thm. 10), are implemented in PyTorch using polynomial zonotope arithmetic based on the CORA toolbox [39]. All of the constraint gradients of (Opt) were computed analytically to improve computation speed. Because polynomial zonotopes can be viewed as polynomials, all of the constraint gradients can be readily computed using the standard rules of differential calculus.

5) Comparisons: We compare SPARROWS to ARMTD [8] where the reachable sets are computed with polynomial zonotopes similar to [9]. We also compare to CHOMP [2], TrajOpt [3], cuRobo [30], and MPOT [29]. We used the ROS MoveIt [45] implementation for CHOMP, Tesseract Robotics's ROS package [46] for TrajOpt, and the original implementation for MPOT [29]. Note that MPOT's signed distance function was modified to account for the obstacles in the planning tasks we present here. Although CHOMP, TrajOpt, cuRobo, and MPOT are not receding-horizon planners, each method provides a useful baseline for measuring the difficulty of the scenarios encountered by SPARROWS. Note that for simplicity, we do not consider self-intersection constraints for SPARROWS or any of the baseline methods. We also include a detailed discussion in App. B on how the parameters in each of the baseline methods were tuned.

B. Runtime Comparison with ARMTD

We compare the runtime performance of SPARROWS to ARMTD while varying the number of obstacles, maximum acceleration, and planning time limit. We measure the mean constraint evaluation time and the mean planning time. Constraint evaluation includes the time to compute the constraints and the constraint gradients. Planning time includes the time required to construct the reachable sets and the total time required to solve (Opt) including constraint gradients evaluations. We consider planning scenarios with $n = 10$, 20, and 40 obstacles each placed randomly such that each obstacle is reachable by the end effector of the robot. We also consider two families of parameterized trajectories (42) such that the maximum acceleration is $\pi/24$ rad/s^2 or $\pi/6$ rad/s^2, respectively. The robot is allowed to have 7, 14, or 21 degrees of freedom by having 1, 2, or 3 Kinova arms in the environment. The maximum planning time is limited to 0.15s, 0.25s, or 0.5s for the 7 DOF arm and 0.5s and 1.0s for the 14 and 21 DOF arms. For each case, the results are averaged over 100 trials.

Methods	mean constraint evaluation time [ms]		
# Obstacles (s)	10	20	40
SPARROWS ($\pi/24$)	**3.1 ± 0.2**	**3.8 ± 0.1**	**5.1 ± 0.1**
ARMTD ($\pi/24$)	4.1 ± 0.3	5.4 ± 0.5	8.3 ± 0.7
SPARROWS ($\pi/6$)	**3.1 ± 0.1**	**3.8 ± 0.1**	5.2 ± 0.1
ARMTD ($\pi/6$)	4.1 ± 0.3	5.4 ± 0.5	8.1 ± 0.7

TABLE II: Mean runtime for constraint and constraint gradient evaluation across 100 task scenes in the **Random Obstacle Scenarios** with a single Kinova arm for SPARROWS and ARMTD under a **0.5s** planning time limit ↓

Methods	mean planning time [s]		
# Obstacles (s)	10	20	40
SPARROWS ($\pi/24$)	**0.14 ± 0.07**	**0.16 ± 0.08**	**0.20 ± 0.09**
ARMTD ($\pi/24$)	0.18 ± 0.08	0.30 ± 0.10	0.45 ± 0.08
SPARROWS ($\pi/6$)	0.16 ± 0.08	0.18 ± 0.08	0.24 ± 0.09
ARMTD ($\pi/6$)	0.24 ± 0.09	0.38 ± 0.10	0.51 ± 0.04

TABLE III: Mean per-step planning time across 100 task scenes in the **Random Obstacle Scenarios** with a single Kinova arm for SPARROWS and ARMTD under a **0.5s** planning time limit ↓. Red indicates that the average planning time limit has been exceeded.

Tables II and III summarize the results for a 7 DOF robot with a 0.5s planning time limit. SPARROWS ($\pi/24$) has the lowest mean constraint evaluation time and per-step planning time followed by SPARROWS ($\pi/6$). For each maximum acceleration condition, SPARROWS' average planning time is almost twice as fast as ARMTD when there are 20 or more obstacles. Note that we see a similar trend when the planning time is limited to 0.25s and 0.15s. Further note that under planning time limits of 0.25s (Table X) and 0.15s (Table XI), ARMTD is unable to evaluate the constraints quickly enough to solve the optimization problem within the allotted time as the number of obstacles increases (Table X). This is consistent with the cases when ARMTD's mean planning time limit exceeds the maximum allowable planning time limits in Tables III, XII, and XIII.

Tables V and IV present similar results for robots with 14 and 21 degrees of freedom. Similar to the previous results, SPARROWS computes the constraints and constraint gradients almost twice as fast as ARMTD. Furthermore, compared to SPARROWS, ARMTD struggles to generate motion plans in the allotted time.

C. Motion Planning Experiments

We evaluate the performance of SPARROWS on two sets of task scenes. The first set of scenes, **Random Obstacle**

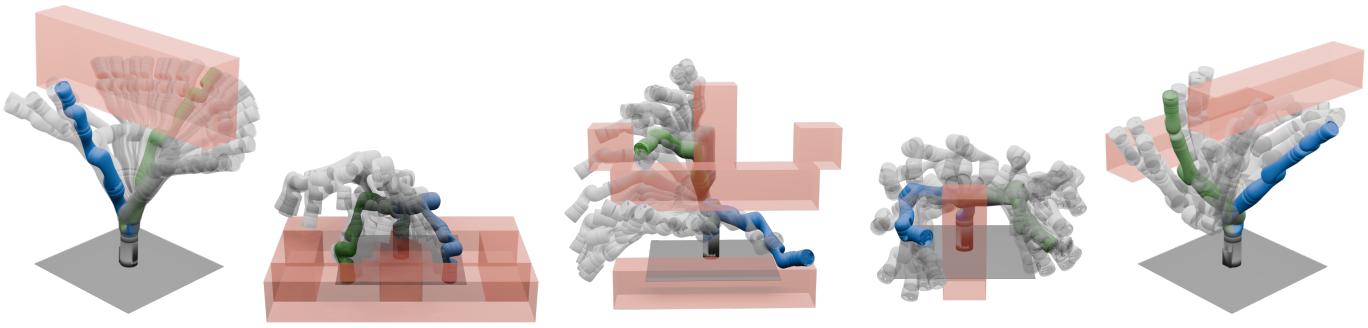

Fig. 4: Subset of **Realistic Scenarios** where SPARROWS succeeds. The start, goal, and intermediate poses are shown in blue, green, and grey (transparent), respectively. Obstacles are shown in red (transparent). The tasks include reaching from one side of a wall to another, between two small bins, from below to above a shelf, around one vertical post, and between two horizontal posts. Video demonstrations can be found at https://roahmlab.github.io/sparrows/.

Methods	mean constraint evaluation time [ms]					
# DOF	14			21		
# Obstacles	5	10	15	5	10	15
SPARROWS ($\pi/24$)	**3.9 ± 0.2**	**4.6 ± 0.2**	**5.4 ± 0.1**	**4.8 ± 0.2**	**6.0 ± 0.1**	**7.2 ± 0.1**
ARMTD ($\pi/24$)	8.4 ± 0.6	9.4 ± 0.7	10.7 ± 0.8	12.9 ± 0.9	14.5 ± 0.7	16.1 ± 0.8
SPARROWS ($\pi/6$)	**3.9 ± 0.3**	**4.6 ± 0.2**	**5.4 ± 0.1**	**4.8 ± 0.1**	**6.0 ± 0.1**	**7.2 ± 0.1**
ARMTD ($\pi/6$)	8.4 ± 0.6	9.4 ± 0.7	10.6 ± 0.7	12.8 ± 0.8	14.5 ± 0.7	16.7 ± 1.4

TABLE IV: Mean runtime for constraint and constraint gradient evaluation across 100 task scenes in the **Random Obstacle Scenarios** with 2 or 3 Kinova arms for SPARROWS and ARMTD under a **1.0s** planning time limit ↓.

Methods	mean planning time [s]					
# DOF	14			21		
# Obstacles	5	10	15	5	10	15
SPARROWS ($\pi/24$)	**0.33 ± 0.10**	**0.37 ± 0.14**	**0.41 ± 0.16**	**0.63 ± 0.14**	**0.67 ± 0.16**	**0.71 ± 0.16**
ARMTD ($\pi/24$)	0.38 ± 0.10	0.57 ± 0.16	0.76 ± 0.17	0.70 ± 0.13	0.98 ± 0.07	1.07 ± 0.03
SPARROWS ($\pi/6$)	0.38 ± 0.12	0.46 ± 0.16	0.50 ± 0.17	0.72 ± 0.16	0.79 ± 0.17	0.85 ± 0.16
ARMTD ($\pi/6$)	0.49 ± 0.16	0.72 ± 0.18	0.91 ± 0.15	0.81 ± 0.15	1.03 ± 0.03	1.06 ± 0.03

TABLE V: Mean per-step planning time across 100 task scenes in the **Random Obstacle Scenarios** with 2 or 3 Kinova arms for SPARROWS and ARMTD under a **1.0s** planning time limit ↓. across 100 task scenes in the **Random Obstacle Scenarios** with 2 or 3 Kinova arms for SPARROWS and ARMTD under a **0.5s** planning time limit. Red indicates that the average planning time limit has been exceeded.

Scenarios, shows that SPARROWS can generate safe trajectories for arbitrary tasks and DOFs. For the 7 DOF tasks, the scene contains a single Kinova arm with arm, the scene contains $n_{\mathcal{O}} = 10$, 20, or 40 axis-aligned boxes that are 20cm on each side. For the 14 and 21 DOF tasks, the scene contains 2 or 3 Kinova arms treated as a single robot with arms, the scene contains $n_{\mathcal{O}} = 5$, 10, or 15 obstacles of the same size. For each number of obstacles, we generate 100 task scenes with obstacles placed randomly such that the start and goal configurations are collision-free. Note that a solution to the goal may not be guaranteed to exist. The second set of tasks, **Realistic Scenarios**, contains 14 tasks with a Kinova arm and the number of obstacles varying from 5 to 20. The **Realistic Scenarios** show that SPARROWS can generate safe trajectories in scenes containing geometric features similar to those found in a household setting. We compare the performance of SPARROWS to ARMTD, CHOMP, TrajOpt, and MPOT on both sets of scenarios for the 7 DOF arm. We further compare the performance of SPARROWS to ARMTD on Random Obstacle Scenarios for the 14 and 21 DOF robots.

In the tests with a 7 DOF arm, SPARROWS and ARMTD are given a planning time limit of 0.5s and a maximum of 150 planning iterations to reach the goal. For the tests with 14 and 21 DOF robots, SPARROWS and ARMTD are given a planning time limit of 1.0s and a maximum of 150 planning iterations. We imposed a planning time limit of 100s, 30s, and 2s on CHOMP, TrajOpt, and cuRobo, respectively. These planning time limits were chosen such that the max planning time limit was greater than 5x the average planning time to solve the sample set of random problems that were drawn from our experimental setup. We include additional results for the average planning times for the baseline methods in Table IX.

A failure occurs if SPARROWS or ARMTD fails to find a plan for two consecutive planning iterations or if a collision occurs. Note that meshes for the robot and obstacles are used to perform ground truth collision checking within the simulator. Finally, SPARROWS and ARMTD are given a straight-line high-level planner for the **Random Obstacle Scenarios** and **Realistic Scenarios**.

Table VI presents the results for the 7 DOF **Random Obstacle Scenarios** where SPARROWS and ARMTD are given a maximum planning time limit 0.5s. SPARROWS ($\pi/6$) achieves the highest success rate across all obstacles followed by SPARROWS ($\pi/24$). A representative example is shown in Fig. 6. Notably, the performance of ARMTD degrades drastically as the number of obstacles is increased.

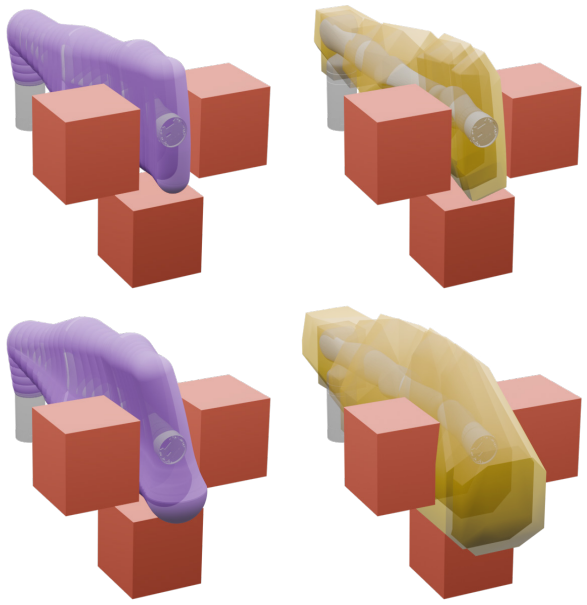

Fig. 5: A comparison of SPARROWS' and ARMTD's reachable sets. (Top row) Both SPARROWS and ARMTD can generate trajectories that allow the arm to fit through a narrow passage when the maximum acceleration is limited to $\frac{\pi}{24}$ rad/s^2. (Bottom row) However, when the maximum acceleration is $\frac{\pi}{6}$ rad/s^2 ARMTD's reachable set collides with the obstacles while SPARROWS' remains collision-free.

This is consistent with ARMTD's slower constraint evaluation presented in Table II, ARMTD's mean planning time exceeding the limit, and ARMTD's overly conservative reachable sets shown in Fig 5. Although ARMTD remains collision-free, these results suggest that the optimization problem is more challenging for ARMTD to solve. In contrast CHOMP, TrajOpt, and MPOT exhibit much lower success rates. Note that TrajOpt and MPOT frequently crash whereas CHOMP does not return feasible trajectories.

Table VII compares the success rate of SPARROWS to ARMTD on **Random Obstacle Scenarios** for higher degrees of freedom. Representative planning scenarios are shown in Fig. 7 and Fig. 8. For 14 degrees of freedom and $n_{\mathcal{O}} = 5$ obstacles, ARMTD $(\pi/24)$ has more successes, though SPARROWS $(\pi/24)$ and SPARROWS $(\pi/6)$ remain competitive. Note, in particular, that SPARROWS $(\pi/6)$'s success rate is more than double that of ARMTD $(\pi/6)$ for 10 and 15 obstacles. For 21 degrees of freedom, the performance gap between SPARROWS and ARMTD is much larger as ARMTD fails to solve any task in the presence of 10 and 15 obstacles.

Table VIII presents results for the **Realistic Scenarios** (Fig. 4). CHOMP outperforms all baseline methods with 10 successes, followed by MPOT which obtains 6 successes and 5 collisions. CHOMP's success rate of 71% on the **Realistic Scenarios** is similar to its success rate on the **Random Obstacles Scenarios** for $n = 10$ obstacles. This is likely due to CHOMP's ability to generate motion plans when the obstacle density is low. Notably, CHOMP also outperforms SPARROWS $(\pi/6)$ which is the best-performing method on the **Random Obstacles Scenarios**. By design, SPARROWS

Methods	# Successes		
# Obstacles	10	20	40
SPARROWS $(\pi/24)$	79	61	34
ARMTD $(\pi/24)$	79	50	2
SPARROWS $(\pi/6)$	**87**	**62**	**40**
ARMTD $(\pi/6)$	56	17	0
CHOMP [2]	76	40	15
TrajOpt [3]	33 (67)	9 (91)	6 (94)
MPOT [29]	58 (42)	23 (77)	9 (91)
cuRobo [30]	59 (41)	45 (55)	22 (78)

TABLE VI: Number of successes out of 100 task scenes in the **Random Obstacle Scenarios** with a single Kinova arm for SPARROWS, ARMTD, CHOMP, TrajOpt, MPOT, and cuRobo, where SPARROWS and ARMTD have a **0.5s** planning time limit ↑. Red indicates the number of failures due to collision.

Methods	# Successes					
# DOF	14			21		
# Obstacles	5	10	15	5	10	15
SPARROWS $(\pi/24)$	92	79	62	**73**	**46**	**29**
ARMTD $(\pi/24)$	**96**	75	34	**73**	0	0
SPARROWS $(\pi/6)$	94	**83**	**68**	67	36	13
ARMTD $(\pi/6)$	83	39	6	41	0	0

TABLE VII: Number of successes out of 100 task scenes in the **Random Obstacle Scenarios** with 2 or 3 Kinova arms for SPARROWS and ARMTD under a **1.0s** planning time limit ↑

and ARMTD are both receding-horizon trajectory planners that may behave myopically when optimizing for a single path to reach the next waypoint. Thus, both methods can get stuck in local minima without a good high-level planner. This is in contrast to the baseline methods which all optimize for the entire plan. To test this, we evaluated the performance of SPARROWS $(\pi/6)$ and ARMTD $(\pi/6)$ on the **Realistic Scenarios** when given access to a high-level planner. Table VIII shows that SPARROWS $(\pi/6)+$HLP can successfully solve all of the tasks when using a high-level planner, whereas ARMTD $(\pi/6)+$HLP is only able to solve 8 of the tasks. Note that no assumptions were made on the safety of waypoints given by the high-level plan.

VI. Conclusion

We present SPARROWS, a method to perform safe, real-time manipulator motion planning in cluttered environments. We introduce the Spherical Forward Occupancy to overapproximate the reachable set of a serial manipulator robot in a manner that is less conservative than previous approaches. We also introduce an algorithm to compute the exact signed distance between a point and a 3D zonotope. We use the spherical forward occupancy in conjunction with the exact signed distance algorithm to compute obstacle-avoidance constraints in a trajectory optimization problem. By leveraging reachability analysis, we demonstrate that SPARROWS is certifiably-safe, yet less conservative than previous methods. Furthermore, we demonstrate SPARROWS outperforming various state-of-the-art methods on challenging motion planning tasks. Despite SPARROWS' performance, we believe there are two promising areas of future research. Firstly, we believe the performance of SPARROWS will be further improved by using

Methods	# Successes
SPARROWS ($\pi/6$) + HLP	**14**
ARMTD ($\pi/6$) + HLP	8
SPARROWS ($\pi/6$)	5
ARMTD ($\pi/6$)	4
SPARROWS ($\pi/24$)	3
ARMTD ($\pi/24$)	3
CHOMP [2]	10
TrajOpt [3]	5 (9)
MPOT [29]	6 (8)
cuRobo [30]	5 (9)

TABLE VIII: Number of successes out of 14 task scenes in the **Realistic Scenarios** with a single Kinova arm for SPARROWS, ARMTD, CHOMP, TrajOpt, MPOT, and cuRobo, where SPARROWS and ARMTD have a **0.5s** planning time limit ↑. HLP denotes methods that have access to an additional high-level planner. Red indicates the number of failures due to collision.

it in tandem with other state-of-the-art trajectory optimization-based planners [47]–[49] or sampling-based planners [50], [51]. For example, we believe there may be instances where methods such as Graphs on Convex Sets (GCS) [48] and SPARROWS could work well together. GCS is a powerful trajectory optimizer that returns near-optimal solutions. The strict safety guarantees of SPARROWS could locally refine solutions from GCS under conditions in which the scene has undergone moderate changes, but trajectories are too expensive to recompute. Furthemore, fast sampling-based methods [23], [50], [51] that only enforce safety in discrete-time could be used to generate high-level waypoints for SPARROWS. Secondly, the major downside of our approach is that it requires the enumeration of vertices of 3D zonotopes prior to planning. However, we believe this provides a unique opportunity to combine rigorous model-based methods with deep learning-based methods for probabilistically-safe robot motion planning. Therefore, future directions aim to incorporate neural implicit scene representations into SPARROWS while accounting for uncertainty.

REFERENCES

[1] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[2] M. Zucker, N. Ratliff, A. D. Dragan, *et al.*, "Chomp: Covariant hamiltonian optimization for motion planning," *The International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1164–1193, 2013.

[3] J. Schulman, Y. Duan, J. Ho, *et al.*, "Motion planning with sequential convex optimization and convex collision checking," *The International Journal of Robotics Research*, vol. 33, no. 9, pp. 1251–1270, 2014.

[4] F. Caccavale, "Inverse dynamics control," in *Encyclopedia of Robotics*, M. H. Ang, O. Khatib, and B. Siciliano, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2020, pp. 1–5.

[5] S. Kousik, S. Vaskov, M. Johnson-Roberson, and R. Vasudevan, "Safe trajectory synthesis for autonomous driving in unforeseen environments," in *ASME 2017 Dynamic Systems and Control Conference*, American Society of Mechanical Engineers Digital Collection, 2017.

[6] L. J. Guibas, A. T. Nguyen, and L. Zhang, "Zonotopes as bounding volumes.," in *SODA*, vol. 3, 2003, pp. 803–812.

[7] M. Spong, S. Hutchinson, and M. Vidyasagar, "Robot modeling and control," 2005.

[8] P. Holmes, S. Kousik, B. Zhang, *et al.*, " Reachable Sets for Safe, Real-Time Manipulator Trajectory Design," in *Proceedings of Robotics: Science and Systems*, Corvalis, Oregon, USA, Jul. 2020.

[9] J. Michaux, P. Holmes, B. Zhang, *et al.*, *Can't touch this: Real-time, safe motion planning and control for manipulators under uncertainty*, 2023.

[10] Z. Brei, J. Michaux, B. Zhang, P. Holmes, and R. Vasudevan, "Serving time: Real-time, safe motion planning and control for manipulation of unsecured objects," *IEEE Robotics and Automation Letters*, vol. 9, no. 3, pp. 2383–2390, 2024.

[11] D. Blackmore and M. Leu, "A differential equation approach to swept volumes," in *[1990] Proceedings. Rensselaer's Second International Conference on Computer Integrated Manufacturing*, May 1990, pp. 143–149.

[12] D. Blackmore and M. Leu, "Analysis of Swept Volume via Lie Groups and Differential Equations," en, *The International Journal of Robotics Research*, vol. 11, no. 6, pp. 516–537, Dec. 1992, Publisher: SAGE Publications Ltd STM.

[13] D. Blackmore, M. C. Leu, and F. Shih, "Analysis and modelling of deformed swept volumes," en, *Computer-Aided Design*, Special Issue: Mathematical methods for CAD, vol. 26, no. 4, pp. 315–326, Apr. 1994.

[14] D. Blackmore, M. Leu, and L. P. Wang, "The sweep-envelope differential equation algorithm and its application to NC machining verification," en, *Computer-Aided Design*, vol. 29, no. 9, pp. 629–637, Sep. 1997.

[15] D. Blackmore, R. Samulyak, and M. C. Leu, "Trimming swept volumes," en, *Computer-Aided Design*, vol. 31, no. 3, pp. 215–223, Mar. 1999.

[16] Lozano-Perez, "Spatial planning: A configuration space approach," *IEEE Transactions on Computers*, vol. C-32, no. 2, pp. 108–120, 1983.

[17] K. Abdel-Malek, J. Yang, D. Blackmore, and K. Joy, "Swept volumes: Fundation, perspectives, and applications," *International Journal of Shape Modeling*, vol. 12, no. 01, pp. 87–127, Jun. 2006, Publisher: World Scientific Publishing Co.

[18] M. Campen and L. P. Kobbelt, "Polygonal boundary evaluation of minkowski sums and swept volumes," *Computer Graphics Forum*, vol. 29, 2010.

[19] Y. J. Kim, G. Varadhan, M. C. Lin, and D. Manocha, "Fast swept volume approximation of complex polyhedral models," *Comput. Aided Des.*, vol. 36, pp. 1013–1027, 2003.

[20] A. Gaschler, R. P. A. Petrick, T. Kröger, O. Khatib, and A. Knoll, "Robot task and motion planning with sets of convex polyhedra," in *Robotics: Science and Systems Conference*, 2013.

[21] N. Perrin, O. Stasse, L. Baudouin, F. Lamiraux, and E. Yoshida, "Fast humanoid robot collision-free footstep planning using swept volume approximations," *IEEE Transactions on Robotics*, vol. 28, no. 2, pp. 427–439, 2012.

[22] C. Ekenna, D. Uwacu, S. Thomas, and N. M. Amato, "Improved roadmap connection via local learning for sampling based planners," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 3227–3234.

[23] S. Murray, W. Floyd-Jones, Y. Qi, D. J. Sorin, and G. D. Konidaris, "Robot motion planning on a chip," in *Robotics: Science and Systems*, 2016.

[24] S. Duenser, J. M. Bern, R. Poranne, and S. Coros, "Interactive robotic manipulation of elastic objects," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 3476–3481.

[25] M. Gaertner, M. Bjelonic, F. Farshidian, and M. Hutter, *Collision-free mpc for legged robots in static and dynamic scenes*, 2021.

[26] B. Brito, B. Floor, L. Ferranti, and J. Alonso-Mora, *Model predictive contouring control for collision avoidance in unstructured dynamic environments*, 2020.

[27] C. Dube, "Self collision avoidance for humanoids using circular and elliptical capsule bounding volumes," in *2013 Africon*, 2013, pp. 1–6.

[28] A. El Khoury, F. Lamiraux, and M. Taïx, "Optimal motion planning for humanoid robots," in *2013 IEEE International Conference on Robotics and Automation*, 2013, pp. 3136–3141.

[29] A. T. Le, G. Chalvatzaki, A. Biess, and J. Peters, *Accelerating motion planning via optimal transport*, 2023.

[30] B. Sundaralingam, S. K. S. Hari, A. Fishman, *et al.*, *Curobo: Parallelized collision-free minimum-jerk robot motion generation*, 2023.

[31] E. Gilbert, D. Johnson, and S. Keerthi, "A fast procedure for computing the distance between complex objects in three-dimensional space,"

Methods	mean planning time [s]			
Scenarios	10 obs	20 obs	40 obs	Realistic
CHOMP [2]	8.83 ± 9.83	18.1 ± 14.1	21.6 ± 7.85	15.0 ± 7.81
TrajOpt [3]	0.920 ± 0.676	1.02 ± 0.363	2.82 ± 0.528	1.09 ± 0.580
MPOT [29]	8.78 ± 4.59	15.7 ± 7.81	30.1 ± 13.8	9.91 ± 6.42
cuRobo [30]	0.0902 ± 0.0468	0.0963 ± 0.0508	0.149 ± 0.0843	0.783 ± 0.739

TABLE IX: Mean planning time to generate a full trajectory across 100 task scenes in the **Random Obstacle Scenarios** and 14 task scenes in the **Realistic Scenarios** with a single Kinova arm for CHOMP, TrajOpt, MPOT, and cuRobo.

IEEE Journal on Robotics and Automation, vol. 4, no. 2, pp. 193–203, 1988.

[32] G. Bergen, "Proximity queries and penetration depth computation on 3d game objects," Jan. 2001.

[33] N. Kochdumper and M. Althoff, "Sparse polynomial zonotopes: A novel set representation for reachability analysis," *IEEE Transactions on Automatic Control*, vol. 66, no. 9, pp. 4043–4058, 2020.

[34] S. Kousik, S. Vaskov, F. Bu, M. Johnson-Roberson, and R. Vasudevan, "Bridging the gap between safety and real-time performance in receding-horizon trajectory design for mobile robots," *The International Journal of Robotics Research*, vol. 39, no. 12, pp. 1419–1469, 2020.

[35] S. Kousik, P. Holmes, and R. Vasudevan, "Safe, aggressive quadrotor flight via reachability-based trajectory design," in *ASME 2019 Dynamic Systems and Control Conference*, American Society of Mechanical Engineers Digital Collection, 2019.

[36] J. Liu, Y. Shao, L. Lymburner, *et al.*, *Refine: Reachability-based trajectory design using robust feedback linearization and zonotopes*, 2022.

[37] X. Zhang, A. Liniger, and F. Borrelli, "Optimization-based collision avoidance," *IEEE Transactions on Control Systems Technology*, vol. 29, no. 3, pp. 972–983, 2021.

[38] A. Paszke, S. Gross, S. Chintala, *et al.*, "Automatic differentiation in pytorch," 2017.

[39] M. Althoff and N. Kochdumper, *Cora 2018 manual*, Available at https://tumcps.github.io/CORA/data/Cora2018Manual.pdf, 2018.

[40] Kinova, *User Guide - KINOVA Gen3 Ultra lightweight robot*. 2022.

[41] Kinovarobotics, *Ros kortex*, Available at https : / / github . com / Kinovarobotics/ros_kortex, 2023.

[42] Dawson-Haggerty et al., *Trimesh*, version 3.2.0, 2019.

[43] S. M. LaValle, "Rapidly-exploring random trees : A new tool for path planning," *The annual research report*, 1998.

[44] A. Wächter and L. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical programming*, vol. 106, pp. 25–57, Mar. 2006.

[45] D. Coleman, I. A. Sucan, S. Chitta, and N. Correll, "Reducing the Barrier to Entry of Complex Robotic Software: a MoveIt! Case Study," *CoRR*, vol. abs/1404.3785, 2014.

[46] TesseractRobotics, *Trajopt_ros*, Available at https : / / github . com / tesseract - robotics / trajopt, commit 12a6d505f55cf6fa6ff5483b1c77af4c7b15b19a, 2023.

[47] A. Amice, H. Dai, P. Werner, A. Zhang, and R. Tedrake, *Finding and optimizing certified, collision-free regions in configuration space for robot manipulators*, 2022.

[48] T. Marcucci, M. Petersen, D. von Wrangel, and R. Tedrake, "Motion planning around obstacles with convex optimization," *Science Robotics*, vol. 8, no. 84, eadf7843, 2023.

[49] V. Vasilopoulos, S. Garg, P. Piacenza, J. Huh, and V. Isler, *Ramp: Hierarchical reactive motion planning for manipulation tasks using implicit signed distance functions*, 2023.

[50] M. Bhardwaj, B. Sundaralingam, A. Mousavian, *et al.*, *Storm: An integrated framework for fast joint-space model-predictive control for reactive manipulation*, 2021.

[51] W. Thomason, Z. Kingston, and L. E. Kavraki, *Motions in microseconds via vectorized sampling-based planning*, 2023.

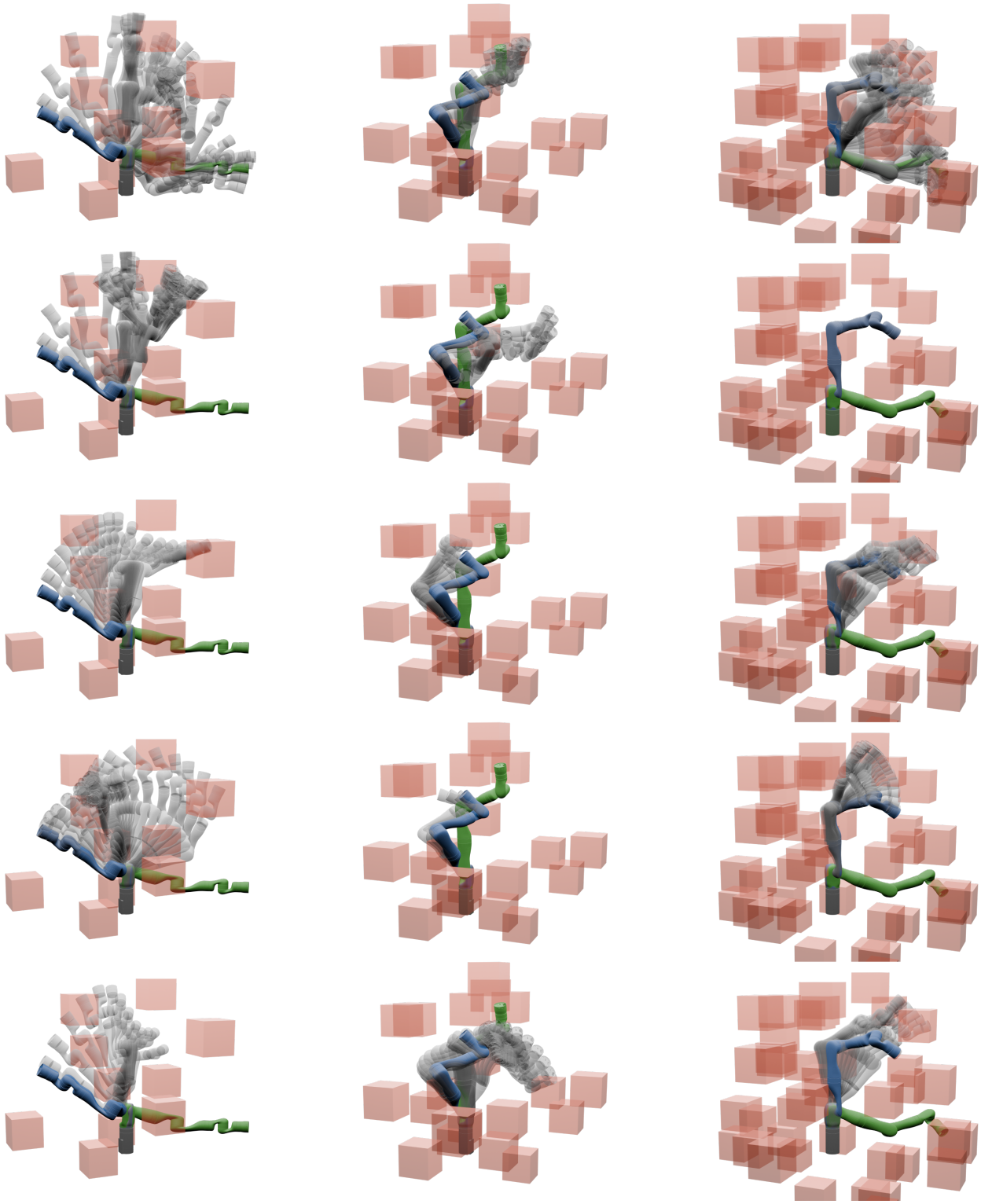

Fig. 6: **Grid of Baseline Comparison Timelapses.** A subset of **Random Obstacle Scenarios** where SPARROWS succeeds and all other methods fail. The start, goal, and intermediate poses are shown in blue, green, and grey (transparent), respectively. Obstacles are shown in red (transparent). The columns, ordered from left to right, correspond to scenes with 10, 20, and 40 obstacles. (Row 1) SPARROWS. (Row 2) ARMTD. (Row 3) TrajOpt. (Row 4) MPOT. (Row 5) cuRobo. Video demonstrations for each of these runs can be found in the accompanying supplementary material.

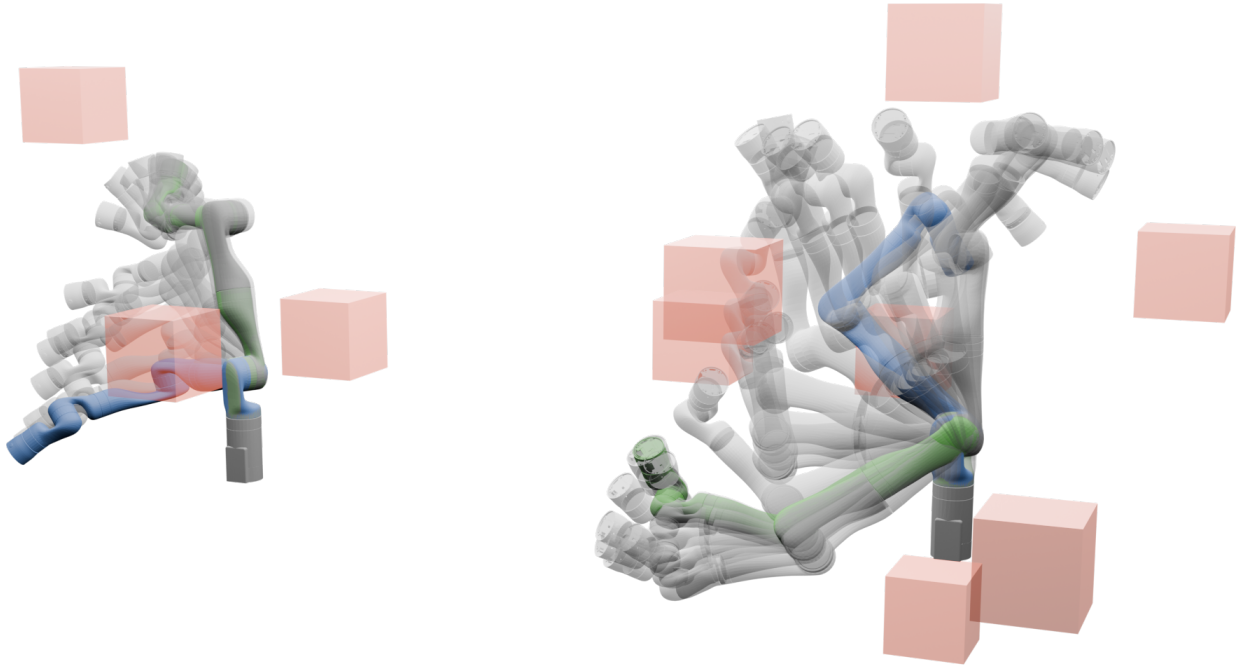

Fig. 7: A 14 DOF demonstration from the **Random Obstacle Scenarios** where SPARROWS succeeds. The start, goal, and intermediate poses are shown in blue, green, and grey (transparent), respectively. Obstacles are shown in red (transparent). A video demonstration can be found in the accompanying supplementary material.

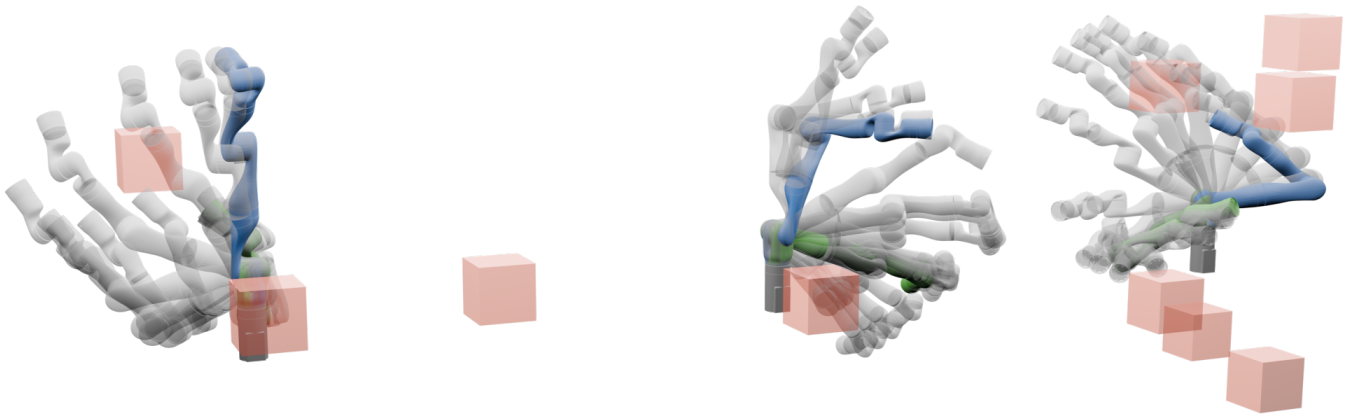

Fig. 8: A 21 DOF demonstration from the **Random Obstacle Scenarios** where SPARROWS succeeds. The start, goal, and intermediate poses are shown in blue, green, and grey (transparent), respectively. Obstacles are shown in red (transparent). A video demonstration can be found in the accompanying supplementary material.

APPENDIX A
SPHERICAL FORWARD OCCUPANCY PROOF

This section provides the proofs for Lemma 9 and Theorem 10 which are visually illustrated in Fig. 9.

A. Proof of Lemma 9

Proof: Using Lem. 8 and Alg. 1 one can compute $\mathbf{p_j}(\mathbf{q}(\mathbf{T_i};\mathbf{K}))$, which overapproximates the position of the j^{th} joint in the workspace. We then split $\mathbf{p_j}(\mathbf{q}(\mathbf{T_i};\mathbf{K}))$ (Fig. 6A) into

$$\begin{aligned}\mathbf{p_j}(\mathbf{q}(\mathbf{T_i};\mathbf{K})) &= \mathcal{PZ}(c,g_i,\alpha_i,x,h_j,y_j) && (43)\\ &= \mathcal{PZ}(g_i,\alpha_i,x,\mathbf{0},\mathbf{0}) \oplus \mathcal{PZ}(\mathbf{0},\mathbf{0},\mathbf{0},h_j,y_j) && (44)\\ &= \mathbf{C_j}(\mathbf{T_i};\mathbf{K}) \oplus \mathbf{u_j}(\mathbf{T_i}), && (45)\end{aligned}$$

where $\mathbf{C_j}(\mathbf{T_i};\mathbf{K})$ is a polynomial zonotope whose generators that are only dependent on $\mathbf{K}$ and $\mathbf{u_j}(\mathbf{T_i})$ is a zonotope that is independent of $\mathbf{K}$. This means slicing $\mathbf{C_j}(\mathbf{T_i};\mathbf{K})$ at a particular trajectory parameter $k \in K$ yields a single point given by $\mathbf{C_j}(\mathbf{T_i};k)$.

We next overapproximate $\mathbf{p_j}(\mathbf{q}(\mathbf{T_i};\mathbf{K}))$. We start by defining the vector $u_{j,i,l}$ such that

$$u_{j,i,l} = \texttt{sup}(\mathbf{u_j}(\mathbf{T_i}) \odot \hat{e}_l) \quad \forall l \in \{1,2,3\}, \tag{46}$$

where $\hat{e}_l$ is a unit vector in the standard orthogonal basis. Together, these vectors $\{u_{j,i,l}\}_{l=1}^{3}$ create an axis-aligned cube that bounds $\mathbf{u_j}(\mathbf{T_i})$ (Fig. 6B). Using the L2 norm of $\sum_{l=1}^{3} u_{j,i,l}$ allows one to then bound $\mathbf{p_j}(\mathbf{q}(\mathbf{T_i};\mathbf{K}))$ (Fig. 6C, left) as

$$\mathbf{p_j}(\mathbf{q}(\mathbf{T_i};\mathbf{K})) \subset \mathbf{C_j}(\mathbf{T_i};\mathbf{K}) \oplus B(\mathbf{0},u_{j,i}), \tag{47}$$

where

$$u_{j,i} = \|u_{j,i,1} + u_{j,i,2} + u_{j,i,3}\|_2. \tag{48}$$

Finally, letting r_j denote the sphere radius defined in Assum. 5 (Fig. 6C, right), we overapproximate the volume occupied by the j^{th} joint as

$$\begin{aligned}S_j(\mathbf{q}(\mathbf{T_i};\mathbf{K})) &= \mathbf{C_j}(\mathbf{T_i};\mathbf{K}) \oplus B(\mathbf{0},u_{j,i}) \oplus B(\mathbf{0},r_j)\\ &= \mathbf{C_j}(\mathbf{T_i};\mathbf{K}) \oplus B(\mathbf{0},r_{j,i}), && (49)\end{aligned}$$

where $r_{j,i} = r_j + u_{j,i}$.

Then for all $i \in N_t$ and $j \in N_q$ we define the Spherical Joint Occupancy ($\mathcal{SJO}$) as the collection of sets $S_j(\mathbf{q}(\mathbf{T_i};\mathbf{K}))$:

$$\mathcal{SJO} = \left\{ S_j(\mathbf{q}(\mathbf{T_i};\mathbf{K})) : \forall i \in N_t, j \in N_q \right\}. \tag{50}$$

Note that for any $k \in K$ we can slice each element of $\mathcal{SJO}$ to get a sphere centered at $\mathbf{C_j}(\mathbf{T_i};k)$ with radius $r_{j,i}$ and the *sliced* Spherical Joint Occupancy (Fig. 6D):

$$S_j(\mathbf{q}(\mathbf{T_i};k)) = B(\mathbf{C_j}(\mathbf{T_i};k), r_{j,i}). \tag{51}$$

$$\mathcal{SJO}(k) = \left\{ S_j(\mathbf{q}(\mathbf{T_i};k)) : \forall i \in N_t, j \in N_q \right\}. \tag{52}$$

■

B. Proof of Theorem 10

Proof: Let $\mathbf{C_j}(\mathbf{T_i};k)$ and $r_{j,i}$ be the center and radius of $S_j(\mathbf{q}(\mathbf{T_i};k))$, respectively. We begin by dividing the line segments connecting the centers and tangent points of $S_j(\mathbf{q}(\mathbf{T_i};k))$ and $S_{j+1}(\mathbf{q}(\mathbf{T_i};k))$ into $2(n_s-2)$ equal line segments of length

$$s_{j,i}(k) = \frac{\left\|\mathbf{C_{j+1}}(\mathbf{T_i};k) - \mathbf{C_j}(\mathbf{T_i};k)\right\|}{2(n_s-2)} \tag{53}$$

and

$$\begin{aligned}s'_{j,i}(k) &= \left(\frac{\left\|\mathbf{C_{j+1}}(\mathbf{T_i};k) - \mathbf{C_j}(\mathbf{T_i};k)\right\|}{2(n_s-2)} - \left(\frac{r_{j+1,i}-r_{j,i}}{2(n_s-2)}\right) \right)^{\frac{1}{2}}\\ &= \left(s_{j,i}(k)^2 - \left(\frac{r_{j+1,i}-r_{j,i}}{2(n_s-2)}\right)^2 \right)^{\frac{1}{2}} && (54)\end{aligned}$$

respectively. By observing that the tangent line is perpendicular to both spheres and then applying the Pythagorean Theorem to the right triangle with leg (Fig. 6E, green) and hypotenuse (Fig. 6E, light blue) lengths given by $(r_{j+1,i}-r_{j,i})$ and $2\cdot(n_s-2)\cdot s_{j,i}(k)$, respectively, one can obtain (54).

We then choose n_s sphere centers (Fig. 6F) such that

$$\bar{c}_{j,i,1}(k) = \mathbf{C_j}(\mathbf{T_i};k), \tag{55}$$

$$\bar{c}_{j,i,n_s}(k) = \mathbf{C_{j+1}}(\mathbf{T_i};k), \tag{56}$$

and

$$\bar{c}_{j,i,m+1}(k) = \mathbf{C_j}(\mathbf{T_i};k) + \frac{2m-1}{2(n_s-2)}\left(\mathbf{C_{j+1}}(\mathbf{T_i};k) - \mathbf{C_j}(\mathbf{T_i};k)\right), \tag{57}$$

where $1 \le m \le n_s-2$.

Next, let

$$\bar{r}_{j,i,1}(k) = r_{j,i} \tag{58}$$

and

$$\bar{r}_{j,i,n_s}(k) = r_{j+1,i}. \tag{59}$$

Before constructing the remaining radii, we first compute $\ell_{j,i,m+1}(k)$, which is the length of the line segment (Fig. 6F, black) that extends perpendicularly from the tapered capsule surface through center $\bar{c}_{j,i,m+1}(k)$:

$$\ell_{j,i,m+1}(k) = r_{j,i} + \frac{2m-1}{2(n_s-2)}\left(r_{j+1,i}-r_{j,i}\right), \tag{60}$$

where $1 \le m \le n_s-2$. We construct the remaining radii such that adjacent spheres intersect at a point on the tapered capsule's tapered surface. This ensures each sphere extends past the surface of the tapered capsule to provide an overapproximation. Then, $\bar{r}_{j,i,m+1}(k)$ is obtained by applying the Pythagorean Theorem to the triangle with legs $\ell_{j,i,m+1}(k)$ (Fig. 6G, black) and $s'_{j,i}(k)$ (Fig. 6G, red), respectively:

$$\begin{aligned}\bar{r}_{j,i,m+1}(k) &= \left(\ell^2_{j,i,m+1} + s'_{j,i}(k)^2\right)^{\frac{1}{2}}\\ &= \left(\ell^2_{j,i,m+1} + s_{j,i}(k)^2 - \left(\frac{r_{j+1,i}-r_{j,i}}{2(n_s-2)}\right)^2\right)^{\frac{1}{2}} && (61)\end{aligned}$$

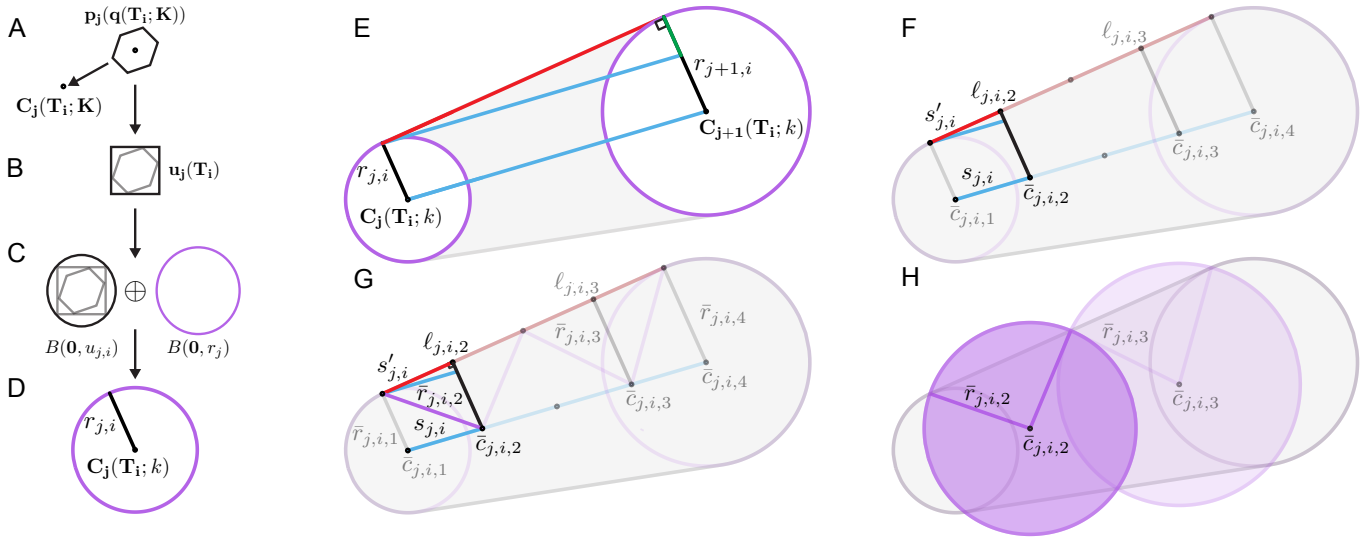

Fig. 9: Illustration of the Spherical Joint Occupancy and Spherical Forward Occupancy construction proofs in 2D.

Thus, for a given $k \in K$, we define $\mathcal{SFO}(\mathbf{q}(\mathbf{T_i};k))$ (Fig. 6H, purple) to be the set

$$\mathcal{SFO}_j(\mathbf{q}(\mathbf{T_i};k)) = \left\{\bar{S}_{j,i,m}(\mathbf{q}(\mathbf{T_i};k)) : 1 \le m \le n_s\right\}, \tag{62}$$

where

$$\bar{S}_{j,i,m}(\mathbf{q}(\mathbf{T_i};k)) = B\left(\bar{c}_{j,i,m}(k), \bar{r}_{j,i,m}(k)\right). \tag{63}$$

By construction, the tapered capsule $TC_j(\mathbf{q}(\mathbf{T_i};k))$ is a subset of $\mathcal{SFO}_j(\mathbf{q}(\mathbf{T_i};k))$ for each $j \in N_q$, $k \in \mathbf{K}$, and $t \in \mathbf{T_i}$. Further note that the construction of $\mathcal{SFO}_j(\mathbf{q}(\mathbf{T_i};k))$ is presented for the 2D case. However, this construction remains valid in 3D if one restricts the analysis to any two dimensional plane that intersects the center of each $\mathbf{C_j}(\mathbf{T_i};k)$ and $\mathbf{C_{j+1}}(\mathbf{T_i};k)$. ■

C. Proof of Lemma 11

There are three cases to consider. In the first case, c is inside the zonotope $\mathcal{O}$ (Fig. 10, left). Let (A,b) be the polytope representation of $\mathcal{O}$ such that $A \in R^{m\times n}$ and $b \in \mathbb{R}^n$. Then each row of $Ac-b$ corresponds to the signed distance to each hyperplane comprising the faces of $\mathcal{O}$. Because $c \in \mathcal{O}$, each row of $Ac-b$ is less than or equal to zero. Therefore, the signed distance to $\mathcal{O}$ is given by

$$s_d(c;\mathcal{O}) = \max(Ac-b). \tag{64}$$

In the second case, c is not inside $\mathcal{O}$ but there exists $i_A \in [1,m]$ such that

$$p_{\text{face}} = \phi(c;\mathcal{H}_{i_A}) \tag{65}$$

$$= c - (Ac-b)_{i_A} \cdot A_{i_A}^T \in \mathcal{O}, \tag{66}$$

where p_{face} is the projection of c onto the $i_A{}^{\text{th}}$ hyperplane $\mathcal{H}_{i_A}$ of $\mathcal{O}$. Then the signed distance to $\mathcal{O}$ is given by

$$s_d(c;\mathcal{O}) = (Ac-b)_{i_A}. \tag{67}$$

In the final case, c is not inside $\mathcal{O}$, but the smallest positive signed distance to each hyperplane of $\mathcal{O}$ (Fig. 10, right) lies on an edge of $\mathcal{O}$. Then the signed distance to $\mathcal{O}$ is given by

$$s_d(c;\mathcal{O}) \min_{E_{i_E}\in E} d(c;E_{i_E}). \tag{68}$$

Appendix B
Baselines

In this section we discuss how we tuned the parameters for the baseline methods.

1) CHOMP: For CHOMP, we varied several parameters including the collision clearance, the collision threshold, the trajectory initialization, and the "enable_failure_recovery" option which allows chomp to select its own parameter during runtime. We varied each of these parameters over at least two orders of magnitude and ran experiments on all combinations to choose the set of parameters that yields the best result. The best parameter set was 0.2 collision clearance and 0.007 collision threshold, with a quintic spline as the trajectory initialization method and the "enable_failure_recovery" option enabled.

2) TrajOpt: For TrajOpt, we varied several parameters including the number of waypoints for the generated trajectory (n_steps), maximum allowed planning time (max_time), and minimum distance allowed for collision checker (margin_buffer). We varied each of these parameters over at least two orders of magnitude and ran experiments on all combinations to choose the set of parameters that yields the best result. The best parameter set was 30 waypoints, 100s planning time, and 0.05m safety bound.

3) cuRobo: For cuRobo, we set the maximum time to solve to be 2 seconds, which was 4 times longer than SPARROWS received on the same set of experiments. After some experimentation, we set the maximum number of attempts is set to be 10 because it seemed to give the highest chance of success in the allotted planning time. We also turn on a geometric-based graph planner [30, Section V] that serves as an initial guess before going into the trajectory optimization stage [30,

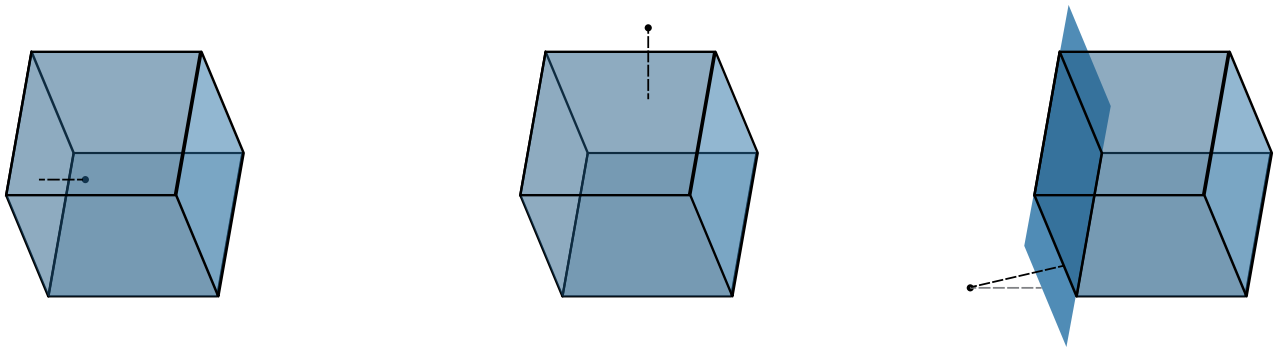

Fig. 10: Illustration of signed distance between a point and a zonotope in 3D. The first panel shows a point inside the zonotope. The second panel shows a point outside the zonotope, but whose projection lies on a single face of the zonotope. The third panel shows a point outside the zonotope, but whose projection lies on an edge of the zonotope.

Section IV]. We turned on the graph planner option because this allows cuRobo to explore a wider range of configurations and thus increases the success rate. The maximum solve time does not seem to be the main issue that limits the performance of cuRobo. For scenarios that cuRobo reaches the goal successfully, the solve time is usually less than 200 ms, which is less than the given maximum time. But note that for both random scenarios and realistic scenarios, we did not assign SPARROWS a similar global planner, but just a simple straight-line planner instead. Nevertheless, SPARROWS is still able to achieve a higher success rate.

4) MPOT: For MPOT, we set the number of particles (candiate plans) per goal to 10 as limited by GPU memory. The maximum number of inner iterations and outer iterations were both set to 1000 which are 10 times the default value.

APPENDIX C
ADDITIONAL RESULTS

Methods	mean constraint evaluation time [ms]		
# Obstacles (s)	10	20	40
SPARROWS ($\pi/24$)	**3.1 ± 0.1**	**3.8 ± 0.1**	**5.1 ± 0.1**
ARMTD ($\pi/24$)	4.1 ± 0.3	5.4 ± 0.5	8.1 ± 0.6
SPARROWS ($\pi/6$)	**3.1 ± 0.1**	**3.8 ± 0.1**	**5.1 ± 0.1**
ARMTD ($\pi/6$)	4.1 ± 0.3	5.4 ± 0.4	7.6 ± 0.3

TABLE X: Mean runtime for constraint and constraint gradient evaluation across 100 task scenes in the **Random Obstacle Scenarios** with a single Kinova arm for SPARROWS and ARMTD under a **0.25s** planning time limit ↓.

Methods	mean constraint evaluation time [ms]		
# Obstacles (s)	10	20	40
SPARROWS ($\pi/24$)	**3.1 ± 0.1**	**3.8 ± 0.1**	**5.1 ± 0.1**
ARMTD ($\pi/24$)	4.1 ± 0.3	5.3 ± 0.1	7.8 ± 0.4
SPARROWS ($\pi/6$)	**3.1 ± 0.1**	**3.8 ± 0.1**	**5.1 ± 0.1**
ARMTD ($\pi/6$)	4.0 ± 0.2	5.2 ± 0.2	8.8 ± 0.7

TABLE XI: Mean runtime for constraint and constraint gradient evaluation across 100 task scenes in the **Random Obstacle Scenarios** with a single Kinova arm for SPARROWS and ARMTD under a **0.15s** planning time limit ↓.

Methods	mean planning time [s]		
# Obstacles (s)	10	20	40
SPARROWS ($\pi/24$)	**0.12 ± 0.04**	**0.14 ± 0.05**	**0.17 ± 0.05**
ARMTD ($\pi/24$)	0.16 ± 0.04	0.24 ± 0.03	0.27 ± 0.02
SPARROWS ($\pi/6$)	0.14 ± 0.04	0.16 ± 0.04	0.20 ± 0.04
ARMTD ($\pi/6$)	0.20 ± 0.04	0.26 ± 0.02	0.27 ± 0.02

TABLE XII: Mean per-step planning time across 100 task scenes in the **Random Obstacle Scenarios** with a single Kinova arm for SPARROWS and ARMTD under a **0.25s** planning time limit ↓. Red indicates that the average planning time limit has been exceeded.

Methods	mean planning time [s]		
# Obstacles (s)	10	20	40
SPARROWS ($\pi/24$)	**0.11 ± 0.02**	**0.13 ± 0.02**	**0.14 ± 0.02**
ARMTD ($\pi/24$)	0.14 ± 0.02	0.16 ± 0.03	0.17 ± 0.03
SPARROWS ($\pi/6$)	0.13 ± 0.02	0.14 ± 0.02	0.15 ± 0.02
ARMTD ($\pi/6$)	0.16 ± 0.02	0.16 ± 0.03	0.18 ± 0.03

TABLE XIII: Mean per-step planning time across 100 task scenes in the **Random Obstacle Scenarios** with a single Kinova arm for SPARROWS and ARMTD under a **0.15s** planning time limit ↓. Red indicates that the average planning time limit has been exceeded.

Methods	# Successes					
# DOF	14			21		
# Obstacles	5	10	15	5	10	15
SPARROWS ($\pi/24$)	**74**	**47**	**27**	0	0	0
ARMTD ($\pi/24$)	70	3	0	0	0	0
SPARROWS ($\pi/6$)	55	42	12	0	0	0
ARMTD ($\pi/6$)	35	0	0	0	0	0

TABLE XIV: Number of successes out of 100 task scenes in the Randon Obstacle Scenarios with 2 or 3 Kinova arms for SPARROWS and ARMTD under a **0.5s** planning time limit ↑.

Methods	mean constraint evaluation time [ms]					
# DOF	14			21		
# Obstacles	5	10	15	5	10	15
SPARROWS ($\pi/24$)	**3.9 ± 0.2**	**4.7 ± 0.1**	**5.4 ± 0.2**	4.9 ± 0.2	**6.0 ± 0.1**	**7.2 ± 0.1**
ARMTD ($\pi/24$)	8.4 ± 0.6	9.3 ± 0.6	10.6 ± 0.4	12.6 ± 0.6	14.3 ± 0.5	17.1 ± 1.3
SPARROWS ($\pi/6$)	**3.9 ± 0.1**	**4.6 ± 0.1**	**5.4 ± 0.1**	**4.8 ± 0.1**	**6.0 ± 0.2**	**7.2 ± 0.2**
ARMTD ($\pi/6$)	8.3 ± 0.6	9.0 ± 0.2	10.6 ± 0.1	13.0 ± 0.7	14.6 ± 0.8	16.4 ± 0.9

TABLE XV: Mean runtime for constraint and constraint gradient evaluation across 100 task scenes in the **Random Obstacle Scenarios** with 2 or 3 Kinova arms for SPARROWS and ARMTD under a **0.5s** planning time limit ↓.

Methods	mean planning time [s]					
# DOF	14			21		
# Obstacles	5	10	15	5	10	15
SPARROWS ($\pi/24$)	**0.32 ± 0.06**	**0.35 ± 0.07**	**0.37 ± 0.08**	0.51 ± 0.02	0.51 ± 0.01	0.52 ± 0.02
ARMTD ($\pi/24$)	0.37 ± 0.06	0.49 ± 0.04	0.52 ± 0.02	0.54 ± 0.02	0.55 ± 0.03	0.55 ± 0.03
SPARROWS ($\pi/6$)	0.37 ± 0.08	0.40 ± 0.08	0.44 ± 0.07	0.53 ± 0.02	0.52 ± 0.01	0.53 ± 0.01
ARMTD ($\pi/6$)	0.42 ± 0.07	0.52 ± 0.02	0.54 ± 0.14	0.53 ± 0.02	0.56 ± 0.02	0.54 ± 0.02

TABLE XVI: Mean per-step planning time across 100 task scenes in the **Random Obstacle Scenarios** with 2 or 3 Kinova arms for SPARROWS and ARMTD under a **0.5s** planning time limit ↓. Red indicates that the average planning time limit has been exceeded.