Reachable Sets for Safe, Real-Time Manipulator Trajectory Design

Patrick Holmes[1], Shreyas Kousik[1], Bohao Zhang[1], Daphna Raz[2],
Corina Barbalata[3], Matthew Johnson-Roberson[2,4], Ram Vasudevan[1,2]

Abstract—**For robotic arms to operate in arbitrary environments, especially near people, it is critical to certify the safety of their motion planning algorithms. However, there is often a trade-off between safety and real-time performance; one can either carefully design safe plans, or rapidly generate potentially-unsafe plans. This work presents a receding-horizon, real-time trajectory planner with safety guarantees, called ARMTD (Autonomous Reachability-based Manipulator Trajectory Design). The method first computes (offline) a reachable set of parameterized trajectories for each joint of an arm. Each trajectory includes a fail-safe maneuver (braking to a stop). At runtime, in each receding-horizon planning iteration, ARMTD constructs a parameterized reachable set of the full arm in workspace and intersects it with obstacles to generate sub-differentiable, provably-conservative collision-avoidance constraints on the trajectory parameters. ARMTD then performs trajectory optimization over the parameters, subject to these constraints. On a 6 degree-of-freedom arm, ARMTD outperforms CHOMP in simulation, never crashes, and completes a variety of real-time planning tasks on hardware.**

I. INTRODUCTION

To maximize utility in arbitrary environments, especially when operating near people, robotic arms should plan collision-free motions in real time. Such performance requires sensing and reacting to the environment as the robot plans and executes motions; in other words, it must perform *receding-horizon planning*, where it iteratively generates a plan while executing a previous plan. This paper addresses **guaranteed-safe receding-horizon trajectory planning for robotic arms**. We call the proposed method Autonomous Reachability-based Manipulator Trajectory Design, or **ARMTD**, introduced in Fig. 1.

Motion planning can be broadly split into three paradigms, depending on whether safety is enforced by (1) a path planner, (2) a trajectory planner, or (3) a tracking controller.

The first paradigm is commonly used for robotic arm planning, wherein the path planner is responsible for safety. One generates a collision-free path, then smooths it and parameterizes it by time (i.e., converts it into a trajectory) [1], [2]. Such methods often have a tradeoff between safety and real-time performance because they represent paths with discrete points in configuration space [3], [4]. Ensuring safety requires
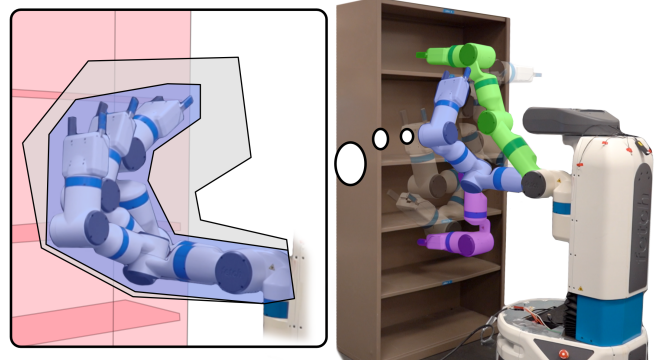
Fig. 1: ARMTD performs safe, real-time receding-horizon planning for a Fetch arm around a cabinet in real time, from a start pose (purple, low shelf) to a goal (green, high shelf). Several intermediate poses are shown (transparent). The callout on the left, corresponding to the blue intermediate pose, shows a single planning iteration, with the shelf in light red. In grey is the arm's reachable set for a continuum of parameterized trajectories over a short time horizon. The smaller blue set is the subset of the reachable set corresponding to the particular trajectory that was selected for this planning iteration, which is guaranteed not to collide with the obstacle. Over many such trials in simulation and on hardware, ARMTD never crashed. See our video: `youtu.be/ySnux2owlAA`.

approximations such as buffering the volume of the arm at each discrete point to account for the discretization, or computing the swept volume along the path assuming, e.g., straight lines between points [5]. If one treats the path as a decision variable in a nonlinear optimization program, the gradient of the distance between the arm's volume and obstacles may "push" each configuration out of collision [6]–[8]. This means the output path can be treated directly as a trajectory, if the optimization uses path smoothness as the cost. However, this relies on several approximations to achieve real-time performance: finite differencing to bound joint speeds and accelerations, collision penalties in the cost instead of hard constraints, and finite differencing [6] or linearization [7] for the collision-avoidance penalty gradient. This necessitates finer discretization to faithfully represent the robot's kinematics. To enable real-time performance without gradients, one can compute many paths offline, then collision-check at runtime [9], [10]; but for arbitrary tasks, it can be unclear how many paths are necessary, or how to ensure safety if the arm's volume changes (e.g., by grasping an object). Another approach to real-time performance is to plan iteratively in a receding-horizon either by gradient descent (with the same drawbacks as above) [8] or assuming the underlying path planner is safe [11]. In summary, in this paradigm, one must discretize finely, or buffer by a large amount, to achieve safety at the expense of performance.

In the second paradigm, the path planner generates a (potentially unsafe) path, then the trajectory planner attempts to track the path as closely as possible while maintaining safety. In

this paradigm, one computes a *reachable set* (RS) for a family of trajectories instead of computing a swept volume for a path. Methods in this paradigm can achieve both safety and real-time performance in receding-horizon planning by leveraging sums-of-squares programming [12]–[14] or zonotope reachability analysis [15]. Unfortunately, the methods in this paradigm suffer from the curse of dimensionality, preventing their use with the high-dimensional models of typical arms.

In the third paradigm, one attempts to ensure safety via the tracking controller, instead of in a path or trajectory. Here, one builds a supervisory safety controller for pre-specified trajectories [16] or a set of safe states [17]. Another approach is to compute a safety buffer and associated controller using Hamilton-Jacobi reachability analysis [18], [19], but the curse of dimensionality has prevented applying this to arms.

To the best of our knowledge, RSs in manipulator planning have only been used for either collision-checking a single, precomputed trajectory [16], [20], or for controlling to a predefined setpoint [21]. In contrast, our proposed ARMTD method generates RSs for a continuum of trajectories, allowing optimization over sets of safe trajectories. Computing such RSs directly is challenging because of the high-dimensional configuration space and nonlinear transformation to workspace used for a typical arm [13], [19].

Our proposed ARMTD method overcomes these challenges by composing a high-dimensional RS in workspace from low-dimensional reachable sets of joint configurations. ARMTD extends the second planning paradigm above by using these RSs to plan safe trajectories in real time. The RS also provides subdifferentiable collision-avoidance, self-intersection, and joint limit constraints for trajectory optimization. Importantly, the RS composition, constraint generation, and gradient evaluation are all parallelizable.

We now provide an overview of ARMTD, also shown in Fig. 2. ARMTD begins by specifying a parameterized continuum of kinematic configuration space trajectories, each of which includes a fail-safe maneuver. Offline, ARMTD computes parameterized joint reachable sets, or JRSs, of these trajectories in configuration space. At runtime (in each receding-horizon), it constructs a parameterized RS from the precomputed JRSs. ARMTD intersects the RS with obstacles to generate provably-correct safety constraints. ARMTD then performs trajectory optimization over the parameters, subject to the safety constraints. If it cannot find a feasible solution within a prespecified time limit, the arm continues executing the trajectory from its previous planning iteration (which includes a fail-safe maneuver), guaranteeing perpetual safety [11], [13]. In this work, we only discuss static environments, but this approach can extend to dynamic environments [14].

A. Contributions

We make the following contributions. First, a method to conservatively construct the RS of high-dimensional redundant robotic manipulators (Sections III–IV). Second, a parallelized method to perform real-time, provably-safe, receding-horizon trajectory optimization (Section IV). Third, a demonstration

in simulation and on hardware, with no collisions (Section V and Supplemental Video), plus a comparison to CHOMP [6]. The remaining sections are Section II (Arm, Obstacles, and Trajectory Parameters) and Section VI (Conclusion). See our video: `youtu.be/ySnux2owlAA`. Our code is available: `github.com/ramvasudevan/arm_planning`. All proofs, plus additional explanations, are available in a supplement: `roahmlab.com/s/RSS_2020_armtd_supplement.pdf`.

B. Notation

The n-dimensional real numbers are $\mathbb{R}^n$, natural numbers are $\mathbb{N}$, the unit circle is $\mathbb{S}^1$, and the set of 3×3 rotation matrices is SO(3). Vectors are either $[x_1, \cdots, x_n]^\top$ or $(x_1, \cdots, x_n)$ depending on if the size/shape is relevant. Let $U, V \subset \mathbb{R}^n$. For a point $p \in U$, $\{p\} \subset U$ is the set containing p. The power set of U is $\mathcal{P}(U)$. The Minkowski sum is $U \oplus V = \{u + v \mid u \in U, \ v \in V\}$. For a matrix $A \in \mathbb{R}^{n \times n}$, $AU = \{Au \mid u \in U\}$. For matrices, $\prod$ performs right multiplication with increasing index (e.g., $\prod_{i=1}^{3} A_i = A_1 A_2 A_3$). Greek lowercase letters in angle brackets are indeterminate variables (e.g., $\langle \sigma \rangle$). Superscripts on points index elements of a set. Subscripts are joint indices or contextual information.

II. Arm, Obstacles, and Trajectory Parameters

The goal of this work is to plan collision-free trajectories for a robotic arm operating around obstacles in a receding-horizon framework. We now discuss the arm and its environment, then our receding-horizon framework and parameterized trajectories.

A. Arm and Obstacles

1) Arm: Consider an arm with $n_q \in \mathbb{N}$ joints (i.e., n_q DOFs) and $n_q + 1$ links, including the 0th link, or *baselink*. We make the following assumptions/definitions. Each joint is a single-axis revolute joint, attached between a *predecessor* link and a *successor* link. The arm is a single kinematic chain from baselink to end effector; link $i - 1$ is joined to link i by joint i for $i = 1, \cdots, n_q$. One can create multi-DOF joints using virtual links of zero volume. The *configuration space* is $Q \subseteq \mathbb{S}^{n_q}$, containing *configurations* $q = (q_1, q_2, \cdots, q_{n_q}) \in Q$. The space of joint velocities is $\dot{Q} \subset \mathbb{R}^{n_q}$. There exists a default configuration $0 \in Q$. The *workspace*, $W \subset \mathbb{R}^3$, is the all points in space reachable by any point on the arm in any configuration. The robot's physical limits are as follows. Each joint i has a minimum and maximum position $q_{i,\text{lim}}^-$ and $q_{i,\text{lim}}^+$, maximum absolute speed $\dot{q}_{i,\text{lim}}$ and maximum absolute acceleration $\ddot{q}_{i,\text{lim}}$.

We now describe the kinematic chain. Each link has a local coordinate frame with the origin located at the link's predecessor joint (the baselink's frame is the global frame). The rotation matrix $R_i(q_i) \in$ SO(3) describes the rotation of link i relative to link $i - 1$ (by joint i). The displacement $l_i \in \mathbb{R}^3$ denotes the position of joint i on link i relative to joint $(i - 1)$ in the frame of link i. The set $L_i \subset \mathbb{R}^3$ denotes the volume occupied by the ith link, with respect to its predecessor joint, in the frame of link i. Let $\text{FO}_i : Q \to \mathcal{P}(W)$ give the forward occupancy of

link i. That is, the ith link occupies the volume

$$\text{FO}_i(q) = \left\{\sum_{j<i}\left(\prod_{n\le j}R_n(q_n)l_j\right)\right\} \oplus \left(\prod_{n\le i}R_n(q_n)L_i\right) \subset W. \quad (1)$$

Let $\text{FO} : Q \to \mathcal{P}(W)$ give the occupancy of the entire arm: $\text{FO}(q) = \bigcup_{i=1}^{n_q}\text{FO}_i(q)$. Note, the first expression in (1) gives the position of joint $(i-1)$ and the second gives the rotated volume of link i.

2) Obstacles: We denote an *obstacle* as a set $O \subset W$. If the arm's volume at $q \in Q$ is intersecting the obstacle, we say the arm is in *collision*, i.e. $\text{FO}(q) \cap O \ne \emptyset$. We assume the following about obstacles. Each obstacle is compact and static with respect to time (note, one can extend ARMTD to dynamic obstacles [14]). At any time, there are at most $n_{\text{obs}} \in \mathbb{N}$, $(n_{\text{obs}} < \infty)$ obstacles in the workspace, and the arm has access to a conservative estimate of the size and location of all such obstacles (we are only concerned with planning, not perception). Let $\mathcal{O} = \{O_1, \cdots, O_{n_O}\}$ denote a set of obstacles.

B. Receding-Horizon Planning and Timing

ARMTD plans in a receding-horizon way, meaning it generates a short plan, then executes it while generating its next short plan. Every such plan is specified over a compact time interval $T \subset \mathbb{R}$. Without loss of generality (WLOG), since time can be shifted to 0 at the beginning of any plan, we denote $T = [0, t_{\text{f}}]$. We further specify that ARMTD must generate a new plan every $t_{\text{plan}} < t_{\text{f}}$ seconds. If a collision-free plan cannot be found within t_{plan} s, the robot must continue the plan from the previous receding-horizon iteration; therefore, we include a fail-safe (braking) maneuver in each plan. The durations t_{f} and t_{plan} are chosen such that $(t_{\text{f}} - t_{\text{plan}})$ is large enough for the arm to stop from its maximum joint speeds given its maximum accelerations. This ensures every plan can include a fail-safe maneuver. We abuse notation to let $q : T \to Q$ denote a trajectory plan and $q_i : T \to Q$ denote the trajectory of the ith joint. A plan is *collision-free* if $\text{FO}(q(t)) \cap O = \emptyset \, \forall t \in T, \, \forall \, O \in \mathcal{O}$. Next, we specify the form of each plan.

C. Trajectory Parameterization

ARMTD plans using parameterized trajectories. We describe the theory, then present our implementation.

1) Theory: Let $K \subset \mathbb{R}^{n_k}$, $n_k \in \mathbb{N}$, be a compact space of *trajectory parameters*, meaning each $k \in K$ maps to a trajectory $q : T \to Q$. We use $q(t;k)$ to denote the configuration parameterized by $k \in K$ at time $t \in T$. So, in each receding-horizon planning iteration, ARMTD attempts to select a single $k \in K$ (via trajectory optimization with obstacles represented as constraints on K.

Definition 1. *We require $q : T \to Q$ to satisfy three properties for all $k \in K$. First, $q(\cdot;k)$ is at least once-differentiable w.r.t. time. Second, $q(0;k) = 0$. Third, $\dot{q}(t_{\text{f}};k) = 0$.*

The second property uses the fact that all joints are revolute, so $q(0;k) = 0$ WLOG. The third property guarantees each parameterized trajectory includes a fail-safe braking maneuver.

Note, the parameterized trajectories are kinematic, not dynamic. This is common in motion planning [6]–[10], because existing controllers can track such trajectories closely (e.g., within 0.01 rad for revolute joints [22], [23]) in the absence of disturbances such as collisions. We find these trajectories sufficient to avoid collision in real-world hardware demonstrations (Sec. V). Also, methods exist for quantifying tracking error [15], [23] and accounting for it at runtime [13], [14].

2) Implementation: We choose a parameterization that is simple yet sufficient for safe planning in arbitrary scenarios (see Sec. V). We define a *velocity parameter* $k^{\text{v}} \in \mathbb{R}^{n_q}$ for the initial velocity $\dot{q}$, and an *acceleration parameter* $k^{\text{a}} \in \mathbb{R}^{n_q}$ that specifies a constant acceleration over $[0, t_{\text{plan}}]$. We write $k^{\text{v}} = (k_1^{\text{v}}, \cdots, k_{n_q}^{\text{v}})$ and similarly for k^{a}. We denote $k = (k^{\text{v}}, k^{\text{a}}) \in K \subset \mathbb{R}^{n_k}$, where $n_k = 2n_q$. The trajectories are given by

$$\dot{q}(t;k) = \begin{cases} k^{\text{v}} + k^{\text{a}}t, & t \in [0, t_{\text{plan}}) \\ \frac{k^{\text{v}} + k^{\text{a}}t_{\text{plan}}}{t_{\text{f}} - t_{\text{plan}}}(t_{\text{f}} - t), & t \in [t_{\text{plan}}, t_{\text{f}}], \end{cases} \quad (2)$$

with $q_i(0;k) = 0$ for all k to satisfy Def. 1. These trajectories brake to a stop over $[t_{\text{plan}}, t_{\text{f}}]$ with constant acceleration.

We require that K is compact to perform reachability analysis (Sec. III). Let K_i denote the parameters for joint i. For each joint i, we specify $K_i = K_i^{\text{v}} \times K_i^{\text{a}}$, where

$$K_i^{\text{v}} = \left[\overline{k_i^{\text{v}}} - \Delta k_i^{\text{v}}, \; \overline{k_i^{\text{v}}} + \Delta k_i^{\text{v}}\right], \quad K_i^{\text{a}} = \left[\overline{k_i^{\text{a}}} - \Delta k_i^{\text{a}}, \; \overline{k_i^{\text{a}}} + \Delta k_i^{\text{a}}\right], \quad (3)$$

with $\overline{k_i^{\text{v}}}, \; \overline{k_i^{\text{a}}}, \; \Delta k_i^{\text{v}}, \; \Delta k_i^{\text{a}} \in \mathbb{R}$ and $\Delta k_i^{\text{v}}, \Delta k_i^{\text{a}} \ge 0$. To implement acceleration limits (i.e., to bound K_i^{a}), we ensure

$$K_i^{\text{a}} = \left[\max\left\{-\ddot{q}_{i,\text{lim}}, \overline{k_i^{\text{a}}} - \Delta k_i^{\text{a}}\right\}, \min\left\{\ddot{q}_{i,\text{lim}}, \overline{k_i^{\text{a}}} + \Delta k_i^{\text{a}}\right\}\right]. \quad (4)$$

Next, we use these parameterized trajectories to build parameterized reachable sets of joint configurations.

III. Offline Reachability Analysis

ARMTD uses short parameterized trajectories of joint angles for trajectory planning. We now describe a Joint Reachable Set (JRS) containing all such parameterized trajectories. All computations in this section are performed offline.

1) Theory: Since each q_i represents a rotation, we examine trajectories of $\cos(q_i)$ and $\sin(q_i)$, as shown in Fig. 2. By Def. 1, $q(\cdot;k)$ is at least once differentiable. We can write a differential equation of the sine and cosine as a function of the joint trajectory, where k is a constant:

$$\frac{d}{dt}\begin{bmatrix} \cos(q_i(t;k)) \\ \sin(q_i(t;k)) \\ k \end{bmatrix} = \begin{bmatrix} -\sin(q_i(t;k))\dot{q}_i(t;k) \\ \cos(q_i(t;k))\dot{q}_i(t;k) \\ 0 \end{bmatrix}. \quad (5)$$

We then define the parameterized JRS of the ith joint:

$$\mathscr{J}_i = \Big\{(c, s, k) \in \mathbb{R}^2 \times K \mid \exists t \in T \text{ s.t. } q_i \text{ as in Def. 1},$$
$$c = \cos(q_i(t;k)), \; s = \sin(q_i(t;k)), \quad (6)$$
$$\text{and } \tfrac{d}{dt}(\cos(q_i(t;k)), \sin(q_i(t;k)), k) \text{ as in (5)}\Big\}.$$

We account for different initial joint angles, and use the JRSs to overapproximate the forward occupancy FO, in Sec. IV.

2) Implementation: We represent (6) using zonotopes, a subclass of polytopes amenable to reachable set computation [24]. A *zonotope* is a set in $\mathbb{R}^n$ in which each element is a linear combination of a *center* $x \in \mathbb{R}^n$ and *generators* $g^1, \cdots, g^p \in \mathbb{R}^n$, $p \in \mathbb{N}$:

$$Z = \left\{ y \in \mathbb{R}^n \;\middle|\; y = x + \sum_{i=1}^{p} \beta^i g^i, \; -1 \le \beta^i \le 1 \right\}. \qquad (7)$$

We denote $Z = (x, g^i, \langle \beta^i \rangle)^p$ as shorthand for a zonotope with center x, a set of generators $\{g^i\}_{i=1}^p$, and a set of indeterminate coefficients $\{\langle \beta^i \rangle\}_{i=1}^p$ corresponding to each generator. When an indeterminate coefficient $\langle \beta^i \rangle$ is *evaluated*, or assigned a particular value, we write β^i (i.e., without angle brackets).

To represent the JRS, we first choose a time step $\Delta t \in \mathbb{R}$ such that $\frac{t_f}{\Delta t} \in \mathbb{N}$ and partition T into $\frac{t_f}{\Delta t}$ closed intervals each of length Δt, indexed by $\mathbb{N}_T = \left\{ 0, 1, \cdots, \frac{t_f}{\Delta t} - 1 \right\}$. We represent $\mathscr{J}_i$ with one zonotope per time interval, which is returned by $J_i : \mathbb{N}_T \to \mathcal{P}(\mathbb{R}^2 \times K)$. For example, the zonotope $J_i(n)$ corresponds to the time interval $[n\Delta t, (n+1)\Delta t]$. We abuse notation and let t index the subinterval of T that contains it, so that $J_i(t) = J_i(\lfloor t/\Delta t \rfloor)$ where $\lfloor \cdot \rfloor$ rounds down to the nearest integer. We use similar notation for the center, generators, and indeterminates.

Next, we make an initial condition zonotope $J_i(0) \subset \mathbb{R}^2 \times K$:

$$J_i(0) = \left(\tilde{x}_i, \left\{ \tilde{g}_i^v, \tilde{g}_i^a \right\}, \left\{ \langle \tilde{\kappa}_i^v \rangle, \langle \tilde{\kappa}_i^a \rangle \right\} \right), \qquad (8)$$

with $\tilde{x}_i = [1, 0, \overline{k_i^v}, \overline{k_i^a}]^\top$, $\tilde{g}_i^v = [0, 0, \Delta k_i^v, 0]^\top$, $\tilde{g}_i^a = [0, 0, 0, \Delta k_i^a]^\top$. The indeterminates $\langle \tilde{\kappa}_i^v \rangle$ and $\langle \tilde{\kappa}_i^a \rangle$ correspond to $\tilde{g}_i^v$ and $\tilde{g}_i^a$. $J_i(0)$ contains K_i^v and K_i^a in the k_i^v and k_i^a dimensions.

Finally, we use an open-source toolbox [25] with the time partition, differential equation (5) and (2), and initial set $J_i(0)$ to overapproximate (6). Importantly, by [26, Thm. 3.3 and Prop. 3.7], one can prove the following:

$$\mathscr{J}_i \subseteq \bigcup_{t \in T} J_i(t). \qquad (9)$$

JRSs are illustrated in Fig. 2. Next, we use the JRSs online to build an RS for the arm and identify unsafe plans in each receding-horizon iteration.

IV. Online Planning

We now present ARMTD's online algorithm for a single receding-horizon iteration (see Alg. 3 and Fig. 2). First, we construct the parameterized RS of the entire arm from the JRS of each joint. Second, we identify unsafe trajectory plans. Third, we optimize over the safe plans to minimize an arbitrary cost function. If no solution is found, we execute the previous plan's fail-safe maneuver. Note, we present self-intersection constraints in the supplement.

A. Reachable Set Construction

1) Theory: Recall that ARMTD plans while the robot is executing its previous plan. Therefore, ARMTD must estimate its future initial condition $(\tilde{q}, \dot{\tilde{q}}) \in Q \times \dot{Q}$ as a result of its previous plan by integrating (5) for t_{plan} seconds. At the beginning of each online planning iteration, we use $(\tilde{q}, \dot{\tilde{q}})$ to
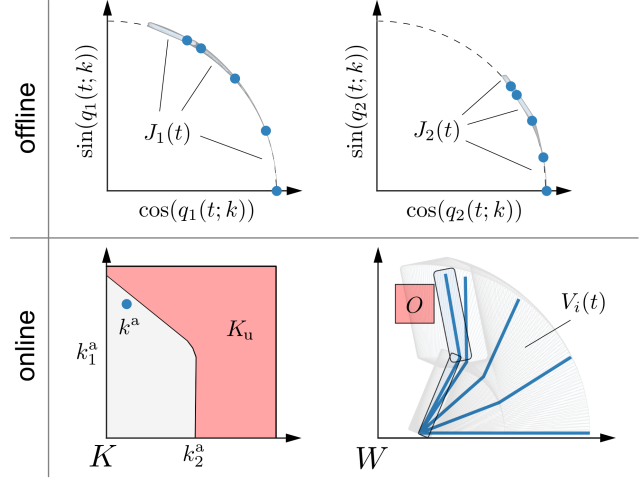

Fig. 2: An overview of the proposed method for a 2-D, 2-link arm. Offline, ARMTD computes the JRSs, shown as the collection of small grey sets $J_i(t)$ overlaid on the unit circle (dashed) in the sine and cosine spaces of two joint angles. Note that each JRS is conservatively approximated, and parameterized by trajectory parameters K. Online, the JRSs are composed to form the arm's reachable set $V_i(t)$ (large light grey sets in W), maintaining a parameterization by K. The obstacle O (light red) is mapped to the unsafe set of trajectory parameters $K_u \subset K$ on the left, by intersection with each $V_i(t)$. The parameter k^a represents a trajectory, shown at five time steps (blue arms in W, and blue dots in joint angle space). The subset of the arm's reachable set corresponding to k^a is shown for the last time step (light blue boxes with black border), critically not intersecting the obstacle, which is guaranteed because $k^a \notin K_u$.

compose the RS of the arm from the low-dimensional JRSs. Denote each link's RS $\mathscr{L}_i$, formed from all $\mathscr{J}_j$ with $j \le i$:

$$\begin{aligned} \mathscr{L}_i = \Big\{ &(Y, k) \in \mathcal{P}(W) \times K \;\Big|\; \exists t \in T \text{ s.t.} \\ &\dot{q}_i(0; k) = \dot{\tilde{q}}_i, \; Y = \text{FO}_i(q(t; k) + \tilde{q}), \\ &\text{and } \big(\cos(q_j(t; k)), \sin(q_j(t; k)), k \big) \in \mathscr{J}_j \; \forall \; j \le i \Big\} \end{aligned} \qquad (10)$$

with FO_i as in (1). Each $\mathscr{L}_i$ is formed by trajectories which start at the given initial conditions $(\tilde{q}, \dot{\tilde{q}})$. The RS of the entire arm, $\mathscr{L} \subset W \times K$, is then $\mathscr{L} = \bigcup_i \mathscr{L}_i$.

2) Implementation: It is important that we overapproximate $\mathscr{L}$ to guarantee safety when planning. To do this, we overapproximate FO for all configurations in each $\mathscr{J}_i$ (see Alg. 2).

First, we fix $\dot{\tilde{q}}$ by obtaining subsets of the JRSs containing trajectories with the given initial velocity. To do so, we note a property of the zonotope JRS:

Lemma 2. *There exist $J_i : \mathbb{N}_T \to \mathcal{P}(\mathbb{R}^2 \times K)$ that overapproximate $\mathscr{J}_i$ as in (9) such that, for each $t \in T$, $J_i(t)$ has only one generator with a nonzero element, equal to Δk_i^v, in the dimension corresponding to k_i^v; we denote this generator $g_i^v(t)$. Similarly, $J_i(t)$ has only one generator $g_i^a(t)$ (distinct from $g_i^v(t)$) with a nonzero element, Δk_i^a, for k_i^a.*

Note, the zonotopes created by the open-source toolbox [25] satisfy Lem. 2. For each $J_i(t)$, we denote the center $x_i(t)$, the generators $\{g_i^v(t), g_i^a(t), g_i^j(t)\}$, and the corresponding indeterminates $\left\{ \langle \kappa_i^v(t) \rangle, \langle \kappa_i^a(t) \rangle, \langle \beta_i^j(t) \rangle \right\}$ for $j = 1, \cdots, p(t) \in \mathbb{N}$. We write $p(t)$ since the number of generators is not necessarily the same for each $J_i(t)$ [25]. For all t except 0, $g_i^v(t)$ and $g_i^a(t)$ may have

Algorithm 1 $Z_{\text{sliced}} = \text{slice}\left(Z, \{\langle\sigma^j\rangle\}_{j=1}^n, \{\sigma^j\}_{j=1}^n\right)$

1: // Let $Z = (x, g^i, \langle\beta^i\rangle)^p$ denote the input zonotope or rotatotope
2: $Z_{\text{sliced}} \leftarrow (x, g^i, \langle\beta^i\rangle)^p$ // allocate output
3: **for** $i = 1, \cdots, p$ // iterate over generator/indeterminate pairs
4:     **for** $j = 1, \cdots, n$ // iterate over input values
5:         **if** $\langle\sigma^j\rangle \in \langle\beta^i\rangle$
6:             $g^i \leftarrow \sigma^j g^i$ // multiply generator by value
7:             $\langle\beta^i\rangle \leftarrow \langle\beta^i\rangle \setminus \langle\sigma^j\rangle$ // remove evaluated indeterminate
8:         **end if**
9:     **end for**
10:     **if** $\langle\beta^i\rangle = \emptyset$ // if fully-sliced, then g^i is no longer needed
11:         $x \leftarrow x + g^i$ and $g^i \leftarrow \emptyset$ // shift center, remove generator
12:     **end if**
13: **end for**

---

nonzero elements in the cosine and sine dimensions, due to nonzero dynamics and linearization error. The generators $g_i^v(t)$ and $g_i^a(t)$ are important because they let us obtain a subset of the JRS corresponding to a particular choice of parameters k_i^v and k_i^a. We refer to this operation as *slicing*, and we call $g_i^v(t)$ and $g_i^a(t)$ k^v-*sliceable* and k^a-*sliceable*, respectively.

To this end, we define `slice` in Alg. 1. We slice a zonotope by taking in a set of indeterminate coefficients and corresponding values with which to evaluate them. We evaluate an indeterminate by multiplying its associated generator by the given value. We then *remove* the corresponding indeterminate from the set. Since any zonotope generator has only one indeterminate, once its indeterminate is evaluated, it is called *fully-sliced*, and added to the center of the zonotope. Later in this section (Def. 4), we construct zonotope-like objects called *rotatotopes*, which have multiple indeterminates per generator (so, a generator could be sliced without being fully-sliced).

For each joint i, recall that each $J_i(t)$ has generator $g_i^v(t)$, with indeterminate $\langle\kappa_i^v(t)\rangle$ and nonzero element Δk_i^v corresponding to the k_i^v dimension. Also, $x_i(t)$ (the center of $J_i(t)$) has the value $\overline{k_i^v}$ in that same dimension. We use $\dot{\tilde{q}}$ to slice each $J_i(t)$:

$$S_i(t) = \text{slice}\left(J_i(t), \langle\kappa_i^v(t)\rangle, (\dot{\tilde{q}} - \overline{k_i^v})/\Delta k_i^v\right) \quad (11)$$

Note, we ensure $\dot{\tilde{q}} \in K_v$ later in this section. We denote $S_i(t) = (x_i^v(t), \{g_i^a(t), g_i^j(t)\}, \{\langle\kappa_i^a(t)\rangle, \langle\beta_i^j(t)\rangle\})^{p(t)}$, where $x_i^v(t)$ is the new (shifted) center and $p(t) \in \mathbb{N}$ is the new number of generators, other than $g_i^a(t)$, left after slicing. $S_i(t)$ contains a set of $\cos(q_i(t;k))$ and $\sin(q_i(t;k))$ reachable for a single value of k_i^v, but for a range of k_i^a. Denote the components of $S_i(t)$ as $x_i^v(t) = [c_i^v, s_i^v, \dot{\tilde{q}}_i, \overline{k_i^a}]^\top$, $g_i^a(t) = [c_i^a, s_i^a, 0, \Delta k_i^a]^\top$ and $g_i^j(t) = [c_i^j, s_i^j, 0, 0]^\top$ for each $j = 1, ..., p(t)$. Note from Lem. 2 that c_i^a and s_i^a are generally non-zero, and Δk_i^a is constant.

The forward occupancy map FO uses rotation matrices formed from the cosine and sine of each joint. By overapproximating these matrices, we can overapproximate FO. To this end, we represent sets of rotation matrices with matrix zonotopes. A *matrix zonotope* $M \subset \mathbb{R}^{n \times n}$ is a set of matrices parameterized by a center X and generators $G^1, \cdots, G^m$:

$$M = \left\{A \in \mathbb{R}^{n \times n} \;\middle|\; A = X + \sum_{j=1}^m G^j \lambda^j, -1 \leq \lambda^j \leq 1\right\}. \quad (12)$$

We use $M = (X, G^j, \langle\lambda^j\rangle)^m$ as shorthand for a matrix zonotope with center X, generators $\{G^j\}_{j=1}^m$, and indeterminate coefficients $\{\langle\lambda^j\rangle\}_{j=1}^m$. Note, superscripts are indices, not exponentiation, of matrix zonotope generators.

We use each sliced zonotope $S_i(t)$ to produce a matrix zonotope $M_i(t)$ that overapproximates the rotation matrices for each joint i at each time t. We do so by reshaping the center and generators of $S_i(t)$ (and keeping its indeterminates), then rotating the resulting matrix zonotope by the initial joint angle $\tilde{q}$; we call this the `makeMatZono` function in Alg. 2. See the supplement for an example of $M_i(t)$. Importantly, $M_i(t)$ satisfies the following property:

Lemma 3. *For any parameterized trajectory* $q : T \rightarrow Q$ *with* $k_i^v = \dot{\tilde{q}}$, *every* $R_i(q_i(t;k)) \in M_i(t)$.

Now we use $M_i(t)$ to overapproximate the link RS $\mathcal{L}_i$. Given the joint displacements l_i and link volumes L_i, we specify $l_j \in \mathbb{R}^3$ as a zonotope with center l_j and no generators, and L_i as a zonotope overapproximating the volume of link i. We multiply the matrix zonotopes $M_i(t)$ by L_i to overapproximate a swept volume, hence the following definition:

Definition 4. *Let* $Z = (x, g^i, \langle\beta^i\rangle)^p$ *be a zonotope and* $M = (X, G^j, \langle\lambda^j\rangle)^m$ *be a matrix zonotope. Let* $MZ := \{y \in \mathbb{R}^n \mid y = Az, A \in M, z \in Z\} \subset \mathbb{R}^n$. *We call* MZ *a* rotatotope, *which can be written:*

$$MZ = \Big\{y \in \mathbb{R}^n \mid y = Xx + \sum_i \beta^i X g^i + \sum_j \lambda^j G^j x +$$
$$+ \sum_{i,j} \beta^i \lambda^j G^j g^i, \; -1 \leq (\beta, \lambda) \leq 1\Big\}, \quad (13)$$

where $i = 1, \cdots, p$ *and* $j = 1, \cdots, m$.

We use the shorthand $MZ = (\hat{x}, \hat{g}^r, \langle\gamma^r\rangle)^s$ where $\hat{x} = Xx$, $s = (p+1)(m+1) - 1$, and the generator and coefficient sets are

$$\{\hat{g}^r\}_{r=1}^s = \{Xg^1, \cdots, Xg^p, G^1 x, \cdots, G^m x, G^1 g^1, \cdots, G^m g^p\}$$
$$\{\langle\gamma^r\rangle\}_{r=1}^s = \{\langle\beta^1\rangle, \cdots, \langle\beta^p\rangle, \langle\lambda^1\rangle, \cdots, \langle\lambda^m\rangle, \langle\beta^1\lambda^1\rangle, \cdots, \langle\beta^p\lambda^m\rangle\}.$$

Rotatotopes are a special class of polynomial zonotopes [25]. Each $\langle\gamma^r\rangle$ for $r > p + m$ is a product of indeterminate coefficients from M and Z. For a pair of indeterminate coefficients $\langle\gamma^1\rangle$ and $\langle\gamma^2\rangle$, the notation $\langle\gamma^1\gamma^2\rangle$ indicates the product $\langle\gamma^1\rangle\langle\gamma^2\rangle$. We call $\langle\gamma^1\rangle$ and $\langle\gamma^2\rangle$ the *factors* of $\langle\gamma^1\gamma^2\rangle$.

As noted earlier, we use `slice` with rotatotopes, for which we now define removing factors generically. We denote the *removal* of the ith indeterminate coefficient of $\langle\gamma^1\gamma^2 \cdots \gamma^n\rangle$ as:

$$\langle\gamma^1\gamma^2 \cdots \gamma^n\rangle \setminus \langle\gamma^i\rangle = \langle\gamma^1\gamma^2 \cdots \gamma^{i-1}\gamma^{i+1} \cdots \gamma^n\rangle. \quad (14)$$

We define $\langle\gamma^1\gamma^2 \cdots \gamma^n\rangle \setminus \langle\gamma^1\gamma^2 \cdots \gamma^n\rangle = \emptyset$. We write $\langle\sigma\rangle \in \langle\gamma^1\gamma^2 \cdots \gamma^n\rangle$ to denote that $\langle\sigma\rangle$ is a factor of $\langle\gamma^1\gamma^2 \cdots \gamma^n\rangle$.

Two useful properties follow from the rotatotope definition:

Lemma 5. *A matrix zonotope times a rotatotope is a rotatotope.*

Lemma 6. *(Zono/rotatotope Minkowski sum) Consider two zonotopes* $X = (x, g_X^i, \langle\zeta^i\rangle)^n$ *and* $Y = (y, g_Y^j, \langle\psi^j\rangle)^m$. *Then* $X \oplus Y = (x + y, \{g_X^i, g_Y^j\}, \{\langle\zeta^i\rangle, \langle\psi^j\rangle\}_{i=1,j=1}^{i=n,j=m}$, *which is a zonotope centered*

Algorithm 2 $\{V_i(t) : i = 1, \cdots, n_q, \ t \in T\} = \texttt{composeRS}(\check{q}, \dot{\check{q}})$

1: **parfor** $t \in T$ // parallel for each time step
2:     **for** $i = 1 : n_q$ // for each joint
3:        $\kappa_i^{\mathrm{y}}(t) \leftarrow (\check{q} - \overline{k_i^{\mathrm{y}}})/(\Delta k_i^{\mathrm{y}})$ // get value for (11)
4:        $S_i(t) \leftarrow \texttt{slice}(J_i(t), \langle \kappa_i^{\mathrm{y}}(t) \rangle, \kappa_i^{\mathrm{y}}(t))$ // slice JRS
5:        $M_i(t) \leftarrow \texttt{makeMatZono}(S_i(t), \widetilde{q})$
6:        $V_i(t) \leftarrow M_i(t) L_i$ // init $V_i(t)$ for link volume RS
7:        $U_i(t) \leftarrow l_{i-1}$ // init rotatotope for joint location
8:        **for** $j = (i-1) : -1 : 1$ // predecessor joints
9:           $V_i(t) \leftarrow M_j^t V_i(t)$ // rotate link volume
10:          $U_i(t) \leftarrow M_j^t U_i(t)$ // rotate joint location
11:        **end for**
12:        **for** $j = (i-1) : -1 : 1$ // predecessor joints
13:          $V_i(t) \leftarrow V_i(t) \oplus U_j(t)$ // stack link on joints
14:        **end for**
15:     **end for**
16: **end parfor**

---

at $x + y$ with all the generators and indeterminates of both X and Y. Similarly, for two rotatotopes, $V = (v, g_V^i, \langle \mu^i \rangle)^n$ and $W = (w, g_W^j, \langle \omega^j \rangle)^m$,

$$V \oplus W = \left(v + w, \{g_V^i, g_W^j\}, \{\langle \mu^i \rangle, \langle \omega^j \rangle\} \right)_{i=1, j=1}^{i=n, j=m}. \quad (15)$$

That is, the Minkowski sum is given by the sum of the centers and the union of the generators/indeterminate sets.

We use rotatotopes to overapproximate the forward occupancy map of each link by *stacking* rotatotopes representing link volume on top of rotatotopes representing joint positions:

Lemma 7. *For any $t \in T$ and $k \in K$, $\mathrm{FO}_i(q(t; k)) \subseteq V_i(t)$, where*

$$V_i(t) = \bigoplus_{j < i} \left(\prod_{n \le j} M_n(t)\{l_j\} \right) \oplus \left(\prod_{n \le i} M_n(t) L_i \right) \subset W. \quad (16)$$

Lem. 7 lets us overapproximate the RS: $\mathscr{L}_i \subseteq \bigcup_{t \in T} V_i(t) \implies \mathscr{L} \subseteq \bigcup_{t,i} V_i(t)$, as shown in Fig. 2, Alg. 2 computes $V_i(t)$.

Though $V_i(t) \subset W$, many of its generators are k^{a}-sliceable, because they are the product of k^{a}-sliceable matrix zonotope generators. Denote $V_i(t) = (\hat{x}_i(t), \hat{g}_i^j(t), \langle \hat{\beta}_i^j(t) \rangle)^{p(t)}$. Formally, the jth generator $\hat{g}_i^j(t)$ is k^{a}-sliceable if there exists at least one $\langle \kappa_n^{\mathrm{a}}(t) \rangle \in \langle \hat{\beta}_i^j(t) \rangle$ with $n \le i$. This means, by slicing by k^{a}, we can obtain a subset of $V_i(t)$ corresponding to that parameter. We make the distinction that a generator $\hat{g}_i^j(t)$ is *fully-k^{a}-sliceable* if *all* of its indeterminates are evaluated when sliced by k^{a}, i.e. $\langle \hat{\beta}_i^j(t) \rangle \subseteq \bigcup_{n \le i} \langle \kappa_n^{\mathrm{a}}(t) \rangle$. Fully-$k^{\mathrm{a}}$-sliceable generators are created by multiplying k^{a}-sliceable generators with each other or with centers in (16). These generators are important because all of their indeterminates are evaluated by the trajectory optimization decision variable k^{a}, which we use in Sec. IV-B2.

B. Constraint Generation

1) Theory: With the RS composed, we now use $\mathscr{L}$ to find all unsafe trajectory parameters $k \in K_{\mathrm{u}} \subseteq K$ that could cause collisions with obstacles. We treat K_{u} as a constraint for trajectory optimization, shown in Fig. 2. Recall $q_{i,\mathrm{lim}}^-$, $q_{i,\mathrm{lim}}^+$, and $\dot{q}_{i,\mathrm{lim}}$ are joint limits. Let $\mathscr{O}$ be a set of obstacles. At

each planning iteration, the unsafe trajectory parameters are $K_{\mathrm{u}} = K_{\mathrm{lim}} \cup K_{\mathrm{obs}}$, where

$$\begin{aligned} K_{\mathrm{lim}} = \{ k \mid \exists \ t \in T \ \text{s.t.} \ q(t; k) < q_{i,\mathrm{lim}}^- \ \text{or} \ q(t; k) > q_{i,\mathrm{lim}}^+ \\ \text{or} \ |\dot{q}(t; k)| > \dot{q}_{i,\mathrm{lim}} \} \end{aligned} \quad (17)$$

$$K_{\mathrm{obs}} = \{ k \mid Y \cap O \ne \emptyset, (Y, k) \in \mathscr{L}, O \in \mathscr{O} \}. \quad (18)$$

2) Implementation: We represent K_{lim} with functions $h_{\mathrm{lim}} : K^{\mathrm{a}} \to \mathbb{R}$. Notice in (2) that $q(t; k)$ is piecewise quadratic in k and $\dot{q}(t; k)$ is piecewise linear in k, so the parameterized trajectory extrema can be computed analytically. We construct h_{lim} from $\dot{\check{q}}_i$, $q_{i,\mathrm{lim}}$, and $\dot{q}_{i,\mathrm{lim}}$, such that $h_{\mathrm{lim}}(k^{\mathrm{a}}) < 0$ when feasible.

To represent K_{obs} (depicted in Fig. 2), first consider a particular k^{a}. We test if the corresponding subset of each rotatotope $V_i(t)$ could intersect any obstacle $O \in \mathscr{O}$. We overapproximate each O by a zonotope, which is always possible for compact, bounded sets [26] that appear in common obstacle representations such as octrees [27] or convex polytopes [28]. To proceed, we must test if two zonotopes intersect:

Lemma 8. *[29, Lem. 5.1] For two zonotopes $X = (x, g^i, \langle \beta^i \rangle)^n$ and $Y = (y, g^j, \langle \beta^j \rangle)^m$, $X \cap Y \ne \emptyset$ iff y is in the zonotope $X_{\mathrm{buf}} = (x, g^i, \langle \beta^i \rangle)^n \oplus (0, g^j, \langle \beta^j \rangle)^m$, where the subscript indicates X is buffered by the generators of Y.*

Since zonotopes are convex polytopes [29], by [26, Theorem 2.1], one can implement Lem. 8 by computing a *half-space representation* $(A_{\mathrm{buf}}, b_{\mathrm{buf}})$ of X_{buf} for which $A_{\mathrm{buf}}z - b_{\mathrm{buf}} \le 0 \iff z \in X_{\mathrm{buf}}$, where the inequality is taken elementwise. Using this representation, $X \cap Y = \emptyset \iff \max(A_{\mathrm{buf}}y - b_{\mathrm{buf}}) > 0$. We can use Lem. 8 for collision avoidance by replacing X (resp. Y) with a zonotope representing the arm (resp. an obstacle).

However, since we use rotatotopes, we need the following:

Lemma 9. *Any rotatotope MZ as in (13) can be overapproximated by a zonotope.*

So, we can overapproximate the intersection of each $V_i(t)$, sliced by k^{a}, with each $O \in \mathscr{O}$. Note, we only slice the fully-k^{a}-sliceable generators of $V_i(t)$, and treat all other generators conservatively by applying Lemma 9. That is, we do not slice any generators that have any indeterminates in addition to $\langle \kappa_i^{\mathrm{a}}(t) \rangle$, and instead use those generators to (conservatively) buffer obstacles.

To check intersection, we separate $V_i(t)$ into two rotatotopes,

$$V_{i,\mathrm{slc}}(t) = \left(x_i(t), g_{\mathrm{slc}}^j, \langle \kappa_{\mathrm{slc}}^j \rangle \right) \ \text{and} \ V_{i,\mathrm{buf}}(t) = \left(0, g_{\mathrm{buf}}^n, \langle \beta_{\mathrm{buf}}^n \rangle \right), \quad (19)$$

such that $V_i(t) = V_{i,\mathrm{slc}}(t) \oplus V_{i,\mathrm{buf}}(t)$, where $V_{i,\mathrm{slc}}(t)$ has only fully-k^{a}-sliceable generators. That is, each $\langle \kappa_{\mathrm{slc}}^j \rangle$ is a product of *only* $\langle \kappa_i^{\mathrm{a}}(t) \rangle$ for one or more $i \in \{1, \cdots, n_q\}$. Note, the number of generators/indeterminates in $V_{i,\mathrm{slc}}(t)$ and $V_{i,\mathrm{buf}}(t)$ is omitted to ease notation. For any $k^{\mathrm{a}} \in K^{\mathrm{a}}$, since every generator of $V_{i,\mathrm{slc}}(t)$ is k^{a}-sliceable, slicing $V_{i,\mathrm{slc}}(t)$ by k^{a} returns a point. We express this with $\texttt{eval} : \mathcal{P}(W) \times K^{\mathrm{a}} \to \mathbb{R}^3$ for which

$$\texttt{eval}(V_{i,\mathrm{slc}}(t), k^{\mathrm{a}}) = \texttt{slice}\left(V_{i,\mathrm{slc}}(t), \{\langle \kappa_i^{\mathrm{a}}(t) \rangle\}_{i=1}^{n_q}, \{\kappa(i)\}_{i=1}^{n_q} \right) \quad (20)$$

where $\kappa(i) = (k_i^{\mathrm{a}} - \overline{k_i^{\mathrm{a}}})/\Delta k_i^{\mathrm{a}}$. Note, $\texttt{eval}$ can be implemented as the evaluation of polynomials.

Algorithm 3 $q_{\text{plan}} = \texttt{makePlan}(\tilde{q}, \dot{\tilde{q}}, q_{\text{prev}}, \mathcal{O}, f)$

1: $\{V_i(t)\} \leftarrow \texttt{composeRS}(\tilde{q}, \dot{\tilde{q}})$ // Sec. IV-A2
2: $(h_{\text{obs}}, h_{\text{lim}}) \leftarrow \texttt{makeCons}(\tilde{q}, \dot{\tilde{q}}, \mathcal{O}, \{V_i(t)\})$ // Sec. IV-B2
3: // solve (24) within t_{plan} or else return q_{prev}
4: $q_{\text{plan}} \leftarrow \texttt{optTraj}(f, h_{\text{obs}}, h_{\text{lim}}, t_{\text{plan}}, q_{\text{prev}})$ // Sec. IV-C2

Now, let A_{obs} and b_{obs} be the halfspace representation of $O_{\text{buf}} = O \oplus V_{i,\text{buf}}(t)$, and let $x = \texttt{eval}(V_{i,\text{slc}}(t), k^a)$. Then,

$$(\{x\} \oplus V_{i,\text{buf}}(t)) \cap O = \emptyset \iff -\max\{A_{\text{obs}}x - b_{\text{obs}}\} < 0 \quad (21)$$

where $\{x\} \oplus V_{i,\text{buf}}(t)$ is overapproximated as a zonotope by applying Lem. 9. We use (21) to overapproximate the parameters K_{obs} (18) with $h_{\text{obs}} : \mathbb{N} \times T \times \mathcal{O} \times K^a \to \mathbb{R}$ for which

$$h_{\text{obs}}(*, k^a) = -\max\{A_{\text{obs}}(*)\texttt{eval}(V_{i,\text{slc}}(t), k^a) - b_{\text{obs}}(*)\}. \quad (22)$$

where $* = (i, t, O)$ for space. Here, $A_{\text{obs}}(i, t, O)$ and $b_{\text{obs}}(i, t, O)$ return the halfspace representation of $O \oplus V_{i,\text{buf}}(t)$. Importantly, for each obstacle, time, and joint, h_{obs} is a max of a linear combination of polynomials in k^a (per (20) and Alg. 1), so we can take its subgradient with respect to k^a [30] (also see [31, Thm. 5.4.5]). This constraint conservatively approximates K_{obs}:

Lemma 10. *If* $k^a \in K_{\text{obs}}$, *then there exists* $i \in \mathbb{N}$, $t \in T$, *and* $O \in \mathcal{O}$ *such that* $h_{\text{obs}}(i, t, O, k^a) \geq 0$.

C. Trajectory Optimization

1) Theory: ARMTD performs trajectory optimization over $K \setminus K_u$ for an arbitrary user-specified cost function $f : K \to \mathbb{R}$ (which encodes information such as completing a task). ARMTD attempts to solve the following within t_{plan}:

$$k_{\text{opt}} = \text{argmin}_k\{f(k) \mid k \notin K_u\}. \quad (23)$$

If no solution is found in time, the robot tracks the fail-safe maneuever from its previous plan.

2) Implementation: We implement (23) as a nonlinear program, denoted $\texttt{optTraj}$ in Alg. 3.

$$\underset{k^a \in K^a}{\text{argmin}}\{f(k^a) \mid h_{\text{obs}}(i, t, O, k^a) < 0, \; h_{\text{lim}}(k^a) < 0\} \quad (24)$$

where the constraints hold for all $i \in \{1, \cdots, n_q\}$, $t \in T$, $O \in \mathcal{O}$.

Theorem 11. *Any feasible solution to* (24) *parameterizes a trajectory that is collision-free and obeys joint limits over the time horizon* T.

ARMTD uses Alg. 3 at each planning iteration. If the arm does not start in collision, this algorithm ensures that the arm is always safe (see [13, Remark 70] or [11, Theorem 1]).

V. DEMONSTRATIONS

We now demonstrate ARMTD in simulation and on hardware using the Fetch mobile manipulator (Fig. 1). ARMTD is implemented in MATLAB, CUDA, and C++, on a 3.6 GHz computer with an Nvidia Quadro RTX 8000 GPU. See our video: `youtu.be/ySnux2owlAA`. Our code is available: `github.com/ramvasudevan/arm_planning`.
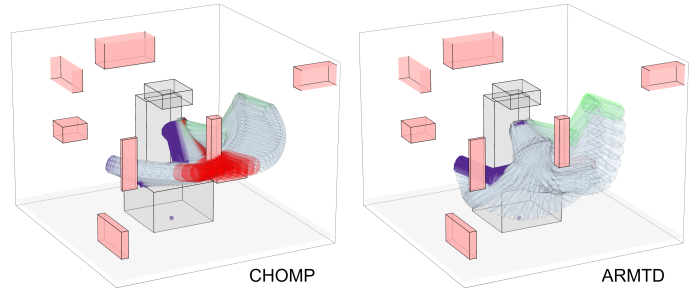

CHOMP   ARMTD

Fig. 3: A Random Obstacles scene with 8 obstacles in which CHOMP [6] converged to a trajectory with a collision (collision configurations shown in red), whereas ARMTD successfully navigated to the goal (green); the start pose is shown in purple. CHOMP fails to move around a small obstacle close to the front of the Fetch.

A. Implementation Details

1) Manipulator: The Fetch arm has 7 revolute DOFs [32]. We consider the first 6 DOFs, and treat the body as an obstacle. The 7th DOF controls end effector orientation, which does not affect the volume used for collision checking. We command the hardware via ROS [33] over WiFi.

2) Comparison: To assess the difficulty of our simulated environments, we ran CHOMP [6] via MoveIt [34] (default settings, straight-line initialization). We emphasize that CHOMP is not a receding-horizon planner [34]; it attempts to find a plan from start to goal with a single optimization program. However, CHOMP provides a useful baseline to measure the performance of ARMTD. To the best of our knowledge, no open-source, real-time receding-horizon planner is available for a direct comparison. Note, we report solve times to illustrate that ARMTD is real-time feasible, but the goal of ARMTD is not to solve as fast as possible; instead, we care about finding provably collision-free trajectories in the allotted time t_{plan}.

3) High-level Planner: Recall that ARMTD performs trajectory optimization using an arbitrary user-specified cost function. In this work, in each planning iteration, we create a cost function for ARMTD using an intermediate waypoint between the arm's current configuration and a global goal. These waypoints are generated by a high-level planner (HLP). Note, the RS and safety constraints generated by ARMTD are independent of the HLP, which is only used for the cost function. To illustrate that ARMTD can enforce safety, we use two different HLPs, neither of which is guaranteed to generate collision-free waypoints. First, a straight-line HLP that generates waypoints along a straight line between the arm and a global goal in configuration space. Second, an RRT* [35] that only ensures the arm's end effector is collision-free. Thus, **ARMTD can act as a safety layer on top of RRT***. Note, we allot a portion of t_{plan} to the HLP in each iteration, and give ARTMD the rest of t_{plan}. We cannot use CHOMP as a receding-horizon planner with these HLP waypoints, because it requires a collision-free goal configuration.

4) Algorithm Implementation: Alg. 2 runs at the start of each ARMTD planning iteration. We use a GPU with CUDA to execute Alg. 2 in parallel, taking approximately 10–20 ms to compose a full RS. The constraint generation step in Alg. 3

is also parallelized across obstacles and time steps (this takes approximately 10–20 ms for 20 obstacles).

We solve ARMTD's trajectory optimization (24) using IPOPT [36]. The cost function f is $\|q(t_f; k) - q_{\text{des}}\|_2^2$, where q_{des} is the waypoint specified by the HLP (straight-line or RRT*) at each planning iteration. We compute analytic gradients/sub-gradients of the cost function and constraints, and evaluate the constraints in parallel. IPOPT takes 100–200 ms when it finds a feasible solution in a scene with 20 random obstacles.

5) Hyperparameters: To reduce conservatism, we partition K_i^v into $n_{\text{JRS}} \in \mathbb{N}$ equally-sized intervals and compute one JRS for each interval. At runtime, for each joint, we pick the JRS containing the initial speed $\dot{q}_i$. In each JRS, we set $\Delta k_i^a = \max\left\{r_{a_2}, \ r_{a_1}|\overline{k_i^v}|\right\}$, with $r_{a_1}, r_{a_2} > 0$ so the range of accelerations scales with the absolute value of the mean velocity of each JRS. This reduces conservativism at low speeds, improving maneuverability near obstacles.

We also use these values: $t_{\text{plan}} = 0.5$ s, $t_f = 1.0$ s, $\Delta t = 0.01$ s, $n_{\text{JRS}} = 400$, $\dot{q}_{i,\text{lim}} = \pi \frac{\text{rad}}{s}$, $\ddot{q}_{i,\text{lim}} = \pi/3 \frac{\text{rad}}{s^2}$, $\overline{k_i^a} = 0 \frac{\text{rad}}{s^2}$, $r_{a_1} = 1/3 s^{-1}$, and $r_{a_2} = \pi/24 \frac{\text{rad}}{s^2}$. For collision checking, we overapproximate the Fetch's links with cylinders of radius 0.146 m.

B. Simulations

1) Setup: We created two sets of scenes. The first set, Random Obstacles, shows that ARMTD can handle arbitrary tasks (see Fig 3). This set contains 100 tasks with random (but collision-free) start and goal configurations, and random box-shaped obstacles. Obstacle side lengths vary from 1 to 50 cm, with 10 scenes for each $n_O = 4, 8, ..., 40$.

The second set, Hard Scenarios, shows that ARMTD guarantees safety where CHOMP converges to an unsafe trajectory. There are seven tasks in the Hard Scenarios set: (1) from below to above a table, (2) from one side of a wall to another, (3) between two vertical posts, (4) from one set of shelves to another, (5) from inside to outside of a box on the ground, (6) from a sink to a cupboard, (7) through a small window. These are shown in the supplementary document.

2) Results: Table I presents ARMTD (with a straight-line HLP) and CHOMP's results for the Random Obstacles scenarios. ARMTD reached 84/100 goals and had 0/100 crashes, meaning ARMTD stopped safely 16/100 times without finding a new safe trajectory. CHOMP reached 82/100 goals and had 18/100 crashes. CHOMP always finds a trajectory, but not necessarily a collision-free one; it can converge to infeasible solutions because it considers a non-convex problem with obstacles as areas of high cost (not as hard constraints). We did not attempt to tune CHOMP to only find feasible plans (e.g., by buffering the arm), since this incurs a tradeoff between safety and performance. Note, in MoveIt, infeasible CHOMP plans are not executed (if detected by an external collision-checker).

We report the mean solve time (MST) of ARMTD over all planning iterations, while the MST for CHOMP is the mean over all 100 tasks. Directly comparing timing is not possible since ARMTD and CHOMP use different planning paradigms; we report MST to confirm ARMTD is capable of real-time planning (note that that ARMTD's MST is less than $t_{\text{plan}} = 0.5$).

Random Obstacles	% goals	% crashes	MST [s]	MNPD
ARMTD + SL	84	0	0.273	1.076
CHOMP	82	18	0.177	1.511

TABLE I: MST is mean solve time (per planning iteration for ARMTD with a straight-line planner, total for CHOMP) and MNPD is mean normalized path distance. MNPD is only computed for trials where the task was successfully completed, i.e. the path was valid.

Hard Scenarios	1	2	3	4	5	6	7
ARMTD + SL	S	S	S	S	S	S	S
ARMTD + RRT*	O	O	O	S	O	S	O
CHOMP	C	C	C	C	C	C	C

TABLE II: Results for the seven Hard Scenario simulations. ARMTD uses straight-line (SL) and RRT* HLPs. The entries are "O" for task completed, "C" for a crash, or "S" for stopping safely without reaching the goal.

We also report the mean normalized path distance (MNPD) of the plans produced by each planner (the mean is taken over all 100 tasks). The normalized path distance is a path's total distance (in configuration space), divided by the distance between the start and goal. For example, the straight line from start to goal has a (unitless) normalized path distance of 1. ARMTD's MNPD is 24% smaller than CHOMP's, which may be because CHOMP's cost rewards path smoothness, whereas ARMTD's cost rewards reaching an intermediate waypoint at each planning iteration (note, path smoothness could be included in ARMTD's cost function).

Table II presents results for the Hard Scenarios. With the straight-line HLP, ARMTD does not complete any of the tasks but also has no collisions. With the RRT* HLP [35], ARMTD completes 5/7 scenarios. CHOMP converges to trajectories with collisions in all of the Hard Scenarios.

C. Hardware

See our video: `youtu.be/ySnux2owlAA`. ARMTD completes arbitrary tasks while safely navigating the Fetch arm around obstacles in scenarios similar to Hard Scenarios (1) and (4). We demonstrate real-time planning by suddenly introducing obstacles (a box, a vase, and a quadrotor) in front of the moving arm. The obstacles are tracked using motion capture, and treated as static in each planning iteration. Since ARMTD performs receding-horizon planning, it can react to the sudden obstacle appearance and continue planning without crashing.

VI. CONCLUSION

This work proposes ARMTD as a real-time, receding-horizon manipulator trajectory planner with safety guarantees. The method proposes novel reachable sets for arms, which enable safety. ARMTD can enforce safety on top of an unsafe path planner such as RRT*, shown in both simulation and on hardware. Of course, ARMTD has limitations: it may not perform in real time without parallelization, is only demonstrated on 6-DOF planning problems, and has not yet been demonstrated planning around humans. However, because ARMTD uses time-varying reachable sets, it can readily extend to dynamic environments, uncertainty such as tracking error, and planning with grasped objects. The results in this work show promise for practical, safe robotic arm trajectory planning.

References

[1] F. Pfeiffer and R. Johanni, "A concept for manipulator trajectory planning," *IEEE Journal on Robotics and Automation*, vol. 3, no. 2, pp. 115–123, Apr. 1987.

[2] T. Kunz and M. Stilman, "Time-Optimal Trajectory Generation for Path Following with Bounded Acceleration and Velocity," in *Robotics: Science and Systems*, 2012.

[3] S. M. LaValle and J. J. Kuffner Jr., "Randomized kinodynamic planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[4] 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.

[5] S. M. LaValle, *Planning Algorithms*. New York, NY, USA: Cambridge University Press, 2006.

[6] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa, "Chomp: Covariant hamiltonian optimization for motion planning," *The International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1164–1193, 2013.

[7] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel, "Motion planning with sequential convex optimization and convex collision checking," *The International Journal of Robotics Research*, vol. 33, no. 9, pp. 1251–1270, 2014.

[8] C. Park, J. Pan, and D. Manocha, "ITOMP: Incremental Trajectory Optimization for Real-Time Replanning in Dynamic Environments," 2012.

[9] S. Murray, W. Floyd-Jones, Y. Qi, D. J. Sorin, and G. Konidaris, "Robot Motion Planning on a Chip," in *Robotics: Science and Systems*, 2016.

[10] T. Kunz, U. Reiser, M. Stilman, and A. Verl, "Real-time path planning for a robot arm in changing environments," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct. 2010, pp. 5906–5911.

[11] K. Hauser, "On responsiveness, safety, and completeness in real-time motion planning," *Autonomous Robots*, vol. 32, no. 1, pp. 35–48, Jan. 2012.

[12] A. Majumdar and R. Tedrake, "Funnel libraries for real-time robust feedback motion planning," *arXiv preprint arXiv:1601.04037*, 2016.

[13] 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," *ArXiv e-prints arXiv:1809.06746*, Sep. 2018.

[14] S. Vaskov, S. Kousik, H. Larson, F. Bu, J. R. Ward, S. Worrall, M. Johnson-Roberson, and R. Vasudevan, "Towards Provably Not-At-Fault Control of Autonomous Robots in Arbitrary Dynamic Environments," in *Proceedings of Robotics: Science and Systems*, FreiburgimBreisgau, Germany, Jun. 2019.

[15] S. Kousik, P. Holmes, and R. Vasudevan, "Safe, Aggressive Quadrotor Flight via Reachability-Based Trajectory Design," Dynamic Systems and Control Conference, vol. 3, Oct. 2019, V003T19A010.

[16] M. Althoff, A. Giusti, S. B. Liu, and A. Pereira, "Effortless creation of safe robots from modules through self-programming and self-verification," *Science Robotics*, vol. 4, no. 31, 2019.

[17] T. G. A. A. A. Singletary P. Nilsson, "Online Active Safety for Robotic Manipulators," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Nov. 2019.

[18] S. L. Herbert, M. Chen, S. Han, S. Bansal, J. F. Fisac, and C. J. Tomlin, "FaSTrack: A modular framework for fast and guaranteed safe motion planning," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, Dec. 2017, pp. 1517–1522.

[19] M. Chen, S. L. Herbert, M. S. Vashishtha, S. Bansal, and C. J. Tomlin, "Decomposition of Reachable Sets and Tubes for a Class of Nonlinear Systems," *IEEE Transactions on Automatic Control*, vol. 63, no. 11, pp. 3675–3688, Nov. 2018.

[20] T. Fraichard and J. J. Kuffner, "Guaranteeing motion safety for robots," *Autonomous Robots*, vol. 32, no. 3, pp. 173–175, 2012.

[21] A. Majumdar, A. A. Ahmadi, and R. Tedrake, "Control and verification of high-dimensional systems with dsos and sdsos programming," in *53rd IEEE Conference on Decision and Control*, IEEE, 2014, pp. 394–401.

[22] B. Paden and R. Panja, "Globally asymptotically stable 'PD+'controller for robot manipulators," *International Journal of Control*, vol. 47, no. 6, pp. 1697–1712, 1988.

[23] A. Giusti and M. Althoff, "Efficient Computation of Interval-Arithmetic-Based Robust Controllers for Rigid Robots," in *2017 First IEEE International Conference on Robotic Computing (IRC)*, Apr. 2017, pp. 129–135.

[24] A. Girard, "Reachability of uncertain linear systems using zonotopes," in *International Workshop on Hybrid Systems: Computation and Control*, Springer, 2005, pp. 291–305.

[25] M. Althoff, "An Introduction to CORA 2015," in *Proc. of the Workshop on Applied Verification for Continuous and Hybrid Systems*, 2015.

[26] M. Althoff, "Reachability analysis and its application to the safety assessment of autonomous cars," PhD thesis, Technische Universität München, 2010.

[27] D. Meagher, "Geometric modeling using octree encoding," *Computer graphics and image processing*, vol. 19, no. 2, pp. 129–147, 1982.

[28] J.-M. Lien and N. M. Amato, "Approximate convex decomposition of polyhedra," in *Proceedings of the 2007 ACM symposium on Solid and physical modeling*, 2007, pp. 121–131.

[29] L. J. Guibas, A. Nguyen, and L. Zhang, "Zonotopes as bounding volumes," in *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, Society for Industrial and Applied Mathematics, 2003, pp. 803–812.

[30] S. Boyd, L. Xiao, and A. Mutapcic, "Subgradient methods," *lecture notes of EE392o, Stanford University, Autumn Quarter*, vol. 2004, pp. 2004–2005, 2003.

[31] E. Polak, *Optimization: algorithms and consistent approximations*. Springer Science & Business Media, 2012, vol. 124.

[32] M. Wise, M. Ferguson, D. King, E. Diehr, and D. Dymesich, "Fetch and freight: Standard platforms for service robot applications," in *Workshop on Autonomous Mobile Service Robots*, 2016.

[33] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA workshop on open source software*, Kobe, Japan, vol. 3, 2009, p. 5.

[34] 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.

[35] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[36] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical programming*, vol. 106, no. 1, pp. 25–57, 2006.