Online Search-Based Collision-Inclusive Motion Planning and Control for Impact-Resilient Mobile Robots

Zhouchu Lu, Zhichao Liu, Merrick Campbell, Member, IEEE, and Konstantinos Karydis, Member, IEEE

Abstract—This article focuses on the emerging paradigm shift of collision-inclusive motion planning and control for impact-resilient mobile robots, and develops a unified hierarchical framework for navigation in unknown and partially observable cluttered spaces. At the lower level, we develop a deformation recovery control and trajectory replanning strategy that handles collisions that may occur at run time, locally. The low-level system actively detects collisions (via embedded Hall effect sensors on a mobile robot built in-house), enables the robot to recover from them, and locally adjusts the postimpact trajectory. Then, at the higher level, we propose a search-based planning algorithm to determine how to best utilize potential collisions to improve certain metrics, such as control energy and computational time. Our method builds upon A* with jump points. We generate a novel heuristic function, and a collision checking and adjustment technique, thus making the A* algorithm converge faster to reach the goal by exploiting and utilizing possible collisions. The overall hierarchical framework generated by combining the global A* algorithm and the local deformation recovery and replanning strategy, as well as individual components of this framework, are tested extensively both in simulation and experimentally. An ablation study draws links to related state-of-the-art search-based collision-avoidance planners (for the overall framework), as well as search-based collision-avoidance and sampling-based collision-inclusive global planners (for the higher level). Results demonstrate our method’s efficacy for collision-inclusive motion planning and control in unknown environments with isolated obstacles for a class of impact-resilient robots operating in 2-D.

Index Terms—Collision-inclusive motion planning and control, motion and path planning, reactive and sensor-based planning, wheeled robots.

I. INTRODUCTION

THERE has been an emerging paradigm shift in mobile robot motion planning and autonomous navigation whereby collisions with obstacles are not by default avoided but instead exploited to improve certain robot planning, control, and navigation metrics [1], [2], [3], [4], [5]. Such collision-inclusive planning and control strategies capitalize on results demonstrating how some forms of collisions can in fact be useful in terms of sensing, localization, control, and agility [6], [7], [8], [9], [10], [11], [12], [13], [14]. Besides the benefits of embracing collisions, robot deployment in realistic (that is, dynamic, cluttered, and irregularly shaped) environments may, at cases, make collision avoidance hard to achieve [15], [16]. For example, detecting all obstacles in the environment can be a challenge, especially when there exist translucent and/or transparent obstacles, such as glass walls, or reflective surfaces [13]. At the same time, using a conservative local planner may fail finding a feasible path to the goal even if one exists [17]. Collision-inclusive motion planners can help address the aforementioned challenges.

Although research on collision-inclusive motion planning has already begun receiving attention, existing methods can be limited in their ways to apply in practical cases. On one hand, methods that evaluate the effect of collision within motion planning [3], [4] do not apply to online problems. On the other hand, existing online collision-inclusive planning methods [9], [11], [18] cannot decide how to use collisions optimally, which could help guide the robot to the goal. Our previous online planning method [2] can evaluate possible collisions in unknown space, which lies outside the field-of-view (FoV) of the robot, but does not consider how to employ collisions optimally within the known (and/or visible) space.

In this article, we propose a unified online collision-inclusive motion planning and control framework that evaluates the effect of possible collisions and decides when it might be preferred to collide with an obstacle (or a surface more broadly) instead of avoiding it. Our framework applies to impact-resilient robots with three core capabilities: collision resilience, collision identification, and postimpact characterization. We design and fabricate in-house a custom omnidirectional holonomic wheeled robot equipped with a collision ring that integrates Hall effect sensors along the arms holding the ring in place (see Fig. I); our robot satisfies all three core capabilities. The robot runs a deformation recovery control and trajectory replanning (DRR) strategy [20] that enables it to recover from a collision and rapidly replan its postimpact trajectory using the information provided by the Hall effect sensors. The DRR strategy acts as

1552-3098 © 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
The local replanner of the unified framework developed herein. We also propose and develop a global search-based planning algorithm based on the collision model generated from the DRR strategy. Similar to [21], our approach explores the space of trajectories using a set of short-duration motion primitives generated by solving an optimal control problem. Instead of pruning those primitives colliding with the obstacles in the global map, our proposed approach can adjust and evaluate them based on the collision model.

Succinctly, this article’s contributions are as follows.

1) We extend the DRR strategy to generate local trajectories when colliding with (non-)convex obstacles.
2) We develop a search-based planner to generate global trajectories and evaluate it in different benchmarks.
3) We propose and evaluate a unified online collision-inclusive motion planning and control framework integrating the DRR strategy and search-based collision-inclusive planning while considering the robot’s FoV.

Our method is systematically evaluated via both simulated and real-world experiments using planar holonomic wheeled robot kinematics in environments that contain isolated convex and nonconvex obstacles. We first test the DRR strategy experimentally to ensure its feasibility and safety when applied to the physical robot. Data collected from this process help identify parameters for the collision model, which is necessary to test the search-based collision-inclusive algorithm in simulation. Comprehensive benchmark comparisons against state-of-art collision-avoidance and collision-inclusive methods demonstrate the differences, similarities and the utility of specific components, as well as of the overall proposed framework. Moreover, experimentation with the physical robot in a single corridor environment is conducted to validate the performance of our unified online collision-inclusive motion planning and control framework.

This article builds upon and significantly extends previous results [2], [20]. Lu et al. [2] focus only at the global planning level and evaluates possible collisions in the unknown (not yet observed) space. Lu et al. [20] focus only at the local control and planning level that utilizes DRR based on a priori given waypoints. This article, in contrast, develops the unified framework that combines the global planning and local control and planning levels together. In this newly developed approach, the global planner can evaluate possible collisions both within and outside the robot’s FoV.

In what follows, we review related works in Section II and introduce our overall system’s structure in Section III. The deformation recovery control and postimpact trajectory replanning components are detailed in Sections IV and V, respectively. The global search-based planner is discussed in Section VI. Extensive benchmark (in simulation) and experimental results are given in Section VII. Finally, Section VIII conclude this article.

II. RELATED WORKS

Collision-free motion planning algorithms handle obstacle avoidance in distinct ways (e.g., [21], [22], [23], [24], and [25]) to derive collision-free trajectories in real time. Typically, such methods split the trajectory generation problem into two parts, which are as follows:

1) planning a collision-free geometric path or using motion primitives;
2) optimizing the path locally to obtain a dynamically feasible time-parameterized trajectory.

When the environment is unknown (or partially known), different strategies have been used based on those two-part framework. Many methods adopt the optimistic assumption [26], [27], which treats the unknown space as collision free. This strategy improves the speed of reaching goals but may not guarantee safety. In contrast, other methods treat the unknown space as obstacle occupied [28] and only allow for motions within the already known free space or FoV-observed free space [29]. Although these restrictions can help ensure safety, they tend to lead to conservative motion.

1To make this present article self-contained, important methods and results from the previous related articles [2], [20] are included herein.
Tordesillas et al. [24] proposed a method that combines these two strategies by planning in both the known-free and unknown spaces. Instead of being overly optimistic about the unknown space, backup trajectories are also planned to enforce safety should the assumption about unknown space being free turns out to be wrong. While this method works well overall, it has put less emphasis on environment perception, lack of which may reduce safety or create overconservative trajectories when the robot is tasked to operate at high speeds [30]. To this end, perception-aware strategies [23], [31], [32] have been proposed to predict unknown dangers and try to discover and avoid those dangers early on. However, prediction of unknown dangers does not necessarily ensure accuracy and usually requires additional computational effort that may limit online implementation.

Different from collision avoidance, there have been efforts on designing impact-resilient robots that can withstand collisions instead (e.g., [2], [4], [11], [13], [14], [18], [19], [20], [33], [34], and [35]). With such robots as hand, one research direction has been to design characterization methods that can make the robot sense the collision and recover from the collision state. Most of such characterization methods have mainly focused on utilizing data from an onboard inertial measurement unit (IMU) [36]. However, IMUs are usually unable to distinguish collisions during aggressive maneuvers and to detect static contacts, resulting in low accuracy in collision detection. Sensors that could detect deformation of the robot during the collision process have been used in the past to provide more accurate collision detection [18], [37]. In related yet distinct previous work [14], we have implemented a passive quadrotor arm design with Hall effect sensors, making the robot able to detect and characterize collisions. The ability to sense and characterize collisions has led to various different methods to replan the local trajectory once the collision is detected [9], [11], [18]. Planning methods using motion patterns, e.g., to move forward in straight lines until collision with environment boundaries, and then, rotate in place and move forward again, have also been proposed [38], [39], [40]. Such methods can run online in environments with non-convex, polygon-shaped obstacles. A different trajectory generation method can be achieved by assuming the robot maintains contact with the obstacle [12]. Although these methods increase robustness and safety of postimpact trajectories, they cannot determine where the robot should collide with the environment to help it redirect toward the globally planned goal.

Related works [3] and [4] propose methods to evaluate and design possible collision spots of the global trajectory. Mote et al. [3] have introduced an empirical algebraic collision model by directly relating pre- and postimpact velocities with no thrust commanded. Then, a mixed-integer planning method based on that model is used to compute collision-inclusive trajectories in a known environment. However, integer constraints are hard to create, and solving a mixed-integer programming problem is usually time consuming, making it impossible to run online for planning in an unknown environments. Furthermore, the approach [3] has been demonstrated with a specific pair of objects over a relatively limited range of conditions (obstacles need to be line segments). Zha and Mueller [4] have proposed a rapidly exploring random tree (RRT)-based planning method to plan global trajectories with collisions. The impulsive model is used to create the postcollision state once the precollision state is generated. Findings from [4] suggest that a collision-inclusive sampling-based planner is likely to find better trajectories in cluttered environments (such as narrow tunnels) as compared to environments that contain isolated obstacles. In addition, such algorithms remain limited in their use for online planning in unknown (or partially known) environments.

Compared to our previous work [2], which developed a global path planner that explicitly tradesoff between risk and collision exploitation only in unknown space, this article proposes a new search-based global planner using a set of short-duration motion primitives, which exploit possible collisions in the environment. The planner treats the unknown space as collision free. Our planner can generate waypoints with explicit information about possible collisions and more reasonable time allocation for the local trajectory generator. Contrary to collision avoidance methods with hard constraints that generate trajectories only in conservative local space [22], [24], our method utilizes gradient-based trajectory optimization (GTO) [23], which typically formulates trajectory generation as a nonlinear optimization problem and incorporates the artificial potential field (APF) to ensure safety. However, since GTO does not guarantee the robot will avoid all possible collisions, especially in unknown environments, we utilize the DRR strategy [20] for local trajectory generation; once a collision is sensed and characterized, DRR can ensure that the robot will recover from the collision and keep progressing toward its global goal.

III. SYSTEM OVERVIEW

A. Overall Framework

Our overall system architecture is shown in Fig. 2. Novel contributions relate to the low-level planner (see Sections IV
and V) and the high-level planner (see Section VI) in partially known environments. The robot may collide with obstacles that were not detected at any time instant that the map (e.g., provided via LiDAR scans) refreshes. Instead of stopping when sensing the collision, the robot locally refines the trajectory and continues to explore the unknown space. To avoid repeated collisions with an obstacle (reminiscent of stacking into local minima), if another collision occurs while the robot follows the locally revised trajectory part, the robot will then stop and invoke the high-level global planner to make more substantial refinements to the trajectory. Both processes run online.

Contrary to collision avoidance algorithms, we do not impose any obstacle-related constraints in trajectory generation, nor do we run a geometric collision check once a trajectory is generated at the low level. Instead, we directly generate a trajectory based on given waypoints. If a collision occurs, the robot receives a signal that a collision has occurred from any of the Hall effect sensors embedded between the main chassis and its deflection surfaces and activates a deformation recovery controller. The controller (see Section IV) makes the robot detach from the collision surface by recovering from the deformation, and determines a postcollision state for the robot so as to facilitate postimpact trajectory replanning. The replanner (see Section V) refines the initial trajectory since collisions change the second-order continuity of the trajectory followed before collision. To do so, the replanner uses the postcollision state computed by the recovery controller as initial state for the refined trajectory generation. The procedure repeats as new collisions occur in the future, in a reactive and online manner. Fig. 3 shows the DRR strategy, along with specific implementation components for experimentation.

We select GTO for postimpact recovery and global trajectory refinement based on [23] that revealed that GTO-based methods are particularly effective for local replanning, which is key for high-speed online motion planning in unknown environments. One drawback of GTO is the presence of local minima that may lead to undesirable solutions. Specifically, GTO may yield a trajectory that intersects with the obstacles in the environment [23]. Our DRR strategy can resolve this issue by offering a way to run a quick replan locally after the collision happens to ensure postimpact consistency.

### B. Problem Assumptions and Notation

The proposed approach applies under the following.

1) The boundary of the environment is known.
2) Operating environments attain the form of confined corridors with isolated convex and nonconvex obstacles, and only planar collisions obstacles are considered.
3) During deformation and until a collided arm recovers its initial length, the tip of the arm remains in contact with the collision surface but does not rotate about the z-axis, and the wheels of the robot contact the ground.
4) The Hall effect sensor can return the information of collision state timely.

Key notation used in this article is shown in Table I.

### IV. DEFORMATION RECOVERY CONTROL

The purpose of our proposed deformation controller is to make the robot recover from a collision and reach a postimpact state that can facilitate the recovery trajectory replanning (which we discuss in the next section).

| Key Notation | Definition |
|--------------|------------|
| $\tau$       | time interval |
| $t_c$        | time instance when the sensing collision |
| $T_r$        | time horizon of deformation recovery |
| $T_{rep}$    | time horizon of replanning |
| $R^{w}_b$    | Rotation matrix from body frame to world frame |
| $R^{w}_c$    | Rotation matrix from collision frame $F_c$ to world frame |
| $F$          | state transition matrix of deformation controller |
| $G$          | input to state matrix of deformation controller |
| $C$          | mapping matrix of polynomial coefficient $\eta$ |
| $Q$          | cost matrix of smoothness term |
| $A_f$        | state transition matrix in free space |
| $B_f$        | input to state matrix related to $a_d$ in free space |
| $J_s$        | smoothness term objective function |
| $J_c$        | objective function of the clearance |
| $J_p$        | penalty on velocity |
| $J_a$        | penalty on acceleration |

---

$^2$The list of waypoints can be computed via any path planning method. It is independent from our proposed collision-inclusive planning algorithm.
Consider a holonomic mobile robot [see Fig. 1(c)], modeled as a point mass $m$. The robot’s main chassis is connected to its deflection surfaces via viscoelastic prismatic joints (see Fig. 4). Note that the springs inside each joint are pretensioned. The robot’s compliant arms can both protect the robot from collision damage, and generate an external force driving it away from obstacles. External forces along each arm are caused via viscoelastic deformations assumed to follow the Voigt model; $k_c$ and $k_d$ denote the spring constant and damping coefficient, respectively. Hall effect sensors are used to measure the amount of deformation along each arm, and to signal collision detection when a user-tuned arm compression threshold is exceeded.\(^3\) The arm design with the bump sensor mechanism is similar to the button mechanism [41] and helps protect the robot from damage caused by collision as well as sense the collision in real time. Collision detection accuracy is related to the number of arms on the robot.

We consider four key quantities related to spring lengths: neutral $l_0$, pretensioned $l_s$, maximum-load $l_c$, and current $l$ (also referred to as deformation vector). These quantities play a significant role in the deformation recovery controller; they are also summarized in Table I, along with other key notation. In single-arm collisions, current spring length vector $l$ is aligned with the unit vector along the colliding arm, pointing from the tip of the arm to the center of the robot along the compliant prismatic joint. For clarity of presentation, we consider in the following single-arm collisions. In multiarm collisions, we compute individual contributions from each colliding arm’s spring, and then, consider their vector sum as the compound deformation vector used in lieu of $l$.

We use three coordinate systems. The world and body frames ($^wR$ denotes the rotation matrix from body to world frames, while $^bR$ denotes the deformation vector expressed in the body frame), and a (local) collision frame $^cF$. This frame is defined at the time instant a collision occurs, $t_c$, and remains fixed for throughout the collision recovery process, $T_c$. Its origin coincides with the origin of the robot when a collision is detected. Basis vector $\{n, t, k\}$ of $^cF$ are defined normal, tangent, and upwards with respect to the deformation vector $l$. Let $\theta$ be the angle of deformation vector $l$ in $^cF$.\(^4\)

The (frame-agnostic) robot collision dynamics is given by

$$m\ddot{l} + k_c\dot{l} + k_d(l-l_0) = ma_{in},$$

where $a_{in}$ is the robot’s body acceleration input as provided by the robot’s motors.

### B. Deformation Controller

The deformation recovery controller’s task is to steer the postimpact state of the robot to a desired one within a time period of $[t_c, t_c + T_c]$. The time horizon $T_c$ is an important hyperparameter tuned by the user. Typically, longer $T_c$ means the robot will recover from collision with longer time and smoother motion pattern. Through a preliminary calibration phase we selected $T_c = 0.5$ s.

The deformation controller operates with respect to the local, collision frame $^cF$. Let the state variable be $^c\mathbf{s} = \begin{bmatrix} \mathbf{p}_x \mathbf{v}_x \end{bmatrix}^T$. The control input is $\mathbf{u} = \begin{bmatrix} u_x & u_y & u_\theta \end{bmatrix}$, where $u_x = (\mathbf{a}_{in} + \frac{k_s}{m}(l_s-l_0)) \cdot n$, $u_y = (\mathbf{a}_{in} - \frac{k_s}{m}(l_s-l_0)) \cdot t$, and $u_\theta = \mathbf{\omega} \cdot \mathbf{k}$ with $^c\mathbf{\omega}$ being the angular velocity of the robot in the collision frame. Note that position control terms include compensation for the force caused by the spring being pretensioned when the robot’s arm is at its rest length. Then, the state-space model of the robot recovering from collision can be expressed as

\[
\begin{bmatrix}
\dot{p}_x \\
\dot{p}_y \\
\dot{\theta}
\end{bmatrix} =
\begin{bmatrix}
v_x \\
v_y \\ \mathbf{u}_\theta
\end{bmatrix},
\]

\[
\begin{bmatrix}
\dot{v}_x \\
\dot{v}_y \\
\mathbf{v}_\theta
\end{bmatrix} =
\begin{bmatrix}
-k_x \mathbf{v}_x + \frac{k_y}{m} \mathbf{v}_y + u_x \\
\frac{k_x}{m} (\mu \text{sign}(v_y) + \tan \theta) \mathbf{v}_x + f_0 - k_d (\mu \text{sign}(v_y) + \tan \theta) \mathbf{v}_y + u_y
\end{bmatrix}
\]

where $f_0 = \mu k_d \text{sign}(v_y)(l_s - l_0) \cdot n$.

Since the robot is holonomic, we can decouple orientation from position control.\(^5\) The orientation and angular velocity errors during recovery time $t \in [t_c, t_c + T_c]$ are $e_R(t) = \frac{1}{2}(^wR^cR - ^wR^d)^n$ and $e_\theta(t) = \omega - ^wR^d \omega_d$, respectively.\(^6\) Index $d$ denotes desired quantities; these are $R_d = R(t_c)$ and $\omega_d = [0 \ 0 \ 0]^T$. (All terms are with respect to collision frame

\[^4\]Note that inability to define the deformation vector may make the collision frame ill-defined. There are three special cases for this to happen. One is when two opposite arms deform exactly equally. In this case, there are two possible solutions to define the direction of the deformation vector along the line connecting the two arms. However, our algorithm still works as it prioritizes motion along the tangent to the collision vector (this would be the case of going through a very narrow curvy corridor). The second case contains asymmetric collisions with three or more arms such that the vector sum is still zero. Then, one can define the collision frame based on the most dominant (in terms of magnitude) individual collision vector. Our algorithm can still work, though it is possible that more solutions to define will occur as the robot tries to navigate through (this would be the case of going through a very narrow curve corridor). The last case is when there is an even (four or greater) number of symmetric collisions (this would be the case of going through a very narrow curvy corridor). The second case contains asymmetric collisions with three or more arms such that the vector sum is still zero. Then, one can define the collision frame based on the most dominant (in terms of magnitude) individual collision vector. Our algorithm can still work, though it is possible that more solutions to define will occur as the robot tries to navigate through (this would be the case of going through a very narrow curve corridor). The last case is when there is an even (four or greater) number of symmetric collisions (this would be the case of going through a very narrow curvy corridor).

\[^5\]In our approach, we seek to make the robot keep the same orientation it has at the instant it collides throughout the collision recovery process. We follow this approach because it can simplify the overall deformation recovery control problem without sacrificing optimality.

\[^6\]The vee map $\vee$ is the inverse of a skew-symmetric mapping.
\( \mathcal{F}_c \). Then
\[
    u_\theta = -K_\theta e_{R,z}(t) - K_c e_{R,z}(t) .
\] (2)

Note that since this is a planar collision problem, the collision recovery orientation controller considers only the z-components of orientation and angular velocity errors.

Regarding collision recovery position control, note that the translation-only motion in (1) is affine. Thus, we can apply feedback linearization. The linearized system matrix \( F \) is
\[
    F = \begin{bmatrix}
        0 & 0 & 1 & 0 \\
        0 & 0 & 0 & 1 \\
        -\frac{k_x}{m} & 0 & -\frac{k_z}{m} & 0 \\
        0 & 0 & 0 & 0
    \end{bmatrix}
\]
with state vector \( s_d = [p_x \ p_y \ v_x \ v_y]^\top \). The control input matrix is \( G = I_{2 \times 2} \) with control input vector \( \nu = [v_x \ v_y] \) given by
\[
    \begin{cases}
    v_x = u_x \\
    v_y = -\frac{k_x}{m} (\text{sign}(v_x) + \tan \theta) x_f + \frac{k_z}{m} (\text{sign}(v_y) + \tan \theta) v_x,
    \end{cases}
\] (3)

We formulate an optimal control problem with fixed time horizon \( T \) based on the linearized system \( \dot{x} = Fx + G\nu \). Using the change of variable \( \tau = t - t_c \), we seek to solve
\[
    \min_{s_d} \int_0^{T_c} \left( s_d(\tau)^\top \Gamma s_d(\tau) + \nu^\top(\tau)H\nu(\tau) \right) d\tau \tag{4a}
\]
\( \text{subject to } \quad \dot{s}_d = F s_d + G \nu \tag{4b} \)
\( \quad \quad -\|l_c - l_s\| \cos \theta \leq p_x \leq 0 \) \hspace{1cm} (4c)
\( \quad s_d(0) = [u_0, 0, v_0, x_0, y_0]^\top \) \hspace{1cm} (4d)
\( \quad s_d(T_c) = [0, p_{T,x}, v_{T,x}, v_{T,y}] \) \hspace{1cm} (4e)

Matrices \( \Gamma = \left[ \begin{array}{cc} l_{2 \times 2} & 0 \\ 0 & l_{2 \times 2} \end{array} \right] \) and \( H = h I_{2 \times 2} \) penalize the displacement during the recovery process and the control input, respectively. There is a tradeoff between the displacement and the control input of the robot. Tuning parameters \( \gamma \) and \( h \) balance this tradeoff to select the controller with minimal control energy and displacement.

Constraint (4c) dictates that the robot should be in contact with the collision surface until the colliding arm’s spring has recovered its original, pretensioned length \( l_s \) (i.e., the arm is no longer compressed) without compressing beyond its linear region \( l_c \). Constraints (4d) and (4e) enforce initial and terminal position and velocity conditions, respectively. In detail, \( p_{0,x} \) is determined by the colliding arm’s Hall effector sensor reading. Since the vector form of the sensor’s reading (that is, \( b_l - b_l \)) is expressed in the body frame, we need transform it to the collision frame \( \mathcal{F}_c \) as per
\[
    p_{0,x} = -[1 \ 0] \ c R^\top \ b R (b_l - b_l) .
\] (5)

We employ this change of variable for clarity. The Problem (4) resets every time a new collision occurs; this gives rise to an LTI system, hence, the change of variable can apply.

Algorithm 1: Deformation Recovery Controller

1. **Function** RECOVERY_CONTROLLER \( \{ b_l - b_l, \ \tau_c, \ \Delta t_e, \ \Gamma, \ \nu \} \)
2. \( \quad \) \( \Delta t_e \leftarrow \nu \ p_{T,x}, \ \nu \ p_{T,y}, \ \nu \ v_{T,x}, \ \nu \ v_{T,y}, \ \nu \ v_{T} \)
3. \( \quad \) \( \nu \nu \leftarrow \nu \ R^\top \ \nu \nu \)
4. \( \quad \) \( \nu \nu \leftarrow \nu \ \text{max normalize}( \ \nu \nu \)
5. \( \quad \) \( \nu \nu \leftarrow 0 \)
6. \( \quad \) \( \nu \nu \leftarrow \nu \gamma \nu \left( [u_0, v_0, x_0, y_0] \right) \)

The velocity components at the collision instant \( v_{0,x} \) and \( v_{0,y} \) are expressed in frame \( \mathcal{F}_c \), and are estimated at run time.\(^8\) Postimpact, the arm needs to be uncompressed (hence, \( p_{T,x} \) is set to 0), but \( p_{T,y} \) is treated as an unconstrained free variable. Postimpact terminal velocity components \( v_{T,x} \) and \( v_{T,y} \) are also expressed in \( \mathcal{F}_c \) and can be set freely. In Section V, we discuss how to generate \( v_{T,x} \) and \( v_{T,y} \) based on the preplanned trajectory. We discretize the linearized system in (4b) with sampling frequency \( f = 10 \) Hz using the Euler method, and solve the corresponding quadratic program with CVXOPT. The process is summarized in Algorithm 1.

Computed control inputs (2) and (4) make the robot detach from the collision surface and help bring it to a temporary postcollision state, which can be used as the initial condition for postimpact trajectory generation. We discuss this next.

V. POSTIMPACT TRAJECTORY REPLANNING

A. Problem Formulation

We formulate the postimpact trajectory generation problem as a quadratic program with equality constraints, i.e.,
\[
    \min_{\eta} J_x(\eta) = \sum_{i=1}^{N_f} \Delta t_i \left\| p_i^{(q)}(t) \right\| dt \tag{6a}
\]

\(^8\)In the experiments conducted in this work, velocity measurements are provided via a motion capture camera system, but the method applies as long as velocity estimates are available, e.g., via optical flow.
subject to
\[ C_{0,i,c,β}^{(0)} I_{i,c,β} = w p_{r,β}, \]  
\[ C_{0,i,c,β}^{(1)} I_{i,c,β} = w v_{r,β}, \]  
\[ C_{Δt_N i,1,β}^{(α)} N_{i,β} N_{i,1,β} = w d_{i,β}(α) \]  
\[ α = \{0, 1 \ldots q - 1\} \]  
\[ C_{Δt_{i+1} i,β}^{(α)} I_{i+1,β} = c_{i+1,β}\]  
\[ α = \{1, 2 \ldots q - 1\}. \]

For polynomial segments, we can rewrite \( J_s \) as
\[ J_s = \sum_{i = i_c}^{N_1} \sum_{β \in \{x,y\}} I_{i,β}^\top Q_β^i(Δt_i) I_{i,β} \]
where \( i_c \) is the segment where the collision happens and \( N_I \) is the number of trajectory segments. Superscript \( q \) denotes the derivative order; for example, \( q = \{1, 2, 3, 4\} \) correspond to min-velocity, min-acceleration, min-jerk, and min-snap trajectories, respectively. Subscript \( β \in \{x,y\} \) indicates the \( x \) and \( y \) components of the trajectory, and \( Δt_i \) is the time duration for \( i \)th polynomial segment. Parameter \( I_{i,β} \) is the vector of coefficients of \( i \)th polynomial. \( C_{Δt_{i+1} i,β}^{(α)} \) maps the coefficients to \( α \)-th order derivative of the start point in segment \( i \), while \( C_{Δt_{i+1} i,β}^{(α)} \) maps the coefficients to the \( α \)-th order derivative of the end point in the segment \( i \).

Constraints (6b) and (6c) impose the initial values for the 0th and the first-order derivatives to match the position and velocity values attained via the collision recovery controller, respectively. Constraint (6d) imposes that the \( α \)-th order derivatives of the end position are fixed. Constraint (6e) imposes that the trajectory will pass through desired waypoints after \( i_c \). Constraint (6f) is imposed to ensure \( α \)-th continuity among polynomial segments.

We solve this quadratic programming (QP) problem given initial (postcollision) and end states, and intermediate waypoints. Then, we perform time scaling as in [22] to reduce the maximum values for planned velocities, accelerations, and higher order derivatives as appropriate, and thus, improve dynamic feasibility of the refined postimpact trajectory.

The solution of the QP problem serves as the initial value for GTO [42], where we change the objective function to
\[ \min \quad λ_s J_s + λ_o J_o + λ_d (J_v + J_a) \]  
where \( J_o \) is the cost to avoid collisions, and \( J_v \) and \( J_a \) are the penalties when candidate velocity and acceleration solutions exceed the dynamic feasibility limit, respectively. Weight parameters \( λ_s \), \( λ_o \), and \( λ_d \) tradeoff between smoothness, trajectory clearance, and dynamical feasibility, respectively.

Similar to [42], we use an exponential cost function. At a position with distance \( d \) to the closest obstacle, the cost \( c_o(d) \) is written as
\[ c_o(d) = α_o \exp \left( -d - d_o \right) / γ_o \]  
where \( α_o \) is the magnitude of the cost function, \( d_o \) is the threshold where the cost starts to rapidly rise, and \( γ_o \) controls the rate of the function’s rise. Then, \( J_o \) can be computed as
\[ J_o = \sum_{i = i_c}^{N_I} \sum_{β \in \{x,y\}} J_o(\beta) \int_0^{Δt_i} c_o(p(t))∥v(t)∥dt \]  
\[ = \sum_{i = i_c}^{N_I} N \sum_{k = 0} c_o(p(t_k))∥v(t_k)∥dt. \]

The formulation of \( J_o \) is similar to (10). The cost function of the acceleration constraint \( c_a(α) \) is also an exponential function similar to \( c_o(d) \) and \( c_v(v) \), since it is can penalize when close to or beyond acceleration bounds while staying flat when away from the bounds. We apply a similar Newton trust region method as in [42] to optimize the objective.

### B. Waypoint Adjustment

In some cases, it may be necessary to adjust the waypoints given by a preplanned trajectory with the information obtained from the collision, and then, solve the aforementioned problem in Section V-A with the adjusted waypoints. Such cases occur when there is no direct line of sight between the collision state and the waypoint at the end of the immediately next trajectory segment following collision recovery. By enabling such waypoint adjustment, the algorithm promotes exploration, and in certain cases, prevents the robot from being trapped in a local minima in which repeated collisions at the same (or very close-by) place could otherwise occur.

With reference to Algorithm 2, we express in the local collision frame \( F_c \), the next waypoint \text{waypoint list}[i_c + 1] (lines 2–4). In line 5, we adjust \text{waypoint list} with the information we get from collision. Details of this process are shown in Fig. 5. We add an additional waypoint \( p_{add} \) to create a path detouring the collided obstacle. Then, we select the shortest path among
Algorithm 2: Post-Impact Waypoint Adjustment.

input: Position after the collision recovery in world frame, \( w_p \); waypoint list of preplanned trajectory; \( \mathcal{w} \); \( \mathcal{R} \); trajectory segment \( i_c \) where the collision happens.

output: waypoint list after adjustment \( \mathcal{w}_{\text{pnext}} \)

parameter: Robot radius \( r_{rob} \)

1 Function \( \text{WAYPOINTADJUSTMENTLINE} (w_p, \mathcal{w} \mathcal{R}, \mathcal{w}_{\text{pnext}}, \mathcal{w}_{\text{p}}) \):

\[ \mathcal{w}_{\text{pnext}} \leftarrow \text{waypoint list} [\mathcal{w}_{\text{p}} + 1] \]

2 Transfer \( \mathcal{w}_{\text{pnext}} \) into \( \mathcal{R} \) frame to get \( \mathcal{w}_{\text{pnext}} \)

3 Transfer \( \mathcal{w}_{\text{p}} \) into \( \mathcal{R} \) frame to get \( \mathcal{w} \)

4 Adjust \( \mathcal{w}_{\text{pnext}} \) as Fig. 5

5 return \( \mathcal{w}_{\text{pnext}} \)

all the possible paths toward the next waypoint \( \mathcal{w}_{\text{pnext}} \) that was originally in the list before collision. Possible \( \mathcal{w}_{\text{padd}} \) waypoints are generated by either using a path generation algorithm (e.g., jump point search) when the complete collision surface can be perceived, or by searching along the y-axis of collision frame \( \mathcal{R} \) by a (user defined) exploration distance \( \epsilon_{\text{explore}} \) when the complete collision surface cannot be reliably perceived (e.g., via LiDAR measurements).

As the robot progresses and reaches the additional waypoint \( \mathcal{w}_{\text{padd}} \) that was added following the collision, it then replans based on latest information provided from the perception module. This happens when the robot either reaches the added waypoint (to ensure that the next waypoint is in free space) or it senses another collision from the deformation sensor. In this case, the original \( \mathcal{w}_{\text{pnext}} \) will change as well. Note that this process runs online. In the case that the robot senses a collision before reaching \( \mathcal{w}_{\text{padd}} \), then it will recover and stop (instead of running the fast replanning approach listed previously) and call the global planner to revise larger parts of the trajectory. If a new waypoint is inserted in the list, we map the path generated by \( w_p \) and waypoints in the list after \( i_c + 1 \) into time domain using a trapezoidal velocity profile. If no new waypoint is inserted, we set the time duration of \( i_c \) segment in (6) as \( \Delta t_{i_c} = t_{i_c + 1} - t_c \), where \( t_{i_c + 1} \) is the time reaching next waypoint \( \mathcal{w}_{i_c + 1} \) in the preplanned trajectory.

VI. SEARCH-BASED COLLISION-INCLUSIVE PLANNING

In this section, we propose the main algorithm to generate the waypoint list and trajectory segments that serve as the input to the DRR strategy.

A. Problem Formulation

Let the system state \( s_d(t) \in S \subset \mathbb{R}^{n \times q} \) contain the configuration and the \((q - 1)\)th-order derivatives in 2D (i.e. \( n = 2 \)). The free state space, \( S_{\text{free}} \subset S \), contains both obstacle-free configurations, \( D_{\text{free}} \), as well as the system’s dynamical constraints, \( P_{\text{free}} \), which include minimum and maximum bounds on velocity \([v_{\text{min}}, v_{\text{max}}]\), acceleration \([a_{\text{min}}, a_{\text{max}}]\), jerk \([j_{\text{min}}, j_{\text{max}}]\), and other higher-order derivatives. We can then write \( S_{\text{free}} = P_{\text{free}} \times D_{\text{free}} = P_{\text{free}} \times [v_{\text{min}}, v_{\text{max}}] \times [a_{\text{min}}, a_{\text{max}}] \times \ldots \times P_{\text{obs}} = P \setminus P_{\text{free}} \) and \( S_{\text{obs}} = P_{\text{obs}} \setminus D_{\text{free}} \) defines the obstacle region.

The differential flatness of some mobile robot systems (e.g., [21]) helps design control inputs from 1D time-parameterized polynomial trajectories independently for each of the \( n \) positions. Hence, \( s_d(t) = [p_D(t)^{top}, \ldots, p_D^{(q-1)}(t)^{top}] \) where \( p_D(t) = \sum_{i=0}^{t} d_i / \tau \) and \( D = [d_0, \ldots, d_q] \in \mathbb{R}^{n \times (q+1)} \), and \( d_i = [d_{i,x}\ d_{i,y}]^{top} \) in (6). To simplify the notation, we re-express the derivatives as \( v(t) = p_D^{top}(t), a(t) = p_D^{top}(t), \) \( J(t) = p_D^{top}(t), \) etc., and drop subscript \( D \).

We can construct the polynomial trajectories via \( P_{(q)}(t) = u(t) \) with controls \( u(t) \in \mathcal{U} = [-u_{\text{max}}, u_{\text{max}}]^n \subset \mathbb{R}^n \). In state-space form, we obtain \( \dot{s}_d(t) = A_f s_d(t) + B_f u(t) \) with

\[
A_f = \begin{bmatrix} 0 & I_n & 0 & \cdots & 0 \\ 0 & 0 & I_n & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & I_n & 0 \\ 0 & \cdots & 0 & 0 & I_n \end{bmatrix}, \quad B_f = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}. \tag{11}
\]

In collision-inclusive planning, we consider a smoothness cost \( J_c(D) = \sum_{k=1}^{K} J_{\beta}(\mathcal{U}(t))^{2} dt = \sum_{k=1}^{K} J_{\beta}(\mathcal{U}(t))^{2} dt. \) The trajectory is not \( q \)-th order differentiable as it would be in collision avoidance. The smoothness of the entire trajectory is the sum of its \( q \)th order differentiable segments. We consider two additional costs. First, \( T_g = \sum_{k=1}^{K} T_k + (K-1)T_r \) penalizes the overall trajectory duration. Then

\[
J_c = \begin{cases} \left( v_{\text{top}}^2 - c v_{\beta}^2 \right)^2 + \sum_{\beta \in (y,z)} \Delta E_\beta / 2 & \forall \zeta(t) = 1 \\ 0 & \text{otherwise} \end{cases} \tag{12}
\]

evaluates the effect of a collision in changing the direction of motion of the robot. \( \Delta E_\beta = (\hat{v}_\beta^+ - \hat{v}_\beta^-)^2 \), where \( \hat{v}_\beta^+ = c v_{\beta}^+ (t + T_r) \) and \( \hat{v}_\beta^- = c v_{\beta}^- (t) \). \( \hat{v}_{\beta}^+ \) can be approximated via Algorithm 4 if \( p_{\text{goal}} \) is known. (We discuss Algorithm 4 in detail in Section VI-D.) We also define an indicator function \( \zeta(t) = \{0, 1\} \) that signals if the robot is colliding at time \( t \).

We can then define the optimization problem

\[
\min_{D,T_g} J_c(D) + \rho_{t} T_g + \rho_{c} J_c(t) \tag{13a}
\]

subject to

\[
\dot{s}_d(t) = A_f s_d(t) + B_f u(t), \forall \zeta(t) = 0 \tag{13b}
\]

\[
\forall t \in [0, T_g] \tag{13c}
\]

\[
s_d(t + T_r) = F_{\text{DRR}}(s_d(t)), \forall \zeta(t) = 1 \tag{13d}
\]

\[
\forall t \in [0, T_g] \tag{13e}
\]

\[
\zeta(t) = 0, \forall t \leq s_d(t + \delta t) \in S_{\text{free}}, \delta t \rightarrow 0 \tag{13e}
\]

\[
\forall t \in [0, T_g] \tag{13e}
\]
\[
\zeta(t) = 1, \text{ if } s_d(t + \delta t) \in S_{\text{obs}}, \delta t \to 0 \\
\forall t \in [0, T_g - T_r] \\
s_d(0) = s_d_0, s_d(T_g) \in S_{\text{goal}} \\
\zeta(0) = 0, \zeta(T_g) = 0 \\
s_d(t) \in S_{\text{free}}, u(t) \in U, \forall t \in [0, T_g] \\
e_{\psi}(t) \in \mathcal{V}^c, \text{ if } \zeta(t) = 1.
\]

Parameters \( \rho_1 > 0 \) and \( \rho_2 > 0 \) regulate the relative importance of trajectory smoothness, duration, and amount of collisions that switch the direction of motion. Conditions (13e) and (13f) determine how the value for \( \zeta(t) \) is being set. In (13i), \( \mathcal{V}^c = [-v_{\text{max},c}, v_{\text{max},c}] \) indicates the maximum collision velocity, which, if exceeded, will lead to the robot flipping over. Thus, we set the pre-collision velocity component along the \( x \)-axis of \( \mathcal{F}_c \) as \( v_x(t) \in \mathcal{V}^c \).

Herein we show that, similar to the collision avoidance motion planning problem [21], safety constraints may be addressed by reformulating problem (13) into a deterministic shortest path one with \( (n \times q) \) state \( S \) and \( n \) control \( U \). Since the dimensionality of \( U \) remains at \( n \), search-based planning (e.g., A* [43]) that discretizes \( U \) using motion primitives can be an effective way to determine in finite-time resolution-complete and optimal (in the discretized space) trajectories.

### B. Motion Primitives

Choosing a number of samples \( r \in \mathbb{Z}^+ \) along each axis \([-u_{\text{max}}, u_{\text{max}}] \), which defines a discretization step: \( du = \frac{u_{\text{max}}}{r} \) and results in \( M = (2r + 1)^n \) motion primitives, is one technique to acquire the discretization \( \mathcal{U}_M \).

Given initial state \( s_{d_0} = [p_0^T, v_0^T, \cdots]^T \), we generate a motion primitive of duration \( \tau > 0 \) that applies piece-wise constant control.

\[
\dot{u}_m(t) = \begin{cases} 
\begin{align*}
& u_m(p_D^{(q-1)}(t) \in [p_D^{(q-1)}_{D,\text{min}}, p_D^{(q-1)}_{D,\text{max}}] \\
& 0 \quad p_D^{(q-1)}(t) \notin [p_D^{(q-1)}_{D,\text{min}}, p_D^{(q-1)}_{D,\text{max}}]
\end{align*}
\end{cases}
\]

(14)

where \( u_m \in \mathcal{U}_M \) for \( t \in [0, \tau] \). Given initial conditions,

\[
p_D(t) = \frac{\dot{u}_m(t) \cdot q_0}{q_1^T} + \cdots + v_0 t + p_0
\]

(15)

is a piece-wise function. The resulting trajectory of (11) is

\[
s_d(t) = e^{A_f t} s_{d_0} + \int_0^t e^{A_f(t-\sigma)} B_f \dot{u}_m(\sigma) d\sigma
\]

(16)

By beginning at \( s_{d_0} \) and applying all primitives to acquire the \( M \) possible states after \( \tau \in [0, \tau_f] \) (Alg. 3), we can create a graph representation of the attainable system states. There will be \( M^2 \) potential states at time \( 2\tau \) if all primitives are applied to each of the \( M \) states once again. The set of reachable states \( S' \) is finite given the free space \( S_{\text{free}} \) is bounded. These enable the construction of a graph the states of which are connected by a motion primitive \( e = (\dot{u}_m, \tau, \xi) \) with \( \xi \) being an integer (discussed in Section IV-D).

**Algorithm 3: Collision-Inclusive Motion Primitive Generation.**

**input:** Initial state \( s_d \in S' \subset S_{\text{free}} \); motion primitive set \( \mathcal{U}_M \); upper-bound of duration \( \tau_f \)

**output:** Reachable set \( \mathcal{R}(s_d) \) from \( s_d \) in one step; costs set \( \mathcal{C}(s_d) \); duration set \( \mathcal{T}(s_d) \); collision states set \( \mathcal{Z}(s_d) \)

**parameter:** Time interval \( \delta t \); recover time \( T_r \) in DRR

1. **Function GETMOTIONPRIMITIVE** \((s_d, \mathcal{U}_M, \tau_f)\):

   - \( \forall \; \forall u_m \in \mathcal{U}_M \)
     - Calculate edge \( e_{m}(t) \) according to (16) for \( t \in [0, \tau_f] \)
     - if \( e(t) \in S_{\text{free}} \) for all \( t \in [0, \tau_f] \) then
       - \( \zeta_m \leftarrow 0 \)
       - \( \tau_m \leftarrow \tau_f \)
       - \( s_{d,m} \leftarrow e_{m}(\tau_f) \)
     - \( \mathcal{R}(s_d) \leftarrow \mathcal{R}(s_d) \cup \{s_{d,m}\} \)
     - \( J_P \leftarrow \int_0^\tau \| \dot{u}_m(t) \| dt \)
     - \( \mathcal{C}(s_d) \leftarrow \mathcal{C}(s_d) \cup \{J_P + \rho_r (\tau + T_r) + J_c\} \)
     - \( \mathcal{T}(s_d) \leftarrow \mathcal{T}(s_d) \cup \{\tau_m\} \)
     - \( \zeta(s_d) \leftarrow \mathcal{Z}(s_d) \cup \{\zeta_m\} \)
   - else
     - \( \zeta_m \leftarrow 1 \)
     - Generate \( s_{d,m}, \tau \) and calculate \( J_c \) or prune this primitive (discussed in VI-D and VI-F).
     - \( \mathcal{R}(s_d) \leftarrow \mathcal{R}(s_d) \cup \{s_{d,m}\} \)
     - \( J_P \leftarrow \int_0^\tau \| \dot{u}_m(t) \| dt \)
     - \( \mathcal{C}(s_d) \leftarrow \mathcal{C}(s_d) \cup \{J_P + \rho_r (\tau + T_r) + J_c\} \)
     - \( \mathcal{T}(s_d) \leftarrow \mathcal{T}(s_d) \cup \{\tau_m\} \)
     - \( \zeta(s_d) \leftarrow \mathcal{Z}(s_d) \cup \{\zeta_m\} \)
   - end
2. return \( \mathcal{R}(s_d), \mathcal{C}(s_d), \mathcal{T}(s_d), \mathcal{Z}(s_d) \)

We construct the graph to explore the free state space \( S_{\text{free}} \) using Algorithm 3. Given the constant time upper-bound \( \tau_f \) and the fully specified state \( s \), the primitive is derived in line 4 using the control input \( u_m \); lines 5–23 check whether the primitive intersects with the obstacles, and then, modify those primitives intersecting with the obstacles. This step will be further discussed Section VI-D. In lines 6–13, we evaluate the end state of a valid primitive not intersecting with the obstacles and we add it to the set of successors of the current node; meanwhile, we estimate the edge cost from the corresponding primitive. In lines 16–21, we modify the end state of the primitive and add it to the set of successors of the current node; meanwhile, we estimate the edge cost related to the corresponding modified primitive. Line 19 shows that we consider \( T_r \) for the robot recovering from the collision using DRR in the cost function. Further modification of the cost function about estimating the cost related to \( J_c \) part will be discussed in Section VI-D. The nodes in the successor set \( \mathcal{R}(s_d) \) are added to the graph after we have checked all the primitives in the finite control input set. Finally, the graph keeps growing until we reach the goal is reached.
C. Deterministic Shortest Trajectory

We can re-formulate (13) as a graph-search problem using the set of motion primitives \( U_M \) and the induced discretization. To do so, we introduce additional constraints for the control input \( u(t) \) in (13) to be piecewise-constant. We introduce an additional variable \( N \in \mathbb{Z}^+ \) so that \( T_g = \sum_{k=0}^{N-1} (\tau_k + \zeta_{k+1} T_f) \), and \( u_k \) is computed by (14) with \( u_k \in U_M \) for \( k = 0, \ldots, N - 1 \) and a constraint in (13h)

\[
u(t) = \sum_{k=0}^{N-1} u_k \mathbb{1}_{(T_k,T_{k+1}]}.\]

By letting \( T_i = \sum_{k=0}^{i-1} \tau_k \) we can force the control trajectory to be a composition of the motion primitives in \( U_M \). Given an initial state \( s_{d,0} \in S^{\text{free}} \), a goal area \( S^{\text{goal}} \) and a finite set of motion primitives \( U_M \) with duration \( \tau > 0 \), we seek to select a series of motion primitives \( u_{0:N-1} \) of length \( N \), such that

\[
\min_{N,u_{0:N-1}} \sum_{k=0}^{N-1} \|u_k\|^2 + \rho_t (\tau_k + \zeta_{k+1} T_f) + \rho_c J_{c,k}\tag{17a}
\]

subject to

\[
\begin{align*}
    s_d(\mathbb{1}) &= A_{df}(\mathbb{1}) s_{d,k} + B_{df}(\mathbb{1}) u_k \in S^{\text{free}} \quad \forall \mathbb{1} \in [0,\tau_k] \tag{17b} \\
    \zeta_k &\in \{0,1\} \quad \forall k \in [0,1,\ldots,N-1] \tag{17c} \\
    \zeta_{k+1} &= 0, \quad \text{if } s_d(t_k + \delta t) \in S^{\text{free}}, \delta t \rightarrow 0 \tag{17d} \\
    \zeta_{k+1} &= 1, \quad \text{if } s_d(t_k + \delta t) \in S^{\text{obs}}, \delta t \rightarrow 0 \tag{17e} \\
    s_{d,k+1} &= s_d(t_k), \forall \zeta_{k+1} = 0 \tag{17f} \\
    s_{d,k+1} &= F_{\text{DRR}}(s_d(t_k), \zeta_{k+1} = 1 \tag{17g} \\
    s_{d,0} &= s_{d,0}, s_{d,N} \in S^{\text{goal}}, \zeta_0 = 0, \zeta_N = 0 \tag{17h} \\
    u_k &\in U_M \tag{17i} \\
    \epsilon_{v_{k,x}} &\in \Psi^c, \text{if } \zeta_k = 1 \tag{17j}
\end{align*}
\]

The optimal cost of (17) is an upper bound to the optimal cost of (13) because (17) is a constrained version of (13). The whole trajectory consists of a set of continuous and collision free primitives of \( \tau_k \) duration and initial state \( s_{d,k} \). If the end state of the primitive \( s_d(t_k) \) is state, which collides with an obstacle, we modify it based on (17g). We modify the final state based on the DRR controller model. We make the modified final state as initial state of next primitive. If the end state of the primitive \( s_d(t_k) \) is collision free, we keep the final state similar the collision-avoidance planner making the final state as initial state of the next primitive as (17g). Reformulating into a discrete control problem enables the use of several motion planning methods that rely on search-based [44] or sampling-based [4] techniques. We choose to adopt an A* technique similar to [21] and concentrate on the creation of effective, guaranteed collision checking and post-collision behavior categorizing methods, as well as an accurate and consistent heuristic since the former can ensure limited time (sub-)optimality.9

D. Collision Checking and Postcollision Behaviors

For a computed edge \( e(t) = [p(t)]^\top v(t)^\top u(t)^\top \cdots]^\top \), in Algorithm 3, we need to check if \( e(t) \in S^{\text{free}} \) for all \( t \in [0,\tau_f] \). For \( e(t) \in S^{\text{free}} \wedge e(t + \delta t) \in S^{\text{obs}} \) with \( \delta t \rightarrow 0 \) for all \( t \in [0,\tau_f] \), we need to modify the edge \( e(t) \) as in lines 16–21 in Algorithm 3. We check collisions in the geometric space \( P^{\text{free}} \subset \mathbb{R}^n \) separately from enforcing dynamic constraints \( P^{\text{free}} \subset \mathbb{R}^{n \times (q-1)} \).

An edge \( e(t) \) is collision-free only if its geometric shape \( p_c(t) \in P^{\text{free}} \) for all \( t \in [0,\tau_f] \).

In general, determining collision points for each motion primitive can be very challenging. Herein we model \( P \) as an occupancy grid map, \( M_o \). Other representations such as polyhedral maps [3], [22], [25] are also possible but often hard to obtain from a robot’s FoV sensor data (e.g., from LiDAR) and hence not pursued herein. Let \( P_c = \{p_c(t_i) | t_i \in [0,\tau_f], i = 1, \ldots, I\} \) be a set of positions that the system traverses along the trajectory. For collision-free primitives, we need \( p_c(t_i) \in P^{\text{free}} \) for all \( i \in [0, \ldots, I] \). The duration of the collision-free trajectory is \( \tau = \tau_f \). For the given polynomial \( e(t), t \in [0,\tau_f] \), the positions \( p_c(t_i) \) are sampled by defining

\[
t_i = \frac{i}{7} \tau_f \quad \text{such that} \quad \frac{\tau}{7} v_{\text{max}} \geq \epsilon_{\text{map}} \tag{18}
\]

where \( \epsilon_{\text{map}} \) is the occupancy grid resolution, and \( v_{\text{max}} = \max\{|v_{\text{min}},|v_{\text{max}}|\} \). This condition ensures that the maximum distance between two consecutive samples will not exceed the map resolution. Since it is an approximation, some cells traversed by \( p_c(t) \) with a portion of the curve within the cell shorter than \( \epsilon_{\text{map}} \) may be missed, but it guarantees the collision-free trajectory does not hit any obstacles.

In not collision free \( e(t) \), the estimated collision time instant \( t_i \) is when \( p_c(t_i) \in P^{\text{free}} \wedge p_c(t_i + \delta t) \in P^{\text{obs}} \) with \( \delta t \approx t_{i+1} - t_i \) for all \( i \in [0, \ldots, I - 1] \). Then, we set the duration \( \tau \) of the collision-inclusive motion primitives in Algorithm 3 to \( t_i \), and modify the end state \( s_{d,c} \) of \( e(t) \) as \( s_{d,c} = \{s_{d,c}, s_{d,c}^+\} \) with \( s_{d,c}^+ = e(t_i) = [p_c(t_i)^\top v_c(t_i)^\top a_c(t_i)^\top \cdots]^\top \). We set the duration of this edge \( \tau = t_i \) and set \( \zeta(\tau) = 1. s_{d,c}^+ = F_{\text{DRR}}(s_{d,c}^-) \) is the postimpact state recovered using the DRR strategy. We discuss how to set \( s_{d,c}^+ \) shortly.

Since \( v, a \) and other higher order derivatives are polynomial functions, we can compute their extrema within the time period \( [0, \tau] \) to check if the respective maximum bounds are violated. The polynomials’ order is less than 5 for \( n \leq 3 \), hence the extrema can be computed quickly in closed form. We eliminate the primitives that cannot be dynamically implemented (i.e. any bounds are exceeded). For the collision-inclusive primitives, we need to check the \( x \) component of the velocity \( v_c^x \) in \( F_c^x \)

9We note here that in principle both a search-based (as herein) and a sampling-based global planner is possible. In Section VII-D, we demonstrate the differences of the two within collision-inclusive motion planning.
corresponding $c v_{c,e,x}^− \in V^c$. We prune those with $c v_{c,e,x}^− \not\in V^c$ to prevent the robot from flipping over after colliding.

To generate the frame $F_c$ required for evaluating the collision-inclusive primitives, we need to get the geometric information of each obstacle that the robot collides on. Given a current scan from the mapping sensor (e.g., a LiDAR), we identify all possible collision surfaces and use regression to fit curve equations to the possible collision surfaces. The value of doing so is that it enables a rapid calculation of the tangent and normal unit vectors at selected possible collision points on those collision surfaces. Basis vectors of $F_c$ are generated as discussed in Section IV-A, whereas the origin of $F_c$ is set to be the estimated position of collision $p_{c,d,e}$.

After generating $F_c$, we are able to generate $s_{d,e}^+$ based on the map $M_o$, which we predict the robot will collide on when arriving at $s_{d,e}$ with the given motion primitive. Given the goal position $p_{goal}$, we are able to set $s_{d,e}^+$ according to Algorithm 4. This way, we can ensure the trajectory generated by the search-based algorithm respect constraint (17g).

![Diagram showing example of performing collision and detouring away from a nonconvex obstacle by generating a new waypoint between the point of collision and the goal.](image)

**Algorithm 4: Post-Collision State Generation.**

```plaintext
input : Pre-collision state in world frame $s_{d,e}^-$; collision frame $F_c$; goal position in world frame $p_{goal}$; $M_o$.
output : Post-collision state in world frame $s_{d,e}^+$; behavior type after collision $\xi$.
parameter: Lower bound and upper bound of the velocity $v_{min}, v_{max}$; upper bound of duration of each primitives $\tau_f, \delta_t$.

1 Function SetPostCollisionState ($s_{d,e}^-$, $p_{goal}$, $M_o$):
2   $v^+ \leftarrow \frac{p_{goal} - p_{c,d}}{\tau_f}$
3   Generate $\psi^+$ based on $F_c$ as what is shown in Fig. 6
4   if $c v_{c,e,x}^+ < 0$ then
5      $c v_{c,e,x}^+ \leftarrow 0$
6      $\xi \leftarrow 2$
7      We generate a intermediate waypoint $p_{add}$ as what is shown in Fig. 6 given $M_o$ based on jump point search algorithm. The waypoint should be the last visible waypoint along the path. $v^+ \leftarrow \frac{p_{add} - p_{c,d}}{\tau_f}$
8   else
9      $\xi \leftarrow 1$
10     $p_{add} \leftarrow \emptyset$
11   end
12 We adjust each components of $v^+$ with a saturation function restricting upper bound and lower bound as $v_{min}$ and $v_{max}$.
13 We set all derivatives of $s_{d,e}^+$ as $p_{c,e}^{(+)} = 0$, for all $q \geq 2$, $p_{c,e}^{(+q)} = v^+$, $p_{c,e}^{(+q)} = p_{c,e}^-$.
14 return $s_{d,e}^+$, $\xi$.
```

**E. Heuristic Function Design.**

A heuristic function that is admissible, informative (i.e. provides a tight approximation of the optimal cost), and consistent (i.e. it can be inflated to obtain solutions with bounded sub-optimality efficiently) is required for efficient graph search to solve (17). Similar to [21], we solve a relaxed form of (13) and arrive at a reasonable heuristic function. The basic concept is to replace the difficult-to-satisfy $s_d(t) \in S_{free}$ and $u(t) \in U$ requirements in (13) with a constraint on $T$. Next, we demonstrate that a relaxation of (13) that includes motion planning may be solved optimally and effectively. We add a constraint to ensure that the robot will travel through the recently added waypoint $p_{add}$, avoiding the obstacle it collided with and preventing repeated collisions with it if $p_{add} \neq \emptyset$.

1) **Lower Bound of Time:** Limits on maximum speed, acceleration, jerk, etc. imposed by $S_{obs}$ and $U$ can help create a lower in (13) of $T$. If $P_{add} = \emptyset$, the minimum time to reach the nearest state $s_{d,goal}$ in the goal region $S_{goal}$ is constrained by $T_v = \frac{\|p_{goal} - p_{c,d}\|}{v_{max}}$. This is because the systems maximum velocity is bounded by $v_{max}$ along each axis. The system’s maximum acceleration is bounded by $a_{max}$, hence, the state.
The aforementioned is a minimum-time optimal control problem with input constraints, which can be solved in closed form along each individual axis to obtain the lower bound $T_a = \min \{ T_{a,x}, T_{a,y}, T_{a,z} \}$ [45, ch. 5]. This procedure applies for constraints in higher order derivatives, but in practice, the computed times are less likely to provide better bounds while requiring higher computational effort. Hence, even though we can define a lower bound on the minimum achievable time via $T_a = \min \{ T_{a}, T_{a'}, \cdots \}$, for computational expediency, we use the efficiently computed (but less tight) bound $T_a = T_{a'}$. For those cases with $p_{add} \neq 0$, we generate $T_a$ and $T_g$ for path segments $p_0 \rightarrow p_{add}$ and $p_{add} \rightarrow p_{goal}$ as $T_a = T_{a',1} = \max(h_{\text{add}})$ and $T_g = T_{a',2}$.

2) Velocity Control Linear Quadratic Minimum Time Heuristic: The lower bound $T_a$ can help relax (13) by replacing the state and input constraints. If $p_{add} = 0$, then

$$\begin{align}
\min_{D,T_g} J_s(D) + \rho_t T_g \\
\text{subject to} \\
\dot{s}_d(t) &= A f s_d(t) + B_f u(t), u(t) = \alpha(t) \\
\forall t \in [0,T_g] \\
\|\alpha(t)\| &\leq \alpha_{\text{max}} \\
\dot{s}_d(0) &= \left[p_{d,0}^T v_{d,0}\right]^T, s(T_a) = \left[p_{goal}^T v_{goal}\right]^T.
\end{align}$$

The relaxed problem (20) is in fact the classical linear quadratic minimum-time problem [46]. The optimal cost generated from (20) according to (22) is

$$h(s_{d,0}) = \delta_{T}^e W^{-1}_T \delta_T + \rho_t T_g.$$  \hspace{1cm} (21)

We define $\delta_{T} = s_{d,goal} - e^{A f} T_s s_{d,0}$ and the controllability Gramian $W_T = \int_{0}^{T_s} e^{A f} t e^{A f^T} B_f dt$.

Let us consider velocity control as an illustrative example of (21). Given $T_g$, $s_{d,0} = p_0$, $s_{d,goal} = p_{goal}$, we can rewrite the optimal cost of (20) shown in (21) as

$$h_v(s_{d,0}) = C^*(T_g) = \frac{\|p_{goal} - p_0\|^2}{T_g} + \rho_t T_g.$$  \hspace{1cm} (22)

By minimizing $C^*$ in (22) with the constraint $T_g \geq T_a$, we are able to obtain the ideal $T_g$. If the positive real root $\text{root}^+ \geq T_g$, then the solution is the positive real root of $\frac{\partial C^*}{\partial T} = 0$. Otherwise, $T_g = T_a$. Furthermore, the optimal cost is $C^*(T_g)$.

For the case where $p_{add} \neq 0$, we modify (22) to

$$h_v(s_{d,0}) = C^*(T_1, T_2) = \frac{\|p_{goal} - p_0\|^2}{T_1} + \rho_t T_1 + \frac{\|p_{goal} - p_{add}\|^2}{T_2} + \rho_t T_2.$$  \hspace{1cm} (23)

Similarly, we are able to derive the optimal $T_1$ and $T_2$ by minimizing $C^*$ in (23) with constraints $T_1 \geq T_a$ and $T_2 \geq T_a$. We can get the solution of this optimization problem by solving the positive real root of $\frac{\partial C^*}{\partial T_1} = 0$ and $\frac{\partial C^*}{\partial T_2} = 0$. The optimal cost then is $C^*(T_1, T_2)$.

F. Jump Point-Based Computation to Improve Efficiency

Following the aforementioned approach results in a collision-free trajectory including specific times needed to reach each waypoint. This is then fed as a prior to create smooth trajectories in higher dimensions. The refined trajectory $s_{d}(t)$ is derived from solving a gradient-based trajectory generation problem similar to the one in Section V-A with given initial and end states $s_{d,0}$ and $s_{d,goal}$ and intermediate waypoints $p_k \in \{0,1, \cdots , N\}$. The time for each trajectory segment $\tau_k$ is also given from the prior trajectory. All $p_k$ are stored in waypoint_list (see Algorithm 2).

To improve computational efficiency and reduce the computational time of our method, we can replace the postimpact motion primitive generation technique introduced in Section VI-D in A* graph search with a more efficient variant that is inspired by jump point search [47]. Specifically, we notice that when the robot needs to add a new waypoint between the collision point $p_c$ of the motion primitive and the goal $p_{goal}$ ($\xi = 2$), we can modify $p_{add} = p_{add}$. Performing this modification will help us eliminate traversing multiple nodes with the same $p_{add}$. This way, the number of nodes we are traversing can reduce, thus reducing the computational time. Even though applying this technique can be at expense of optimality of the solution, solving the planning problem with less computational time can be more important in practice.

If colliding with an obstacle (as shown in Fig. 6), we modify $p_{add}$ and duration $\tau$ as $\tau = \tau + \tau_{add}$ with $\tau_{add} = \frac{||p_{goal} - p_c||}{||v_c||}$. The cost will be updated with new $\tau = \tau + \tau_{add}$. When we go through edges with $\xi = 2$, we split the trajectory of this given edge with two segments, given the start and the end waypoints as $p_0$ and $p_c$ for the first segment, $p_c$ and $p_{add}$ for the second segment. The time duration of the first segment is $\tau - \tau_{add}$ and the time duration of the second segment is $\tau_{add}$. We set the $\xi = 0$ and $\xi = 0$ with respect to the waypoint $p_c^\perp$.

G. Trajectory Refinement

The trajectory resulting from following the aforementioned procedure gives not only the collision-free path, but also the time for reaching the respective waypoints. Thus, we are able to use it as a prior to generate a smoother trajectory in higher dimension for controlling the actual robot. The refined trajectory $s_{d}(t)$ is derived from solving a gradient-based trajectory generation problem similar to the one in Section V-A with given initial and end states $s_{d,0}$ and $s_{d,goal}$ and intermediate waypoints $p_k \in \{0,1, \cdots , N\}$. The time for each trajectory segment $\tau_k$ is also given from the prior trajectory. All $p_k$ are stored in waypoint_list (see Algorithm 2).
We apply a two-step optimization strategy similar to [42], which can be summarized as follows.

1) First, optimize the collision cost of the path generated from waypoints only. Positions of intermediate waypoints on the initial path are left as free variables, and will be pushed away from the obstacles.

2) Second, revise the time scaling of the trajectory according to current waypoints’ positions. Then, optimize the objective with additional smoothness and dynamical penalty terms.

The output trajectory comprises \( N_c + 1 \) continuous polynomial trajectory segments. The differential variable of the waypoint with \( \xi_k \geq 1 \) is fixed end variable in the collision-inclusive method, which indicates \( s_{d,i} = s_i^d, k \in \{0, 1, \ldots, N_c\}, i \in \{1, \ldots, N_c\} \) \( \forall \xi_k \geq 1 \). The differential variable of the next fixed initial state is \( s_{d,i+1} = s_i^d, k \in \{0, 1, \ldots, N_c\}, i \in \{1, \ldots, N_c\} \) \( \forall \xi_k \geq 1 \). The collision state waypoints \( \xi_k \geq 1 \) are generated from a grid map with augmented obstacles. We need to adjust those waypoints before trajectory generation by relocating them so that the distance to the closest block is \( d_c \leq r_{rob} \); this way we ensure that planned collisions occur. The output of the search-based algorithm may have two colliding and reflecting states that are close-by (see Fig. 7). In this case, if \( p_i \) is visible to both \( p_{i-2} \) and \( p_{i+1} \), we can delete \( p_{i-1} \) to reduce redundant collisions. We also disregard obstacles that the robot planned to collide on \( (\xi_k \geq 1) \) when computing the potential field for trajectory generation in the second step for computational expediency. The trajectory after refinement is \( n \)th order continuous.

It is important to note that even though the refinement step produces a smoother trajectory, the refined course might be dynamically infeasible; we need to perform time scaling as in [22] to reduce the maximum dynamics of the refined trajectory. The refined trajectory might collide with the obstacle in the trajectory segment that is checked to be collision free according to Section VI-D. In such cases, our DRR strategy ensures robustness and safety.

VII. EXPERIMENTAL RESULTS

We validate the effectiveness of our unified framework for collision-inclusive motion planning and control by presenting several benchmark testing results in simulation and via physical experimentation with our robot.

1) First, we test the deformation controller on the robot to evaluate its performance and generate a postcollision model, which is required in simulation.

2) Second, we test the local DRR trajectory generation component experimentally with our robot.

3) Then, we test our global planning method in a double corridor environment and compare it with state-of-the-art search-based collision avoidance and sampling-based collision-inclusive methods.

4) After that we test the overall planning strategy in simulation in unknown, partially observable environments.

5) Finally, we evaluate the overall method with our impact-resilient robot experimentally.

A. Experimental Setup and Implementation Details

Testing the deformation controller (see Section VII-B) and DRR (see Section VII-C) experimentally takes place in a 2.0 × 2.5 m area with a rectangular pillar serving as a static polygon-shaped obstacle. The overall method is tested experimentally (see Section VII-F) in a 2.5 × 3.5 m area with a long rectangular pillar in the middle to form a U-shaped corridor environment.

We use the two active omnidirectional impact-resilient wheeled robots we built in-house (see Fig. 1). The main chassis is connected to a deflection “ring” via 4 or 8 arms that feature a viscoelastic prismatic joint each. Each arm has embedded Hall effect sensors to measure the length of the arm and detect collisions along each of their direction when the deformation exceeds a certain threshold. In physical experiments, odometry feedback is provided by a 12-camera VICON motion capture system. An onboard Intel NUC mini PC (2.3 GHz i7 CPU; 16 GB RAM) processes odometry data and sends control commands to the robot at 10 Hz. The robot is equipped with a single-beam LiDAR (RPLidar A2) with 8-m range to detect the obstacles in the environment.

The robot may flip when colliding with velocities over a bound. To identify a theoretical collision velocity bound to avoid flipping, we use an energy conservation argument. Assume the kinetic energy before collision transfers into elastic potential energy of the arm, and the gravitational potential energy of the robot with small flipping angle counters the negative work input from the controller, i.e.,

\[
E_{k,t}(v_{max}) = E_{cp}(l_e) - E_{cp}(l_s) + E_{gp}(\sigma_{max}) + m a_{in,max}(l_s - l_e).
\]

Then

\[
v_{max} = \sqrt{\frac{k[(l_e - l_0)^2 - (l_s - l_0)^2]}{2m} + g(\rho - l_s + l_e) \sin \sigma_{max} + a_{in,max}(l_s - l_e)}.
\]

The robot’s radius is 0.3 m. The difference between the initial and neutral position of each arm is \( l_s = 30.0 \) mm, the maximum load length is \( l_e = 15.0 \) mm, and the neutral length is \( l_0 = 41.5 \) mm. The spring coefficient is \( k = 2.31 \) N/mm. We set the largest flip angle \( \sigma_{max} = 3^\circ \). For the four-arm robot, the maximum acceleration input is \( a_{in,max} = 5.0 \text{ m/s}^2 \), and its mass...
is 6.0 kg. Then, we compute an upper theoretical velocity bound of $v_{\text{max}} \approx 0.7$ m/s.\footnote{The eight-arm robot features motors with higher torque and different gear ratio that increase $\alpha_{\text{in, max}}$ and despite the mass increase to 8 kg, the same upper theoretical velocity bound remains valid.}

Simulated comparison against other methods (see Section VII-D) takes place in a double-corridor environment, whereas simulated benchmark testing of our method when noise is added takes place in the same double-corridor environment but with added isolated obstacles added as well (see Section VII-E). We further consider a similar environment that features non-convex obstacles (see Section VII-E).

We use a rigid cylinder body to emulate the robot. A numerical model is generated from the experiments for the deformation recovery controller to determine the output velocity after collision. The output velocity is generated by adding uniform random noise to the reference velocity $v_T$. Then, we use a ray-casting algorithm to emulate the LiDAR (we consider the range of the LiDAR can cover all visible operating space). We implement simulation benchmarks in a python environment. All simulations run on a workstation with Intel Core Xeon-E2146 G CPU.

B. Experimental Testing of the Deformation Controller

To examine the deformation controller’s effect in local trajectory generation, we command the robot to collide with an obstacle and then apply the proposed deformation recovery controller. We perform 10 trials of various input–output velocity combinations [20, Table 2]. Collision detection is very accurate; only 9 out of 249 were not detected.

Results suggest that the deformation controller generates a negative velocity to make the robot detach from the obstacle after collision. Actual output velocity $v_{\text{out}}$ is determined by the actual input velocity $\tilde{v}_{\text{in}}$ and the set output value $v_{\text{out, set}}$ though the latter may not be reached in practice. That is because feedback linearization is not robust to system parameter uncertainties that occur in practice. We observe that the velocity along the normal to collision direction is closer to the set velocity than the velocity along the tangent direction. This is because most of the uncertainties in system parameters enter as unmodeled friction dynamics along the tangent direction. Furthermore, the sensor is more accurate when the input velocity is along the normal direction; the average value of deformation detected in this case is 29% larger.

C. Experimental Testing of the DRR Strategy

We test our DRR strategy with a trajectory generated based on using the online safe trajectory generation method in [48] with time scaling as in [22] without collision checking. We compare the strategies in two cases:

1) when the previous path does not intersect with the collision surface;

2) when the previous path intersects with the collision surface.

Case 1 tests the condition in Fig. 5(a), i.e., no waypoint is added as per Algorithm 2. Case 2 tests Fig. 5, i.e., a waypoint is added to the list. In case 2, we run RRT* to generate a collision free path and perform path simplification to remove nodes without affecting the path’s collision safety [20, Fig. 4]. The path simplification technique removes intermediate waypoints between two waypoints if a line segment between those two does not intersect with the obstacle. Then, we use the trajectory generation strategy in [48]. We perform ten trials for each case. Instances of DRR and all experimental trajectories are shown in Figs. 8 and 9, respectively.

Even though we design a collision-free desired trajectory with the strategy in [48], the robot may still collide with the environment given for instance unmodeled dynamics such as drift. In case 1, there are 3 out of 10 trials that the robot in fact collides with the obstacle applying trajectory generation [48] that aims to avoid collisions. Table II shows statistics on mean arrival times, path lengths, and control energy.

In case 1, for DRR, mean arrival times $T_{\text{end}}$ and path lengths $\bar{s}$ decrease by 25% and 6%, while the control energy increases by 2.8% on average. However, the error in the end point increases by 25%. In case 2, mean arrival times and path lengths increase by 5.2% and 27%, and the control energy increases by 258%. This is because the output velocity of DRR is not flat since the robot decelerates, and then, accelerates during boundary following. The path generated by the boundary following is not the shortest. However, since the path between the collision point and the new inserted waypoint is close to the obstacle surface, the existence of the obstacle decreases the control error in free space. The
error in the end point decreases by 12%. These results show the tradeoff between online reactive execution (whereby collision checking is skipped) and collision avoidance.

D. Simulated Tests of the Collision-Inclusive Global Planner

To test our proposed framework in simulation, we first benchmark it in a double corridor environment to test our search-based collision-inclusive global planner. We compare our method for global planning against the following two methods:

1) a search-based collision-avoidance motion planning algorithm [21];

2) an RRT*-based (sampling-based) collision-inclusive planning algorithm adapted from [4] to ensure fair comparisons.\(^{12}\)

In all tests, the dynamic limits are set as \(v_{\text{max}} = 5.0 \, \text{m/s}^2\). The holonomic robot only translates but does not rotate during the process.\(^{13}\) We set the upper bound of the robot velocity \(v_{\text{max}} = 2.0 \, \text{m/s}\). The cost function in all methods considers \(\rho_c = 1.0\), with the ablation studies to determine this value being demonstrated qualitatively in Fig. 10 and expanded in more depth in Table V.

Results from testing the global planner are shown in Fig. 10; both collision avoidance as per [21] (panel a) and collision-inclusive (our method, panels b–f) results are highlighted. We demonstrate our method’s results with and without implementing jump points. We also consider the following three cases for varying values of the parameter \(\rho_c\), which affects how much collisions are being penalized in the cost function:

1) \(\rho_c = 1.0\) (panels b and e), which corresponds to minimal penalty;

2) \(\rho_c = 10.0\) (panels c and f), which corresponds to a medium penalty;

3) \(\rho_c = 100.0\) (panel d), which corresponds to a severe penalty.

It can be readily verified that both cases of \(\rho_c = \{1.0, 10.0\}\) can lead to paths that contain collisions, although in some cases (especially when jump points are considered), a higher \(\rho_c\) value of 10.0 may make the output trajectory unnecessarily complex and suboptimal (panel f). As such, if collisions are to be considered, setting \(\rho_c = 1.0\) should be preferred. At very high \(\rho_c\) values and \(\lambda_{\nu} = 10.0.\)\(^{14}\) Parameter \(\rho_c\) in the cost function is one of the most critical ones as it determines how much collisions are being penalized. We select \(\rho_c = 1.0\), with the ablation studies to determine this value being demonstrated qualitatively in Fig. 10 and expanded in more depth in Table V.

### Table II

Comparison of Trajectory Generation Strategy in [48] (Collision-Avoidance) and DRR (Collision-Inclusive) Strategies

| Strategy in [48] | DRR (our method) |
|------------------|------------------|
|                  | Case 1 | Case 2 | Case 1 | Case 2 |
| \(T_{\text{end}}\) [s] | 7.7   | 8.71  | 6.16  | 9.17  |
| \(\text{STD}(T_{\text{end}})\) | 0.0   | 0.22  | 0.31  | 0.31  |
| \(\bar{x}\) [m] | 3.153 | 3.448 | 2.977 | 4.38  |
| \(\text{STD}(\bar{x})\) | 0.117 | 0.495 | 0.150 | 0.451 |
| \(\bar{E}_c\) [m²/s²] | 56.83 | 80.62 | 58.42 | 255.96 |
| \(\text{STD}(\bar{E}_c)\) | 29.34 | 54.38 | 32.92 | 133.54 |

### Table III

Comparison of Global Planners’ Performance Between Our Method and That to Prune Primitives

| \(r = 1.0\) [m/s²] | \(\bar{e}_{\text{std}}\) [m] | Comp. Time(s) | \(N_f\) | Traj. Time(s) | Ctr. Cost([m²/s²]) |
|---------------------|-----------------------------|---------------|--------|--------------|-------------------|
| method [21] without pruning | 5.0 | 31.86 | 1427 | 71.5 | 13.01 |
| Our method without pruning \(\rho_c = 1.0\) | 5.0 | 34.60 | 1340 | 69.1 | 13.25 |
| method [21] with pruning | 0.5 | 18629.12 | 102836 | 91.1 | 201.98 |
| Our method with pruning \(\rho_c = 1.0\) | 0.5 | 15168.97 | 69450 | 90.9 | 200.72 |

\(^{12}\)No open-source python code is available for either [4] or [21], so we implemented both ourselves to the best possible extent.

\(^{13}\)Constant orientation is maintained via a separate stabilizing controller.

\(^{14}\)The values were selected empirically to improve trajectory refinement.
Fig. 10. Search-based collision avoidance global planner [21] (top panels) and our proposed search-based collision-inclusive global planner for three different $\rho_c$ values. Higher $\rho_c$ values (bottom panels) penalize collisions more, thus recovering behaviors that resemble collision avoidance. The collision-inclusive with jump points case when $\rho_c = 100.0$ is very similar to the case without jump points in panel (d) in the sense of number of closed nodes $N_p$ (c.f., Table V, and hence, not shown here for brevity. (a) Collision-avoidance without node pruning. (b) Collision-inclusive without jump points ($\rho_c = 1.0$). (c) Collision-inclusive without jump points ($\rho_c = 10.0$). (d) Collision-inclusive without jump points ($\rho_c = 100.0$). (e) Collision-inclusive with jump points ($\rho_c = 1.0$). (f) Collision-inclusive with jump points ($\rho_c = 10.0$).

(of 100.0), we observe that our method can recover collision avoidance behaviors (c.f., panels a and d). This highlights our global planner’s ability to switch between collision-inclusive and collision-avoidance planning on-demand by only updating the value of a single parameter.

We also demonstrate the utility of formulating the motion primitives as discussed in Section VI-B as compared to directly pruning dynamically infeasible primitives. Our proposed method can feature primitives of longer duration $\tau$, which in fact helps increase the efficiency of exploring the space. Table III provides comparative numbers for both collision avoidance [21] and our collision-inclusive method. Results verify that our selected primitives generation method can explore the space with significantly less computational time when compared to the approach of pruning infeasible states.

Table IV contains the results from the ablation study on the resolution parameter $r$. We found that in the environment with simple obstacles (as in Fig. 14(a) shown later), a high resolution of $r = 0.5$ leads to computational times for both collision-avoidance and collision-inclusive methods that are much higher since the graph is denser. The computational time of the collision-inclusive method is higher than the collision-avoidance method since we modify those primitives that intersect with the obstacles instead of pruning them altogether. With lower resolution $r = 1.0$ or $r = 2.0$, the computational time of both collision-avoidance and collision-inclusive methods rapidly decreases. When $r = 1.0$, collision-avoidance and our collision-inclusive method have comparable computational performance. However, as we further increase the resolution ($r = 2.0$), the computational time of our collision-inclusive method gets much lower than the collision-avoidance method, which appears to be affected less by this change. These observations suggest that with lower resolution, collision-inclusive primitives can explore the space with higher efficiency. However, the trajectory time and control cost are higher than applying higher resolution. Similar observations can be made when testing in a more complicated environment [see Fig. 14(b)]. Therefore, taken computational time, trajectory time, and control cost into consideration, we

| $\nu_0$ | Comp. [ms] | $N_p$ | Traj. Time [s] | Ctrl. Cost [m^2/r^3] | Succ. Rate [%] |
|---------|------------|------|---------------|----------------------|-------------|
| search based method [21] | 29.02 | 1427 | 71.5 | 12.01 | 100.0 |
| Our method no jump point $\rho_c = 1.0$ | 34.60 | 1380 | 69.1 | 13.25 | 100.0 |
| Our method with jump point $\rho_c = 1.0$ | 30.75 | 1388 | 73.3 | 10.10 | 100.0 |
| Our method no jump point $\rho_c = 10.0$ | 48.28 | 1701 | 74.1 | 10.74 | 100.0 |
| Our method with jump point $\rho_c = 10.0$ | 44.32 | 1747 | 76.7 | 12.60 | 100.0 |
| Our method no jump point $\rho_c = 100.0$ | 44.93 | 1588 | 71.5 | 12.01 | 100.0 |
| Our method with jump point $\rho_c = 100.0$ | 41.74 | 1602 | 71.5 | 12.01 | 100.0 |

| $\nu_0$ | Comp. [ms] | $N_p$ | Traj. Time [s] | Ctrl. Cost [m^2/r^3] | Succ. Rate [%] |
|---------|------------|------|---------------|----------------------|-------------|
| sampling based method [4] | mean | 124.94 | 211 | 101.0 | 9.86 | 70.0 |
| | std | 55.63 | 38.2 | 9.86 | 2.91 |
| | min | 93.72 | 183 | 84.2 | 5.77 |
| | max | 259.40 | 238 | 114.8 | 15.27 |

| $\nu_0$ | Comp. [ms] | $N_p$ | Traj. Time [s] | Ctrl. Cost [m^2/r^3] | Succ. Rate [%] |
|---------|------------|------|---------------|----------------------|-------------|
| search based method [21] | 22.41 | 1113 | 71.7 | 14.02 | 100.0 |
| Our method no jump point $\rho_c = 1.0$ | 24.45 | 1012 | 81.0 | 15.03 | 100.0 |
| Our method with jump point $\rho_c = 1.0$ | 23.78 | 1077 | 78.9 | 8.29 | 100.0 |
| Our method no jump point $\rho_c = 10.0$ | 37.30 | 1388 | 74.9 | 13.28 | 100.0 |
| Our method with jump point $\rho_c = 10.0$ | 36.67 | 1417 | 74.9 | 13.28 | 100.0 |
| Our method no jump point $\rho_c = 100.0$ | 30.97 | 1145 | 71.7 | 14.02 | 100.0 |
| Our method with jump point $\rho_c = 100.0$ | 30.38 | 1145 | 71.7 | 14.02 | 100.0 |

| $\nu_0$ | Comp. [ms] | $N_p$ | Traj. Time [s] | Ctrl. Cost [m^2/r^3] | Succ. Rate [%] |
|---------|------------|------|---------------|----------------------|-------------|
| sampling based method [4] | mean | 88.50 | 169 | 117.0 | 14.18 |
| | std | 43.88 | 55.7 | 10.58 | 5.14 |
| | min | 17.09 | 63 | 103.6 | 8.28 |
| | max | 132.83 | 235 | 134.9 | 21.83 |
select $r = 1.0$ for which both collision-avoidance and collision-inclusive method have better results.

Furthermore, we conduct a more extensive analysis to evaluate the effect of different values of parameter $\rho_c$ in more detail. Table V contains more detailed results and also presents comparisons against the sampling-based (RRT*) method in [4], which was adapted herein to feature a trapezoidal velocity pattern to connect any two nodes in the tree to better match our search-based global planning method and enable fair comparisons. Due to the nondeterministic nature of this method, we perform ten trials and report statistics. In all other cases (that are deterministic), we perform a single trial.

Both our collision-inclusive method and collision avoidance in [21] can generate kinodynamically feasible trajectories. When the initial velocity is $v_0 = [0 \ 0] \text{ m/s}$ and $\rho_c = 1.0$, our method without jump points tends to generate a path with the shortest duration compared to both the collision-inclusive planner with jump points and the collision-avoidance planner. However, the control cost for doing so is slightly higher. The computational time of the collision-inclusive planner with jump points is the second-lowest among all the methods. Comparing these results with those obtained by the RRT* method in [4], we find the RRT*-based approach is time consuming since node rewiring requires significant computational time (about 88% of total time). Also, results are not deterministic compared to the search-based method. Thus, we deduce that the search-based collision-inclusive method with jump points can be the global planner in our unified collision-inclusive motion planning and control framework with parameters selected in this section.

### E. Simulated Tests of our Unified Collision-Inclusive Method

We first test our unified collision-inclusive motion planning and control strategy in a double corridor environment with online sensing, and compare its performance against that of a collision avoidance framework similar to [24]. In the collision avoidance framework, the global planner is the search-based method in [21]; we also make the optimistic assumption of treating the unknown space as free. The local trajectory generation method is based on gradients [42] and time duration adjustment [22]. We design a backup safety trajectory to ensure the robot will stop at the frontier. Then, we test both methods in a double corridor environment populated with circular isolated obstacles of increasing density. In all cases, each method is run for ten times with the same initial configuration and parameter settings.

We test with and without additive estimation noise in the global planner. Position estimation noise is zero-mean truncated Gaussian with variance of 0.3 and bounds of $\pm 0.9$. Velocity estimation noise is zero-mean truncated Gaussian with variance of 0.1 and bounds of $\pm 0.3$. Comparative results are presented in Figs. 11 and 12. Output trajectories of our method (with added noise) are shown in Fig. 13.

With reference to Fig. 11, when replanning every 5 s, our method generates shorter paths with a lower trajectory time on average. When the obstacle density is low (9.3%), our method generates trajectories with higher control energy; however, when the obstacle density increases (13.5% and 20.7%), our method requires lower control energy since the robot can utilize the obstacles to change its heading.\(^{15}\) When replanning every 10 s, our method consistently generates paths with lower length on average. Our method also has lower trajectory times and control energy.

With $\rho_c = 100$, $T_{\text{rep}} = 5.0$ s, our algorithm causes oscillations around the obstacle by avoiding collisions, which increase path length and trajectory time. However, its ability to use collisions makes its output trajectory better than the collision-avoidance strategy in terms of control energy, trajectory and path length. With $T_{\text{rep}} = 10.0$ s, the path length, trajectory time and control energy of the collision-inclusive trajectory is higher since there

\(^{15}\)In collision avoidance, and with $T_{\text{rep}} = 5.0$ s, the robot can get trapped oscillating in a area to avoid collisions and cannot reach the goal; however, adding some random behavior may help the robot break the tie.
is no safety maneuver making the robot stop before the frontier; hence, it will have to turn sharply and possibly oscillate when replanning. The computational time of the collision-inclusive planning is higher since it visits more nodes in the graph.

Furthermore, success rates of the proposed collision-inclusive method and collision avoidance are shown in Fig. 12. Our method has higher success rates as it addresses overconservativeness in collision avoidance to ensure safety.

We then test our proposed unified collision-inclusive motion planning and control strategy in environments with nonconvex obstacles (see Table VI and Fig. 14). With reference to the environment shown in Fig. 14(a), and using a replanning time interval $T_{rep} = 5.0$ s, our strategy can reach the goal area with higher success rates since unmodeled dynamics in physical testing make the robot collide with the obstacle even if the reference trajectory generated from collision avoidance is designed to be collision free. Furthermore, by utilizing collisions, the robot can reach the goal faster while requiring less control energy by trading off the average path length.

### VIII. CONCLUSION

#### A. Summary of Contributions and Main Findings

In this article, we proposed a unified collision-inclusive motion planning and control framework applied for navigation in unknown environment. A global search-based method was devised to generate a path that contains explicit information about collisions. The effect of the collisions was explored in the global planner. The local planner was enhanced by a lower level deformation recovery control and trajectory replanning strategy, which enabled the robot to detect and recover from collisions and move toward the goal. The deformation controller was designed based on robot dynamics, which herein was a holonomic omnidirectional wheeled robot.

The planning system was evaluated extensively through several benchmark comparisons in simulation as well as via physical experimental testing. The conducted ablation study demonstrated the utility of certain key design choices made in this work (e.g., not pruning primitives altogether), and evaluated the effect of key parameters (e.g., how much collisions are to be penalized via parameter $\rho_c$). The proposed collision-inclusive planning method was implemented in simulation first, and then, integrated with state estimation, mapping, and control into our custom-made robot platform to check the feasibility of the method in physical world experiments. Results showed that the proposed method was robust and can generate fast and safe trajectories compared to collision-avoidance methods. Overall, this work pushed forward the state-of-the-art in collision-inclusive motion planning and control, and provided a competitive alternative to traditional collision avoidance methods for a class of
Fig. 13. Simulated trajectories of the unified collision-inclusive motion planning and control framework with online sensing and noise added to the input of the global planner in the double corridor for environment with increasing obstacle density. (a) $T_{\text{rep}} = 5.0$, obstacle density $= 9.3\%$. (b) $T_{\text{rep}} = 5.0$, obstacle density $= 13.5\%$. (c) $T_{\text{rep}} = 5.0$, obstacle density $= 20.7\%$. (d) $T_{\text{rep}} = 10.0$, obstacle density $= 9.3\%$. (e) $T_{\text{rep}} = 10.0$, obstacle density $= 13.5\%$. (f) $T_{\text{rep}} = 10.0$, obstacle density $= 20.7\%$.

Fig. 14. Simulated trajectories generated from our method in environments with isolated nonconvex obstacles. (a) $T_{\text{rep}} = 5.0\text{s}$. (b) $T_{\text{rep}} = 5.0\text{s}$. (c) $T_{\text{rep}} = 10.0\text{s}$. (d) $T_{\text{rep}} = 10.0\text{s}$.

Fig. 15. Composite images of a sample experiment with our unified collision-inclusive motion planning and control framework (left) and of a collision avoidance sample experiment (right). Snapshots shown every 2.5 s. (See supplementary video file for more details.)

impact-resilient mobile robots operating in partially observable environments populated with isolated (non-)convex obstacles.

B. Discussion of Key Selections in Our Framework

1) Application to Other Robots in 2-D and 3-D: We considered the family of omnidirectional wheeled robots (see Fig. 1).

Yet, we anticipate that our proposed framework can apply to other impact-resilient robots in 2-D (e.g., wheeled [11] or aerial [13], [14], [18], [37], [49] robots) provided that they can adjust their position and redirect postimpact by using the collision to save energy. The omnidirectional wheeled robot employed here was one example along those types of robots. The higher level part of the framework can readily apply in 3-D for such systems; same holds for the overall methodology as in whole. However, the proposed lower level collision recovery would need to be adjusted to consider the 3-D dynamics for postimpact stabilization [14].

2) Use of Motion Primitives: Besides the use of motion primitives (as herein) other methods are possible. For instance, direct control of the kinematic model (1) of the specific robot considered herein, or use of fixed motion patterns (e.g., as in [38], [39], and [40]) can be viable alternatives. However, use of motion primitives at the higher level provides a unified way to make the proposed framework applicable to all the aforementioned types of robots and extendable from 2-D to 3-D, and hence, it was preferred to over simpler approaches that would have worked specifically for the omnidirectional robot we tested with herein but would be hard to scale to other types of robots. Furthermore, use of primitives allows for more flexibility, which is critical to help determine where the robot should collide with...
the environment to help it redirect toward the globally planned goal; this was achieved by directly using information on the velocity as per (12).

3) Choosing a Search- or Sampling-Based Global Planner: We showed that it is possible to derive collision-inclusive planning frameworks with the global planner being either search-based or sampling-based. Each has its own strengths and weaknesses, and as a matter of fact, our results are consistent with observations made in collision-avoidance methods. Consistent with collision avoidance, a user can choose which approach to select (search-based over sampling-based global planner) according to their application needs; our proposed framework can accommodate both. We highlight here that the sampling-based global planner can be further optimized by biasing search toward free space to increase the computational efficiency (e.g., [50]). Integration of the sampling-based planner into our overall real-time framework would require further adaptations of the collision-inclusive RRT* planner to make it online (faster nearest neighbor search, minimal cost path generation, and optimized rewiring methods). Similar to collision-avoidance online RRT* methods (e.g., [51] and [52]), a collision-inclusive anytime planning algorithm was required to extend the RRT* method for planning collision-inclusive trajectories online.

C. Directions for Future Work

The framework developed herein lays the basis toward a general method for collision-inclusive motion planning and control, and created multiple opportunities for future research along these lines. These include extension to other robots and to systems with higher order dynamics, evaluation of direct controllers against motion primitives (as well as different parameterizations of the latter), and integration of sampling-based planners into the overall framework.

Furthermore, at its current form, our method does not consider the perception model of the robot in online planning; extension of the algorithm to consider the perception problem based on the collision-inclusive method was another interesting direction of future research. Finally, we showed that it was possible to handle navigation in environments populated with isolated nonconvex environments; however, study of navigation in more cluttered environments (e.g., maze-like) is a direction of research enabled by this work.

ACKNOWLEDGMENT

The authors would like to thank H. Teng for his help implementing the mapping package used in this work. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies.

REFERENCES

[1] Z. Lu and K. Karydis, “Optimal steering of stochastic mobile robots that undergo collisions with their environment,” in Proc. IEEE/RSJ Int. Conf. Robot. Biomimetics, 2019, pp. 668–675.

[2] Z. Lu, Z. Liu, G. J. Correa, and K. Karydis, “Motion planning for collision-resilient mobile robots in obstacle-cluttered unknown environments with risk reward trade-offs,” in Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., 2020, pp. 7064–7070.

[3] M. Mote, M. Egerstedt, E. Feron, A. Bylard, and M. Pavone, “Collision-inclusive trajectory optimization for free-flying spacecraft,” J. Guid., Control, Dyn., vol. 43, no. 7, pp. 1247–1258, 2020.

[4] J. Zha and M. W. Mueller, “Exploiting collisions for sampling-based multicopter motion planning,” in Proc. IEEE/RSJ Int. Conf. Robot. Automat., 2021, pp. 7943–7949.

[5] T. Lew et al., “Contact inertial odometry: Collisions are your friends,” in Proc. Int. Symp. Robot. Res., 2019, pp. 938–958.

[6] T. Schmickl et al., “Get in touch: Cooperative decision making based on robot-to-robot collisions,” Auton. Agents Multi-Agent Syst., vol. 18, no. 1, pp. 133–155, 2009.

[7] K. Karydis, D. Zarrouk, I. Poulakakis, R. S. Fearing, and H. G. Tanner, “Planning with the star (s),” in Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., 2014, pp. 3033–3038.

[8] D. W. Haldane, M. M. Plecnik, J. K. Yim, and R. S. Fearing, “Robotic vertical jumping agility via series-elastic power modulation,” Sci. Robot., vol. 1, no. 1, 2016.

[9] Y. Mulgaonkar, A. Makineni, L. Guerrero-Bonilla, and V. Kumar, “Robust aerial robot swarms without collision avoidance,” IEEE Robot. Automat. Lett., vol. 3, no. 1, pp. 596–603, Jan. 2018.

[10] S. Mayya, P. Pierpaoli, G. Nair, and M. Egerstedt, “Localization in densely packed swarms using interrobot collisions as a sensing modality,” IEEE Trans. Robot., vol. 35, no. 1, pp. 21–34, Feb. 2019.

[11] A. Stager and H. G. Tanner, “Composition of local potential functions with reflection,” in Proc. IEEE Int. Conf. Robot. Automat., 2019, pp. 5558–5564.

[12] N. Khedekar, F. Mascarich, C. Papachristos, T. Dang, and K.Alexis, “Contact-based navigation path planning for aerial robots,” in Proc. IEEE Int. Conf. Robot. Automat., 2019, pp. 4161–4167.

[13] Y. Mulgaonkar, W. Liu, D. Thakur, K. Danilidis, C. J. Taylor, and V. Kumar, “The Tiercel: A novel autonomous micro aerial vehicle that can map the environment by flying into obstacles,” in Proc. IEEE Int. Conf. Robot. Automat., 2020, pp. 7448–7454.

[14] Z. Liu and K. Karydis, “Toward impact-resilient quadrotor design, collision characterization and recovery control to sustain flight after collisions,” in Proc. IEEE Int. Conf. Robot. Automat., 2021, pp. 183–189.

[15] M. H. A. S. Matveev, and A. V. Savkin, “Algorithms for collision-free navigation of mobile robots in complex cluttered environments: A survey,” Robotica, vol. 33, no. 3, pp. 463–497, 2015.

[16] S. Campbell, W. Naeem, and G. W. Irwin, “A review on improving the autonomy of unmanned surface vehicles through intelligent collision avoidance manoeuvres,” Annu. Rev. Control, vol. 36, no. 2, pp. 267–283, 2012.

[17] T. Okunev, Z. Taylor, R. Siegwart, and J. Nieto, “Safe local exploration for replanning in cluttered unknown environments for microaerial vehicles,” IEEE Robot. Automat. Lett., vol. 3, no. 3, pp. 1474–1481, Jul. 2018.

[18] P. De Petris, H. Nguyen, T. Dang, F. Mascarich, and K. Alexis, “Collision-tolerant autonomous navigation through manhole-size confined environments,” in Proc. IEEE Int. Symp. Saf., Secur., Rescue Robot., 2020, pp. 1–6.

[19] A. Stager and H. G. Tanner, “Stochastic behavior of robots that navigate by interacting with their environment,” in Proc. IEEE 53th Conf. Decis. Control, 2016, pp. 6871–6876.

[20] Z. Lu, Z. Liu, and K. Karydis, “Deformation recovery control and post-impact trajectory replanning for collision-resilient mobile robots,” in Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., 2021, pp. 2030–2037.

[21] S. Liu, N. Atanasov, K. Mohta, and V. Kumar, “Search-based motion planning for quadrotors using linear quadratic minimum time control,” in Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., 2017, pp. 2872–2879.

[22] S. Liu et al., “Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-D complex environments,” IEEE Robot. Automat. Lett., vol. 2, no. 3, pp. 1688–1695, Jul. 2017.

[23] B. Zhou, J. Pan, F. Gao, and S. Shen, “Raptor: Robust and perception-aware trajectory replanning for quadrotor fast flight,” IEEE Trans. Robot., vol. 37, no. 6, pp. 1992–2009, Dec. 2021.

[24] J. Tordesillas, B. T. Lopez, M. Everett, and J. P. How, “Faster: Fast and safe trajectory planner for navigation in unknown environments,” IEEE Trans. Robot., vol. 38, no. 2, pp. 922–938, Apr. 2022.
[25] R. Deits and R. Tedrake, “Efficient mixed-integer planning for UAVs in cluttered environments,” in Proc. IEEE Int. Conf. Robot. Automat., 2015, pp. 42–49.

[26] F. Gao, W. Wu, Y. Lin, and S. Shen, “Online safe trajectory generation for quadrotors using fast marching method and Bernstein basis polynomial,” in Proc. IEEE Int. Conf. Robot. Automat., 2018, pp. 344–351.

[27] J. Tordesillas, B. T. Lopez, J. Carter, J. Ware, and J. P. How, “Real-time planning with multi-fidelity models for agile flights in unknown environments,” in Proc. IEEE Int. Conf. Robot. Automat., 2019, pp. 725–731.

[28] D. Liu, M. Cong, and Y. Du, “Episodic memory-based robotic planning under uncertainty,” IEEE Trans. Ind. Electron., vol. 64, no. 2, pp. 1762–1772, Feb. 2017.

[29] B. T. Lopez and J. P. How, “Aggressive collision avoidance with limited field-of-view sensing,” in Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., 2017, pp. 1358–1365.

[30] K. Mohta et al., “Fast, autonomous flight in GPS-denied and cluttered environments,” J. Field Robot., vol. 35, no. 1, pp. 101–120, 2018.

[31] C. Richter and N. Roy, “Learning to plan for visibility in navigation of unknown environments,” in Proc. Int. Symp. Exp. Robot., 2016, pp. 387–398.

[32] E. Heiden, K. Hausman, G. S. Sukhatme, and A.-a. Agha-mohammadi, “Planning high-speed safe trajectories in confidence-rich maps,” in Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., 2017, pp. 2890–2896.

[33] A. Briod, P. Komatowski, J.-C. Zufferey, and D. Floreano, “A collision-resilient flying robot,” J. Field Robot., vol. 31, no. 4, pp. 496–509, 2014.

[34] D. W. Haldane et al., “ Integrated manufacture of exoskeletons and sensing structures for folded millirobots,” J. Mechanisms Robot., vol. 7, no. 2, 2015, Art. no. 021011.

[35] T. Li et al., “Agile and resilient insect-scale robot,” Soft Robot., vol. 6, no. 1, pp. 133–141, 2019.

[36] A. Battistion, I. Sharf, and M. Nahon, “Attitude estimation for collision recovery of a quadcopter unmanned aerial vehicle,” Int. J. Robot. Res., vol. 38, no. 10–11, pp. 1286–1306, 2019.

[37] A. Briod et al., “Contact-based navigation for an autonomous flying robot,” in Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., 2013, pp. 3987–3992.

[38] A. Q. Nilles, Y. Ren, I. Becerra, and S. M. LaValle, “A visibility-based approach to computing non-deterministic bouncing strategies,” Int. J. Robot. Res., vol. 40, no. 10–11, pp. 1196–1211, 2021.

[39] T. Alam, L. Bobadilla, and D. A. Shell, “ Minimalist robot navigation and coverage using a dynamical system approach,” in Proc. IEEE Int. Conf. Robot. Comput., 2017, pp. 249–256.

[40] J. S. Lewis and J. M. O’Kane, “Planning for provably reliable navigation using an unreliable, nearly sensorless robot,” Int. J. Robot. Res., vol. 32, no. 11, pp. 1342–1357, 2013.

[41] P. E. Sandin, Robot Mechanisms and Mechanical Devices Illustrated, New York, NY, USA: McGraw-Hill, 2003.

[42] F. Gao, Y. Lin, and S. Shen, “Gradient-based online safe trajectory generation for quadrotor flight in complex environments,” in Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., 2017, pp. 3681–3688.

[43] M. Likhachev, G. J. Gordon, and S. Thrun, “A*: Anytime A* with provable bounds on sub-optimality,” in Proc. Adv. Neural Inf. Process. Syst., 2003, vol. 16, pp. 767–774.

[44] B. Zhou, F. Gao, L. Wang, C. Liu, and S. Shen, “Robust and efficient quadrotor trajectory generation for fast autonomous flight,” IEEE Robot. Automat. Lett., vol. 4, no. 4, pp. 3529–3536, Oct. 2019.

[45] F. L. Lewis, D. Vrabie, and V. L. Syrmos, Optimal Control, Hoboken, NJ, USA: Wiley, 2012.

[46] E. Verriest and F. Lewis, “On the linear quadratic minimum-time problem,” IEEE Trans. Autom. Control, vol. 36, no. 7, pp. 859–863, Jul. 1991.

[47] D. Harabor and A. Grastien, “Online graph pruning for pathfinding on grid maps,” in Proc. AAAI Conf. Artif. Intell., 2011, vol. 25, pp. 1114–1119.

[48] C. Richter, A. Bry, and N. Roy, “ Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments,” in Robotics Research, Berlin, Germany: Springer, 2016, pp. 649–666.

[49] J. Zha, X. Wu, J. Kroeger, N. Perez, and M. W. Mueller, “A collision-resistant aerial vehicle with icosahedron tensegrity structure,” in Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., 2020, pp. 1407–1412.

[50] R. Sandström, D. Uwacu, J. Denny, and N. M. Amato, “Topology-guided roadmap construction with dynamic region sampling,” IEEE Robot. Automat. Lett., vol. 5, no. 4, pp. 6161–6168, Oct. 2020.

[51] J. D. Hernández, E. Vidal, G. Vallicrosa, E. Galceran, and M. Carreras, “Online path planning for autonomous underwater vehicles in unknown environments,” in Proc. IEEE Int. Conf. Robot. Automat., 2015, pp. 1152–1157.

[52] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, “Anytime motion planning using the RRT,” in Proc. IEEE Int. Conf. Robot. Automat., 2011, pp. 1478–1483.

Zhouyu Lu received the B.S. degree in electrical engineering from the Nanjing University of Aeronautics and Astronautics, Nanjing, China, in 2017. He is currently working toward the Ph.D. degree in electrical engineering from the University of California, Riverside, CA, USA.

His research interests include motion planning for autonomous robots.

Zhichao Liu received the B.S. degree in control technology and instruments from the Beijing University of Posts and Telecommunications, Beijing, China, in 2014, and the M.S. degree in electrical and systems engineering from the University of Pennsylvania, Philadelphia, PA, USA, in 2018. He is currently working toward the Ph.D. degree in electrical engineering with the University of California, Riverside, CA, USA.

His research interests include compliant and soft robotics with applications to aerial and legged robots.

Merrick Campbell (Member, IEEE) received the B.S. degree in mechanical engineering from the University of California, Los Angeles, Los Angeles, CA, USA, in 2016, and the M.S. degree in electrical engineering from the University of California, Riverside, Riverside, CA, in 2022.

His research interests include physical leaf sampling and proximal soil sensing for robot-assisted precision irrigation.

Konstantinos Karydis (Member, IEEE) received the Diploma (Hons.) degree in mechanical engineering from the National Technical University of Athens, Athens, Greece, in 2010, and the Ph.D. degree in mechanical engineering from the University of Delaware, Newark, DE, USA, in 2015. He is currently an Associate Professor of electrical and computer engineering with the University of California, Riverside (UCR), Riverside, CA, USA. Before joining the UCR as an Assistant Professor in 2017, he was a Postdoctoral Researcher in robotics with the Department of Mechanical Engineering and Applied Mechanics, University of Pennsylvania. His research interests include robot motion planning, learning and control under uncertainty in precision agriculture, emergency response, and assistive robotics applications.

Dr. Karydis was the recipient of an NSF CAREER Award in 2021.