Interaction-Aware Multi-Robot Kinodynamic Motion Planning

Akmaral Moldagalieva[1], Joaquim Ortiz-Haro[2], Wolfgang Hönig[1]

***Abstract*— Kinodynamic motion planning for a multi-robot system with different dynamics and actuation limits is a challenging problem. The difficulty increases with the presence of an aerodynamic interaction force that occur in aerial robots flying in close-proximity. Due to these complexities, existing planners either rely on simplified assumption like ignoring robot dynamics, interaction forces or produce highly suboptimal solutions. This paper presents a kinodynamic motion planner for a heterogeneous team of robots that respects robot dynamics and directly reasons about interaction forces between aerial robots operating in close-proximity. Our method, db-ECBS, generalizes the multi-agent path finding method Enhanced Conflict-Based Search (ECBS) to the continuous domain by using the single-robot kinodynamic motion planner discontinuity-bounded A*. Db-ECBS operates on three levels. Initially, individual robot trajectories are computed using a graph search that allows bounded discontinuities between precomputed motion primitives. The second level identifies inter-robot collisions, interaction force violations and resolves them by imposing constraints on the first level. The third and final level uses the resulting solution with discontinuities as an initial guess for a joint space trajectory optimization. The procedure is repeated with a reduced discontinuity bound resulting in a anytime, probabilistically complete, and asymptotically bounded suboptimal planner. We provide a benchmark of 65 problems with six different dynamics. We demonstrate that db-ECBS produces trajectories that are less than half the cost of existing planners. We show that the interaction-awareness is in particular important for very dense scenarios.**

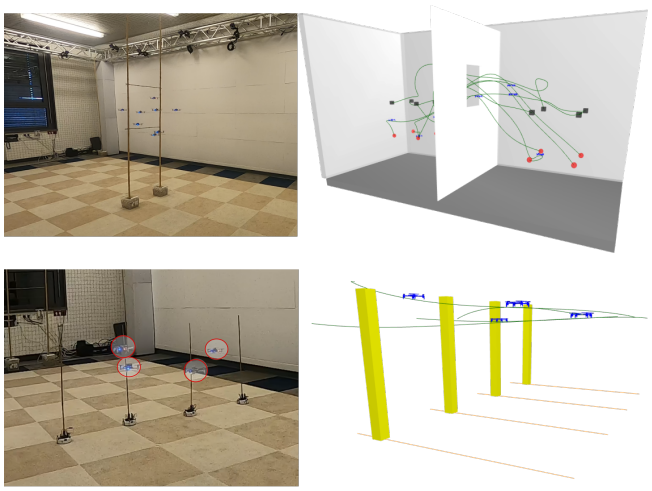


Fig. 1. Top row: *Window* example with eight flying robots. Robots are required to swap their positions by passing through a small window. Red circles show the starting position, while dark boxes represent the final state. Bottom row: Example with four ground and four flying robots. Ground robots move forward in a straight line, while flying robots pass through 0.6m high bamboo sticks that are mounted on the moving ground robots to reach their goals.

I. Introduction

Multi-robot systems have a wide-range of real-world applications including delivery, collaborative transportation, and search-and-rescue. These scenarios require multiple robots to operate in close proximity to each other for the sake of efficiency. For example, a group of flying robots operating independently in a shared space, such as warehouse, where some are scanning shelves, while others transporting items.

One of the essential requirements to enhance the autonomy of a team of robots in these settings is being able to reach the goal quickly while avoiding collisions with obstacles and other robots. Moreover, the planned motions are required to respect the robots' dynamics which impose constraints on their velocity or acceleration.

Considering the complexity of Multi-Robot Motion Planning (MRMP), the majority of existing solutions either make simplified assumptions like ignoring the robot dynamics or actuation limits [1, 2] or produce highly suboptimal solutions [3]. Moreover, they ignore aerodynamic interaction forces between aerial robots operating in close-proximity. A major challenge of close-proximity operation is that small distances between flying robots create an aerodynamic interaction force. Ignoring this force during planning might result in dangerous formations, which may cause robots to crash.

In this paper, we present an anytime, probabilistically-complete and asymptotically bounded suboptimal motion planner for a heterogeneous team of robots. Our method takes into account constraints imposed by robot dynamics and directly reasons about interaction forces between aerial robots in close formations.

Our approach leverages the Multi-Agent Path Finding (MAPF) bounded-suboptimal solver Enhanced Conflict-Based Search (ECBS) [2], the single-robot kinodynamic motion planner discontinuity-bounded A* (db-A*) [4], and nonlinear trajectory optimization.

Our algorithm, called Discontinuity-Bounded Enhanced Conflict-Based Search (db-ECBS), performs a three-level search. On the low-level, the single-robot motion is planned separately for each robot relying on pre-computed motion primitives obeying the robot's dynamics. The computed

[1] Technical University of Berlin, Berlin, Germany moldagalieva@tu-berlin.de.

[2] Machines in Motion Laboratory, New York University, USA

We thank Nan Cai, Jan Achermann, and Dennis Schmidt for help with the robot experiments.

Code: https://github.com/IMRCLab/db-CBS/tree/dbecbs-tro

Video: https://youtu.be/OcG-59Pq3oY

The research was partially funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - 448549715.

trajectories may result in inter-robot collisions, or interaction force violations which are resolved one-by-one in the second-level search. Different from ECBS, we allow the trajectories to be dynamically infeasible up to a given discontinuity bound during these two searches. Then, we use the output trajectories as an initial guess for a joint space trajectory optimization on the third level. This is repeated with a lower discontinuity bound in case of failure or if a lower-cost solution is desired.

The main contribution of this work is a new kinodynamic motion planner for multi-robot systems that respects robot dynamics and reasons directly about interaction forces occurring in flying robot teams. Our planner, db-ECBS, is anytime, probabilistically-complete and asymptotically bounded suboptimal. We provide a benchmark on a set of different 65 problems across six robot dynamics. We compare our method with other kinodynamic motion planning methods with the identical objective of computing time-optimal trajectories. We demonstrate that our approach consistently finds a solution when the environment forces dense formations with flying robots. Finally, we show our planner in application with two test scenarios on physical robots including flying and ground robots (Fig. 1).

II. Related Work

Multi-Agent Path Finding (MAPF) assumes a discrete state space represented as a graph. A robot can move from one vertex along an edge to an adjacent neighbor in one step; robots cannot occupy the same vertex or traverse the same edge at the same timestep. For collision checking, vertices are often placed in a grid, but any-angle extensions exist [5]. Finding an optimal MAPF solution is NP-hard [6]. Current algorithms plan paths for each robot individually and then resolve conflicts by fixing velocities [7], assigning priorities to robots [8] or adding path constraints as in an optimal MAPF solver Conflict-Based Search (CBS) [1, 2]. Although MAPF has practical applications in real-world problems, it ignores robot dynamics and can result in kinodynamically infeasible solutions. The planned paths may be followed using a controller and plan-execution framework [9]. However, this is suboptimal and does not generalize to all robot types.

Kinodynamic motion planning remains challenging, even for single robots. Search-based methods with motion primitives have been adapted to a variety of robotic systems [10, 11], including high-dimensional systems [12]. Those motion primitives are on a state lattice, thus they are able to connect states properly and can be designed manually [13]. Afterwards, any variant of discrete path planning can be used without modifications. Sampling-based planners expand a tree of collision-free and dynamically feasible motions [14–16]. Even though solutions have probabilistic completeness guarantees, they are suboptimal and require post-processing to smooth the trajectory. Optimization-based planners rely on an initial guess and optimize locally [17], for example by using sequential convex programming (SCP) [18, 19]. Hybrid methods can combine search and sampling [20], search and optimization [21] or sampling and optimization [22].

TABLE I
A comparison of related work

	[16]	[23]	[24]	[25]	[3]	Ours
Anytime	✓	✗	✗	✓	✗	✓
Probabilistically Complete	✓	✗	✓	✓	✗	✓
Asymptotically Optimal	✓	✗	✓	✓	✗	✓
Dynamics-agnostic	✓	✓	✓	✓	✗	✓
Interaction-aware	✗	✗	✗	✗	✓	✓

For multi-robot kinodynamic motion planning, single-robot planners applied to the joint space can be used [16], but do not scale well beyond a few robots. Extensions of CBS can tackle the MRMP problems. For example, discretized workspace grid cells can be connected via predefined motion primitives [26]. However, computing primitives remains challenging for higher-order dynamics.

Another option is to use model predictive control as the low-level planner to generate safe plans for multiple robots simultaneously [27], however this method can fail when the initial guess is far from a dynamically feasible solution. K-CBS [24] uses any sampling-based planner as the low-level planner and merges the search space of two robots if the number of conflicts between the robots exceeds a threshold. In the worst case, all robots may be merged but probabilistic completeness is achieved. Similar to CBS, this method does not scale to a large number of robots. Scalable and efficient multi-robot motion planner Safe Multi-Agent Motion Planning with Nonlinear Dynamics and Bounded Disturbances (S2M2) [23] combines Mixed-Integer Linear Programs (MILP) and control theory by computing piecewise linear paths that a controller should be able to track within a safe region. These safe regions can be large, which makes the method incomplete.

While the above-mentioned methods enhance multi-robot motion planning, they ignore the aerodynamic interaction forces between aerial robots. When two robots fly in close proximity there is a non-negligible force between them in z-direction [28] caused by airflow between robots. This force can cause the lower robot to deviate from its planned trajectory. Interaction forces can be measured in a lab settings using two side-by-side robots [29], however it is unclear how this affects the flight of multiple multirotors. Propeller velocity field model can be used to estimate the interaction force between two multirotors [30]. However, it is difficult to extend this method to multi-robot case. Residual forces of up to 4 multirotots can be measured within a laboratory environment using a load-stand setup [31], and has shown to negatively affect the flight performance of multirotors. Conservative strategies of avoiding these aerodynamic disturbances are limiting the relative position of multirotors in the 2D horizontal plane [32], enforcing large inter-robot separations by approximating each robots shape as an ellipsoid [3, 33] or cylinder with large radii [34]. However, these methods result in sparse robot formations and assume simplified robot dynamics.

Deep neural networks offer precise and efficient aerodynamic effect estimation due to their ability to recognize

patterns within large datasets. Recent works augment single multirotors dynamics model by fusing learned residual forces with blade-element-momentum (BEM) theory [35]. This hybrid model accurately captures complex aerodynamic effects allowing high-speed flight maneuvers. For multi-robot settings, learned residual interaction forces can be combined with multirotor nominal dynamics model in order to design a stable controller [36] and interaction-aware motion planning [37, 38] for close-proximity flight scenarios. It has been shown that the learned residual dynamics reduce the tracking error, so robots do not deviate from their planned trajectories [39]. However, current state-of-the-art interaction-aware multirobot motion planners are limited to a simplified 2D model being tested with only 3 multirotors [37] and not scalable with more than 2 robots due to computational burden of the approach [38]. Table I summarizes a comparison between state-of-the-art multi-robot motion planners and our planner db-ECBS.

III. Problem Definition

We consider N robots and denote the state of the i^{th} robot with $\mathbf{x}^{(i)} \in \mathcal{X}^{(i)} \subset \mathbb{R}^{d_{x^{(i)}}}$, which is actuated by controlling actions $\mathbf{u}^{(i)} \in \mathcal{U}^{(i)} \subset \mathbb{R}^{d_{u^{(i)}}}$. Robots with identical dynamics, geometric shapes are considered to be of the same type. The workspace the robots operate in is given as $\mathcal{W} \subseteq \mathbb{R}^{d_w}$ ($d_w \in \{2, 3\}$). The collision-free space is $\mathcal{W}_{\text{free}} \subseteq \mathcal{W}$.

We define the *relative state* between robot i and robot j with $\mathbf{x}^{(ij)} = [\mathbf{x}^{(i)} - \mathbf{x}^{(j)}]$. The set of the relative states and neighbor robot types of the i^{th} robot is given as $\mathbf{r}^{(i)} = \{\mathbf{x}^{(ij)}, \langle \mathcal{I}(j) \rangle | j \in \mathcal{N}(i)\}$. Here, $\mathcal{N}(\cdot)$ is a function for defining neighboring robots, which relies on an interaction volume. We assume $j \in \mathcal{N}(i)$, i.e., the robot j is a neighbor of the robot i iff the position of the robot j is within the interaction volume of i. The $\langle \mathcal{I}(j) \rangle$ retrieves the type of the neighboring robot j.

We assume that each robot $i \in \{1, \dots, N\}$ has dynamics:

$$\dot{\mathbf{x}}^{(i)} = \mathbf{f}^{(i)}(\mathbf{x}^{(i)}, \mathbf{u}^{(i)}, \boldsymbol{\psi}^{(i)}(\mathbf{r}^{(i)})), \tag{1}$$

where $\boldsymbol{\psi}^{(i)}(\cdot)$ is an additional disturbance force of the i^{th} robot created by interactions with other neighboring robots.

The Jacobian of $\mathbf{f}^{(i)}$ with respect to $\mathbf{x}^{(i)}$, $\mathbf{u}^{(i)}$ are assumed to be available in order to use gradient-based optimization.

With zero-order hold discretization, (1) can be framed as

$$\mathbf{x}^{(i)}_{k+1} \approx \text{step}(\mathbf{x}^{(i)}_k, \mathbf{u}^{(i)}_k, \boldsymbol{\psi}^{(i)}(\mathbf{r}^{(i)}_k))\Delta t, \tag{2}$$

where Δt is a time step and sufficiently small to ensure that the Euler integration holds.

Example 1: Consider a homogeneous system of two double integrator robots with states $\mathbf{x}^{(1)} = [\mathbf{p}^{(1)}, \mathbf{v}^{(1)}]$, $\mathbf{x}^{(2)} = [\mathbf{p}^{(2)}, \mathbf{v}^{(2)}]$. Here, $\mathbf{p} = [x, y, z] \in \mathbb{R}^3$ are the position, $\mathbf{v} = [v_x, v_y, v_z] \in \mathbb{R}^3$ are the velocity. The control inputs $\mathbf{u^{(1)}}, \mathbf{u^{(2)}} \in \mathbb{R}^3$, where $\mathbf{u} = [a_x, a_y, a_z]$ represents the acceleration. The dynamics are

$$\dot{\mathbf{p}}^{(1)} = \mathbf{v}^{(1)}, \tag{3a}$$
$$\dot{\mathbf{v}}^{(1)} = \mathbf{a}^{(1)} + \boldsymbol{\psi}, \tag{3b}$$
$$\dot{\mathbf{p}}^{(2)} = \mathbf{v}^{(2)}, \tag{3c}$$
$$\dot{\mathbf{v}}^{(2)} = \mathbf{a}^{(2)} - \boldsymbol{\psi}. \tag{3d}$$

The disturbance force $\boldsymbol{\psi}$ can be described as the repulsive force which depends on the relative position between robots:

$$\boldsymbol{\psi} = \frac{\lambda(\mathbf{p}^{(1)} - \mathbf{p}^{(2)})}{\left\|\mathbf{p}^{(1)} - \mathbf{p}^{(2)}\right\|^3}, \tag{4}$$

where λ is a user-defined scaling factor.

We use $\mathbf{X}^{(i)} = \langle \mathbf{x}^{(i)}_0, \mathbf{x}^{(i)}_1, \dots, \mathbf{x}^{(i)}_{K^{(i)}} \rangle$ as a sequence of states of the i^{th} robot sampled at times $0, \Delta t, \dots, K^{(i)}\Delta t$ and $\mathbf{U}^{(i)} = \langle \mathbf{u}^{(i)}_0, \mathbf{u}^{(i)}_1, \dots, \mathbf{u}^{(i)}_{K^{(i)}-1} \rangle$ as a sequence of actions applied to the i^{th} robot for times $[0, \Delta t), [\Delta t, 2\Delta t), \dots, [(K^{(i)} - 1)\Delta t, K^{(i)}\Delta t)$. Our goal is to move the team of N robots from their start states $\mathbf{x}^{(i)}_s \in \mathcal{X}^{(i)}$ to their goal states $\mathbf{x}^{(i)}_f \in \mathcal{X}^{(i)}$ as fast as possible without collisions. This can be formulated as the following optimization problem:

$$\min_{\{\mathbf{X}^{(i)}\},\{\mathbf{U}^{(i)}\},\{K^{(i)}\}} \sum_{i=1}^{N} K^{(i)} \tag{5}$$
$$\text{s.t.} \begin{cases} \mathbf{x}^{(i)}_{k+1} = \text{step}(\mathbf{x}^{(i)}_k, \mathbf{u}^{(i)}_k, \boldsymbol{\psi}^{(i)}(\mathbf{r}^{(i)}_k)) & \forall i\ \forall k, \\ \mathbf{u}^{(i)}_k \in \mathcal{U}^{(i)}, \quad \mathbf{x}^{(i)}_k \in \mathcal{X}^{(i)} & \forall i\ \forall k, \\ \|\boldsymbol{\psi}^{(i)}(\mathbf{r}^{(i)}_k)\| \leq \psi^{(i)}_{\text{max}} & \forall i\ \forall k, \\ \mathcal{B}^{(i)}(\mathbf{x}^{(i)}_k) \in \mathcal{W}_{\text{free}} & \forall i\ \forall k, \\ \mathcal{B}^{(i)}(\mathbf{x}^{(i)}_k) \cap \mathcal{B}^{(j)}(\mathbf{x}^{(j)}_k) = \emptyset & \forall i \neq j\ \forall k, \\ \mathbf{x}^{(i)}_0 = \mathbf{x}^{(i)}_s, \quad \mathbf{x}^{(i)}_{K^{(i)}} = \mathbf{x}^{(i)}_f & \forall i, \end{cases}$$

where $\mathcal{B}^{(i)} : \mathcal{X}^{(i)} \to 2^{\mathcal{W}}$ is a function that maps the configuration of the i^{th} robot to a collision shape. $\mathbf{x}^{(i)}_s \in \mathcal{X}^{(i)}$ is the start state, $\mathbf{x}^{(i)}_f \in \mathcal{X}^{(i)}$ is the goal state and $\psi^{(i)}_{\text{max}}$ is the maximum endurable interaction force for the i^{th} robot. The objective is to minimize the sum of the arrival time of all robots.

IV. Background

We first describe db-A* and CBS with its bounded suboptimal variant ECBS in more detail, since our work generalizes them into a multi-robot case operating in continuous space and time.

db-A* is a generalization of A* for kinodynamic motion planning of a single robot that searches a graph of *motion primitives*, i.e., precomputed motions respecting the robot's dynamics [4].

A single motion primitive can be defined as a tuple $\langle \mathbf{X}, \mathbf{U}, K \rangle$, consisting of state sequences $\mathbf{X} = \langle \mathbf{x}_0, ..., \mathbf{x}_K \rangle$ and control sequences $\mathbf{U} = \langle \mathbf{u}_0, ..., \mathbf{u}_{K-1} \rangle$ which obey the dynamics $\mathbf{x}_{k+1} = \text{step}(\mathbf{x}_k, \mathbf{u}_k)$. These motion primitives are used as graph edges to connect states, representing graph nodes, with user-configurable discontinuity δ.

As in A*, db-A* explores nodes based on $f(\mathbf{x}) = g(\mathbf{x}) + h(\mathbf{x})$, where $g(\mathbf{x})$ is the cost-to-come. The node with the lowest f-value is expanded by applying valid motion primitives. The output of db-A* is a δ-discontinuity-bounded solution as defined below.

Definition 1: Sequences $\mathbf{X} = \langle \mathbf{x}_0, \dots, \mathbf{x}_K \rangle$, $\mathbf{U} = \langle \mathbf{u}_0, \dots, \mathbf{u}_{K-1} \rangle$ are *δ-discontinuity-bounded* solutions iff the following conditions hold:

$$\begin{aligned} d(\mathbf{x}_{k+1}, \mathrm{step}(\mathbf{x}_k, \mathbf{u}_k)) \leq \delta \quad \forall k, \\ \mathbf{u}_k \in \mathcal{U}, \quad \mathbf{x}_k \in \mathcal{X}, \quad \mathcal{B}(\mathbf{x}_k) \in \mathcal{W}_{\mathrm{free}} \quad \forall k, \\ d(\mathbf{x}_0, \mathbf{x}_s) \leq \delta, \quad d(\mathbf{x}_K, \mathbf{x}_f) \leq \delta, \end{aligned} \tag{6}$$

where d is a *metric* $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, which measures the distance between two states.

CBS is an optimal MAPF solver which works in two levels. The high-level search finds conflicts between single-robot paths, while at the low-level individual paths are updated to fulfill time-dependent constraints using a single-robot planner. Each high-level node contains a set of constraints and, for each robot, a path which obeys the given constraints. The cost of the high-level node is equal to the sum of all single-robot path costs. When a high-level node P is expanded, it is considered as the goal node, if its individual robot paths have no conflicts. If the node P has no conflicts, then its set of paths is returned as the solution. Otherwise, two child nodes, P_1 and P_2 are added. Both child nodes inherit constraints of P and have an additional constraint to resolve the last conflict. For example, if the collision happened at vertex s at time t between robots i and j, then the node P_1 has an additional constraint $\langle i, s, t \rangle$, which prohibits the robot i to occupy vertex s at time t. Likewise, the constraint $\langle j, s, t \rangle$ is added to P_2. Afterwards, given the set of constraints for each node, the low-level planner is executed for the affected robot, only.

Enhanced CBS (**ECBS**) is a bounded-suboptimal variant of CBS. It takes a suboptimality factor ω and returns a solution which is guaranteed to be within an ω-factor from the optimal solution. The value of ω defines the trade-off between runtime and solution quality. Compared to CBS, in ECBS both the high- and the low-level searches use *Focal search* in order to perform bounded suboptimal search. The focal search maintains an additional set of nodes for the search. This set contains all nodes cost *less or equal* to $\omega \cdot LB$, where the LB is the lower bound on the optimal cost. Focal search can use an inadmissible *conflict heuristic* for node expansion. In ECBS, this heuristic prioritizes nodes with a lower number of conflicts with other robots.

V. Discontinuity-Bounded Conflict-Based Search (db-CBS) - Overview

We summarize the multi-robot kinodynamic motion planner db-CBS first, thus extensions can be explained better. The planner adapts CBS to the kinodynamic case. The structure of db-CBS consists of the following steps:

1) A single-robot motion with discontinuous jumps is computed for each robot using db-A*.
2) Collisions between individual motions are resolved one-by-one.
3) Discontinuous motions are repaired into smooth and feasible trajectories with optimization.
4) Steps are repeated with lower discontinuity bound and more motion primitives.

Although the two-level framework of CBS remains unchanged in this approach, the definition of conflicts and constraints need to be changed for the continuous domain. A conflict is defined as $C = \langle i, j, \mathbf{x}_k^{(i)}, \mathbf{x}_k^{(j)}, k \rangle$ for a collision between robot i with state $\mathbf{x}_k^{(i)}$ and robot j with state $\mathbf{x}_k^{(j)}$ identified at time k. The resulting constraint for robot i is $\langle i, \mathbf{x}_k^{(i)}, k \rangle$, which prevents it to be within a distance of δ to state $\mathbf{x}_k^{(i)}$ at time k. Similarly, the constraint for robot j is $\langle j, \mathbf{x}_k^{(j)}, k \rangle$. The notion of a discontinuity δ defines the constraint as an actual volume (around a point), which is crucial for efficiency and completeness guarantees. Db-CBS focuses on scenarios without interaction forces.

The high-level search of db-CBS is shown in Alg. 1. Its major changes compared to CBS are highlighted with yellow.

Initially, db-A* is used to compute heuristics $\mathcal{H}^{(i)}$ for each robot $i \in \{1, \dots, N\}$ by performing a reverse search from the given goal state $\mathbf{x}_f^{(i)}$ using a subset of the provided set of motion primitives $\mathcal{M}^{(i)}$. The expanded states and their g-values define a lower-bound of the cost to the goal and are thus an admissible heuristic (up to the choice of motion primitives). The values can be looked up efficiently at runtime by storing them in a kd-tree.

Db-CBS finds a solution in an iterative manner and decreases the value of the discontinuity allowed in the final path with each iteration. `AddPrimitives` adds more motion primitives from a pre-computed set of motions (Line 10). The nodes to explore are maintained using OPEN $\mathcal{O}$ priority queue. In $\mathcal{O}$, the nodes are sorted by the lowest value of $f(\mathbf{x}) = g(\mathbf{x}) + h(\mathbf{x})$, where $g(\mathbf{x})$ is the cost-to-come.

In each iteration, the root node of the constraint tree is initialized by running the low-level planner for each individual robot separately with the given set of motion primitives, current δ and no constraints (Line 14). Then the root node is added to $\mathcal{O}$ (Line 17). These motions are required to avoid robot-obstacle collisions and specified constraints. If single-robot motions are found successfully, they are validated for inter-robot collision (Line 23) by checking sequentially at each timestep if a collision between any two robots occurred. The checking terminates when the earliest conflict is detected. This conflict is resolved by creating constraints and computing trajectories with the low-level planner for each constrained robot (Line 33-Line 39).

Db-CBS uses an extension of db-A* that can directly consider dynamic constraints efficiently for the low-level search. If the motions have no conflicts, they are used as an initial guess for the trajectory optimization (Line 25).

A. db-A* with Dynamic Obstacles

In CBS and db-CBS, constraints arise directly from conflicts and represent states to be avoided during the low-level

search, which requires planning with dynamic obstacles. In CBS, the most common approach is to use A* in a space-time search space or SIPP [40]. Inspired by SIPP, in this work db-A* is extended in order to solve single-robot path planning which is consistent with constraints. The extended db-A* is given in Alg. 2. The notation $\mathbf{x} \oplus m$ indicates that the motion m is applied to state $\mathbf{x}$. Recall that a constraint $\langle i, \mathbf{x}_c, k \rangle$ enforces $d(\mathbf{x}_c, \mathbf{x}_k^{(i)}) > \delta$. All constraints are handled during the expansion of neighboring states, only motions that are at least δ away from any constrained state are included (Line 18 - Line 21). In addition, avoiding dynamic obstacles might require to reach a state with a slower motion. Thus, a list of safe arrival time, parent node, and motion to reach this state from the parent are kept (Line 3). When a potentially better path is found, the previous solution motions are kept rather than being removed (Line 32). For the latter, each node keeps a list of safe arrival time, parent nodes to be approached from, and motions that can be used in order to be reached from the goal state (see also Line 3 and Line 32). This is necessary in case one of the dependent children nodes has a dynamic constraint imposed that requires a slower motion. Conceptually, this is similar to SIPP, except that safe arrival intervals are not stored but potential arrival time points, since not all robots are able to wait. Empirically, storing the arrival times compared to always creating new nodes is significantly faster in experiments (Sec. VII-E).

B. Trajectory Optimization

The trajectory optimization is performed in the joint configuration space with:

$$\min_{\{\mathbf{X}^{(i)}\},\{\mathbf{U}^{(i)}\},\Delta t} \sum_{i=1}^{N} \sum_{k=0}^{K-1} J(\mathbf{x}_k^{(i)}, \mathbf{u}_k^{(i)}) \tag{7}$$

$$\text{s.t.} \begin{cases} \begin{bmatrix} \mathbf{x}_{k+1}^{(1)} \\ \mathbf{x}_{k+1}^{(2)} \\ \vdots \\ \mathbf{x}_{k+1}^{(N)} \end{bmatrix} = \begin{bmatrix} \mathbf{x}_k^{(1)} \\ \mathbf{x}_k^{(2)} \\ \vdots \\ \mathbf{x}_k^{(N)} \end{bmatrix} + \begin{bmatrix} \mathbf{f}^{(1)}(\mathbf{x}_k^{(1)}, \mathbf{u}_k^{(1)}) \\ \mathbf{f}^{(2)}(\mathbf{x}_k^{(2)}, \mathbf{u}_k^{(2)}) \\ \vdots \\ \mathbf{f}^{(N)}(\mathbf{x}_k^{(N)}, \mathbf{u}_k^{(N)}) \end{bmatrix} \Delta t \ \ \forall k, \\ \left[\mathbf{u}_k^{(1)}, \mathbf{u}_k^{(2)}, \ldots, \mathbf{u}_k^{(N)}\right]^\top \in \bar{\mathcal{U}} \ \ \forall k, \\ \left[\mathbf{x}_k^{(1)}, \mathbf{x}_k^{(2)}, \ldots, \mathbf{x}_k^{(N)}\right]^\top \in \bar{\mathcal{X}} \ \ \forall k, \\ \mathbf{g}\left(\left[\mathbf{x}_k^{(1)}, \mathbf{x}_k^{(2)}, \ldots, \mathbf{x}_k^{(N)}\right]^\top\right) \geq \mathbf{0} \ \ \forall k, \\ \mathbf{x}_0^{(i)} = \mathbf{x}_s^{(i)}, \quad \mathbf{x}_{K^{(i)}}^{(i)} = \mathbf{x}_f^{(i)} \ \ \forall i. \end{cases}$$

Here, the cost is $J(\mathbf{x}_k^{(i)}, \mathbf{u}_k^{(i)}) = \Delta t + \beta ||\mathbf{u}_k^{(i)}||^2$ with a regularization β, and $\bar{\mathcal{U}} = \times_i \mathcal{U}^{(i)}$, $\bar{\mathcal{X}} = \times_i \mathcal{X}^{(i)}$ are the joint action space and joint state space, respectively. The signed-distance function $\mathbf{g}()$ performs robot-robot and robot-obstacle collision avoidance for each state. Since the ultimate objective is to minimize the sum of arrival times, the goal constraints are added at different timesteps $K^{(i)}$ for each robot based on the discontinuity-bounded initial guess. The optimization variables are state and control actions of all robots, and the length of the time interval Δt. The number of time steps is fixed. The trajectory optimization problem (7) can be solved with nonlinear, constrained optimization.

C. Properties

CBS is a complete and optimal algorithm for MAPF, i.e., if a solution exists it will find it and the first reported solution has the lowest possible cost (sum of costs of all agents). The proof [1] uses two arguments: i) for completeness, it is shown that the search will visit all states that can contain the solution, i.e., no potential solution paths are pruned, and ii) for optimality, it is shown that CBS visits the solutions in increasing order of costs (f-value) and can therefore not have missed a lower-cost solution once it terminates.

The continuous time and continuous space renders the full enumeration proof argument infeasible for MRMP problems. Instead, the probabilistic completeness (PC; the probability of finding a solution if one exists is 1 in the limit over search effort) and asymptotic optimality (AO; the cost difference between the reported solution and an optimal solution approaches zero in the limit over search effort) are considered. It's assumed that there exists a finite $\delta > 0$ such that the trajectory with discontinuity is optimized successfully. It's shown that db-CBS is both probabilistically complete and asymptotically optimal and therefore an anytime planner [25], similar to sampling-based methods such as SST*. The original CBS algorithm never terminates if there is no feasible solution. A common solution is to first compute an upper-bound of the cost using a complete and suboptimal polynomial algorithm and then terminate CBS once this cost is exceeded. Similarilily, in db-CBS we assume that there is a finite, user-defined upper-bound on the cost, which terminates the search if the solution cost exceeds it.

Theorem 1: The db-CBS motion planner in Alg. 1 is asymptotically optimal, i.e.

$$\lim_{n \to \infty} P(\{c_n - c^* > \epsilon\}) = 0, \ \forall \epsilon > 0, \tag{7}$$

where c_n is the cost in iteration n and c^* is the optimal cost.

Theorem 2: The db-CBS motion planner in Alg. 1 is probabilistically complete.

Theorem 3: The db-CBS motion planner in Alg. 1 is anytime, i.e.

$$\lim_{n \to \infty} P(\{c_n \leq c_m\}), \ \forall n < m, \tag{8}$$

where c_n is the cost in iteration n and c_m is the cost in iteration m.

VI. DB-ECBS - ENHANCEMENTS TO DB-CBS

Db-ECBS builds upon db-CBS by introducing the bounded-suboptimality through the use of a suboptimality factor ω. This parameter allows db-ECBS to find a suboptimal solution which is guaranteed to have a cost no more than ω times the cost of the db-CBS. Db-ECBS uses a FOCAL $\mathcal{F}$ priority queue in addition to $\mathcal{O}$. $\mathcal{F}$ prioritizes nodes by the lowest value f_h, the focal heuristic which may be inadmissible. In our case, f_h is equal to the number of robot-robot conflicts encountered in the high-level node. Sec. VI-B explains the focal heuristic function used by our planner.

Changes related to db-ECBS are marked in blue in Alg. 1. Before picking the best node P in the high-level search, $\mathcal{F}$ is updated from $\mathcal{O}$ using `UpdateFocalSet` (Line 19). This function ensures that $\mathcal{F} = \{n | n \in \mathcal{O}, n.cost \leq \omega \cdot LB\}$, where LB is the lower-bound of the best node in $\mathcal{O}$. Db-ECBS uses db-A^*_ω as a low-level planner, which is explained in Sec. VI-A. If the optimization is successful, then additional motions are extracted from the optimization output with `ExtractPrimitives` (Line 30).

In case the optimization fails to return a feasible solution, `AddPrimitives` adds more motion primitives from a pre-computed set of motions (Line 10). In both cases, new iteration is initiated with new settings. After each successful iteration, the upper-bound on the cost is computed using `UpdateUpperBound`(Line 28) for each robot separately and used for the successor iteration by the low-level planner.

A. Discontinuity-Bounded A^*_ω

Discontinuity-bounded A^*_ω (db-A^*_ω) is a bounded-suboptimal variant of the optimal planner db-A*. The algorithm is shown in Alg. 2 with major changes compared to db-A* marked in blue. Figure 2 provides a visual representation. The planner keeps track of nodes to explore using two priority queues - OPEN $\mathcal{O}$ and FOCAL $\mathcal{F}$. $\mathcal{F}$ contains a subset of nodes from $\mathcal{O}$. In each iteration, $\mathcal{F}$ is updated with `UpdateFocalSet` (Line 6). Unlike in the high-level search, this function ensures that $\mathcal{F} = \{n | n \in \mathcal{O}, n.cost \leq \omega \cdot LB$ and $n.cost < ub\}$, where LB is the f-score of the best node in $\mathcal{O}$, and ub is the upper-bound on the cost.

$\mathcal{O}$ and prioritizes nodes based on a heuristic f_h (Line 8). The heuristics we use are described in Sec. VI-B. When the best node is picked from $\mathcal{F}$, the distance to the goal is checked (Line 10). If the distance is small or equal to δ, then the trajectory is returned as a solution. Otherwise, we expand this node with applicable collision-free motion primitives (Line 13-Line 15). Each motion $m \in \mathcal{M}'$ is checked for the collision with a list of constrained states. A motion m is assumed not applicable if $m.\mathbf{x}_k$ is within a δ-distance from the constrained state $c.\mathbf{x}$ at time k (Line 18-Line 21). Also, each $m \in \mathcal{M}'$ is checked for the collision with the neighboring robot trajectories (Line 17). If $n.\mathbf{x} \oplus m$ results in a collision with other robot trajectories, then f_h gets updated. A new state is added to $\mathcal{O}$ if they are not within $(1-\alpha)\delta$ of previously discovered nodes. Otherwise, the previously discovered node gets updates if the new cost-to-come cost or focal heuristic are reduced. (Line 29). The search terminates when a node that's within a δ-distance to the goal state is found.

Algorithm 1: High-Level Search of the db-CBS/db-ECBS

/ Main changes from CBS to db-CBS are in yellow, from db-CBS to db-ECBS are in blue. */*

Input: $\{\mathbf{x}_s^{(i)}\}, \{\mathbf{x}_f^{(i)}\}, \mathcal{W}_{\text{free}}, N, \omega$

Result: $\{\mathbf{X}^{(i)}\}, \{\mathbf{U}^{(i)}\}$

1 $\mathcal{M}^{(r)} = \emptyset \ \forall r \in \{1, \dots, N\}$ ▷ *Initial set of motion primitives*

2 **for** $r \in N$ **do**

3 $\quad \mathcal{M}^{(r)} \leftarrow \mathcal{M}^{(r)} \cup$ `AddPrimitives`(r) ▷ *Add motion primitives for each robot dynamics*

4 $\delta \leftarrow$ `SetDelta()` ▷ *User-defined discontinuity bound*

5 $solved =$ False

6 **for** $n = 1, 2, \dots$ **do**

7 $\quad$ **if** *solved* **then**

8 $\quad\quad \delta \leftarrow$ `UpdateDelta()` ▷ *Update discontinuity bound*

9 $\quad$ **for** $r \in N$ **do**

10 $\quad\quad \mathcal{M}^{(r)} \leftarrow \mathcal{M}^{(r)} \cup$ `AddPrimitives`(r) ▷ *Add more motions for each robot dynamics*

11

12 $\mathcal{S} \leftarrow$ `Node`$(solution : \emptyset, constraints : \emptyset)$ ▷ *Root node*

13 **foreach** *robot* $i \in \mathcal{R}$ **do**

14 $\quad \mathcal{S}.sol[i] \leftarrow$ `db-A*`$_\epsilon (\mathbf{x}_s^{(i)}, \mathbf{x}_f^{(i)}, \mathcal{W}_{\text{free}}, \mathcal{M}^{(i)}, \delta, None, \omega, None)$ ▷ *Single-robot planner with no constraints*

15 $\mathcal{S}.cost \leftarrow$ `GetSolutionCost`$(\mathcal{S}.solution)$

16 $\mathcal{S}.LB \leftarrow$ `GetLowerBound`$(\mathcal{S}.solution)$ ▷ *Compute the lower bound*

$\mathcal{S}.fh \leftarrow$ `HighLevelFocalHeuristic`$(\mathcal{S}.solution)$ ▷ *Compute the focal heuristic-value*

17 $\mathcal{O} \leftarrow \{\mathcal{S}\}$ ▷ *Initialize open priority queue*

18 **while** $|\mathcal{O}| > 0$ **do**

19 $\quad$ `UpdateFocalSet`$(\mathcal{O}, \mathcal{F}, \omega)$ ▷ *Rebuild the focal set*

20 $\quad LB \leftarrow \mathcal{O}.top().LB$ ▷ *Get the lower bound*

21 $\quad P \leftarrow$ `PriorityQueuePop`$(\mathcal{F})$ ▷ *Lowest focal heuristic-value*

22 $\quad \mathcal{O}.Erase(P)$

23 $\quad C \leftarrow$ `GetEarliestConflict`$(P.solution)$ ▷ *Check for collisions between individual motions*

24 $\quad$ **if** $C = \emptyset$ **then**

25 $\quad\quad \{\mathbf{X}^{(i)}\}, \{\mathbf{U}^{(i)}\} \leftarrow$ `Optimization`$(P.solution)$

26 $\quad\quad$ **if** $\{\mathbf{X}^{(i)}\}, \{\mathbf{U}^{(i)}\}$ *successfully computed* **then**

27 $\quad\quad\quad$ `Report`$(\{\mathbf{X}^{(i)}\}, \{\mathbf{U}^{(i)}\})$ ▷ *New solution found*

28 $\quad\quad\quad ub =$ `UpdateUpperBound`$((J(\mathbf{X}, \mathbf{U}, T))$ ▷ *cost bound*

29 $\quad\quad\quad$ **for** $r \in N$ **do**

30 $\quad\quad\quad\quad \mathcal{M}^{(r)} \leftarrow \mathcal{M}^{(r)} \cup$ `ExtractPrimitives`$(\mathbf{X}^{(r)}, \mathbf{U}^{(r)})$

31 $\quad\quad$ **else**

32 $\quad\quad\quad \langle i, j, \mathbf{x}^{(i)}, \mathbf{x}^{(j)}, k \rangle \leftarrow C$ ▷ *Extract conflict*

33 $\quad\quad\quad$ **foreach** $c \in \{i, j\}$ **do**

34 $\quad\quad\quad\quad P' \leftarrow$ `Node`$(solution : P.solution, constraints : P.constraints \cup \{\langle c, \mathbf{x}^{(c)}, k \rangle\})$

35 $\quad\quad\quad\quad P'.solution[c] \leftarrow$ `db-A*`$_\epsilon (\mathbf{x}_s^{(c)}, \mathbf{x}_f^{(c)}, \omega, \mathcal{W}_{\text{free}}, \mathcal{M}^{(c)}, \delta, P'.constraints)$

36 $\quad\quad\quad\quad P'.cost =$ `GetSolutionCost`$(P'.solution)$

37 $\quad\quad\quad\quad P'.LB \leftarrow$ `GetLowerBound`$(P'.solution)$

38 $\quad\quad\quad\quad P'.f_h \leftarrow$ `HighLevelFocalHeuristic`$(P'.solution)$

39 $\quad\quad\quad\quad$ `PriorityQueueInsert`$(\mathcal{O}, P')$

40 $\quad\quad\quad\quad$ **if** $P'.cost \leq LB \cdot \omega$ **then**

41 $\quad\quad\quad\quad\quad$ `PriorityQueueInsert`$(\mathcal{F}, P')$

42

B. Heuristic Functions

The original ECBS designs a focal search with conflict heuristic in order to prioritize nodes with fewer conflicts. In our work, we adapt this focal heuristic to be applicable to motion primitives in order to guide the search on both high and low levels. In general, the focal heuristic is estimated as the sum of conflicts for each timestep:

$$f_h = \sum_{k=0}^{K-1} c^{(k)}, \tag{9}$$

where the computation of $c^{(k)}$ depends on the conflict heuristic function.

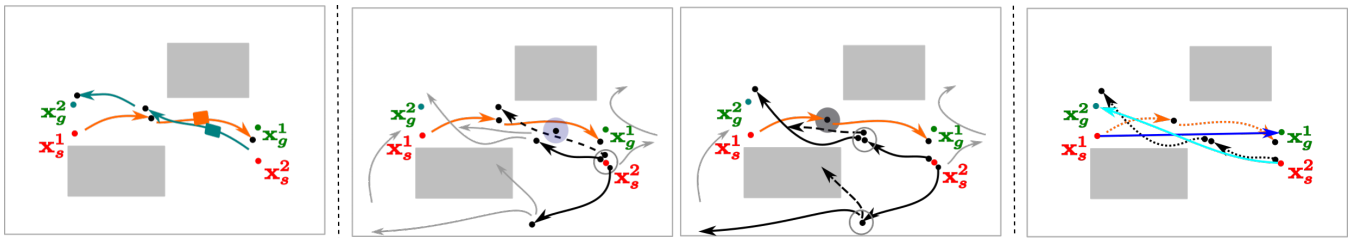


Fig. 2. Visual representation of db-ECBS with db-A^*_ω. Motion primitives are represented as grey edges, while start and goal states are given as $\mathbf{x}_s$ and $\mathbf{x}_g$ respectively. a) A single-robot motion with discontinuous jumps is computed for each robot separately using db-A*. These computed trajectories are checked for inter-robot collisions and for interaction force violations. Once the *earliest* conflict between a single pair of robots *or* an interaction force violation is detected, it results in a constraint for each involved robots. b) This conflict or interaction force violation is resolved by invoking db-A^*_ω with the obtained constraints for certain robots. Motion primitives that start with a discontinuity lower than $\alpha\delta$ (gray circumference), collision-free and at least δ-away from the constrained state (light-blue circumference arose from inter-robot collision, and dark-blue circumference arose from interaction force violation) are considered as applicable for the expansion. The applicable motion primitives are shown with solid black edges. Motions that are in collision with the environment are discarded and not used for the expansion(dashed edge). Motions that cause the interaction force between robots exceed the ψ_{max} are penalized by incrementing the f_h. The search is terminated when a node close to a goal is reached. c) Once the collision-free trajectories obtained they are used as an initial guess for the optimization to repair discontinuities. Time-optimized trajectories are given in blue and cyan respectively.

- *High-level focal heuristic:* All robot trajectories from their start to goal states are checked for collision at each timestep. The number of conflicts for each timestep is 1 iff there is a collision between any pair of robots.

$$c^{(k)} = \begin{cases} 1 & \text{if } d(\mathbf{x}_i^{(k)}, \mathbf{x}_j^{(k)}) < \delta, \quad \exists i \neq j \\ 0 & \text{otherwise.} \end{cases} \tag{10}$$

- *Low-level focal heuristic L_1:* Given trajectories of neighboring robots, the focal heuristic for the i^{th} robot is equal to the sum of all conflicts its current motion has with other robot motions for each timestep. L_1 is slow due to state-by-state collision checking, however more accurate. In practice we notice that for $N > 12$ robot cases it is more preferable to have an accurate low-level focal heuristic, analysis are given in Sec. VII-E.

$$c^{(k)} = \sum_{i=1}^{N} \sum_{i \neq j, j=1}^{N} \begin{cases} 1 & \text{if } d(\mathbf{x}_i^{(k)}, \mathbf{x}_j^{(k)}) < \delta, \\ 0 & \text{otherwise.} \end{cases} \tag{11}$$

- *Low-level focal heuristic L_2:* Given the trajectories of neighboring robots, the number of conflicts is 1 iff there is a collision between the current motion of the i^{th} robot and any number of robots similar to (10). L_2 is less accurate, however, fast. Comparison details are in Sec. VII-E.

C. Interaction-Aware Planning

Interaction-awareness is an additional feature which is applicable to both the db-CBS and db-ECBS.

We denote the aerodynamic interaction force on the i^{th} robot with $\boldsymbol{\psi}^{(i)}(\cdot)$, which alters the robot dynamics as given in (1). Augmenting the state of the robot changes the definition of a motion primitive. A single motion primitive is a tuple $\langle \mathbf{X}, \mathbf{U}, K \rangle$, consisting of state sequences $\mathbf{X} = \langle \mathbf{x}_0, ..., \mathbf{x}_K \rangle$, control sequences $\mathbf{U} = \langle \mathbf{u}_0, ..., \mathbf{u}_{K-1} \rangle$, and interaction force sequences $\boldsymbol{\Psi} = \langle \boldsymbol{\psi}(\mathbf{r}_0), \ldots, \boldsymbol{\psi}(\mathbf{r}_K) \rangle$, which respect the dynamics given in (2). Those motions primitives are connected in a graph search with bounded discontinuities, see Fig. 2.

The distance between states $\mathbf{x}_m$ and $\mathbf{x}_n$ is computed as $d(\mathbf{x}_m, \mathbf{x}_n) + d'(\boldsymbol{\psi}(\mathbf{r}_m), \boldsymbol{\psi}(\mathbf{r}_n))$, where d' is a *metric* $d' : \mathcal{X} \times \mathcal{X} \to \mathbb{R}^3 \times \mathbb{R}^3$ which measures the difference between residual forces. Thus, Definition 1 becomes

Definition 2: Sequences $\mathbf{X} = \langle \mathbf{x}_0, \ldots, \mathbf{x}_K \rangle$, $\mathbf{U} = \langle \mathbf{u}_0, \ldots, \mathbf{u}_{K-1} \rangle$, $\boldsymbol{\Psi} = \langle \boldsymbol{\psi}_0, \ldots, \boldsymbol{\psi}_K \rangle$ are *δ-discontinuity-bounded* solutions iff the following conditions hold:

$$\begin{aligned} d(\mathbf{x}_{k+1}, \text{step}(\mathbf{x}_k, \mathbf{u}_k, \boldsymbol{\psi}(\mathbf{r}_k))) \leq \delta, \forall k, \\ \mathbf{u}_k \in \mathcal{U}, \quad \mathbf{x}_k \in \mathcal{X}, \forall k, \\ \|\boldsymbol{\psi}(\mathbf{r}_k)\| \leq \psi_{max}, \quad \mathcal{B}(\mathbf{x}_k) \in \mathcal{W}_{\text{free}}, \forall k, \\ d(\mathbf{x}_0, \mathbf{x}_s) + d'(\boldsymbol{\psi}(\mathbf{r}_0), \boldsymbol{\psi}(\mathbf{r}_s)) \leq \delta, \\ d(\mathbf{x}_K, \mathbf{x}_f) + d'(\boldsymbol{\psi}(\mathbf{r}_K), \boldsymbol{\psi}(\mathbf{r}_f)) \leq \delta. \end{aligned} \tag{12}$$

Definition 2 enforces the start and goal states to be connected with an error of δ in dynamics, and bounds the interaction force to stay within the limit. We use a neural networks to compute the interaction forces between robots, details are given in Sec. VII-C.

Algorithmic Changes:

At the *low-level search*, only the focal heuristic estimation is adapted to take into account the aerodynamic force between robot pairs that have a relative position *less* than a `threshold`. For example, while planning for a robot i, the focal heuristic f_h at time k gets incremented if the collision constraint *or* $\|\boldsymbol{\psi}^{(i)}(\mathbf{r}(i))\| \leq \psi^{(i)}_{max}$ is violated. The value of `threshold` depends on the robot dynamics and geometric shapes.

At the *high-level search* the constraints derive not only from inter-robot collision, but additionally from aerodynamic interaction force violations. For example, while checking for inter-robot collisions between robots i and j at timestamp k, the constraint arise either from inter-robot collisions, *or* if one of the $\boldsymbol{\psi}^{(i)}(\mathbf{r}^{(i)}), \boldsymbol{\psi}^{(j)}(\mathbf{r}^{(j)})$ exceeds the maximum endurable limit ψ_{max}. Intuitively, changes made to the

Algorithm 2: db-A^*/db-A^*_ω

/ Main changes compared to A* are in yellow, compared to db-A* are in blue. */*

Input: $\mathbf{x}_s, \mathbf{x}_f, \mathcal{W}_{\text{free}}, \mathcal{M}, \delta, \mathcal{C}, \omega, ub, \{\mathbf{X}_{tmp}\}$
Result: $\mathbf{X}, \mathbf{U}, K$ or Infeasible

1 $\mathcal{T}_m \leftarrow$ `NearestNeighborInit`$(\mathcal{M})$ ▷ *Use start states of motions (excl. position)*
2 $\mathcal{T}_n \leftarrow$ `NearestNeighborInit`$(\{\mathbf{x}_s\})$ ▷ *capture explored vertices (incl. position)*
3 $\mathcal{O} \leftarrow \{$`Node`$(\mathbf{x}: \mathbf{x}_s, g: 0, h: h(\mathbf{x}_s), f_h: 0, A: \{(g: 0, p: None, a: None, f_h: 0)\})\}$
4 $\mathcal{F} \leftarrow \{$`Node`$(\mathbf{x}: \mathbf{x}_s, g: 0, h: h(\mathbf{x}_s), f_h: 0, A: \{(g: 0, p: None, a: None, f_h: 0)\})\}$ ▷ *Initialize the open and focal priority queues*
5 **while** $|\mathcal{O}| > 0$ **do**
6 `UpdateFocalSet`$(\mathcal{O}, \mathcal{F}, \omega, ub)$ ▷ *Rebuild the focal set*
7 $LB \leftarrow \mathcal{O}.top().f$ ▷ *Get the lower bound*
8 $n \leftarrow$ `PriorityQueuePop`$(\mathcal{F})$ ▷ *Lowest focal heuristic-value*
9 $\mathcal{O}.Erase(n)$
10 **if** $d(n.\mathbf{x}, \mathbf{x}_f) \leq \delta$ **then**
11 **return** $\mathbf{X}, \mathbf{U}, K$ ▷ *Traceback solution*
▷ *Find applicable motion primitives with discontinuity up to* $\alpha\delta$
12 $\mathcal{M}' \leftarrow$ `NearestNeighborQuery`$(\mathcal{T}_m, n.\mathbf{x}, \alpha\delta)$
13 **foreach** $m \in \mathcal{M}'$ **do**
14 **if** $n.\mathbf{x} \oplus m \notin \mathcal{W}_{\text{free}}$ **then**
15 **continue** ▷ *entire motion is not collision-free*
16 $g_t \leftarrow n.g + cost(m)$ ▷ *tentative g-score for this action*
17 $f_{ht} \leftarrow n.f_h +$ `LowLevelFocalHeuristic`$(m, \{\mathbf{X}_{tmp}\})$ ▷ *tentative focal heuristic-value for this action*
18 **foreach** $c \in \mathcal{C}$ **do**
19 $\mathbf{x}' = m[\lfloor \frac{c.k - n.g}{\Delta t} \rfloor]$ ▷ *Motion primitive state for checking*
20 **if** $d(\mathbf{x}', c.\mathbf{x}) \leq \delta$ **then**
21 **continue** loop Line 13 ▷ *m does not obey all constraints*
▷ *find already explored nodes within* $(1-\alpha)\delta$
22 $\mathcal{N}' \leftarrow$ `NearestNeighborQuery`$(\mathcal{T}_n, n.\mathbf{x} \oplus m, (1-\alpha)\delta)$
23 **if** $\mathcal{N}' = \emptyset$ **then**
24 `PriorityQueueInsert`$(\mathcal{O}, $`Node`$(\mathbf{x}: n.\mathbf{x} \oplus m, g: g_t, h: h(n.\mathbf{x} \oplus m), f_h: f_{ht}, A: \{(g: g_t, p: n, a: m, f_h: f_{ht})\})$
25 `NearestNeighborAdd`$(\mathcal{T}_n, n.\mathbf{x} \oplus m)$
26 **else**
27 **foreach** $n' \in \mathcal{N}'$ **do**
28 **if** $g_t < n'.g \mid f_{ht} < n'.f_h$ ▷ *Motion is better than a known motion* **then**
29 $n'.g = g_t$ ▷ *Update cost*
30 $n'.f_h = f_{ht}$ ▷ *Update focal heuristic*
31 $n'.A = n'.A \cup \{(g: g_t, p: n, a: m, f_h: f_{ht})\}$ ▷ *Add parent*
32 `PriorityQueueUpdate`$(\mathcal{O}, n')$
33 **if** $n'.f \leq LB \cdot \omega$ **then**
34 `PriorityQueueInsert`$(\mathcal{F}, n')$ ▷ *Add to focal set*
35
36
37 **return** Infeasible

low-level and high-level search makes safe motions more preferable to extend to find a solution.

For the *trajectory optimization* part we consider the residual force $\psi(\cdot)$ as part of the robot's state, thus it's value is constrained to stay within a pre-defined bound.

D. Properties

ECBS is a complete and bounded suboptimal algorithm for MAPF, i.e., if a solution exists it will find it and the first reported solution has a cost that is at most ω times the lowest possible cost (sum of costs of all agents). The proof [2] uses the same argument as CBS, and keeps track of the lower bound of the possible solution cost in order to guarantee bounded suboptimality. As in the db-CBS case, this argument does not directly transfer to the continuous time and continuous space case, because enumerating all possible solutions is not possible. However, we can reason about probablistic completeness and asymptotic bounded suboptimality, as in sampling-based planners [16].

1) Completeness and Anytime Behavior: In db-ECBS in continuous domain, we assume that there is a finite, user-defined value for an upper-bound cost. Thus, the planner terminates search it the solution cost exceeds the given upper-bound on cost.

Db-ECBS never expands a node with cost higher than the upper bound on cost, which is computed for each robot separately. Upper-bound on the cost for each individual robot ensures that the solution returned for each robot stays within the bound. Upper-bound on the cost for each single-robot is computed with:

$$ub_i = c_i - (h - h_i), \tag{13}$$

where c_i is the cost in iteration i, h_i is an admissible cost to reach the goal from start state for the robot i, and h is the sum of all robots.

We assume that there exists a finite $\delta > 0$ such that the trajectory with discontinuity is optimized successfully.

Theorem 4: The db-ECBS motion planner in Alg. 1 is asymptotically bounded suboptimal, i.e.

$$\lim_{n\to\infty} P(\{c_n - \omega c^* > \epsilon\}) = 0, \ \forall \epsilon > 0, \tag{14}$$

where c_n is the cost in iteration n, c^* is the optimal cost, and $\omega \geq 1$ is a user-provided suboptimality bound.

Proof: Each iteration in Line 6 operates on a discrete search problem over the graph implicitly defined by the finite set of motion primitives. A bounded suboptimal solution is found, if one exists, within this discretization by [2, p. 6]. Subsequent iterations add more motion primitives and reduce δ as in [4]; therefore, a larger discrete search graph is produced, for which the bounded suboptimality proof of ECBS still holds. In the limit ($|\mathcal{M}| \to \infty$, $\delta \to 0$), db-ECBS will report the optimal solution. This argument is the same as the detailed version in [41, Theorem 1 and 2], for combining PRM and CBS. A formal version of this proof, including bounding the actual probability, is given in [4, Theorem 2]. ■

Theorem 5: The db-ECBS motion planner in Alg. 1 is probabilistically complete.

Proof: This directly follows from Theorem 4, since asymptotic optimality implies probabilistic completeness. ■

Theorem 6: The db-ECBS motion planner in Alg. 1 is anytime, i.e.

$$\lim_{n\to\infty} P(\{c_n \leq c_m\}), \ \forall n < m, \tag{15}$$

where c_n is the cost in iteration n and c_m is the cost in iteration m.

Proof: From Theorem 5, an initial feasible solution is found in finite time, if one exists. Once an initial solution is found, the upper-bound is set for each robot individually and

the lowest total cost is computed. At the high-level by adding more primitives at each iteration Line 6 the planner accepts a solution if the total cost is decreased than the lowest total cost. Since the lowest cost solution is saved, the algorithm can stop at any time. ■

This result compares to the baseline algorithms as follows. SST* is probabilistically complete and asymptotically near-optimal, where the near-optimality stems from a fixed hyper-parameter similar to the time-varying δ in db-ECBS. K-CBS is probabilistically complete and could achieve asymptotic optimality by using an incremental cost bound similar to AO-x [42]. S2M2 is neither probabilistically complete nor asymptotically optimal, because it uses pessimistic approximations and priority-based search. Concrete examples in which S2M2 fails to find a solution are shown in Sec. VII.

Db-ECBS can naturally plan for heterogeneous teams of robots, where individual team members may have unique dynamics or collision shapes. This does not require any algorithmic changes, as the low-level planner can use different motion primitives and constraints are defined for each robot themselves, and conflicts can be detected using arbitrary collision shapes. On the optimization side, nonlinear optimization and stacked states which naturally extends to varied state and action dimensions.

VII. Results

We evaluate db-ECBS on 65 problems with six different robot dynamics in various environments. Some of the problems and dynamics are selected from previous works [25], [43].

A. Dynamic Systems and Environments

1) **Unicycle (1^{st} order)** with 3-dimensional state space $[x, y, \theta] \in SE(2)$ and a 2-dimensional $[v, \omega] \in \mathcal{U} \subset \mathbb{R}^2$ control space with dynamics defined in [44, Eq. (13.18)]. Bounds used in this work are $v \in [-0.5, 0.5]$ m/s and $\omega \in [-0.5, 0.5]$ rad/s.
2) **Unicycle (S2M2)** is a unicycle 1^{st} order with a spherical collision shape and higher angular velocity bounds of $\omega \in [-2, 2]$ rad/s.
3) **Unicycle (2^{nd} order)** with 5-dimensional state space $[x, y, \theta, v, \omega] \in \mathcal{X} \subset \mathbb{R}^5$, a 2-dimensional $[\dot{v}, \dot{\omega}] \in \mathcal{U} \subset \mathbb{R}^2$ control space, and dynamics defined in [44, Eq. (13.46)]. We use $v \in [-0.5, 0.5]$ m/s, $\omega \in [-0.5, 0.5]$ rad/s, $\dot{v} \in [-0.25, 0.25]$ m/s^2, and $\dot{\omega} \in [-0.25, 0.25]$ rad/s^2.
4) **Double integrator 2D** with 4-dimensional state space $[x, y, v_x, v_y] \in \mathcal{X} \subset \mathbb{R}^4$, a 2-dimensional $[a_x, a_y] \in \mathcal{U} \subset \mathbb{R}^2$ control space with dynamics given in [44, (Eq. (13.35))]. We use $v \in [-0.5, 0.5]$ m/s, $a \in [-2.0, 2.0]$ m/s^2, where these bounds imply per element.
5) **Double integrator 3D** with 6-dimensional state space $[x, y, z, v_x, v_y, v_z] \in \mathcal{X} \subset \mathbb{R}^6$, a 3-dimensional $[a_x, a_y, a_y] \in \mathcal{U} \subset \mathbb{R}^3$ control space. The dynamics are similar to [44, (Eq. (13.35))], but in 3D. We use $v \in [-0.5, 0.5]$ m/s, $a \in [-2.0, 2.0]$ m/s^2.
6) **Car with trailer** with 4-dimensional state space $[x, y, \theta_0, \theta_1] \in \mathcal{X} \subset \mathbb{R}^4$, a 2-dimensional $[v, \phi] \in \mathcal{U} \subset \mathbb{R}^2$ control space, and dynamics shown in [44, Eq. (13.19), Fig. 13.6]. We have an additional constraint $|\angle(\theta_0, \theta_1)| < \pi/4$ in order to keep the angle between the car and trailer below the threshold. We use $v \in [-0.1, 0.5]$ m/s, $\phi \in [-\pi/3, \pi/3]$, $l = 0.25$ m, and $d_1 = 0.5$ m, where l and d_1 are given in in [44].

For testing scenarios with no interaction-awareness we focus on cases where the workspace has small dimensions, thus it is challenging for robots to find collision-free motions in a time-optimal way. Some problem instances are created manually to test canonical cases from the literature. Moreover, we create additional environments with random obstacles, robot types, and start/goal states. In these environments, we compare our method with K-CBS [24], S2M2 [23], joint space SST* [16] and db-CBS [25].

For testing the interaction-awareness of our planner we consider scenarios where flying robots have to fly in close-proximity to each other in order to reach their goals. In these environments we compare two version of our method with MAPF/C+POST [3].

We analyze success rate (p), computational time until the first solution is found (t^{st}), and cost of the first solution (J^{st}), and cost of the final solution (J^{f}) for instances with anytime property enabled. Note, the cost is a time, which is equal to the sum of control duration over the all single-robot path. An instance is not solved successfully if an incorrect solution is returned, or no solution is found after the time limit. To compare relative improvements over baselines, we use a notion of regret, e.g., $r^{\text{s2m2}} = (J^{\text{s2m2}} - J^{\text{db-ecbs}})/J^{\text{s2m2}}$.

B. Algorithms

We use OMPL [45] and FCL [46] for SST*, and the latter for K-CBS, db-CBS and db-ECBS. For fair comparison, we set the low-level planner of K-CBS to SST*, and modify the collision checking to support robots of any shape. For S2M2, we use the Euler integration to propagate the robot state and bound control actions. For K-CBS and S2M2 we use the respective publicly available implementations from the authors. For K-CBS, we disable merging. When comparing with S2M2, we use a spherical first order unicycle model and quadrupled angular velocity bounds $\omega \in [-2, 2]$ rad/s, since the provided code was unable to solve most of our very dense planning problems otherwise. Moreover, there is a trade-off in setting a parameter for bloated obstacles; a small gain leads to collisions with the environment, while a large gain results in infeasible solutions. We carefully tuned this and other hyperparameters to maximize the success rate of all algorithms. For MAPF/C+POST we use the publicly available MATLAB implementation. For db-CBS and **db-ECBS** Algs. 1 and 2 are implemented in C++, and the benchmarking script is written in Python. For optimization, we use Dynoplan [47], which uses the differential dynamic programming (DDP) solver Crocoddyl [48]. We use a workstation (AMD Ryzen Threadripper PRO 5975WX @ 3.6 GHz, 128 GB RAM, Ubuntu 22.04).

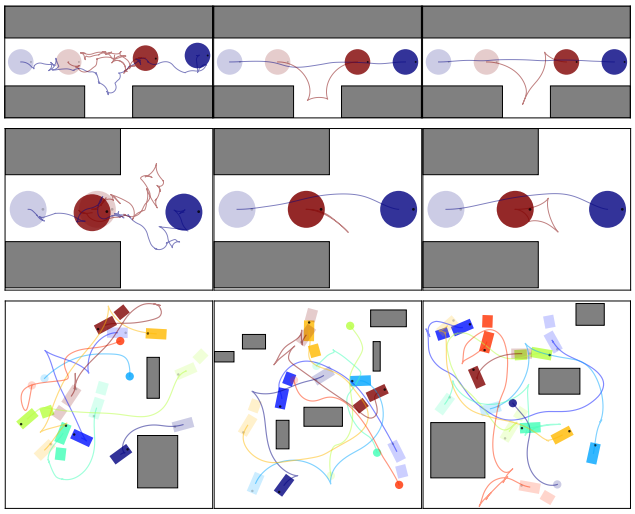


Fig. 3. Top row: SST*, db-CBS, db-ECBS with *alcove* (left to right). Middle row: SST*, db-CBS, db-ECBS with *at goal*. Last row shows *random hetero* with random start and goal states for $N = 8$ where only db-ECBS is able to return a solution.

C. Benchmarking

Non-Interaction Aware Planning:

We compare algorithms on problems from [25]. An instance is not solved if if an incorrect solution is returned, or no solution is found after 5 min. First, we consider canonical examples often seen in the literature, see rows 1–3 in Table II and Fig. 3.

- *Swap:* the environment has no obstacles and two robots have to swap their positions, i.e, each robot's start position is the goal position of the other robot. All planners except SST* have a $100\,\%$ success rate. Although S2M2 is the fastest to find a solution within 0.1 s, the solution cost of db-CBS has the lowest cost of 13.3 s.
- *Alcove:* the environment requires one robot to move into an alcove temporarily in order to let the second robot pass. K-CBS fails for all runs, while S2M2, db-CBS and db-ECBS remain consistent to find a solution. The time to generate the solution for db-ECBS is 4.3 s, which is around five times faster than of db-CBS.
- *At Goal:* similar to *Alcove*, but one robot is already at its goal state and needs to move away temporarily to let the second robot pass. S2M2 and K-CBS fails with this instance. Although, SST* is the fastest to find a solution within 1.2 s, the solution cost is as twice large as of db-CBS, which has the lowest cost of 15.4 s.

Both *alcove* and *at goal* should be solvable by K-CBS for finite merge bounds (a feature that was not readily available in the provided code). However, in this case the algorithm essentially becomes joint space SST* and would produce solutions with a high cost as seen in Table II.

We validate the planners scalability and effectiveness in heterogeneous systems with the following problem instances:

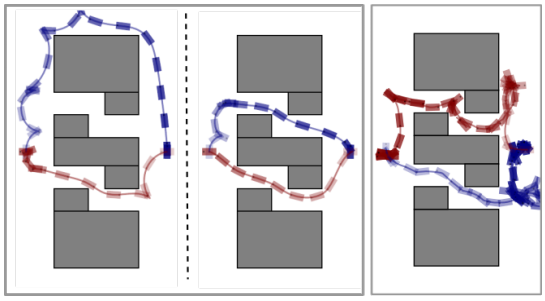
- *Robot Number:* obstacles, start and goal states for up to eight robots are generated randomly, see rows 4–6 in Table II. Here, we assume homogeneous robots. Both S2M2 and K-CBS handle $N = 4$ cases relatively well, with K-CBS showing better performance with a success rate of $70\,\%$. Only db-CBS and db-ECBS finds a solution for $N = 8$ problems, however the success rate of db-CBS is $10\,\%$.
- *Heterogeneous Systems* obstacles, start and goal states for a team of up to eight heterogeneous robots are generated randomly, see rows 7–9 in Table II. The results of SST* and K-CBS remain similar to homogeneous systems test. The success rate of db-CBS is 50% for $N = 8$ scenarios, while db-ECBS has 100% success rate. S2M2 is not tested, since it does not support all robot dynamics considered here.

Note that in these random instances the shown median of J is not directly comparable unless the success rate $p = 1$. In the table the positive regret r for all baselines shows that db-CBS consistently computes lower-cost solutions.

Note that all problem instances consider 5×5 environments, which is significantly smaller compared to scenarios S2M2 and K-CBS were previously validated on.

We conclude that for problem instances without interaction forces, the performance of db-CBS outperforms db-ECBS with respect to computation time and with respect to the solution cost. However, as the number of robots increases, db-CBS' success rate drops, as can be seen in Table II, while db-ECBS consistently returns a solution within our time limit.

We test our planner db-ECBS with SST* on the *Kink* example to show the anytime property. As Fig. 4 shows, db-ECBS improves initial solution by decreasing the cost of 60.9 s to 45.2 s within less than 2 s. The final solution passes through corridors to reach goal states. On the other hand, SST* is able to find paths through corridors from an initial trial, but with cost of 90.7 s and fails to improve it within the time limit of 2 min.



a) db-ECBS b) SST*

Fig. 4. *Kink* example with two unicycle robots (1^{st} order). a) Our planner db-ECBS finds the first solution at $t = 7.3$ s, and computes better solution at $t = 8.28$ s. b) SST* finds a solution at $t = 11.93$ s, and does not improve it within the time limit of 120 s.

Interaction Aware Planning: We validate the interaction-reasoning property of db-ECBS by comparing it with a conservative method, which assumes to have an ellipsoid shape around each robot, MAPF/C+POST [43] and sampling-based interaction-aware kinodynamic motion planner from [37].

We deploy *deep sets* [49] to approximate the value of the residual force $\boldsymbol{\psi}(\cdot)$.

$$\boldsymbol{\psi}^{(i)}(\mathbf{r}^{(i)}) \approx \boldsymbol{\rho}\left(\sum_{x^{(ij)} \in r^{(i)}} \boldsymbol{\eta}\left(x^{(ij)}\right)\right) := \hat{\boldsymbol{\psi}}^{(i)}. \quad (16)$$

Here, $\boldsymbol{\eta}(\cdot)$ and $\boldsymbol{\rho}(\cdot)$ are two DNNs, where the former predicts the "contributions" from each neighbor, while the latter returns the total effect. The details on the network architecture is given in [37].

Example 2: Consider a heterogeneous system of small, small and large robots. Then, we have:

$$\boldsymbol{\psi}^{(3)} \approx \boldsymbol{\rho}_{large}\left(\boldsymbol{\eta}_{small}\left(x^{(31)}\right) + \boldsymbol{\eta}_{small}\left(x^{(32)}\right)\right). \quad (17)$$

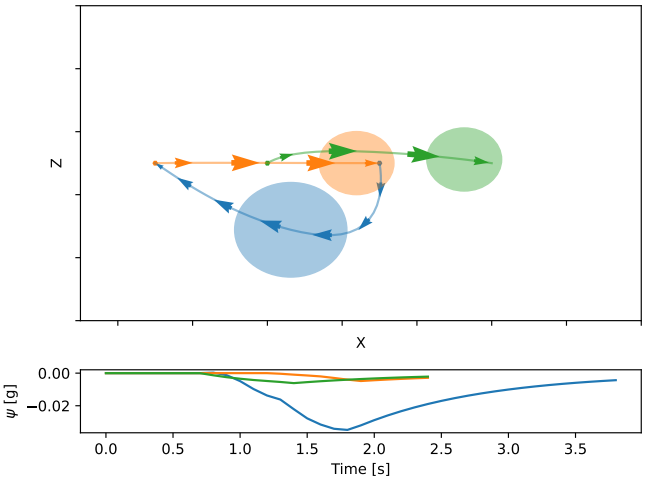


Fig. 5. *Swap-hetero* example with two small robots (orange and green), and one large robot (blue). Top row: robot trajectories in 2D xz-plane with arrows representing velocity every 0.5 s, and circles defining robot collision geometric shapes. Our planner db-ECBS finds a solution where large robot flies under small robots. This configuration results in lower solution cost and stays within the interaction force limit. Bottom row: interaction force of each robot over time.

We consider three scenarios of different difficulty levels. Here, db-ECBS-C stands for db-ECBS with conservative assumption of having an ellipsoid shape around each robot to handle the interaction force during planning. The ellipsoid has a radii of $\mathbf{e} = (e_x, e_y, e_z)$, where $0 < e_x = e_y \leq e_z$. From [43], $e_x = e_y = 0.12\text{m}$, and $e_z = 0.3\text{m}$ is determined to be a safe vertical distance. Db-ECBS-R stands for db-ECBS using deep sets to capture residual forces between robots.

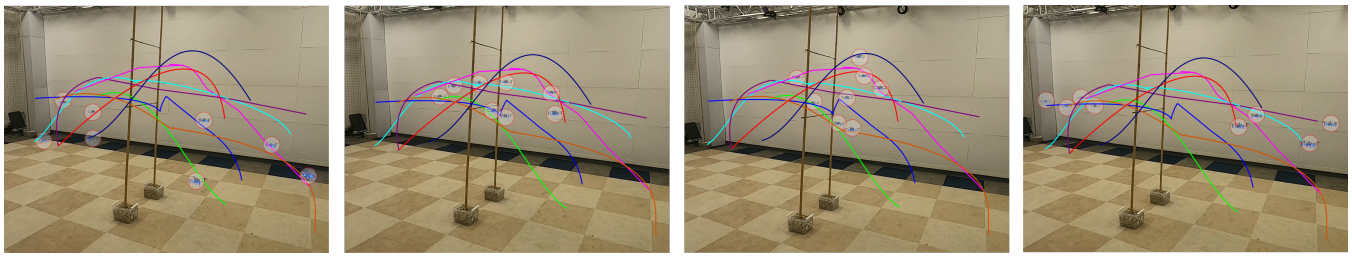
- *Wall:* the environment has relatively large dimensions $(7.5 \times 6.5 \times 2.5\,\text{m}^3)$. Robots are required to pass to the other side of the wall by passing through three small-sized windows, see Fig. 1. An instance is not solved if an incorrect solution is returned, or no solution is found after 2.5 h. For all instances MAPF/C+POST is the fastest to find a solution, but with the cost twice as large as db-ECBS-C and db-ECBS-R. Db-ECBS-C finds a solution faster compared to the db-ECBS-R with the lower solution cost for almost all instances, see rows 1–6 Table III.
- *Window:* it has a small dimensions of $2 \times 4 \times 2\,\text{m}^3$. The difficulty comes with the challenge of having many robots on both sides of the window, where each team is supposed to pass through a single window of 0.5×0.5 m^2 dimensions, see Fig. 1, top row. This scenario requires robots to operate in close-proximity to each other. The time limit for this instance is 10 min. The performance of db-ECBS-R remain consistent to find a solution with 100% success rate compared to the conservative variant with better solution cost, see rows 7–8 Table III. The failure of db-ECBS-C might be due to the small size of the environment, where robots have to move in close-proximity to each other. Because of an ellipsoid collision shape of robots, the inter-robot collision occurs more in this environment, so the planner needs more time to resolve all conflicts.
- *Swap-hetero:* three-robot swapping problem, see Fig. 5. The robots are small, small and large, where they differ with geometric shapes. The environment is of $2 \times 0.4 \times 1\,\text{m}^3$ dimensions. The narrow size of the $y-$axis enforces robots to stay vertically aligned. However, this formation can be dangerous or expensive if the large robot's trajectory goes over the small ones. This is due to the huge aerodynamic interaction force on small ones coming from the large one. The time limit for planners is 2 min. Our db-ECBS finds the first solution within 0.30 s with cost 13.4 s. Then improves it to the cost of 8.6 s at 1.31 s. The baseline from [37] was reported with a similar cost for this problem instance, but fails to find a solution within our time limit.

We conclude that for problem instances with interaction forces, the conservative assumption of having an ellipsoidal collision shape for each robot is efficient if the environment has large dimensions, thus the interaction force between robots always remain within an endurable limit. However, this method fails in scenarios where the environment has small dimensions, which require robots to fly in close-proximity. In the latter case, the planner relies on an accurate residual force estimation using the NN.

D. Physical Robots

The real-world experiments are conducted inside a $7 \times 4 \times 2.75\,\text{m}^3$ room equipped with a motion capture system with twelve Optitrack cameras. We use Bitcraze Crazyflie 2.1 drones for flying robots and control them using Crazyswarm2 [50]. For ground robots we use Polulu 3pi+ 2040 differential-drive robots.

Two scenarios are considered. We first test the *Window* example with $N = 8$ homogeneous flying robots. The flying robots are modeled with 3D double integrator dynamics. Trajectories are obtained with interaction-aware db-ECBS, see Fig. 6. The second scenario has $N = 4$ ground robots and $N = 4$ flying robots. Heterogeneous team of robots are requested to swap their positions. We attach a 60cm bamboo



a) t=1.08s b) t=2.0s c) t=4.06s d) t=5.1s

Fig. 6. Trajectories over time for eight flying robots with *Window* example.

TABLE II

CANONICAL EXAMPLES, SCALABILITY WITH VARYING ROBOT NUMBERS, AND HETEROGENEOUS SYSTEMS. MEDIAN VALUES OVER 10 TRIALS PER ROW. BOLD ENTRIES ARE THE BEST FOR THE ROW, —NO SOLUTION FOUND, ⋆ NOT TESTED.

#	Instance	SST*				S2M2				k-CBS				db-CBS				db-ECBS		
		p	$t[s]$	$J[s]$	$r[\%]$	p	$t[s]$	$J[s]$	$r[\%]$	p	$t[s]$	$J[s]$	$r[\%]$	p	$t[s]$	$J[s]$	$r[\%]$	p	$t[s]$	$J[s]$
1	swap	0.5	5.3	29.2	55	**1.0**	**0.1**	16.0	7	**1.0**	13.2	29.7	45	**1.0**	0.8	**13.3**	**-12**	**1.0**	4.3	14.9
2	alcove	0.6	5.2	47.2	34	**1.0**	**0.4**	29.2	12	0.0	—	—	—	**1.0**	21.4	**23.9**	**-7**	**1.0**	4.3	25.6
3	at goal	0.7	**1.2**	35.3	33	0.0	—	—	—	0.0	—	—	—	**1.0**	3.1	**15.4**	**-54**	**1.0**	3.5	23.7
4	rand (N=2)	0.1	16.3	40.8	43	0.8	**0.1**	**28.8**	**-26**	**1.0**	2.0	71.3	50	**1.0**	1.7	30.2	-5	**1.0**	10.8	34.1
5	rand (N=4)	0.0	—	—	—	0.3	**0.5**	**52.0**	**-11**	0.7	36.5	141.7	47	**1.0**	6.0	65.5	-10	**1.0**	23.3	72.7
6	rand (N=8)	0.0	—	—	—	0.0	—	—	—	0.0	—	—	—	0.1	**56.5**	**142.3**	**-19**	**0.9**	60.2	164.4
7	rand het (N=2)	0.2	6.5	37.0	55	⋆	⋆	⋆	⋆	0.9	4.0	35.4	54	**1.0**	**0.9**	18.4	**-3**	**1.0**	3.7	**18.0**
8	rand het (N=4)	0.0	—	—	—	⋆	⋆	⋆	⋆	0.8	46.4	80.6	42	**0.9**	**4.1**	**44.2**	**-24**	**0.9**	8.5	50.0
9	rand het (N=8)	0.0	—	—	—	⋆	⋆	⋆	⋆	0.0	116.5	155.1	17	0.5	**17.9**	**90.3**	**-11**	**1.0**	23.7	118.8

TABLE III

Wall AND *Window* EXAMPLES. MEDIAN VALUES OVER 5 TRIALS PER ROW. BOLD ENTRIES ARE THE BEST FOR THE ROW, ⋆ NOT TESTED.

#	Instance	MAPF/C+POST				db-ECBS-C				db-ECBS-R			
		p	$t^{\mathrm{st}}[s]$	$J^{\mathrm{st}}[s]$	$J^{f}[s]$	p	$t^{\mathrm{st}}[s]$	$J^{\mathrm{st}}[s]$	$J^{f}[s]$	p	$t^{\mathrm{st}}[s]$	$J^{\mathrm{st}}[s]$	$J^{f}[s]$
1	wall2	**1.0**	**1.0**	60.7	60.7	**1.0**	84.9	**28.2**	**26.4**	**1.0**	101.3	32.7	26.6
2	wall4	**1.0**	**1.6**	147.3	147.3	**1.0**	186.3	**63.1**	58.0	**1.0**	227.4	71.6	**56.7**
3	wall8	**1.0**	**3.2**	297.1	297.1	**1.0**	428.3	**136.5**	**128.8**	**1.0**	553.6	161.3	133.2
4	wall10	**1.0**	**4.0**	373.0	373.0	**1.0**	706.7	**165.7**	**165.6**	**1.0**	1373.4	195.6	181.1
5	wall12	**1.0**	**4.9**	439.8	439.8	**1.0**	874.8	**210.8**	**201.8**	**1.0**	1054.3	248.6	211.2
6	wall16	**1.0**	**7.7**	657.4	657.4	**1.0**	1704.6	**286.9**	**276.7**	**1.0**	3411.0	329.7	290.7
7	window8	⋆	⋆	⋆	⋆	0.2	434.9	72.1	72.1	**1.0**	**132.8**	**66.4**	**66.4**
8	window10	⋆	⋆	⋆	⋆	0.4	441.7	117.5	117.5	**1.0**	**336.6**	**99.9**	**99.9**

stick to each ground robots to increase the complexity, see Fig. 2. Experiments with physical robots are available in the supplemental video.

E. Ablation Studies of db-ECBS

We analyze some key components in the algorithm to study their impact on the overall performance of the db-ECBS planner.

1) Analysis of Adding New States in db-A^*_ω: We first evaluate two ways of adding new states to the search during the low-level planning as discussed in Sec. V-A: *(i)* adding all nodes that has been explored during the expansion (*always-add nodes*), *(ii)* allow update an already explored node if a better path is found (*rewiring*). We compare and discuss how the choice of adding nodes affects the computation time and overall solution cost of db-ECBS. A summary of the results is shown in Fig. 7. The computation time remain quite the same for up to $N = 4$ case, while the difference between them starts to be noticeably high with $N > 8$ cases. This can be because with *always-add nodes* the number of nodes increases in general, which leads to have many nodes with identical cost-to-come (g) value. While with *rewiring* the number of nodes to process remains low, thus the number of expanded nodes to reach the goal state remain low. The solution cost between two methods remain similar for $N \leq 4$ similar, and diverges significantly with more robot scenarios.
2) Analysis of Focal Heuristics: We discuss two focal heuristic functions from Sec. VI-B that is used by the low-level planner db-A^*_ω: *(i)* L_1 - state-by-state collision checking of all possible robot pairs at each timestep during the single-robot path planning, *(ii)* L_2 - single collision checking per timestep combining all robot states. L_2 is computationally efficient for problem instances with $N \leq 12$, as shown in Fig. 8, this is because inter-robot collision occur with less number of robots, so less accurate estimation of collisions is

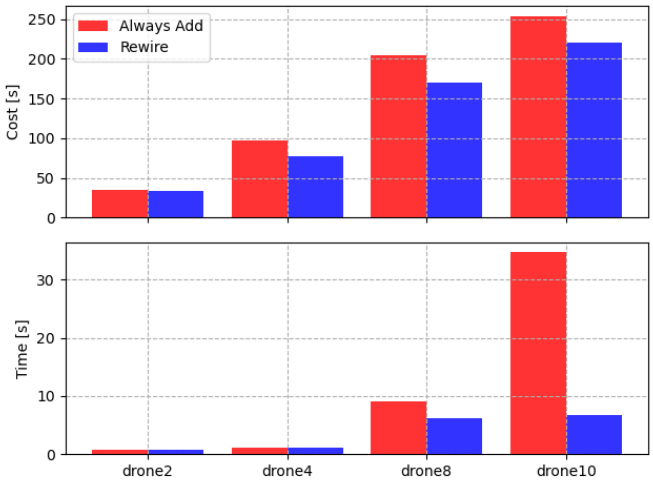


Fig. 7. Analysis of two different methods of adding nodes to the search during the single-robot path planning: *always-add nodes*, *rewiring*. The test is done in the *Wall* example with up to $N = 10$ flying robots. Note, the cost represents the cost of the discrete search without trajectory optimization.

sufficient. However, with increasing number of robots L_1 outperforms the former one, since it provides more accurate value for the heuristic, preventing more high-level runs. In terms of the solution cost, both have similar values across all instances.

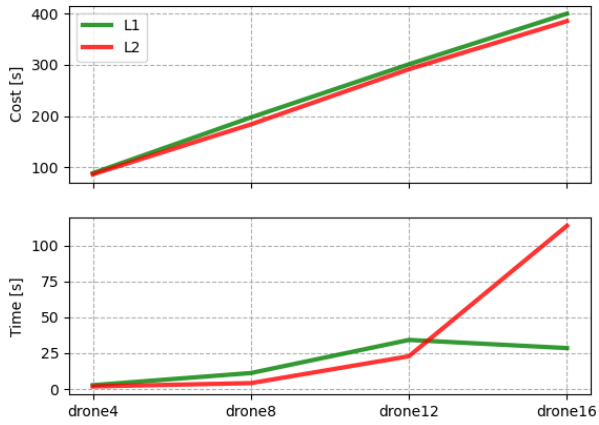


Fig. 8. Analysis of two different methods of computing the focal heuristics during the single-robot path planning: L_1, L_2. The test is done in the *Wall* example with up to $N = 16$ flying robots. Note, the cost and time represents the cost and time of the discrete search without trajectory optimization.

3) Computation Time: We evaluate how much computational time is spent on some key components of our planner, and discuss how it varies from one problem instance to another. Fig. 9 shows computation time spent only on the search part of db-A^*_ω for a single robot. We keep the discontinuity bound fixed, and terminate the planner when the first feasible solution is found. The analysis on the how different values of the discontinuity bound changes the runtime is thoroughly discussed in [51]. For problem instances with small dimensions, like *alcove*, *hetero8* and *wall* the focal heuristic computation is the most expensive operation. The collision checking also takes big portion of time, especially small environments where the obstacle-robot and inter-robot collision happen frequently.
In practice we notice that the trajectory optimization in the joint state space gets slower with the increasing number of robots. For analysis we focus on db-ECBS-R. For example, in order to solve the problem instance *Wall* with $N = 4$, the discrete search takes 1.5 s, while the optimization takes 35 s. The discrete search for the same instance with $N = 8$ needs 3.1 s, while the optimization takes 190.6 s.

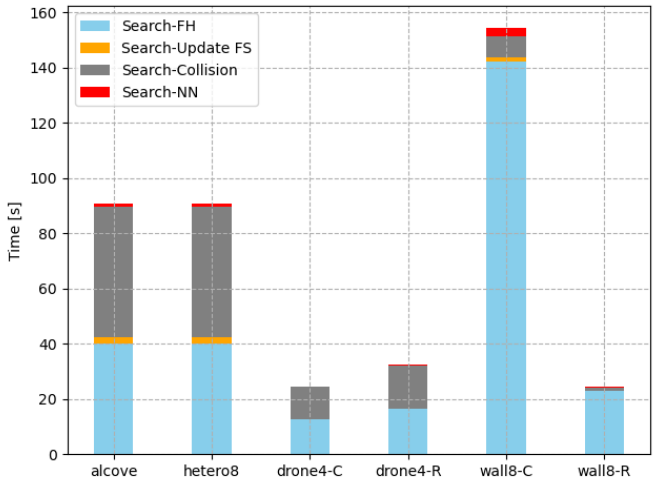


Fig. 9. Analysis on time spent on some key components of the low-level planner db-A^*_ω. The labels represent problem instance names. The times are measured for the first successful iteration of db-ECBS where all robot trajectories are feasible. We report the statistics for only the first robot.

F. Limitations and Future Work

There are three main limitations of db-ECBS. First, motion primitives used in the low-level search are environment-depended. For example, for large-sized environments longer motion primitives are preferred over short ones for the efficiency. This shortage can be addressed by the deployment of generative models like Diffusion models to pre-compute motion primitives conditioning on the environment. Second, focal heuristic estimation is computationally expensive as Fig. 9 shows, due to expensive collision checking. In order to overcome this, the collision shape of motion primitives needs to be dynamic. Third, the trajectory optimization in the joint state space of all robots suffers from large-scale robot teams. With increasing number of robots the optimization starts to fail to return a feasible solution. Developing a meta-optimization strategy, which is able to combine robots dynamically for different time intervals might solve this problem.

VIII. Conclusion

In this paper, we present db-ECBS, an anytime, probabilistically complete and asymptotically bounded suboptimal motion planner for a heterogeneous team of robots operating in close-proximity. Our planner considers robot dynamics, control bounds and reason about aerodynamic interaction force between flying robots. Db-ECBS solves the multi-robot kinodynamic motion planning problem by finding sub-optimal collision-free trajectories with a bounded discontinuity and optimizes them in joint configuration space. We evaluate db-ECBS on a diverse set of challenging environments with and without interaction forces. The results from instances without aerodynamic interaction forces demonstrate that db-ECBS is the most robust compared to other planners on the benchmark when the number of robots increases. The results from instances with interaction forces between flying robots show that db-ECBS consistently finds a solution when the environment forces dense formations. Finally, with physical robot experiments we demonstrate that trajectories obtained with our interaction-aware planner can be tracked safely.

References

[1] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent path finding," in *AAAI Conference on Artificial Intelligence (AAAI)*, 2012.

[2] M. Barer, G. Sharon, R. Stern, and A. Felner, "Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem," in *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)*, T. Schaub, G. Friedrich, and B. O'Sullivan, Eds., ser. Frontiers in Artificial Intelligence and Applications, vol. 263, IOS Press, 2014, pp. 961–962.

[3] W. Hönig, J. A. Preiss, T. K. S. Kumar, G. S. Sukhatme, and N. Ayanian, "Trajectory planning for quadrotor swarms," *IEEE Transactions on Robotics (T-RO)*, vol. 34, no. 4, pp. 856–869, 2018.

[4] W. Hönig, J. O. de Haro, and M. Toussaint, "db-A*: Discontinuity-bounded search for kinodynamic mobile robot motion planning," in *International Conference on Intelligent Robots and Systems (IROS)*, 2022, pp. 13 540–13 547.

[5] K. S. Yakovlev and A. Andreychuk, "Any-angle pathfinding for multiple agents based on SIPP algorithm," in *International Conference on Automated Planning and Scheduling (ICAPS)*, 2017, p. 586.

[6] J. Yu and S. M. LaValle, "Structure and intractability of optimal multi-robot path planning on graphs," in *AAAI Conference on Artificial Intelligence (AAAI)*, 2013, pp. 1443–1449.

[7] R. Cui, B. Gao, and J. Guo, "Pareto-optimal coordination of multiple robots with safety guarantees," *Autonomous Robots*, vol. 32, no. 3, pp. 189–205, 2012.

[8] M. Cáp, P. Novák, M. Selecký, J. Faigl, and J. Vokffnek, "Asynchronous decentralized prioritized planning for coordination in multi-robot system," in *International Conference on Intelligent Robots and Systems (IROS)*, 2013, pp. 3822–3829.

[9] W. Hönig, T. K. S. Kumar, L. Cohen, H. Ma, H. Xu, N. Ayanian, and S. Koenig, "Multi-agent path finding with kinematic constraints," in *International Conference on Automated Planning and Scheduling (ICAPS)*, 2016, pp. 477–485.

[10] M. Likhachev and D. Ferguson, "Planning long dynamically feasible maneuvers for autonomous vehicles," *Int. J. Robotics Res.*, vol. 28, no. 8, pp. 933–945, 2009.

[11] M. Pivtoraiko and A. Kelly, "Kinodynamic motion planning with state lattice motion primitives," in *International Conference on Intelligent Robots and Systems (IROS)*, 2011, pp. 2172–2179.

[12] M. Dharmadhikari, T. Dang, L. Solanka, J. Loje, H. Nguyen, N. Khedekar, and K. Alexis, "Motion primitives-based path planning for fast and agile exploration using aerial robots," in *International Conference on Robotics and Automation (ICRA)*, 2020, pp. 179–185.

[13] B. J. Cohen, S. Chitta, and M. Likhachev, "Search-based planning for manipulation with motion primitives," in *International Conference on Robotics and Automation (ICRA)*, 2010, pp. 2902–2908.

[14] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics (T-RO)*, vol. 12, no. 4, pp. 566–580, 1996.

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

[16] Y. Li, Z. Littlefield, and K. E. Bekris, "Asymptotically optimal sampling-based kinodynamic planning," *The International Journal of Robotics Research (IJRR)*, vol. 35, no. 5, pp. 528–564, 2016.

[17] M. Toussaint, "A tutorial on newton methods for constrained trajectory optimization and relations to slam, gaussian process smoothing, optimal control, and probabilistic inference," in *Geometric and Numerical Foundations of Movements*, ser. Springer Tracts in Advanced Robotics, J. Laumond, N. Mansard, and J. Lasserre, Eds., vol. 117, 2017, pp. 361–392.

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

[19] R. Bonalli, A. Cauligi, A. Bylard, and M. Pavone, "Gusto: Guaranteed sequential trajectory optimization via sequential convex programming," in *International Conference on Robotics and Automation (ICRA)*, IEEE, 2019, pp. 6741–6747.

[20] B. Sakcak, L. Bascetta, G. Ferretti, and M. Prandini, "Sampling-based optimal kinodynamic planning with motion primitives," *Autonomous Robots*, vol. 43, no. 7, pp. 1715–1732, 2019.

[21] R. Natarajan, H. Choset, and M. Likhachev, "Interleaving graph search and trajectory optimization for aggressive quadrotor flight," *IEEE Robotics and Automation Letters (RA-L)*, vol. 6, no. 3, pp. 5357–5364, 2021.

[22] S. Choudhury, J. D. Gammell, T. D. Barfoot, S. S. Srinivasa, and S. A. Scherer, "Regionally accelerated batch informed trees (RA-BIT*): A framework to integrate local information into optimal path planning," in *International Conference on Robotics and Automation (ICRA)*, 2016, pp. 4207–4214.

[23] J. Chen, J. Li, C. Fan, and B. C. Williams, "Scalable and safe multi-agent motion planning with nonlinear dynamics and bounded disturbances," in *AAAI Conference on Artificial Intelligence (AAAI)*, 2021, pp. 11 237–11 245.

[24] J. Kottinger, S. Almagor, and M. Lahijanian, "Conflict-based search for multi-robot motion planning with kinodynamic constraints," in *International Conference on Intelligent Robots and Systems (IROS)*, 2022, pp. 13 494–13 499.

[25] A. Moldagalieva, J. Ortiz-Haro, M. Toussaint, and W. Hönig, "Db-cbs: Discontinuity-bounded conflict-based search for multi-robot kinodynamic motion planning," in *International Conference on Robotics and Automation (ICRA)*, 2024, pp. 14 569–14 575.

[26] L. Cohen, T. Uras, T. K. S. Kumar, and S. Koenig, "Optimal and bounded-suboptimal multi-agent motion planning," in *International Symposium on Combinatorial Search (SoCS)*, 2021.

[27] A. Tajbakhsh, L. T. Biegler, and A. M. Johnson, "Conflict-based model predictive control for scalable multi-robot motion planning," *CoRR*, vol. abs/2303.01619, 2023. arXiv: 2303.01619.

[28] G. Shi, X. Shi, M. O'Connell, R. Yu, K. Azizzadenesheli, A. Anandkumar, Y. Yue, and S.-J. Chung, "Neural lander: Stable drone landing control using learned dynamics," in *International Conference on Robotics and Automation (ICRA)*, 2019, pp. 9784–9790.

[29] D. Shukla and N. Komerath, "Multirotor drone aerodynamic interaction investigation," *Drones*, vol. 2, no. 4, 2018, ISSN: 2504-446X.

[30] K. P. Jain, T. Fortmuller, J. Byun, S. A. Mäkiharju, and M. W. Mueller, "Modeling of aerodynamic disturbances for proximity flight of multirotors," in *International Conference on Unmanned Aircraft Systems (ICUAS)*, 2019, pp. 1261–1269.

[31] J. Gielis, A. Shankar, R. Kortvelesy, and A. Prorok, "Modeling aggregate downwash forces for dense multirotor flight," *CoRR*, vol. abs/2312.03488, 2023. arXiv: 2312.03488.

[32] X. Du, C. E. Luis, M. Vukosavljev, and A. P. Schoellig, "Fast and in sync: Periodic swarm patterns for quadrotors," in *International Conference on Robotics and Automation (ICRA)*, IEEE, 2019, pp. 9143–9149.

[33] X. Zhou, J. Zhu, H. Zhou, C. Xu, and F. Gao, "Ego-swarm: A fully autonomous and decentralized quadrotor swarm system in cluttered environments," in *International Conference on Robotics and Automation (ICRA)*, 2021, pp. 4101–4107.

[34] M. Debord, W. Hönig, and N. Ayanian, "Trajectory planning for heterogeneous robot teams," in *International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 7924–7931.

[35] L. Bauersfeld, E. Kaufmann, P. Foehn, S. Sun, and D. Scaramuzza, "Neurobem: Hybrid aerodynamic quadrotor model," *RSS: Robotics, Science, and Systems*, 2021.

[36] G. Shi, W. Hönig, Y. Yue, and S.-J. Chung, "Neural-swarm: Decentralized close-proximity multirotor control using learned interactions," in *International Conference on Robotics and Automation (ICRA)*, 2020, pp. 3241–3247.

[37] G. Shi, W. Hönig, X. Shi, Y. Yue, and S.-J. Chung, "Neural-swarm2: Planning and control of heterogeneous multirotor swarms using learned interactions," *IEEE Transactions on Robotics (T-RO)*, vol. 38, no. 2, pp. 1063–1079, 2022.

[38] J. Li, L. Han, H. Yu, Y. Lin, Q. Li, and Z. Ren, "Nonlinear mpc for quadrotors in close-proximity flight with neural network downwash prediction," in *2023 62nd IEEE Conference on Decision and Control (CDC)*, 2023, pp. 2122–2128.

[39] H. Smith, A. Shankar, J. Gielis, J. Blumenkamp, and A. Prorok, "So(2)-equivariant downwash models for close proximity flight," *IEEE Robotics and Automation Letters (RA-L)*, vol. 9, no. 2, pp. 1174–1181, 2024.

[40] S. Hu, D. D. Harabor, G. Gange, P. J. Stuckey, and N. R. Sturtevant, "Multi-agent path finding with temporal jump point search," in *International Conference on Automated Planning and Scheduling (ICAPS)*, 2022, pp. 169–173.

[41] I. Solis, J. Motes, R. Sandström, and N. M. Amato, "Representation-optimal multi-robot motion planning using conflict-based search," *IEEE Robotics and Automation Letters (RA-L)*, vol. 6, no. 3, pp. 4608–4615, 2021.

[42] K. Hauser and Y. Zhou, "Asymptotically optimal planning by feasible kinodynamic planning in a state-cost space," *IEEE Transactions on Robotics (T-RO)*, vol. 32, no. 6, pp. 1431–1443, 2016.

[43] W. Hönig, J. A. Preiss, T. K. S. Kumar, G. S. Sukhatme, and N. Ayanian, "Trajectory planning for quadrotor swarms," *IEEE Transactions on Robotics (T-RO)*, vol. 34, no. 4, pp. 856–869, 2018.

[44] S. M. LaValle, *Planning Algorithms*. 2006, 1362 pp., ISBN: 978-1-139-45517-6.

[45] I. A. Şucan, M. Moll, and L. E. Kavraki, "The open motion planning library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, 2012.

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

[47] J. O.-H. et. al., *Dynoplan*, https://github.com/quimortiz/dynoplan.

[48] C. Mastalli, R. Budhiraja, W. Merkt, G. Saurel, B. Hammoud, M. Naveau, J. Carpentier, L. Righetti, S. Vijayakumar, and N. Mansard, "Crocoddyl: An efficient and versatile framework for multi-contact optimal control," in *International Conference on Robotics and Automation (ICRA)*, 2020, pp. 2536–2542.

[49] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Póczos, R. Salakhutdinov, and A. J. Smola, "Deep sets," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, Eds., 2017, pp. 3391–3401.

[50] J. A. Preiss, W. Hönig, G. S. Sukhatme, and N. Ayanian, "Crazyswarm: A large nano-quadcopter swarm," in *International Conference on Robotics and Automation (ICRA)*, 2017, pp. 3299–3304.

[51] J. Ortiz-Haro, W. Hönig, V. N. Hartmann, and M. Toussaint, "Idb-a*: Iterative search and optimization for optimal kinodynamic motion planning," *IEEE Transactions on Robotics (T-RO)*, pp. 1–19, 2024.