Multi-Agent Path Planning and Trajectory Generation for Confined Environments

Hikmet Beyoglu, Stephan Weiss, Bernhard Rinner

***Abstract*— Planning the collision-free, simultaneous movement of multiple agents is a fundamental problem in multi-robot systems and becomes particularly challenging in highly confined environments. This paper addresses this problem by first searching for collision-free paths in a discretized environment and then optimizing the agents' dynamically feasible trajectories along the discrete paths with respect to energy and time. Our approach extends the available movement options at each planning step of the enhanced conflict-based search and results in shorter path lengths, faster planning times, and reduced number of waiting events for agents at waypoints. We compare our new approach with the original algorithm in a simulation study and investigate the performance in confined spaces.**

I. Introduction

Multi-agent systems (MAS) gain importance and relevance for various fields and applications. Unmanned aerial vehicles (UAVs) are a particular MAS field with innovative applications in transportation, communication, surveillance, disaster support, and entertainment, to name a few examples [1], [2], [3], [4]. In all these applications, the behavior of individual agents needs to be coordinated to complete the given mission successfully and efficiently [5]. This paper tackles path planning and trajectory generation as a particular coordination problem where several agents concurrently move from a start to an end constellation [6], [7]. These coordinated, simultaneous movements are highly relevant in various UAV applications including flight shows, multi-coverage, and warehouse delivery. We first plan collision-free paths for all agents in a discrete space and generate then dynamically feasible, snap- and time-optimized trajectories for all UAVs. Figure 1 depicts a low dimensional (for legibility) constellation change of six agents through a window as an example.

Our approach is an offline path planning and trajectory generation method for simultaneous, collision-free, and fast movement of agents in environments with and without obstacles. Our approach extends the available movement options at each planning step of enhanced conflict-based search (ECBS) [8] and results in the following benefits, which are particularly present in confined environments: shorter path lengths, faster planning times, and reduced waiting events for agents. These waiting events are problematic for generating energy-efficient and time-optimized trajectories due to deceleration, hovering, and acceleration at specific points. Our contribution can be summarized as follows: (i) a discrete path planning algorithm dubbed as *Extended Enhanced Conflict-Based Search* (EECBS) followed by generating piece-wise, energy- and time-optimized dynamically feasible continuous trajectories, (ii) a comparison of the state of the art ECBS and our EECBS algorithms, and (iii) a demonstration of the trajectory generation and detailed study of performance in confined spaces.

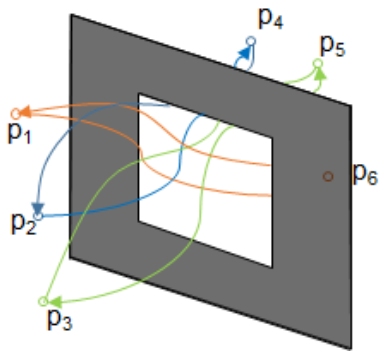


Fig. 1: Problem sketch. Six agents move concurrently from start to goal positions p_i through a window without collisions.

The remainder of the paper is organized as follows: Section II discusses related work. Section III sketches the problem under study and Section IV describes our path planning and trajectory generation approach. Section V discusses the simulation study and Section VI concludes the paper.

II. Related Work

Single and multiple agent path planning in both static and dynamic environments has been extensively studied (e.g., [9]). In general, the goal of all planning algorithms is to provide collision-free paths for all agents during the mission. A typical strategy is to formulate planning as an optimization problem. Particularly, mixed integer linear programming (MILP) is used to model collision constraints with binary variables in [10]. However, this approach is computationally expensive and not suitable for a large number of agents. Sequential convex programming (SCP) can be used to reduce the computation time [11]. SCP is also utilized to generate energy-optimized trajectories for small quadrotor teams in [12]. While this approach is beneficial for small teams, it

All authors are with the University of Klagenfurt, Austria (dronehub Klagenfurt: uav.aau.at). Hikmet Beyoglu (hikmet.beyoglu@hotmail.com) and Bernhard Rinner (bernhard.rinner@aau.at) are with the Institute of Networked and Embedded Systems and Stephan Weiss (stephan.weiss@aau.at) is with the Control of Networked Systems Group.

This work was partially supported by the EU-H2020 project BUG-WRIGHT2 (GA 871260)

does not scale well.

Park et al. [13] discretize the space to perform initial path planning, build then safe and relative safe flight corridors, and finally formulate a convex optimization problem to provide jerk-optimal paths. However, this approach requires a large total flight distance, even for a small number of agents, since all agents are arranged to conclude the mission with the same number of trajectory segments.

Luis et al. [6] propose to use model predictive control (MPC) for trajectory planning and show that a high success rate with a moderate computational complexity compared to the previous methods is possible. They introduce on-demand collision avoidance with soft constraints to improve scalability and success rate. Ladinig et al. [7] extend MPC with potential fields to further improve the success rate and the transition time.

Robinson et al. [14] apply a combination of the partial differential equations solver pseudospectral (PS) and a reachability analysis based on the surface method level set (LS) to generate high-quality and time-optimal trajectories. This approach is limited to two-dimensional environments, and its success rate decreases drastically for a large number of agents. Zou et al. [15] use buffered Voronoi cells (BVC), where each robot continually computes its cell and plans its path within the BVC in a receding horizon fashion. BVC is considerably faster than MPC and SCP approaches, but can not guarantee a collision-free solution in obstacle-dense spaces due to deadlocks. Deadlocks in multi-agent constellation change occur when two or more robots block each other such that none of them can continue its path without colliding [16].

As shown in [17], previous work (e.g., [18]) relies on snap-optimized polynomials as smooth and energy-optimized trajectories. The polynomial coefficients are determined by solving a quadratic programming (QP) problem that optimizes a cost function of the path derivatives while maintaining several constraints. The imposed analytical solution to the formulated QP problem in [18] has a problematic issue by generating sensitive coefficients that are arduous and sometimes impossible to be executed by agents. The issue drastically increases with long paths requiring the optimizer to generate piecewise polynomials with many segments as trajectories. Richter et al. [17] tackle the problem of sensitive polynomials by a constrained QP problem as a base for solving a re-formulated unconstrained QP problem to optimize polynomial trajectory segments jointly. This approach is numerically stable for high-order polynomials and large numbers of segments.

III. Problem Statement

Figure 1 depicts the problem under study. For a given team of A agents, we search for trajectories providing simultaneous, collision-free, fast, and energy-efficient transitions from a start to an end constellation.

A group of A agents is located at positions p_i for each agent $i \in \{1, \ldots, A\}$ in the Cartesian coordinate system where a constellation is defined as static and collision-free positions of all A agents. The environment is limited to a given flight volume specified by a lower bound p_{min} and an upper bound p_{max} and is discretized with resolution r into $r \times r \times r$ cells. The flight volume can contain static obstacles reducing the available flight volume, i.e., cells occupied by obstacles can not be entered by any agent. Collision constraints can be formulated as minimum distance between agents or agents and obstacles [6]. This constraint can be simplified in the discretized space by guaranteeing that no cell in the available flight volume is occupied by more than one agent. Additionally, the agent dynamics at any time t adheres to the following physical limits

$$\begin{aligned} p_{\min} \leq p_i(t) \leq p_{\max} \\ v_{\min} \leq v_i(t) \leq v_{\max} \\ a_{\min} \leq a_i(t) \leq a_{\max}. \end{aligned} \tag{1}$$

We adopt the model from [7] for the agent dynamics described by velocity $v(t)$ and acceleration $a(t)$. In the discretized space, an agent can move to any of the 26 neighboring cells in one time step, resulting in a movement in forward, backward, left, right, up, down, and any diagonal direction in 3D space.

The overall objective is to generate trajectories such that all agents move concurrently from a start constellation s_i to an end constellation e_i ($i \in \{1, \ldots, A\}$) satisfying physical limits and collisions constraints and optimizing flight time and energy.

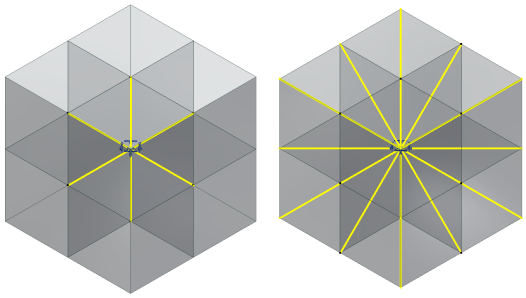
IV. Path Planning and Trajectory Generation

We follow a two-step approach for planning the simultaneous movement of multiple agents. In a first step, we plan collision-free paths in discrete space for all agents with an extension of enhanced conflict-based search. Our EECBS algorithm operates in a 3D grid topology and exploits 26 adjacencies as potential movement actions (compared to ECBS using only 6 adjacencies). In a second step, we generate optimized non-discretized trajectories based on the EECBS paths. We adopt the formulation of Sharon et al. [19] for the description of the planning algorithm.

A. Conflict-based search algorithm

Conflict-based search (CBS) is a centralized multi-agent path finding algorithm that operates on two levels. At the high level, a search is performed on a conflict tree (CT) which is a tree based on conflicts between individual agents. At the low level, fast single-agent searches are performed to satisfy the constraints imposed by the high level CT node [19]. These constraints limit the occupation of a vertex and the movement along an edge in the CT by multiple agents at the same time [20].

Sharon et al. [19] showed that CBS outperforms traditional planning algorithms A*, enhanced partial expansion A*, and increasing cost tree search in terms of the generated paths length. However, CBS faces problems with scalability and computation time.



(a) (b)

Fig. 2: Projection of a 3D grid cell topology and the available movement actions for ECBS (a) and EECBS (b), respectively.

B. Extended enhanced conflict-based search algorithm

Barer et al. [8] introduced the enhanced CBS (ECBS) algorithm as an approach to overcome the limitations of CBS and to control the trade-off between run time and solution quality. ECBS is a bounded sub-optimal solver which returns a solution that is guaranteed to be less or equal to $w \cdot C^*$, where $w = 1+\epsilon$ is user-defined suboptimality factor and C^* is the cost of the optimal solution [8].

The idea behind bounded sub-optimal CBS is to shrink the list of nodes to be examined in both low and high-level searches based on the focal search (FS), which is a widely used bounded sub-optimal search algorithm [8]. FS is organized by two lists of nodes (open and focal) and can be applied in both low and high-level searches. In low-level search, the open list includes all possible nodes to be taken in the graph and the focal list, which is a subset of the open list, includes all nodes n whose costs will be examined. In high-level search, the open list includes all possible nodes n in the conflict tree that can be examined, and the focal list includes all the nodes that will be examined, i.e., whether their costs satisfy the condition

$$f_1(n) \leq w \cdot f_{1\text{min}} \tag{2}$$

where f_{1min} is the minimal f_1 value in the open list.

Heuristic f_2 is used to select which node from the focal list to expand. For example, the Euclidean distance to the goal can be used as f_2 in the low level, whereas the node with the lowest number of conflicts can be used in the high level. FS is represented as *focal_search*(f_1, f_2).

The available movement actions at each planning step strongly influence the performance of ECBS. To the best of our knowledge, path planning in a 3D grid based on ECBS has only used orthogonal movement directions resulting in 6 possible movement actions: left, right, forward, backward, up and down (e.g., [13]). By including all diagonal movement directions, our EECBS planning algorithm results in 26 available movement actions which significantly enlarges planning options. Figure 2 depicts the different movement actions in ECBS and EECBS.

In the following we describe the EECBS algorithm. Consider a node N in a conflict tree. At the low level of EECBS, the focal search is performed for each agent a_i such that the low level generates paths, based on the extended action potential set, with cost of at most $w \cdot Open_{LB,i}$, where $Open_{LB,i}$ is the lower cost bound on the minimal path for a_i that satisfies the set of constraints of conflict tree node N and is computed by the focal search. Thus, the bound is given as $N_{LB} = \sum_{i=1}^{k} N_{LB,i}$ [20]. At the high level, the focal search is performed with a focal list of all conflict tree nodes $N \in open$ such that $N_{cost} \leq w \cdot LB$. The open list $open$ includes all potential nodes and $LB = min_{N \in open} N_{LB}$. LB is a lower bound on the sum of costs of any conflict-free solution. If a conflict-free solution is found while expanding a conflict tree node in the focal list, it is guaranteed to be a w-approximate of the optimal solution.

Algorithm 1 presents a pseudo code representation of the EECBS algorithm whose notation is adopted from [20]. In line 1, the open list is constructed with the root node which is the first node in the conflict tree that includes a proposed path for each agent from the low level. Lines 2 and 3 initialize the open and focal lists with the root node, respectively. The cost lower bound LB is initialized with the cost of the root node in line 4. EECBS iterates through the remaining lines as long as there are nodes in the open list. A node N with the minimum cost is selected (line 6). If N has no conflict, a solution is found and EECBS terminates (line 7). Otherwise, N is removed from both lists and the total cost of N is checked whether it is smaller than the lower bound LB (line 10). If true, LB is set to the new bound and nodes that satisfy the new bound are assigned to the focal list (line 12). Then a conflict from N_{conf} is selected (line 13), two child nodes of N are generated, and the A* algorithm of the low level is called to resolve the conflict (line 15). This low-level search checks all orthogonal and diagonal neighboring cells resulting in 26 potential movement actions. If no solution is found, a waiting event is introduced (line 18). The newly generated children are added to the open list in line 20. Lines 21 to 23 check if any of the children satisfy condition 2 and add them to the focal list. Since EECBS is able to generate paths with diagonal movements, we use the Chebyshev distance instead of the Manhattan distance as admissible heuristic.

C. Energy- and time-optimized trajectory generation

EECBS path planning results in sequences of grid cells. We use the center of these cells as waypoints for the trajectory generation and perform an optimization process to generate smooth and energy-efficient flight paths based on piecewise polynomials. Representing trajectories with polynomials suits highly dynamic ground and aerial robots [17]. Trajectory generation is modeled as a quadratic programming (QP) problem that optimizes a cost function on the path derivatives while maintaining a number of constraints. We adopt techniques used in [17], [21] to generate energy- and subsequently time-optimized piecewise polynomials.

Algorithm 1: EECBS Path Planning

Input: Planning instance and suboptimality factor w

generate root CT node R with an initial solution
initialize open list $open := \{R\}$
initialize focal list $focal := \{R\}$
$LB := R_{LB}$
while $open \neq \emptyset$ **do**
  $N :=$ CT node with minimum distance in $focal$
  if $N_{conf} = \emptyset$ **then**
    return N_{sol}
  remove N from $open$ and $focal$
  if $min_{N \in open}\{N_{LB}\} < LB$ **then**
    $LB := min_{N \in open}\{N_{LB}\}$
    $focal := \{N \in open : N_{LB} \leq w \cdot LB\}$
  pick a conflict in N_{conf}
  generate two child CT nodes N^1 and N^2 of N
  call low-level-search for each N^i and calculate N^i_{sol}, N^i_{cost} and N^i_{conf} for $i \in \{1, 2\}$
  Low-level perform searches for N^i_{sol} in a 3D grid cell topology that includes 26 adjacencies.
  if $N^i_{sol} = \emptyset$ **then**
    add waiting event
    perform new low-level search
  add N^i to $open$ for $i = 1, 2$
  for $i \in \{1, 2\}$ **do**
    if $N^i_{cost} \leq w \cdot LB$ **then**
      add N^i to $focal$
return no solution

The trajectory of robot i, $p^i(t) \in \mathbb{R}^3$, can be represented as M-segment piecewise standard polynomials

$$p^i(t) = \begin{cases} \sum_{w=0}^{7} c_{w,1} t^w & t \in [T_0, T_1] \\ \sum_{w=0}^{7} c_{w,2} t^w & t \in [T_1, T_2] \\ \vdots & \\ \sum_{w=0}^{7} c_{w,M} t^w & t \in [T_{M-1}, T_M] \end{cases}$$

where k is the segment's number, $c_{w,k}$ is the w^{th} coefficient in k^{th} segment, such that $k \in [1, M]$, $w \in [0, 7]$. The polynomials in $p^i(t)$ are calculated by solving the optimization function

$$f^i \;=\; \min \int_{T_0}^{T_M} \left\| \frac{d^{k_r} p^i}{dt^{k_r}} \right\|_2^2 dt \qquad (3)$$

as a QP. Similar to [7] we minimize the integral of the square of the norm of the snap ($k_r = 4$) resulting in a 7th order polynomial. Snap-optimized polynomial splines are effective quadrotor trajectories, since the motor commands and attitude accelerations of the vehicle are proportional to the snap [18]. The following constraints are imposed for the optimization:

- Pre-selected initial and goal waypoints constraints

$$p^i_{k=1}(T_0) = s_i \;,\; p^i_{k=M}(T_M) = e_i$$

  where s_i is the start position and e_i is the end position of agent i.

- Intermediate waypoints continuity constraints for position, velocity, acceleration and jerk:

$$p^{(n)}_{k-1}(T_k) = p^{(n)}_k(T_k) \;,\; n = 0, 1, 2, 3$$

- Velocity, acceleration, and jerk should be zero at the start and goal waypoints for all agents in the mission.

$$\left.\frac{d^r p^i(t)}{dt^r}\right|_{t=j} = 0 \;,\; j = T_0, T_M;\; r = 1, 2, 3$$

- Physical limit constraints (1).

More specific constraints, such as downwash or safety distances, can be added if needed.

V. Simulation Study

We evaluate the performance of the EECBS path planning algorithm with three different scenarios. The first scenario compares EECBS with ECBS in obstacle-free environments under various space resolutions. The second scenario shows the EECBS behavior in environments with randomly located obstacles at various densities. The third scenario investigates the EECBS behavior for constellation changes through a window with various sizes.

A. Simulation setup

We adopted and extended the code by Vedder et al. [22] and Park et al. [23] for the implementation of our EECBS algorithm. We implemented the simulation framework in Python 3.9.6 and performed all simulations on an Intel Core laptop with 8 GB RAM running at 2.60 GHz.

Our evaluation is based on the following performance parameters. The *success rate* corresponds to the percentage of successes over all simulated cases, i.e., the number of collision-free constellation changes over all simulated scenarios. The *total cost* is equal to the aggregated path lengths of all agents in a scenario where the path length is measured by the number of traversed waypoints. The *cost per agent* is equal to the average path length of a single agent in a scenario, and the *longest path* corresponds to the agent with maximal cost, i.e., the longest path of an agent for a scenario. The *number of waiting events* is the number of waypoints along the agents' paths where an agent has to wait in order to avoid a collision with another crossing agent. The *run time* corresponds to the time required for computing the paths of all agents for a constellation change.

In our simulation study we varied the number of agents, the resolution of the 3D environment, and the density or size of obstacles. Each simulation is executed 30, 10, and 10 times in obstacle-free, obstacle-dense, and window-crossing environments, respectively, and the start and end positions of the constellation and the obstacles (for the second scenario) are placed randomly. We use the mean values of the performance parameters in our simulation study.

B. Obstacle-free environment

For the comparison of ECBS and EECBS in obstacle-free environments we computed paths for constellation changes for 1 to 100 agents in discretized 3D environments with resolution $r \in \{8, 20, 100\}$. Thus, the environment is composed by either $8 \times 8 \times 8$, $20 \times 20 \times 20$, or $100 \times 100 \times 100$ cells. Figure 3 shows that EECBS is superior to ECBS in these scenarios.

1) Success rate: Both ECBS and EECBS achieve a 100% success rate for all cases starting from 1 up to 100 agents, under all studied resolution scenarios in an obstacle-free environment.

2) Total cost: As Figure 3a shows the total cost of ECBS and EECBS almost linearly increases with the number of agents. Due to the increased number of movement actions available, EECBS outperforms ECBS in all cases. In particular, EECBS achieves a total cost reduction of 56%, 54% and 54% in environments with resolution $8 \times 8 \times 8$, $20 \times 20 \times 20$ and $100 \times 100 \times 100$, respectively.

3) Cost per agent: Figures 3b depicts the cost per agent as a function of the number of agents of ECBS and EECBS. The achieved cost per agent is approximately constant for both algorithms. EECBS outperforms ECBS in all cases, i.e., by 46%, 48% and 46% in environments with resolution $8 \times 8 \times 8$, $20 \times 20 \times 20$ and $100 \times 100 \times 100$, respectively.

4) Longest path: Figures 3c shows the longest path as a function of the number of agents for ECBS and EECBS. For both algorithms the longest paths slightly increase with growing number of agents. Similarly to the previous performance parameters, EECBS outperforms ECBS in all cases. EECBS is able to reduce the longest path by 47%, 50% and 50% in environments with resolution $8 \times 8 \times 8$, $20 \times 20 \times 20$ and $100 \times 100 \times 100$, respectively.

5) Waiting events: Figure 3d depicts the introduced waiting events at waypoints along the generated paths to avoid collisions with crossing agents. In general, the number of waiting events increase with increasing number of agents and decreasing resolution, because the likelihood of crossing agents increase in environments with dense paths. The difference between ECBS and EECBS is most noticeable for $r = 8$, where ECBS introduces 17 waiting events and EECBS less than 3 waiting events on average. This difference decreases for $r = 20$ to 1 waiting event for EECBS and 2 waiting events for ECBS, respectively. Both algorithms avoid introducing waiting events for $r = 100$.

6) Run time: Figures 3e shows the run time for different resolutions for the path generation in our simulation framework as a function of the number of agents. For all three resolutions the run time of ECBS and EECBS is almost identical for 1 to 45 agents. For larger number of agents, the run time for ECBS increases faster than for EECBS. It is worth mentioning that the ECBS run time has a larger variation, which can be recognized by fluctuation of the run time (mean) values. This effect is particularly strong for $r = 100$ and is caused by a large number of potential conflicts in the ECBS search. Thus, even though EECBS needs to check more options (26 versus 6), the reduction in conflicts to solve due to the added evasion options has a higher impact on the run time making EECBS the winning method.

C. Environment with obstacles

In this scenario we evaluate the performance of EECBS in environments with randomly placed obstacles. We are particularly interested in identifying performance degradation in obstacle-dense environments. In this study, obstacles have a fixed size of 2 neighboring cells and are randomly placed in the environment. The occupancy of the obstacles is varied from 0% to 95% of the available environment which means that 0% to 95% of all cells are occupied by obstacles. We computed paths for constellation changes of 1 to 50 agents in environments with resolution $r \in \{12, 50\}$. Figure 4 summarizes the performance of EECBS in an environments with these resolutions.

1) Success rate: EECBS is able to achieve a 100% success rate for all simulated cases with 1 to 50 agents, varying obstacles occupy and environments with $12 \times 12 \times 12$ and $50 \times 50 \times 50$ resolution.

2) Total cost: Figure 4a shows the total cost of the generated paths as a function of the number of agents. The total cost linearly increases independent of the obstacle occupancy in the environment. The aggregated average path length (total cost) is naturally larger in environments with larger resolution.

3) Cost per agent: Figures 4b depicts the cost per agent as a function of the number of agents. EECBS cost per agent slightly varies for all numbers of agents independent of the obstacle occupation. The average path length (cost per agent) is approximately 6 and 25 for resolutions 12 and 50, respectively.

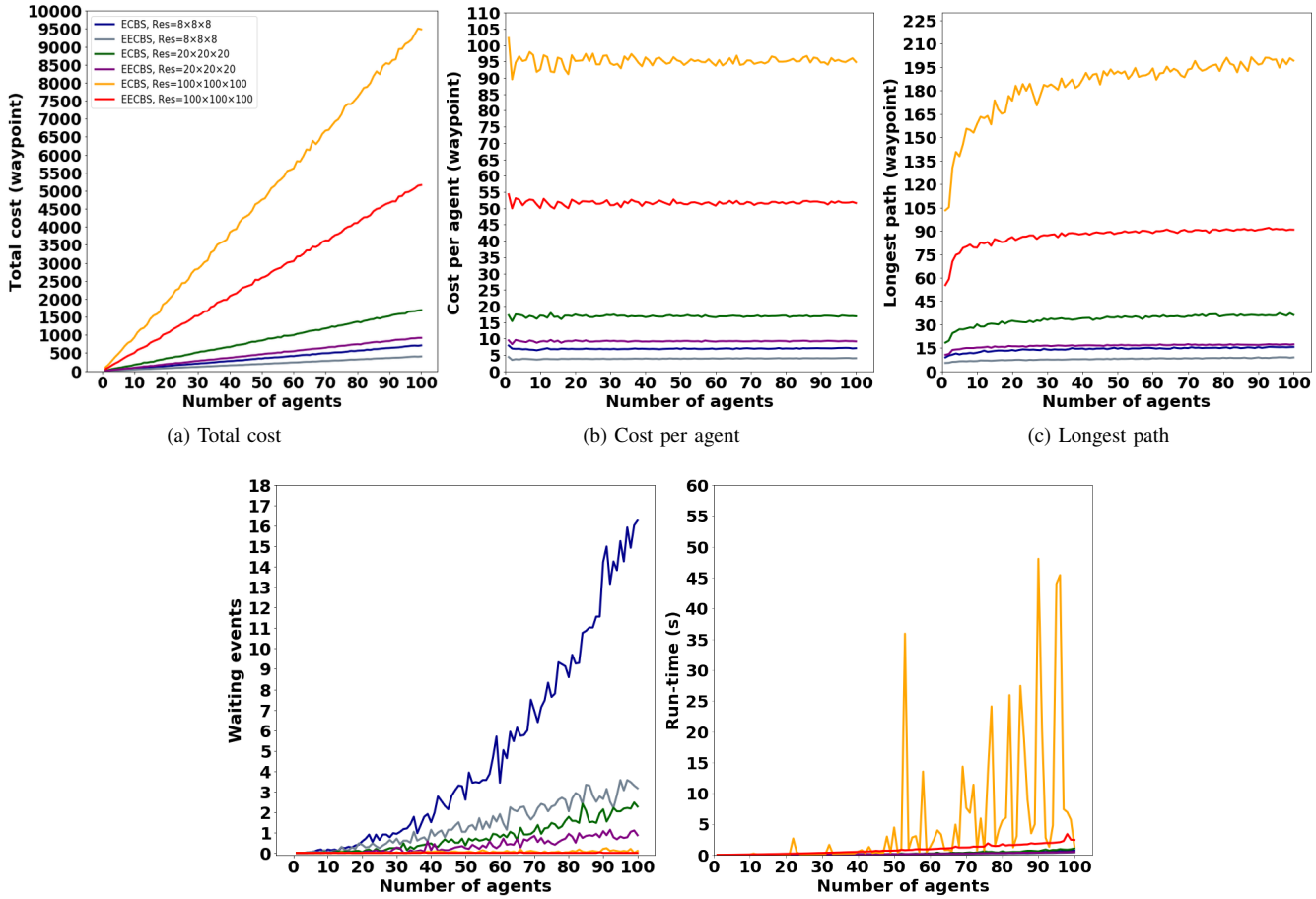
4) Longest path: Figure 4c shows the longest path as function of the number of agents. The length of the longest path increases for a low number of agents and saturates to 11 and 44 waypoints for constellation changes with more than 15 agents for resolutions 12 and 50, respectively.

5) Waiting events: As can bee seen in Figure 4d, EECBS hardly introduces waiting events. The maximum average number is less than 0.1 for all simulated constellation changes in an environment with $r = 50$. In an environment with $r = 12$, the number of waiting events slightly increase with the number of agents. Even with an occupation of 80% the waiting events do not exceed 1 waypoint for large constellation changes.

6) Run time: Figure 4e shows the run time for the path generation in our simulation framework as a function of the number of agents. The run time as well as the variation of the mean run time increase with the number of agents. The mean value stays below 2.1 seconds for the environment with $r = 50$. For an environment with $r = 12$ the mean run time is always less than 0.1 seconds.

D. Constellation change through a window

We finally evaluate the performance of EECBS for constellation changes through a window with variable size.



(a) Total cost (b) Cost per agent (c) Longest path

(d) Waiting events (e) Run time

Fig. 3: Comparison of ECBS and EECBS in obstacle-free environments with three different resolutions $r \in \{8, 20, 100\}$ for 1 to 100 agents. Legend: — ECBS $8 \times 8 \times 8$; — EECBS $8 \times 8 \times 8$; — ECBS $20 \times 20 \times 20$; — EECBS $20 \times 20 \times 20$; — ECBS $100 \times 100 \times 100$; — EECBS $100 \times 100 \times 100$.

In this scenario the window is placed in the center of the environment and start and end positions are randomly placed on the opposite sides of the window such that every agent must pass through the window. The window border percentage D represents the ratio of window border area over the entire cross section area. Figure 5 depicts a window with a 2-cell border in an environment with $r = 12$ resulting in a window border percentage $D = 55\%$. We computed the paths for constellation changes of 1 to 50 agents in environments with resolution $r \in \{12, 100\}$.

1) Success rate: EECBS achieves a 100% success rate for all window border sizes in the $100 \times 100 \times 100$ environment. In the $12 \times 12 \times 12$ environment, EECBS is not able to achieve 100% success rate for an 80% window border percentage (cp. Figure 6e). Due to the tight space limitations of this scenario, the success rate significantly drops for more than 24 agents and EECBS can not find a solution for more than 41 agents. Consequently, the graph of the other performance parameters in Figure 6 terminate at 41 agents.

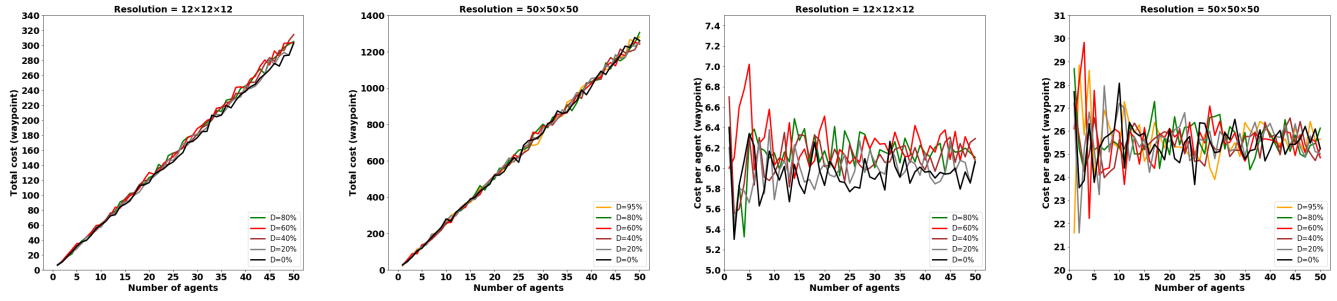
2) Total cost: As depicted in Figure 6a the total cost linearly increases with the number of agents for both environment resolutions. In the most confined scenario with $r = 12$ and $D = 80\%$, the total cost increases more than in the other cases.

3) Cost per agent: A similar behavior can also be seen for the cost per agent (Figure 6b). Except for the most confined scenario, the average path length slightly varies around values which are almost identical.

4) Longest path: Figure 6c depicts the length of the longest path as function of the number of agents. In general, the longest path increase with a low number of agents and remains then almost constant. However, the scenario with $r = 12$ and $D = 80\%$ results in a steadily increasing longest path.

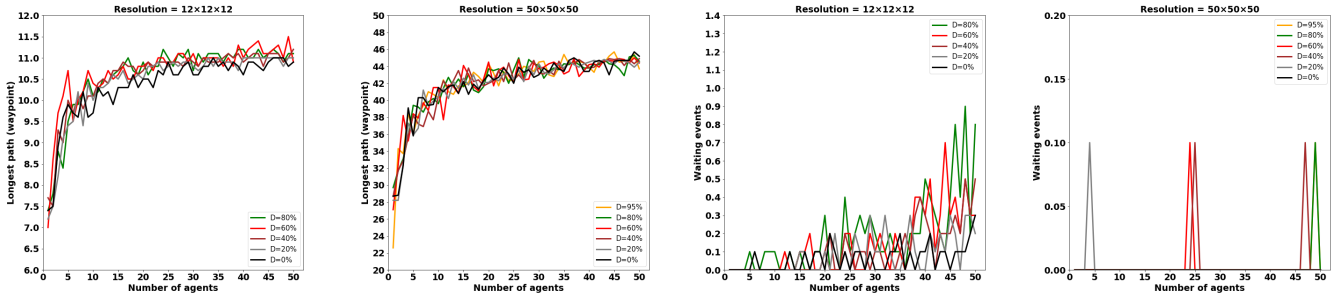
5) Waiting events: EECBS does hardly introduce any waiting events in the environment with $r = 100$. In the low resolution environment EECBS needs to include a few waiting events for successful path planning. The severe planning restrictions can also be seen by the significant increase of waiting events for window border size $D = 80\%$.

6) Run time: The run time is a good indicator for the complexity of the path planning problem. We can classify our



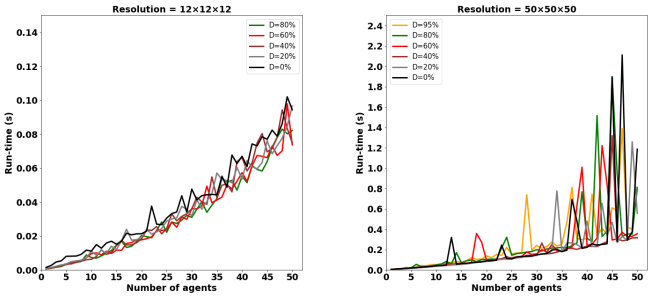
(a) Total cost

(b) Cost per agent



(c) Longest path

(d) Waiting events



(e) Run time

Fig. 4: Performance of EECBS in $12 \times 12 \times 12$ and $50 \times 50 \times 50$ environments with varying obstacle occupancy.

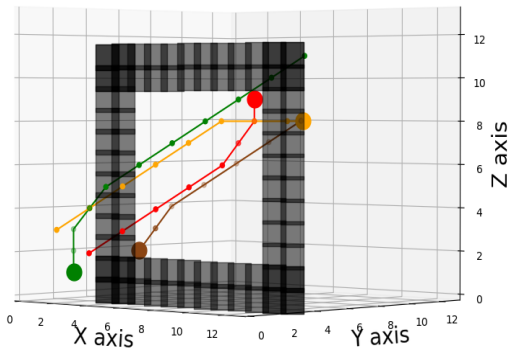
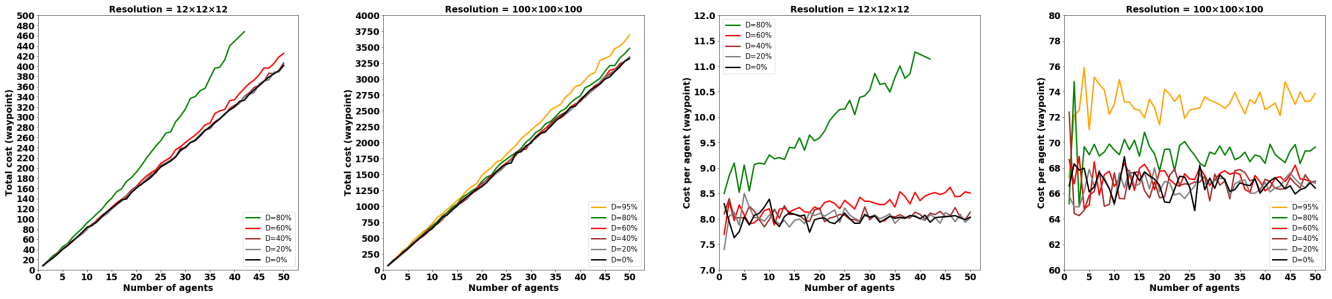


Fig. 5: Constellation change of 4 agents through a window.

scenarios into three categories. Scenarios with $r = 12$ and $D \in \{0\%, 20\%, 40\%, 60\%\}$ (Figure 6f left) can be solved in less than 0.25 s. Scenarios with $r = 100$ can be solved in less than 0.8 s for $D = 0\%$ but require up to 450 s for $D = 95\%$ (Figure 6f right). Finally, Figure 6f center depicts a dramatic increase of the run time when the success rate starts to drop. In this case, EECBS traverses many unsuccessfully many branches in the conflict tree resulting orders of magnitude larger run times.

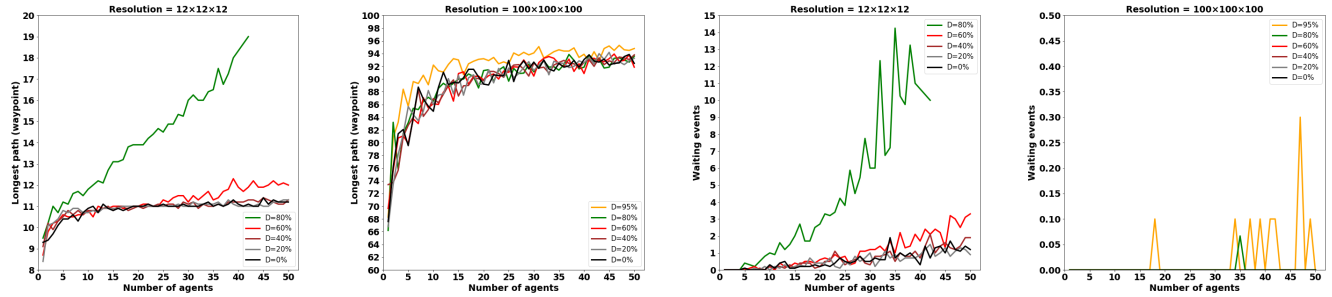
E. Example: Trajectory generation

Figure 7 depicts a constellation change example of 10 agents in an environment with resolution $r = 8$. The trajectories have been optimized based on Equation 3 and result in an overall transition time of 9.1 s. The trajectories fulfill all optimization constraints and allow collision-free,



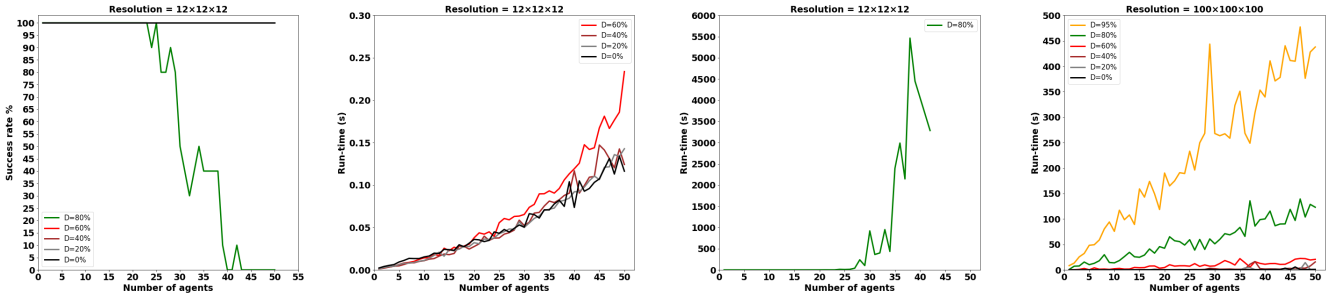
(a) Total cost

(b) Cost per agent



(c) Longest path

(d) Waiting events



(e) Success rate

(f) Run time

Fig. 6: Performance of EECBS in environments with resolution $r \in \{12, 100\}$ and varying window border size $D \in \{0\%, 20\%, 40\%, 60\%, 80\%, 95\%\}$.

synchronous movement of the ten agents. Figure 8 shows the required velocity and acceleration of an agent for traversing the trajectory. Both velocity and acceleration stay below the actuator limits of 1 m/s and 1 m/s^2, respectively.

VI. Conclusion

We present a multi-agent path planning and trajectory generation approach for constellation changes in environments with and without obstacles. As demonstrated in our simulation study, the proposed EECBS algorithm achieves improvements over the traditional ECBS algorithm. In obstacle-free environments, total cost, average cost, and longest path were reduced by approximately 50% in all considered resolutions. Furthermore, EECBS hardly introduces waitings to avoid collisions and computes the paths in shorter run times. The improvements of our EECBS algorithm, i.e., shorter path lengths, faster planning times, and reduced waitings of agents, become particularly present in environments with scarce space to move, i.e., in confined environments characterized by low resolution, high obstacle density, and many agents. These improvements are primarily due to the increased movement actions at each planning step as compared to the original ECBS algorithm. Consequentially, EECBS was able to find collision-free paths in almost all simulated scenarios—except for constellation changes of more than 40 agents through a window with $D = 80\%$ and $r = 12$. We finally demonstrate the trajectory generation by a snap- and time-optimization of the discrete, multi-agent paths.

We foresee several directions as future work. First, we deploy the generated trajectories on real UAVs, such as

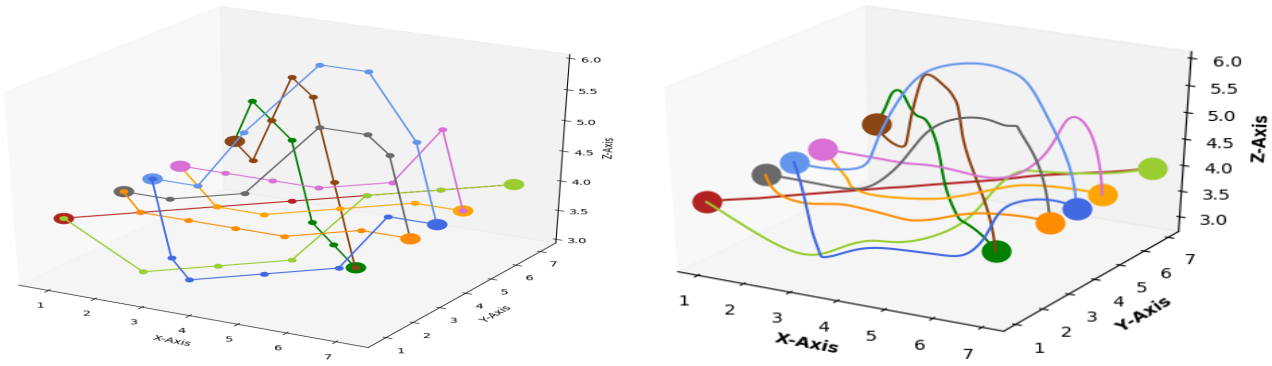


Fig. 7: Trajectory generation for a constellation change scenario. EECBS returns discrete paths for all agents (left). The waypoints of these paths serve as input for snap-optimized trajectory generation (right).

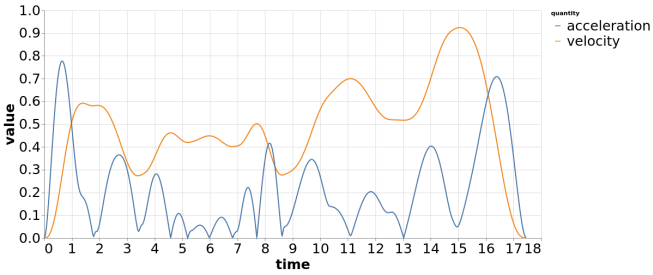


Fig. 8: Velocity and acceleration of an agent following its trajectory.

Crazyflie drone platforms, and compare the simulation results with real experiments. Second, we evaluate other graph-based planning algorithms for environments with scarce mobility space and perform more complex optimizations for the trajectory generation.

References

[1] R. Shakeri, M. A. Al-Garadi, A. Badawy, A. Mohamed, T. Khattab, A. K. Al-Ali, K. A. Harras, and M. Guizani, "Design challenges of multi-UAV systems in cyber-physical applications: A comprehensive survey and future directions," *IEEE Commun. Surveys & Tutorials*, vol. 21, no. 4, pp. 3340–3385, 2019.

[2] B. Rinner, C. Bettstetter, H. Hellwagner, and S. Weiss, "Multidrone systems: More than the sum of the parts," *IEEE Computer*, vol. 54, no. 5, pp. 34–43, 2021.

[3] P. Mazdin and B. Rinner, "Distributed and communication-aware coalition formation and task assignment in multi-robot systems," *IEEE Access*, vol. 9, pp. 35 088–35 100, 2021.

[4] A. Khan, B. Rinner, and A. Cavallaro, "Cooperative robots to observe moving targets: Review," *IEEE Transactions on Cybernetics*, vol. 48, no. 1, 2018.

[5] Y. Rizk, M. Awad, and E. W. Tunstel, "Cooperative heterogeneous multi-robot systems: A survey," *ACM Computing Surveys (CSUR)*, vol. 52, no. 2, pp. 1–31, 2019.

[6] C. Luis and A. Schoellig, "Trajectory generation for multiagent point-to-point transitions via distributed model predictive control," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 375–382, 2019.

[7] P. Ladinig, B. Rinner, and S. Weiss, "Time and energy optimized trajectory generation for multi-agent constellation changes," in *Proc. IEEE Intern. Conf. on Robotics and Automation (ICRA)*, 2021.

[8] M. Barer, G. Sharon, R. Stern, and A. Felner, "Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem," in *The International Symposium on Combinatorial Search (SOCS)*, 2014.

[9] R. K. Dewangan, A. Shukla, and W. W. Godfrey, "Survey on prioritized multi robot path planning," in *Proc. IEEE International Conference on Smart Technologies and Management for Computing, Communication, Controls, Energy and Materials (ICSTM)*, 2017, pp. 423–428.

[10] T. Schouwenaars, B. De Moor, E. Feron, and J. How, "Mixed integer programming for multi-vehicle path planning," in *Proc. European Control Conference (ECC)*, 2001, pp. 2603–2608.

[11] Y. Chen, M. Cutler, and J. How, "Decoupled Multiagent Path Planning via Incremental Sequential Convex Programming," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2015, pp. 5954–5961.

[12] F. Augugliaro, A. P. Schoellig, and R. D'Andrea, "Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach," in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 1917–1922.

[13] J. Park, J. Kim, I. Jang, and H. J. Kim, "Efficient multi-agent trajectory planning with feasibility guarantee using relative bernstein polynomial," in *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 434–440.

[14] D. R. Robinson, R. T. Mar, K. Estabridis, and G. Hewer, "An efficient algorithm for optimal trajectory generation for heterogeneous multi-agent systems in non-convex environments," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 1215–1222, 2018.

[15] D. Zhou, Z. Wang, S. Bandyopadhyay, and M. Schwager, "Fast, online collision avoidance for dynamic vehicles using buffered voronoi cells," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 1047–1054, 2017.

[16] M. Jager and B. Nebel, "Decentralized collision avoidance, deadlock detection, and deadlock resolution for multiple mobile robots," in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No.01CH37180)*, vol. 3, 2001, pp. 1213–1219 vol.3.

[17] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," in *Proc. International Symposium of Robotics Research*, 2016, pp. 649–666.

[18] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *Proc. IEEE International Conference on Robotics and Automation*, 2011, pp. 2520–2525.

[19] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artificial Intelligence*, vol. 219, pp. 40–66, 2015.

[20] T. Huang, B. Dilkina, and S. Koenig, "Learning node-selection strategies in bounded-suboptimal conflict-based search for multi-agent path finding," in *Proc. International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2021, pp. 611–619.

[21] W. Hönig, "UAV trajectories," https://github.com/whoenig/uav_trajectories, 2019.

[22] W. Hönig, K. Vedder, and B. Şenbaşlar, "libMultiRobotPlanning," https://github.com/whoenig/libMultiRobotPlanning/tree/79678825103cf944979d5d83657770d83b907afa, 2019.

[23] J. Park and J. T. Torres, "swarm simulator," https://github.com/qwerty35/swarm_simulator, 2020.