2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 14-18, 2024. Abu Dhabi, UAE

Priority-Based Deadlock Recovery for Distributed Swarm Obstacle Avoidance in Cluttered Environments

Jiacheng He, Fangguo Zhao, Shaohao Zhu, Shuo Li and Jinming Xu

***Abstract*— We propose a novel hierarchical priority mechanism for deadlock recovery of distributed swarm via on-demand collision avoidance in cluttered dynamic environments. The proposed priority mechanism dynamically assigns certain priority and an optimized detour point for each agent based on its spatial context to avoid deadlocks which are predicted by properly designed deadlock conditions; as a byproduct, this priority mechanism allows us to effectively resolve livelocks as well. The resulting optimization problem is then solved by polar reformulation and alternating minimization methods. Simulation results demonstrate that, in both static and dynamic environments, our method (termed PriDRAM) outperforms the baseline Alternating Minimization Swarm (AMSwarm) method which does not explicitly account for deadlock recovery, with a 10.5% improvement in average smoothness and a 4.8% reduction in flight time. Moreover, for narrow passages, our method shows a superior performance against the Distributed Linear Safe Corridor (DLSC) method, with a more reasonable passing order and an achievement of up to 40% reduction in flight path length. Finally, we verify the efficacy of our proposed method with a Crazyflie 2.1 quadrotor swarm.**

I. Introduction

Distributed quadrotor swarms have enormous potential applications in various areas, such as mapping [1], cargo delivery [2], search and rescue [3], surveillance [4]. Real-world applications often require multi-quadrotors to perform tasks in cluttered wild and dense urban areas, which underscores the importance of a reliable obstacle avoidance method for enhancing the adaptability of quadrotor swarms.

For distributed swarm obstacle avoidance, several works have been proposed that explicitly formulate the free space navigation as a low-dimensional linear problem [5], including Optimal Reciprocal Collision Avoidance (ORCA) [6], Buffered Voronoi Cells (BVC) [7], Relative Safe Flight Corridor (RSFC) [8] [9]. However, all of these approaches impose hyperplane constraints that will restrain the feasible solution space for optimizing future inputs [10], leading to conservative motion performance or even failure to find feasible solutions in complex environments.

To address the above issue, on-demand collision avoidance methods based on a predict-avoid paradigm have been adopted for point-to-point navigation [10] [11], which incorporate non-convex obstacle avoidance constraints directly into the optimization problem. To deal with the non-convexity of obstacle constraints, Sequential Convex Programming (SCP) [12] [13] is employed to linearize them, which leads to reduced flight time and increased aggressiveness for distributed drone swarm. Recently, an AMSwarm [14] method is proposed to solve the above problem based on polar reformulation and Alternating Minimization (AM) which further reduces computation time and achieves better motion performance. However, these above methods still suffer from an 'oscillation' problem, where robots synchronously jiggle their avoiding direction [15].

The occurrence of deadlock, where robots block each other during motion in a narrowed space, has become a significant issue in distributed obstacle avoidance [16] [17]. Numerous studies are conducted that employ simple heuristic rules to address deadlocks. For example, Pierson et al. [18] use the right-hand rule to direct agents to the right during deadlocks based on the BVC framework. While effective in free spaces, this method can be hindered by static obstacles. In [9], a priority planning was adopted to sequentially calculate each agent's trajectory, which may incur extra computation time and become unfair in allocation of priorities. In [19], a detour point technique is employed that manually specifies subgoals to resolve deadlocks, which may loose generalization. There have been other attempts that integrate the above techniques [20] [21]. For instance, in [20], the authors propose a Distributed Linear Safe Corridor (DLSC) method for collision avoidance in dynamic maze-like environments, which utilizes right-hand rule to address deadlock initially; upon failure, mode-based subgoal planning will be then employed for deadlock resolution. Additionally, an optimization-based technique is recently proposed in [22] to achieve deadlock recovery under the BVC framework, which can implicitly assign certain priority for each agent.

In this paper, we propose a new deadlock recovery framework integrated with a novel hierarchical priority mechanism based on on-demand collision avoidance, which only requires the predicted trajectory positions and goal points of neighbors, as the same spirit in [20] [23]. Note that, different from BVC-based approaches, predicting and recovering from deadlocks poses great challenges due to the absence of explicit geometric feasible region representation, especially for on-demand collision avoidance-based methods.

The contributions of this paper are as follows:

1) We propose a novel hierarchical priority mechanism that dynamically allocates priorities and optimized detour points to agents, achieving up to a 40% reduction in flight path length compared to the state-of-the-art DLSC method [20]. Moreover, the proposed method

The authors are with the College of Control Science and Engineering, Zhejiang University, Hangzhou 310027, China. Correspondence to jimmyxu@zju.edu.cn (Jinming Xu). This work was supported in parts by NSFC under Grants 62088101, 62373323, 62203385 and in parts by the Key Laboratory of Collaborative Sensing and Autonomous Unmanned Systems (Key Lab of CS&AUS) of Zhejiang Province. Code and videos can be found at https://github.com/JiachengHekdk/PriDRAM

979-8-3503-7769-9/24/$31.00 ©2024 IEEE

can effectively overcome livelocks as well.
2) We develop proper deadlock prediction conditions tailored for on-demand collision avoidance, which can accurately identify deadlocks.
3) We conduct extensive simulations and real-world experiments that demonstrate excellent trajectory planning in both random forests and narrow passages.

II. Preliminaries

A. On-Demand Collision Avoidance

The on-demand method relies on a predict-and-avoid paradigm for collision avoidance [11]. It constrains only a specific sample of the trajectory curve if agent i detects a predicted collision based on the future actions shared among communicative agents. The obstacle avoidance inequality constraint for the sample is as follows:

$$\|\mathbf{\Theta}^{-1}(\mathbf{p}_i[k]-\mathbf{p}_j[k]\| \geq r_{\min}, \forall k, \forall j \in \mathcal{N}_i, \tag{1}$$

where $\mathbf{p}_i(k)$ is the position of agent i at time step k. $\mathbf{\Theta}$ is a diagonal matrix with elements (a, b, c) to mitigate the impact of downwash airflow between quadrotors. $r_{\min}$ is the minimum safe distance between agents.

B. Polar Reformulation and Alternating Minimization

To convert the QCQP trajectory optimization problem into a QP problem, we utilize polar reformulation introduced in [14] [24] for collision avoidance constraints.

For such an inequality constraint $\|\mathbf{M}(\mathbf{v_i}-\mathbf{v_j})\| \geq r$, where $\mathbf{M}$ is a diagonal positive definite matrix, this inequality constraint can be equivalently represented as $\mathbf{M}(\mathbf{v_i}-\mathbf{v_j}) - \mathbf{d}_{ij}[\cos\boldsymbol{\alpha}\sin\boldsymbol{\beta}, \sin\boldsymbol{\alpha}\cos\boldsymbol{\beta}, \cos\boldsymbol{\beta}]^T$ with $d_{ij} \geq r$, where $\boldsymbol{\alpha}$ is the azimuthal angle, $\boldsymbol{\beta}$ is the polar angle and $\mathbf{d}_{ij}$ is the magnitude of the vector from $\mathbf{v_j}$ to $\mathbf{v_i}$. Using the polar reformulation, we can rewrite the collision avoidance constraints (1) as:

$$\mathbf{\Theta}^{-1}(\mathbf{p}_i[k]-\mathbf{p}_j[k]) - \mathbf{d}_{ij}\omega(\boldsymbol{\alpha},\boldsymbol{\beta}) = 0, d_{ij} \geq r_{\min}, \tag{2}$$

which is convex concerning $\mathbf{p}_i$. The transformed multi-variable optimization problem can be converted into an unconstrained optimization problem with implicit convex structure using a method close to the Bregman iteration method [25]. Subsequently, the alternating minimization method is employed to optimize each variable to obtain the optimal solution sequentially. This method significantly accelerates the computation time per agent.

III. Problem Formulation

We consider a swarm composed of N agents labeled by $i \in \mathcal{V} = \{1, 2..., N\}$ and a convex obstacle space $\mathcal{O}$ consisting of a static obstacle set denoted as $\mathcal{M}_s$ and a dynamic obstacles set denoted as $\mathcal{M}_d$. Our goal is to generate a collision-free and dynamically feasible point-to-point trajectory for each drone in a cluttered environment, with the starting point denoted as $\mathbf{P}_{i,s}$, and the endpoint as $\mathbf{p}_{i,g}$. We aim to resolve deadlock and livelock issues. Similar to [10] [14], we formulate a distributed trajectory optimization problem based on receding horizon fashion.

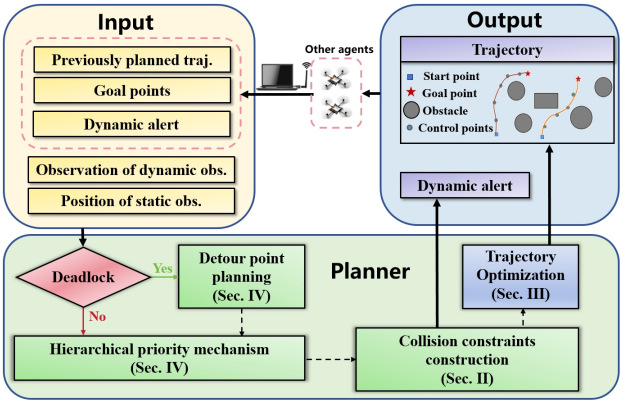


Fig. 1: The proposed deadlock recovery framework. Dashed lines and solid lines represent inter-module and intra-module information transmission respectively.

At each time step, quadrotors with known linear dynamics exchange the planned trajectory computed from the previous time step and re-optimize ego trajectories until all drones reach their goal points without collision. **Fig. 1** shows the framework of our algorithm.

A. Trajectory Parameterization

Each quadrotor's trajectory is parameterized using Bernstein polynomials of degree n denoted as $\mathbf{p}_i : [0, T] \rightarrow \mathbb{R}^d$ [26], this implies the existence of n+1 control points $\mathbf{c}_i = [c_i^0 ..., c_0^n] \in \mathbb{R}^d$, where d represents the dimension of space. The x-position trajectory of the i-th quadrotor can be represented as:

$$p_{i,x}(t) = \sum_{m=0}^{n} c_{i,x}^m \binom{n}{m} \left(\tfrac{t}{T}\right)^m \left(1-\tfrac{t}{T}\right)^{n-m}. \tag{3}$$

We can obtain a discrete trajectory representation by discretizing time, dividing T into K instants, each with a size of δt, i.e, $T = (K-1)\delta t$. The linear relationship between the trajectory and control points can be represented as $\mathbf{p}_i[t] = \mathbf{G}[t]\mathbf{c}_i$, where $\mathbf{G}$ is a pre-defined matrix in terms of t. According to the properties of Bernstein curves [27], the derivative of the trajectory can also be expressed using the initial control points of the Bernstein curve.

B. Model of Flying Agent

We assume each agent is equipped with an underlying position controller that follows a discrete linear system:

$$\mathbf{x}_i[k+1] = \mathbf{A}\mathbf{x}_i[k] + \mathbf{B}\mathbf{u}_i[k], \tag{4}$$

where $\mathbf{x} = [\mathbf{p}, \mathbf{v}] \in \mathbb{R}^6$ are the position and velocity of quadrotors and the inputs $\mathbf{u} \in \mathbb{R}^3$ is position reference signal. To prevent agents from exceeding dynamic limits, we impose the following constraints:

$$\mathbf{u}_{\min}^{(h)} \leq \mathbf{u}_i^{(h)}[k] \leq \mathbf{u}_{\max}^{(h)} \quad h = 0, 1, 2, \tag{5}$$

where h represents the h-th deviation of input.

C. Dynamic Obstacle

Regarding dynamic obstacles $j \in \mathcal{M}_d$, each agent can observe them within a limited range and predict their future trajectories as follows:

$$\hat{\mathbf{p}}_j[k+1] = \bar{\mathbf{p}}_j[k] + \delta t \cdot \bar{\mathbf{v}}_j[k], \tag{6}$$

where $\hat{\mathbf{p}}_j$ represents the predicted trajectory of dynamic obstacles, $\bar{\mathbf{p}}$ and $\bar{\mathbf{v}}$ respectively denote the observed position and observed velocity of dynamic obstacles.

D. Formulating as an Optimization Problem

At each planning step, quadrotor i locally solves an optimization problem, formulated as follows:

$$\min_{\mathbf{p}_i} \quad \sum_{k=K-\pi}^{K-1} \|\mathbf{p}_i[k] - \mathbf{p}_{i,g}\|^2_{\mathbf{R}_1} + \sum_{k=0}^{K-1} \|\mathbf{p}_i^{(h)}[k]\|^2_{\mathbf{R}_2} \tag{7a}$$

$$\text{s.t} \quad \mathbf{p}_i^{(h)}[0] = \mathbf{P}_{i,s}^{(h)}, \quad \forall h \in \{1,2,3\} \tag{7b}$$

$$\mathbf{p}_{\min}^{(h)} \le \mathbf{p}_i^{(h)}[k] \le \mathbf{p}_{\max}^{(h)}, \forall k, \forall h \in \{1,2,3\} \tag{7c}$$

$$\|\mathbf{\Theta}^{-1}(\mathbf{p}_i[k] - \mathbf{p}_j[k]\| \ge r_{\min}, \forall k, \forall j \in \mathcal{N}_i \cap \mathcal{P}_i, \tag{7d}$$

where k is the discrete-time index, K is the planning horizon length, $\|\cdot\|$ denotes the Euclidean norm. We note that in the initial constraint (7b) we use a capital $\mathbf{P}_{i,s}$ to distinguish the starting point $\mathbf{P}_{i,s}$ because the initial conditions need to be updated at each planning step.

The cost function consists of two terms. The first term is to orient the agent to the goal point and we only penalize the last π steps in a prediction horizon. The second term is to minimize the energy required by the control commands. $\mathbf{R}_1$ and $\mathbf{R}_2$ are the weights of the two terms respectively. For convenience, we use $\mathbf{p}_j[k]$ to represent both neighboring agents and obstacles simultaneously. An on-demand obstacle avoidance approach was employed to enable less conservative maneuvers. Additionally, we have predefined a priority set $\mathcal{P}_i$ to prevent the occurrence of livelock. The detailed explanation can be found in Section IV-A.

E. AM-based Optimization

In Section II-B, we discussed the polar reformulation method, which can transform constraints (7d) into a convex constraint concerning the optimization variable $\mathbf{p}_i$. According to Section III-A, the linear relationship between the trajectory and control points can be represented with a matrix $\mathbf{G}$. The reformulated optimization problem thus becomes:

$$\min_{\mathbf{c}_i, \boldsymbol{\alpha}_i, \boldsymbol{\beta}_i, \mathbf{d}_i} \quad \frac{1}{2}\mathbf{c}_i^T \mathbf{Q} \mathbf{c}_i + \mathbf{q}^T \mathbf{c}_i \tag{8a}$$

$$\text{s.t} \quad \mathbf{A}_{\text{ineq}} \mathbf{c}_i \le \mathbf{b}_{\text{ineq}} \tag{8b}$$

$$\mathbf{A}_{\text{eq}} \mathbf{c}_i = \mathbf{b}_{\text{eq}}(\boldsymbol{\alpha}_i, \boldsymbol{\beta}_i, \mathbf{d}_i) \tag{8c}$$

$$\mathbf{c}_i \in \mathcal{C}_1, \tag{8d}$$

$$\boldsymbol{\alpha}_i \in \mathcal{C}_2, \boldsymbol{\beta}_i \in \mathcal{C}_3, \mathbf{d}_i \in \mathcal{C}_4, \tag{8e}$$

where $\mathbf{Q} = \mathbf{G}^T \mathbf{R}_1 \mathbf{G} + \mathbf{G}^{(h)T} \mathbf{R}_2 \mathbf{G}^{(h)}$, the vector $\mathbf{q} = \mathbf{G}^T \mathbf{p}_{i,g}$. The initial condition set $\mathcal{C}_1$ denoted in (8d) indicates that the first h control points are determined by the current state. The set constraints $\mathcal{C}_2$, $\mathcal{C}_3$, $\mathcal{C}_4$ defined in (8e) restrict parameters in polar coordinates. Then we relax affine constraint (8b) and non-convex equality (8c) as penalties as in [25]:

$$\begin{aligned} \min_{\mathbf{c}_i \in \mathcal{C}_1, \boldsymbol{\alpha}_i \in \mathcal{C}_2, \boldsymbol{\beta}_i \in \mathcal{C}_3, \mathbf{d}_i \in \mathcal{C}_4} \quad & \frac{1}{2}\mathbf{c}_i^T \mathbf{Q} \mathbf{c}_i + \mathbf{q}^T \mathbf{c}_i - \boldsymbol{\lambda}_i^T \mathbf{c}_i + \\ & \frac{\rho}{2}\|\mathbf{A}_{\text{ineq}} \mathbf{c}_i - \mathbf{b}_{\text{ineq}} + \mathbf{s}_i\|^2 + \\ & \frac{\rho}{2}\|\mathbf{A}_{\text{eq}} \mathbf{c}_i - \mathbf{b}_{\text{eq}}(\boldsymbol{\alpha}_i, \boldsymbol{\beta}_i, \mathbf{d}_i)\|^2, \end{aligned} \tag{9}$$

where $\mathbf{s}_i$ serves as a slack variable to relax the inequality constraint. After problem reformulation, we can employ the AM method to solve (8a-8e). This transforms each subproblem into a readily solvable QP form [14].

IV. Priority-based obstacle avoidance

In this section, our attention is directed towards the design of obstacle avoidance schemes. A well-designed obstacle avoidance strategy is pivotal in addressing deadlock and livelock, significantly impacting the safety and computation time of trajectory planning. We propose a priority-based obstacle avoidance method integrating with optimization-based detour point to address the above issues.

A. Design of Priority Level

We construct a priority queue by considering the environmental complexity and the distance of each quadrotor to the goal point, aiming to prevent the livelock in symmetric scenarios and deadlock in narrowed space (c.f., **Fig. 2**).

- **Level 0**: This level indicates that it has already reached the vicinity of the goal point, i.e,$\|\mathbf{p}_i[0] - \mathbf{p}_{i,g}\| \le d_g$. To prevent it from blocking the flight trajectories of other agents, it will be assigned the lowest priority.
- **Level 1**: This level indicates that the agent is in a normal flying state. In this state, the agent will locally determine the priority order through a priority function:

$$f(i) = \mathcal{J}_{i,c} + \frac{d_g}{\|\mathbf{p}_i[0] - \mathbf{p}_{i,g}\|} \tag{10}$$

$$\mathcal{J}_{i,c} = \sum_{j \in \mathcal{N}_i} e^{(d_g - \|\mathbf{\Theta}^{-1}(\mathbf{p}_i - \mathbf{p}_j)\|)}, \tag{11}$$

  where $f_i(x)$ is the priority value of i-th quadrotor, d_g is the goal threshold, and $\mathcal{J}_{i,c}$ is an exponential function enables the agent i to "pretend" to perceive the complexity of the surrounding environment. Additionally, the distance between the agent and the goal point is added to the priority function, following the same principles as in [20] [16]. To prevent overly complex obstacle avoidance constraints for certain agents, we do not employ a priority-based obstacle avoidance method for agents flying in the same direction:

$$\textbf{sign}((\mathbf{p}_{i,g} - \mathbf{p}_i[0]) \cdot (\mathbf{p}_{j,g} - \mathbf{p}_j[0])) \ge 0, \tag{12}$$

- **Level 2**: This level indicates that agents observe nearby dynamic obstacles that require emergency obstacle avoidance action:

$$\|\mathbf{p}_i[0] - \hat{\mathbf{p}}_j[0]\| \le d_e, \quad j \in \mathcal{M}_d, \tag{13}$$

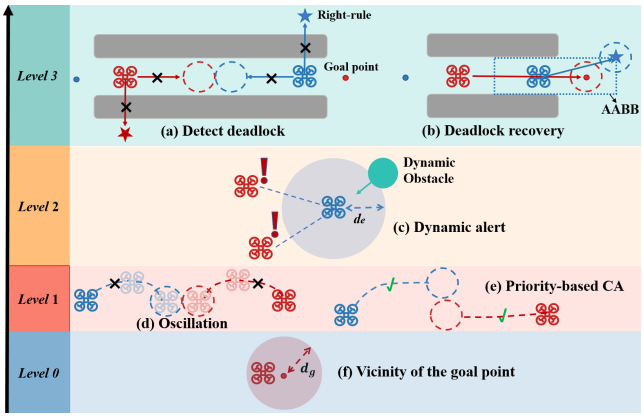


Fig. 2: Illustration of priority levels. (a) The agent detects a deadlock, and due to the presence of static obstacles, the right-hand rule cannot be applied. (b) The agent moves away from the nearest high-priority agent to achieve deadlock recovery. (c) Agent near dynamic obstacles sends alerts to its neighbors. (d) The agents experience trajectory oscillation (livelock) in a symmetrical environment without priority. (e) The agent is in a normal flight state using priority-based collision avoidance. (f) The agent has reached the vicinity of the goal point.

considering that agents might demonstrate uncontrollable emergency avoidance strategies upon detecting dynamic obstacles, we assign them a higher priority.

- **Level 3**: This level indicates that agents are in a state of deadlock recovery. When the target position of agent i differs from its initial goal point, it can be considered that agent i has encountered a deadlock (c.f., Section IV-B). Therefore, other agents can locally observe this through previously shared information.

Following the above definition of priority levels, we can define a high-priority set for agent i as follows:

$$\mathcal{P}_i = \begin{cases} j & \text{if Level}(i) < \text{Level}(j) \\ j | f(j) > f(i) & \text{if Level}(i) = \text{Level}(j). \end{cases} \tag{14}$$

B. Deadlock Recovery

As with [20] [21], some trajectory optimization methods based on BVC obstacle avoidance can predict and detect deadlock through the explicitly defined feasible region. However, for our method via on-demand collision avoidance, predicting and detecting deadlock locally is challenging. Therefore, to accurately identify deadlocks, we design proper deadlock prediction conditions as below:

1) $\|\mathbf{v}_i[0]\| \leq v_{th}$,
2) $\|\mathbf{p}_i[0] - \mathbf{p}_{i,g}\| > d_g$,
3) $\mathbf{p}_i[K] \cap \mathcal{O} \cap \mathcal{N}_i \neq \emptyset$,

where v_{th} is the velocity threshold to be properly determined according to a certain context.

When a deadlock is detected, we first attempt to resolve it using the right-hand rule. However, the right-hand rule method cannot guarantee deadlock resolution in environments with static obstacles [22]. As a result, we propose a subgoal planning technique that integrates the priority-based approach with the detour point method.

In Section IV-A, we have defined methods to establish priorities between different levels and within levels. Therefore, when a deadlock occurs for each agent, we can drive the ego agent to move away from the closest high-priority agent denoted as jc. To determine a proper detour point, we solve the following optimization problem:

$$\max_{\mathbf{sg}_i} \quad \|\mathbf{\Theta}^{-1}(\mathbf{sg}_i - \mathbf{p}_{jc}[0])\| + \mathbf{\Theta}^{-1}(\mathbf{sg}_i - \mathbf{p}_{jc}[0]) \cdot \mathbf{n}_{ijc} \tag{15a}$$

$$\text{s.t} \quad \mathbf{sg}_i \in \mathcal{F}_i \tag{15b}$$

$$\|\mathbf{sg}_i - \mathbf{p}_i[0]\| \leq d_c \tag{15c}$$

$$\mathbf{p}_{\min} \leq \mathbf{sg}_i \leq \mathbf{p}_{\max}, \tag{15d}$$

where $\mathbf{sg}_i$ is the detour point of agent i. The first term of the objective function ensures the detour point be away from the nearest high-priority agent, while the second term guarantees lateral distancing from it. $\mathcal{F}_i$ denotes the Axis-Aligned Bounding Box (AABB) of the agent i [28]. d_c is a tunable parameter avoiding the overshooting of subgoals. If a subgoal cannot be found, the agent will remain stationary. $\mathbf{n}_{ijc}$ is a direction vector from agent i to agent jc, which is defined as follows:

$$\mathbf{n}_{ijc} = \frac{\mathbf{\Theta}^{-1}(\mathbf{p}_i[0] - \mathbf{p}_{jc}[0])}{\|\mathbf{\Theta}^{-1}(\mathbf{p}_i[0] - \mathbf{p}_{jc}[0])\|}. \tag{16}$$

C. Priority-based Obstacle Avoidance

We propose a priority-based obstacle avoidance method that incorporates the deadlock recovery approach in Section IV-B, outlined in **Algorithm 1**. As mentioned, the algorithm is designed to be executed in a distributed manner by a group of agents with communication capabilities.

In **lines 1-2** we assume all agents are at Level 1, and compute the priority values for all agents locally using (10). Then, in **lines 3-4**, check if any agents have reached the vicinity of the goal point, and if so, set their priority level to zero. Subsequently, **lines 5-9** are used to update obstacle constraints by checking if the predicted trajectory would collide with obstacles. This updates the current step's neighboring obstacle sets $\mathcal{M}_s$ and $\mathcal{M}_d$. When the dynamic obstacle set $\mathcal{M}_d$ is not empty, the current priority is updated to Level 2 and broadcast alert messages **isDynamic()** to surrounding agents. Afterward, in **lines 10-15**, we predict deadlocks and employ the deadlock recovery approach using detour points. Finally, by previously shared information among agents, we update the priority levels of other agents to obtain the high-priority set $\mathcal{P}_i$, thus updating neighboring agents $\mathcal{N}_i$ as inter-obstacle constraints. Note that the proposed deadlock recovery method can not guarantee solving deadlocks in arbitrary situations, as the same spirits in [7] [20].

V. Experimental Results

In this section, we present simulation and real-world experiment results of our method PriDRAM and compare it with the state-of-the-art baselines in two classic cluttered environments, the first is a random forest in which our algorithm is compared against the SCP [11] and AMswarm

TABLE I: Performance Comparision in Random Forests

Metric	Method	The number of agents(No dyn obs.)			The number of agents		
		12	24	36	12	24	36
Success rate(%)	SCP	87	76	45	-	-	-
	AMswarm	**100**	**89**	**84**	**100**	**83**	**61**
	PriDRAM (ours)	**100**	84	75	**100**	79	57
Averge smoothness(ms^{-2})	SCP	3.54 ± 0.06	3.92 ± 0.13	4.51 ± 0.02	-	-	-
	AMswarm	3.46 ± 0.18	4.17 ± 0.09	4.78 ± 0.17	3.44 ± 0.12	4.52 ± 0.12	4.87 ± 0.10
	PriDRAM (ours)	**2.90 ± 0.02**	**3.86 ± 0.04**	**4.28 ± 0.09**	**3.16 ± 0.05**	**4.18 ± 0.05**	**4.56 ± 0.07**
Runtime per agent(ms)	SCP	2.03 ± 0.02	3.34 ± 0.08	6.12 ± 0.59	-	-	-
	AMswarm	0.61 ± 0.01	0.56 ± 0.02	0.67 ± 0.01	0.72 ± 0.03	0.60 ± 0.02	**0.65 ± 0.06**
	PriDRAM (ours)	**0.59 ± 0.05**	**0.54 ± 0.03**	**0.61 ± 0.08**	**0.63 ± 0.06**	**0.58 ± 0.04**	0.68 ± 0.01
Flight time(s)	SCP	5.59 ± 0.08	6.78 ± 0.02	6.92 ± 0.03	-	-	-
	AMswarm	5.24 ± 0.23	5.59 ± 0.13	6.01 ± 0.32	5.15 ± 0.21	**6.44 ± 0.26**	6.76 ± 0.48
	PriDRAM (ours)	**4.67 ± 0.03**	**5.48** ± 0.19	**5.89 ± 0.33**	**5.01 ± 0.23**	6.47 ± 0.16	**6.65 ± 0.15**

The Flight Time and Average Smoothness values are extracted from successful trials in 100 simulations. The number following ± represents the standard deviation, and the bold number highlights the best results.

Algorithm 1: The Priority-based Obstacle Avoidance

Initialization: Priority ← Level 1;
$\mathcal{P}_i \leftarrow$ Solve (10) ;
if *($\|\boldsymbol{p}_i[0] - \boldsymbol{p}_{i,g}\| \leq d_g$)* **then**
  Set Priority ← Level 0;
$\mathcal{M}_s \leftarrow InitializeObstacle()$;
$\mathcal{M}_d \leftarrow InitializeDynamicObstacle()$;
if *($\mathcal{M}_d \neq \emptyset$)* **then**
  Broadcast ***isDynamic()***;
  Set Priority ← Level 2;
if *Dectect* ***isDeadlock()*** **then**
  if ***isRightrule()*** **then**
    $\mathbf{sg}_i \leftarrow \mathbf{p}_i[0] + \frac{\mathbf{p}_{i,g} - \mathbf{p}_i[0]}{\|\mathbf{p}_{i,g} - \mathbf{p}_i[0]\|} \times \mathbf{z}$;
  else
    $\mathbf{sg}_i \leftarrow$ Solve (15a);
  Set Priority ← Level 3;
Receive ***isDynamic()*** from j;
Receive ***isDeadlock()*** from j;
$\mathcal{P}_i \leftarrow$ Solve (14) ;
$\mathcal{N}_i \leftarrow InitializeInterObstacle(\mathcal{P}_i)$;

[14] methods, the second is a narrow passage for comparison with the DLSC [20] method. All simulations are performed in an open-source C++ multi-robot simulation[1], with results visualized in Matlab R2020b. All experiments were conducted on a laptop with an Intel Core i7-12700H CPU at 2.70 GHz and 16 GB of RAM.

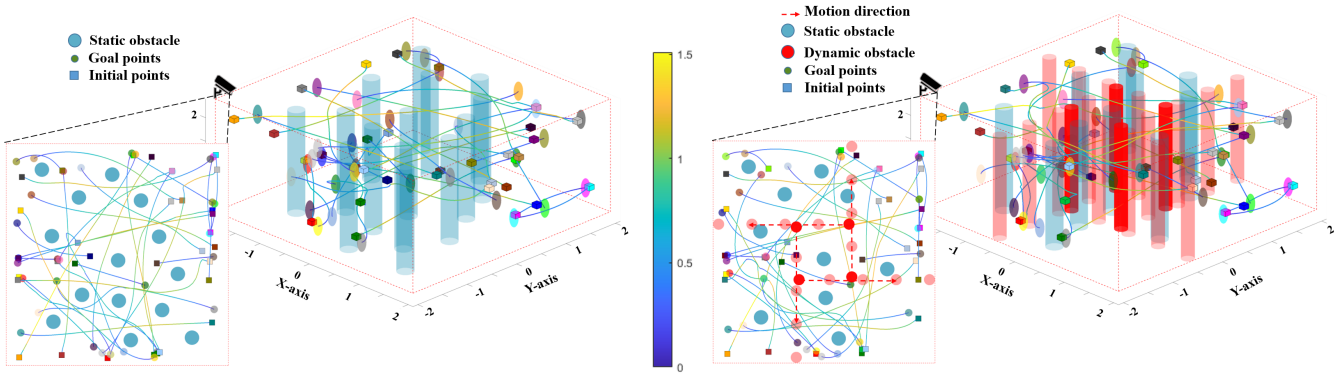
We set the collision model of the agent with $r_{\min} = 0.15m$ and the scaling matrix $\mathbf{\Theta} = diag([1, 1, 2])$ based on the specification of Crazyflie2.1. The maximum velocity and acceleration of the agents are $v_{\max} = 1.5m/s$, $a_{\max} = 2.8m/s^2$, respectively. We used a Bernstein polynomial with the degree $n = 10$, the prediction horizon $K = 30$, and the planning horizon of the proposed algorithm is 3s. Therefore, the discretization of time intervals $\delta t = 0.1s$. Our replanning period is 0.1s. Dynamic obstacles can be detected within a 1m range of the agent, and triggering priority Level 3 when the distance between the dynamic obstacle and the agent is less than 0.6m, i.e., $d_e = 0.6m$. The goal threshold $d_g = 0.2m$. Readers are referred to [14] for additional parameters associated with the optimization process.

To evaluate the performance of the proposed method in deadlock situations, we designed three scenarios as follows: i) A cluttered environment with 10 cylindrical static obstacles and 4 dynamic obstacles; ii) a second scenario featuring 14 cylindrical static obstacles within a volume of $4m \times 4m \times 2m$; and iii) a third scenario with a narrow passage constructed with static and dynamic obstacles.

A. Random forests

We conducted 100 trials for each swarm size $N \in [12, 24, 36]$, randomly deploying the start and goal points. A trial was deemed successful if all agents reached their targets within 20 seconds. The simulation results, as illustrated in **Fig. 3** and detailed in **Table I**, demonstrate the performance in random forests. Our method reduces the runtime per agent by 4.8% compared to AMSwarm and by 80.4% compared to SCP. Furthermore, our approach improves average smoothness by about 10.5% over AMSwarm. This enhancement stems from our priority-based strategy, which reduces trajectory oscillation i.e., the livelock issue in symmetrical environments. In terms of flight time, our method shows an average reduction of 4.8% compared to AMSwarm and 17.1% compared to SCP. The above results show that our proposed method has comparable performance with AMswarm in deadlock-free environments, which indicates that our method will not lead to deadlock misjudgments or compromise motion performance. However, since the priority-based approach will increase the local obstacle avoidance difficulty for agents with low priority, our method has a lower success rate than AMSwarm.

[1]https://github.com/utiasDSL/AMSwarm



(a) 36 agents with 14 static obstacles

(b) 36 agents with 10 static obstacles, 4 dynamic obstacles

Fig. 3: Swarm trajectory in random forests with (a) static and (b) dynamic obstacles. The bottom-left image provides a top-down view. Light blue cylinders represent static obstacles, while red cylinders denote dynamic obstacles. Dashed arrow lines indicate the motion direction of the dynamic obstacles. Ellipsoids represent the goal positions of drones, and cubes denote their starting positions.

B. Narrow passage

To evaluate the performance of our proposed algorithm in deadlock resolution, we set up a narrow passage in a 6m×3m×2m environment. Additionally, in this experiment, we use a 2D perspective to validate our method. Therefore, we represent quadrotors as compressed disks. The width of the passage is insufficient to allow two agents to pass side by side, as shown in **Fig. 4**. We tested our algorithm in both static and dynamic environments. In the dynamic scenario, the dynamic obstacle has an initial position of $[3m, 0m]$, a velocity of $1m/s$ along the negative x-axis direction.

TABLE II: Performance Comparision in Static Narrow Passage

Metric	Method	Number of agents 2	5	8
D_f(m)	DLSC	3.58	6.04	7.64
	PriDRAM (ours)	**3.49**	**3.63**	**3.99**
T_c (ms)	DLSC	**5.7**	**6.6**	**6.8**
	PriDRAM (ours)	10.0	11.5	12.9

T_c is computation time per agent. D_f is the average fight distance, and the bold number highlights the best results.

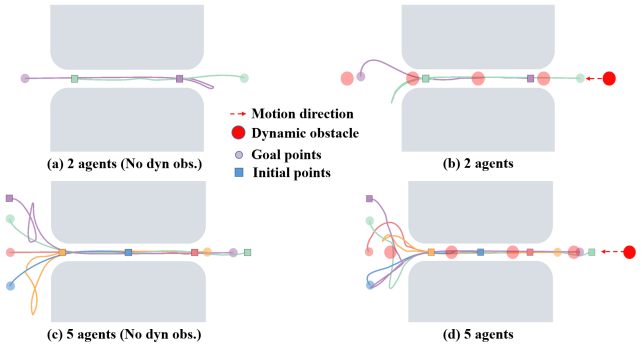


Fig. 4: Swarm trajectory in a narrow passage. Gray geometric shapes represent static obstacles, squares represent the starting points of drones, small circles represent the termination positions of drones, and red large circles represent dynamic obstacles.

In **Table II**, it is observed that our method outperforms DLSC in terms of the average flight distance of agents in the 5 agents and 8 agents scenario, with an improvement of around 40%. After comparing the flight trajectories of these two methods, we found that our method, by introducing priorities, has a more reasonable passing order in narrow passage environments (c.f., **Fig. 5**), avoiding unnecessary flight actions. There were no collision incidents during all simulations, and all agents satisfied the dynamic constraints.

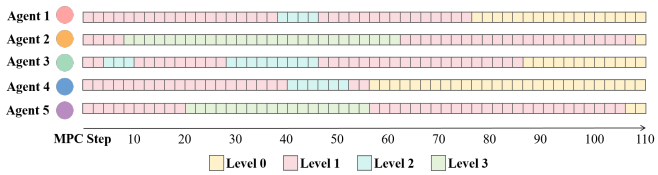


Fig. 5: Priority levels graph. Priority change graph for the movement of five drones in a narrow passage with dynamic obstacles (Fig. 4d). Each square represents two MPC steps.

C. Real-World Experiments

We conduct real-world experiments using a Crazyflie 2.1 swarm within a confined passage measuring $4m \times 1m$. The trajectory calculations were performed on a laptop, while position commands are to be sent to the onboard position controller [29]. The real-time position of each quadrotor was obtained by an Optitrack motion capture system.

We test the proposed algorithm in a narrow passage environment, where the quadrotors could only move in the x-y plane. **Fig. 6a-c** show that a group of 2 to 4 quadrotors can successfully navigate through a static narrow passage without collisions while preserving a reasonable passing order. Besides, one quadrotor was designated as a moving obstacle with an initial position at [2.5m, 0m, 0m] and a velocity of 1m/s along the negative x-axis, as shown by the red trajectory in **Fig. 6d**. Despite the dynamic obstacle, we successfully managed the flight of two quadrotors through the narrow passage. However, due to the abscence of an overhead motion capture system, the drones experienced positioning loss while navigating the narrow passages due

to obstacles. This limitation restricted the number of drones in the real-world demonstration.

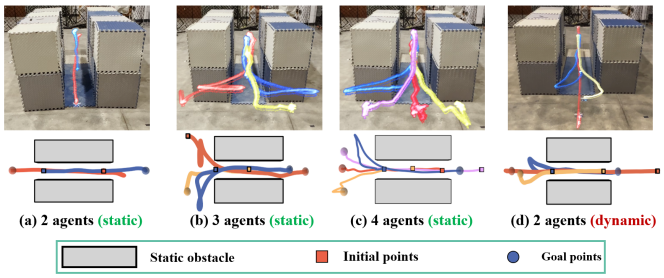


Fig. 6: Real-world experiments for a quadrotor swarm. (a)-(c) show a group of 2 to 4 agents navigating the narrow passage, and (d) shows a group of 2 agents with a dynamic obstacle (red).

VI. Conclusion

We have proposed a novel priority-based deadlock recovery method, PriDRAM, for distributed quadrotor swarms, enabling on-demand collision avoidance in cluttered environments. The proposed priority mechanism dynamically assigns different priorities and optimizes detour points for agents based on their spatial contexts to mitigate deadlocks. Our method shows comparable performance with the AM-swarm method in deadlock-free environments and exhibits more aggressive motion capabilities than DLSC when handling deadlocks. Future work will focus on exploring swarm collaborative navigation in non-convex configuration spaces, increasing the size of the quadrotor swarm, and extending its applications to more complex scenarios.

References

[1] L. Yang, B. Li, W. Li, H. Brand, B. Jiang, and J. Xiao, "Concrete defects inspection and 3d mapping using cityflyer quadrotor robot," *IEEE/CAA Journal of Automatica Sinica*, vol. 7, no. 4, 2020.

[2] L. Pei, J. Lin, Z. Han, L. Quan, Y. Cao, C. Xu, and F. Gao, "Collaborative planning for catching and transporting objects in unstructured environments," *IEEE Robotics and Automation Letters*, vol. 9, no. 2, pp. 1098–1105, 2024.

[3] K. Choutri, M. Lagha, and L. Dala, "A fully autonomous search and rescue system using quadrotor uav," *International Journal of Computing and Digital Systems*, vol. 10, no. 01, pp. 403–414, 2021.

[4] B. P. Duisterhof, S. Li, J. Burgués, V. J. Reddi, and G. C. de Croon, "Sniffy bug: A fully autonomous swarm of gas-seeking nano quadcopters in cluttered environments," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 9099–9106.

[5] J. Snape, J. Van Den Berg, S. J. Guy, and D. Manocha, "The hybrid reciprocal velocity obstacle," *IEEE Transactions on Robotics*, vol. 27, no. 4, pp. 696–706, 2011.

[6] J. Van Den Berg, J. Snape, S. J. Guy, and D. Manocha, "Reciprocal collision avoidance with acceleration-velocity obstacles," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 3475–3482.

[7] 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.

[8] J. Park and H. J. Kim, "Online trajectory planning for multiple quadrotors in dynamic environments using relative safe flight corridor," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 659–666, 2020.

[9] J. Park, D. Kim, G. C. Kim, D. Oh, and H. J. Kim, "Online distributed trajectory planning for quadrotor swarm with feasibility guarantee using linear safe corridor," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4869–4876, 2022.

[10] C. E. Luis, M. Vukosavljev, and A. P. Schoellig, "Online trajectory generation with distributed model predictive control for multi-robot motion planning," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 604–611, 2020.

[11] C. E. Luis and A. P. 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.

[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 *2012 IEEE/RSJ international conference on Intelligent Robots and Systems*, 2012, pp. 1917–1922.

[13] Y. Chen, M. Cutler, and J. P. How, "Decoupled multiagent path planning via incremental sequential convex programming," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 5954–5961.

[14] V. K. Adajania, S. Zhou, A. K. Singh, and A. P. Schoellig, "Amswarm: An alternating minimization approach for safe motion planning of quadrotor swarms in cluttered environments," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 1421–1427.

[15] A. Yorozu, H. Hailu, and A. Ohya, "Experimental investigation of mutual collision avoidance behavior for multiple mobile robots," in *2021 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, 2021, pp. 1164–1169.

[16] Y. Chen, C. Wang, M. Guo, and Z. Li, "Multi-robot trajectory planning with feasibility guarantee and deadlock resolution: An obstacle-dense environment," *IEEE Robotics and Automation Letters*, vol. 8, no. 4, pp. 2197–2204, 2023.

[17] J. Grover, C. Liu, and K. Sycara, "The before, during, and after of multi-robot deadlock," *The International Journal of Robotics Research*, vol. 42, no. 6, pp. 317–336, 2023.

[18] A. Pierson, W. Schwarting, S. Karaman, and D. Rus, "Weighted buffered voronoi cells for distributed semi-cooperative behavior," in *2020 IEEE international conference on robotics and automation (ICRA)*, 2020, pp. 5611–5617.

[19] X.-S. Wu, H. Yue, Q.-M. Liu, X. Zhang, and C.-F. Shao, "A new heuristics model of simulating pedestrian dynamics based on voronoi diagram," *Chinese Physics B*, vol. 30, no. 1, p. 018902, 2021.

[20] J. Park, Y. Lee, I. Jang, and H. J. Kim, "Dlsc: Distributed multi-agent trajectory planning in maze-like dynamic environments using linear safe corridor," *IEEE Transactions on Robotics*, 2023.

[21] M. Abdullhak and A. Vardy, "Deadlock prediction and recovery for distributed collision avoidance with buffered voronoi cells," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 429–436.

[22] Y. Chen, M. Guo, and Z. Li, "Deadlock resolution and feasibility guarantee in mpc-based multi-robot trajectory generation," *arXiv preprint arXiv:2202.06071*, 2022.

[23] E. Soria, F. Schiano, and D. Floreano, "Distributed predictive drone swarms in cluttered environments," *IEEE Robotics and Automation Letters*, vol. 7, no. 1, pp. 73–80, 2021.

[24] V. K. Adajania, H. Masnavi, F. Rastgar, K. Kruusamae, and A. K. Singh, "Embedded hardware appropriate fast 3d trajectory optimization for fixed wing aerial vehicles by leveraging hidden convex structures," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 571–578.

[25] G. Taylor, R. Burmeister, Z. Xu, B. Singh, A. Patel, and T. Goldstein, "Training neural networks without gradients: A scalable admm approach," in *International conference on machine learning*, 2016, pp. 2722–2731.

[26] R. Van Parys and G. Pipeleers, "Online distributed motion planning for multi-vehicle systems," in *2016 European Control Conference (ECC)*, 2016, pp. 1580–1585.

[27] G. M. Phillips and G. M. Phillips, "Bernstein polynomials," *Interpolation and Approximation by Polynomials*, pp. 247–290, 2003.

[28] J. Tordesillas and J. P. How, "Mader: Trajectory planner in multiagent and dynamic environments," *IEEE Transactions on Robotics*, vol. 38, no. 1, pp. 463–476, 2021.

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