2023 IEEE International Conference on Robotics and Automation (ICRA 2023)
May 29 - June 2, 2023. London, UK

Robust MADER: Decentralized and Asynchronous Multiagent Trajectory Planner Robust to Communication Delay

Kota Kondo, Jesus Tordesillas, Reinaldo Figueroa,
Juan Rached, Joseph Merkel, Parker C. Lusk, and Jonathan P. How

***Abstract*— Although communication delays can disrupt multiagent systems, most of the existing multiagent trajectory planners lack a strategy to address this issue. State-of-the-art approaches typically assume perfect communication environments, which is hardly realistic in real-world experiments. This paper presents Robust MADER (RMADER), a decentralized and asynchronous multiagent trajectory planner that can handle communication delays among agents. By broadcasting both the newly optimized trajectory and the committed trajectory, and by performing a delay check step, RMADER is able to guarantee safety even under communication delay. RMADER was validated through extensive simulation and hardware flight experiments and achieved a 100% success rate of collision-free trajectory generation, outperforming state-of-the-art approaches.**

SUPPLEMENTARY MATERIAL

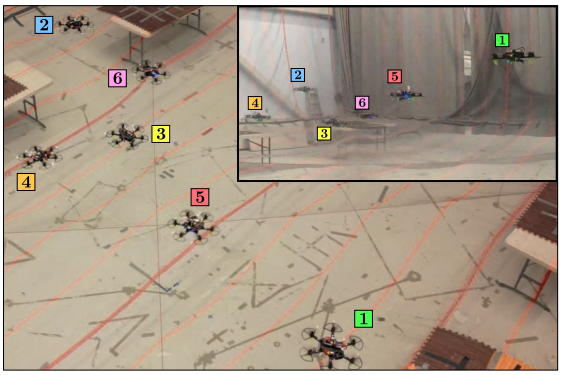
Video: https://youtu.be/vH09kwJOBYs
Code: https://github.com/mit-acl/rmader

I. INTRODUCTION

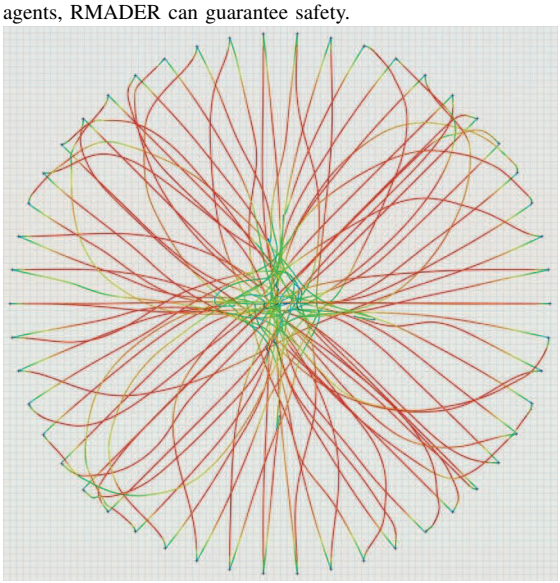
Multiagent UAV trajectory planning has been extensively studied in the literature for its wide range of applications. These planners can be centralized [1], [2], [3] (one machine plans every agent's trajectory) or decentralized [4], [5], [6] (each agent plans its own trajectory). Decentralized planners are more scalable and robust to failures of the centralized machine. Despite these advantages, a decentralized scheme requires communication between the agents, and communication delays could potentially introduce failure in the trajectory deconfliction between the agents, which is essential to guarantee safety [7]. multiagent planners can also be classified according to whether or not they are asynchronous. In an asynchronous setting, each agent independently triggers the planning step without considering the planning status of other agents. Asynchronous approaches do not require a synchronous mechanism among agents and therefore more scalable than synchronous approaches, but they are also more susceptible to communication delays since agents are planning and executing trajectories independently.

Many decentralized state-of-the-art trajectory planners do not consider communication delays or explicitly state assumptions about communication. For example, the planners presented in **SCP** [8], **decNS** [9], and **LSC** [10] are decentralized and synchronous, but SCP and decNS implicitly and LSC explicitly assume a perfect communication environment without any communication delays.

Aerospace Controls Laboratory, MIT, 77 Massachusetts Ave, Cambridge, MA, USA {kkondo, jtorde, reyfp, jrached, jamerkel, plusk, jhow}@mit.edu



(a) Hardware experiments: 6 UAVs running RMADER onboard. Despite the existence of communication delays between the agents, RMADER can guarantee safety.



(b) Simulation experiments: 50 UAVs in a circle configuration successfully exchange their positions despite the 100 ms communication delay introduced on purpose. The color denotes the velocity (red higher and blue lower).

Fig. 1. RMADER Hardware and Simulation Results

The algorithm **decMPC** [11] is decentralized, but it requires synchronicity and communication delays to be within a fixed planning period. **decGroup** [12] is a decentralized semi-asynchronous planner, which solves joint optimization as a group. decGroup cooperatively tackles the path-planning problem but implicitly assumes no communication delays. **ADPP** [13] is asynchronous[1] and decentralized, but it as-

[1]As in [4], we define asynchronous planning to be when the agent triggers trajectory planning independently without considering the planning status of other agents. However, ADPP [13] implements a prioritized asynchronous approach, meaning plannings are not fully independently triggered.

TABLE I. State-of-the-art Decentralized Multiagent Planners.

Method	Asynchronous?	Handles Comm. Delay?	Hardware Demonstration
decNS [9]	No	No	No
SCP [8]	No	No	Yes
LSC [10]	No	No	Yes
decMPC [11]	No	Yes	No
decGroup [12]	Yes/No[2]	No	Yes
ADPP [13]	Yes[3]	No	Yes
MADER [4]	Yes	No	No
EGO-Swarm [5]	Yes	No	Yes
AsyncBVC [14]	Yes	Yes	No
RMADER (proposed)	Yes	Yes	Yes

[2] decGroup triggers joint-optimization in dense environments and switches to a centralized, synchronous planner.
[3] Asynchronous but requires priority information for planning.

sumes perfect communication without delay. Our previous work **MADER** [4] is asynchronous and decentralized but assumes no communication delays. **EGO-Swarm** [5] also proposes a decentralized, asynchronous planner that requires agents to periodically broadcast a trajectory at a fixed frequency, and each agent immediately performs collision checks upon receiving the message. **EGO-Swarm** is the first fully decentralized, asynchronous trajectory planner successfully demonstrating hardware experiments, yet it still suffers from a collision due to communication delays, as shown in Section III. **AsyncBVC** [14] proposes an asynchronous decentralized trajectory planner that can guarantee safety even with communication delays. However, the future trajectories are constrained by past separating planes, which can overconstrain the solution space and hence increase the conservatism. Also, they only presented simulation results with up to 4 agents, and no hardware experiments were implemented. In addition, it relies on discretization when solving the optimization problem, meaning that safety is only guaranteed on the discretization points. Our approach instead is able to guarantee safety in a continuous approach by leveraging the MINVO basis [15].

To address these shortcomings, we propose **Robust MADER** (**RMADER**), a decentralized and asynchronous multiagent trajectory planner capable of generating collision-free trajectories in the presence of realistic communication delays. As shown in Table I, RMADER is the first approach to demonstrate decentralized, asynchronous trajectory planning robust to communication delays. RMADER builds on convex MADER, which is a modified version of the nonconvex MADER presented in our previous work [4] (more details are available in Appendix I). Our contributions include:

1) An algorithm that guarantees collision-free trajectory generation even with the existence of communication delays among vehicles.
2) Extensive simulations comparing our approach to state-of-the-art methods under communication delays that demonstrate **a 100% success rate** of collision-free trajectory generation (see Table V).
3) Extensive set of decentralized hardware experiments using 6 UAVs, and achieving velocities up to $2.8\,\mathrm{m/s}$.

II. Trajectory Deconfliction

In MADER [4] and RMADER, UAVs plan trajectories asynchronously and broadcast the results to each other. Each agent uses these trajectories as constraints in the optimization problem. Assuming no communication delays exist, safety can be guaranteed using our previous approach presented in MADER (summarized in Section II-A). This safety guarantee, however, breaks when an agent's planned trajectory is received by other agents with some latency. Section II-B shows how RMADER guarantees safety even with communication delays. We use the definitions shown in Table II.

A. MADER Deconfliction

MADER [4] guarantees collision-free trajectories under ideal communication through the use of the planning stages shown in Fig. 2. An agent plans its initial trajectory during **Optimization** (**O**), followed by **Check** (**C**) to ensure its plan does not lead to a collision. Finally, **Recheck** (**R**) is used to check if the agent received any trajectory updates from other agents during **C** - if so, an agent starts over planning at **O**. Although MADER does not have explicit safety guarantees in the presence of communication delays, its trajectories are still collision free for cases 1 and 2 shown in Fig. 2. However, collisions may occur in cases 3 and 4 of Fig. 2. These four cases are summarized in Fig. 2 and Table III.

B. Robust MADER Deconfliction

To achieve robustness to communication delays, we replace the **Recheck** with **Delay Check** (**DC**), where each agent repeatedly checks if its newly optimized trajectory conflicts with other agents' trajectories. If an agent detects conflicts during **DC**, it discards the new trajectory and starts another **O** while executing its previous trajectory. If no collisions are detected in **DC**, it starts executing the new trajectory. To guarantee collision-free trajectory generation, **DC** needs to be longer than the possible longest communication delay (i.e., $\delta_{\mathrm{DC}} \geq \delta_{\max}$). That way, an agent can always keep at least one collision-free trajectory. It could, however, not be ideal for introducing such a long δ_{DC}, and therefore, in Section III we also tried $\delta_{\mathrm{DC}} < \delta_{\max}$ and measure its performance. Fig. 3 shows how RMADER

TABLE II. Definitions of the different delay quantities: Note that, by definition, $0 \leq \delta_{\mathrm{introd}} \leq \delta_{\mathrm{actual}} \leq \delta_{\max}$. See also Figs 6f and 7 for the actual histogram of the delays in simulation and hardware, respectively.

δ_{actual}	Actual communication delays among agents.
$\delta_{\max}$	Possible maximum communication delay.
δ_{introd}	Introduced communication delay in simulations.
δ_{DC}	Length of **Delay Check** in RMADER. To guarantee safety, $\delta_{\max} \leq \delta_{\mathrm{DC}}$ must be satisfied.

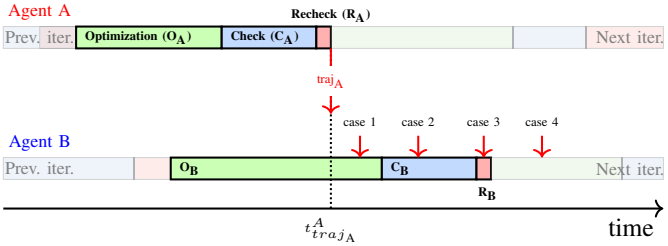


Fig. 2. MADER deconfliction: Agent A solves $\mathbf{O_A}$ to find its optimal trajectory, constrained by other agents' trajectories. Agent A then begins $\mathbf{C_A}$ to determine if that generated trajectory has any conflicts with trajectories received during $\mathbf{O_A}$. Finally, in $\mathbf{R_A}$, Agent A checks if it received any trajectories during $\mathbf{C}$. The four cases shown in the figure correspond to different communication delays, resulting in traj_A being received by Agent B at different times. Designed to guarantee safety when there is no communication delays, MADER also guarantees safety when traj_A arrives during $\mathbf{O_B}$ (Case 1) or $\mathbf{C_B}$ (Case 2), but could cause collisions if traj_A is received during/after (Case 3/4) $\mathbf{R_B}$.

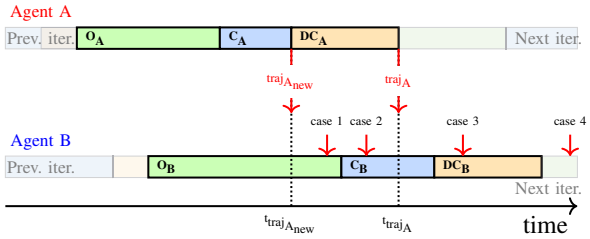


Fig. 3. RMADER deconfliction: After $\mathbf{C_A}$ Agent A keeps executing the trajectory from the previous iteration, $\text{traj}_{A_{prev}}$, while checking potential collisions of newly optimized trajectory, $\text{traj}_{A_{new}}$. This is because $\text{traj}_{A_{new}}$ might have conflicts due to communication delays and need to be checked in $\mathbf{DC_A}$, and $\text{traj}_{A_{prev}}$ is ensured to be collision-free. If collisions are detected in either $\mathbf{C_A}$ or $\mathbf{DC_A}$, Agent A keeps executing $\text{traj}_{A_{prev}}$ (i.e., $\text{traj}_A \leftarrow \text{traj}_{A_{prev}}$). If $\mathbf{DC_A}$ does not detect collisions, Agent A broadcasts and starts implementing $\text{traj}_{A_{new}}$ (i.e., $\text{traj}_A \leftarrow \text{traj}_{A_{new}}$).

TABLE III. Safety guarantees under communication delays: Depending on when traj_A is received by Agent B, the deconfliction takes place at different stages. MADER does not guarantee safety if traj_A is received during $\mathbf{R_B}$ or during the following iteration, while RMADER guarantees safety in all the cases. Note that in RMADER, if Agent B does not receive traj_A by the end of $\mathbf{DC_B}$, then the deconfliction is performed by Agent A (specifically, in $\mathbf{C_A}$ or $\mathbf{DC_A}$) and not by Agent B. Agent A will use $\text{traj}_{B_{new}}$ and/or traj_B for this.

	MADER			
When traj_A received	$\mathbf{O_B}$ (Case 1)	$\mathbf{C_B}$ (Case 2)	$\mathbf{R_B}$ (Case 3)	**Next iter.** (Case 4)
traj_A deconflicted? When?	Yes $\mathbf{C_B}$	Yes $\mathbf{R_B}$	No	No

	RMADER			
When $\text{traj}_{A_{new}}$/traj_A received	$\mathbf{O_B}$ (Case 1)	$\mathbf{C_B}$ (Case 2)	$\mathbf{DC_B}$ (Case 3)	**Next iter.** (Case 4)
$\text{traj}_{A_{new}}$/traj_A deconflicted? When?	Yes $\mathbf{C_B}$	Yes $\mathbf{DC_B}$	Yes $\mathbf{DC_B}$	Yes $\mathbf{C_A}$ or $\mathbf{DC_A}$

TABLE IV. Differences between MADER and RMADER

MADER	RMADER
Upon successful $\mathbf{C}$ and $\mathbf{R}$, the newly optimized trajectory is broadcast to other agents	Upon successful $\mathbf{C}$, $\text{traj}_{J_{new}}$ is broadcast. After $\mathbf{DC}$, the committed trajectory traj_J (which is either $\text{traj}_{J_{new}}$ and $\text{traj}_{J_{prev}}$ depending on whether $\mathbf{DC}$ is satisfied or not) is broadcast
$\mathbf{R}$ is a Boolean check to see if the agent received traj. in $\mathbf{C}$	$\mathbf{DC}$ is a sequence of collision checks
$\mathbf{R}$ is very short	$\mathbf{DC}$ lasts δ_{DC} seconds

Algorithm 1 Robust MADER - Agent B

Require: traj_B, a feasible trajectory
1: **while** not goal reached **do**
2: $\quad\text{traj}_{B_{new}} = \mathbf{Optimization}()$
3: $\quad$**if** $\mathbf{Check}(\text{traj}_{B_{new}}) ==$ False **then**
4: $\quad\quad$Go to Line 2
5: $\quad$**end if**
6: $\quad$Broadcast $\text{traj}_{B_{new}}$
7: $\quad$**if** $\mathbf{Delay\ Check}(\text{traj}_{B_{new}}) ==$ False **then**
8: $\quad\quad\text{traj}_B \leftarrow \text{traj}_{B_{prev}}$, and go to Line 11
9: $\quad$**end if**
10: $\quad\text{traj}_B \leftarrow \text{traj}_{B_{new}}$
11: $\quad$Broadcast traj_B
12: **end while**

Algorithm 2 Delay Check - Agent B

1: **function** $\mathbf{Delay\ Check}(\text{traj}_{B_{new}})$
2: $\quad$**for** δ_{DC} seconds **do**
3: $\quad\quad$**if** $\text{traj}_{B_{new}}$ collides with any trajectory in $\mathcal{Q}_B$ **then**
4: $\quad\quad\quad$**return** False
5: $\quad\quad$**end if**
6: $\quad$**end for**
7: $\quad$**return** True
8: **end function**

deals with communication delays. Furthermore, Table III shows all the possible cases in which communication delays could occur and how these are handled by RMADER to generate collision-free trajectories even with communication delays. The pseudocode of RMADER deconfliction is given in Algorithm 1. First, Agent B runs $\mathbf{O}$ to obtain $\text{traj}_{B_{new}}$ and broadcasts it if $\mathbf{C}$ is satisfied (Line 6). This $\mathbf{C}$ aims to determine if $\text{traj}_{B_{new}}$ has any conflicts with trajectories received in $\mathbf{O}$. Then, Agent B commits either $\text{traj}_{B_{prev}}$ or $\text{traj}_{B_{new}}$ - if $\mathbf{DC}$ detects conflicts, Agent B commits to $\text{traj}_{B_{prev}}$ (Line 8), and if $\mathbf{DC}$ detects no conflicts, $\text{traj}_{B_{new}}$ (Line 10). This committed trajectory is then broadcast to the other agents (Line 11). Table IV highlights the differences between MADER and RMADER.

Agent B stores the trajectories received from other agents in a set $\mathcal{Q}_B$; for example, for each Agent J, Agent B will store the committed trajectory of Agent J, traj_J, and possibly the newly optimized trajectory $\text{traj}_{J_{new}}$ if any new committed trajectory has still not been broadcast. Figure 4 shows the way Agent B stores trajectories from Agent A. $\mathcal{Q}_B$ is used in $\mathbf{O}$, $\mathbf{Check}$, and $\mathbf{DC}$, where Agent B checks for collision against all the trajectories stored in $\mathcal{Q}_B$. Note that $\mathcal{Q}_B$ is updated in parallel with $\mathbf{O}$ and $\mathbf{DC}$. At the beginning of $\mathbf{O_B}$, an agent generates an optimal trajectory, $\text{traj}_{B_{new}}$, using all the trajectories stored in $\mathcal{Q}_B$ as constraints, and then, Agent B checks for $\text{traj}_{B_{new}}$'s potential collisions against $\mathcal{Q}_B$, which is updated in optimization. Finally, Agent B repeatedly checks for collisions against $\mathcal{Q}_B$ in $\mathbf{DC_B}$, which lasts for δ_{DC}. Note that $\mathbf{C}$ is not necessary to guarantee safety in RMADER - $\mathbf{O}$

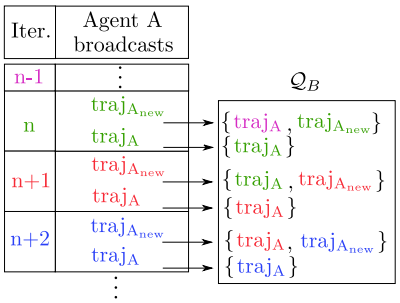
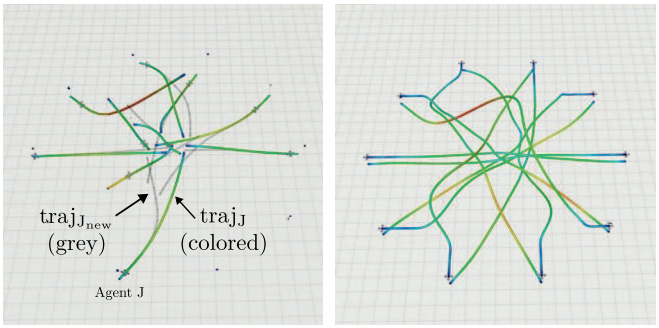


Fig. 4. Agent B stores in $\mathcal{Q}_B$ the last committed trajectory of Agent A. It will also contain the newly optimized trajectory $\text{traj}_{A_{new}}$ while the new committed trajectory has still not been received by Agent B.



(a) For Agent J, the colored trajectory is the committed (safety-guaranteed) trajectory (traj_J), and the grey trajectory is the newly optimized trajectory ($\text{traj}_{J_{new}}$).

(b) Actual trajectories flown by the agents. All 10 agents successfully swap their positions in a circle configuration.

Fig. 5. 10 agents employing RMADER exchange their positions in a circle of radius 20 m. In the colored trajectories, red represents a high speed while blue denotes a low speed.

followed by **DC** alone can generate collision-free trajectories as long as $\delta_{DC} \geq \delta_{max}$ holds. Though **C** detects collisions before broadcasting any (possibly conflicted) trajectories and allows an agent to start another **O**, which prevents unnecessary communication.

III. Simulation Results

We tested Slow EGO-Swarm, EGO-Swarm [5], MADER [4], and RMADER (proposed) on a `general-purpose-N2` Google Cloud instance with 32 Intel Core i7s. In each scenario, we conducted **100 simulations** with 10 agents positioned in a $10\,\text{m}$ radius circle, exchanging positions diagonally as shown in Fig. 5. Note that this paper convexified MADER optimization problem as detailed in Appendix I, and we used the convex optimization problem for both MADER and RMADER. The maximum dynamic limits (velocity, acceleration, and jerk) for these algorithms are set to $10\,\text{m/s}$, $20\,\text{m/s}^2$, and $30\,\text{m/s}^3$. EGO-Swarm carries out a sequential startup - agents commits their first trajectory in a pre-determined order to avoid unnecessary trajectory conflicts. We also introduced $0.25\,\text{s}$-apart startup into MADER and RMADER.

Slow EGO-Swarm is EGO-Swarm with smaller dynamic limits. We first tested EGO-Swarm with default parameters provided in [5] and saw a significant number of conflicts. Therefore we increased the weights of the collision costs in EGO-Swarm's cost function up to 1000 (we tried more than 1000, but it did not change the results) while other weights (s.t. trajectory feasibility) are on the order of single digits; however, we still observed collisions (as seen in the second row of Table V). We thus decreased the maximum velocity and acceleration of EGO-Swarm down to $5\,\text{m/s}$ and $10\,\text{m/s}^2$, which we define as Slow EGO-Swarm.

Although we introduced a fixed δ_{introd} for simulated communications, δ_{actual} can be larger due to the simulation computer's computational limitations. The communication delays observed in simulation are shown in Fig. 6f for five nominal values of δ_{introd} (δ_{introd} = 0, 50, 100, 200, and 300 ms). As long as $\delta_{DC} \geq \delta_{max}$ holds, RMADER can generate collision-free trajectories.

Table V and Fig. 6 showcase each approach's performance in simulations. RMADER was implemented in the case of (1) $\delta_{DC} \geq \delta_{max}$ ($\delta_{DC} >$ 100th percentile of δ_{actual}) and (2) $\delta_{max} \geq \delta_{DC}$ ($\delta_{DC} \approx$ 75th percentile of δ_{actual}). When $\delta_{DC} \geq \delta_{max}$ holds, collision-free trajectory planning is guaranteed, and therefore RMADER generates 0 collisions for all the δ_{introd}, while other approaches suffer collisions. As expected, the longer δ_{introd} more collisions Slow EGO-Swarm, EGO-Swarm, and MADER generate. In the case of (2) $\delta_{max} \geq \delta_{DC}$, although safety is not theoretically guaranteed, since δ_{DC} is long enough, RMADER succeeds to generate collision-free trajectories. Note that Case (2) could have collisions in case agents have conflicted trajectories and their trajectories fall into the rest of ≈25%.

It is also worth mentioning that RMADER's robustness to communication delays is obtained by layers of conflict checks and agents periodically occupying two trajectory spaces, which can result in generating conservative trajectories and trading off UAV performance. *Avg. Number of Stops* in Table V, for instance, suggests more stoppage than other approaches. As $\int \|\mathbf{a}\|^2\, dt$ and $\int \|\mathbf{j}\|^2\, dt$ show RMADER's trajectories are less smooth than MADER, and RMADER takes longer *Travel Time* than others (Slow EGO-Swarm takes more but that is because of its smaller dynamic limits, and therefore a direct comparison is not fair).

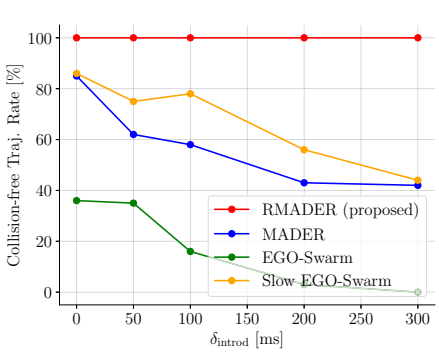
IV. Hardware Experiments

A total of 10 hardware experiments (5 flights for each) demonstrate RMADER's robustness to communication delays as well as MADER's shortcomings. Each flight test had 6 UAVs in the $9.2\,\text{m} \times 7.5\,\text{m} \times 2.5\,\text{m}$ flight space and lasted ≈1 min. All the planning and control run onboard the UAV, and the state estimation is obtained by fusing IMU measurements with an external motion capture system. A safety mechanism running in parallel reports potential collisions and sends commands to the UAVs to avoid colliding.

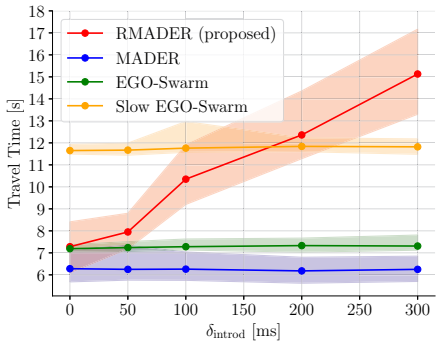
During the MADER hardware experiments due to the effects of communication delays, **7** potential collisions were detected by the safety mechanism. RMADER, on the other hand, did not generate conflicts. A snapshot of one of the RMADER experiments is shown in Fig. 1a, and a successful trajectory deconfliction despite the communication delay is shown in Fig. 8. The maximum velocities and averate flight distances achieved during the MADER and RMADER

TABLE V. Cases $\delta_{\text{introd}} = 0$ ms, $\delta_{\text{introd}} = 50$ ms, $\delta_{\text{introd}} = 100$ ms, (see Fig. 6f for actual message delays). The bold values represent the case where $\delta_{\text{DC}} \geq \delta_{\text{max}}$, which is the necessary condition to ensure safety.

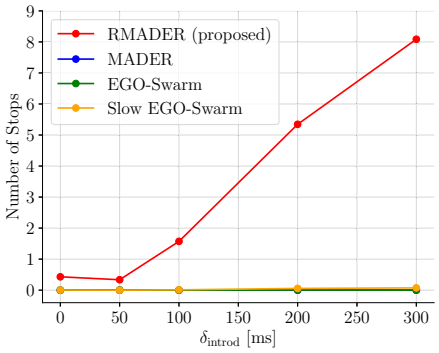
Method	δ_{DC} [ms]	Collision [%]	Avg number of stops	$\int \|\|\mathbf{a}\|\|^2 dt$ [m^2/s^3]	$\int \|\|\mathbf{j}\|\|^2 dt$ [m^2/s^5]	Travel Time [s] Avg	Travel Time [s] Max
Slow EGO-Swarm	N/A	14|25|22	0|0|0	109.8|113.2|113.5	15388.2|15491.5|15486.2	11.65|11.67|11.76	11.93|11.99|12.97
EGO-Swarm	N/A	64|84|84	0.004|0.0|0.01	662.5|700.7|787.9	90721.9|94611.9|104160.3	7.19|7.24|7.28	7.38|7.51|7.63
MADER (convex)	N/A	15|38|42	0.0|0.001|0.0	78.09|74.19|74.74	1595.9|1643.6|1638.5	6.28|6.25|6.26	7.15|7.35|7.04
RMADER (proposed)	**100|130|200** (>100th percentile of δ_{actual} and $\delta_{\text{DC}} \geq \delta_{\text{max}}$ holds)	**0|0|0**	**0.46|0.347|1.751**	**127.7|147.9|190.5**	**2939.4|3712.4|5942.1**	**7.28|7.95|10.35**	**8.41|8.80|11.91**
	25|56|105 (≈75th percentile of δ_{actual} so $\delta_{\text{DC}} \geq \delta_{\text{max}}$ does not hold)	0|0|0	0.001|0.007|0.086	99.52|112.0|137.7	1844.3|2142.3|3056.2	6.80|6.87|7.30	7.66|8.02|8.89



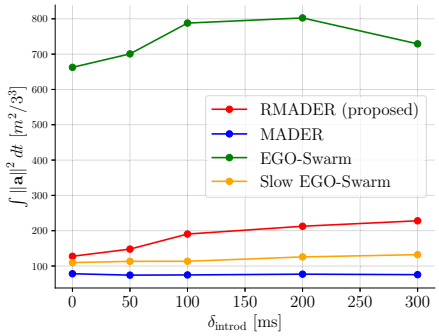
(a) Collision-free Trajectory Rate



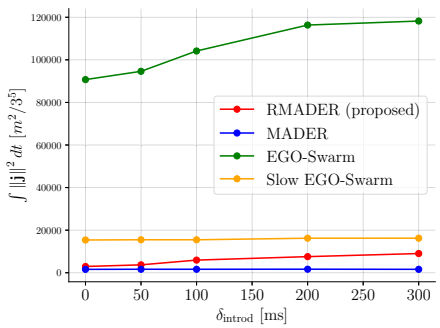
(b) Travel Time - shaded parts indicate its maximum and minimum value.



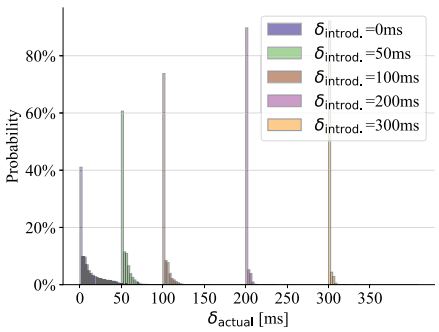
(c) Number of Stops



(d) Trajectory Smoothness (Acceleration)



(e) Trajectory Smoothness (Jerk)



(f) Distribution of δ_{actual} in simulations. Due to computer's computational limits, messages do not travel instantly.

Fig. 6. 100 Flight Simulation Results: Fig. 6a shows RMADER generates collision-free trajectory at 100%, while other state-of-the-art approaches fail when communication delays are introduced. To maintain collision-free trajectory generation, RMADER periodically occupies two trajectories, and other agents need to consider two trajectories as a constraint, which could lead to conservative plans - longer *Travel Time* and more *Avg. Number of Stops*. This is a trade-off between safety and performance. MADER reports a few collided trajectory because $\delta_{\text{actual}} > 0$ ms.

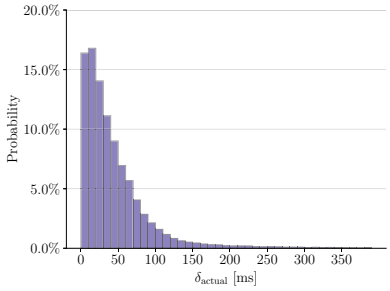


Fig. 7. Distribution of δ_{actual} in hardware experiments: Both MADER and RMADER were tested in 5 flight experiments. Compared to simulations (see the case $\delta_{\text{introd}} = 0$ ms in Fig. 6f), δ_{actual} is much larger in hardware.

TABLE VI. MADER hardware experiments

	Exp. 1	**Exp. 2**	**Exp. 3**	**Exp. 4**	**Exp.5**
Max vel. [m/s]	2.7	2.5	2.7	2.7	3.0
Avg. travel distance [m]	70.3	67.6	61.1	61.1	65.2

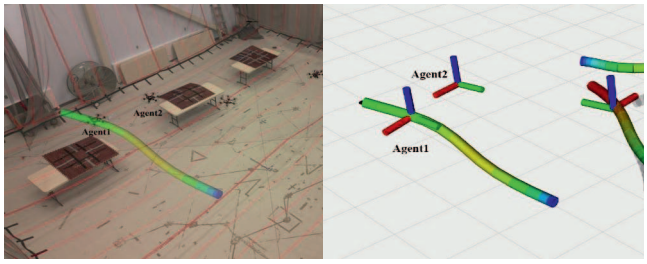
hardware experiments are shown in Tables VI and VII. The UAVs achieved maximum velocity of 2.8 m/s.

V. CONCLUSIONS AND FUTURE WORK

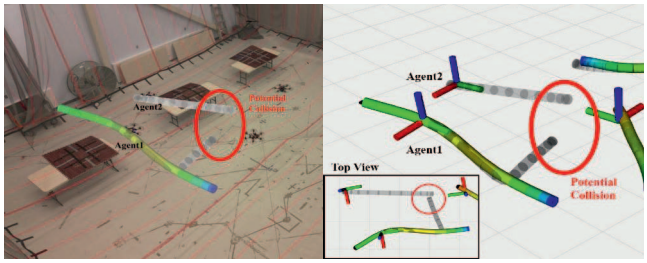
We proposed RMADER, a decentralized and asynchronous multiagent trajectory planner that is robust to communication delays. The key property of RMADER is

TABLE VII. RMADER hardware ecperiments

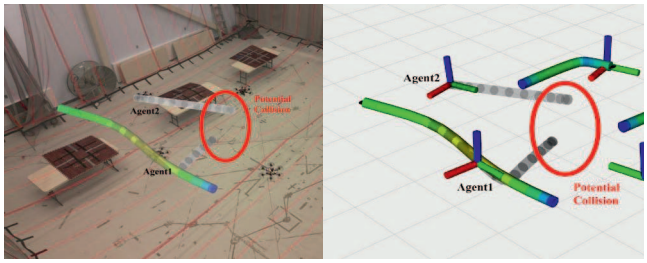
	Exp. 6	Exp. 7	Exp. 8	Exp. 9	Exp. 10
Max vel. [m/s]	2.6	2.7	2.8	2.7	2.7
Avg. travel distance [m]	45.6	58.2	58.4	58.3	54.8



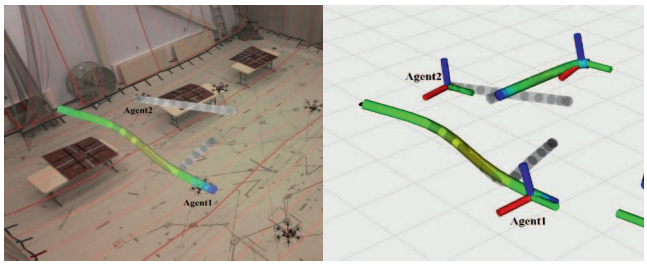
$t = 0\,\mathrm{s}$: Agent 1 is following its trajectory



$t = 0.15\,\mathrm{s}$: Agent 1 and Agent 2 published their $\mathrm{traj}_{\mathrm{new}}$ only $10\,\mathrm{ms}$ apart. Due to communication delays each agent did not consider the other trajectory, and thus these two trajectories are in conflicts. Note that we have a $1.5\,\mathrm{m}$-tall boundary box, and thus these trajectories are in collision.



$t = 1.01\,\mathrm{s}$: During Delay Check both agents detected conflicts and did not commit their trajectory.



$t = 1.97\,\mathrm{s}$: Collision avoided.

Fig. 8. RMADER Successful Deconfliction under Communication Delays

that it guarantees safety even when there are communication delays. RMADER guarantees collision-free trajectories by introducing a delay check mechanism and keeping at least one collision-free trajectory available throughout planning. Simulation and hardware experiments showed RMADER's robustness to communication delays and the trade-off between safety and performance. Potential future work includes large-scale hardware experiments.

TABLE VIII. Convex MADER vs Nonconvex MADER

Method	Computation Time [ms] Avg	Max	$\int \|\mathbf{a}\|^2\,dt$ [$\mathrm{m^2/s^3}$]	$\int \|\mathbf{j}\|^2\,dt$ [$\mathrm{m^2/s^5}$]	Number of Stops	Travel Time [s]	Travel Distance [m]
convex MADER	**31.08**	**433.0**	**103.5**	**2135.0**	0.18	16.05	**75.24**
nonconvex MADER	39.23	724.0	441.93	20201.8	**0.16**	**9.93**	75.80

APPENDIX I
CONVEX VS NONCONVEX MADER

Our prior work MADER [4] formulated a nonconvex optimization problem by using both the control points and the separating planes as decision variables [4, Section VI-D]. This could, however, cause expensive onboard computation. Therefore we re-formulated the problem as convex by fixing the separating planes in the optimization (i.e., by not including these planes as decision variables). In addition, to generate smoother trajectories, we added a constraint on the maximum jerk.

We compared both version on a `general-purpose-N2` Google Cloud instance with 32 Intel® Core i7. The flight space contains 250 dynamic and static obstacles, and the UAV must fly through the space to reach a goal $75\,\mathrm{m}$ away. Maximum velocity/acceleration/jerk are set to $10\,\mathrm{m/s}$ / $20\,\mathrm{m/s^2}$ / $30\,\mathrm{m/s^3}$. The performance was measured in terms of *Computation Time*, trajectory smoothness indicated by $\int \|\mathbf{a}\|^2\,dt$ and $\int \|\mathbf{j}\|^2\,dt$, *Number of Stops*, *Travel Time*, and *Travel Distance*. The results are shown in Table VIII, where all data is the average of **100 simulations**. The notation $\int \|\mathbf{a}\|^2\,dt$ and $\int \|\mathbf{j}\|^2\,dt$ refers to the time integral of squared norm of the acceleration and jerk along the trajectory, respectively. Higher values therefore represent a less smooth trajectory. *Number of Stops* is the number of times the UAV had to stop on its way to the goal. Table VIII indicates that convex MADER is computationally less expensive and generates smoother trajectories, but nonconvex MADER performs better in terms of the *Number of Stops* and *Travel Time*. Since convex MADER has a computational advantage and can generate smoother trajectories, we implemented convex MADER for MADER and RMADER in all the simulations and hardware experiments in this paper.

ACKNOWLEDGMENT

We would like to thank Nick Rober, Lakshay Sharma, Miguel Calvo-Fullana, Andrea Tagliabue, Dong-Ki Kim, and Jeremy Cai, for their help and insightful comments. This research is funded by Boeing Research & Technology.

REFERENCES

[1] J. Park, J. Kim, I. Jang, and H. J. Kim, "Efficient Multi-Agent Trajectory Planning with Feasibility Guarantee using Relative Bernstein Polynomial," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, May 2020, pp. 434–440, iSSN: 2577-087X.

[2] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artificial Intelligence*, vol. 219, no. C, pp. 40–66, Feb. 2015. [Online]. Available: https://doi.org/10.1016/j.artint.2014.11.006

[3] 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, Apr. 2018, conference Name: IEEE Robotics and Automation Letters.

[4] 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, 2022.

[5] X. Zhou, J. Zhu, H. Zhou, C. Xu, and F. Gao, "EGO-Swarm: A Fully Autonomous and Decentralized Quadrotor Swarm System in Cluttered Environments," Nov. 2020, arXiv:2011.04183 [cs] version: 1. [Online]. Available: http://arxiv.org/abs/2011.04183

[6] P. C. Lusk, X. Cai, S. Wadhwania, A. Paris, K. Fathian, and J. P. How, "A Distributed Pipeline for Scalable, Deconflicted Formation Flying," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 5213–5220, Oct. 2020, conference Name: IEEE Robotics and Automation Letters.

[7] J. Gielis, A. Shankar, and A. Prorok, "A Critical Review of Communications in Multi-robot Systems," *Current Robotics Reports*, Aug. 2022. [Online]. Available: https://doi.org/10.1007/s43154-022-00090-9

[8] 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)*, May 2015, pp. 5954–5961, iSSN: 1050-4729.

[9] S. Liu, K. Mohta, N. Atanasov, and V. Kumar, "Towards Search-based Motion Planning for Micro Aerial Vehicles," Oct. 2018, arXiv:1810.03071 [cs]. [Online]. Available: http://arxiv.org/abs/1810.03071

[10] 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, Apr. 2022, conference Name: IEEE Robotics and Automation Letters.

[11] C. Toumieh and A. Lambert, "Decentralized Multi-Agent Planning Using Model Predictive Control and Time-Aware Safe Corridors," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 11 110–11 117, Oct. 2022, conference Name: IEEE Robotics and Automation Letters.

[12] J. Hou, X. Zhou, Z. Gan, and F. Gao, "Enhanced Decentralized Autonomous Aerial Swarm with Group Planning," Mar. 2022, arXiv:2203.01069 [cs]. [Online]. Available: http://arxiv.org/abs/2203.01069

[13] M. Cáp, P. Novák, M. Selecký, J. Faigl, and J. Vokffnek, "Asynchronous decentralized prioritized planning for coordination in multi-robot system," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Nov. 2013, pp. 3822–3829, iSSN: 2153-0866.

[14] S. Baskin and G. Sukhatme, "Asynchronous Real-time Decentralized Multi-Robot Trajectory Planning," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct. 2022.

[15] J. Tordesillas and J. P. How, "MINVO Basis: Finding Simplexes with Minimum Volume Enclosing Polynomial Curves," *Computer-Aided Design*, vol. 151, no. C, Oct. 2022. [Online]. Available: https://doi.org/10.1016/j.cad.2022.103341