2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 23-27, 2022, Kyoto, Japan

Asynchronous Real-time Decentralized Multi-Robot Trajectory Planning

Baskın Şenbaşlar and Gaurav S. Sukhatme

***Abstract*— We present a novel overconstraining and constraint-discarding method for asynchronous, real-time, decentralized, multi-robot trajectory planning that ensures collision avoidance. Our approach utilizes communication between robots. The communication medium is best-effort: messages may be dropped, re-ordered or delayed. Robots conservatively constrain themselves against others assuming they may be working with outdated information, and discard constraints when they receive update messages from others. Our method can augment existing synchronized decentralized receding horizon planning algorithms that utilize separating hyperplanes for collision avoidance thereby making them applicable to asynchronous setups. As an example, we extend an existing model predictive control based, synchronized, decentralized multi-robot planner using our method. We show our method's effectiveness under asynchronous planning and imperfect communication by comparing our extension to the base version. Our extension does not result in any collisions or synchronization-induced deadlocks to which the base version is prone.**

I. INTRODUCTION

Trajectory planning is a central problem for effective multi-robot navigation systems, especially when robots are in close proximity to each other (Fig. 1). It is a fundamental challenge in emerging industries like automated warehouses [1], autonomous driving [2], and automated intersection management [3]. One approach to solve multi-robot trajectory planning problems is to utilize central coordination wherein a central entity plans trajectories for all robots, keeps track of the execution of trajectories, and re-plans as necessary. Challenges in centralized planning systems include communication maintenance between the robots and the central entity, synchronization of robot movements, computational complexity when the number of robots or the environment complexity is high, and the necessity of relaying map information to the central entity if it is not known a priori. To alleviate these challenges, decentralized trajectory planning algorithms have been proposed. In these, each robot plans its own trajectory and coordinates with other robots to avoid collisions. In some scenarios, decentralization of trajectory planning is required because some robots are not controlled by the same entity. In others, it is attractive because it makes the planning problem tractable and the navigation system robust – robots can react to unexpected situations using on-board capabilities without having to wait for a central entity to command them.

Both authors are with the Department of Computer Science, University of Southern California, Los Angeles, CA 90007, United States. baskin.senbaslar@usc.edu, gaurav@usc.edu. Sukhatme holds concurrent appointments as a Professor at USC and as an Amazon Scholar. This paper describes work performed at USC and is not associated with Amazon.

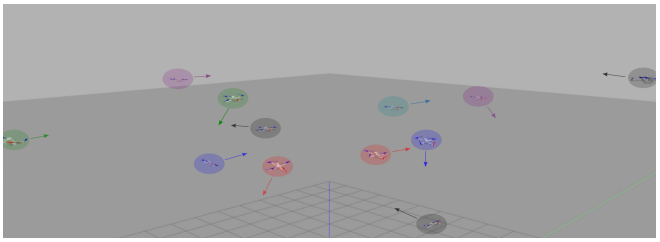

Fig. 1. Twelve quadrotors navigating in close proximity to each other. Multi-robot trajectory planning is a central problem in close proximity scenarios. Our approach ensures collision and synchronization-induced deadlock avoidance between robots when planning is decentralized and asynchronous.

One often-overlooked problem in decentralized multi-robot planning algorithms is the inherent asynchronicity that stems from the planning system. First, since each robot plans for itself, synchronization of when planning starts is not practical. Since planning start timestamps are not synchronized, robots use different snapshots of the environment while planning, which causes disagreements between them on the definitions of *safety*. Second, planning end timestamps between robots are not synchronized, which means that different robots start executing plans at different times, causing disagreements during plan switching, leading to disagreements on the definitions of *safety* as well. Additionally, there is asynchronicity introduced by the communication network – when robots use a communication channel to coordinate their actions, their messages may get delayed, dropped or re-ordered.

We propose a novel approach which ensures that robots do not collide with each other when planning is asynchronous and the communication medium is an imperfect best-effort network. Our contributions are as follows:

- We define **mutually computable separating hyperplane trajectories** that form the basis of our approach.
- Using these trajectories, we propose **a novel overconstraining and constraint-discarding system for decentralized multi-robot asynchronous planning** that ensures safety in the face of message delays, drops, and re-orderings.
- To illustrate the utility of our approach, we present an **implementation of AsyncBVC (Asynchronous Buffered Voronoi Cell)**, an extension of BVC [4] based on our system. In simulations, we show that AsyncBVC outperforms BVC in terms of number of synchronization-induced deadlocks and number of collisions under asynchronous planning in obstacle free environments. We also show the performance of AsyncBVC under message delays, drops and re-orderings.

2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) | 978-1-6654-7927-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/IROS47612.2022.9981760

II. RELATED WORK

We categorize decentralized multi-robot planning algorithms into reactive planners (or controllers) and long horizon planners. Given the state of the environment, the states of other robots and the state of the planning robot, reactive planners plan for a single action to execute. Long horizon planners plan for multiple actions or long trajectories. Generally, reactive planners work fast ($> 50\,\mathrm{Hz}$), hence the effects of asynchronous planning are minimal. However, long horizon planners may take longer durations ($0.5\,\mathrm{Hz} - 10\,\mathrm{Hz}$) since the planned trajectory is longer. While the effects of asynchronous planning can be felt at any timescale, they are more readily visible in long horizon planners.

Reactive approaches. Optimal Reciprocal Collision Avoidance (ORCA) [5] is a reactive planner that emits velocity commands. Given the current positions and velocities of other robots, each robot computes a safe velocity command to execute that is as close as possible to the given preferred velocity command. Another reactive planner that accepts a preferred action and finds a safe action close to the desired action is described in [6]. They utilize safety barrier certificates to make sure that the robots will not leave safe spaces after applying the chosen actions. GLAS [7] is a learning-based reactive planner in which a neural network is trained to output robot actions given the states of other robots and the environment state. The neural network output is combined with a classical controller to ensure safety. Another learning-based approach [8] utilizes graph neural networks that learn what action to execute given the state of environment and what to communicate with other robots for collision avoidance. A third learning-based approach [9] outputs quadrotor motor commands given the states of other robots in the environment. Since reactive approaches plan for a single action to execute, they are prone to deadlocks and can be combined with long horizon planners for better performance. The approaches described above assume synchronized planning in which each robot uses the same snapshot of the environment. In these settings, since the planning frequency is high, the effects of asynchronous planning are minimized and often not readily visible in practice.

Long horizon approaches. BVC [4] is a model predictive control-based long horizon planner that utilizes buffered Voronoi cells (BVCs). In each planning iteration, each robot computes its Voronoi cell in the Voronoi tesselation of the environment, buffers it to account for robot's collision shape and plans actions to execute using a quadratic program making sure that the trajectory stays within the buffered Voronoi cell. Since each robot does the same, robot-robot collisions are avoided. Another approach [10] plans for piecewise Bézier curves using BVCs for collision avoidance. It plans for trajectories (parts of which may be outside the BVC), but makes sure that the trajectory is contained within the BVC until the next planning iteration. Another model predictive control approach is presented in [11]. They compute a feasible, smooth Bézier curve for each robot and use Bézier curve samples in the robot's controller behavior model and plan using a quadratic optimization formulation. Another optimization approach that plans for piecewise Bézier curves [12] ensures collision avoidance using relative safe flight corridors. The method of [13] extends that of [10] to robots with generalized collision shapes by utilizing support vector machines instead of Voronoi diagrams. Similar to the reactive approaches, these approaches assume synchronized planning. However, unlike reactive approaches, when planning is done in low frequency, these approaches may result in collisions. MADER [14] attempts to solve problems stemming from asynchronous planning using communication by utilizing a plan-commit scheme, in which each robot i plans a trajectory and commits it if no other robot planned a trajectory that collides with the planned trajectory of robot i during the planning process of robot i. However, they assume that the communication channel is failure and delay-free. When a robot sends a message, it is assumed that other robots receive it instantly without message drops. This is an unrealistic assumption for practical communication networks.

Centralized offline approaches exist to solve the multi-robot trajectory planning problem. We do not review them here since our focus is on decentralized real-time planning algorithms. As we will discuss, our method can be applied to several synchronized real-time decentralized multi-robot planning algorithms that enforce robot-robot collision avoidance using separating hyperplanes between robot geometries (examples include [4], [10], [13]), in order to extend them to asynchronous planning systems. In this paper, we show how to do this in practice with one such approach (based on [4]).

III. PROBLEM

Consider a team of N robots tasked with navigating from their start positions $\mathbf{s}_i \in \mathbb{R}^d$ to their corresponding goal positions $\mathbf{g}_i \in \mathbb{R}^d$ without collisions where $i \in \{1, \ldots, N\}$ is the index of a robot. The robots may potentially all have different shapes and sizes. We model robots as non-rotating rigid bodies. Let $\mathcal{R}_i : \mathbb{R}^d \to \mathcal{P}(\mathbb{R}^d)$ be the collision shape function of robot i, such that $\mathcal{R}_i(\mathbf{p})$ is the subset of $\mathbb{R}^d$ occupied by robot i when it is placed at position $\mathbf{p}$. Here, $d \in \{2, 3\}$ is the dimension of the Euclidean space that the robot operates in and $\mathcal{P}(\mathbb{R}^d)$ is the power set of $\mathbb{R}^d$. We define $\mathcal{R}_i(\mathbf{p})$ as the Minkowski sum $\mathcal{R}_i^{\mathbf{0}} \oplus \{\mathbf{p}\}$, where $\mathcal{R}_i^{\mathbf{0}}$ is the subset of $\mathbb{R}^d$ occupied by robot i when it is placed at the origin $\mathbf{0}$. For simplicity of exposition, we assume that the environment is obstacle-free.

The general problem is to compute trajectories $\mathbf{f}_i(t) : [0, T] \to \mathbb{R}^d$ for each robot i along with the team navigation duration T such that

- $\mathbf{f}_i(t)$ is dynamically feasible according to i^{th} robot's dynamics,
- $\mathcal{R}_i(\mathbf{f}_i(t)) \cap \mathcal{R}_j(\mathbf{f}_j(t)) = \emptyset \; \forall t \in [0, T] \; \forall i \neq j$, i.e. robots do not collide with each other,
- $\frac{d^k \mathbf{f}_i(0)}{dt^k} = \frac{d^k \mathbf{f}_i(T)}{dt^k} = \mathbf{0} \; \forall k \geq 1 \; \forall i \in \{1, \ldots, N\}$, i.e. robots are stationary at the start and the end of the computed trajectories,

- $\mathbf{f}_i(0) = \mathbf{s}_i, \mathbf{f}_i(T) = \mathbf{g}_i \ \forall i \in \{1, \ldots, N\}$, i.e. the computed trajectory for each robot commences at its start position and ends at the prescribed goal position.

We focus on a subset of decentralized receding horizon planning style approaches to solve the general problem above. In decentralized receding horizon planning, robots plan long trajectories in real-time for themselves using onboard capabilities. They execute the planned trajectories for a short duration, and re-plan. Our focus is on algorithms that utilize separating hyperplanes between robots as collision avoidance constraints.

Robots may communicate with each other using a best-effort communication medium, but each robot plans a trajectory *only* for itself. The delay of the communication medium need not be bounded and can theoretically grow to infinity. The delay may freely vary from message to message. Messages sent through the medium may get lost or reordered.

Planning start and end times across robots need not be synchronized and planning algorithms are not necessarily proven to be failure-free.

IV. PRELIMINARIES

A hyperplane $\mathcal{H}$ is defined by a normal vector $\mathcal{H}_\mathbf{n} \in \mathbb{R}^d$ and an offset $\mathcal{H}_a \in \mathbb{R}$ such that $\mathcal{H} = \{\mathbf{p} \in \mathbb{R}^d \mid \mathcal{H}_\mathbf{n} \cdot \mathbf{p} + \mathcal{H}_a = 0\}$. A hyperplane $\mathcal{H}$ bounds two half-spaces, namely $\mathcal{H}^+ = \{\mathbf{p} \in \mathbb{R}^d \mid \mathcal{H}_\mathbf{n} \cdot \mathbf{p} + \mathcal{H}_a \geq 0\}$ and $\mathcal{H}^- = \{\mathbf{p} \in \mathbb{R}^d \mid \mathcal{H}_\mathbf{n} \cdot \mathbf{p} + \mathcal{H}_a \leq 0\}$. Given two sets $\mathcal{A} \subset \mathbb{R}^d$ and $\mathcal{B} \subset \mathbb{R}^d$, a separating hyperplane $\mathcal{H}$ between $\mathcal{A}$ and $\mathcal{B}$ satisfies either $\mathcal{A} \subseteq \mathcal{H}^+$ and $\mathcal{B} \subseteq \mathcal{H}^-$ or $\mathcal{A} \subseteq \mathcal{H}^-$ and $\mathcal{B} \subseteq \mathcal{H}^+$, i.e. $\mathcal{A}$ is in one side of the hyperplane and $\mathcal{B}$ is in the other side. $\mathcal{A}$ and $\mathcal{B}$ are called linearly separable, if and only if there exists an hyperplane that separates them. If both $\mathcal{A}$ and $\mathcal{B}$ are disjoint non-empty convex sets, there exists a separating hyperplane between them by the separating hyperplane theorem.

Definition 1: **Commutative Deterministic Separating Hyperplane Computation Algorithm**. A commutative deterministic separating hyperplane computation algorithm Ω accepts two linearly separable sets $\mathcal{A}$ and $\mathcal{B}$ as arguments, and computes a separating hyperplane between $\mathcal{A}$ and $\mathcal{B}$ such that the computed hyperplanes do not depend on the order of the arguments, i.e. $\Omega(\mathcal{A}, \mathcal{B}) = \Omega(\mathcal{B}, \mathcal{A})$, and the computed hyperplane is same for each call with the same arguments, i.e. there is no randomization.

Definition 2: **Mutually Computable Separating Hyperplane**. Given two linearly separable sets $\mathcal{A}$ and $\mathcal{B}$ and an hyperplane $\mathcal{H}$ that separates them, $\mathcal{H}$ is called a mutually computable separating hyperplane for $\mathcal{A}$ and $\mathcal{B}$ if and only if there exists a commutative deterministic separating hyperplane computation algorithm Ω such that $\Omega(\mathcal{A}, \mathcal{B}) = \mathcal{H}$.

V. MOTIVATION

Commutative deterministic separating hyperplane computation algorithms are used by several state-of-the-art decentralized real-time receding horizon planning algorithms for collision avoidance. At each planning iteration, robots sense each others' geometries (positions and shapes). Each robot computes separating hyperplanes between itself and other robots using a shared commutative deterministic separating hyperplane computation algorithm. Robots constrain their movements to the regions bounded by the computed separating hyperplanes during planning. Since the commutative deterministic separating hyperplane computation algorithm is shared among robots, each pair of robots compute exactly the same hyperplane using only geometry sensing: each robot in the pair constrains itself against the other using this hyperplane. Therefore, regions used to constrain robot movements are disjoint between robots.

In [4] and [10] robots are modeled as hyperspheres and Voronoi diagrams are utilized for computing separating hyperplanes. Voronoi diagrams between hyperspherical shapes are unique, hence the algorithm is deterministic. For hyperspherical disjoint sets, Voronoi hyperplane computation algorithm is commutative, hence robots can compute the same hyperplane between each other using only geometry sensing. In [13] robots are modeled as convex shapes and a hard-margin support vector machine (SVM) is used for computing separating hyperplanes. The hard-margin SVM problem is convex with a unique solution. Therefore it is commutative and deterministic.

When robots are in close proximity to each other, the success of these approaches depends on two important assumptions that are hard to realize in practice: i) pairs of robots must use the same inputs during separating hyperplane computation - this holds only if the assumption of synchronization of planning start times between robots holds, and ii) robots start executing the plans at the same time, which assumes synchronization of planning end times between robots. In physical deployments, such synchronization is not realistic. This creates a mismatch between the separating hyperplanes computed and/or used by robots as shown in Fig. 2. The mismatch of hyperplanes introduces safety regions that intersect with each other, which results in unsafe behavior in tight scenarios.

We introduce mutually computable separating hyperplane *trajectories* as a solution to this problem. Instead of constraining a robot by a single hyperplane for each other robot, we constrain it with a carefully pruned trajectory of hyperplanes.

VI. APPROACH

A. Assumptions

We assume that robot clocks are synchronized prior to navigation (using for example, the Network Time Protocol (NTP) [15]). This ensures that the timestamps used by the robots share the same frame of reference. Second, we assume that robots can sense each others' geometries and can identify each other instantly. In reality, assuming visual sensing, there is a time delay after light hits the camera until the robot detects the geometry of another robot and identifies it. We omit this delay. Frame-by-frame tracking and identification of robots can be done in $30\,\mathrm{Hz}$ with state-of-the art object detectors [16]. Third, we assume that robots share a commutative deterministic separating hyperplane

(a) Initial robot geometries and enforced separating hyperplanes.

(b) Green hyperplane is used by the yellow robot during planning.

(c) Black region is considered safe by both robots when yellow robot finishes planning.

(d) Brown hyperplane is used by the blue robot during planning.

(e) Black region is considered safe by both robots when blue robot finishes planning.

Fig. 2. **Hyperplane mismatch leads to unsafe behavior.** Blue and yellow robots utilize Voronoi hyperplanes to create safe regions within which to plan. However, when planning start and end times between robots are not synchronized, the Voronoi hyperplanes computed do not match. Fig. 2a shows robots geometries (shapes and positions) and enforced Voronoi half-spaces, which are computed at the previous planning iteration, for each robot just before the yellow robot starts planning (assuming perfect synchronization before the current planning iteration). The green half-space in Fig. 2b is the Voronoi half-space that the yellow robot computes and utilizes during planning. While the yellow robot is planning its own trajectory, robots keep moving according to their previous plans. When the yellow robot finishes planning as shown in Fig. 2c, the safe regions of the blue and yellow robots intersect (intersection shown in black). Both robots are allowed to navigate through the black region, which would result in collisions if they did so. Robots keep moving before the blue robot starts planning. The brown half-space in Fig. 2d is the Voronoi half-space the blue robot utilizes during planning. While the blue robot is planning, robots continue moving. When the blue robot finishes planning (Fig. 2e), the safe regions of the robots intersect at another location.

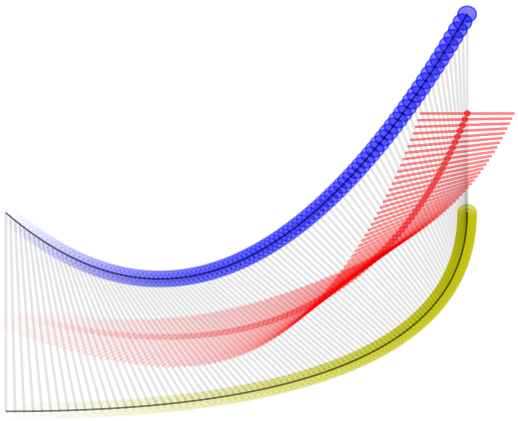

Fig. 3. **Hyperplane trajectory.** Sampled trajectories of blue and yellow robots are given. The red sampled trajectory is the trajectory of middle points of robots at each sampling step. Red hyperplanes constitute the sampled mutually computable Voronoi hyperplane trajectory between robots.

computation algorithm Ω, which is given to them prior to navigation.

B. Mutually Computable Separating Hyperplane Trajectories

Without the loss of generality, assume that the navigation starts at timestamp 0. Let $\tilde{T}$ be the current timestamp. Let $\mathbf{f}_i(t) : [0, \tilde{T}] \to \mathbb{R}^d$ and $\mathbf{f}_j(t) : [0, \tilde{T}] \to \mathbb{R}^d$ be the trajectories that robot i and robot j traversed until the current timestamp respectively. The mutually computable separating hyperplane trajectory $\mathcal{H}_{i,j}(t) : [0, \tilde{T}] \to \mathcal{H}^d$ between robot i and robot j induced by Ω is defined as

$$\mathcal{H}_{i,j}(t) = \Omega(\mathcal{R}_i(\mathbf{f}_i(t)), \mathcal{R}_j(\mathbf{f}_j(t))) \ \forall t \in [0, \tilde{T}]$$

where $\mathcal{H}^d$ is the set of all hyperplanes in $\mathbb{R}^d$.

Notice that $\mathcal{H}_{i,j}$ can be computed without communication by both robot i and robot j independently as it requires geometry sensing only. Also, $\mathcal{H}_{j,i} = \mathcal{H}_{i,j}$ since Ω is commutative. An example sampled mutually computable hyperplane trajectory is shown in Fig. 3 when the commutative deterministic hyperplane computation algorithm is the Voronoi hyperplane computation algorithm.

C. Overconstraining & Constraint-discarding using Mutually Computable Separating Hyperplane Trajectories and Inter-Robot Communication

Our approach is based on the following idea: If the trajectories of each pair (i, j) of robots are constrained by a shared hyperplane, collision avoidance between them can be ensured. In order to ensure that their trajectories are constrained by a shared hyperplane at all times, we constrain them with all hyperplanes in tail portions of the $\mathcal{H}_{i,j}$ in a specific way.

Each robot i stores a *tail timestamp variable* $T_{i,j} \leq \tilde{T}$ for every other robot j denoting the timestamp after which the hyperplane trajectory should be used to constrain the trajectory of robot i against robot j where $\tilde{T}$ is the current timestamp. Specifically, at any instant $\tilde{T}$, if robot i starts planning, it constrains its full trajectory with all hyperplanes $\mathcal{H}_{i,j}(t) \ \forall t \in [T_{i,j}, \tilde{T}]$ to avoid collision with robot j. Initially, we set $T_{i,j} = 0 \ \forall i \neq j$. Hence, robots use full mutually computable separating hyperplane trajectories to avoid collisions with each other. This ensures that robot i and robot j will not collide with each other for the initial values of $T_{i,j}$ since they are stationary until the end of their first planning iterations and they share $\mathcal{H}_{i,j}(0)$ afterwards.

Remark 1: For a given robot i, its planning start timestamps are strictly increasing.

Lemma 1: Assume that robot j successfully planned using information at time τ. The following facts hold for its constraints against every other robot i:

1) Robot j is constrained by the hyperplane $\mathcal{H}_{j,i}(\tau)$ (which is equal to $\mathcal{H}_{i,j}(\tau)$ since Ω is commutative).
2) In the future, there can be no case in which robot j uses a hyperplane from timespan $[0, \tau)$ to constrain itself against robot i in which it does not also use the hyperplane at time τ.

Proof: Since $T_{j,i}$ is less than or equal to the current timestamp $\tilde{T}$ at all times and $\tau = \tilde{T}$ at the start of the planning, robot j uses $\mathcal{H}_{j,i}(\tau)$ as a constraint. By Remark 1, planning start timestamps of robot j strictly increase. Therefore in the future, any planning start timestamp τ' will be greater than τ. If robot j uses any hyperplane from timespan

$[0, \tau)$ during planning in the future, it means that $T_{j,i} < \tau$. Since it uses hyperplanes in timespan $[T_{j,i}, \tau']$, it has to use the hyperplane at time τ as well. ■

Lemma 1 suggests that if a robot j plans successfully using the information at timestamp τ, there is no reason for another robot i to use hyperplanes against robot j before timestamp τ, because robot j is currently constrained by the hyperplane $\mathcal{H}_{i,j}(\tau)$ at time τ and there will not be any case in the future at which it is constrained by a hyperplane in timespan $[0, \tau)$ in which it is not constrained by the hyperplane at timestamp τ. Therefore, every other robot i can *prune* its hyperplane trajectory $\mathcal{H}_{i,j}$ against robot j by setting $T_{i,j} = \tau$. Updating $T_{i,j}$ with this rule ensures that trajectories of each pair (i, j) of robots share a constraining hyperplane at all times because i) $\mathcal{H}_{i,j}(0)$ is shared by robots i and j initally, and ii) pruning is done in a way that makes sure that there is at least one shared hyperplane at all times. Sharing a constraining hyperplane at all times guarantees collision avoidance.

Conveying the information to every other robot i that robot j planned successfully at timestamp τ is done through the communication medium. Whenever robot j successfully plans a trajectory, it broadcasts the planning success signal (j, τ), stating that it planned successfully using the information at timestamp τ, and there is no reason for other robots to constrain themselves against robot j using hyperplanes in timespan $[0, \tau)$. Until the message arrives, every other robot i *overconstrains* itself against robot j with hyperplanes in $[T_{i,j}, \tau) \cup [\tau, \tilde{T}]$, but *discards constraints* generated from hyperplanes in timespan $[T_{i,j}, \tau)$ when message arrives by setting $T_{i,j} = \tau$. When message re-ordering in the communication medium is possible, $T_{i,j} = \max(T_{i,j}, \tau)$ is used.

If a planning success signal from robot j gets lost in the communication medium, other robots i do not update their $T_{i,j}$ values and keep overconstraining themselves against robot j until a signal from robot j arrives. This allows robots to assume the worst about robot j (i.e., it planned successfully at $T_{i,j}$ and it is not known if it planned successfully again in timespan $(T_{i,j}, \tilde{T}]$) until a message arrives. $T_{i,j}$ represents the last known timestamp to robot i at which robot j has planned a trajectory successfully.

Note that, if an upper bound U exists on the time difference between two successful planning iterations for all robots, constraints can be discarded even without communication. Here, $T_{i,j}$ is the timestamp at which robot j has planned (or after which robot j must have planned) according to robot i. Robot i simply makes sure that $\tilde{T} - T_{i,j} \leq U$ by setting $T_{i,j} = \max(T_{i,j}, \tilde{T} - U)$ at each iteration for asynchronous safety because robot j plans successfully at least once in every time window of duration U.

VII. EXPERIMENTS

A. Buffered Voronoi Cell (BVC) Planner

We validate our approach using a decentralized model predictive control-based planner that uses Voronoi diagrams for collision avoidance [4]. It models robots as hyperspheres parameterized by robot radii. At each planning iteration, Voronoi hyperplanes between robots are computed, buffered to account for robot radii, and used to constrain robot positions. The original formulation of the BVC planner uses discrete single integrator dynamics; we use its extension to discrete linear dynamics with position outputs. Each robot i solves the following convex quadratic program during planning, making sure that the output (position) is contained in the buffered Voronoi cell of the robot.

$$\min_{\mathbf{u}_0,\dots,\mathbf{u}_{M-1}} \sum_{k=1}^{M} \lambda_k \left\|\mathbf{p}_k - \mathbf{g}_i\right\|_2^2 + \sum_{k=0}^{M-1} \theta_k \left\|\mathbf{u}_k\right\|_2^2 s.t.$$

$$\begin{aligned}
\mathbf{x}_{k+1} &= \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k && \forall k \in \{0,\dots,M-1\}\\
\mathbf{p}_k &= \mathbf{C}\mathbf{x}_k && \forall k \in \{0,\dots,M\}\\
\mathcal{R}_i(\mathbf{p}_k) &\in \mathcal{V}_i && \forall k \in \{0,\dots,M\}\\
\mathbf{u}_{min} &\preceq \mathbf{u}_k \preceq \mathbf{u}_{max} && \forall k \in \{0,\dots,M-1\}\\
\mathbf{x}_{min} &\preceq \mathbf{x}_k \preceq \mathbf{x}_{max} && \forall k \in \{0,\dots,M\}
\end{aligned}$$

where $\mathbf{g}_i$, $\mathcal{V}_i$ and $\mathcal{R}_i$ are the goal position, Voronoi cell, and hyperspherical collision shape function of robot i, M is the number of steps to plan for (or the planning horizon), $\mathbf{u}_k$ is the input to apply from timestep k to timestep $k+1$, $\mathbf{x}_k$ is the state of robot at timestep k, $\mathbf{p}_k$ is the position of robot at timestep k, $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ are matrices that describe the dynamics of the robot, $\mathbf{u}_{min}$ and $\mathbf{u}_{max}$ are the limits for inputs, $\mathbf{x}_{min}$ and $\mathbf{x}_{max}$ are the limits for states. The cost of the optimization problem is a weighted combination of the distance of intermediate and final robot positions to the goal position and the input magnitudes. λ_ks and θ_ks are weight parameters of the cost function. The constraint $\mathcal{R}_i(\mathbf{p}_k) \in \mathcal{V}_i$ is enforced by buffering the Voronoi hyperplanes with the robot radius, and constraining the $\mathbf{p}_k$ with the buffered Voronoi hyperplanes.

B. Asynchronous Buffered Voronoi Cell (AsyncBVC) Planner

In order to ensure safety of BVC under asynchronous planning, we integrate mutually computable separating hyperplane trajectories to the BVC planner by replacing the Voronoi cells with the Voronoi hyperplane trajectories.

Each robot uses Voronoi hyperplane computation algorithm as the commutative deterministic hyperplane computation algorithm Ω. They construct mutually computable separating hyperplane trajectories, or Voronoi hyperplane trajectories specifically, as described in Section VI-B and keep track of tail timestamp variables as described in Section VI-C. We call this algorithm asynchronous buffered Voronoi cell planner, or AsyncBVC for short.

Each robot i solves the following optimization problem:

$$\min_{\mathbf{u}_0,\dots,\mathbf{u}_{M-1}} \sum_{k=1}^{M} \lambda_k \left\|\mathbf{p}_k - \mathbf{g}_i\right\|_2^2 + \sum_{k=0}^{M-1} \theta_k \left\|\mathbf{u}_k\right\|_2^2 s.t.$$

$$\begin{aligned}
\mathbf{x}_{k+1} &= \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k && \forall k \in \{0,\dots,M-1\}\\
\mathbf{p}_k &= \mathbf{C}\mathbf{x}_k && \forall k \in \{0,\dots,M\}\\
\mathcal{R}_i(\mathbf{p}_k) &\in \mathcal{H}_{i,j}^{-}(t) \;\; \forall k \in [0,\dots,M] \;\; \forall j \neq i \;\; \forall t \in [T_{i,j}, \tilde{T}]\\
\mathbf{u}_{min} &\preceq \mathbf{u}_k \preceq \mathbf{u}_{max} && \forall k \in \{0,\dots,M-1\}\\
\mathbf{x}_{min} &\preceq \mathbf{x}_k \preceq \mathbf{x}_{max} && \forall k \in \{0,\dots,M\}
\end{aligned}$$

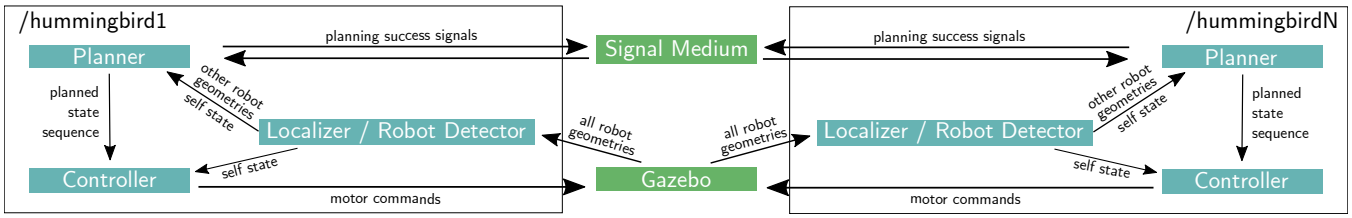

Fig. 4. **System design.** Each robot has 3 components: a planner, a controller and a localizer/robot detector. The planner plans for a sequences of states, which gets fed into the controller. The controller sends motor commands to the simulator (Gazebo) to track the planned sequence. The localizer/robot detector connects directly to the simulator and feeds other robot geometries and ego robot state to the planner and the controller. A signal medium process acts as a message broker between planners of different robots. We introduce communication delays and message drops using the signal medium process.

where $\mathcal{H}^-_{i,j}(t)$ is the robot i's side of the hyperplane $\mathcal{H}_{i,j}(t)$ against robot j and $\tilde{T}$ is the current timestamp. Similarly to BVC, the constraint $\mathcal{R}_i(\mathbf{p}_k) \in \mathcal{H}^-_{i,j}(t)$ is enforced by buffering the hyperplane with the robot radius, and constraining the $\mathbf{p}_k$ with the buffered hyperplane.

This is a convex quadratic problem with linear constraints. However, since time is continuous, there are infinitely many hyperplanes in the trajectory $\mathcal{H}_{i,j}$, which results in infinitely many constraints in the optimization problem. We solve this problem by using a sequence of hyperplanes between robot i and robot j using a fine discretization of the full trajectory. Whenever robot i detects the geometry of and identifies another robot j, we compute a new separating hyperplane between them using Ω and add it to the separating hyperplane list of robot i against robot j. This is an approximation we employ for computational tractability. Whenever robot i receives a planning success signal (j, τ) from robot j, we discard hyperplanes that are generated before τ from the sequence of robot i against robot j.

If the optimization fails for a robot i, it keeps using the previous plan. When robot i's planner fails, it does not broadcast a planning success signal, and other robots keep constraining themselves against robot i using the constraints starting from and including the timestamp robot i last succeeded. Hence, the collisions are avoided even under a planner failure, since at any point of time, plans of any pair of robots are constrained by at least one common separating hyperplane.

C. Experiment Design

We compare the behavior of BVC and AsyncBVC when message drops, delays, re-orderings and timestamp mismatches are introduced using simulations. We use the RotorS MAV Gazebo simulator [17] running on a desktop computer with 16 core Intel i9-9900 CPU @ 3.10GHz, Ubuntu 20.04 operating system and ROS Noetic. We plan in 3D using AscTec Hummingbird quadrotors already integrated into the RotorS simulator.

In all experiments, 4 robots are placed in a square formation where they are at $[-10\ 0\ 5.0]^T$, $[10\ 0\ 5.0]^T$, $[0\ -10\ 5.0]^T$, $[0\ 10\ 5.0]^T$ initially. Robots at the opposite ends of the square swap positions.

The general design of our simulation system is shown in Fig. 4. Each robot's planning pipeline has 3 main components: a planner, a controller and a localizer/robot detector. The planner is set to either BVC or AsyncBVC. In all experiments, we use spheres with radii $r = 0.4\,\mathrm{m}$ to model robots. The real radius of AscTec Hummingbirds is $0.27\,\mathrm{m}$. We utilize extra $0.13\,\mathrm{m}$ as the buffer zone to compensate for the controller trajectory tracking errors. In all experiments, we use discretized double integrator dynamics during planning for the robots hence control actions $\mathbf{u}_k$ are acceleration commands, and states $\mathbf{x}_k$ are stacked position and velocity vectors in both BVC and AsyncBVC. We set input upper and lower bounds to $\mathbf{u}_{max} = -\mathbf{u}_{min} = [5\,\mathrm{m}/s^2\ 5\,\mathrm{m}/s^2\ 5\,\mathrm{m}/s^2]^T$, state upper bounds to $\mathbf{x}_{max} = [\infty\ \infty\ \infty\ 2\,\mathrm{m}/s\ 2\,\mathrm{m}/s\ 2\,\mathrm{m}/s]^T$ and state lower bounds to $\mathbf{x}_{min} = [-\infty\ -\infty\ r\ -2\,\mathrm{m}/s\ -2\,\mathrm{m}/s\ -2\,\mathrm{m}/s]^T$ i.e. the maximum acceleration along any axis is $5\,\mathrm{m/s^2}$, maximum velocity along any axis is $2\,\mathrm{m/s}$, and all positions except those with z coordinates less than the robot radii r are allowed. We plan for 20 steps with discretization timestep of $0.2\,\mathrm{s}$ (i.e. $4\,\mathrm{s}$ long trajectories). We set $\lambda_k = 2$ for all k and $\theta_k = 1$ for all k. The planner sends planned state sequences to the controller for execution. We use one of the controllers integrated into RotorS simulator which is based on [18]. It sends motor commands to the simulator process to track the planned state sequence. The localizer/robot detector component simply receives perfect state and shape information from Gazebo simulator and feeds them to the planner and controller. The frequency of robot detection is $30\,\mathrm{Hz}$, which results in discretization step length of $33\,\mathrm{ms}$ for the separating hyperplane trajectories. Planners on different robots send planning success signals to each other through the signal medium process, which acts as a message broker between planners simulating a communication restrictive environment. BVC does not need communication between planners, therefore signal medium is not utilized when the planner is BVC. When the planner is set to AsyncBVC, it uses the signal medium to broadcast planning success signals described in Section VI-C. We introduce artificial communication delays, message drops and message re-orderings to the system using the signal medium process. We model communication delays using exponential probability density functions of the form

$$P_{delay}(x;\alpha) = \begin{cases} \alpha e^{-\alpha x} & x \geq 0 \\ 0 & x < 0 \end{cases},$$

where $1/\alpha \geq 0$ is the mean delay. The most probable delay in this distribution is 0, and the probability decreases as the delay increases. Message re-orderings are naturally

generated by different per-message delays generated from the given delay distribution.

We model message drop probability with a Bernoulli distribution of the form

$$P_{drop}(x;\beta) = \begin{cases} \beta & x \text{ is drop} \\ 1-\beta & x \text{ is no-drop.} \end{cases}$$

D. Evaluation Metrics

We evaluate the performance of algorithms using 4 metrics: number of robots that are involved in at least one collision during navigation (# Coll.), number of robots that reach their goal positions (# Goal Reaching), number of robots that get stuck in synchronization-induced deadlocks (# Deadlocks), and the maximum navigation time among robots from their start to goal position (Makespan).

Synchronization-induced deadlocks occur because of separating hyperplane mismatches between robots when they are close to each other as shown in Fig 5. Since we model robots with spheres larger than the actual robot dimensions (which we call collision shapes), planners plan for avoiding the spheres containing the actual robots. When a separating hyperplane mismatch occurs when robots are close to each other, their collision shapes may intersect with each other even if the real robots do not collide. This creates a situation at which planners cannot find collision free solutions because the robot is initially in a collision state. This results in a deadlock because the planner fails continuously. We call this a synchronization-induced deadlock.

In the simulations, robots that are involved in collisions can continue navigation because the controller can recover from them as the speed of the robots is not high.

E. Effects of Planning Start and End Time Mismatch

First, we check the behavior of BVC and AsyncBVC when the planning start and end times do not match between robots. We change the planning frequency f to create this mismatch. If planning frequency of robots is set to f, a planning iteration occurs every $1/f$ seconds. This creates a maximum of $1/f$ seconds mismatch between planning start times of a pair robots. The planning end time is the sum of planning start time and planning duration. There is no bound on planning duration, but it does not take longer than 100 ms in our experiments, thereby creating a mismatch between planning end times no more than $1/f + 0.1$ seconds. Note that we do not use this value as an upper bound between the timestamps of two successful planning iterations because there is no guarantee that the planner always succeeds. We thus set the upper bound to ∞. We set mean delay time $1/\alpha = 0$ and message drop probability $\beta = 0$ in all experiments, meaning that the network has no delay and does not drop messages, to show the affects of only planning start/end time mismatches.

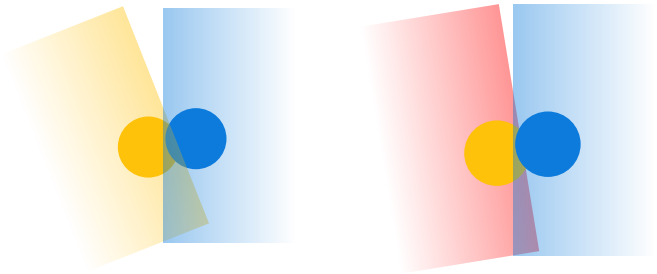

Fig. 5. **Synchronization-induced deadlock due to separating hyperplane mismatch.** (left) Robots go into a collision state because of a separating hyperplane mismatch and (right) planning fails for the yellow robot because it initially violates the new red hyperplane.

TABLE I

EFFECTS OF PLANNING START/END TIME MISMATCH. METRICS ARE AVERAGED OVER 5 RUNS IN EACH SCENARIO.

	Freq.	2 Hz	1 Hz	0.5 Hz
	Max. Start Mismatch	0.5 s	1 s	2 s
BVC	# Coll.	0	2	2.4
	# Deadlocks	0	0.4	2.2
	# Goal Reaching	4	3.6	1.8
AsyncBVC	# Coll.	0	**0**	**0**
	# Deadlocks	0	**0**	**0**
	# Goal Reaching	4	**4**	**4**

The results of our comparisons are listed in Table I. We run each experiment 5 times and report the average of the metrics across runs. We do not report the makespan of navigation between algorithms because the difference is not significant when the communication is instantaneous and failure-free.

In experiment 1, we set planning frequency to 2 Hz, which creates a maximum planning start time mismatch of 0.5 s. In this case, neither BVC nor AsyncBVC results in any synchronization-induced deadlocks or collisions and all robots reach their goal positions. In experiment 2, we decrease planning frequency to 1 Hz, which creates a maximum planning start time mismatch of 1 s. In this case, 2 out of 4 robots using BVC collide at least once on average, 0.4 out of 4 robots deadlock on average, and 3.6 out of 4 robots reach their goals on average. AsyncBVC does not result in any collisions or deadlocks. In experiment 3, we decrease planning frequency further to 0.5 Hz, which creates a maximum planning start time mismatch of 2 s. BVC suffers from this mismatch siginficantly. On average 2.4 out of 4 robots collide at least once, and 2.2 out of 4 get stuck in synchronization-induced deadlocks, leaving 1.8 robots on average reaching their goals. AsyncBVC does not result in any collisions or deadlocks.

F. Effects of Best-Effort Communication Medium

Next, we investigate the effects of communication delays and message drops on the performance of AsyncBVC. We do not investigate the behavior of BVC in this set of experiments because BVC does not utilize communication. We set planning frequency $f = 1$ Hz in all experiments which creates a maximum planning start time mismatch of 1 s. As before, we set upper bound of time difference between successful planning iterations to ∞.

The results of our experiments are summarized in Table II. As before, we run each experiments 5 times and report the average of the metrics across runs. In experiment 1, we show the performance of AsyncBVC when there

TABLE II

EFFECTS OF COMMUNICATION DELAYS AND MESSAGE DROPS TO ASYNCBVC. METRICS ARE AVERAGED OVER 5 RUNS IN EACH SCENARIO. $1/\alpha$ IS THE MEAN DELAY AND β IS THE MESSAGE DROP PROBABILITY.

#	$1/\alpha$	β	# Coll.	# Deadlocks	# Goal Reaching	Makespan
1	0	0	0	0	4	20.82 s
2	1 s	0	0	0	4	25.57 s
3	1 s	0.1	0	0	4	26.68 s
4	2 s	0.1	0	0	4	28.31 s
5	2 s	0.2	0	0	4	28.92 s
6	10 s	0.2	0	0	4	40.88 s
7	10 s	0.5	0	0	4	48.04 s
8	10 s	0.75	0	0	4	92.26 s
9	—	1.0	0	4	0	∞

is no communication delay, i.e. mean delay is 0, and no message drops, i.e. message drop probability is 0. In this case, no robots collide with each other, and no robots get stuck in deadlocks. The makespan of the navigation, i.e. the maximum navigation duration of all robots, is 20.82 s. From experiment 2 to experiment 8, we increase mean delay from 0 to 10 s and message drop probability from 0 to 0.75. In experiment 8, for example, 3 out of 4 messages get lost in the medium, and 1 that does not get lost arrives 10 s late on average. In all experiments from 2 to 8, no robots collide with each other, and no robots deadlock. Makespan increases from 25.57 s (experiment 2) to 92.26 s (experiment 8).

When the message drop probability is set to 1.0 in experiment 9, i.e. when all messages are dropped, all robots get deadlocked, because they are overly constrained by the full separating hyperplane trajectories. This problem can be solved if there is an upper bound on time difference between two successful planning iterations for the robots in the team as explained in Section VI-C.

VIII. CONCLUSION

In this paper, we present a novel overconstraining and constraint-discarding method for asynchronous real-time decentralized trajectory planning algorithms in multi-robot teams. Each robot overly constrains itself against other robots using mutually computable separating hyperplane trajectories until it receives a planning success signal from them. A portion of the constraints are discarded after a success signal is received from another robot. Our approach is a principled way for dealing with asynchronous planning and imperfect communication in real-time decentralized multi-robot systems.

Based on these ideas, we extend the BVC planner [4], which assumes synchronization in planning, to adapt it to an asynchronous setting. We call our extension AsyncBVC. We compare AsyncBVC with BVC and show that AsyncBVC does not result in any collisions or synchronization-induced deadlocks, while BVC is prone to these phenomena. We also demonstrate the encouraging behavior of AsyncBVC under message drops, message re-orderings and message delays.

In the future, we plan to apply our overconstraining and constraint-discarding method to other planners that utilize separating hyperplanes between robots for robot-robot collision avoidance.

ACKNOWLEDGMENT

We thank James A. Preiss, Wolfgang Hönig, and Jingyao Ren for their inputs and comments on writing of this paper.

REFERENCES

[1] P. Wurman, R. D'Andrea, and M. Mountz, "Coordinating hundreds of cooperative, autonomous vehicles in warehouses," *AI Magazine*, vol. 29, pp. 9–20, 2008.

[2] A. Furda and L. Vlacic, "Enabling safe autonomous driving in real-world city traffic using multiple criteria decision making," *IEEE Intelligent Transportation Systems Magazine*, vol. 3, no. 1, pp. 4–17, 2011.

[3] K. Dresner and P. Stone, "A multiagent approach to autonomous intersection management," *Journal of Artificial Intelligence Research*, vol. 31, pp. 591–656, 2008.

[4] 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.

[5] J. Alonso-Mora, A. Breitenmoser, M. Rufli, P. Beardsley, and R. Siegwart, "Optimal reciprocal collision avoidance for multiple non-holonomic robots," in *Distributed Autonomous Robotic Systems: The 10th International Symposium*, 2013, pp. 203–216.

[6] L. Wang, A. D. Ames, and M. Egerstedt, "Safety barrier certificates for collisions-free multirobot systems," *IEEE Transactions on Robotics*, vol. 33, no. 3, pp. 661–674, 2017.

[7] B. Riviere, W. Hönig, Y. Yue, and S.-J. Chung, "GLAS: Global-to-local safe autonomy synthesis for multi-robot motion planning with end-to-end learning," *IEEE Robotics and Automation Letters*, vol. PP, pp. 1–1, 2020.

[8] Q. Li, F. Gama, A. Ribeiro, and A. Prorok, "Graph neural networks for decentralized multi-robot path planning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 11 785–11 792.

[9] S. Batra, Z. Huang, A. Petrenko, T. Kumar, A. Molchanov, and G. S. Sukhatme, "Decentralized control of quadrotor swarms with end-to-end deep reinforcement learning," in *5th Conference on Robot Learning, CoRL 2021*. PMLR, 2021.

[10] B. Şenbaşlar, W. Hönig, and N. Ayanian, "Robust trajectory execution for multi-robot teams using distributed real-time replanning," in *Distributed Autonomous Robotic Systems (DARS)*, 2019, pp. 167–181.

[11] C. Luis, M. Vukosavljev, and A. Schoellig, "Online trajectory generation with distributed model predictive control for multi-robot motion planning," *IEEE Robotics and Automation Letters*, vol. PP, pp. 1–1, 2020.

[12] 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, 2021.

[13] B. Şenbaşlar, W. Hönig, and N. Ayanian, "RLSS: Real-time Multi-Robot Trajectory Replanning using Linear Spatial Separations," p. arXiv:2103.07588, 2021. [Online]. Available: https://arxiv.org/abs/2103.07588

[14] J. Tordesillas and J. P. How, "MADER: Trajectory planner in multi-agent and dynamic environments," *IEEE Transactions on Robotics*, 2021.

[15] D. Mills, J. Martin, J. Burbank, and W. Kasch, "Network time protocol version 4: Protocol and algorithms specification," Internet Requests for Comments, RFC 5905, June 2010. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc5905

[16] A. Bochkovskiy, C. Wang, and H. M. Liao, "YOLOv4: Optimal speed and accuracy of object detection," *CoRR*, vol. abs/2004.10934, 2020. [Online]. Available: https://arxiv.org/abs/2004.10934

[17] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart, *Robot Operating System (ROS): The Complete Reference (Volume 1)*. Cham: Springer International Publishing, 2016, ch. RotorS—A Modular Gazebo MAV Simulator Framework, pp. 595–625.

[18] T. Lee, M. Leok, and N. H. McClamroch, "Geometric tracking control of a quadrotor uav on se(3)," in *49th IEEE Conference on Decision and Control (CDC)*, 2010, pp. 5420–5425.