MADER: Trajectory Planner in Multi-Agent and Dynamic Environments

Jesus Tordesillas and Jonathan P. How

Abstract—This paper presents MADER, a 3D decentralized and asynchronous trajectory planner for UAVs that generates collision-free trajectories in environments with static obstacles, dynamic obstacles, and other planning agents. Real-time collision avoidance with other dynamic obstacles or agents is done by performing outer polyhedral representations of every interval of the trajectories and then including the plane that separates each pair of polyhedra as a decision variable in the optimization problem. MADER uses our recently developed MINVO basis ([1]) to obtain outer polyhedral representations with volumes $2.36$ and $254.9$ times, respectively, smaller than the Bernstein or B-Spline bases used extensively in the planning literature. Our decentralized and asynchronous algorithm guarantees safety with respect to other agents by including their committed trajectories as constraints in the optimization and then executing a collision check-recheck scheme. Finally, extensive simulations in challenging cluttered environments show up to a $33\%$ reduction in the flight time, and a $88.8\%$ reduction in the number of stops compared to the Bernstein and B-Spline bases, shorter flight distances than centralized approaches, and shorter total times on average than synchronous decentralized approaches.

Index Terms—UAV, Multi-Agent, Trajectory Planning, MINVO basis, Optimization.

Supplementary Material

The code and the video associated with this paper will be released soon (waiting for approval).

I. INTRODUCTION AND RELATED WORK

While efficient and fast UAV trajectory planners for static worlds have been extensively proposed in the literature [2]–[14], a 3D real-time planner able to handle environments with static obstacles, dynamic obstacles and other planning agents still remains an open problem (Fig. 1).

To be able to guarantee safety, the trajectory of the planning agent and the ones of other obstacles/agents need to be encoded in the optimization (see Fig. 2). A common representation of this trajectory in the optimization is via points discretized along the trajectory [15]–[20]. However, this does not usually guarantee safety between two consecutive discretization points and alleviating that problem by using a fine discretization of the trajectory can lead to a very high computational burden. To reduce this computational burden, polyhedral outer representations of each interval of the trajectory are extensively used in the literature, with the added benefit of ensuring safety at all times (i.e., not just at the discretization points). A common way to obtain this polyhedral outer representation is via the convex hull of the control points of the Bernstein basis...
(basis used by Bézier curves) or the B-Spline basis [10], [13], [21], [22]. However, these bases do not yield very tight (i.e., with minimum volume) tetrahedra that enclose the curve, leading to conservative results.

MADER addresses this conservatism at its source and leverages our recently developed MINVO basis [1] to obtain control points that generate the \( n \)-simplex (a tetrahedron for \( n = 3 \)) with the minimum volume that completely contains each interval of the curve. Global optimality of this tetrahedron obtained by the MINVO basis is guaranteed both in position and velocity space.

When other agents are present, the problem between the trajectories also needs to be solved. Most of the state-of-the-art approaches either rely on centralized algorithms [23]–[25] and/or on imposing an ad-hoc priority such that an agent only avoids other agents with higher priority [26]–[30]. Some decentralized solutions have also been proposed [28], [31], [32], but they require synchronization between the replans of different agents. The challenge then is how to create a decentralized and asynchronous planner that solves the deconfliction problem and guarantees safety and feasibility for all the agents.

MADER solves this deconfliction in a decentralized and asynchronous way by including the trajectories other agents have committed to as constraints in the optimization. After the optimization, a collision check-recheck scheme ensures that the trajectory found is still feasible with respect to the trajectories other agents have committed to while the optimization was happening.

To impose collision-free constraints in the presence of static obstacles, a common approach is to first find convex decompositions of free space and then force (in the optimization problem) the outer polyhedral representation of each interval to be inside these convex decompositions [10], [33], [34]. However, this approach can be conservative, especially in cluttered environments in which the convex decomposition algorithm may not find a tight representation of the free space. In the presence of dynamic obstacles, these convex decompositions become harder, and likely intractable, due to the extra time dimension.

To be able to impose collision-free constraints with respect to dynamic obstacles/agents, MADER imposes the separation between the polyhedral representations of each trajectory via planes. Moreover, MADER overcomes the conservatism of a convex decomposition (imposed ad-hoc before the optimization) by including a parameterization of these separating planes as decision variables in the optimization problem. The solver can thus choose the optimal location of these planes to determine collision avoidance. Including this plane parameterization reduces conservatism, but it comes at the expense of creating a nonconvex problem, for which a good initial guess is imperative. For this initial guess, we present a search-based algorithm that handles dynamic environments and obtains both the control points of the trajectory and the planes that separate it from other obstacles/agents.

The contributions of this paper are therefore summarized as follows (see also Fig. 2):

- Decentralized and asynchronous planning framework that solves the deconfliction between the agents by imposing as constraints the trajectories other agents have committed to, and then doing a collision check-recheck scheme to guarantee safety with respect to trajectories other agents have committed to during the optimization time.
- Collision-free constraints are imposed by using a novel polynomial basis in trajectory planning: the MINVO basis. In position space, the MINVO basis yields a volume 2.36 and 254.9 times smaller than the extensively-used Bernstein and B-Spline bases, respectively.
- Formulation of the collision-free constraints with respect to other dynamic obstacles/agents by including the planes that separate the outer polyhedral representations of each interval of every pair of trajectories as decision variables.
- Extensive simulations and comparisons with state-of-the-art baselines in cluttered environments. The results show up to a 33% reduction in the flight time, a 88.8% reduction in the number of stops (compared to Bernstein/B-Spline bases), shorter flight distances than centralized approaches, and shorter total times on average than synchronous decentralized approaches.
Table I: Notation used in this paper

| Symbol | Meaning |
|--------|---------|
| p, v, a, j | Position, Velocity, Acceleration and Jerk, $\in \mathbb{R}^3$. |
| $\mathbf{x}$ | State vector: $\mathbf{x} := [\mathbf{p}^T \; \mathbf{v}^T \; \mathbf{a}^T]^T, \quad \in \mathbb{R}^p$ |
| $m$ | $m + 1$ is the number of knots of the B-Spline. |
| $n$ | $n + 1$ is the number of control points of the B-Spline. |
| $p$ | Degree of the polynomial of each interval of the B-Spline. In this paper we will use $p = 3$. |
| $J$ | Set that contains the indexes of all the intervals of a B-Spline $J := \{0, 1, ..., m - 2p - 1\}$. |
| $u$ | $u := \text{Number of agents + Number of obstacles}$ |
| $s$ | Index of the planning agent. |
| $l$ | $l$ is the index of the control point. $l \in L$ for position, and $l \in L_\{n - 1, n\}$ for acceleration. |
| $I$ | Index of the obstacle/agent, $i \in I$. |
| $L_0$ | $L = \{0, 1, ..., u\}$. |
| $c$ | Vertex of a polyhedron, $\in \mathbb{R}^3$. |
| $\eta_s$ | Each entry $\eta_s$ is the length of each side of the AABB of the planning agent (i.e. agent whose index is $s$), $\in \mathbb{R}^3$. |
| $\xi_{ij}$ | Set of vertices of the polyhedron that completely encloses the trajectory of the obstacle/agent $i$ during the initial and final times of the interval $j$ of the agent $s$. |
| $\psi$ | $\psi$ is the length of each side of the axis-aligned box (AABB) of the agent/obstacle $i$. |
| $q, v, a$ | Position, velocity, and acceleration control points, $\in \mathbb{R}^3$. |
| $b$ | $b$ is the notation for the basis used: MINVO ($b = \text{MV}$), Bernstein ($b = \text{Bo}$), or B-Spline ($b = \text{BS}$). |
| $Q_b^{ij} := \{q_{j1}, ..., q_{j+3}\}$ | Set of position control points of interval $j$ of the trajectory of the agent $s$ using the basis $b$. Analogous definition for the velocity control points: $V_b^{ij} := \{v_{j1}, ..., v_{j+3}\}$. |
| $Q_{b}^{ij} := [q_{j1} \ldots q_{j+3}]$ | Matrix whose columns contain the position control points of the interval $j$ of the trajectory of the agent $s$ using the basis $b$. Analogous definition for the velocity control points: $V_{b}^{ij} := [v_{j1} \ldots v_{j+3}]$. |
| $\pi_{ij}$ | Plane $\pi_{ij}$, $x + d_{ij} = 0$ that separates $\xi_{ij}$ from $Q_{b}^{ij}$. |
| $\mathbf{l}$ | $\mathbf{l}$ is an abs. value, element-wise inequality, Minkowski sum and convex hull. |
| $\alpha \leq b, \in \mathbb{R}$ | Column vector of ones, element-wise absolute value, element-wise inequality, Minkowski sum and convex hull. |
| conv( ) | Unless otherwise noted, this colormap in the trajectories will represent the norm of the velocity (blue 0 m/s and red $v_{\text{max}}$). |

II. DEFINITIONS

This paper will use the notation shown in Table I, together with the following two definitions:

- **Agent:** Element of the environment with the ability to exchange information and take decisions accordingly (i.e., an agent can change its trajectory given the information received from the environment).

- **Obstacle:** Element of the environment that moves on its own without consideration of the trajectories of other elements in the environment. An obstacle can be static or dynamic.

Note that here we are calling dynamic obstacles what some works in the literature call non-cooperative agents.

This paper will also use clamped uniform B-Splines, which are B-Splines defined by $n + 1$ control points $\{q_0, \ldots, q_n\}$ and $m + 1$ knots $\{t_0, t_1, \ldots, t_m\}$ that satisfy:

$$t_0 = \cdots = t_p < t_{p+1} < \cdots < t_{m-p-1} < t_{m-p} = \cdots = t_m$$

and where the internal knots are equally spaced by $\Delta t$ (i.e. $\Delta t := t_{k+1} - t_k$, $\forall k = \{p, \ldots, m - p - 1\}$). The relationship $m = n + p + 1$ holds, and there are in total $m - 2p = n + p + 1$ intervals. Each interval $j \in J$ is defined in $t \in [t_{p+j}, t_{p+j+1}]$. In this paper we will use $p = 3$ (i.e. cubic B-Splines). Hence, each interval will be a polynomial of degree 3, and it is guaranteed to lie within the convex hull of its 4 control points $\{q_j, q_{j+1}, q_{j+2}, q_{j+3}\}$. Moreover, clamped B-Splines are guaranteed to pass through the first and last control points ($q_0$ and $q_n$). The velocity and acceleration of a B-Spline are B-Splines of degrees $p - 1$ and $p - 2$ respectively, whose control points are given by ([13]):

$$v_l = \frac{p}{t_{l+p+1} - t_{l+1}} (q_{l+1} - q_l), \quad \forall l \in L \setminus \{n\}$$

$$a_l = \frac{(p - 1)}{t_{l+p+1} - t_{l+2}} (v_{l+1} - v_l), \quad \forall l \in L \setminus \{n - 1, n\}$$

III. ASSUMPTIONS

This paper relies on the following three assumptions:

- Let $p_{i}^{\text{real}}(t)$ denote the real future trajectory of an obstacle $i$, and $p_{i}(t)$ the one obtained by a given tracking and prediction algorithm. The smallest dimensions of the axis-aligned box $D_{ij}$ for which $p_{i}^{\text{real}}(t) \in \text{conv} \left(D_{ij} \oplus p_{i}(T_j) \right)$ $\forall t \in [t_{p+j}, t_{p+j+1}]$ is satisfied will be denoted as $2 (\alpha_{ij} + \beta_{ij}) \in \mathbb{R}^3$. Here $\alpha_{ij}$ represents the error associated with the prediction and $\beta_{ij}$ the one associated with the discretization of the trajectory of the obstacle. The values $\alpha_{ij}$, $\beta_{ij}$ and $\gamma_i$ are assumed known. This assumption is needed to be able to obtain an outer polyhedral approximation of the Minkowski sum of a bounding box and any continuous trajectory of an obstacle (Sec. IV-C).
Table II: Polyhedral representations of interval \( j \) from the point of view of agent \( s \). Here, \( R_{ij}^{MV} \) denotes the set of MINVO control points of every interval of the trajectory of agent \( i \) that falls in \([t_m + j \Delta t, t_{m + (j + 1) \Delta t}]\) (timespan of the interval \( j \) of the trajectory of agent \( s \)).

| Agent \( s \) | Trajectory Inflation | Polyhedral Repr. |
|-----------------|------------------------|-----------------|
| Other agents \( i \in I \) | B-Spline | \( B_i' = \mathbb{R} \) inflated with \( \eta_i \) | \( \operatorname{conv} \{ Q_i^{MV} \} \) |
| Obstacles \( i \in I \) | Any | \( B_i' = \mathbb{R} \) inflated with \( \eta_i + 2 (\delta_i + \alpha_i) \) | \( \operatorname{conv} \{ B_i' \oplus p_i (T_i) \} \) |

- An agent can communicate without delay with other agents that are within a sphere of radius \( 4r \). All the agents have the same reference time, but trigger the planning iterations asynchronously. The reason of the value \( 4r \) is that the point \( d \) (see Table I) can be chosen at most at a distance \( r \) from the current position of the UAV (\( \bullet \)), which means that the final position of the new trajectory can be at most at \( 2r \) from \( \bullet \). As this is true for all the agents, each agent needs to be able to communicate with agents that are within a distance of \( 4r \) to be able to guarantee safety.

- Two agents do not commit to a new trajectory at the same time. Note that, as time is continuous, the probability of this assumption not being true is essentially zero. The reason behind this assumption is to guarantee that it is safe for a UAV to commit to a trajectory at \( t = t_i \) having checked all the committed trajectories of other agents at \( t < t_i \) (this will be explained in detail in Sec. VI).

IV. POLYHEDRAL REPRESENTATIONS

To avoid the computational burden of imposing infinitely-many constrains to separate two trajectories, we need to compute a tight polyhedral outer representation of every interval of the optimized trajectory (trajectory that agent \( s \) is trying to obtain), the trajectory of the other agents and the trajectory of other obstacles (see also Table II).

A. Polyhedral Representation of the trajectory of the agent \( s \)

When using B-Splines, one common way to obtain an outer polyhedral representation for each interval is to use the polyhedron defined by the control points of each interval. As the functions in the B-Spline basis are positive and form a partition of unity, this polyhedron is guaranteed to completely contain the interval. However, this approximation is far from being tight, leading therefore to great conservatism both in the position and in the velocity space. To mitigate this, [22] used the Bernstein basis for the constraints in the velocity space. Although this basis generates a polyhedron smaller than the B-Spline basis, it is still conservative, as this basis does not minimize the volume of this polyhedron. We instead use both in position and velocity space our recently derived MINVO basis [1] that, by construction, is a polynomial basis that attempts to obtain the simplex with minimum volume that encloses a given polynomial curve. As shown in Fig. 3, this basis achieves a volume that is 2.36 and 254.9 times smaller (in the position space) and 1.29 and 5.19 times smaller (in the velocity space) than the Bernstein and B-Spline bases respectively. For each interval \( j \), the vertexes of the MINVO control points \( (Q_j^{MV} \text{ and } V_j^{MV} \text{ for position and velocity respectively}) \) and the B-Spline control points \( (Q_j^{BS}, V_j^{BS}) \) are related as follows:

\[
Q_j^{MV} = Q_j^{BS} A_{pos}^{-1} \quad V_j^{MV} = V_j^{BS} A_{vel}^{-1}
\]

where the matrices \( A \) are known, and are available in our recent work [1] (for the MINVO basis) and in [35] (for the Bernstein and B-Spline bases).

B. Polyhedral Representation of the trajectory of other agents

We first increase the sides of \( B_i \) by \( \eta_i \) to obtain the inflated box \( B_i' \). Now, note that the trajectory of the agent \( i \neq s \) is also a B-Spline, but its initial and final times can be different from \( t_{iu} \) and \( t_f \) (initial and final times of the trajectory that agent \( s \) is optimizing). Therefore, to obtain the polyhedral representation of the trajectory of the agent

![Figure 3: Comparison of the volumes, areas and lengths obtained by the MINVO basis (ours), Bernstein basis (used by the Bézier curves) and B-Spline basis for an interval (---) of a given uniform B-Spline (---). In the acceleration space, the three bases generate the same control points.](image_url)
Figure 4: Example of a trajectory avoiding a dynamic obstacle. The obstacle has a box-like shape and is moving following a trefoil knot trajectory. The trajectory of the obstacle is divided into as many segments as the optimized trajectory has. An outer polyhedral representation (whose edges are shown as black lines) is computed for each of these segments, and each segment of the trajectory avoids these polyhedra.

Figure 5: Collision-free constraints between agent 2 and both the obstacle 0 and agent 1. This figure is from the view of agent 2.

Figure 6: To impose collision-free constraints, MADER uses polyhedral representations of each interval of the trajectories of other agents/obstacles. On the left a given scenario with dynamic obstacles and on the right the polyhedral representations obtained (in red).

For each interval $j$ we first increase the sides of $B_i$ by $\eta_s + 2(\beta_{ij} + \alpha_{ij})$, and denote this inflated box $B'_i$. Here $\beta_{ij}$ and $\alpha_{ij}$ are the values defined in Sec. III. We then place $B'_i$ in $p_i(T_j)$, where $p_i(T_j)$ denotes the set of positions of the obstacle $i$ at the times $T_j$ (see Table I) and compute the convex hull of all the vertexes of these boxes. Given the first assumption of Sec. III, this guarantees that the convex hull obtained is an outer approximation of all the 3D space occupied by the obstacle $i$ (inflated by the size of the agent $s$) during the interval $j$. The static obstacles are treated in the same way, with $p_i(t) = \text{constant}$. An example of these polyhedral representations is shown in Fig. 6.
V. Optimization and Initial Guess

A. Collision-free constraints

Once the polyhedral approximation of the trajectories of the other obstacles/agents have been obtained, we enforce the collision-free constraints between these polyhedra and the ones of the optimized trajectory as follows: we introduce the planes \( \pi_{ij} \) (characterized by \( n_{ij} \) and \( d_{ij} \)) that separate them as decision variables in the optimization problem and force this way the separation between the vertexes in \( C_{ij} \) and the MINVO control points \( Q_{j}^{MV} \) (see Figs. 4 and 5):

\[
\begin{align*}
    n_{ij}^T c + d_{ij} > 0 & \quad \forall c \in C_{ij}, \forall i, j \in J \\
    n_{ij}^T q + d_{ij} < 0 & \quad \forall q \in Q_{j}^{MV}, \forall j \in J
\end{align*}
\]

B. Other constraints

The initial condition (position, velocity and acceleration) is imposed by \( x(t_{in}) = x_{in} \). Note that \( p_{in}, v_{in} \) and \( a_{in} \) completely determine \( q_0, q_1 \) and \( q_2 \), so these control points are not included as decision variables.

For the final condition, we use a final stop condition imposing the constraints \( v(t_f) = 0 \) and \( a(t_f) = 0 \). These conditions require \( q_{n-2} = q_{n-1} = q_n \), so the control points \( q_{n-1} \) and \( q_n \) can also be excluded from the set of decision variables. The final position is included as a penalty cost \( \|q_{n-2} - g\|_2^2 \) in the objective function, weighted with a parameter \( \omega \geq 0 \). Here \( g \) is the goal (projection of the \( g_{term} \) onto a sphere \( S \) of radius \( r \) around \( d \), see Table I). Note that, as we are using clamped uniform B-Splines with a final stop condition, \( q_{n-2} \) coincides with the last position of the B-Spline.

To force the trajectory generated to be inside the sphere \( S \), we impose the constraint

\[
\|q - d\|_2^2 \leq r^2 \quad \forall q \in Q_{j}^{MV}, \forall j \in J
\]

Moreover, we also add the constraints on the maximum velocity and acceleration:

\[
\begin{align*}
    \text{abs}(v) & \leq v_{max} \quad \forall v \in V_{j}^{MV}, \forall j \in J \\
    \text{abs}(a) & \leq a_{max} \quad \forall a \in L \setminus \{n-1, n\}
\end{align*}
\]

where we are using the MINVO velocity control points for the velocity constraint. For the acceleration constraint, the B-Spline and MINVO control points are the same (see Fig. 3).

C. Control effort

The evaluation of a cubic clamped uniform B-Spline in an interval \( j \in J \) can be done as follows [35]:

\[
p(t) = Q_{j}^{BS} A_{pos}^{BS}(j) \begin{bmatrix} u_j^3 \\ u_j^2 \\ u_j \\ 1 \end{bmatrix} = u_j
\]

where \( u_j := \frac{t-t_i}{t_{i+1}-t_i}, \quad t \in [t_{p+j}, t_{p+j+1}] \) and \( A_{pos}^{BS}(j) \) is a known matrix that depends on each interval. Specifically, and with the knots chosen, we will have \( A_{pos}^{BS}(1) \neq A_{pos}^{BS}(2) = \ldots = A_{pos}^{BS}(m - 2p - 3) \neq A_{pos}^{BS}(m - 2p - 2) \neq A_{pos}^{BS}(m - 2p - 1) \).

Therefore, as the jerk is constant in each interval (since \( p = 3 \), the control effort is:

\[
\int_{t_i}^{t_f} \|\dot{y}(t)\|^2 dt \propto \sum_{j \in J} \left\|Q_{j}^{BS} A_{pos}^{BS}(j) \begin{bmatrix} 6 \\ 0 \\ 0 \end{bmatrix} \right\|_2^2
\]

D. Optimization Problem

Given the constraints and the objective function explained above, the optimization problem solved is as follows:

\[
\min_{Q_{j}^{BS}, p_{in}, d_{ij}} \sum_{j \in J} \left\|Q_{j}^{BS} A_{pos}^{BS}(j) \begin{bmatrix} 6 \\ 0 \\ 0 \end{bmatrix} \right\|_2^2 + \omega \left\|q_{n-2} - g\right\|_2^2
\]

s.t.

\[
\begin{align*}
    x(t_{in}) & = x_{in} \\
    v(t_i) & = v_f = 0 \\
    a(t_i) & = a_f = 0 \\
    n_{ij}^T c + d_{ij} & > 0 \quad \forall c \in C_{ij}, \forall i, j \in J \\
    n_{ij}^T q + d_{ij} & < 0 \quad \forall q \in Q_{j}^{MV}, \forall j \in J \\
    \|q - d\|_2^2 & \leq r^2 \quad \forall q \in Q_{j}^{MV}, \forall j \in J \\
    \text{abs}(v) & \leq v_{max} \quad \forall v \in V_{j}^{MV}, \forall j \in J \\
    \text{abs}(a) & \leq a_{max} \quad \forall a \in L \setminus \{n-1, n\}
\end{align*}
\]

This problem is clearly nonconvex since we are minimizing over the control points and the planes \( \pi_{ij} \) (characterized by \( n_{ij} \) and \( d_{ij} \)). We solve this problem using the augmented Lagrangian method [36], [37], and with the globally-convergent method-of-moving-asymptotes (MMA) [38] as the subsidiary optimization algorithm. The interface used for these algorithms is NLopt [39]. The time allocated per trajectory is chosen before the optimization as \( t_f - t_{in} = \frac{\|g - d\|_2}{v_{max}} \).

E. Initial Guess

To obtain an initial guess (which consists of both the control points \( \{q_0, \ldots, q_n\}^{BS} \) and the planes \( \pi_{ij} \)), we use the Octopus Search algorithm shown in Alg. 1. The Octopus Search takes inspiration from A* [40], but it is designed to work with B-Splines, handle dynamic obstacles/agents,
and use the MINVO basis for the collision check. Each control point will be a node in the search. All the open nodes are kept in a priority queue $Q$, in which the elements are ordered in increasing order of $l$, where $l$ is the sum of the distances (between successive control points) from $q_0$ to the current node, $h$ is the distance from the current node to the goal, and $\epsilon$ is the bias.

The way the algorithm works is as follows: First, we compute the control points $q_0$, $q_1$, $q_2$, which are determined from $p_m$, $v_m$ and $a_m$. After adding $q_2$ to the queue $Q$ (line 3), we run the following loop until there are no elements in $Q$: First we store in $q_1$ the first element of $Q$, and remove it from $Q$ (lines 5-6). Then, we store in a set $M$ velocity samples for $v_l$ that satisfy both $v_{max}$ and $a_{max}$. After this, we discard the current $q_1$ if any of these conditions are true (l.s. denotes linearly separable):

1) $Q_{L-3}^{MV}$ is not l.s. from $C_{l-3}$ for some $i \in I$.
2) $l = (n - 2)$ and $Q_{n-4}^{MV}$ is not l.s. from $C_{i,n-4}$ for some $i \in I$.
3) $l = (n - 2)$ and $Q_{n-3}^{MV}$ is not l.s. from $C_{i,n-3}$ for some $i \in I$.
4) $\|q_i - d\|_2 > r$.
5) $\|q_i - q_k\|_\infty \leq \epsilon'$ for some $q_k$ already added to $Q$.
6) Cardinality of $M$ is zero.

Condition 1 ensures that the convex hull of $Q_{L-3}^{MV}$ does not collide with any interval $l - 3$ of other obstacle/agent $i \in I$. The linear separability is checked by solving the following feasibility linear problem for the interval $j = l - 3$ of every obstacle/agent $i \in I$:

$$
\begin{align*}
\mathbf{n}_{ij}^T e + d_{ij} &> 0, \quad \forall e \in C_{ij} \\
\mathbf{n}_{ij}^T q + d_{ij} &< 0, \quad \forall q \in Q_{L-3}^{MV}
\end{align*}
$$

where the decision variables are the planes $\pi_{ij}$ (defined by $n_{ij}$ and $d_{ij}$). We solve this problem using GLPK [41]. Note that we also need to check the conditions 2 and 3 due to the fact that $q_{n-2} = q_{n-1} = q_a$ and hence the choice of $q_{n-2}$ in the search forces the choice of $q_{n-1}$ and $q_n$. In all these three previous conditions, the MINVO control points are used.

As in the optimization problem we are imposing the trajectory to be inside the sphere $S$, we also discard $q_i$ if condition 4 is not satisfied. Additionally, to keep the search computationally tractable, we discard $q_i$ if it is very close to another $q_k$ already added to $Q$ (condition 5): we create a voxel grid of voxel size $2\epsilon'$, and add a new control point to $Q$ only if no other point has been added before within the same voxel. Finally, we also discard $q_i$ if there are not any feasible samples for $v_l$ (condition 6).

Then, we check if we have found all the control points and if $q_{n-2}$ is sufficiently close to the goal $g$ (distance less than $\epsilon^\#$). If this is the case, the control points $q_{n-1}$ and $q_n$ (which are the same as $q_{n-2}$ due to the final stop condition) are added to the list of the corresponding control points, and are returned together with all the separating planes $\pi_{ij} \forall i \in I, \forall j \in J$ (lines 10-13). If the goal has not been reached yet, we use the velocity samples $M$ to generate $q_{i+1}$ and add them to $Q$ (lines 14-17). If the algorithm is not able to find a trajectory that reaches the goal, the one found that is closest to the goal is returned (line 18).

Fig. 7 shows an example of the trajectories found by the Octopus Search algorithm in an environment with a dynamic obstacle following a trefoil knot trajectory.

VI. DECONFLICTION

To guarantee that the agents plan trajectories asynchronously while not colliding with other agents that are also constantly replanning, we use a deconfliction scheme divided in these three periods (see Fig. 8):
Figure 8: Deconfliction between agents. Each agent includes the trajectories other agents have committed to as constraints in the optimization. After the optimization, a collision check-recheck scheme is performed to ensure feasibility with respect to trajectories other agents have committed to while the optimization was happening. In this example, agent B starts the optimization after agent A, but commits to a trajectory before agent A. Hence, when agent A finishes the optimization it needs to check whether the trajectory found collides or not with the trajectory agent B committed to at $t = t_{B,q}^A$. If it collides, agent A will simply keep executing the trajectory available at $t = t_{A,h}^1$. If it does not collide, agent A will do the Recheck step to ensure no agent has committed to any trajectory during the Check period, and if this Recheck step is satisfied, agent A will commit to the trajectory found.

- The **Optimization** period happens during $t \in (t_1, t_2]$. The optimization problem will include the polyhedral outer representations of the trajectories $p_i(t), i \in I$ in the constraints. All the trajectories other agents commit to during the Optimization period are stored.
- The **Check** happens during $t \in (t_2, t_3]$. The goal of this period is to check whether the trajectory found in the optimization collides with the trajectories other agents have committed to during the optimization. This collision check is done by performing feasibility tests solving the Linear Program 1 for every agent $i$ that has committed to a trajectory while the optimization was being performed (and whose new trajectory was not included in the constraints at $t_1$). A boolean flag is set to true if any other agent commits to a new trajectory during this Check period.
- The **Recheck** period aims at checking whether agent A has received any trajectory during the Check period, by simply checking if the boolean flag is true or false. As this is a single Boolean comparison in the code, it allows us to assume that no trajectories have been published by other agents while this recheck is done, avoiding therefore an infinite loop of rechecks.

With $h$ denoting the replanning iteration of an agent A, the

Figure 9: At $t = t_{1,A}^1$, agent A chooses the point $d$ (●) along the current trajectory that it is executing, with an offset $\delta t$ from the current position ○. Then, it allocates $\kappa \delta t$ seconds to obtain an initial guess. The closest trajectory found to $g$ is used as the initial guess if the search has not finished by that time. Then, the nonconvex optimization runs for $\mu \delta t$ seconds, choosing the best feasible solution found if no local optimum has been found by then. $\kappa$ and $\mu$ satisfy $\kappa > 0$, $\mu > 0$, $\kappa + \mu < 1$. 
time allocation for each of these three periods described above is explained in Fig. 9: to choose the initial condition of the iteration $h$, Agent A first chooses a point $d$ along the trajectory found in the iteration $h-1$, with an offset of $\delta t$ seconds from the current position $q$. Here, $\delta t$ should be an estimate of how long iteration $h$ will take. To obtain this estimate, and similar to our previous work [10], we use the time iteration $h-1$ took multiplied by a factor $\alpha \geq 1$: $\delta t = \alpha (t_{4h-1}^{A,h-1} - t_{1}^{A,h-1})$. Agent A then should finish the replanning iteration $h$ in less than $\delta t$ seconds. To do this, we allocate a maximum runtime of $\kappa \delta t$ seconds to obtain an initial guess, and a maximum runtime of $\mu \delta t$ seconds for the nonconvex optimization. Here $\kappa > 0$, $\mu > 0$ and $\kappa + \mu < 1$, to give time for the Check and Recheck. If the Octopus Search takes longer than $\kappa \delta t$, the trajectory found that is closest to the goal is used as the initial guess. Similarly, if the nonconvex optimization takes longer than $\mu \delta t$, the best feasible solution found is selected.

Fig. 8 shows an example scenario with only two agents A and B. Agent A starts its $h$-th replanning step at $t_{1}^{A,h}$, and finishes the optimization at $t_{2}^{A,h}$. Agent B starts its $q$-th replanning step at $t_{2}^{B,q}$. In the example shown, agent B starts the optimization later than agent A ($t_{2}^{B,q} > t_{2}^{A,h}$), but solves the optimization earlier than agent A ($t_{2}^{B,q} < t_{4}^{A,h}$). As no other agent has obtained a trajectory while agent B was optimizing, agent B does not have to check anything, and commits directly to the trajectory found. However, when agent A finishes the optimization at $t_{2}^{A,h} > t_{2}^{B,q}$, it needs to check if the trajectory found collides with the one agent B has committed to. If they do not collide, agent A will perform the Recheck by ensuring that no trajectory has been published while the Check was being performed.

An agent will keep executing the trajectory found in the previous iteration if any of these four scenarios happens:

1) The trajectory obtained at the end of the optimization collides with any of the trajectories received during the Optimization.
2) The agent has received any trajectory from other agents during the Check period.
3) No feasible solution has been found in the Optimization.
4) The current iteration takes longer than $\delta t$ seconds.

Under the third assumption explained in Sec. III (i.e., two agents do not commit to their trajectory at the very same time), this deconfliction scheme explained guarantees safety with respect to the other agents, which is proven as follows:

- If the planning agent commits to a new trajectory in the current replanning iteration, this new trajectory is guaranteed to be collision-free because it included all the trajectories of other agents as constraints, and it was checked for collisions with respect to the trajectories other agents have committed to during the planning agent’s optimization time.
- If the planning agent does not commit to a new trajectory in that iteration (because one of the scenarios 1–4 occur), it will keep executing the trajectory found in the previous iteration. This trajectory is still guaranteed to be collision-free because it was collision-free when it was obtained and other agents have included it as a constraint in any new plans that have been made recently. If the agent reaches the end of this trajectory (which has a final stop condition), the agent will wait there until it obtains a new feasible solution. In the meanwhile, all the other agents are including its position as a constraint for theirs trajectories, guaranteeing therefore safety between the agents.

VII. RESULTS

A. Single-Agent simulations

To highlight the benefits of the MINVO basis with respect to the Bernstein or B-Spline bases, we first run the
algorithm proposed in a corridor-like environment (73 m × 4 m × 3 m) depicted in Fig. 10, that contains 100 randomlydeployed dynamic obstacles of sizes 0.8 m × 0.8 m × 0.8 m. All the obstacles follow a trajectory whose parametric equations are those of a trefoil knot [42]. The velocity and acceleration constraints for the UAV are \( v_{\text{max}} = 5 \text{ m/s} \) and \( a_{\text{max}} = \begin{bmatrix} 20 & 20 & 9.6 \end{bmatrix}^T \text{ m/s}^2 \). The velocity profile for \( v_x \) is shown in Fig. 11. For the same given velocity constraint \( v_{\text{max}} = 5 \text{ m/s} \), the mean velocity \( v_x \) achieved by the MINVO basis is 4.15 m/s, higher than the ones achieved by the Bernstein and B-Spline basis (3.23 m/s and 2.79 m/s respectively).

We now compare the time it takes for the UAV to reach the goal using each basis in the same corridor environment but varying the total number of obstacles (from 50 obstacles to 250 obstacles). Moreover, and as a stopping condition is not a safe condition in a world with dynamic obstacles, we also report the number of times the UAV had to stop. The results are shown in Table III and Fig. 12. In terms of number of stops, the use of the MINVO basis achieves reductions of 86.4% and 88.8% with respect to the Bernstein and B-Spline bases respectively. In terms of the time to reach the goal, the MINVO basis achieves reductions of 22.3% and 33.9% compared to the Bernstein and B-Spline bases respectively. The reason behind all these improvements is the tighter outer polyhedral approximation of each interval of the trajectory achieved by the MINVO basis in the velocity and position spaces.

### B. Multi-Agent simulations without obstacles

We now compare MADER with the following different state-of-the-art algorithms:

- Sequential convex programming (SCP, [23], code).
- Relative Bernstein Polynomial approach (RBP, [27], code).
- Distributed model predictive control (DMPC, [31], code).
- Decoupled incremental sequential convex programming (dec_iSCP, [28], code).
- Search-based motion planning ([32], code), both in its sequential version (decS_Search) and in its non-sequential version (decNS_Search).

To classify these different algorithms, we use the following definitions:

- **Decentralized:** Each agent solves its own optimization problem.

Table III: Comparison of the number of stops and time to reach the goal in a corridor-like environment using different bases and with different number of obstacles.

| 50 obstacles | 100 obstacles | 150 obstacles | 200 obstacles | 250 obstacles |
|--------------|--------------|--------------|--------------|--------------|
| Basic        | Stops        | Time (s)     | Stops        | Time (s)     | Stops        | Time (s)     | Stops        | Time (s)     |
| MINVO (ours) | 1.1 ± 0.14   | 16 ± 4.6     | 17 ± 3.1     | 22 ± 0.8     | 4.2 ± 1.8    | 22.65 ± 2.10 | 9.1 ± 1.4    | 24.94 ± 3.64 |
| Bezier       | 1.4 ± 0.20   | 19.61 ± 1.08 | 22.95 ± 1.84 | 32 ± 3.04    | 11.1 ± 2.77  | 32.69 ± 4.61 | 14.10 ± 4.09 | 59.81 ± 9.70 |
| B-Spline     | 4.6 ± 2.50   | 19.61 ± 1.67 | 26.17 ± 3.04 | 20.97 ± 3.24 | 12.1 ± 2.77  | 32.69 ± 4.61 | 14.10 ± 4.09 | 59.81 ± 9.70 |

Figure 12: Time to reach the goal and number of times the UAV had to stop for different number of obstacles. 5 simulations were performed for each combination of basis (MINVO, Bernstein and B-Spline) and number of obstacles. The shaded area is the 1σ interval, where σ is the standard deviation.

- **Replanning:** The agents have the ability to plan several times as they fly (instead of planning only once before starting to fly). The algorithms with replanning are also classified according to whether they satisfy the real-time constraint in the replanning: algorithms that satisfy this constraint are able to replan in less than \( \delta t \) or at least have a trajectory they can keep executing in case no solution has found by then (see Fig. 9). Algorithms that do not satisfy this constraint allow replanning steps longer than \( \delta t \) (which is not feasible in the real world), and simulations are performed by simply having a simulation time that runs completely independent of the real time.

- **Asynchronous:** The planning is triggered independently by each agent without considering the planning status of other agents. Examples of synchronous algorithms include the ones that trigger the optimization of all the agents at the same time or that impose that one agent cannot plan until another agent has finished.

- **Discretization for inter-agent constraints:** The collision-free constraints between the agents are imposed only on a finite set of points of the trajectories. The discretization step will be denoted as \( h \) seconds.

In the test scenario, 8 agents are in a 8 × 8 m square, and they have to swap their positions. The velocity and
acceleration constraints used are \( v_{max} = 1.7 \cdot 1 \text{ m/s} \) and \( a_{max} = 0.2 \cdot 1 \text{ m/s}^2 \), with a drone radius of 15 cm. Moreover, we define the safety ratio as \( \min_{i,j} d_{min}^i/(r_i + r_j) \) [27], where \( d_{min}^i \) is the minimum distance over all the pairs of agents \( i \) and \( j \), and \( r_i, r_j \) denote their respective radii. To ensure safety we need to have a safety ratio \( < 1 \). For the RBP and DMPC algorithms, the downwash coefficient \( c \) was set to \( c = 1 \) (so that the drone is modeled as a sphere as in all the other algorithms).

The results obtained, together with the classification of each algorithm, are shown in Table IV. For the algorithms that replan as they fly, we show the following times (see Fig. 13): \( t_{start} \) (earliest time a UAV starts flying), \( t_{end} \) (last time a UAV starts flying), \( t_{start} \) (earliest time a UAV reaches the goal) and the total time \( t_{total} \) (time when all the UAVs have reached their goals). Note that the algorithm decNS_Search is synchronous (link), and it does not satisfy the real-time constraints in the replanning iterations (link). Several conclusions can be drawn from Table IV:

- Algorithms that use discretization to impose inter-agent constraints are in general not safe due to the fact that the constraints may not be satisfied between two consecutive discretization points. A smaller discretization step may solve this, but at the expense of very high computation times.
- Compared to the centralized solution that generates safe trajectories (RBP), MADER achieves a shorter overall flight distance. The total time of MADER is also shorter than the one of RBP.
- Compared to decentralized algorithms (DMPC, dec_iSCP, decS_Search and decNS_Search), MADER is the one with the shortest total time, except for the case of decNS_Search with \( u = 5 \text{ m/s}^3 \). However, for this case the flight distance achieved by MADER is 6.3 m shorter. Moreover, MADER is asynchronous and satisfies the real-time constraints in the replanning, while decNS_Search does not.
- From all the algorithms shown in Table IV, MADER is the only algorithm that is decentralized, has replanning, satisfies the real-time constraints in the replanning and is asynchronous.

C. Multi-Agent simulations with static and dynamic obstacles

We now test MADER in multi-agent environments that have also static and dynamic obstacles. For this set of experiments, we use \( \alpha_j = \beta_j = 3 \cdot 1 \text{ cm} \), \( r_j = 0.1 \text{ s} \) \( \forall j \) and a drone radius of 5 cm. We test MADER in the following two environments:

- Circle environment: the UAVs start in a circle formation and have to swap their positions while...
Figure 14: Results for the circle environment, that contains 24 static obstacles (pillars) and 25 dynamic obstacles (boxes). The case with 32 agents is shown in Fig. 1a.

Figure 15: Results for the sphere environment, that contains 18 static obstacles (pillars) and 52 dynamic obstacles (boxes and horizontal poles). The case with 32 agents is shown in Fig. 1b.
Table V: Results for MADER in the circle and sphere environments.

| Environment | Num of Agents | $t_{1\text{st\_start}}$ (s) | $t_{1\text{st\_end}}$ (s) | $t_{2\text{nd\_end}}$ (s) | Total (s) | Flight Distance per agent (m) | Safety ratio between agents | Number of stops per agent |
|-------------|---------------|-----------------|-----------------|-----------------|---------|-----------------------------|-------------------------|-------------------------|
| Circle      | 4             | 0.339           | 0.563           | 10.473          | 11.403  | 21.052                      | 5.444                   | 0.000                   |
|             | 8             | 0.904           | 0.764           | 7.344           | 7.910   | 21.025                      | 1.899                   | 0.000                   |
|             | 16            | 0.905           | 0.674           | 7.777           | 8.947   | 22.737                      | 4.824                   | 0.188                   |
|             | 32            | 0.532           | 1.195           | 9.092           | 18.820  | 22.105                      | 1.834                   | 1.500                   |
| Sphere      | 4             | 0.452           | 0.584           | 10.291          | 11.124  | 20.827                      | 4.252                   | 0.000                   |
|             | 8             | 0.425           | 0.618           | 9.204           | 12.561  | 21.684                      | 1.903                   | 0.125                   |
|             | 16            | 0.363           | 0.845           | 8.909           | 13.715  | 21.284                      | 1.905                   | 0.125                   |
|             | 32            | 0.357           | 1.725           | 9.170           | 18.275  | 22.284                      | 1.155                   | 1.000                   |

flying in a world with 24 static obstacles of size $0.4 \times 8 \times 0.4$ m and 25 dynamic obstacles of size $0.6 \times 0.6 \times 0.6$ m following a trefoil knot trajectory [42]. The radius of the circle the UAVs start from is 10 m.

- **Sphere environment**: the UAVs start in a sphere environment and have to swap their positions while flying in a world with 18 static obstacles of size $0.4 \times 8 \times 0.4$ m, 17 dynamic obstacles of size $0.4 \times 4 \times 0.4$ m (moving in z) and 35 dynamic obstacles of size $0.6 \times 0.6 \times 0.6$ m following a trefoil knot trajectory. The radius of the sphere the UAVs start from is 10 m.

The results can be seen in Table V and in Figs. 1, 14 and 15. All the safety ratios between the agents are > 1, and the flight distances achieved (per agent) are approximately 21.5 m. With respect to the number of stops, in all the cases the UAV has to stop fewer than 0.19 times, except for the most crowded case (when there are 32 agents), in which on average each UAV needs to stop 1 time and 1.5 times for the sphere and circle environments respectively.

VIII. CONCLUSIONS

This work presented MADER, a decentralized and asynchronous planner that handles static obstacles, dynamic obstacles and other agents. By using the MINVO basis, MADER obtains outer polyhedral representations of the trajectories that are 2.36 and 254.9 times smaller than the volumes achieved using the Bernstein and B-Spline bases. To ensure non-conservative, collision-free constraints with respect to other obstacles and agents, MADER includes as decision variables the planes that separate each pair of outer polyhedral representations. Safety with respect to other agents is guaranteed in a decentralized and asynchronous way by including their committed trajectories as constraints in the optimization and then executing a collision check-recheck scheme. Extensive simulations in dynamic multi-agent environments have highlighted the improvements of MADER with respect to other state-of-the-art algorithms in terms of number of stops, computation/execution time and flight distance.

IX. ACKNOWLEDGEMENTS

The authors would like to thank Pablo Tordesillas, Michael Everett, Dr. Kasra Khosoussi, Andrea Tagliabue and Parker Lusk for helpful insights and discussions. Thanks also to Andrew Torgesen, Jeremy Cai, and Stewart Jamieson for their comments on the paper. Research funded in part by Boeing Research & Technology.

REFERENCES

[1] J. Tordesillas and J. How, “MINVO basis: Finding simplexes with minimum volume enclosing polynomial curves,” arXiv preprint, 2020.
[2] X. Zhou, Z. Wang, C. Xu, and F. Gao, “Ego-planner: An esdf-free gradient-based local planner for quadrotors,” arXiv preprint arXiv:2008.08835, 2020.
[3] N. Bucki, J. Lee, and M. W. Mueller, “Rectangular pyramid partitioning using integrated depth sensors (rappids): A fast planner for multicopter navigation,” arXiv preprint arXiv:2005.03124, 2020.
[4] B. Zhou, J. Pan, F. Gao, and S. Shen, “Raptor: Robust and perception-aware trajectory replanning for quadrotor fast flight,” arXiv preprint arXiv:2007.03465, 2020.
[5] P. Florence, J. Carter, and R. Tedrake, “Integrated perception and control at high speed: Evaluating collision avoidance maneuvers without maps,” in Workshop on the Algorithmic Foundations of Robotics (WAFR), 2016.
[6] P. R. Florence, J. Carter, J. Ware, and R. Tedrake, “Nanomap: Fast, uncertainty-aware proximity queries with lazy search over local 3d data,” in 2018 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2018, pp. 7631–7638.
[7] B. T. Lopez and J. P. How, “Aggressive 3-D collision avoidance for high-speed navigation,” in Robotics and Automation (ICRA), 2017 IEEE International Conference on. IEEE, 2017, pp. 5759–5765.
[8] ——, “Aggressive collision avoidance with limited field-of-view sensing,” in Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on. IEEE, 2017, pp. 1358–1365.
[9] J. Tordesillas, B. T. Lopez, J. Carter, J. Ware, and J. P. How, “Real-time planning with multi-fidelity models for agile flights in unknown environments,” in 2019 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2019.
[10] J. Tordesillas, B. T. Lopez, and J. P. How, “FASTER: Fast and safe trajectory planner for flights in unknown environments,” in 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2019.
[11] J. Chen, T. Liu, and S. Shen, “Online generation of collision-free trajectories for quadrotor flight in unknown cluttered environments,” in Robotics and Automation (ICRA), 2016 IEEE International Conference on. IEEE, 2016, pp. 1476–1483.
[12] F. Gao, W. Wu, W. Gao, and S. Shen, “Flying on point clouds: Online trajectory generation and autonomous navigation for quadrotors in cluttered environments,” Journal of Field Robotics, vol. 36, no. 4, pp. 710–733, 2019.
[13] B. Zhou, F. Gao, L. Wang, C. Liu, and S. Shen, “Robust and efficient quadrotor trajectory generation for fast autonomous flight,” IEEE Robotics and Automation Letters, vol. 4, no. 4, pp. 3529–3536, 2019.
[14] H. Oleynikova, M. Burri, Z. Taylor, J. Nieto, R. Siegwart, and E. Galceran, “Continuous-time trajectory optimization for online uav replanning,” IEEE Robotics and Automation Letters, vol. 4, no. 4, 2020.
15. D. Mellinger, A. Kushlevy, and V. Kumar, “Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams,” in 2012 IEEE international conference on robotics and automation. IEEE, 2012, pp. 477–483.

16. N. D. Potdar, G. C. de Croon, and J. Alonso-Mora, “Online trajectory planning and control of a mav payload system in dynamic environments,” Autonomous Robots, pp. 1–25, 2020.

17. H. Zhu and J. Alonso-Mora, “Chance-constrained collision avoidance for mavs in dynamic environments,” IEEE Robotics and Automation Letters, vol. 4, no. 2, pp. 776–783, 2019.

18. M. Szmuk, C. A. Pascucci, and B. AÇıkme¸ se, “Real-time quadrotor path planning for mobile obstacle avoidance using convex optimization,” in 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2018, pp. 1–9.

19. F. Gao and S. Shen, “Quadrotor trajectory generation in dynamic environments using semi-definite relaxation on nonconvex convex optimization,” in 2017 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2017, pp. 6354–6361.

20. J. Lin, H. Zhu, and J. Alonso-Mora, “Robust vision-based obstacle avoidance for micro aerial vehicles in dynamic environments,” arXiv preprint arXiv:2002.04920, 2020.

21. 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. 4, no. 2, pp. 177–195, 2018.

22. L. Tang, H. Wang, P. Li, and Y. Wang, “Real-time trajectory generation for quadrotors using b-spline based non-uniform kinodynamic search,” in 2019 IEEE International Conference on Robotics and Biomimetics (ROBIO). IEEE, 2019, pp. 1133–1138.

23. F. Augugliaro, A. P. Schoellig, and R. D’Andrea, “Generation of collision-free trajectories for a quadcopter fleet: A sequential convex programming approach,” in 2012 IEEE/RSJ international conference on Intelligent Robots and Systems. IEEE, 2012, pp. 1917–1922.

24. A. Kulshlevy, D. Mellinger, C. Powers, and V. Kumar, “Towards a swarm of agile micro quadrotors,” Autonomous Robots, vol. 35, no. 4, pp. 287–300, 2013.

25. W. Wu, F. Gao, L. Wang, B. Zhou, and S. Shen, “Temporal scheduling and optimization for multi-mav planning,” Ph.D. dissertation, Hong Kong University of Science and Technology, 2019.

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

27. J. Park, J. Kim, I. Jang, and H. J. Kim, “Efficient multi-agent trajectory planning with feasibility guarantee using relative Bernstein polynomial,” arXiv preprint arXiv:1909.10219, 2019.

28. 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). IEEE, 2015, pp. 5954–5961.

29. A. R. Conn, N. I. Gould, and P. Toint, “A globally convergent augmented lagrangian algorithm for optimization with general constraints and simple bounds,” SIAM Journal on Numerical Analysis, vol. 28, no. 2, pp. 545–572, 1991.

30. E. G. Birgin and J. M. Martínez, “Improving ultimate convergence of an augmented lagrangian method,” Optimization Methods and Software, vol. 23, no. 2, pp. 177–195, 2008.

31. K. Svanberg, “A class of globally convergent optimization methods based on conservative convex separable approximations,” SIAM journal on optimization, vol. 12, no. 2, pp. 555–573, 2002.

32. S. G. Johnson, “The Nlopt nonlinear-optimization package,” http://www-users.cs.umn.edu/~bdodson/nlopt, 2020.

33. P. E. Hart, N. J. Nilsson, and B. Raphael, “A formal basis for the heuristic determination of minimum cost paths,” IEEE transactions on Systems Science and Cybernetics, vol. 4, no. 2, pp. 100–107, 1968.

34. “GLPK: GNU Linear Programming Kit,” https://www.gnu.org/software/glpk/, 2020.

35. J. Zhao, J. Park, and B. Schatz, “Real-time trajectory generation for micro aerial vehicles in dynamic environments,” in 2015 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2015, pp. 42–49.