[Name]: 3D Flying in Multi-Agent and Dynamic Environments

Jesus Tordesillas Torres, and Jonathan P. How

***Abstract*—This paper presents [Name], a 3D decentralized trajectory planner for UAVs able to generate collision-free trajectories in uncertain environments with static obstacles, dynamic obstacles and other agents. [Name] also uses the MINVO basis to perform collision check both in position and velocity space. By minimizing the volume of the 3-simplex that encloses a given polynomial curve, the MINVO basis obtains a volume that is 2.36 and 254.8 times smaller than the Bernstein or B-Spline bases, which are the ones extensively used in the planning state-of-the-art literature. Real-time collision check with other dynamic obstacles or agents is achieved by dividing the trajectory in a finite number of convex sets, including the planes that separates two convex sets as decision variables of the optimization problem. A deconfliction scheme between the agents, where the priority is decided in a decentralized way by the solution time of the optimization is also proposed. Uncertainty in the detection and prediction of the obstacles is also taken into account. Finally, extensive simulation in challenging cluttered environments that contain static obstacles, dynamic obstacles and dynamic agents validates the proposed algorithm.**

Initial Guess via Octopus Search ***Abstract*—**

SUPPLEMENTARY MATERIAL

The code used for this paper is available here:

https://www.github.com

A video is available here:

I. INTRODUCTION

a 3D real-time planner able to handle in static obstacles, dynamic obstacles and other agents is an open problem. The polyhedral outer approximation of a trajectory, the collision check between two trajectories, and the deconfliction scheme make this problem specially hard. , The

–> polyhedral approximation of an interval:

–> sampling, discretization does not solve the problem with

-> Deconfliction scheme (centralized, decentralized and

-> derived in previous work).

The Collision check between an agent and a dynamic obstacle (or other agent), and discretization usually needs computation

With respect to the **collision check**, a polyhedral representation of each interval via the convex hull of the control points of the Bernstein or B-Spline bases have been extensively proposed and used in the planning literature.

The authors are with the Aerospace Controls Laboratory, MIT, 77 Massachusetts Ave., Cambridge, MA, USA {jtorde, jhow}@mit.edu

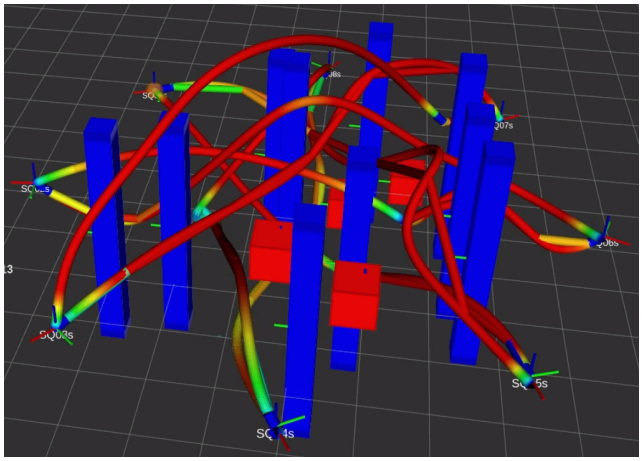

Figure 1: [Change figure] Agents using [name_algorithm] to plan trajectories in an environment with dynamic obstacles (in red), static obstacles (in blue) and other agents.

However, these bases do not succeed in obtaining the tightest (i.e. with minimum volume) tetrahedron that encloses the curve, leading therefore to conservative result. This paper addresses this problem at its source, and leverages our recently found MINVO basis [1] to obtain the control points that generate the n-simplex (tetrahedron for $n = 3$) that contains the curve and that has minimum volume.

Initial Guess

To solve the **deconfliction** problem between the trajectories of the agents, most of the state-of-the-art algorithms rely either on centralized algorithms, or on imposing a ad-hoc priority between the agents (usually via sequential optimization). We propose instead a way to naturally solve to deconflict the trajectories in a decentralized way by deciding the priority based on the time each agent is able to finish the optimization. In this way, each agent triggers the planning optimizations asynchronously with respect to other agents, including as hard constraints the trajectories other agents have already committed to.

The contributions of this paper are summarized as follows:

- Formulation
- Use of the MINVO basis to do the collision check, achieving a volume 2.36 and 254.8 times smaller than the extensively-used Bernstein and B-Spline respectively.

II. Related work

[2] use the control points of the B-Spline, not Bezier/MINVO

III. Notation and Definitions

Obstacles are inflated with radius of the UAV
This project will use the following notation:

Symbol	Meaning
$\mathbf{v}$	Vertex. $\in \mathbb{R}^3$
$\boldsymbol{p}, \boldsymbol{v}, \boldsymbol{a}, \boldsymbol{j}$	Position, Velocity, Acceleration and Jerk. $\in \mathbb{R}^3$
$\mathbf{x}$	State vector: $\mathbf{x} := \begin{bmatrix} \boldsymbol{p}^T & \boldsymbol{v}^T & \boldsymbol{a}^T \end{bmatrix}^T \in \mathbb{R}^9$
J	Set that contains the indexes of all the intervals $J = \{0, 1, ..., m-2p-1\}$
I	Set that contains the indexes all the obstacles/agents, except the agent s. $I = \{0, 1, ..., m-2p-1\} \backslash s$
i	Index of the obstacle/agent. $i \in I$
j	Index of the interval $j \in 0, 1, ..., J$
s	Index of the planning agent
$\mathcal{I}_j^i$	Polyhedron that contains the interval j of the trajectory of the obstacle/agent i.
$\{\cdot\}_{MV}$	Set of points using the MINVO (MV), Bezier (Be), or B-Spline (BS) Basis
$\mathbf{1}$	Column vector of ones

Let us also introduce the distinction between an agent and an obstacle:

- **Agent:** Element of the environment with the ability of taking decisions. I.e. an agent can change its trajectory given the information received from the environment.
- **Obstacles:** Element of the environment that does not take decisions, and therefore, it keeps executing its own trajectory, regardless of the decisions of the agents. The obstacles can be static (trajectory $= constant$) or dynamic (trajectory $= f(t)$) .

Note that, with the previous definitions, the sometimes called in the literature non-cooperative agents, are considered as dynamic obstacles.

two following common definitions and their respective notations:

IV. Assumptions

This paper relies on the following two assumptions:

- The existence of a prediction algorithm that estimates the future trajectory $\boldsymbol{p}_e(t)$ of other obstacles that are inside a sphere of radius r around the drone, with a bounded error with respect to the real trajectory $\boldsymbol{p}_{gt}(t)$:

$$\left\|\boldsymbol{p}_{gt}(t) - \boldsymbol{p}_e(t)\right\|_\infty \leq \delta(t)$$

and with the following smoothness condition ($\alpha := \frac{t_1+\Delta t-t}{\Delta t}$):

$$\max_{t\in[t_1,t_1+\Delta t]} \|\boldsymbol{p}_e(t) - (\alpha \boldsymbol{p}_e(t_1) + (1-\alpha)\boldsymbol{p}_e(t_1+\Delta t))\|_\infty \leq \beta$$

This condition bounds the deviation of the estimated trajectory (between two discretization points $\boldsymbol{p}_e(t_1)$ and $\boldsymbol{p}_e(t_1+\Delta t)$ of the trajectory) with respect to the straight line that passes through those discretization points.

- An agent can communicate without delay with other agents that are within an sphere of radius r. All the agents have the same reference time, but trigger the planning iterations asynchronously

==========

The space $S(t)$ occupied by an obstacle or agent is modeled as:

$$S(t) = \boldsymbol{p}(t) \oplus B(t)$$

where $\oplus$ is the Minkowski sum, and $B(t)$ is the 3D axis-aligned bounding box (AABB) of the obstacle. To model the uncertainty in the obstacles detection and prediction, we inflate the size of each side $B_i(t)$ ($i = \{1, 2, 3\}$) of this bounding box as follows:

$$B_i(t) = \beta_i B_{i,0} + \gamma_i(t - t_0), \quad t \in [t_0, t_f]$$

where $\beta_i \geq 1$ models the uncertainty in the current position, and $\gamma_i \geq 0$ models the uncertainty of the future trajectory. For the agents, we use $B_i(t) = B_{i,0} \quad \forall i$, and for the static obstacles $B_i(t) = \beta_i B_{i,0}$.

V. Collision Check

Imposing that a trajectory has to be ind. Many optimal control solvers are able to impose this by doing a very fine discretization, and imposing the constraints on this discretization points, but at the expense of high computation times, making them rarely applicable for real-time applications. That is why most of the path planning works that use B-Splines rely on their convex hull property, which guarantees that the interval j of the trajectory completely lies within the convex hull of the control points $\{\boldsymbol{q}_j, \boldsymbol{q}_{j+1}, \boldsymbol{q}_{j+2}, \boldsymbol{q}_{j+3}\}$. This however, can be quite conservative, as this convex hull usually does not tightly enclose the interval. [3] proposes to use a B-Spline to generate the trajectory, but then to check the collision use the control points of the Bézier curve associated with each interval. This reduces the conservatism metioned before, but sometimes these Beziér convex are also quite conservative . In this paper we propose to leverage our work [Cite MINVO's paper], where we derived the MINVO basis, a polynomial basis that attempts to obtain the simplex with minimum volume that encloses a given polynomial curve. It can be shown that this problem can be reduced to this other one:

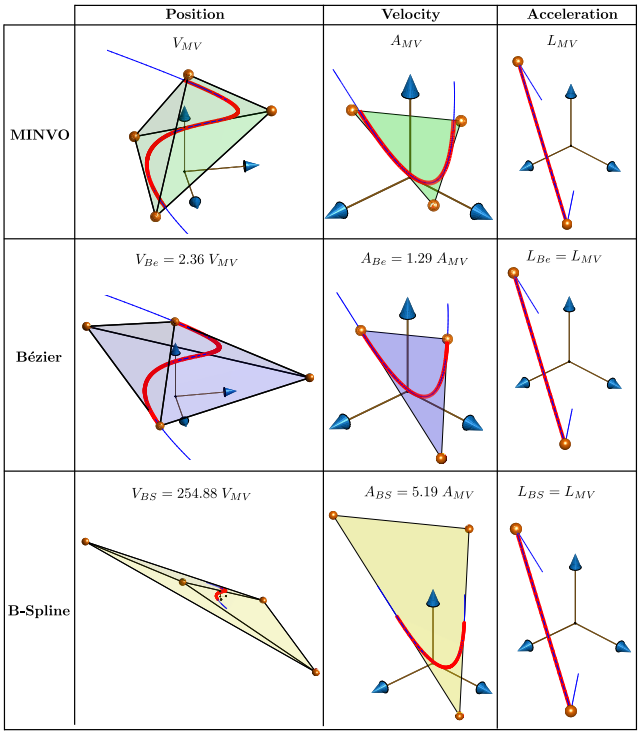

Figure 2: Comparison ... The blue line represents an uniform B-Spline, and the red line is one interval of this B-Spline. The vertexes of each tetrahedron are the control points obtained for that interval using the basis MINVO, Bernstein and B-Spline. Every arrow has length = 1 u

For 3D, this basis generates a convex hull that is 2.36 times smaller than the one Bézier generates. We have therefore that, for any given third order polynomial curve:

$$V_{MINVO} \approx \frac{V_{Bezier}}{2.36} \qquad V_{MINVO} \approx \frac{V_{B-Spline}}{254.88}$$

===

Once the convex hull has been obtained, the other question is how to guarantee that this convex hull does not intersect with any of the obstacles (i.e. convex hull is collision free). [2] proposes a conservative way to do it by using the triangular inequality. Here we instead propose to find introduce the planes as decision variable and force in this way the strong separation between the two polyhedrons ($n^T x + d \leq -\epsilon$) and $n^T x + d \geq \epsilon$ for some small $\epsilon > 0$.

1) Justify need of a simple polyhedral outer approximation for a curve (intractable to do it as opt control solvers do). There are two key aspects here:
   a) But what outer approx should we use? –>Explain collision Check comparison between the three ways (B-Spline, Bezier and MINVO (ours))+ . Explain here also the approach
   b) What method guarantees collision check. Conservative check is what paper XX did (the one that had the inequality). An exact check is the way we do it (find plane)

VI. Problem

Clamped uniform B-Splines: They are defined with $n+1$ control points $\{\boldsymbol{q}_0, ..., \boldsymbol{q}_n\}$, $m+1$ knots $\{t_0, t_1, ..., t_m\}$, where each interval of the B-Spline is a polynomial of degree p. The distibution of the knots satisfy

$$\underbrace{t_0 = ... = t_p}_{p+1 \text{ knots}} < \underbrace{t_{p+1} < ... < t_{m-p-1}}_{\text{Internal Knots}} < \underbrace{t_{m-p} = ... = t_m}_{p+1 \text{ knots}}$$

and where the internal knots are equally spaced ($t_{i+1} = t_i + \Delta t \ \ \forall i = p, ..., m-p-1$ ($\Delta t := t_{p+2} - t_{p+1}$).

The relationship $m = n + p + 1$ holds.

There are derefore $m - 2p$ intervals $\{\mathcal{I}_0, ..., \mathcal{I}_{m-2p-1}\}$, each of which is a polynomial of degree p. In this paper we will use $p = 3$ (i.e. cubic B-Splines).

Clamped guarantees that the B-Spline passes through the first and last control points.

The interval $\mathcal{I}_{i-3}$ of the curve is guaranteed to lie within the convex hull of the control points $\boldsymbol{q}_{i-3}$, $\boldsymbol{q}_{i-2}$, $\boldsymbol{q}_{i-1}$ and $\boldsymbol{q}_i$

The evaluation of a cubic clampled uniform B-Spline in the interval $\mathcal{I}_{i-3}$ can be done as follows (cite):

[This is for the interval [0,1]!!!]

$$\boldsymbol{p}_{\mathcal{I}_{i-3}}(t) = \underbrace{\begin{bmatrix} \boldsymbol{q}_{i-3} & \boldsymbol{q}_{i-2} & \boldsymbol{q}_{i-1} & \boldsymbol{q}_i \end{bmatrix}}_{:=\boldsymbol{Q}} \underbrace{\frac{1}{6}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 0 & 4 \\ -3 & 3 & 3 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}}_{:=\boldsymbol{A}_{BS}} \underbrace{\begin{bmatrix} u^3 \\ u^2 \\ u \\ 1 \end{bmatrix}}_{:=\boldsymbol{u}}$$

where $u := \frac{t - t_i}{t_{i+1} - t_i}$

First note that:

$$\frac{d^j \boldsymbol{p}(t)}{dt^j} = \frac{d^j \boldsymbol{p}(t)}{du^j} \underbrace{\frac{d^j u}{dt^j}}_{=1/\Delta t^j} = \frac{1}{\Delta t^j} \boldsymbol{Q} \boldsymbol{A}_{BS} \frac{d^j \boldsymbol{u}}{dt^j}$$

Hence, we have that the jerk (which is constant for each interval) is:

$$\boldsymbol{j}_{\mathcal{I}_{i-3}}(t) = \frac{1}{\Delta t^3} \boldsymbol{Q} \boldsymbol{A}_{BS} \begin{bmatrix} 6 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \frac{1}{\Delta t^3} \boldsymbol{Q} \begin{bmatrix} -1 \\ 3 \\ -3 \\ 1 \end{bmatrix}$$

Therefore, and assumming a third integrator model, the control effort is:

$$\int_{t_0}^{t_f} \|\boldsymbol{j}(t)\|^2 \, dt = \sum_{i=3}^{n} \left\| \boldsymbol{j}_{\mathcal{I}_{i-3}} \right\|^2 \propto \sum_{i=3}^{n} \left\| -\boldsymbol{q}_{i-3} + 3\boldsymbol{q}_{i-2} - 3\boldsymbol{q}_{i-1} + \boldsymbol{q}_i \right\|^2$$

Velocity is also a b-spline –> impose vmax and amax in their splines bbox.

Hence, the problem we are attempting to solve is:

normal of the obstacles

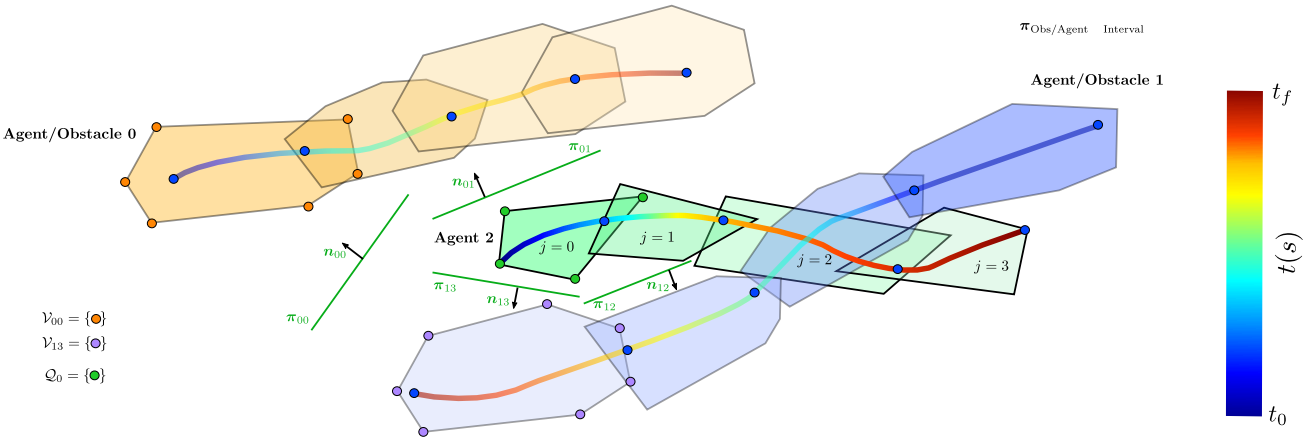

Figure 3: Caption. [NO, pues t0 y tf se refieren a la optimization del agent 3 (cuyos tiempos se usan para los intervalos de los obstaculos). t0 y tf no son los tiempos de optimization de los otros obstaculos)CHANGE THE COLORS, USE DIFFERENT tf/t0 FOR EACH AGENT]This is the view of the problem from the perspective of agent 2. Assume that the obstacles have been inflated with the radius (or max dim of its bounding box) of the agent 3. [The control points in agent 2 should match from interval to interval. Or maybe is better to plot the MINVO control points directly]

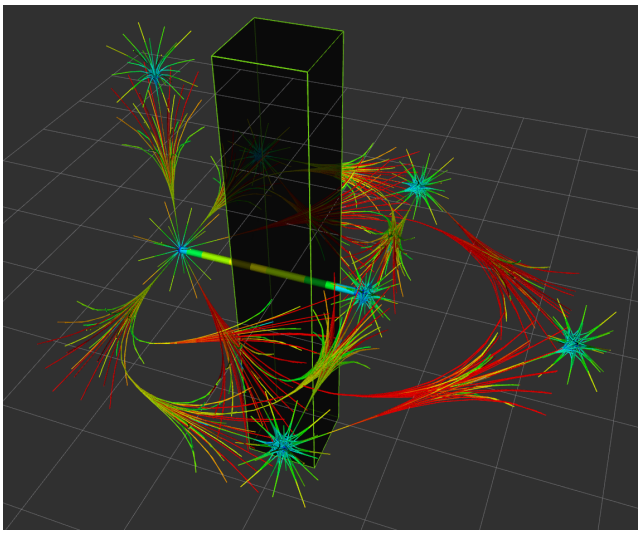

Figure 4: CHANGE THIS FIGURE//PUT AN EXAMPLE WITH DYNAMIC OBSTACLES?? Example of the Octopus Search that generates the initial guess for the nonconvex optimization. An static obstacle is shown here for visualization purposes, but Alg. 1 also works for dynamic obstacles

obstacle; i ////I :=Num_of_obstacles+agents (\neq me) in local map

index interval: j///$J := m - 2p - 1$

Let us define

index obstacle/agent: i (say that we treat them indistintively)

Time allocation is given

$\|\boldsymbol{v}(t)\|_\infty \leq v_{max}$

First/last qi are given by pos and velocity

A final stop condition is used in the optimization.

$$
\begin{aligned}
\min_{\mathcal{Q}_j, \boldsymbol{n}_{ij}, d_{ij}} \quad & \sum_{j=0}^{n-3} \left\| -\boldsymbol{q}_j + 3\boldsymbol{q}_{j+1} - 3\boldsymbol{q}_{j+2} + \boldsymbol{q}_{j+3} \right\|^2 + \omega \left\| \boldsymbol{q}_N - \boldsymbol{g} \right\|^2 \\
\text{s.t.} \quad & \\
& \mathbf{x}(t_0) = \mathbf{x}_{init} \\
& \boldsymbol{v}(t_f) = \mathbf{0} \\
& \boldsymbol{a}(t_f) = \mathbf{0} \\
& \boldsymbol{n}_{ij}^T \boldsymbol{v} + \boldsymbol{d}_{ij} > 0 \quad \forall \boldsymbol{v} \in \mathcal{V}_{ij},\ \forall i \in I, j \in J \\
& \boldsymbol{n}_{ij}^T \boldsymbol{q} + \boldsymbol{d}_{ij} < 0 \quad \forall \boldsymbol{q} \in \mathcal{Q}_j,\ \forall j \in J \\
& \left\| \frac{p(\mathbf{q}_{j+1} - \mathbf{q}_j)}{t_{j+p+1} - t_{j+1}} \right\|_\infty \leq v_{max} \quad \forall j \in J \\
& \left\| \frac{(p-1)}{t_{i+p+1} - t_{i+2}} \left(\frac{p(\mathbf{q}_{j+2} - \mathbf{q}_{j+1})}{t_{j+p+2} - t_{j+2}} - \frac{p(\mathbf{q}_{j+1} - \mathbf{q}_j)}{t_{j+p+1} - t_{j+1}} \right) \right\|_\infty \leq a_{max} \quad \forall j \in J
\end{aligned}
$$

[CUIDADO QUE EN ESTE OPT PROBLEM DEBERIA USAR MINVO CONTROL POINTS]

where $\boldsymbol{g}$ is the intermediate goal, and $\omega \geq 0$ is the weight of the soft terminal cost. Note that this problem is clearly a nonconvex problem, since we are minimizing over the control points *and* the planes $\boldsymbol{\pi}_{ij}$ (charaterized by $\boldsymbol{n}_{ij}$ and d_{ij}). We solve this problem using the augmented Lagrangian method [4], [5], and with the globally-convergent method-of-moving-asymptotes (MMA) [6] as the subsidiary optimization algorithm. Both of these algorithms are interfaced using NLOPT [7].

A. Initial Guess

To obtain an initial guess (which consists of both the control points $\{\boldsymbol{q}_0, ..., \boldsymbol{q}_n\}$ and the planes $\boldsymbol{\pi}_{ij}$), we propose to use the *Octopus Search* algorithm shown in Alg. 1. The *Octopus Search* is a A^*-like algorithm. This guess handles dyn obstacles. We keep all the open nodes in a priority queue Q, in which the elements are ordered in increasing order of $f = g + \epsilon h$. We use g as the sum of the distances between control points to the current node. For the heuristics h we use the distance from the current node to the goal.

As $\boldsymbol{p}(t_0)$, $\boldsymbol{v}(t_0)$, and $\boldsymbol{a}(t_0)$ are given, the position control points $\{\boldsymbol{q}_0, \boldsymbol{q}_1, \boldsymbol{q}_2\}$, the velocity control points $\{\boldsymbol{v}_0, \boldsymbol{v}_1\}$

and the acceleration control point $\boldsymbol{a}_0$ are completely determined. Both $\boldsymbol{v}_0$ and $\boldsymbol{a}_0$ are guaranteed to satisfy the bounds v_{max} and a_{max} because they are obtained from the trajectory in the previous iteration (i.e. $\boldsymbol{v}_0 = \boldsymbol{v}(t_0)$ and $\boldsymbol{a}_0 = \boldsymbol{a}(t_0)$), which was feasible. To ensure that $\|\boldsymbol{v}_1\|_\infty \leq v_{max}$, we can adjust Δt so that it satisfies:

$$\begin{cases} \frac{(p-1)\left(-sign(\boldsymbol{a}_0^i)v_{max}\mathbf{1}-\mathbf{v}_0^i\right)}{\boldsymbol{a}_0^i} \leq \Delta t \leq \frac{(p-1)\left(sign(\boldsymbol{a}_0^i)v_{max}\mathbf{1}-\mathbf{v}_0^i\right)}{\boldsymbol{a}_0^i} \\ 0 \leq \Delta t \end{cases} \quad \forall i \in \{1,2,3\}$$

(for the cases $\boldsymbol{a}_0^i = 0$, the condition $\left|\boldsymbol{v}_1^i\right| \leq v_{max}$ is satisfied, because $\boldsymbol{v}_1^i = \boldsymbol{v}_0^i$.

(see line 3)

Moreover, as $\boldsymbol{v}(t_0) = \mathbf{0}$, and $\boldsymbol{a}(t_0) = \mathbf{0}$, we have that $\boldsymbol{q}_{N-2} = \boldsymbol{q}_{N-1} = \boldsymbol{q}_N$.

The way a new node is expanded works as follows: First we sample uniformly in v_x,v_y and v_z. Note that:

$$\boldsymbol{v}_j = \frac{p\left(\boldsymbol{q}_{j+1} - \boldsymbol{q}_j\right)}{t_{j+p+1} - t_{j+1}} \qquad \boldsymbol{a}_{j-1} = \frac{(p-1)\left(\boldsymbol{v}_j - \boldsymbol{v}_{j-1}\right)}{t_{j+p} - t_{j+1}}$$

And therefore we can ensure the bounds v_{max} and a_{max} if we take a sample $\boldsymbol{v}_j$ in this interval:

$$\max\left(-\boldsymbol{v}_{max}, -\boldsymbol{a}_{max}d + \boldsymbol{v}_{j-1}\right) \leq \boldsymbol{v}_j \leq \min\left(\boldsymbol{v}_{max}, \boldsymbol{a}_{max}d + \boldsymbol{v}_{j-1}\right)$$

where the operators max, min and $\leq$ are element wise, $d := \frac{t_{j+p}-t_{j+1}}{(p-1)}$, $\boldsymbol{v}_{max} := v_{max}\mathbf{1}$, and $\boldsymbol{a}_{max} := a_{max}\mathbf{1}$. The next control point for a sample is then given by $\boldsymbol{q}_{j+1} \leftarrow \boldsymbol{q}_j + \frac{t_{j+p+1}-t_{j+1}}{p}\boldsymbol{v}_j$. EXPLAIN WHAT I'M DOING TO ENFORCE that aNm2 is inside bounds (see computeUpperAndLowerConstraints in code). Now the only reason why it may not be feasible is when filling (either because it collides, or because it doesn't satisfy the accel constraints)

Once $\boldsymbol{q}_{j+1}$ have been expanded, we convert the control points $\{\boldsymbol{q}_{j-2}, \boldsymbol{q}_{j-1}, \boldsymbol{q}_j, \boldsymbol{q}_{j+1}\}_{BS}$ to the MINVO basis, $\{\cdot\}_{MV}$ and discard the current $\boldsymbol{q}_{j+1}$ if any of these conditions are true:

- $\{\boldsymbol{q}_{j-2}, \boldsymbol{q}_{j-1}, \boldsymbol{q}_j, \boldsymbol{q}_{j+1}\}_{MV}$ is not l.s. from $\mathcal{I}_{j-2}^i$ for some i
- $j = (N-3)$ and $\{\boldsymbol{q}_{j-1}, \boldsymbol{q}_j, \boldsymbol{q}_{N-2}, \boldsymbol{q}_{N-2}\}_{MV}$ is not l.s. from $\mathcal{I}_{j-1}^i$ for some i
- $j = (N-3)$ and $\{\boldsymbol{q}_j, \boldsymbol{q}_{N-2}, \boldsymbol{q}_{N-2}, \boldsymbol{q}_{N-2}\}_{MV}$ is not l.s. from $\mathcal{I}_j^i$ for some i
- $\left\|\boldsymbol{q}_{j+1} - \boldsymbol{q}_k\right\|_\infty \leq \epsilon$ for some $\boldsymbol{q}_k$ already added to Q

Note that we need to check the second and third conditions due to the fact that $\boldsymbol{q}_{N-2} = \boldsymbol{q}_{N-1} = \boldsymbol{q}_N$. The linear separability is checked by solving the following feasibility linear problem for each interval j of the obstacle i:

$$\begin{aligned} \boldsymbol{n}_{ij}^T\boldsymbol{v} + \boldsymbol{d}_{ij} > 0 \qquad & \forall \boldsymbol{v} \in \mathcal{V}_{ij},\ \forall i \in I, j \in J \\ \boldsymbol{n}_{ij}^T\boldsymbol{q} + \boldsymbol{d}_{ij} < 0 \quad & \forall \boldsymbol{q} \in \{\boldsymbol{q}_{j+1}, \boldsymbol{q}_j, \boldsymbol{q}_{j-1}, \boldsymbol{q}_{j-2}\},\ \forall j \in J \end{aligned}$$

Algorithm 1: Octopus Search

```
1  Function GetInitialGuess():
2      Compute q_0, q_1 and q_2 from p_0, v_0 and a_0
3      Adjust Δt so that ||v_1||_∞ ≤ v_max is satisfied
4      Add q_2 to Q
5      while Q is not empty do
6          q_i ←First element of Q
7          if dist(q_i, goal) < r and i = (n − 2) then
8              q_{n−1} ← q_{n−2}
9              q_n ← q_{n−2}
10             return {q_0, q_1, q_2, ..., q_{n−2}, q_{n−1}, q_n} ∪ π_ij
11         Remove first element of Q
12         V ←Uniformly sample v_j satisfying v_max and a_max
13         for every v_i in V do
14             q_{j+1} ← q_j + ((t_{j+p+1} − t_{j+1})/p) v_j
15             if any of the conditions [Reference] is true then
16                 continue with next v_i
17             else
18                 Store in q_{j+1} a pointer to q_j and the planes
                     π_ij
19                 Add q_{j+1} to Q
20     return Straight Line guess
```

where the decision variables are the planes $\boldsymbol{\pi}_{ij}$ (defined by$\boldsymbol{n}_{ij}$ and d_{ij}). We solve this problem using GLPK [8].

The last condition creates like a voxel grid of voxel size 2ϵ, allowing only to add the point $\boldsymbol{q}_{j+1}$ to Q if no other point has been added before within the same voxel.

If all the previous conditions are false, $\boldsymbol{q}_{j+1}$ is not discarded, and it is appended to Q (lines 18 and 19).

Guaranteed to be a nearly-feasible guess. ??

[Change r in the algorithm above]

VII. Deconfliction

The deconfliction scheme is divided in three periods **Optimization**, **check** and **recheck**:

- The **Optimization** period happens $t \in (t_1, t_2]$, and include all the trajectories of the obstacles in the local map at $t = t_1$ as hard constraints .
- 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 recived during the optimization.
- The **Recheck** period aims at checking whether the agent A has received any trajectory during the Check period. It is implemented as a single Boolean comparison in the code, which allows us to assume that no trajectories have been published by other agents while this recheck is done (and hence an infinite loop of rechecks is not needed).

Plot with the order of magnitude of timings of all the checks

Should I include also the trajectories of the obstacles??

Fig. 5 shows one example scenario.

To choose the initial condition of the iteration k, Agent A first chooses a point along the trajectory found in the

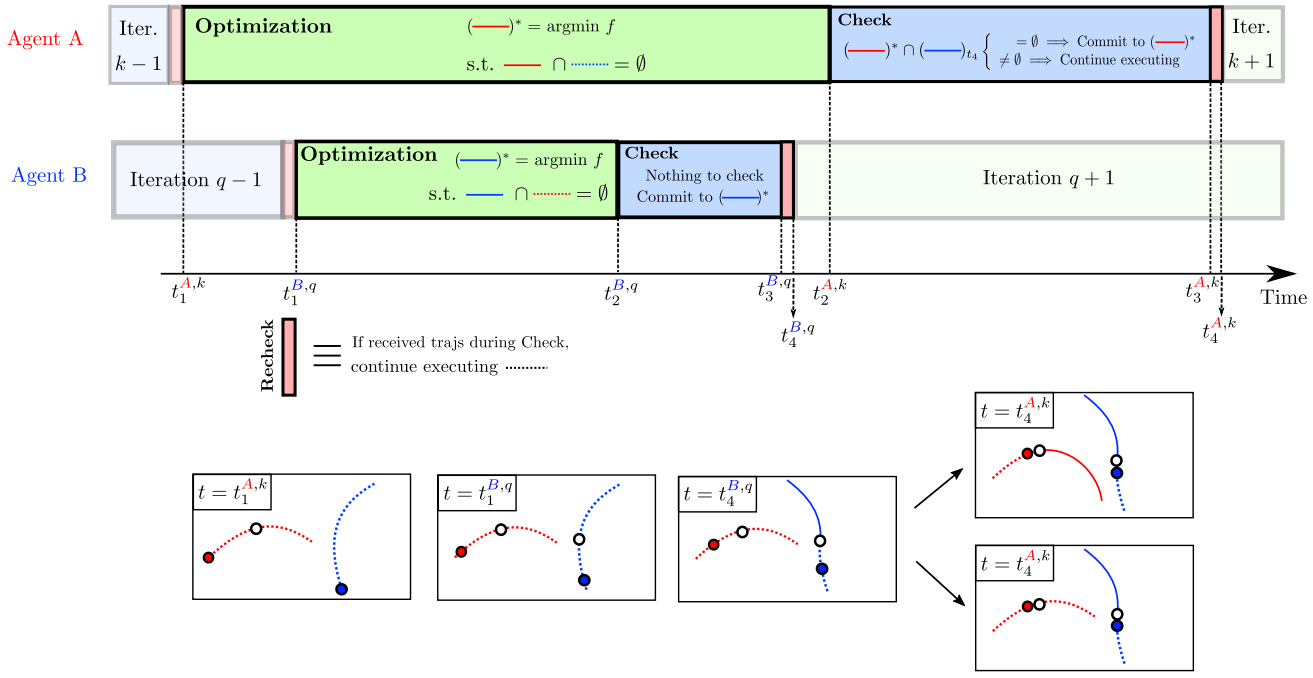

Figure 5: Note that as soon as one iteration finishes, the next one starts (i.e. $t_4^{A,k} = t_1^{A,k+1}$ and $t_4^{B,q} = t_1^{B,q+1}$).[Fade "iteration" words]

iteration $k-1$, with an offset of δt seconds from the current position ●. Similar to our previous work [9], to obtain an estimate of the total time the iteration k may take, we use the time the previous iteration took multiplied by a factor $\alpha > 1$: $\delta t = \alpha \left(t_4^{A,k-1} - t_1^{A,k-1} \right)$ (see Fig. 6). The agent A then should finish the replanning iteration k in less than δt 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 $\kappa + \mu < 1$(usually $\kappa + \mu \approx 0.8$) to give time to the check and recheck. If the octopus search takes longer than $\kappa \delta t$, the closest path to the goal found is chosen as initial guess. Similarly, if the nonconvex optimization takes longer than $\mu \delta t$, the best feasible solution is selected.

If the replanning iteration k takes longer than δt, or becomes infeasible, then the agent will keep executing the trajectory found in the iteration $k-1$ if any of these four scenarios happen:

- The trajectory obtained at the end of the optimization collides with any the trajectories received during the Optimization.
- The agent has received any trajectory from other agents during the Check period.
- No feasible solution has been found in the Optimization
- Iteration k takes longer than δt seconds.

The previous deconfliction scheme guarantees safety wrt the other agents assuming $t_4^{A,k} \neq t_4^{B,q}$

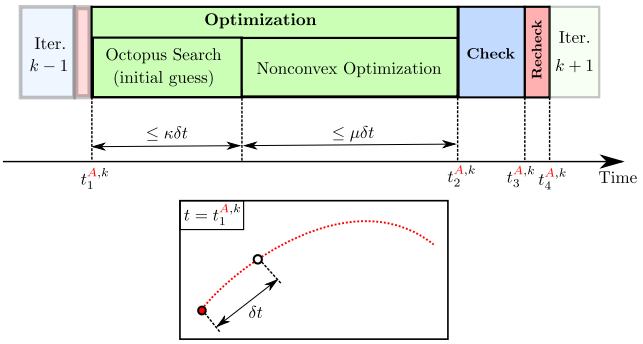

Figure 6: At $t = t_1^{A,k}$, agent A chooses a point ○ along the current trajectory that it is executing, with an offset δt from the current position ●. Then, it allocates $\kappa \delta t$ seconds to obtain an initial guess. The closest trajectory found to $\boldsymbol{g}$ is used as the initial guess if the search has not finished after this 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. κ and μ satisfy $\kappa > 0$, $\mu > 0$, $\kappa + \mu < 1$ [Fade "iteration" words]

VIII. Results

A. Single-Agent

We first run the algorithm proposed in a corridor-like enviroment (50 m × 4 m × 3 m) depicted in Fig. that contains both static and dynamic obstacles. All the obstacles follow a trajectory whose parametric equations are those of a trefoil knot [Cite], with random offsets, centers andd scales. We test the algorithm running

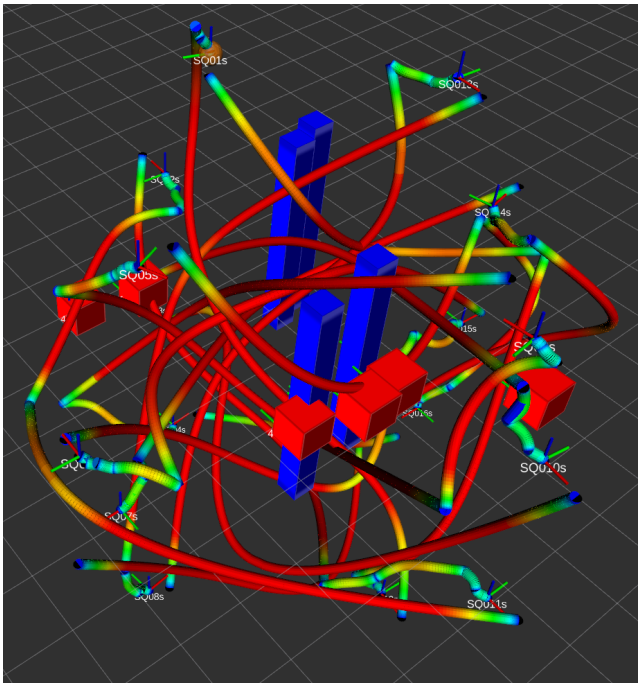

Figure 7: [Change figure] Agents using [name_algorithm] to plan trajectories in an environment with dynamic obstacles (in red), static obstacles (in blue) and other agents.

B. Multi-Agent

IX. Conclusions and Future Work

This work derived . The key features of

The exciting results of this project naturally lead to the following questions:

- Is

X. Acknowledgements

The authors would like to thank Michael Everett, Kasra Khosoussi, Andrea Tagliabue and Parker Lusk for helpul insights and discussions.

References

[1] Jesus Tordesillas and Jonathan How. MINVO basis: Finding simplexes with minimum volume enclosing polynomial curves. 2019.

[2] Boyu Zhou, Fei Gao, Luqi Wang, Chuhao Liu, and Shaojie Shen. Robust and efficient quadrotor trajectory generation for fast autonomous flight. *IEEE Robotics and Automation Letters*, 4(4):3529–3536, 2019.

[3] Lvbang Tang, Hesheng Wang, Peng Li, and Yong 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)*, pages 1133–1138. IEEE, 2019.

[4] Andrew R Conn, Nicholas IM Gould, and Philippe Toint. A globally convergent augmented lagrangian algorithm for optimization with general constraints and simple bounds. *SIAM Journal on Numerical Analysis*, 28(2):545–572, 1991.

[5] Ernesto G Birgin and José Mario Martínez. Improving ultimate convergence of an augmented lagrangian method. *Optimization Methods and Software*, 23(2):177–195, 2008.

[6] Krister Svanberg. A class of globally convergent optimization methods based on conservative convex separable approximations. *SIAM journal on optimization*, 12(2):555–573, 2002.

[7] Steven G. Johnson. The Nlopt nonlinear-optimization package. http://github.com/stevengj/nlopt, 2020.

[8] GLPK: GNU Linear Programming Kit. https://www.gnu.org/software/glpk/, 2020.

[9] Jesus Tordesillas, Brett T Lopez, and Jonathan 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.