Online Trajectory Generation With Distributed Model Predictive Control for Multi-Robot Motion Planning

Carlos E. Luis, Marijan Vukosavljev, and Angela P. Schoellig

***Abstract*—We present a distributed model predictive control (DMPC) algorithm to generate trajectories in real-time for multiple robots. We adopted the *on-demand collision avoidance* method presented in previous work to efficiently compute non-colliding trajectories in transition tasks. An event-triggered replanning strategy is proposed to account for disturbances. Our simulation results show that the proposed collision avoidance method can reduce, on average, around 50% of the travel time required to complete a multi-agent point-to-point transition when compared to the well-studied Buffered Voronoi Cells (BVC) approach. Additionally, it shows a higher success rate in transition tasks with a high density of agents, with more than 90% success rate with 30 palm-sized quadrotor agents in a 18 m^3 arena. The approach was experimentally validated with a swarm of up to 20 drones flying in close proximity.**

***Index Terms*—Motion and path planning, distributed robot systems, collision avoidance, model predictive control.**

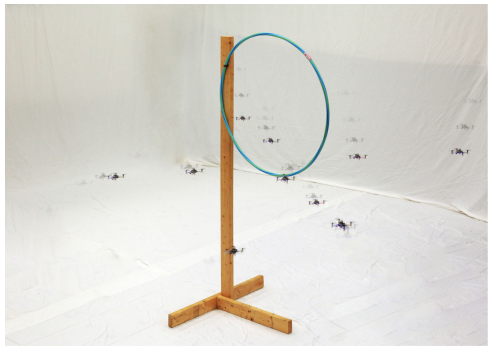


Fig. 1. A ten-drone transition task through a hula-hoop solved using our proposed online trajectory generation method. Our distributed computation allows for real-time multi-robot motion planning, enabling complex transition tasks to be performed. A video of the performance is found at `http://tiny.cc/online-dmpc`.

I. Introduction

ONLINE trajectory generation is key to execute missions in dynamic or unknown environments. In particular, multi-robot tasks are especially challenging due to a high number of decision-making agents sharing the same space. In such settings, the planning algorithms must compute collision-free and goal-oriented trajectories, taking into account the state of the environment and neighbouring agents.

A wide variety of techniques exist to tackle the multi-robot trajectory generation problem. First, optimization-based techniques such as Sequential Convex Programming (SCP) [1], [2] and Distributed Model Predictive Control (DMPC) [3], [4] have successfully solved point-to-point trajectory generation problems for multiple agents. Second, discrete planning strategies such as Rapidly-exploring Random Trees (RRT) [5] have been extended to the multi-agent case. Third, a combination of discrete planning and continuous optimization has been developed to coordinate multiple robots in cluttered environments [6]. GPU-accelerated approaches can reduce the runtime of these offline planners [7].

Real-time trajectory generation is required for quick adaptation in dynamic environments, but it remains challenging to implement for robot swarms. Optimal Reciprocal Collision Avoidance (ORCA) and all its variants have pushed towards real-time trajectory generation [8], providing experimental validation with various robotic platforms in planar environments [9]. A similar approach achieves collision avoidance through the concept of Buffered Voronoi Cells (BVC) [10], showing initial results of online trajectory generation in 2D with multiple quadrotors operating at a fixed height. The BVC concept has been recently used in tandem with discrete planners [11], primarily to avoid deadlocks in scenarios where plain BVC would get trapped and fail the task.

Robust MPC frameworks such as tube MPC have been developed for distributed multi-agent systems under uncertainty, both with linear [12] and nonlinear [13] dynamics. Although both approaches provide proofs and simulation results, they are not real-time implementable with current hardware and solver capabilities.

We present a novel real-time, multi-vehicle motion planning framework that significantly outperforms existing methods in terms of the success rate to complete transition tasks in agent-dense environments, as shown in Fig. 1. To the best of our knowledge, this letter presents the first results on real-time motion planning for drone swarms of up to 20 drones, executed

Manuscript received September 10, 2019; accepted December 18, 2019. Date of publication January 6, 2020; date of current version January 20, 2020. This letter was recommended for publication by Associate Editor Prof. Javier Alonso-Mora and Editor Nak Young Chong upon evaluation of the reviewers' comments. This work was supported in part by the NSERC research and equipment Grants RTI 2018-00847, CRDPJ 528161-18, CREATE 466088, and in part by the CFI JELF/ORF Grant #33000. *(Corresponding author: Carlos E. Luis.)*

The authors are with the Dynamic Systems Lab, Institute for Aerospace Studies, University of Toronto, Toronto, ON M5S, Canada (e-mail: carlos.luis@robotics.utias.utoronto.ca; mario.vukosavljev@mail.utoronto.ca; schoellig@utias.utoronto.ca).

This letter has supplementary downloadable material available at http://ieeexplore.ieee.org, provided by the authors.

Digital Object Identifier 10.1109/LRA.2020.2964159

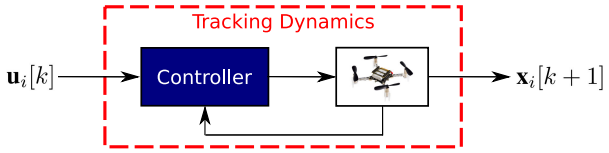


Fig. 2. Block diagram of the control system of agent i. Here we depict the agent as a Crazyflie 2.0 quadrotor, which is our experimental platform.

from a single off-board computer. The proposed algorithm is implemented in a centralized fashion, and relies on information sharing between agents. We developed a DMPC formulation of the problem that includes state feedback, which enables online replanning and therefore increases overall robustness. As such, our framework provides an essential functionality for higher-level planners that specify complex team missions in terms of goal locations to be visited by the agents. Compared to our offline approach [4], our main contributions are threefold: (*i*) a multi-agent motion planning framework based on distributed model predictive control, which allows for real-time trajectory generation, (*ii*) an event-triggered replanning strategy for robust execution of plans and (*iii*) a thorough empirical evaluation of the method.

Our approach contrasts from current online methods (e.g., [11]) in that:

- It is purely optimization-based, in the form of a standard and efficiently-solvable Quadratic Program (QP).
- It uses *on-demand collision avoidance* instead of the BVC method for partitioning the free space, resulting in less conservative movement and faster transition times.

The rest of the letter is organized as follows: Section II introduces the problem. Section III formalizes the DMPC method and Section IV introduces the trajectory replanning strategy. The algorithm for input updates is presented in Section V. Finally, Section VI and VII provide simulation and experimental results of our approach with teams of drones.

II. Problem Statement

Given N agents with known linear dynamics, a finite 3-dimensional workspace $\mathcal{W} \subset \mathbb{R}^3$, desired end positions $\mathbf{p}_{d,i} \in \mathcal{W}$ for each agent i and static obstacle set $\mathcal{E} \subset \mathcal{W}$, compute inputs $\mathbf{u}_i[k] \in \mathbb{R}^3$ for each agent such that:

- the agents do not collide with each other or with the obstacles;
- the agents remain within $\mathcal{W}$ for all time;
- there exists a time T_f after which the agents remain sufficiently close to their desired positions.

A. The Agents

We assume every agent i is equipped with a controller for position trajectory tracking and $\mathbf{u}_i$ is a position reference, as shown in Fig. 2.

Furthermore, assume each agent i obeys some known trajectory tracking dynamics given by a discrete linear system:

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

For example, in this letter we consider the system (1) to represent a quadrotor with an underlying position controller [14], for which the input ($\mathbf{u}_i[k] \in \mathbb{R}^3$) is a position reference signal, and the states ($\mathbf{x}_i[k] \in \mathbb{R}^6$) are the position and velocity of the vehicle, i.e., $\mathbf{x}_i[k] = (\mathbf{p}_i[k], \mathbf{v}_i[k])$. This results in a second-order system defining the dynamics, with $\mathbf{A}_i \in \mathbb{R}^{6\times6}$, $\mathbf{B}_i \in \mathbb{R}^{6\times3}$. One may adopt more complex inputs (e.g., adding a velocity reference), or more complex systems (e.g., other differentially flat robots) as long as the dynamics can be represented by a linear system.

III. Online Distributed Model Predictive Control (DMPC)

In this section we formalize the MPC optimization problem solved in real-time for each agent. The approach is based on the offline method presented in [4].

A. Trajectory Parameterization

Our approach is based on receding horizon control, meaning that at the discrete time step k_t, corresponding to the continuous time instant t_0, we recompute the input sequence to be applied over a finite horizon of K time steps. Given a desired discrete time step duration h, we get the continuous time horizon duration $t_h = (K-1)h$. We parameterize the continuous input signal $\mathbf{u}_i(t)$ for $t \in [t_0,\, t_0 + t_h]$ as a concatenation of l Bézier curves, similar to [6]. For more details on Bézier curves we refer the reader to [15].

We select Bézier curves since we can impose smoothness requirements in the input and can easily represent its derivatives. In order to define a Bézier curve in $\mathbb{R}^3$ of arbitrary degree p and duration T, first we must construct the $p+1$ Bernstein polynomials of degree p:

$$S_{m,p}(t) = \binom{p}{m}(1-t/T)^{p-m}(t/T)^m \quad \forall t \in (0,\, T), \tag{2}$$

with $m = 0, 1, \ldots, p$. Now, an n-dimensional Bézier curve of degree p is defined as $\mathbf{S}(t) = \sum_{i=0}^{p} \mathbf{P}_m S_{m,p}(t)$ with $\mathbf{P}_m \in \mathbb{R}^3$. The set $\mathcal{P} = \{\mathbf{P}_0, \mathbf{P}_1, \ldots, \mathbf{P}_p\}$ represents the $p+1$ control points that uniquely characterize the curve. The control points are a finite parameterization of the continuous curve and serve as the optimization variables to compute the agents' trajectories over the horizon.

Samples of $\mathbf{S}(t)$ and its derivatives can be computed as a linear combination of its control points [15], which will be used in what follows to build a convex optimization problem.

B. The Agent Prediction Model

We introduce the notation $\hat{(\cdot)}[k|k_t]$, which represents the predicted value of $(\cdot)[k_t + k]$ with the information available at k_t and $k \in \{0, \ldots, K-1\}$, where K is the horizon length. The prediction model of agent i is given by

$$\hat{\mathbf{x}}_i[k+1|k_t] = \mathbf{A}_i\hat{\mathbf{x}}_i[k|k_t] + \mathbf{B}_i\hat{\mathbf{u}}_i[k|k_t]. \tag{3}$$

Using (3) we can represent the (stacked) predicted state sequence over the horizon, $\mathbf{X}_i \in \mathbb{R}^{6K}$, as

$$\mathbf{X}_i = \mathbf{A}_{0,i}\bar{\mathbf{x}}_i[k_t] + \mathbf{\Lambda}_i\mathbf{U}_i, \tag{4}$$

where $\mathbf{U}_i \in \mathbb{R}^{3K}$ is the stacked input sequence, $\bar{\mathbf{x}}_i[k_t]$ is the measured state at k_t, and matrices $\mathbf{A}_{0,i} \in \mathbb{R}^{6K\times 6}$ and $\mathbf{\Lambda}_i \in \mathbb{R}^{3K\times 3K}$. We note that $\mathbf{U}_i$ is a *sampled* representation of the input, and it can be obtained from a linear combination of the control points of a *continuous* Bézier curve. We define $\boldsymbol{\mathcal{U}}_i \in \mathbb{R}^{3l(p+1)}$ as the decision vector to optimize, which represents the control points of the l Bézier curves of degree p.

C. Input Continuity

Trajectory smoothness is enforced through equality constraints. First, the initial control point of the input is chosen to be equal to a constant vector; the way this constant vector is constructed is the subject of Section IV. Second, continuity between the l Bézier curves is guaranteed up to a certain derivative by forcing the endpoint of a curve to match the beginning of the next curve, i.e., the difference between control points must be equal to zero [11].

Using linear relationships between the control points of the Bézier curve and the control points of its derivatives, we build a tuple $(\mathbf{A}_{\text{eq}}, \mathbf{b}_{\text{eq}})$ that represents the input continuity constraints of the form $\mathbf{A}_{\text{eq}}\boldsymbol{\mathcal{U}}_i = \mathbf{b}_{\text{eq}}$ for each agent i.

D. Physical Limits of the Robot

Since the agents have limited actuation and the workspace is limited as defined by $\mathcal{W}$, we must encode such limitations within the optimization. For dynamic feasibility we impose the following constraints

$$\boldsymbol{\gamma}_{\min}^{(c)} \leq \frac{d^c}{dt^c}\mathbf{u}_i(t) \leq \boldsymbol{\gamma}_{\max}^{(c)}, \quad c = \{0, 1, \ldots, r\}, \tag{5}$$

where $\boldsymbol{\gamma}_{\min}^{(c)}$ and $\boldsymbol{\gamma}_{\max}^{(c)}$ are the given maximum and minimum values of the c^{th} derivative of the input.

In general, imposing these constraints has posed difficulties in past work. One option proposed in the literature is to exploit the convex hull property of Bézier curves, although this may impose overly conservative bounds [16]. A second option, as suggested in [11], is to not impose the constraints at all and check afterwards if the trajectories comply with the constraints; if not, the problem needs to be resolved for constraint satisfaction. In this work we propose a third alternative, in which we obtain specific samples of the input and its derivatives (as a linear combination of the control points) and limit those appropriately through linear inequality constraints of the form $\mathbf{A}_{\text{ineq}}\boldsymbol{\mathcal{U}}_i = \mathbf{b}_{\text{ineq}}$. The procedure involves computing a linear transformation between control points and polynomial coefficients in the power basis [15], which then can be multiplied by vectors of the form $\{1, t_0, \ldots, t_0^p\}$ to obtain the exact value of $\mathbf{S}(t_0)$ and its derivatives.

This method avoids the conservativeness of using the convex hull property and the potential need to resolve the problem as in [11]. The values for $\boldsymbol{\gamma}_{\min}^{(c)}$ and $\boldsymbol{\gamma}_{\max}^{(c)}$ were picked based on experimental understanding of the specific robotic platform being used. In the case of quadrotors we found that limiting the acceleration ($c = 2$) led to good tracking performance of the underlying controller.

E. Optimization-Based Collision Avoidance

For collision avoidance we require the following inequality to hold throughout trajectory execution

$$\left\|\mathbf{\Theta}^{-1}(\mathbf{p}_i[k_t] - \mathbf{p}_j[k_t])\right\|_2 \geq r_{\min}, \quad \forall j \neq i, \tag{6}$$

where $\mathbf{\Theta}$ is a scaling matrix to obtain general ellipsoid safety boundaries, and $r_{\min}$ is the minimum distance between two agents before collision.

We explored two approaches: Buffered Voronoi Cells (BVC) [10], [11] and on-demand collision avoidance [4]. Both methods rely on the same principle of imposing hyperplane constraints that limit the available free space over which the agent is allowed to optimize its future inputs. In Fig. 3(a) we present a simple collision avoidance scenario with two agents in 2D.

In the BVC method, the agents are restricted to remain within their own Buffered Voronoi Cell, $\mathcal{V}_i$, for a time τ of their horizon. In this work, we define a Buffered Voronoi Cell similar to [10] but including the scaling matrix:

$$\mathcal{V}_i = \left\{ \mathbf{p} \in \mathbb{R}^3 \;\middle|\; \frac{\mathbf{\Theta}^{-2}(\mathbf{p}_i - \mathbf{p}_j)^\intercal(\mathbf{p} - \mathbf{p}_i)}{d_{i,j}} \geq \frac{r_{\min} - d_{i,j}}{2} \right\}, \forall j \neq i, \tag{7}$$

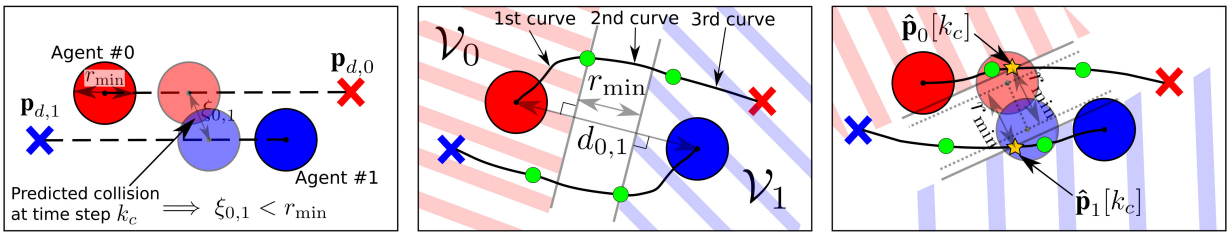
where $d_{i,j} = \|\mathbf{\Theta}^{-1}(\mathbf{p}_i - \mathbf{p}_j)\|_2$, and $\mathbf{p}_i$, $\mathbf{p}_j$ are the measured positions of agents i and j at time step k_t. Fig. 3(b) shows the BVCs calculated (shaded areas) for our two-agent example. The condition in (7) defines a linear constraint on the position of the agents to achieve collision avoidance. Let $\mathcal{P}_{i,1}$ be the set of control points of agent i corresponding to the first Bézier curve of the input. To achieve collision avoidance we impose the constraint $\mathcal{P}_{i,1} \in \mathcal{V}_i$, which translates to $p + 1$ constraints on the control points.

Collision-free updates are achieved with this method, as shown in Fig. 3(b).

On the other hand, the on-demand method of [4] relies on a predict-avoid paradigm for collision avoidance. It assumes communicative agents that share with the team a representation of their future actions. In our case, since the input and the state are closely related (reference trajectory and measured position), we have two options for collision avoidance:

- **State space**: constraints are imposed on the predicted states $\mathbf{X}_i$ of the agents, which can be obtained as a linear combination of the optimal inputs using (4). This results in collision-free *predicted* positions over the horizon.
- **Input space:** constraints are imposed on the inputs $\mathbf{U}_i$ directly, resulting in collision-free *reference* positions over the horizon.

For the general dynamics in system (1), non-intersecting trajectories in the input space would not necessarily achieve collision avoidance.



(a) Colliding scenario (b) BVC collision avoidance (c) On-demand collision avoidance

Fig. 3. Two-agent transition scenario in 2D. The agents are represented by a circle of certain radius. The X marks the intended goal of each agent. In (a) the dashed lines represent the nominal (colliding) trajectories, where the translucent circles represent the position of each agent at time step k_c in which the first collision is predicted. In (b) we show the input update using the BVC method. The green dots represent the concatenation points of the Bézier curves. The first curve is constrained to lie within the coloured zone for each agent. In (c) the agents update their inputs using on-demand collision avoidance, leveraging the predicted collision information to build the separating hyperplanes. The star represents the sample of the updated input which was constrained to be within the coloured zone.

Agent i detects the first predicted collision (in the state space) with any neighbour j at time step $k_{c,i}$ whenever

$$\xi_{ij} = \left\| \boldsymbol{\Theta}^{-1} \left(\hat{\mathbf{p}}_i[k_{c,i}|k_t - 1] - \hat{\mathbf{p}}_j[k_{c,i}|k_t - 1] \right) \right\|_2 \geq r_{\min}, \tag{8}$$

does not hold. For input space detection it suffices to replace predicted positions with predicted inputs (position reference). We define a subset Ω_i of neighbours of agent i for which collision constraints are constructed, defined as:

$$\Omega_i = \{ j \in \{1, \dots, N\} \mid \xi_{ij} < g(r_{\min}),\, j \neq i \}, \tag{9}$$

where $g(r_{\min})$ models the area around the agent for which collision avoidance is required. In this work we used $g(r_{\min}) = 2r_{\min}$. As proposed in [4], we can procure collision avoidance in the state space by enforcing a first-order approximation of the constraint

$$\left\| \boldsymbol{\Theta}^{-1} \left(\hat{\mathbf{p}}_i[k_{c,i} - 1|k_t] - \hat{\mathbf{p}}_j[k_{c,i}|k_t - 1] \right) \right\|_2 \geq r_{\min} + \varepsilon_{ij}, \quad \forall j \in \Omega_i, \tag{10}$$

where $\varepsilon_{ij} < 0$ are slack decision variables for relaxation.

Note that on-demand avoidance only constrains a specific sample of the curve at $k_{c,i}$, as shown with the yellow stars in Fig. 3(c). This sample must lie within a partition of the space given by the linearization of (10), whereas BVC constrains a complete segment of the curve. Comparing the resulting trajectories in Fig. 3(b) and Fig. 3(c), it is clear that on-demand avoidance leads to less conservative maneuvers than the BVC method. In Section VI we analyze how these insights impact the ability to complete multi-agent transition tasks.

In both cases, to implement collision avoidance we need only add an inequality constraint tuple $(\mathbf{A}_{\text{coll}}, \mathbf{b}_{\text{coll}})$ that satisfies $\mathbf{A}_{\text{coll}} \boldsymbol{\mathcal{U}}_i \leq \mathbf{b}_{\text{coll}}$.

F. Cost Function

We search to minimize a cost function which results from the sum of various terms. In this section we omit the subindex i for the tuning parameters of each term of the cost function, but each agent could have different values.

1) Error to Goal: this term drives the agent to its goal location. We aim to minimize the sum of errors between the positions at the last $\kappa < K$ time steps of the horizon and the goal location $\mathbf{p}_{d,i}$. The quadratic cost function is defined as

$$\mathcal{J}_{i,\text{error}} = \sum_{k=K-\kappa}^{K} q_k \left\| \hat{\mathbf{p}}_i[k|k_t] - \mathbf{p}_{d,i} \right\|_2^2, \tag{11}$$

where $q_k > 0$ are the positive weights of each time step. This expression can be formulated as a quadratic form in terms of the inputs and the measured state at k_t.

2) Energy: we minimize a weighted combination of the sum of squared derivatives, as in [6], [17]. The cost is defined as

$$\mathcal{J}_{i,\text{energy}} = \sum_{c=0}^{r} \alpha_c \int_0^{t_h} \left\| \frac{d^c}{dt^c} \hat{\mathbf{u}}_i(t) \right\|_2^2 dt, \tag{12}$$

where $\alpha_c > 0$ is a scalar weight for each derivative of the input, until the r^{th} derivative. This term can be evaluated in closed form to get a quadratic form in terms of $\boldsymbol{\mathcal{U}}_i$ [17].

3) Collision Constraint Violation: we implement on-demand collision avoidance as soft constraints, which requires a penalty term to be added in the cost function to limit the amount of relaxation of the constraints. For that we consider both quadratic and linear penalty costs

$$\mathcal{J}_{i,\text{violation}} = \zeta \left\| \varepsilon_{ij} \right\|_2^2 + \xi \varepsilon_{ij}, \tag{13}$$

where ζ and ξ are the weights of each term.

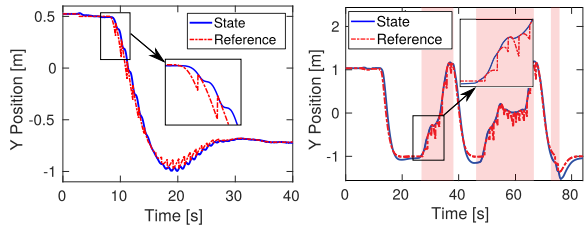
A similar approach can be used to relax the constraints in the BVC method, with the difference that for each neighbour j we add penalty terms for the $p + 1$ constraints on the control points of the first Bézier curve segment.

All the elements previously mentioned compose the following standard QP problem:

$$\begin{aligned} &\underset{\boldsymbol{\mathcal{U}}_i, \varepsilon_{ij}}{\text{minimize}}\ \mathcal{J}_{i,\text{error}} + \mathcal{J}_{i,\text{energy}} + \mathcal{J}_{i,\text{violation}} \\ &\text{subject to } \mathbf{A}_{\text{eq}} \boldsymbol{\mathcal{U}}_i = \mathbf{b}_{\text{eq}}, \mathbf{A}_{\text{in}} \boldsymbol{\mathcal{U}}_i \leq \mathbf{b}_{\text{in}}, \\ &\mathbf{A}_{\text{coll}} \boldsymbol{\mathcal{U}}_i \leq \mathbf{b}_{\text{coll}}, \varepsilon_{ij} \leq 0 \quad \forall j \in \Omega_i. \end{aligned} \tag{14}$$

IV. Event-Triggered Replanning

Choosing the initial condition for the input at each planning cycle to be equal to the current state of the robot was proposed in [11], but it has certain limitations. First, if we require C^r-continuity on the inputs, then we need to reliably measure the



(a) Continuous replanning (b) Event-triggered replanning

Fig. 4. Experimental data comparison of replanning strategies. Continuous replanning creates discontinuities in the reference signals that cause jittering in the state of the agent. The event-triggered strategy remedies this behaviour by introducing discontinuities only when the agents are being perturbed.

r^{th} derivative of the robot's position. Second, for imperfect trajectory tracking or systems with slow dynamics, this replanning strategy consistently causes (potentially big) discontinuities of the input to match the state of the robot, as shown in Fig. 4(a). Such discontinuities cause undesired jittering in the robot and slow down its progress to complete the task.

To address these concerns, we propose an event-triggered replanning strategy, in which we reset the input to match the states of the agent only whenever we detect the agent has been perturbed. To detect such an event, we heuristically designed an activation function that serves to detect disturbances to the agent. An example of such an activation function for second-order tracking dynamics is:

$$f_n[k_t] = \frac{(\mathrm{p}_{i,n}[k_t] - \mathrm{u}_{i,n}[k_t])^5}{-(\mathrm{v}_{i,n}[k_t] + \mathrm{sgn}(\mathrm{v}_{i,n}[k_t])\varepsilon)}, \quad n = 1, 2, 3 \tag{15}$$

where the subscript n represents the spatial component ($[x, y, z]$) of the vectors associated with agent i. The term $(\mathrm{p}_{i,n}[k_t] - \mathrm{u}_{i,n}[k_t])$ is the trajectory tracking error, and the term $\mathrm{sgn}(\mathrm{v}_{i,n}[k_t])\varepsilon$ with a small scalar $\varepsilon \ll 1$ is used to avoid singularities in $f_n[k_t]$. We assume $|\mathrm{v}_{i,n}[k_t]| > 0$, which is realistic in real-world operation due to noise in state estimation.

The intuition behind (15) is that we want to reset our reference signal whenever the tracking error grows large. However, designing an appropriate threshold value for the tracking error is tricky due to its high variability during execution. Instead, $f_n[k_t]$ is designed to detect whenever the error is growing but the velocity is either small or growing in the opposite direction of the error. To detect these scenarios, we define the robot is operating normally if the inequality

$$f_{\min} < f_n[k_t] < f_{\max} \tag{16}$$

holds for every element of $f_n[k_t]$. The values of $f_{\min}$ and $f_{\max}$ must be chosen by extracting the extrema of $f_n[k_t]$ under normal operation. If (16) does not hold, then the agent is being disturbed and we set the initial position and velocity of the Bézier curve to match the states of the vehicle, while setting higher-order derivatives to zero. In summary,

$$\mathbf{u}_{0,i}[k_t] = \begin{cases} \hat{\mathbf{u}}_i[1|k_t - 1], & \text{if } f_{\min} < f_n[k_t] < f_{\max}, \\ (\bar{\mathbf{x}}_i[k_t], \mathbf{0}), & \text{else.} \end{cases} \tag{17}$$

In order to validate the proposed replanning strategy, we conducted an experiment with our quadrotor platform while a human operator perturbed it along its path. The task of the quadrotor was to reach a y-coordinate of -1 m. The reference signal and state of the quadrotor are shown in Fig. 4(b), where the red segments mean the agent was being disturbed. During these disturbed stages we observe how the reference signal is frequently reset to match the state of the robot. The replanning helps the quadrotor continue its task whenever the disturbance is removed. Under normal operation (white segments) the replanning is not required, which leads to a smooth reference signal that avoids the shortcomings observed in Fig. 4(a). Note that we assume that the perturbations not detected by the activation function (15) can be rejected by the underlying controller, which we validated experimentally. The video that accompanies this letter showcases the strategy working in experiments with quadrotors. These disturbances would have led to crashing and mission failure with typical offline approaches (e.g., [4]), since there is no adaptation of the pre-planned reference signal, and the underlying control system would have been unable to reject the perturbations.

Algorithm 1: Input Updates for Agent i.

Input: Current states of all agents ($\mathbf{x}[k_t]$), target location ($\mathbf{p}_{d,i}$)
Output: Commands to be applied from t_0 to $t_0 + h$ with sampling of T_s ($\bar{\mathbf{u}}_i$)

1: `setTargetLocation`($\mathbf{p}_{d,i}$)
2: $\boldsymbol{\Pi}[k_t - 1] \leftarrow$ `receiveAgentPredictions()`
3: $\mathbf{u}_{0,i}[k_t] \leftarrow$ `getInitRef`($\bar{\mathbf{x}}_i[k_t]$, $\hat{\mathbf{u}}_i[1|k_t - 1]$)
4: ($\mathbf{A}_{\text{coll}}$, $\mathbf{b}_{\text{coll}}$) $\leftarrow$ `getCollision` ($\bar{\mathbf{x}}[k_t]$, $\boldsymbol{\Pi}[k_t - 1]$)
5: QP $\leftarrow$ `buildQP` ($\mathbf{A}_{\text{coll}}$, $\mathbf{b}_{\text{coll}}$, $\mathbf{u}_{0,i}[k_t]$, $\bar{\mathbf{x}}_i[k_t]$)
6: $\boldsymbol{\mathcal{U}}_i \leftarrow$ `solve`(QP)
7: $\boldsymbol{\Pi}_i[k_t] \leftarrow$ `broadcastUpdatedHorizon` ($\boldsymbol{\mathcal{U}}_i$, $\bar{\mathbf{x}}_i[k_t]$)
8: $\hat{\mathbf{u}}_i[1|k_t] \leftarrow$ `updateInitialReference` ($\boldsymbol{\mathcal{U}}_i$)
9: $\bar{\mathbf{u}}_i \leftarrow$ `getSampledInput` ($\boldsymbol{\mathcal{U}}_i$)
10: **return** $\bar{\mathbf{u}}_i$

V. The Algorithm

In this section we describe the core algorithm used to update the optimal input sequence for each agent, outlined in Algorithm 1. As stated, the algorithm is conceived to be executed in a distributed fashion by a group of agents with communication capabilities. It takes as inputs the measured state of each agent and the desired location of agent i. For execution we consider two different time bases: one with a coarse time step h, used for the MPC planning, and one with a refined time step T_s used for commanding the agents at a higher rate. With this definition, the output of Algorithm 1 is the set of inputs for agent i in the time frame in-between planning cycles, i.e., $t \in [t_0,\ t_0 + h]$ with sample rate T_s. In other words, the output are subsamples of the input between $\hat{\mathbf{u}}_i[0|k_t]$ and $\hat{\mathbf{u}}_i[1|k_t]$. This subsampling process is exact and not an approximation, due to the chosen continuous parameterization of trajectories.

In line 1 we build the error penalties given by (11), which is only required if the setpoint $\mathbf{p}_{d,i}$ of the agent changes. In line 2 each agent receives the latest predictions of all the other agents

through some communication channel, which is considered ideal in this letter, with no delays or packet drops. In lines 3–9 we update the input sequence of each agent. Note that the first time the algorithm is executed, the latest predictions may be initialized trivially by assuming static neighbours. First, in line 3 we apply the event-triggered replanning strategy to decide the value of the initial condition of the input. The collision avoidance constraint (BVC or on-demand) is constructed in line 4. Note that BVC would not require the prediction information, meaning that there is no communication between agents, but instead it would require the measured state of the agents. Conversely, on-demand avoidance only requires the predictions and not the measured states. Lines 5–6 build and solve the standard quadratic programming problem outlined in (14). Once the solution vector $\boldsymbol{\mathcal{U}}_i$ is obtained from the QP solver, we can then sample the resulting Bézier curves to obtain a sampled representation of the input (or the state). Line 7 updates the horizon for the agent and broadcasts the information to the rest of the team. Line 8 updates the initial condition of the reference to be used in the next planning cycle, in the case where replanning is not required. Lastly, line 9 samples the resulting Bézier curve with period T_s to obtain the sequence to be applied for $t \in [t_0,\, t_0 + h]$.

Since our physical platform does not have inter-agent communication capabilities, Algorithm 1 is implemented in a centralized manner from an offboard computer. The centralized implementation of Algorithm 1 is executed in parallel for all the agents, since there is no inter-agent data dependency for the input updates.

VI. Simulation Results

We created a simulation environment in MATLAB 2017a and executed it on a PC with Intel Xeon CPU with 8 cores and 16 GB of RAM, running at 3 GHz. The agents were modeled after the Crazyflie 2.0 quadrotor, using $r_{\min} = 0.3\,\text{m}$ and $\boldsymbol{\Theta} = \text{diag}([1, 1, 2])$, but a collision was declared using $r_{\text{coll}} = 0.2\,\text{m}$ (closer to the physical size of the quadrotor) and $\boldsymbol{\Theta}_{\text{coll}} = \text{diag}([1, 1, 2.25])$. The trajectory tracking dynamics were identified by fitting a second-order model to experimental data from the step response of the system depicted in Fig. 2. We selected a step of $h = 0.2\,\text{s}$, which means that trajectories are replanned at only 5 Hz.

For the input sequence we chose Bézier curves with $p = 5$, $l = 3$ and $t_h = 3\,\text{s}$, where each segment had a fixed duration of 1 second. Additionally, we imposed actuation limits with $\gamma^{(2)}_{\max} = -\gamma^{(2)}_{\min} = 1\,\text{m/s}^2$. After tuning the cost function, we selected $\kappa = 3, q_k = 100, \alpha_2 = 0.008, \zeta = 1$ and $\xi = -5 \times 10^4$. For the replanning function in (15) we chose $\epsilon = 0.01$, $f_{\min} = -0.01$, and $f_{\max} = 0.8$. We also added noise in the measured state $\bar{\mathbf{x}}_i[k_t]$ based on empirical data gathered from an overhead motion capture system.

A. Comparison of Collision Avoidance Methods

We compared four different optimization-based collision avoidance methods in random transition scenarios: 1) BVC as proposed in [11] (without the discrete planner component), 2) BVC using soft constraints, 3) On-demand collision avoidance applied in the state space and 4) On-demand collision avoidance in the input space. Discrete planning was removed for comparison purposes, but it should be noted that, in general, they can improve the performance of any of the methods. For instance, discrete planners may provide intermediate goal points for the agents to complete the transition task, fitting seamlessly with our current setup.

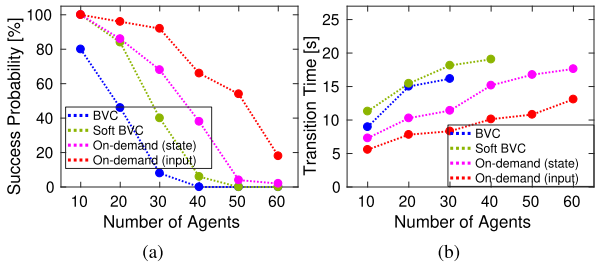


Fig. 5. Simulation performance comparison of various collision avoidance strategies. We considered different numbers of agents in a fixed volume of 18 m^3. For each swarm size, 50 different random test cases were generated and averaged.

We considered a fixed-volume, obstacle-free workspace of 18 m^3 (roughly the size of our indoor flight arena), with randomly generated initial and final locations for all agents. The number of agents varied from 10 to 60, in order to test the algorithms as the agent density increased. A trial was considered successful if all agents were able to reach their goals without collisions and within 20 seconds. After each simulation we ran a collision check (using r_{coll} and $\boldsymbol{\Theta}_{\text{coll}}$) and a goal check (allowing 10 cm distance from the target location) to determine if the test was successful.

In Fig. 5 we show the performance obtained using each method. The success probability for each swarm size considered is highlighted in Fig. 5(a). We notice that as the number of agents increases (ergo, a denser workspace) the effectiveness of the BVC methods decay drastically. Using soft constraints helps, but ultimately the approach is too conservative to resolve transition scenarios with a high density of agents.

On the other hand, the on-demand collision avoidance strategy shows better performance when applied in the input space, especially in high agent density workspaces. Input-space collision avoidance achieved more than 90% success rate with swarm sizes up to 30 agents. We observe a significant decline in performance after 30 agents in all the tested methods. This is a weakness of our approach given the need to relax collision constraints in order to find solutions. As the density grows, then higher relaxations will be required to solve the transitions, which may result in collisions.

One explanation to the performance difference between state and input space avoidance resides on the agent model. In the identified dynamics, the position of the agents is, essentially, a delayed version of the input signal (with some overshoot). Thus, by treating collision avoidance in the input space, the agents are preemptively avoiding each other, which ultimately leads to less collisions during execution. Also, by using the inputs as opposed to predicted states, the collision avoidance is less sensitive to the model's accuracy. We note that for other linear

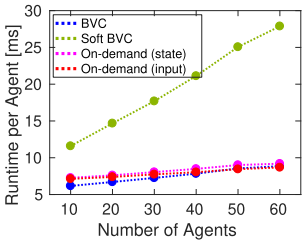


Fig. 6. Comparison of the average runtime per agent to update the inputs using our on-demand collision avoidance and the BVC method. The data shown is the average over 50 randomly generated tests for each swarm size considered.

systems where the state and the input are completely different quantities, performing input-space collision avoidance is not guaranteed to be safe (since non-colliding inputs do not translate into non-colliding positions) and state-space avoidance should be preferred.

In the same order of ideas, Fig. 5(b) shows that, on average, using on-demand collision avoidance leads to faster transition times than the BVC methods, averaging around 50% transition time reductions. These numbers match the analysis made on Section III-E using Fig. 3. We also analyzed the average travelled distance by all the agents as a measure of optimality. The simulation data suggested that all the methods produce trajectories of almost the same length, with a slight advantage to the soft BVC method which produces, on average, trajectories around 3% shorter than the rest.

B. Runtime Benchmark

We compared the computation time per agent to update their input sequence. In Fig. 6 the results are presented, where we specifically show the average time per agent to solve the associated QP problem.

To formally analyze the scaling of both algorithms, define N_{i,k_t} to be the number of nearby neighbours of agent i to be considered for collision avoidance at time step k_t. The amount of inequality constraints on both BVC and on-demand methods scale with $\mathcal{O}(N_{i,k_t})$. The soft BVC method adds additional $(p+1) \times N_{i,k_t}$ slack variables to relax the constraints, while the on-demand methods add only N_{i,k_t} new decision variables to the problem. In Fig. 6 we observe the empirical runtime of the considered methods. The soft BVC method has the slowest runtime, due to the added slack variables and overall bigger problems to be solved. For the other three methods the runtime is fairly similar, with a slight advantage to the BVC method.

VII. Experimental Results

The online generation method outlined in this letter (with on-demand avoidance) was implemented in C++, using ROS to manage the drone swarm and qpOASES [18] as the QP solver. In this section we provide experimental results using our Crazyflie 2.0 swarm testbed. All the inputs were computed from a single computer and broadcast to the swarm through a radiolink, alongside the estimated position of each individual agent given by a motion capture system. The computer specs and algorithm parameters are the same as Section VI, with the exception of $\xi = -1 \times 10^3$ and the addition of $T_s = 0.05\,\text{s}$ (trajectories were being sent to the swarm at 20 Hz).

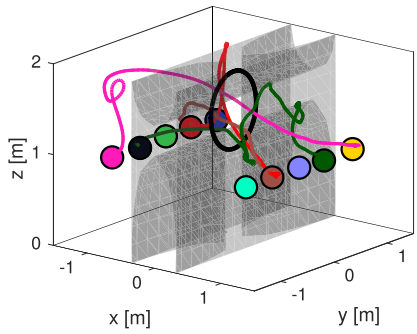


Fig. 7. A 10-drone transition passing through a hula-hoop (denoted by the black circle). The forbidden space is defined by four ellipsoids acting as static obstacles. The coloured circles denote the initial locations of the agents, and the corresponding coloured lines are the followed trajectories towards the antipodal goal locations (showing only four trajectories for clarity).

A video summarizing the experimental results can be found at `http://tiny.cc/online-dmpc`.

A. Obstacle-Free Transitions

We considered different swarm sizes, ranging from 2 to 20 drones. For each swarm size, three independent flights were executed, where each flight consisted in 30 seconds of randomly generated transitions. The agents were restricted to move in a $3 \times 3 \times 2\,\text{m}^3$ volume, and we used $r_{\min} = 0.35\,\text{m}$ as the safety distance.

For each test, we recorded the average computation time to execute Algorithm 1 and the minimum inter-agent distance during the flight. The results are summarized in Table I. As expected, the average computation time increases as we add more agents to the problem, since all computations are being executed by a single computer. The interesting result is that the scaling we obtain in runtime is pseudo-linear, since we are able to parallelize the computation thanks to the distributed nature of the approach.

The minimum inter-agent distance decreases as we increase the number of agents, i.e., there is less available space to move collision-free. Since the optimizer is allowed to violate the collision constraint, the original margin of $r_{\min} = 0.35\,\text{m}$ is violated if required. Such scenarios of violation appear more often the higher the agent density in the workspace. Although this is suboptimal from a safety perspective, experiments show that as long as a sufficiently large $r_{\min}$ is chosen, the amount of violation incurred while optimizing will still allow the agents to move collision-free.

B. Transition Tasks with Static Obstacles

We tasked a group of drones to exchange positions with each other by passing through a hula-hoop with a 85 cm diameter. The environment was divided by an invisible wall with a passage-way defined by the hula-hoop. In Fig. 7 we show the 10-drone transition scenario solved in experiments. Note that static obstacles are added to the problem as new "neighbours" for each agent, with

TABLE I
EXPERIMENTAL RESULTS SUMMARY FOR RANDOM TRANSITION TASKS INVOLVING INCREASING NUMBER OF AGENTS

# Agents	2	4	6	8	10	12	14	16	18	20
Avg. solve time of Alg. 1 [ms]	3.3	6.2	9.9	13.8	16.9	17.6	20.3	20.5	23.4	28.3
Std. solve time of Alg. 1 [ms]	0.4	0.9	1.7	3.3	5.1	7.6	8.0	8.9	10.2	11.4
Min. distance [cm]	36.2	32.2	31.1	30.0	29.2	28.6	29.1	26.0	26.1	25.3

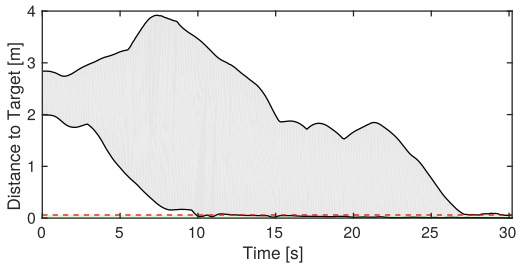


Fig. 8. Distance to target envelope (minimum and maximum over time) of the 10-drone hula-hoop transition task. The light green section near the bottom represents the zone where transition success is declared: a 6 cm radius of the target location. In this case the transition was completed in $T_f = 28\,\mathrm{s}$.

their own ellipsoidal parameters, which means that the runtime complexity scales linearly with the number of static obstacles.

The restricted zone was modeled as the union of four ellipsoids. They are shaped in such a way that they are intersecting and provide a small gap of $30 \times 30\,\mathrm{cm}^2$ for the agents to pass through. In the tracked trajectories we observe that some of the agents were able to fly directly through the circle, while others took detours in order to let other agents pass first. The distance-to-goal envelope shown in Fig. 8 demonstrates how the agents make progress over time to decrease the distance towards their goal, eventually converging to it within some tolerance region.

Several different tuning parameters were tried while solving this particular task. While using input space collision avoidance, a wide range of penalty gains and maximum accelerations worked well to solve the task; the completion time T_f varied from 20.1 to 48.4 seconds in 18 different trials, with $\xi \in [-1.5, -0.5] \times 10^3$. In most cases, a penalty of $\xi < -2 \times 10^3$ resulted in oscillatory behaviour near the obstacles, which sometimes led to deadlocks. On the other hand, the success rate using state space collision avoidance was much lower, resulting in deadlocks much more frequently than with input-space avoidance.

Non-convex obstacles could lead to agents getting stuck in local minima, in which case the use of discrete planners to provide intermediate waypoints may be required.

VIII. CONCLUSION AND FUTURE WORK

In this letter we presented a framework for multi-robot online trajectory generation based on distributed model predictive control (DMPC). In transition tasks, our method has a higher success rate and lower travel times than using the Buffered Voronoi Cells method. The simulations indicated more than 90% success rate with up to 30 palm-sized quadrotor agents in a $18\,\mathrm{m}^3$ arena.

The parallelization of the method leads to high scalability. In experiments we were able to send trajectories in real-time (20 Hz) for a swarm of 20 quadrotors. Our approach showed satisfactory results in a complicated transition scenario passing through a hula-hoop, and robust replanning in the presence of unmodeled disturbances.

One interesting area of future work includes accommodating a more realistic communication channel in the formulation, in order to deal with delays and packet drops.

REFERENCES

[1] F. Augugliaro, A. P. Schoellig, and R. D'Andrea, "Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2012, pp. 1917–1922.

[2] Y. Chen, M. Cutler, and J. P. How, "Decoupled multiagent path planning via incremental sequential convex programming," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2015, pp. 5954–5961.

[3] R. Van Parys and G. Pipeleers, "Distributed model predictive formation control with inter-vehicle collision avoidance," in *Proc. Asian Control Conf.*, 2017, pp. 2399–2404.

[4] C. E. Luis and A. P. Schoellig, "Trajectory generation for multiagent point-to-point transitions via distributed model predictive control," *IEEE Robot. Autom. Lett.*, vol. 4, no. 2, pp. 375–382, Apr. 2019.

[5] M. Čáp, P. Novák, J. Vokrínek, and M. Pěchouček, "Multi-agent RRT: Sampling-based cooperative pathfinding," in *Proc. Int. Conf. Auton. Agents Multi-Agent Syst.*, 2013, pp. 1263–1264.

[6] W. Hönig, J. A. Preiss, T. S. Kumar, G. S. Sukhatme, and N. Ayanian, "Trajectory planning for quadrotor swarms," *IEEE Trans. Robot.*, vol. 34, no. 4, pp. 856–869, Aug. 2018.

[7] M. Hamer, L. Widmer, and R. D'Andrea, "Fast generation of collision-free trajectories for robot swarms using GPU acceleration," *IEEE Access*, vol. 7, pp. 6679–6690, 2018.

[8] J. Van Den Berg, J. Snape, S. J. Guy, and D. Manocha, "Reciprocal collision avoidance with acceleration-velocity obstacles," in *Int. Conf. Robot. Autom.*, 2011, pp. 3475–3482.

[9] J. Alonso-Mora, P. Beardsley, and R. Siegwart, "Cooperative collision avoidance for nonholonomic robots," *IEEE Trans. Robot.*, vol. 34, no. 2, pp. 404–420, Apr. 2018.

[10] D. Zhou, Z. Wang, S. Bandyopadhyay, and M. Schwager, "Fast, on-line collision avoidance for dynamic vehicles using buffered voronoi cells," *IEEE Robot. Autom. Lett.*, vol. 2, no. 2, pp. 1047–1054, Apr. 2017.

[11] B. Şenbaşlar, W. Hönig, and N. Ayanian, "Robust trajectory execution for multi-robot teams using distributed real-time replanning," in *Distrib. Auton. Robot. Syst.*, 2019, pp. 167–181.

[12] B. Hernandez and P. Trodden, "Distributed model predictive control using a chain of tubes," in *Proc. UKACC 11th Int. Conf. Control*, 2016, pp. 1–6.

[13] A. Nikou and D. V. Dimarogonas, "Decentralized tube-based model predictive control of uncertain nonlinear multiagent systems," *Int. J. Robust Nonlinear Control*, vol. 29, no. 10, pp. 2799–2818, 2019.

[14] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2011, pp. 2520–2525.

[15] K. I. Joy, "Bernstein polynomials," *On-Line Geometric Modeling Notes*, vol. 13, 2000.

[16] T. Mercy, R. Van Parys, and G. Pipeleers, "Spline-based motion planning for autonomous guided vehicles in a dynamic environment," *IEEE Trans. Control Syst. Technol.*, vol. 26, no. 6, pp. 2182–2189, Nov. 2018.

[17] R. Charles, A. Bry, and N. Roy, "Polynomial trajectory planning for quadrotor flight," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2013.

[18] H. Ferreau, C. Kirches, A. Potschka, H. Bock, and M. Diehl, "qpOASES: A parametric active-set algorithm for quadratic programming," *Math. Program. Comput.*, vol. 6, no. 4, pp. 327–363, 2014.