Trajectory Generation for Multiagent Point-To-Point Transitions via Distributed Model Predictive Control

Carlos E. Luis and Angela P. Schoellig

***Abstract*—This letter introduces a novel algorithm for multiagent offline trajectory generation based on distributed model predictive control. Central to the algorithm's scalability and success is the development of an on-demand collision avoidance strategy. By predicting future states and sharing this information with their neighbors, the agents are able to detect and avoid collisions while moving toward their goals. The proposed algorithm can be implemented in a distributed fashion and reduces the computation time by more than 85% compared to previous optimization approaches based on sequential convex programming, while only having a small impact on the optimality of the plans. The approach was validated both through extensive simulations and experimentally with teams of up to 25 quadrotors flying in confined indoor spaces.**

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

I. Introduction

GENERATING collision-free trajectories when dealing with multiagent systems is a safety-critical task. In missions that require cooperation of multiple agents, such as warehouse management [1], we often must safely drive agents from their current locations to a set of final positions. Solving this task, known as multiagent point-to-point transition, is therefore an integral part of any robust multiagent system.

There are two main variations of the multiagent point-to-point transition problem: the labelled and the unlabelled agent problem. In the former, each agent has a fixed final position that cannot be exchanged with another agent [2], [3]; in the latter, the algorithm is free to assign the goals to the agents, as to ease the complexity of the transition problem [4]. This letter focuses on the labelled agent problem.

A common approach is to formulate this as an optimization problem. One of the first techniques developed relied on Mixed Integer Linear Programming (MILP), modelling collision constraints with binary variables [2]. This method is computationally expensive and not suited for large groups of agents.

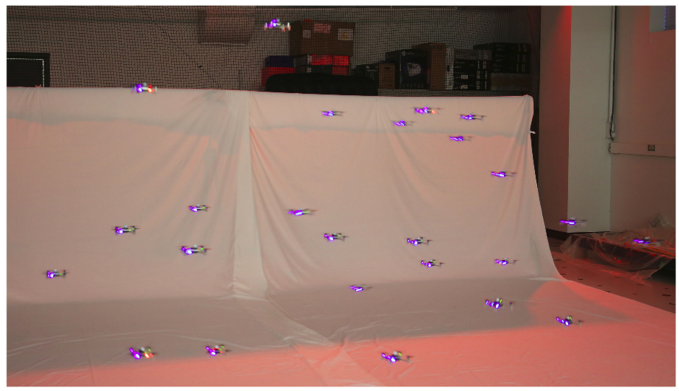


Fig. 1. A group of 25 Crazyflie 2.0 quadrotors performing a point-to-point transition using our distributed model predictive control (DMPC) algorithm. A video of the performance is found at `http://tiny.cc/dmpc-swarm`.

More recently, Sequential Convex Programming (SCP) [5] has been used to achieve faster computation compared to MILP. In [3], SCP is used to compute optimal-energy trajectories for quadrotor teams. Although useful for small teams, the algorithm does not scale well with the number of agents. A decoupled version of that algorithm was proposed in [6], [7], which provides better scalability at the cost of suboptimal solutions. However, the required decoupling leads to a sequential greedy strategy (i.e., turning agent trajectories previously solved for into obstacles for subsequent agents) with decreased success rate as the number of agents increases.

Discrete approaches divide the space into a grid and use known discrete search algorithms [8], limiting the initial and final locations to be vertices of the underlying grid. Other approaches combine optimization techniques and predefined behaviours to manage collisions in 2D [9].

Distributed optimization approaches can effectively include pair-wise distance constraints [10]. Furthermore, the computational effort is distributed among the agents and therefore reduced compared to centralized approaches. Optimal reciprocal collision avoidance (ORCA) leverages velocity obstacles to guarantee collision-free trajectories for holonomic [11] and nonholonomic [12] agents. While provably safe, the method may be overly conservative by assuming a constant velocity profile over the time horizon. Techniques based on potential fields have been used for decentralized collision avoidance [13], but they are susceptible to deadlocks.

Manuscript received September 10, 2018; accepted December 15, 2018. Date of publication January 1, 2019; date of current version January 11, 2019. This letter was recommended for publication by Associate Editor F. Arrichiello and Editor N. Y. Chong upon evaluation of the reviewers' comments. This work was supported in part by NSERC research and equipment under Grants RTI 2018-00847, CRDPJ 528161-18, and 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 1A1, Canada (e-mail: carlos.luis@robotics.utias.utoronto.ca; schoellig@utias.utoronto.ca).

This letter has supplementary downloadable material available at http://ieeexplore.ieee.org, provided by the authors. The Supplemental Material contains a video demonstrating the algorithms proposed in the letter using a swarm of 25 quadrotors. This material is 18.3 MB in size.

Digital Object Identifier 10.1109/LRA.2018.2890572

Distributed model predictive control (DMPC) [14] has been used in coordination tasks such as formation control [15], [16], but not explicitly for point-to-point transitions. Particularly interesting are synchronous implementations of DMPC [17], where the agents simultaneously update their predictions, reducing runtime by parallel computing.

Previous DMPC approaches achieved collision avoidance by either (1) using compatibility constraints that limit the position deviation of agents between prediction updates [18] or (2) imposing separating hyperplane constraints between the agents at every time step of the prediction horizon [15]. Both strategies are not well suited for transition tasks: strategy (1) drastically reduces the mobility of agents, especially in cluttered environments, while strategy (2) lacks scalability and is overly conservative, as demonstrated in Section V-A. In contrast, inspired by the incremental inclusion of *all* collision constraints over an *infinite* horizon proposed in [6], we introduce *on-demand collision avoidance* in a DMPC framework, where we detect and resolve only the *first* collision in the *finite* prediction horizon, reducing computation time and increasing the success rate for transition tasks. Our method is further enhanced by the use of soft collision constraints, as in [19].

The key contributions of this letter are three-fold: we introduce a novel on-demand collision avoidance strategy for DMPC, present a fast DMPC algorithm for multiagent point-to-point transitions, and provide a thorough empirical analysis of our method via simulations and real quadrotor experiments, as well as comparisons to existing approaches. To the best of our knowledge, our method is the first to be fast enough for midflight trajectory generation with 25 drones (computations are done upon request during flight).

The rest of the letter is organized as follows: Section II states the problem. Section III introduces the optimization formulation to solve it. The algorithm is presented in Section IV and demonstrated in simulation (Section V) and experiments with a swarm of quadrotors (Section VI).

II. Problem Statement

The goal is to generate collision-free trajectories that drive N agents from initial to final locations within a given amount of time, subject to state and actuation constraints. We aim to generate such trajectories offline and execute them with our experimental platform, the Crazyflie 2.0 quadrotor.

A. The Agents

The agents are modeled as unit masses in $\mathbb{R}^3$, with double integrator dynamics. This simplified model of a quadrotor with an underlying position controller is used to achieve faster computations. Higher-order models can be accommodated with minimum modifications in what follows. We use $\mathbf{p}_i[k]$, $\mathbf{v}_i[k]$, $\mathbf{a}_i[k]$ to represent the discretized x, y, z position, velocity and accelerations of agent i at time step k, where accelerations are the inputs. With a discretization step h, the dynamic equations are given by

$$\mathbf{p}_i[k+1] = \mathbf{p}_i[k] + h\mathbf{v}_i[k] + \frac{h^2}{2}\mathbf{a}_i[k], \tag{1}$$

$$\mathbf{v}_i[k+1] = \mathbf{v}_i[k] + h\mathbf{a}_i[k]. \tag{2}$$

B. Constraints

We constrain the motion of the agents to match the physics of the vehicle. First, the agents have limited actuation, which bounds its minimum and maximum acceleration,

$$\mathbf{a}_{\min} \leq \mathbf{a}_i[k] \leq \mathbf{a}_{\max}. \tag{3}$$

Secondly, the agents must remain within a volume (e.g., an indoor flying arena). We impose:

$$\mathbf{p}_{\min} \leq \mathbf{p}_i[k] \leq \mathbf{p}_{\max}. \tag{4}$$

C. Collision Avoidance

The collision avoidance constraint is designed such that the agents safely traverse the environment. In the case of quadrotors, aerodynamic effects from neighbouring agents may lead to crashes. Thus, we model the collision boundary for each agent as an ellipsoid elongated along the vertical axis to capture the downwash effect of the agents' propellers, similar to [8]. The collision constraint between agents i and j is defined using a scaling matrix $\mathbf{\Theta}$,

$$\left\| \mathbf{\Theta}^{-1} \left(\mathbf{p}_i[k] - \mathbf{p}_j[k] \right) \right\|_n \geq r_{\min}, \tag{5}$$

where n is the degree of the ellipsoid ($n = 2$ is a usual choice) and $r_{\min}$ is the minimum distance between agents in the xy plane. The scaling matrix $\mathbf{\Theta}$ is defined as $\mathbf{\Theta} = \mathrm{diag}(a, b, c)$. We choose $a = b = 1$ and $c > 1$. Thus, the required minimum distance in the vertical axis is $r_{z,\min} = cr_{\min}$. Note that the constraint in (5) checks whether agent j (or i), modelled as a 3D point, is inside an ellipsoid centered around agent i (or j).

III. Distributed Model Predictive Control

The problem formulated in Section II can be translated into an optimization problem. In single-agent standard model predictive control (MPC), an optimization problem is solved at each time step, which finds an optimal input sequence over a given prediction horizon based on a model that describes the agent's dynamics. The first input of that sequence is applied to the real system and the resulting state is measured, which is the starting point for the next optimization problem. In an offline planning scenario such as ours, we do not measure the agent's state after applying an input (since there is no physical agent yet), instead we apply the input directly to the model to compute the next step of the generated trajectory. The same procedure is repeated until the whole trajectory is generated. This methodology can be applied in a distributed fashion, where each agent executes the iterative optimization to generate trajectories, but with the possibility of sharing information with neighbouring agents.

A. The Synchronous Algorithm

Our approach is based on synchronous DMPC, where the agents share their previously predicted state sequence with their neighbours before simultaneously solving the next optimization problems. At every discrete-time index k_t, each agent simultaneously computes a new input sequence over the horizon following these steps:

1) Check for future collisions using the latest predicted states of the neighbours, computed at time step $k_t - 1$.
2) Build the optimization problem, including state and actuation constraints, and collision constraints *only if required*.
3) After obtaining the next optimal sequence, the first element is applied to the model and the agents move one step ahead. Future states are predicted over the horizon and shared with the other agents.

Predicting collisions and including constraints only if needed is the basic idea behind on-demand collision avoidance. We only include those constraints associated with the first predicted collisions. The process is repeated until all agents reach their desired goals. Below we derive the mathematical setup of the optimization problem.

B. The Agent Prediction Model

Using the dynamics in (1) and (2), we can develop a linear model to express the agents' states over a horizon of fixed length K. First 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. In what follows, $k \in \{0, \dots, K-1\}$ is the discrete-time index of the prediction horizon. The dynamic model of agent i is given by

$$\begin{bmatrix} \hat{\mathbf{p}}_i[k+1|k_t] \\ \hat{\mathbf{v}}_i[k+1|k_t] \end{bmatrix} = \begin{bmatrix} \mathbf{I_3} & h\mathbf{I}_3 \\ \mathbf{0}_3 & \mathbf{I}_3 \end{bmatrix} \begin{bmatrix} \hat{\mathbf{p}}_i[k|k_t] \\ \hat{\mathbf{v}}_i[k|k_t] \end{bmatrix} + \begin{bmatrix} (h^2/2)\mathbf{I}_3 \\ h\mathbf{I}_3 \end{bmatrix} \hat{\mathbf{a}}_i[k|k_t], \tag{6}$$

with $\mathbf{I}_3$ being a 3×3 identity matrix and $\mathbf{0}_3$ a 3×3 matrix of zeros. We select the acceleration as the model's input (and variable to optimize). A compact representation is

$$\hat{\mathbf{x}}_i[k+1|k_t] = \mathbf{A}\hat{\mathbf{x}}_i[k|k_t] + \mathbf{B}\hat{\mathbf{u}}_i[k|k_t], \tag{7}$$

where $\hat{\mathbf{x}}_i \in \mathbb{R}^6$, $\mathbf{A} \in \mathbb{R}^{6\times6}$, $\mathbf{B} \in \mathbb{R}^{6\times3}$ and $\hat{\mathbf{u}}_i \in \mathbb{R}^3$ (model input). Define the initial state at instant k_t, $\mathbf{X}_{0,i} = \mathbf{x}_i[k_t]$. Then we can write the position sequence $\mathbf{P}_i \in \mathbb{R}^{3K}$ as an affine function of the input sequence $\mathbf{U}_i \in \mathbb{R}^{3K}$,

$$\mathbf{P}_i = \mathbf{A}_0\mathbf{X}_{0,i} + \mathbf{\Lambda}\mathbf{U}_i, \tag{8}$$

where $\mathbf{\Lambda} \in \mathbb{R}^{3K\times3K}$ is defined as

$$\mathbf{\Lambda} = \begin{bmatrix} \mathbf{\Psi B} & \mathbf{0}_3 & \dots & \mathbf{0}_3 \\ \mathbf{\Psi A B} & \mathbf{\Psi B} & \dots & \mathbf{0}_3 \\ \vdots & \ddots & \ddots & \vdots \\ \mathbf{\Psi A}^{K-1}\mathbf{B} & \mathbf{\Psi A}^{K-2}\mathbf{B} & \dots & \mathbf{\Psi B} \end{bmatrix}, \tag{9}$$

with matrix $\mathbf{\Psi} = \begin{bmatrix} \mathbf{I_3} & \mathbf{0}_3 \end{bmatrix}$ selecting the first three rows of the matrix products (those corresponding to the position states). Lastly, $\mathbf{A}_0 \in \mathbb{R}^{3K\times6}$ reflects the propagation of the initial state,

$$\mathbf{A}_0 = \begin{bmatrix} (\mathbf{\Psi A})^\intercal & (\mathbf{\Psi A}^2)^\intercal & \dots & (\mathbf{\Psi A}^K)^\intercal \end{bmatrix}^\intercal. \tag{10}$$

C. Objective Function

The objective function that is minimized to compute the optimal input sequence has three main components: trajectory error, control effort and input variation. A similar formulation can be found in [20].

1) Trajectory Error Penalty: This term drives the agents to their goals. We aim to minimize the sum of errors between the positions at the last κ time steps of the horizon and the desired final position $\mathbf{p}_{d,i}$. The error term is defined as

$$e_i = \sum_{k=K-\kappa}^{K} \left\| \hat{\mathbf{p}}_i[k|k_t] - \mathbf{p}_{d,i} \right\|_2. \tag{11}$$

This term can be turned into a quadratic cost function in terms of the input sequence using (8),

$$J_{e,i} = \mathbf{U}_i^\intercal(\mathbf{\Lambda}^\intercal\tilde{\mathbf{Q}}\mathbf{\Lambda})\mathbf{U}_i - 2(\mathbf{P}_{d,i}^\intercal\tilde{\mathbf{Q}}\mathbf{\Lambda} - (\mathbf{A}_0\mathbf{X}_{0,i})^\intercal\tilde{\mathbf{Q}}\mathbf{\Lambda})\mathbf{U}_i, \tag{12}$$

where $\tilde{\mathbf{Q}} \in \mathbb{R}^{3K\times3K}$ is a positive definite and block-diagonal matrix that weights the error at each time step. A value of $\kappa = 1$ leads to $\tilde{\mathbf{Q}} = \text{diag}(\mathbf{0}_3, \dots, \mathbf{Q})$ with matrix $\mathbf{Q} \in \mathbb{R}^{3\times3}$ chosen as a diagonal positive definite matrix. Higher values of κ lead to more aggressive agent behaviour with agents moving faster towards their goals, but may also lead to overshooting at the target location.

2) Control Effort Penalty: We also aim to minimize the control effort using the quadratic cost function

$$J_{u,i} = \mathbf{U}_i^\intercal\tilde{\mathbf{R}}\mathbf{U}_i. \tag{13}$$

Similarly, $\tilde{\mathbf{R}} \in \mathbb{R}^{3K\times3K}$ is positive definite and block-diagonal, $\tilde{\mathbf{R}} = \text{diag}(\mathbf{R}, \dots, \mathbf{R})$, where $\mathbf{R} \in \mathbb{R}^{3\times3}$ weights the penalty on the control effort.

3) Input Variation Penalty: This term is used to minimize variations of the acceleration, leading to smooth input trajectories. We define the quadratic cost

$$\delta_i = \sum_{k=0}^{K-1} \left\| \hat{\mathbf{u}}_i[k|k_t] - \hat{\mathbf{u}}_i[k-1|k_t] \right\|_2. \tag{14}$$

To transform (14) into a quadratic form, first we define a matrix $\mathbf{\Delta} \in \mathbb{R}^{3K\times3K}$,

$$\mathbf{\Delta} = \begin{bmatrix} \mathbf{I_3} & \mathbf{0}_3 & \mathbf{0}_3 & \dots & \mathbf{0}_3 & \mathbf{0}_3 \\ -\mathbf{I}_3 & \mathbf{I}_3 & \mathbf{0}_3 & \dots & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & -\mathbf{I}_3 & \mathbf{I}_3 & \dots & \mathbf{0}_3 & \mathbf{0}_3 \\ \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \dots & -\mathbf{I}_3 & \mathbf{I}_3 \end{bmatrix}, \tag{15}$$

and introduce the vector $\mathbf{U}_{i*} \in \mathbb{R}^{3K}$ to include the term $\mathbf{u}_i[k_t - 1]$ (previously applied input),

$$\mathbf{U}_{i*} = \begin{bmatrix} \mathbf{u_i[k_t - 1]}^\intercal & \mathbf{0}_{3\times1}^\intercal & \dots & \mathbf{0}_{3\times1}^\intercal \end{bmatrix}^\intercal. \tag{16}$$

Finally, we write (14) in quadratic form as

$$J_{\delta,i} = \mathbf{U}_i^\intercal(\mathbf{\Delta}^\intercal\tilde{\mathbf{S}}\mathbf{\Delta})\mathbf{U}_i - 2(\mathbf{U}_{i*}^\intercal\tilde{\mathbf{S}}\mathbf{\Delta})\mathbf{U}_i, \tag{17}$$

where $\tilde{\mathbf{S}} \in \mathbb{R}^{3K\times3K}$ is positive definite and block-diagonal, defined as $\tilde{\mathbf{S}} = \text{diag}(\mathbf{S}, \dots, \mathbf{S})$, where $\mathbf{S} \in \mathbb{R}^{3\times3}$ weights the penalty on control variation. The cost function $\mathcal{J}_i$ is obtained by adding together (12), (13) and (17),

$$\mathcal{J}_i(\mathbf{U}_i) = J_{e,i} + J_{u,i} + J_{\delta,i}. \tag{18}$$

D. Physical Limits

When computing the input sequence over the horizon, the agents must satisfy constraints (3) and (4). Define $\mathbf{P}_{\min}, \mathbf{P}_{\max}, \mathbf{U}_{\min}, \mathbf{U}_{\max} \in \mathbb{R}^{3K}$ to be

$$\begin{aligned} \mathbf{P}_{\min} &= [\mathbf{p}_{\min}^{\intercal} \cdots \mathbf{p}_{\min}^{\intercal}]^{\intercal}; \quad \mathbf{P}_{\max} = [\mathbf{p}_{\max}^{\intercal} \cdots \mathbf{p}_{\max}^{\intercal}]^{\intercal} \\ \mathbf{U}_{\min} &= [\mathbf{a}_{\min}^{\intercal} \cdots \mathbf{a}_{\min}^{\intercal}]^{\intercal}; \quad \mathbf{U}_{\max} = [\mathbf{a}_{\max}^{\intercal} \cdots \mathbf{a}_{\max}^{\intercal}]^{\intercal}. \end{aligned} \tag{19}$$

The physical limits are formulated as

$$\begin{aligned} \mathbf{P}_{\min} - \mathbf{A}_0 \mathbf{X}_{0,i} \leq \mathbf{\Lambda} \mathbf{U}_i \leq \mathbf{P}_{\max} - \mathbf{A}_0 \mathbf{X}_{0,i} \\ \mathbf{U}_{\min} \leq \mathbf{U}_i \leq \mathbf{U}_{\max}. \end{aligned} \tag{20}$$

Lastly, we can vertically stack both inequality constraints in (20) to obtain a single expression: $\mathbf{A}_{\text{in}} \mathbf{U}_i \leq \mathbf{b}_{\text{in}}$.

E. Convex Optimization Problem, No Collision Case

If agent i does not detect any future collisions, then it updates its input sequence by solving:

$$\begin{aligned} &\underset{\mathbf{U}_i}{\text{minimize}} \ \mathcal{J}_i(\mathbf{U}_i) \\ &\text{subject to } \mathbf{A}_{\text{in}} \mathbf{U}_i \leq \mathbf{b}_{\text{in}}. \end{aligned} \tag{21}$$

The formulation in (21) results in a quadratic programming problem with 3 K decision variables and 12 K inequality constraints, which scales independently of N.

F. On-demand Collision Avoidance With Soft Constraints

The previous formulation is useful for scenarios where the agents can follow straight lines to their goals without colliding. In a more general setting, agents must avoid each other constantly to reach their goals. To implement on-demand collision avoidance, we leverage the predictive nature of DMPC to detect colliding trajectories and impose constraints to avoid the *first* predicted collision. This strategy differs from [6] since we do not attempt to incrementally resolve *all* predicted collisions, only the most relevant one.

Agent i detects a collision at time step $k_{c,i}$ of the previously considered horizon whenever the inequality

$$\xi_{ij} = \left\| \mathbf{\Theta}^{-1} \left(\hat{\mathbf{p}}_i[k_{c,i}|k_t - 1] - \hat{\mathbf{p}}_j[k_{c,i}|k_t - 1] \right) \right\|_n \geq r_{\min} \tag{22}$$

does not hold with a neighbour j. Note that at solving time k_t, the agents only have information of the other agents computed at $k_t - 1$, meaning that the collision is predicted to happen at time $k_{c,i} + k_t - 1$. In what follows, $k_{c,i}$ represents the *first* time step of the horizon where agent i predicts a collision with any neighbour. We include collision constraints with the subset of agents Ω_i defined as

$$\Omega_i = \{ j \in \{1, \ldots, N\} \mid \xi_{ij} < f(r_{\min}), i \neq j \},$$

where $f(r_{\min})$ models the radius around the agent, which defines the neighbours to be considered as obstacles when solving the problem. For example, we may include all agents within a radius 3 times bigger than the collision boundary, then $f(r_{\min}) = 3\, r_{\min}$. Limiting Ω_i to be the subset of neighbours within a radius of agent i intends to safely reduce the amount of collision constraints in the optimization.

If the agent detects collisions, it must include collision constraints to compute the new input sequence. To account for infeasibility issues while solving the optimization problem, we formulate the following relaxed collision constraint:

$$\left\| \mathbf{\Theta}^{-1} \left(\hat{\mathbf{p}}_i[k_{c,i} - 1|k_t] - \hat{\mathbf{p}}_j[k_{c,i}|k_t - 1] \right) \right\|_n \geq r_{\min} + \varepsilon_{ij}, \tag{23}$$

where $\varepsilon_{ij} < 0$ is a new decision variable that relaxes the constraint. Note that at k_t, we aim to optimize the value of $\hat{\mathbf{p}}_i[k_{c,i} - 1|k_t]$ to satisfy (23). The constraint is linearized using a Taylor series expansion about the previous predicted position of agent i at time $k_{c,i} + k_t - 1$, namely $\hat{\mathbf{p}}_i[k_{c,i}|k_t - 1]$,

$$\boldsymbol{\nu}_{ij}^{\intercal} \hat{\mathbf{p}}_i[k_{c,i}|k_t] - \varepsilon_{ij} \xi_{ij} \geq \rho_{ij} \tag{24}$$

with $\boldsymbol{\nu}_{ij} = \mathbf{\Theta}^{-n} (\hat{\mathbf{p}}_i[k_{c,i}|k_t - 1] - \hat{\mathbf{p}}_j[k_{c,i}|k_t - 1])^{n-1}$ and $\rho_{ij} = r_{\min} \xi_{ij} + \xi_{ij}^n + \boldsymbol{\nu}_{ij}^{\intercal} \hat{\mathbf{p}}_i[k_{c,i}|k_t - 1]$. On the left-hand side of (24), we note that the constraint is imposed on the position at time $k_t + k_{c,i}$ ($\hat{\mathbf{p}}_i[k_{c,i}|k_t]$), which is one time step after the predicted collision. This choice was made based on an empirical assessment of the algorithm's performance on a wide range of transition scenarios. It was found that by imposing the constraint one time step after the predicted collision, the agents exhibited more preemptive collision avoidance capabilities and were able to complete the transitions faster on average.

To turn the collision constraint into an affine function of the decision variables, first we augment the previous formulation to include the relaxation variables. Consider $\mathbf{E}_i \in \mathbb{R}^{n_{c,i}}$, with $n_{c,i} = \dim(\Omega_i)$, defined as the stacked vector of all ε_{ij}. We now introduce the augmented decision vector $\boldsymbol{\mathcal{U}}_i \in \mathbb{R}^{3K + n_{c,i}}$, obtained by concatenating vectors $\mathbf{U}_i$ and $\mathbf{E}_i$. The matrices derived above can be easily augmented to account for the augmented decision vector, by completing them with zeros where multiplied with the vector $\mathbf{E}_i$. We turn (24) into an affine function of the decision variables,

$$\boldsymbol{\mu}_{ij}^{\intercal} \mathbf{\Lambda} \mathbf{U}_i - \varepsilon_{ij} \xi_{ij} \geq \rho_{ij} - \boldsymbol{\mu}_{ij}^{\intercal} \mathbf{A}_0 \mathbf{X}_{0,i}, \tag{25}$$

where $\boldsymbol{\mu}_{ij} \in \mathbb{R}^{3K}$ is defined as

$$\boldsymbol{\mu}_{ij} = \begin{bmatrix} \mathbf{0}_{3(k_{c,i}-1)\times 1}^{\intercal} & \boldsymbol{\nu}_{ij}^{\intercal} & \mathbf{0}_{3(K-k_{c,i})\times 1}^{\intercal} \end{bmatrix}^{\intercal}. \tag{26}$$

By stacking the inequalities in (25) for the $n_{c,i}$ colliding neighbours, we obtain the complete collision constraint,

$$\mathbf{A}_{\text{coll}} \boldsymbol{\mathcal{U}}_i \leq \mathbf{b}_{\text{coll}}. \tag{27}$$

Additionally, we impose $-\varepsilon_{\max} \leq \varepsilon_{ij} \leq 0$ in order to bound the amount of relaxation allowed. We also consider the following linear and quadratic cost terms to penalize the relaxation on the collision constraint:

$$f_{\varepsilon,i} = \varrho \begin{bmatrix} \mathbf{0}_{3K\times 1}^{\intercal} & \mathbf{1}_{n_{c,i}\times 1}^{\intercal} \end{bmatrix}^{\intercal}, \mathbf{H}_{\varepsilon,i} = \zeta \begin{bmatrix} \mathbf{0}_{3K\times 3K} & \mathbf{0}_{3K\times n_{c,i}} \\ \mathbf{0}_{n_{c,i}\times 3K} & \mathbf{I}_{n_{c,i}} \end{bmatrix}$$

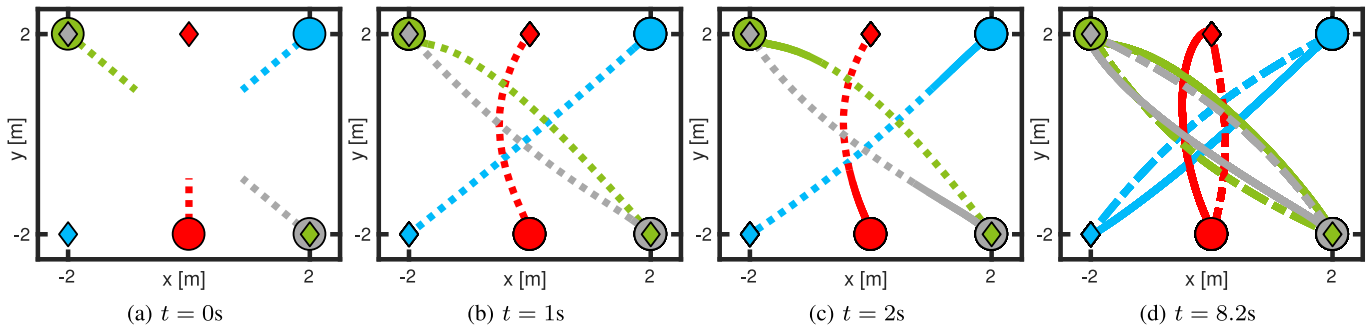
where ϱ, $\zeta > 0$ are scalar tuning parameters, measuring how much the relaxation is penalized. The augmented cost function in the collision avoidance case is defined as

$$\mathcal{J}_{\text{aug},i}(\boldsymbol{\mathcal{U}}_i) = \mathcal{J}(\mathbf{U}_i) + \boldsymbol{\mathcal{U}}_i^{\intercal} \mathbf{H}_{\varepsilon,i} \boldsymbol{\mathcal{U}}_i - f_{\varepsilon,i}^{\intercal} \boldsymbol{\mathcal{U}}_i. \tag{28}$$

Finally, the convex optimization problem with collision avoidance for agent i is formulated as

$$\begin{aligned} &\underset{\boldsymbol{\mathcal{U}}_i}{\text{minimize}} \ \mathcal{J}_{\text{aug},i}(\boldsymbol{\mathcal{U}}_i) \\ &\text{subject to } \mathbf{A}_{\text{in,aug}} \boldsymbol{\mathcal{U}}_i \leq \mathbf{b}_{\text{in,aug}}. \end{aligned} \tag{29}$$

The subscript 'aug' indicates the use of augmented state matrices, as outlined before. The inequality tuple $(\mathbf{A}_{\text{in,aug}}, \mathbf{b}_{\text{in,aug}})$ is



(a) $t = 0$s (b) $t = 1$s (c) $t = 2$s (d) $t = 8.2$s

Fig. 2. Four-agent position exchange scenario in 2D solved using Algorithm 1. Circles and diamonds represent initial and final locations, respectively. Dotted lines in (a)–(c) represent the predicted positions over a 3-second horizon, solid lines are the generated trajectories and dashed lines in (d) are the trajectories generated by the centralized approach in [3]. Using the optimality criteria of the sum of travelled distances by all agents, the distributed plan is only slightly suboptimal when compared to the centralized approach.

obtained by vertically stacking the physical limits, the collision constraint and the relaxation variable bounds. The augmented problem has $3\,K + n_{c,i}$ decision variables and $12\,K + 3\,n_{c,i}$ inequality constraints.

IV. The Algorithm

The proposed DMPC algorithm for point-to-point transitions is outlined in Algorithm 1. It requires as input the initial and desired final locations for N agents $(\mathbf{p}_0, \mathbf{p}_f)$, and outputs the trajectories that complete the transition. Variables $\mathbf{p}, \mathbf{v}$ and $\mathbf{a}$ are defined as the concatenation of the transition trajectories for every agent, while $\mathbf{\Pi}$ is the concatenation of the latest predicted positions for all agents.

In line 1, every $\mathbf{\Pi}_i$ is initialized as a line from initial to final location with a constant velocity profile. Each agent's states are initialized to be at the corresponding initial position with zero velocity. The main loop (lines 3–11) repeatedly solves optimization problems for the N agents, building the transition trajectory until they arrive at their goals or a maximum number of time steps is exceeded. Convergence of the transition (line 10) is declared once *all* the agents are within a small radius of their goals. Note that for $k_t = 0$, we consider $\mathbf{a}_i[-1] = \mathbf{0}_{3\times 1}$. The inner loop (lines 4–9) can be solved either sequentially or in parallel, since there is no data dependency between the problems.

To build and solve the corresponding QP (line 5), first we check for predicted collisions over the horizon, as described in Section III-F. If no collisions are detected, we solve the reduced problem in (21), otherwise we solve the collision avoidance problem in (29). If the optimizer finds a solution to the QP, then we can propagate the states using (7) and obtain the predicted position and velocity over the horizon (lines 6–9). Lastly, if a solution for the transition was found, we interpolate the solution with time step T_s to obtain a higher resolution trajectory. An optional step is to scale the solution, as suggested in [6], to push the accelerations to the maximum allowed. Finally, in line 15 we perform a collision check by verifying that $\left\|\mathbf{\Theta}^{-1}\left(\mathbf{p}_i[k_t] - \mathbf{p}_j[k_t]\right)\right\|_n \geq r_{\min} - \varepsilon_{\text{check}}$ holds for every i, j and k_t of the *interpolated* solution. The value of $\varepsilon_{\text{check}} \geq \varepsilon_{\max}$ is user-defined and must reflect the safety limit of the physical agents, such that the algorithm can decide whether the solution is safe to execute or not. If the solution passes all sanity checks, then the algorithm is deemed successful, otherwise an empty solution is returned.

A. Example Scenario

To illustrate how DMPC manages colliding trajectories, Fig. 2 shows a transition problem for four agents in the plane. Initially, as shown in Fig. 2a, the agents follow a direct path towards their desired final locations. In Fig. 2 b, collisions are detected and considered in the optimization problem. After a few time steps, the agents obtain the non-colliding plan seen in Fig. 2c. The trajectories generated with a centralized approach are quite different than the DMPC trajectories, as shown in Fig. 2 d. However, the sum of travelled distance of all agents is fairly similar in both cases, with only a 1.7% increase for the distributed approach.

B. Limitations and Associated Mitigation Strategies

We now discuss the limitations of the proposed algorithm, along with associated mitigation strategies to overcome them.

1) *Infeasibility*: the optimization problem becomes infeasible when the constraint (27) cannot be satisfied given the acceleration and relaxation limits. Feasibility of the problem can be guaranteed, however, by locally increasing the relaxation bound $\varepsilon_{\max}$ until the constraint is satisfied. In line 5 of Algorithm 1 we apply this technique to ensure recursive feasibility of the problem. The variable $\varepsilon_{\max}$ is reset to its original value once a solution is found.
2) *Collisions*: the use of on-demand collision avoidance with soft constraints does not guarantee collision-free trajectories. The use of soft constraints may lead to partial violations of the collision constraints along the trajectory. Moreover, since the trajectory is specified in discrete-time, there may be collisions occurring between time steps [3]. Higher values of ϱ and ζ penalize the violation of the collision constraint more, rendering the agents more wary of avoiding collisions.
3) *Oscillations and deadlocks*: oscillations occur due to a lack of central coordination, where agents oscillate between possible trajectories to avoid a collision. An agent may get trapped in a local minima where it oscillates indefinitely and never reaches its goal (deadlock). Higher values of κ and $\mathbf{Q}$ encourage aggressiveness towards reaching the goal.

Algorithm 1: DMPC for Point-to-Point Transitions.

Input : Initial and final positions
Output: Position, velocity and acceleration trajectories

$[\mathbf{\Pi}, \mathbf{x}[0]] \leftarrow$ InitAllPredictions$(\mathbf{p}_0, \mathbf{p}_f)$
$k_t \leftarrow 0$, AtGoal $\leftarrow$ false
while *not AtGoal and* $k_t < K_{\max}$ **do**
  foreach *agent* $i = 1, ..., N$ **do**
    $\hat{\mathbf{a}}_i[k|k_t] \leftarrow$Build&SolveQP$(\mathbf{x}_i[k_t], \mathbf{a}_i[k_t - 1], \mathbf{\Pi})$
    if *QP feasible* **then**
      $\hat{\mathbf{x}}_i[k+1|k_t] \leftarrow$ GetStates$(\mathbf{x}_i[k_t], \hat{\mathbf{a}}_i[k|k_t])$
      $\mathbf{\Pi}_i \leftarrow \hat{\mathbf{p}}_i[k+1|k_t]$
      $\mathbf{x}_i[k_t+1], \mathbf{a}_i[k_t] \leftarrow \hat{\mathbf{x}}_i[1|k_t], \hat{\mathbf{a}}_i[0|k_t]$
  AtGoal $\leftarrow$ CheckGoal$(\mathbf{p}[k_t], \mathbf{p}_f)$
  $k_t \leftarrow k_t + 1$
if *AtGoal* **then**
  $[\mathbf{p}, \mathbf{v}, \mathbf{a}] \leftarrow$ ScaleTrajectory$(\mathbf{p}, \mathbf{v}, \mathbf{a}, \|\mathbf{a}_{\max}\|)$
  $[\mathbf{p}, \mathbf{v}, \mathbf{a}] \leftarrow$ Interpolate$(\mathbf{p}, \mathbf{v}, \mathbf{a}, T_s)$
  CheckCollisions$(\mathbf{p}, r_{\min} - \varepsilon_{\text{check}})$
return $[\mathbf{p}, \mathbf{v}, \mathbf{a}]$

We observed that oscillations are often present in the predictions of agents, but vanish after a few MPC cycles and do not appear in the generated trajectories. Failure to avoid collisions can be minimized by tuning the cost function appropriately, achieved by a good compromise between aggressiveness towards the goal and penalization of the constraint relaxation.

V. Simulations

This section provides a simulation analysis of the DMPC algorithm. Implementation was done in MATLAB 2017a (using a sequential implementation of Algorithm 1) and executed on a PC with an Intel Xeon CPU with 8 cores and 16 GB of RAM, running at 3 GHz. The agents were modelled based on the Crazyflie 2.0 platform, using $r_{\min} = 0.35$ m, $a_{\max} = -a_{\min} = 1$ m/s^2, and $c = 2$ (to avoid downwash).

A. Comparison of Collision Avoidance Strategies in DMPC

To validate our on-demand collision avoidance scheme with soft constraints, we compared the performance to two other methods: (1) using hard collision constraints in every time step of the horizon (as in [15]) and (2) implementing our on-demand collision avoidance with hard constraints (i.e., constraint (23) without the relaxation variable). All methods were tested in scenarios with random sets of initial and final positions. We kept the density of the workspace (defined as agent/m^3) constant and varied the amount of agents from 20 to 200. All three approaches shared the time step parameters $h = 0.2$ s and $T_s = 0.01$ s. We used a horizon length $K = 15$, parameter $\kappa = 1$, a maximum relaxation of $\varepsilon_{\max} = 0.05$ m for the optimizer, a maximum relaxation of $\varepsilon_{\text{check}} = 0.05$ m for the safety check and a maximum time to complete the transition $T_{\max} = 20$ s. Fig. 3a shows the success rate of DMPC for point-to-point transitions using the different collision avoidance schemes. If we use hard constraints

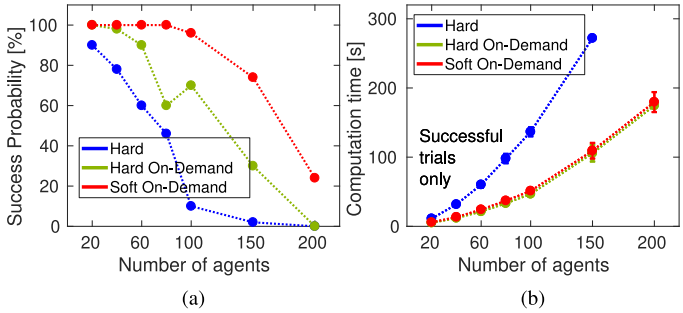


Fig. 3. Performance comparison of different collision avoidance strategies in DMPC, for an increasing number of agents within a workspace with a fixed agent density of 1 agent/m^3. For every swarm size considered, 50 different random test cases were generated.

at every time step (blue lines), the success rate suffers due to the inability of the agents to arrive at their final locations. The agents display conservative behaviour to maintain collision-free updates along their predictions, which may preclude progress towards the goal. On the other hand, the use of on-demand collision avoidance with hard constraints may lead to infeasible optimization problems, since the agents may be unable to avoid collisions within their acceleration limits. Our soft constraint strategy resolves the problem and achieves more than 75% success rate with up to 150 agents, clearly outperforming the other two methods. The decrease in success rate for 200 agents is partially due to insufficient time to complete the transition leading to 55% of the failures; with more agents and a fixed agent density (i.e., a larger environment) the average time to complete a random transition increases. This may mean that 55% of the transitions are infeasible independent of the algorithm used. In addition, the introduction of more decision-making agents leads to more collisions (45% of the failures). In Fig. 3b we highlight the reduction in computation time with our on-demand collision avoidance strategy.

B. Comparison to SCP-Based Approaches

We compared the performance of our proposed DMPC scheme with two state-of-the-art algorithms: centralized [3] and decoupled [6] SCP. We used the same simulation parameters as in Section V-A, but the volume of the workspace was kept fixed at 4 m^3, and the number of agents ranged from 4 to 20. We increased the value of κ to 2 to encourage agents to move to their goals, which showed better performance for high-density environments. Since the centralized and decoupled approaches require a fixed arrival time, we first solved each test using DMPC and determined the required time to complete the transition, and then set that as the arrival time of the SCP methods. Similar results were obtained by setting a fixed arrival time for the SCP methods for every trial (i.e., not based on the DMPC completion time), and are omitted. If DMPC failed to solve, the arrival time was set to $T_{\max} = 20$ s. Both SCP methods were executed until convergence was achieved or the problem was deemed infeasible.

Fig. 4a shows the probability of success as the density of agents increases. The proposed DMPC algorithm was able to

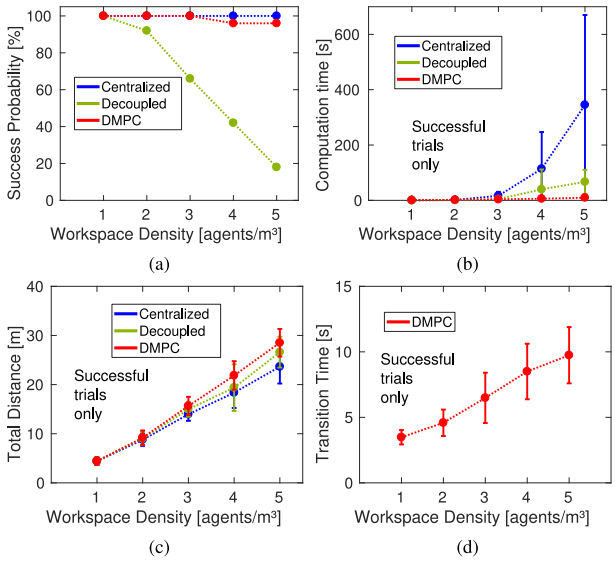


Fig. 4. Performance comparison of DMPC against SCP-based approaches, in a fixed $4\,\mathrm{m}^3$ volume. For every density considered, 50 different random test cases were generated.

find a solution in more than 95% of the trials, for every density scenario considered. The centralized approach was able to find a solution in every case, while the decoupled approach failed increasingly with increasing density.

As for the computation time, Fig. 4b shows a reduction of up to 97% in computation time with respect to centralized SCP and of 85% with decoupled SCP. The runtime variance observed in the other two approaches is due to the test-by-test variance in arrival time, as seen in Fig. 4d. Note that this DMPC implementation does not exploit the parallelizable nature of the algorithm yet and already achieves significantly lower runtimes.

To measure the optimality of the generated trajectories we analysed the sum of travelled distances by the agents, as highlighted in Fig. 4c. Our distributed approach produces longer paths on average, with respect to both the centralized and decoupled SCP. The suboptimality increases with workspace density, since the agents actively adjust their trajectories to avoid collisions, and oftentimes those adjustments lead to non-optimal paths towards their goals.

VI. Experiments

In this section we present experimental results using Algorithm 1 as an offline trajectory planner for a swarm of Crazyflies 2.0. The algorithm was implemented in C++ using OOQP as the solver. A video of the performance is found at `http://tiny.cc/dmpc-swarm`.

A. Parallel DMPC

Leveraging the parallel nature of the inner loop of Algorithm 1, we can design a strategy that parallelizes the computation. The idea is to equally split the N agents into smaller clusters to be solved in parallel using a multicore processor. The optimization problems of the agents inside a cluster are solved sequentially, but with the advantage of iterating through

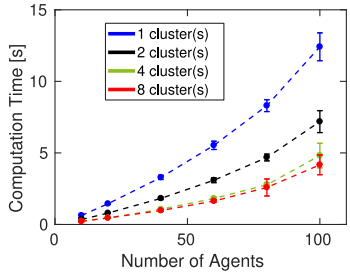


Fig. 5. Average computation time for different numbers of clusters. For each swarm size, we gathered data of 30 successful transitions and reported the mean and standard deviation (vertical bars) of the runtime.

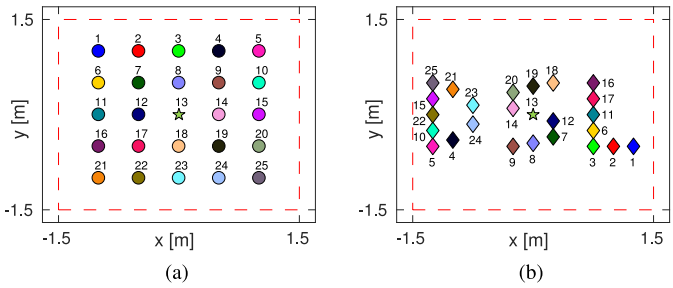


Fig. 6. A 25-agent transition scenario: (a) initial grid configuration, (b) target 'DSL' configuration. Circles and diamonds (of matching colour) represent initial and final locations for all agents, respectively. The star in the middle represents an agent acting as a static obstacle. The bounding box in dashed red lines represents the workspace boundaries.

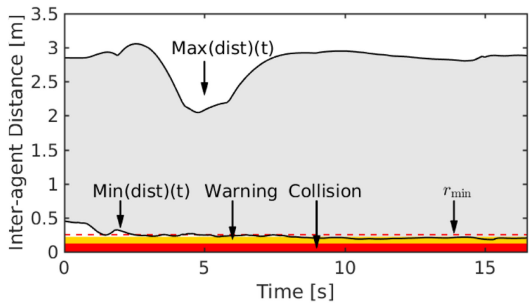
fewer agents. After all the clusters finish solving their QPs, they exchange the updated predictions and repeat the process.

In Fig. 5 we compare different numbers of clusters tested on a wide variety of transition scenarios. It was found that 8 clusters led to the best result for our computing hardware (CPU with 8 cores). This parallel strategy (8 clusters) reduced the computation time by more than 60% compared to using a purely sequential execution (1 cluster).

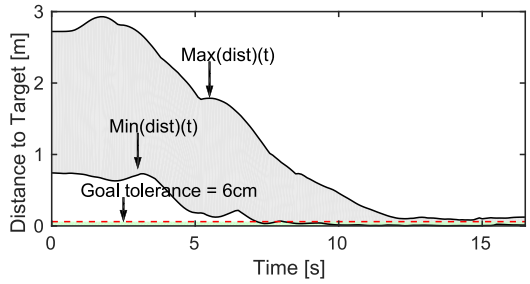
B. Swarm Transition

To perform the pre-computed transition motion on the quadrotors, we communicated via radio link with each drone and sent the following information at 100 Hz: (1) position setpoints and (2) position estimates from an overhead motion capture system. The setpoints were tracked using an on-board position controller based on [21]. One transition scenario is depicted in Fig. 6, in which the swarm was to transition from a 5×5 grid to a 'DSL' configuration. The difficulty of this particular scenario was increased by the central agent acting as a static obstacle (i.e., obstacle with fixed position).

We required $r_{\min} = 0.25\,\mathrm{m}$ with $\varepsilon_{\text{check}} = 0.03\,\mathrm{m}$. The DMPC algorithm was able to find a solution for this scenario in 1.8 seconds. In Fig. 7 a, the curves delimiting the gray area correspond to the minimum and maximum inter-agent distance at each time instant for six independent executions of the transition. Although trajectories are planned such that any inter-agent distance must remain above the warning zone (yellow band), the experimental curve goes slightly below that value. The warning zone is, in practice, a safety margin to compensate for unmodelled



(a)



(b)

Fig. 7. Experimental data from the transition depicted in Fig. 6, showing maximum and minimum distance values over 6 independent trials: (a) pairwise distances, (b) distances to target locations.

phenomena in our planning algorithm, such as imperfect trajectory tracking, time delays, and aerodynamics. Taking all these factors into account, it is natural for the minimum distance curve to go farther below than planned; however, it still remains above the collision zone. It is critical for the warning zone to be large enough, as to absorb any mismatch between the idealized planning and the real world. Its size is directly controlled by $r_{\min}$, which must be carefully chosen for robust trajectory executions.

Finally, Fig. 7b shows that the agents' progress towards their goal and are able to complete the transition up to some small tolerance. Once the agents enter the tolerance region below the dashed red line, they were commanded to hover in place. The on-board position controller reported a maximum error of close to 3 cm during hover, which explains why the maximum distance curve remains slightly above the tolerance region after all agents reached their goals.

In addition to the showcased scenario, the system has been tested on many randomly generated transitions, as can be seen in the video that accompanies this letter.

VII. Conclusions

The DMPC algorithm developed in this letter enables fast multiagent point-to-point trajectory generation. Using model-based predictions, the agents detect and avoid future collisions while moving to their goal locations. We introduced on-demand collision avoidance with soft constraints in a DMPC framework to enhance the scalability and success rate over previous approaches. As compared to SCP-based methods, we drastically reduce computational complexity, with only a small impact on the optimality of the plans. Our formulation allows for parallel computing, which further reduces the runtime.

We validated our method through an extensive empirical analysis using randomly generated transition tasks. Experimental results further validate our approach, which can be used to quickly calculate and execute transition trajectories for large teams of quadrotors, enabling new capabilities in applications such as drone shows.

References

[1] E. Guizzo, "Three engineers, hundreds of robots, one warehouse," *IEEE Spectrum*, vol. 45, no. 7, pp. 26–34, Jul. 2008.
[2] T. Schouwenaars, B. De Moor, E. Feron, and J. How, "Mixed integer programming for multi-vehicle path planning," in *Proc. Eur. Control Conf.*, 2001, pp. 2603–2608.
[3] 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.
[4] M. Turpin, N. Michael, and V. Kumar, "Decentralized formation control with variable shapes for aerial robots," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2012, pp. 23–30.
[5] S. Boyd, "Sequential convex programming," Lecture Notes, Stanford Univ., Stanford, CA, USA, 2008.
[6] Y. Chen, M. Cutler, and J. P. How, "Decoupled multiagent path planning via incremental sequential convex programming," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2015, pp. 5954–5961.
[7] 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 Robot. Autom. Lett.*, vol. 3, no. 2, pp. 1215–1222, Apr. 2018.
[8] J. A. Preiss, W. Hönig, N. Ayanian, and G. S. Sukhatme, "Downwash-aware trajectory planning for large quadrotor teams," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 250–257.
[9] S. Tang and V. Kumar, "A complete algorithm for generating safe trajectories for multi-robot teams," in *Robotics Research*. New York, NY, USA: Springer, 2018, pp. 599–616.
[10] S. Bhattacharya and V. Kumar, "Distributed optimization with pairwise constraints and its application to multi-robot path planning," in *Robotics: Science and Systems VI*, vol. 177, Cambridge, MA, USA: MIT Press, 2011.
[11] J. Van Den Berg, J. Snape, S. J. Guy, and D. Manocha, "Reciprocal collision avoidance with acceleration-velocity obstacles," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2011, pp. 3475–3482.
[12] J. Alonso-Mora, A. Breitenmoser, M. Rufli, P. Beardsley, and R. Siegwart, "Optimal reciprocal collision avoidance for multiple non-holonomic robots," in *Distributed Autonomous Robotic Systems*. New York, NY, USA: Springer, 2013, pp. 203–216.
[13] H. Rezaee and F. Abdollahi, "A decentralized cooperative control scheme with obstacle avoidance for a team of mobile robots," *IEEE Trans. Ind. Electron.*, vol. 61, no. 1, pp. 347–354, Jan. 2014.
[14] E. Camponogara, D. Jia, B. H. Krogh, and S. Talukdar, "Distributed model predictive control," *IEEE Control Syst.*, vol. 22, no. 1, pp. 44–52, Feb. 2002.
[15] 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.
[16] H. Sayyaadi and A. Soltani, "Decentralized polynomial trajectory generation for flight formation of quadrotors," *Proc. Inst. Mech. Eng., Part K, J. Multi-body Dyn.*, vol. 231, no. 4, pp. 690–707, 2017.
[17] L. Dai, Q. Cao, Y. Xia, and Y. Gao, "Distributed MPC for formation of multi-agent systems with collision avoidance and obstacle avoidance," *J. Franklin Inst.*, vol. 354, no. 4, pp. 2068–2085, 2017.
[18] P. Wang and B. Ding, "A synthesis approach of distributed model predictive control for homogeneous multi-agent system with collision avoidance," *Int. J. Control*, vol. 87, no. 1, pp. 52–63, 2014.
[19] A. Papen, R. Vandenhoeck, J. Bolting, and F. Defay, "Collision-free rendezvous maneuvers for formations of unmanned aerial vehicles," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 282–289, 2017.
[20] P. Ru, "Nonlinear model predictive control for cooperative control and estimation," Ph.D. dissertation, The University of Texas at Arlington, TX, USA, 2017.
[21] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2011, pp. 2520–2525.