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

Carlos E. Luis and Angela P. Schoellig

Abstract—This paper introduces a novel algorithm for multiagent offline trajectory generation based on distributed model predictive control (DMPC). By predicting future states and sharing this information with their neighbours, the agents are able to detect and avoid collisions while moving towards their goals. The proposed algorithm computes transition trajectories for dozens of vehicles in a few seconds. It reduces the computation time by more than 85% compared to previous optimization approaches based on sequential convex programming (SCP), with only causing a small impact on the optimality of the plans. We replaced the previous compatibility constraints in DMPC, which limit the motion of the agents in order to avoid collisions, by relaxing the collision constraints and enforcing them only when required. The approach was validated both through extensive simulations for a wide range of randomly generated transitions and with teams of up to 25 quadrotors flying in confined indoor spaces.

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 crop inspection [1] and warehouse management [2], it is desirable to safely drive the 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 component 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 other agents [3], [4]; in the latter, the algorithm is free to assign the goals to the agents, as to ease the complexity of the transition problem [5]. This paper will focus 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 as binary variables [3]. This method, although viable, is computationally expensive and not suited for large groups of agents.

More recently, Sequential Convex Programming (SCP) [6] has been used for faster computation compared to MILP. In [4], 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 is proposed in [7], [8], 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 as obstacles for subsequent agents) that decreases the success rate as the number of agents increases.

Discrete approaches divide the space in a grid and use known discrete search algorithms [9], but limit the initial and final locations to be vertices of the underlying grid. Other mixed approaches combine optimization techniques and predefined behaviours to manage collisions in 2D [10].

Distributed optimization approaches can effectively include pair-wise distance constraints [11]. 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 [12] and non-holonomic [13] agents. While provably safe, the method may be conservative by assuming a constant velocity profile over a time horizon, effectively limiting the motion of the agents. Distributed model predictive control (DMPC) [14] has been applied 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, further reducing execution time thanks to parallel computing.

Both ORCA and DMPC require explicit coordination of the agents in order to guarantee collision avoidance. In ORCA, the agents are aware of the collision avoidance policies of nearby neighbours, and coordinate to share the responsibility of avoiding the collision. In DMPC, a compatibility constraint is usually introduced to limit the position deviation between prediction updates [18]. In this paper, we...
empirically demonstrate that, in moderately large swarms, agents without these explicit coordination schemes still exhibit high collision avoidance capabilities. Furthermore, the removal of the explicit coordination has the potential of reducing algorithm complexity and conservative behaviour of the agents.

The key contributions of this paper are three-fold: we present a fast and parallelizable DMPC algorithm for multiagent point-to-point transitions, replace previous conservative compatibility constraints in DMPC by relaxing the collision constraints, and demonstrate the efficacy of the method with quadrotor transitions in high-density scenarios.

The rest of the paper 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 is a simplified model of a quadrotor equipped with an underlying position controller. We use $p_i[k], v_i[k], a_i[k]$ to represent the discretized 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

$$p_i[k+1] = p_i[k] + hv_i[k] + \frac{h^2}{2}a_i[k],$$

$$v_i[k+1] = v_i[k] + ha_i[k].$$

B. Constraints

We constrain the motion of the agents to match the physical world. First, the agents have limited actuation, which bounds its maximum acceleration,

$$a_{\text{min}} \leq a_i[k] \leq a_{\text{max}}.$$  \hspace{1cm} (3)

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

$$p_{\text{min}} \leq p_i[k] \leq p_{\text{max}}.$$  \hspace{1cm} (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 [9]. The collision constraint between agents $i$ and $j$ is defined using a scaling matrix $\Theta$,

$$\|\Theta^{-1}(p_i[k] - p_j[k])\|_n \geq r_{\text{min}},$$  \hspace{1cm} (5)

where $n$ is the degree of the ellipsoid ($n = 2$ is a usual choice) and $r_{\text{min}}$ is the minimum distance between agents in the XY plane. The scaling matrix $\Theta$ is defined as

$$\Theta = \begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{bmatrix}.$$  \hspace{1cm} (6)

We choose $a = b = 1$ and $c > 1$. Thus, the required minimum distance in the vertical axis is $r_{z,\text{min}} = cr_{\text{min}}$.

III. DISTRIBUTED MODEL PREDICTIVE CONTROL

The problem formulated in Sec. II can be naturally posed as 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 $k_i$, 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 $k_i - 1$.
2) Build the optimization problem, including state and actuation constraints, and collision constraints if we detect collisions.
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 other agents.

The process is repeated until all agents reach their desired goals. Below we formalize 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 $(\cdot)[k|k_i]$, which
represents the predicted value of $\langle \cdot | k_t + k \rangle$ with the information available at $k_t$. In what follows, $k \in \{0, \ldots, K - 1\}$ is the discrete-time index of the prediction horizon. The dynamic model of agent $i$ is given by

$$
\begin{bmatrix}
\dot{p}_i[k + 1|k_t] \\
\dot{v}_i[k + 1|k_t]
\end{bmatrix} = 
\begin{bmatrix}
I_3 & h I_3 \\
0_3 & I_3
\end{bmatrix}
\begin{bmatrix}
p_i[k|k_t] \\
v_i[k|k_t]
\end{bmatrix} + 
\begin{bmatrix}
h^2/2 I_3 \\
h I_3
\end{bmatrix}
\gamma_i[k|k_t],
$$

(7)

with $I_3$ being a $3 \times 3$ identity matrix and $0_3$ a $3 \times 3$ matrix of zeros. We select the acceleration as the model’s input (and variable to optimize). A compact representation is

$$
\dot{x}_i[k + 1|k_t] = A \gamma_i[k|k_t] + B \gamma_i[k|k_t],
$$

(8)

where $x_i \in \mathbb{R}^6$, $A \in \mathbb{R}^{6 \times 6}$, $B \in \mathbb{R}^{6 \times 3}$ and $\gamma_i \in \mathbb{R}^3$. Define the initial state at instant $k_t$, $x_{0,i} = x_i[k_t]$. Then we can write the position sequence $P_i \in \mathbb{R}^{3K}$ as an affine function of the input sequence $U_i \in \mathbb{R}^{3K}$,

$$
P_i = A_0 x_{0,i} + \Lambda U_i,
$$

(9)

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

$$
\Lambda = 
\begin{bmatrix}
\Psi B & 0_3 & \ldots & 0_3 \\
\Psi AB & \Psi B & \ldots & 0_3 \\
\vdots & \ddots & \ddots & \vdots \\
\Psi A^{K-1}B & \Psi A^{K-2}B & \ldots & \Psi B
\end{bmatrix},
$$

(10)

with matrix $\Psi = [I_3 \ 0_3]$ selecting the first three rows of the matrix products (those corresponding to the position states). Lastly, $A_0 \in \mathbb{R}^{3K \times 6}$ reflects the propagation of the initial state,

$$
A_0 = [(\Psi A)^T \ (\Psi A^2)^T \ \ldots \ (\Psi A^K)^T]^T.
$$

(11)

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 [19].

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 $\kappa$ time steps of the horizon and the desired final position $p_{d,i}$. The error term is defined as

$$
e_i = \sum_{k=K-\kappa}^K \|\hat{p}_i[k|k_t] - p_{d,i}\|_2.
$$

(12)

This term can be turned into a quadratic cost function in terms of the input sequence using (9),

$$
J_{e,i} = U_i^T (A^T \tilde{Q} A) U_i - 2 (P_{d,i}^T \tilde{Q} A - (A_0 x_{0,i})^T \tilde{Q} A) U_i,
$$

(13)

where $\tilde{Q} \in \mathbb{R}^{3K \times 3K}$ is a positive semidefinite and block-diagonal matrix that weights the error at each time step. A value of $\kappa = 1$ leads to $\tilde{Q} = \text{diag}(0_3, \ldots, 0_3)$ with matrix $Q \in \mathbb{R}^{3 \times 3}$ chosen as a diagonal positive semidefinite matrix. Higher values of $\kappa$ 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

$$
J_{u,i} = U_i^T \tilde{R} U_i.
$$

(14)

Similarly, $\tilde{R} \in \mathbb{R}^{3K \times 3K}$ is positive semidefinite and block-diagonal, $\tilde{R} = \text{diag}(R, \ldots, R)$, where $R \in \mathbb{R}^{3 \times 3}$ 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

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

(15)

To transform (15) into a quadratic form, first we define a matrix $\Delta \in \mathbb{R}^{3K \times 3K}$,

$$
\Delta = 
\begin{bmatrix}
I_3 & 0_3 & 0_3 & \ldots & 0_3 & 0_3 \\
- I_3 & I_3 & 0_3 & \ldots & 0_3 & 0_3 \\
0_3 & - I_3 & I_3 & \ldots & 0_3 & 0_3 \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots \\
0_3 & 0_3 & 0_3 & \ldots & - I_3 & I_3
\end{bmatrix},
$$

(16)

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

$$
U_{i*} = [u_i[k_t - 1]^T \ 0_{3 \times 1}^T \ldots \ 0_{3 \times 1}^T]_i.
$$

(17)

Finally, we write (15) in quadratic form as

$$
J_{\delta,i} = U_i^T (\Delta^T \tilde{S} \Delta U_i - 2 (U_{i*}^T \tilde{S} \Delta U_i),
$$

(18)

where $\tilde{S} \in \mathbb{R}^{3K \times 3K}$ is positive semidefinite and block-diagonal, defined as $\tilde{S} = \text{diag}(S, \ldots, S)$, where $S \in \mathbb{R}^{3 \times 3}$ weights the penalty on control variation. The total cost function $J_i$ is obtained by adding together (13), (14) and (18),

$$
J_i(U_i) = U_i^T (A^T \tilde{Q} A + \tilde{R} + \Delta^T \tilde{S} \Delta U_i - 2 (P_{d,i}^T \tilde{Q} A - (A_0 x_{0,i})^T \tilde{Q} A + U_i^T \tilde{S} \Delta U_i).
$$

(19)

D. Physical Limits

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

$$
P_{\text{min}} = [p_{\text{min}}^T \ p_{\text{min}}^T \ p_{\text{min}}^T]_i; \quad P_{\text{max}} = [p_{\text{max}}^T \ p_{\text{max}}^T \ p_{\text{max}}^T]^T,
$$

$$
U_{\text{min}} = [a_{\text{min}}^T \ a_{\text{min}}^T \ a_{\text{min}}^T]^T; \quad U_{\text{max}} = [a_{\text{max}}^T \ a_{\text{max}}^T \ a_{\text{max}}^T]^T.
$$

(20)

The physical limits are formulated as

$$
P_{\text{min}} - A_0 x_{0,i} \leq \Lambda U_i \leq P_{\text{max}} - A_0 x_{0,i},
$$

$$
U_{\text{min}} \leq U_i \leq U_{\text{max}}.
$$

(21)

Lastly, the inequality constraints of the problem are stacked and brought in the form $A_u U_i \leq b_u$.

E. Convex Optimization Problem, No Collision Case

If agent $i$ does not detect any future collisions, then it updates its input sequence by solving the following opti-
minimize \( J_t(U_i) \) subject to \( A_m U_i \leq b_m \). \( (22) \)

The formulation in (22) results in a quadratic programming problem with 3K decision variables and 12K inequality constraints, which means it scales independently of the number of agents.

**F. Collision Avoidance As Soft Constraints**

The previous formulation is useful for simple planning 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. In order to avoid collisions, we leverage the predictive nature of DMPC to detect colliding trajectories using an adapted version of (5). Agent \( i \) detects a collision violation at time step \( k_{c,i} \) of the previously computed horizon whenever the inequality

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

(23)

does not hold with any 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 \( \Omega_i \) 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 \( f(r_{\min}) = 3r_{\min} \). The main idea is that by limiting the size of the subset \( \Omega_i \), the amount of inequality constraints for agent \( i \) can be reduced while still including all the critical information to compute its next action.

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

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

(24)

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 \( \hat{p}_i[k_{c,i} | k_t - 1] \) to satisfy (24). 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 \( \hat{p}_i[k_{c,i} | k_t - 1] \).

\[
u^T_{ij} \hat{p}_i[k_{c,i} | k_t] - \varepsilon_{ij} \xi_{ij} \geq \rho_{ij}
\]

(25)

with \( \nu_{ij} = \Theta_i^{-n} (\hat{p}_i[k_{c,i} | k_t - 1] - \hat{p}_j[k_{c,i} | k_t - 1]) \) and \( \rho_{ij} = r_{\min} \xi_{ij} + \xi_{ij}^{(n)} + \nu^T_{ij} \hat{p}_i[k_{c,i} | k_t - 1] \). On the left-hand side of (25), we note that the constraint is imposed on the position at time \( k_t + k_{c,i} \) (\( \hat{p}_i[k_{c,i} | k_t] \)), 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. Furthermore, one can argue that the separating plane constraint in (25), although applied only at time \( k_t + k_{c,i} \), implicitly constraints the positions in the previous time steps of the horizon as well.

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 \( E_i \in \mathbb{R}^{n_{c,i}} \), with \( n_{c,i} = \dim(\Omega_i) \), defined as the stack vector of all \( \varepsilon_{ij} \). We now introduce the augmented decision vector \( U_i \in \mathbb{R}^{3K+n_{c,i}} \), obtained by concatenating vectors \( U_i \) and \( E_i \). 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 \( E_i \). We cast (25) into an affine function of the decision variables,

\[
\mu^T_{ij} A U_i - \varepsilon_{ij} \xi_{ij} \geq \rho_{ij} - \mu^T_{ij} A_0 X_0, i
\]

(26)

where \( \mu_{ij} \in \mathbb{R}^{3K} \) is defined as

\[
\mu_{ij} = \begin{bmatrix} 0^T_{3(k_{c,i}-1)} & \nu^T_{ij} & 0^T_{3(K-k_{c,i})} \end{bmatrix}^T.
\]

(27)

By stacking the inequalities in (26) for the \( n_{c,i} \) colliding
the trajectory to complete the transition. Variables 

\[ \mathbf{b}_{\text{coll}} \leq \mathbf{A}_{\text{coll}} \mathbf{U}_i \]  

(28)

Additionally, we impose \(-\varepsilon_{\text{max}} \leq \varepsilon_{ij} \leq 0\) 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:

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

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

\[ J_{\text{aug},i}(\mathbf{U}_i) = J(\mathbf{U}_i) + \mathbf{U}_i^T \mathbf{H}_{\varepsilon,i} \mathbf{U}_i - f_{\varepsilon,i}^T \mathbf{U}_i. \]  

(29)

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

\[
\begin{align*}
\text{minimize} & \quad J_{\text{aug},i}(\mathbf{U}_i) \\
\text{subject to} & \quad \mathbf{A}_{\text{in},\text{aug}} \mathbf{U}_i \leq \mathbf{b}_{\text{in,aug}}.
\end{align*}
\]  

(30)

The subscript ’aug’ indicates the use of augmented state matrices, as outlined before. The inequality tuple \((\mathbf{A}_{\text{in},\text{aug}}, \mathbf{b}_{\text{in,aug}})\) is obtained by stacking the physical limits, collision constraint and relaxation variable bounds. The augmented problem has \(3K + n_{c,i}\) decision variables and \(12K + 3n_{c,i}\) 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 trajectory to complete the transition. Variables \( \mathbf{p}, \mathbf{v}, \mathbf{a} \) are defined as the concatenation of the transition trajectories for every agent, while \( \Pi \) is the concatenation of the latest predicted positions for all agents.

In line 1, every \( \Pi_i \) is initialized as a line pointing 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. Note that for \( k_i = 0 \), 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 individual problems of each agent.

To build and solve the corresponding QP (line 5), first we check for predicted collisions over the horizon, as described in Sec. III-F. If no collisions are detected, we solve the reduced problem in (22), otherwise we solve the collision avoidance problem in (30). If the optimizer finds a solution to the QP, then we can propagate the states using (8) 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. An optional step is to scale the solution, as suggested in [7], to push the acceleration limits to the maximum allowed. Finally, we verify that no collisions were violated after interpolation (note that the scaling step only compresses the position profile of each agent, which has no effect on the pairwise distances). If the solution passes all the sanity checks, then the algorithm is deemed successful, otherwise an empty solution is returned.

A. Example Scenario

To illustrate how DMPC is able to manage 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. 2b, collisions are detected and the constraints start acting within the optimization problem. After a few time steps, the agents compute 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. 2d. However, the sum of travelled distance of all agents is fairly similar in both cases, with only a 1.7% increase in the distributed approach. A more thorough comparison is provided in Sec. V.

B. Repair Strategies

Since the algorithm acts as an offline trajectory planner, whenever it fails to find a solution we can apply repair strategies and retry solving the problem. We identify possible reasons for algorithm failure together with a repair strategy.

1) QP infeasible: An agent is unable to avoid collisions given the acceleration and relaxation limits. A proposed repair strategy is to locally retry solving the QP with an increased value for \( \varepsilon_{\text{max}} \).

2) Unavoided collision: This can happen when solving the problem (collision predicted at \( k_{c,i} = 0 \) means

---

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

**Input**: Initial and final positions

**Output**: Position, velocity and acceleration trajectories

1. \( \Pi, \mathbf{x}[0] \leftarrow \text{InitAllPredictions}(\mathbf{p}_0, \mathbf{p}_f) \)
2. \( k_i \leftarrow 0 \)
3. **while not AtGoal and \( k_i < K_{\text{max}} \) do**
4.   **foreach** agent \( i = 1, \ldots , N \) **do**
5.     \( \hat{\mathbf{a}}_i[k_i] \leftarrow \text{QP}(\mathbf{x}_i[k_i], \mathbf{a}_i[k_i - 1], \Pi) \)
6.     **if** QP feasible **then**
7.         \( \hat{\mathbf{x}}_i[k+1|k_i] \leftarrow \text{GetStates}(\mathbf{x}_i[k_i], \hat{\mathbf{a}}_i[k_i]) \)
8.         \( \Pi_i \leftarrow \hat{\mathbf{p}}_i[k+1|k_i] \)
9.         \( \mathbf{x}_i[k_i+1], \mathbf{a}_i[k_i] \leftarrow \hat{\mathbf{x}}_i[1|k_i], \hat{\mathbf{a}}_i[0|k_i] \)
10. \( \text{AtGoal} \leftarrow \text{CheckGoal}(\mathbf{p}[k_i], \mathbf{p}_f) \)
11. \( k_i \leftarrow k_i + 1 \)
12. **if** AtGoal **then**
13.     \( [\mathbf{p}, \mathbf{v}, \mathbf{a}] \leftarrow \text{ScaleTrajectory}(\mathbf{p}, \mathbf{v}, \mathbf{a}, \|a_{\text{max}}\|) \)
14.     \( [\mathbf{p}, \mathbf{v}, \mathbf{a}] \leftarrow \text{Interpolate}(\mathbf{p}, \mathbf{v}, \mathbf{a}, T_s) \)
15. \( \text{CheckCollisions}(\mathbf{p}) \)
16. **return** \( [\mathbf{p}, \mathbf{v}, \mathbf{a}] \)
Fig. 3. Probability of success comparison of DMPC against other approaches, in a fixed 4m³ volume. For every density considered, 50 different random test cases were generated within the feasible set.

that it is unavoidable), or it could be detected after the interpolation step. Solving with different values for \(\rho\) and/or \(\zeta\), or a tighter lower bound \(\varepsilon_{\text{max}}\), may lead to a solution.

3) Not reaching goal within \(K_{\text{max}}\) time steps: The lack of coordination between the agents may lead to oscillations or deadlocks, in which case the agents may not reach their goal location within the maximum allowed time. Higher values of \(\kappa\) may help in avoiding such situations.

Strategy (1) is actively used inside the QP function of Algorithm 1. The other two strategies are not part of the proposed method, but one could add an additional outer loop that checks for the failure of the algorithm and applies a repair strategy if required.

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 16GB of RAM, running at 3GHz. The agents were modelled based on the Crazyflie 2.0 platform, using \(r_{\text{min}} = 0.35\text{m}, \quad d_{\text{max}} = -d_{\text{min}} = 1\text{m/s}^2, \quad \text{and} \quad c = 2\) (to avoid downwash effects).

A. Comparison To Other Approaches

To test the capabilities of DMPC to compute transition trajectories, we compared its performance with two other state-of-the-art algorithms: centralized [4] and decoupled [7] SCP. Scenarios with random sets of initial and final positions were generated and solved using all three algorithms. The volume of the workspace was kept fixed at 4m³, and the number of agents ranged from 4 to 20. 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. If DMPC failed to solve, the arrival time was set to as the maximum allowed time to complete the transition, \(T_{\text{max}} = 20s\).

All three approaches shared the time step parameters \(h = 0.2s\) and \(T_s = 0.01s\), as well as the agent specifications. For DMPC we used a horizon length \(K = 15\), parameter \(\kappa = 2\) and a maximum relaxation of \(\varepsilon_{\text{max}} = 0.05\text{m}\). After solving the problem with each method, the solution was interpolated and analysed for possible collisions. For all the methods we allowed a collision tolerance of 5cm after interpolation, meaning that for success we required at least a distance greater than \(r_{\text{min}} = 0.3\text{m}\).

Fig. 3a shows the probability of success as the density of agents increases. The proposed DMPC algorithm was able to 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. 3b reveals the biggest advantage of our approach: a reduction of up to 97% in computation time with respect to centralized SCP and of 85% with decoupled SCP. The various design decisions to speed up the algorithm, such as including collision constraints only when needed, and only on one time step, have a noticeable impact and are the main reason of the scalability of the method. The significant runtime variance observed in the other two approaches is due to the test-by-test variance in arrival time, as seen in Fig. 3d. Note that even with the highly reduced runtimes, the tested implementation of DMPC was not taking advantage of its parallelizable nature.

To measure the optimality of the generated trajectories we analysed the sum of travelled distances by the agents, as highlighted in Fig. 3c. Our distributed approach produces longer paths on average, with respect to both the centralized and decoupled SCP. The suboptimality increases with the density of the environment, since the agents actively adjust their trajectories to avoid collisions, and oftentimes those adjustments lead to non-optimal paths towards their goals.

B. Failure Analysis

We previously analysed the performance of DMPC over increasingly dense environments, but with a relatively low number of agents. To evaluate the algorithm’s performance with a higher number of agents, we considered scenarios with a fixed workspace density of 1 agent/m³ and a wide range of number of agents. The results are summarized in Fig. 4. From the failure sources listed in Sec. IV-B, only collision failures were identified for this test case. Even though the density of agents was constant, the proposed DMPC algorithm had a higher failure rate as the number of agents increased. The inclusion of more decision-making agents exposes a limitation of our method, mainly due to the lack of explicit
coordinated between the agents. In particular, our collision avoidance strategy assumes that the neighboring agents are ‘passive’ and follow their predicted path, which is not true since they are also changing plans to avoid collisions. Despite this limitation, our algorithm remains effective for moderately large swarms with a 90% success rate with up to 100 agents.

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

The number of cores of a CPU constrain the maximum number of clusters we can deploy. In Fig. 5 we tested different numbers of clusters to solve a wide variety of transition scenarios. It was found that 8 clusters led to the best result for our computing hardware. In this case, each cluster ran separately on one of the eight cores of our CPU. This parallel strategy (8 clusters) reduced the computation time by more than 60% compared to using 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 and velocity estimates from an overhead motion capture system. The setpoints were tracked using an on-board position controller based on [20]. 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 constant position over the horizon).

We required \( r_{\min} = 0.25 \text{m} \) with \( \varepsilon_{\max} = 0.03 \text{m} \). The DMPC algorithm was able to find a solution for this case in 1.8 seconds. In Fig. 7a, the curves delimiting the gray area correspond to the minimum and maximum inter-agent distance in each step 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 unmodeled phenomena in our planning algorithm, such as imperfect trajectory tracking, time delays and aerodynamics. Also, since we were using an overhead motion capture system, the reflective markers sitting on top of the drones may be occluded while performing convoluted transitions. During these occlusion periods, the state estimation of our system relied on a constant acceleration profile to predict the current position, introducing biases in the position estimation. Taking all these phenomena 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 needs to 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
duced the versatility of the method to quickly generate intricate transition trajectories mid-flight.

As future work, we plan to investigate coordination paradigms that enable our algorithm to find a solution for even larger swarms, without compromising the scalability demonstrated in this work.

REFERENCES


