Time and Energy Optimized Trajectory Generation for Multi-Agent Constellation Changes

Paul Ladinig, Bernhard Rinner and Stephan Weiss

***Abstract*— Planning the simultaneous movement of multiple agents represents a challenging coordination problem, and ideally safety and efficiency are jointly addressed. This paper introduces a planning algorithm for fast and energy-efficient trajectories with reduced collision potential from a start to an end constellation. This new approach combines trajectory approximation based on model predictive control, collision avoidance with potential fields, and flight energy optimization with minimum snap trajectories. Our approach results in unprecedented transition times and success rates with less energy consumption, as shown in simulation and real experiments with 16 drones.**

I. Introduction and Related Work

Multi-robot systems are typically represented as a Multi-Agent System (MAS), and the coordination of the individual agents is a fundamental topic in MAS research. Coordination is thus essential for many multi-robot applications including logistics [1], [2], aerial monitoring [3], and surveillance [4]. Planning the agents' movement represents a challenging coordination problem where safety and efficiency should ideally be jointly addressed. This paper contributes to this specific problem and proposes a planning algorithm for fast, energy-efficient, and simultaneous trajectories with reduced collision potential from a start to an end constellation for large teams of agents. The innovation of our approach lies in the merging of i) trajectory approximation based on model predictive control, ii) collision avoidance with potential fields, and iii) flight energy optimization with minimum snap trajectories. As depicted in Figure 1, we extensively evaluate this approach in simulation and demonstrate the feasibility in experiments with constellation changes of 16 drones.

There is various related work on multi-agent constellation changes. Trajectories can be generated using Potential Field (PF) approaches [5], [6] where the computation effort scales well with the number of agents while keeping potential collisions low. However, they experience problems with deadlocks and, to the best of our knowledge, PF methods have not been used for time- or energy-optimized trajectory planning. Bio-inspired approaches [7], on the other hand, support problem-solving in non-convex environments. Examples include genetic algorithms [8] or particle swarm optimization [9], but their high computational complexity limits trajectory planning for small to medium-sized teams of agents. An alternative approach is to cast trajectory planning as an optimization problem and use established solvers to compute optimal solutions. For example, the Sequential Convex Programming (SCP) approach [10], [11] optimizes for trajectory length and/or control effort but also faces problems with computation time and potential collisions for a large number of agents. These issues are addressed in [12] by generating computationally fast and guaranteed feasible trajectories using grid- and optimization-based approaches. However, this method yields suboptimal solutions with respect to path lengths and thus transition times. Luis et al. [13] propose to use Model Predictive Control (MPC) for trajectory planning. They show that a high success rate with a moderate computational complexity as compared to

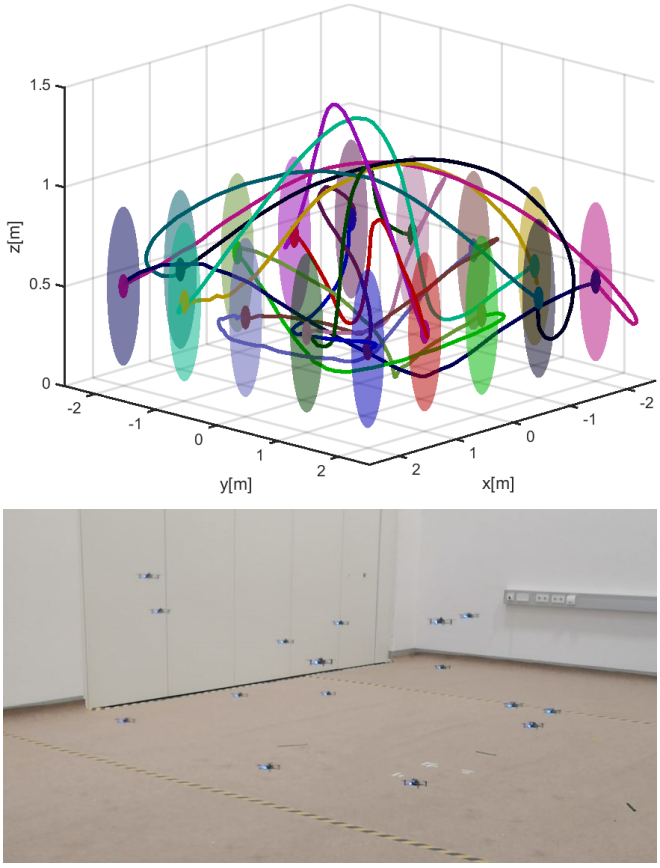


Fig. 1: Constellation changes of 16 agents in 3D space. Dots and ellipses represent the start and end constellation, solid lines the tracked trajectories from real flights (top). Crazyflie drones executing the trajectories in our dronehall (bottom).

All authors are with the University of Klagenfurt, Austria (dronehub Klagenfurt: uav.aau.at). Paul Ladinig and Stephan Weiss are with the Control of Networked Systems Group, and Bernhard Rinner is with the Institute of Networked and Embedded Systems. email: {firstname.lastname}@aau.at. The authors thank Michał Barciś for his support with the experiments.

This work was supported by the EU-H2020 project BUGWRIGHT2 (GA 871260) and the doctoral school KPK-NAV of the University of Klagenfurt.

Accepted February/2021 for ICRA 2021, DOI follows ASAP ©IEEE.

the previous methods is possible. Furthermore, this approach includes a dynamic model to generate trajectories specifically designed for their robotic platform. Mellinger et al. [14] exploit the agent's dynamics by minimizing the snap of a trajectory resulting in a trajectory generation specifically designed for aggressive drone flights. However, this method is restricted to individual trajectories and is not adapted to collective planning.

Our approach to collision-aware, fast, yet energy-efficient simultaneous trajectory generation for large teams of agents leverages the trajectory approximation in [13] with improved performance through our PF extension and the energy optimization in [14] to achieve unprecedented transition times and success rate with less energy. The success rate is defined as the ratio between collision-free and overall examined constellation changes. We use MPC to compute initial trajectories for coarse constellation changes and apply PF to significantly reduce collisions which would occur due to discretization in [13]. We then specify elliptically shaped spatial constraints around these initial trajectories and apply minimum snap optimization for computing the final trajectories within these volume constraints around the initial trajectories. Our approach results in simultaneous, fast, and energy-efficient drone movements from a start to an end constellation. The contribution includes i) a novel framework using MPC, PF-based collision avoidance, and energy-optimized trajectory generation, ii) a simulation study which shows a significant reduction of the energy consumption as compared to previous work while at the same time achieving faster constellation changes, and iii) experiments with real drones demonstrating the feasibility of our approach and validating our simulation results.

II. Problem Statement

Figure 1 depicts the problem under study. For a given team of N agents, we search for trajectories providing simultaneous, collision-free, fast, and energy-efficient transitions from a start to an end constellation. We adopt the notation of [13] for the formal problem specification.

A. Agent Model

A group of N agents is located at positions $\mathbf{p}_i$ for each agent $i \in 1,\ldots,N$ in the Cartesian coordinate system where a constellation is defined as static and collision-free positions of all N agents. The position of agent i at timestep k is denoted by $\mathbf{p}_i[k] \in \mathbb{R}^3$. Each agent's dynamics are described by their velocity $\mathbf{v}_i[k]$ and acceleration $\mathbf{a}_i[k]$. These dynamics are modelled as follows

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

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

using a discretization step h and obeying the physical limits:

$$\begin{aligned} \mathbf{p}_{\min} &\leq \mathbf{p}_i[k] \leq \mathbf{p}_{\max} \\ \mathbf{v}_{\min} &\leq \mathbf{v}_i[k] \leq \mathbf{v}_{\max} \\ \mathbf{a}_{\min} &\leq \mathbf{a}_i[k] \leq \mathbf{a}_{\max}. \end{aligned} \tag{3}$$

The agents' workspace is limited to a given flight volume specified by a lower bound $\mathbf{p}_{\min}$ and an upper bound $\mathbf{p}_{\max}$. The agents' velocity and acceleration are bounded as well. We choose the bounds $v_{\max} = -v_{\min} = 5\,\mathrm{m/s}$ and $a_{\max} = -a_{\min} = 1\,\mathrm{m/s^2}$ in each axis. Furthermore, we assign a start position $\mathbf{p}_{\text{start}}$ and an end position $\mathbf{p}_{\text{end}}$ to each agent.

B. Collision Constraints

We adopt the collision constraints from [13] for maintaining a minimum distance among all agents. This collision constraint is formulated as

$$\left\| \Theta^{-1}\left(\mathbf{p}_i[k] - \mathbf{p}_j[k]\right) \right\|_2 \geq r_{\min} + \varepsilon_{ij}, \quad \forall k,i,j \,|\, i \neq j \tag{4}$$

where the scaling matrix $\Theta = \mathrm{diag}(a,b,c)$ models an ellipsoid around each agent. The parameters are set to $a = b = 1$ and $c = 2$ to increase the vertical distance between agents to avoid downwash interference.

During trajectory generation it may happen that a fixed spatial constraint based on $r_{\min}$ cannot be met. Therefore, the collision constraint in Eq. 4 is relaxed by ε_{ij} with $-\varepsilon_{\max} \leq \varepsilon_{ij} \leq 0$ to avoid infeasible trajectories. ε_{ij} is initially set to zero and is decreased if necessary.

C. Constellation Change

Based on the agent model in Eqs. 1 and 2 we search for trajectories $\mathbf{r}_T$ that transfer agents from their start positions $\mathbf{p}_{\text{start}}$ to their end positions $\mathbf{p}_{\text{end}}$ and fulfill the physical limit constraints in Eq. 3 and the collision constraint in Eq. 4. The objective of the constellation change is to minimize the transition time for all agents and to minimize the agents' energy consumption during the transition.

III. Optimized Trajectory Generation

Our trajectory generation approach consists of two parts. First, we calculate initial trajectories between the start and end constellation by extending the MPC method [13] with PF to further improve the success rate. Second, we optimize these initial trajectories to better exploit the agent's dynamics by fitting snap-minimized trajectories [14] based on dynamic and volumetric constraints that are derived from the MPC solution. In the following, we mathematically describe our approach and present the pseudo code in Algorithms 1-3.

A. Model Predictive Control

We define the optimization function as quadratic program (QP) based on the agent model in Eqs. 1 and 2:

$$\begin{aligned} \min_{\mathbf{U}_i} \quad & \mathbf{U}_i^T \mathbf{H} \mathbf{U}_i + \mathbf{f}^T \mathbf{U}_i, \ \forall i \\ \text{s.t.} \quad & \mathbf{A}\mathbf{U}_i \leq \mathbf{b} \end{aligned} \tag{5}$$

The cost function described by $\mathbf{H}$ and $\mathbf{f}$ governs the transition of each agent. It penalizes the spatial displacement of agents to their end position, their absolute control effort and their control variation during their transition. Thus, agents are smoothly directed to the end constellation. The physical limit constraint Eq. 3 and the collision avoidance constraint Eq. 4 are incorporated using the matrices $\mathbf{A}$ and $\mathbf{b}$.

The trajectories are generated by iteratively solving the QP and obtaining the output $\mathbf{U}$ which provides the predicted accelerations over a predefined time horizon with length K. All predicted accelerations are propagated using Eqs. 1 and 2 to obtain the predicted future positions which are shared amongst other agents. Hence, collisions at iteration k can be avoided based on the predicted future positions from iteration $k-1$. In each iteration, the first predicted positions and velocities are stored as the latest state of the trajectories. This is repeated until either the end constellation is reached or no feasible solution can be found.

B. Potential Fields

To improve the success rate and computation time, [13] uses so-called soft constraints for trajectory generation. However, these constraints can lead to partial violations of the collision constraint and subsequently to collisions. Therefore we suggest avoiding these collisions by using PF. To avoid collisions at a propagation step in Eqs. 1 and 2 we simply assess whether the collision constraint in Eq. 4 holds true for the newly propagated positions $\mathbf{p}_i[k+1]$ of all agents i. If a collision is detected, we execute the following PF method.

Inspired by [15], we define a force $\mathbf{F}_{\text{net}}$[1] as:

$$\mathbf{F}_{\text{net}}\left(\mathbf{p}[k],i\right)=\mathbf{F}_{\text{att}}\left(\mathbf{p}_i[k]\right)-\frac{1}{N}\sum_{\text{j}=1,\text{j}\neq\text{i}}^{N}\mathbf{F}_{\text{rep}}\left(\mathbf{p}_i[k],\mathbf{p}_j[k]\right) \quad (6)$$

The force $\mathbf{F}_{\text{net}}$ determines the position change at time step k. Here, the attractive force $\mathbf{F}_{\text{att}}$ directs agents to their end positions, and the repulsive force $\mathbf{F}_{\text{rep}}$ prevents collisions with other agents:

$$\begin{aligned}\mathbf{F}_{\text{att}}\left(\mathbf{p}_i[k]\right)&=\frac{\mathbf{p}_{i_{\text{end}}}-\mathbf{p}_i[k]}{\left\|\mathbf{p}_{i_{\text{end}}}-\mathbf{p}_i[k]\right\|_2}\\ \mathbf{F}_{\text{rep}}\left(\mathbf{p}_i[k],\mathbf{p}_j[k]\right)&=\frac{\mathbf{p}_j[k]-\mathbf{p}_i[k]}{\left\|\mathbf{p}_j[k]-\mathbf{p}_i[k]-r_{\min}-\varepsilon_{\max}\right\|_2^2}\end{aligned} \quad (7)$$

Note that we have to substitute $\mathbf{p}_j[k]$ by $\mathbf{p}_j[k+1]$ if the position update at time k is calculated for agent j before agent i. Using the PF method in a collision scenario changes the position and velocity propagation into

$$\begin{bmatrix}\mathbf{p}_i[k+1]\\ \mathbf{v}_i[k+1]\end{bmatrix}=\begin{bmatrix}\mathbf{p}_i[k]\\ -\mathbf{v}_i[k]\end{bmatrix}+\begin{bmatrix}\mathbf{F}_{\text{net}}\left(\mathbf{p}[k],i\right)\\ 2\mathbf{F}_{\text{net}}\left(\mathbf{p}[k],i\right)\end{bmatrix}. \quad (8)$$

The velocity propagation is derived by rewriting Eq. 1 as $\mathbf{a}_i[k]=2\left(\mathbf{p}_i[k+1]-\mathbf{p}_i[k]-\mathbf{v}_i[k]\right)$ using $h=1$ and inserting it in Eq. 2. Furthermore, we choose a maximum force value $\text{F}_{\max}$ in order to limit the position update $\mathbf{p}_i[k+1]$. The resulting force $\mathbf{F}_{\text{net}}$ is scaled such that the inequality $\|\mathbf{F}_{\text{net}}\|_2\leq \text{F}_{\max}$ holds true.

The state update with Eq. 8 essentially replaces Eqs. 1 and 2 in case of detected collisions. Afterward, we share the existing $K-1$ predicted states with the other agents and continue computing the trajectory using the MPC algorithm.

[1] In this formulation, the forces $\mathbf{F}_{\text{net}}$, $\mathbf{F}_{\text{att}}$ and $\mathbf{F}_{\text{rep}}$ correspond to spatial displacements and not physical forces.

C. Minimum Snap Trajectories

Once the initial trajectory generation using MPC and PF is completed, we continue with its refinement. We first explain the generation of trajectories using snap minimization and then describe how to generate snap-minimized trajectories within the dynamic and volumetric constraints derived from the initial trajectories.

As proposed by [14], a trajectory $\mathbf{r}_{i_T}(t)=\left[x_{i_T}(t),y_{i_T}(t),z_{i_T}(t)\right]^T$ can be defined as m piecewise polynomial functions of order n.

$$\mathbf{r}_{i_T}(t)=\begin{cases}\sum_{w=0}^{n}r_{i_T w1}t^w & t_0\leq t<t_1\\ \sum_{w=0}^{n}r_{i_T w2}t^w & t_1\leq t<t_2\\ \vdots & \\ \sum_{w=0}^{n}r_{i_T wm}t^w & t_{m-1}\leq t\leq t_m\end{cases} \quad (9)$$

Here, m represents the number of all polynomials within a trajectory. We choose $n=7$ to generate smooth trajectories down to snap. The polynomials in $\mathbf{r}_T(t)$ are calculated by solving the optimization function

$$\begin{aligned}f_{i_T}\;=\;&\min\int_{t_0}^{t_m}\left\|\frac{d^{k_r}\mathbf{r}_{i_T}}{dt^{k_r}}\right\|_2^2 dt\\ &\text{s.t.}\;\left.\frac{d^p\mathbf{r}_{i_T}}{dt^p}\right|_{t=t_j}=0,\;j=0,m;\;p=1,\ldots,k_r\end{aligned} \quad (10)$$

as QP. For quadrotor drones, [14] proposes to minimize the integral of the square of the norm of the snap and therefore $k_r=4$. Note, that this parameter depends on the physical platform and can be adapted to fit other robotic hardware. The following constraints are defined for the optimization:

- All derivatives are set to zero at the start and the end of a trajectory.
- Acceleration and velocity limits are defined according to Eq. 3.
- The third derivative of $\mathbf{r}_{i_T}(t)$ must be differentiable.
- Each trajectory must be located within a spatio-temporal volume derived from the MPC-PF trajectories.

The use of snap-minimized trajectories provides two advantages: First because the actuation of quadrotors is algebraically related to the snap, the exploitation of their dynamics is precisely possible. This allows for aggressive flight maneuvers. Second, minimizing the snap directly results in energy minimization (cf. [16]).

D. Volume Constraints

We finally describe the trajectory generation by exploiting volume constraints which can be specified as (inspired by [14])

$$\mathbf{p}_i[k]-\Delta\mathbf{p}_{c_i}[k]\leq\left.\mathbf{r}_{i_T}(t)\right|_{t=hk}\leq\mathbf{p}_i[k]+\Delta\mathbf{p}_{c_i}[k]. \quad (11)$$

At the discrete time step k and the continuous time $t=hk$ we constrain the trajectory $\mathbf{r}_{i_T}(t)$ to be located within a volume $2\Delta\mathbf{p}_{c_i}[k]$ around the position $\mathbf{p}_i[k]$. In order to generate the volume for agent i at time step k, we first calculate the distance $r_{\text{n}}[k]$ to the nearest neighbor j_{n}

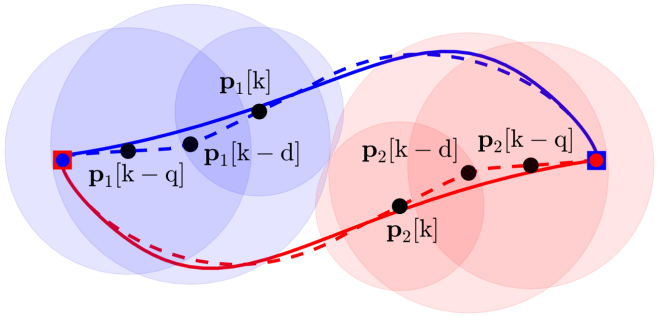


Fig. 2: 2D visualization of the volume constraints i.e. $r_n[k]$ at selected time steps (translucent circles) and initial (dashed lines) and snap-minimized (solid lines) trajectories. Black circles represent the volume's center at different time steps. These volume constraints result in energy-efficient evasive actions during snap minimization.

$$r_{\mathrm{n}}[k] = \left\| \Theta^{-1} \left(\mathbf{p}_i[k] - \mathbf{p}_{j_{\mathrm{n}}}[k] \right) \right\|_2 . \tag{12}$$

We then obtain $\Delta\mathbf{p}_{c_i}$ using r_{n} and subtracting the smallest allowed distance among the agents to provide a safety distance

$$\Delta\mathbf{p}_{c_i}[k] = \frac{r_{\mathrm{n}}[k] - r_{\min} + \varepsilon_{\max}}{2} \mathbf{1}_{3\times 1}. \tag{13}$$

The division by two ensures that the volume is spanned around the center $\mathbf{p}_i[k]$. Finally, the optimization function is achieved by incorporating Eq. 11 into Eq. 10. Note that the trajectories must lie within the flight volumes specified by Eq. 3.

E. Algorithm

Algorithm 1 describes our *Snap-Optimized MPC approach with Potential-Field extension (SOMPF)*. The algorithm starts with initializing the trajectories (line 3) as straight lines between the start and end constellation. The predicted future positions are stored in Π, and each agent's state is stored in $\mathbf{x} = \left[\mathbf{p}^T\ \mathbf{v}^T\right]^T$. During the while loop (lines 5-20), the trajectories are iteratively generated until all end positions are reached or a maximum number of iterations $k_{\max}$ is reached. Here, the notation $\hat{(\cdot)}\,[k|k_t]$ refers to the prediction of $(\cdot)\,[k+k_t]$ based on the available information at time k_t. The parameter $k \in \{0,\ldots,K-1\}$ denotes the prediction steps of the prediction horizon with length K.

The QP in Eq. 5 is solved based on the current state, the previous acceleration and the future predictions (line 7). It outputs the predicted accelerations over the time horizon. If the QP is feasible, the accelerations are propagated using Eqs. 1 and 2 and the result is stored in Π (lines 9-10). If a collision is detected, we apply the PF method (Algorithm 2) to calculate the future state (line 12). Otherwise, the predicted state is stored (line 14). Once the end constellation is reached (line 18), all trajectories are scaled such that the dynamic limits in Eq. 3 are reached but not exceeded (line 22). This decreases the transition times. The scaled trajectories are then refined with Algorithm 3 (line 23). Finally, if all trajectories are collision-free, the trajectory is returned.

Algorithm 2 executes the state update using the PF method. This algorithm receives the current state $\mathbf{x}[k_t]$ of all agents and, if already computed, the available states $\mathbf{x}[k_t+1]$ of the next time step. The states at time step k_t+1 are needed considering the already successful state generation of other agents.

If agent i encounters a collision with another agent, the repulsive force $\mathbf{F}_{i_{\mathrm{net}}}$ is computed based on Eq. 6 with the position information at time step k_t or k_t+1, respectively (line 3). The resulting force $\mathbf{F}_{i_{\mathrm{net}}}$ is limited (lines 4-6) and then used to calculate the next state $\mathbf{x}_i[k_t+1]$ with Eq. 8 (line 7). Finally, the acceleration is computed by the difference of the velocity $\mathbf{v}_i$ at time step k_t and k_t+1 (line 8), and the new state and acceleration are returned (line 9).

Algorithm 1 SOMPF trajectory generation

1: **Input:** Start $\mathbf{p}_{\mathrm{start}}$ and end position $\mathbf{p}_{\mathrm{end}}$
2: **Output:** Position trajectory $\mathbf{r}(t)$
3: $\left[\Pi, \mathbf{x}[0]\right] \leftarrow \mathrm{InitAllPredicitions}\left(\mathbf{p}_{\mathrm{start}}, \mathbf{p}_{\mathrm{end}}\right)$
4: $k_t \leftarrow 0, \mathrm{AtGoal} \leftarrow \mathrm{false}$
5: **while** *not AtGoal and* $k_t < k_{\max}$ **do**
6: **for** agent $i = 1, ..., N$ **do**
7: $\hat{\mathbf{a}}_i[k|k_t] \leftarrow \mathrm{Build\&SolveQP}\left(\mathbf{x}_i[k_t], \mathbf{a}_i[k_t-1], \Pi\right)$
8: **if** *QP feasible* **then**
9: $\hat{\mathbf{x}}_i[k+1|k_t] \leftarrow \mathrm{GetStates}\left(\mathbf{x}_i[k_t], \hat{\mathbf{a}}_i[k|k_t]\right)$
10: $\Pi_i \leftarrow \hat{\mathbf{p}}_i[k+1|k_t]$
11: **if** $\mathrm{CollisionCheck}\left(\hat{\mathbf{x}}_i[1|k_t]\right) == true$ **then**
12: $\mathbf{x}_i[k_t+1], \mathbf{a}_i[k_t] \leftarrow \mathrm{PF}\left(\mathbf{x}[k_t], \mathbf{x}[k_t+1]\right)$
13: **else**
14: $\mathbf{x}_i[k_t+1], \mathbf{a}_i[k_t] \leftarrow \hat{\mathbf{x}}_i[1|k_t], \hat{\mathbf{a}}_i[0|k_t]$
15: **end if**
16: **end if**
17: **end for**
18: $\mathrm{AtGoal} \leftarrow \mathrm{CheckGoal}\left(\mathbf{p}_i[k_t+1], \mathbf{p}_{\mathrm{end}}\right)$
19: $k_t \leftarrow k_t + 1$
20: **end while**
21: **if** *AtGoal* **then**
22: $[\mathbf{p}, \mathbf{v}, \mathbf{a}, h_{\mathrm{scaled}}] \leftarrow \mathrm{ScaleMPC}\left(\mathbf{p}, \mathbf{v}, \mathbf{a}, \|\mathbf{v}_{\max}\|, \|\mathbf{a}_{\max}\|\right)$
23: $\mathbf{r}(t), t_m \leftarrow \mathrm{MinSnap}\left(\mathbf{p}, \mathbf{v}, \mathbf{a}, \mathbf{p}_{\min}, \mathbf{p}_{\max}, h_{\mathrm{scaled}}, h\right)$
24: **if** $\mathrm{CollisionCheck}\left(\mathbf{r}(t)\right) == true$ **then**
25: $\mathbf{r}(t) \leftarrow \emptyset$
26: **end if**
27: **end if**
28: **return** $\mathbf{r}(t)$

Algorithm 2 Potential field propagation

1: **Input:** Current positions $\mathbf{p}$ and available accelerations $\mathbf{a}[k_t]$
2: **Output:** New state $\mathbf{x}_i[k_t+1]$ and acceleration $\mathbf{a}_i[k_t]$
3: $\mathbf{F}_{\mathrm{net}} \leftarrow \mathrm{CalcForce}\left(\mathbf{p}, i\right)$
4: **if** $\|\mathbf{F}_{\mathrm{net}}\| > F_{\max}$ **then**
5: $\mathbf{F}_{\mathrm{net}} \leftarrow \mathrm{LimitToMaxValue}\left(\mathbf{F}_{\mathrm{net}}, F_{\max}\right)$
6: **end if**
7: $\mathbf{x}_i[k_t+1] \leftarrow \mathrm{GetState}\left(\mathbf{x}_i[k_t], \hat{\mathbf{a}}_i[0|k_t], \mathbf{F}_{\mathrm{net}}\right)$
8: $\mathbf{a}_i[k_t] \leftarrow \mathbf{v}_i[k_t+1] - \mathbf{v}_i[k_t]$
9: **return** $\mathbf{x}_i[k_t+1], \mathbf{a}_i[k_t]$

Algorithm 3 optimizes the trajectories generated by MPC and PF with respect to their snap. It starts with the generation of the bounding volumes $\Delta\mathbf{p}_c$ for each agent at all time steps (line 3) and then iteratively refines the trajectories lowering the transition times (lines 5-9). The initial transition time is given by the MPC solution. In each iteration, the scaled time step h_{scaled} from the previous iteration is used for the snap optimization by solving Eq. 10 (line 7). The resulting trajectories are then scaled to the dynamic limits for obtaining faster transition times (line 8). The iterations are repeated until no further time reduction is achieved or the iteration threshold $l_{\max}$ is reached.

Algorithm 3 Calculate minimum snap trajectory

1: **Input:** Positions $\mathbf{p}$, velocity $\mathbf{v}$ and acceleration $\mathbf{a}$ trajectories; position boundaries $\mathbf{p}_{\min}$, $\mathbf{p}_{\max}$; original h and scaled h_{scaled} time step
2: **Output:** Position trajectory $\mathbf{r}(t)$
3: $\Delta\mathbf{p}_c \leftarrow \text{GenerateVolumina}(\mathbf{p}, \mathbf{p}_{\min}, \mathbf{p}_{\max})$
4: $l_s \leftarrow 0$
5: **while** $(h_{\text{scaled}} < h)$ *and* $(l_s \leq l_{\max})$ **do**
6: $\quad h \leftarrow h_{\text{scaled}}$
7: $\quad \mathbf{r}(t) \leftarrow \text{FitMinSnapTrajectories}(\mathbf{p}, \Delta\mathbf{p}_c, h)$
8: $\quad [\mathbf{r}(t), h_{\text{scaled}}] \leftarrow \text{ScaleSOMPF}(\mathbf{r}(t), \|\mathbf{v}_{\max}\|, \|\mathbf{a}_{\max}\|, h)$
9: $\quad l_s \leftarrow l_s + 1$
10: **end while**
11: **return** $\mathbf{r}(t)$

IV. Evaluation

We evaluate the performance of our SOMPF trajectory generation method compared to the MPC approach with a simulation study. We have implemented Algorithms 1-3 in MATLAB R2019b and executed all simulations on an Intel Core PC with 8 GB RAM running at 3.40 GHz. The computation times are measured with the cputime function, and the QPs are solved with the quadprog function. For a fair comparison with [13], we used a similar parameter setting, i.e., $h = 0.2$, $r_{\min} = 0.35\,m$, $\varepsilon_{\max} = 0.05\,m$ and $k_{\max} = 1000$. Furthermore, we set $\mathrm{F}_{\max} = 0.02\,\mathrm{m}$ so that the position update is kept small during the position update by the PF method.

A. Density-Dependent Performance

We run the MPC of [13] and our SOMPF algorithm in a fixed volume of $4\,m^3$ and vary the team size from 4 to 24 in multiple of 4 agents, thus varying the agent density from 1 to 6 agents per m^3. Fig. 3 compares the following performance parameters of our simulation: the transition time, i.e., the mean movement time from a start to an end point, the mean computation time, the normalized trajectory length, and the optimization success, i.e., the ratio between the constellation changes with feasible optimization and all considered constellation changes. Note that the update of h_{scaled} in SOMPF with $I_{max} = 4$ (green plots) is slightly modified, i.e., h_{scaled} is lowered by 10% at each iteration.

Fig. 3a shows that the transition times of the MPC method is reduced for all densities using the proposed method.

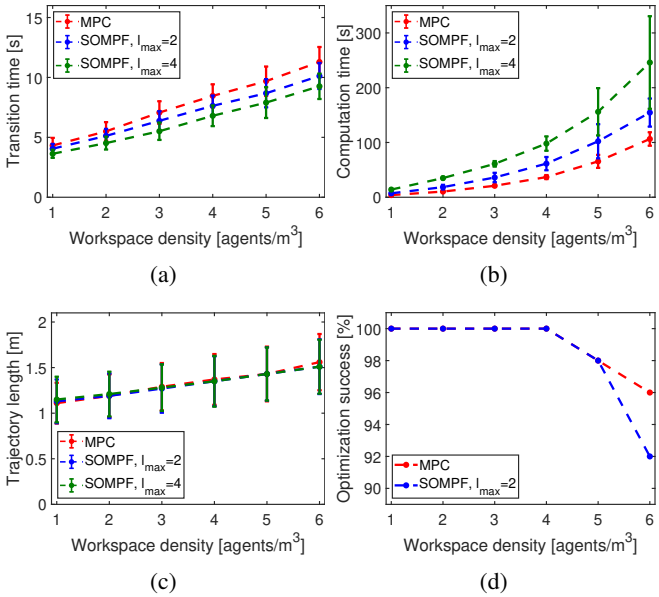
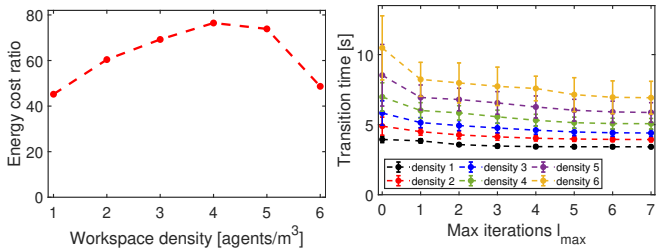


Fig. 3: Comparison of the original MPC algorithm with our SOMPF method. For each workspace density we perform 100 simulations and determine (a) the transition times, (b) the computation times, (c) the trajectory lengths, and (d) the total optimization success. For graphs (a)-(c), the mean and standard deviation values are plotted, and SOMPF is executed with at most 2 or 4 optimization cycles, respectively. All generated trajectories are collision-free.

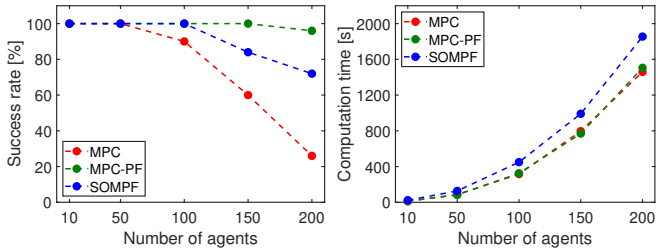
The transition time is reduced by up to 18% at density 6. Fig. 3b depicts that the improvement of the transition times comes with increased computation time. However, this figure indicates that MPC and SOMPF lie in the same order of computational complexity. The computation time increases with the maximum numbers of optimization cycles $l_{\max}$. As Fig. 4b suggests, $l_{\max} > 2$ has little performance gain in practice. Furthermore, our method yields very similar trajectory lengths and optimization successes compared to the MPC method, see Figs. 3c and 3d. The small decrease in the optimization success of 4% at a density of 6 is due to infeasible constraints in Eq. 10 which result from the MPC solution. Collisions do not appear throughout all densities.

Fig. 4a depicts the energy cost ratio of the trajectories generated with MPC and SOMPF. We define the energy cost ratio as the quotient of the average trajectory costs of MPC $(\overline{f_T}\,\big|\,\Delta\mathbf{p}_c = \mathbf{0})$ and the average costs of SOMPF $(\overline{f_T})$ with $l_{\max} = 2$ (cf. Eq. 10). By setting $\Delta\mathbf{p}_c = \mathbf{0}$, we force the trajectories to lie exactly on the given points of the MPC approach. We show that SOMPF reduces the costs by at least a factor of 40 compared to the MPC approach. Furthermore, we notice that the cost saving has its peak at a density of 4. Our explanation for this is as follows: At low densities, the task of generating trajectories is less complex due to the reduced risk of collision. This allows MPC to generate smoother and hence energy-efficient trajectories. At high densities, the volume constraints are tighter on average, and the trajectories generated by Eq. 10 are therefore spatially closer to the MPC solution. Fig. 4b shows the improvement of the transition time as a function of the maximum number



(a) (b)

Fig. 4: (a) Energy cost ratio of trajectories generated by MPC and SOMPF. (b) Transition times for different densities and maximum number of optimization cycles $l_{\max}$. Transition times can be hardly improved with more than 4 cycles.



(a) (b)

Fig. 5: Success rate and computation time dependent on the number of agents with fixed density of 1 agent/m^3. SOMPF uses at most 2 optimization cycles ($l_{max} = 2$).

of optimization cycles.

B. Success Rate and Computation Time

Fig. 5 plots the success rate and computation as a function of the team size. In these simulation scenarios, we keep the density fixed at 1 agent/m^3 and vary the number of agents from 10 to 200. We compare three trajectory generation approaches: the MPC approach (red line), the MPC approach including our PF extension (green line), and our SOMPF approach (MPC with PF and minimum snap refinement, blue line). Because of the collision susceptibility of the MPC method with higher numbers of agents, our PF based extension reaches a noticeably improved success rate (cf. Fig 5a). Similarly, SOMPF outperforms the MPC success rate but is worse than MPC-PF. Although our algorithms show improved success rates, we cannot guarantee feasibility. Infeasible trajectory generations happen when individual optimization problems in Eqs. 5 and 10 cannot be solved within the given constraints. Furthermore, trajectories that cannot be generated within $k_{\max}$ iterations are also declared infeasible. Fig. 5b compares computation time of the three approaches. The PF extension does hardly increase the computation time of MPC. On the other hand, SOMPF shows a slight increase in computation time which follows a similar trend.

V. Practical Experiments

This section summarizes results from our experiments with Crazyflie 2.0 drone platforms. All experiments are conducted in our dronehall, and an Optitrack motion capture system is used for precise position measurements of the drones. We apply the provided control system [17] for executing the computed trajectories. We have implemented a constellation change of 16 drones with $r_{\min} = 0.4m$ arranged in the quadratic shape of $4m \times 4m$ at a fixed altitude.

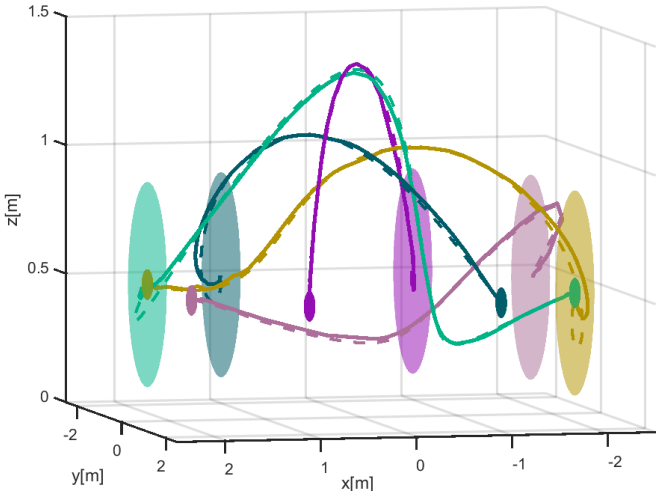


Fig. 6: Visualization of 5 selected simulated (dashed line) and experimentally measured (solid Line) trajectories. Dots denote start positions and ellipses show end positions.

Fig. 6 compares selected simulated and measured trajectories. The average displacement error between simulation and real experiment is as small as $7.43\,cm$ with a standard deviation of $0.02\,cm$, whereas the maximum error is $20.27\,cm$. The constellation change takes $8.97\,s$ in simulation and $8.79\,s$ in practice and achieves an energy cost ratio of 240.03. The computation time is $107.44\,s$.

VI. Conclusion

We present an energy-efficient trajectory generation for fast constellation changes in multi-agent systems. Our Snap-Optimized Model-predictive control approach with Potential-Field extension (SOMPF) achieves, compared to state of the art, i) faster constellation transitions (up to 18% faster), ii) at lower energy costs (at least 40 times lower cost terms), and iii) improved success rate for large swarms. Our approach requires slightly larger computation times while residing in the same order of computational complexity.

The improvement in the success rate for larger swarms is due to our extension of including a PF approach upon collision detection in the original MPC approach. The drastic energy reduction arises from our approach to include a snap-minimization step with volume constraints to best leverage the vehicle dynamics while at the same time maintaining minimal distances to other agents.

The presented simulations yield statistical relevant evaluation data to do fair comparisons against state of the art. In addition, the real experiments with 16 real crazyflie platforms changing constellations in our dronehall demonstrate the feasibility of our concept in real-world setups.

REFERENCES

[1] K. Gao, J. Xin, H. Cheng, D. Liu, and J. Li, "Multi-mobile Robot Autonomous Navigation System for Intelligent Logistics," in *Proceedings of the Chinese Automation Congress*, 2018, pp. 2603–2609.

[2] K. Kuru, D. Ansell, W. Khan, and H. Yetgin, "Analysis and Optimization of Unmanned Aerial Vehicle Swarms in Logistics: An Intelligent Delivery Platform," *IEEE Access*, vol. 7, pp. 15 804–15 831, 2019.

[3] V. Mersheeva and G. Friedrich, "Multi-UAV Monitoring with Priorities and Limited Energy Resources," in *Proceedings of the Twenty-Fifth International Conference on International Conference on Automated Planning and Scheduling*, 2015, pp. 347–355.

[4] J. Scherer and B. Rinner, "Multi-uav surveillance with minimum information idleness and latency constraints," *IEEE Robotics and Automation Letters*, vol. 5, pp. 4812–4819, 2020.

[5] S. Ge and Y. Cui, "Dynamic Motion Planning for Mobile Robots Using Potential Field Method," *Autonomous Robots*, vol. 13, no. 3, pp. 207–222, 2002.

[6] Y.-b. Chen, G.-c. Luo, Y.-s. Mei, J.-q. Yu, and X.-l. Su, "UAV path planning using artificial potential field method updated by optimal control theory," *International Journal of Systems Science*, vol. 47, no. 6, pp. 1407–1420, 2016.

[7] L. Yang, J. Qi, D. Song, J. Xiao, J. Han, and Y. Xia, "Survey of Robot 3D Path Planning Algorithms," *Journal of Control Science and Engineering*, vol. 2016, 2016.

[8] A. Tuncer and M. Yildirim, "Dynamic path planning of mobile robots with improved genetic algorithm," *Computers & Electrical Engineering*, vol. 38, no. 6, pp. 1564–1572, 2012.

[9] J. Kennedy and R. Eberhart, "Particle Swarm Optimization," in *Proceedings of IEEE International Conference on Neural Networks*, vol. 4, 1995, pp. 1942–1948.

[10] F. Augugliaro, A. P. Schoellig, and R. D'Andrea, "Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 1917–1922.

[11] Y. Chen, M. Cutler, and J. How, "Decoupled Multiagent Path Planning via Incremental Sequential Convex Programming," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2015, pp. 5954–5961.

[12] J. Park, J. Kim, I. Jang, and J. Kim, "Efficient multi-agent trajectory planning with feasibility guarantee using relative bernstein polynomial," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2020, pp. 434–440.

[13] C. Luis and A. Schoellig, "Trajectory Generation for Multiagent Point-To-Point Transitions via Distributed Model Predictive Control," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 375–382, 2019.

[14] D. Mellinger and V. Kumar, "Minimum Snap Trajectory Generation and Control for Quadrotors," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2011, pp. 2520–2525.

[15] K. P. O'Keeffe, H. Hong, and S. H. Strogatz, "Oscillators that sync and swarm," *Nature communications*, vol. 8, no. 1, pp. 1–13, 2017.

[16] N. Kreciglowa, K. Karydis, and V. Kumar, "Energy Efficiency of Trajectory Generation Methods for Stop-and-Go Aerial Robot Navigation," in *Proceedings of the International Conference on Unmanned Aircraft Systems*, 2017, pp. 656–662.

[17] J. A. Preiss, W. Hönig, G. S. Sukhatme, and N. Ayanian, "Crazyswarm: A large nano-quadcopter swarm," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 3299–3304, software available at https://github.com/USC-ACTLab/crazyswarm.