Distributed Trajectory Generation for Multiagent Systems

by

Carlos E. Luis

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science
University of Toronto Institute for Aerospace Studies
University of Toronto

Abstract

Distributed Trajectory Generation for Multiagent Systems

Carlos E. Luis
Master of Applied Science
University of Toronto Institute for Aerospace Studies
University of Toronto
2019

Collision-free trajectory generation is a fundamental functionality of multiagent systems. As the number of agents increases, scalable algorithms are required to efficiently compute motion plans. In this thesis we developed a multiagent trajectory generation framework based on distributed model predictive control (DMPC). First, we introduced *on-demand collision avoidance* to solve the planning problem completely offline. We then extended the approach to online generation of the trajectories, and presented an event-triggered replanning strategy. The framework is validated through a wide variety of simulation tests and experiments with a swarm of up to 25 quadrotors flying in an indoor environment.

Acknowledgements

Looking back at the past two years, I am left with a deep sense of gratitude towards many people that supported me throughout this journey. I would like to start with my biggest supporters: thank you Mom for always being there for me, either to give me life advice or the ingredients of your famous bolognese sauce. Thank you Dad for teaching me how to be an adult and fend for myself. Thank you Adri for the laughs and the "big sis" advice. And special thanks to María, for your constant love and support. Gracias por tanto, los amo.

I want to express my gratitude to my supervisor, Professor Angela Schoellig. First, for seeing the potential in me and accepting me as her student. Second, for allowing me to explore my own ideas and push me in the right direction when needed. Lastly, for all the advise and support on finding my career path. The biggest lesson Angela taught me was to never lose sight of the big picture, and push the boundaries of what is currently possible. Needless to say, the time spent in Angela's lab has been the most academically enriching experience of my life. Everything was well worth the effort.

The Dynamic Systems Lab provided a friendly environment to do my research and develop this thesis. I am glad I met such an incredibly talented group of roboticists from all over the world. Beyond academic discussions, I was able to learn a lot from the different cultural backgrounds of my labmates. Thank you all: Karime, Chris, SiQi, Melissa, Adam, Bhavit, Sep, Michael, Mario, Wenda, Ke, Jeremy and Keenan. I will miss the interesting lunch-time conversations.

I would like to acknowledge the ever-growing robotics community at the University of Toronto, specifically at UTIAS. Special mention goes to the members of STARS lab and ASRL, with whom I share my love for this field. I would also like to thank Professor Hugh Liu, who agreed to review my thesis. I hope to read amazing work being published by the UTIAS robotics community.

"The true lover of knowledge naturally strives for truth, and is not content with common opinion, but soars with undimmed and unwearied passion till he grasps the essential nature of things."

– Plato

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background & Motivation

Current robotic applications such as manufacturing and surveillance benefit from the use of coordinated multiple robots. There are missions that would be too time-consuming or even impossible for a single robot to accomplish. Cooperation is regarded as the key to exploit all the benefits of multi-agent systems, hence the motivation of studying the algorithms that make these cooperating behaviours arise.

Researchers have tried to mimic the complex group dynamics seen in nature, such as a flock of birds flying in formation. Observations imply that there is no central unit coordinating these animals, instead they are driven only by their individual observations and interactions with their neighbours. One of the first milestones achieved in the field was replicating the behaviour of a flock of birds, by defining the local rules that drive each member of the team [3].

The intuitive way to model these behaviours is as a distributed system, in which each agent is self-contained both in its sensing and its actions. The general problem to be solved is that of achieving consensus within a distributed network of agents, which means that they are able to coordinate and achieve a common global goal, such as meeting at a point (rendezvous problem) or to assemble a formation [4].

One fundamental functionality of any multiagent system is collision-free motion. For instance, in warehouse management [5], 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 and the core problem studied in this thesis.

As the number of agents to be controlled increases, the scalability of the algorithms becomes a key factor for their use with real robots. Thus, the main motivation of this work

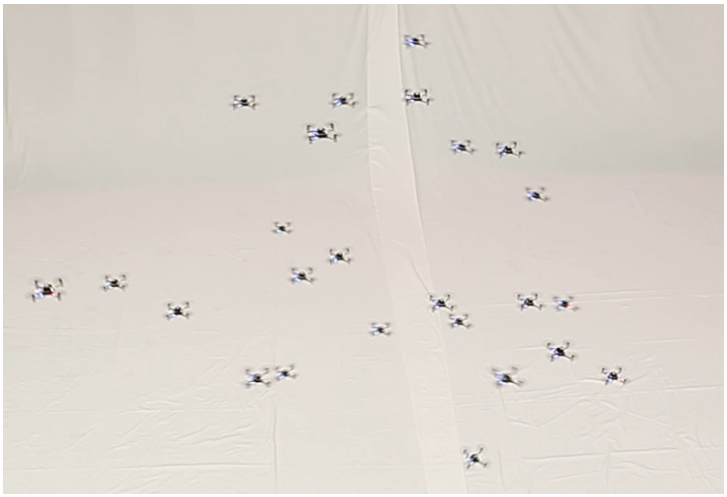


Figure 1.1: A group of 25 quadrotors Crazyflie 2.0 flying in close proximity in an indoor flying arena.

is to develop a scalable trajectory generation framework that solves the point-to-point transition problem for general multiagent systems. The algorithms are demonstrated with a swarm of small-sized quadrotors flying indoors, as depicted in Fig. 1.1.

Our first approach to tackle the problem was to solve it completely offline, designing an algorithm that outputs full trajectories for the robot team to complete the transition. With scalability in mind, we explored distributed optimization-based approaches to solve the problem in a few seconds for dozens of agents.

In an effort to add robustness to the trajectory execution, we adapted the offline approach to instead work as an online trajectory generator. This new approach replans the trajectories of the agents based on the current sensed states, allowing the agents to adapt whenever disturbances occur along the execution of the motion plans.

1.2 Related Work

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 [1,6]; in the latter, the algorithm is free to assign the goals to the agents, as to ease the complexity of the transition problem [7]. This thesis focuses on the labelled agent problem.

A common approach is to formulate an optimization problem. One of the first techniques developed relied on Mixed Integer Linear Programming (MILP), modelling collision constraints with binary variables [6]. This method is computationally expensive and

not suited for large groups of agents.

More recently, Sequential Convex Programming (SCP) [8] has been used to achieve faster computation compared to MILP. In [1], 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 [9, 10], 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 [11], limiting the initial and final locations to be vertices of the underlying grid. Other discrete planning strategies like Rapidly-exploring Random Trees (RRT) [12] have been extended to the multi-agent case. Also, a combination of discrete planning and continuous optimization has been developed to coordinate multiple robots in cluttered environments [13]. GPU-accelerated approaches can drastically reduce the runtime of these offline motion planners [14].

Other approaches combine optimization techniques and predefined behaviours to manage collisions in 2D [15]. Lyapunov barrier functions have also been used to compute multiagent collision-free trajectories [16].

Distributed optimization approaches can effectively include pair-wise distance constraints [17]. 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 [18] and non-holonomic [19] 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 [20], but they are susceptible to deadlocks.

Distributed model predictive control (DMPC) [21] has been used in coordination tasks such as formation control [22, 23], but not explicitly for point-to-point transitions. Particularly interesting are synchronous implementations of DMPC [24], 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 [25] or (2) imposing separating hyperplane constraints between the agents at every time step of the prediction horizon [22]. Both strategies are not well suited for transition tasks: strategy (1) drastically reduces the mobility of agents, especially in cluttered envi-

ronments, while strategy (2) lacks scalability and is overly conservative, as demonstrated in Ch. 2. In contrast, inspired by the incremental inclusion of *all* collision constraints over an *infinite* horizon proposed in [9], in Ch. 2 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 [26].

Despite the advances in scalability and safety of the algorithms, online trajectory generation for large groups of robots remains a challenge. ORCA and all its variants have pushed towards real-time trajectory generation, with convincing results in various robotic platforms in planar environments [27]. A similar approach achieves collision avoidance through the concept of Buffered Voronoi Cells (BVC) [28], showing initial results of online trajectory generation in 2D with multiple quadrotors operating at a fixed height. The BVC concept has been recently used in tandem with discrete planners [29], primarily to avoid deadlocks in scenarios where plain BVC would get trapped and fail the task.

Some work has also considered uncertain robot location and sensing. In [30], a probabilistic collision avoidance method is considered using a chance-constrained nonlinear MPC framework, with successful results in experiments with quadrotors sharing a workspace with a human. Other robust MPC frameworks such as tube MPC have been developed for distributed multi-agent systems, both with linear [31] and nonlinear [32] dynamics. Although both approaches provide proofs and simulation results, they are not real-time implementable with current hardware and solver capabilities. More recently, [33] developed a reciprocal collision avoidance method under sensing uncertainty for single-integrator agents.

In Ch. 3 we extend the method in Ch. 2 (publication [2]) to include online replanning of the trajectories. As such, our framework provides an essential functionality for higher level planners that specify complex team missions in terms of goal locations to be visited by the agents.

Our approach contrasts from current online methods (e.g., [29]) in that:

- It is purely optimization-based, in the form of a standard QP. No discrete planner is running in the background, which reduces the computation time.
- It uses *on-demand collision avoidance* instead of the BVC (or ORCA) method for partitioning the free space, resulting in less conservative movement and faster transition times.

Our results suggest that the proposed method is well-suited for online trajectory

generation, with average computation times of 28 ms for 20 quadrotors, all from a single offboard computer. Moreover, we demonstrate how our method creates less conservative plans than BVC, which ultimately leads to faster completion of the task and higher probability of solving scenarios with high agent density.

1.3 Contributions

The main contributions of this thesis are now listed:

- An offline trajectory generation method based on distributed model predictive control. The work on this topic resulted in the publication in [2]. A video demonstrating the results is found at `http://tiny.cc/dmpc-swarm`.
- An online trajectory generation method using real-time distributed model predictive control. The work on this topic resulted in the paper in [34]. A video of the performance is found at `http://tiny.cc/online-dmpc`.

1.4 Thesis Overview

The thesis is organized as follows: Ch. 2 presents the offline trajectory generation framework. In Ch. 3 we develop a framework for real-time trajectory generation. In Ch. 4 we summarize our results, give concluding remarks and propose future research directions.

Chapter 2

Multiagent Offline Trajectory Generation

2.1 Introduction

This chapter proposes a novel approach to solve the point-to-point transition problem offline. In the context of this work, offline means that the whole transition trajectories are computed prior to their execution. The two main assumptions are as follows: i) the environment is static and known in advance and ii) the agents will not suffer disturbances while executing their trajectories. In Ch. 3 we relax assumption ii) by replanning trajectories online.

Despite these assumptions, offline trajectory generation is still useful and commonly used in a wide variety of tasks. In particular, in structured environments where the agents are guaranteed to remain unperturbed, then offline trajectory generation is enough to solve the point-to-point transition problem.

The key contributions of this chapter 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 chapter is organized as follows: Sec. 2.2 states the problem. Sec. 2.3 introduces the optimization formulation to solve it. The algorithm is presented in Sec. 2.4 and demonstrated in simulation (Sec. 2.5) and experiments with a swarm of quadrotors

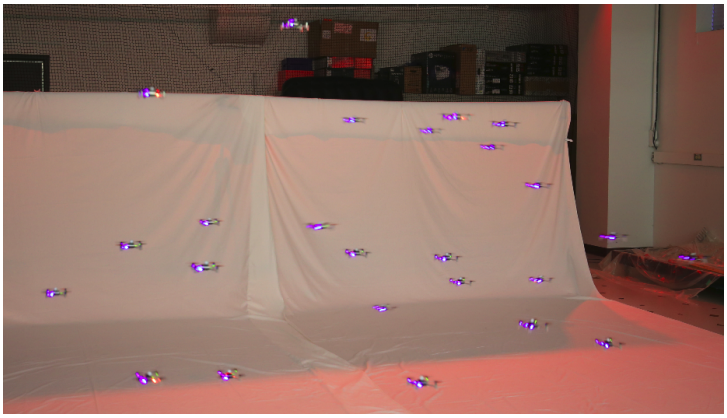


Figure 2.1: A group of 25 Crazyflie 2.0 quadrotors performing a point-to-point transition using our distributed model predictive control (DMPC) algorithm. A video of the performance is found at `http://tiny.cc/dmpc-swarm`.

(Sec. 2.6). A picture of our 25-drone swarm testbed is shown in Fig. 2.1, along with an accompanying video showcasing our method and results.

2.2 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.

2.2.1 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

$$\mathbf{p}_i[k+1] = \mathbf{p}_i[k] + h\mathbf{v}_i[k] + \frac{h^2}{2}\mathbf{a}_i[k], \tag{2.1}$$

$$\mathbf{v}_i[k+1] = \mathbf{v}_i[k] + h\mathbf{a}_i[k]. \tag{2.2}$$

2.2.2 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,

$$\mathbf{a}_{\min} \leq \mathbf{a}_i[k] \leq \mathbf{a}_{\max}. \tag{2.3}$$

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

$$\mathbf{p}_{\min} \leq \mathbf{p}_i[k] \leq \mathbf{p}_{\max}. \tag{2.4}$$

2.2.3 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 [11]. The collision constraint between agents i and j is defined using a scaling matrix $\mathbf{\Theta}$,

$$\left\| \mathbf{\Theta}^{-1} \left(\mathbf{p}_i[k] - \mathbf{p}_j[k] \right) \right\|_n \geq r_{\min}, \tag{2.5}$$

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 $\mathbf{\Theta} = \text{diag}(a, b, c)$. 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 (2.5) checks whether agent j (or i), modelled as a 3D point, is inside an ellipsoid centered around agent i (or j).

2.3 Distributed Model Predictive Control

The problem formulated in Sec. 2.2 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.

2.3.1 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.

2.3.2 The Agent Prediction Model

Using the dynamics in (2.1) and (2.2), 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

$$\begin{bmatrix} \hat{\mathbf{p}}_i[k+1|k_t] \\ \hat{\mathbf{v}}_i[k+1|k_t] \end{bmatrix} = \begin{bmatrix} \mathbf{I}_3 & h\mathbf{I}_3 \\ \mathbf{0}_3 & \mathbf{I}_3 \end{bmatrix} \begin{bmatrix} \hat{\mathbf{p}}_i[k|k_t] \\ \hat{\mathbf{v}}_i[k|k_t] \end{bmatrix} + \begin{bmatrix} (h^2/2)\mathbf{I}_3 \\ h\mathbf{I}_3 \end{bmatrix} \hat{\mathbf{a}}_i[k|k_t], \tag{2.6}$$

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

$$\hat{\mathbf{x}}_i[k+1|k_t] = \mathbf{A}\hat{\mathbf{x}}_i[k|k_t] + \mathbf{B}\hat{\mathbf{u}}_i[k|k_t], \tag{2.7}$$

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 $\mathbf{U}_i \in \mathbb{R}^{3K}$,

$$\mathbf{P}_i = \mathbf{A}_0\mathbf{X}_{0,i} + \mathbf{\Lambda}\mathbf{U}_i, \tag{2.8}$$

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

$$\mathbf{\Lambda} = \begin{bmatrix} \mathbf{\Psi B} & \mathbf{0}_3 & \dots & \mathbf{0}_3 \\ \mathbf{\Psi A B} & \mathbf{\Psi B} & \dots & \mathbf{0}_3 \\ \vdots & \ddots & \ddots & \vdots \\ \mathbf{\Psi A}^{K-1}\mathbf{B} & \mathbf{\Psi A}^{K-2}\mathbf{B} & \dots & \mathbf{\Psi B} \end{bmatrix}, \tag{2.9}$$

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,

$$\mathbf{A}_0 = \begin{bmatrix}(\mathbf{\Psi A})^\intercal & (\mathbf{\Psi A}^2)^\intercal & \dots & (\mathbf{\Psi A}^K)^\intercal\end{bmatrix}^\intercal. \tag{2.10}$$

2.3.3 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 [35].

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

$$e_i = \sum_{k=K-\kappa}^{K} \left\|\hat{\mathbf{p}}_i[k|k_t] - \mathbf{p}_{d,i}\right\|_2. \tag{2.11}$$

This term can be turned into a quadratic cost function in terms of the input sequence using (2.8),

$$J_{e,i} = \mathbf{U}_i^\intercal(\mathbf{\Lambda}^\intercal\tilde{\mathbf{Q}}\mathbf{\Lambda})\mathbf{U}_i - 2(\mathbf{P}_{d,i}^\intercal\tilde{\mathbf{Q}}\mathbf{\Lambda} - (\mathbf{A}_0\mathbf{X}_{0,i})^\intercal\,\tilde{\mathbf{Q}}\mathbf{\Lambda})\mathbf{U}_i, \tag{2.12}$$

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

Control effort penalty

We also aim to minimize the control effort using the quadratic cost function

$$J_{u,i} = \mathbf{U}_i^\intercal \tilde{\mathbf{R}} \mathbf{U}_i. \tag{2.13}$$

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

Input variation penalty

This term is used to minimize variations of the acceleration, leading to smooth input trajectories. We define the quadratic cost

$$\delta_i = \sum_{k=0}^{K-1} \left\| \hat{\mathbf{u}}_i[k|k_t] - \hat{\mathbf{u}}_i[k-1|k_t] \right\|_2. \tag{2.14}$$

To transform (2.14) into a quadratic form, first we define a matrix $\boldsymbol{\Delta} \in \mathbb{R}^{3K\times 3K}$,

$$\boldsymbol{\Delta} = \begin{bmatrix} \mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \ldots & \mathbf{0}_3 & \mathbf{0}_3 \\ -\mathbf{I}_3 & \mathbf{I}_3 & \mathbf{0}_3 & \ldots & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & -\mathbf{I}_3 & \mathbf{I}_3 & \ldots & \mathbf{0}_3 & \mathbf{0}_3 \\ \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \ldots & -\mathbf{I}_3 & \mathbf{I}_3 \end{bmatrix}, \tag{2.15}$$

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

$$\mathbf{U}_{i*} = \begin{bmatrix} \mathbf{u}_i[k_t - 1]^\intercal & \mathbf{0}_{3\times 1}^\intercal & \ldots & \mathbf{0}_{3\times 1}^\intercal \end{bmatrix}^\intercal. \tag{2.16}$$

Finally, we write (2.14) in quadratic form as

$$J_{\delta,i} = \mathbf{U}_i^\intercal (\boldsymbol{\Delta}^\intercal \tilde{\mathbf{S}} \boldsymbol{\Delta}) \mathbf{U}_i - 2(\mathbf{U}_{i*}^\intercal \tilde{\mathbf{S}} \boldsymbol{\Delta}) \mathbf{U}_i, \tag{2.17}$$

where $\tilde{\mathbf{S}} \in \mathbb{R}^{3K\times 3K}$ is positive definite and block-diagonal, defined as $\tilde{\mathbf{S}} = \text{diag}(\mathbf{S}, \ldots, \mathbf{S})$, where $\mathbf{S} \in \mathbb{R}^{3\times 3}$ weights the penalty on control variation. The cost function $\mathcal{J}_i$ is obtained by adding together (2.12), (2.13) and (2.17),

$$\begin{aligned}\mathcal{J}_i(\mathbf{U}_i) = \mathbf{U}_i^\intercal(\mathbf{\Lambda}^\intercal\tilde{\mathbf{Q}}\mathbf{\Lambda} + \tilde{\mathbf{R}} + \mathbf{\Delta}^\intercal\tilde{\mathbf{S}}\mathbf{\Delta})\mathbf{U}_i \\ - 2(\mathbf{P}_{d,i}^\intercal\tilde{\mathbf{Q}}\mathbf{\Lambda} - (\mathbf{A}_0\mathbf{X}_{0,i})^\intercal\,\tilde{\mathbf{Q}}\mathbf{\Lambda} + \mathbf{U}_{i*}^\intercal\tilde{\mathbf{S}}\mathbf{\Delta})\mathbf{U}_i.\end{aligned} \tag{2.18}$$

2.3.4 Physical Limits

When computing the input sequence over the horizon, the agents must satisfy constraints (2.3) and (2.4). Define $\mathbf{P}_{\min}, \mathbf{P}_{\max}, \mathbf{U}_{\min}, \mathbf{U}_{\max} \in \mathbb{R}^{3K}$ to be

$$\begin{aligned}\mathbf{P}_{\min} = [\mathbf{p}_{\min}^\intercal \ldots \mathbf{p}_{\min}^\intercal]^\intercal; \quad \mathbf{P}_{\max} = [\mathbf{p}_{\max}^\intercal \ldots \mathbf{p}_{\max}^\intercal]^\intercal \\ \mathbf{U}_{\min} = [\mathbf{a}_{\min}^\intercal \ldots \mathbf{a}_{\min}^\intercal]^\intercal; \quad \mathbf{U}_{\max} = [\mathbf{a}_{\max}^\intercal \ldots \mathbf{a}_{\max}^\intercal]^\intercal.\end{aligned} \tag{2.19}$$

The physical limits are formulated as

$$\begin{aligned}\mathbf{P}_{\min} - \mathbf{A}_0\mathbf{X}_{0,i} \leq \mathbf{\Lambda}\mathbf{U}_i \leq \mathbf{P}_{\max} - \mathbf{A}_0\mathbf{X}_{0,i} \\ \mathbf{U}_{\min} \leq \mathbf{U}_i \leq \mathbf{U}_{\max}.\end{aligned} \tag{2.20}$$

Lastly, we can vertically stack both inequality constraints in (2.20) to obtain a single expression: $\mathbf{A}_{\text{in}}\mathbf{U}_i \leq \mathbf{b}_{\text{in}}$.

2.3.5 Convex Optimization Problem, No Collision Case

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

$$\begin{aligned}\underset{\mathbf{U}_i}{\text{minimize}} \quad & \mathcal{J}_i(\mathbf{U}_i) \\ \text{subject to} \quad & \mathbf{A}_{\text{in}}\mathbf{U}_i \leq \mathbf{b}_{\text{in}}.\end{aligned} \tag{2.21}$$

The formulation in (2.21) results in a quadratic programming problem with $3K$ decision variables and $12K$ inequality constraints, which scales independently of N.

2.3.6 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 [9] 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

$$\xi_{ij} = \left\| \boldsymbol{\Theta}^{-1} \left(\hat{\mathbf{p}}_i[k_{c,i}|k_t - 1] - \hat{\mathbf{p}}_j[k_{c,i}|k_t - 1] \right) \right\|_n \geq r_{\min} \tag{2.22}$$

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 $f(r_{\min}) = 3r_{\min}$. 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:

$$\left\| \boldsymbol{\Theta}^{-1} \left(\hat{\mathbf{p}}_i[k_{c,i} - 1|k_t] - \hat{\mathbf{p}}_j[k_{c,i}|k_t - 1] \right) \right\|_n \geq r_{\min} + \varepsilon_{ij}, \tag{2.23}$$

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 (2.23). 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 $\hat{\mathbf{p}}_i[k_{c,i}|k_t - 1]$,

$$\boldsymbol{\nu}_{ij}^{\mathsf{T}} \hat{\mathbf{p}}_i[k_{c,i}|k_t] - \varepsilon_{ij} \xi_{ij} \geq \rho_{ij} \tag{2.24}$$

with $\boldsymbol{\nu}_{ij} = \boldsymbol{\Theta}^{-n} (\hat{\mathbf{p}}_i[k_{c,i}|k_t - 1] - \hat{\mathbf{p}}_j[k_{c,i}|k_t - 1])^{n-1}$ and $\rho_{ij} = r_{\min} \xi_{ij}^{n-1} - \xi_{ij}^n + \boldsymbol{\nu}_{ij}^{\mathsf{T}} \hat{\mathbf{p}}_i[k_{c,i}|k_t - 1]$. On the left-hand side of (2.24), 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 (2.24) into an affine function of the decision variables,

$$\boldsymbol{\mu}_{ij}^{\mathsf{T}} \boldsymbol{\Lambda} \mathbf{U}_i - \varepsilon_{ij} \xi_{ij} \geq \rho_{ij} - \boldsymbol{\mu}_{ij}^{\mathsf{T}} \mathbf{A}_0 \mathbf{X}_{0,i}, \tag{2.25}$$

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

$$\boldsymbol{\mu}_{ij} = \begin{bmatrix} \mathbf{0}_{3(k_{c,i}-1)\times 1}^{\mathsf{T}} & \boldsymbol{\nu}_{ij}^{\mathsf{T}} & \mathbf{0}_{3(K-k_{c,i})\times 1}^{\mathsf{T}} \end{bmatrix}^{\mathsf{T}}. \tag{2.26}$$

By stacking the inequalities in (2.25) for the $n_{c,i}$ colliding neighbours, we obtain the complete collision constraint,

$$\mathbf{A}_{\text{coll}} \boldsymbol{\mathcal{U}}_i \leq \mathbf{b}_{\text{coll}}. \tag{2.27}$$

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:

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

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

$$\mathcal{J}_{\text{aug},i}(\boldsymbol{\mathcal{U}}_i) = \mathcal{J}(\mathbf{U}_i) + \boldsymbol{\mathcal{U}}_i^{\mathsf{T}} \mathbf{H}_{\varepsilon,i} \boldsymbol{\mathcal{U}}_i - f_{\varepsilon,i}^{\mathsf{T}} \boldsymbol{\mathcal{U}}_i. \tag{2.28}$$

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

$$\begin{aligned} \underset{\boldsymbol{\mathcal{U}}_i}{\text{minimize}} \quad & \mathcal{J}_{\text{aug},i}(\boldsymbol{\mathcal{U}}_i) \\ \text{subject to} \quad & \mathbf{A}_{\text{in,aug}} \boldsymbol{\mathcal{U}}_i \leq \mathbf{b}_{\text{in,aug}}. \end{aligned} \tag{2.29}$$

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

2.4 The Algorithm

The proposed DMPC algorithm for point-to-point transitions is outlined in Alg. 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 Sec. 2.3.6. If no collisions are detected, we solve the reduced problem in (2.21), otherwise we solve the collision avoidance problem in (3.22). If the optimizer finds a solution to the QP, then we can propagate the states using (2.7) 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 [9], 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.

Algorithm 1: DMPC for Point-to-Point Transitions

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

1 $[\mathbf{\Pi}, \mathbf{x}[0]] \leftarrow$ `initAllPredictions` $(\mathbf{p}_0, \mathbf{p}_f)$
2 $k_t \leftarrow 0$, AtGoal $\leftarrow$ false
3 **while** *not AtGoal and* $k_t < K_{\max}$ **do**
4 **foreach** *agent* $i = 1, ..., N$ **do**
5 $\hat{\mathbf{a}}_i[k|k_t] \leftarrow$ `build&SolveQP` $(\mathbf{x}_i[k_t], \mathbf{a}_i[k_t - 1], \mathbf{\Pi})$
6 **if** *QP feasible* **then**
7 $\hat{\mathbf{x}}_i[k+1|k_t] \leftarrow$ `getStates` $(\mathbf{x}_i[k_t], \hat{\mathbf{a}}_i[k|k_t])$
8 $\mathbf{\Pi}_i \leftarrow \hat{\mathbf{p}}_i[k+1|k_t]$
9 $\mathbf{x}_i[k_t+1], \mathbf{a}_i[k_t] \leftarrow \hat{\mathbf{x}}_i[1|k_t], \hat{\mathbf{a}}_i[0|k_t]$
10 AtGoal $\leftarrow$ `checkGoal` $(\mathbf{p}[k_t], \mathbf{p}_f)$
11 $k_t \leftarrow k_t + 1$
12 **if** *AtGoal* **then**
13 $[\mathbf{p}, \mathbf{v}, \mathbf{a}] \leftarrow$ `scaleTrajectory` $(\mathbf{p}, \mathbf{v}, \mathbf{a}, \|\mathbf{a}_{\max}\|)$
14 $[\mathbf{p}, \mathbf{v}, \mathbf{a}] \leftarrow$ `interpolate` $(\mathbf{p}, \mathbf{v}, \mathbf{a}, T_s)$
15 `checkCollisions` $(\mathbf{p}, r_{\min} - \varepsilon_{\text{check}})$
16 **return** $[\mathbf{p}, \mathbf{v}, \mathbf{a}]$

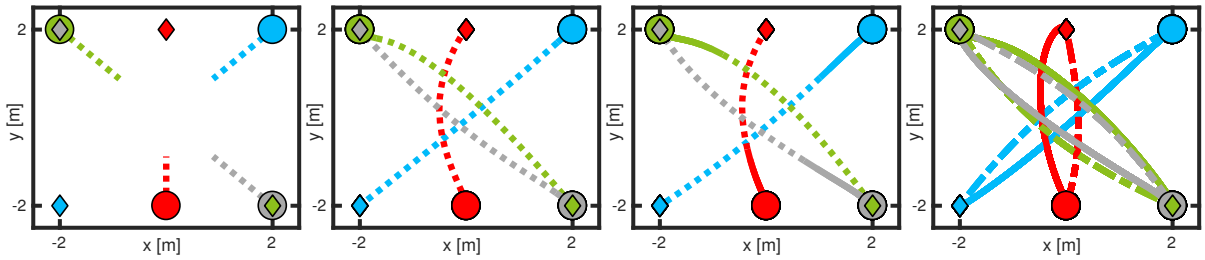
2.4.1 Example Scenario

To illustrate how DMPC manages colliding trajectories, Fig. 2.2 shows a transition problem for four agents in the plane. Initially, as shown in Fig. 2.2a, the agents follow a direct path towards their desired final locations. In Fig. 2.2b, collisions are detected and considered in the optimization problem. After a few time steps, the agents obtain the non-colliding plan seen in Fig. 2.2c. The trajectories generated with a centralized approach are quite different than the DMPC trajectories, as shown in Fig. 2.2d. 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.

2.4.2 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 (2.27) 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 Alg. 1 we apply this technique to ensure recursive



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

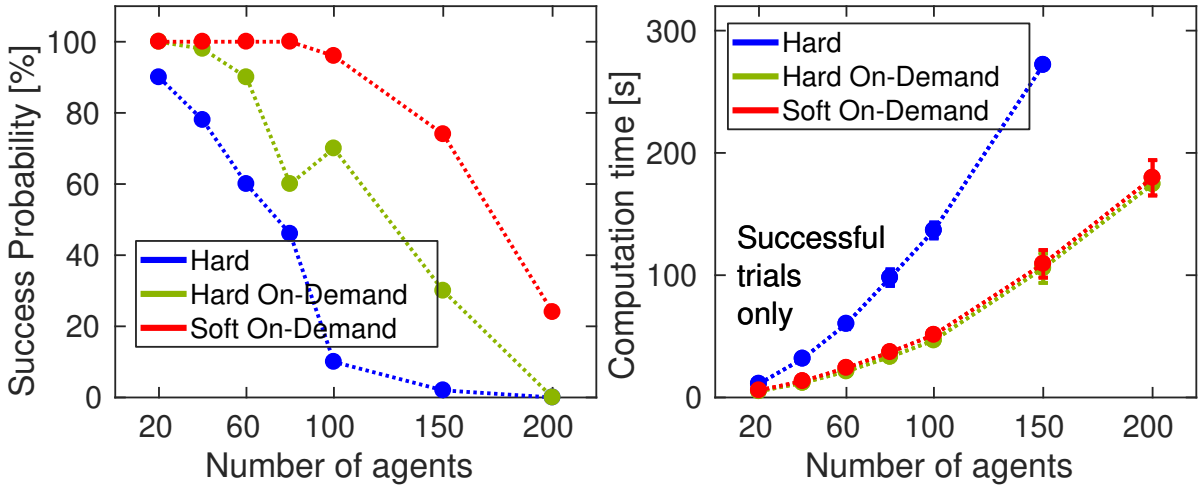
Figure 2.2: Four-agent position exchange scenario in 2D solved using Alg. 1. Circles and diamonds represent initial and final locations, respectively. Dotted lines in (a) - (c) represent the predicted positions over a 3-second horizon, solid lines are the generated trajectories and dashed lines in (d) are the trajectories generated by the centralized approach in [1]. 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.

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 [1]. 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.

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.



(a) (b)

Figure 2.3: 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.

2.5 Simulations

This section provides a simulation analysis of the DMPC algorithm. Implementation was done in MATLAB 2017a (using a sequential implementation of Alg. 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 $r_{\min} = 0.35\,\text{m}$, $a_{\max} = -a_{\min} = 1\,\text{m/s}^2$, and $c = 2$ (to avoid downwash).

2.5.1 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 [22]) and (2) implementing our on-demand collision avoidance with hard constraints (i.e., constraint (2.23) 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 $h = 0.2\,\text{s}$ and $T_s = 0.01\,\text{s}$. We used a horizon length $K = 15$, parameter $\kappa = 1$, a maximum relaxation of $\varepsilon_{\max} = 0.05\,\text{m}$ for the optimizer, a maximum relaxation of $\varepsilon_{\text{check}} = 0.05\,\text{m}$ for the safety check and a maximum time to complete the transition $T_{\max} = 20\,\text{s}$. Fig. 2.3a shows the success rate of DMPC for point-to-point transitions using the different collision

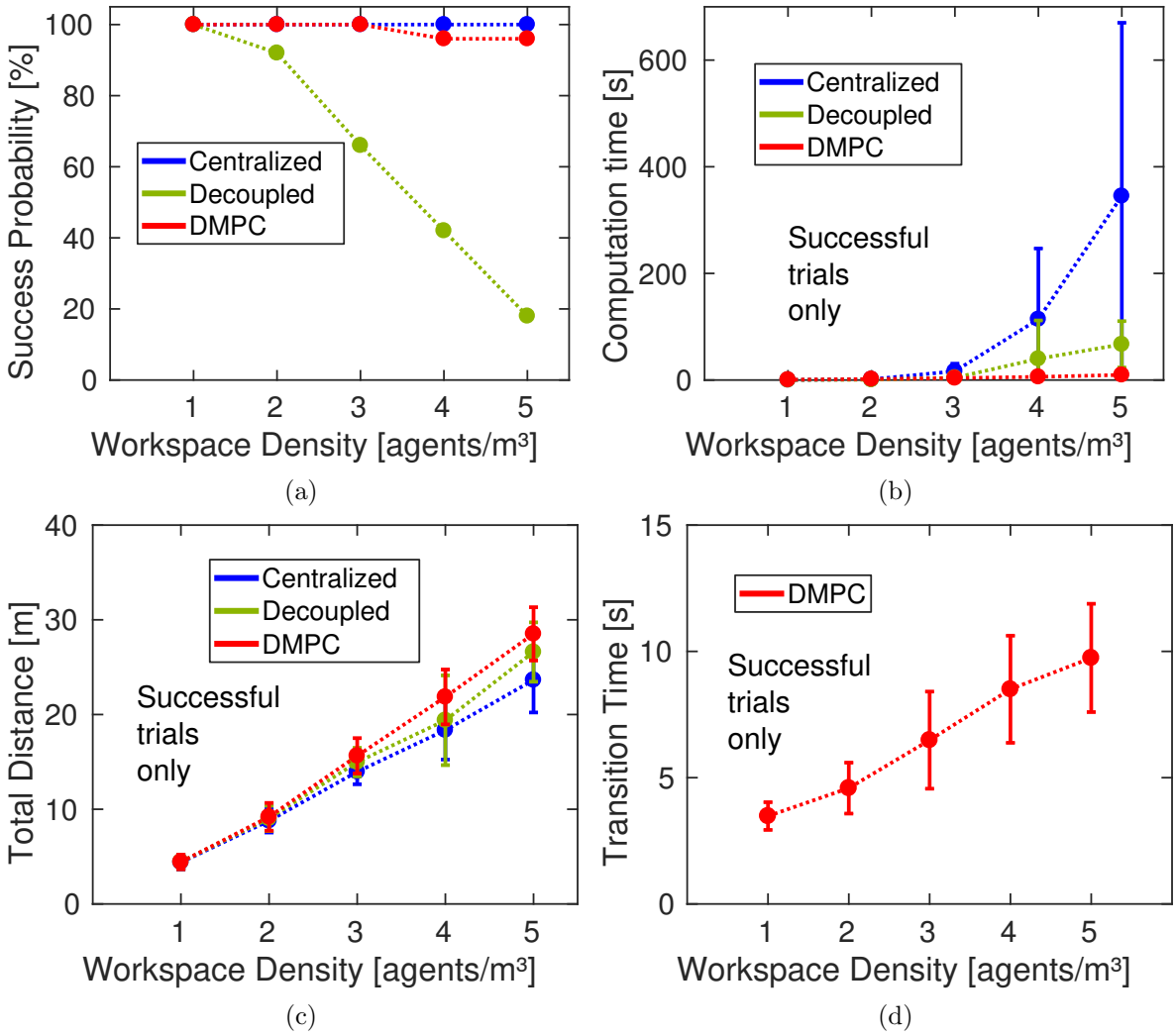


Figure 2.4: Performance comparison of DMPC against SCP-based approaches, in a fixed 4m^3 volume. For every density considered, 50 different random test cases were generated.

avoidance schemes. If we use hard constraints 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. 2.3b we highlight the reduction in computation time with our on-demand collision avoidance strategy.

2.5.2 Comparison to SCP-Based Approaches

We compared the performance of our proposed DMPC scheme with two state-of-the-art algorithms: centralized [1] and decoupled [9] SCP. We used the same simulation parameters as in Sec. 2.5.1, but the volume of the workspace was kept fixed at $4\,\mathrm{m}^3$, 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

Fig. 2.4a 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. 2.4b 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. 2.4d. 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. 2.4c. 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.

2.6 Experiments

In this section we present experimental results using Alg. 1 as an offline trajectory planner for a swarm of Crazyflies 2.0. The algorithm was implemented in C++ using OOQP as

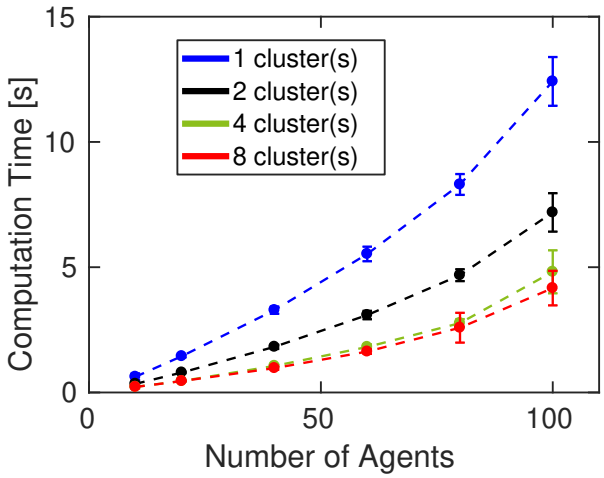


Figure 2.5: Average computation time for different numbers of clusters. For each swarm size, we gathered data of 30 successful transitions and reported the mean and standard deviation (vertical bars) of the runtime.

the solver. A video of the performance is found at `http://tiny.cc/dmpc-swarm`.

2.6.1 Parallel DMPC

Leveraging the parallel nature of the inner loop of Alg. 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 fewer agents. After all the clusters finish solving their QPs, they exchange the updated predictions and repeat the process.

In Fig. 2.5 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).

2.6.2 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 [36]. One transition scenario is depicted in Fig. 2.6, in which the swarm was to transition from a 5×5 grid

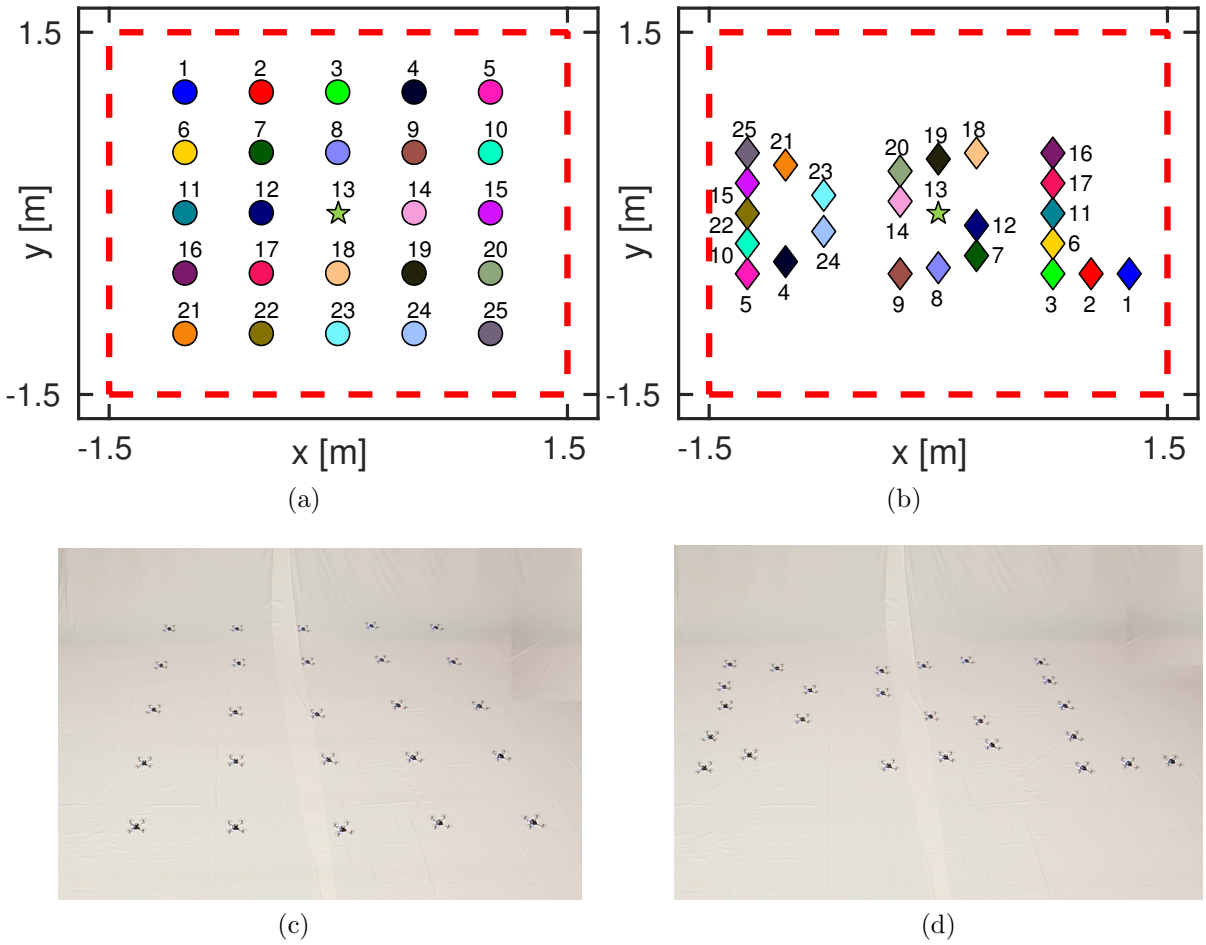
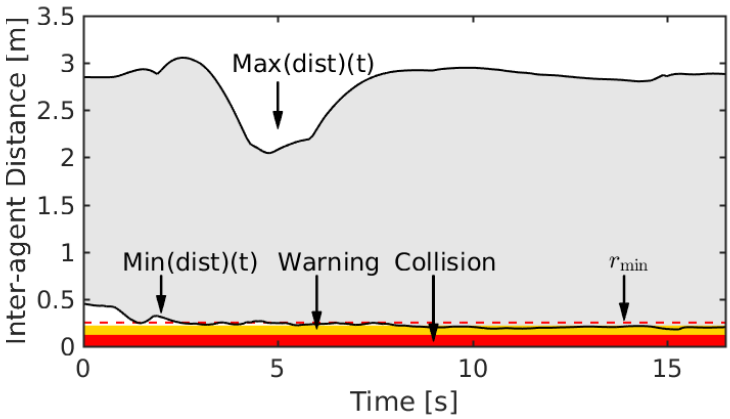
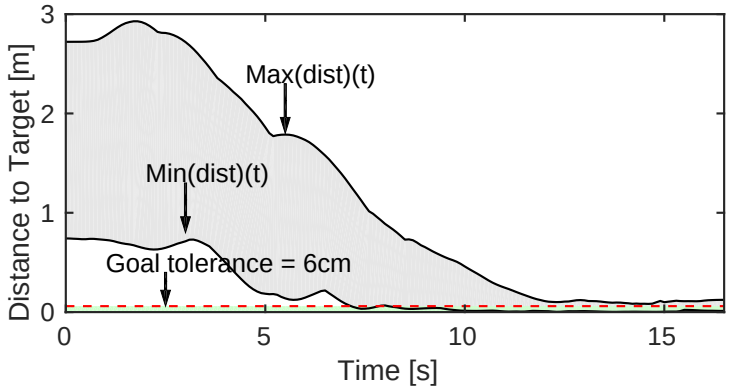


Figure 2.6: A 25-agent transition scenario: (a) initial grid configuration, (b) target 'DSL' configuration. Circles and diamonds (of matching colour) represent initial and final locations for all agents, respectively. The star in the middle represents an agent acting as a static obstacle. The bounding box in dashed red lines represents the workspace boundaries. Figures (c)-(d) are the initial and final configuration snapshots from our experiments.



(a)



(b)

Figure 2.7: Experimental data from the transition depicted in Fig. 2.6, showing maximum and minimum distance values over 6 independent trials: (a) pairwise distances, (b) distances to target locations.

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. 2.7a, 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 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. 2.7b 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 onboard 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 this video `http://tiny.cc/dmpc-swarm`.

2.7 Summary

The DMPC algorithm developed in this chapter 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.

Chapter 3

Multiagent Online Trajectory Generation

3.1 Introduction

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

In this chapter we extend the approach presented in Ch. 2 to allow for online replanning of the transition trajectories. As such, our framework adds robustness to the trajectory execution by recomputing trajectories in real-time using the sensed states of the agents. Essentially, we propose a *closed-loop* motion planning scheme, whereas the framework in Ch. 2 was *open-loop*.

The main contributions of this chapter are threefold: (*i*) a multiagent motion planning framework based on distributed model predictive control, which allows for real-time trajectory generation, (*ii*) an event-triggered replanning strategy for robust execution of plans and (*iii*) a thorough empirical evaluation of the method. To the best of our knowledge, this chapter presents the first results on real-time motion planning for drone swarms of up to 20 drones, executed from a single off-board computer.

The rest of the chapter is organized as follows: Sec. 3.2 introduces the problem at hand. Sec. 3.3 formalizes the optimization behind DMPC and Sec. 3.4 introduces the trajectory replanning strategy. The algorithm for input updates is presented in Sec. 3.5. Finally, Sec. 3.6 and Sec. 3.7 provide simulation and experimental results of our approach

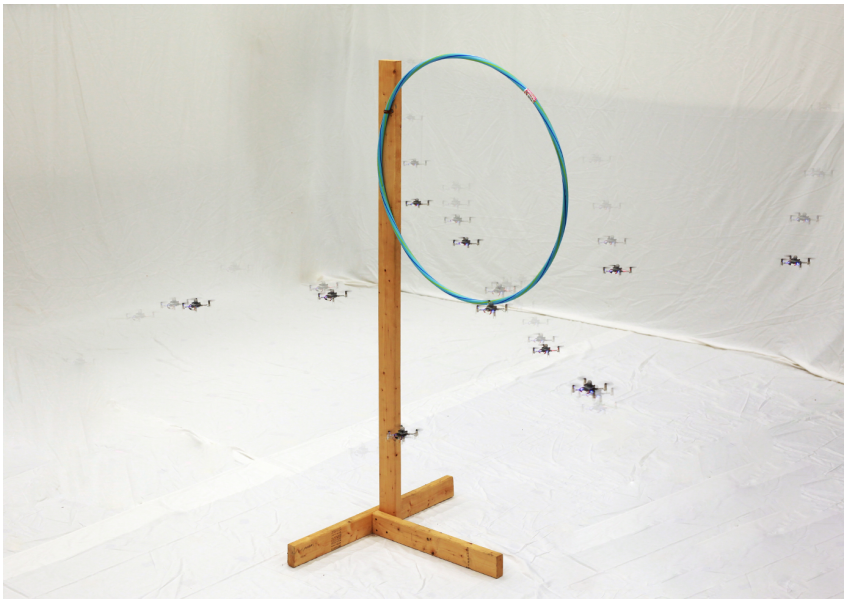


Figure 3.1: A ten-drone transition task through a hula-hoop solved using our proposed online trajectory generation method. Our distributed computation allows for real-time multi-robot trajectory generation, enabling complex transition tasks to be performed. A video of the performance is found at `http://tiny.cc/online-dmpc`.

with teams of drones.

3.2 Problem Statement

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

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

3.2.1 The Agents

We assume every agent i is equipped with a controller for position trajectory tracking and $\mathbf{u}_i$ is a position reference, as shown in Fig. 3.2.

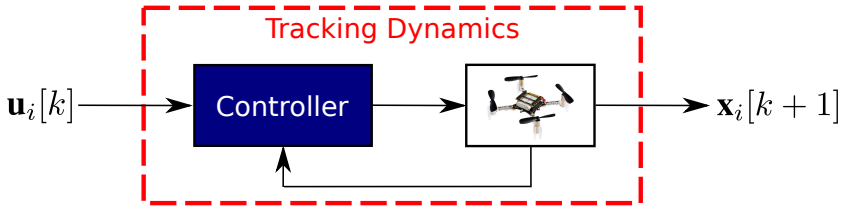


Figure 3.2: Block diagram of the control system of agent i. Here we depict the agent as a Crazyflie 2.0 quadrotor, which is our experimental platform.

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

$$\mathbf{x}_i[k+1] = \mathbf{A}_i\mathbf{x}_i[k] + \mathbf{B}_i\mathbf{u}_i[k]. \tag{3.1}$$

For example, in this chapter we will consider the system (3.1) to represent a quadrotor with an underlying position controller [36], for which the input ($\mathbf{u}_i[k] \in \mathbb{R}^3$) is a position reference signal, and the states ($\mathbf{x}_i[k] \in \mathbb{R}^6$) are the position and velocity of the vehicle, i.e., $\mathbf{x}_i[k] = (\mathbf{p}_i[k], \mathbf{v}_i[k])$. This derives in a second order system defining the dynamics, with $\mathbf{A}_i \in \mathbb{R}^{6\times6}$, $\mathbf{B}_i \in \mathbb{R}^{6\times3}$.

Note that system (3.1) differs conceptually to the one presented in the previous chapter (see (2.7)) since now we consider the state $\mathbf{x}_i[k]$ to represent the *measured* state of the agent, whereas in Ch. 2 the state represented the position and velocity profiles of the *reference* trajectory.

3.3 Online Distributed Model Predictive Control

In this section we formalize the optimization problem solved in real-time within a receding horizon control framework. The approach is based on the offline method presented in Sec. 2.3, now accommodating the new input parameterization and the trajectory tracking dynamics.

3.3.1 Trajectory Parameterization

Our approach is based on receding horizon control, meaning that at the time step k_t, corresponding to the time instant t_0, we recompute the input sequence to be applied over a finite horizon of K time steps. Given a desired time step duration h, we get the continuous time horizon duration $t_h = (K-1)/h$. We parameterize the continuous input

signal $\mathbf{u}_i(t)$ for $t \in [t_0,\, t_0 + t_h]$ as a concatenation of l Bézier curves, similar to [13]. For a summary on Bézier curves and Bernstein polynomials we refer the reader to [37].

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

$$B_{m,p}(t) = \binom{p}{m} (1 - t/T)^{p-m} (t/T)^m \quad \forall t \in (0,\, T), \tag{3.2}$$

with $m = 0, 1, \ldots, p$. Now, an n-dimensional Bézier curve of degree p is defined as:

$$\mathbf{B}(t) = \sum_{m=0}^{p} \mathbf{P}_m B_{m,p}(t) \tag{3.3}$$

with $\mathbf{P}_m \in \mathbb{R}^n$. The set $\mathcal{P} = \{\mathbf{P}_0, \mathbf{P}_1, \ldots, \mathbf{P}_p\}$ represents the $p+1$ control points that uniquely characterize the curve. Since the control points are a finite parameterization of the continuous curve, they serve as an optimization variable to compute the agents' trajectories.

Note that (3.3) implies that the individual components of a Bézier curve in $\mathbb{R}^n$ are decoupled. This means that a Bézier curve in $\mathbb{R}^n$ can be seen as n Bézier curves in $\mathbb{R}$. In what follows, the matrix products and derivations are thought for Bézier curves in $\mathbb{R}$, but they are immediately applicable for curves in higher dimensions as per their decoupling principle.

Expressing the Bézier curve in the power basis $\{1,\, t,\, \ldots, t^p\}$ is useful to obtain samples of the curve. Consider the power basis representation of the Bernstein polynomials in (3.2):

$$\bar{B}_{k,p}(t) = \sum_{m=k}^{p} (-1)^{m-k} \binom{p}{m} \binom{m}{k} (t/T)^m. \tag{3.4}$$

with $k = 0, 1, \ldots, p$. Then we can write the equivalence

$$\mathbf{B}(t) = \sum_{i=0}^{p} \mathbf{P}_m B_{m,p}(t) = \sum_{k=0}^{p} \mathbf{S}_k \bar{B}_{k,p}(t) \tag{3.5}$$

with $\mathbf{S}_k \in \mathbb{R}^n$, and the set $\mathcal{S} = \{\mathbf{S}_0, \mathbf{S}_1, \ldots, \mathbf{S}_p\}$ representing the $p+1$ polynomial coefficients of the power basis. We can then define a transformation matrix $\boldsymbol{\beta}$ such that

$\mathcal{S} = \boldsymbol{\beta}\mathcal{P}$, allowing us to easily change between polynomial bases. Define

$$\boldsymbol{\beta}_{[m,k]} = \begin{cases} (-1)^{m-k} \binom{p}{m} \binom{m}{k}, & \text{if } m \leq k,\ k = 0, 1, \ldots, p \\ 0, & \text{otherwise} \end{cases}, \tag{3.6}$$

where $\boldsymbol{\beta}_{[m,k]}$ represents the element at the m^{th} row and k^{th} column of matrix $\boldsymbol{\beta}$.

By definition, the derivative of a Bézier curve of degree p is another Bézier curve of degree $p-1$ [37]. The derivative of the Bernstein polynomials is given by the linear equation

$$\frac{d}{dt} B_{m,p}(t) = p(B_{m-1,p-1} - B_{m,p-1}). \tag{3.7}$$

Using (3.7) we can define a set of matrices $\{\boldsymbol{\Sigma}, \boldsymbol{\Sigma}^{(1)}, \ldots, \boldsymbol{\Sigma}^{(r)}\}$ that map the original control points ($\mathcal{P}$) to the control points of the r^{th} derivative of $\mathbf{B}(t)$ ($\mathcal{P}^{(r)}$). To obtain the power basis coefficients of $\mathbf{B}^{(r)}(t)$, first define the transformation matrix $\boldsymbol{\beta}^{(r)}$ using (3.6) with $k = 0, 1, \ldots, p-r$. Then, we can represent the set $\mathcal{S}^{(r)} = \{\mathbf{S}_0^{(r)}, \mathbf{S}_1^{(r)}, \ldots, \mathbf{S}_{p-r}^{(r)}\}$ of $p-r+1$ power basis coefficients of $\mathbf{B}^{(r)}(t)$ with $\mathcal{S}^{(r)} = \boldsymbol{\beta}^{(r)}\boldsymbol{\Sigma}^{(r)}\mathcal{P}$.

One last important aspect of Bézier curves is they can be sampled for any $t \in [0, T]$, and such samples are a linear combination of the polynomial coefficients (and thus, of the control points). Suppose we want to compute K samples of the curve, represented by the set $\mathcal{B} = \{\mathbf{B}(t_0), \mathbf{B}(t_1), \ldots, \mathbf{B}(t_{K-1})\}$. The set can be computed as:

$$\begin{pmatrix} \mathbf{B}(t_0) \\ \mathbf{B}(t_1) \\ \vdots \\ \mathbf{B}(t_{K-1}) \end{pmatrix} = \begin{pmatrix} 1 & t_0 & \ldots & t_0^p \\ 1 & t_1 & \ldots & t_1^p \\ \vdots & \vdots & \vdots & \vdots \\ 1 & t_{K-1} & \ldots & t_{K-1}^p \end{pmatrix} \begin{pmatrix} \mathbf{S}_0 \\ \mathbf{S}_1 \\ \vdots \\ \mathbf{S}_p \end{pmatrix}, \tag{3.8}$$

with the relationship $\mathcal{B} = \mathbf{T}\mathcal{S} = \mathbf{T}\boldsymbol{\beta}\mathcal{P}$. The same procedure applies for sampling derivatives of the curve, using the relationship $\mathcal{B}^{(r)} = \mathbf{T}\mathcal{S}^{(r)}$.

3.3.2 The Agent Prediction Model

Using the linear trajectory tracking model of (3.1) and a series of inputs, we can compute the agents' states over a horizon of fixed length K. We introduce the notation $\hat{(\cdot)}[k|k_t]$, which represents the predicted value of $(\cdot)[k_t + k]$ with the information available at k_t

and $k \in \{0, \ldots, K-1\}$. The prediction model of agent i is given by

$$\hat{\mathbf{x}}_i[k+1|k_t] = \mathbf{A}_i \hat{\mathbf{x}}_i[k|k_t] + \mathbf{B}_i \hat{\mathbf{u}}_i[k|k_t]. \tag{3.9}$$

Using (3.9) we can represent the (stacked) predicted state sequence over the horizon, $\mathbf{X}_i \in \mathbb{R}^{6K}$, as

$$\mathbf{X}_i = \mathbf{A}_{0,i} \bar{\mathbf{x}}_i[k_t] + \mathbf{\Lambda}_i \mathbf{U}_i, \tag{3.10}$$

where $\mathbf{U}_i \in \mathbb{R}^{3K}$ is the stacked input sequence, $\bar{\mathbf{x}}_i[k_t]$ is the measured state at time step k_t, and $\mathbf{\Lambda}_i \in \mathbb{R}^{3K \times 3K}$ is defined as

$$\mathbf{\Lambda}_i = \begin{bmatrix} \mathbf{B}_i & \mathbf{0}_3 & \ldots & \mathbf{0}_3 \\ \mathbf{A}_i\mathbf{B}_i & \mathbf{B}_i & \ldots & \mathbf{0}_3 \\ \vdots & \ddots & \ddots & \vdots \\ \mathbf{A}_i^{K-1}\mathbf{B}_i & \mathbf{A}_i^{K-2}\mathbf{B}_i & \ldots & \mathbf{B}_i \end{bmatrix}. \tag{3.11}$$

Lastly, the matrix $\mathbf{A}_{0,i} \in \mathbb{R}^{6K \times 6}$ is defined as

$$\mathbf{A}_{0,i} = \begin{bmatrix} (\mathbf{A}_i)^\intercal & (\mathbf{A}_i^2)^\intercal & \ldots & (\mathbf{A}_i^K)^\intercal \end{bmatrix}^\intercal. \tag{3.12}$$

We note that $\mathbf{U}_i$ is a *sampled* representation of the input, and it can be obtained from a linear combination of the control points of a *continuous* Bézier curve. We define $\boldsymbol{\mathcal{U}}_i \in \mathbb{R}^{3l(p+1)}$ as the decision vector of the optimization, which represents the control points of the l Bézier curves of degree p.

3.3.3 Input Continuity

Trajectory smoothness is enforced through equality constraints. First, the initial control point of the input is chosen to be equal to a constant vector; the way this constant vector is constructed is the subject of Sec. 3.4. Second, continuity between the l Bézier curves is guaranteed up to a certain derivative by forcing the endpoint of a curve to match the beginning of the next curve, i.e., the difference between control points must be equal to zero [29].

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

3.3.4 Dynamic Feasibility

Since the agents have limited actuation and the environment may have limited dimensions, we must encode such limitations within the optimization. It is natural to pose these constraints as inequalities that must be satisfied at every input update. For dynamic feasibility we impose the following constraints

$$\boldsymbol{\gamma}_{\min}^{(c)} \leq \frac{d^c}{dt^c}\hat{\mathbf{u}}_i(t) \leq \boldsymbol{\gamma}_{\max}^{(c)}, \quad c = \{0, 1, \ldots, r\}, \tag{3.13}$$

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

One option proposed in the literature to implement these constraints is to exploit the convex hull property of Bézier curves. If we limit the control points of the curve to lie within a convex region, the curve will be entirely contained within that region. This may, however, impose overly conservative bounds [38]. A second option, as suggested in [29], is to not impose the constraints at all and check afterwards for dynamic feasibility; if it does not, the problem needs to be resolved one more time to guarantee constraint satisfaction. In this work we propose a third alternative, in which we leverage the derivations in Sec. 3.3.1 to obtain samples of the input and its derivatives, and limit those appropriately through linear inequality constraints of the form $\mathbf{A}_{\text{in}}\boldsymbol{\mathcal{U}}_i = \mathbf{b}_{\text{in}}$. This method avoids the conservativeness of using the convex hull property (since we apply the constraints over the actual curve) and the potential need to resolve the problem as in [29].

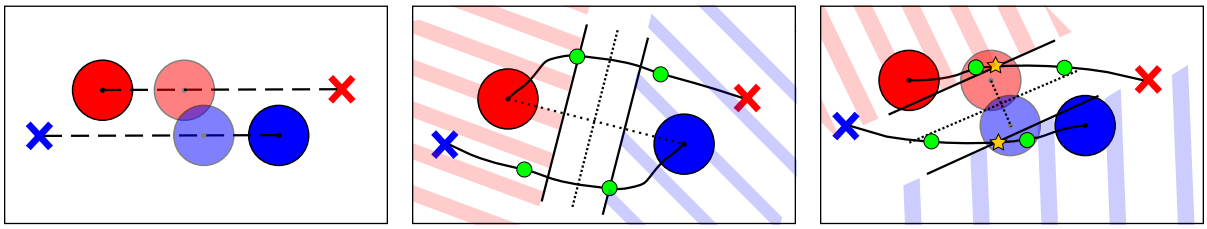
3.3.5 Optimization-Based Collision Avoidance

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

$$\left\|\boldsymbol{\Theta}^{-1}(\mathbf{p}_i[k_t] - \mathbf{p}_j[k_t])\right\|_2 \geq r_{\min}, \quad \forall j \neq i, \tag{3.14}$$

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

We explored two approaches: buffered Voronoi cells (BVC) [28, 29] and on-demand collision avoidance [2]. Both methods rely on the same principle of imposing hyperplane constraints that limit the available free space over which the agent is allowed to optimize its future inputs. In Fig. 3.3a we present a simple collision avoidance scenario with two agents in 2D. Would the agents continue on their intended trajectories, they would collide at a certain timestep indicated by the translucent circles.



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

Figure 3.3: Two-agent transition scenario in 2D. The agents are represented by a circle of radius $r_{\min}/2$. The X marks the intended goal of each agent. In (a) the dashed lines represent the nominal (colliding) trajectories, where the translucent circles represent the position of each agent at time step $k_{c,i}$ in which the first collision is predicted. In (b) we show the input update using the BVC method. The green dots represent the concatenation points of the Bézier curves. The first segment is constrained to lie within the coloured zone for each agent. In (c) the agents update their inputs using on-demand collision avoidance. The star represents the sample of the input constrained to be within the coloured zone.

Buffered Voronoi Cells

In the BVC method, the agents are restricted to remain within their own Voronoi cell, $\mathcal{V}_i$, for a time τ of their horizon. In this work, we define a Buffered Voronoi Cell similar to [28] but including the scaling matrix:

$$\mathcal{V}_i = \left\{ \mathbf{p} \in \mathbb{R}^3 \;\middle|\; \frac{\boldsymbol{\Theta}^{-2}(\mathbf{p}_i - \mathbf{p}_j)^\intercal(\mathbf{p} - \mathbf{p}_i)}{d_{i,j}} \geq \frac{r_{\min} - d_{i,j}}{2} \right\}, \forall j \neq i, \tag{3.15}$$

where $d_{i,j} = \left\| \boldsymbol{\Theta}^{-1}(\mathbf{p}_i - \mathbf{p}_j) \right\|_2$, and $\mathbf{p}_i$, $\mathbf{p}_j$ are the measured positions of agents i and j at time step k_t. Fig. 3.3b shows the BVCs calculated (shaded areas) for our two-agent example. The condition in (3.15) defines a linear constraint in the position of the agents to achieve collision avoidance. Let $\mathcal{P}_{i,1}$ be the set of control points of agent i corresponding to the first Bézier curve of the input. To achieve collision avoidance we impose the constraint $\mathcal{P}_{i,1} \in \mathcal{V}_i$, which translates to $p+1$ constraints on the control points. This constraint exploits the convex hull property of Bézier curves, which guarantees that the first segment of the input will lie within $\mathcal{V}_i$ if the optimization problem is feasible. Collision-free updates are achieved with this method, as shown in Fig. 3.3b.

On-demand Collision Avoidance

On the other hand, the on-demand collision avoidance methodology presented in Sec. 2.3.6 relies on a predict-avoid paradigm for collision avoidance. It assumes communicative

agents that share with the team a representation of their future actions. In this case, since we are considering the tracking dynamics into our formulation, we have two options for collision avoidance:

- **State space**: constraints are imposed on the predicted states $\mathbf{X}_i$ of the agents, which can be obtained as a linear combination of the optimal inputs using (3.10). This results in collision-free *predicted* positions over the horizon.

- **Input space:** constraints are imposed on the inputs $\mathbf{U}_i$ directly, resulting in collision-free *reference* positions over the horizon.

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

$$\xi_{ij} = \left\| \boldsymbol{\Theta}^{-1} \left(\hat{\mathbf{p}}_i[k_{c,i}|k_t - 1] - \hat{\mathbf{p}}_j[k_{c,i}|k_t - 1] \right) \right\|_2 \geq r_{\min}, \tag{3.16}$$

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

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

where $g(r_{\min})$ models the area around the agent for which collision avoidance is required. In this work we used $g(r_{\min}) = 2r_{\min}$.

Leveraging this information, the agents predict future collisions (in the input or state space, depending on which modality we use) and include separating hyperplane constraints on the first time step with a predicted collision (marked with a yellow star in Fig. 3.3c). Let $k_{c,i}$ be the time step in which agent i predicts the first collision with a neighbour j (translucent circles in Fig. 3.3c). We can procure collision avoidance in the state space by enforcing a first-order approximation of the constraint

$$\left\| \boldsymbol{\Theta}^{-1} \left(\hat{\mathbf{p}}_i[k_{c,i} - 1|k_t] - \hat{\mathbf{p}}_j[k_{c,i}|k_t - 1] \right) \right\|_2 \geq r_{\min} + \varepsilon_{ij},\ \forall j \in \Omega_i, \tag{3.17}$$

where $\varepsilon_{ij} < 0$ are slack decision variables that relax the constraints (refer to Sec. 2.3.6 for details on the linearization). A similar constraint can be used for collision avoidance in the input space:

$$\left\| \boldsymbol{\Theta}^{-1} \left(\hat{\mathbf{u}}_i[k_{c,i} - 1|k_t] - \hat{\mathbf{u}}_j[k_{c,i}|k_t - 1] \right) \right\|_2 \geq r_{\min} + \varepsilon_{ij},\ \forall j \in \Omega_i. \tag{3.18}$$

Note that on-demand avoidance only constrains a specific sample of the curve to lie within a partition of the space, whereas BVC constrains a complete segment of the curve. Comparing the resulting trajectories in Fig. 3.3b and Fig. 3.3c, it is clear that on-demand avoidance leads to less conservative maneuvers than the BVC method. In Sec. 3.6 we analyze how these insights impact the ability to complete multi-agent transition tasks.

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

3.3.6 Cost Function

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

Error to goal

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

$$\mathcal{J}_{i,\text{error}} = \sum_{k=K-\kappa}^{K} q_k \left\| \hat{\mathbf{p}}_i[k|k_t] - \mathbf{p}_{d,i} \right\|_2^2, \tag{3.19}$$

where $q_k > 0$ are the positive weights of each time step.

Energy

We minimize a weighted combination of the sum of squared derivatives, as in [13, 39]. The cost is defined as

$$\mathcal{J}_{i,\text{energy}} = \sum_{c=0}^{r} \alpha_c \int_0^{t_h} \left\| \frac{d^c}{dt^c} \hat{\mathbf{u}}_i(t) \right\|_2^2 dt, \tag{3.20}$$

where $\alpha_c > 0$ is a scalar weight for each derivative of the input, until the r^{th} derivative. This term can be evaluated in closed form to get a quadratic form in terms of $\boldsymbol{\mathcal{U}}_i$ [39].

Collision constraint violation

We implement on-demand collision avoidance as soft constraints, which requires a penalty term to be added in the cost function to limit the amount of relaxation of the constraints.

For that we consider both quadratic and linear penalty costs

$$\mathcal{J}_{i,\text{violation}} = \zeta \left\|\varepsilon_{ij}\right\|_2^2 + \xi\varepsilon_{ij}, \tag{3.21}$$

where ζ and ξ are the weights of each term.

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

All the element previously mentioned compose the following standard QP problem, for which efficient solvers exist:

$$\begin{aligned} \underset{\boldsymbol{\mathcal{U}}_i,\, \varepsilon_{ij}}{\text{minimize}} \quad & \mathcal{J}_{i,\text{error}} + \mathcal{J}_{i,\text{energy}} + \mathcal{J}_{i,\text{violation}} \\ \text{subject to} \quad & \mathbf{A}_{\text{eq}}\boldsymbol{\mathcal{U}}_i = \mathbf{b}_{\text{eq}}, \\ & \mathbf{A}_{\text{in}}\boldsymbol{\mathcal{U}}_i \leq \mathbf{b}_{\text{in}}, \\ & \mathbf{A}_{\text{coll}}\boldsymbol{\mathcal{U}}_i \leq \mathbf{b}_{\text{coll}}, \\ & \varepsilon_{ij} \leq 0 \quad \forall j \in \Omega_i. \end{aligned} \tag{3.22}$$

3.4 Event-Triggered Replanning

Choosing the initial condition for the input to be equal to the current state of the robot was proposed in [29], but it has certain limitations. First, if we require C^r-continuity on the inputs, then we need to reliably measure the r^{th} derivative of the robot's position. Second, for imperfect trajectory tracking or systems with slow dynamics, this replanning strategy constantly causes (potentially big) discontinuities of the input to match the state of the robot, as shown in Fig. 3.4. Such discontinuities cause undesired jittering in the robot and slow down its progress to complete the task.

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

$$f_n[k_t] = \frac{(\mathrm{p}_{i,n}[k_t] - \mathrm{u}_{i,n}[k_t])^5}{-(\mathrm{v}_{i,n}[k_t] + sgn(\mathrm{v}_{i,n}[k_t])\varepsilon)}, \quad n = 1, 2, 3 \tag{3.23}$$

where the subscript n represents the spatial component ($[x, y, z]$) of the vectors associated with agent i. The term $(\mathrm{p}_{i,n}[k] - \mathrm{u}_{i,n}[k])$ is the trajectory tracking error, and the term

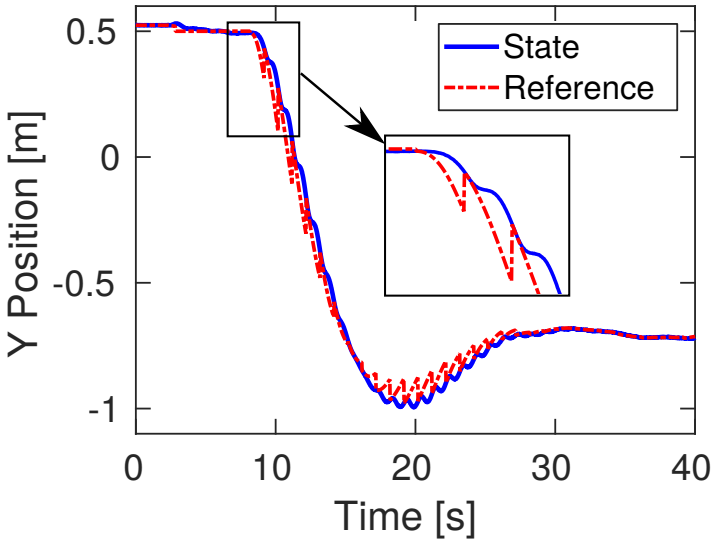


Figure 3.4: Experimental data of a quadrotor flight when using online trajectory generation based on DMPC [2] with replanning every second. The discontinuities in the reference signal causes undesired behaviour.

$sgn(\mathrm{v}_{i,n}[k])\varepsilon$ with a small scalar $\varepsilon \ll 1$ is used to avoid singularities in $f_n[k]$. We assume $|\mathrm{v}_{i,n}[k]| > 0$, which is realistic in real-world operation due to noise in state estimation.

The intuition behind (3.23) is that we want to reset our reference signal whenever the tracking error grows large. However, designing an appropriate threshold value for the tracking error is tricky due to its high variability during execution. Instead, $f_n[k]$ is designed to detect whenever the error is growing but the velocity is either small or growing in the opposite direction of the error. To detect these scenarios, we define the robot is operating normally if the inequality

$$f_{\min} < f_n[k] < f_{\max} \tag{3.24}$$

holds for every element of $f_n[k]$. The values of $f_{\min}$ and $f_{\max}$ must be chosen by extracting the extrema of $f_n[k]$ under normal operation. If (3.24) does not hold, then the agent is being disturbed and we set the initial position and velocity of the Bézier curve to match the states of the vehicle, while setting higher-order derivatives to zero. In summary,

$$\mathbf{u}_{0,i}[k_t] = \begin{cases} \hat{\mathbf{u}}_i[1|k_t-1], & \text{if } f_{\min} < f_n[k] < f_{\max}. \\ (\mathbf{x}_i[k_t], \mathbf{0}), & \text{else} \end{cases} \tag{3.25}$$

In order to validate the proposed replanning strategy, we conducted an experiment with our quadrotor platform while a human operator perturbed it along its path. The

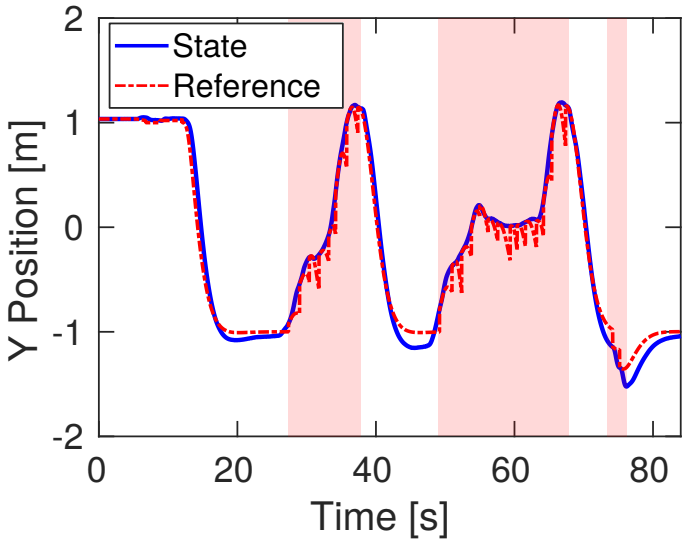


Figure 3.5: Experimental data of a quadrotor using event-triggered replanning with the activation function in (3.23). The robot was perturbed by a human during the highlighted segments in red.

task of the quadrotor was to reach a y-coordinate of -1 m. The reference signal and state of the quadrotor are shown in Fig. 3.5, where the red segments mean the agent was being disturbed. During these disturbed stages we observe how the reference signal is constantly reset to match the state of the robot. The replanning helps the quadrotor continue its task whenever it stops being disturbed. Under normal operation (white segments) the replanning is not required, which leads to a smooth reference signal that avoids the shortcomings observed in Fig. 3.4.

3.5 Algorithm

In this section we describe the core algorithm used to update the optimal input sequence for all the agents, outlined in Alg. 2. As stated, the algorithm is conceived to be executed from a single offboard computer, which then communicates the commands to each agent. It takes as inputs the measured state of each agent, the desired locations and the previously computed prediction horizons of each agent. For execution we consider two different time bases: one with a coarse time step h, used for the MPC planning, and one with a refined time step T_s used for commanding the agents at a higher rate. With this definition, the output of Algorithm 1 is the set of inputs for each agent in the time frame in-between planning cycles, i.e., $t \in [t_0, t_0 + h]$ with sample rate T_s. This is equivalent to obtaining subsamples of the input between $\hat{\mathbf{u}}_i[0|k_t]$ and $\hat{\mathbf{u}}_i[1|k_t]$.

Algorithm 2: Multi-agent Online Planner Input Updates

Input : Current states of all agents ($\mathbf{x}[k_t]$), target locations ($\mathbf{p}_d$), prediction horizon of all agents ($\mathbf{\Pi}[k_t - 1]$)

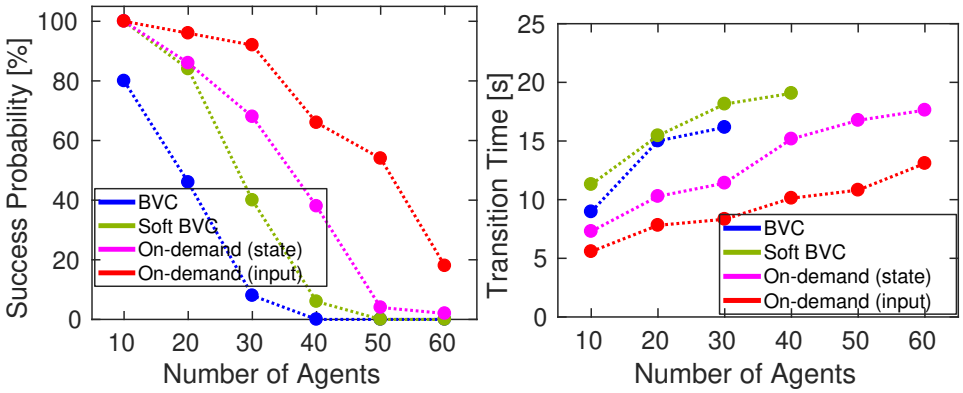
Output: Commands to be applied from t_0 to $t_0 + h$ with sampling of T_s ($\bar{\mathbf{u}}$)

17 `setTargetLocations` ($\mathbf{p}_d$)
18 **foreach** *agent* $i = 1, ..., N$ **do**
19 $\quad \mathbf{u}_{0,i}[k_t] \leftarrow$ `getInitRef` ($\mathbf{x}_i[k_t]$, $\hat{\mathbf{u}}_i[1|k_t - 1]$)
20 $\quad (\mathbf{A}_{\text{coll}}, \mathbf{b}_{\text{coll}}) \leftarrow$ `getCollision` ($\mathbf{x}[k_t]$, $\mathbf{\Pi}[k_t - 1]$)
21 $\quad$ QP $\leftarrow$ `buildQP` ($\mathbf{A}_{\text{coll}}$, $\mathbf{b}_{\text{coll}}$, $\mathbf{u}_{0,i}[k_t]$, $\mathbf{x}_i[k_t]$)
22 $\quad \boldsymbol{\mathcal{U}}_i \leftarrow$ `solve` (QP)
23 $\quad \mathbf{\Pi}_i[k_t] \leftarrow$ `updateHorizon` ($\boldsymbol{\mathcal{U}}_i$, $\mathbf{x}_i[k_t]$)
24 $\quad \hat{\mathbf{u}}_i[1|k_t] \leftarrow$ `updateInitialReference` ($\boldsymbol{\mathcal{U}}_i$)
25 $\quad \bar{\mathbf{u}}_i \leftarrow$ `getSampledInput` ($\boldsymbol{\mathcal{U}}_i$)
26 **return** $\bar{\mathbf{u}}$

In line 1 we build the error penalties given by (3.19), which is only required if the setpoint $\mathbf{p}_{d,i}$ of the agents change. The loop in lines 2-9 updates the input sequence of each agent, and can be executed in parallel since there are no data dependencies between the individual optimization problem of each agent. First, in line 3 we apply the event-triggered replanning strategy to decide the value of the initial condition of the input. The collision avoidance constraint (either BVC or on-demand) is constructed in line 4. Note that BVC would not require the prediction information, but instead would require the measured state of the agents. Conversely, on-demand avoidance only requires the predictions and not the measured states. Lines 5-6 build and solve the standard quadratic programming problem outlined in (3.22). Once the solution vector $\boldsymbol{\mathcal{U}}_i$ is obtained from the QP solver, we can then sample the resulting Bezier curves to obtain a sampled representation of the input (or the state). Note that the updated information is *not* used by subsequent agents, which allows for parallelization of the input updates of each agent. Line 8 updates the initial condition of the reference to be used in the next planning cycle, in the case where replanning is not required. Lastly, line 9 samples the resulting Bezier curve with period T_s to obtain the sequence to be applied for $t \in [t_0, t_0 + h]$.

3.6 Simulation Results

We created a simulation environment in MATLAB 2017a and executed on a PC with Intel Xeon CPU with 8 cores and 16 GB of RAM, running at 3 GHz. The agents were modeled after the Crazyflie 2.0 quadrotor, using $r_{\min} = 0.3\,\text{m}$ and $\mathbf{\Theta} = \text{diag}([1, 1, 2])$, but



(a) (b)

Figure 3.6: Simulation performance comparison of various collision avoidance strategies. We considered different numbers of agents in a fixed volume of 18 m^3. For each swarm size, 50 different random test cases were generated and the results were averaged.

a collision was declared using $r_{\text{coll}} = 0.2\,\text{m}$ (closer to the physical size of the quadrotor) and $\mathbf{\Theta}_{\text{coll}} = \text{diag}([1, 1, 2.25])$. The trajectory tracking dynamics were identified by fitting a second-order model to experimental data from the step response of the system depicted in Fig. 3.2. We selected a step of $h = 0.2\,\text{s}$, which means that trajectories are replanned at 5 Hz.

For the input sequence we chose Bézier curves with $p = 5$, $l = 3$ and $t_h = 3\,\text{s}$, where each segment had a fixed duration of 1 second. Additionally, we imposed actuation limits with $\gamma^{(2)}_{\max} = -\gamma^{(2)}_{\min} = 1\,\text{m/s}^2$. After tuning the cost function, we selected $\kappa = 3$, $q_k = 100$, $\alpha_2 = 0.008$, $\zeta = 1$ and $\xi = -5 \times 10^4$. For the replanning function in (3.23) we chose $\epsilon = 0.01$, $f_{\min} = -0.01$, and $f_{\max} = 0.8$. Additionally, we added noise in the measured state $\bar{\mathbf{x}}_i[k_t]$ based on empirical data gathered from an overhead motion capture system, to simulate a more realistic scenario.

3.6.1 Comparison of Collision Avoidance Methods

We compared four different optimization-based collision avoidance methods in random transition scenarios: 1) BVC as proposed in [29] (without the discrete planner component), 2) BVC using soft constraints, 3) On-demand collision avoidance applied in the state space and 4) On-demand collision avoidance in the input space.

We considered a fixed-volume, obstacle-free workspace of 18 m^3 (roughly the size of our indoor flight arena), with randomly generated initial and final locations for all agents. The number of agents varied from 10 to 60, in order to test the algorithms as the agent density increased. A trial was considered successful if all agents were able to reach their goals without collisions and within 20 seconds. More specifically, after each simulation

we ran a collision check (using r_{coll} and $\mathbf{\Theta}_{\text{coll}}$) and a goal check (allowing 10 cm distance from the target location) to determine if the test was successful.

In Fig. 3.6 we show the performance obtained using each method. The success probability for each swarm size considered is highlighted in Fig. 3.6a. We notice that as the number of agents increases (ergo, a denser workspace) the effectiveness of the BVC methods decay drastically. Using soft constraints helps to some extent, but ultimately the approach is too conservative to resolve transition scenarios with a high density of agents.

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

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

In the same order of ideas, Fig. 3.6b shows that, on average, using on-demand collision avoidance leads to faster transition times than the BVC methods, averaging around 50% transition time reductions. These numbers match the analysis made on Sec. 3.3.5 using Fig. 3.3.

3.6.2 Runtime Benchmark

We compared the computation time per agent to update their input sequence. In Fig. 3.7 the results are presented, where we specifically show the average time per agent to solve the associated QP problem.

To formally analyze the scaling of both algorithms, define N_{i,k_t} to be the number of nearby neighbours of agent i to be considered for collision avoidance at time step k_t. The amount of inequality constraints on both BVC and on-demand methods scale with $\mathcal{O}(N_{i,k_t})$. The soft BVC method adds additional $(p+1) \times N_{i,k_t}$ slack variables to relax

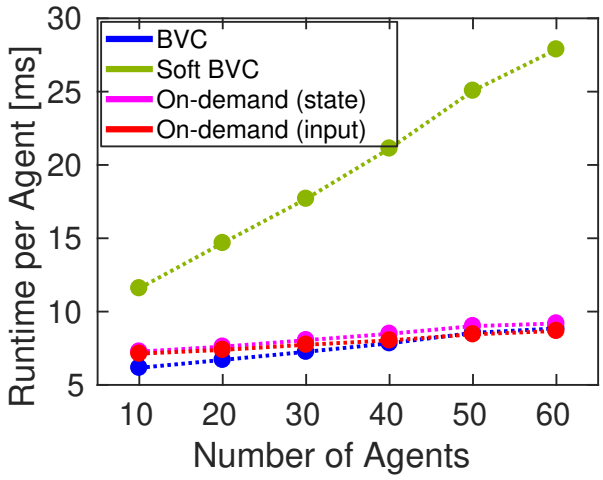


Figure 3.7: Comparison of the average runtime per agent to update the inputs using our on-demand collision avoidance and the BVC method. The data shown is the average over 50 randomly generated tests for each swarm size considered.

the constraints, while the on-demand methods add only N_{i,k_t} new decision variables to the problem. In Fig. 3.7 we observe the empirical runtime of the considered methods. The soft BVC method incurs in the slowest runtime, due to the added slack variables and overall bigger problems to be solved. For the other three methods the runtime is fairly similar, with a slight advantage to the BVC method.

3.7 Experimental Results

The online generation method outlined in this paper (with on-demand avoidance) was implemented in C++, using ROS to manage the drone swarm and qpOASES [40] as the QP solver. We parallelized the input updates for the agents by dividing them in clusters that were solved in separate CPUs of the host machine [2]. In this section we provide experimental results using our Crazyflie 2.0 swarm testbed. All the inputs were computed from a single computer and broadcasted to the swarm through a radiolink, alongside the estimated position of each individual agent given by a motion capture system. The computer specs and algorithm parameters are the same as Sec. 3.6, with the exception of $\xi = -1 \times 10^3$ and the addition of $T_s = 0.05\,\text{s}$, meaning that trajectories were being sent to the swarm at 20 Hz.

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

Table 3.1: Experimental results summary for random transition tasks involving increasing number of agents.

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

3.7.1 Obstacle-Free Transitions

The method was tested in several randomly generated transition tasks in an indoor flight arena. We considered different swarm sizes, ranging from 2 to 20 drones. For each swarm size, three independent flights were executed, where each flight consisted of 30 seconds of randomly generated transitions (the setpoint of each agent was changed every 5 seconds to obtain a continuous movement of the swarm). The agents were restricted to move in a $3 \times 3 \times 2\,\mathrm{m}^3$ volume, and we used $r_{\min} = 0.35\,\mathrm{m}$ as the safety distance during planning.

For each test, we recorded the average time required to update the inputs of all the agents of the swarm, as well as the minimum inter-agent distance during the flight. The results are summarized in Tab. 3.1. As expected, the average computation time increases as we add more agents to the problem, since all computations are being executed by a single computer. We should note here that there are other sources of overhead in the system when adding more drones: state estimation and communication, for instance. The interesting result is that the scaling we obtain in runtime is better than linear, since we are able to parallelize the computation due to the distributed nature of the approach. Also noteworthy is that the standard deviation of the computation increases with the number of vehicles; there is a significant variability in the sizes and complexity of the QPs being solved for each agent, depending on how close they are to other agents and to the boundaries of the workspace.

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

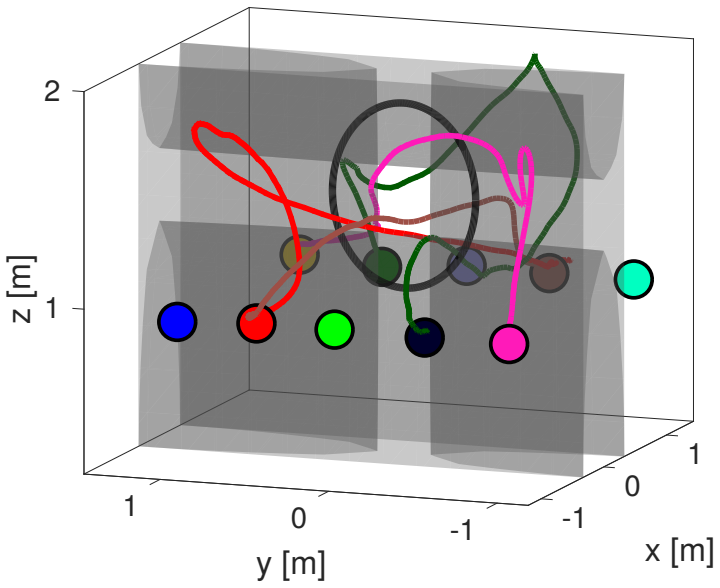


Figure 3.8: A 10-drone transition scenario passing through a hula-hoop (denoted by the black circle). The forbidden space is denoted by four ellipsoids acting as static obstacles. The coloured dots denote the initial locations of the agents, and the corresponding coloured lines are the followed trajectories towards the goal (only 4 showing for clarity).

3.7.2 Transition Tasks With Static Obstacles

To showcase the versatility of our approach to solve complex transition scenarios in a workspace with static obstacles, we tasked a group of drones to exchange positions with each other by passing through a hula-hoop with a 85 cm diameter. The environment was divided by an invisible wall with a passage-way defined by the hula-hoop. In Fig. 3.8 we show the 10-drone transition scenario solved in experiments. Note that static obstacles are added to the problem as new "neighbours" for each agent, with their own r_{obs} and $\boldsymbol{\Theta}_{\text{obs}}$, which means that the runtime complexity scales linearly with the number of static obstacles.

The restricted zone was modeled as the union of four ellipsoids. They are shaped in such a way that they are intersecting and provide a small gap of 30 x 30 cm for the agents to pass through. With this window size, at most two Crazyflie quadrotors were able to pass at the same time through the opening. In the tracked trajectories we observe that some of the agents were able to fly directly through the circle, while others took detours in order to let other agents pass first. This phenomenon is quite interesting since there is no explicit priority encoded in the agents, but collaboration still arises from within the distributed optimization. The distance-to-goal envelope shown in Fig. 3.9 demonstrates how the agents make progress over time to decrease the distance

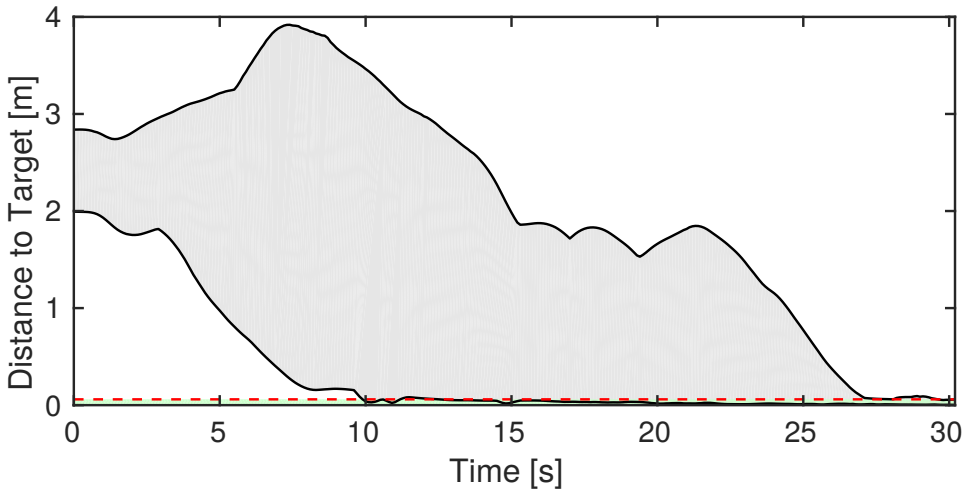


Figure 3.9: Distance to target envelope (minimum and maximum over time) of the 10-drone hula-hoop transition task. The light green section represents the zone where transition success is declared: a 6 cm radius of the target location. In this case the transition was completed in $T_f = 28\,\mathrm{s}$.

towards their goal, eventually converging to it within some tolerance region. Note that, in general, the envelope is not monotonically decreasing, which means that oftentimes the agents deviate from the direct path to the goal in order to avoid collisions.

Several different tuning parameters were tried while solving this particular task. While using input space collision avoidance, a wide range of penalty gains and maximum accelerations worked well to solve the task; the completion time varied from 20.1 to 48.4 seconds in 18 different trials. On the other hand, the success rate using state space collision avoidance was much lower. The agents tended to have oscillatory behaviour when approaching the obstacles and were able to complete the transition only a fraction of the trials. One hypothesis is that adding the tracking dynamics into the collision avoidance constraints leads to less preemptive maneuvers. Since in our case the state of the agents is basically a delayed version of the input, then the optimizer naturally delays the evasive maneuvers, which ends up being counterproductive towards efficiently completing the transition.

3.8 Summary

In this chapter we presented a framework for multi-robot online trajectory generation based on distributed model predictive control (DMPC). In transition tasks, our method has a higher success rate and lower travel times than using the Buffered Voronoi Cells

method. The simulations indicated more than 90% success rate with up to 30 palm-sized quadrotor agents in a 18 m^3 arena.

The parallelization of the method and its formulation as a simple Quadratic Program (QP) leads to high scalability. In experiments we were able to generate trajectories in real-time (20 Hz) for a swarm of 20 drones, where all computations regarding the motion planning were executed from a single off-board computer.

Aside from testing thoroughly with swarms of varying sizes in obstacle-free environments, our approach showed satisfactory results in a complicated transition scenario passing through a hula-hoop and robust replanning in the presence of unmodeled disturbances.

Chapter 4

Conclusions and Future Work

4.1 Summary of Contributions

As a first contribution, a multiagent offline trajectory generation framework was presented in Ch. 2. Using concepts from distributed receding horizon control, the proposed algorithm produces smooth and collision-free motion plans that can be easily tracked by off-the-shelf controllers. On-demand collision avoidance was introduced as an efficient way of including collision constraints in a distributed way, achieving convincing collision avoidance while not limiting the motion of the agents. When compared to similar optimization-based approaches, our method achieves a significant (85%) runtime reduction while maintaining a high success rate. To the best of our knowledge, at the time of publication of this thesis, our framework is the only that has demonstrated trajectory generation for swarms of 25 quadrotors with runtimes below 2 seconds. The contents of this chapter were published in [2].

In Ch. 3 we adapted the offline framework presented in Ch. 2 for real-time trajectory generation, which is the second main contribution of this thesis. By combining the concepts of Bezier curves and on-demand collision avoidance, our method is able to produce smooth motion plans that are collision-free, while not being too conservative. Event-triggered replanning was introduced to overcome jittering effects incurred by the constant replanning of trajectories. A comparison with the Buffered Voronoi Cells method shows that our method has a higher success rate in random transition tasks, while keeping real-time tractability. Experiments proved that our method is able to resolve complicated transition scenarios, such as groups of quadrotors passing through a hula-hoop. The scalability test shows that the runtime of updating the inputs for the robot teams scales sub-linearly with the number of agents. The algorithm scalability was proven flying a team of 20 quadrotors, where all the motion plans where computed from

a single off-board computer.

4.2 Future Work

A possible extension for both offline and online trajectory generation is to add GPU-accelerated computations, as initially proposed in [14].

On-demand collision avoidance has proven to be effective empirically, by testing several random transition scenarios. A future research direction would be a rigorous study to determine the best time step(s) to impose collision constraints. In this work we considered the first time step with collisions to be the most safety critical and thus the one to be included in the optimization. While it has an intuitive explanation, at any given point it may be more desirable to include multiple time steps into the optimization.

In the results presented, the robot-to-robot communication was considered to be instantaneous and without loss of information. A more realistic scenario where the robots are equipped with a communication network will require to model more accurately their communication channel. Studying the impact of delays and lack of information is an interesting future research direction. The work in [7] considered real communication between robots and it may serve as inspiration.

An interesting research direction is to use trajectory prediction of nearby agents whenever communication is lost or unreliable. That is, each agent predicts a possible trajectory of the dynamic objects within its sensing range, which then informs the motion planning algorithm. We refer the reader to [41] for a recent survey on motion prediction. In these order of ideas, probabilistic optimization frameworks are well-suited to include uncertainties about neighbouring agents positions and/or intentions [30].

This thesis presents interesting future venues for multi-robot applications. From drone entertainment shows to multi-drone cooperative sensing, the algorithms presented in this work may serve as the low-level motion planning framework required for more complicated coordination tasks.

Even though we have shown real-time motion planning is currently possible for several quadrotors, the implementation in more realistic scenarios can be challenging. The motion planning algorithms will have to evolve to account for local sensing of static/dynamic obstacles and local communication (as opposed to centralizing the communications on an off-board computer). These considerations will likely result in more complicated optimization problems to be solved, but hopefully the advances in computing power and optimization solvers efficiency will still allow for real-time implementations.

Bibliography

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

[2] C. E. Luis and A. P. 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.

[3] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," *ACM SIGGRAPH computer graphics*, vol. 21, no. 4, pp. 25–34, 1987.

[4] K. Oh, M. Park, and H. Ahn, "A survey of multi-agent formation control," *Automatica*, vol. 53, pp. 424–440, 2015.

[5] E. Guizzo, "Three engineers, hundreds of robots, one warehouse," *IEEE Spectrum*, vol. 45, no. 7, pp. 26–34, 2008.

[6] T. Schouwenaars, B. De Moor, E. Feron, and J. How, "Mixed integer programming for multi-vehicle path planning," in *European Control Conference (ECC)*, 2001, pp. 2603–2608.

[7] M. Turpin, N. Michael, and V. Kumar, "Decentralized formation control with variable shapes for aerial robots," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2012, pp. 23–30.

[8] S. Boyd, "Sequential convex programming," *Lecture Notes, Stanford University*, 2008.

[9] Y. Chen, M. Cutler, and J. P. How, "Decoupled multiagent path planning via incremental sequential convex programming," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 5954–5961.

[10] 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 Robotics and Automation Letters*, vol. 3, no. 2, pp. 1215–1222, 2018.

[11] J. A. Preiss, W. Hönig, N. Ayanian, and G. S. Sukhatme, "Downwash-aware trajectory planning for large quadrotor teams," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 250–257.

[12] M. Čáp, P. Novák, J. Vokrínek, and M. Pěchouček, "Multi-agent rrt: sampling-based cooperative pathfinding," in *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2013, pp. 1263–1264.

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

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

[15] S. Tang and V. Kumar, "A complete algorithm for generating safe trajectories for multi-robot teams," in *Robotics Research*. Springer, 2018, pp. 599–616.

[16] L. Wang, A. D. Ames, and M. Egerstedt, "Safety barrier certificates for collisions-free multirobot systems," *IEEE Transactions on Robotics*, 2017.

[17] 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. MIT Press, 2011.

[18] J. Van Den Berg, J. Snape, S. J. Guy, and D. Manocha, "Reciprocal collision avoidance with acceleration-velocity obstacles," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2011, pp. 3475–3482.

[19] 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*. Springer, 2013, pp. 203–216.

[20] H. Rezaee and F. Abdollahi, "A decentralized cooperative control scheme with obstacle avoidance for a team of mobile robots," *IEEE Transactions on Industrial Electronics*, vol. 61, no. 1, pp. 347–354, 2014.

[21] E. Camponogara, D. Jia, B. H. Krogh, and S. Talukdar, "Distributed model predictive control," *IEEE Control Systems*, vol. 22, no. 1, pp. 44–52, 2002.

[22] R. Van Parys and G. Pipeleers, "Distributed model predictive formation control with inter-vehicle collision avoidance," in *Asian Control Conference (ACC)*, 2017.

[23] H. Sayyaadi and A. Soltani, "Decentralized polynomial trajectory generation for flight formation of quadrotors," *Proceedings of the Institution of Mechanical Engineers, Part K: Journal of Multi-body Dynamics*, vol. 231, no. 4, pp. 690–707, 2017.

[24] L. Dai, Q. Cao, Y. Xia, and Y. Gao, "Distributed mpc for formation of multi-agent systems with collision avoidance and obstacle avoidance," *Journal of the Franklin Institute*, vol. 354, no. 4, pp. 2068–2085, 2017.

[25] P. Wang and B. Ding, "A synthesis approach of distributed model predictive control for homogeneous multi-agent system with collision avoidance," *International Journal of Control*, vol. 87, no. 1, pp. 52–63, 2014.

[26] 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.

[27] J. Alonso-Mora, P. Beardsley, and R. Siegwart, "Cooperative collision avoidance for nonholonomic robots," *IEEE Transactions on Robotics*, vol. 34, no. 2, pp. 404–420, 2018.

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

[29] B. Şenbaşlar, W. Hönig, and N. Ayanian, "Robust trajectory execution for multi-robot teams using distributed real-time replanning," in *Distributed Autonomous Robotic Systems*. Springer, 2019, pp. 167–181.

[30] H. Zhu and J. Alonso-Mora, "Chance-constrained collision avoidance for mavs in dynamic environments," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 776–783, 2019.

[31] B. Hernandez and P. Trodden, "Distributed model predictive control using a chain of tubes," in *UKACC 11th International Conference on Control (CONTROL)*, 2016, pp. 1–6.

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

[33] G. Angeris, K. Shah, and M. Schwager, "Fast reciprocal collision avoidance under measurement uncertainty," *arXiv preprint arXiv:1905.12875*, 2019.

[34] C. E. Luis, M. Vukosavljev, and A. P. Schoellig, "Online trajectory generation with distributed model predictive control for multi-robot motion planning," *arXiv preprint arXiv:1909.05150*, 2019.

[35] P. Ru, "Nonlinear model predictive control for cooperative control and estimation," Ph.D. dissertation, 2017.

[36] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2011, pp. 2520–2525.

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

[38] T. Mercy, R. Van Parys, and G. Pipeleers, "Spline-based motion planning for autonomous guided vehicles in a dynamic environment," *IEEE Transactions on Control Systems Technology*, no. 99, pp. 1–8, 2017.

[39] R. Charles, A. Bry, and N. Roy, "Polynomial trajectory planning for quadrotor flight," in *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA*, 2013.

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

[41] A. Rudenko, L. Palmieri, M. Herman, K. M. Kitani, D. M. Gavrila, and K. O. Arras, "Human motion trajectory prediction: A survey," *arXiv preprint arXiv:1905.06113*, 2019.