EPFL

Thèse n° 9542

Model Predictive Control of Aerial Swarms

Présentée le 25 février 2022

Faculté des sciences et techniques de l’ingénieur
Laboratoire de systèmes intelligents
Programme doctoral en robotique, contrôle et systèmes intelligents

pour l’obtention du grade de Docteur ès Sciences

par

Enrica SORIA

Acceptée sur proposition du jury

Prof. P. P. Ramdya, président du jury
Prof. D. Floreano, directeur de thèse
Prof. N. Ayanian, rapporteuse
Prof. G. de Croon, rapporteur
Prof. C. N. Jones, rapporteur

École
polytechnique
fédérale
de Lausanne

2022

To Rosalba and Livio.

Acknowledgments

I am incredibly grateful to my supervisor Prof. Dario Floreano for having offered me the opportunity to do research in such an exciting laboratory and work on a topic that I deeply enjoyed during these years. His support and kind advice throughout these years have been crucial to the achievements of this thesis. I thank the jury members, Prof. Colin Jones, Prof. Guido de Croon, and Prof. Nora Ayanian, for having accepted to review and improve the quality of this work. I equally thank Prof. Pavan Ramdya for having served as the jury president.

I wish to individually thank each of my colleagues who have accompanied me on this journey over these years. Their presence in happy and difficult moments has always been a key motivation for thriving and achieving goals in all circumstances. In particular, I thank Fabian Schilling for having been an extraordinarily supportive office mate and for all the insightful conversations we shared across the years. I thank Vivek Ramachandran, Valentin Wuest, Enrico Ajanic, William Stewart, Euan Judd, Carine Rognon, and Julien Lecoeur for having offered me their helpful point of view on scientific and personal matters on numerous occasions. In particular, I acknowledge Vivek's consistent engagement in improving the atmosphere of our doctoral school and having been cheerful confident. I thank Enrico Ajanic and Kevin Holdcroft for having shared the role of EDRS student representatives during the last two years with seriousness and commitment. I acknowledge the help of postdocs, and among all Fabrizio Schiano, who has encouraged me to improve the quality of my work and have dedicated their time and effort to it. I feel privileged for having shared my time with each and every member of LIS and for having built bonds with them that will survive the time. I also thank Michelle Waelti and Corinne Lebet for their constant help and honorable commitment in dealing with the administrative aspects of the Ph.D.

Some of the work presented in this thesis has been the fruit of a collaboration with Master's students. For this, I particularly thank Victor Delafontaine, Andrea Giordano, Yoann Lapijover, Hugo Birch, and Samuele Lanzanova. I owe my thanks to Mauro Pfister, who has contributed as an intern to perfecting many aspects of this thesis, the hardware setup used for the experiments, and the graphic animations among others.

I would like to thank the open-source community for constantly and freely sharing their work with the world. The democratization of knowledge is one of the most exciting aspects of our era and their effort will never be acknowledged enough.

I feel an immense sense of gratitude towards my family, who has always satisfied my physical and emotional needs without asking for anything in return. Chiefs of the list are my parents Rosalba and Livio. To the first, I owe innumerable hours of a kind care, and to the second an

example of integrity and perseverance. To my brother Alberto, I owe deep gratitude for having shared a happy childhood and adolescence and for his discrete presence through the most joyful and unpleasant moments in life. I thank my uncle Carlo, called ’Pio’. He never loses an occasion for demonstrating his deep affection to my brother and me. I have the immense lack of having four healthy grandparents Remigia, Pierina, Mario, and Giuseppe, who have been my second parents on different occasions and though me life lessons with touch and kindness. Finally, I deeply thank Cesarina for having hosted and taken care of me during my undergraduate studies. Her morning coffee was a good motivation for getting up every day.
Many are the friends who left their sign in my life. To them too, I owe who I am. First, I thank Serena for always being present for me. I also thank Elisa, Pierangelo, Alessandro, Simone, Marco, and Matteo, for the adventures I get to share with them. I thank Giulia, Chiara, and Martina, for being such inspiring friends.
Finally, I owe to Anthony the taste of life. I thank him for offering me love and support every day and for his openness to sharing with me countless personal and professional experiences.

"The important thing is not to stop questioning. Curiosity has its own reason for existence. One cannot help but be in awe when he contemplates the mysteries of eternity, of life, of the marvelous structure of reality. It is enough if one tries merely to comprehend a little of this mystery each day."
- Albert Einstein

Lausanne, February 14, 2022 E. S.

Abstract

Aerial robot swarms can have a large socio-economic impact. They can perform time-critical missions faster than a single robot and access dangerous environments without compromising human safety. However, swarm deployment is often limited to free environments where no obstacle interferes with the robots' flight. Their trajectories are computed before the flight in many applications, and the robots are perceptually blind to one another. In these cases, all decisions are taken by a central computer that is informed about all robots' states, thus increasing their chance of failure in case of signal interruptions, limiting their robustness to failure, and preventing their scalability in size. Drones should have the autonomy to make their own decisions based on local information, and they should integrate a safe obstacle avoidance behavior to offer more versatility for their application in real-world environments.

We propose a predictive approach to swarm control inspired by flocking birds to address these limitations. In particular, we develop methods that enable drones to coordinate their motion by predicting their own future trajectory and that of their neighbors based on the current state information and the knowledge of a model. We first present a centralized predictive algorithm that computes the trajectories of the drones in real-time and improves the synchronization and order of the swarm flight compared to current state-of-the-art algorithms. We show in extensive simulations that our algorithm is robust to different obstacle densities, swarm speeds, and inter-agent distances. Then, we formulate a distributed predictive algorithm with the same qualitative advantages as its centralized counterpart but scalable in the swarm size. We also show that our approach is tolerant to a wide range of sensor noise. We extend and analyze the usage of this algorithm also for pure sensor-based swarms that cannot use explicit communication. Finally, in relation to pure sensor-based swarms, we also analyze the applicability of state-of the art reactive swarm models to drones with limited field of view sensors and the scalability of the same models to large vision-based swarms which account for occlusions. We validate the algorithms developed in this thesis in extensive simulation and in the real world with a fleet of hand-sized quadcopters in controlled indoor environments with obstacles.

Keywords: aerial robot swarms, multi-robot systems, model predictive control, convex optimization, collective motion, flocking algorithms, path planning, genetic optimization, vision-based control

Résumé

Le vol de robots aériens en essaims peut avoir un impact socio-économique considérable. Ces robots peuvent effectuer des missions collectives plus rapidement qu'un seul individu et accéder à des environnements dangereux sans compromettre la sécurité des opérateurs humains. Cependant, le déploiement en essaim est souvent limité à des environnements libres où aucun obstacle n'interfère avec le vol des robots. Dans de nombreuses applications, leurs trajectoires sont calculées avant le vol et les robots sont perceptuellement aveugles les uns aux autres. Dans ces cas, toutes les décisions sont prises par un ordinateur central qui est informé de tous les états des robots, augmentant ainsi leur risque d'accident en cas d'interruption du signal, limitant leur robustesse à l'échec et empêchant l'augmentation du nombre d'individus contrôlés simultanément. Une autre approche consiste à donner aux drones l'autonomie nécessaire, pour prendre leurs propres décisions sur la base d'informations d'origine locale, ces derniers manifestant ainsi un comportement d'évitement d'obstacles pour offrir plus de polyvalence pour leur application dans des environnements réels.

Pour répondre à ces limitations, nous proposons une approche prédictive de contrôle des essaims robotiques inspirée par les essaims d'oiseaux. En particulier, nous développons des méthodes qui permettent aux drones de coordonner leur mouvement en prédisant leur trajectoire future et celle de leurs voisins sur la base des informations d'état actuel et de la connaissance d'un modèle de vol. D'abord, nous présentons un algorithme prédictif centralisé qui calcule les trajectoires des drones en temps réel et améliore la synchronisation et l'ordre du vol de l'essaim par rapport aux algorithmes de pointe actuels. Grâce à des simulations approfondies, nous montrons que notre algorithme est robuste à différentes densités d'obstacles, vitesses d'essaim et distances inter-agents. Dans les chapitres suivants, nous formulons un algorithme prédictif distribué avec les mêmes avantages qualitatifs que son homologue centralisé, mais qui peut accroître la taille de l'essaim. Nous montrons par ailleurs que notre approche est tolérante à une large gamme de bruit des capteurs. Nous étendons l'utilisation de cet algorithme également pour les essaims qui ne se basent pas sur de la communication explicite, mais sur des capteurs tels que la vision pour mesurer les distances relatives entre les individus. Enfin, concernant les essaims qui n'utilisent que des capteurs locaux, nous analysons l'applicabilité des modèles d'essaim purement réactifs pour des robots à champ de vision limité. Nous vérifions aussi l'adaptabilité de ces mêmes modèles basés sur la vision et qui tiennent compte des occlusions à des essaims de grande taille. Nous validons les algorithmes proposés dans cette thèse dans des environnements virtuels avec des nombreux obstacles ainsi que dans le monde réel avec une flotte de quadricoptères légers, tenant dans une main.

Mots clés : essaims de robots aériens, systèmes multi-robots, modèle de contrôle prédictif, optimisation convexe, mouvement collectif, algorithmes d' essaim, planification de trajectoires, optimisation génétique, contrôle basé sur la vision

Table of contents

Acknowledgments **i**

Abstract (English/Français) **iii**

List of figures **xi**

List of tables **xiii**

1 Introduction **1**

1.1 Motivation 1
1.2 General approach 3
1.3 Thesis outline 4

2 Algorithms for aerial drone swarms **9**

2.1 Preliminaries and notation 9
2.2 Potential field-based models 10
2.2.1 Potential field 10
2.2.2 Reynolds swarm model 10
2.2.3 Olfati-Saber model 13
2.2.4 Vasarhelyi model 15
2.3 Optimization-based models 17
2.3.1 Sequential Convex Programming 17
2.3.2 Optimal Reciprocal Collision Avoidance 17
2.3.3 Buffered Voronoi Cells 18
2.4 Learning-based models 18
2.4.1 Local to local learning models 18
2.4.2 Global to local learning models 18
2.5 Neighbor selection 19
2.5.1 Neighbor selection based on the Euclidean distance 19
2.5.2 Neighbor selection based on the topological distance 20
2.5.3 Neighbor selection based on Voronoi tessellation 20
2.5.4 Neighbor selection based on line-of-sight occlusions 21
2.6 Swarm performance metrics 22
2.6.1 Order 23

2.6.2 Agent-agent safety . . . 23
2.6.3 Agent-obstacle safety . . . 23
2.6.4 Union . . . 24
2.6.5 Connectivity . . . 24
2.6.6 Mission completion time . . . 24
2.6.7 Trajectory length . . . 25

3 Centralized predictive control of aerial drone swarms 27
3.1 Introduction . . . 28
3.2 Method . . . 29
3.2.1 PF swarm model . . . 31
3.2.2 Agents' dynamics . . . 33
3.2.3 NMPC swarm model . . . 33
3.2.4 Simulation setup . . . 38
3.2.5 Drone experimental setup . . . 38
3.3 Results . . . 39
3.3.1 Comparison of PF and NMPC aerial swarms . . . 41
3.3.2 Environments with different obstacle densities . . . 42
3.3.3 Scalability to different inter-agent distances and speeds . . . 44
3.4 Discussion . . . 45

4 Distributed predictive control of aerial drone swarms 51
4.1 Introduction . . . 51
4.2 Methods . . . 53
4.2.1 Model of a flying agent . . . 54
4.2.2 Trajectory parameterization . . . 54
4.2.3 Migration . . . 55
4.2.4 Agents' reciprocal avoidance . . . 55
4.2.5 Agents' cohesion . . . 57
4.2.6 Obstacle avoidance . . . 57
4.2.7 Control effort . . . 58
4.2.8 Desired trajectory . . . 59
4.2.9 Implementation of Bezier curves . . . 59
4.2.10 Swarm performance metrics . . . 59
4.3 Results . . . 61
4.3.1 Scalability in the agent number and noise robustness . . . 62
4.3.2 Adaptability to different environments . . . 64
4.3.3 Comparison of collision avoidance methods . . . 66
4.3.4 Computational complexity . . . 67
4.3.5 Hardware experiments . . . 67
4.4 Discussion . . . 68

5 Reynolds swarms with limited visual sensing 71

5.1 Introduction . . . 71
5.2 Method . . . 72
  5.2.1 Notation . . . 72
  5.2.2 Limited field of view . . . 73
  5.2.3 Reynolds swarm model . . . 75
  5.2.4 Agents dynamics . . . 75
  5.2.5 Swarm performance metrics . . . 76
5.3 Results . . . 76
  5.3.1 Fixed Reynolds coefficients . . . 78
  5.3.2 Optimized Reynolds coefficients . . . 79
5.4 Discussion . . . 80

6 Scalable vision-based swarms in the presence of occlusions 83
6.1 Introduction . . . 83
6.2 Method . . . 85
  6.2.1 Neighbor selection . . . 86
  6.2.2 Sensing noise . . . 88
6.3 Experimental setup . . . 89
  6.3.1 Experimental parameters . . . 89
6.4 Results . . . 90
  6.4.1 Performance across swarm sizes . . . 91
  6.4.2 Performance across swarm densities . . . 96
6.5 Conclusions . . . 98

7 Sensor-based predictive control of aerial swarms 101
7.1 Introduction . . . 101
7.2 Method . . . 102
  7.2.1 Model of a flying agent . . . 103
  7.2.2 Inter-agent collision avoidance . . . 103
  7.2.3 DMPC swarm model . . . 104
  7.2.4 Neighbor selection . . . 105
  7.2.5 Neighbor predicted trajectory . . . 106
  7.2.6 PF swarm model . . . 106
  7.2.7 Swarm performance metrics . . . 107
7.3 Results . . . 108
  7.3.1 Scalability in the swarm size . . . 108
  7.3.2 Comparison between swarm models . . . 109
7.4 Discussion . . . 111

8 Conclusion 113
8.1 Limitations of the predictive control approach . . . 114
8.2 Possible directions for future work . . . 115
8.3 Closing remarks . . . 116

A Waypoint navigation of vision-based drone swarms **117**
- A.1 Introduction 117
- A.2 Method 119
  - A.2.1 The visual field of view 119
  - A.2.2 Vision-based swarm model 120
  - A.2.3 Position-based swarm model 122
  - A.2.4 Swarm performance metrics 123
- A.3 Results 124
  - A.3.1 Vision-based swarm model: the effect of individual and visual parameters 125
  - A.3.2 Vision-based swarm model: goal-oriented flight 126
  - A.3.3 Swarm models comparison 126
  - A.3.4 Hardware experiments 127
- A.4 Discussion 127

B Open-source software **129**
- B.1 Introduction 129
- B.2 Related Work 131
- B.3 Software architecture 132
  - B.3.1 Drone 133
  - B.3.2 Swarm 134
  - B.3.3 Swarm algorithms 134
  - B.3.4 Graphical User Interfaces (GUIs) 135
  - B.3.5 Plotting tools 136
  - B.3.6 Performance analysis 136
- B.4 Comparison of swarm algorithms and computational time analysis 138
- B.5 Conclusions and future work 139

C Publications **141**

Bibliography **143**

Curriculum vitae **155**

List of figures

2.1 Reynolds swarm in 2D and 3D 11
2.2 Cohesion and separation forces in the Reynolds model 12
2.3 Inter-agent equilibrium distance of the Reynolds model for different swarm sizes 13
2.4 Olfati-Saber distance-matching potential 14
2.5 Olfati-Saber distance-matching force 14
2.6 Trajectories of agents flying with the Vasarhelyi's model 17
2.7 Neighbor selection based on the Euclidean distance 19
2.8 Neighbor selection based on the topological distance 20
2.9 Neighbor selection based on Voronoi tessellation 21
2.10 Neighbor selection based on vision 21
2.11 Illustration of the performance metrics used for the evaluation of the swarm flight 22

3.1 Experimental setup of NMPC drone swarm flying in cluttered environments . . 30
3.2 Experimental setup of NMPC drone swarm flying in cluttered environments . . 34
3.3 Comparison between the PF-based model and our NMPC swarm model in simulation 42
3.4 Comparison of the PF and NMPC swarm deployment in different environments 43
3.5 Scalability of the NMPC swarm in inter-agent distance and speed 46
3.6 Scalability of the PF swarm in inter-agent distance and speed 47
3.7 Performance of the simulated PF and the NMPC swarms for varying inter-agent distances and speeds 48

4.1 Self-organized distributed predictive swarm flying in an artificial forest 52
4.2 Modelling of the environments 52
4.3 Schematic overview of the DMPC method 54
4.4 Swarm performance in forest-like environment at different noise levels 62
4.5 Number of completed missions in the forest-like environment 63
4.6 Swarm trajectories in the forest like environment at different noise levels 63
4.7 Swarm performance comparison in the funnel and forest-like environments . . 64
4.8 Number of completed missions in the funnel-like environment 65
4.9 Swarm of 36 drones in a funnel-like environment 65
4.10 Comparison of collision avoidance methods for the DMPC swarm model 66

4.11 Experimental results of the DMPC swarm in a forest-like environment 68

5.1 Illustration of the width and azimuth angles for an agent with limited FOV . . . 73
5.2 Sensing configurations of an agent with different width and azimuth values . . 74
5.3 Reynolds swarm with limited-FOV agents 74
5.4 Simulation workflow for testing sensor configurations with Reynolds model . . 77
5.5 Visual configurations with high order scores 79
5.6 Simulation results of swarms with limited FOV and different Reynolds gains . . 81
5.6 Simulation results of swarms with limited FOV and different Reynolds gains . . 82

6.1 Scalability of nearest neighbor distance for varying group sizes and densities. . 87
6.2 Neighbor selection strategies: metric, visual, topological, and Voronoi-based. . 88
6.3 Performance of neighbor selection methods for the experiments on the scalability of vision-based drone swarms. 92
6.4 Swarm trajectories using visual and Voronoi neighbor selection methods. . . . 93
6.5 Swarm trajectories using myopic and topological neighbor selection methods. 94
6.6 Schematic representation of the switching topologies caused by visual occlusions. 95
6.7 Swarm performance of neighbor selection methods for varying group density. . 97

7.1 Neighbor selection method for sensor-based drone swarms 103
7.2 Snapshots of predicted trajectories of the sensor-based DMPC swarm 104
7.3 Aggregated performance of the sensor-based DMPC model for different swarm sizes 109
7.4 Size scalability of the sensor-based DMPC swarm 110
7.5 Replanning variance and neighbor prediction error for the DMPC sensor-based swarm model. 111

A.1 Vision-based swarm experiment in our motion tracking hall 118
A.2 Schematic representation of the vision-based swarm method that operates directly on the visual FOV 121
A.3 Neighbors detection and estimation of the relative positions 122
A.4 Performance of the vision-based swarm model 123
A.5 Scalability of the vision-based swarm 125
A.6 Goal-oriented flight of the two swarm models 125
A.7 Hardware experiment of the vision-based swarm with four drones 127

B.1 3D swarm visualizations in SwarmLab 130
B.2 *Aerial swarms* publications 130
B.3 SwarmLab simulation workflow 133
B.4 Maps with varying obstacle density 136
B.5 Comparison of two swarm algorithms with SwarmLab 137
B.6 Real-time factor for varying sizes of the swarm 139

List of tables

3.1 Optimized parameter values of the PF swarm model 32
3.2 Parameter setting of the evolutionary optimization for the PF swarm. 32
3.3 Parameter description of the NMPC swarm model 38
3.4 Metrics to measure the performance of the NMPC drone swarm 41
3.5 Swarm and environment configurations for the NMPC swarm model experiments 44
3.6 Aggregate performance of the simulated PF and the NMPC swarms at varying obstacle densities 45
3.7 Aggregate performance of the simulated PF and the NMPC swarms at varying inter-agent distances 49
3.8 Aggregate performance of the simulated PF and the NMPC swarms at varying speeds 50

4.1 Performance metrics for the DMPC swarm 60
4.2 DMPC swarm model parameters 61
4.3 Empirical runtime of the DMPC swarm model for different swarm sizes 66
4.4 Comparison of simulation and hardware swarm performance 69

5.1 List of the different Reynolds gain triplets used in the simulation. 78
5.2 Parameters of the genetic algorithm used to determine the optimal Reynolds gains for swarms with limited FOV 80

6.1 Neighbor selection methods used for the experiments 87
6.2 Parameters used during the experiments on the scalability of vision-based drone swarms 90

7.1 Swarm model parameters used for the sensor-based DMPC model. 107

A.1 Optimal parameters for the vision-based swarm model 124
A.2 Comparison of the swarm performance with the two models 127

1 Introduction

Lausanne, 2052.
Thousands of bionic bees fly around the city. They join their natural counterpart in buzzing soft clouds that gently skims by buildings and soar over trees. They pollinate crops and guarantee adequate nourishment for the populace. The sound of sirens penetrates the tranquility of the city. A group of winged ambulances and firefighters cross the sky in the direction of a dense plume of smoke. They extinguish the fire coming from an abandoned hangar and save a couple of unfortunate passers-by engulfed in the flames. On a regular Monday morning, the sky traffic is harmonious, the vehicles stream is fluid despite their heterogeneity. A window cleaner sits in his comfortable chair and recollects his first manually cleaning job while monitoring the work of his self-flying mini-helicopter cleaning apparatus on a screen. He is sure that his son would not believe him if he told the story of how he hung against a wall of glass, secured by a rope like a circus act.

1.1 Motivation

Aerial robotic swarms often appear in sci-fi movies as a symbol of future technologies [1] [2] [3]. Their development holds the promise of large socio-economic impacts. The synergistic flight of multiple aerial robots can enable many real-world applications. For example, aerial robots can be deployed to create detailed 3D maps, monitor crops, search for people in dangerous areas, and deliver medicine to otherwise inaccessible places [1, 2]. Deploying drones in swarms, as opposed to a single robot, allows them to complete time-critical tasks faster and more efficiently. Most impressively, the collaboration between multiple drones can enable entirely new applications that are beyond the capabilities of a single drone such as cooperative transportation and construction [3] or mobile sensor networks [4, 5]. However, commercial drone swarms deployed today are still far from autonomous.

---

[1] Black Mirror, Season 3, Episode 6, "Hated in the Nation" (October 2016)
[2] "Prometheus" (June 2012)
[3] Star Wars, Episode III, "Revenge of the Sith" (May 2005)

Hundreds of drones have been deployed in aerial light shows by companies such as Intel [4], EHang [5], and Verity Studios [6]. However, the drones in these examples were controlled centrally and the trajectories were pre-programmed which did not allow for adaptation to unforeseen changes in the environment. In addition, these swarms fly in open environments where no obstacles interfere with their flight. As a result, their lack of autonomy severely limits their usage for many applications.

Drone swarm control is traditionally divided into two categories according to its algorithmic design: centralized and decentralized [6]. Centralized swarm models require the presence of a central computing node that determines the robot commands based on the knowledge of all positions and velocities and then sends them to the swarm [7, 8, 9, 10]. The central computer can optimize for global objectives, such as the total energy required by the agents to complete their missions. However, it can represent a major vulnerability for the swarm since the central decision-maker constitutes a single point of failure. To deploy a centralized swarm, frequent communication between the central computer and the agents needs to be orchestrated, and if the communication is interrupted the mission is compromised. Alternatively, in decentralized approaches, decision-making is shared among all agents, thereby improving robustness against one individual's failure [11, 12, 13, 14]. Additionally, in decentralized control strategies, each agent's decisions only depend on a limited number of neighbors, therefore allowing the scalability of swarm size.

From the fluid wavelike movements of starlings flocks to the swift turning maneuvers of bee swarms, nature displays impressive examples of coordinated flight, simply achieved by decentralized decision-making [15, 16, 17, 18, 19]. In particular, animals rely on local sensing capabilities to perceive other swarm members [19, 20]. Early work suggested that the collective motion of a biological swarm can be described by the combination of three behavioral rules that apply to each agent simultaneously [21]. These rules consist of (a) *cohesion*, which brings each agent closer to its neighbors, (b) *repulsion*, which drives each agent away from its neighbors to avoid collisions, and (c) *alignment*, which steers each agent towards the average heading of its neighbors. For navigating environments with obstacles, the addition of a fourth rule, collision avoidance, is necessary to steer the agents around the obstacles [21, 22, 13]. Several works proposing variations of these rules have shown successful deployment of drone swarms in the real world. However, their validation is often limited to obstacle-free environments. To allow drone swarms to operate in cluttered environments and therefore improve their versatility for varied applications, it is crucial to integrate a safe collision avoidance behavior while operating in a decentralized manner. These requirements motivate this thesis.

---

[4] "Intel Drone Light Show: Intel's 50th Anniversary", https://www.intel.com/content/www/us/en/technology-innovation/videos/drone-light-show-50th-anniversary-video.html

[5] "EHang Egret's 1374 drones dancing over the City Wall of Xian, achieving a Guinness World Records title", https://www.ehang.com/news/365.html

[6] "The Globe and Mail: Mini-drone use on the rise to light up big concerts like Celine Dion and Drake", https://veritystudios.com/news/globe-and-mail-celine

1.2 General approach

The pioneering work of Reynolds [21] is at the origin of swarm robotics. Mathematically, Reynolds rules can be synthesized by virtual Potential Fields (PFs), i.e., vector fields describing how forces act at various positions in space. PFs encode the desired behaviors of the swarm. They regulate the inter-agent distance among neighboring individuals similar to a spring-mass system, adjust the velocity of the agents, steer them towards a common direction, and regulate their distance to obstacles [22].

The advantage of PF swarm models is that they are purely reactive, meaning that their decisions are solely based on the current sensory information and thus have low computational complexity [21, 22]. For this reason, PF models are convenient for the implementation on real robotic systems, either in obstacle-free environments [23, 12], or in environments with convex obstacles [13]. In the latter case, collision avoidance is obtained by defining virtual repulsive agents (called shill agents) located along the obstacles' boundaries. However, these shill agents present the inconvenience of slowing down the swarm as it approaches the obstacles [21, 24]. This effect becomes prominent in environments with a high density of obstacles, where PF swarms can significantly slow down. The slowdown can be attenuated by weakening the repulsion potentials, albeit at the expense of the swam safety, because some agents may collide. Moreover, to account for the idiosyncrasies of the real world, these models often include a significant number of parameters that have complex interdependencies [13, 25]. As a consequence, they often require the adoption of optimization techniques, such as evolutionary algorithms, to identify a viable instantiation of the parameters. Each of these instantiations is specific to the swarm's preferred speed and inter-agent distance and to the environmental layout [23, 13, 26]. In practice, in an open environment with a varying number of agents in the swarm, it would be unfeasible to find a suitable set of static parameters. Parameters maps could be pre-computed, rising in computational complexity with the number of considered scenarios. Finally, due to the absence of optimality considerations, these methods result in high variability of the inter-agent distances.

In this thesis, we qualitatively and quantitatively analyze the efficiency of the current state-of-the-art aerial swarm models in a variety of challenging real-world conditions (i.e., presence of obstacles, sensor noise, limited perception). Then, we propose a method to remove the difficulties anticipated above that consist of endowing swarming agents with prediction-based control. It has been recently advocated that some form of predictive control, in the form of an internal model of the actions of their conspecifics, may also be leveraged by biological swarms where the apparent synchronization of coordinated maneuvers, such as a flock of starlings or a school of fish, cannot be explained by a purely reactive system [27]. Inspired by this hypothesis, the methodology proposed in this thesis endows flying agents with a model of swarm behavior based on Model Predictive Control (NMPC).

Model Predictive Control (MPC) is a method that computes the control action of a system as the solution of a constrained optimization problem [28, 29]. MPC leverages a mathematical representation of the system to predict and optimize its future behavior in an iterative process. Differently from PF control, MPC can explicitly handle constraints, such as physical limitations (e.g., flight speed and acceleration ranges of a drone) [30, 31, 32], and environmental restrictions (e.g., no-flight zones) [32, 33, 34]. However, the recursive online solution of constrained optimization problems is associated with higher computational costs, and therefore the adoption of predictive controllers in robotics is relatively recent [35]. MPC has shown promising results in simulation on multi-vehicle systems [36, 37, 33, 38, 39, 40, 41, 34], while, at the time this thesis started, the efficacy of online and self-organized predictive flight of swarms in the real world had yet to be proven. It is only recently that this technique has shown its surprising potential for collective flight in open environments [42].

This thesis extends the current literature on aerial drone swarms by analyzing how state-of-the-art PF-based models behave with various swarm sizes and in adverse conditions of the real world, e.g. in the presence of sensor noise, in increasingly cluttered environments, and with field-of-view limitations. To analyze the quality of the swarm flight, we propose a number of swarm performance metrics based on the agents states, such as positions and velocities. Then, we propose alternative swarm models based on MPC that can make the swarm fly safely in a range of different real-world environments populated with obstacles. These models have proven effective for operating with diverse swarm configurations such as preferred inter-agent distances and agent speed. Lastly, we consider the relaxation of the inter-agent communication requirements and explore the applicability of the MPC model to agents that cannot communicate their predicted states but must rely only on the use of on-board sensors with line-of-sight visibility such as cameras or depth sensors.

1.3 Thesis outline

In the following section, we provide a brief outline of the thesis and summarize the contents of each chapter. The summaries are based on the abstracts of the publications mentioned in the respective chapters.

Chapter 2: Algorithms for aerial drone swarms

We describe state-of-the-art algorithms that synthesize the collective motion of aerial swarms. These algorithms usually reproduce the biological behavior of cohesion, collision avoidance, and migration to a common destination. Among them, the PF algorithms are one of the most popular choices for the hardware implementation of robot swarms and constitute a baseline for comparing the algorithms that we design and present in the subsequent chapters. We then present plausible methods for neighbor selection that we used in the design of decentralized drone swarm models. Finally, we define the metrics that will be used to evaluate the flight quality of drone swarms.

Chapter 3: Centralized predictive control of aerial drone swarms

We propose a novel swarm model based on nonlinear MPC that removes the difficulties of state-of-the-art PF models presented in the previous chapter. Our predictive model incorporates the swarm behaviors of PF models and optimizes those behaviors under the knowledge of the agents' dynamics and environment. It generates self-organized, safe, and cohesive trajectories by solving an optimization problem in real-time. We show that our approach improves the speed, order, and safety of the swarm compared to the PF approach. Our model is independent of the environment layout and scalable in the swarm speed and inter-agent distance. We validate our model with a swarm of five quadrotors that can successfully navigate in a real-world indoor environment populated with obstacles.

Chapter 4: Distributed predictive control of aerial drone swarms

The centralized MPC model of the previous chapter lacks scalability in the swarm size. Here, we propose a distributed version of the same model. In simulation, we show that the distributed model is scalable to large numbers of agents and suitable for deployment in different environments, specifically a forest and a funnel-like environment. Furthermore, our results show that the agents are capable of collision-free flight with noisy sensor measurements for a noise level of up to 70% of the magnitude of the agent safety distance. Real-world experiments with a swarm of up to 16 quadrotors in an indoor artificial environment validate our method.

Chapter 5: Reynolds swarms with limited visual sensing

The MPC algorithms presented above require the use of explicit communication. Communication-based drone swarm models are fragile to outages and delays, especially when flying in high-density formations or in cluttered environments. From this chapter on, we remove the hypothesis of explicit communication between agents. Instead, we assume the availability of on-board visual sensors that allow the agents to estimate relative positions and velocities to their neighbors. When considering vision-based agents, the modeling of perceptual factors such as the limited field of view, i.e., the angular extent through which an agent is sensitive to the world, becomes fundamental. In this chapter, we analyze the effects of limited visual sensing on the performance of a commonly used and computationally efficient PF swarm model, i.e., the Reynolds model. We study how the reduction in the field of view and the orientation of the visual sensors affect the performance of the swarm. As Nature suggests, our results confirm that lateral vision is essential for coordinating the movements of individuals within a swarm. Moreover, agents benefit from omnidirectional vision to avoid collisions. We achieve the results presented in this paper through extensive Monte-Carlo simulations and integrate them with the use of genetic algorithm optimization.

Chapter 6: Scalable vision-based swarms in the presence of occlusions

We address the scalability of vision-based swarms with regard to group size and density. Vision-based swarms rely on the detection of neighbors but usually neglect mutual visual occlusions because they operate in small groups. We extend a PF-based algorithm with a realistic model of visual occlusions that discards agents if they are obstructed by closer ones. We evaluate the visibility model in simulation with up to one thousand agents. In particular, we find that small agent displacements have considerable effects on neighbor visibility and lead to control discontinuities. We show that the destabilizing effects of visibility switches, i.e., agents continuously becoming visible or invisible, can be mitigated if agents select their neighbors from adjacent Voronoi regions. The results show that Voronoi-based interactions enable vision-based swarms to remain collision-free, ordered, and cohesive in the presence of occlusions.

Chapter 7: Sensor-based predictive control of aerial swarms

We extend our previous work on predictive swarm models to purely sensor-based agents. Instead of communicating their future trajectories, the agents predict them based on the local knowledge of the environment and momentary neighbor information. We evaluate our model in simulation in a forest-like environment at different swarm sizes and we show the swarm can avoid collisions, while the flight synchrony worsen and the trajectory lengths increase compared to the communication-based DMPC model. We also compare the sensor-based swarm performance with a potential-field model from the state of the art and show that the DMPC swarm has overall better flight performance across different sizes.

Chapter 8: Conclusion

We summarize the thesis and conclude with a discussion of the implications, significance, limitations, and possible directions for future work.

Appendix A: Waypoint navigation of vision-based drone swarms

We present an alternative swarm model that operates directly on a segmentation of the visual field of view. We show that this model can provide collision-free and goal-directed flight without estimating relative positions. We extensively study this approach in simulation for goal-oriented navigation missions. We compare the results with a position-based swarm model and we validate our approach with a swarm of four drones flying in a controlled indoor environment.

Appendix B: Open-source software

We briefly describe a software package to simulate drone swarms which we developed during this thesis and we made publicly available on Github.

Appendix C: Publications

The work presented in this thesis is based on the following publications:

- E. Soria, F. Schiano, D. Floreano, "The influence of limited visual sensing on the Reynolds flocking algorithm," in *The Third IEEE International Conference on Robotic Computing (IRC)*, Naples, Feb. 2019 [26].
- E. Soria, F. Schiano, D. Floreano, "SwarmLab: a MATLAB drone swarm simulator," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Las Vegas, Feb. 2020 [43]
- E. Soria, F. Schiano, D. Floreano, "Predictive control of aerial swarms in cluttered environments," in *Nature Machine Intelligence*, vol. 3, pp. 545-554, May 2021 [10].
- E. Soria, F. Schiano, D. Floreano, "Distributed predictive drone swarms in cluttered environments," in *IEEE Robotics and Automation Letters (RA-L)*, vol. 7, no. 1, pp. 73-80, Jan. 2022 [44].
- E. Soria, H. Birch, F. Schilling, D. Floreano, "Waypoint Navigation of Vision-based Aerial Swarms without Estimation of Relative Positions," in *IEEE International Conference on Robotics and Automation (ICRA)*, (under review).
- F. Schilling, E. Soria, D. Floreano, "On the Scalability of Vision-based Drone Swarms in the Presence of Occlusions," in *IEEE Access*, (under review).

2 Algorithms for aerial drone swarms

We commence this thesis by describing state-of-the-art algorithms that synthesize collective motion for the navigation of aerial drone swarms. We focus on those algorithms that are decentralized and can generate the agents' trajectories online, i.e., during flight. The purpose of these models is threefold. Firstly, and most importantly, collisions among agents should be avoided. Secondly, the swarm should remain cohesive. Thirdly and finally, the swarm should be able to perform collective migration towards a common destination. According to the technique they use, we divide the swarm models into the following categories: potential field-based, optimization-based, and learning-based models. In the second part, we introduce possible techniques that one may use for selecting the neighbors of an agent. Neighbor selection is at the basis of decentralized swarm models in that each agent of the swarm reduces its focus to a subset of the swarm agents and its decision only depends on their states and not the whole swarm state.

2.1 Preliminaries and notation

In this thesis, we consider a set of N agents (i.e., components of the swarm) labeled by $i \in \{1, 2, 3, \dots, N\}$. The position, velocity, and acceleration of the agent i are denoted by $\boldsymbol{p}_i, \boldsymbol{v}_i, \boldsymbol{a}_i \in \mathbb{R}^2$ or $\mathbb{R}^3$ depending on whether we consider two-dimensional flight on a plane at constant altitude or flight in the three dimensions. We will specify in each chapter of our dissertation which one is considered. For the agents, we will consider single-integrator, double-integrator, or different linear dynamics that we will introduce in the following chapters. Hence, the control input to the agent, $\boldsymbol{u}_i$, can represent a position, velocity, or acceleration command depending on the context. We let $d_{ij} = \|\boldsymbol{p}_j - \boldsymbol{p}_i\|$ where $\|\cdot\|$ denotes the Euclidean norm. We model the swarm with a directed sensing graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where the vertex set $\mathcal{V} = \{1 \dots N\}$ represents the agents, and the edge set $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ contains the pairs of agents $(i, j) \in \mathcal{E}$ for which agent i can sense agent j. We denote $\mathcal{N}_i = \{j \in \mathcal{V} | (i, j) \in \mathcal{E}\} \subset \mathcal{V}$ as the set of neighbors of an agent i in $\mathcal{G}$ and $|\mathcal{N}_i|$ as its cardinality[1]. To describe swarm models in cluttered environments, we

---

[1] Note that both the set of edges $\mathcal{E}$ and the one of neighbors $\mathcal{N}_i$ of a specific agent i are time-varying.

introduce a set of M obstacles labeled by $m \in \mathcal{M} = \{1, \dots, M\}$. For convenience, most of the considered obstacles are cylinders. In the following, k denotes the index of a discrete-time step with duration dt. A generic variable x evaluated at instant k, $x(k)$ can be denoted by x^k if that variable is discrete. To keep our notation concise, we omit the time dependency when it is clear from the context.

2.2 Potential field-based models

2.2.1 Potential field

In physics, a Potential Field (PF) V is any field that obeys Laplace's equation. This equation is:

$$\nabla^2 V = 0 \tag{2.1}$$

where ∇ symbol denoted the gradient operator. In $\mathbb{R}^3$ with a Euclidean metric, the gradient of a generic function f, if it exists, is given by: $\nabla f(x, y, z) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}\right)$ [45]. Some common examples of PFs include electrical, magnetic, and gravitational fields.

A force field is a vector field that describes a non-contact force acting on a particle at various positions in space. Specifically, a force field is a vector field $\mathbf{F}(x, y, z)$, where $\mathbf{F}$ is the force that a particle would feel if it were at the point $(x, y, z) \in \mathbb{R}^3$ [46]. Any conservative force field $\mathbf{F}$ can be expressed as the negative gradient of a potential V:

$$\mathbf{F} = -\nabla V \tag{2.2}$$

Newton's second law of motion relates the force acting on a rigid body to two variables: the acceleration $\boldsymbol{a}$ and the mass m of the rigid body. It holds:

$$\mathbf{F} = m\boldsymbol{a} \tag{2.3}$$

A PF-based algorithm uses the artificial PF to regulate a robot around in a certain space [24]. By using the formulas above, we can transform artificial potentials into acceleration inputs to the robots. Then, the dynamic equations of the robot let us trace its velocity and position over time. In swarm robotics, artificial PFs can be defined to regulate the inter-agent distance among neighboring individuals similar to a spring-mass system, adjust the velocity of the agents, steer them towards a common direction, and regulate their distance to obstacles [22].

2.2.2 Reynolds swarm model

Reynolds' pioneering work on collective motion [21] opened the way to the fast-growing literature of swarm robotics. He suggested that the collective motion of a biological swarm can be described by the combination of three behavioral rules that apply to each agent

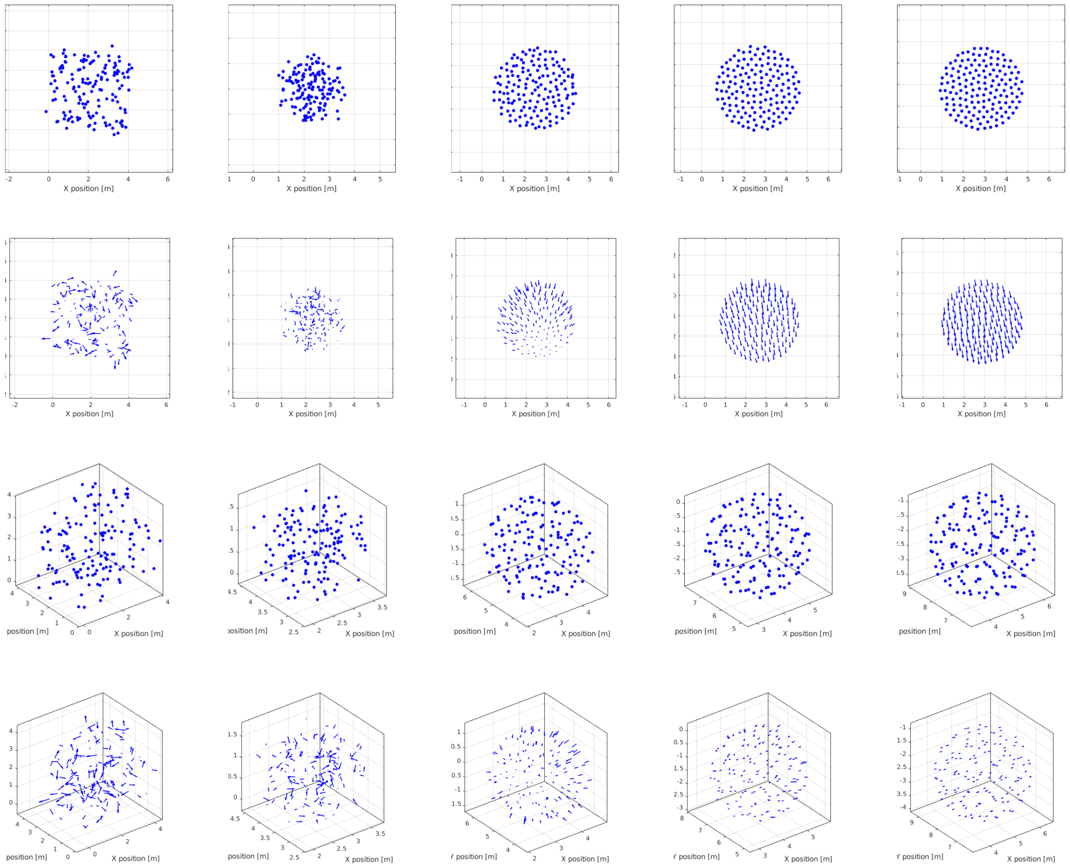


Figure 2.1 – **Reynolds swarm in 2D and 3D**. Snapshots at $t \in 0.1, 5, 10, 15, 20$ s of the position and velocities of the Reynolds swarm of 150 agents in $\mathbb{R}^2$ (row 1 and 2) and $\mathbb{R}^3$(row 3 and 4). Reynolds parameter values are the same as in 2.3. Here, we consider point-mass dynamics and $\mathcal{N}_i = \mathcal{V} \setminus \{i\}$. At $t = 0$ s, the agents positions and velocities are uniformly randomly distributed. Then, the swarm expands and contracts (see velocities at $t = 10$ s) while the agents reorganize into a lattice formation. The agents have almost reached an equilibrium at $t = 20$ s where they occupy the vertices of a lattice formation while constantly translating with the average of the initial velocities.

simultaneously. These rules consist of (a) *cohesion*, which brings each agent closer to its neighbors, (b) *repulsion*, which drives each agent away from its neighbors to avoid collisions, and (c) *alignment*, which steers each agent towards the average heading of its neighbors. Although these rules are defined locally, i.e., every agent regulates its flight with respect to a limited set of neighboring agents, they have proven to produce a globally coordinated motion. Mathematically, Reynolds model can be written as:

$$\tilde{\boldsymbol{u}}_i = \underbrace{\frac{c_{\text{coh}}}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} (\boldsymbol{p}_j - \boldsymbol{p}_i)}_{\text{cohesion}} - \underbrace{\frac{c_{\text{sep}}}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} \frac{\boldsymbol{p}_j - \boldsymbol{p}_i}{d_{ij}^2}}_{\text{separation}} + \underbrace{\frac{c_{\text{align}}}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} (\boldsymbol{v}_j - \boldsymbol{v}_i)}_{\text{alignment}} \tag{2.4}$$

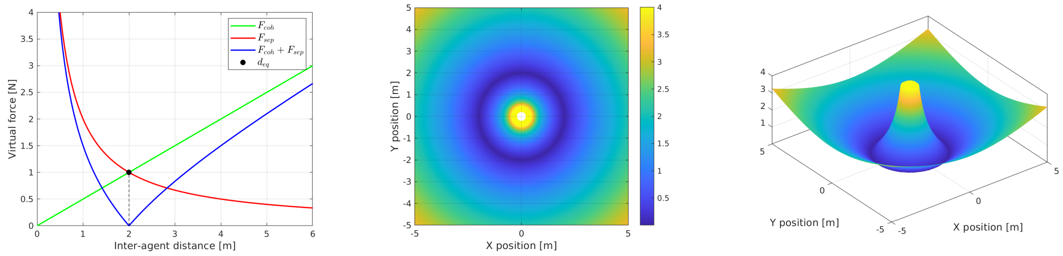


Figure 2.2 – **Cohesion and separation forces in the Reynolds model.** On the left, magnitude of the cohesion and separation forces in the Reynolds swarm model as functions of the inter-agent distance d_{ij}. We consider agents of unitary mass. In the middle and on the right, magnitude of the cohesion and separation force field in 2D generated by an agent positioned in $\boldsymbol{p}_i = (0,0)$. The parameters are $c_{\text{coh}} = 0.5$, $c_{\text{sep}} = 2$, and $a_{\max} = 4\ m/s^2$. For $N = 2$, it holds $d_{\text{eq}} = 2$.

where $\tilde{\boldsymbol{u}}_i$ is the ideal acceleration command for agent i, while c_{coh}, c_{sep}, and c_{align} are the weights referred to cohesion, alignment, and separation, respectively. We illustrate in 2.1 the 2D and 3D swarm patterns generated by the Reynolds rules with point-mass agents.

However, when working on real robots the behavior is generally more complex. The dynamics do not follow the point-mass model, the actuation is limited, and sensing is imperfect. To account for the actuation limitations, the ideal acceleration must be cut off at a maximum value, $a_{\max}$:

$$\boldsymbol{u}_i = \frac{\tilde{\boldsymbol{u}}_i}{\|\tilde{\boldsymbol{u}}_i\|} \min(\|\tilde{\boldsymbol{u}}_i\|, a_{\max}) \tag{2.5}$$

However, in some situations applying the cutoff leads to collisions.

The cohesion and separation forces define the equilibrium distance d_{eq} between any pair of neighbors, i.e., the distance at which the sum of the forces acting on an agent is null. This distance is given by the condition:

$$\sum_{j \in \mathcal{N}_i} \left(F_{\text{coh}}(d_{ij}) + F_{\text{sep}}(d_{ij}) \right) = 0 \tag{2.6}$$

For $N = 2$, we can compute it analytically: $d_{\text{eq}}(N = 2) = \sqrt{\frac{c_{\text{sep}}}{c_{\text{coh}}}}$ (Fig. 2.2). The equilibrium distance usually decreases when we add agents to the swarm, but it can vary depending on the choice of the neighbor set. In Fig. 2.3 we show the trend for a growing number of agents, and $\mathcal{N}_i = \mathcal{V} \setminus \{i\}$.

When the neighbor selection method is symmetric (i.e., if i is neighbor of j then j is neighbor of i) we can analytically determine the final velocity of the swarm. In that case, forces acting on neighboring agents are symmetric, meaning that whenever an agent i exerts a force on another agent j, j simultaneously exerts a force equal in magnitude and opposite in direction on i. Since no external force is applied, the momentum is conserved and hence the sum of the

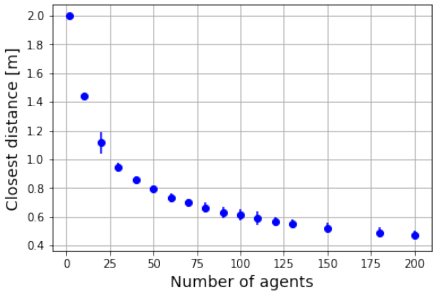


Figure 2.3 – **Inter-agent equilibrium distance of the Reynolds model for different swarm sizes**. Average and standard deviation of the equilibrium inter-agent distances (measured as $\min_j d_{ij}(t = 200\ s)$) of the Reynolds model in $\mathbb{R}^3$ for different swarm sizes ($N = \{2, 10, \ldots, 200\}$). Here, we consider $\mathcal{N}_i = \{j \in \mathcal{V} | j \neq i\}$. The Reynolds parameters are $c_{\text{coh}} = 0.5$, $c_{\text{sep}} = 2$, and $c_{\text{align}} = 0.5$. The distances have been measured at $t = 200\ s$ to avoid the influence of the initial conditions.

velocities in the system is conserved at any instant k, i.e., $\sum_{i\in\mathcal{V}} \boldsymbol{v}_i(k) = \sum_{i\in\mathcal{V}} \boldsymbol{v}_i(0) = \boldsymbol{V}_0$. At the equilibrium configuration, every agent has the same velocity $\boldsymbol{V}_0/N$ that depends on the initial velocities of the agents.

Based on Reynolds model, many variants have been developed. For goal-directed flight, alignment is replaced by migration, which steers each agent in a preferred migration direction [23, 47]. For navigating environments with obstacles, the addition of a fourth rule, collision avoidance, is necessary to steer the agents around the obstacles [12, 13].

2.2.3 Olfati-Saber model

Olfati-Saber first proposed the idea of formalizing Reynolds swarm rules with the definition of PFs [22]. Mathematically, Olfati-Saber combined the cohesion and separation rules relative to an agent i into a single PF $U_{\text{dm},i}$ for the distance matching, which has the advantage of presenting the reference distance d_{ref} as an explicit parameter. For $N = 2$, the reference distance is equal to the equilibrium inter-agent distance. The formula for the distance-matching potential $U_{\text{dm},i}$ is (Fig. 2.4):

$$U_{\text{dm},i} = \frac{1}{|\mathcal{N}_i(k)|} \sum_{j\in\mathcal{N}_i} \rho(d_{ij}/r)\sigma(d_{ij} - d_{\text{ref}}) \tag{2.7}$$

where $\rho(\cdot)$ is a weight function defining the influence of neighbor j on i, r is the perception radius of the agents, and $\sigma(\cdot)$ is a scalar function defining the intensity of cohesion or repulsion to neighbor j, depending on whether the two agents are closer or farther than the equilibrium

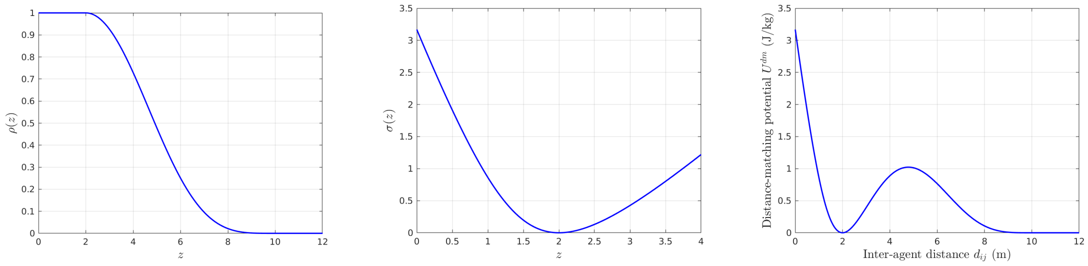


Figure 2.4 – **Olfati-Saber distance-matching potential**. From left to right, weight function ρ, shape function σ, and distance-matching potential U^{dm}

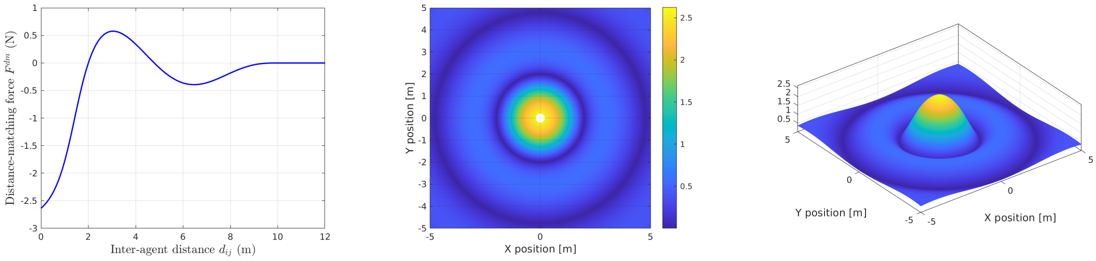


Figure 2.5 – **Olfati-Saber distance-matching force**. On the left, distance-matching force as a function of the inter-agent distance. In the middle and on the right, plots of the magnitude of the 2D field generated by the same force.

distance. Their definitions are:

$$\rho\left(\frac{d_{ij}}{r}\right) = \begin{cases} 1, & \frac{d_{ij}}{r} \in [0,\delta] \\ 1/2^2\left[1+\cos\left(\pi\frac{((d_{ij}/r)-\delta)}{(1-\delta)}\right)\right]^2, & \frac{d_{ij}}{r} \in [\delta,1] \\ 0, & \text{otherwise} \end{cases} \tag{2.8}$$

$$\sigma(d_{ij}-d_{\text{ref}}) = \frac{a+b}{2}\left[\sqrt{1+(d_{ij}-d_{\text{ref}}+c)^2}-\sqrt{1+c^2}\right]+\frac{(a-b)(d_{ij}-d_{\text{ref}})}{2} \tag{2.9}$$

where the constant parameters δ defines the weight function $\rho(\cdot)$, while a, b, and c define the shape of $\sigma(\cdot)$. The force for the distance matching can be obtained from the potential as:

$$\mathbf{F}_{\text{dm},i} = \nabla U_{\text{dm},i} \tag{2.10}$$

The alignment rule, renamed to velocity matching, is expressed by the same formula as in the Reynolds model. When a migration point (or waypoint) is provided, the alignment behavior can be replaced by another formula that steers the agents towards a migration point $\boldsymbol{p}_{\text{mig}} \in \mathbb{R}^3$ with a preferred speed v_{ref}:

$$\boldsymbol{F}_{\text{vm},i}(\boldsymbol{v}_i) = \gamma\left(\|\boldsymbol{p}_{\text{mig}}-\boldsymbol{p}_i\|\right)\left(v_{\text{ref}}\mathbf{u}_{\text{mig},i}-\boldsymbol{v}_i\right) \tag{2.11}$$

where $\gamma(\cdot)$ is a function weighting the velocity-matching force, i.e., $\gamma(z) = \min(1, z/d_{\text{tol}})$, and $\mathbf{u}_{\text{mig},i} = (\boldsymbol{p}_{\text{mig}} - \boldsymbol{p}_i)/\|\boldsymbol{p}_{\text{mig}} - \boldsymbol{p}_i\|$ is the unit vector directed from agent i to the migration point $\boldsymbol{p}$mig.

2.2.4 Vasarhelyi model

Standing on the models above, Vasarhelyi et al. proposed an adaptation for the navigation in confined environments and the presence of no-flight regions [13]. Their model includes the rules of self-propulsion for matching a preferred speed, repulsion to prevent inter-drone collisions, friction to reduce velocity oscillations, obstacle avoidance to avoid collisions with obstacles, and lastly wall avoidance to keep the swarm within the boundaries of a confined flight area.

The repulsion is active when neighboring agents are closer than the perception radius r and push them further apart. Its formula is:

$$\boldsymbol{v}_{\text{rep},ij} = \begin{cases} c_{\text{rep}}(r - d_{ij})\frac{\boldsymbol{p}_i - \boldsymbol{p}_j}{d_{ij}} & \text{if } d_{ij} < r \\ 0 & \text{otherwise} \end{cases} \tag{2.12}$$

where c_{rep} is the constant weight of the repulsion term. The total repulsion term is given by the sum of individual terms:

$$\boldsymbol{v}_{\text{rep},i} = \sum_{j \neq i} \boldsymbol{v}_{\text{rep},ij} \tag{2.13}$$

The friction term synchronizes motion to achieve collective swarming behavior, but it also serves as a damping medium, reducing self-excited oscillations emerging due to the delayed and noisy response to, for example, repulsion. Its formula is:

$$\boldsymbol{v}_{\text{fric},ij} = \begin{cases} C_{\text{fric}}(v_{ij} - v_{\text{fricmax},ij})\frac{\boldsymbol{v}_i - \boldsymbol{v}_j}{v_{ij}} & \text{if } v_{ij} > v_{\text{fricmax},ij} \\ 0 & \text{otherwise} \end{cases} \tag{2.14}$$

$v_{\text{fricmax},ij}$ is defined by:

$$v_{\text{fricmax},ij} = \max(v_{\text{fric}}, D(d_{ij} - r_{0,\text{fric}}, a_{\text{fric}}, c_{\text{fric}})) \tag{2.15}$$

where c_{fric}, $r_{0,\text{fric}}$, v_{fric}, and a_{fric} are constant parameters. The function $D(\cdot)$ is a smooth velocity decay function in space with constant acceleration at high speeds and exponential approach in time at low speeds. It is defined by:

$$D(d_{ij}, a_{\text{fric}}, c_{\text{fric}}) = \begin{cases} 0 & \text{if} d_{ij} < 0 \\ c_{\text{fric}} d_{ij} & \text{if } 0 < c_{\text{fric}} d_{ij} < a_{\text{fric}}/c_{\text{fric}} \\ \sqrt{2a_{\text{fric}} d_{ij} - a_{\text{fric}}^2/c_{\text{fric}}^2} & \text{otherwise} \end{cases} \tag{2.16}$$

Similarly to repulsion, the total friction term is given by the sum of individual terms:

$$\boldsymbol{v}_{\mathrm{fric},i} = \sum_{j \neq i} \boldsymbol{v}_{\mathrm{fric},ij} \tag{2.17}$$

Obstacle and wall avoidance are obtained by defining virtual repulsive agents (called shill agents) located along the obstacle and wall boundaries. These virtual agents are heading perpendicular to the obstacle or wall surface with a certain speed, v_{shill}. The real agents close to them should relax their velocity to the velocity of the shill agents through the formula:

$$\boldsymbol{v}_{\mathrm{fric},is} = \begin{cases} (v_{is} - v_{\mathrm{shillmax},is}) \frac{\boldsymbol{v}_i - \boldsymbol{v}_s}{v_{is}} & \text{if } v_{is} > v_{\mathrm{shillmax},is} \\ 0 & \text{otherwise} \end{cases} \tag{2.18}$$

where s is an index referring to a shill agent on a wall or obstacle, and $v_{\mathrm{fricmax},ij}$ is defined by:

$$v_{\mathrm{shillmax},ij} = D(d_{is} - r_{0,\mathrm{shill}}, a_{\mathrm{shill}}, c_{\mathrm{shill}}) \tag{2.19}$$

Finally, the self-propulsion term is obtained from the previous velocity of the agent as:

$$\boldsymbol{v}_{\mathrm{flock},i} = \frac{\boldsymbol{v}_i}{\|\boldsymbol{v}_i\|} v_{\mathrm{ref}} \tag{2.20}$$

where v_{ref} is the preferred speed of the swarm.

At any instant, the velocity command for agent i resulting from the contributions above is:

$$\tilde{\boldsymbol{u}}_i = \boldsymbol{v}_{\mathrm{flock},i} + \boldsymbol{v}_{\mathrm{rep},i} + \boldsymbol{v}_{\mathrm{fric},i} + \sum_{w \in \mathcal{W}_i} \boldsymbol{v}_{\mathrm{wall},im} + \sum_{m \in \mathcal{M}_i} \boldsymbol{v}_{\mathrm{obstacle},im} \tag{2.21}$$

After summing the contributions, a cutoff at $v_{\max}$ is applied on the velocity commands $\tilde{\boldsymbol{u}}_i$ and the velocities commanded to the agents become:

$$\boldsymbol{u}_i = \frac{\tilde{\boldsymbol{u}}_i}{\|\tilde{\boldsymbol{u}}_i\|} \min(\|\tilde{\boldsymbol{u}}_i\|, v_{\max}) \tag{2.22}$$

To search the large parameter space and find a parameter combination suitable to the deployment settings, Vasarhelyi et al. propose the use of evolutionary optimization. The selected parameters maximize the swarm order while minimizing the number of collisions within the agents and with the environment. However, the parameter choice gives no guarantees of collision avoidance, especially when tested on different swarm configurations (i.e., inter-agent distances and preferred speed) or different environmental settings (i.e., obstacle density and size).

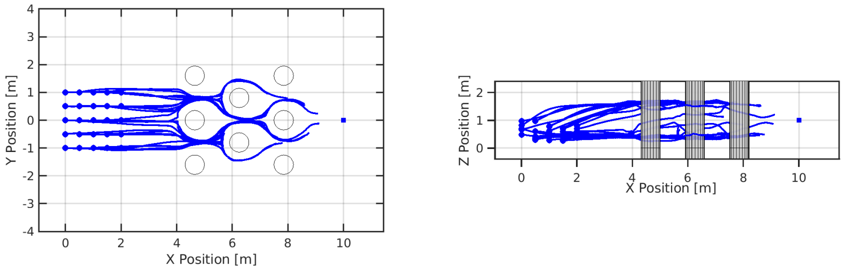


Figure 2.6 – **Trajectories of agents flying with the Vasarhelyi's model**. Top and side views, on the left and right respectively, of the agent trajectories flying through cylindrical obstacles with Vasarhelyi's model. The goal is located at $(10, 0, 1)$, while the floor and ceiling are set at $z = 0\ m$ and $z = 2\ m$. The agent's initial positions are indicated with dots.

2.3 Optimization-based models

A wide variety of optimization-based techniques exist to tackle the tasks of multi-robot trajectory generation and collision avoidance. These tasks are more generic and do not require all the behaviors that we defined for the swarm (i.e., cohesion, collision avoidance, and alignment or migration to a common destination). However, as these techniques work on some common ground, we describe them here for completeness and use them in the following chapters for comparison.

2.3.1 Sequential Convex Programming

Sequential Convex Programming (SCP) has been successfully applied to point-to-point trajectory generation for multiple agents [48, 49]. This method solves sequential optimization problems with convex approximations of collision constraints. However, in scenarios where the spaces are non-convex, these algorithms can fail to find feasible solutions because the convex approximations lead to a sequence of infeasible optimization problems. An alternative approach based on an iterative SCP scheme (iSCP) solves this issue by tightening the collision constraints incrementally, thus forming a sequence of more relaxed, feasible intermediate optimization problems [50].

2.3.2 Optimal Reciprocal Collision Avoidance

Optimal Reciprocal Collision Avoidance (ORCA) and all its variants have pushed towards real-time trajectory generation [51, 52]. Real-time trajectory generation is required for quick adaptation in dynamic environments. ORCA builds on the concept of reciprocal velocity obstacle and computes the set of non-colliding velocities for an agent. The selected robot velocity lies in this set and minimizes its distance to the preferred velocity (which is usually determined by the goal location of the robot and the preferred traveling speed). It implicitly assumes that all other agents make similar collision-avoidance reasoning while providing a simple approach to navigate multiple agents safely and smoothly amongst each other without explicit communication between them. Variations have been elaborated to minimize the discomfort of travelers supposing that the agents are vehicles carrying humans [53]. The variation minimizes the instantaneous accelerations while providing collision-less trajectories.

2.3.3 Buffered Voronoi Cells

A similar approach to ORCA achieves collision avoidance through the concept of Buffered Voronoi Cells (BVC) [54]. BVCs are Voronoi cells with edges retracted by a safety radius for the robot so that if the robot's center point is in the BVC, its body will be entirely within the Voronoi cell. The BVC concept has been recently used in tandem with discrete planners [55], primarily to avoid deadlocks in scenarios where plain BVC would not find a viable solution and the mission would fail.

2.4 Learning-based models

2.4.1 Local to local learning models

The decentralised drone swarms described previously rely on knowing the positional data of other agents which is either communicated or acquired visually. A different approach consists of learning some swarm model outputs directly from the visual inputs of the agents. Recent work has done this with the help of a convolutional neural network that learns the Reynolds model by imitation [47]. We call this approach local to local since both the expert and the learned network use only local information. The network could learn not only robust inter-agent collision avoidance but also the cohesion of the swarm in a sample-efficient manner. By visualizing the regions with the most influence on the motion of an agent, the authors showed that the neural controller effectively learns to localize other agents.

2.4.2 Global to local learning models

Different from the previous work, where a decentralized expert model is used to learn a decentralized swarm behavior, other works imitate the policy of centralized controllers using global information at training time while they require only local information and communications at test time. We call this approach global to local. A remarkable example is based on aggregation graph neural networks that use time-varying signals and time-varying network support [56]. They demonstrate the performance on communication graphs that change as the robots move. The common local controller of a single robot can exploit information from distant teammates using only local communication interchanges. Cohesive and safe collective motion is generated in empty environments. Interestingly, the authors examine how a decreasing communication radius and faster velocities increase the value of multi-hop information.

Another example uses deep imitation learning to solve the online multi-agent trajectory generation task in the presence of obstacles [57]. In this work, every individual has a different target position and cohesion is not a requirement. However, the interesting addition is the introduction of a differentiable safety module that ensures collision-free operation, thereby allowing for end-to-end policy training.

2.5 Neighbor selection

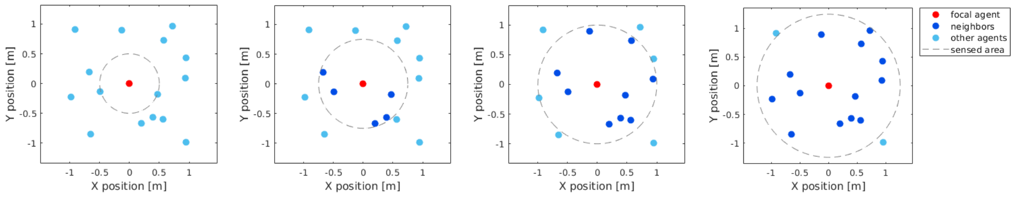
Careful consideration of the neighbor selection method is important for all swarm models since it introduces the notion of locality (e.g., in communication and perception) as opposed to all-to-all information transfer. In the following, we present different ways of selecting the neighbor set $\mathcal{N}_i$ of an agent i.

2.5.1 Neighbor selection based on the Euclidean distance

Neighbor selection based on the Euclidean distance chooses only those agents that fall within a radius r centered around the focal agent i (Fig. 2.7). We can formalize metric neighbor selection as the set:

$$\mathcal{N}_i = \left\{ j \in \mathcal{V},\ j \neq i \mid \mid d_{ij} < r \right\} \tag{2.23}$$

where r denotes the maximum perception range of the agents. Defining the set of neighbors based on a Euclidean distance is the most popular means of neighbor selection in the literature [21, 58, 22, 59]. In robotic implementations, r can be interpreted as a perception radius for vision-based swarms or a communication range for swarms that can exchange information via wireless links, for example. With the assumption that all agents are homogeneous and equally sized, we can use the metric perception range to represent visual acuity, i.e., the minimum size that another agent spans on the retina of the focal agent before it can no longer be perceived.



(a) $r = 0.5\ m$ (b) $r = 0.75\ m$ (c) $r = 1\ m$ (d) $r = 1.25\ m$

Figure 2.7 – **Neighbor selection based on the Euclidean distance**. The sensing radius r increases from left to right, i.e., $r = 0.5, 0.75, 1,$ and $1.25\ m$. The number of neighbors (in blue) of the focal agent (in red) grows with r.

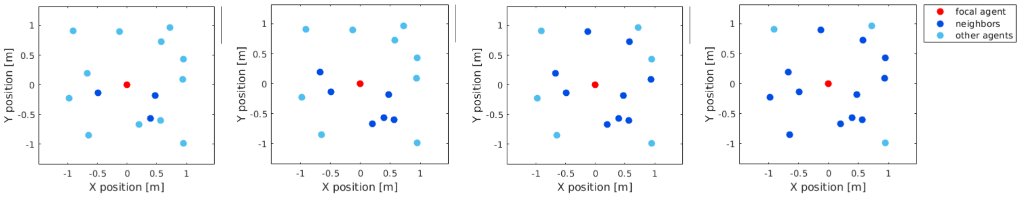
2.5.2 Neighbor selection based on the topological distance

Neighbor selection based on the topological distance chooses only the n nearest neighbors of the focal agent i (Fig. 2.8). In this case, we can write the set of neighbors as:

$$\mathcal{A}_i = \left\{ n\text{-}\underset{j \in \mathcal{V},\, j \neq i}{\operatorname{argmin}}\, d_{ij} \right\} \tag{2.24}$$

where the n-argmin operator selects at most the n nearest neighbors.

Topological neighbor selection is a popular method due to its explanatory success in natural swarms [19, 60] and is often used in models of collective motion to maintain group cohesion [59, 61].



(a) $n = 3$ (b) $n = 6$ (c) $n = 9$ (d) $n = 12$

Figure 2.8 – **Neighbor selection based on the topological distance**. The number of neighbors n increases from left to right, i.e., $n = 3$, 6, 9, and 12.

2.5.3 Neighbor selection based on Voronoi tessellation

Neighbor selection based on the Voronoi tessellation chooses only those agents whose Voronoi regions share a border with the focal agent (Fig. 2.9). We can write the set of Voronoi neighbors as:

$$\mathcal{N}_i = \{ j \in \mathcal{V},\ j \neq i \mid V_i \cap V_j \neq \emptyset \} \tag{2.25}$$

where $\emptyset$ denotes the empty set and V_i the Voronoi region of agent i in the d−dimensional space which can be defined as:

$$V_i = \left\{ \mathbf{q} \in \mathbb{R}^d,\ j \in \mathcal{V},\ j \neq i \mid \| \boldsymbol{q} - \boldsymbol{p}_i \| \leq \| \boldsymbol{q} - \boldsymbol{p}_j \| \right\}. \tag{2.26}$$

In other words, the Voronoi region of an agent can be described as the set of all points that are closer to itself than to any other agent. Neighbor selection based on the Voronoi tessellation can be seen as topological interactions that are parameter-free and automatically balanced in space [61]. Moreover, it can be shown that the average number of Voronoi neighbors is at most six for the planar case (i.e., $d = 2$) [62].

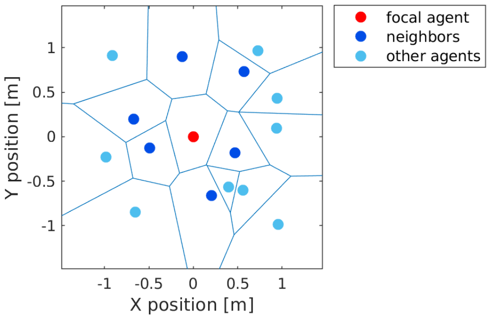


Figure 2.9 – **Neighbor selection based on Voronoi tessellation**. The Voronoi tessellation is highlighted in light blue. Only agents that share an edge of their Voronoi cell with the focal agent (in red) are considered neighbors (in blue). This neighbor selection method does not depend on any parameter.

2.5.4 Neighbor selection based on line-of-sight occlusions

Neighbor selection based on line-of-sight occlusions chooses only those agents that are not occluded by closer ones as seen from the perspective of the focal agent (Fig. 2.10). The set of visible agents can be written as:

$$\mathcal{N}_i = \{j \in \mathcal{V},\ j \neq i,\ j \neq k \mid \neg\left(\|\boldsymbol{u}_{ij} - \boldsymbol{u}_{ik}\| < \hat{r}_{ij} + \hat{r}_{ik} \wedge d_{ij} < d_{ik}\right)\} \tag{2.27}$$

where $\boldsymbol{u}_{ij} = \|\boldsymbol{p}_j - \boldsymbol{p}_i\| / d_{ij}$ and $\hat{r}_{ij} = r_{\text{agent}} / d_{ij}$ are the projections of the agent position and radius onto the unit circle, respectively. According to this definition, partially occluded agents are considered invisible, i.e., only the closest set of agents with an uninterrupted line of sight are contained in the visible set. This assumption is reasonable for monocular vision since the relative distance to other agents can only be reliably estimated if all of their spatial extents are visible.

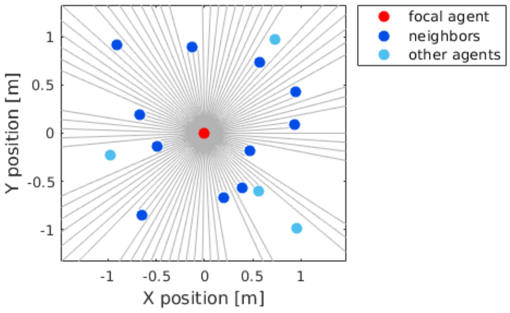


Figure 2.10 – **Neighbor selection based on vision**. Only agents which are fully visible from the focal agents (in red) are considered neighbors (in blue). This neighbor selection method does not depend on any parameter.

2.6 Swarm performance metrics

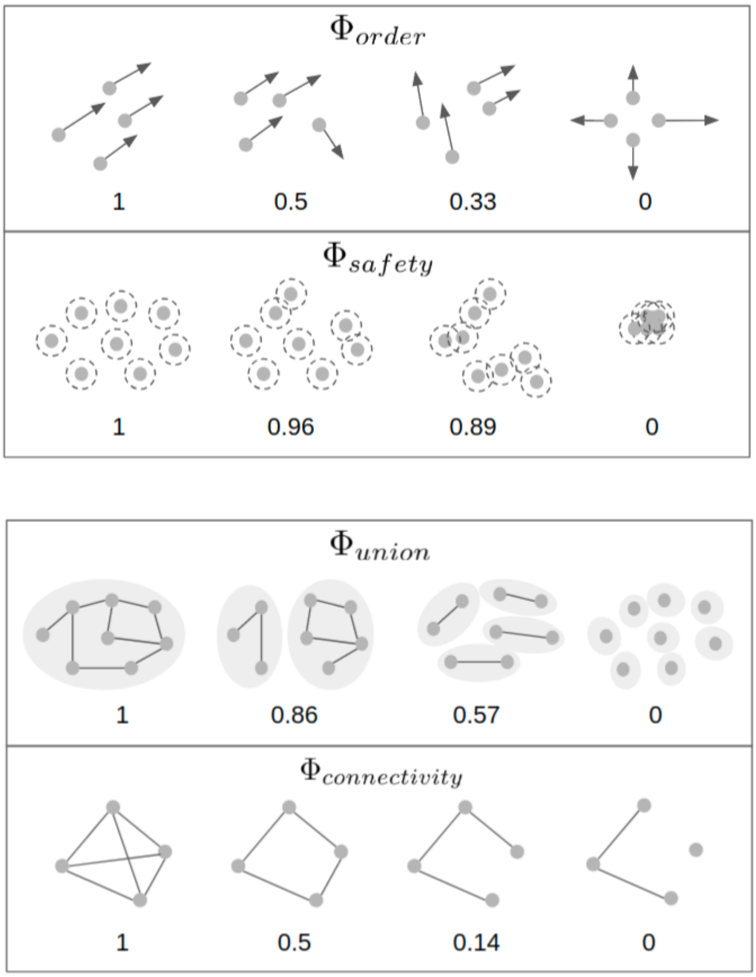


Figure 2.11 – **Illustration of the performance metrics used for the evaluation of the swarm flight**. In particular, they are: order (Φ_{order}), safety (Φ_{safety}), union (Φ_{union}) and connectivity ($\Phi_{\text{connectivity}}$). In general, 1 corresponds to a high score, while 0 refers to a poor score. Significant levels are chosen for every metric. In particular, for the safety metric, important changes in the flock configuration already happen in the highest half-range.

We define here relevant metrics that will be used for the evaluation of the swarm flight quality in the following chapters. These metrics are inspired by both the robotics and biology literature and adapted to this context.

2.6.1 Order

The order metric Φ^k_{order} captures the correlation of the agents' movements and gives an indication about how ordered the flock is. At every instant k, it is expressed by:

$$\Phi^k_{\text{order}} = \frac{1}{N|\mathcal{N}_i|} \sum_{i,j\in\mathcal{N}_i} \frac{\dot{\boldsymbol{x}}^k_i \cdot \dot{\boldsymbol{x}}^k_j}{\|\dot{\boldsymbol{x}}^k_i\| \|\dot{\boldsymbol{x}}^k_i\|}. \tag{2.28}$$

It is equal to 1 when all the agent velocities are aligned towards the same direction and it is equal to 0 when agent velocities are diametrically opposed two by two. To evaluate the global performance of the swarm during the experiment time T_{max}, the performance is averaged over time, as:

$$\Phi_{\text{order}} = \frac{\sum\limits_{k=1}^{K} \Phi^k_{\text{order}}}{K} \tag{2.29}$$

The same reasoning applies to other metrics.

2.6.2 Agent-agent safety

To measure the safety against inter-agent collisions we count the number of collisions $N_{\text{coll,agent}}$ and we compute the agent-agent safety metric $\Phi^k_{\text{safe,agent}}$, which measures the risk of collisions among the members of the flock at a given temporal instant. We define $r_{\text{safe,agent}}$ as the radius of a virtual sphere that surrounds the agent where the presence of other agents should be avoided for safety reasons. For real drones, it corresponds to the dimension of the robot, plus an arbitrary margin which is higher for more conservative approaches. If the number of violations (or collisions) is $N_{\text{coll,agent}}(k) = |\{(i,j)\ s.t.\ j \neq i\ \wedge d_{ij}(k) < r_{\text{safe,agent}}\}|$, then the agent-agent safety metric at instant k is:

$$\Phi^k_{\text{safe,agent}} = 1 - \frac{N_{\text{coll,agent}}}{N(N-1)}. \tag{2.30}$$

While $N_{\text{coll,agent}} \in \mathbb{N}$, $\Phi_{\text{safe,agent}} \in [0,1]$. In particular, $\Phi_{\text{safe,agent}} = 0$ when all possible pairs of agents are colliding and $\Phi_{\text{safe,agent}} = 1$ when no collision happens. In practice, since each agent occupy a volume it is not possible to get $\Phi_{\text{safe,agent}} = 0$ for large swarms. However, for a swarm of a given size, it is always true that a higher number of inter-agent collisions corresponds to a lower agent-agent safety value. Additionally, we use the minimum inter-agent distance $\min(d_{ij})(k)$ between all pairs of agents as a complementary measurement. The latter indicates whether collisions are happening at a given time k.

2.6.3 Agent-obstacle safety

To measure the safety against obstacle collisions we count the number of obstacle collisions $N_{\text{coll,obs}}$ and we compute the agent-obstacle safety metric $\Phi^k_{\text{safe,obs}}$, which measures the risk of collisions among the members of the flock at a given temporal instant. We define $r_{\text{safe,obs}}$ as the radius of a virtual sphere that includes the obstacles where the presence of other agents should be avoided. If the number of obstacle collisions is $N_{\text{coll,agent}}(k) = |\{(i, j)\ s.t.\ j \neq i\ \wedge d_{ij}(k) < r_{\text{safe,agent}}\}|$, then the agent-obstacle safety metric at instant k is:

$$\Phi^k_{\text{safe,agent}} = 1 - \frac{N_{\text{coll,agent}}}{N(N-1)}. \tag{2.31}$$

We then measure the minimum distance to obstacles $\max(d_{ij})$ as the minimum of the agent distances to all obstacles over the experiment time.

2.6.4 Union

The union metric Φ^k_{union} reflects how scattered the group members are and it counts the number of independent subgroups that originate during the experiment. We define N_{cc} as the number of connected components of the undirected graph that corresponds to the flock topology, then at time t_k it holds:

$$\Phi^k_{\text{union}} = 1 - \frac{N_{cc} - 1}{N - 1}. \tag{2.32}$$

Complementarily, we measure the maximum inter-agent distance $\max(d_{ij})$ over the experiment time.

2.6.5 Connectivity

The connectivity metric $\Phi^k_{\text{connectivity}}$ is defined from the algebraic connectivity [63] of the swarm underlying graph. Also known as the *connectivity eigenvalue*, this is the second smallest eigenvalue of the Laplacian matrix [63] associated with the undirected graph $\mathcal{G}'$ obtained from $\mathcal{G}$ and it is usually denoted by λ_2. Algebraic connectivity has been extensively used in swarm robotics [64, 65] because the magnitude of this value reflects crucial qualities of the graph. We define the value of the metric at time t_k as:

$$\Phi^k_{\text{connectivity}} = \frac{\lambda_2}{N} \tag{2.33}$$

where λ_2 is defined in 5.2.1. Notice that $\Phi^k_{\text{connectivity}} \neq 0$ only when $\Phi^k_{\text{union}} = 1$. In this sense, the connectivity metric is complementary to the union metric.

2.6.6 Mission completion time

The mission completion time T is the time that the swarm takes to get to the final destination. If the swarm migrates according to a preferred velocity $\boldsymbol{v}_{\text{ref}}$ we consider the mission completed when the swarm crosses a finish line, while if the swarm migrates towards a goal destination

$\boldsymbol{p}_{\text{mig}}$ then we consider the mission completed when the center of the swarm is located around $\boldsymbol{p}_{\text{mig}}$ up to a tolerance distance r_{tol}.

2.6.7 Trajectory length

The trajectory length L_{traj} is the average distance flown by the swarm agents from the beginning of the experiment until they reach the goal destination or the finish line, depending on the context, or until the experiment end if none of the previous conditions is verified.

$$L_{\text{traj}} = \frac{\sum_{i \in \mathcal{V}} \sum_{k=1}^{K} \|\boldsymbol{p}_i^k - \boldsymbol{p}_i^{k-1}\|}{N} \tag{2.34}$$

with K being the minimum between the time index corresponding to the mission completion and the experiment end time index.

3 Centralized predictive control of aerial drone swarms

In the previous chapter, we introduced classical models of aerial swarms, and in particular those based on potential fields. These models describe global coordinated motion as the combination of local interactions that happen at the individual level. Despite their explanatory success, they fail to guarantee rapid and safe collective motion when applied to aerial robotic swarms flying in cluttered environments of the real world, such as forests and urban areas. Moreover, these models necessitate a tight coupling with the deployment scenarios to induce consistent swarm behaviors. Here, we propose a predictive model that incorporates the local principles of potential field models in an objective function and optimizes those principles under the knowledge of the agents' dynamics and environment. We show that our approach improves the speed, order, and safety of the swarm, it is independent of the environment layout and scalable in the swarm speed and inter-agent distance. Our model is validated with a swarm of five quadrotors that can successfully navigate in a real-world indoor environment populated with obstacles.

The work presented in this chapter is adapted from [10]:

- E. Soria, F. Schiano, D. Floreano, "Predictive control of aerial swarms in cluttered environments," in *Nature Machine Intelligence*, vol. 3, pp. 545-554, May 2021.

Simulation and hardware experimental data that support the findings of this study can be downloaded from https://doi.org/10.5281/zenodo.4379168.

The code that supports the findings of this study can be downloaded from https://doi.org/10.5281/zenodo.4379503.

A press release realized by Mediacom is available at https://actu.epfl.ch/news/ helping-drone-swarms-avoid-obstacles-without-hitti/.

3.1 Introduction

In this chapter, we propose a novel drone swarm method that removes the difficulties of PF-based methods, and specifically the slow-down effect caused by shill agents and the usability of the model for a range of swarm configurations and different environments. This method consists of endowing swarming agents with prediction-based control. We show that aerial swarms with predictive control display faster flight while guaranteeing safe navigation in cluttered environments, they can adapt to diverse obstacle densities, and they are scalable to changes in the inter-agent distance and swarm's speed. It has been recently advocated that some form of predictive control, in the form of an internal model of the actions of their conspecifics, may also be leveraged by biological swarms where the apparent synchronization of coordinated maneuvers, such as a flock of starlings or a school of fish, cannot be explained by a purely reactive system [27]. Inspired by this hypothesis, the proposed method endows flying agents with a model of swarm behavior based on Nonlinear Model Predictive Control (NMPC).

Model Predictive Control (MPC) is a method that computes the control action of a system as the solution of a constrained optimization problem [28, 29]. MPC leverages a mathematical representation of the system to predict and optimize its future behavior in an iterative process. Differently from PF control, MPC can explicitly handle constraints, such as physical limitations (e.g., flight speed and acceleration ranges of a drone) [30, 31, 32, 66], and environmental restrictions (e.g., no-flight zones) [32, 33, 34]. However, the recursive online solution of constrained optimization problems is associated with higher computational costs, and therefore the adoption of predictive controllers in robotics has spread only recently [35].

MPC has shown promising results in simulation on multi-vehicle systems. Examples include the stabilization of multiple agents in obstacle-free environments [36, 37], in the presence of obstacles [33], and the generation of collision-free trajectories for groups of robots with known target locations [38, 39, 40]. NMPC is a variant of MPC that can handle the nonlinearities of a system or its constraints [29]. This advantage comes at the cost of being more computationally demanding. In the simulation, NMPC has been used to control leader-follower formations of drones without obstacles [41], and to control 2D quadrotor formations in the presence of convex obstacles [34].

Less work has been done on the use of MPC with multiple real drones, notably due to the difficulty of real-time implementation. Linear MPC has been used for trajectory planning in the presence of virtual obstacles in a leader-follower configuration, where a drone (the follower) has to keep a constant distance from a virtual agent (the leader) [67]. However, in leader-follower approaches, the leader has the extra knowledge of the group trajectory, which is either preprogrammed or provided by an external source. This aspect introduces an asymmetry in the agents' roles and adds a single point of failure in the swarm [68]. MPC has been used for the online generation of collision-free trajectories for a group of drones in environments with obstacles, where every drone is individually assigned an initial position and a target destination [32]. Instead, the model presented here is meant to coordinate the navigation of the swarm as a unique entity and guarantee internal order, in lieu of generating the trajectories separately. Concurrently, we avoid imposing a rigid formation or a fixed topology to the swarm, which may impact the freedom and fluidity of the agents' movements. Finally, NMPC has been shown to be capable of dealing with non-convex collision avoidance constraints in real multi-drone systems when the agents are assigned intersecting paths, although they were flying in empty environments [69].

In the proposed NMPC model, the objective function to be optimized is made of three components inspired from PF swarm models: (a) separation, which drives the inter-agent distances to a preferred value, (b) propulsion, which propels the agents with a preferred speed value, and (c) direction, which steers the swarm along a preferred direction. The separation component incorporates the effects of cohesion and repulsion in a single rule, in such a way that the inter-agent distance between neighboring individuals appears as an explicit parameter. The propulsion and direction components taken together in the NMPC swarm model replicate the effect of the migration term in the PF swarm models. However, keeping two terms with two independent parameters allows separate adjustments of their relative strengths. A fourth rule, (d) control effort is added to minimize the agents' accelerations, thereby smoothing flight trajectories and increasing energy efficiency. Each drone regulates its flight based on the knowledge of its neighbors and its own state and predicts its own trajectory and those of its neighbors thanks to a linearized dynamical model. The drones' neighbors are selected within a topological range, i.e. for every drone only a constant number of nearest neighbors is considered [19]. The proposed NMPC model integrates a set of constraints to ensure safety distances among drones and with obstacles. We implement a centralized version of our NMPC model and we compare simulation results to a PF model. We show that predictive controllers can safely fly the swarm in cluttered environments while significantly increasing the flight speed and synchronization of the swarm. Also, we show that the performance of the proposed NMPC model is independent of the obstacle density and environmental layout, differently from PF models. Additionally, we test the scalability of the proposed model to variations of desired inter-agent distance and swarm speed. We perform systematic experiments in simulation and validate the results with a swarm of five palm-sized quadrotors.

3.2 Method

In this work, we consider a swarm of N agents labeled by $i \in \{1,\ldots,N\}$. The position, velocity, and control input of the i-th agent are denoted by $\boldsymbol{p}_i$, $\boldsymbol{v}_i$, $\boldsymbol{u}_i \in \mathbb{R}^3$, respectively. Let $d_{ij} = \|\boldsymbol{p}_j - \boldsymbol{p}_i\|$ represent the distance between the center of two agents i and j, where $\|\cdot\|$ denotes the Euclidean norm. We model the swarm with a directed sensing graph $\mathcal{G} = (\mathcal{V},\mathcal{E})$, where the vertex set $\mathcal{V} = \{1,\ldots,N\}$ represents the agents, and the edge set $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ contains the pairs of agents $(i,j) \in \mathcal{E}$ for which agent i can sense agent j. We denote as $\mathcal{N}_i = \{j \in \mathcal{V} | (i,j) \in \mathcal{E}\} \subset \mathcal{V}$ the set of neighbors of an agent i in $\mathcal{G}$, and $|\cdot|$ indicates the cardinality of a set. We define the neighbors set utilizing a topological range, i.e., the set $\mathcal{N}_i$ contains the n nearest

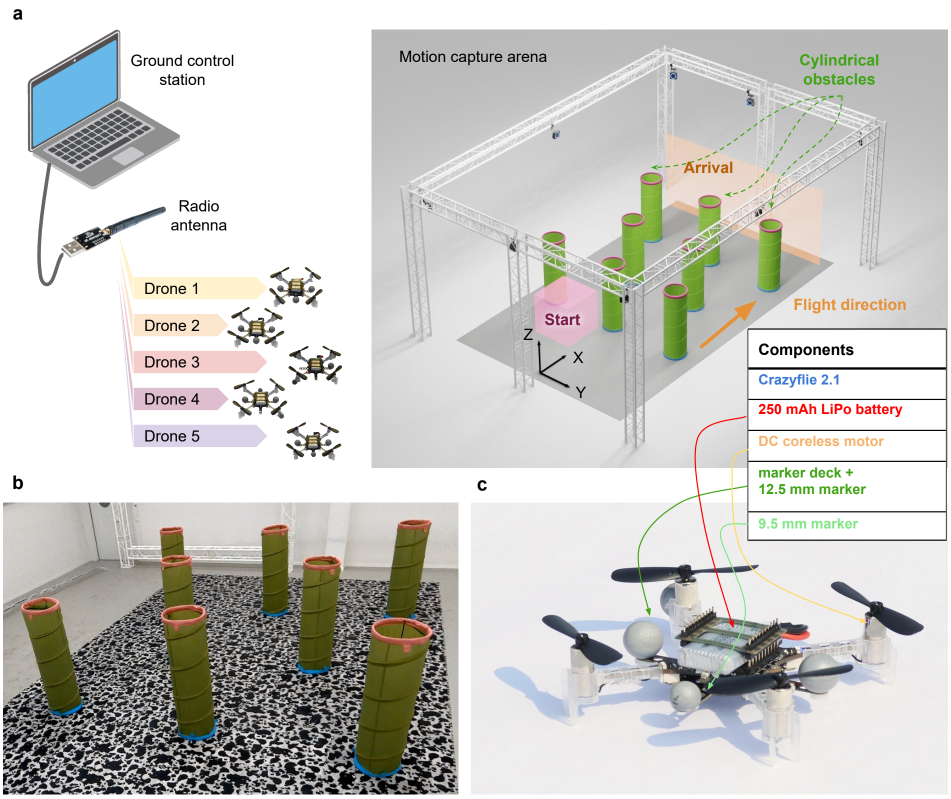


Figure 3.1 – **Experimental setup of NMPC drone swarm flying in cluttered environments.** (a) Illustration of the experimental setup and the environment configuration. A ground control station, equipped with a radio transmitter, computes and sends run-time control commands to the drones. The swarm flies in the 3D space of an indoor flying arena. The drones take off from initial random positions within a predefined start area (red zone). Drones swarm along the preferred migration direction (orange arrow). The mission is completed when all drones cross the arrival plane on the opposite side of the region. (b) Indoor test environment populated with cylindrical obstacles. (c) Components of the drones used for the hardware experiments.

neighbors of agent i. This choice is convenient for keeping the cardinality of the neighbor set constant, and it has also been shown to hold true for biological swarms [19]. Other studies have investigated different neighborhood approaches based on the Voronoi partition or ad-hoc attraction topologies [70]. However, we discarded those methods because they would introduce discontinuities in the objective function of our predictive swarm model, or they would constrain the swarm to a fixed formation. To reproduce a forest-like environment, we introduce M cylindrical obstacles labeled by $m \in \mathcal{M} = \{1, \ldots, M\}$. We denote as dim the distance between an agent i and the symmetry axis of cylinder m. In our simulations, the dynamics of the agents is reproduced in discrete time. We let $\boldsymbol{p}_i(k)$, $\boldsymbol{v}_i(k)$, $\boldsymbol{u}_i(k) \in \mathbb{R}^3$ be the position, velocity, and control input of the i-th agent at the time $t(k) = kdt$, respectively. For brevity, in the following, we will omit the time dependency when clear from the context.

3.2.1 PF swarm model

The PF model we present is inspired by a state-of-the-art model that allows drone swarm navigation in confined environments [13]. From the original model, we include the rule of repulsion to prevent inter-drone collisions, friction to reduce velocity oscillations, and obstacle avoidance to avoid collisions with obstacles. For the mathematical definition of these rules, we refer the reader to [13]. To ensure goal-directed flight in open environments, we added two rules: migration to provide a preferred velocity vector, and cohesion to keep agents together. We denote the migration velocity with $\boldsymbol{v}_{\text{ref}} = v_{\text{ref}}\boldsymbol{u}_{\text{ref}}$, where v_{ref} is the preferred speed and $\boldsymbol{u}_{\text{ref}}$ is the preferred direction. Then, the migration term, equal for every agent, corresponds to:

$$\boldsymbol{v}_{\text{mig}} = v_{\text{ref}}\boldsymbol{u}_{\text{ref}} \tag{3.1}$$

If the repulsion is active when neighboring agents are closer than the preferred distance d_{ref} and push them further apart, the cohesion is active when they are farther than d_{ref} to bring them closer. Repulsion and cohesion are inactive when two agents are precisely at the distance d_{ref}. The cohesion exerted on an agent i from a neighbor j is:

$$\boldsymbol{v}_{\text{coh},ij} = \begin{cases} c_{\text{coh}}(d_{ij} - d_{\text{ref}})\frac{\boldsymbol{p}_j - \boldsymbol{p}_i}{d_{ij}} & \text{if } d_{ij} < d_{\text{ref}} \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

where we choose the pairwise gain of cohesion equal to the repulsion gain $c_{\text{coh}} = c_{\text{rep}}$ and the cutoff for the minimum cohesion range equal to the repulsion range d_{ref}. The total cohesion effect calculated for agent i with respect to its neighbors is:

$$\boldsymbol{v}_{\text{coh},i} = \sum_{j \in \mathcal{N}_i} \boldsymbol{v}_{\text{coh},ij} \tag{3.3}$$

At any instant, the velocity for agent i resulting from the contributions above is:

$$\tilde{\boldsymbol{v}}_i = \boldsymbol{v}_{\text{mig}} + \boldsymbol{v}_{\text{coh},i} + \boldsymbol{v}_{\text{rep},i} + \boldsymbol{v}_{\text{fric},i} \sum_{s \in \mathcal{M}_i} \boldsymbol{v}_{\text{obstacle},is} \tag{3.4}$$

After summing the contributions, we apply a cutoff on the acceleration at $a_{\max}$ according to:

$$\boldsymbol{a}_i = \frac{\tilde{\boldsymbol{a}}_i}{\|\tilde{\boldsymbol{a}}_i\|} \min(\|\tilde{\boldsymbol{a}}_i\|, a_{\max}) \tag{3.5}$$

where $\tilde{\boldsymbol{a}}_i(k+1) = (\tilde{\boldsymbol{v}}_i(k+1) - \tilde{\boldsymbol{v}}_i(k))/dt$. Then, we apply a cutoff on the speed at $v_{\max}$, and get the velocity command $\boldsymbol{v}_i$ of the i-th agent:

$$\boldsymbol{v}_i = \frac{\tilde{\boldsymbol{v}}_i}{\|\tilde{\boldsymbol{v}}_i\|} \min(\|\tilde{\boldsymbol{v}}_i\|, v_{\max}) \tag{3.6}$$

To search the large parameter space of the PF swarm model, we used evolutionary optimization for highest-order flight and lowest number of collisions. The evaluation of the swarm behavior is based on a single fitness function that sums three independent values (Φ_{order}, $\Phi_{\text{agent-safety}}$, and $\Phi_{\text{obs-safety}}$) smaller or equal to 1 (ideal case). The fitness is determined in simulations where the swarm is initialized with random positions in an environment where obstacles are randomly placed. The parameter values and their description are detailed in Table 3.1.

Parameter	Symbol	Unit	Value
Repulsion	c_{rep}	$1/s$	0.29
	d_{ref}	m	0.8
Cohesion	c_{coh}	$1/s$	0.29
	d_{ref}	m	0.8
Friction	c_{fric}	$1/s$	3.34
	C_{fric}	–	0.06
	v_{fric}	m/s	0.63
	a_{fric}	m/s^2	0.05
	$r_{0,\text{fric}}$	m	6.98
Obstacle avoidance	c_{shill}	$1/s$	2.99
	v_{shill}	m/s	0.81
	a_{shill}	m/s^2	1.17
	$r_{0,\text{shill}}$	m	0.10
Migration	$\boldsymbol{v}_{\text{mig}}$	m/s	$(0.5, 0, 0)$

Table 3.1 – **Optimized parameter values of the PF swarm model**. The parameter values are obtained via evolutionary optimization. For the complete explanation on the meaning of the parameters, please refer to [13].

Parameter	Value
Population size	15
Number of generations	60
Maximum stall generation	6
Selection function	tournament
Fitness scaling	proportional

Table 3.2 – **Parameter setting of the evolutionary optimization for the PF swarm**. The optimization algorithm used to compute the optimal parameter values for the PF swarm model was obtained from https://www.mathworks.com/help/gads/genetic-algorithm.html. The most important parameter values are shown in the table below. Parameters not listed here were used with default values. The fitness function used to instantiate the PF swarm model is a linear combination of the order (Φ_{order}), agent-agent safety ($\Phi_{\text{agent-safety}}$), and agent-obstacle safety ($\Phi_{\text{obs-safety}}$), with gains 0.1, 1, and 1, respectively.

3.2.2 Agents' dynamics

The NMPC swarm model supposes the availability of the agents' dynamic model. We assume that every drone of the swarm obeys a discrete linear system, given by:

$$\boldsymbol{x}_i(k+1) = A_i\boldsymbol{x}_i(k) + B_i\boldsymbol{u}_i(k) \tag{3.7}$$

where A_i and B_i are constant matrices. In this chapter, we consider the system to represent a quadrotor with an underlying acceleration controller. The input $\boldsymbol{u}_i$ is an acceleration command and the state $\boldsymbol{x}_i = [\boldsymbol{p}_i, \boldsymbol{v}_i] \in \mathbb{R}^6$ is a vector containing the position and velocity.

We assume that the velocities and acceleration inputs of the agents are bounded by constant vectors $\boldsymbol{v}_{\text{min}}$, $\boldsymbol{v}_{\text{max}}$ and $\boldsymbol{u}_{\text{min}}$, $\boldsymbol{u}_{\text{max}}$ respectively. This translates into the inequalities:

$$\boldsymbol{v}_{\text{min}} \leq \boldsymbol{v}_i \leq \boldsymbol{v}_{\text{max}} \tag{3.8}$$

$$\boldsymbol{u}_{\text{min}} \leq \boldsymbol{u}_i \leq \boldsymbol{u}_{\text{max}} \tag{3.9}$$

Let $x = [\boldsymbol{x}_1, \boldsymbol{x}_2, \dots \boldsymbol{x}_N] \in \mathbb{R}^{6N}$ the positions and velocities of the agents of the swarm, and $\boldsymbol{u} = [\boldsymbol{u}_1, \boldsymbol{u}_2, \dots \boldsymbol{u}_N] \in \mathbb{R}^{3N}$. The system defining the motion of the swarm can be written as:

$$\boldsymbol{x}(k+1) = A\boldsymbol{x}(k) + B\boldsymbol{u}(k) \tag{3.10}$$

where A and B are block diagonal matrices with blocks $A_1, \dots, A_N$ and $B_1, \dots, B_N$, respectively. Parameter values of the agents' dynamics are detailed in Table **??**.

3.2.3 NMPC swarm model

For our NMPC swarm model, we defined behavioral rules similar to those of the PF model. These rules are encoded as four terms of a cost function, including separation, propulsion, direction, and control effort. At each time step, each agent updates its neighbor set and computes the cost associated with the four swarm rules only considering neighboring agents. All agents' contributions are then summed in a global cost function that defines our centralized model. The dynamic model of the agents determines the evolution of the agents' state over a constant time window, called the prediction horizon. These predictions are optimized by the global cost function, whose solution gives the control inputs for the swarm over the so-called control horizon (see Fig. 3.2). The prediction and control horizons are finite and shift forward at every time step. In the following, they will be denoted as $T_P = Pdt$ and $T_C = Cdt$ respectively, with $P \geq C$ and $P, C \in \mathbb{N}^+$.

We let $(\cdot)(k+l|k)$ represent the predicted value of $(\cdot)(k+l)$ with the information available at time $t(k)$ and $l \in \{0, \dots, P\}$. Then, the continuity condition on the swarm state is written as:

$$\boldsymbol{x}(k|k) = \boldsymbol{x}(k) \tag{3.11}$$

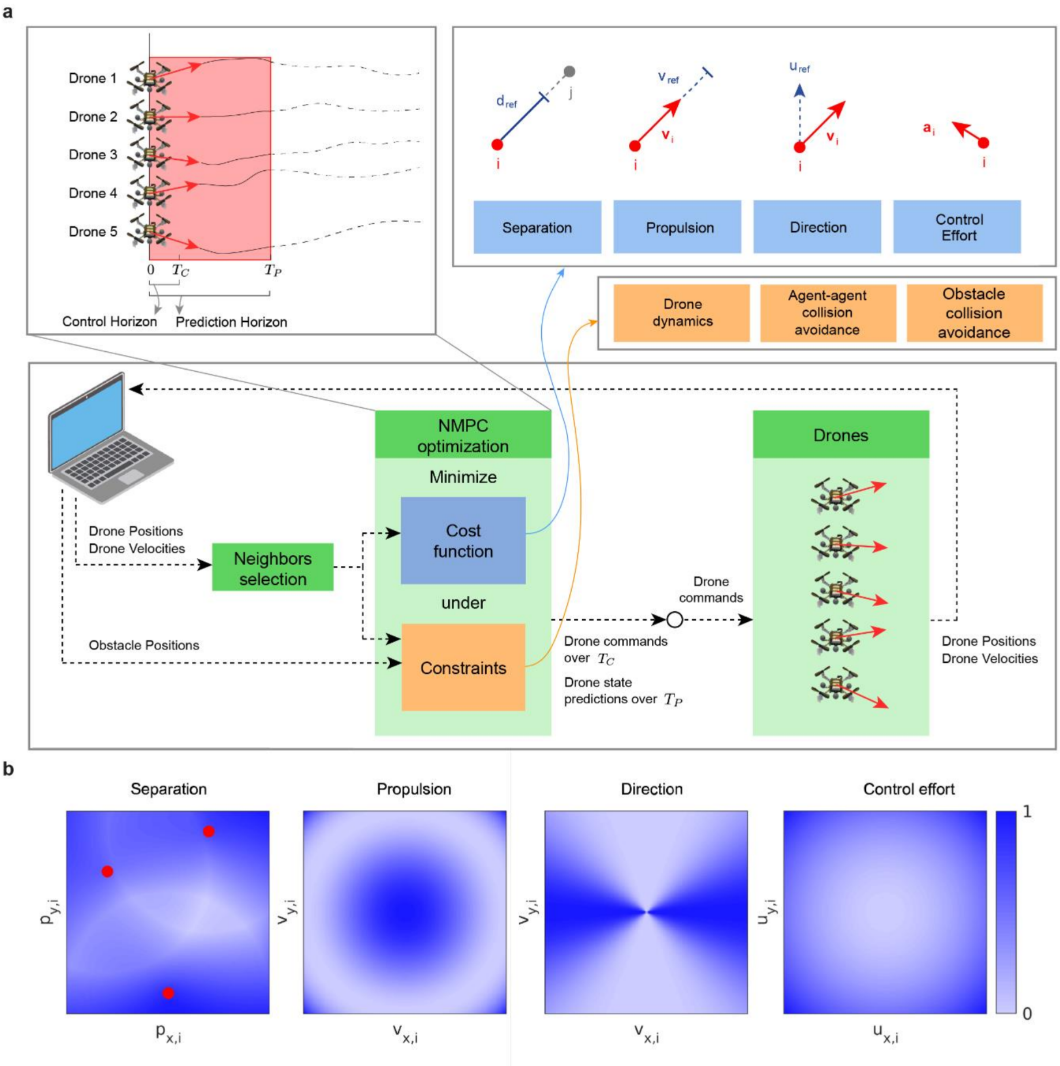


Figure 3.2 – **Experimental setup of NMPC drone swarm flying in cluttered environments.** (a) Illustration of the experimental setup and the environment configuration. A ground control station, equipped with a radio transmitter, computes and sends run-time control commands to the drones. The swarm flies in the 3D space of an indoor flying arena. The drones take off from initial random positions within a predefined start area (red zone). Drones swarm along the preferred migration direction (orange arrow). The mission is completed when all drones cross the arrival plane on the opposite side of the region. (b) Indoor test environment populated with cylindrical obstacles. (c) Components of the drones used for the hardware experiments.

The separation term for agent i and time $t(k)$ is:

$$J_{\text{sep},i}(k) = \sum_{j\in\mathcal{N}_i} \sum_{l=1}^{P} \frac{w_{\text{sep}}}{|\mathcal{N}_i|} (\|\boldsymbol{p}_j(k+l|k) - \boldsymbol{p}_i(k+l|k)\|^2 - {d_{\text{ref}}}^2)^2 \tag{3.12}$$

The separation component incorporates the effects of cohesion and repulsion and drives the inter-agent distances to the preferred value d_{ref}. It is important to notice that commanding inter-agent distances instead of relative positions, as done in problems of formation keeping [33, 34, 36, 37], introduces a non-convexity in the separation term. The propulsion term is:

$$J_{\text{prop},i}(k) = \sum_{l=1}^{P} w_{\text{prop}}(\|\boldsymbol{v}_i(k+l|k)\|^2 - {v_{\text{ref}}}^2)^2 \tag{3.13}$$

The direction term is:

$$J_{\text{dir},i}(k) = \sum_{l=1}^{P} w_{\text{dir}}\left(1 - \frac{(\boldsymbol{v}_i(k+l|k)\cdot\boldsymbol{u}_{\text{ref}})^2}{\|\boldsymbol{v}_i(k+l|k)\|^2}\right)^2 \tag{3.14}$$

The combined action of the propulsion 3.13 and direction 3.14 terms contribute to the so-called migration behavior of the swarm that regulates the two components of the swarm's velocity, i.e., magnitude and direction, respectively. The choice of two separate terms that independently act on the swarm's velocity components is necessary to maintain constant flight speed during obstacle avoidance maneuvers. The control effort is:

$$J_{\text{u},i}(k) = \sum_{l=0}^{P-1} w_{\text{u}}\|\boldsymbol{u}_i(k+l|k)\|^2 \tag{3.15}$$

where w_{sep}, w_{prop}, w_{dir} , and w_u represent the constant weights associated with the cost function terms.

To prevent the agents from colliding with their neighbors or the obstacles, we associated with the cost function two sets of collision avoidance constraints:

$$d_{ij}(k+l|k)^2 \geq d^2_{\text{agent-safety}} \quad i \in \mathcal{V},\ j \in \mathcal{N}_i \tag{3.16}$$

$$d_{im}(k+l|k)^2 \geq d^2_{\text{obs-safety}} \quad i \in \mathcal{V},\ m \in \mathcal{M} \tag{3.17}$$

where $d_{\text{agent-safety}}$ is the safety distance between two agents' positions and $d_{\text{obs-safety}}$ is the safety distance that an agent should keep from the obstacle's position. While in this study we consider collision avoidance with all obstacles (see Eq. 3.17), the model does not necessarily require it. Indeed, the obstacles that do not interfere with the agents' predicted trajectories are discarded by the optimization process. A strategy for reducing the number of constraints in the model consists of considering only the subset of obstacles that are on the collision course with the agents. While this strategy would represent an approximation of more comprehensive modeling of all obstacles, as presented here, the advantages of the model-based approach described here would still hold.

We let $\boldsymbol{X}(k) \in \mathbb{R}^{6NP}$ the stacked sequence of the predicted states $\boldsymbol{x}(k+l|k)$ over the horizon $l \in \{1,\dots,P\}$ and $\boldsymbol{U}(k) \in \mathbb{R}^{3NP}$ the stacked sequence of the predicted control inputs $\boldsymbol{u}(p|k)$ over the horizon $l \in \{0,\dots,P-1\}$. Then, the terms of the cost function and the constraints define the following non-convex optimization problem:

$$\min_{\boldsymbol{X}(k),\boldsymbol{U}(k)} \sum_{i=1}^{N} \left(J_{\text{sep},i}(k) + J_{\text{prop},i}(k) + J_{\text{dir},i}(k) + J_{\text{u},i}(k)\right) \tag{3.18}$$

$$\text{subject to:} \tag{3.19}$$

$$\boldsymbol{x}(k+l+1|k) = A\boldsymbol{x}(k+l|k) + B\boldsymbol{u}(k+l|k) \tag{3.20}$$

$$\boldsymbol{x}(k|k) = \boldsymbol{x}(k) \tag{3.21}$$

$$\boldsymbol{v}_{\text{min}} \le \boldsymbol{v}_i \le \boldsymbol{v}_{\text{max}} \tag{3.22}$$

$$\boldsymbol{u}_{\text{min}} \le \boldsymbol{u}_i \le \boldsymbol{u}_{\text{max}} \tag{3.23}$$

$$d_{ij}(k+l|k)^2 \ge d^2_{\text{agent-safety}} \tag{3.24}$$

$$d_{im}(k+l|k)^2 \ge d^2_{\text{obs-safety}} \tag{3.25}$$

with $l \in \{1,\dots,P\}$, $i \in \mathcal{V}$, $j \in \mathcal{N}_i$, and $m \in \mathcal{M}$.

Parameter	**Symbol**	**Unit**	**Description**
Separation	d_{ref}	m	**Preferred inter-agent distance**. The preferred distance value for neighboring pairs of drones. Larger values create sparser swarms.
	w_{sep}	$1/m$	**Separation weight**. The strength of the separation behavior of neighboring drones (similar to the spring constant in a spring model). The separation encapsulates both the repulsion and the attraction behaviors that tend to bring two agents at the reference inter-agent distance. The repulsion acts when the inter-agent distance is larger than the reference, vice versa the attraction kicks in. High values result in emphasized repulsive and attractive behaviors. We used $w_{\text{sep}} = 1\ m^{-1}$.
Propulsion	v_{ref}	m/s	**Preferred swarm speed**. The preferred speed value of the swarm agents.
	w_{prop}	s/m	**Propulsion weight**. The strength of the propulsion behavior of the drones. Higher values of this gain penalize more the deviation of the swarm speed from the preferred value. However, good speed tracking happens at the expense of the other two behaviors, i.e., separation and direction. We used $w_{\text{prop}} = 5\ m/s$.

Continued on next page

Table 3.2 – continued from previous page

Parameter	**Symbol**	**Unit**	**Description**
Direction	$\boldsymbol{u}_{\text{ref}}$	–	**Preferred swarm direction**. A unit vector expressing the preferred direction of flight of the swarm.
	w_{dir}	–	**Direction weight**. The strength of the direction term. High values give this term a high priority. We used $w_{\text{dir}} = 5$.
Control effort	w_{u}	s^2/m	**Control effort weight**. The strength of the control effort term. High values penalize high accelerations. We used $w_{\text{u}} = 0.4\ s^2/m$.
Collision avoidance	$d_{\text{agent-safety}}$	m	**Safety agent-agent distance**. The distance that every pair of agents should maintain to guarantee agent-agent collision avoidance. It is measured between the centers of two agents, and it depends on the agents' size (r_{agent}). We used $d_{\text{agent-safety}} = 2r_{\text{agent}}$.
	$d_{\text{obs-safety}}$	m	**Safety agent-obstacle distance**. The distance that every pair of agent-obstacle should maintain to guarantee collision avoidance. It is measured between the center of an agent and the center of an obstacle, and it depends on the agents' size (r_{agent}) and the obstacle size (r_{obs}). In the case of real-world experiments, we also accounted for an extra safety margin (r_{margin}), hence $d_{\text{obs-safety}} = r_{\text{agent}} + r_{\text{obs}} + r_{\text{margin}}$.
Agents dynamics	A_i	–	**State matrix**. A 6x6 matrix representing a double integrator agent, the quadrotor. Our hardware experiments confirm that when the agents are closed-loop controlled by an onboard low-level controller (as described in the Drone experimental setup section) a linear model can describe them appropriately within the bounds that we have defined.
	B_i	–	**Input matrix**. 6x3 matrix representing an acceleration input.
	$v_{\max}$	m/s	**Maximum speed**. Maximum linear speed in x, y, and z of the agents. We used $v_{\max} = 5/\sqrt{3}\ m/s$.

Continued on next page

Table 3.2 – continued from previous page

Parameter	Symbol	Unit	Description
	$u_{\max}$	m/s^2	**Maximum acceleration**. Maximum linear acceleration in x, y, and z of the agents. We used $a_{\max} = 2/\sqrt{3}\ m/s^2$.

Table 3.3 – **Parameter description of the NMPC swarm model**. List and description of the NMPC swarm model parameters.

3.2.4 Simulation setup

We implemented our NMPC model in MATLAB with the help of acados [71], an open-source library for fast nonlinear optimal control. This software relies on C code generation for speeding up the computation in real-time applications. The system dynamics and the constraints of the problem are discretized by the library over the prediction horizon to obtain a structured Nonlinear Program (NLP). Then, the NLP is approximated through Sequential Quadratic Programming (SQP) that iteratively solves convex Quadratic Program (QP) sub-problems. After applying a condensing step, a linear algebra solver, HPIPM, based on the Interior Point (IP) method finds the solution of the sub-problems []. We run our simulations on a DELL Precision Tower with a 3.6 GHz Intel Core i7-7700 processor and 16 GB 2400 MHz RAM, where we set the maximum number of SQP to 7 and the maximum number of QP iterations to 7.

3.2.5 Drone experimental setup

In our experiments, we used five Bitcraze Crazyflie 2.1 quadrotors (Fig. 3.1c). Each quadrotor is equipped with a 3-axis accelerometer, a 3-axis gyroscope, a pressure sensor, and a marker deck for hosting passive reflective markers. The microcontroller is a STM32F4 running at 168MHz, on which both state estimation and low-level control are running. An OptiTrack motion capture system was used to track the position of the robots. All the acceleration commands for the drones were computed on a single computer with our NMPC model, integrated into position and velocity commands, and broadcast to the swarm through a radio-link alongside the estimated position of each drone. The estimated positions were used by the drones to perform the lower-level control loops and track the commands sent. The positions and velocities used by the swarm model were predicted with the agents' dynamic model. To guarantee the transferability of the NMPC swarm model to hardware experiments, we decreased the number of maximum SQP to 4. This was sufficient to compute converging solutions of the NLP in less than 0.1 s.

3.3 Results

For the performance assessment of the swarm models, we set up a forest-like environment that consists of a rectangular flight region populated with cylindrical obstacles (Fig. 3.1a). At the experiment onset, we place five drones at random positions within a predefined start area on one side of the region (Fig. 3.1a, red zone) and let the swarm fly through the region along the migration direction (Fig. 3.1a, orange arrow). The mission is completed when all drones cross the arrival plane (Fig. 3.1a, orange plane) on the opposite side of the region.

We assess the quality of the aerial swarm's flight considering eight different metrics. The mission completion time T measures the time that the swarm requires to cross the region. The inter-agent distance error E_d measures the agents' deviation from the preferred distance d_{ref}, and the inter-agent distance range R_d measures the range in which the inter-agent distances vary (defined by the minimum and maximum inter-agent distance over time). The speed error E_v measures the deviation of the agents' speeds from the preferred migration speed v_{ref}, and the speed range R_v measures the range in which the agents' speeds vary. E_d, R_d, E_v and R_v take values greater than or equal to 0 (ideal case). We determine the swarm's level of synchronization by calculating the directional correlation of the agents' movements, expressed by the so-called order Φ_{order}. Φ_{order} takes values between -1 (complete disorder) and 1 (perfect order). Finally, the agent-agent safety $\Phi_{\text{agent-safety}}$ assesses the ability of the swarm's agents to avoid collisions among themselves, and the agent-obstacle safety $\Phi_{\text{obs-safety}}$ assesses the ability of the agents to avoid collisions with the obstacles. $\Phi_{\text{agent-safety}}$ and $\Phi_{\text{obs-safety}}$ take values between 0 (complete unsafety) and 1 (perfect safety, i.e., zero collisions) (see Table 3.4 for mathematical formulation). To evaluate the overall performance of the swarm during a mission, we compute the average and standard deviation of these metrics. For the instantaneous evaluation of the swarm over time, we additionally plot the inter-agent distance and speed, and the distance to obstacles, from which we can appreciate their respective errors and ranges, and the occurrence of collisions.

We extensively tested the proposed NMPC swarm model in simulation and compared it to a reactive PF model that has been recently described and validated on 30 real drones []. In addition to the repulsion and obstacle avoidance rules, the PF model includes a friction rule to reduce velocity oscillations. In order to ensure cohesive goal-directed flight in open environments, we added the rules of cohesion and migration to the PF model. As in previous work [], we used evolutionary optimization to search the large parameter space of the PF swarm model, and favored swarms with highly ordered flight ($\Phi_{\text{order}} = 1$) and a low number of agent-agent and agent-obstacle collisions ($\Phi_{\text{agent-safety}} = 1$, $\Phi_{\text{agent-safety}} = 1$) (see Table 3.2 and 3.1). The purpose of the experimental comparison between NMPC swarming and PF swarming is to emphasize behavioral differences and performance advantages of the proposed NMPC swarm model. However, the choice of a swarm model for the deployment on physical drones should also consider computational resources, which are significantly larger for NMPC swarming.

Below we present three sets of simulation experiments: (i) we compare the performance metrics of the two models in the same environmental conditions, (ii) we investigate the adaptability of the PF and NMPC swarm models to environments with different obstacle density, and (iii) we study the scalability of the NMPC swarm model at different preferred speeds and inter-drone distances. Finally, we experimentally validate the NMPC swarm model with five palm-sized drones (Fig. 3.1c) flying through a room with cylindrical obstacles (Fig. 3.1b).

Metric	Formula	Description
T	–	**Mission completion time.** The time that takes for all agents to navigate the environment and cross the arrival plane.
$E_d(k)$	$\sum_{i\in\mathcal{V}} \frac{\sum_{j\in\mathcal{N}_i} \lvert d_{ij}(k)-d_{\text{ref}}\rvert}{N\lvert\mathcal{N}_i\rvert d_{\text{ref}}}$	**Inter-agent distance error.** The error of the inter-agent distances normalized by d_{ref}. A value equal to 1 indicates an inter-agent distance error of 100%. While a value equal to 0 indicates that all distances between neighboring agents match the preferred value d_{ref}.
$R_d(k)$	$\max_{i\in\mathcal{V},j\in\mathcal{N}_i}(d_{ij}(k)) - \min_{i\in\mathcal{V},j\in\mathcal{N}_i}(d_{ij}(k))$	**Inter-agent distance range.** The range in which the inter-agent distances vary, normalized by d_{ref}.
$E_v(k)$	$\sum_{i\in\mathcal{V}} \frac{\lvert v_i(k)-v_{\text{ref}}\rvert}{N v_{\text{ref}}}$	**Speed error.** The error of the agents' speeds normalized by v_{ref}. A value equal to 1 indicates a speed error of 100%. While a value equal to 0 indicates that all agents' speeds match the preferred value v_{ref}.
$R_v(k)$	$\max_{i\in\mathcal{V}}(v_i(k)) - \min_{i\in\mathcal{V}}(v_i(k))$	**Speed range.** The range in which the agents' speeds vary, normalized by v_{ref}.
$\Phi_{\text{order}}(k)$	$\sum_{i\in\mathcal{V}} \frac{\sum_{j\in\mathcal{N}_i} \boldsymbol{v}_i(k)\cdot\boldsymbol{v}_j(k)}{N\lvert\mathcal{N}_i\rvert \lVert\boldsymbol{v}_i(k)\rVert \lVert\boldsymbol{v}_j(k)\rVert}$	**Order.** The correlation between the agents' flight directions. It is equal to 1 when all agents' velocities are pointing in the same direction (ideal case), while it is equal to 0 when, for instance, they are diametrically opposed in pairs.

Continued on next page

Table 3.3 – continued from previous page

Metric	Formula	Description
$\Phi_{\text{agent-safety}}(k)$	$1 - \frac{N_{\text{agent-coll}}(k)}{N(N-1)}$ with $N_{\text{agent-coll}}(k) = \lvert\{d_{ij}(k) \mid i \in \mathcal{V}, j \neq i, d_{ij}(k) < d_{\text{obs-coll}}\}\rvert$	**Agent-agent safety.** A measure of the number of collisions among the swarm's agents. It is equal to 1 when no collision is registered (ideal case), while it is equal to 0 when all pairs of agents are colliding (worst case).
$\Phi_{\text{obs-safety}}(k)$	$1 - \frac{N_{\text{agent-coll}}(k)}{N(N-1)}$ with $N_{\text{agent-coll}}(k) = \lvert\{d_{ij}(k) \mid i \in \mathcal{V}, j \neq i, d_{ij}(k) < d_{\text{obs-coll}}\}\rvert$	**Agent-obstacle safety.** A measure of the number of collisions between the swarm agents and the obstacles. Like the agent-agent safety, the agent- obstacle safety is equal to 1 when no collision is registered (ideal case), while it is equal to 0 when all agents are colliding with at least one obstacle (worst case).

Table 3.4 – **Metrics to measure the performance of the NMPC drone swarm.** The table describes the metrics used to assess the performance of the swarm models. Except for the mission completion time T, which is global, i.e. it refers to the entire mission, all other metrics are defined instantaneously, i.e. for every time instant $t(k)$ with $k \in \{0,\dots,K\}$ and $T = Kdt$. Therefore, to evaluate a mission we consider the average and standard deviation of the metrics. The computation of E_d, E_v, R_d, R_v excludes an initial transient period of $\tilde{t} = 3\ s$.

3.3.1 Comparison of PF and NMPC aerial swarms

Both PF and NMPC swarms navigated around the obstacles without collisions (Fig. 3.3e), but the NMPC swarm completed the mission due to the ability of the NMPC swarm to track the preferred speed v_{ref} more consistently ($E_v = 0.02 \pm 0.02$, $R_v = 0.08 \pm 0.07$) than PF swarm ($E_v = 0.39 \pm 0.15$, $R_v = 0.47 \pm 0.15$) (Fig. 3.3c). The NMPC swarm also generated a smaller inter-agent distance error ($E_d = 0.11 \pm 0.02$) and range ($R_d = 0.55 \pm 0.18$) compared to the PF swarm ($E_d = 0.26 \pm 0.15$, $R_d = 0.90 \pm 0.26$) (Fig. 3.3b). The NMPC model generated almost perfectly ordered flight manoeuvres throughout the entire flight ($\Phi_{\text{order}} = 0.98 \pm 0.02$) while the PF model displayed lower and more variable order ($\Phi_{\text{order}} = 0.78 \pm 0.17$) (Fig. 3.3d). Neither the NMPC nor the PF swarm presented agent-agent or agent-obstacle collisions ($\Phi_{\text{agent-safety}} = 1 \pm 0$, $\Phi_{\text{obs-safety}} = 1 \pm 0$) (Fig. 3.3e). While optimizing the swarm's objectives, the NMPC model reduced the minimum distance to obstacles to 0.03 m. In comparison, the PF swarm achieved a minimum distance to obstacles of 0.14 m. This difference is due to the fact that in the PF model the obstacles apply a repulsion force on the agents in their proximity, while in the

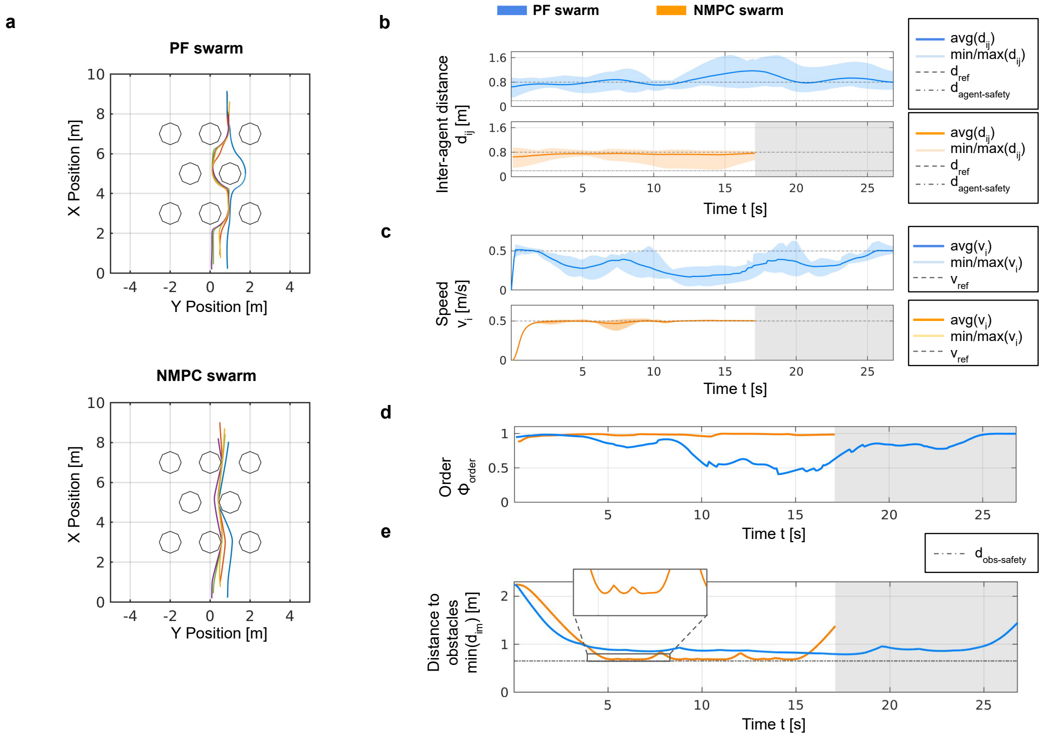


Figure 3.3 – **Comparison between the PF-based model and our NMPC swarm model in simulation experiments**. **(a)** Top views of the 3D trajectories of five drones flying in a cluttered environment with the PF (top) and the NMPC models (bottom). The circular objects on the map corresponds to cylindrical obstacles. **(b)** Inter-agent distance average (solid line) and range (shaded region). The curve on top (blue) refers to the PF swarm, while the one at the bottom (orange) refers to the NMPC swarm. **(c)** Swarm speed average (solid line) and range (shaded region). **(d)** Order metric, Φ_{order} . **(e)** Distance to obstacles, $\min(d_{im})$ expressed as the minimum distance between the swarm's agents and the set of obstacles.

NMPC model there is no penalty for approaching the obstacles. As a consequence, when implementing the NMPC model on a real-world swarm, the user should carefully choose a safety margin.

3.3.2 Environments with different obstacle densities

We tested the PF and the NMPC swarm models for three different obstacle densities (Case A: 0.06, B: 0.12, and C: 0.20) to quantify the impact on the swarms' performance. The obstacles occupy random positions on the map, but they have a homogenous distribution (Fig. 3.4a and 3.4d). The initial positions of the drones are random. In Fig. 3.4, we show the evolution of the inter-agent distance and speed for the scenario with the highest obstacle density (Case C) and for both models. The results show that the inter-agent distance error is smaller with NMPC swarms ($E_d = 0.11 \pm 0.02$) than with PF swarms ($E_d = 0.27 \pm 0.12$), and the inter-agent distance

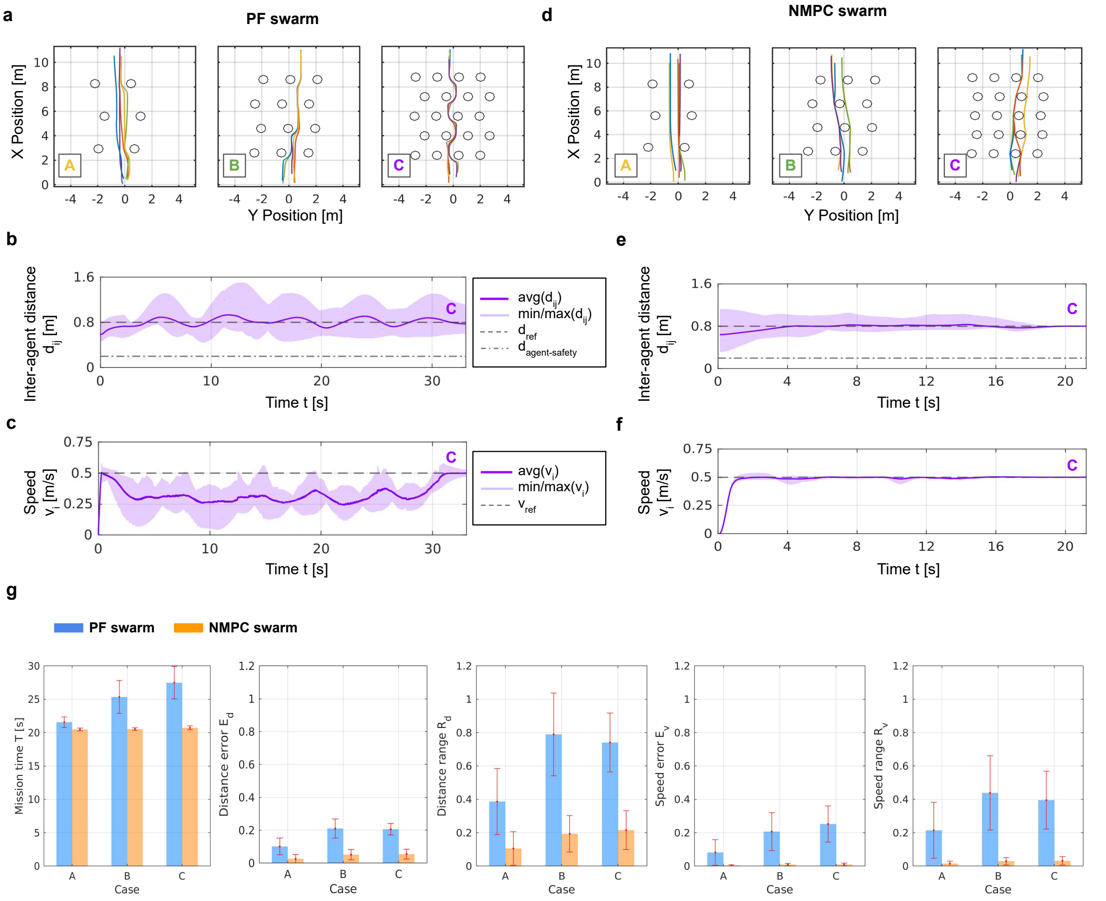


Figure 3.4 – **Comparison of the PF and NMPC swarm deployment in environments with different obstacle densities**. (a, d) Top views of the 3D simulated trajectories of the PF and the NMPC swarms in environments with three different obstacle densities. The density increases from left to right (Case A: 0.06, B: 0.12, and C: 0.20). (b, c) Inter-agent distance and speed of the PF swarm in Case C. (e, f) Inter-agent distance and speed of the NMPC swarm in Case C. (g) Aggregated results (average and standard deviation) of 10 stochastic simulations of the PF (blue) and NMPC (orange) swarm models in Cases A, B, and C. The represented metrics are the mission time T, the distance error E_d, the distance range R_d, the speed error E_v, and the speed range R_v (see Table 3.4).

range is shorter for NMPC swarms ($R_d = 0.56 \pm 0.18$) than with PF swarms ($R_d = 0.90 \pm 0.26$) (Fig. 3.4b and 3.4e). The NMPC swarms tracked the preferred speed vref more precisely ($E_v = 0.03 \pm 0.02$) than the PF swarms ($E_v = 0.39 \pm 0.15$), and the speed range was shorter ($R_v = 0.08 \pm 0.07$ and 0.47 ± 0.15, respectively) (Fig. 3.4c and 3.4f). The faster speed of NMPC swarms resulted in faster mission completion time than the PF swarms ($T = 21.5$ s and 34.1 s, respectively).

To assess the reproducibility of the results, we performed ten stochastic simulations for each

Parameter	Unit	Description	Value
d_{ref}	m	Preferred (or reference) value for the inter-agent distance	0.8
v_{ref}	m/s	Preferred (or reference) value for the swarm speed	0.5
$\boldsymbol{u}_{\text{ref}}$	-	Preferred migration direction	(1 0 0)
L_{map}	m	Length of an edge of the square flight region (or map)	10
r_{obs}	m	Obstacles radius	0.35
ρ_{obs}	$1/m^2$	Obstacle density	Case A: 0.06, Case B: 0.12, Case C: 0.20

Table 3.5 – **Swarm and environment configurations of the simulation experiments with different obstacle densities**. The same configurations are used for both the PF and the NMPC swarm models.

of the three obstacle densities and for the two swarm models, and we report here aggregated performance results (Fig. 3.4g). While the speed error in the NMPC swarm is small and constant for all obstacle densities (Case A: $E_v = 0.01 \pm 0.01$, B: 0.01 ± 0.01, C: 0.01 ± 0.01), it is larger and increases with larger obstacles densities in the PF swarm (Case A: $E_v = 0.08 \pm 0.07$, B: 0.21 ± 0.11, C: 0.25 ± 0.11). As a consequence, the mission completion time of the PF swarm is increased when increasing the obstacle density (Case A: $T = 21.56 \pm 0.81$ s, B: 25.35 ± 2.46 s, C: 27.48 ± 2.43 s), while for the NMPC swarm it is shorter and it stays almost constant across the different densities (Case A: $T = 20.47 \pm 0.22$ s, B: 20.54 ± 0.21 s, C: 20.72 ± 0.28 s). Also, the PF swarm's order deteriorates when increasing the obstacle density (Case A: $\Phi_{\text{order}} = 0.98 \pm 0.03$, B: 0.92 ± 0.08, C: 0.81 ± 0.08), while for the NMPC swarm it stays almost constant (Case A: $\Phi_{\text{order}} = 0.99 \pm 0.01$, B: 0.98 ± 0.02, C: 0.98 ± 0.02). While the NMPC swarm produces collision-free movements in all cases, for the PF swarm we observe some agent-obstacle collisions at high obstacle densities (Case A: $\Phi_{\text{obs-safety}} = 1 \pm 0$, B: $(99.98 \pm 0.06)10^{-2}$, C: $(99.99 \pm 0.02)10^{-2}$). The aggregated performance results are summarized in Table 3.6.

3.3.3 Scalability to different inter-agent distances and speeds

We assess the scalability of the proposed NMPC model to different values of the preferred inter-agent distance (Case A: $d_{\text{ref}} = 0.5$ m, B: 1.0 m, and C: 1.5 m, see Fig 3.7a-c) and speed (Case A: $v_{\text{ref}} = 0.5$ m/s, B: 1.0 m/s, and C: 1.5 m/s, see Fig 3.7 d-f) in the same environmental conditions. We analyze the swarm's inter-agent distance and speed and quantify their respective errors and ranges. The results show that at different inter-agent distance levels the swarm inter-agent distance converged to the preferred value with comparable errors (Case A: $E_d = 0.05 \pm 0.06$, B: 0.01 ± 0.02, C: 0.02 ± 0.03, see Fig. 3.7b). The swarm's speed error is almost zero in the three cases (see Fig. 3.7c), and it resulted in similar mission times (Case A: $T = 20$ s, B: 21 s, and C:

Metric	Unit	Case A		Case B	
		PF	NMPC	PF	NMPC
T	s	21.56 ± 0.81	20.47 ± 0.21	25.34 ± 2.45	20.54 ± 0.21
E_d	–	0.10 ± 0.05	0.03 ± 0.02	0.21 ± 0.05	0.05 ± 0.03
E_v	–	0.08 ± 0.08	0.00 ± 0.00	0.21 ± 0.11	0.01 ± 0.01
R_d	–	0.39 ± 0.20	0.10 ± 0.11	0.79 ± 0.25	0.19 ± 0.11
R_v	–	0.21 ± 0.17	0.01 ± 0.01	0.44 ± 0.22	0.03 ± 0.02
Φ_{order}	–	0.98 ± 0.03	0.99 ± 0.01	0.92 ± 0.08	0.98 ± 0.02
$\Phi_{\text{agent-safety}}$	–	1 ± 0	1 ± 0	1 ± 0	1 ± 0
$\Phi_{\text{obs-safety}}$	–	1 ± 0	1 ± 0	$(99.98 \pm 0.06)10^{-2}$	1 ± 0

Metric	Unit	Case C	
		PF	NMPC
T	s	27.48 ± 2.43	20.72 ± 0.28
E_d	–	0.20 ± 0.05	0.05 ± 0.03
E_v	–	0.25 ± 0.11	0.01 ± 0.01
R_d	–	0.74 ± 0.18	0.22 ± 0.12
R_v	–	0.39 ± 0.17	0.03 ± 0.02
Φ_{order}	–	0.81 ± 0.08	0.98 ± 0.02
$\Phi_{\text{agent-safety}}$	–	1 ± 0	1 ± 0
$\Phi_{\text{obs-safety}}$	–	$(99.99 \pm 0.02)10^{-2}$	1 ± 0

Table 3.6 – **Aggregate performance of the simulated PF and the NMPC swarms at varying obstacle densities.** The table reports numeric results (in terms of average and standard deviation) of stochastic simulations of the PF and the NMPC swarm models deployed in environments with three different obstacle densities (Case A: 0.06, B: 0.12, and C: 0.20). For every model and configuration, we ran 10 experiments. The computation of E_d, E_v, R_d, R_v excludes an initial transient period of $\tilde{t} = 3$ s.

21.2 s). We did not observe collisions. Regarding the experiments on the scalability in speed, the speed error E_v was close to zero in the three cases (Fig. 3.7e), while the mission times were decreasing with the increase of the speed (Case A: $T = 20.2$ s, B: 10.5 s, and C: 7.5 s). However, the variability of the inter-agent distance in Case C is higher ($R_d = 0.46 \pm 0.05$) than in Cases A ($R_d = 0.13 \pm 0.11$) and B ($R_d = 0.19 \pm 0.03$) (Fig. 3.7f). Indeed, when the agents turn around the obstacle in the middle of the scene, they rearrange and increase their distance. Also in these experiments, we did not observe collisions. Comparative results on the PF swarm are in Fig. 3.6. Aggregate results of stochastic simulations for each of the preferred inter-agent distance and speed values, and for both the PF and the NMPC models are in Fig. 3.7, and in Tables 3.7 and 3.8.

3.4 Discussion

This chapter shows that a Nonlinear Model Predictive Control (NMPC) model achieves a faster and more synchronized flight in cluttered environments as compared to state-of-the-art

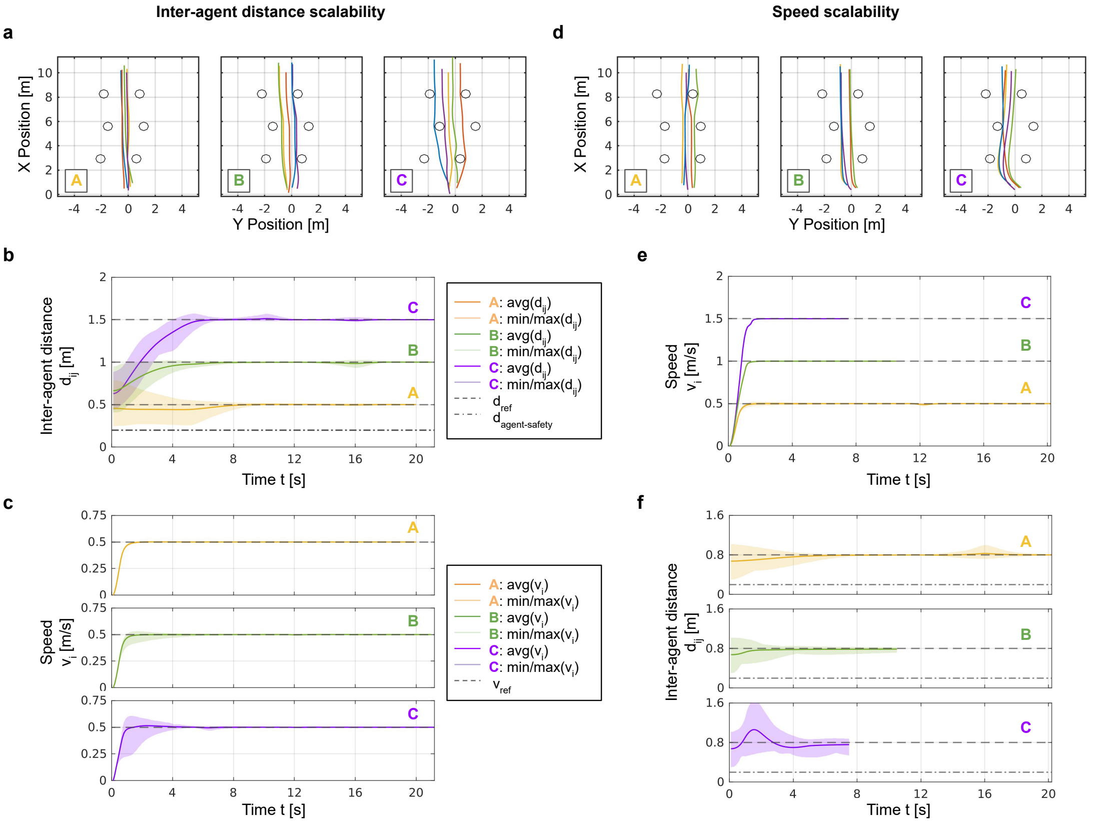


Figure 3.5 – **Scalability of the NMPC swarm in inter-agent distance and speed.** On the left, simulation results on the scalability of the NMPC swarm model in the inter-agent distance for three preferred distance values (Case A: d_{ref}=0.5 m, B: 1.0 m, and C: 1.5 m). On the right, simulation results on the scalability in the swarm speed for three preferred speed values (Case A: v_{ref}=0.5 m/s, B: 1.0 m/s, and C: 1.5 m/s). (a, d) Top views of the 3D trajectories of the swarm. (b, c) Inter-agent distance and speed for the experiment on the inter-agent distance scalability. (e, f) Inter-agent distance and speed for the experiment on the speed scalability. The obstacle size and density are the same for the six cases.

models based on potential fields (PFs). NMPC swarms report no collisions in cluttered environments, they better attain and maintain target speeds, and they remain more ordered and cohesive. The benefits brought by predictive controllers to robotic aerial swarms confirm a parallel with biological systems, where individuals are thought to enhance their synchronization by future state projection [27].

In robotics, the advantages of the NMPC method are promising for applications that require navigation in crowded scenarios, such as the exploration of urban environments, collapsed buildings, or forests [72, 73]. Also, vision-based swarms could benefit from all these features since the reliability of reciprocal visual detection of the drones strongly depends on their distance, and NMPC swarms showed that they can better maintain target inter-agent

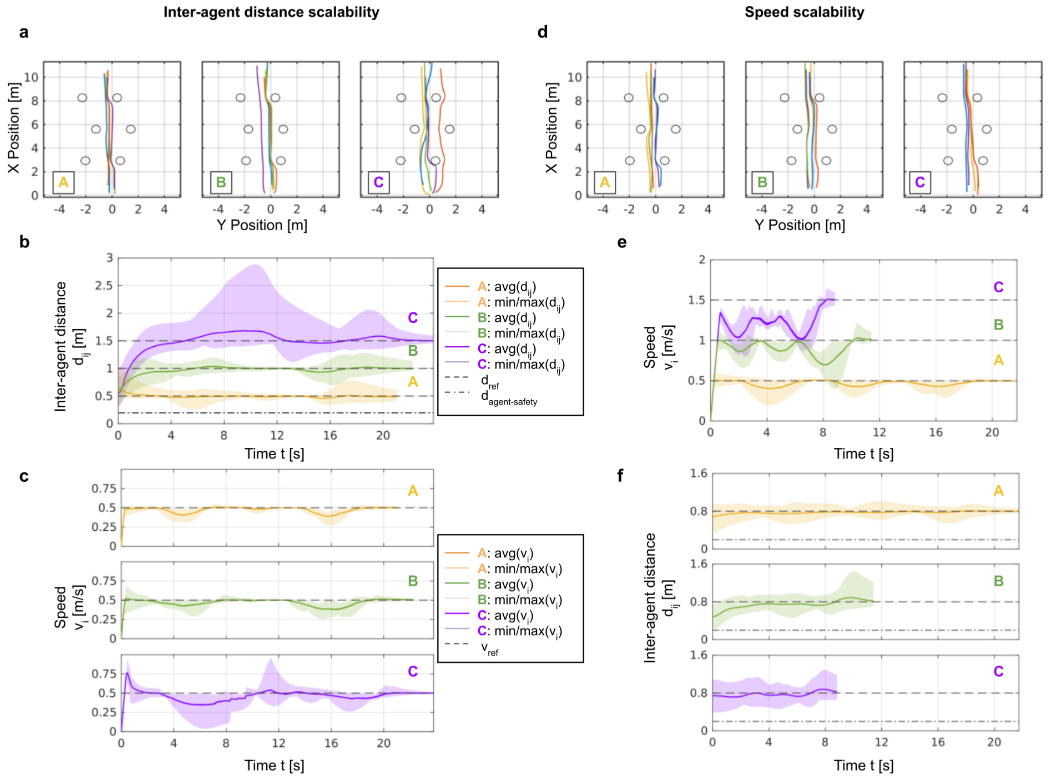


Figure 3.6 – **Scalability of the PF swarm in inter-agent distance and speed.** On the left, simulation results on the scalability of the NMPC swarm model in the inter-agent distance for three preferred distance values (Case A: d_{ref}=0.5 m, B: 1.0 m, and C: 1.5 m). On the right, simulation results on the scalability in the swarm speed for three preferred speed values (Case A: v_{ref}=0.5 m/s, B: 1.0 m/s, and C: 1.5 m/s). (a, d) Top views of the 3D trajectories of the swarm. (b, c) Inter-agent distance and speed for the experiment on the inter-agent distance scalability. (e, f) Inter-agent distance and speed for the experiment on the speed scalability. The obstacle size and density are the same for the six cases.

distances [47, 74]. Overall, predictive methods can improve the autonomy of swarm operations as well as the safety of the swarm and the environment, which are both essential elements to build public confidence in the use of swarms [75].

For our experiments, we relied on a central computing node that generates the motion of the agents at run time according to local interactions only. This assumption simplifies the implementation since it requires only one computer, acting as a ground control station, instead of several onboard computers that the agents would carry. However, the NMPC model requires a higher amount of computational resources than the PF model and scales worse with the swarm size. It will be interesting to develop a decentralized NMPC model where the computational costs are independent of the number of agents. Work in this direction will allow to scale our approach to swarms of larger size.

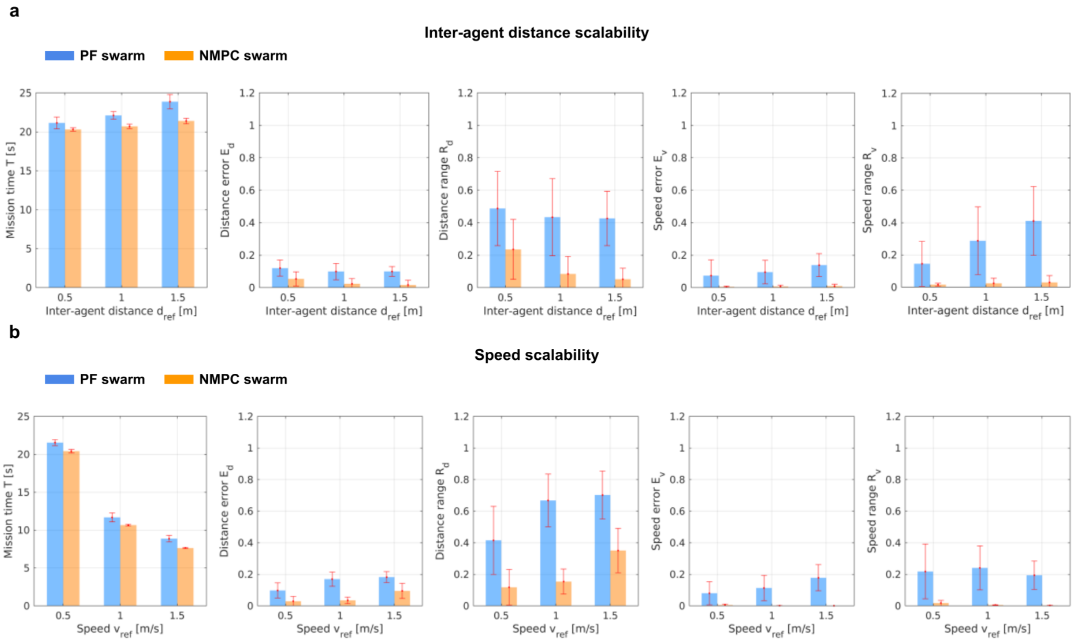


Figure 3.7 – **Aggregate performance of the simulated PF and the NMPC swarms for varying inter-agent distances and speeds.** (a) Aggregate results of the PF and NMPC swarms for three different inter-agent distances (Case A: $d_{\text{ref}} = 0.5\ m$, B: $1.0\ m$, and C: $1.5\ m$). (b) Aggregate results of the PF and NMPC swarms for three different swarm speeds (Case A: $v_{\text{ref}} = 0.5\ m/s$, B: $1.0\ m/s$, and C: $1.5\ m/s$). We performed 10 stochastic simulations for every configuration. Numeric results of the same experiments can be found in Tables 3.7 and 3.8.

Finally, our results motivate future works to address research questions in the design of robust swarm models in dynamic environments. Thanks to their recursive structure, MPC controllers offer a promising method to allow navigation in scenarios with moving obstacles. However, a generalization of the proposed model to dynamic environments would require theoretical and numerical investigation on the conditions for stability, as well as a reliable estimation of the obstacles' motion.

Metric	**Unit**	**Case A**		**Case B**	
		PF	**NMPC**	**PF**	**NMPC**
T	s	21.15 ± 0.75	20.31 ± 0.22	22.11 ± 0.49	20.71 ± 0.31
E_d	–	0.12 ± 0.05	0.05 ± 0.04	0.09 ± 0.05	0.02 ± 0.03
E_v	–	0.07 ± 0.10	0.00 ± 0.00	0.10 ± 0.07	0.01 ± 0.01
R_d	–	0.49 ± 0.23	0.23 ± 0.18	0.43 ± 0.24	0.08 ± 0.10
R_v	–	0.14 ± 0.14	0.01 ± 0.01	0.29 ± 0.21	0.02 ± 0.03
Φ_{order}	–	0.99 ± 0.02	0.99 ± 0.01	0.96 ± 0.05	0.98 ± 0.03
$\Phi_{\text{agent-safety}}$	–	$(99.98 \pm 0.13)10^{-2}$	1 ± 0	1 ± 0	1 ± 0
$\Phi_{\text{obs-safety}}$	–	1 ± 0	1 ± 0	$(99.99 \pm 0.04)10^{-2}$	1 ± 0

Metric	**Unit**	**Case C**	
		PF	**NMPC**
T	s	23.87 ± 0.91	21.40 ± 0.35
E_d	–	0.10 ± 0.03	0.02 ± 0.03
E_v	–	0.14 ± 0.07	0.01 ± 0.01
R_d	–	0.43 ± 0.17	0.05 ± 0.07
R_v	–	0.41 ± 0.21	0.03 ± 0.04
Φ_{order}	–	0.91 ± 0.13	0.96 ± 0.11
$\Phi_{\text{agent-safety}}$	–	1 ± 0	1 ± 0
$\Phi_{\text{obs-safety}}$	–	1 ± 0	1 ± 0

Table 3.7 – **Aggregate performance of the simulated PF and the NMPC swarms at varying inter-agent distances.** The table reports numeric results (in terms of average and standard deviation) of stochastic simulations of the PF and the NMPC swarm models deployed in environments with three different inter-agent distances (Case A: 0.5 m, B: 1.0 m, and C: 1.5 m). For every model and configuration, we ran 10 experiments. The computation of E_d, E_v, R_d, R_v excludes an initial transient period of $\tilde{t} = 3$ s.

Metric	**Unit**	**Case A**		**Case B**	
		PF	**NMPC**	**PF**	**NMPC**
T	s	21.54 ± 0.40	20.42 ± 0.25	11.68 ± 0.58	10.66 ± 0.11
E_d	–	0.10 ± 0.05	0.03 ± 0.03	0.17 ± 0.04	0.03 ± 0.02
E_v	–	0.08 ± 0.08	0.00 ± 0.00	0.11 ± 0.08	0.00 ± 0.00
R_d	–	0.41 ± 0.21	0.12 ± 0.11	0.67 ± 0.17	0.15 ± 0.08
R_v	–	0.22 ± 0.17	0.02 ± 0.02	0.24 ± 0.14	0.01 ± 0.00
Φ_{order}	–	0.98 ± 0.03	0.99 ± 0.01	0.98 ± 0.08	0.96 ± 0.02
$\Phi_{\text{agent-safety}}$	–	1 ± 0	1 ± 0	1 ± 0	1 ± 0
$\Phi_{\text{obs-safety}}$	–	1 ± 0	1 ± 0	$(99.87 \pm 0.37)10^{-2}$	1 ± 0

Metric	**Unit**	**Case C**	
		PF	**NMPC**
T	s	8.87 ± 0.44	7.64 ± 0.07
E_d	–	0.18 ± 0.03	0.09 ± 0.05
E_v	–	0.17 ± 0.08	0.00 ± 0.00
R_d	–	0.70 ± 0.15	0.35 ± 0.14
R_v	–	0.19 ± 0.09	0.00 ± 0.00
Φ_{order}	–	0.98 ± 0.08	0.94 ± 0.02
$\Phi_{\text{agent-safety}}$	–	1 ± 0	1 ± 0
$\Phi_{\text{obs-safety}}$	–	$(99.95 \pm 0.17)10^{-2}$	1 ± 0

Table 3.8 – **Aggregate performance of the simulated PF and the NMPC swarms at varying speeds.** The table reports numeric results (in terms of average and standard deviation) of stochastic simulations of the PF and the NMPC swarm models deployed in environments with three different speeds (Case A: 0.5 m/s, B: 1.0 m/s, and C: 1.5 m/s). For every model and configuration, we ran 10 experiments. The computation of E_d, E_v, R_d, R_v excludes an initial transient period of $\tilde{t} = 3$ s.

4 Distributed predictive control of aerial drone swarms

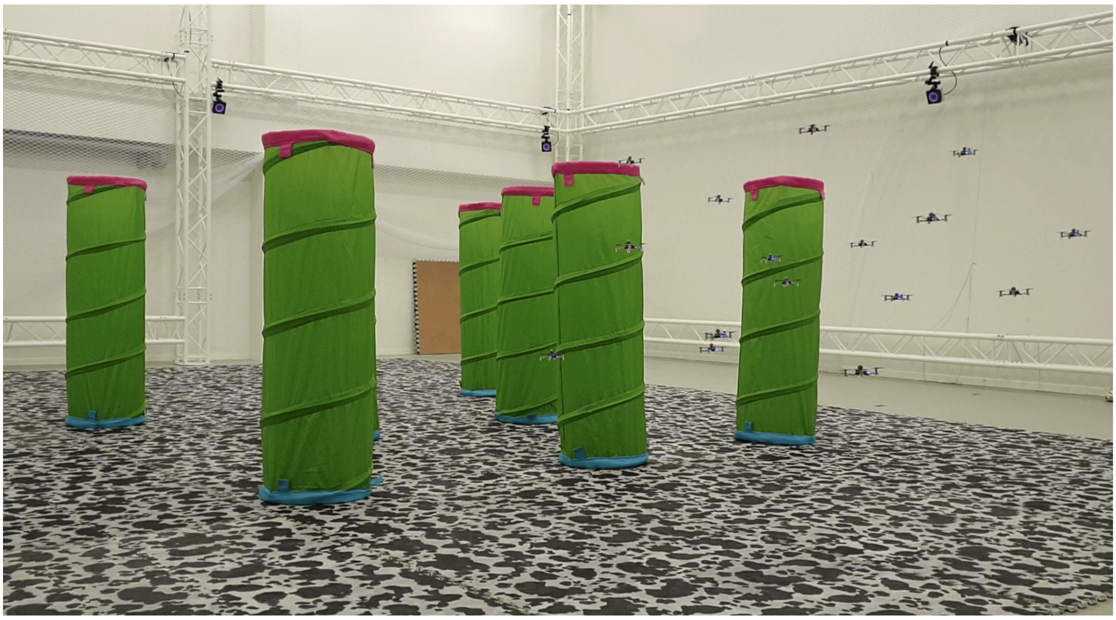
In the previous chapter, we showed that predictive models have the potential to incorporate safe collision avoidance capabilities and give the agents the ability to anticipate and synchronize their trajectories in real-time. However, our previous model makes use of a centralized formulation, which does not allow scalability to a large number of agents. Here, we propose a distributed predictive swarm model that generates self-organized, safe, and cohesive trajectories by solving an optimization problem in real-time. In simulation, we show that our method is scalable to large numbers of agents and suitable for deployment in different environments, specifically a forest and a funnel-like environment. Furthermore, our results show that the agents are capable of collision-free flight with noisy sensor measurements for a noise level of up to 70% of the magnitude of the agent safety distance. Real-world experiments with a swarm of up to 16 quadrotors in an indoor artificial environment validate our method.

The work presented in this chapter is adapted from [44]:

- E. Soria, F. Schiano, D. Floreano, “Distributed predictive drone swarms in cluttered environments,” in *IEEE Robotics and Automation Letters (RA-L)*, vol. 7, no. 1, pp. 73-80, Jan. 2022.

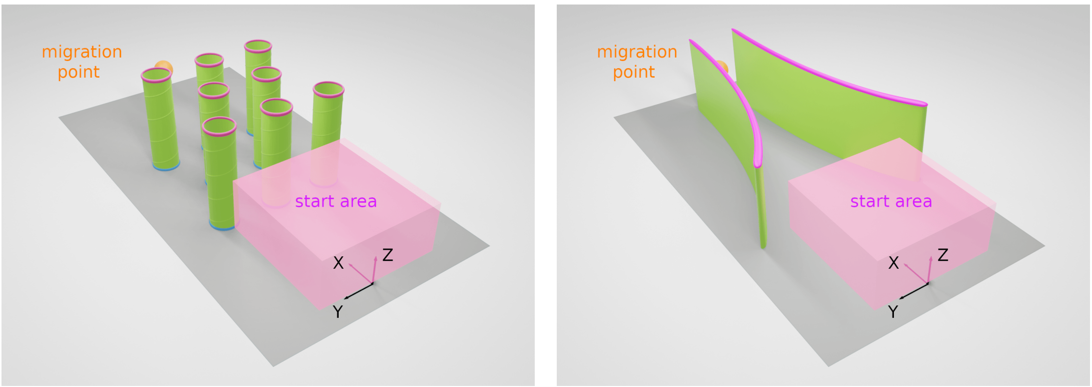
4.1 Introduction

The results from the previous chapter suggest that predictive controllers can improve the safety of aerial swarms by predicting and optimizing the agents’ future behavior in an iterative process. Model Predictive Control (MPC) computes the control action of a system as the solution to an optimization problem that explicitly accounts for the robot dynamics and actuation constraints. Moreover, the computations can be shared among all agents according to a Distributed MPC (DMPC) formulation [69, 32, 76]. With DMPC, every robot solves an optimal problem locally and then communicates its solution to the others to allow global coordination. Following this control scheme, multiple drones can reliably avoid reciprocal collisions when assigned intersecting trajectories [69]. This approach is extended in [32],



(a)

Figure 4.1 – **Self-organized predictive swarm flying in an artificial forest.** Picture of a swarm of 16 drones flying in a forest-like environment in our indoor experimental facility. A supplementary video of our experiments is found at https://youtu.be/1Vg1jdw2Ruk.



(a) Forest-like environment

(b) Funnel-like environment

Figure 4.2 – **Modelling of the environments.** On the left, modeling of the forest-like environment filled with cylindrical obstacles. On the right, funnel-like environment made of two curved surfaces that gradually reduce the flight volume towards the migration point. In both environments, the allowed flight workspace is set to $[8.5, 8.5, 1.1]\ m^3$ and the migration point is in $\boldsymbol{p}_{\text{mig}} = [7.5, 0, 0.6]\ m$). The swarm flies from the start region (pink cube) in the foreground of the scene towards the migration point (orange sphere) in the background.

where the authors present a DMPC motion planner that allows the real-time and collision-free trajectory generation for swarms of up to 20 drones with a single off-board computer.

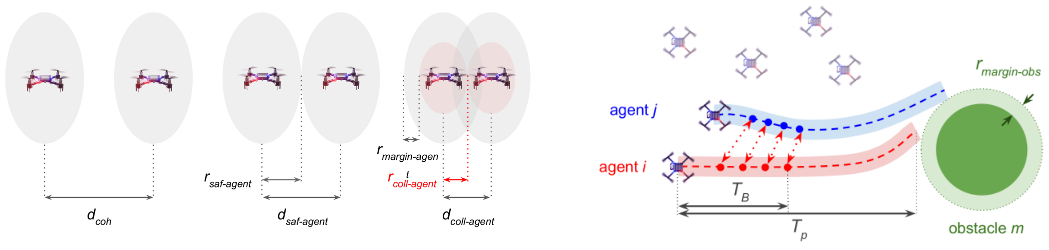
Their on-demand Collision Avoidance (CA) method reduces the computation and travel time compared to Buffered Voronoi Cells (BVC) CA methods [54, 55]. These works show the remarkable potential of modern optimization-based motion planners for solving CA problems of collective systems, although they are designed for individual point-to-point transitions and do not generate self-organized cohesive flight similar to biological swarms.

In our previous work [10], we showed that the collective behavior of biological swarms could be reproduced with an NMPC (Nonlinear MPC) model. The results indicated improved safety and flight synchronization at different obstacle densities, inter-agent distances, and speeds compared to purely reactive approaches based on artificial potential fields[13]. However, the centralized nature of this model allowed the real-time control of only five drones and prevented it from scaling to a large number of drones. Also, the non-convex formulation of the optimization problem required using a computationally expensive scheme based on SQP (Sequential Quadratic Programming) [77].

Here, we present a novel and scalable DMPC swarm model that allows a safe and cohesive flight of aerial swarms in cluttered environments (Fig. 4.1). We show its scalability in the swarm size and its robustness to noise by systematically analyzing the swarm performance at different agents number and noise levels. The swarm performance is studied and compared for two different environments: a forest and funnel-like environment. We also compare the performance of the presented DMPC swarm model with different reciprocal CA methods, i.e., BVC, on-demand, and continuous CA. Finally, we validate the proposed algorithm in real-world experiments with up to 16 palm-sized quadrotors.

4.2 Methods

We consider a swarm composed of N agents labeled by $i \in \mathcal{V} = \{1, \dots, N\}$ and a set of M static obstacles labeled by $m \in \mathcal{M} = \{1, \dots, M\}$ (Fig. 4.2). The swarm can be modeled with a directed sensing graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where the vertex set $\mathcal{V}$ represents the agents, and the edge set $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ contains the pairs of agents (i, j) for which agent i can sense agent j. The state of the i-th agent is represented by $\boldsymbol{x}_i = (\boldsymbol{p}_i, \boldsymbol{v}_i) \in \mathbb{R}^6$ and is made of its position $\boldsymbol{p}_i \in \mathbb{R}^3$ and velocity $\boldsymbol{v}_i \in \mathbb{R}^3$. In the following, k denotes the index of a discrete time step with duration dt. In our DMPC scheme, at every step k, each agent i computes its neighborhood $\mathcal{N}_i{}^k = \{j \in \mathcal{V} \mid (i, j) \in \mathcal{E}\}$ according to the topological metric, which is defined as the set of the n nearest neighbors of agent i at time k [19]. If we indicate with $|\cdot|$ the cardinality of a set, then $|\mathcal{N}_i^k| = n$ is fixed for all instants k, although the neighbors in the set may vary at different k. Then, the agents determine their optimal trajectory w.r.t. their neighbors by solving a constrained optimization problem over a fixed time window called the *prediction horizon* and denoted as $T_P = P\ dt$, $P \in \mathbb{N}^+$ (Fig. 4.3a). The optimization problem aims at minimizing a cost function, which encodes the swarm rules, under constraints that include the dynamic limitations of the agent and the trajectory smoothness. The swarm rules comprise the migration, which steers the agents towards a common goal, the regulation of the inter-



(a) Inter-agent distances

(b) Agent's predicted trajectory

Figure 4.3 – **Schematic overview of the method**. In 4.3a, inter-agent distance parameters: on the left, the cohesion distance $d_{\text{coh}} = 1.30\ m$ (i.e., the maximum allowed distance between neighboring drones), in the middle, the safety distance $d_{\text{saf-agent}} = 0.30\ m$, and on the right, the collision distance $d_{\text{coll-agent}} = 0.14\ m$. In 4.3b, illustration of the predicted trajectory for the focal agent i. At every time step k, the focal agent i determines the neighbor set $\mathcal{N}_i^k$, then computes the distance to the migration point, the relative distances to its neighbors, and potential obstacle collisions over the horizon $T_P = 3.0\ s$. Agents' cohesion and reciprocal avoidance with the proposed continuous CA method are enforced over $T_B = 1.8\ s$. Trajectory replanning happens every $0.2\ s$.

agent distance, which consists of cohesion and agents' reciprocal avoidance, and obstacle avoidance, which steers the agents away from obstacles. Additionally, the control effort rule minimizes the energy spent on maneuvering.

4.2.1 Model of a flying agent

Every agent of the swarm obeys a discrete linear system:

$$\boldsymbol{x}_i(k+1) = \boldsymbol{A}_i \boldsymbol{x}_i(k) + \boldsymbol{B}_i \boldsymbol{u}_i(k) \tag{4.1}$$

where $\boldsymbol{A}_i$ and $\boldsymbol{B}_i$ are constant matrices modeling the dynamics of the Crazyflie 2.1 quadrotor with an underlying position controller. To account for the dynamic feasibility, we limit the position commands by the dimensions of the environment and the acceleration by constant vectors. Hence, it holds $\boldsymbol{p}_{\text{min}} \leq \boldsymbol{u}_i(k) \leq \boldsymbol{p}_{\text{max}}$ and $\boldsymbol{a}_{\text{min}} \leq \boldsymbol{a}_i(k) \leq \boldsymbol{a}_{\text{max}}$.

To quantify the effects of noisy sensor measurements on the flight performance, we model the noise on the agents' positions with a random normal distribution with zero average and equal standard deviation in the three dimensions, σ_p.

4.2.2 Trajectory parameterization

We model the agents' trajectories with l Bezier curves in $\mathbb{R}^3$ of duration T_l, in the same spirit as [32, 78]. This parameterization allows us to define continuous input trajectories to the drones from a finite set of control points. A 3-dimensional Bezier curve of order d is uniquely characterized by a set of $d+1$ control points $\tilde{\mathcal{U}} = \{\tilde{\boldsymbol{u}}_0, ..., \tilde{\boldsymbol{u}}_d\} \in \mathbb{R}^{3(d+1)}$ and the trajectory of agent i is defined by $l(d+1)$ control points $\tilde{\boldsymbol{U}}_i \in \mathbb{R}^{3l(d+1)}$. In the following, $\boldsymbol{x}_i$ and $\boldsymbol{u}_i$ are considered as function of the new unknown $\tilde{\boldsymbol{U}}_i$. At each time k, we can rewrite the dynamic feasibility conditions described above as:

$$\boldsymbol{A}_{\text{dyn},i}\tilde{\boldsymbol{U}}_i^k \leq \boldsymbol{b}_{\text{dyn},i} \tag{4.2}$$

To obtain smooth trajectories, we impose continuity requirements $\mathcal{C}^2$ on the input curve. Samples of the Bezier curves and their derivatives are obtained by linear combinations of the control points. Hence, at each time k, the continuity conditions translate in linear equality constraints of the form:

$$\boldsymbol{A}_{\text{cont},i}\tilde{\boldsymbol{U}}_i^k = \boldsymbol{b}_{\text{cont},i} \tag{4.3}$$

4.2.3 Migration

A *migration point* $\boldsymbol{p}_{\text{mig}} \in \mathbb{R}^3$ known by all agents orients the motion of the swarm in a common direction. The advantage of choosing a migration point over a migration velocity as in [10] is that, even if the agents' path is deviated by some obstacles, the agents will correct their direction to reach the expected destination. We let $(\cdot)(k+\pi|k)$ represent the predicted value of $(\cdot)(k+\pi)$ with the information available at time step k. Then, the migration term is defined as:

$$J^k_{\text{mig},i} = \sum_{\pi=1}^{P} q_{\text{mig}} \|\boldsymbol{p}_i(k+\pi|k) - \boldsymbol{p}_{\text{mig}}\|_2^2 \tag{4.4}$$

where q_{mig} is the weight of the migration behavior.

4.2.4 Agents' reciprocal avoidance

Safety among agents is ensured by requiring neighboring couples of agents to fly at a distance larger than a *safety inter-agent distance* $d_{\text{saf-agent}}$ (Fig. 4.3a). To model the down-wash effect of the quadrotors, 2-norm distances between agents are scaled according to a weight matrix E, with positive diagonal elements $E_{xx} = E_{yy} = 1$ and $E_{zz} < 1$ that defines an ellipsoidal distance $\|\cdot\|_E$. Here, we explore three methods for Collision Avoidance (CA): (i) BVC [54, 55], (ii) on-demand [32], and (iii) continuous CA, the method that we propose. All methods are based on the principle of imposing hyperplane constraints that limit the available space over which the agent optimizes its future trajectory to avoid collisions with its neighbors. In the following, we describe each of them.

BVC CA. In the BVC method, each agent i is forced to stay within its Buffered Voronoi Cell $\mathcal{V}_i$ for a time T_l corresponding to the first Bezier curve of its input trajectory. Let $d_{ij} = \|\boldsymbol{p}_i - \boldsymbol{p}_j\|_E$ be the 2-norm scaled distance between agents i and j, then the Buffered Voronoi Cell of agent

i is:

$$\mathcal{V}_i = \left\{ \frac{(\boldsymbol{p}_i - \boldsymbol{p}_j)^T E^{-2} (\boldsymbol{p} - \boldsymbol{p}_i)}{d_{ij}} \geq \frac{d_{\text{saf-agent}} - d_{ij}}{2}, \ \forall j \in \mathcal{N}_i \right\} \tag{4.5}$$

Condition 4.5 translates into a linear constraint per each of the $(d+1)$ control points of the Bezier curve. For more details on the BVC method we refer to [54, 55, 32].

On-demand CA. This method is based on an event-triggered strategy. It assumes communicative agents that share with their neighbors their predicted actions (i.e., $\boldsymbol{u}_i(k+\pi|k)$, $\pi \in \{0, ..., P-1\}$), and imposes constraints only if potential collisions are detected. On-demand CA only constrains one sample of the agents' trajectory, corresponding to the time of the first detected collision. If agent i detects the first collision at time $k_{\text{coll},i}$, then avoidance constraints are enforced with all its neighbors at that time. Based on the results in [32], we write the constraints in the input space as:

$$\| \boldsymbol{u}_i(k_{\text{coll},i}|k) - \boldsymbol{u}_j(k_{\text{coll},i}|k) \|_E \geq d_{\text{saf-agent}} \tag{4.6}$$

which results in collision-free position commands. Then, we linearize them with a first-order Taylor expansion. For more details on the on-demand method we refer to [32].

Continuous CA. In this method, the constraints have the same formula as in the on-demand method, but they are enforced over an entire trajectory segment rather than a single sample. We define the *braking time* $T_{\text{B}} = B\ dt \leq T_P$ as the minimum time required by an agent that flies at maximum speed to brake until zero velocity. Continuous CA constraints for agent i are enforced with the neighbors $j \in \mathcal{N}_i^k$ over the horizon T_{B}. Hence, neighboring agents need to share their predicted actions over the braking horizon (i.e., $\boldsymbol{u}_i(k+\pi|k)$, $\pi \in \{0, ..., B-1\}$). As for on-demand CA, these constraints are written in the input space. They are:

$$\| \boldsymbol{u}_i(k+\pi|k) - \boldsymbol{u}_j(k+\pi|k) \|_E \geq d_{\text{saf-agent}} \tag{4.7}$$

with $\pi \in \{0, ..., B-1\}$. Because of the larger number of constraints, continuous CA leads to more conservative maneuvers than on-demand CA. Also in this case, we approximate the constraints with a first-order Taylor expansion.

For all methods, we introduce a set of slack variables $\mathcal{E}_i^k$ that relax the hyperplane constraints and make it more likely for the optimization problem to find a viable path. Each variable $\epsilon_{ij} \geq 0$ indicates the amount by which the avoidance constraint between agent i and neighbor j is violated. For example, for continuous CA, the relaxed constraints are:

$$\| \boldsymbol{u}_i(k+\pi|k) - \boldsymbol{u}_j(k+\pi|k) \|_E \geq d_{\text{saf-agent}} - \epsilon_{ij}(k+p|k) \tag{4.8}$$

More generally, for all CA methods, we indicate the set of linear reciprocal CA constraints for agent i at time k as:

$$\boldsymbol{A}^k_{\text{saf-agent},i}[(\tilde{\boldsymbol{U}}^k_i)^T, (\mathcal{E}^k_i)^T]^T \leq \boldsymbol{b}^k_{\text{saf-agent},i} \tag{4.9}$$

$$-\mathcal{E}^k_i \leq 0 \tag{4.10}$$

The cost associated with the violation of these constraints includes linear and quadratic terms in ϵ_{ij} with constant weights l_{saf} and q_{saf}, respectively. For example, for continuous CA, the total violation cost is:

$$J^k_{\text{saf-agent},i} = \sum_{j \in \mathcal{N}_i} \sum_{\pi=0}^{B-1} \Big(l_{\text{saf}} \epsilon_{ij}(k+\pi|k) + q_{\text{saf}} \epsilon^2_{ij}(k+\pi|k) \Big) \tag{4.11}$$

4.2.5 Agents' cohesion

The swarm behavior of cohesion requires neighboring couples of drones to stay closer than the *cohesion distance* d_{coh}. We introduce slack variables δ_{ij} for agent i and neighbors $j \in \mathcal{N}^k_i$ and we formulate the cohesion constraint over the horizon T_{B} in the input space as:

$$\| \boldsymbol{u}_i(k+\pi|k) - \boldsymbol{u}_j(k+\pi|k) \|_E \leq d_{\text{coh}} + \delta_{ij}(k+\pi|k) \tag{4.12}$$

If we indicate with $\boldsymbol{\Delta}^k_i$ the set of slack variables for agent i at time k, then the set of cohesion constraints approximated by a first-order Taylor expansion is denoted as:

$$\boldsymbol{A}^k_{\text{coh},i}[(\tilde{\boldsymbol{U}}^k_i)^T, (\boldsymbol{\Delta}^k_i)^T]^T \leq \boldsymbol{b}^k_{\text{coh},i} \tag{4.13}$$

$$-\boldsymbol{\Delta}^k_i \leq 0 \tag{4.14}$$

The cost associated with the violation of the constraints is:

$$J^k_{\text{coh},i} = \sum_{j \in \mathcal{N}_i} \sum_{\pi=0}^{B-1} \Big(l_{\text{coh}} \delta_{ij}(k+\pi|k) + q_{\text{coh}} \delta^2_{ij}(k+\pi|k) \Big) \tag{4.15}$$

where l_{coh} and q_{coh} are constant weights.

4.2.6 Obstacle avoidance

The obstacle avoidance behavior is produced by requiring the drones to fly at a *safety distance* $d_{\text{saf-obs}}$ from the obstacles. To keep this behavior local, each agent only considers the first obstacle on the collision course with its predicted trajectory. In this work, we consider convex obstacles that we model with 3D ellipsoids with arbitrary axes dimensions. If a drone i detects its first collision with obstacle m at instant $k_{\text{coll},i}$ it adds an anti-collision constraint with it to its optimization problem. We introduce the slack variable $\zeta_{im} \geq 0$ referred to agent i and obstacle m and consider the following constraint:

$$\|\boldsymbol{u}_i(k_{\text{coll},i}|k) - \boldsymbol{p}_m\|_E \geq d_{\text{saf-obs}} - \zeta_{im}(k_{\text{coll},i}|k) \tag{4.16}$$

As done before, we indicate with $\mathcal{Z}_i^k$ the obstacle avoidance slack variables for agent i at time k, and write the first-order approximation of the above constraint as:

$$\boldsymbol{A}^k_{\text{saf-obs},i}[(\tilde{\boldsymbol{U}}_i^k)^T, (\mathcal{Z}_i^k)^T]^T \leq \boldsymbol{b}^k_{\text{saf-obs},i} \tag{4.17}$$

$$-\mathcal{Z}_i^k \leq 0 \tag{4.18}$$

The cost associated with the violation of the constraint is:

$$J^k_{\text{saf-obs},i} = l_{\text{saf}}\zeta_{im}(k_{\text{coll},i}|k) + q_{\text{saf}}\zeta^2_{im}(k_{\text{coll},i}|k) \tag{4.19}$$

4.2.7 Control effort

The *control effort* is responsible for minimizing the energy required by the control commands. It is defined by a weighted sum of the second squared derivative of the input commands that penalizes acceleration and deceleration of an agent:

$$J^k_{\text{effort},i} = \sum_{\pi=0}^{P-1} q_{\text{effort}} \left\| \frac{d^2}{dt^2}\boldsymbol{u}_i(k+\pi|k) \right\|_2^2 \tag{4.20}$$

where q_{effort} is the weight of the control effort rule.

4.2.8 Desired trajectory

To calculate the desired trajectory at each time step k, every drone i solves the following QP problem, which includes all the above cost terms and constraints:

$$\begin{aligned}
&\min_{\tilde{\boldsymbol{U}}_i^k, \mathcal{E}_i^k, \boldsymbol{\Delta}_i^k, \mathcal{Z}_i^k} J_{\text{mig},i}^k + J_{\text{saf-agent},i}^k + J_{\text{coh},i}^k + J_{\text{saf-obs},i}^k + J_{\text{effort},i}^k \\
&\text{subject to:} \\
&\quad \boldsymbol{A}_{\text{dyn},i} \tilde{\boldsymbol{U}}_i^k \leq \boldsymbol{b}_{\text{dyn},i} \\
&\quad \boldsymbol{A}_{\text{cont},i} \tilde{\boldsymbol{U}}_i^k = \boldsymbol{b}_{\text{cont},i} \\
&\quad \boldsymbol{A}_{\text{saf-agent},i}^k [(\tilde{\boldsymbol{U}}_i^k)^T, (\mathcal{E}_i^k)^T]^T \leq \boldsymbol{b}_{\text{saf-agent},i}^k \\
&\quad \boldsymbol{A}_{\text{coh},i}^k [(\tilde{\boldsymbol{U}}_i^k)^T, (\boldsymbol{\Delta}_i^k)^T]^T \leq \boldsymbol{b}_{\text{coh},i}^k \\
&\quad \boldsymbol{A}_{\text{saf-obs},i}^k [(\tilde{\boldsymbol{U}}_i^k)^T, (\mathcal{Z}_i^k)^T]^T \leq \boldsymbol{b}_{\text{saf-obs},i}^k \\
&\quad -\mathcal{E}_i^k \leq 0 \\
&\quad -\boldsymbol{\Delta}_i^k \leq 0 \\
&\quad -\mathcal{Z}_i^k \leq 0
\end{aligned} \tag{4.21}$$

4.2.9 Implementation of Bezier curves

A three-dimensional Bezier curve of degree d and duration T is defined as:

$$B(t) = \sum_{m=0}^{d} P_m B_{m,d}(t) \tag{4.22}$$

where $P = \{P_0, P_1, \ldots, P_d\}$ is the set of $d+1$ control points that characterize that curve and $B_{m,d}(t)$ are the $d+1$ Bernstein polynomials of degree d:

$$B_{m,d}(t) = \binom{d}{m} (1 - t/T)^{p-m} (t/T)^m \quad \forall t \in (0, T) \tag{4.23}$$

Expressing the Bezier curve in the power basis $\{1, t, \ldots, t^d\}$ is useful to obtain the samples of the curve. If we denote with $S = \{S_0, S_1, \ldots, S_d\}$ the $d+1$ polynomial coefficients of the power basis, then the linear transformation between polynomial bases can be defined by a constant matrix β as $S = \beta P$ (see [79] for details on how to compute β). Another important aspect of Bezier curves is that samples at $t \in [0, T]$ are a linear combination of the polynomial coefficients (and thus, of the control points) that we denote with $B = TS$. Hence, it holds $B = T\beta P$.

4.2.10 Swarm performance metrics

We assess the performance of the swarm's flight according to six different metrics. The mission completion time T measures the time that the swarm requires to complete a mission. A mission is completed if the swarm average position reaches the migration point up to a tolerance distance d_{tol} and if, at the same time, all the drones are within the distance d_{coh} from their neighbors. The trajectory length L_{traj} measures the average of the agents' flown distances until they complete the mission. The minimum and the maximum inter-agent distances, $\min(d_{ij})$ and $\max(d_{ij})$, measure the minimum and the maximum distance among neighboring couples of drones over the mission. The minimum distance to the obstacles $\min(d_{im})$ measures the minimum distance between all agents and all obstacles. Finally, the order Φ_{order} measures the average correlation of the agents' directed movements. It is often used to quantify the synchronization of the agents' flight [13, 17], and in formula, it is written as:

$$\Phi_{\text{order}} = \sum_{k\in\{1,\dots,K\}} \sum_{i\in\mathcal{V}} \sum_{j\in\mathcal{N}_i^k} \frac{\boldsymbol{v}_i(k)\boldsymbol{v}_j(k)}{KNn\|\boldsymbol{v}_i(k)\|\|\boldsymbol{v}_j(k)\|} \tag{4.24}$$

where $K = \min(\lceil T/dt \rceil, \lceil T_{\max}/dt \rceil)$ and $\lceil\cdot\rceil$ is the ceiling function.

Table 4.1 – **Swarm performance metrics.** Description of the swarm performance metrics used to evaluate the swarm flight. We denote with K the minimum between the time index of mission completion and the experiment end time.

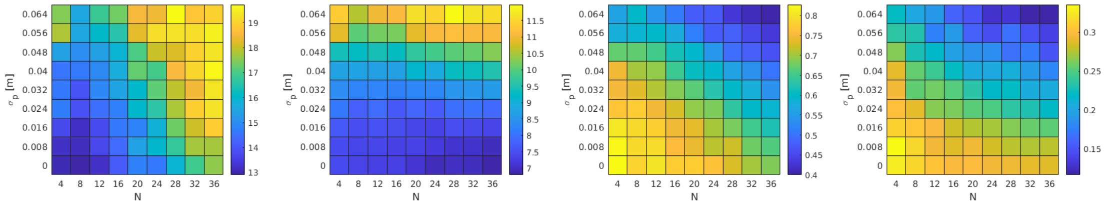
Metric	Formula	Description
T	–	**Mission completion time**. The time that takes for the swarm to reach the migration point up to a tolerance distance d_{tol} while, at the same time, having maximum inter-agent distances within the bound d_{coh}.
L_{traj}	$\sum_{k\in\{1,\dots,K\}} \frac{\sum_{i\in\mathcal{V}} \|\boldsymbol{p}_i(k)-\boldsymbol{p}_i(k-1)\|}{N}$	**Trajectory length**. Average trajectory length of the agents until they complete the mission or the experiment ends.
Φ_{order}	$\frac{\sum_{k\in\{1,\dots,K\}}\sum_{i\in\mathcal{V}}\sum_{j\in\mathcal{N}_i} \boldsymbol{v}_i(k)\boldsymbol{v}_j(k)}{KNn\|\boldsymbol{v}_i(k)\|\|\boldsymbol{v}_j(k)\|}$	**Swarm order**. The correlation between the agents' flight directions. It is equal to 1 when all agents' velocities are pointing in the same direction (ideal case), while it is equal to 0 when, for instance, they are diametrically opposed in pairs.
$\min(d_{ij})$	$\min_{i\in\mathcal{V},\ j\in\mathcal{N}_i,\ k\in\{1,\dots,K\}}(d_{ij}(k))$	**Minimum inter-agent distance**. Minimum distance between any agents and their neighbors over the mission.
$\max(d_{ij})$	$\max_{i\in\mathcal{V},\ j\in\mathcal{N}_i,\ k\in\{1,\dots,K\}}(d_{ij}(k))$	**Maximum inter-agent distance**. Maximum distance between any agents and their neighbors over the mission.
$\min(d_{im})$	$\min_{i\in\mathcal{V},\ m\in\mathcal{M},\ k\in\{1,\dots,K\}}(d_{im}(k))$	**Minimum distance to obstacles**. Minimum distance between any agents and any obstacles over the mission.

4.3 Results

We implemented our swarm model in MATLAB 2020b, and executed our simulations on a computer equipped with Intel Core i7-8750H CPU with 12 cores and 16 GB of RAM. For the solution of the optimization problem, we used the active set algorithm [80]. The parameter values used in simulation experiments are detailed in A.1.

Parameter	Description	Unit	Value
dt	Control time step	s	0.2
$d\tau$	Simulation time step	s	0.01
$T_{\max}$	Simulation time	s	20
T_P	Prediction horizon	s	3
T_B	Brake time	s	1.8
l	Number of Bezier curves	–	3
d	Bezier curve order	–	5
T_l	Bezier curve duration	s	1
$\boldsymbol{p}_{\text{mig}}$	Migration point	m	$[7.5, 0, 1]$
$\boldsymbol{p}_{\min}$	Minimum allowed position	m	$[0, -4.25, 0.4]$
$\boldsymbol{p}_{\max}$	Maximum allowed position	m	$[8.5, 4.25, 1.5]$
$\boldsymbol{a}_{\min}$	Minimum acceleration	m/s^2	$[-1, -1, -1]$
$\boldsymbol{a}_{\max}$	Maximum acceleration	m/s^2	$[1, 1, 1]$
q_{mig}	Quadratic migration weight	–	2
l_{saf}	Linear safety weight	–	10000
q_{saf}	Quadratic safety weight	–	100
l_{coh}	Linear cohesion weight	–	1000
q_{coh}	quadratic cohesion weight	–	10
d_{coh}	Cohesion distance	m	1.3
$d_{\text{saf-agent}}$	Inter-agent safety distance	m	0.3
$d_{\text{coll-agent}}$	Inter-agent collision distance	m	0.14
$r_{\text{margin-agent}}$	Agent safety margin	m	0.08
$d_{\text{saf-obs}}$	Obstacle safety distance	m	0.2
$d_{\text{coll-obs}}$	Obstacle collision distance	m	0.07
$r_{\text{margin-obs}}$	Obstacle safety margin	m	0.14
n	Number of nearest neighbors	–	3
E_{xx}	Distance scaling in x	–	1
E_{yy}	Distance scaling in y	–	1
E_{zz}	Distance scaling in z	–	0.5
r_{obs}	Cylindrical obstacle radius	m	0.35

Table 4.2 – **Swarm model parameters.** Description and values of the DMPC swarm model parameters used in our simulation experiments.



(a) Mission completion time (b) Avg. trajectory length (c) Avg. order (d) Min. inter-agent distance

Figure 4.4 – **Swarm performance in forest-like environment**. On top, simulation results for swarms with different agent numbers and noise levels. For each parameter combination (i.e., one bin in the heatmaps), the results show the average of 10 random simulations. Specifically, 4.4a reports the mission completion time T, 4.4b reports the average trajectory length L_{traj}, 4.4c reports the average order Φ_{order}, and 4.4d reports the minimum inter-agent distance $\min(d_{ij})$. At the bottom, aggregate average and standard deviation of the same metrics for three different noise levels: in blue $\sigma_p = 0\ m$, in orange $\sigma_p = 0.024\ m$, and in yellow $\sigma_p = 0.048\ m$. Collisions between agents happen at noise levels $\sigma_p \geq 0.056\ m$. Instead, collisions with the obstacles already happen at $\sigma_p \geq 0.040\ m$.

4.3.1 Scalability in the agent number and noise robustness

We run swarm simulations for 9 swarm sizes $N \in \{4, 8, 12, 16, 20, 24, 28, 32, 36\}$ and 9 noise levels, $\sigma_p \in \{0, 0.008, 0.016, 0.024, 0.032, 0.040, 0.048, 0.056, 0.064\}\ m$ in the forest-like environment (Fig. 4.2a). For every configuration, we run 10 random simulations and averaged the results. In the computation of the swarm performance, we only considered the missions that the swarm could complete within $T_{\max} = 20\ s$ (Fig. 4.5).

The results show that the coordination of a large swarm requires a longer time than a small swarm. Analogously, an increase in the noise level requires extra time for the swarm to reach the migration point (Fig. 4.4a). The increased mission time in the presence of noise can be explained by an increment in the agents' trajectory lengths (Fig. 4.4b). Instead, for swarms of large sizes, the trajectory lengths seem almost stationary for a given noise level (Fig. 4.4b), necessarily implying a lower speed. The average swarm order remarkably reduces when the swarm size and noise level increase (Fig. 4.4c). Specifically, when passing from zero noise level to noise level $\sigma_p = 0.048\ m$ the order decreases of about 21 to 32% depending on the swarm size ($\Phi_{\text{order}} \approx 0.83$ to 0.68 for $N = 4$ to 36 at $\sigma_p = 0\ m$, while $\Phi_{\text{order}} \approx 0.67$ to 0.46 for $N = 4$ to 36 at $\sigma_p = 0.048\ m$). The reduction of order in the presence of noise, indicating a decrease in the correlation of the agents' movements, is due to the agents' need of adjusting their directions in order to avoid collisions (Fig. 4.6i). This tendency increases with the agent number. With no noise, the minimum inter-agent distance is approximately steady around the safety value ($\min(d_{ij}) \approx 0.3\ m$) independently on the swarm size (Fig. 4.4d). Instead, in the presence of noise, the minimum distance decreases when the swarm size increases (Fig. 4.4d). On average, collisions between drones happen from a noise level $\sigma_p = 0.056\ m$, which is approximately 70% of the magnitude of the safety margin $r_{\text{margin-agent}}$ and 80% of agent collision radius

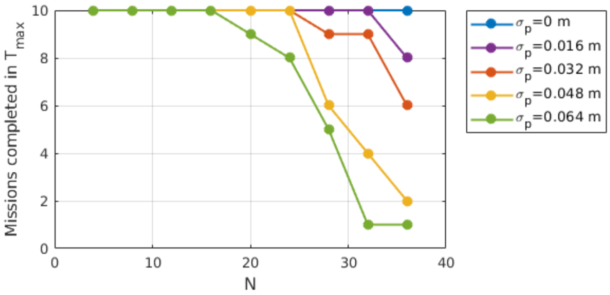
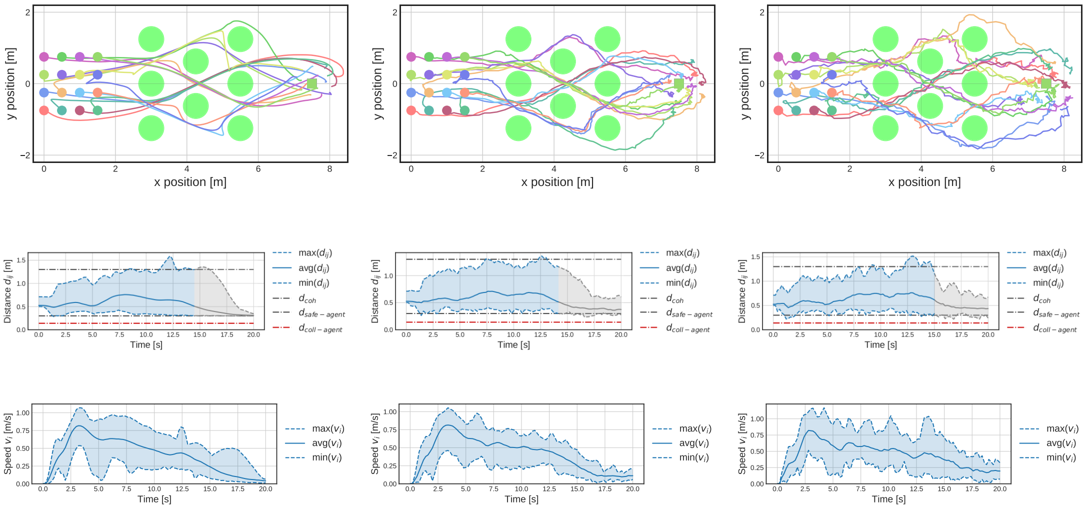


Figure 4.5 – **Number of completed missions in time $t \leq T_{\max}$.** Number of completed missions (out of 10 random simulations) for swarms of different agent numbers N at different noise levels σ_p, flying in the forest-like environment. We consider the mission completed only if the swarm gets to the migration point before a maximum time of $T_{\max} = 20\ s$. For a noise level $\sigma_p = 0\ m$, the swarm could complete all missions.



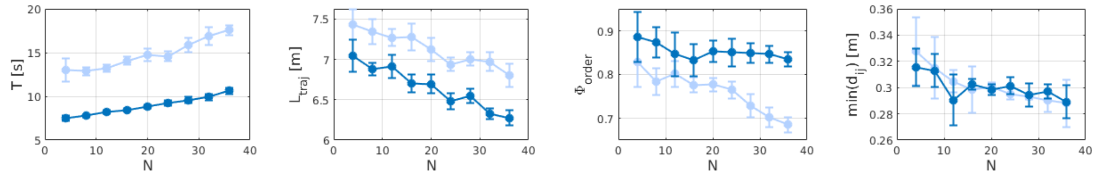
(g) Noise level $\boldsymbol{\sigma_p} = 0\ m$ (h) Noise level $\boldsymbol{\sigma_p} = 0.024\ m$ (i) Noise level $\boldsymbol{\sigma_p} = 0.048\ m$

Figure 4.6 – **Swarm trajectories at different noise levels**. Simulated trajectories and inter-agent distances for a swarm of 16 drones in a forest-like environment at three noise levels ($\boldsymbol{\sigma_p} = 0$, $0.024\ m$, and $0.048\ m$). Although we report no collisions in all cases, the trajectories are visibly smoother at zero noise level (4.6g) than in the presence of noise (4.6h and 4.6i). The middle and bottom rows show the envelope of the distance between neighboring drones d_{ij} and the drones speed v_i. The plots are grayed out from the mission completion time T until the simulation end time $T_{\max}$. The swarm completes the mission in $T = 14.6$, 14.2, and $15\ s$ at low, medium, and high noise level respectively.

$r_{\text{coll-agent}}$. Finally, the minimum distance to obstacles showed the same trend as the minimum inter-agent distance. We did not observe a significant trend for the maximum inter-agent distance, which slightly fluctuates above the cohesion distance ($\max(d_{ij}) = 1.40 \pm 0.09\ m$ on average).

By comparing the swarm flight at three different noise levels (i.e., $\sigma_p = 0$, 0.024, and 0.048 m), we notice that the trajectories are visibly smoother in the absence of noise than in the presence of little or high noise (Fig. 4.6). This result is consistent with the order results, which show decreasing values at high noise levels (Fig. 4.4c). Furthermore, the minimum inter-agent distance decreases with an increase in the noise ($\min(d_{ij}) = 0.29$ and 0.22 m at $\sigma_p = 0$ and 0.048 m, respectively) (Fig. 4.6i). However, the distance envelope stays above the collision distance, which indicates that no collisions happen. The maximum inter-agent distance slightly exceeds the cohesion value in all cases, but presents its maximum value in the case of $\sigma_p = 0.048\ m$ ($\max(d_{ij}) = 1.52\ m$) (Fig. 4.6i). This violation happens in favor of the obstacle avoidance behavior, which has priority over cohesion. In all cases, after the swarm completes the mission (at time T), the agents concentrate around the migration point and orbit around it at low speed while reducing their inter-agent distance to minimize the distance to the migration point and without violating the collision avoidance constraints (Fig. 4.6).

4.3.2 Adaptability to different environments



(a) Mission completion time (b) Avg. trajectory length (c) Avg. order (d) Min. inter-agent distance

Figure 4.7 – **Swarm performance comparison in the funnel and forest-like environments**. Aggregate results (average and standard deviation of 10 random simulations) of the swarm performance in the funnel-like (in blue) and forest-like (in light blue) environments at zero noise level and for different agent numbers N. For all agent numbers, we report zero agent-agent collisions.

To evaluate the swarm flight quality in different environments, we simulate our swarm model in a funnel-like environment (Fig. 4.2b) for the same swarm sizes as in the forest-like environment and compare the results.

In the funnel-like environment, the swarm flies overall shorter trajectories to get to the migration point compared to the forest-like environment (Fig. 4.7b). As a result, the mission times are generally shorter (Fig. 4.7a). The mission completion time and the average trajectory length present the same trends in both environments: while the first increases with the agent number, the second slightly decreases. The order is almost steady across the swarm sizes

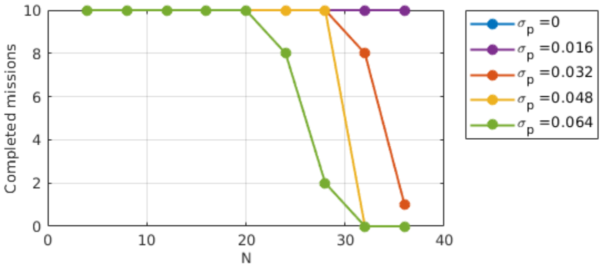
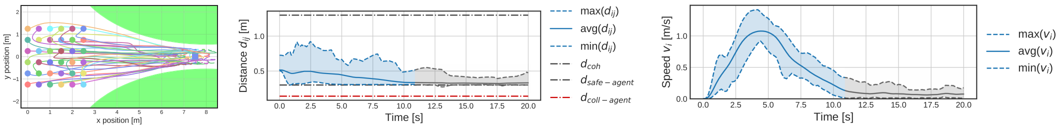


Figure 4.8 – **Number of completed missions in time $t \leq T_{\max}$.** Number of completed missions (out of 10 random simulations) for swarms of different agent numbers N at different noise levels σ_p, flying in the funnel-like environment. We consider the mission completed only if the swarm gets to the migration point before a maximum time of $T_{\max} = 20\ s$. For a noise level $\sigma_p = 0\ m$, the swarm could complete all missions.



(a) Trajectories (b) Inter-agent distance (c) Speed

Figure 4.9 – **Swarm of 36 drones in a funnel-like environment**. Results of a swarm of 36 agents flying in a funnel-like environment. From left to right, trajectories, inter-agent distance, and speed. The swarm completes the mission at $T = 10.8\ s$, instant from which the plots are grayed out.

(Fig. 4.7c) since all agents fly consistently along the x-direction. Instead, in the forest-like environment, the swarm is less ordered due to the obstacle avoidance maneuvers, and this effect is amplified at large swarm sizes. Finally, we report zero collisions and the maximum inter-agent distance stayed below the cohesion distance for all agent numbers. However, in the funnel environment we notice a phenomenon of swarm compression characterized by a decrease of the average inter-agent distance towards the end of the funnel and due to the reduced volume.

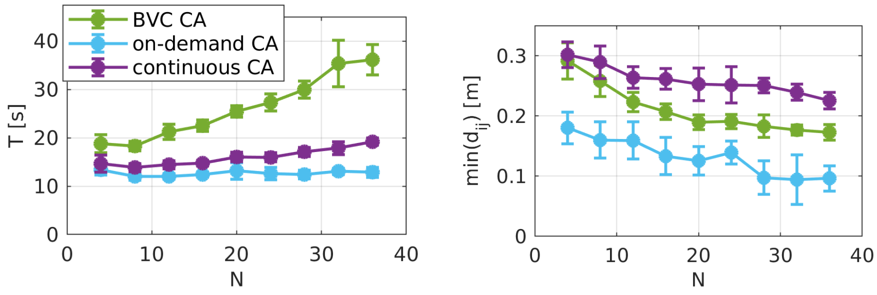
The reduced volume of the funnel causes a phenomenon of swarm compression that is characterized by a decrease of the average inter-agent distance over time and it is accentuated towards the end of the funnel, where the two surfaces draw closer. This tendency is evident for large swarms (i.e., $N = 36$ drones, Fig.4.9). The speed highlights two phases: in the first phase, the agents accelerate and their speed grows to get to the migration point as quickly as possible. In the second phase, the agents decelerate. By the time the swarm attains the migration point, they have little speed left and orbits around that point without colliding.

Solve time	**Swarm size**								
	4	**8**	**12**	**16**	**20**	**24**	**28**	**32**	**36**
Avg $[ms]$	9.3	9.3	9.4	9.8	10.7	11.3	12.1	12.3	16.1
Std $[ms]$	2.6	1.7	2.0	2.4	3.0	3.4	4.4	4.9	8.4

Table 4.3 – **Empirical runtime per agent for different swarm sizes**. Empirical runtime results (average and standard deviation) for the algorithm implementing continuous CA method. The shown data is the average over 10 random trials.

4.3.3 Comparison of collision avoidance methods

We compared the performance of the optimization-based swarm model with the three reciprocal CA methods. In order to test the algorithms as the agent density increases, we run 10 random simulations in the forest-like environment for all the swarm sizes. For brevity and clarity, we present here the results at a fixed noise level (i.e., $\sigma_p = 0.024\ m$), although similar trends can be observed for other noise levels too.



(a) Mission completion time

(b) Min. inter-agent distance

Figure 4.10 – **Comparison of collision avoidance methods**. Comparison of the average performance metrics for different CA methods: BVC, on-demand, and continuous CA. The shown data is the average over 10 random trials for each swarm size.

Being the least conservative, on-demand CA leads to the fastest mission completion times for all swarm sizes (20% and 49% of average time reduction over all swarm sizes compared to the continuous and BVC methods, respectively (Fig. 4.10a). However, continuous CA outperforms the other two methods in terms of safety. This evidence is supported by the plot of the minimum inter-agent distance (Fig. 4.10b). For continuous CA, the minimum inter-agent distance stays close to the safety value (ranging from 0.30 to 0.23 m for swarms of 4 to 36 agents). Instead, for the other two methods it is significantly lower (ranging from 0.29 to 0.18 m and from 0.18 to 0.10 m for swarms of 4 and 36 agents and for the BVC and the on-demand methods, respectively). Indeed, with the BVC and on-demand methods we recorded occasional collisions.

4.3.4 Computational complexity

We formally analyze the computational complexity of the presented algorithm with the three reciprocal CA methods in terms of computation time per agent to build constraints and solve the QP. In our notation, $|\mathcal{N}_i{}^k|$ indicates the number of considered neighbors for agent i at time step k, which is constant and equal to n. The number of reciprocal CA constraints for each algorithm scales linearly with the number of neighbors, i.e., $O(n)$. The BVC method adds $(d+1)n$ slack variables for relaxing the reciprocal avoidance constraints, where d is the Bezier curve order. The on-demand method adds only n slack variables and hence has the lowest complexity, while our method adds Bn new decision variables. Since in our method the number of variables increases with B, increasing the agents' maximum speed increases the computational complexity. Solving a standard QP problem has complexity $O(v^3)$, where v is the number of decision variables, which is linear in the planning horizon P and the number of neighboring robots n. Hence, for each robot, the total computational complexity is $O(v^3)$.

Numerical values for the runtime depend on the used hardware and solver capabilities. Here, we report the empirical runtime of the algorithm running on the ground station with continuous CA in the forest-like environment and for different swarm sizes (Table 4.3). We expect the runtime per agent to be independent of the swarm size since we chose agents' neighborhood to have a fixed cardinality (i.e., n). Instead, we notice that it increases as we add more agents to the swarm. We explain this fact with an increase in the swarm density, which implies more complex QPs to be solved for each agent, due to its closer proximity to the obstacles, neighbors, and environment boundaries.

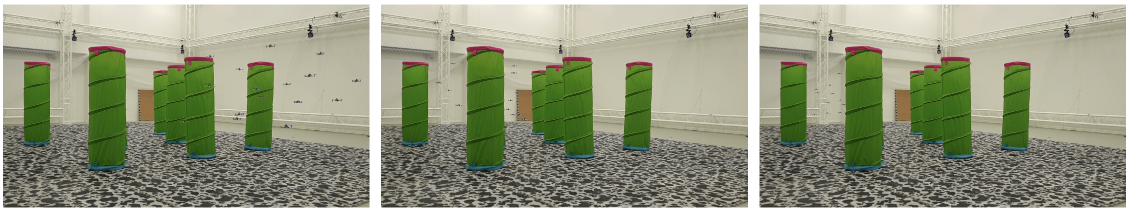
4.3.5 Hardware experiments

We implemented the swarm model in Python, where we used numba[1] to speed up the mathematical computations, OSQP to solve the QP [81], and the Crazyswarm interface to communicate with the Crazyflie 2.1 drones [82]. An Optitrack motion capture system acquired the drone positions and streamed them to the ground station. Then, the ground station computed the optimal predicted trajectories online and for all drones in parallel and broadcast them to the swarm via two radio links. In our experiments, local communication between drones is mimicked by the ground station exchanging information between threads. Instead, full on-board implementation would require direct communication between neighboring robots. Moreover, drones should embed sensors to estimate their relative positions to obstacles and more powerful computers to solve the optimization problems on-board. The model parameters are the same as above, except for the migration point ($\boldsymbol{p}_{\text{mig}} = [6, 0, 1]\ m$) to fit our experimental room.

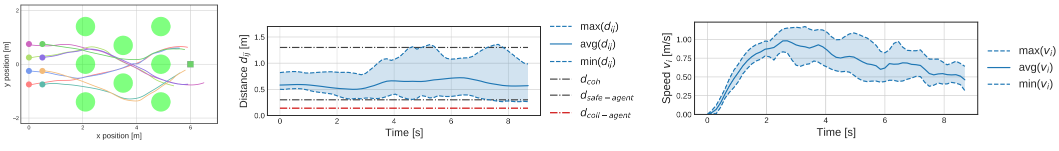
The swarms of 8 and 16 drones reach the migration point in comparable times and cover comparable distances (see Table 4.4). However, the trajectory lengths in hardware experiments are slightly longer than in simulation because of the small positional errors in real-world

---

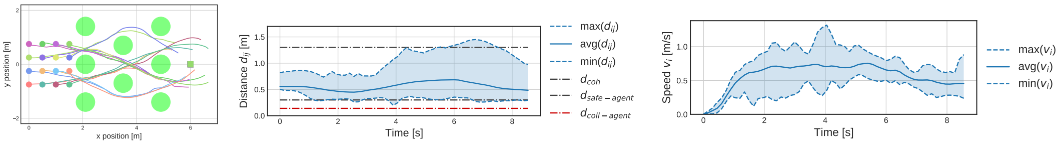
[1] https://numba.pydata.org/



(a) Time $t = 0\ s$ (b) Time $t = 4\ s$ (c) Time $t = 8\ s$



(d) Trajectories (e) Inter-agent distance (f) Speed



(g) Trajectories (h) Inter-agent distance (i) Speed

Figure 4.11 – **Experimental results with 16 drones in a forest-like environment**. Snapshots at three instants (from left to right, $t = 0,\ 4,\ 8\ s$) of a swarm of 16 drones flying from the start area (in the foreground) to the migration point (in the background). At the bottom, trajectories, inter-agent distance, and speed for swarms of 8 and 16 drones over the mission completion time T.

experiments. The average order decreases when passing from 8 to 16 drones ($\Phi_{\text{order}} = 0.88$ and 0.85, respectively). This result follows the simulation experiments ($\Phi_{\text{order}} = 0.93$ and 0.88 for 8 and 16 agents, respectively) and confirms the above statistic results in the forest-like environment: larger swarms decrease their order to insure collision avoidance in cluttered environments. The minimum inter-agent distance remains above the collision threshold in all cases. However, it decreases when increasing the swarm size ($\min(d_{ij}) = 0.26$ and $0.20\ m$ for the swarms of 8 and 16 drones, respectively). Finally, the agents do not collide with obstacles.

4.4 Discussion

In this chapter, we described a distributed MPC model for aerial swarms that results in self-organized, safe, and cohesive flight in cluttered environments. The proposed DMPC algorithm with the continuous collision avoidance method generates collection-free trajectories even in the presence of sensor noise at levels up to 70% of the magnitude of the agent safety margin distance. We validated the proposed algorithm in two types of simulated environments with obstacles, and on 16 palm-sized drones flying in a real forest-like indoor environment.

Metric	**Simulation**		**Hardware**	
	8 agents	16 agents	8 agents	16 agents
T $[s]$	9.2	9.0	8.9	8.7
L_{traj} $[m]$	5.88	5.47	5.93	5.49
Φ_{order} $[-]$	0.93	0.88	0.88	0.85
$\min(d_{ij})$ $[m]$	0.31	0.26	0.26	0.20
$\max(d_{ij})$ $[m]$	1.31	1.26	1.31	1.45
$\min(d_{im})$ $[m]$	0.22	0.18	0.09	0.08

Table 4.4 – **Comparison of simulation and hardware swarm performance.** Comparison of the performance metrics of two swarms of 8 and 16 drones between simulation and hardware experiments.

While this work paves the way for large and safe decentralized aerial drone swarms, future work should focus on the challenges for a full on-board implementation. Work in this direction will address communication issues for large multi-agent systems such as communication delays, packets losses, interference, and synchronization.

Additionally, future work will explore swarm navigation in non-convex configuration space. In the presence of concave obstacles, the current obstacle avoidance strategy may lead to deadlocks due to the presence of local minima. Hence, the integration of alternative techniques based on topological planning as in [42] should be investigated to solve this issue.

5 Reynolds swarms with limited visual sensing

Our previous work on predictive swarms relies on the assumption that each robot can communicate its predicted trajectory to a local neighborhood. Communication-based swarm models are fragile to outages and delays, especially when flying in high-density formations or in cluttered environments. From this chapter on, we remove the hypothesis of explicit communication between agents. Instead, we consider sensor-based agents that can estimate relative positions and velocities to their neighbors. In nature, the central sensor modality often used for achieving swarming is vision. Hence, in this chapter, we analyze the effects of limited visual sensing on the behavior of potential field swarm models from the state-of=the-art. We study how the reduction in the field of view and the orientation of the visual sensors affect the performance of the Reynolds algorithm used to control the swarm. As Nature suggests, our results confirm that lateral vision is essential for coordinating the movements of individuals. Moreover, agents benefit from omnidirectional vision to avoid collisions. We achieve the results presented in this paper through extensive Monte-Carlo simulations and integrate them with the use of genetic algorithm optimization.

The work presented in this chapter is adapted from [26]:

- E. Soria, F. Schiano, and D. Floreano, "The influence of limited visual sensing on the Reynolds flocking algorithm," in *IEEE Third International Conference on Robotic Computing (IRC)*, 2019, pp. 138-145, doi: 10.1109/IRC.2019.00028.

5.1 Introduction

An evidence from nature is that several animals rely on visual information to achieve navigation, collision avoidance, and remain connected with the other individuals of the group [83, 84, 20, 85]. Although visual organs are limited in range and angular span, birds fly in perfect choreographed synchronization. Sensor limitations are seldom taken into account in the study of aerial swarms since they can be hard to model [86]. However, understanding their effects can be fundamental to bring aerial swarms from lab-conditions to real-world scenarios.

The visual processes of animals include several aspects. In particular, a distinction can be made between *physical* and *perceptual* properties. Examples of the former include the geometric field that the eyes can perceive, their rapidity to adapt to light and environmental changes and the range of the detected light wavelengths. The visual perception, instead, is defined as the ability of animals to assimilate information from the surroundings and it is an open field of research for psychologists, neuroscientists, and molecular biologists.

In this thesis chapter, we focus on analyzing the impact of the physical geometric properties of visual sensing, specifically the width angle of the field of view (FOV) and its orientation, on the performance of a Reynolds flocking. Because the aimed behavior of a swarm is often application-dependent, we quantify the effects of the two aforementioned visual properties with different metrics that can be relevant to different scenarios. In some applications, the central interest can be to remain connected [64, 65], while in others, it may be more crucial to control the number of subgroups that originate during the operation. An example of the second case is patrolling [87], where the swarm is allowed to split, but a minimum number of agents per cluster is needed to enable stereo vision.

The rest of the section is organized as follows. Section 5.2 reviews the notation, the concepts of limited sensing and the Reynolds swarming model, and the metrics used for the evaluation of the swarm with limited visual sensing. Then, Section 5.3 presents the results of our work, followed by Section 5.4 that concludes the chapter and outlines possible future directions.

5.2 Method

5.2.1 Notation

In this work, we consider a set of N point-mass agents labeled by $i \in \{1,2,3,\dots,N\}$. The position, velocity and acceleration of the agent i are denoted by $\boldsymbol{p}_i, \boldsymbol{v}_i, \boldsymbol{a}_i \in \mathbb{R}^2$, respectively. In order to keep our notation concise we let $d_{ij} = \|\boldsymbol{p}_j - \boldsymbol{p}_i\|$ where $\|\cdot\|$ denotes the Euclidean norm, and $\bar{\boldsymbol{p}}_{ij} = \boldsymbol{p}_{ij}/d_{ij}^2$. We then let $\boldsymbol{I}_N \in \mathbb{R}^{N\times N}$ represent the identity matrix of dimension N, $\mathbf{0}_N \in \mathbb{R}^N$ a vector of all zeros and $\mathbf{0}_{N\times N} \in \mathbb{R}^{N\times N}$ a matrix of all zeros. The operator $\text{diag}(\cdot)$ returns a square diagonal matrix with the elements of the input vector on the main diagonal and the operator $[(\cdot)]$ returns a matrix containing a vertical stacking of the arguments.

We model the swarm with a directed sensing graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where the vertex set $\mathcal{V} = \{1\dots N\}$ represents the agents, and the edge set $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ contains the pairs of agents $(i, j) \in \mathcal{E}$ for which agent i can sense agent j. We denote as $\mathcal{N}_i = \{j \in \mathcal{V} | (i, j) \in \mathcal{E}\} \subset \mathcal{V}$ the set of neighbors of an agent i in $\mathcal{G}$ and as $|\mathcal{N}_i|$ its cardinality[1]. We also define δ_{ij}, a function of two agents i and j, which takes value 0 if $j \notin \mathcal{N}_i$ and 1 if $j \in \mathcal{N}_i$. Another concept borrowed from algebraic graph theory is algebraic connectivity [63], also known as *connectivity eigenvalue*. This is the second smallest eigenvalue of the Laplacian matrix [63] associated with the undirected

---

[1] Note that both the set of edges $\mathcal{E}$ and the one of neighbors $\mathcal{N}_i$ of a specific agent i are time-varying. However, we will omit their time dependency throughout the chapter for brevity.

graph $\mathcal{G}'$ obtained from $\mathcal{G}$ and it is usually denoted by λ_2. The algebraic connectivity has been extensively used in swarm robotics [64, 65] because the magnitude of this value reflects crucial qualities of the graph. However, its mathematical details are beyond the scope of this thesis.

5.2.2 Limited field of view

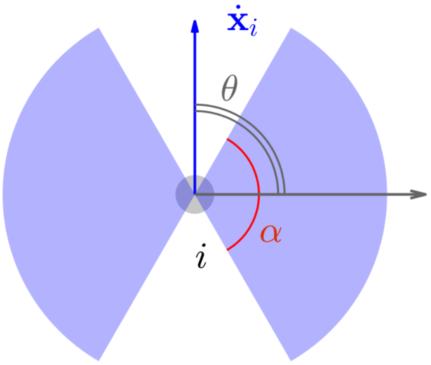


Figure 5.1 – **Illustration of the width and azimuth angles for an agent with limited FOV.** Width α and azimuth θ. In particular, this configuration is associated with values $\alpha = 120°$ and $\theta = 90°$. $\dot{\boldsymbol{x}}$ corresponds to the agent velocity and is aligned to its heading.

In this chapter, we study the influence of geometric visual constraints on the ability of swarming. For the sake of an easier visualization and interpretability of the results, we choose to implement our model in two dimensions. We endow every agent with two eyes, each having sight over a portion of the surroundings, i.e. a circular sector centered on the agent's position. The eyes are symmetrically placed about the direction of the agent's velocity. Since we are interested in dense swarms, we assume that the radius of perception is bigger than the size of the swarm, therefore we do not add a constraint on the visual range. To model the described configuration, we define two distinct parameters:

- the FOV *width* (α) is the angular span that an eye can detect and it ranges from 0° to 180°. On a UAV this would correspond to the FOV of an on-board camera;
- the FOV *azimuth* (θ) describes the visual direction of the eye and it varies from $\alpha/2$ to 90°. The choice to limit θ to 90° is justified by the analogy with the orientation of the birds eyes [88]. On a UAV this would correspond to the angle at which the camera center is placed w.r.t. the x-axis of the body-frame of the robot.

Fig. 5.1 provides an illustration for the two parameters, while Fig. 5.2 shows different configurations of the FOV for varying values of width α and azimuth θ.

Because of its limited FOV, an agent i can only sense a portion of the surrounding space. This defines the set of swarm members that i can perceive at time t, namely its neighborhood $\mathcal{N}_i$.

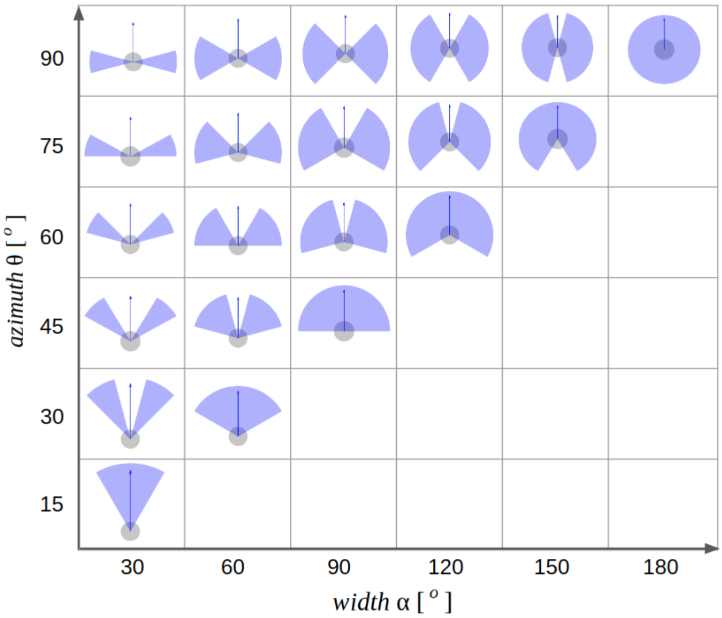


Figure 5.2 – **Sensing configurations of an agent with different width and azimuth values.** Note that the positive values on the θ-axis refer to the *right eye*, while respective values for the *left eye* have a negative sign for the hypothesis of symmetry that we assume. When $\theta = \alpha/2$ the two sensed regions overlap on one edge and merge in a unique circular sector. For every other azimuth value, the agent perceives two non-intersecting sectors and for $\theta = 90°$ (first row) the eyes face opposite directions.

Fig. 5.3 illustrates an example of the neighborhood of i, for a sensor configuration of $\alpha = 120°$ and $\theta = 90°$.

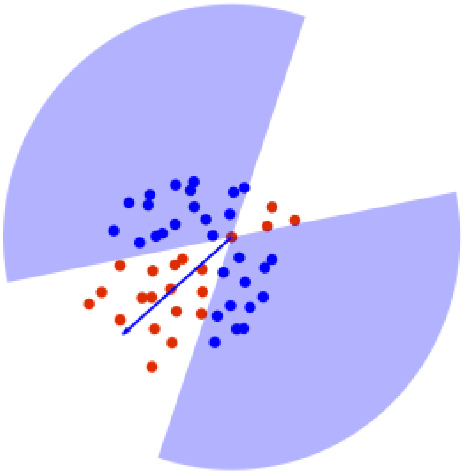


Figure 5.3 – **Reynolds swarm with limited-FOV agents**. The highlighted agent i has a visual sector of width $\alpha = 120°$ and azimuth $\theta = 90°$. The swarm members in blue are the ones perceived by i (the neighbors $\mathcal{N}_i$), while the unperceived members are in red.

5.2.3 Reynolds swarm model

For the mathematical definition of the Reynolds model, we refer to 2.2. In the Reynolds model, the choice of the parameter values (i.e., c_{coh}, c_{sep}, and c_{align}) is not unique and in many situations it is application-dependent. For instance, in operations of maximal area coverage, increasing the separation gain could help to amplify the spreading of the drones. Instead, in operations that require the swarm to squeeze through narrow canyons, the cohesion could be increased to make the group fit into a reduced space.

We restrict the gain values to the range 1 to 10. Notice that a separation gain equal to 1, $c_{\text{sep}} = 1$, means that whenever two agents are situated at 1 m of distance from each other, they are repulsed from each other with an acceleration of magnitude 1 m/s^2. Similar considerations can be done for the other gains.

Let us denote $\boldsymbol{p} = [\boldsymbol{p}_1, \boldsymbol{p}_2, \cdots \boldsymbol{p}_N]$, $\boldsymbol{v} = [\boldsymbol{v}_1, \boldsymbol{v}_2, \cdots \boldsymbol{v}_N]$, $\boldsymbol{u} = [\boldsymbol{u}_1, \boldsymbol{u}_2, \cdots \boldsymbol{u}_N] \in \mathbb{R}^{2N}$ respectively containing the positions, velocities, and commands of the agents of the swarm. The equation defining the motion of the swarm can be written as:

$$\boldsymbol{u} = c_{\text{align}} A\boldsymbol{v} + c_{\text{coh}} A\boldsymbol{p} - c_{\text{sep}} H \tag{5.1}$$

where $A \in \mathbb{R}^{2N\times 2N}$ is a matrix composed by blocks $A_{ij} = (\delta_{ij}/|\mathcal{N}_i|)\boldsymbol{I}_2 \in \mathbb{R}^{2\times 2}$ if $i \neq j$ and $A_{ij} = -\boldsymbol{I}_2 \in \mathbb{R}^{2\times 2}$ if $i = j$, and $H \in \mathbb{R}^{2N\times 2N}$ is composed by blocks $H_{ij} = (1/|\mathcal{N}_i|)\,\text{diag}(\bar{\boldsymbol{p}}_{ij}) \in \mathbb{R}^{2\times 2}$ if $i \neq j$ and $H_{ij} = \boldsymbol{0}_{2\times 2}$ if $i = j$.

5.2.4 Agents dynamics

In our simulations, the dynamics of the agents are reproduced in discrete time according to a double integrator model [58]. For every time step $t_k = k \cdot dt,\ k \in \{1, 2, 3 \ldots, K\}$ and every agent i, it holds

$$\boldsymbol{a}_i^k = \boldsymbol{u}_i^k \tag{5.2}$$

$$\boldsymbol{v}_i^k = \boldsymbol{v}_i^{k-1} + \boldsymbol{a}_i^k\, dt \tag{5.3}$$

$$\boldsymbol{p}_i^k = \boldsymbol{p}_i^{k-1} + \boldsymbol{v}_i^k\, dt \tag{5.4}$$

$$\tag{5.5}$$

where dt represents the step used for the temporal discretization of the system, and $\boldsymbol{p}_i^k \in \mathbb{R}^2$, $\boldsymbol{v}_i^k$, and $\boldsymbol{a}_i^k$ are the position, velocity, and acceleration of the i-th agent at time t_k, respectively.

To narrow the gap between simulation and reality we consider physical constraints on the magnitudes of velocities and accelerations, expressed by $\|\boldsymbol{v}_i^k\| \leq v_{\max}$ and $\|\boldsymbol{a}_i^k\| \leq a_{\max}$.

5.2.5 Swarm performance metrics

When limiting the FOV, the guarantee that the agents form a single group where all components move at the same speed in the same direction does not hold anymore. Indeed, two agents that do not perceive each other act independently. Under this condition, the swarm can split into multiple subgroups that do not affect each other's movements, and collisions can occur. To evaluate the collective performance, four relevant metrics are introduced. Several metrics for describing the correlation of the agents' movements and the collision risk have been adopted in the literature for describing both robotic systems and animal groups [13], [89], [17]. Furthermore, in swarm robotics, algebraic connectivity has proven to be useful in many applications, especially the ones where a flow of information has to be ensured among all the robots of the swarm (e.g., exploration and target tracking). By taking into account a metric directly related to this quantity we aim to make our analysis relevant also for those users who are interested in maintaining the connectivity of the graph while swarming with limited sensors. In this work, all metrics are scaled to be equal to 1 in the best-case scenario. Namely, they are:

- the instantaneous *order* metric, Φ^k_{order};
- the instantaneous *safety* metric, Φ^k_{safety};
- the instantaneous *union* metric, Φ^k_{union};
- the instantaneous *connectivity* metric, $\Phi^k_{\text{connectivity}}$.

For the definition, please refer to 2.6.

To evaluate the global performance of the swarm during the time T of a simulation, the metrics that have been defined above for a generic time instant t_k are averaged over the time window. Then, the order metric of the swarm referred to a simulation of length T is

$$\Phi_{\text{order}} = \frac{\sum_{k=1}^{K} \Phi^k_{\text{order}}}{K} \tag{5.6}$$

and the same holds for the other metrics.

5.3 Results

In this section, we present the results about the effects of limited visual sensing (modeled using the two parameters of width and azimuth introduced in Section 5.2.2) on the swarm performance metrics described in Section 5.2.5. An extensive analysis of their influence is carried out by running the Reynolds algorithm at different sensor configurations.

Besides depending on the visual configuration, the performance results also vary w.r.t. the Reynolds gains c_{align}, c_{coh} and c_{sep}. As we mentioned in Section 5.2.2, the choice of the right combination of those constants can be application-dependent and, therefore, non-unique. Consequently, we study how the performance metrics vary according to this choice.

In the first place, we select fixed triplets for the Reynolds gains and we analyze the metrics when the alignment, the cohesion, or the separation effect are privileged, once at a time. An additional case is studied for equal values of the gains.

Another approach can be preferred when we are aware of the metric w.r.t. our swarm should be optimized. Therefore, in a subsequent study, we analyze the results for Reynolds gains that are optimized for each of the metrics considered, one at a time. Genetic algorithms are applied to find the optimal swarm parameters. In the following, we present the simulation setup we used.

All simulations have been run in MATLAB R2017b. The workflow we used is summarized in Fig. 5.4. Specifically, we consider a swarm of $N = 50$ agents. The width angle α is varied with constant steps of 10° in the from 10° to 180°, while θ is increased with steps of 5° in the range from $\alpha/2$ to 90°. For each sensor configuration (α, θ) we apply a Monte-Carlo method that repeatedly selects random samples of the initial conditions for the swarm, i.e. initial positions and velocities of the agents $(\boldsymbol{x}_0, \dot{\boldsymbol{x}}_0)$. A batch of 100 simulations is run, and the final score is the average over the batch.

At $t_0 = 0\ s$ the agents are initialized with random positions and velocities. The former follow a uniform distribution over a cube of 10 m edge, while the latter obey a multi-normal law with mean $\mathbf{0}_{2N}\ m/s$ and covariance $3\boldsymbol{I}_{2N}\ m^2/s^2$. The discretization time step used in simulation is $\delta t = 0.05\ s$ and the total time is $T = 50\ s$.

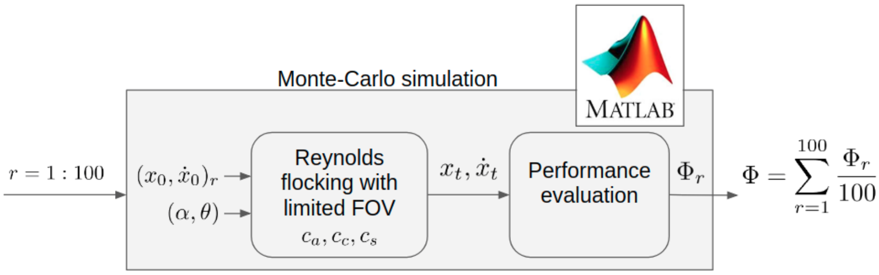


Figure 5.4 – **Simulation workflow for testing sensor configurations with Reynolds model.** Reynolds gains are c_{align}, c_{coh}, and c_{sep}. The outputs of the workflow are the performance metrics Φ_{order}, Φ_{safety}, Φ_{union} and $\Phi_{\text{connectivity}}$, averaged over the $r = 100$ repetitions of the Monte-Carlo method.

5.3.1 Fixed Reynolds coefficients

In this section, we evaluate the swarm flight properties when the Reynolds gains are fixed to triplets that privilege one effect over the others: alignment, cohesion, or separation. Finally, a triplet with equal gains is evaluated. The values applied in our simulations as summarized in Table 5.1.

c_{align}	c_{coh}	c_{sep}
10	1	1
1	10	1
1	1	10
1	1	1

Table 5.1 – List of the different Reynolds gain triplets used in the simulation.

As a general consideration, we may reasonably expect that a decline in the performance of the swarm is proportional to a diminution of the width angle α. Indeed, a decrease in the visual angle of the agents determines a reduction in the information that they can capture from the surroundings. This tendency is generally confirmed by the union and the connectivity metrics for all the choices of gains, in Fig. 5.6(I-IV,C-D). Instead, this is not the case for the order and the safety metrics in Fig. 5.6(I-IV,A-B), for which the trends shown are more complex.

If we observe the order metric Φ_{order} in Fig. 5.6(I), it is immediately noticeable that the configurations on the half-diagonals (from top left to the center) perform better than the average (the yellow color in the figures corresponds to high scores), and this fact is independent of the choice of gains. Fig.5.5 presents a graphical excerpt of the sensor configurations corresponding to these diagonal regions, from which we can infer the visual portions that they have in common: two thin lateral sectors. High values in order metric are also present in the upper regions of the figures, where the azimuth angle is $\theta = 90°$. Again, this holds for every choice of the Reynolds gains. These upper regions correspond to visual configurations with diametrically opposed sensors, having sight on the lateral sides. These results corroborate the hypothesis that lateral vision is important in biology, especially for those species exhibiting collective motion. Indeed, the majority of birds in nature present a narrow binocular sector that varies on average between 20° and 30° and a wide FOV that covers well the two lateral regions [88].

The results about the safety metric Φ_{safety} are highly dependent on the Reynolds gains. In fact, in Fig. 5.6(I-IV,B) we can observe very different patterns. Moreover, it can be noticed that this metric takes values in a very limited range. This is partly due to the normalization factor, which counts the number of all possible pairs of agents. Therefore, for a large swarm, the addition of one collision would correspond to a very limited decrease in the safety value. As intuition suggests, our results confirm that the safety metric deteriorates when the cohesion gain increases (see Fig. 5.6(II,B)). However, unexpectedly, the lowest safety values correspond to the biggest FOV widths. This can be explained by referring to the union metric Φ_{union} in

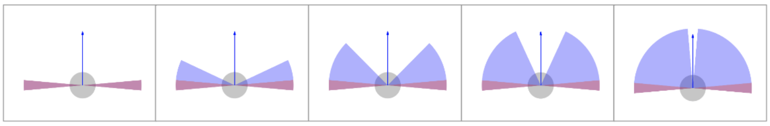


Figure 5.5 – **Visual configurations with high order scores**. Configurations of the FOV corresponding to the left half-diagonal of the (α, θ)-space (from top left to the centre) and associated with high scores in the order metric. In particular, from left to right, they are: $(\alpha = 10, \theta = 90)$, $(30, 80)$, $(50, 70)$, $(70, 60)$ and $(90, 50)$, where the width and azimuth angles are expressed in degrees [°]. The intersection between all of them is highlighted in red.

Fig. 5.6(II,C). High values of Φ_{union} correspond to the tendency of the agents to create a unique group, thereby increasing the chance of collision.

One non-trivial remark about the union metric in Fig. 5.6(III,C), is that for very small values of width α the values of the azimuth θ performing the best are the extreme ones, either close to 0° or to 90°. The explanation of this phenomenon involves more analysis of the system and it will be part of our future work.

As anticipated, a non-zero value of the connectivity metric $\Phi_{\text{connectivity}}$ is only possible when Φ^k_{union} takes the maximum value, 1, at least at some time instants. During a simulation, the swarm may split and rejoin. In such instances, the instantaneous connectivity metric $\Phi^k_{\text{connectivity}}$ passes from positive to null values and vice versa, and the global value of $\Phi_{\text{connectivity}}$ is the average over time. Globally, we notice that the larger the width angle, the higher the score.

5.3.2 Optimized Reynolds coefficients

The approach of finding optimal Reynolds parameters through genetic algorithms has been applied in previous work [13] and it is justified by the high non-linearity of the relationship that links the parameters to the performance function.

In Table 5.2 we resume the parameters used in the genetic algorithm.

Fig. 5.6(V,A-D) shows the results of the four metrics associated with the swarm simulations with optimized parameters. One consideration is that the plots display more discontinuities over the configuration-space (α, θ) compared to the previous results. A motivation for this is that genetic algorithms rely on a stochastic approach and they do not guarantee the convergence to the global optima. In addition, to keep the total simulation time affordable, our optimization algorithm involved a reduced number of individuals and generations.

In general, the trends highlighted in Section 5.3.1 are confirmed in this section. Indeed, similarly to the previous cases, the order metric Φ_{order} shows the best performing values on the diagonal and in the upper area. To a more accurate analysis, it seems that in the diagonal region the genetic algorithm led to some improvements, i.e. the yellow portion in Fig. 5.6(V,A)

Parameter	Value
Variables	$c_{\text{align}}, c_{\text{coh}}, c_{\text{sep}}$
Range for the variables	$[1,10] \times [1,10] \times [1,10] \in \mathbb{R}^3$
Fitness function	Φ_{order}, Φ_{safety}, Φ_{union}, $\Phi_{\text{connectivity}}$
Population size	10
Number of generations	10
Scaling operator	’proportional’
Selection operator	’tournament’
Crossover operator	’scattered’
Mutation operator	’gaussian’

Table 5.2 – **Parameters of the genetic algorithm used to determine the optimal Reynolds gains for swarms with limited FOV**. The algorithm is part of the built-in functions in MATLAB. The optimization is performed w.r.t. each of the metric functions and for every sensor configuration (α, θ).

is wider. The safety metric Φ_{safety} globally presents high values, with some outliers in the upper region, where poorer results were already present for fixed Reynolds gains. The union metric Φ_{union} varies in a smaller range, indicating a global enhancement. Finally, the connectivity metric $\Phi_{\text{connectivity}}$ does not display substantial modifications and presents positive scores only on the right corner where the width angle is wide.

5.4 Discussion

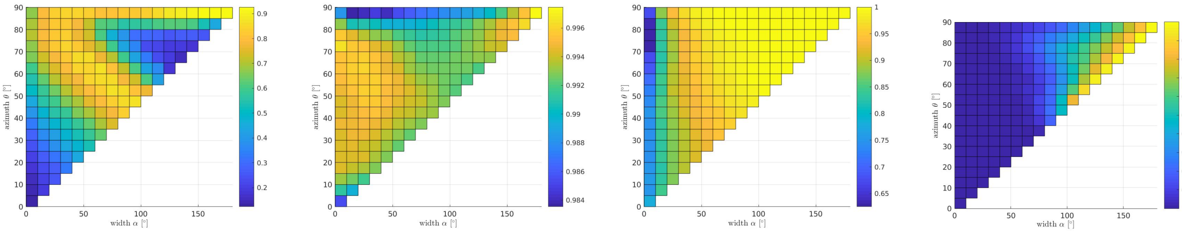
In this chapter, we presented a numerical analysis of the impact of limited visual sensing on swarm systems, from the perspective of different and complementary performance metrics.

We believe that the results presented in this thesis chapter can be a starting point for filling the gap between simulations and reality when implementing swarm algorithms. Moreover, this analysis could be used to solve the problem of optimal sensor placement, when the application of the swarming system is known a priori and a choice of the sensors and their orientation has to be made.

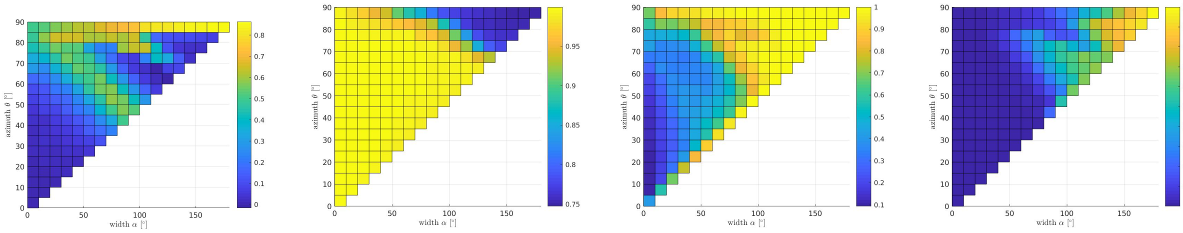
The results presented are promising and create new avenues of research that are worth investigating. More complex models should be taken into account to observe the effects of different drone dynamics on the Reynolds model, e.g. fixed-wing or quadrotor drones, and how the sensory limitations affect their flight in swarms. Moreover, noise modeling and delays should be taken into account to narrow the gap between simulation and reality.

In addition to the limited FOV, we would like to consider other limitations of the visual sensors such as limited range and possible occlusions generated by other agents [86]. Indeed, in the real-world, visual perception is influenced by the distance. Higher definition measurements are obtained for nearby neighbors and worse for far-away ones. Moreover, every object detected by a camera generates a blind cone behind it, occluding the view over a part of the

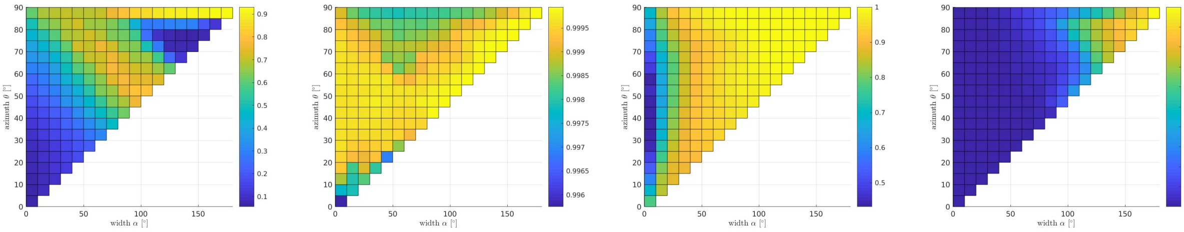
(A) Order (B) Safety (C) Union (D) Connectivity



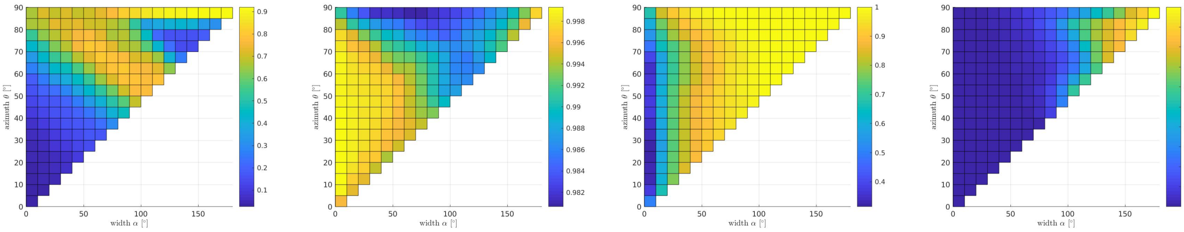
(I) Reynolds gains: $c_{\text{align}} = 10$, $c_{\text{coh}} = 1$, $c_{\text{sep}} = 1$



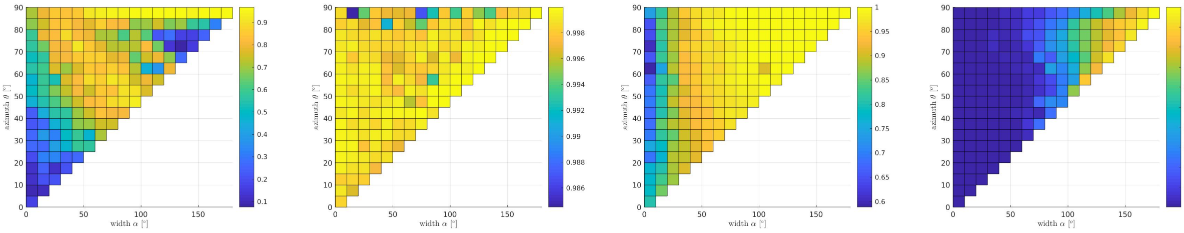
(II) Reynolds gains: $c_{\text{align}} = 1$, $c_{\text{coh}} = 10$, $c_{\text{sep}} = 1$



(III) Reynolds gains: $c_{\text{align}} = 1$, $c_{\text{coh}} = 1$, $c_{\text{sep}} = 10$



(IV) Reynolds gains: $c_{\text{align}} = 1$, $c_{\text{coh}} = 1$, $c_{\text{sep}} = 1$



(V) Reynolds gains: optimized for each configuration

Figure 5.6 – **Simulation results of swarms with limited FOV and different Reynolds gains.** Every row is associated with a triplet of the Reynolds gains, whose values are specified in the first column. The columns are referred to the different performance metrics. From left to right, order (Φ_{order}), safety (Φ_{safety}), union (Φ_{union}), and connectivity ($\Phi_{\text{connectivity}}$). The bottom row shows the results of the swarm performance when the Reynolds gains of every sensor configuration are optimized w.r.t. a given metric. In the plots, the x axis represents the width angle α [°], varying from 0° to 180°, while the y axis represents the azimuth angle θ [°], varying from 0° to 90°. (Continue on the next page.)

Figure 5.6 – **Simulation results of swarms with limited FOV and different Reynolds gains.** (Continued from the previous page.) Every bin is computed as the average of 100 simulations with randomized initial conditions, i.e. positions and velocities of the swarm's agents. Notice that the color maps are rescaled according to the ranges of the metrics values.

surrounding space. Finally, in order to make our approach more realistic, we think that is important to add obstacles along the navigation path and analyze how the behavior of the swarm adapts

6 Scalable vision-based swarms in the presence of occlusions

This chapter addresses the scalability of vision-based drone swarms in terms of group size and density. For this, we consider swarm navigation in open environments without obstacles and we employ a potential-field-based model. We evaluate the visibility model with up to one thousand point mass agents, showing that occlusions have adverse effects on the inter-agent distances and velocity alignment as the swarm scales up, both in terms of group size and density. In particular, we find that small agent displacements have considerable effects on neighbor visibility and lead to control discontinuities. We show that the destabilizing effects of visibility switches, i.e., agents continuously becoming visible or invisible, can be mitigated if agents select their neighbors from adjacent Voronoi regions. The results show that Voronoi-based interactions enable vision-based swarms to remain collision-free, ordered, and cohesive in the presence of occlusions. These results are consistent across group sizes and agent densities.

The work presented in this chapter is adapted from [90]:

- F. Schilling, E. Soria, and D. Floreano, "On the scalability of vision-based drone swarms in the presence of occlusions," in *IEEE Access*, vol. 1, no. 1, pp. 1–13, (submitted) Aug. 2021.

6.1 Introduction

Vision-based relative localization methods rely entirely on local information to detect other agents, thus removing the dependence on external localization systems and additional communication infrastructure. Moreover, vision is arguably the ideal sensory modality for localization on aerial robots since cameras are small, lightweight, and provide extremely high information density at comparatively low power consumption [91]. Multi-robot systems that use a vision-based approach to mutual localization have recently emerged in the form of leader-follower formations [92, 47, 93, 94] and the first aerial flocks [95, 96, 97]. Important perceptual factors such as visual occlusions, i.e., agents that are obstructed by others, are usually neglected in these swarms because of their small group size. However, these factors become a deterrent for larger swarms, especially when they have to fly in dense configurations.

While some swarm roboticists explicitly make use of visual occlusions to solve collaborative transport problems [98] and robotic shepherding tasks [99], the most thorough treatment of visibility constraints can be found in the collective motion literature. Using computer vision techniques, researchers are able to reconstruct the poses and visual fields of individual animals and show that visual perception best explains how information about food sources and predators transfers within the group [19, 20, 100, 101]. How individuals select and react to their neighbors is one of the fundamental questions in the study of collective motion and agent-based swarm models provide an indispensable tool to test and verify different hypotheses [102, 103, 15, 104]. Notable examples of neighbor selection methods include *metric* (i.e., within a metric radius) [22], *topological* (i.e., the set of *n* nearest neighbors) [61], or *voronoi*-based (i.e., from adjacent Voronoi regions) [105] interactions. Recently, different forms of *visual* neighbor selection have gained popularity due to their biological plausibility [19, 20, 100, 101]. For example, research on swarm models with a limited field of view shows that lateral vision is crucial for collision-free collective motion [106, 26] and may explain why flocking birds have almost omnidirectional vision [88]. Simulations of large schools of fish show that visual obstructions lead to more realistic group shapes and densities than purely metric interactions [107]. Simulations of large vision-based flocks show that bird density can be regulated effectively if individuals only react to the projection of their neighbors [108]. Other researchers show that many natural behaviors such as milling and polarized flocking emerge from purely visual interactions even in the absence of a spatial representation of neighbors [109]. Although these models offer interesting collective behaviors, they often make modeling choices that are geared towards a particular species or result in undesirable behavior for robotic swarms since they lead to frequent collisions.

In this chapter, we tackle visibility constraints arising from occlusions from a robotics perspective with the goal of synthesizing large and compact vision-based drone swarms. In particular, we study the effect of occlusions on the performance (i.e., collision avoidance, cohesion, and velocity alignment) of vision-based swarms as they scale from low densities and a handful of agents to high-density swarms with thousands of individuals. To this end, we propose a *visual* neighbor selection model that offers a perceptually plausible alternative to the ubiquitous but unrealistic *metric* selection of neighbors, i.e., methods that assume agents can sense all neighbors within a given radius. We simulate vision-based swarms of up to one thousand point mass agents and program them to perform collective waypoint navigation using a simple attractive/repulsive swarm algorithm. The results show that swarms in which agents react to all *visible* neighbors perform poorly, especially at high densities and as the group size increases beyond tens of agents. However, by limiting visual interactions to their Voronoi neighbors, we can successfully synthesize collision-free, cohesive, and ordered vision-based swarms. A comparison of Voronoi interactions with other common neighbor selection methods (i.e., metric and topological) reveals their superiority in large, high-density swarms. We validate the scalability of the resulting swarm algorithm at different densities and group sizes with quadcopter dynamics using a simulator with realistic physics and noise levels. The analysis shows that visually-constrained Voronoi interactions are both perceptually plausible and highly effective for the coordination of large aerial robot swarms in which agents rely purely on local visual information for control.

6.2 Method

We aim to synthesize a vision-based swarm that remains as compact as possible and collision-free while performing collective waypoint navigation. We define this objective since it enables many practical applications such as cooperative mapping, aerial deliveries, and search & rescue. In the following, we restrict ourselves to swarms that operate in two-dimensional planar configurations.

For the navigation of the swarm, we use a variant of the Reynolds algorithm that synthesizes cohesion and collision avoidance, as well as a migration behavior (Sec. 6.2). To obtain a swarm algorithm that is plausible for vision-based swarms, we define the notion of agent visibility in the form of a neighbor selection strategy that is based on a realistic occlusion model (Sec. 6.2.1). Since vision-based detection is an inherently stochastic process, we further model sensing noise on the range and bearing measurements (Sec. 6.2.2).

The motion of each agent can be described by single-integrator dynamics of the form

$$\boldsymbol{p}_i^{k+1} = \boldsymbol{p}_i^k + \boldsymbol{v}_i^k dt \tag{6.1}$$

where k denotes the index of the discrete-time step with duration dt.

In the remainder of the section, we skip the dependence on the discrete-time step k for notational brevity and clarity. However, all computations in this section are performed at every time step without exception.

Swarm algorithm

The objective of the swarm is to perform waypoint navigation while avoiding inter-agent collisions and staying together as a group. We formulate this objective as an artificial potential field that is inspired by the Reynolds swarm algorithm [21]. The motion of an agent is composed of an attractive/repulsive potential that provides separation and cohesion between agents (Sec. 6.2), as well as a migratory potential responsible for goal-directed navigation (Sec. 6.2).

The motion of an agent is composed of a social term that captures agent-to-agent interactions and a migration term that introduces the navigation objective. The velocity command of an agent can be written as

$$\tilde{\boldsymbol{v}}_i = \boldsymbol{v}_{\mathrm{soc},i} + \boldsymbol{v}_{\mathrm{mig},i} \tag{6.2}$$

where $\boldsymbol{v}_{\mathrm{soc},i}$ and $\boldsymbol{v}_{\mathrm{mig},i}$ denote the respective social (Eq. 6.3) and migration terms (Eq. 6.4). In order to obtain a final velocity command that is feasible even under the actuation constraints of a physical robot, we limit the maximum speed as $\boldsymbol{v}_i = \tilde{\boldsymbol{v}}_i / \|\tilde{\boldsymbol{v}}_i\| \min(\|\tilde{\boldsymbol{v}}_i\|, v_{\max})$. The velocity command $\boldsymbol{v}_i$ can then be used directly for the motion update to obtain the agent positions of the next time step (Eq. 6.1).

Separation and cohesion

Cohesion and collision avoidance can be achieved with an attractive/repulsive potential that keeps the agents at an equilibrium distance. The cohesion term keeps the swarm together by attracting agents to the average position of their neighbors. The separation term leads to collision avoidance by repulsing nearby agents from each other. We can express these rules more formally as:

$$\boldsymbol{v}_{\mathrm{soc},i} = c_{\mathrm{coh}} \underbrace{\frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} \boldsymbol{d}_{ij}}_{\text{cohesion}} - c_{\mathrm{sep}} \underbrace{\sum_{j \in \mathcal{N}_i} \frac{\boldsymbol{d}_{ij}}{\|\boldsymbol{d}_{ij}\|^2}}_{\text{separation}} \tag{6.3}$$

where c_{coh} and c_{sep} are gains that regulate the strength of the attraction and repulsion, respectively.

Note that we do not scale the separation velocity command by the number of agents. This formulation has the advantage that minimum inter-agent distances remain quasi-constant as the group size increases and thus reduces the need for readjusting the control gains (Fig. 6.1). We further use the analytical solution to the above equations for three agents as a first approximation of the desired inter-agent distance d_{ref}. This allows us to express an approximate reference distance by using a separation gain of the form $c_{\mathrm{sep}} = (d_{\mathrm{ref}})^2/2\ m/s$ and keeping the cohesion gain fixed at $c_{\mathrm{coh}} = 1\ m/s$. Note that in general, the separation gain slightly overestimates the reference distance for larger swarms since it does not take the number of neighbors into account. It is nevertheless a useful approximation that spares us the tedious task of finding the reference distance empirically for each agent swarm scale separately.

Migration

The purpose of the migration term is to give the agents a navigation goal by steering them towards a waypoint. The migration term can be written as

$$\boldsymbol{v}_{\mathrm{mig},i} = c_{\mathrm{mig}} \frac{\boldsymbol{p}_{\mathrm{mig}} - \boldsymbol{p}_i}{\|\boldsymbol{p}_{\mathrm{mig}} - \boldsymbol{p}_i\|} \tag{6.4}$$

where c_{mig} denotes the gain for modulating the migration speed.

6.2.1 Neighbor selection

Neighbor selection methods define the set of neighbors of an agent i, $\mathcal{N}_i \subseteq \mathcal{V}$, and are defined in Chap. 2. We summarize in Table 6.1 the different methods used in this chapter. In particular,

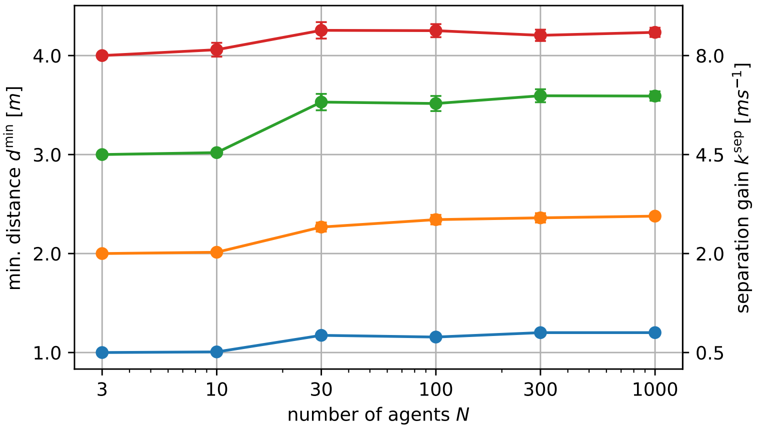


Figure 6.1 – **Scalability of nearest neighbor distance for varying group sizes and densities.** Scalability of minimum nearest neighbor distances to increasing numbers of agents using the baseline *metric* neighbor selection model, i.e. agents within the perception radius are detected irrespective of whether they are occluded. Each line represents the minimum equilibrium distance between nearest neighbors obtained from different separation gains as the swarm size increases (mean and std. dev. over ten trials, all other parameters constant). Aside from a noticeable increase of inter-agent distances between ten and thirty agents that occurs due to the saturation of the perception range with agents, the inter-agent distances remain constant across different group sizes (note the logarithmic scale).

Name	Set notation
metric	$\mathcal{N}_i^{\text{metric}}(r)$
visual	$\mathcal{N}_i^{\text{visual}}(r, r_{\text{agent}})$
visual + myopic	$\mathcal{N}_i^{\text{visual}}(2d_{\text{ref}}, r_{\text{agent}})$
visual + topological	$\mathcal{N}_i^{\text{visual}}(r, r_{\text{agent}}) \cap \mathcal{N}_i^{\text{topo}}(n)$
visual + voronoi	$\mathcal{N}_i^{\text{visual}}(r, r_{\text{agent}}) \cap \mathcal{N}_i^{\text{voronoi}}$

Table 6.1 – **Neighbor selection methods used for the experiments.**

we call *metric* method the selection method based on the Euclidean distance. We define as *visual* method the combination of the metric and the occlusion-based methods. In other words, the visual method only counts the neighbors within a perception radius r which are also fully unoccluded.

Note that the adjacency matrix is not necessarily symmetric and the resulting graph may be directed. The metric and voronoi neighbor selection mechanism result in an undirected graph and a symmetric adjacency matrix, whereas visual and topological neighbor selection are generally asymmetric and directed. In other words, the visibility between a pair of agents $i \sim j$ does not imply that the inverse relationship $j \sim i$ is true

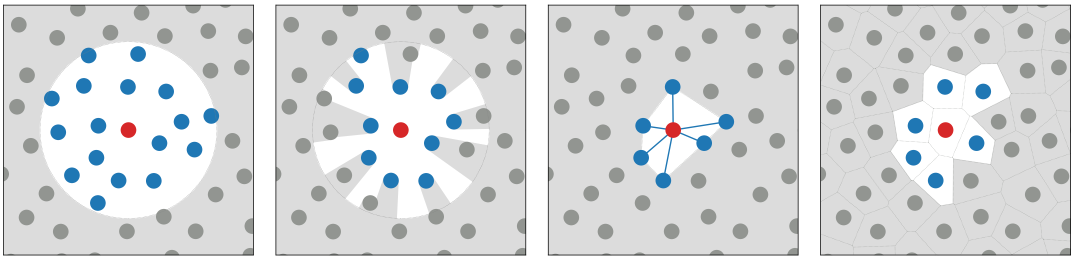

(a) Metric (b) Visual (c) Topological (d) Voronoi

Figure 6.2 – **Neighbor selection strategies.** Schematic visualization of different neighbor selection strategies: (a) metric, (b) visual, (c) topological, and (d) Voronoi-based. We take the perspective of a focal agent within a swarm (central red disk) that selects agents (blue disks) and discard others (gray disks) depending on the following selection criteria: (a) *metric* selects all agents within a metric perception radius, (b) *visual* selects all visible agents within a metric radius, i.e., all agents that appear large enough and are not occluded by others, assuming agents are equally sized and have an omnidirectional camera at their center, (c) *topological* selects only the n closest agents (here $n = 6$), irrespective of their distance, and (d) *voronoi* selects only those agents that belong to a neighboring Voronoi region.

6.2.2 Sensing noise

We model the visual relative localization inaccuracies in two independent components: range and bearing. We model range noise as a function that varies linearly with relative distance from the observer whereas the bearing noise is constant over the field of view [110, 111, 93, 97]. More formally, we define the noisy version of range and bearing with which agent i detects agent j as:

$$\hat{d}_{ij} = d_{ij}(1+\omega_d), \qquad \omega_d \sim \mathcal{N}(0,\sigma_d) \tag{6.5}$$

$$\hat{\psi}_{ij} = \psi_{ij} + \omega_\psi, \qquad \omega_\psi \sim \mathcal{N}(0,\sigma_\psi) \tag{6.6}$$

where ω_d and ω_ψ are independent and identically distributed white noise with zero mean and standard deviation of σ_d and σ_ψ, respectively. The noisy relative position can then be constructed from polar coordinates as

$$\hat{\boldsymbol{d}}_{ij} = \begin{bmatrix} \hat{d}_{ij}\cos(\hat{\psi}_{ij}) \\ \hat{d}_{ij}\sin(\hat{\psi}_{ij}) \end{bmatrix} \tag{6.7}$$

where $\hat{\boldsymbol{d}}_{ij}$ can serve directly as an input to the social term of the swarm algorithm (Eq. 6.3). The exact values for range and bearing noise depend on several factors such as camera resolution, lens quality, calibration accuracy, and target deformation.

6.3 Experimental setup

We briefly describe the experimental setup and parameters (Sec. 6.3.1), as well as the simulation environments that are used to obtain the experimental results (Sec. 6.4).

6.3.1 Experimental parameters

We perform ten repeated runs of migration experiments to make statistical statements about the scalability of the swarm using different neighbor selection methods, group sizes, swarm densities, agent dynamics, and noise levels.

The specific parameter values we use are informed by our previous experiments with real vision-based quadcopters in indoor [47] and outdoor environments [97], as well as the literature on vision-based drone localization [110, 95, 111, 112, 74, 113, 114, 115, 92]. We choose the radius of an agent as $r_{\text{agent}} = 0.25\ m$ since it reflects a common physical size of quadcopter platforms used in robotic experiments. The perception radius $r = 10\ m$ is chosen as the distance at which other drones were no longer reliably detected during outdoor experiments. The time delta $dt = 0.1\ s$ is chosen as a reasonable amount of time to solve the visual perception, state estimation, and control problems in real-time. The desired inter-agent distance is set to $d_{\text{ref}} = 1\ m$ to generate the most compact formation that simultaneously provides enough safety margin against potential collisions.

In order to provide a fair comparison of the visual neighbor selection methods, we choose parameter values that result in comparable numbers of neighbors as the group size increases (Fig. 6.3d). In particular, we set the maximum number of agents for topological neighbor selection to $n = 6$ since it reflects the average number of Voronoi neighbors for planar configurations [62]. We further let $r = 2d_{\text{ref}}$ for myopic interactions since it approaches an average number of six neighbors as the group size increases. We provide an overview of the neighbor selection methods used during the experiments in Table 6.1.

At the beginning of each experiment, the agents are spawned randomly within a circular region. The initial positions are sampled uniformly in a non-overlapping fashion using rejection sampling such that no pair of agents are closer than their desired reference distance d_{ref}. The area of the circular region is chosen such that the agent density ρ_N remains constant for different numbers of agents. The agents exhibit no motion at the beginning of the experiment, i.e., their initial velocities are set to zero. The agents are given a constant navigation direction $\boldsymbol{d}_{\text{mig}} = [1,0]$ along the horizontal axis which can be seen as a migratory route along the magnetic field [16]. We let the swarm develop its collective motion for a total of $T = 200\ s$ composed of 2000 isochronous discrete time steps k with duration $dt = 0.1\ s$. At each time step, the agents select their neighbors according to the indicated neighbor selection function (Fig. 6.2) and compute their motion command (Sec. 6.2). We set the separation and cohesion gains to $c_{\text{sep}} = 1\ m/s$ and $c_{\text{coh}} = 1\ m/s$ to provide an approximate nearest neighbor distance of $d_{\text{ref}} = 1\ m$. The separation gain is set to $c_{\text{mig}} = 0.5\ m/s$ which provides goal-directed motion

Description	Notation	Value
Agent radius	r_{agent}	0.25 m
Reference distance	d_{ref}	1 m
Perception radius	r	10 m
Bearing noise	σ_β	1°
Range noise	σ_d	0.05 m
Maximum topological neighbors	n	6
Maximum speed	$v_{\max}$	1 m/s
Separation gain	c_{sep}	1 m/s
Cohesion gain	c_{coh}	1 m/s
Migration gain	c_{mig}	0.5 m/s
Time step	dt	0.1 s
Simulation duration	T	200 s

Table 6.2 – **Parameters used during the experiments.**

without overpowering the attractive/repulsive commands. We set the maximum speed an agent can sustain to $v_{\max} = 1\ m/s$. A concise overview of the experimental parameters is provided in Table 6.2.

In order to provide a fair comparison across vastly different group sizes, we compute the metrics over the last quarter of the simulation, i.e. considering only the final 500 time steps. Particularly for large swarm sizes, we avoid computing metrics during an initial transient period in which agents have not yet aggregated to their final configuration. We refer to the time range during which we compute the metrics as the equilibrium period for convenience.

We report the minimum nearest neighbor distances as a minimum over time over the equilibrium period since it reveals whether collisions occur. For the order and union metrics, we report time averages over the equilibrium period. The mean and standard deviations are computed over the ten independent runs with random initial conditions.

6.4 Results

We report results on two sets of complementary simulation experiments: (i) we compare several neighbor selection methods with increasing numbers of agents to show their performance for different swarm sizes (Sec. 6.4.1), (ii) we evaluate the neighbor selection methods for increasing inter-agent distances to show the effect of varying agent number densities on the swarm performance (Sec. 6.4.2).

6.4.1 Performance across swarm sizes

We assess the performance of the swarm for all neighbor selection methods and six increasing group sizes $N \in \{3, 10, 30, 100, 300, 1000\}$. We set the reference distance $d_{\text{ref}} = 1\,\text{m}$ constant throughout the experiments to keep the agent number density fixed and to allow a direct comparison of the effect of group size.

Visual neighbor selection

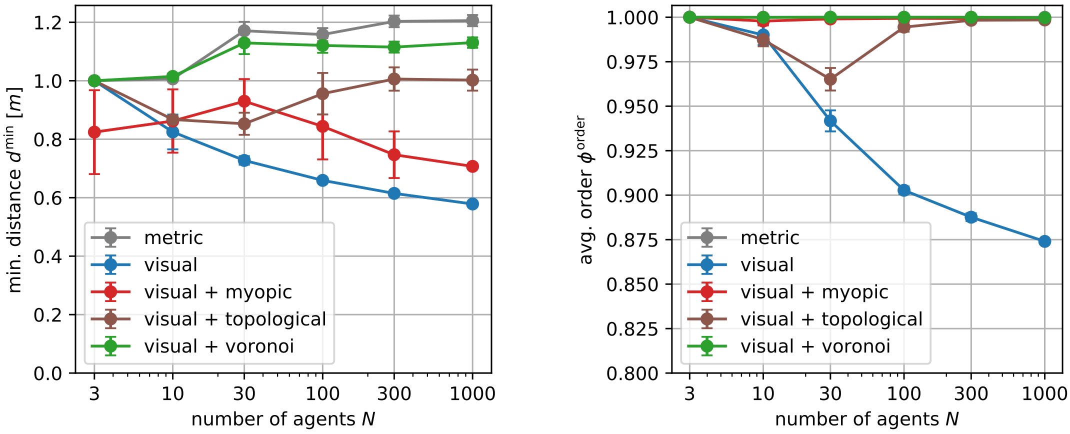
Purely *visual* neighbor selection shows the overall lowest performance as the group size increases. There is a considerable performance penalty in the distance and order metrics (Fig. 6.3a and 6.3b). The minimum distance is tracked well only for a group size of 3 agents ($d_{\text{min}} = 1.0 \pm 0.0\,\text{m}$; Fig. 6.3a). The distance gradually approaches the collision threshold of $2r_{\text{agent}} = 0.5\,\text{m}$ and reaches its minimum at 1000 agents ($d_{\text{min}} = 0.58 \pm 0.0\,\text{m}$; Fig. 6.3a). The order metric shows a similar trend since the agents start out perfectly ordered for 3 agents ($\Phi_{\text{order}} = 1.0 \pm 0.0$; Fig. 6.3b). However, for larger group sizes, the order metric decreases monotonously until reaching its minimum at 1000 agents ($\Phi_{\text{order}} = 0.87 \pm 0.0$; Fig. 6.3b). The swarm stays cohesive as a single unit across all group sizes ($\Phi_{\text{union}} = 1.0 \pm 0.0\,\text{m}$; Fig. 6.3c). Generally, using *visual* neighbor selection, the swarm performance decreases as soon as occlusions start to emerge (Fig. 6.3d; Fig. 6.3d). There is no performance penalty for 3 agents using *visual* neighbor selection since they predominantly occur in equilateral triangle formations in which there are no occlusions (i.e., $N_i = 2$). For larger group sizes, an increasing number of agents within the perception radius is occluded (32% occluded for $N = 10$; up to 90% occluded for $N = 1000$).

Qualitatively, the trajectories of agents using purely *visual* neighbor selection are jittery (Fig. 6.4a). The agents migrate with considerable deviations from the optimal linear trajectory in the migration direction. In particular, the relative positions of the agents within the swarm are not fixed but rather subject to frequent topology switches. For instance, agents that initially belong to the swarm periphery move towards the swarm center (Fig. 6.4a; blue line) and vice versa.

The topology switches can be explained by considering that an agent within the swarm is exposed to constant changes of its neighbor set (Fig. 6.6). Small agent displacements result in considerable changes of perspective that cause neighbors to appear and disappear from the visible set (Fig. 6.6a and 6.6b: 11 agents appear and 4 disappear, for example). Here, the focal agent is exposed to a total of 32 visibility switches (8 ± 1.22 switches per timestep) over the course of four consecutive seconds of the experiment.

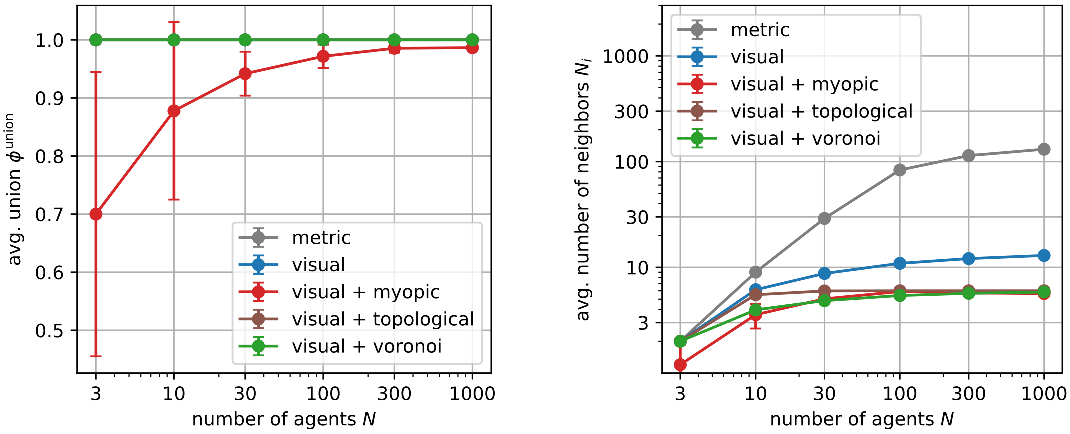
Alternatives to purely visual neighbor selection

Neighbor selection based on the Voronoi tesselation shows the highest performance of all neighbor selection methods across group sizes. The minimum distance, order, and union



(a) Distance

(b) Order

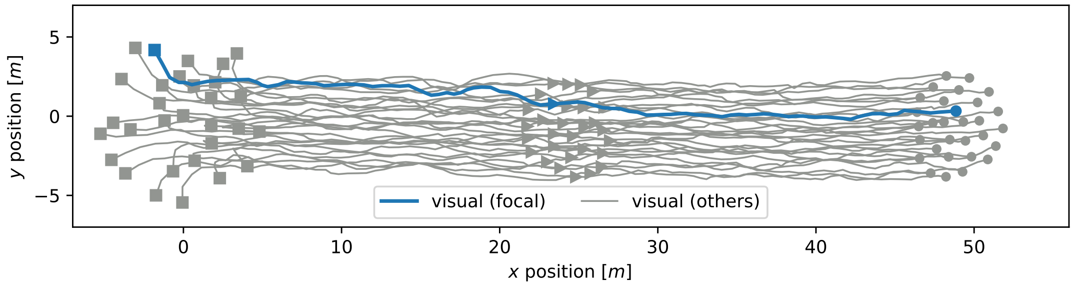


(c) Union

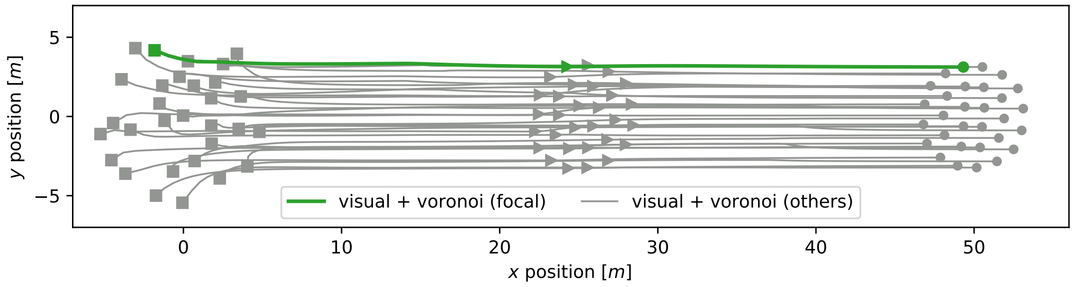
(d) Neighbors

Figure 6.3 – **Swarm performance of neighbor selection methods for varying group sizes.** We show the effect of different neighbor selection methods on the (a) minimum nearest neighbor distance d_{min}, (b) average order Φ_{order}, (c) average union Φ_{union}, and (d) the average number of neighbors N_i, expressed as a function of the number of agents N (note the logarithmic scale). Here, $d_{\mathrm{ref}} = 1\,\mathrm{m}$. The neighbors are selected as follows: 1) *metric* selects all agents within the perception radius $r = 10\,\mathrm{m}$, 2) *visual* selects all visible agents, 3) *visual* + *myopic* selects all visible agents within a smaller radius $r = 2\,\mathrm{m}$, 4) *visual* + *topological* selects the $n = 6$ topologically closest visible neighbors, and 5) *visual* + *voronoi* selects the neighbors from adjacent Voronoi regions. The Voronoi neighbor selection method scales most predictably with the number of vision-based agents, i.e., distance, order, and union remain quasi-constant as the swarm size increases.

metrics show performance comparable to *metric* neighbor selection (Fig. 6.3a, 6.3b, and 6.3c). In particular, the minimum distance is tracked even closer to the reference distance of $d_{\mathrm{ref}} = 1\,\mathrm{m}$ for increasing group size (for 1000 agents: $d_{\mathrm{min}} = 1.13 \pm 0.02\,\mathrm{m}$ for *visual* and



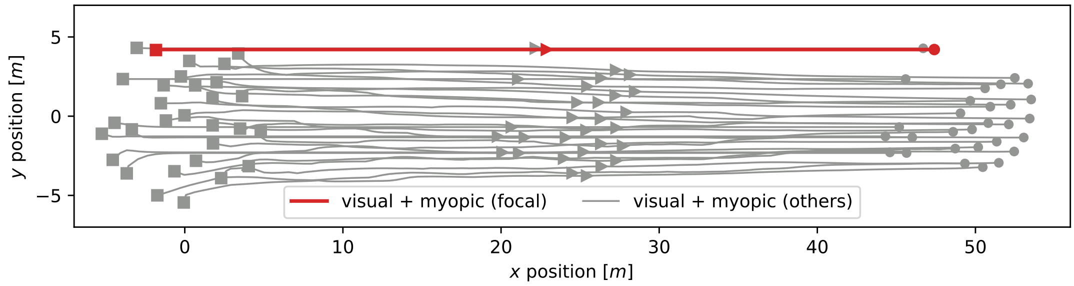
(a) Visual



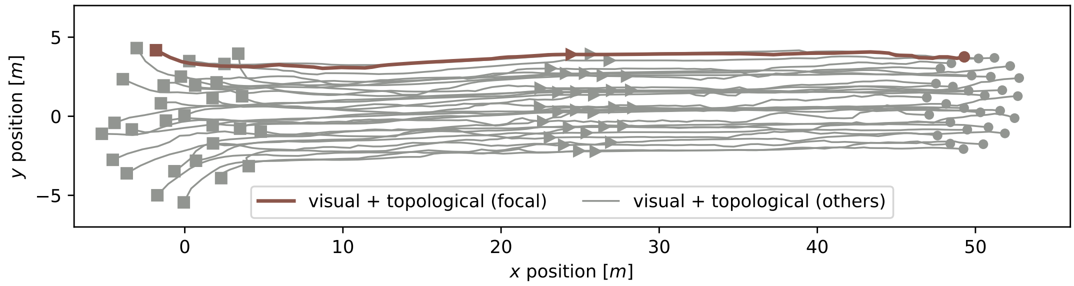
(b) Visual + voronoi

Figure 6.4 – **Swarm trajectories using visual and Voronoi neighbor selection methods.** Example trajectories of a swarm of thirty agents during a single run of the collective migration experiment using (a) visual and (b) Voronoi neighbor selection mechanisms. We use the same random seed to create equal initial conditions and highlight an arbitrary focal agent (colored, thick line) to reveal its motion among the other agents (grey, thin lines). The agents start from their initial positions (solid squares) on the left and migrate along the horizontal axis (solid triangles) to the right side of the virtual arena (solid disks). (a) Visual neighbor selection leads to control discontinuities and disorder; agents frequently change positions inside the swarm. (b) Visual and Voronoi neighbor selection together result in collision-free, ordered, and cohesive migration (see Fig. 6.5 for continuation).

$d_{\min} = 1.21 \pm 0.02\,\mathrm{m}$ for *metric*, for example; Fig. 6.3a). This can be explained by considering that *metric* swarms have a significantly larger number of neighbors compared those based on *visual* + *voronoi* neighbor selection for group sizes $N > 3$ (Fig. 6.3d). For example, at $N = 1000$ agents, the *metric* neighbor set contains around 22 times the number of agents than it does for *visual* + *topological* neighbor selection (on average 11.2 ± 8.5 times the number of neighbors for all group sizes; Fig. 6.3d). Recall that the swarm algorithm computes the separation term as a sum of reciprocal distances (Eq. 6.3). Therefore, each neighbor has an additive contribution towards the repulsion (albeit a very small one for distant agents) that explains the slightly larger distances. The agents are perfectly ordered and cohesive for all group sizes ($\Phi_{\text{order}} = 1.0 \pm 0.0$ and $\Phi_{\text{union}} = 1.0 \pm 0.0$, respectively; Fig. 6.3b and 6.3c). Qualitatively, the paths taken by *visual* + *voronoi* swarms are generally linear and smooth (Fig. 6.4b). The swarm performs collision-free, ordered, and cohesive collective migration. Switches in the neighbor set do occur but are infrequent and do not lead to unsafe situations or disorder (e.g., changes in neighbor



(a) Visual + myopic

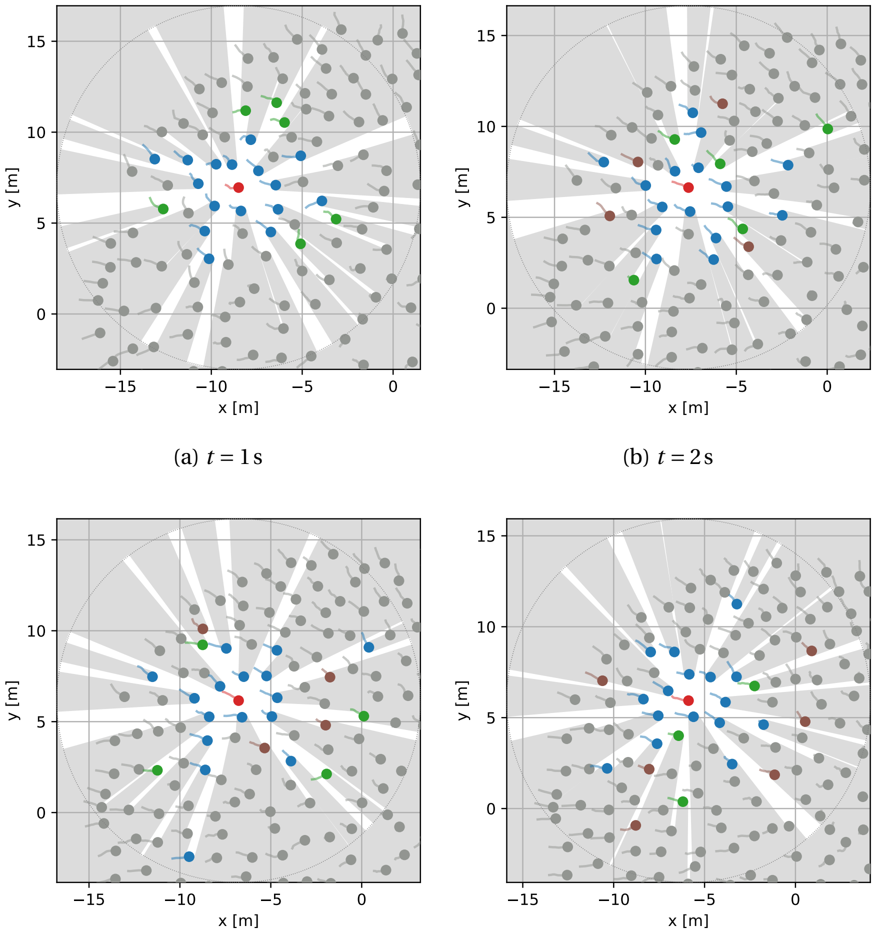


(b) Visual + topological

Figure 6.5 – **Swarm trajectories using myopic and topological neighbor selection methods.** Example trajectories of a swarm of thirty agents during a single run of the collective migration experiment using (a) myopic and (b) topological neighbor selection mechanisms (see Fig. 6.4 for description). (a) Myopic visual interactions mitigate the discontinuities but lead to fragmentation. (b) Visuo-topological interactions mitigate strong discontinuities but swarms are not well-ordered, especially for peripheral agents.

configuration at $x \approx 23\,\mathrm{m}$; Fig. 6.4b).

Swarms that use *visual + myopic* or *visual + topological* neighbor selection do not perform as well as those using *visual + voronoi* selection for different group sizes. Generally, *visual + myopic* swarms exhibit low cohesion and easily fragment into several subgroups (Fig. 6.3c). Fragmentation occurs because agents that exit the perception radius are usually found within small subgroups or entirely isolated due to their limited perception range (see subgroups and isolated agent; Fig. 6.5a). The fragmentation phenomenon also skews the minimum distance metric towards lower values with large standard deviations compared to other neighbor selection methods (average of $d_{\mathrm{min}} = 0.82 \pm 0.12\,\mathrm{m}$ across group sizes; Fig. 6.3a). This occurs because isolated agents are usually far away from any other agent (see isolated agent; Fig. 6.5a). We verified that minimum distances to nearest neighbors are usually well-tracked within subgroups of at least three agents. The union metric is always below $\Phi_{\mathrm{union}} < 1$ which indicates that fragmentation occurs for all group sizes (Fig. 6.3c). Cohesion is lowest for small groups and approaches, but never reaches, a value of $\Phi_{\mathrm{union}} = 1$ that would indicate a single-unit cohesive swarm ($\Phi_{\mathrm{union}} = 0.7 \pm 0.25$ for $N = 3$, up to $\Phi_{\mathrm{union}} = 0.98 \pm 0.0$ for $N = 1000$; Fig. 6.3c). Note that larger groups exhibit higher union performance since the metric is normalized by



(a) $t = 1\,\mathrm{s}$ (b) $t = 2\,\mathrm{s}$

(c) $t = 3\,\mathrm{s}$ (d) $t = 4\,\mathrm{s}$

Figure 6.6 – **Schematic representation of the switching topologies caused by visual occlusions.** Visual representation of the switching topologies caused by occlusions during a collective migration experiment. We show the perspective of an arbitrary focal agent (central red disk) over the course of four isochronous time steps $t \in \{1\,\mathrm{s}, 2\,\mathrm{s}, 3\,\mathrm{s}, 4\,\mathrm{s}\}$. The focal agent uses *visual* neighbor selection and therefore perceives only agents within its perception radius that are in a direct line of sight (blue disks), whereas occluded agents are invisible (grey disks). We further highlight visibility switches, i.e., when an agent that has been occluded since the previous time step becomes visible (green disks) and when a previously visible agent becomes occluded (brown disks). A total of 32 visibility switches occur over the course of four seconds.

group size, i.e., larger groups consist of fewer subgroups relative to the overall group size. Swarms with *visual* + *myopic* neighbor selection are effectively ordered ($\Phi_{\text{order}} = 1.0 \pm 0.0$; Fig. 6.3b) Qualitatively, apart from fragmentation, larger subgroups tend to have irregular shapes that are less circular compared to other neighbor selection methods (see the largest subgroup; Fig. 6.5a).

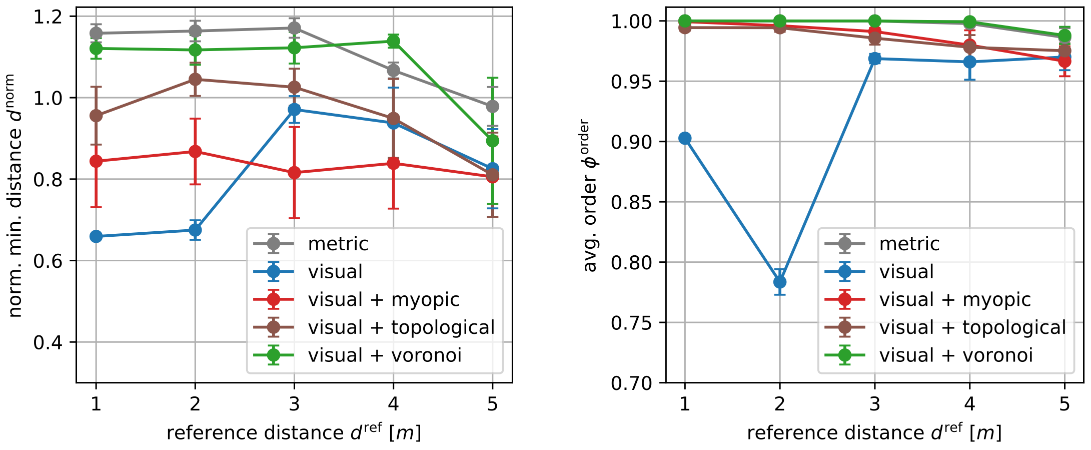
Swarms that use *visual* + *topological* neighbor selection do not exhibit consistent performance accross swarm sizes. Especially for intermediate group sizes of 10, 30, and 100 agents, both minimum distances and order metrics suffer a decrease in performance (Fig. 6.3a and 6.7b, respectively). For the respective distances and order metrics, the minimum performance occurs at 30 agents ($d_{\min} = 0.85 \pm 0.04\,\mathrm{m}$ and $\Phi_{\mathrm{order}} = 0.97 \pm 0.01$; Fig. 6.3a and 6.3b, respectively). We can explain this behavior by considering that agents *always* select the six closest visible neighbors, irrespective of where they are located. Agents that belong to the swarm center tend to have six neighbors that are spaced around them at approximately equal angles from each other. Conversely, agents on the periphery consider only neighbors in one direction which are subject to occlusions. This leads to similar visual switching topologies as for the purely *visual* neighbor selection, albeit less severe since even the most distant nearest neighbor for $n = 6$ is usually in close proximity. The effect of occlusions is mostly mitigated for larger swarm sizes $N > 100$ since a smaller proportion of agents is located on the periphery relative to the swarm center. We do not observe fragmentation with *visual* + *topological* neighbor selection for any group size ($\Phi_{\mathrm{union}} = 1.0 \pm 0.0$; Fig. 6.3c). Qualitatively, *visual* + *topological* interactions generate paths that are not perfectly straight (Fig. 6.5b). We also observe swarms that exhibit rotations, as well as ones that periodically switch between a set of recurring configurations.

6.4.2 Performance across swarm densities

We evaluate the swarm performance for all neighbor selection methods and for five levels of increasing inter-agent distances $d_{\mathrm{ref}} \in \{1\,\mathrm{m}, 2\,\mathrm{m}, 3\,\mathrm{m}, 4\,\mathrm{m}, 5\,\mathrm{m}\}$. We let $N = 100$ to fix the group size and to enable a direct comparison between agent number densities. We define the normalized minimum nearest neighbor distance as $d_{\mathrm{norm}} = d_{\min} / d_{\mathrm{ref}}$ to make the minimum distances more easily comparable for different agent densities.

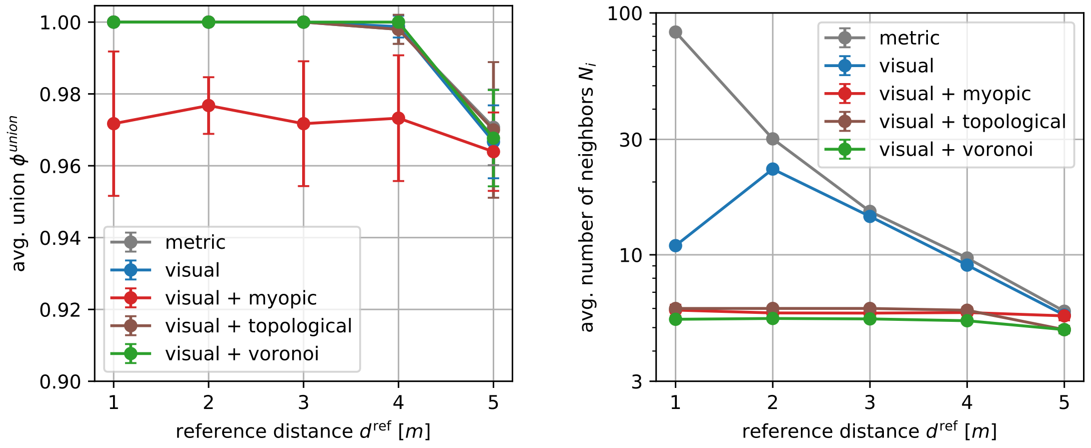
Visual neighbor selection

Purely *visual* neighbor selection does not show consistent performance for different swarm densities. The performance penalty in distance and order is especially severe for agents in high-density configurations with small reference distances (Fig. 6.7a and 6.7b, respectively). The normalized distance is much lower than the desired reference of $d_{\mathrm{norm}} \geq 1$ and has its minimum for $d_{\mathrm{ref}} \in \{1\,\mathrm{m}, 2\,\mathrm{m}\}$ ($d_{\mathrm{norm}} = 0.66 \pm 0.01$ and $d_{\mathrm{norm}} = 0.67 \pm 0.02$, respectively; Fig. 6.7a). For larger reference distances, $d_{\mathrm{ref}} \in \{3\,\mathrm{m}, 4\,\mathrm{m}\}$, the normalized distance stabilizes again to larger values ($d_{\mathrm{norm}} = 0.97 \pm 0.03$ and $d_{\mathrm{norm}} = 0.94 \pm 0.09$, respectively; Fig. 6.7a) Note that the minimum distance, order, and union metrics for large reference distances $d_{\mathrm{ref}} = 5\,\mathrm{m}$ decrease for all neighbor selection methods. A reference distance of $d_{\mathrm{ref}} = r/2 = 5\,\mathrm{m}$ effectively renders all neighbor selection methods *myopic* and fragmentation starts to occur. The union metric indicates that this is indeed the case for $d_{\mathrm{ref}} = 5\,\mathrm{m}$ since all neighbor selection methods show comparable mean performance to *myopic* swarms (average of all neighbor selection methods $\Phi_{\mathrm{union}} = 0.97 \pm 0.01$; Fig. 6.7c). The order metric reaches its minimum at $d_{\mathrm{ref}} = 2\,\mathrm{m}$ ($\Phi_{\mathrm{order}} = 0.78 \pm 0.01$; Fig. 6.7b). The minimum order coincides with the maximum of the



(a) Distance

(b) Order



(c) Union

(d) Neighbors

Figure 6.7 – **Swarm performance of neighbor selection methods for varying group density.** Swarm performance during the collective migration experiment for different neighbor selection methods and group size $N = 100$. We show the effect of different neighbor selection methods on the (a) normalized minimum nearest neighbor distance d_{norm}, (b) average order Φ_{order}, (c) average union Φ_{union}, and (d) average number of neighbors N_i, expressed as a function of the reference distance d_{ref}. The neighbors are selected as follows: *metric* selects all agents within the perception radius $r = 10\,\text{m}$, *visual* selects all visible agents, *visual* + *myopic* selects all visible agents within a smaller radius $r = 2\,\text{m}$, *visual* + *topological* selects the $n = 6$ topologically closest visible neighbors, and *visual* + *voronoi* selects the neighbors from adjacent Voronoi regions. With the exception of *myopic* conditions (at $\boldsymbol{d}_{\text{ref}} = 5\,\text{m}$), the Voronoi neighbor selection method scales most predictably with the density of the vision-based swarm and the performance remains quasi-constant as the reference distance increases.

average number of visible neighbors at $d_{\text{ref}} = 2\,\text{m}$ ($N_i = 22.62 \pm 0.04$). This indicates that order follows an inverse relationship with the number of visible neighbors: if more agents are visible, the likelihood of visual topology switches that lead to disorder increases (Fig. 6.6). The neighbor graph also highlights that the effect of occlusions is maximized at intermediate densities. At high densities, the nearest neighbors occlude most agents in all directions (87% occluded for $d_{\text{ref}} = 1\,\text{m}$; Fig. 6.7d). Conversely, the effect of occlusions diminishes at lower densities since the agents are not large enough to break the line of sight (5% occluded for $d_{\text{ref}} = 3\,\text{m}$, for example; Fig. 6.7d).

Alternatives to purely visual neighbor selection

The Voronoi-based neighbor selection provides the highest and most consistent performance across different group densities. The distance, order, and union metrics remain stable for all but the lowest density level ($d_{\text{ref}} = 5\,\text{m}$) at which interactions are rendered *myopic* (Fig. 6.7a, 6.7b, and 6.7c; Sec. 6.4.2 for discussion of *myopic* interactions). The normalized distance, order, and union remain stable for high and intermediate swarm densities (average $d_{\text{norm}} = 1.12 \pm 0.03\,\text{m}$, $\Phi_{\text{order}} = 1.0 \pm 0.0$, and $\Phi_{\text{union}} = 1.0 \pm 0.0$; Fig. 6.7a, 6.7b, 6.7c, respectively).

Swarms with *visual* + *myopic* and *visual* + *topological* interactions perform comparatively poorly to *visual* + *voronoi* neighbor across group densities. The *visual* + *myopic* neighbor selection method shows consistently low performance in terms of distance and union metrics (Fig. 6.7a and 6.7c). Myopic interactions effectively reduce the negative impact of occlusions. However, they also induce low distances and fragmentation (average $d_{\text{norm}} = 0.84 \pm 0.02$ and $\Phi_{\text{union}} = 0.97 \pm 0.01$ across reference distances; Fig. 6.7a and 6.7c, respectively). Swarms with *visual* + *topological* interactions can avoid the fragmentation issues but their minimum distances fluctuate for different densities (e.g., $d_{\text{norm}} = 1.05 \pm 0.04$ for $d_{\text{ref}} = 2\,\text{m}$ and $d_{\text{norm}} = 0.95 \pm 0.10$ for $d_{\text{ref}} = 4\,\text{m}$; Fig. 6.7a).

6.5 Conclusions

Methods for multi-agent coordination often make unrealistic assumptions about the information that is available to the individual agent. One of the most pervasive simplifying assumptions is that vision-based agents can sense the state of *all* surrounding neighbors within a metric perception radius, even if they are obstructed by closer ones. Here, we break this common assumption and construct a simple yet realistic model of visibility that selects neighbors only if (i) they appear large enough in the field of view, and (ii) are not occluded by other agents. Extensive swarm simulations with the visual occlusion model show that perfectly ordered metric-based swarms become disordered and unsafe when agents react to all of their visible neighbors. These adverse effects can be attributed to small perspective changes that continuously influence the set of visible neighbors, thus causing the agents to move in reaction to the new neighbor configuration. We show that this interplay between visibility constraints and collective motion can lead to severe instabilities for vision-based swarms, especially for large numbers of agents and high swarm densities.

Selecting a subset of visible neighbors from adjacent Voronoi regions significantly improves the swarm performance (i.e., collision avoidance, velocity alignment, and group cohesion) across group sizes and densities. Controlled experiments with subsets of the visual neighbors show that Voronoi-based interactions are a more effective countermeasure against occlusions than metric and topological ones. The main drawback of metric and topological neighbor selection methods is their dependence on specific parameters, namely the perception range and the number of nearest neighbors, respectively. Choosing favorable values for these parameters that provide high performance at all group sizes and densities may be impossible for vision-based swarms. In particular, swarms that select too many neighbors suffer from the adverse effects of occlusions, and selecting too few neighbors inevitably leads to fragmentation. Voronoi-based interactions provide an elegant solution to this problem since they are both parameter-free and spatially balanced [61].

The occlusion model presented here is undoubtedly useful but it neglects an important aspect of vision-based relative localization: errors due to misdetections. False positives (i.e., detecting an agent that is not there) and false negatives (i.e., not detecting an agent that is defacto there) inevitably occur in real-world conditions but are notoriously difficult to model. The main difficulty is that the distribution of misdetections depends not only on the used hardware and detection algorithm but also on environmental conditions such as background clutter and lighting conditions. Multi-target filtering algorithms can alleviate errors due to sensing noise and false positive detections to some extent but are largely ineffective against false negatives [97].

We argue that occlusions should not be neglected when designing algorithms for vision-based swarms since they are comparatively easy to model. We consider the occlusion model presented here simple enough to be a drop-in replacement for algorithms that would otherwise default to purely metric interactions. Simple agent-based simulations can thus prevent significant hardware damage by considering occlusions early in the algorithm design and before they are implemented on real robots. The validation presented here is specifically geared towards drones but we expect the results to translate well to other types of vision-based robots.

7 Sensor-based predictive control of aerial swarms

In this chapter, we extend our previous work on predictive swarm models to purely sensor-based agents. While removing the communication requirements between agents, we make use of a simple but perceptually realistic neighbor selection method that discards occluded agents. Instead of communicating their future trajectories, the agents predict them based on the local knowledge of the environment. We evaluate our model in simulation in a forest-like environment at different swarm sizes and we show the swarm can avoid collisions, while the flight synchrony worsen and the trajectory lengths increase compared to the communication-based DMPC model. We also compare the sensor-based swarm performance with a potential-field model from the state of the art and show that the DMPC swarm has overall better flight performance across different sizes.

7.1 Introduction

In communication-based swarms, drones are typically equipped with wireless communication devices that allow the exchange of state information such as positions and velocities with each other [23, 14, 10]. While this approach has enabled successful swarm deployments, it presents inherent limitations. Firstly, it lacks scalability since the bandwidth requirement of wireless communication scales quadratically with the number of individuals [116]. As a result, its utilization in large swarms leads to compounding delays whose durations are difficult to estimate and thus require dampening and interpolation [12, 13]. Secondly, this approach lacks flexibility since all agents must adhere to the same communication protocol and localize themselves in the same reference frame.

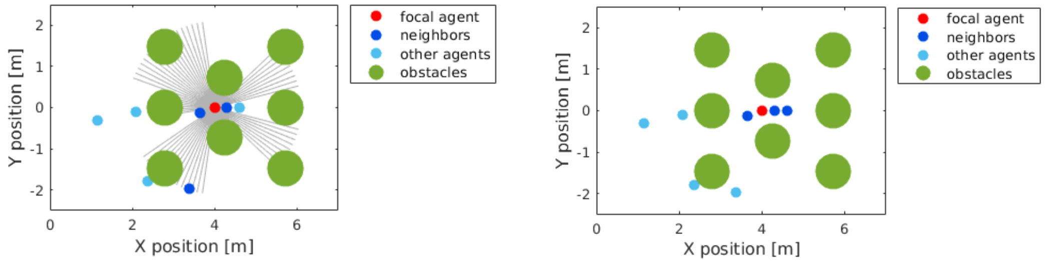
Decentralized multi-robot systems that do not rely on communication have recently emerged and are generally based on visual or depth on-board sensing [47, 117, 95, 96, 118]. When using these sensor modalities, occlusions become an important factor to be considered. Occlusions make some agents to be obstructed and hence not perceivable by others. While in swarms with low density they can be neglected without consequences, in large swarms flying in dense configurations their effects can become fatal.

Among the most popular decentralized models for the collision-free navigation of drone swarms that do not necessarily involve communication we find Potential Field (PF)-based methods. They are based on the design of artificial forces that replicate the behaviors of biological swarms such as cohesion, collision avoidance, and migration to a common destination [21, 22]. Their implementation only requires the knowledge of momentary information on neighboring agents, and specifically relative positions and, in some cases, velocities [23, 13, 96]. Both variables can be estimated on-board with visual sensors [118, 96]. Other models are based on optimization techniques such as Optimal Reciprocal Collision Avoidance (ORCA) and Sequential Convex Programming (SCP). They provide a safer framework since they explicitly include anti-collision constraints. ORCA computes the set of non-colliding velocities for an agent and then selects the robot command to optimize its path towards the final destination [51, 52]. SCP methods solve sequential optimization problems with convex approximations of collision constraints. They can explicitly consider the robot actuation constraints in the model [48, 49]. Both require the knowledge of neighbor positions and velocities. Buffered Voronoi Cells generate more conservative paths but use only relative positions [54]. Finally, end-to-end learning has provided a novel alternative for the generation of safe swarm trajectories by imitation of the techniques above, both in the absence and presence of obstacles [47, 57].

In previous work [10], we show that future state prediction can improve the safety and synchrony of flying agents when navigating cluttered environments as compared to other approaches. Our method is based on Distributed Model Predictive Control (DMPC) and incorporates the swarm behaviors of cohesion, migration, and collision avoidance within a constrained quadratic problem. However, this method makes use of local-based communication. In this work, we remove the communication requirement between agents, and we elaborate a DMPC swarm model that requires only line-of-sight sensor-based detection. Hence, our neighbor selection method discards occluded agents. We compare the communication-based model with the new sensor-based model in simulation and we quantify the performance drop at different swarm sizes. Then, we compare the sensor-based DMPC model to a popular PF-based swarm model from the state of the art and we show the benefits of state prediction.

7.2 Method

We consider a set of N homogeneous agents labeled by $i \in \mathcal{V} = \{1, \dots, N\}$ and a set of M static obstacles labeled by $m \in \mathcal{M} = \{1, \dots, M\}$. The swarm can be modeled with a directed sensing graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where the vertex set $\mathcal{V}$ represents the agents, and the edge set $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ contains the pairs of agents (i, j) for which agent i can access the state information of agent j by either communication or local state estimation. The state of the i-th agent is represented by $\boldsymbol{x}_i = (\boldsymbol{p}_i, \boldsymbol{v}_i) \in \mathbb{R}^6$ and is made of its position $\boldsymbol{p}_i \in \mathbb{R}^3$ and velocity $\boldsymbol{v}_i \in \mathbb{R}^3$. The input $\boldsymbol{u}_i$ represents a position command. In the following, k denotes the index of a discrete time step with duration dt, $\boldsymbol{O}_d$ denotes the zero matrix of dimension d, and $\boldsymbol{I}_d$ denotes the identity matrix of the same dimension d.



(a) Visual neighbors

(b) Topological neighbors

Figure 7.1 – **Neighbor selection methods**. On the left, vision-based neighbor selection. On the right, communication-based neighbor selection. Both figures show a top view of the swarm flying in the forest-like environment. Cylindrical obstacles (green) appear as circles. In the vision-based method, the focal agent (red) can localize only non-occluded agents within the perception range $r = 2\ m$ (blue). Instead, the state of agents occluded by obstacles or other agents (light blue) can not be inferred. In the communication-based method, neighbors are selected according to the topological metric (i.e., the n-nearest agents to the focal agent).

7.2.1 Model of a flying agent

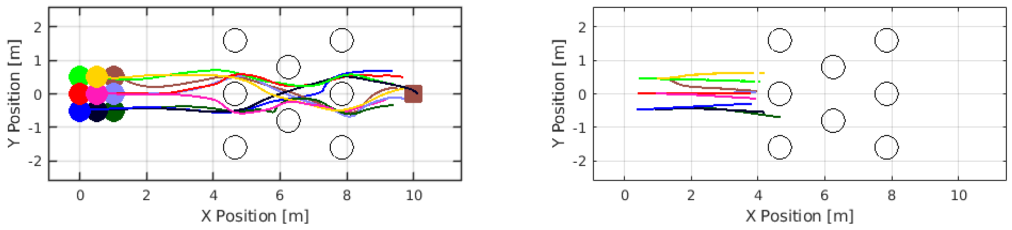
Every agent is approximated by a point-mass model that can be written as:

$$\boldsymbol{x}_i(k+1) = \boldsymbol{A}_i \boldsymbol{x}_i(k) + \boldsymbol{B}_i \boldsymbol{u}_i(k) \tag{7.1}$$

where $\boldsymbol{A}_i = [\boldsymbol{0}_3\ \boldsymbol{0}_3; -\boldsymbol{I}_3/dt\ \boldsymbol{0}_3]$ and $\boldsymbol{B}_i = [\boldsymbol{I}_3; \boldsymbol{I}_3/dt]$ are constant matrices. To account for the environment boundaries and the dynamic feasibility, we limit the position commands and the acceleration by constant vectors, i.e., $\boldsymbol{p}_{\text{min}} \leq \boldsymbol{u}_i(k) \leq \boldsymbol{p}_{\text{max}}$ and $\boldsymbol{a}_{\text{min}} \leq \boldsymbol{a}_i(k) \leq \boldsymbol{a}_{\text{max}}$. We model the agents' trajectories with l Bezier curves in $\mathbb{R}^3$ of duration T_l, in the same spirit as [32, 78]. This parameterization allows us to define continuous input trajectories to the drones from a finite set of control points. A 3-dimensional Bezier curve of order d is uniquely characterized by a set of $d+1$ control points $\tilde{\mathcal{U}} = \{\tilde{\boldsymbol{u}}_0, ..., \tilde{\boldsymbol{u}}_d\} \in \mathbb{R}^{3(d+1)}$ and the trajectory of agent i is defined by $l(d+1)$ control points $\tilde{\boldsymbol{U}}_i \in \mathbb{R}^{3l(d+1)}$. In the following, $\boldsymbol{x}_i$ and $\boldsymbol{u}_i$ are considered as function of the new unknown $\tilde{\boldsymbol{U}}_i$.

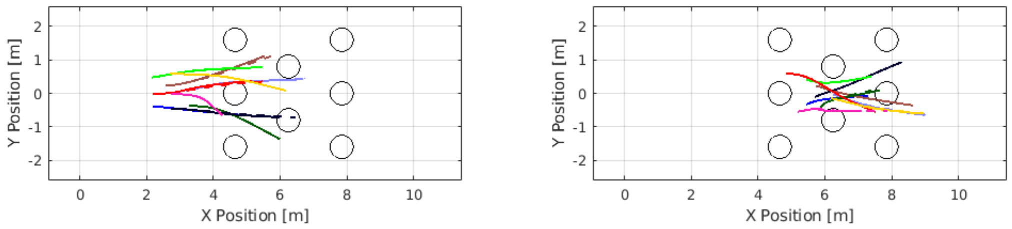
7.2.2 Inter-agent collision avoidance

A critical aspect of swarm flight is collision avoidance between agents. With sensor-based DMPC, agents compute their neighbor trajectories at every time step, but they are not guaranteed that these predictions are correct. Hence, safety constraints have to be conservative to prevent collisions in all situations. For this, we use conservative constraints that account for the the maximum braking distance d_{brake} of an agent. In particular, if we approximate the volume of an agent with a sphere of radius r_{agent}, the constraint that an agent should satisfy to



(a) Trajectories

(b) Time: 2s



(c) Time: 4s

(d) Time: 8s

Figure 7.2 – **Snapshots of predicted trajectories of the sensor-based DMPC swarm**. 7.2a top view of the trajectories of a swarm of 9 agents. 7.2b, 7.2c, and 7.2d, snapshots of the predicted trajectories of the swarm agents at 2, 4, and 8 s. Approximations of the neighbor predicted trajectories are dashed. The predicted trajectories can sometimes intersect obstacles because only the first obstacle on the collision course in time order is included in the optimization problem.

avoid collisions with a neighbor j is:

$$d_{ij}(k+\pi|k) \geq 2(r_{\text{agent}} + d_{\text{brake}}) \tag{7.2}$$

where $d_{\text{brake}} = \frac{1}{2} a_{\min} dt^2 + v_{\max} dt$ is the braking time for an agent flying at its maximum speed.

7.2.3 DMPC swarm model

In the DMPC swarm model, every drone i calculates its desired trajectory at each time step k over a fixed time window called the *prediction horizon* and denoted as $T_P = P\ dt$, $P \in \mathbb{N}^+$.

The constrained optimization problem aims at minimizing a cost function that encodes the swarm behavior and comprise a term for migration $J^k_{\text{mig},i}$, which steers the agents towards a common goal, the regulation of the inter-agent distance, which consists of cohesion $J^k_{\text{coh},i}$ and agents' reciprocal avoidance $J^k_{\text{saf-agent},i}$, and the obstacle avoidance $J^k_{\text{saf-obs},i}$, which steers the agents away from obstacles. Additionally, the control effort term minimizes the energy spent on maneuvering $J^k_{\text{effort},i}$. The problem is expresses by:

$$
\begin{aligned}
&\min_{\tilde{\boldsymbol{U}}_i^k, \mathcal{E}_i^k, \boldsymbol{\Delta}_i^k, \mathcal{Z}_i^k} J^k_{\text{mig},i} + J^k_{\text{saf-agent},i} + J^k_{\text{coh},i} + J^k_{\text{saf-obs},i} + J^k_{\text{effort},i} \\
&\text{subject to:} \\
&\boldsymbol{A}_{\text{dyn},i} \tilde{\boldsymbol{U}}_i^k \leq \boldsymbol{b}_{\text{dyn},i} \\
&\boldsymbol{A}_{\text{cont},i} \tilde{\boldsymbol{U}}_i^k = \boldsymbol{b}_{\text{cont},i} \\
&\boldsymbol{A}^k_{\text{saf-agent},i} [(\tilde{\boldsymbol{U}}_i^k)^T, (\mathcal{E}_i^k)^T]^T \leq \boldsymbol{b}^k_{\text{saf-agent},i} \\
&\boldsymbol{A}^k_{\text{coh},i} [(\tilde{\boldsymbol{U}}_i^k)^T, (\boldsymbol{\Delta}_i^k)^T]^T \leq \boldsymbol{b}^k_{\text{coh},i} \\
&\boldsymbol{A}^k_{\text{saf-obs},i} [(\tilde{\boldsymbol{U}}_i^k)^T, (\mathcal{Z}_i^k)^T]^T \leq \boldsymbol{b}^k_{\text{saf-obs},i} \\
&-\mathcal{E}_i^k \leq 0 \\
&-\boldsymbol{\Delta}_i^k \leq 0 \\
&-\mathcal{Z}_i^k \leq 0
\end{aligned}
\tag{7.3}
$$

where $\tilde{\boldsymbol{U}}_i$ is the parameterized trajectory of agent i to be optimized. The constraints include dynamic limitations of the agents, trajectory continuity and smoothness, soft constraints on the inter-agent safety, safety against obstacles, and cohesion. $\mathcal{E}_i^k$, $\boldsymbol{\Delta}_i^k$, and $\mathcal{Z}_i^k$ are slack variables that relax corresponding hyperplane constraints and make it more likely for the optimization problem to find a viable path.

Although the problem formulation is the same for both communication and sensor-based models, they present two key differences. The first concerns the neighbor selection method, the second is how trajectories of neighboring drones are obtained. We explain both of them in the following.

7.2.4 Neighbor selection

In the communication-based swarm model, we assume that each agent can exchange information with its local neighbors by explicit communication. We define the neighborhood of agent i, $\mathcal{N}_i{}^k = \{j \in \mathcal{V} \mid (i, j) \in \mathcal{E}\}$ as the set of the n nearest neighbors of agent i at time k [19].

Instead, in the sensor-based swarm, the agents cannot explicitly communicate their state or their optimized trajectories, but they infer their neighbors' state with on-board line-of-sight sensors, such as for example omnidirectional cameras. It is, therefore, reasonable to consider that only the state of the agents within a perception range r can be measured, while the state of agents outside this area can not. For the sensor-based model, we define the neighborhood of an agent i, $\mathcal{N}_i{}^k$, as the subset of the agents which are visible from i (Fig. 7.1). With the assumption that all agents are homogeneous and equally sized, we can use the perception range to represent visual acuity, i.e., the minimum size that another agent spans on the retina of the focal agent before it can no longer be perceived. In addition, full or partially occluded agents are considered invisible, i.e., only agents with an uninterrupted line of sight are contained in the visible set. This assumption is reasonable for monocular vision since the relative distance to other agents can only be reliably estimated if all of their spatial extent is visible. The neighbor set of the communication-based method has a fixed cardinality, while the neighbor set of the vision-based method has a variable cardinality.

7.2.5 Neighbor predicted trajectory

In the communication-based model, the neighbors of agent i, $\mathcal{N}_i^k$, are communicated its predicted trajectory at time k and use it for computing their desired trajectory. Specifically, the neighbors predicted trajectory intervenes in the cohesion and agent collision avoidance behaviors. Instead, in the vision-based model, agents are not allowed to communicate. The agents predict the trajectory of their neighbors by solving a DMPC problem for each of them. This approximated problem considers only the behaviors of migration, obstacle avoidance, and control effort. The cohesion and inter-agent avoidance are excluded because they would require recursive knowledge on the neighbor set, while due to the local sensing agent i can not exhaustively infer the set of neighbors $\mathcal{N}_j$ of its neighbor j. This is equivalent to solving the problem where only agent j would be present in the environment.

7.2.6 PF swarm model

We implement Vasarhelyi's model as presented in [13] to which we add a cohesion term. The command to agent i, expressed as a velocity, is:

$$\boldsymbol{u}_i = \boldsymbol{v}_{\mathrm{flock},i} + \boldsymbol{v}_{\mathrm{rep},i} + \boldsymbol{v}_{\mathrm{fric},i} + \sum_{w\in\mathcal{W}_i} \boldsymbol{v}_{\mathrm{wall},im} + \sum_{m\in\mathcal{M}_i} \boldsymbol{v}_{\mathrm{obs},im} + \boldsymbol{v}_{\mathrm{coh},i} \tag{7.4}$$

where the terms are self-propulsion, to match a preferred velocity, repulsion, to avoid inter-agent collisions, our cohesion term, to maintain the agents close, friction, to reduce agent oscillations in dense configurations, repulsion from the boundaries of the environments (i.e., walls), and repulsion form obstacles. The cohesion formula is:

$$\boldsymbol{v}_{\mathrm{coh},ij} = \begin{cases} c_{\mathrm{coh}}(d_{ij} - d_{\mathrm{coh}}) \frac{\boldsymbol{p}_i - \boldsymbol{p}_j}{d_{ij}} & \text{if } d_{ij} > d_{\mathrm{ref}} \\ 0 & \text{otherwise} \end{cases} \tag{7.5}$$

where c_{coh} is the constant weight of the cohesion term, d_{coh} is the cohesion inter-agent distance, and d_{ij} is the scalar distance between agents i and j. The total repulsion term is given by the sum of individual terms:

$$\boldsymbol{v}_{\mathrm{coh},i} = \sum_{j\in\mathcal{N}_i} \boldsymbol{v}_{\mathrm{coh},ij} \tag{7.6}$$

Table 7.1 – **Swarm model parameters.** Description and values of the DMPC swarm model parameters used in our simulation experiments.

Parameter	Description	Unit	Value
dt	Control time step	s	0.2
$d\tau$	Simulation time step	s	0.2
T_{max}	Simulation time	s	30
T_P	Prediction horizon	s	3
l	Number of Bezier curves	–	3
d	Bezier curve order	–	5
T_l	Bezier curve duration	s	1
$\boldsymbol{p}_{\text{mig}}$	Migration point	m	$[14, 0, 1]$
$\boldsymbol{p}_{\text{min}}$	Minimum allowed position	m	$[0, -7, 0]$
$\boldsymbol{p}_{\text{max}}$	Maximum allowed position	m	$[0, 7, 2]$
$\boldsymbol{a}_{\text{min}}$	Minimum acceleration	m/s^2	$[-1, -1, -1]$
$\boldsymbol{a}_{\text{max}}$	Maximum acceleration	m/s^2	$[1, 1, 1]$
d_{coh}	Cohesion distance	m	2
r_{agent}	Agent radius	m	0.07
n	Number of nearest neighbors	–	5
E_{xx}	Distance scaling in x	–	1
E_{yy}	Distance scaling in y	–	1
E_{zz}	Distance scaling in z	–	0.5
r_{obs}	Cylindrical obstacle radius	m	0.35

7.2.7 Swarm performance metrics

We assess the performance of the swarm's flight according to six different metrics. The mission completion time T measures the time that the swarm requires to complete a mission. A mission is completed if the swarm average position reaches the migration point up to a tolerance distance d_{tol} and if, at the same time, all the drones are within the distance d_{coh} from their neighbors. The trajectory length L_{traj} measures the average of the agents' flown distances until they complete the mission. The minimum and the maximum inter-agent distances, $\min(d_{ij})$ and $\max(d_{ij})$, measure the minimum and the maximum distance among neighboring couples of drones over the mission. The minimum distance to the obstacles $\min(d_{im})$ measures the minimum distance between all agents and all obstacles. Finally, the order Φ_{order} measures the average correlation of the agents' directed movements. It is often used to quantify the synchronization of the agents' flight [13, 17], and in formula it is:

$$\Phi_{\text{order}} = \sum_{k \in \{1,\dots,K\}} \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{N}_i^k} \frac{\boldsymbol{v}_i(k)^T \cdot \boldsymbol{v}_j(k)}{KNn \|\boldsymbol{v}_i(k)\| \|\boldsymbol{v}_j(k)\|} \tag{7.7}$$

where $K = \min(\lceil T/dt \rceil, \lceil T_{\text{max}}/dt \rceil)$ and $\lceil \cdot \rceil$ is the ceiling function.

Additionally, for the DMPC swarm models we measure the replanning variance $\mathbb{V}p$ and for the sensor-based DMPC we measure the neighbor prediction error $\mathbb{E}p$. The first measures the discrepancy between consecutive planned trajectories and it is expressed by:

$$\begin{aligned}\mathbb{V}p = \frac{1}{NK} & \sum_{i\in\mathcal{V}} \sum_{k\in\{1,\dots,K\}} \sum_{\pi=\{1,\dots,P-1\}} \\ & [\boldsymbol{p}_i(k+\pi|k+1)^T \cdot \boldsymbol{p}_i(k+\pi|k+1)+ \\ & -\boldsymbol{p}_i(k+1+\pi|k)^T \cdot \boldsymbol{p}_i(k+1+\pi|k)]\end{aligned} \tag{7.8}$$

where $(\cdot)(k+\pi|k)$ represent the predicted value of $(\cdot)(k+\pi)$ with the information available at time step k. The second measures the squared difference between the neighbors approximated and exact predicted trajectories, and it is expressed by:

$$\begin{aligned}\mathbb{E}p = \frac{1}{NK} & \sum_{i\in\mathcal{V}} \sum_{k\in\{1,\dots,K\}} \sum_{\pi=\{1,\dots,P\}} \\ & [\hat{\boldsymbol{p}}_i^j(k+\pi|k)^T \cdot \hat{\boldsymbol{p}}_i^j(k+\pi|k)+ \\ & -\boldsymbol{p}_i(k+\pi|k)^T \cdot \boldsymbol{p}_i(k+\pi|k)]\end{aligned} \tag{7.9}$$

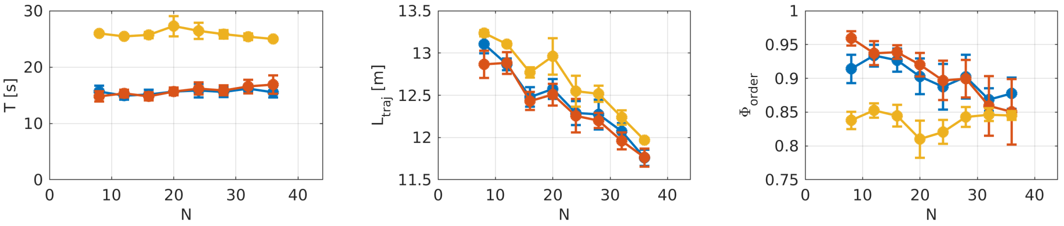
where $\hat{(\cdot)}_i^j$ represent the value of $(\cdot)_i$ of i predicted by j.

7.3 Results

7.3.1 Scalability in the swarm size

We run simulations for 8 swarm sizes $N \in \{8, 12, 16, 20, 24, 28, 32, 36\}$ in a forest-like environment with cylindrical obstacles. For every configuration, we average the performance of 10 random simulations of length $T_{\max} = 30\ s$ and we show the aggregated results in Fig. B.5.

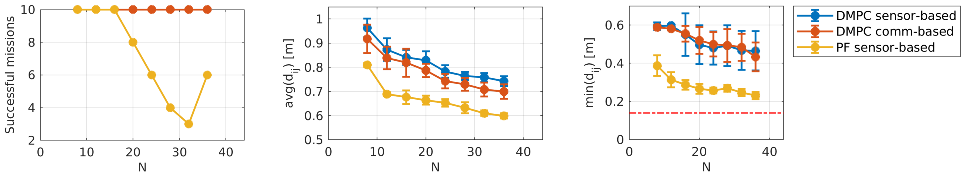
First, focusing the attention on the sensor-based DMPC swarm model, we can see that the mission completion time increases with the number of agents and, inversely, the trajectory length decreases ((Fig. 7.3b and 7.3a)). This implies that the average speed of the agents decreases when the swarm is more packed. While the agents look for a free path, they slow down to avoid collisions. Similarly, also the order decreases at higher swarm sizes. This phenomenon is due to changes in the swarm topology to fit the morphology of the environment. While some agents avoid the obstacle on the right, others fly to its left and the swarm topology varies. Once they pass the obstacle, they reunite and update their neighbor sets. Changes in the neighbor sets are frequently accompanied by variations in the flight direction and hence decreased order. Up to 36 agents, we observe zero collisions between agents and with obstacles (Fig. 7.3d), which means that both distances stay above the collision value (Fig. 7.3f). However, the average inter-agent distance between agents decreases when the swarm size increases (Fig. 7.3e).



(a) Mission completion time

(b) Trajectory length

(c) Order



(d) Number of missions without collisions

(e) Average inter-agent distance

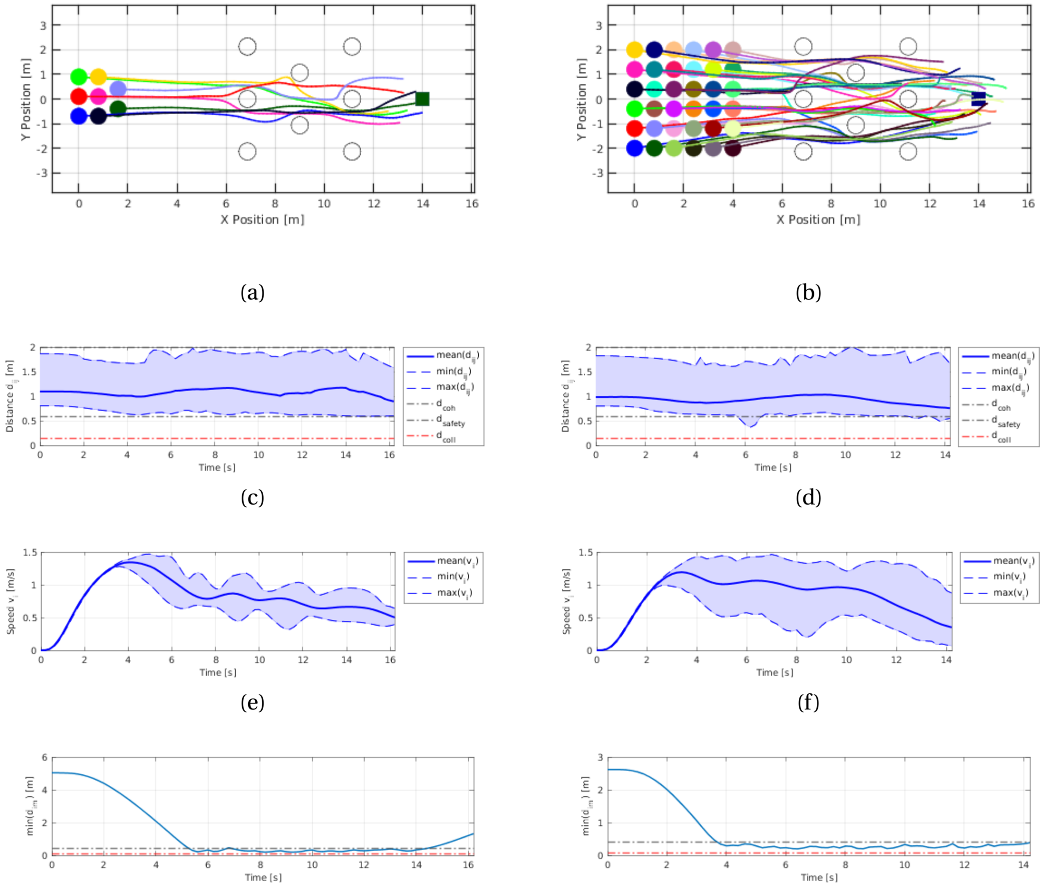
(f) Minimum inter-agent distance

Figure 7.3 – **Aggregated performance at different swarm sizes**. Aggregated simulation results on the swarm performance in a forest-like environment and at different swarm sizes (i.e., 8, 12, 16, 20, 24, 28, 32, and 36 agents). We compare three swarm models: communication-based DMPC (orange), sensor-based DMPC (blue), and sensor-based PF (yellow). For each configuration (i.e, swarm size and model), we run 10 random simulations. In the plots of the mission completion time (7.3a), trajectory length (7.3b), order (7.3c), average inter-agent distance (7.3e), and minimum inter-agent distance (7.3f) we report average and standard deviation of the same metrics. Additionally, we plot in dashed red the collision threshold. In 7.3d, we report the total number of missions without inter-agent nor obstacle collisions.

7.3.2 Comparison between swarm models

The parameters for the sensor-based PF model were optimized for the absence of collisions and the fastest mission time with a swarm size of 16 agents in the same environment.

When comparing the three swarm models (communication-based DMPC, sensor-based DMPC, and sensor-based PF), we observe that the communication-based DMPC model present the best performance for all the metrics considered. This result conforms to expectations since the communication-based model uses perfect information on the neighbors predicted trajectory and not an approximation based on local sensor information like the other two models. While the two DMPC models present simular results in terms of mission completion time, and trajectory length, the PF model presents visibly longer mission times and trajectories at all swarm sizes. The longer mission completion times of the PF model is due to the fact that the swarm tries to match the preferred speed and slows down around obstacles because of the negative artificial collision avoidance forces. Hence, the agents seldom fly at the maximum speed. Although the preferred speed can be increased within the maximum boundary, having a value too close to the maximum value leaves little room for positive adjustments that may be necessary for avoiding collisions. Longer agents' trajectories



(a) (b)

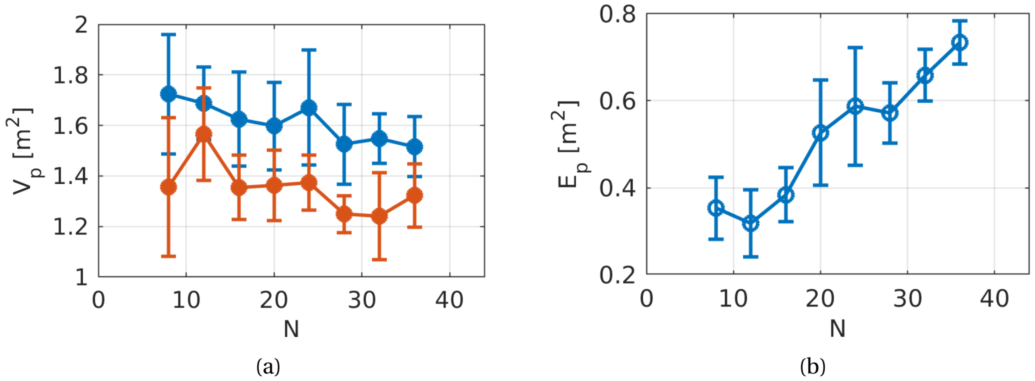
(c) (d)

(e) (f)

(g) $N = 8$ agents (h) $N = 36$ agents

Figure 7.4 – **Size scalability of the sensor-based DMPC swarm**. Simulation results on the swarm flight in a forest-like environment and at two different swarm sizes (i.e., 8 and 36 agents, from left to right). From top to bottom, top-view of the trajectories, inter-agent distance envelope, speed envelope, and closest distance to obstacles.

are due to the difference in the avoidance maneuvering. The agents of the DMPC models take less-conservative paths which almost touch the obstacles, while the PF model agents make more prominent turns which increase the total travelled distances. As for the order, differently from DMPC swarms, PF swarms have comparable performance at different swarm sizes. However, order performance is overall lower for PF swarms than for DMPC swarms. The DMPC models result in safe trajectories, sensor-based PF model presents collisions for swarms larger than 16 agents, the size for which the parameters were selected. Collisions happened mostly between agents and obstacles. Inter-agent distances are on average shorter for PF swarms than DMPC swarms. Although PF swarms try to match a reference distance $d_{\text{ref}} = 0.8\ m$, agents often move closer apart to avoid collisions with obstacles. It is important to notice that the inter-agent safety constraints of both the communication-based and the



(a)

(b)

Figure 7.5 – **Replanning variance and neighbor prediction error**. On the left, replanning variance, $\mathbb{V}p$. On the right, neighbor prediction error, $\mathbb{E}p$.

sensor-based DMPC models are often relaxed for swarm size larger than 12 agents. However, the standard deviation of the minimum inter-agent distance is larger for the sensor-based DMPC model due to the approximation of the neighbors predicted trajectories (Fig. 7.5).

7.4 Discussion

In this chapter, we described a DMPC swarm model that replaces explicit communication with local sensing and neighbors' trajectory prediction. We compared the flight performance of this model with the performance of the communication-based DMPC model and showed its drawbacks for the application to large drone swarms in cluttered environments. However, the results show that at all sizes the sensor-based DMPC model can provide zero-collision trajectories, differently from the PF approach. It is important to notice that, to avoid collisions, we set the minimum distance between any pair of agents to be larger than twice the maximum braking distance. This condition depends on the agents' maximum speed, acceleration, and the update time of the control loop. In particular, larger swarm's speeds determine longer braking distance, and hence the anti-collision conditions require larger minimum inter-agent distances. As a result, also the maximum allowed density of the swarm depends on these factors. If the swarm can travel fast, then the density allowing safe flight lowers. In other words, the swarm requires more free volume to avoid collisions.

In the future, we would like to extend the sensor-based DMPC model and consider strategies for emergency braking to avoid collisions at high swarm densities, when the constraints relaxation could potentially infringe the safety boundaries. This imply the use of adaptive parameters that adapt the swarm laws relative importance in different phases of the swarm flight. Besides, we hardware implementation will be a necessary step to validate the usage of this model for real-world swarm missions.

8 Conclusion

In this thesis, we investigated the conditions under which current state-of-the-art approaches to aerial swarm navigation break. In particular, we showed that the state-of-the-art swarm models based on artificial potential fields are unlikely to generate safe flight in cluttered environments, especially when the obstacle density varies along the way (Chap. 3 and 7). In this situations, empirical parameter adjustment is necessary although inconvenient for the quick deployment of aerial swarms. Moreover, we analyzed the used of potential field-based algorithms for the deployment of agents with limited field-of-view sensing (Chap. 5). From the results, we concluded that parameter instantiations adapted to the robots' sensor configuration can marginally improve specific performance metrics. However, omnidirectional sensors certainly present comparatively larger performance advantages.

To remove the difficulties specific to these state-of-art models, we proposed novel algorithms that can safely steer swarms of flying robots in cluttered environments of the real world. These algorithms are based on the prediction of the future state of the agents' neighbors. In particular, we showed that a centralized NMPC swarm model improves the swarm's speed when flying in the presence of obstacles. Instead of slowing down when approaching obstacles, the agents can track the preferred speed better and end their mission faster. Due to the anticipatory behavior of the agents, their synchronization is better than for potential field-based methods. As a result, the swarm is safer and can fly in a range of different environment without requiring parameter adjustments for each of them. Finally, the presented approach is robust to changes in the swarm inter-agent distance and speed, up to a certain extent (Chap. 3). This property increases the versatility of the swarm, giving it the ability to fly coarsely over large areas for terrain mapping or fly more compactly in confined spaces for indoor exploration. We also showed that a distributed version of the MPC swarm model maintains these advantages with the additional benefit of being scalable in the agent number (Chap. 4). Then, we proposed and analyzed the use of the DMPC swarm model for agents that rely on sensors and cannot communicate information with their neighbors (Chap. 7). Reducing the amount of available information necessarily results in a poorer performance. However, although the swarm is less synchronized, collisions can still be avoided for all the considered swarm sizes. Instead, PF-based models did not generate a safe flight for all the sizes.

In the following, we address the limitations of the predictive approaches presented in this thesis (Sec. 8.1) and discuss possible directions for future work (Sec. 8.2).

8.1 Limitations of the predictive control approach

In general, centralized optimization problem has the advantage of presenting better convergence properties than their distributed counterpart. The optimality of the solution to the problem including the full state system information is equal or better than the solution to the problem having only partial information. Furthermore, the setup and update of the system is easier and less time consuming. However, centralized formulations like the one in Chap. 3 are not scalable in the swarm size. This means that despite an increase in the hardware and software capabilities of the central node and communication bandwidth, the number of agents supported will not increase appreciably. On the other side, we showed that a decentralized algorithm can be scalable in the agent number while still providing similar qualitative flight properties as the centralized counterpart.

A limitation of our work regards the simplification assumptions that we made on the environment. While the forest and the funnel-like environments represent two common use cases, they do not cover all environmental configurations that a swarm may encounter while flying in urban or natural real-world environments. The latter may present obstacles of different sizes and concave shapes. To fly around concave obstacles without freezing into local minima, strategies such as topological planning [42, 119, 120] could replace our obstacle avoidance strategy, which is purely based on the closest distance to an obstacle and disregards the obstacle shape. In addition, some missions may involve more complex navigation tasks than going straight to a goal destination, as we assume in our work. For example, swarms may have to navigate maze-like environments like the complex road network surrounded by high buildings [121]. A higher-level planning strategy could compute short-term position goals in these scenarios, and a predictive controller could steer the swarm to those goals. Hence, it would be the high-level planner's responsibility to resolve conflicting situations such as dead-ends or keep a record of already explored paths.

Our systematic experiments on the DMPC swarm model concluded that our algorithm is robust to positional sensor noise. However, soft constraints on collision avoidance can be violated in favor of lower control effort or when a solution can not be otherwise found. Hence, a theoretical analysis of convergence would help understand the situations in which collisions may occur and define operating conditions under which collision avoidance is theoretically guaranteed [38, 122, 123].

Although sensor-based MPC must be envisaged in conditions when communication is not available or possible, communication-based MPC provides better performance and should hence be preferred when possible. Communication can be combined with state estimation from other sensor measurements to reduce the downsides of communication, such as delays or package drops. The combination of the two methods can add resilience against faults.

8.2 Possible directions for future work

This thesis has covered the case of collective navigation in environments with static obstacles. The extension of the presented techniques to dynamic obstacles would allow more flexibility to swarm applications. Indeed, most of the environments include moving obstacles, and some of them require special care not to be hit or injured by flying vehicles. Moving obstacles can not be treated like other swarm members because they presumably do not follow the same behavioral rules. However, their state, i.e., momentary position and velocity, can be estimated with the same techniques. Since we first started our work, many studies have been published on precise pose estimation of flying vehicles [118, 117, 110]. Therefore, we would find interesting to extend this research to swarm navigation in the presence of dynamic obstacles.

All the models developed in this thesis assume that all the swarm agents have the same morphology and sensors, or in other words, that the swarm is homogeneous. To extend their applicability, future work could consider heterogeneous swarms [124, 125]. The extension of predictive models to heterogeneous swarms implies specific considerations on the swarm safety when different agents have different sizes, dynamics, and physical limitations. Intuitively, an agent with higher inertia and requiring more time to brake should fly more cautiously than a highly agile agent to avoid collisions in dangerous scenarios. Since collaborative swarm algorithms assume that individuals of the same swarm share the responsibility for collision avoidance, this consideration is not only individual. Instead, agents of heterogeneous swarms can either exchange extra information on their dynamics or infer it from the observation of their flight.

Touching on another aspect, the presented swarm models have constant parameters. However, in certain situations, adaptive parameters could increase the safety and flexibility of the swarm [126]. For example, while flying in the presence of humans, safety may be preferred over time efficiency. In contrast, time optimality may be privileged over energy consumption and safety once past the inhabited region. Adapting the swarm parameters in function of the mission and environment may be convenient on these occasions. The task of parameter selection could be granted to a high-level classifier algorithm, deciding on the context-specific needs of the swarm.

All the presented predictive algorithms have been implemented on a ground computer, including the distributed algorithm. Future work should concentrate on embedding the computation on the drone controllers [14, 127, 128, 129]. Given the limited computational power of the selected platform (i.e., Crazyflie 2.1), we expect a direct implementation of the proposed algorithms on the drone microcontroller to be hardly achievable. For this reason, different and more complete hardware solutions should be explored. An alternative approach could involve the simplification of the algorithm to get a computationally lighter solution. Works in these two directions are presented in [42] and [14].

Finally, variants of the presented algorithms could enable a wider range of applications. While we consider drone swarms flying to the same destination, most applications such as last-centimeter delivery, crop monitoring, surveillance, etc., would benefit from assigning different intermediate goals to each agent to distribute packages or maximize an area coverage. In these tasks, the agents' density would be reduced as compared to cohesive swarm flight although trajectory crossing is still possible and require specific attention. Transitions from this state to cohesive flight may happen for homing purposes once that the individual missions are completed. Another application where cohesive and individual flight may coexist is when flying in densely packed areas. Recent work [130] has shown that in those situations, lining improves the travel time, similarly to how the road traffic is managed [126, 131]. While cars have individual destinations, they mainly travel in convoys and follow other vehicles' flow.

8.3 Closing remarks

This thesis would not have been possible without relying on third-party software tools and libraries, and the importance of free software in research cannot be understated. Notable examples include the crazyswarm software [82], the Robot Operating System (ROS) [132], the Gazebo physics simulator [133], the RotorS drone simulation framework [134], the PX4 open-source drone autopilot [135], the CrazyS extension for simulating Crazyflies [136], the MRS UAV system [137], and many others.

This work represents a step towards the future that have depicted in the prelude to Chap. 1. Although an infinity of applications of this work are possible, we invite the readers of this document to think critically about their pertinence and consequences for our society.

A Waypoint navigation of vision-based drone swarms

Vision-based drone swarms have recently emerged as a biologically plausible and scalable alternative to swarms that rely on wireless communication for motion coordination. Instead of sharing absolute positions wirelessly, vision-based swarms rely on visual perception to infer the relative positions of neighboring drones and transform them into motion commands. Here, we show that a simple swarm model that operates directly on a segmentation of the visual field of view can provide collision-free and goal-directed flight without estimating relative positions. We extensively study this approach in simulation for goal-oriented navigation missions. We compare the results with a position-based swarm model and we validate our approach with a swarm of four drones flying in a controlled indoor environment.

The work presented in this chapter is adapted from []:

- E. Soria, H. Birch, F. Schilling, D. Floreano, "Waypoint Navigation of Vision-based Aerial Swarms without Estimation of Relative Positions," in *IEEE International Conference on Robotics and Automation (ICRA)*, (under review).

A.1 Introduction

Reynolds rules are at the basis of many robotics implementations [23, 12, 13]. However, in these works, drones communicate their positions to their neighbors.

Researchers have attempted to relax the dependence on communication between agents and to instead rely entirely on visual inputs for real-world control of swarms. Visual sensors are ideal for the deployment of small, autonomous drones since they can passively extract a large amount of information from a scene, all while being low-cost, lightweight, and power-efficient. In recent years, vision-based swarms have therefore become an active area of research. Most works are based on extracting relevant state variables such as relative positions from visual inputs, which are then used for control [115, 97]. However, the interpretation of the image data and the extraction of useful information, such as the relative range and bearing to other robots,

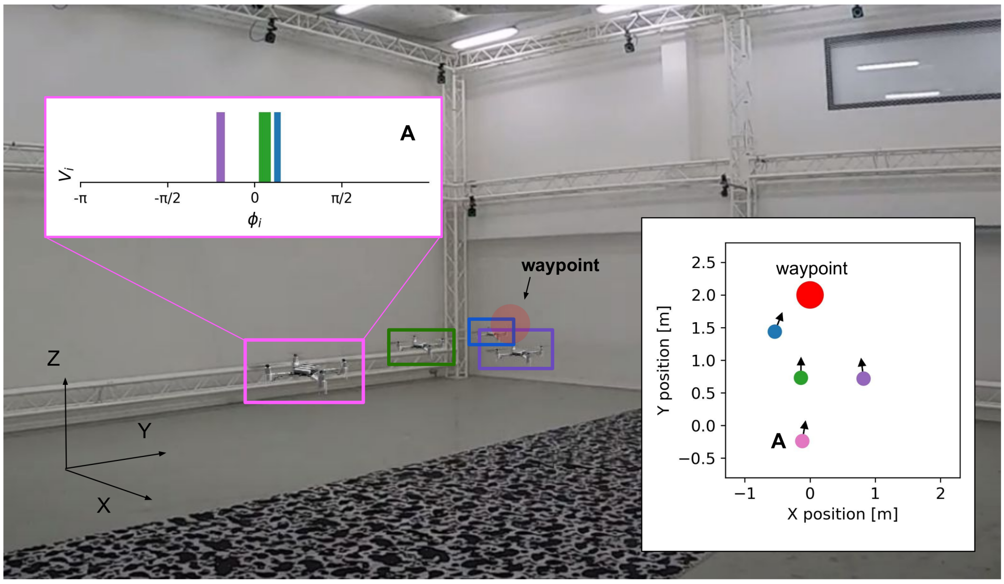


Figure A.1 – **Vision-based swarm experiment in our motion tracking hall.** The behavior of each agent depends on the projection of its visual FOV and the waypoint position (in red). The visual FOVs are simulated on the ground control station by processing the agents' positions and velocities. For convenience, we highlight in pink agent A and its FOV.

is highly challenging. Some researchers have proposed to ease this task by equipping the drones with easily detectable visual markers [110, 95, 112]. This approach requires the use of specialized hardware. Moreover, the complexity of multi-agent state tracking depends on the size and density of the swarm, hence motivating the exploration of alternative vision-based approaches that avoid this task.

Bastien et al. recently demonstrated that collective motion can be originated with a model-based approach operating directly on the visual Field of View (FOV) [109]. Differently from other swarm models [21, 22, 10, 13, 12], their model does not require the agents to compute their neighbors' relative positions. Depending on the parameter values, this model can generate polarized movements, rotating configurations, or disordered motion around an area. However, to be suitable for robotics swarm implementations, this model should consider the limited actuation of the robots and account for the navigation towards a preferred direction. Most importantly, only parameter configurations producing collision-free movements are viable.

In this chapter, we extend Bastien's method to produce goal-directed flight in vision-based aerial swarms without estimation of relative positions (Fig. A.1). We add realistic limitations of drones by bounding the linear and angular components of the computed motion commands. Then, we add a migration rule that orients the agents' motion towards known waypoints. We optimize the parameters of the vision-based swarm model for collision-free waypoint navigation with the use of a particle swarm optimization algorithm. Then, we extensively analyze the flight performance in simulation and we compare it to a state-of-the-art method based on the visual inference of the neighbors' relative positions. This comparison allows us to highlight and quantify the differences between swarms that operate directly on the visual FOV and swarms that rely on the computation of intermediate state variables. To assess the validity of the proposed method for real-world swarm deployment, we perform hardware experiments with a swarm of 4 drones in an indoor facility. The results confirm that the swarm is capable of collision-free and cohesive flight while transitioning between common target destinations.

A.2 Method

In this work, we consider N identical drones of radius R labeled by $i \in \mathcal{V} = \{1, 2, 3, \dots, N\}$. To simplify the description, we consider two-dimensional flight in a horizontal plane, which is also the case for many real-world applications like monitoring and area coverage. The position, velocity and acceleration of agent i are denoted by $\boldsymbol{p}_i$, $\boldsymbol{v}_i$, and $\boldsymbol{a}_i$, respectively, while the heading and the angular speed are indicated with $\psi_i \in [-\pi, \pi]$ and ω_i, respectively. The agents' state variables are discretized with a constant time step dt, and k denotes the index of the time step.

In the following, we introduce the modeling of the visual FOV and the equations of motion for two swarm models: (a) the swarm model operating directly on the FOV, and (b) the swarm model using relative positions. For convenience, we call the former vision-based model and the latter position-based model. Both models use visual inputs and produce the same swarm behaviors: cohesion and repulsion, resulting from a virtual force component based on vision, and migration, resulting from a virtual force component based on individual factors.

A.2.1 The visual field of view

We consider that each agent is endowed with a visual sensor centered w.r.t. the x-axis of the body-frame, and we assume that the robot's heading is aligned with the velocity vector (Fig. A.2, top left). To model the visual configurations, we define the *FOV width* $\phi^{\max}$ as half of the angular span that the sensor can sense. The visual angle ϕ_i spans $[-\phi^{\max}, \phi^{\max}]$ and $\phi_i = 0$ coincides with the heading angle ψ_i. The FOV width can range from 0 (absence of vision) to π (omidirectional vision). For this work, we consider omnidirectional vision. In our implementation, we discretize the visual angle ϕ_i with a constant discrete step $d\phi_i$.

$V_i(\psi_i, k)$ is a scalar function that represents the projection of the visual FOV experienced by agent i at time k in one dimension, and ψ_i is the swiping angle of this field. As hinted by [109], in order to produce collective motion a minimalist swarm model based on visual interactions only accounts for the presence or absence of agents and not their identity. Hence, $V_i(\psi_i, k)$ takes values in $\{0, 1\}$ (Fig. A.2, top right). The neighborhood of an agent i at time step k, indicated with $\mathcal{N}_i(k)$, is the set of agents which are not visually occluded w.r.t. i. Only the neighbors $j \in \mathcal{N}_i(k)$ influence the movements of i. For clarity, in the following, we will omit the dependency on time k when clear from the context.

A.2.2 Vision-based swarm model

The equations of the swarm model that operates directly on the visual FOV are inspired by [109]. According to this model, the motion of an agent i is defined by the sum of two forces, a visual force $\boldsymbol{F}_i^{\text{vis}}$, that is the reaction of an agent to its FOV, and an individual force $\boldsymbol{F}_i^{\text{ind}}$, that depends on individual factors. It holds:

$$\widetilde{\boldsymbol{a}}_i = \boldsymbol{F}_i^{\text{vis}}(V_i) + \boldsymbol{F}_i^{\text{ind}}(\boldsymbol{v}_i) \tag{A.1}$$

Both forces are represented by a linear and an angular component, separately determining the linear and angular components of the agent input command $\widetilde{a}_i$ and $\widetilde{\omega}_i$.

The visual force accounts for the agents' response to the perceived neighbors and produces a short-range repulsion and a long-range attraction. Its components are:

$$F_{v,i}^{\text{vis}} = \alpha_v \int_{-\pi}^{\pi} (-V_i(\phi_i) + \beta_v (\partial_{\phi_i} V_i(\phi_i))^2) \cos(\phi_i) d\phi_i \tag{A.2}$$

$$F_{\psi,i}^{\text{vis}} = \alpha_\psi \int_{-\pi}^{\pi} (-V_i(\phi_i) + \beta_\psi (\partial_{\phi_i} V_i(\phi_i))^2) \sin(\phi_i) d\phi_i \tag{A.3}$$

These formulas indicate that the agents react to the angular area and edges of the objects in their visual projection, with strength given by the coefficients α_v and β_v for the linear component and α_ψ and β_ψ for the angular component. A representation of the linear component of the visual force is shown in Fig. A.2, at the bottom right. The parameters β_v and β_ψ determine the front-back and left-right equilibrium distance, respectively, at which two agents would converge if no other force was acting. For symmetry, we choose $\beta_v = \beta_\psi$, and the equilibrium distance d_0 can therefore be computed as $d_0 = R/\beta_v = R/\beta_\psi$.

The individual force defines the so-called migration behavior of the agents. Differently from [109], we encode a preferred heading ψ_0 and a preferred agents' speed v_0. The parameter ψ_0 orients the motion of the swarm in a preferred direction and it is useful for applications like collective exploration, where the destination to be visited is known by the swarm. It holds:

$$F_{v,i}^{\text{ind}} = \gamma_v (v_0 - v_i) \tag{A.4}$$

$$F_{\psi,i}^{\text{ind}} = \gamma_\psi (\psi_0 - \psi_i) \tag{A.5}$$

where γ_v and γ_ψ are constant weights for the linear and angular components of the individual force.

Goal-oriented flight is obtained by computing the preferred flight direction of an agent i, $\psi_{0,i}$, from the coordinates of a waypoint $\boldsymbol{p}^{\text{wp}} = (x^{\text{wp}}, y^{\text{wp}})$ as:

$$\psi_{0,i} = \operatorname{atan}\left(\frac{y^{\text{wp}} - y_i}{x^{\text{wp}} - x_i}\right) \tag{A.6}$$

This strategy allows the swarm to visit a sequence of goal destinations $\boldsymbol{p}_m^{\text{wp}} \in \{\boldsymbol{p}_1^{\text{wp}}, \boldsymbol{p}_2^{\text{wp}}, \dots, \boldsymbol{p}_M^{\text{wp}}\}$

and we say that one waypoint is reached when the center of the swarm attains it up to a tolerance distance d^{tol}.

Since the agents react to the projected angular areas in their FOV, their behavior depends on the inter-agent distances with their neighbors and their size. To remove the dependency of the swarm behavior on the latter, we can divide the parameters v_0 and α_v and the agents' initial positions by the characteristic length of the agent R (i.e., $v := v_0/R$ and $\chi := \alpha_v/R$).

To account for the limited actuation of the robots, we bound the linear and angular components by $a^{\max}$ and $\omega^{\max}$:

$$a_i = \operatorname{sign}(\widetilde{a}_i)\,\max\left(|\widetilde{a}_i|, a^{\max}\right) \tag{A.7}$$

$$\omega_i = \operatorname{sign}(\widetilde{\omega}_i)\,\max\left(|\widetilde{\omega}_i|, \omega^{\max}\right) \tag{A.8}$$

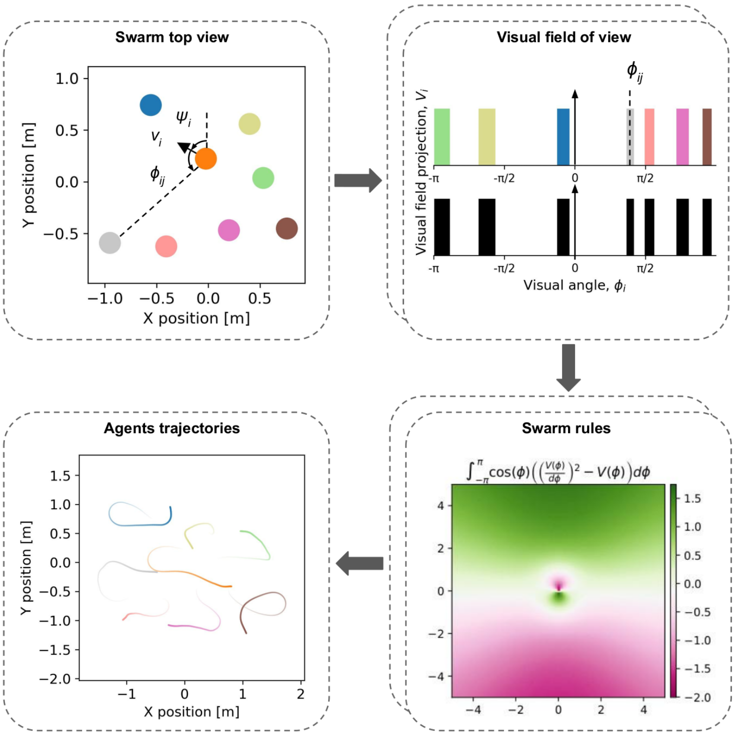
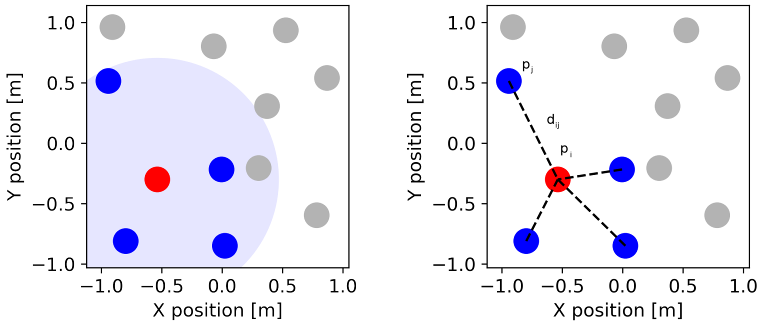


Figure A.2 – **Schematic representation of the vision-based swarm method that operates directly on the visual FOV.** All robots are endowed with an omnidirectional visual sensor. The 1D projection of the agents' visual FOV is used to compute their motion commands through a set of equations that accounts for both visual and individual factors. As a whole, the agents' motion commands result in cohesive and collision-free collective flight.



(a) Visible neighbors

(b) Relative positions

Figure A.3 – **Neighbors detection and estimation of the relative positions.** In the swarm model using intermediate state variables, each agent i detects the visible neighbors (dark blue) within the area defined by r (light blue). These are the neighbors for which it will compute relative positions. Notice that occluded neighbors within the visible area are excluded from the neighbor set.

A.2.3 Position-based swarm model

We present here an alternative model that uses vision to compute intermediate state variables, i.e. relative positions to neighboring agents, instead of operating directly on the visual FOV. This model is inspired by [22]. However, for the position-based model, we define the *FOV range* r as the maximum distance for which a robot can detect another robot. We assume that agent i can detect a neighbor j and estimate its relative position only if j lies within a distance r from i (i.e., $d_{ij} = \|\boldsymbol{p}_j - \boldsymbol{p}_i\| < r$) and j is entirely visible from i. Hence, in this model partially occluded agents are excluded from $\mathcal{N}_i$ (Fig. A.3).

The visual component regulates the inter-agent distances to an equilibrium value d_0 based on the detected neighbors' positions. It is defined by an artificial potential field U_i^{vis} as $\boldsymbol{F}_i^{\text{vis}} = \nabla U_i^{\text{vis}}$. The formula for U_i^{vis} is:

$$U_i^{\text{vis}} = \frac{1}{|\mathcal{N}_i(k)|} \sum_{j \in \mathcal{N}_i} \rho(d_{ij}/r)\sigma(d_{ij} - d_0) \tag{A.9}$$

where $\rho(\cdot)$ is a weight function defining the influence of neighbor j on i, while $\sigma(\cdot)$ is a scalar function defining the intensity of cohesion or repulsion to neighbor j, depending on whether the two agents are closer or farther than d_0. Their definitions are:

$$\rho(z) = \begin{cases} 1, & z \in [0, \delta] \\ 1/2^2[1 + \cos\left(\pi \frac{(z-\delta)}{(1-\delta)}\right)]^2, & z \in [\delta, 1] \\ 0, & \text{otherwise} \end{cases} \tag{A.10}$$

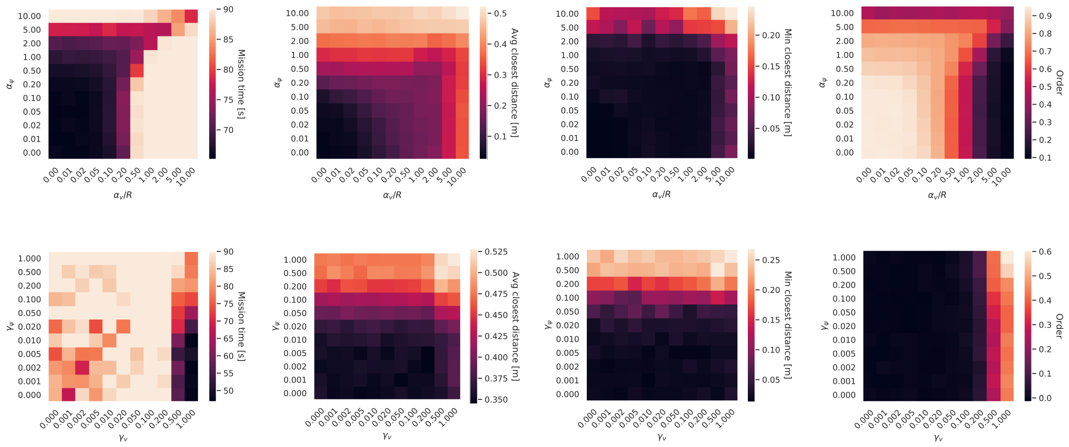


Figure A.4 – **Performance of the vision-based swarm model.** On top, results for different visual parameters. At the bottom, results for different individual parameters. From left to right, we show: mission time T, average of the closest inter-agent distances $\overline{d}$, minimum inter-agent distance d^{min}, and swarm's order Φ^{order}. Each performance value is the average of 10 random simulations.

$$\sigma(z) \quad = \quad (a+b)/2\left[\sqrt{1+(z+c)^2}-\sqrt{1+c^2}\right] \quad + \quad ((a-b)z)/2 \tag{A.11}$$

where the constant parameters δ defines the the weighting function $\rho(\cdot)$, while a, b, and c define the shape of $\sigma(\cdot)$.

The individual force $\boldsymbol{F}_i^{\text{ind}}$ steers the agents towards a waypoint $\boldsymbol{p}^{\text{wp}}$ with preferred speed v_0:

$$\boldsymbol{F}_i^{\text{ind}}(\boldsymbol{v}_i) = \gamma\left(\|\boldsymbol{p}_{\text{wp}}-\boldsymbol{p}_i\|\right)\left(v_0\mathbf{u}_i^{\text{wp}}-\boldsymbol{v}_i\right) \tag{A.12}$$

where $\gamma(\cdot)$ is a function weighting the individual force, i.e. $\gamma(z) = \min(1, z/d^{\text{tol}})$, and $\mathbf{u}_i^{\text{wp}} = (\boldsymbol{p}_{\text{wp}}-\boldsymbol{p}_i)/\|\boldsymbol{p}_{\text{wp}}-\boldsymbol{p}_i\|$ is the unit vector directed from agent i to waypoint $\boldsymbol{p}^{\text{wp}}$. As before, we bound the acceleration commands (Eq. A.7).

A.2.4 Swarm performance metrics

We assess the performance of the swarm's flight with five metrics. The mission completion time T measures the time that the swarm requires to reach the final destination $\boldsymbol{p}_M^{\text{wp}}$. We consider a mission completed if the swarm reaches the last waypoint before the experiment end time T^{max}. The average trajectory length L^{traj} measures the agents' average flown distances until the mission completion. The average and minimum closest inter-agent distance $\overline{d}$ and d_{min} measure the average and minimum of the closest inter-agent distance of all agents over time,

Parameter	Description	Unit	Value
dt	Discretized time step	s	0.1
$d\phi$	Discretized visual angle	–	0.00613
N	Number of agents	–	16
R	Agent radius	m	0.1
v_0	Preferred speed	m/s	0.2
$a^{\max}$	Maximum acceleration	m/s^2	0.75
$\omega^{\max}$	Maximum angular speed	$1/s$	$\pi/3$
α_v	Linear visual coeff.	$1/s^2$	0.64
α_ψ	Angular visual coeff.	$1/s^2$	8.37
β_v	Linear visual coeff.	–	1/12.5
β_ψ	Angular visual coeff.	–	1/12.5
γ_v	Linear drag	$1/s$	0.83
γ_ψ	Angular drag	$1/s$	0.37
d^{tol}	Tolerance distance	m	0.5

Table A.1 – **Optimal parameters for the vision-based swarm model.** Parameters description and values for the swarm model operating directly on the visual FOV. The values are obtained from an optimization process that minimizes two metrics, numbers of collisions and mission completion time.

respectively. It holds:

$$\overline{d} = \frac{1}{K}\sum_k \min_{ij}\left(d_{ij}(k)\right) \tag{A.13}$$

$$d^{\min} = \min_{ijk}\left(d_{ij}(k)\right) \tag{A.14}$$

where $K = \min(\lceil T/dt \rceil, \lceil T^{\max}/dt \rceil)$ is the number of time steps of an experiment and $\lceil\cdot\rceil$ is the ceiling function. The average order Φ^{order} measures the average correlation of the agents' directed movements and it is used to quantify the synchronization of the agents flight [13, 17]. It holds:

$$\Phi^{\text{order}} = \sum_{k\in\{1,\ldots,K\}} \sum_{i\in\mathcal{V}} \sum_{j\in\mathcal{N}_i(k)} \frac{\boldsymbol{v}_i(k)\cdot\boldsymbol{v}_j(k)}{KN|\mathcal{N}_i(k)|\|\boldsymbol{v}_i(k)\|\|\boldsymbol{v}_j(k)\|} \tag{A.15}$$

.

A.3 Results

To study the flight performance of the vision-based model, we define a mission where the swarm visits four waypoints located at the corners of a square. Parameters associated with the agents' radius, the equilibrium inter-agent distance, and the preferred speed are chosen to fit a plausible deployment scenario for a swarm of mini-drones (i.e., $R = 0.1\ m$, $d_0 = 1.25\ m$, and $v_0 = 0.2\ m/s$), while the remaining ones are optimized with a particle swarm optimization algorithm over a continuous parameter space (Table A.1). The optimization is aimed at minimizing the mission time and the number of collisions between agents. The cost of the populations in our particle swarm algorithm was determined by a 90 s simulation of the system. The parameter set retained was the best of three independent runs. In all runs, we used a population size of 40 and terminated after 40 generations. The initial positions of the agents are randomly sampled from a uniform distribution over a squared region with the additional constraint of being non-colliding (i.e., $d_{ij} > 2R$).

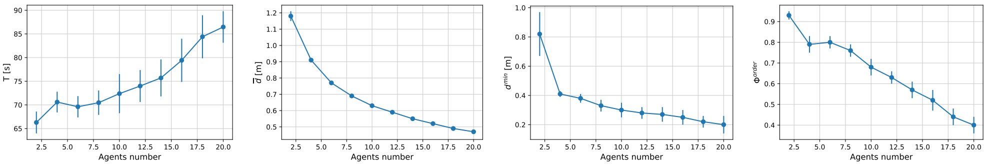


Figure A.5 – **Scalability of the vision-based swarm.** Performance results of the vision-based swarm for different swarm sizes. For each swarm size, we plot the average and standard deviation of 20 random simulations.

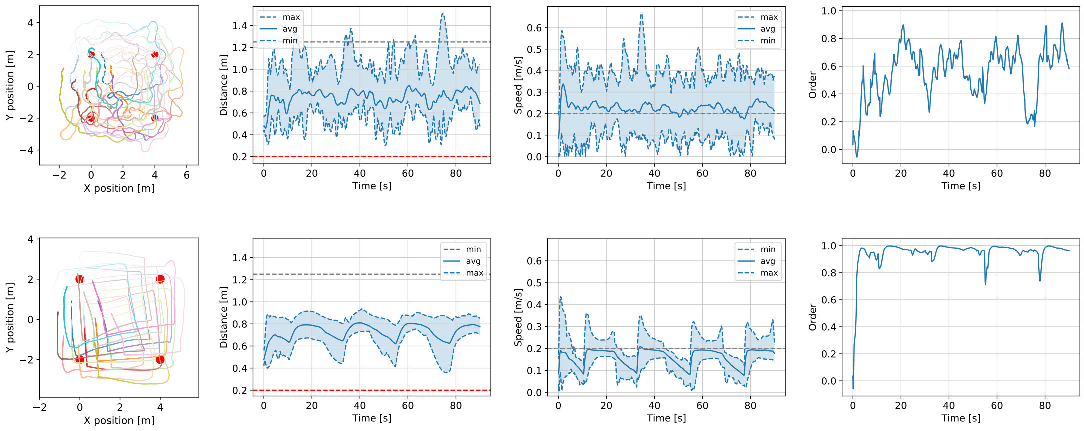


Figure A.6 – **Goal-oriented flight of the two swarm models.** On top, vision-based swarm model. At the bottom, position-based swarm model. From left to right, trajectories of 16 agents waypoints positions in red, envelope of the closest inter-agent distance (i.e., minimum, average, and maximum) with d_0 in gray and collision distance $2R$ in red, speed envelope, and swarm order. The experiments last for $T^{\max} = 90$ s. The trajectories of the agents fade over time.

A.3.1 Vision-based swarm model: the effect of individual and visual parameters

To analyze the effect of single parameters on the swarm's behavior, we run stochastic simulations for varying values of the visual (α_v, α_ψ) and individual (γ_v, γ_ψ) parameters, separately. While we vary visual parameters, individual parameters are set to their optimal values and vice versa. We summarize the results of the performance metrics in Fig. A.4.

Only small values of the visual parameters, α_v and α_ψ, make the swarm complete the mission. However, the inter-agent distance decreases and the number of collisions rises. This effect can be interpreted as a decrease in the agents' reaction strength to their neighbors. On the contrary, high values of the visual parameters allow the swarm to avoid collisions, and this effect holds for both the linear and angular components α_v and α_ψ. We conclude that small values of the visual parameters lead to an ordered swarm that flies straight to the waypoints, while high values of the visual parameters (and especially α_v) lead to a more disordered flight.

High values of the linear drag (i.e., $\gamma_v = 0.5$ or 1) lead to the completion of the mission and improved order. The inter-agent distance seems to be independent of the linear drag. Instead, the angular drag reveals a strong connection with the inter-agent distance. The higher is the angular drag, the higher the minimum and average inter-agent distance are.

A.3.2 Vision-based swarm model: goal-oriented flight

To analyze the scalability of the vision-based model with the number of agents, we run 20 random simulations for increasing swarm sizes N, from 2 to 20. Aggregated results are reported in Fig.A.5. While the mission time increases with the swarm size, the average and minimum closest inter-agent distances decrease dramatically with an increase in the number of agents. Specifically, with two agents, the average closest distance approaches the reference distance value ($\overline{d} \approx d_0$ for $N = 2$), while for larger swarms its value drops ($\overline{d} = 0.48\ m$ for $N = 20$). Finally, small swarm sizes present a well ordered flight ($\Phi^{\text{order}} \approx 0.9$ for $N = 2$), while for larger swarms the order decreases with a linear trend.

We present here the simulation results of a goal-oriented flight with 16 agents for the vision-based swarm model (see Fig. A.6, top). The agents' positions are initialized about the origin, then the swarm flies through the four waypoints clockwise. While the closest inter-agent distances vary significantly over time, the average stays about 0.8 m and the minimum does not cross the safety value. Hence, we register zero collisions. At the simulation start ($t = 0\ s$), the agents' speeds rapidly raise and the average converges to the preferred value v_0. The fluctuations account for the visual inputs to keep cohesion and avoid collisions. We can notice four major drops in the order. The first, at the beginning of the simulation, corresponds to the initial transient where the agents try to match the preferred inter-agent distance and speed values. The other three drops correspond to the moments when the swarm change of goal destination. The swarm completes the mission time in $T = 74.6\ s$ and the average trajectory length is $L^{\text{traj}} = 16.95\ m$.

A.3.3 Swarm models comparison

In the vision-based model, agents avoid collisions by turning, which causes a continuous reorganization in their topology and low overall order (Fig. A.6, top). However, these continuous movements allow a smoother transition between waypoints and do not necessitate a slowdown. Instead, in the position-based model, the agents translate in a rigid structure (Fig. A.6, bottom). Their trajectories are shorter, but the swarm slows down to reach the

Metric	Unit	Vision-based model	Position-based model
T	s	79.42 ± 4.56	77.08 ± 0.59
L^{traj}	m	17.34 ± 0.81	12.25 ± 0.08
Φ^{order}	–	0.52 ± 0.05	0.95 ± 0.00
$\overline{d}$	m	0.52 ± 0.01	0.60 ± 0.01
d^{min}	m	0.25 ± 0.04	0.31 ± 0.09

Table A.2 – **Comparison of the swarm performance with the two models.** Aggregated performance results (average and standard deviations) for a swarm of 16 agents and 20 random simulations.

waypoint position, hence completing the mission in comparable times. Both models guarantee collision avoidance (i.e, $d^{\text{min}} > 2R$), although the position-based model achieves higher minimum distances. As expected, the knowledge of relative positions helps to stabilize the minimum inter-agent distance. Aggregate results for simulations with 16 drones are presented in Table A.2.

A.3.4 Hardware experiments

For the hardware validation, we flew a swarm of 4 Crazyflie 2.1 drones in our indoor flying arena. The room is equipped with a motion capture system to detect the drones' positions (Fig. A.1). The vision was simulated by our ground control station, which computed the visual inputs by processing the agents' positions and estimated velocities. The agents' commands were also computed by the ground station online, and broadcast to each agent as position commands in Cartesian coordinates. The model parameters are the same as in the simulation experiments. Although the speed and order oscillate more in hardware than in simulation experiments with the same configuration, the swarm achieved cohesive and collision-free flight (Fig. A.7).

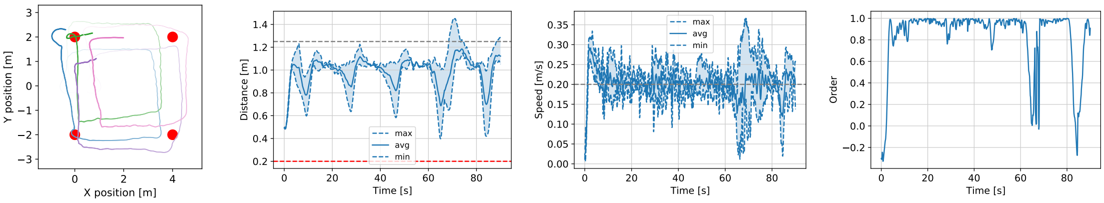


Figure A.7 – **Hardware experiment of the vision-based swarm with four drones.** On top, top view of the drones' trajectories and distance envelope. At the bottom, speed envelope and order.

A.4 Discussion

In this chapter, we showed that a swarm model that operates directly on the visual field of view can enable collision-free, goal-directed flight of drone swarms. We provided extensive simulation results and comparison with a more traditional approach that uses relative positions to compute the motion commands.

The results show that the agents do not need knowledge on relative positions to avoid collisions and navigate collectively through waypoints. Interestingly, when acting upon the visual field directly, the agents move in a less ordered manner but complete the mission in a comparable time. This approach presents a baseline for the development of autonomous swarms of drones with limited computation capabilities. The only requirement is the ability to segment drones in the images of the visual sensor. Future work should focus on acquiring and segmenting images onboard to make the drones independent of the ground computer. A proper quantification of the robustness against noise in the visual segmentation is necessary prior to further hardware developments.

B Open-source software

This thesis has lead to the development of several software packages. Among these, one was useful for the initial investigation on the different swarm models and has provided a code base for the software practicals of the Aerial Robotics course (Spring 2021). We describe its structure and functioning here. SwarmLab is a software entirely written in MATLAB*, that aims at the creation of standardized processes and metrics to quantify the performance and robustness of swarm algorithms, and in particular, it focuses on drones. We showcase the functionalities of SwarmLab by comparing two decentralized algorithms from the state of the art for the navigation of aerial swarms in cluttered environments, Olfati-Saber's and Vasarhelyi's. We believe that SwarmLab is relevant for the robotics research community and for education, since it allows fast algorithm development, the automatic collection of simulated data, the systematic analysis of swarming behaviors with performance metrics inherited from the state of the art.*

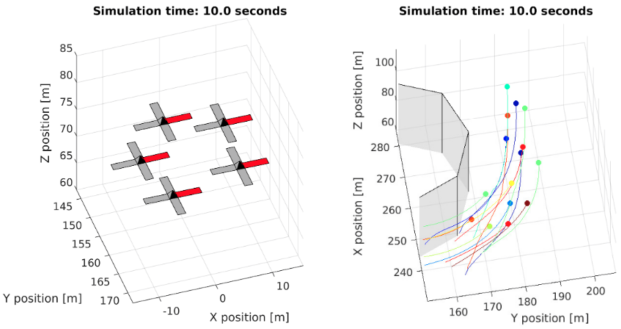
The work presented in this chapter is adapted from [43]:

- E. Soria, F. Schiano, D. Floreano, "SwarmLab: a MATLAB Drone Swarm Simulator," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 8005-8011, doi: 10.1109/IROS45743.2020.9340854.

Supplementary video: https://youtu.be/xMXA9OWSxe8.
SwarmLab is available on Github: https://github.com/lis-epfl/swarmlab.

B.1 Introduction

The first step towards the deployment of drone swarm in real-world scenarios is simulation[138]. The development of algorithms and applications for autonomous aerial vehicles requires the availability of a suitable simulation framework for rapid prototyping and simulation in reproducible scenarios. This is desirable in all robotics fields, but it is especially relevant for collective systems such as drone swarms, where errors can propagate through the individuals and lead to catastrophic results [139]. Although multiple open-source frameworks exist for



(a)

(b)

Figure B.1 – **3D swarm visualizations in SwarmLab**. In (a), 5 quadcopter drones coordinate in collective flight, while, in (b), a swarm of 15 drones simulated with point-mass dynamics executes a collision avoidance maneuver around an obstacle. Both snapshots are captured at $10s$ of simulation.

simulating aerial robots [140, 141, 142, 143], the majority are focused on the realism of a single robot and cannot manage a large number of drones in real-time. On the other side, simulators that support multiple robots do not implement the nonlinear robot's dynamics or they require the interaction with several programming languages. Besides, there is no framework that provides ready-to-use control algorithms, debugging tools and performance analysis functionalities for aerial swarms. The potential user has to develop their own tools compatible with the chosen framework, which are not standard and prone to error.

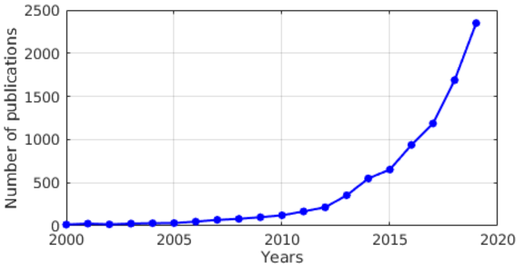


Figure B.2 – ***Aerial swarms* publications**. Number of publications containing the words *'aerial swarms'* between the years 2000 and 2019. Source: http://apps.webofknowledge.com.

We propose SwarmLab, a simulator for single drones and drone swarms. Its main goal is to propose an alternative to existing robotic simulation solutions, that is explicitly centered on drone swarms (see Fig. B.1). We used MATLAB[1], a scripting programming platform that allows us to implement and debug algorithms rapidly, use a large database of built-in functionalities, and create plots and videos with minimum effort. SwarmLab allows both accurate simulation of one drone, and efficient simulation of swarming behaviors with hundreds of agents. Concerning swarming, which is our focus, we provide support for fast instantiation of drone swarms, and the creation of environments with obstacles. We also include control algorithms from the state of the art, extensive plotting and debugging tools, 3D visualization

---

[1] https://mathworks.com/products/matlab.html

functionalities, and performance analysis tools. These features make our software relevant for the booming biological and robotics research communities in the field of aerial swarms (see Fig. B.2), and for education.

The rest of the chapter is organized as follows. We discuss available alternatives for robotic and, more specifically, drone simulations, while highlighting the gap that we aim to fill with SwarmLab. We describe the architecture of the software and guides potential users through its main functionalities. Finally, we show how SwarmLab can be used for a comparative analysis of swarm algorithms and analyzes the computational time required for different simulation configurations.

B.2 Related Work

Currently, many robotic simulators are available on the market. A subset of them allows drone swarm simulations, but still require a considerable amount of time and programming languages for the prototyping and testing of aerial swarms algorithms, with the inconveniences stated in Sec 4.1. A complete survey of the state of the art in robotic simulation is beyond the scope of this work (see [144, 145, 146] for a more detailed overview and performance comparison). Instead, we aim to highlight the main features they offer and point out the needs that led to the development of SwarmLab.

Among the most well-known open-source 3D simulators for robots, we find Gazebo, WeBots, V-REP, ARGoS, and, more recently, AirSim [133, 140, 141, 142, 147, 143]. Gazebo [133, 140] can simulate the physics and dynamics of any mechanical structure modeled with joints, and it offers a large library of ready-to-use models, drones included. Gazebo also allows integration with flight controller stacks for Software-In-The-Loop (SITL) drone simulation [135]. Alternatively, RotorS is an extension to Gazebo designed for multirotors that includes example controllers, besides additional models and simulated sensors [148, 149]. Users can code their functionalities in C++ and interface through ROS. WeBots, recently released open-source, uses the same physics engine of Gazebo, but provides APIs for a large number of programming languages and includes drone models [141] natively. The V-REP simulator, now continued under the name of CoppeliaSim, offers features for easier editing of robots and other models [142]. Development can be performed by means of the built-in Lua interpreter or by using a C or Python API. More oriented to swarm robotics, ARGoS represents a lightweight alternative that offers a good tradeoff between scalability and extensibility [147]. It allows the user to simulate a larger number of robots and it provides the possibility to use physics engines of different types, but it does not include drone models natively. Robots can be programmed either through Lua scripts or in C++. Specifically dedicated to drones and cars, AirSim [143] is a more recent simulator built on Unreal Engine [2] and as Gazebo, it allows SITL integration of flight controllers such as PX4. In AirSim, multi-agent simulations are easy to set up, and custom functionalities can be coded thanks to C++ and Python APIs.

---

[2] https://www.unrealengine.com

All the simulators above are based on powerful 3D rendering engines, and they are mainly coded in C++. As a consequence, they provide graphical realism. However, they often require familiarity with more than one programming language. Also, they necessitate the addition of custom features for simulating an aerial swarm, which makes these simulators unsuitable for quick tests. V-REP and WeBots include drone models natively, but drone swarm control and navigation algorithms must be designed, coded, and tuned by the user.

The simulators mentioned above are general-purpose robotic simulators. Instead, specific to drone simulation, we find the work by Beard et al. [121]. They describe fixed-wing drone systems with a waterfall architecture, where high-level blocks steer the drone to a goal destination, and lower-level blocks simulate physics and sensors. Driven by educational purposes, the authors released open-source templates in MATLAB and Simulink[3]. However, this work does not include quadcopter dynamics and swarming functionalities. However, it constitutes the foundation of SwarmLab.

To the best of our knowledge, the only publicly available simulators geared towards aerial swarms are robotsim[4] [13] and the work of D'Urso et al. [150]. The first is a simulator written in C and, although it goes in the direction of SwarmLab, no drone dynamics are implemented and architectural modularity is missing. The second is a software middleware that coordinates available tools (Gazebo, ArduCopter[5] and ns-3[6]) for the realistic simulation of the physics, graphics, flight control stack, and communication of interconnected drones and computers. This software is thought as a bridge towards a real-world implementation. However, its realism comes at the expense of its ease of use. Indeed, this simulator does not have the advantage of being contained within a single software such as MATLAB. Moreover, none of the simulators mentioned in this section provide functionalities for plotting, analysis, and performance assessment of the collective motion, which represents a limitation that we intend to overcome.

B.3 Software architecture

SwarmLab is written in MATLAB for several reasons. Firstly, this is a scripting language that operates at a high level of abstraction and therefore does not require extensive programming experience. Secondly, this framework provides several built-in toolboxes for design, control, analysis, and visualization of the studied systems, that reduce even further the programming effort and make it widely popular among the scientific community for education and research applications. Moreover, code generation features are available to automatically translate the code to C/C++ and reduce the computational time or embed the algorithms on the robots controllers. SwarmLab follows the Object Oriented Programming (OOP) paradigm and its modular structure is made of the following main components:

---

[3] https://github.com/randybeard/mavsim_template_files

[4] https://github.com/csviragh/robotsim

[5] https://ardupilot.org/copter/

[6] https://www.nsnam.org/

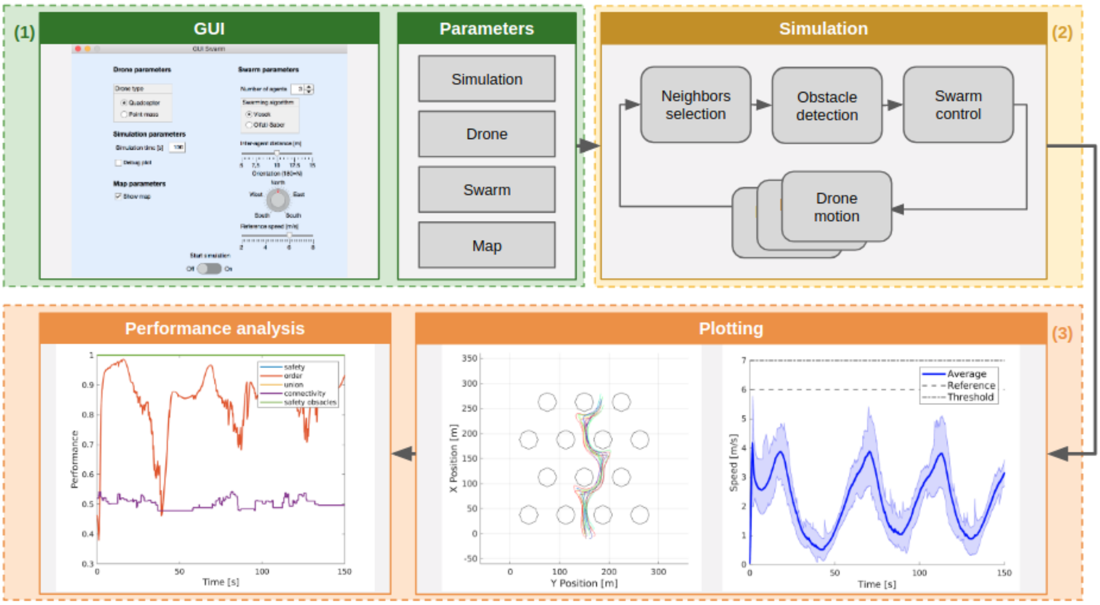


Figure B.3 – **SwarmLab simulation workflow**. From the top left, in clockwise order: (1) in the GUI, the user sets the parameters related to the simulation, drone typology, swarm algorithm and environment. Alternatively, parameters can be set in specific MATLAB scripts. Then, he launches the simulation; (2) the main simulation loop computes control commands for the drones, based on the information of the map and neighboring drones; (3) both real-time and post-simulation plotting of the state variables help the user with the analysis and debugging of the swarming behavior. Moreover, at the end of the simulation the user can inspect the swarm performance metrics.

- parameter scripts for the single drone, swarm, and environment definitions;
- the *Drone* and *Swarm* classes;
- graphical classes that allow run-time and offline 3D visualization of the *Drone*/*Swarm*, their state variables and performance;
- example scripts and a README file that guide the user through the main functionalities of the simulator.

B.3.1 Drone

The *Drone* class represents the building block for simulating a swarm. This class supports the definition of quadcopters or fixed-wing drones, based on the models in [151] and [121] respectively. A *Drone* instance is defined by:

- parameters related to the chosen platform (e.g., mass, aerodynamic and control parameters),

- current state vector: $(p_n, p_e, p_d, u, v, w, \phi, \theta, \psi, p, q, r) \in \mathbb{R}^{12}$. This vector is respectively composed by the north, east and down position coordinates in the inertial frame, the linear velocity measured along the x, y, z axes of the body frame, three Euler angles describing the drone orientation, i.e. roll, pitch, and yaw, and the angular velocities measured in the body frame,
- path planning variables, including a list of waypoints,
- graphic variables for the visualization of the drone and the plotting of the state variables.

The methods provided by this class allow the creation of new instances, the computation of the kinematics and dynamics based on the physical parameters, and the control of the drone thanks to one of the two autopilots tailored for either quadcopters or fixed-wing drones. Moreover, for the simulation of a single-drone mission, high-level functionalities for path navigation are provided, following the same structure of [121].

B.3.2 Swarm

The *Swarm* class contains the necessary properties and methods to instance, initialize and manage *Swarm* objects. These objects are made of an ensemble of dynamic agents of type *Drone*. Their fundamental properties are:

- *drones*: a vector of *Drone* objects,
- *nb_agents*: the number of agents included in the swarm,
- *algorithm*: the selected algorithm for swarm navigation.

The workflow of a swarm simulation is summarized in Fig. B.3. The user can start a simulation either by running an example script or by interacting with a Graphical User Interface (GUI) that accounts for real-time changes of the swarm parameters. When the simulation starts, a number of *Drone* instances are created and added to the *Swarm*. Also, the user can decide to instance a *swarm viewer* to visualize the evolution of the swarm state during the simulation time. The main simulation loop computes at every iteration the control commands for every drone of the swarm and updates their states. Control commands for a given drone i only depend on its neighbors $\mathcal{N}_i$. Depending on the user's choice, the *Swarm* class uses a different swarm algorithm to compute commands for every drone. Alternatively, the user can implement and test their own control algorithm as a method of the *Swarm* class where the drones' states are accessible, by following the available examples.

B.3.3 Swarm algorithms

In SwarmLab, we implemented and adapted two representative algorithms belonging to the category of decentralized swarming. The reason for this choice is that a decentralized approach can make the system easily scalable and robust to the failures of a single individual. The first algorithm is authored by Olfati-Saber, who proposes a formal theoretical framework for the design and analysis of swarm algorithms based on potential fields and graph theory [22]. It is based on the construction of a *collective potential* that penalizes the deviation of the agents from a lattice shape. In addition, a *consensus* term makes the agents agree on their speed and velocity direction. At the equilibrium, in the absence of obstacles, the agents occupy positions at a constant distance from their neighbors and translate with constant velocity. The second algorithm we implemented is an adaptation of the recent Vasarhelyi's algorithm, defined by the following rules: *repulsion* to avoid inter-agent collisions, *velocity alignment* to steer the agents to an average direction, and *self-propulsion* to match a preferred speed value [13]. In addition, the algorithm includes friction forces that reduce oscillations and ease the implementation on real robots. Finally, both algorithms propose an *obstacle avoidance* behavior to deal with convex obstacles. In several engineering applications (e.g., mapping, area coverage, search and rescue), we require the swarm to fly in a specific direction. To this aim, we allow the selection between the consensus on velocity in Olfati-Saber's algorithm, or the velocity alignment in Vasarhelyi's algorithm and a so-called *migration* term that penalizes deviations from a given velocity.

In decentralized approaches, one agent's movement is only influenced by local information coming from its neighbors. Neighbors selection can be operated according to different metrics. Two widely adopted ones are the euclidean and the topological distances [20, 19]. The euclidean distance defines $\mathcal{N}_i$ as the set of agents $j \neq i$ within a constant radius of influence r from agent i. The cardinality of this set depends on the density of the swarm. Instead, the topological distance defines $\mathcal{N}_i$ as the number nn of nearest agents to i (see Chap. 2.5). In the latter case, the cardinality does not depend on the density. In our software, both distances are implemented, and they can be set before starting the simulation.

For the navigation in cluttered environments, we provide a *map* that generates cylindrical obstacles with parametric size and density (see Fig. B.4). In both Olfati-Saber's and Vasarhelyi's algorithms, the obstacle avoidance behavior is modeled via virtual agents. These are additional agents to which we assign a position and velocity that depend on the obstacles configuration, and they act on drones as if they were normal agents.

SwarmLab offers two modalities for swarm simulation: the high-fidelity mode simulates quadcopter drones, where realistic dynamics and control are implemented, while the second approximates the drone dynamics with the dynamics of a point mass, whose state is defined by inertial position and velocity. This is meant to trade-off simulation fidelity and computational efficiency (see Sec. B.5 for more details).

B.3.4 Graphical User Interfaces (GUIs)

For introducing the user to the simulator functionalities we provide two GUIs: one for selecting the parameters related to single drones simulations and one for aerial swarms simulations. The

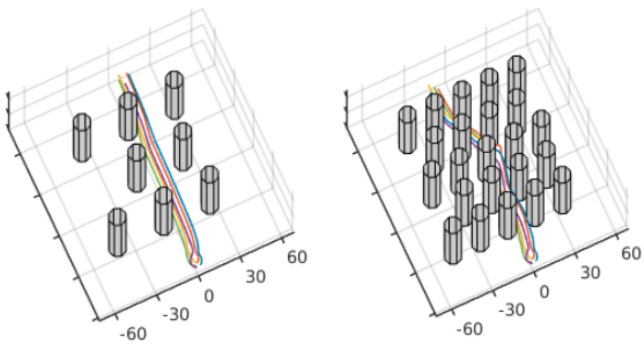


Figure B.4 – **Maps with varying obstacle density**. Cylindrical obstacles are distributed on the map to reproduce a forest-like environment. The obstacle density increases from left to right. Thanks to the obstacle avoidance behavior, the swarm agents are able to avoid collisions with the environment.

latter is split into sections that allow the user to select the drone dynamics, either quadcopter or point-mass, the main swarm parameters such as the number of drones, the preferred value of the inter-agent distance, the speed and orientation of the swarm motion, and simulation parameters such as the simulation time duration, the presence of debugging plots and the creation of a map with obstacles.

B.3.5 Plotting tools

One of the most critical parts of programming is verifying the validity of the code and algorithms. To this aim, a user needs tools to analyze the state of the system and find the origin of potential faults. SwarmLab allows the tracking of: (i) inter-agent distance and distance to obstacles, in order to detect collisions, (ii) swarm speed, useful for instance to monitor slow-down effects in front of obstacles, (iii) acceleration, to observe its variability and, hence, the efficiency of the algorithm. State plotting is possible both during the simulation, *run-time*, and at the end, *offline*. Run-time is useful for debugging, while offline is practical when the user does not want to slow down the simulation with the addition of graphic features. Single-drone plotting can be used simultaneously to observe the state of a specific drone in the swarm.

B.3.6 Performance analysis

The presence of obstacles in the environment can threaten the ability of the agents to remain cohesive during their mission and prevent them from flying smoothly in the migration direction. In these situations, the swarm may split into multiple subgroups with no influence on one another, and collisions may occur. To evaluate the collective navigation performance during flight, we use five metrics adopted in previous work [26]. These metrics were inspired by robotic and biological studies of aerial swarms:

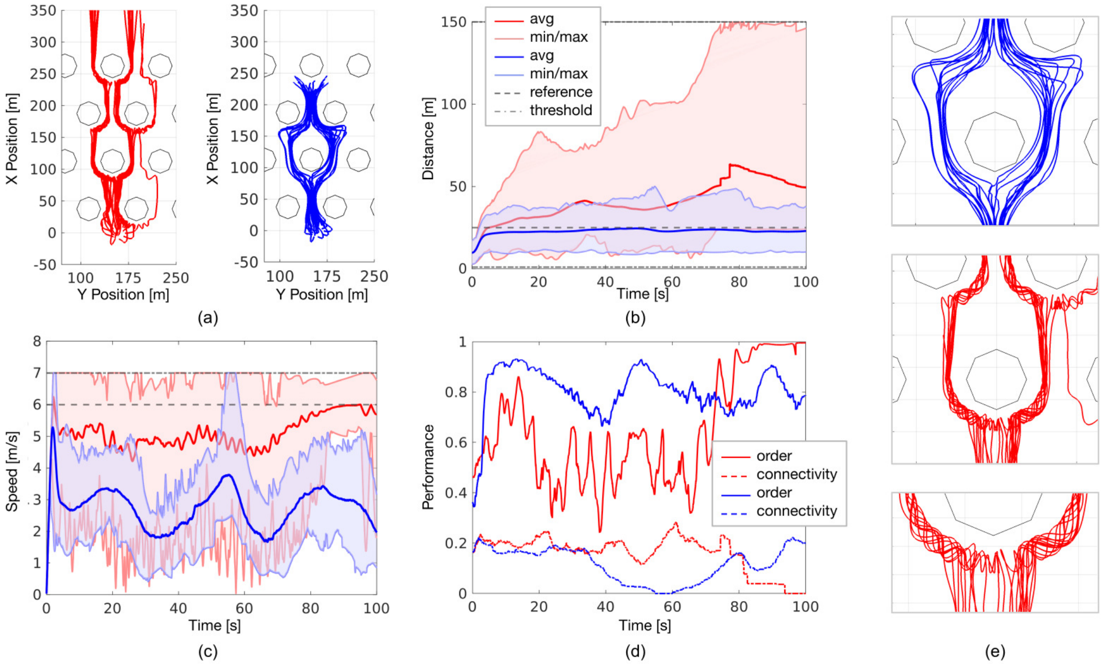


Figure B.5 – **Comparison of two swarm algorithms in SwarmLab**. Olfati-Saber's plots are in red, while Vasarhelyi's plots are in blue. The simulation time is $100s$ for both algorithms. In (a) we observe the top view of the trajectories of 25 agents, flying in an obstacle field from lower to higher values of the x position. In (b) and (c) inter-agent distances and speeds are compared, in terms of average, minimum, and maximum values. The reference values are in dashed lines, while the collision threshold, the radius of influence r, and the maximum speed are in dashdotted lines. Two of the presented performance metrics are compared in (d), the order (Φ_o) and the connectivity (Φ_c). Finally, (e) shows the zoom in of the trajectories of the agents around obstacles, from above.

- the *order* metric, Φ_{order}: it captures the correlation of the agents' movements and gives an indication about how ordered the flock is.

- The *safety* metrics, $\Phi_{safe,agent}$ and $\Phi_{safe,obs}$: they respectively measure the risk of collisions among the swarm agents or between agents and obstacles.

- The *union* metric, Φ_{union}: it counts the number of independent subgroups that originates during the simulation.

- The *connectivity* metric, $\Phi_{connectivity}$: it is defined from the algebraic connectivity of the sensing graph that underlines the considered swarm configuration.

For the performance metric definition, please refer to 2.6.

B.4 Comparison of swarm algorithms and computational time analysis

In this section, we present the results of the comparison of two swarm algorithms enabled by SwarmLab. Moreover, we present an analysis of the computational time of the simulator in different modes.

To compare swarm algorithms, we present a use case where 25 agents fly in a cluttered environment. We select point-mass dynamics, and we perform the neighbor selection with $nn = 10$ and $r = 150\ m$. The agents' initial positions are randomly selected in a cubic volume, and the swarm is let navigate over 100 s in the direction of increasing values of the x position. Both swarm algorithms described in Sec. B.3.2 are tested and the graphical outputs are reported in Fig. B.5. We notice that Olfati-Saber's algorithm prioritizes the tracking of the speed reference value (see Fig. B.5c), while Vasarhelyi's one allows the agent to slow down in front of obstacles to better match the reference inter-agent distance (see Fig. B.5b). The minimum distance threshold in Fig. B.5b is never crossed with both algorithms, which means that no inter-agent collisions occur. By examining the trajectories, in Fig. B.5a and Fig. B.5e, we see that the obstacle avoidance behavior of the second algorithm allows a smoother interaction of the agents with obstacles and reduces their oscillations, while in the first case, both in the trajectories and speed we observe prominent oscillations. Concerning the order Φ_{order}, better performance is obtained with Vasarhelyi's algorithm (see Fig. B.5d). Indeed, oscillations around obstacles prevent ordered flight in the case of Olfati-Saber's swarming. On the contrary, at the end of the simulation, Olfati-Saber's swarm order is higher. Indeed, once that the agents quit the obstacle field, in the free space, their velocity converges to the migration one. Contrarily, while the agents fly among obstacles, connectivity $\Phi_{\text{connectivity}}$ is slightly better in Olfati-Saber's case, and vice versa in the free space. Being connectivity related to the speed of the information flow among the agents, good values are preferred in scenarios where information-sharing among the agents is crucial (e.g., cooperative localization).

To evaluate the computational time, we run the two swarm algorithms with up to 1024 agents without any graphical output. The simulation time is arbitrarily set to 20 seconds. The hardware used is a DELL Precision Tower with a 3.6 GHz Intel Core i7-7700 processor and 16 GB 2400 MHz RAM. The results are reported in Fig. B.6, where the computational time is expressed in terms of *real-time factor*. A *real-time factor* equal to one means that the computational time required by the computer to run the simulation is equal to the simulation time. Instead, a value equal to two indicates that the computational time is twice the simulation time. The trend we notice is the same for both swarm algorithms and both drone typologies. As expected, when modelling the drones as point-masses the real-time factor is significantly lower. For instance, when we consider a swarm of 64 agents the real-time factor is close to 0.5 for the Vasarhelyi's algorithm and 0.9 for the Olfati-Saber's algorithm. Instead, when a full nonlinear quadcopter dynamics is used with the same amount of agents, the real-time factor increases up to 4.6 for the Vasarhelyi's algorithm and 5.2 for the Olfati-Saber's one.

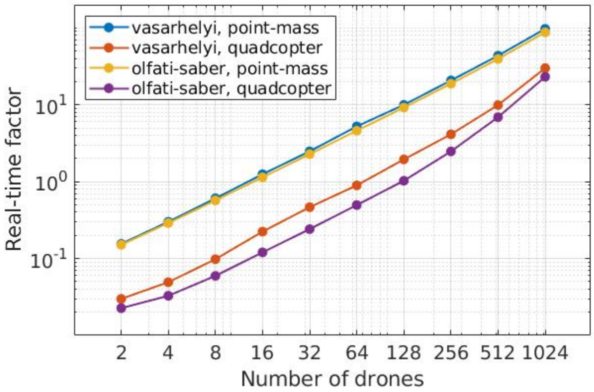


Figure B.6 – **Real-time factor for varying sizes of the swarm**. The number of drones goes from 2 to 1024. Two swarm algorithms (Vasarhelyi's and Olfati-Saber's) and two dynamics (point-mass and quadcopter) are compared.

B.5 Conclusions and future work

We presented a versatile and scalable drone swarm simulator entirely written in MATLAB that integrates built-in functionalities for collective navigation, debugging of the algorithms, and performance analysis. We believe that this framework can serve as a development tool and a comparative platform for the growing research community in aerial swarms, and for education. With reduced coding effort, the user can change parameters, edit their code, run and test it in a single scripted programming language. Regarding future work, we will focus on the improvement of the computational time to allow faster simulation of large swarms. For this, we will consider automatic C/C++ code generation from MATLAB. Moreover, noise modeling and delays should be considered to narrow the gap between simulation and reality. Finally, another challenge for future works is the integration of automatic parameter tuning for the swarm algorithms as done in [13]. This will allow to optimize the swarming behavior for a given environment or task with respect to the implemented performance metrics.

C Publications

Articles published in peer-reviewed journals:

- E. Soria, F. Schiano, D. Floreano, “Predictive control of aerial swarms in cluttered environments,” in *Nature Machine Intelligence*, vol. 3, pp. 545-554, May 2021 [10].
- E. Soria, F. Schiano, D. Floreano, “Distributed predictive drone swarms in cluttered environments,” in *IEEE Robotics and Automation Letters (RA-L)*, vol. 7, no. 1, pp. 73-80, Jan. 2022 [44].
- F. Schilling, E. Soria, D. Floreano, “On the Scalability of Vision-based Drone Swarms in the Presence of Occlusions,” in *IEEE Access*, (under review).

Articles submitted to peer-reviewed conferences:

- E. Soria, F. Schiano, D. Floreano, “The influence of limited visual sensing on the Reynolds flocking algorithm,” in *The Third IEEE International Conference on Robotic Computing (IRC)*, Naples, Feb. 2019 [26].
- E. Soria, F. Schiano, D. Floreano, “SwarmLab: a MATLAB drone swarm simulator,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Las Vegas, Feb. 2020 [43].
- E. Soria, H. Birch, F. Schilling, D. Floreano, “Waypoint Navigation of Vision-based Aerial Swarms without Estimation of Relative Positions,” in *IEEE International Conference on Robotics and Automation (ICRA)*, (under review).

Bibliography

[1] D. Floreano and R. J. Wood, “Science, technology and the future of small autonomous drones,” *Nature*, vol. 521, no. 7553, pp. 460–466.

[2] S. Chung, A. A. Paranjape, P. Dames, S. Shen, and V. Kumar, “A Survey on Aerial Swarm Robotics,” *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 837–855.

[3] G. Loianno and V. Kumar, “Cooperative Transportation Using Small Quadrotors Using Monocular Vision and Inertial Sensing,” *IEEE Robotics and Automation Letters (RA-L)*, vol. 3, no. 2, pp. 680–687, 2018.

[4] M. Varga, “Fixed-wing drones for communication networks.”

[5] S. Hauert, “Evolutionary Synthesis of Communication-Based Aerial Swarms.”

[6] M. Coppola, K. N. McGuire, C. De Wagter, and G. C. H. E. de Croon, “A Survey on Swarming With Micro Air Vehicles: Fundamental Challenges and Constraints,” *Front. Robot. AI*, vol. 7, 2020.

[7] A. Kushleyev, D. Mellinger, C. Powers, and V. Kumar, “Towards a swarm of agile micro quadrotors,” *Autonomous Robots*, vol. 35, no. 4, pp. 287–300.

[8] J. A. Preiss, W. Honig, G. S. Sukhatme, and N. Ayanian, “Crazyswarm: A large nano-quadcopter swarm,” in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 3299–3304.

[9] A. Weinstein, A. Cho, G. Loianno, and V. Kumar, “Visual Inertial Odometry Swarm: An Autonomous Swarm of Vision-Based Quadrotors,” *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1801–1807.

[10] E. Soria, F. Schiano, and D. Floreano, “Predictive Control of Aerial Swarms in Cluttered Environments,” *Nat. Mach. Intell.*, vol. 3, pp. 545–554, 2021.

[11] C. Virágh, G. Vásárhelyi, N. Tarcai, T. Szörényi, G. Somorjai, T. Nepusz, and T. Vicsek, “Flocking algorithm for autonomous flying robots,” *Bioinspiration & Biomimetics*, vol. 9, no. 2, 2014.

[12] G. Vasarhelyi, C. Viragh, G. Somorjai, N. Tarcai, T. Szorenyi, T. Nepusz, and T. Vicsek, "Outdoor flocking and formation flight with autonomous aerial robots," in *IEEE Int. Conf. Intel. Rob. Sys. (IROS)*, 2014, pp. 3866–3873.

[13] G. Vásárhelyi, C. Virágh, G. Somorjai, T. Nepusz, A. E. Eiben, and T. Vicsek, "Optimized flocking of autonomous drones in confined environments," *Science Robotics*, vol. 3, no. 20, 2018.

[14] K. N. McGuire, C. D. Wagter, K. Tuyls, H. J. Kappen, and G. C. H. E. de Croon, "Minimal navigation solution for a swarm of tiny flying robots to explore an unknown environment," *Science Robotics*, vol. 4, no. 35, p. eaaw9710, 2019.

[15] I. D. Couzin, J. Krause, N. R. Franks, and S. A. Levin, "Effective leadership and decision-making in animal groups on the move," vol. 433, p. 4.

[16] G. Dell'Ariccia, G. Dell'Omo, D. P. Wolfer, and H.-P. Lipp, "Flock flying improves pigeons' homing: GPS track analysis of individual flyers versus small groups," *Animal Behaviour*, vol. 76, no. 4, pp. 1165–1172.

[17] M. Nagy, Z. Ákos, D. Biro, and T. Vicsek, "Hierarchical group dynamics in pigeon flocks," *Nature*, vol. 464, no. 7290, pp. 890–893.

[18] M. Yomosa, T. Mizuguchi, G. Vásárhelyi, and M. Nagy, "Coordinated Behaviour in Pigeon Flocks," *PLOS ONE*, vol. 10, no. 10, p. e0140558.

[19] M. Ballerini, N. Cabibbo, R. Candelier, A. Cavagna, E. Cisbani, I. Giardina, V. Lecomte, A. Orlandi, G. Parisi, A. Procaccini, M. Viale, and V. Zdravkovic, "Interaction ruling animal collective behavior depends on topological rather than metric distance: Evidence from a field study," *Proceedings of the National Academy of Sciences*, vol. 105, no. 4, pp. 1232–1237, 2008.

[20] A. Strandburg-Peshkin, C. R. Twomey, N. W. F. Bode, A. B. Kao, Y. Katz, C. C. Ioannou, S. B. Rosenthal, C. J. Torney, H. S. Wu, S. A. Levin, and I. D. Couzin, "Visual sensory networks and effective information transfer in animal groups," *Current Biology*, vol. 23, no. 17, pp. R709–R711, 00144.

[21] C. W. Reynolds, "Flocks, Herds, and Schools: A Distributed Behavioral Model," *Computer Graphics*, vol. 21, pp. 25–43, 1987.

[22] R. Olfati-Saber, "Flocking for Multi-Agent Dynamic Systems: Algorithms and Theory," *IEEE Transactions on Automatic Control*, vol. 51, no. 3, pp. 401–420, 2006.

[23] S. Hauert, S. Leven, M. Varga, F. Ruini, A. Cangelosi, J.-C. Zufferey, and D. Floreano, "Reynolds Flocking in Reality with Fixed-Wing Robots: Communication Range vs. Maximum Turning Rate," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, p. 6.

[24] Y. Koren and J. Borenstein, “Potential field methods and their inherent limitations for mobile robot navigation,” in *IEEE International Conference on Robotics and Automation*. IEEE Comput. Soc. Press, 1991, pp. 1398–1404.

[25] H. Hildenbrandt, C. Carere, and C. K. Hemelrijk, “Self-organized aerial displays of thousands of starlings: A model,” vol. 21, no. 6, pp. 1349–1359.

[26] E. Soria, F. Schiano, and D. Floreano, “The influence of limited visual sensing on the Reynolds flocking algorithm,” *IEEE Third International Conference on Robotic Computing (IRC)*, pp. 138–145, 2019.

[27] I. D. Couzin, “Synchronization: The Key to Effective Communication in Animal Collectives,” *Trends in Cognitive Sciences*, vol. 22, no. 10, pp. 844–846, 2018.

[28] F. Borrelli, A. Bemporad, and M. Morari, *Predictive Control for Linear and Hybrid Systems:*, 1st ed. Cambridge University Press, 2017.

[29] L. Grune and J. Pannek, *Nonlinear Model Predictive Control. Theory and Algorithms*. Springer London, 2011.

[30] T. Baca, D. Hert, G. Loianno, M. Saska, and V. Kumar, “Model Predictive Trajectory Tracking and Collision Avoidance for Reliable Outdoor Deployment of Unmanned Aerial Vehicles,” in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6753–6760.

[31] D. Falanga, P. Foehn, P. Lu, and D. Scaramuzza, “PAMPC: Perception-Aware Model Predictive Control for Quadrotors,” *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, p. 8, 2018.

[32] C. E. Luis, M. Vukosavljev, and A. P. Schoellig, “Online Trajectory Generation With Distributed Model Predictive Control for Multi-Robot Motion Planning,” *IEEE Robot. Autom. Lett. (RA-L)*, vol. 5, no. 2, pp. 604–611, 2020.

[33] T. Keviczky, F. Borrelli, K. Fregene, D. Godbole, and G. J. Balas, “Decentralized Receding Horizon Control and Coordination of Autonomous Vehicle Formations,” *IEEE Transactions on Control Systems Technology*, vol. 16, no. 1, pp. 19–33.

[34] R. Van Parys and G. Pipeleers, “Distributed model predictive formation control with inter-vehicle collision avoidance,” in *Asian Contr. Conf. (ASCC)*, 2017, pp. 2399–2404.

[35] U. Eren, A. Prach, B. B. Koçer, S. V. Raković, E. Kayacan, and B. Açıkmeşe, “Model Predictive Control in Aerospace Systems: Current State and Opportunities,” *Journal of Guidance, Control, and Dynamics*, vol. 40, no. 7, pp. 1541–1566.

[36] W. B. Dunbar and R. M. Murray, “Distributed receding horizon control for multi-vehicle formation stabilization,” *Automatica*, vol. 42, no. 4, pp. 549–558, 2006.

[37] R. Raffard, C. Tomlin, and S. Boyd, "Distributed optimization for cooperative agents: Application to formation flight," in *2004 43rd IEEE Conference on Decision and Control (CDC) (IEEE Cat. No.04CH37601)*, vol. 3, pp. 2453–2459 Vol.3.

[38] T. Schouwenaars, J. How, and E. Feron, "Decentralized Cooperative Trajectory Planning of Multiple Aircraft with Hard Safety Guarantees," in *AIAA Guidance, Navigation, and Control Conference and Exhibit.* American Institute of Aeronautics and Astronautics.

[39] Y. Kuwata and J. P. How, "Robust Cooperative Decentralized Trajectory Optimization using Receding Horizon MILP," in *American Control Conference.* IEEE, pp. 522–527.

[40] A. Richards and J. How, "Implementation of Robust Decentralized Model Predictive Control," in *AIAA Guidance, Navigation, and Control Conference and Exhibit*, ser. Guidance, Navigation, and Control and Co-Located Conferences. American Institute of Aeronautics and Astronautics.

[41] I. K. Erunsal, R. Ventura, and A. Martinoli, "Nonlinear model predictive control for 3d formation of multirotor micro aerial vehicles with relative sensing in local coordinates," *arXiv preprint arXiv:1904.03742*, 2019.

[42] X. Zhou, J. Zhu, H. Zhou, C. Xu, and F. Gao, "EGO-Swarm: A Fully Autonomous and Decentralized Quadrotor Swarm System in Cluttered Environments," *arXiv*, 2020.

[43] E. Soria, F. Schiano, and D. Floreano, "SwarmLab: A Matlab Drone Swarm Simulator," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* IEEE, pp. 8005–8011.

[44] E. Soria., F. Schiano, and D. Floreano, "Distributed predictive drone swarms in cluttered environments," *IEEE Robot. Autom. Lett. (RA-L)*, vol. 7, no. 1, pp. 73–80, 2021.

[45] D. Bachman, *Advanced Calculus Demystified.* McGraw-Hill Professional, Jun 2007.

[46] J. E. Marsden and A. Tromba, *Vector Calculus.* W. H. Freeman, 2003.

[47] F. Schilling, J. Lecoeur, F. Schiano, and D. Floreano, "Learning Vision-based Flight in Drone Swarms by Imitation," *IEEE Robotics and Automation Letters*, 2019.

[48] J. Schulman, J. Ho, A. Lee, I. Awwal, H. Bradlow, and P. Abbeel, "Finding Locally Optimal, Collision-Free Trajectories with Sequential Convex Optimization," in *Robotics: Science and Systems IX.* Robotics: Science and Systems Foundation, 2013.

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

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

[51] J. van den Berg, Ming Lin, and D. Manocha, "Reciprocal Velocity Obstacles for real-time multi-agent navigation," in *2008 IEEE International Conference on Robotics and Automation*. IEEE, 2008, pp. 1928–1935.

[52] 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.

[53] N. Dousse, G. Heitz, F. Schill, and D. Floreano, "Human-Comfortable Collision-Free Navigation for Personal Aerial Vehicles," vol. 2, no. 1, pp. 358–365.

[54] 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.

[55] B. Şenbaşlar, W. Hönig, and N. Ayanian, "Robust Trajectory Execution for Multi-robot Teams Using Distributed Real-time Replanning," in *Distr. Auton. Rob. Sys.* Springer International, 2019, pp. 167–181.

[56] E. Tolstaya, F. Gama, J. Paulos, G. Pappas, V. Kumar, and A. Ribeiro. Learning Decentralized Controllers for Robot Swarms with Graph Neural Networks.

[57] B. Rivière, W. Hoenig, Y. Yue, and S.-J. Chung, "GLAS: Global-to-Local Safe Autonomy Synthesis for Multi-Robot Motion Planning with End-to-End Learning," vol. 5, no. 3, pp. 4249–4256.

[58] H. G. Tanner, A. Jadbabaie, and G. J. Pappas, "Stable Flocking of Mobile Agents, Part I: Fixed Topology," in *Conference on Decision and Control*, vol. 2, p. 8.

[59] B. T. Fine and D. A. Shell, "Unifying microscopic flocking motion models for virtual, robotic, and biological flock members," *Autonomous Robots*, vol. 35, no. 2-3, pp. 195–219, 00012.

[60] Y. Shang and R. Bouffanais, "Influence of the number of topologically interacting neighbors on swarm dynamics," *Scientific Reports*, vol. 4, no. 1, p. 4184, 2014.

[61] M. Camperi, A. Cavagna, I. Giardina, G. Parisi, and E. Silvestri, "Spatially balanced topological interaction grants optimal cohesion in flocking models," *Interface Focus*, vol. 2, no. 6, pp. 715–725, 2012.

[62] A. Okabe, *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*, 2nd ed., ser. Wiley Series in Probability and Statistics. Wiley, 2000.

[63] M. Fiedler, "Laplacian of graphs and algebraic connectivity," *Banach Center Publications*, vol. 25, no. 1, pp. 57–70, 1989.

[64] T. Nestmeyer, P. R. Giordano, H. H. Bülthoff, and A. Franchi, "Decentralized simultaneous multi-target exploration using a connected network of multiple robots," *Autonomous Robots*, vol. 41, no. 4, pp. 989–1011, 2017.

[65] P. Robuffo Giordano, A. Franchi, C. Secchi, and H. H. Bülthoff, "A passivity-based decentralized strategy for generalized connectivity maintenance," *The International Journal of Robotics Research (IJRR)*, vol. 32, no. 3, pp. 299–323, 2013.

[66] D. Hanover, P. Foehn, S. Sun, E. Kaufmann, and D. Scaramuzza, "Performance, Precision, and Payloads: Adaptive Nonlinear MPC for Quadrotors," vol. 7, no. 2, pp. 690–697.

[67] W. Zhao and T. H. Go, "Quadcopter formation flight control combining mpc and robust feedback linearization," *Journal of the Franklin Institute*, vol. 351, no. 3, pp. 1335–1355, 2014.

[68] W. Ren and N. Sorensen, "Distributed coordination architecture for multi-robot formation control," *Robotics and Autonomous Systems*, vol. 56, no. 4, pp. 324–333, 2008.

[69] M. Kamel, J. Alonso-Mora, R. Siegwart, and J. Nieto, "Robust collision avoidance for multiple micro aerial vehicles using nonlinear model predictive control," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 236–243.

[70] A. Strandburg-Peshkin, D. R. Farine, I. D. Couzin, and M. C. Crofoot, "Shared decision-making drives collective movement in wild baboons," *Science*, vol. 348, no. 6241, pp. 1358–1361, 2015.

[71] R. Verschueren, G. Frison, D. Kouzoupis, N. van Duijkeren, A. Zanelli, R. Quirynen, and M. Diehl, "Towards a modular software package for embedded optimization," *IFAC-PapersOnLine*, vol. 51, no. 20, pp. 374–380, 2018.

[72] V. Kumar and N. Michael, "Opportunities and challenges with autonomous micro aerial vehicles," *The International Journal of Robotics Research*, vol. 31, no. 11, pp. 1279–1291, 2012.

[73] M. Petrlík, T. Báca, D. Hert, M. Vrba, T. Krajník, and M. Saska, "A robust uav system for operations in a constrained environment," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2169–2176, 2020.

[74] K. R. Sapkota, S. Roelofsen, A. Rozantsev, V. Lepetit, D. Gillet, P. Fua, and A. Martinoli, "Vision-based Unmanned Aerial Vehicle detection and tracking for sense and avoid systems," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 1556–1561.

[75] E. R. Hunt and S. Hauert, "A checklist for safe robot swarms," *Nat. Mach. Intell.*, vol. 2, no. 8, pp. 420–422, 2020.

[76] H. Cheng, Q. Zhu, Z. Liu, T. Xu, and L. Lin, "Decentralized navigation of multiple agents based on ORCA and model predictive control," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3446–3451.

[77] V. Kungurtsev and M. Diehl, "Sequential quadratic programming methods for parametric nonlinear optimization," *Computational Optimization and Applications*, vol. 59, no. 3, pp. 475–509, 2014.

[78] R. Van Parys and G. Pipeleers, "Online distributed motion planning for multi-vehicle systems," in *2016 European Control Conference (ECC)*. IEEE, 2016, pp. 1580–1585.

[79] C. E. Luis, "Distributed Trajectory Generation for Multiagent Systems," *Master Thesis*, 2019.

[80] L. Hei, J. Nocedal, and R. A. Waltz, "A numerical study of active-set and interior-point methods for bound constrained optimization," in *Modeling, Simulation and Optimization of Complex Processes*. Springer Berlin Heidelberg, 2008, pp. 273–292.

[81] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: an operator splitting solver for quadratic programs," *Mathematical Programming Computation*, vol. 12, no. 4, pp. 637–672, 2020.

[82] J. A. Preiss, W. Honig, G. S. Sukhatme, and N. Ayanian, "Crazyswarm: A large nano-quadcopter swarm," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 3299–3304.

[83] G. R. Martin, "Understanding bird collisions with man-made objects: A sensory ecology approach: Bird collisions," *Ibis*, vol. 153, no. 2, pp. 239–254.

[84] R. Kern, N. Boeddeker, L. Dittmar, and M. Egelhaaf, "Blowfly flight characteristics are shaped by environmental features and controlled by optic flow information," *Journal of Experimental Biology*, vol. 215, no. 14, pp. 2501–2514.

[85] N. Linander, M. Dacke, and E. Baird, "Bumblebees measure optic flow for position and speed control flexibly within the frontal visual field," *Journal of Experimental Biology*, vol. 218, no. 7, pp. 1051–1059.

[86] F. Schiano and P. Robuffo Giordano, "Bearing Rigidity Maintenance for Formations of Quadrotor UAVs," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[87] L. E. Parker, "Distributed algorithms for multi-robot observation of multiple moving targets," *Autonomous Robots*, vol. 12, no. 3, pp. 231–255, 2002.

[88] G. R. Martin, "Visual fields and their functions in birds," *Journal of Ornithology*, vol. 148, no. S2, pp. 547–562.

[89] U. Mehmood, N. Paoletti, D. Phan, R. Grosu, S. Lin, S. D. Stoller, A. Tiwari, J. Yang, and S. A. Smolka, "Declarative vs Rule-based Control for Flocking Dynamics," in *IEEE/ACM International Symposium on Applied Computing*, 2018, pp. 816–823.

[90] F. Schilling, E. Soria, and D. Floreano, “On the Scalability of Vision-based Drone Swarms in the Presence of Occlusions,” *IEEE Access*, p. (submitted), 2021.

[91] T. Qin, P. Li, and S. Shen, “VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator,” *IEEE Transactions on Robotics (T-RO)*, vol. 34, no. 4, pp. 1004–1020, 2018.

[92] P. M. Wyder, Y.-S. Chen, A. J. Lasrado, R. J. Pelles, R. Kwiatkowski, E. O. A. Comas, R. Kennedy, A. Mangla, Z. Huang, X. Hu, Z. Xiong, T. Aharoni, T.-C. Chuang, and H. Lipson, “Autonomous drone hunter operating by deep learning and all-onboard computations in GPS-denied environments,” *PLOS ONE*, vol. 14, no. 11, p. e0225092, 2019.

[93] M. Vrba and M. Saska, “Marker-Less Micro Aerial Vehicle Detection and Localization Using Convolutional Neural Networks,” *IEEE Robotics and Automation Letters (RA-L)*, vol. 5, no. 2, pp. 2459–2466, 2020.

[94] Y. Tang, Y. Hu, J. Cui, F. Liao, M. Lao, F. Lin, and R. Teo, “Vision-aided Multi-UAV Autonomous Flocking in GPS-denied Environment,” *IEEE Transactions on Industrial Electronics*, pp. 1–1, 2018.

[95] M. Saska, T. Baca, J. Thomas, J. Chudoba, L. Preucil, T. Krajnik, J. Faigl, G. Loianno, and V. Kumar, “System for Deployment of Groups of Unmanned Micro Aerial Vehicles in GPS-denied Environments Using Onboard Visual Relative Localization,” *Auton. Robots*, vol. 41, no. 4, pp. 919–944. [Online]. Available: https://doi.org/10.1007/s10514-016-9567-z

[96] P. Petracek, V. Walter, T. Baca, and M. Saska, “Bio-inspired compact swarms of unmanned aerial vehicles without communication and external localization,” *Bioinspir. Biomim.*, 2020.

[97] F. Schilling, F. Schiano, and D. Floreano, “Vision-Based Drone Flocking in Outdoor Environments,” *IEEE Robotics and Automation Letters (RA-L)*, vol. 6, no. 2, pp. 2954–2961, 2021.

[98] J. Chen, M. Gauci, W. Li, A. Kolling, and R. Groß, “Occlusion-Based Cooperative Transport with a Swarm of Miniature Mobile Robots,” *IEEE Transactions on Robotics (T-RO)*, vol. 31, no. 2, pp. 307–321, 2015.

[99] J. Hu, A. E. Turgut, T. Krajník, B. Lennox, and F. Arvin, “Occlusion-Based Coordination Protocol Design for Autonomous Robotic Shepherding Tasks,” *IEEE Transactions on Cognitive and Developmental Systems*, pp. 1–1, 2020.

[100] S. B. Rosenthal, C. R. Twomey, A. T. Hartnett, H. S. Wu, and I. D. Couzin, “Revealing the hidden networks of interaction in mobile animal groups allows prediction of complex behavioral contagion,” vol. 112, no. 15, pp. 4690–4695, 2015.

[101] J. D. Davidson, M. M. G. Sosna, C. R. Twomey, V. H. Sridhar, S. P. Leblanc, and I. D. Couzin, “Collective detection based on visual information in animal groups,” *Journal of the Royal Society Interface*, vol. 18, no. 180, p. 20210142, 2021.

[102] T. Vicsek and A. Zafeiris, “Collective motion,” *Physics Reports*, vol. 517, no. 3-4, pp. 71–140.

[103] I. D. Couzin, J. Krause, R. James, G. D. Ruxton, and N. R. Franks, “Collective Memory and Spatial Sorting in Animal Groups,” *Journal of Theoretical Biology*, vol. 218, no. 1, pp. 1–11.

[104] A. E. Turgut, H. Çelikkanat, F. Gökçe, and E. Şahin, “Self-organized flocking in mobile robot swarms,” *Swarm Intelligence*, vol. 2, no. 2-4, pp. 97–120, 2008.

[105] M. Lindhe, P. Ogren, and K. Johansson, “Flocking with Obstacle Avoidance: A New Distributed Coordination Algorithm Based on Voronoi Partitions,” in *IEEE International Conference on Robotics and Automation (ICRA)*, 2005, pp. 1785–1790.

[106] J. Holland and S. K. Semwal, “Flocking Boids with Geometric Vision, Perception and Recognition,” p. 8, 2009.

[107] H. Kunz and C. K. Hemelrijk, “Simulations of the social organization of large schools of fish whose perception is obstructed,” *Applied Animal Behavior Science*, vol. 138, no. 3, pp. 142–151, 2012.

[108] D. J. G. Pearce, A. M. Miller, G. Rowlands, and M. S. Turner, “Role of projection in the control of bird flocks,” *Proceedings of the National Academy of Sciences of the United States of America (PNAS)*, vol. 111, no. 29, pp. 10 422–10 426, 2014.

[109] R. Bastien and P. Romanczuk, “A model of collective behavior based purely on vision,” *Science Advances*, vol. 6, no. 6, p. eaay0792, 2020.

[110] S. Roelofsen, D. Gillet, and A. Martinoli, “Reciprocal collision avoidance for quadrotors using on-board visual detection,” in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 4810–4817. [Online]. Available: http://ieeexplore.ieee.org/document/7354053/

[111] D. Dias, R. Ventura, P. Lima, and A. Martinoli, “Onboard vision-based 3D relative localization system for multiple quadrotors,” in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 1181–1187.

[112] V. Walter, N. Staub, A. Franchi, and M. Saska, “UVDAR System for Visual Relative Localization with application to Leader-Follower Formations of Multirotor UAVs,” *IEEE Robotics and Automation Letters (RA-L)*, pp. 1–1, 2019.

[113] A. Rozantsev, V. Lepetit, and P. Fua, “Detecting Flying Objects Using a Single Moving Camera,” *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 39, no. 5, pp. 879–892, 2017.

[114] R. Opromolla, G. Fasano, and D. Accardo, "A Vision-Based Approach to UAV Detection and Tracking in Cooperative Applications," *Sensors (Basel)*, vol. 18, no. 10, 2018.

[115] M. Vrba, D. Hert, and M. Saska, "Onboard Marker-Less Detection and Localization of Non-Cooperating Drones for Their Safe Interception by an Autonomous Aerial System," *IEEE Robotics and Automation Letters (RA-L)*, vol. 4, no. 4, pp. 3402–3409, 2019.

[116] T. Cieslewski, S. Choudhary, and D. Scaramuzza, "Data-Efficient Decentralized Visual SLAM," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 2466–2473.

[117] M. Vrba and M. Saska, "Marker-Less Micro Aerial Vehicle Detection and Localization Using Convolutional Neural Networks," vol. 5, no. 2, pp. 2459–2466.

[118] F. Schilling, F. Schiano, and D. Floreano, "Vision-Based Drone Flocking in Outdoor Environments," *IEEE Robot. Autom. Lett. (RA-L)*, vol. 6, no. 2, pp. 2954–2961, 2021.

[119] X. Zhou, Z. Wang, H. Ye, C. Xu, and F. Gao. (2020) EGO-Planner: An ESDF-free Gradient-based Local Planner for Quadrotors.

[120] X. Zhou, Z. Wang, X. Wen, J. Zhu, C. Xu, and F. Gao. Decentralized Spatial-Temporal Trajectory Planning for Multicopter Swarms.

[121] R. W. Beard and T. W. McLain, *Small Unmanned Aircraft: Theory and Practice.* Princeton University Press, oCLC: ocn724663112.

[122] E. J. Rodríguez-Seda, D. M. Stipanović, and M. W. Spong, "Guaranteed Collision Avoidance for Autonomous Systems with Acceleration Constraints and Sensing Uncertainties," vol. 168, no. 3, pp. 1014–1038.

[123] 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.

[124] M. Debord, W. Hönig, and N. Ayanian, "Trajectory Planning for Heterogeneous Robot Teams," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 7924–7931.

[125] G. Shi, W. Hönig, X. Shi, Y. Yue, and S.-J. Chung, "Neural-Swarm2: Planning and Control of Heterogeneous Multirotor Swarms using Learned Interactions," 2021.

[126] M. Bujarbaruah, X. Zhang, H. E. Tseng, and F. Borrelli, "Adaptive MPC for Autonomous Lane Keeping," *arxiv*, 2018.

[127] B. P. Duisterhof, S. Li, J. Burgués, V. J. Reddi, and G. C. H. E. de Croon, "Sniffy Bug: A Fully Autonomous Swarm of Gas-Seeking Nano Quadcopters in Cluttered Environments," *arxiv*, 2021.

[128] S. Li, C. De Wagter, and G. C. H. E. de Croon, "Self-supervised Monocular Multi-robot Relative Localization with Efficient Deep Neural Networks," *arxiv*, 2021.

[129] D. Palossi, A. Loquercio, F. Conti, E. Flamand, D. Scaramuzza, and L. Benini, "A 64-mW DNN-Based Visual Navigation Engine for Autonomous Nano-Drones," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8357–8371, 2019.

[130] B. Balazs and G. Vasarhelyi, "Coordinated Dense Aerial Traffic with Self-Driving Drones," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 6365–6372.

[131] W. Schwarting, J. Alonso-Mora, L. Pauli, S. Karaman, and D. Rus, "Parallel autonomy in automated vehicles: Safe motion generation with minimal intervention," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1928–1935.

[132] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS - an open-source Robot Operating System," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2009, p. 6.

[133] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 3. IEEE/RSJ, 2004, pp. 2149–2154 vol.3.

[134] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart, "RotorS: A Modular Gazebo MAV Simulator Framework," in *Robot Operating System (ROS)*, ser. Studies in Computational Intelligence. Springer, Cham, 2016, pp. 595–625.

[135] L. Meier, D. Honegger, and M. Pollefeys, "PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 6235–6240.

[136] G. Silano, E. Aucone, and L. Iannelli, "CrazyS: A Software-In-The-Loop Platform for the Crazyflie 2.0 Nano-Quadcopter," in *2018 26th Mediterranean Conference on Control and Automation (MED)*. IEEE, pp. 1–6.

[137] T. Baca, M. Petrlik, M. Vrba, V. Spurny, R. Penicka, D. Hert, and M. Saska, "The MRS UAV System: Pushing the Frontiers of Reproducible Research, Real-world Deployment, and Education with Autonomous Unmanned Aerial Vehicles," *IEEE Transactions on Robotics (T-RO)*, 2020.

[138] P. S. Andrews, S. Stepney, and J. Timmis, "Simulation as a scientific instrument," in *Proceedings of the 2012 workshop on complex systems modelling and simulation, Orleans, France*, 2012.

[139] B. Huang, C. Yu, and B. D. O. Anderson, "Understanding Error Propagation in Multihop Sensor Network Localization," vol. 60, no. 12, pp. 5811–5819, 2013.

[140] J. Meyer, A. Sendobry, S. Kohlbrecher, U. Klingauf, and O. von Stryk, “Comprehensive simulation of quadrotor uavs using ros and gazebo,” in *3rd Int. Conf. on Simulation, Modeling and Programming for Autonomous Robots (SIMPAR)*, 2012.

[141] O. Michel, “Cyberbotics ltd. webots™: professional mobile robot simulation,” *International Journal of Advanced Robotic Systems*, vol. 1, no. 1, 2004.

[142] E. Rohmer, S. P. N. Singh, and M. Freese, “V-REP: A versatile and scalable robot simulation framework,” in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, pp. 1321–1326.

[143] S. Shah, D. Dey, C. Lovett, and A. Kapoor, “AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles,” in *Field and Service Robotics*. Springer International Publishing, 2018, pp. 621–635.

[144] L. Pitonakova, M. Giuliani, A. Pipe, and A. Winfield, *Feature and Performance Comparison of the V-REP , Gazebo and ARGoS Robot Simulators*. Springer International Publishing, 2018.

[145] A. Staranowicz and G. L. Mariottini, “A survey and comparison of commercial and open-source robotic simulator software,” in *Proceedings of the 4th International Conference on Pervasive Technologies Related to Assistive Environments*, 2011.

[146] M. S. P. de Melo, J. G. da Silva Neto, P. J. L. da Silva, J. M. X. N. Teixeira, and V. Teichrieb, “Analysis and comparison of robotics 3d simulators,” in *2019 21st Symposium on Virtual and Augmented Reality (SVR)*. IEEE, 2019, pp. 242–251.

[147] C. Pinciroli, V. Trianni, R. O’Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. Di Caro, F. Ducatelle, M. Birattari, L. M. Gambardella, and M. Dorigo, “ARGoS: A modular, parallel, multi-engine simulator for multi-robot systems,” *Swarm Intelligence*, vol. 6, no. 4, pp. 271–295, 2012.

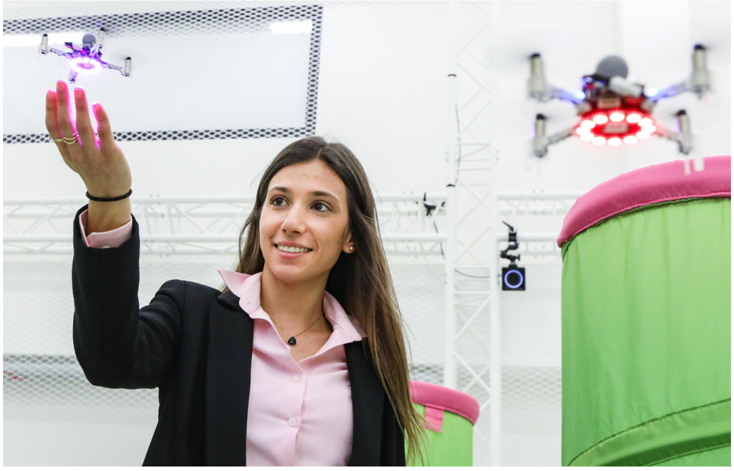
[148] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart, *Robot Operating System (ROS): The Complete Reference (Volume 1)*. Cham: Springer International Publishing, 2016, ch. RotorS—A Modular Gazebo MAV Simulator Framework, pp. 595–625.

[149] C. McCord, J. P. Queralta, T. N. Gia, and T. Westerlund, “Distributed progressive formation control for multi-agent systems: 2d and 3d deployment of uavs in ros/gazebo with rotors,” in *2019 European Conference on Mobile Robots (ECMR)*. IEEE, 2019.

[150] F. D’Urso, C. Santoro, and F. F. Santoro, “An integrated framework for the realistic simulation of multi-UAV applications,” *Computers & Electrical Engineering*, vol. 74, pp. 196–209.

[151] S. Bouabdallah and R. Siegwart, “Full control of a quadrotor,” in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Ieee, 2007, pp. 153–158.

Curriculum Vitae - Enrica Soria



First / last name: **Enrica Soria**

Date of birth: 10.11.1992

Place of birth: Nizza Monferrato

Nationality: Italian

Mobile: +41 (0) 76 372 23 61

Email: enrica.soria@epfl.ch / enrica.s92@gmail.com

ORCID: 0000-0002-1364-6439

Education

- 01.2018 - 01.2022 (expected date of Ph.D. defense): **Ph.D. Thesis**, EPFL (Switzerland). Advisor: Prof. Dario Floreano; doctoral school: Robotics, Control, and Intelligent Systems (EDRS);
    - Thesis title: “Predictive control of aerial swarms with limited sensing”;
    - Courses: Business concepts training by Innosuisse; Design of experiments; Aerial robotics; State-of-the-art techniques in neuroscience.
- 09.2014 - 12.2016: **M.Sc. in Mathematical Engineering** (double degree), Polytechnic of Turin and Polytechnic of Milan (Italy). Advisors: Prof. Paolo Brandimarte and Prof. Luigi Preziosi; score: **110/110 with honors**;
    - Specialization in ‘Physical Modelling and Simulation’;
    - Main courses: Nonlinear systems for engineering applications; Linear systems and control theory; Convex optimization; Object-oriented programming; Statistical models; Discrete events models; Stochastic processes; Business analysis.
- 09.2014 - 12.2016: **honor program Alta Scuola Politecnica** (https://www.asp-poli.it/), Polytechnic of Turin and Polytechnic of Milan (Italy);
    - Project: “Drone technology for monitoring glaciers and water resources”;
    - Tasks: collection and analysis of temporal and geographical data series; creation and analysis of DSMs. Roles: team leader and budget supervisor;
    - Courses: Digital Innovations; Design methods and processes; The energy question.
- 09.2011 - 10.2014: **BS.c. in Mathematics for Engineering Applications**, Polytechnic of Turin (Italy); score: **110/110 with honors**.

Work experience

- 01.2018 - 01.2022: **Ph.D. Research Assistant**, EPFL (Lausanne, Switzerland). Advisor: Prof. Dario Floreano;
- 04.2017 - 10.2017: **R&D Engineer**, Amadeus IT Group (Sophia Antipolis, France). Advisor:

Jeremie Barlet;
  - Definition of the security access system and computation of the services access metrics;
  - Implementation of a big-data monitoring system based on ElasticSearch and Kibana.
- 05.2016 - 01.2017: **R&D Intern**, senseFly (Cheseaux-sur-Lausanne, Switzerland). Advisors: Dr. Arnaud Gelas, Prof. Paolo Brandimarte;
  - M.Sc. thesis title: “Clustering algorithms for 3D surface splitting problems”;
  - Design and implementation of a C++ algorithm for optimally splitting a wide area to be mapped by a professional drone for cartography.

Teaching activities

- Spring 2020 - 2021: teaching assistant; course: Aerial Robotics; EPFL;
  - Development of the software exercises on the drone’s dynamics, control, estimation, and path-planning algorithms;
  - Teaching practicals;
  - Creation and management of the students’ Github repository for the course.
- Spring 2020 - 2021: semester project supervisor; student: Samuele Lanzanova; title: “Communication-faulty drone swarms in heterogeneous environments”;
- Fall 2020 - 2021: semester project supervisor; student: Hugo Birch; title: “Vision-based drone swarms with limited visual sensing”;
- Fall 2020 - 2021: semester project supervisor; student: Thomas Oliver Kimble; title: “Predictive drone swarms with a limited field of view”;
- Spring 2019 - 2020: teaching assistant; course: Aerial Robotics; EPFL;
  - Maintenance of the Gazebo software exercises and creation of online documentation;
  - Teaching practicals.
- Fall 2019 - 2020: semester project supervisor; student: Andrea Giordano; title: “SwarmLab: a user-friendly package for drone swarm simulation”;
- Spring 2018 - 2019: teaching assistant; course: Aerial Robotics; EPFL;
  - Teaching practicals.
- Fall 2018 - 2019: semester project supervisor; student: Yoann Lapijover; title: “Swarming algorithms for quadcopters”;
- Fall 2018 - 2019: semester project supervisor; student: Victor Delafontaine; title: “Versatile Simulator for a swarm of quadcopters”;

Participation in conferences, workshops, summer schools

- 06.2021: Summer School on “**Foundations and mathematical guarantees of data-driven control**”, online due to COVID-19 restrictions;

- 10.2020: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), online due to COVID-19 restrictions;
- 11.2019: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Macau;
- 07.2019: IEEE RAS Summer School on **Multi-Robot Systems** (MRS), Prague (Czech Republic);
- 02.2019: IEEE International Conference on Robotic Computing (IRC), Naples (Italy);
- 12.2018: BMI Symposium on “State-of-the-art techniques in neuroscience”; Lausanne (Switzerland);
- 05.2018: ICTP Conference on “**Collective behavior**” (smr 3201), Trieste (Italy)

Technical skills

- **Mathematical engineering**: control of linear and nonlinear systems; statistical models; discrete events models; stochastic processes; machine learning;
- **Robotics**: kinematics and dynamics; controller design; data acquisition and processing; SITL/HITL experiments; crazyswarm infrastructure; Optitrack motion capture system;
- **Video editing and illustration**: Inkscape, Illustrator, Photoshop, Premiere Pro;
- **Software**: Linux/Windows; Python; Matlab; Simulink; Latex; MS Office; C++; C;
- **Software packages**: ROS, Gazebo, Casadi, acado, acados, OSQP.

Personal skills

- Effective **communication and presentation**;
    - One-year participation in speaking classes with Toastmasters (international clubs).
- **Social commitment**;
    - Participation in multiple humanitarian missions across Switzerland (2018-2020) and Kenya (2019) with Drone Adventures (no-profit association based in Lausanne).

Languages

- **English**: full professional proficiency
- **French**: full professional proficiency
- **Italian**: native proficiency
- **German**: elementary proficiency