POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

Autonomous navigation of quadrotor swarms

FLORIAN KARYDES

Département de génie électrique

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*

Génie électrique

Janvier 2020

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

Ce mémoire intitulé :

Autonomous navigation of quadrotor swarms

présenté par **Florian KARYDES**

en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*

a été dûment accepté par le jury d'examen constitué de :

Richard GOURDEAU, président
David SAUSSIÉ, membre et directeur de recherche
Luca Giovanni GIANOLI, membre

DEDICATION

To Papi Jo,
Who inspired my passion for engineering.

ACKNOWLEDGEMENTS

I would like to thank my supervisor David Saussié, who, two years ago, gave me the chance to pursue my double degree at Polytechnique Montreal in the Automation and Systems section of the Electrical Engineering Department.

This research project was funded by the Consortium for Research and Innovation in Aerospace in Quebec (CRIAQ), Prompt and Humanitas Solutions. Therefore, I would like to thank Abdo Shabah and Luca Giovanni Gianoli, respectively CEO and ICT lead scientist at Humanitas Solutions for their trust.

I would like to thank my colleagues at the Mobile Robotics and Automated Systems Laboratory (MRASL) and at Humanitas Solutions for their unwavering cheerful and welcoming disposition. A special thought for Vincent Dabin who happens to be a pretty good pianist in addition to being a good friend.

This international experience would not have been the same without all the beautiful and amazing people I have met (or joined) on this side of the Atlantic Ocean and I thank them all for that. But my supervisor said "No more than a hundred pages", and I already failed, so I will have to omit some names. First, I thank Maxime Folcolini who most probably gave me the drive to cross the ocean and lent me a perfect inflatable mattress when I first arrived in Montreal. Next, I have to thank Charley Gros for teaching me the ropes of life in Montreal and eventually letting me use his garage when I needed a roof. For the most insane adventures I may have lived in Canada, I give credit to Mohamed Larbi Sentissi and Sebastien Vicens. Moreover, I want to thank my love, Laura Duval, for the support and the joy she brought me throughout my master's degree. Last but not least, I would like to thank my parents, Catherine and Lionel Karydes, for giving me the will to learn, for believing in my projects and for supporting me financially. I would have never made it this far into the maze of sciences if not for you. I love you all.

RÉSUMÉ

La mise sur le marché de composants toujours plus performants et compétitifs en termes de coût, ainsi que le développement rapide des technologies de commande et de navigation en robotique, nous ont amenés à envisager le contrôle d'un large essaim de quadrirotors. Diverses solutions impliquant des drones existent déjà pour différentes applications: inventaire forestier, gestion du littoral, suivi du trafic, etc. Parmi celles-ci, la recherche et le sauvetage en situation d'urgence représentent à nos yeux la possibilité la plus intéressante et constitue, de fait, la première motivation de notre travail. Par conséquent, une large revue de littérature sur la question est fournie. Ce travail se concentre sur le contrôle de l'essaim lui-même, et non sur l'application finale.

Tout d'abord, un modèle mathématique de la dynamique du quadrirotor est présenté et plusieurs lois de commande numérique sont synthétisées. Ces dernières implémentent les modes de fonctionnement nécessaires aux algorithmes de navigation, à savoir : commande en vitesse, commande en position et commande en suivi. Ensuite, deux solutions originales et complémentaires de contrôle d'essaim sont proposées.

D'une part, un algorithme d'essaimage pour la navigation extérieure est développé. Contrairement à la plupart des travaux trouvés dans la littérature, la solution proposée ici gère non seulement le maintien, mais aussi l'initialisation de la formation. Plus spécifiquement, un modèle de formation hexagonale est introduit. Ensuite, les places en formation sont attribuées de façon optimale à l'aide de l'*algorithme hongrois*. Enfin, les agents se déplacent jusqu'à la place qui leur est assignée tout en évitant les autres agents avec un algorithme de navigation inspiré du *Artificial Potential Field*. De plus, cette solution tient compte de contraintes de conception réalistes et a été intégrée avec succès dans un logiciel embarqué de quadrotor déjà existant et opérationnel. Les résultats de simulations *Software-In-The-Loop* sont fournis.

D'autre part, une solution d'essaimage pour la navigation intérieure est étudiée. L'algorithme proposé implémente un certain nombre de comportements individuels simples, de sorte qu'un grand essaim peut suivre un meneur dans des environnements encombrés en se fiant uniquement aux informations locales. Des simulations préliminaires sont effectuées et les résultats montrent qu'il serait possible de faire fonctionner, conformément au besoin étudié, un essaim de cent quadrirotors avec l'algorithme proposé. En particulier, l'essaim est capable de suivre le meneur, de maintenir la connectivité, d'éviter les collisions entre agents, d'éviter les obstacles et même de se faufiler dans des espaces étroits.

ABSTRACT

The ever-growing hardware capabilities and the rapid development of robotic control and navigation technologies have led us to consider the control of an entire swarm of quadrotors. Drone-based solutions have been developed for different applications: forest inventory, coastal management, traffic monitoring, etc... Among these, the Search And Rescue application represents for us a very promising field of application and constitutes the first motivation of our work. As a result, a wide literature review on the matter is provided. Nevertheless, this work focuses on the swarm control itself, and not on the end user application.

First, a mathematical model of the quadrotor dynamics is presented and several digital control laws are designed. The latter provide operating modes useful for the navigation algorithms, namely: velocity control, position control and tracking control. Then, two original and complimentary swarming solutions are proposed.

On the one hand, a swarming algorithm for outdoor navigation is developed. Unlike most of the works reviewed in the literature, our solution handles not only the maintenance but also the initialization of the formation. More specifically, an hexagonal formation pattern is introduced. Then, positions are optimally assigned using the *Hungarian algorithm*. Finally, the agents move to their assigned position while avoiding collisions with the other fleet members thanks to a navigation algorithm inspired from Artificial Potential Field. In addition, this solution accounts for realistic design constraints and was successfully integrated into already existing quadrotor onboard software. Software-In-The-Loop simulation results are provided.

On the other hand, a swarming solution for indoor navigation is investigated. The proposed algorithm enforces a certain set of expected individual simple behaviors such that a large swarm can follow a leader through cluttered environments relying only on local information. Preliminary simulations are run and the results show that it is possible to operate a swarm of a hundred quadrotors with the proposed algorithm. In particular, the swarm is able to follow the leader, maintain connectivity, avoid collisions with the other agents, avoid obstacles, and even squeeze to pass through narrow spaces.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ACRONYMS

ADC	Analog-Digital Converter
ADRC	Active Disturbance Rejection Control
ANFIS	Adaptive-Network-based Fuzzy Inference System
APF	Artificial Potential Field
AS	Asymptotic Stability
BS	Base Station
BSC	Backstepping Control
CAS	Collision Avoidance System
CCW	Counterclockwise
CEO	Chief Executive Officer
CF	Complementary Filter
CPU	Central Processing Unit
CRIAQ	Consortium for Research and Innovation in Aerospace in Quebec
CW	Clockwise
DC	Direct Current
DCM	Direct Cosine Matrix
DIM	Dense Image Matching
DoF	Degrees of Freedom
EKF	Extended Kalman Filter
ESC	Electronic Speed Control
FIS	Fuzzy Inference System
FL	Fuzzy Logic
GA	Genetic Algorithm
GNN	Grossberg Neural Network
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
HEAVEN	Heterogeneous Embedded Ad hoc Virtual Emergency Network
HITL	Hardware-In-The-Loop
ICT	Information and Communication Technology
IMU	Inertial Measurement Unit
LAP	Linear Assignment Problem
LAPJV	Linear Assignment Problem solver Jonker-Volgenent
LIDAR	Light Detection and Ranging
LiPo	Lithium Polymer

LMI	Linear Matrix Inequality
LOE	Loss Of Efficiency
LOS	Line-of-Sight
LQG	Linear-Quadratic-Gaussian
LQR	Linear–Quadratic Regulator
MAS	Multi-Agents System
MAV	Micro Air vehicle
MEMS	Micro-ElectroMechanical Systems
MILP	Mixed Integer Linear Program
MIQP	Mixed-Integer Quadratic Program
MOPP	Multi-objective Path Planning
MPC	Model Predictive Control
MRASL	Mobile Robotics and Automated Systems Laboratory
NED	North-East-Down
NN	Neural Network
OAS	Obstacle Avoidance System
OS	Operating System
PD	Proportional-Derivative
PF	Particle Filter
PHD	Probability Hypothesis Density
PID	Proportional-Integral-Derivative
PMIC	Pulse-Modulated Intermittent Control
PSG-IMC	Probabilistic Swarm Guidance using Inhomogeneous Markov Chain
PSO	Particle Swarm Optimization
RADAR	Radio Direction and Ranging
RANSAC	Random Sample Consensus
RC	Radio Controlled
REMODE	Regularized Monocular Depth
RL	Reinforcement Learning
ROS	Robot Operating System
RTOS	Real-Time Operating System
SAR	Search And Rescue
SFM	Structure From Motion
SIFT	Scale-Invariant Feature Transform
SITL	Software-In-The-Loop
SLAM	Simultaneous Localization and Mapping

SMC	Sliding Mode Control
SONAR	Sound Navigation and Ranging
SVO	Semidirect Visual Odometry
UAV	Unmanned Aerial Vehicle
UKF	Unscented Kalman Filter
VM	Virtual Machine
VRP	Vehicle Routing Problem
WGS84	World Geodetic System 84
ZOH	Zero-Order Hold

x	Scalar
$\mathbf{x}$	Vector
$\mathbf{X}$	Matrix
$\mathtt{x}$	Frame
$\mathbf{x}^{\mathtt{f}}$	$\mathbf{x}$ coordinates in frame $\mathtt{f}$
$\boldsymbol{\omega}_{\mathtt{a/b}}$	Rotation rate of frame $\mathtt{a}$ w.r.t. frame $\mathtt{b}$
i, j, m, n	Elements index
k	Period index
d_x	Distance x
T_x	Time x
$\mathtt{b}$	Body frame
$\mathtt{o}$	NED frame
$\mathtt{h}$	Hexagonal frame
ψ	Quadrotor yaw angle
θ	Quadrotor pitch angle
ϕ	Quadrotor roll angle
$\mathbf{r}$	Quadrotor position
$\mathbf{v}$	Quadrotor velocity
$\mathbf{u}$	Control input
$\bar{\mathbf{x}}$	Reference input on $\mathbf{x}$ variable
$\mathbf{e}_x$	Tracking error on $\mathbf{x}$ variable
T_s	Sampling time
K_x	Control gain x
N	Number of agents
$\mathcal{G}_x$	Graph x
$\mathcal{V}_x$	Set of nodes of graph x
$\mathcal{E}_x$	Set of edges of graph x

LIST OF APPENDICES

CHAPTER 1 INTRODUCTION

1.1 Background

Since the 1990s, researchers and companies around the world never stopped designing Unmanned Aerial Vehicles (UAVs) with different shapes, technologies, and purposes. Originally used as practice targets for military training, UAVs development experienced an important shift when the Dayton-Wright Airplane Company invented a pilotless aerial torpedo that would explode at a preset time during World War I [1].

Nowadays, UAVs are heavily used for military applications: in 2013, at least 50 countries operated UAVs for intelligence and defense. A certain democratization of UAVs has been observed in the last years, as the use of UAVs has become vey popular in many civilian applications such as photography, mapping, survey, and security.

A specific type of UAVs, the quadrotors, have become quite popular among the research community [2]. Indeed, the four-rotor design allows them to be relatively simple in design, yet highly reliable and maneuverable. Therefore, they have become perfect platforms for safe and low-cost experiments in navigation and control strategies for 6 Degrees of Freedoms (DoFs) mobile robots.

More recently, the distributed control of multiple quadrotors and Multi-Agents Systems (MASs), has been extensively studied [3]. Indeed, using MASs composed of several cheaper and smaller units may present numerous advantages compared with one unique super-efficient expensive unit. Apart from the obvious financial risk related to a system failure when dealing with a single and centralized solution, MASs provide high resilience and robustness, enhanced sensing capabilities and lower physical risks when it comes to development. In various domains, such as exploration, search, rescue and surveillance, UAVs' MASs could be used to map dangerous or inaccessible areas, help find survivors after major disasters, identify risks such as gas leaks or structure vulnerabilities, provide an ad hoc communication network for rescue teams and detect intruders in restricted areas [4].

1.2 Aim

This project is financed and supervised by Humanitas Solutions, a team of designers, developers, and researchers building resilient solutions for critical environments. Thus, it is integrated into a higher-level workflow that targets emergency response and especially Search

And Rescue (SAR) applications. Even though the SAR matter is investigated in the literature review and may have influenced some development choices, this project focuses on the swarm control itself, not on the end-user application, with the end goal of designing a quadrotor swarm offering autonomous navigation capabilities. For instance, the solutions proposed here should allow one rescuer without specific training to operate a large swarm of UAV from the operation base to the intervention field.

1.2.1 General objective

The general objective of this project is to design and develop swarming solutions to allow a group of quadrotors to fly in formation through free and cluttered environments, in a fully distributed and autonomous fashion.

1.2.2 Specific objectives

To support the general objective, five specific objectives have been identified.

Objective #1 - Provide modularity and scalability to the swarm
The agents have to communicate and coordinate with each other to achieve the mission in a distributed manner. Nevertheless, the dependencies must be contained to a minimum level to ensure a maintainable and scalable system.

Objective #2 - Ensure robustness and fault tolerance
The swarm must be robust to single agent faults. Moreover, it should be able to endure communication errors or external disturbances. Also, the control law has to be compliant with material capabilities.

Objective #3 - Perform collision and obstacle avoidance
The navigation algorithm has to prevent quadrotors from colliding with each other and crashing against environment obstacles.

Objective #4 - Maintain connectivity among the swarm
Every agent in the swarm has to keep a way to interact with all the others.

Objective #5 - Guarantee real-time efficiency
The swarming problem mixes computation, communication, and control. Therefore, we need to consider how digitalization, i.e., quantification and sampling, may affect real-time performances of the system.

Objective #6 - Provide reliable proof of performances
Simulations account only for a restrictive part of what we know from the real-world phe-

nomenons. Thus, the most reliable way to asses performances remains experiments. Therefore, efforts have to be made to get as close as possible from an operational prototype, e.g., Software-In-The-Loop (SITL) or Hardware-In-The-Loop (HITL) simulations.

1.3 Scope

The autonomous navigation problem is very complex and interdisciplinary, especially when it comes to indoor navigation. While a comprehensive solution would have to consider on-board state estimation, localization, mapping, and even communication issues, this project solely focuses on the formation control part of the problem. This also implies that the high-level path planning of the entire swarm in the global frame, as well as the low-level controller of agents, are not extensively studied here. Moreover, only formations in the horizontal plane (i.e., the same altitude for all agents) are considered. Finally, two distinct swarming solutions are proposed for indoor and outdoor environments.

1.4 Organization

The remainder of this work is organized as follows. Chapter 2 provides a wide review on UAVs' swarm control, navigation and applications in the SAR context. Chapter 3 describes the dynamics of the quadrotors and presents several simple control laws that enable different flight modes which are relevant for formation control. Chapter 4 discusses the design of a simple swarming solution for outdoor environments without obstacle avoidance. This solution is implemented in ROS and some advanced SITL simulations are realized using the software solutions under development at Humanitas Solutions. Chapter 5 presents a distributed leader-follower formation algorithm suitable for indoor environments (i.e., with obstacle avoidance) inspired by the work of Sakai et al. [5]. Finally, Chapter 6 summarizes the important results and achievements of this work and mentions potential future projects and improvements.

CHAPTER 2 LITERATURE REVIEW

2.1 Overview

The aim of this chapter is to provide a global view of both research challenges and general aspects related to quadrotor swarm control for SAR applications. Before going into further details, let us clarify the literature review scope and organization (Figure 2.1). Hence, the review will follow a top-down approach: from SAR applications to estimation & tracking methods. Note that hardware solutions themselves will not be discussed here unless they interfere in a specific way with the software solution (e.g. estimation solutions are associated with a particular type of sensor).

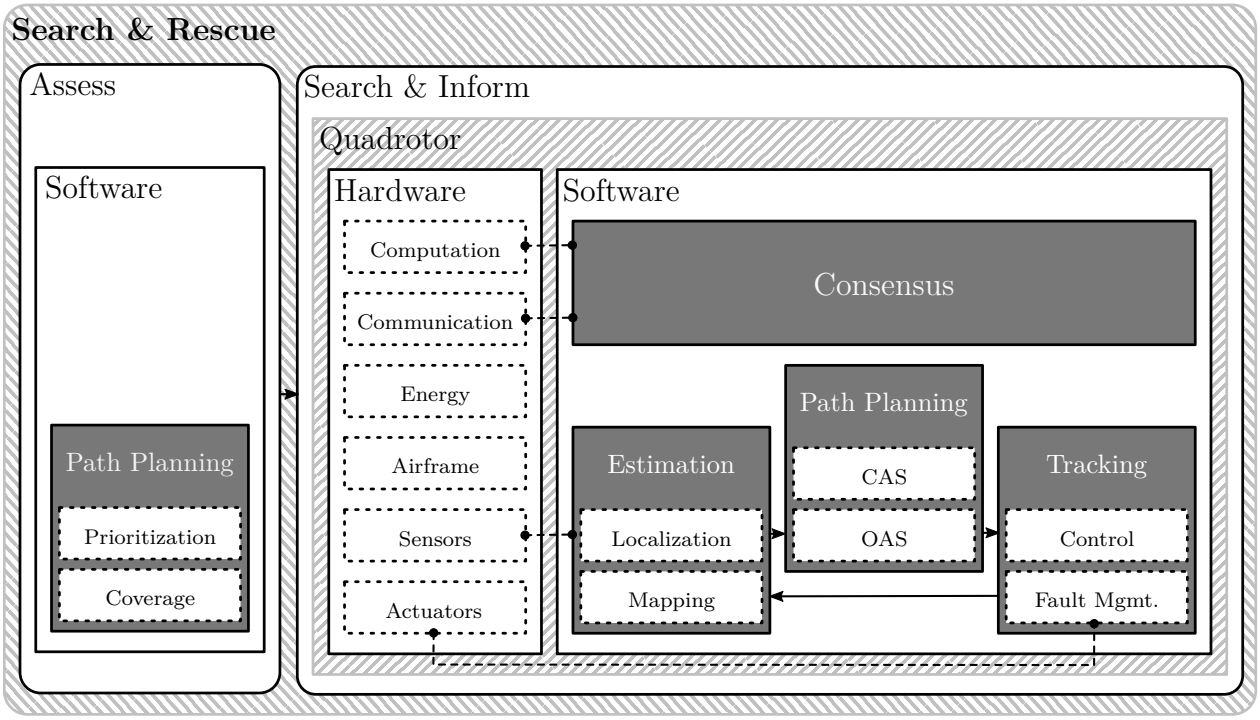


Figure 2.1 Review scope

2.2 Search & Rescue

We first focus on the SAR problem and the related methods. Qi et al. [6] reported that manual SAR operations are usually conducted by several small teams, hardly coordinated.

Each team counts around twelve professionals and is responsible for a small area ($\approx 5\,\mathrm{km}^2$). They decomposed the search and rescue procedure into seven phases :

1. Collect and analyze data about the assigned search region;
2. Limit access to hazardous areas;
3. Determine search strategy and priorities;
4. Execute search strategy;
5. Rescue survivors;
6. Provide medical care;
7. Analyze performances and adapt methods for the future.

This decomposition is not really relevant while considering a quadrotor swarm instead of a team of humans. Hence, we will be considering only three relevant axes in the following : *assess*, *search* and *inform*.

2.2.1 Assess

The first phase, called the *assess* phase, gathers all the steps that occur before the swarm take-off, i.e., analyzing collected knowledge to define priorities and develop a search strategy. Indeed, we may be able to collect some existing maps of the area, satellite imagery, and various other information about the region such as population density or transport corridors. Based on such data, San Juan et al. [7] proposed to build a *risk/occupancy map* using a Fuzzy Inference System (FIS).

2.2.2 Search

The second phase occurs from the take-off to the landing of the swarm. During this phase, the swarm will try to apply the predefined *search* applyins strategy real-time modifications only when necessary. Traditionally, the search has been conducted using predetermined standard search patterns such as rectangular sweep [8]. However, though still limited, the ever-growing computation capabilities of quadrotors enable non-standard approaches to the search problem that promise to be more effective. Dames and Kumar [9] addressed the problem of autonomous localization of an unknown number of targets with a stochastic solution, namely the Probability Hypothesis Density (PHD) filter. Note that the developed

solution is able to assess the end of the mission. Ji et al. [10] designed an adaptive density function based on an online updated probability map and an uncertainty map. Those are then used to balance target detection and area exploration. Li and Duan [11] modeled the search problem with Game Theory. A search strategy is updated online using a potential game approach. Then optimal coverage is guaranteed with binary log-linear learning. Hayat et al. [12] developed an optimization-based solution for multi-objective UAV swarm SAR using a Genetic Algorithm (GA). Their solution aims to minimize both search duration (i.e., area coverage) and network connectivity according to a certain weighting. Finally, San Juan et al. [7] proposed four algorithms relying on graph theory : an Artificial Potential Field (APF) approach, a Fuzzy Logic (FL) approach, an Adaptive-Network-based Fuzzy Inference System (ANFIS) and a Particle Swarm Optimization (PSO) approach. Note that the FL and the PSO solutions are tuned using GA. According to their tests, the ANFIS approach has been shown to be the best one.

2.2.3 Inform

Simultaneously to the *search* mission, the swarm will have to ensure a communication link toward the Base Station (BS) and even provide a communication network to rescuers in the field. This problem is addressed by Hayat et al. [12] through three different communication strategies :

- *Information First* (IF) strategy: once a target is found, a messenger (i.e., one designated UAV among the swarm) flies on a direct path to bring the target information to BS (i.e. *data ferrying*). This allows a fast response of the rescue team.
- *Connectivity First* (CF) strategy: at the time a target is found, the swarm will have to establish a communication link from the target to the BS. This enables the reactive monitoring of the target.
- *Simultaneous Information & Connectivity* (SIC) strategy: this hybrid solution aims to ensure enough connectivity to link the target to the BS while a messenger is sent to notify the BS.

All three strategies are implemented using the Multi-objective Path Planning (MOPP) algorithm that provides GA-optimal coverage.

2.3 Swarm

Formation control of a UAVs swarm is a special case of the consensus problem. Consensus means that all the agents of the swarm reach an agreement on certain variables of interest in order to fulfill a defined goal [13]. Primary goals of such a system in a SAR context are to ensure cohesion of the swarm and coverage of the area while avoiding obstacles in the environment and collisions among the swarm members.

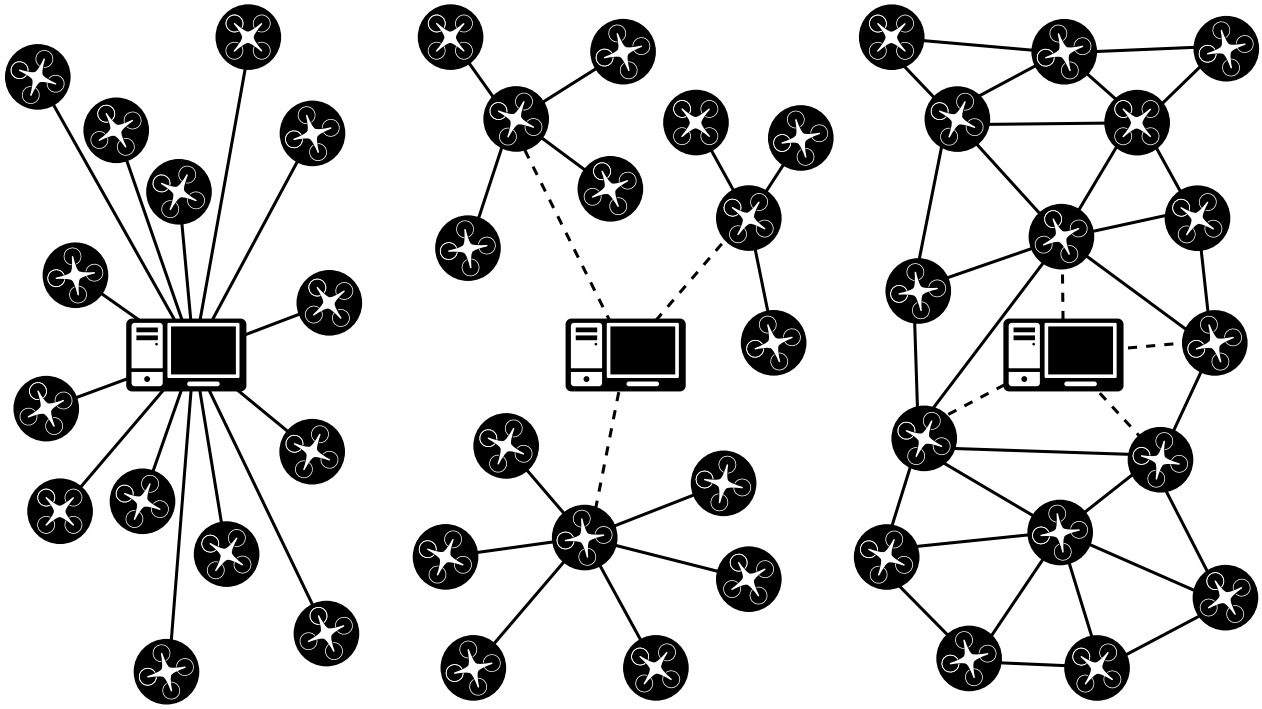
2.3.1 Architectures

MAS can be organized along different architectures (Figure 2.2) that we classify in two groups. The first includes centralized architectures based on a BS that collects all information, deals with most of the computational load and makes decisions for the agents to follow. Due to the dependency on the the BS, this approach is not the most promising. Moreover, a faulty communication link between the agents and the BS will have a strong impact on the system performance. In the second group we find decentralized architectures letting the UAVs taking part in decision and planning procedures, increasing in this way the overall system robustness. A distinction can be made between two different variants of decentralized architectures. First, we consider a fully decentralized architecture where all UAVs will be totally autonomous and independent. This configuration provides strong resilience but does not fully exploit the full power of a swarm, as each UAV acts independently or in small groups. For this reason, we will prefer a distributed architecture according to which the UAVs are demanded to directly cooperate and communicate in order to make a collective decision, in other words, to achieve a consensus. In addition, Ge et al. [14] proposed a cluster architecture for MAS. The agents therein were split into subgroups, potentially resulting in larger swarms.

2.3.2 Models

MASs are often modeled using graph theory for both formation and communication modeling. We define here some graph basic properties. In brief, a graph is a set of *nodes* connected by *edges*. We distinguish *directed* graphs from *undirected* graphs. The undirected graphs feature exclusively bidirectional edges (e.g., each node can receive from and respond to connected nodes) [15]. Conversely, directed graphs imply unidirectional connections. The symmetry in undirected graphs results in great model simplifications and easier controller designs. However, depending on the algorithm one might want to implement, using a directed graph model may be a more efficient [16]. We can also enrich the model by weighting either the edges [14] or the nodes [17]. Most of the solutions developed for MAS control rely on a classic

graph with positive weights, meaning agents' actions converge to the same objective (i.e., collaborative behavior). However, recent studies on real-world cases (e.g., social networks) have shown that antagonistic behaviors may enhance the overall performance of the group. In order to reproduce that kind of antagonistic interactions, Altafini [18] introduced a *signed* graph with positive and negative weights for consensus formation control. We have just seen that modeling choices will have a strong impact on the performances of the final control solution. Namely, properties attributed to the graph model directly influence system performance. In that regard, Sun et al. [19] proposed an energy-optimized formation control that involves graph topology optimization by estimating, at each iteration, the minimal rigid graph that fits the actual formation. This help minimizes communication complexity and energy dissipation.



(a) (b) (c)

Figure 2.2 (a) Centralized. (b) Decentralized. (c) Distributed architecture.

2.3.3 Strategies

Formation control of MAS has been a concern for researchers for more than a decade and several strategies have been developed accordingly. In their attempt to propose standardized formations for spacecraft control, Beard et al. [20] presented three formation strategies :

- *Leader-follower* approach: at least one agent is identified as the leader and tracks the trajectory specified by the path planner, while the rest of the swarm, namely the followers, keep tracking the leader's position by considering some prescribed and possibly time-varying offsets [13]. This is called *morphous* formation control as distances between agents are explicitly specified through a *formation function*. One may also consider variants such as multi-leader strategy [17] or switching leader strategy [21].

- *Behavior-based* approach : several expected behaviors are implemented in parallel on each agent [22]. In other words, it is a multi-objective control law that will try to achieve tracking, obstacle avoidance, cohesion, collision avoidance, connectivity, etc. simultaneously. Thus, this approach is categorized as amorphous formation control.

- *Virtual structure* approach: the swarm is considered as a single system, called virtual structure. The virtual structure approach is, a special case of the leader-follower approach which considers virtual leaders that are not actual agents.

2.3.4 Control

Back in 1986, Craig Reynolds, an artificial life and computer graphics expert, developed the *Boids* (short for "bird-oid" objects), an artificial life program which simulates the flocking behavior of birds [23]. Even though its work was initially intended for computer animation, C. Reynolds laid the foundations of behavior-based swarm control. In particular, he showed that the complex motion observed in a flock of birds, a herd of land animals or a school of fishes emerge from a set of simple distributed rules as that correspond to :

- Collision Avoidance: avoid collisions with nearby agents;

- Velocity Matching: attempt to match velocity with nearby agents;

- Flock Centering: attempt to stay close to nearby agents.

More complex rules can be added, such as obstacle avoidance and goal seeking.

Sabattini et al. [22] developed a behavior-based solution for distributed control of MAS while maintaining connectivity. This solution implements two linear control laws in a complementary manner: one for tracking, and one for connectivity maintenance. Xiang et al. [24], on the other hand, achieved distributed robust nonlinear control of UAV MAS in a leader-follower manner with a full order Sliding Mode Control (SMC). Note that, in addition to handling non-linearity, SMC is robust to both model uncertainties and external disturbances, which makes it a suitable candidate for the UAV MAS control problem. Jasim and Gu [25] also managed to achieve leader-follower distributed robust nonlinear control but decided to use suboptimal H_∞ control instead. Bandyopadhyay et al. [26] proposed to deal with distributed control of a large-scale MAS. They designed a Probabilistic Swarm Guidance using Inhomogeneous Markov Chain (PSG-IMC).

Concerning Obstacle Avoidance Systems (OASs) and Collision Avoidance Systems (CASs), Wang et al. [27] developed a *sense and avoid* dual-mode control strategy that involves a "safe mode" for obstacle-free environment operations and a "danger mode" activated when there is a chance of a collision with another agent or when there are obstacles on the path. In danger mode, collisions/obstacles avoidance is handled by a modified Grossberg Neural Network (GNN) and a visibility graph [28]. A more common way to ensure collision and obstacle avoidance would be the APF as proposed by Zhao et al. [29]. Assuming a leader-follower strategy, the basic idea of the APF method is that each follower is driven by two kinds of artificial forces generated by other agents of the swarm: an attractive force leading the agent to the desired position and a repulsive force avoiding collision among the swarm. Usually, those two forces and their ratio depend on the distances between agents. The agents are then moved toward the minimum of the potential field thus created in order to achieve control while ensuring collision avoidance. Note that for obstacle avoidance, one will just need to consider obstacles as fixed agents. Nevertheless, this method suffers from a strong drawback that needs to be addressed. Indeed, the potential field created will most likely present local minima that may prevent the agents from converging to their expected position and formation. Also, Sakai et al. [5] designed a complete CAS/OAS for UAV MAS enabling formation reorganization to go through narrow spaces and ensuring connectivity maintenance by only using geometric constraints (i.e., distances and Line-of-Sight (LOS) preservation). Nguyen et al. [30] used *path deformation* to deal with obstacle avoidance. In this case, close obstacle avoidance is completely integrated with path planning, applying a transformation to the predetermined path in the neighborhood of the obstacle. The main advantage of this method is its extreme simplicity, which leads to a straightforward design and limited computation load.

2.3.5 Communication

As far as the formation control problem is concerned, the main communication constraint consist in maintaining connectivity among the MAS members. Sabattini et al. [22] distinguished two approaches for connectivity maintenance :

- *The Local connectivity* preservation approach aims at preserving the original set of links throughout the mission. Of course, this strategy is highly restrictive as visibility and range constraints imply a roughly fixed formation;
- *The Global connectivity* preservation approach relaxes this restriction by enabling links to be removed and added at will as long as the overall connectivity is maintained.

To deal with global connectivity preservation, Sabattini et al. [22] introduced the concept of *critical agent*. An agent identifies itself as critical if at least one of its neighbors, as described by the communication graph, is isolated. In other words, critical agent disconnection will cause a split of the communication graph. Thus, their swarm control solution involves detecting the critical agent and limiting control law effects on those to avoid splitting the graph. Sakai et al. [5] achieved connectivity preservation in obstacle environments by limiting control output to preserves LOS between neighbor agents as well as maximum distance constraints. Their solution also features a set of rules that enable the swarm to safely change its network topology (i.e., its communication graph).

It is worth pointing out that the communication system in swarm suffers numerous limitations, constraints, and errors that have to be considered at the design stage. For example, one may experience packet disorders, packet dropouts, quantization errors, and transmission delays [3]. Communication capabilities may also be limited by antenna ranges, network topology complexity, and computation power.

2.3.6 Digitization

Even though control and communication are often modeled and studied as continuous processes, it cannot be ignored that the final implementation will be discrete both in time and amplitude. As a matter of fact, the way we deal with that digitization problem will greatly impact system performance, especially when it comes to swarm control where control and communication tasks happen in an interdependent manner. In their survey, Ge et al. [31] listed several strategies to deal with the sampling. We first consider *time-triggered* sampling. In such an approach, sampling action is driven by time but this can be formalized in a different way. We will be distinguishing *synchronous* sampling when all agents sample signals

at the same moment from *asynchronous* sampling when sampling happens at different times for all agents. Also, considering only one agent, we can define the sampling period to either *uniform*, *non-uniform* or even *random* (i.e., the period can be fixed over time, evolve in a deterministic way or even in a stochastic fashion). As a result, we can list six time-triggered sampling mechanisms : *Synchronous Uniform Sampling* (SUS), *Synchronous Non-uniform Sampling* (SNS), *Synchronous Random Sampling* (SRS), *Asynchronous Uniform Sampling* (AUS), *Asynchronous Non-uniform Sampling* (ANS) and *Asynchronous Random Sampling* (ARS). When it comes to performances, ARS would be the most efficient approach as it may require less sampling action on the overall and no strict synchronization in the swarm. But it will also require more effort when it comes to design. An even more efficient way to perform sampling is synchronous or asynchronous *Event-Triggered* (also called *self-triggered* [32]) sampling (SES/AES). However, one might have to be careful with the event-triggered sampling mechanism as it presents a major drawback known as *Zeno phenomenon*. In fact, under certain circumstances, the sampling period of an event-triggered mechanism may drop to zero and cause the failure of the system. So far, we only discussed time digitization. Li et al. [16] addressed amplitude digitization (i.e., quantization) for networks with limited bandwidth proposing an efficient quantization adaptation mechanism. One should remark that sampled control law remains unchanged during the whole sampling period because of the use of Zero-Order Hold (ZOH) to keep control inputs constant. This method requires that the controller works continuously during each sampling period, which may result in computational overload. In that regard, Liu et al. [33] proposed a Pulse-Modulated Intermittent Control (PMIC) solution that unifies impulsive and sampled control. PMIC can adapt the sampling period by defining a proper pulse function so that controllers only need to work during a fraction of each sampling period.

2.3.7 Challenges & Objectives

In brief, Recchiuto and Sgorbissa [4] summarized the swarm control challenge as a certain set of expected characteristics to achieve :

- Cooperative and distributed architecture: task load needs to be shared among all the swarm agents available. Note that this kind of architecture is more resilient than a centralized one (e.g., a BS that controls each UAV independently);
- Modularity and scalability: the agents have to communicate with each other to achieve the mission in a distributed manner. Nevertheless, the dependencies must be contained to a minimum level to ensure a maintainable and scalable system;

- Robustness and fault tolerance: the MAS must be robust to single agent faults. Moreover, it should be able to endure communication errors or external disturbances;
- Real-time and efficiency: the MAS control problem involves computation, communication, and control. Therefore, we need to pay close attention to how digitization may affect real-time performance of the system.

2.4 Path Planning

The efficiency of a search strategy strictly depends on how the UAVs compute their corresponding flying paths. Therefore, some path planning techniques and solutions have to be reviewed as well. In SAR applications, path planning should optimize the coverage of the area of interest. Also, considering the MAS context, path planning has to ensure collision and obstacle avoidance. Thus, coverage and collisions/obstacles avoidance will be considered here as the two main objectives of path planning.

2.4.1 Strategies

We distinguish two complementary strategies for path planning. Indeed, path planning happens in two different forms. On the one hand, we have *offline path planning*, which consists in designing the main path before take off, i.e., during the *assess* phase. On the other hand, we consider *online path planning* according to which flying paths are computed during the flight, essentially during the *search* phase. *Online path planning* deals with on-the-fly optimization of the path and collision/obstacle avoidance. Note that *online path planning* is subject to strong real-time constraints and complexity limitations due to poor embedded computation power on the UAVs. At the opposite, *offline path planning* does not suffer such constraints but may rely on inaccurate a priori information. Therefore, an optimal path planning system should combine these two strategies.

2.4.2 Algorithms

Avellar et al. [8] studied the problem of maximum coverage of ground areas using UAV swarms. In the first approach, they introduced some basic geometric considerations such as *optimal sweep direction*. We consider a rectangular area to be covered and assume that we use a sweeping pattern to explore the area. Then, while the two dimensions of the area are different, the sweep direction determines the number of turns required to cover the area. Furthermore, we know that the UAVs sensing capability is degraded during a turn so that

such a maneuver represents a certain loss of time and energy. Therefore, the *optimal sweep direction* is the sweep direction that reduces the number of turns. Note that this is especially true for fixed-wing UAVs but is still relevant for quadrotors. Then, they solved the Vehicle Routing Problem (VRP) using graph theory to model the path for a unique UAV to cover the area and Mixed Integer Linear Program (MILP) to split the path between a team of UAVs. They even manage to find the optimal number of UAVs to complete the mission by considering the maximum flight time and the setup time of UAVs.

A widely used method for path planning is the APF technique. We already introduced this technique for collision avoidance in the formation control problem. Let consider here a more global definition of APF for path planning applications. Let us suppose that the target to be reached by the agent has been characterized by a negative potential field, while all areas or objects to avoid by positive potential fields. Then, the planned path slides along a valley of the potential field toward its minimum, i.e., the target. This method provides fast convergence with a relatively simple implementation. Unfortunately, like said previously, this solution may fail to perform in presence of *local minima* in the potential field. Barraquand et al. [34] managed to avoid the *local minima* problem by constructing proper potential field using numerical techniques and *bitmap* representation.

Path planning can also be considered through the optimization prism. Mellinger et al. [35] formulate the path planning problem (i.e., optimal trajectory generation and collision/obstacle avoidance) using Mixed-Integer Quadratic Programs (MIQPs). Relying on *differential flatness* of quadrotors dynamics, the optimal trajectory generation is achieved using quadratic programs while collision avoidance is ensured by integer constraints, hence the MIQP. Even though this approach enables aggressive trajectories, it cannot a priori handle decentralized architecture and is not able to deal with a large swarm or too cluttered environments.

Last but not least, learning-based methods have also been applied to path planning. Zhang et al. [36] successfully combined the Model Predictive Control (MPC) technique with Reinforcement Learning (RL) to develop a computationally efficient controller handling obstacle avoidance. As MPCs represents a very high computational load, the technique cannot be applied right away on quadrotors. Instead, the team had the idea to train a *neural network* to reproduce MPCs behavior at a much lower cost. In order to do so, they used a "remote" MPC supported by BS and external sensors such as a Vicon system to control the quadrotor. Then they fed the Neural Network (NN) with the collected flight data. This approach presents good results in simulation, but unfortunately, no experiment has been realized. Also, NNs cannot give any mathematical proof or guaranty on stability, therefore this method may not be applicable to critical applications like SAR.

One could also consider the nature of the trajectory generated. Indeed, the simplest trajectory could be defined by a waypoint set while a more complex one may include not only position but also velocity, acceleration, and even jerk data. Recall that jerk is the time derivative of acceleration. In that regard, Conover [37] developed a jerk-augmented Linear–Quadratic Regulator (LQR) that would track jerk-augmented trajectories derived from high-order smooth polynomial position trajectories. This kind of enriched path planning, combined with a suitable controller, enables a more precise tracking and aggressive maneuvers.

In order to better deal with each different situation, the path planner could feature different flight modes. For instance, Wang et al. [27] proposed to evaluate the risk of collision to switch the path generation algorithm accordingly: choosing a more conservative trajectory planning when the risk is high and using a more efficient one when the risk is low. Also, Faessler et al. [38] developed an emergency-landing procedure that relies only on the most reliable sensors (e.g., Inertial Measurement Unit (IMU)) to avoid damaging the quadrotors when other sensing systems (e.g., Global Positioning System (GPS), odometry, etc.) fail, especially in the experimental phase. This mode enhances the robustness of the overall system significantly.

Table 2.1 Path planning solutions

	Pros	**Cons**	**Reference**
Sweep + MILP	Simple design and implementation. Optimal swarm size. Include setup time.	No prioritization. No online updates. Irrelevant for indoor applications.	Avellar et al. [8]
APF + Bitmap	Simple design and implementation. Fast convergence.	Local minima issue.	Barraquand et al. [34]
MIQP	Optimal trajectories. Aggressive maneuvers.	Complex design and implementation. Centralized solution.	Mellinger et al. [35]
MPCs + RL	Low computation cost. High robustness.	Complex design. No theoretical proof.	Zhang et al. [36]

2.5 Tracking

Designing efficient trajectories is not enough. We still have to ensure that our quadrotor is able to follow a moving target, hence to solve the tracking problem. Note that we already discussed the control and decision-making of the whole by assuming, in most cases, the presence of individual lower-level controllers. Here we will deal with those lower-level controllers. Although, the main objective of such controllers is to ensure a minimal tracking error, other desirable capabilities have to be envisioned. For instance, high convergence speed, fault-tolerant control or even fault recovery. Also, the tracking problem features some important constraints that should be modeled or at least considered prior to synthesis. Among them, let us list actuator saturation, payload change, transmission delays or wind disturbances.

2.5.1 Models

The first step toward controller design is the dynamic model of the open-loop system. Depending on targeted applications and performance, we will prefer either one reduced simplified system model or, at the opposite, a high-fidelity model. Classic quadrotor models most likely belong to the first category. Indeed, an overwhelming majority of researchers chose to simplify the quadrotor model and get rid of its nonlinearities by linearizing it at hovering point. This enables a quite easy synthesis as it decouples the outputs of the model. But the linearization hypothesis only holds in a close neighborhood of the hovering point such that aggressive maneuvers can only be achieved using multiple linearized models and a *sequential composition controller* [37]. The opposite extreme strategy consists of using the full-order nonlinear model. This may lead to strenuous control synthesis, high complexity design, which is not desirable. Thus, we will prefer a reduced-order nonlinear system obtained by neglecting some minor aerodynamic effects and actuators dynamics. Using such a model, several researchers have been able to design nonlinear controllers that achieve at least Lyapunov Asymptotic Stability (AS). In addition, Miao et al. [39] noticed that the reduced dynamics could be put under *strict feedback form* which implies that Backstepping Control (BSC) synthesis can be applied right away without much effort.

Note that the choice of the body frame of reference of the quadrotor influences the form of the dynamics equation. Indeed, two different configurations have been used by researchers (Figure 2.3) :

- the *plus* "+" configuration with pitch and roll axes aligned with the quadrotor's arms;
- the *cross* "×" configuration with pitch and roll axes shifted by 45° from the arms.

The yaw axis traditionally points toward the belly of the quadrotor. Clearly, the + configuration eases dynamics modelling and somehow simplifies the dynamic equations. Nevertheless, one would prefer the × configuration because it easily allows to place a camera pointed forward (i.e, along x/roll axis) without incurring in obstructions from the airframe.

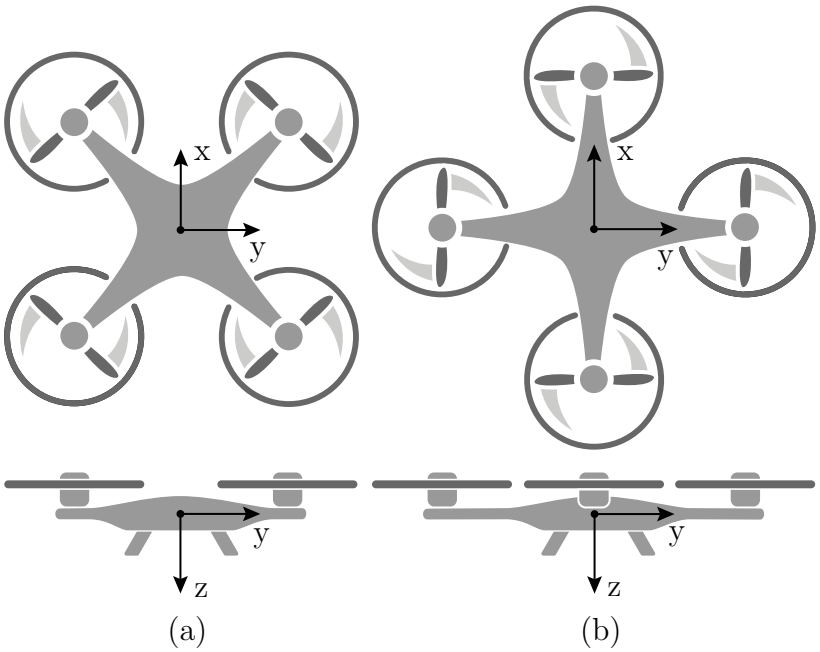


Figure 2.3 (a) "×" configuration. (b) "+" configuration.

A determinant point in the quadrotor kinematic and dynamic modelling is the orientation representation. Traditionally, we define the orientation of flying objects (e.g., plane, helicopter, satellites, ...) using the famous *Euler angles*: roll, pitch, and yaw. Unfortunately, that representation suffers a singularity. Indeed, when the axes of two of the three possible rotations become co-linear, it follows the loss of one degree of freedom in rotation: we call that phenomenon the *gimbal lock*. That said, Euler angles may be convenient for understanding and communication of results but it may bring some problems when it comes to achieving acrobatics. Instead, we could simply use rotation matrices, also called DCMs or matrices of the $SO(3)$ group. This representation does not have any singularity but suffers another major drawback. Indeed, controlling a DCM would mean working with six fully coupled state variables, which could represent a large computation cost. Hopefully, a last system for representing rotations was introduced by Hamilton [40] : the *unit quaternion*. That representation does not suffer any singularity. In addition, it is more compact and faster to compute than DCM or even Euler angles. Thus, we will prefer unit quaternions to other rotation representations for the model and will use Euler angles to communicate our

results.

2.5.2 Structure

Although we are aware that several control structures exist for quadrotor control, let us only discuss the most widely applied (Figure 2.4). First, notice that to generate a linear displacement of the quadrotor in the horizontal plane, we have to tilt it in the chosen direction. Therefore we will be considering two nested control loops. The inner-loop will have to stabilize the quadrotor at hovering and ensure attitude and altitude control. The outer-loop will generate altitude and attitude control inputs for the inner loop in order to track velocity or/and position references from the path planner. At a lower level, we could also consider an innermost loop to deal with motors dynamics [41].

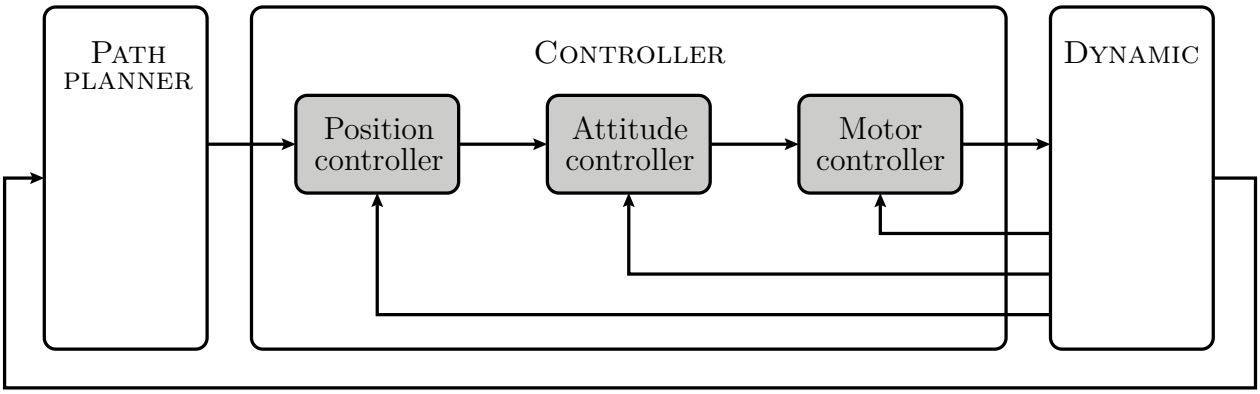


Figure 2.4 Standard control structure

2.5.3 Design

Recall that the linearized model presents decoupled inputs-outputs. That said, a large variety of linear controllers have been successfully applied to quadrotor control. The structure of the control stays roughly the same, namely state feedback with integral action (i.e., Proportional-Integral-Derivative (PID) control). The synthesis, however, can be conducted through different methods. Clearly, *pole assignment* or, in a more complete fashion, *eigenstructures assignment* can be applied but may require some experience to be properly tuned. Also, Linear-Quadratic-Gaussian (LQG) method has been widely used to design optimal controllers.

Unfortunately, linear control can only achieve local stability. Hence, we have to consider nonlinear control to ensure global stability. For instance, Jia et al. [42] introduced a hybrid nonlinear control solution that combines BSC with SMC. The integral action is added to the control loop in order to ensure zero steady state error. The proposed controller does not

present the chattering phenomenon usually observed with SMC and seems to be more robust than other traditional controllers (e.g., PID, LQR, BSC).

Still, the control design listed above does not consider external disturbances or model uncertainties at synthesis level. Therefore, some researchers have turned to robust control design. For instance, Jasim and Gu [25] used the H_∞-synthesis and μ-synthesis method to design a suboptimal robust controller that outperforms a back-stepping controller in presence of disturbances and uncertainties. In brief, the H_∞-synthesis method translates the control problem expressed in terms of frequential performances (i.e., frequential weighting functions) into a mathematical optimization problem. Then, it solves the optimization problem using a numerical method such as the *D-K iteration procedure*. Thus, H_∞ controllers can satisfy the stabilization requirement along with guaranteed performances. Nevertheless, proper frequential weighting functions can be hard to find. Ataei et al. [43] proposed an MPCs based solution to design a quadrotor robust controller. They notably introduced a relaxation technique to reduce the computational cost of LMIs solving processed by the controller in presence of a large number of uncertainties and disturbances. Dabin [44] implemented a linear Active Disturbance Rejection Control (ADRC) on a mini-quadrotor in order to ensure proper trajectory tracking in presence of wind. The ADRC idea is to use an uncertainty/disturbance estimator to adapt the linear feedback control accordingly. Ranjbaran and Khorasani [45] developed an adaptive controller to ensure global stability in presence of actuators' Loss Of Efficiency (LOE). To do so, they designed a parameter estimator along with update law to adapt control accordingly. Note that the quadrotor model used is nonlinear but the LOE model developed is linear.

2.6 Estimation

A major issue in operating quadrotor is being capable of retrieving accurate data about its states and parameters from sensor measurements to feed the control loop. First, we need to estimate the orientation and rotation rate of the quadrotor. This information is critical for the controller in order to stabilize the hovering equilibrium. Therefore, orientation estimation needs to be highly reliable and will be treated individually for each agent. Then, we need to compute the quadrotor's position, translational velocity, acceleration and even jerk [37] in order to achieve trajectory tracking. Note that that information could be retrieved in a distributed fashion as each agent may be able to collect information on position and velocity of its neighbors. As for swarm control, a distributed solution will present better computational efficiency and higher robustness in case of sensor failure. At agent level, we may also need to estimate some parameters such as sensor biases or even actuator efficiency to enable fault

Table 2.2 Tracking solutions

	Pros	**Cons**	**Reference**
Linear ctrl.	Simple design and implementation.	Only locally stable.	Conover [37]
Adaptive ctrl.	Simple design. Good robustness.	Only locally stable. High computation cost.	Dabin [44] Ranjbaran and Khorasani [45]
BSC + SMC	High robustness.	Elaborate design.	Jia et al. [42]
H_∞ ctrl.	Guaranteed robust performances.	Hard to tune.	Jasim and Gu [25]
MPCs	High robustness.	Complex design. High computation cost.	Ataei et al. [43]

detection. At the swarm level, it would be interesting to estimate other parameters such as transmission delays or network topology [46].

Nevertheless, states and parameters estimation in quadrotors applications could be tricky. Indeed, complete estimator solutions have been studied and developed for years now and are widely used on board of commercial aircraft or satellites. But low-cost quadrotors bring new challenges for estimator design. In fact, cheap sensors suffer high noise levels, biases and scale factors which lead to serious issues estimating attitude. Moreover, quadrotors' kinematics are highly nonlinear such that filters used in classical navigation systems may fail to provide accurate attitude estimates. Then, cost and weight constraints also limit the computational power on board. In addition, we have to consider that some measurements could be temporarily compromised or unavailable [39], especially for indoor applications. For instance, GPS signal will not be available inside a building. Also, high luminosity variations at the door crossing may result in failure of visual-based estimations. At last, some estimation systems may require burdensome offline calibration prior to take-off [38] and this has to be considered in the performance evaluation of the estimation solution.

2.6.1 Techniques & Sensors

Navigation techniques can be categorized into two main classes. On the one hand, *position fixing* techniques rely on external sources of information to determine the absolute position of the system. Those techniques provide accurate estimations but depend entirely on the availability of external sources, which severely degrades the robustness of the solution. On the other hand, *dead reckoning* techniques measure both distance and direction traveled from a start position or configuration, by some form of motion sensing (velocity and/or acceleration measurements). These techniques only provide positioning relative to the starting point. Moreover, the position error obtained grows with time because the successive distance and direction measurement errors accumulate (i.e., error integration over time). Nevertheless, *dead reckoning* techniques present some advantages with respect to *position fixing*. Indeed, they are self-contained, fully autonomous and usually perform at a higher rate. Also, they provide direct acceleration and velocity estimates, which are often necessary to the controller. In contrast, *position fixing* techniques would have to differentiate the position signal to retrieve velocity and acceleration, which is very sensitive to noise. An efficient estimation solution will often combine both *position fixing* techniques and *dead reckoning* techniques through sensor fusion algorithms to obtain an accurate and reliable estimate.

Position Fixing

A famous position fixing solution is the Global Navigation Satellite Systems (GNSSs) such as the American GPS. An embedded GPS receiver can determine its own position using signals emitted from satellites to measure the distance from those satellites and triangulate its position (i.e., *ranging* solution). GPS accuracy depends on the number of satellites visible by the receiver. Hence, GPS may provide very good performances for outside applications but cannot be used for indoor applications.

Other *ranging and bearing* solutions such as Sound Navigation and Ranging (SONAR), Radio Direction and Ranging (RADAR), and Light Detection and Ranging (LIDAR) can also be used for position fixing assuming that we have some sort of map of the environment. Those present quite interesting performance and have been widely used for robot navigation. But they may be hard to implement on quadrotors due to strict payload limitations.

More basic position fixing solutions are available for quadrotors. For instance, a barometer is a simple way to determine the altitude of the quadrotors using the atmospheric pressure model. Nevertheless, the accuracy of such a solution may not be relevant for indoor applications.

Absolute orientation can be estimated from observation of Earth gravity vector measured by accelerometers and the geomagnetic field vector measured by magnetometers. Unfortunately, this absolute estimate accuracy will greatly depend on body acceleration and magnetic perturbations. Also, we need to know where our quadrotors are on Earth as the geomagnetic vector is not the same everywhere.

More recently, *pattern recognition* techniques have been developed to achieve position fixing using cameras. Even though those demonstrate good performance, especially for indoor navigation, they may require too much computation to be implemented right away on quadrotors. Note that different types of cameras are available for such applications. Monocular cameras are the most basic solution and will only provide 2D information. Binocular cameras and RGB-D cameras, however, can provide information about image depth but often require a higher computation power.

It is also important to mention *motion capture* systems such as Vicon as they are widely used for experimental studies. Those systems use several cameras installed in the environment to track some markers attached to the quadrotor. Then, it computes the states of the quadrotors with such high fidelity that it is often used as ground truth for experimental studies.

Dead Reckoning

A classical dead reckoning solution would be *inertial navigation*. This technique involves inertial sensors such as accelerometers and gyroscopes that respectively measure linear acceleration and rotational acceleration. In theory, we could easily compute the velocity, the angular rate, the position and the orientation of the quadrotor by successive integration assuming that we know what is the initial state of our quadrotor. But like we said before, the sensors may suffer some bias, scale factor and high-level noise which imply important errors in the estimate after integration. Moreover, the initial state of the quadrotor may not be available.

More recently, *visual odometry* techniques have been developed. Classic *odometry* consists in using wheel sensors (encoders) to measure and then integrate a ground vehicle's velocity. In the same fashion, *visual odometry* uses optical flow captured by cameras to estimate the quadrotor velocity and integrate it to get the position. Once again, those techniques suffer the dead reckoning usual drawbacks, namely error integration and tbe need for initial state knowledge.

2.6.2 Data Fusion

We have seen that either position fixing or dead reckoning cannot possibly be used right away for navigation. That is why we want to perform *data fusion* (also called *sensor fusion*). Indeed, if we manage to design a filter that properly combines the benefit of position fixing or dead reckoning, we might be able to compute an accurate estimate. We consider orientation estimation using an IMU [47]. Most IMUs embed a three-axis magnetometer along with a three-axis gyroscope and a three-axis accelerometer. All of them are commonly built using Micro-ElectroMechanical Systems (MEMS). Then, the basic idea behind data fusion for IMU is to mitigate the drift error due to gyro measurement integration with an absolute estimate from accelerometers and magnetometers when available. Therefore, we finally obtain an accurate measurement of orientation at a high rate update. The same result on the absolute position can be achieved using cameras by combining visual odometry and pattern recognition (i.e., *loop closure*). Remark that the filter that will perform data fusion will have to deal with signals having different sampling periods and possibly suffering from delays [48].

Clearly, the traditional Luenberger observer does not enable sensor fusion and cannot be applied here due to high-level sensor noise. Therefore, a straightforward approach for sensor fusion is to use deterministic linear Complementary Filters (CFs) to combine sensors measurements in the frequency domain [49]. An effective alternative is to use extended stochastic linear estimation techniques derived from the famous *Kalman Filter* [50]. For instance, the Extended Kalman Filter (EKF) has been extensively applied for orientation estimation [51]. Although these methods provide effective orientation estimations and have been widely applied for quadrotor orientation estimation, highly nonlinear kinematics of quadrotors could lead them to fail. Julier et al. [52] proposed a new nonlinear filter to accommodate highly nonlinear systems. By propagating a sampled estimate distribution through the real nonlinear kinematics, this new filter gets rid of linearization errors of the EKF. What will be called the Unscented Kalman Filter (UKF) has since been proved to be a good alternative to the EKF for highly nonlinear systems even though they present a slightly heavier computation cost [53]. Note that other stochastic nonlinear filters exist such as the Particle Filter (PF) but their computation cost makes them irrelevant to quadrotor implementations. Hamel and Mahony [47] proposed a nonlinear CF exploiting the structure of the rotation matrix in order to compute high-quality orientation estimates from typical low-cost IMU. Their solution has been shown to perform well even if only one vector among the gravity vector and the geomagnetic vector is available.

Recently, position estimation has been addressed through the Simultaneous Localization and Mapping (SLAM) framework [54]. SLAM tackles the estimation problem by constructing and updating a map of the environment while simultaneously keeping track of an agent's location within it. Note that this kind of solution is of great interest when it comes to SAR applications.

Tomic et al. [48] developed a complete solution including hardware and software design for urban SAR powered by autonomous quadrotors. They proposed a light version of existing SLAM solutions to be implemented on a quadrotor. The odometry data is traditionally fused with the IMU data using an EKF. But no geometric map is built. Instead, the correction for drift errors is achieved by recognizing known landmarks in the environment. Moreover, the odometry system is fed by a binocular camera and a LIDAR which enable the system to perform both indoor and outdoor. Indeed, an indoor environment consists of walls that can easily be detected by a LIDAR. In contrast, lack of luminosity in such an environment could cause the camera odometry to fail. Conversely, sunlit outdoor environments contain light in the part of the spectrum that coincides with that used by infrared laser scanners which may prevent the LIDAR odometry from providing a good estimate. But such environments have many natural features and good lighting conditions, which make them perfectly suited for visual odometry systems.

In contrast, Verykokou et al. [55] only addressed the mapping problem. Indeed, in SAR applications, having a complete and accurate map of the affected area could significantly improve disaster response. A fast 3D environment modeling using images from quadrotors and Structure From Motion (SFM) open-source algorithms is presented. In brief, SFM process consists of three steps. First, features are extracted in each image using a feature point detector and descriptor. One of the most widely used feature detectors is the Scale-Invariant Feature Transform (SIFT) [56], which are scale and rotation invariant and robust to changes in illumination. Then, the descriptors are matched. One of the matching algorithms that track features from one image to another is the *Lukas–Kanade tracker* [57]. It is common then to filter the matches to remove incorrect matches. The Random Sample Consensus (RANSAC) algorithm is usually chosen to reject the outliers [58]. After that, a dense point cloud is obtained using a Dense Image Matching (DIM) algorithm. And finally, the dense cloud is transformed into a mesh representation and the texture of the derived surface model is applied to obtain a complete 3D map.

Faessler et al. [38] proposed a low-cost solution for the SLAM problem using a micro-quadrotor. The solution developed features only two sensor units: one IMU and one monocular camera looking downward. Visual odometry is performed using the Semidirect Visual

Odometry (SVO) algorithm introduced by Forster et al. [59]. Data fusion between IMU and odometry is processed using the EKF-based multi-sensor fusion algorithm designed by Lynen et al. [60]. From the images streamed by the quadrotor, a dense 3D map is constructed in real time on the ground station using the Regularized Monocular Depth (REMODE) algorithm previously developed in Pizzoli et al. [61]. This system has been shown to perform well both in indoor and outdoor experiments.

Table 2.3 Sensing solutions

	Pros	**Cons**
GNSS	Absolute positioning in global frame.	Not available indoor.
RADAR **LIDAR**	High range. High accuracy	Expensive. High computation cost. Heavy and large.
Mono-Camera	Low cost. Low computation load	No depth information.
Stereo-Camera **RGB-D Cam.**	Depth information	Expensive. High computation load. Heavy and large.
IMU	Low cost. Gives absolute orientation and relative position. Fully self-contained	Need for global position. Poor accuracy. High-level noise.

Table 2.4 Data fusion solutions

	Pros	**Cons**	**Reference**
Linear CF	Simple implementation.	Hard to tune. Does not handle strong nonlinearities.	Zimmermann and Sulzer [49]
EKF	Simple design.	Does not handle strong nonlinearities.	Sabatini [51]
UKF & UKF	Handle strong nonlinerities.	Complex implementation. High computation load.	Crassidis and Markley [53]
Nonlinear CF	Simple implementation.	Hard to tune.	Hamel and Mahony [47]
Full SLAM	Accurate dense 3D map.	Centralized remote computation.	Faessler et al. [38]
Light SLAM	Reduced computation requirements.	No geometric map.	Tomic et al. [48]
SFM	Accurate dense 3D map.	Centralized remote computation.	Verykokou et al. [55]

CHAPTER 3 MODEL & CONTROL

3.1 Model

3.1.1 System Description

A quadrotor, also called quadcopter, is a UAV lifted and propelled by four horizontal rotors attached to a rigid cross airframe. Unlike helicopters whose main rotor axis can tilt to generate the desired thrust, quadrotor's control is achieved by differential control of the thrust generated by each rotor. Pitch, roll and altitude controls are then straightforward to understand. To ensure no resulting moment from rotors at hovering, rotor i rotates Clockwise (CW) if i is odd and Counterclockwise (CCW) if i is even (Figure 3.1). Thus, yaw control is managed by introducing a difference in the average speed of the two CW/CCW pairs of rotors. Note that the system is underactuated because only three rotations and one translation are actuated, leaving two remaining Degrees of Freedom, namely the translational motion in the horizontal plane.

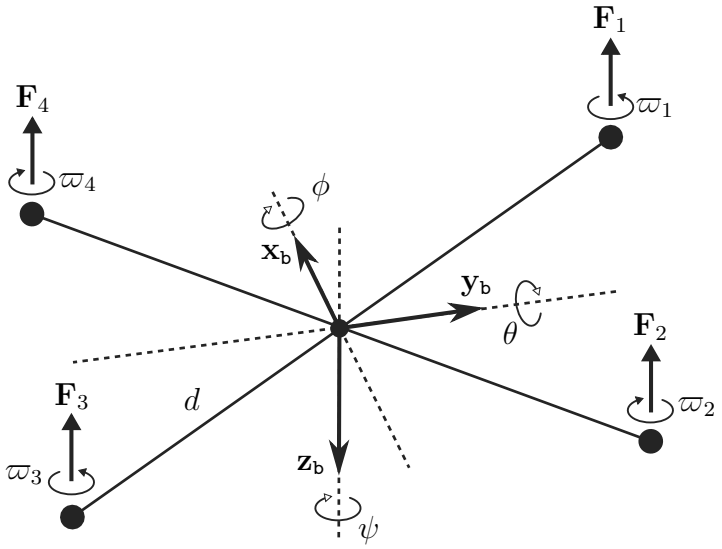


Figure 3.1 Quadrotor & body frame in "×" configuration

3.1.2 Frame of References

Let $\{\mathtt{o}\}$ denote the North-East-Down (NED) frame assumed inertial with unit vectors along the axes denoted by $\{\mathbf{x}_\mathtt{o}, \mathbf{y}_\mathtt{o}, \mathbf{z}_\mathtt{o}\}$. Let $\{\mathtt{b}\}$ be the body frame attached to the quadrotor

gravity center with unit vectors $\{\mathbf{x}_\mathtt{b}, \mathbf{y}_\mathtt{b}, \mathbf{z}_\mathtt{b}\}$. The transformation of the frame $\{\mathtt{o}\}$ into frame $\{\mathtt{b}\}$ presented on Figure 3.2 can be described by a ZYX Euler rotation sequence given by :

1. A *yaw* rotation $\psi \in [-\pi, \pi]$ about the original z-axis.
2. A *pitch* rotation $\theta \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ about the intermediate y-axis.
3. A *roll* rotation $\phi \in [-\pi, \pi]$ about the transformed x-axis.

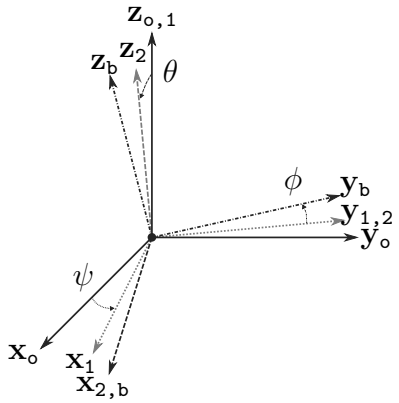


Figure 3.2 Euler angles & reference frames

Then the orientation of the body frame $\{\mathtt{b}\}$ with respect to frame $\{\mathtt{o}\}$ is given by the rotation matrix $\mathbf{R}_\mathtt{b}^\mathtt{o} \in \mathcal{SO}(3)$ such that we have $\mathbf{R}_\mathtt{b}^\mathtt{o} = \begin{bmatrix}\mathbf{x}_\mathtt{b}^\mathtt{o} & \mathbf{y}_\mathtt{b}^\mathtt{o} & \mathbf{z}_\mathtt{b}^\mathtt{o}\end{bmatrix}$. From the ZYX Euler angles sequence, we can compute

$$\mathbf{R}_\mathtt{b}^\mathtt{o} = \begin{bmatrix} \cos\theta\cos\psi & -\cos\phi\sin\psi + \sin\phi\sin\theta\cos\psi & \sin\phi\sin\psi + \cos\phi\sin\theta\cos\psi \\ \cos\theta\sin\psi & \cos\phi\cos\psi + \sin\phi\sin\theta\sin\psi & -\sin\phi\cos\psi + \cos\phi\sin\theta\sin\psi \\ -\sin\theta & \sin\phi\cos\theta & \cos\phi\cos\theta \end{bmatrix}. \quad (3.1)$$

Alternatively, if $\mathbf{R}_\mathtt{b}^\mathtt{o}[3,1] \neq \pm 1$, we have

$$\psi = \operatorname{atan2}\left(\mathbf{R}_\mathtt{b}^\mathtt{o}[2,1], \mathbf{R}_\mathtt{b}^\mathtt{o}[1,1]\right), \ \theta = -\arcsin\left(\mathbf{R}_\mathtt{b}^\mathtt{o}[3,2]\right), \ \phi = \operatorname{atan2}\left(\mathbf{R}_\mathtt{b}^\mathtt{o}[3,2], \mathbf{R}_\mathtt{b}^\mathtt{o}[3,3]\right). \quad (3.2)$$

And for $\mathbf{R}_\mathtt{b}^\mathtt{o}[3,1] = \pm 1$ (i.e. in case of gimbal lock), we can define

$$\psi = \operatorname{atan2}\left(-\mathbf{R}_\mathtt{b}^\mathtt{o}[3,1]\mathbf{R}_\mathtt{b}^\mathtt{o}[1,2], -\mathbf{R}_\mathtt{b}^\mathtt{o}[3,1]\mathbf{R}_\mathtt{b}^\mathtt{o}[1,3]\right), \qquad \theta = -\mathbf{R}_\mathtt{b}^\mathtt{o}[3,1]\frac{\pi}{2}, \qquad \phi = 0. \quad (3.3)$$

3.1.3 Rigid Body Dynamics

The rigid body equations of motion of the airframe [62] are given by :

$$\dot{\mathbf{r}} = \mathbf{v}, \quad (3.4)$$

$$m\dot{\mathbf{v}} = mg\mathbf{z}_\mathtt{o} + \mathbf{R}_\mathtt{b}^\mathtt{o}\mathbf{F}, \quad (3.5)$$

$$\dot{\mathbf{R}}_\mathtt{b}^\mathtt{o} = \mathbf{R}_\mathtt{b}^\mathtt{o}\left[\boldsymbol{\omega}_{\mathtt{b}/\mathtt{o}_\times}\right], \quad (3.6)$$

$$\mathbf{J}\dot{\boldsymbol{\omega}}_{\mathtt{b}/\mathtt{o}} = -\boldsymbol{\omega}_{\mathtt{b}/\mathtt{o}} \times \mathbf{J}\boldsymbol{\omega}_{\mathtt{b}/\mathtt{o}} + \mathbf{M}, \quad (3.7)$$

where $\mathbf{r} \in \mathbb{R}^3$ is the position vector of the quadrotor in frame $\{\mathtt{o}\}$, $\mathbf{v} \in \mathbb{R}^3$ is the linear velocity in frame $\{\mathtt{o}\}$ with respect to $\{\mathtt{o}\}$, $\boldsymbol{\omega}_{\mathtt{b}/\mathtt{o}} \in \mathbb{R}^3$ is the angular velocity in frame $\{\mathtt{b}\}$ with respect to $\{\mathtt{o}\}$. The constants g, m and $\mathbf{J} \in \mathbb{R}^{3\times3}$ are respectively gravity acceleration, quadrotor mass and inertia matrix. The vectors $\mathbf{F} \in \mathbb{R}^3$ and $\mathbf{M} \in \mathbb{R}^3$ here expressed in frame $\{\mathtt{b}\}$, are respectively non-conservative forces and moments applied on the quadrotor, so essentially thrust and turning torque from rotors.

The notation $[\mathbf{a}_\times]$ denotes the skew-symmetric matrix, such that $[\mathbf{a}_\times]\,\mathbf{b} = \mathbf{a}\times\mathbf{b}$ for the vector cross product $\times$ and any vector $\mathbf{a}, \mathbf{b} \in \mathbb{R}^3$. To be more specific, for $\boldsymbol{\omega}_{\mathtt{b}/\mathtt{o}} = \begin{bmatrix}\omega_1 & \omega_2 & \omega_3\end{bmatrix}^\top$, the related skew-symmetric matrix is

$$[\boldsymbol{\omega}_\times] = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix}. \tag{3.8}$$

3.1.4 Dominant Aerodynamics

We will only provide here a simple model for aerodynamics involved in quadrotors to get a basic understanding of how it works. There are many aerodynamic phenomena (e.g., blade flapping [63]) associated with any rotorcraft that will not be discussed here because assumed to be negligible under normal operation of the quadrotor.

The force $\mathbf{F}$ on the quadrotor is the sum of thrusts developed by the four rotors

$$\mathbf{F} = \sum_{i=1}^{4} \mathbf{F}_i, \tag{3.9}$$

where $\mathbf{F}_i = -k_T\varpi_i^2\mathbf{z}_\mathtt{b}$ with k_T the blades' bearing parameter and ϖ_i the rotor i angular rate. The moment $\mathbf{M}$ on the airframe is the sum of a moment $\mathbf{P} \in \mathbb{R}^3$ induced by the force discussed above, a moment $\mathbf{Q} \in \mathbb{R}^3$ induced by the rotation of rotors and a gyroscopic moment $\mathbf{G} \in \mathbb{R}^3$ resulting from the rotations of the frame and the rotors

$$\mathbf{M} = \mathbf{P} + \mathbf{Q} + \mathbf{G}, \tag{3.10}$$

$$\text{with} \qquad \mathbf{P} = \sum_{i=1}^{4} \mathbf{r}_i \times \mathbf{F}_i, \qquad \mathbf{Q} = \sum_{i=1}^{4} (-1)^{i+1}\boldsymbol{\tau}_i, \qquad \mathbf{G} = \sum_{i=1}^{4} (-1)^{i}\boldsymbol{\sigma}_i \times \boldsymbol{\omega}_{\mathtt{b}/\mathtt{o}}, \tag{3.11}$$

where $\mathbf{r}_i \in \mathbb{R}^3$ is the vector from gravity center to rotor i, $\boldsymbol{\tau}_i = (k_D\varpi_i^2 + J_m\dot{\varpi}_i)\mathbf{z}_\mathtt{b}$ and $\boldsymbol{\sigma}_i = J_m\varpi_i\mathbf{z}_\mathtt{b}$ with k_D the drag parameter of blades and J_m the inertial momentum of rotors.

3.1.5 Control Inputs Decoupling

Since the motor dynamics are fast with respect to the rigid body dynamics and aerodynamics, we will assume that rotor speeds can be instantly achieved and will be considered as the true control inputs of the system. This also implies that the inertial momentum of rotors has to be negligible (i.e., $J_m \approx 0\,\mathrm{kg\,m^2}$). Assuming that all four rotors are at same distance d from the gravity center, then, considering an "×" configuration (Figure 3.1) we obtain

$$\mathbf{r}_1 = \frac{d}{\sqrt{2}}(\mathbf{x}_\mathtt{b} + \mathbf{y}_\mathtt{b}), \quad \mathbf{r}_2 = \frac{d}{\sqrt{2}}(\mathbf{x}_\mathtt{b} - \mathbf{y}_\mathtt{b}), \quad \mathbf{r}_3 = \frac{d}{\sqrt{2}}(-\mathbf{x}_\mathtt{b} - \mathbf{y}_\mathtt{b}), \quad \mathbf{r}_4 = \frac{d}{\sqrt{2}}(-\mathbf{x}_\mathtt{b} + \mathbf{y}_\mathtt{b}). \tag{3.12}$$

The expression of $\mathbf{F}$ and $\mathbf{M}$ in body frame are then:

$$\mathbf{F} = \begin{bmatrix} 0 \\ 0 \\ -T \end{bmatrix}^\mathtt{b} = \begin{bmatrix} 0 \\ 0 \\ -k_T(\varpi_1^2 + \varpi_2^2 + \varpi_3^2 + \varpi_4^2) \end{bmatrix}^\mathtt{b}, \tag{3.13}$$

$$\mathbf{M} = \begin{bmatrix} M_x \\ M_y \\ M_z \end{bmatrix}^\mathtt{b} = \begin{bmatrix} \frac{dk_T}{\sqrt{2}}\left(-\varpi_1^2 + \varpi_2^2 + \varpi_3^2 - \varpi_4^2\right) \\ \frac{dk_T}{\sqrt{2}}\left(\varpi_1^2 + \varpi_2^2 - \varpi_3^2 - \varpi_4^2\right) \\ k_D\left(\varpi_1^2 - \varpi_2^2 + \varpi_3^2 - \varpi_4^2\right) \end{bmatrix}^\mathtt{b}. \tag{3.14}$$

Let $\boldsymbol{\varpi} = \begin{bmatrix} \varpi_1^2 & \varpi_2^2 & \varpi_3^2 & \varpi_4^2 \end{bmatrix}^\top$ be the real control input vector and $\tilde{\mathbf{u}} = \begin{bmatrix} T & M_x & M_y & M_z \end{bmatrix}^\top$ be the substitute decoupled control input vector. Then, we define the invertible matrix $\mathbf{T}$

$$\tilde{\mathbf{u}} = \mathbf{T}\boldsymbol{\varpi}. \tag{3.15}$$

such that

$$\mathbf{T} = \begin{bmatrix} -k_T & -k_T & -k_T & -k_T \\ -\frac{dk_T}{\sqrt{2}} & \frac{dk_T}{\sqrt{2}} & \frac{dk_T}{\sqrt{2}} & -\frac{dk_T}{\sqrt{2}} \\ \frac{dk_T}{\sqrt{2}} & \frac{dk_T}{\sqrt{2}} & -\frac{dk_T}{\sqrt{2}} & -\frac{dk_T}{\sqrt{2}} \\ k_D & -k_D & k_D & -k_D \end{bmatrix} \quad \text{and} \quad \mathbf{T}^{-1} = \begin{bmatrix} -\frac{1}{4k_T} & -\frac{\sqrt{2}}{4dk_T} & \frac{\sqrt{2}}{4dk_T} & \frac{1}{4k_D} \\ -\frac{1}{4k_T} & \frac{\sqrt{2}}{4dk_T} & \frac{\sqrt{2}}{4dk_T} & -\frac{1}{4k_D} \\ -\frac{1}{4k_T} & \frac{\sqrt{2}}{4dk_T} & -\frac{\sqrt{2}}{4dk_T} & \frac{1}{4k_D} \\ -\frac{1}{4k_T} & -\frac{\sqrt{2}}{4dk_T} & -\frac{\sqrt{2}}{4dk_T} & -\frac{1}{4k_D} \end{bmatrix}.$$

That way, we can easily obtain the $\boldsymbol{\varpi}$ that generates the desired T and $\mathbf{M}$

$$\boldsymbol{\varpi} = \mathbf{T}^{-1}\tilde{\mathbf{u}}. \tag{3.16}$$

3.2 Control

3.2.1 Architecture

As observed in the literature and depicted in Figure 3.3, the controllers discussed in the following sections consist of two nested control loops. The inner-loop will ensure the control of the quadrotor's orientation: this is the attitude controller. The outer-loop will compute orientation reference inputs for the attitude controller to track velocity or position references. Such a control architecture can be applied to quadrotors because their rotational dynamics is assumed to be much faster than their translational dynamics. This is especially true in our application because we do not target aggressive behavior, but rather smooth motion. While the attitude controller will not be discussed in the following, several control laws (i.e., several flight modes) will be developed for the translational dynamics to generate different behaviors useful for formation control. Note that the simulations in this section are run in MATLAB Simulink 2019b.

3.2.2 Design Model

Low-level control of quadrotors is not the main interest of this work. Therefore, the quadrotor's rotational dynamics will be assumed fully and perfectly stabilized and controlled by an existing attitude controller. Note that this inner control loop can be implemented using simple linear controllers such as a Proportional-Derivative (PD) [64] or more specific non-linear controllers [65]. As a result, the control input considered for translational control will be the resultant thrust force in frame $\{\mathtt{o}\}$.

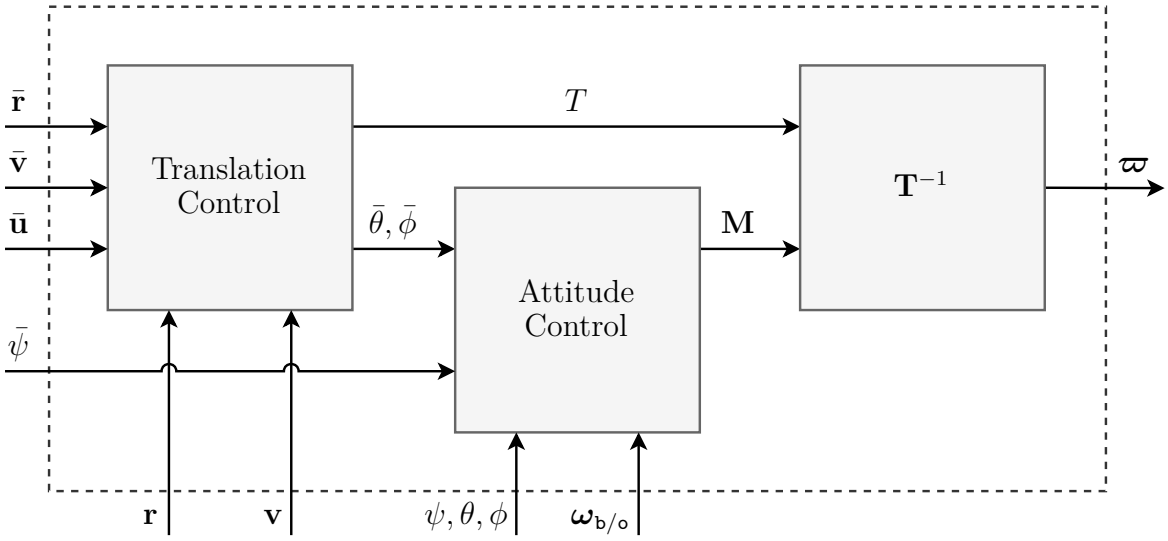


Figure 3.3 Control architecture

Under these assumptions, the quadrotor dynamics consist in three decoupled double integrators such that 3.4 and 3.5 simply become

$$\dot{\mathbf{r}} = \mathbf{v}, \tag{3.17}$$

$$\dot{\mathbf{v}} = \mathbf{u}, \tag{3.18}$$

with

$$\mathbf{r} = \begin{bmatrix} r_x & r_y & r_z \end{bmatrix}^\top, \quad \mathbf{v} = \begin{bmatrix} v_x & v_y & v_z \end{bmatrix}^\top, \quad \mathbf{u} = \begin{bmatrix} u_x & u_y & u_z \end{bmatrix}^\top = g\mathbf{z}_\mathfrak{o} + \frac{1}{m}\mathbf{R}_\mathfrak{b}^\mathfrak{o}\mathbf{F}, \tag{3.19}$$

where $\mathbf{u}$ is the control input vector of the translational subsystem to be controlled. Note that the state vector is assumed fully known at any time such that no output vector or function needs to be defined.

For completeness and practical implementation purposes, the relation between $\mathbf{u}$ the control inputs of the outer-loop and $\begin{bmatrix} \theta & \phi & T \end{bmatrix}^\top$ the reference inputs of the inner-loop is presented. Making small angles approximation for ϕ and θ and for any value of ψ, it can be obtain that

$$\begin{cases} u_x &= -g\left(\cos(\psi)\theta + \sin(\psi)\phi\right), \\ u_y &= -g\left(\sin(\psi)\theta - g\cos(\psi)\phi\right), \\ u_z &= g - \dfrac{1}{m}T, \end{cases} \Leftrightarrow \begin{cases} \theta &= -\dfrac{1}{g}\left(\cos(\psi)u_x + \sin(\psi)u_y\right), \\ \phi &= -\dfrac{1}{g}\left(\sin(\psi)u_x - \cos(\psi)u_y\right), \\ T &= m(g - u_z). \end{cases} \tag{3.20}$$

It worth pointing out that the dynamics along the three different directions are fully decoupled and strictly identical. Therefore, only one direction will be considered from now on. Then, the following discrete-time model can be derived from 3.17 and 3.18 using the ZOH method

$$r(k+1) = r(k) + T_s v(k) + \frac{T_s^2}{2}u(k), \tag{3.21}$$

$$v(k+1) = v(k) + T_s u(k), \tag{3.22}$$

where T_s is the controller sampling time and $|u(k)| < u_{max}$ is the saturated control input that accounts for the small angles approximation and the actuator limitations (i.e., maximal roll/pitch angles and maximal thrust). Note that the effects of quantization that occurs in digital control are ignored. This assumption holds as long as the values of the states r and v and the control input u remain in a certain interval and the Analog-Digital Converter (ADC) has sufficient resolution. In practice, this is verified because the r, v and u are respectively limited by battery life, aerodynamic performance and actuator physics.

3.2.3 Velocity Control

Synthesis

The goal of the velocity control law is to bring the error $e_v(k) = v(k) - \bar{v}(k)$ to zero where $\bar{v}$ is the desired velocity. In order to account for disturbances and model uncertainties, we introduce x_{Iv} the integral of the error e_v. Then, we propose the unbounded control law:

$$u_\infty(k) = -K_{Iv}x_{Iv}(k) - K_{Pv}v(k) + K_{Fv}\bar{v}(k), \tag{3.23}$$

with K_{Iv}, K_{Pv} and K_{Fv}, respectively the integral, the proportional and the feed forward gains. Figure 3.4 displays the block diagram of the velocity control.

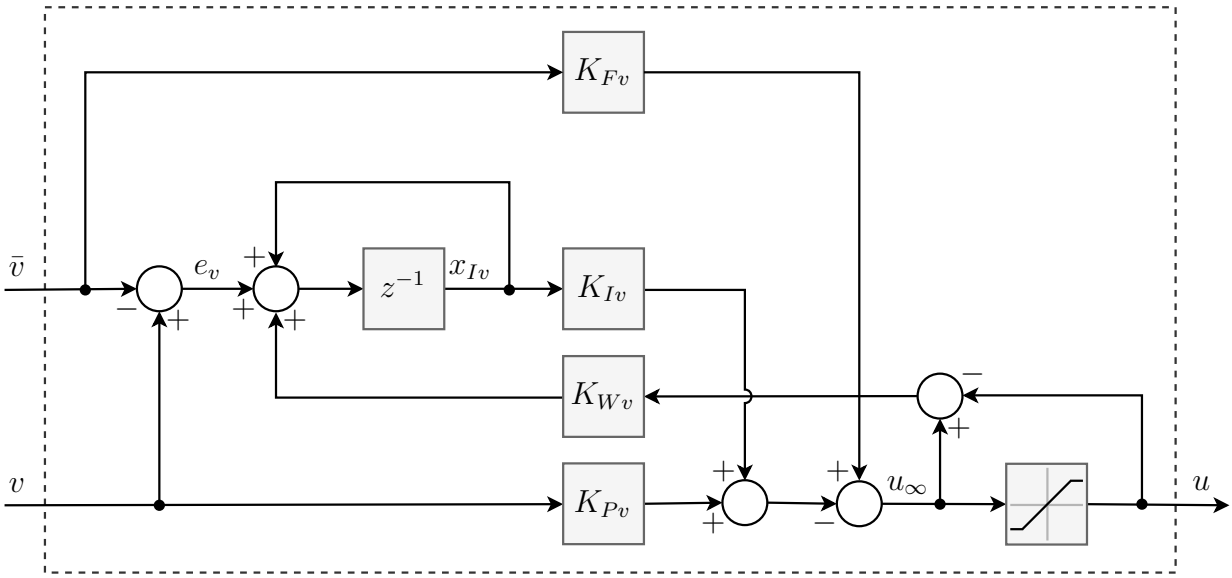


Figure 3.4 Velocity control block diagram

Therefore, we obtain the following closed-loop dynamics

$$x_{Iv}(k+1) = x_{Iv}(k) + v(k) - \bar{v}(k) + K_{Wv}\left[u_\infty(k) - u(k)\right], \tag{3.24}$$

$$v(k+1) = -T_s K_{Iv} x_{I(v)}(k) + (1 - T_s K_{Pv})v(k) + T_s K_{Fv}\bar{v}(k), \tag{3.25}$$

where $u = \text{sat}(u_\infty)$ and $\text{sat}(\cdot)$ is a symmetric saturation function defined as

$$\text{sat}(a) = \begin{cases} -u_{sat}, & \text{if } a \leq -u_{sat}, \\ a, & \text{if } -u_{sat} < a < u_{sat}, \\ u_{sat}, & \text{if } u_{sat} \leq a, \end{cases} \tag{3.26}$$

with $0 < u_{sat} < u_{max}$, a tuning parameter to set maximal acceleration of agents. The term in K_{Wv} is introduced to prevent the wind-up of the error integral x_{Iv} in case of control input saturation. Nevertheless, the controller will first be synthesized for the nominal case with no saturation (i.e., $u(k) - \text{sat}\left(u(k)\right) = 0$) and the anti wind-up gain K_{Wv} will be tuned later through simulations. Thus, the closed-loop nominal transfer function in $\mathcal{Z}$ plane is

$$\frac{v(z)}{\bar{v}(z)} = \frac{T_s K_{Fv}\left(z - \left(1 - \dfrac{K_{Iv}}{K_{Fv}}\right)\right)}{z^2 + \left(T_s K_{Pv} - 2\right) z + \left(T_s K_{Iv} - T_s K_{Pv} + 1\right)}. \tag{3.27}$$

Then, K_{Iv} and K_{Pv} are chosen such that the closed loop system has two real poles $0 < p_{v,1} \leq p_{v,2} < 1$ and K_{Fv} is adjusted such that the zero of the transfer function compensates the slowest pole $p_{v,2}$:

$$K_{Pv} = \frac{2 - p_{v,1} - p_{v,2}}{T_s}, \qquad K_{Iv} = K_{Pv} + \frac{p_{v,1} p_{v,2} - 1}{T_s}, \qquad K_{Fv} = \frac{K_{Iv}}{1 - p_{v,2}}. \tag{3.28}$$

Tuning

Given a maximal expected reference input step $\bar{v}_{max}$, we want to choose the poles $p_{v,1}$ and $p_{v,2}$ such that the the response time is minimized and the control input remains below u_{sat}. For $T_s = 0.10\,\text{s}$, $u_{sat} = 3.4\,\text{m}\,\text{s}^{-2}$ and $v_{max} = 0.50\,\text{m}\,\text{s}^{-1}$, simulations have shown that $p_{v,1} = 0.30$ and $p_{v,2} = 0.80$ lead to acceptable time responses. Figure 3.5 shows a settling time of $0.3\,\text{s}$ and no overshoot. While $p_{v,1}$ sets the closed-loop response time, $p_{v,2}$ can be set to a bigger value to moderate the obtained control gains. Then K_{Wv} is adjusted so that no significant overshoot is observed for a reference input step ten times bigger than $\bar{v}_{max}$ as displayed on Figure 3.6. The corresponding gains are

$$K_{Pv} = 9.0\,\text{s}^{-1}, \qquad K_{Iv} = 1.4\,\text{s}^{-1}, \qquad K_{Fv} = 7.0\,\text{s}^{-1}, \qquad K_{Wv} = 0.5\,\text{s}.$$

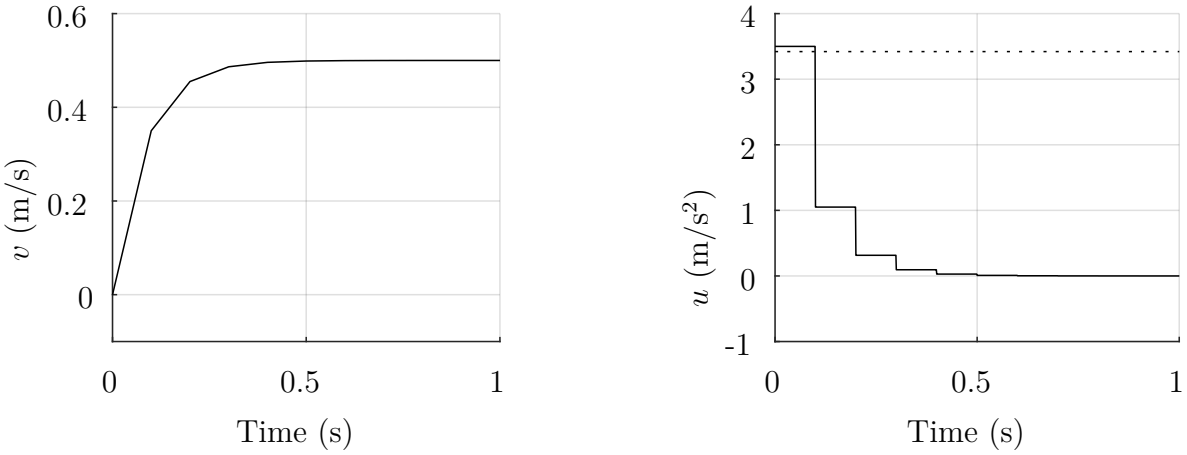


Figure 3.5 Maximal velocity step response without saturation

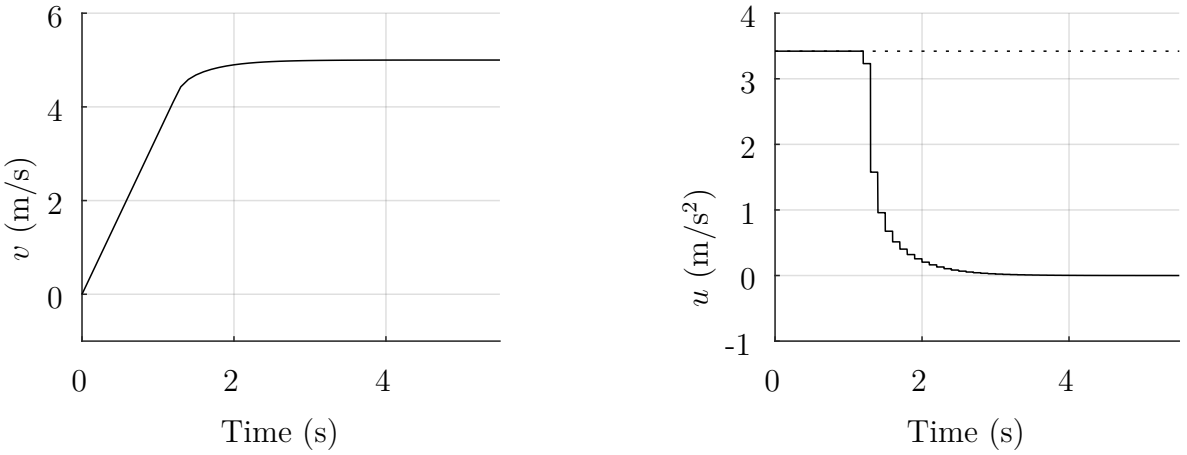


Figure 3.6 Excessive velocity step response with saturation

3.2.4 Position Control

Synthesis

Position control is about bringing the error $e_r(k) = r(k) - \bar{r}(k)$ to zero where $\bar{r}$ is the desired position. We denote x_{Ir} the integral of the error e_r and propose the unbounded control law

$$u_\infty(k) = -K_{Ir}x_{Ir}(k) - K_{Pr}r(k) - K_{Dr}v(k) + K_{Fr}\bar{r}(k), \tag{3.29}$$

with K_{Ir}, K_{Pr}, K_{Dr} and K_{Fr}, respectively the integral, the proportional, the derivative and the feed forward gains. Figure 3.7 shows the block diagram of the position control.

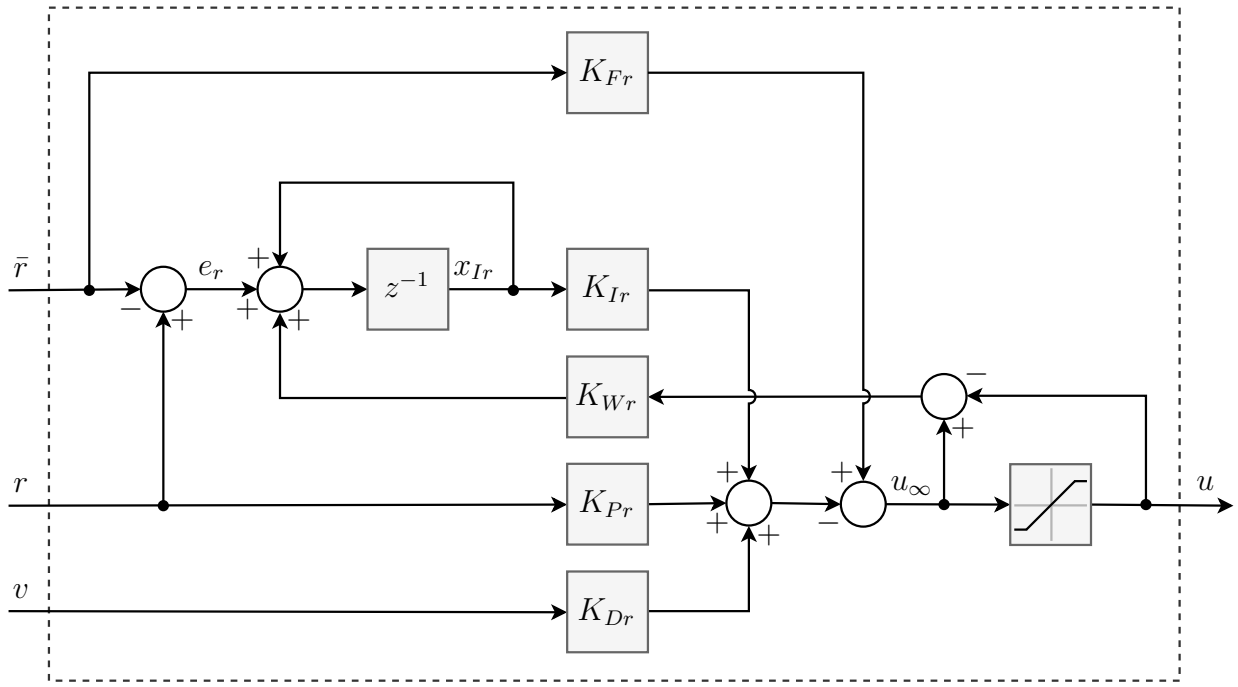


Figure 3.7 Position control block diagram

Thus, the closed-loop system dynamics follows

$$x_{Ir}(k+1) = x_{Ir}(k) + r(k) - \bar{r}(k) + K_{Wr}\left[u_\infty(k) - u(k)\right], \tag{3.30}$$

$$r(k+1) = -\frac{T_s^2}{2}K_{Ir}x_{Ir}(k) + \left(1 - \frac{T_s^2}{2}K_{Pr}\right)r(k) + \left(T_s - \frac{T_s^2}{2}K_{Dr}\right)v(k) + \frac{T_s^2}{2}K_{Fr}\bar{r}(k), \tag{3.31}$$

$$v(k+1) = -T_sK_{Ir}x_{Ir}(k) - T_sK_{Pr}r(k) + (1 - T_sK_{Dr})v(k) + T_sK_{Fr}\bar{r}(k), \tag{3.32}$$

where K_{Wr} is the anti wind-up gain of the position controller. Therefore, the closed-loop nominal (i.e., without saturation) transfer function in $\mathcal{Z}$ plane is

$$\frac{r(z)}{\bar{r}(z)} = \frac{T_s^2K_{Fr}}{2}\frac{(z+1)(z-b_{r,0})}{z^3 + a_{r,2}z^2 + a_{r,1}z + a_{r,0}}, \tag{3.33}$$

where

$$a_{r,2} = \frac{T_s^2}{2}K_{Pr} + T_sK_{Dr} - 3, \qquad a_{r,0} = \frac{T_s^2}{2}(K_{Ir} - K_{Pr}) + T_sK_{Dr} - 1, \tag{3.34}$$

$$a_{r,1} = \frac{T_s^2}{2}K_{Ir} - 2T_sK_{Dr} + 3, \qquad b_{r,0} = 1 - \frac{K_{Ir}}{K_{Fr}}. \tag{3.35}$$

Then, K_{Ir}, K_{Pr} and K_{Dr} are chosen such that the closed loop system has two conjugate poles $p_{r,1/2}$ and one real pole $p_{r,3}$. K_{Fv} is adjusted such that the zero of the transfer function compensates the real pole $p_{r,3}$:

$$K_{Dr} = \frac{\bar{a}_{r,0} - \bar{a}_{r,1} + \bar{a}_{r,2} + 7}{4T_s}, \qquad K_{Ir} = \frac{2}{Ts^2}\left(\bar{a}_{r,1} + 2T_s K_{Dr} - 3\right), \tag{3.36}$$

$$K_{Pr} = \frac{2}{Ts^2}\left(\bar{a}_{r,2} - T_s K_{Dr} + 3\right), \qquad K_{Fr} = \frac{K_{Ir}}{1 - \bar{b}_{r,0}}, \tag{3.37}$$

where

$$\bar{a}_{r,2} = -\left(p_{1,r} + p_{2,r} + p_{3,r}\right), \qquad \bar{a}_{r,1} = p_{r,1}p_{r,2} + p_{r,2}p_{r,3} + p_{r,3}p_{r,1}, \tag{3.38}$$

$$\bar{a}_{r,0} = -p_{r,1}p_{r,2}p_{r,3}, \qquad \bar{b}_{r,0} = p_{r,3}. \tag{3.39}$$

Tuning

Given a maximal expected reference input step $\bar{r}_{max}$, we want to choose the poles $p_{r,1/2}$ and $p_{r,3}$ such that the the response time is minimized and the control input remains below u_{sat}. Instead of directly choosing pole values, let us introduce two tuning parameters : $0 < \omega_r < \frac{2\pi}{T_s}$ and $0 < \xi_r < 1$ such that

$$p_{r,1} = e^{T_s s_r}, \qquad p_{r,2} = p_{r,1}^*, \qquad p_{r,3} = |p_{r,1}|, \qquad s_r = -\omega_r\left(\xi_r + j\sqrt{1-\xi_r^2}\right). \tag{3.40}$$

For $T_s = 0.1\,\mathrm{s}$, $u_{sat} = 2.6\,\mathrm{m\,s^{-2}}$ and $\bar{r}_{max} = 50.0\,\mathrm{m}$, simulations have shown that $\omega_r = 0.23$ and $\xi_r = 0.71$ lead to acceptable time responses. Figure 3.8 shows a settling time of 13 s and an overshoot below 5%. Then K_{Wr} is adjusted so that no significant overshoot is observed on Figure 3.9 for a reference input step ten times bigger than $\bar{r}_{max}$. The gains used are

$$K_{Pr} = 0.10\,\mathrm{s^{-2}}, \quad K_{Ir} = 0.00084\,\mathrm{s^{-2}}, \quad K_{Dr} = 0.48\,\mathrm{s^{-1}}, \quad K_{Fr} = 0.052\,\mathrm{s^{-2}}, \quad K_{Wr} = 50\,\mathrm{s^2}.$$

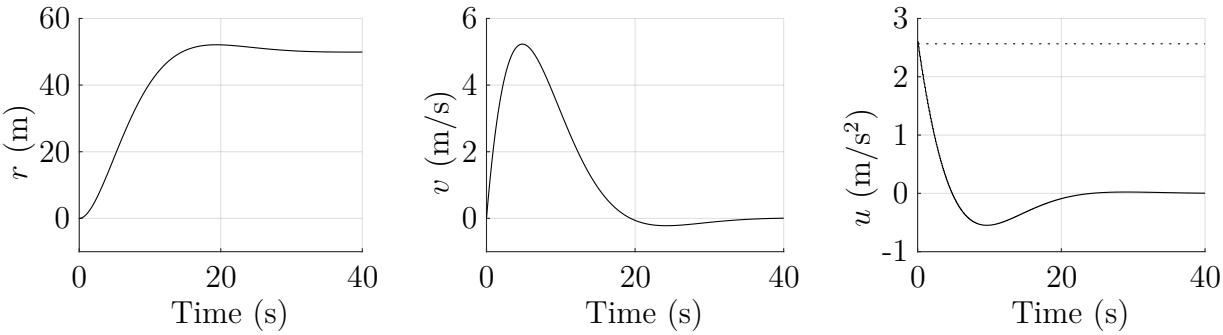


Figure 3.8 Maximal position step response without saturation

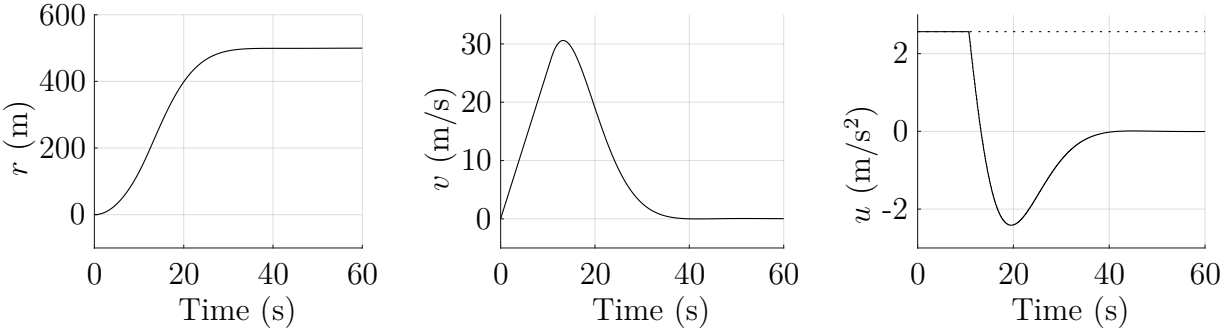


Figure 3.9 Excessive position step response with saturation

3.2.5 Tracker Control

Synthesis

The objective of tracker control law is to enable some sort of following behavior where the controlled agent follows another agent while maintaining a static offset. This is achieved if the controller manages to bring the errors $e_r(k) = r(k) - \bar{r}(k)$ and $e_v(k) = v(k) - \bar{v}(k)$ to zero where $\bar{r}$ and $\bar{v}(k)$ are respectively the offset position and the velocity of a targeted moving agent with control input $\bar{u}(k)$. For design purposes, we assume that agents can communicate their state and control input to others without any delay (i.e., $\bar{r}(k)$, $\bar{v}(k)$ and $\bar{u}(k)$ are available for the controller at epoch k). Nevertheless, further simulations will assess the effect of communication delays on control performance. Once again, we introduce x_{It} the integral of the error e_r. Then, we propose the unbounded control law

$$u_\infty(k) = -K_{It}x_{It}(k) - K_{Pt}e_r(k) - K_{Dt}e_v(k) + \bar{u}(k), \tag{3.41}$$

with K_{Ir}, K_{Pr} and K_{Dr}, respectively the integral, the proportional and the derivative gains. Figure 3.10 presents the block diagram of the tracker control.

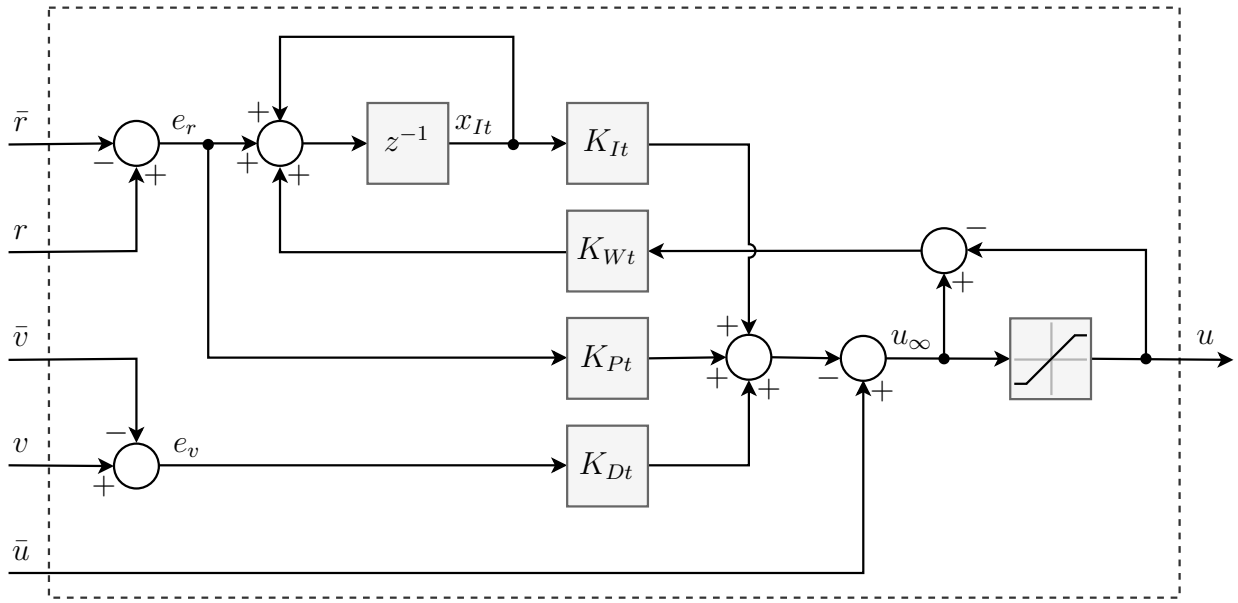


Figure 3.10 Tracker control block diagram

Then, the closed-loop error dynamics is

$$x_{It}(k+1) = x_{It}(k) + e_r(k) + K_{Wt}\left[u_\infty(k) - u_(k)\right], \tag{3.42}$$

$$e_r(k+1) = -\frac{T_s^2}{2}K_{It}x_{It}(k) + \left(1 - \frac{T_s^2}{2}K_{Pt}\right)e_r(k) + \left(T_s - \frac{T_s^2}{2}K_{Dt}\right)e_v(k), \tag{3.43}$$

$$e_v(k+1) = -T_sK_{It}x_{It}(k) - T_sK_{Pt}e_r(k) + (1 - T_sK_{Dt})\,e_v(k), \tag{3.44}$$

where K_{Wt} is the anti wind-up gain of the target controller. Note that this dynamic is identical to the one for position control. Then, the gains that place the two conjugate poles $p_{t,1/2}$ and one real pole $p_{t,3}$ are obtained just like the ones of position controller.

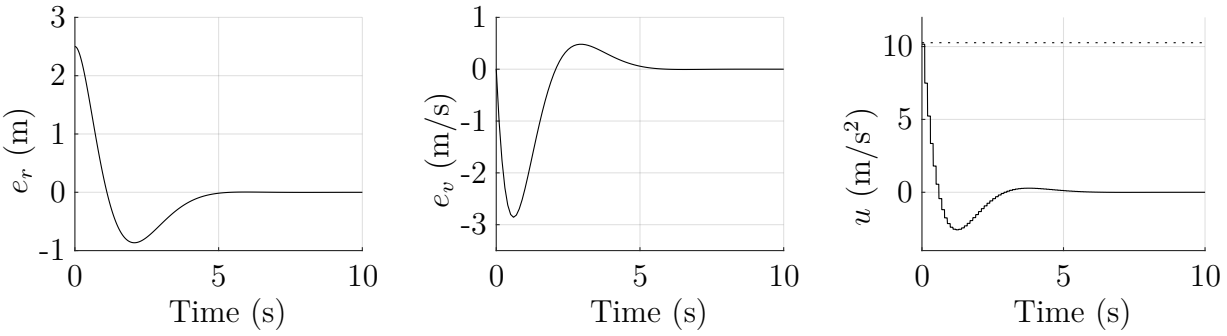


Figure 3.11 Maximal initial position error response without saturation

Tuning

Given a maximal expected initial position error $e_r(0) = e_{r,max}$ and no velocity initial error $e_v(0) = 0\,\mathrm{m\,s^{-1}}$, we want to choose the poles $p_{t,1/2}$ and $p_{t,3}$ such that the response time is minimized and the control input remains below $3u_{sat}$. Here, we want to take advantage of saturation to avoid any major undershoot. Poles are chosen using the method described for position control. For $T_s = 0.1\,\mathrm{s}$, $u_{sat} = 3.4\,\mathrm{m\,s^{-2}}$ and $e_{r,max} = 2.5\,\mathrm{m}$, simulations have shown that $\omega_t = 1.5$ and $\xi_r = 0.71$ lead to an acceptable time response. Figure 3.11 shows the time response of the system without saturation and we can measure a settling time of 5 s and a massive undershoot of 30 %. Then K_{Wt} is adjusted so that no significant undershoot is observed on Figure 3.12 for an initial position error ten times bigger than $e_{r,max}$. Finally, Figure 3.13 displays the time response of the system for an initial position error equal to $e_{r,max}$ and we can observe a settling time of 3 s with no undershoot. The corresponding gains are

$$K_{Pt} = 4.1\,\mathrm{s^{-2}}, \qquad K_{It} = 0.20\,\mathrm{s^{-2}}, \qquad K_{Dt} = 2.9\,\mathrm{s^{-1}}, \qquad K_{Wt} = 5\,\mathrm{s^2}.$$

In addition, Figure 3.14 shows the tracking error time response when the target of the tracker controller is an agent with position controller and reference input step $\bar{r}_{max} = 50.0\,\mathrm{m}$ for a maximal expected communication delay $\tau_{max} = 0.3\,\mathrm{s}$. Under this conditions, the tracking error position remains below 1.6 m.

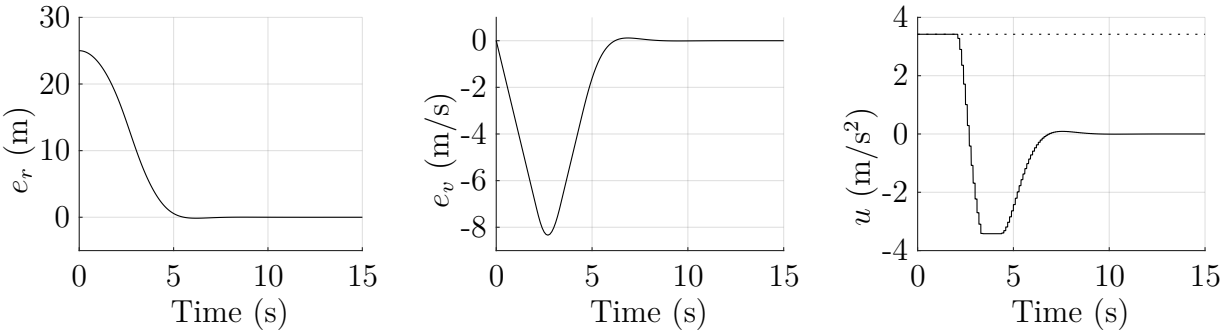


Figure 3.12 Excessive initial position error response with saturation

3.2.6 Velocity Limiter

Synthesis

In cases where the controller does not explicitly set the velocity of the agent (i.e., position and tracker control), we want to dynamically limit the velocity of the agent and even stop the agent (i.e., limit set to zero). Such velocity limiter will notably come in handy in Subsection 4.3.1. Hence, the goal of the velocity limiter is to ensure $|v(k)| < \bar{v}_{lim}(k)$ where

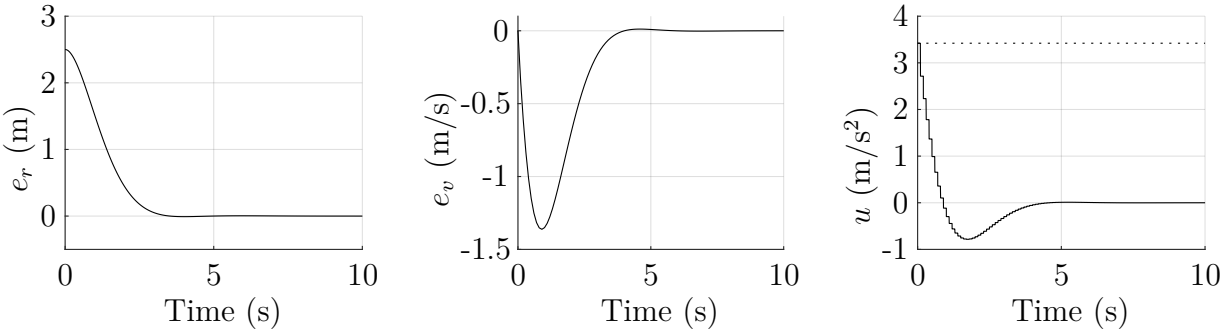


Figure 3.13 Maximal initial position error response with saturation

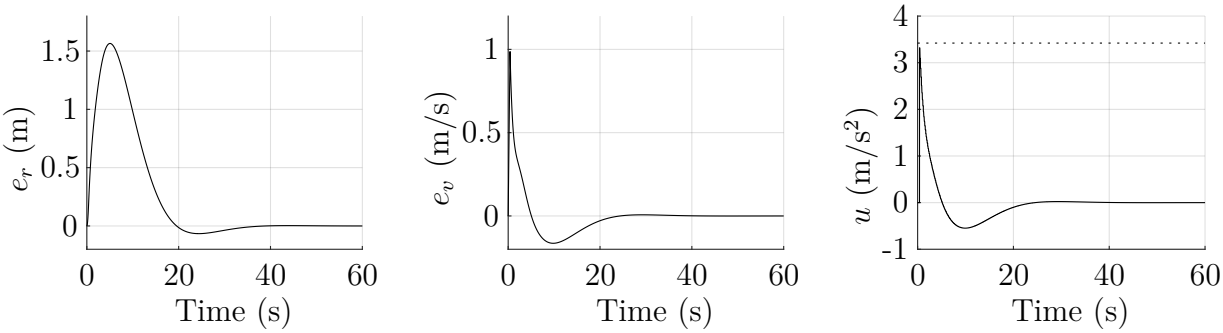


Figure 3.14 Tracking error response

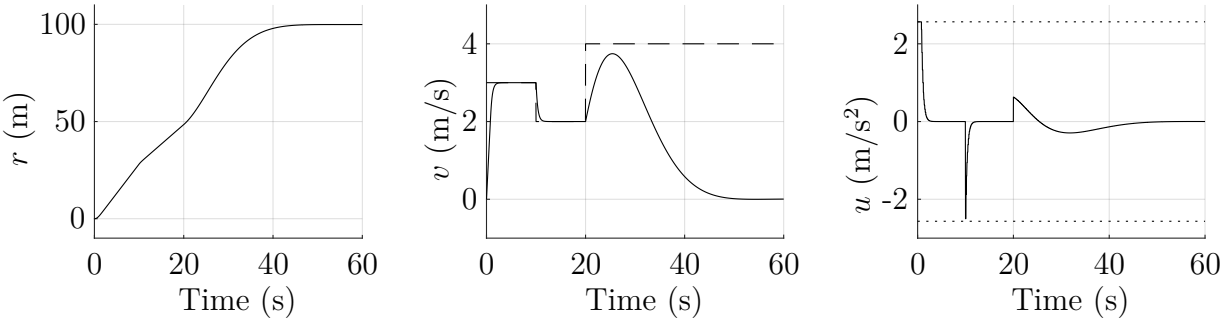


Figure 3.15 Velocity limit steps response without noise

$\bar{v}_{lim}(k) > 0$ is the velocity limit reference input. The velocity limiter is designed to be added in the controller just before saturation. As such, it takes u_∞, v and $\bar{v}_{lim}$ as inputs and generates u_{lim}, the u_∞ substitute that enforce velocity limitation and finally $u(k) = \text{sat}(u_{lim})$. The switching unbounded control law u_{lim} is defined as follows

$$u_{lim}(k) = \begin{cases} \min(u_\infty(k), u_+(k)), & \text{if } \bar{v}_{lim}(k) > \varepsilon_{lim} \text{ and } v(k) > 0, \\ \max(u_\infty(k), u_-(k)), & \text{if } \bar{v}_{lim}(k) > \varepsilon_{lim} \text{ and } v(k) \leq 0, \\ u_\sim(k), & \text{if } \bar{v}_{lim}(k) \leq \varepsilon_{lim}, \end{cases} \tag{3.45}$$

with

$$u_+(k) = -K_{lim}\left(v(k) - \bar{v}_{lim}(k)\right), \quad u_-(k) = -K_{lim}\left(v(k) + \bar{v}_{lim}(k)\right), \quad u_\sim(k) = -K_{lim}v(k),$$

where $0 < K_{lim} < \dfrac{2}{T_s}$ and $0 < \varepsilon_{lim}$ are two tuning parameters. K_{lim} influences the dynamics of the velocity limiter (i.e., fast dynamics for big values of K_{lim}). ε_{lim} is introduced to avoid the chattering effect typical of switching control laws. Let assume that we do not take this precaution (i.e., $\varepsilon_{lim} = 0$) and the estimate/measurement of v manifests some error/noise. Then for $v_{lim} = 0$ and $u_\infty > 0$ (alternatively, $u_\infty < 0$), when v finally reach v_{lim}, the error/noise prevents v from being exactly zero. Thus, u_{lim} will constantly switch between u_∞ and u_+ (respectively, u_-) : that is the chattering effect.

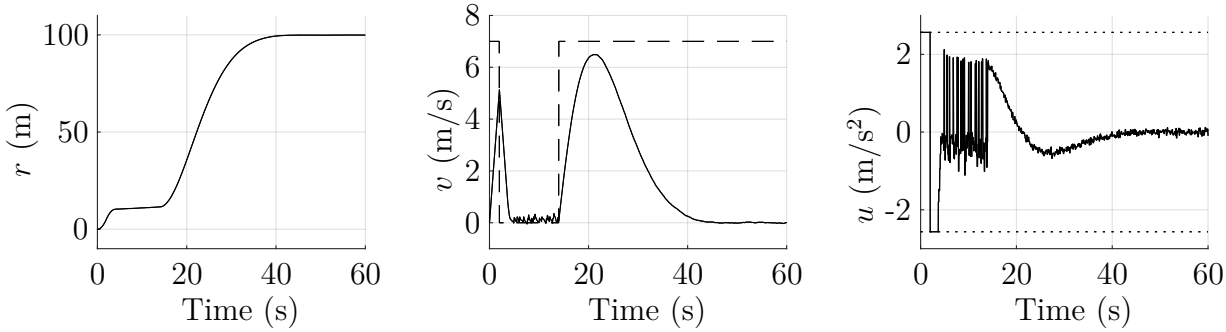


Figure 3.16 Velocity limit steps response with chattering

Tuning

Let us assume that v is free of any error. We want to choose K_{lim} such that the velocity limiter exhibits the fastest dynamics while the control input u_{lim} remains below u_{sat} for a velocity limit step $\bar{v}_{lim,max}$. For $T_s = 0.1\,\text{s}$, $u_{sat} = 2.6\,\text{m}\,\text{s}^{-2}$ and $\bar{v}_{lim,max} = 1.00\,\text{m/s}$, simulations have shown that $K_{lim} = 2.50\,\text{s}^{-1}$ leads to acceptable time responses. Figure 3.15

displays the response of an agent using the position controller augmented by the velocity limiter. At $t = 0\,\mathrm{s}$, the position controller experience a position reference input step of $100\,\mathrm{m}$ while the velocity limit reference input (i.e. the dashed line) is set to $3\,\mathrm{m/s}$. At $t = 10\,\mathrm{s}$, the velocity limit steps to $2\,\mathrm{m/s}$ (i.e., $\bar{v}_{lim,max}$ below). Then, at $t = 30\,\mathrm{s}$, the velocity limiter is released (i.e., set to a high value) so that the position controller operates freely and r finally reaches $100\,\mathrm{m}$. Note that the control input peak observed at $t = 10\,\mathrm{s}$ for a velocity limit reference step $\bar{v}_{lim,max}$ remains below u_{sat}. Now, let us assume that v manifests some estimation errors such that v is the sum of the true velocity value and a discrete-time zero-mean Gaussian noise of sampling time T_s and variance σ_n^2. Then, taking $\varepsilon_{lim} = 3\sigma$ allows us to globally avoid the chattering effect as the switching control will be theoretically triggered by estimation error only 3 times every 1000 periods. To illustrate this, Figure 3.16 and Figure 3.17 show the response of an agent using the position controller augmented by the velocity limiter for $T_s = 0.1\,\mathrm{s}$, $u_{sat} = 2.6\,\mathrm{m\,s^{-2}}$ and $\sigma_n = 0.1\,\mathrm{m/s}$. In the simulations, the same noise has also been added to the position estimate. At $t = 0\,\mathrm{s}$, the position controller experiences a position reference input step of $100\,\mathrm{m}$. At $t = 1\,\mathrm{s}$, the velocity limit reference input is set to zero and released a dozen of seconds later. On the one hand, Figure 3.16 exhibits control input chattering when no action is taken to avoid it. On the other hand, Figure 3.17 displays the system response when $\varepsilon_{lim} = 3\sigma_n$. On the latter, we can still see some strong oscillation of the control input when the velocity limiter stops the agent (i.e., between $t = 5\,\mathrm{s}$ and $t = 15\,\mathrm{s}$). This is simply the noise on v amplified by gain K_{lim}. Note that $0.1\,\mathrm{m/s}$ is a big variance for velocity estimation error in steady-state, this accounts for a worst case scenario.

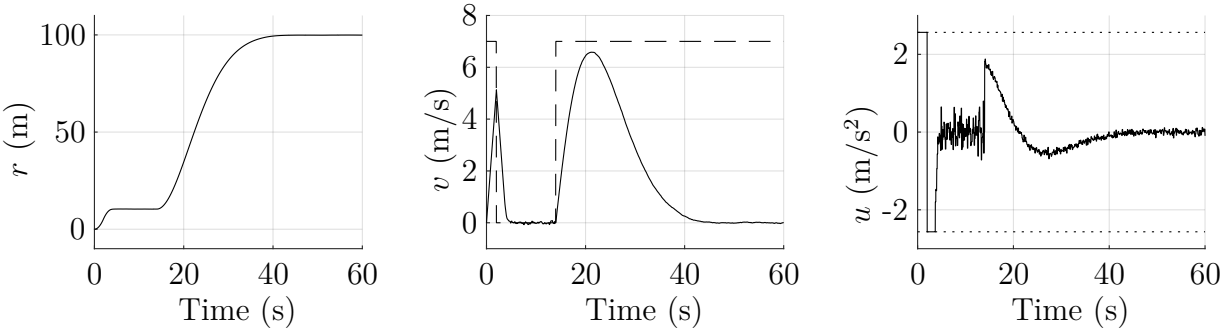


Figure 3.17 Velocity limit steps response without chattering

CHAPTER 4 OUTDOOR SWARMING

4.1 Introduction

Swarm control has recently become a popular topic in the research community. As such, numerous formation control solutions have been proposed. The most common strategy found in the literature is the leader-follower formation which provides an intuitive way to deal with swarm control. Namely, agents in the swarm (i.e. the followers) try to maintain a certain position (i.e., an offset) with respect to an identified agent (i.e., the leader) while the latter moves. However, most of the reviewed work assumes that the followers know their corresponding positions in the formation and does not address how the agents are assigned their respective offsets (e.g., [25], [13] and [66]). This implies manual initialization procedures and limits the autonomy of the solution. With this in mind, we decided to develop a complete solution that not only handles formation maintenance, but also formation initialization where positions in formation are assigned to agents.

This chapter first presents the problem setting based on the material under development at Humanitas Solutions. Then, a position assignment algorithm and a formation control method are combined to produce an original swarming solution for outdoor environments. Finally, SITL simulations are performed to evaluate the solution performance.

4.2 Problem Setting

As mentioned above, this swarming algorithm has to be integrated into the preexisting solution developed by Humanitas Solutions and tested in their proprietary simulator. Therefore, some parameters specific of the system design have to be introduced.

The dynamics of quadrotors and embedded attitude controller match the assumptions made in Chapter 3 so that the controllers previously developed can be successfully implemented. Both the translation control and the swarming algorithm are constrained by design to run in the same thread with sampling time T_s. Also, let d_c and d_r denote respectively the collision-free distance and the reference neighbors distance so that $d_q < d_c < d_r$, where d_q is the quadrotors' total diameter. In practice, we sets $d_c \approx 3d_q$ and $d_r \approx 3d_c$.

Communications between agents in the swarm are handled by Heterogeneous Embedded Ad hoc Virtual Emergency Network (HEAVEN), a distributed networking solution developed by Humanitas Solutions. HEAVEN provides multi-hop capability that allows information ex-

change between all agents, as long as the swarm communication graph is connected. Two agents are said to be neighbors in the communication graph (i.e., single hop) if the physical distance between them remains below the communication range d_{com}. The communication delay $\tau_{i,j}$ between agent i and j is proportional to the topological distance between them $\text{dist}(i,j)$ (i.e., number of hops). Hence, we have $\tau_{i,j} = \text{dist}(i,j)\tau_{com}$ where τ_{com} is the delay for single-hop communication. In practice, we observed that $\tau_{com} \leq 3T_s$. This delay is generated by onboard processes. Actually, the single-hop round-trip-time of HEAVEN has been measured to be around a millisecond. More details on graph theory are given in Section 5.2.5.

As for leader control, we consider a simple path following mission where the leader is required to visit a certain sequence of way-points (i.e., a path). If the given path is a trip home (i.e., last way-point corresponds to the take-off position), then followers also return at their take-off position (also called home). Indeed, Humanitas Solutions's embedded mission manager makes agents go straight back home when idle, regardless of collisions. Therefore, our solution has to work out a collision-free way back home before handing the agent control back to the mission manager. That said, we only focus on the swarming solution developed here and only round-trip path (i.e., starting and ending at home) are to be tested.

4.3 Swarming Algorithm

4.3.1 Workflow

Overview

In the initial state, N agents are hovering at their respective home with the same altitude, the user has sent the mission and selected the leader. The agent with index $i = N$ is defined as leader and the other $N - 1$ agents as followers. Then, the swarming algorithm, outlined in Figure 4.2, for both leader and followers starts. After a short initialization sequence, the leader waits until all followers have reached their auto-assigned position in formation. After that, the leader starts moving along a given path using the position control of Section 3.2.4 while followers maintain their offset with respect to the leader using the tracker control of Section 3.2.5. The leader monitors followers' states (i.e., current algorithm sequence) and position, and waits for them when necessary. Then, when all way-points of the given leader's path have been visited (i.e., the leader is back home), the followers go back to their respective homes.

Communication

For the entire duration of the swarming task, all agents broadcast a standardized message every T_s. Those messages help to maintain a local view of the swarm in the form of an unordered map `swarmMap` with the followings fields :

- `agentId` : Agent's unique HEAVEN address and key of the map;
- `swarmId` : Swarm's unique identifier to allow simultaneous swarms operation;
- `timeStamp` : Time of the last update of the entry;
- `taskState` : Current algorithm state {`INIT`; `FORM`, `FLOW`, `LEAD`, `STOP`};
- `px`, `py`, `pz` : Agent GPS position (latitude, longitude, relative altitude);
- `vx`, `vy`, `vz` : Agent velocity in NED frame;
- `ux`, `uy`, `uz` : Agent acceleration control input in NED frame.

If an entry has not been updated for more than T_{msg}, then the entry is removed from the map in order to keep the map accurate.

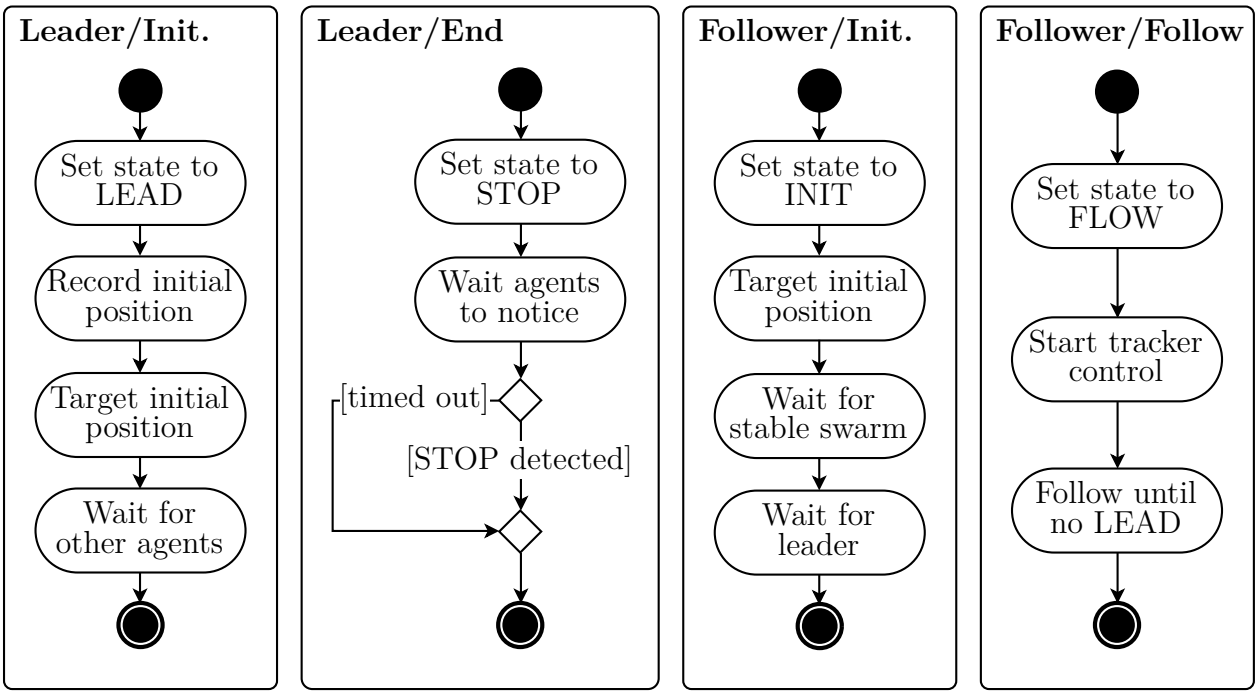


Figure 4.1 Activities (part. 1)

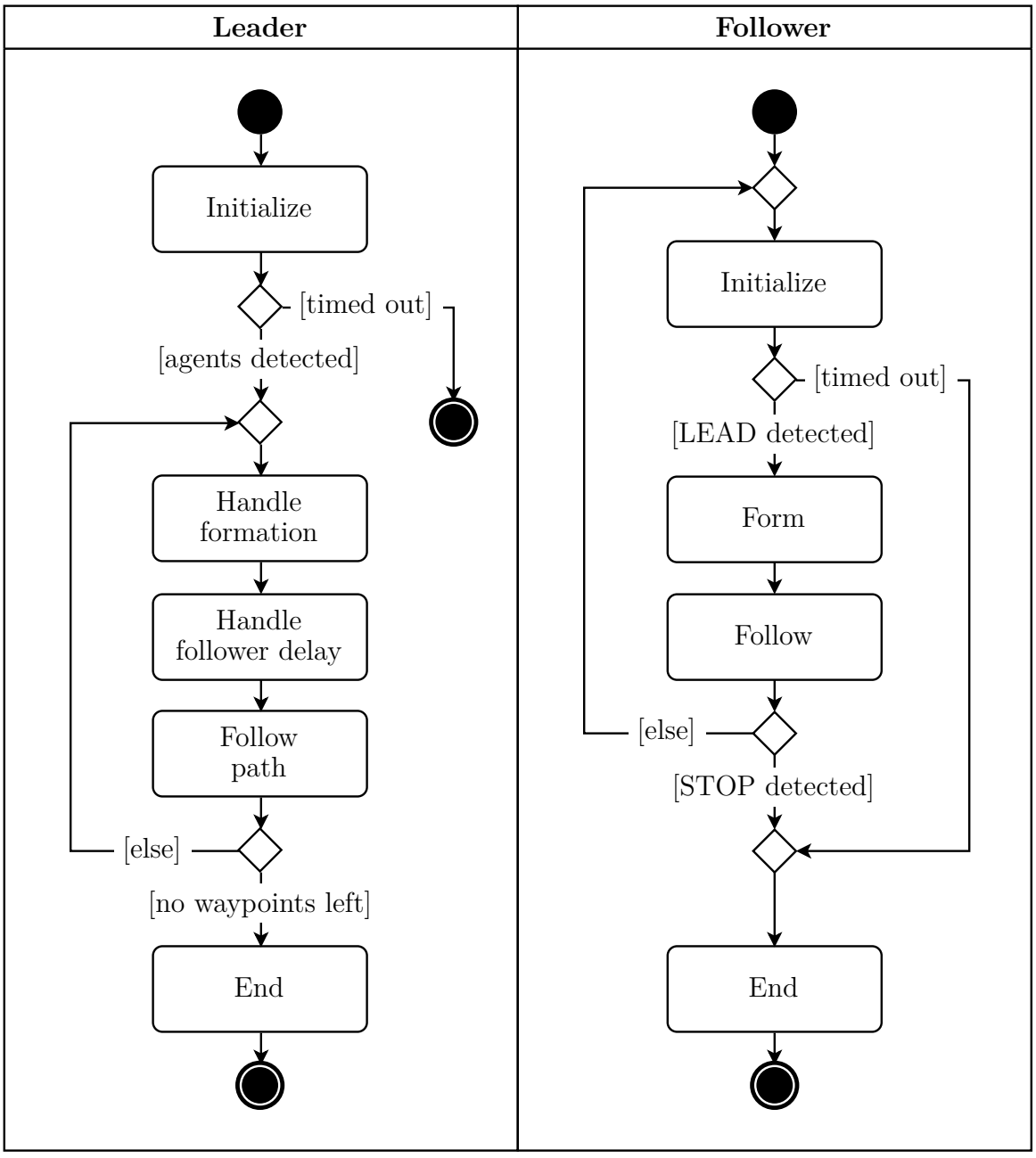


Figure 4.2 Main activity diagram

Initialization Sequence

Figure 4.1 shows initialization sequence details for both leader and followers. Here, the leader sets his state to `LEAD`, records and maintains his initial position (i.e., home) and waits to receive a message from another agent with the same `swarm_id`. If no other agent has been detected after a time T_{led}, the leader aborts the swarming mission. As for the followers, they set their state to `INIT`, record and maintain their initial position, then wait for the swarm composition to stabilize. In practice, the swarm is said to be stable if no agent has been added or removed from the local map of the swarm for a time T_{ini}. If no leader (i.e., an agent with `state` = `LEAD`) has been detected after a time T_{fol}, the followers launch the stop sequence. If a leader is detected, the followers record the `agentId` of the leader.

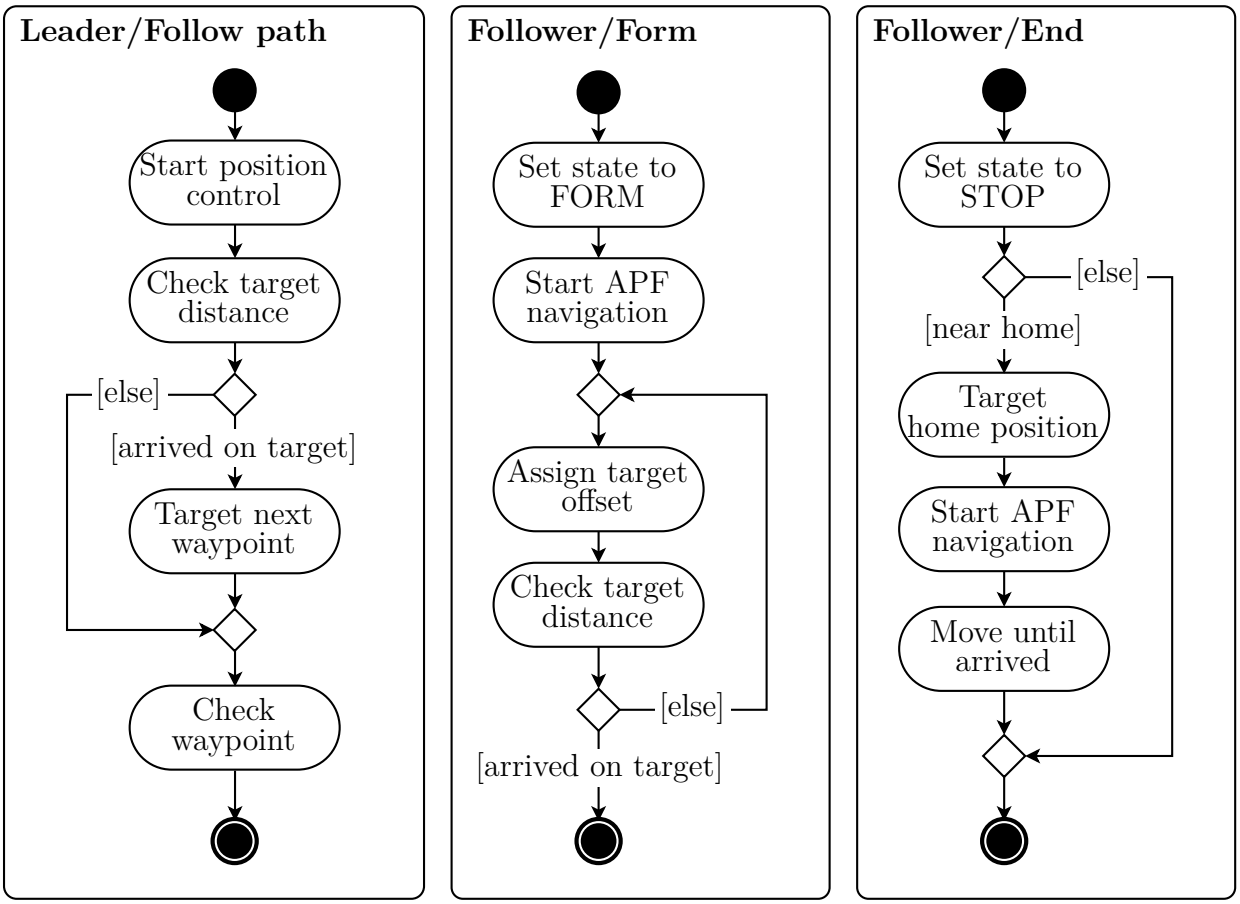


Figure 4.3 Activities (part. 2)

Formation Sequence

Once the leader has been initialized, it stands still until all followers are ready to follow (i.e., at assigned positions in formation). In the meantime, followers execute the formation sequence displayed in Figure 4.1. They set their state to `FORM` and use the APF navigation algorithm detailed in 4.3.3 to reach their formation offset while avoiding other agents. Every T_s, they update their assigned offset using a distributed algorithm explained in 4.3.2. Thus, the offset is continuously updated to handle possible inaccuracies in the local swarm map.

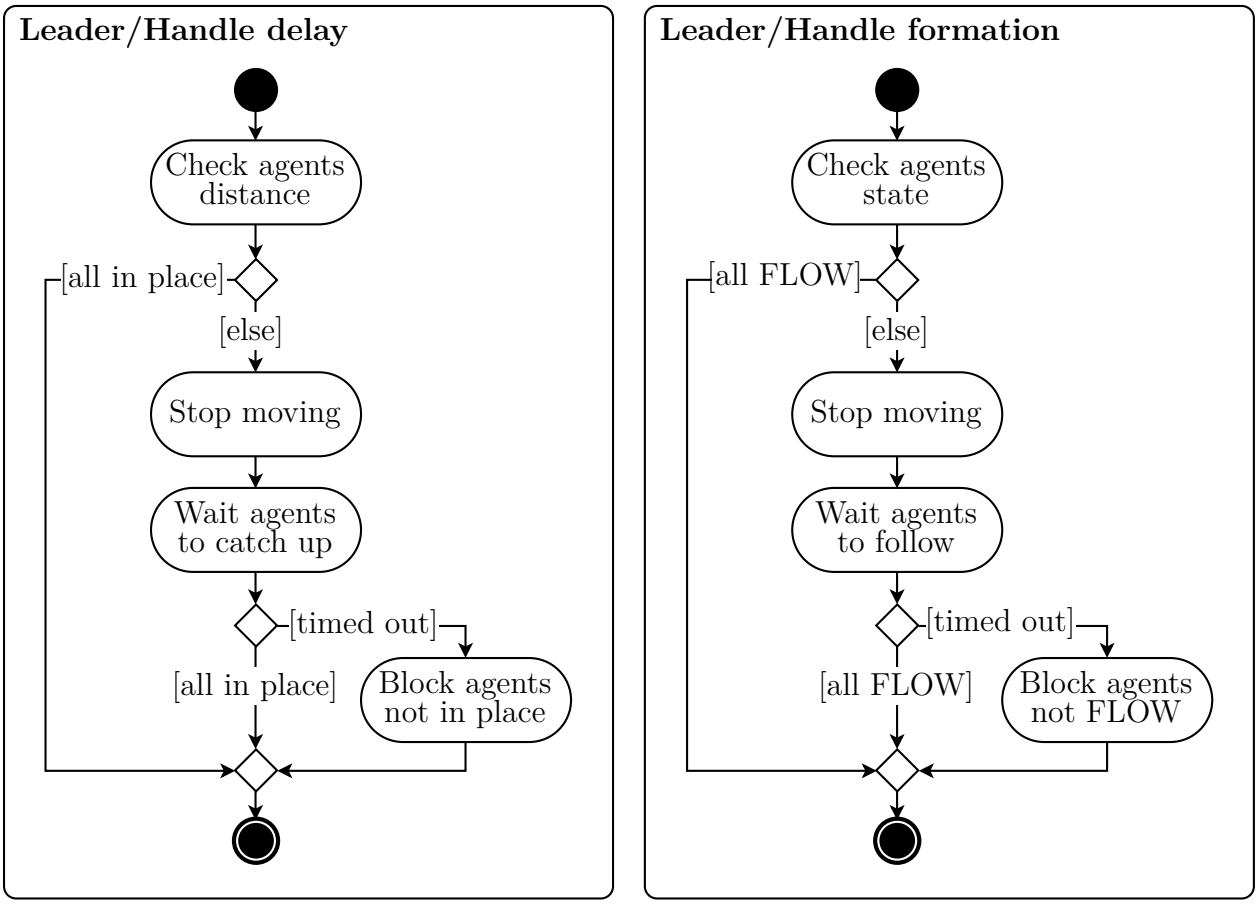


Figure 4.4 Activities (part. 3)

Navigation Sequence

When the followers finally arrive at the assigned offsets, they set their state to `FLOW` and use the tracker control to maintain their offset with respect to the leader, as shown in Figure 4.3. As long as all followers are actually following, the leader moves along the path just as described in Figure 4.1. However, if followers are unable to accurately maintain their offset or if they happen to change their state, Figure 4.4 shows that the leader stop (i.e., velocity limiter set to zero) and wait for the followers to catch-up. To be more precise, the leader wait T_{del} for followers to catch-up and T_{for} for followers to switch to state `FLOW`. The followers that still do not meet the leader's requirements after these times are ignored for the rest of the mission.

In practice, the "all in place" condition is implemented as $\|\boldsymbol{\Delta}_{\mathcal{W}}\| < \Delta_{\mathcal{W},th}$ where $\boldsymbol{\Delta}_{\mathcal{W}}$ integrates the variations of the barycenter of the swarm with respect to the leader. Specifically, we define

$$\boldsymbol{\Delta}_{\mathcal{W}}(k+1) = \frac{\boldsymbol{\Delta}_{\mathcal{W}}(k) + \boldsymbol{\varepsilon}_{\mathcal{W}}(k+1) - \boldsymbol{\varepsilon}_{\mathcal{W}}(k)}{2}, \tag{4.1}$$

with

$$\boldsymbol{\varepsilon}_{\mathcal{W}}(k) = \frac{1}{|\mathcal{W}|} \sum_{i \in \mathcal{W}} \mathbf{r}_i(k) - \mathbf{r}_N(k), \tag{4.2}$$

where $\mathcal{W}$ is the set of agents with state `FLOW` at period k. It has been found in simulation that an appropriate threshold value is $\Delta_{\mathcal{W},th} = \frac{d_r}{2}$.

Stop Sequence

Once the leader has visited all way-points along the path, the leader runs the stop sequence described in Figure 4.3. It sets his state to `STOP` and waits T_{end} for the followers to set their states to `STOP` as well. Then, the leader task is completed and the control is handed back to the mission manager. As soon as a follower does not detect any `LEAD` and at least one `STOP`, it executes the stop sequence presented in Figure 4.1. It sets its state to `STOP` and uses the APF navigation algorithm to go back home while avoiding other agents. Once arrived at home, the follower task is completed and the mission manager takes over.

4.3.2 Position Assignment

Formation Pattern

Before investigating how to assign positions in formation to each follower, one has to choose a set of offsets (i.e., positions with respect to the leader) to be assigned. Recall that only formations in the horizontal plane are considered here.

We define the formation undirected simple graph $\mathcal{G}_h(\mathbf{h}) = (\mathcal{V}_h, \mathcal{E}_h)$ where the parameter $\mathbf{h} = \begin{bmatrix}\mathbf{h}_0^\top & \mathbf{h}_1^\top & \mathbf{h}_2^\top & \dots & \mathbf{h}_M^\top\end{bmatrix}^\top \in \mathbb{R}^{2\times M+1}$ is the set of all $M \geq N-1$ offsets to be assigned and the leader position $\mathbf{h}_0$. Without lack of generality, we set $\mathbf{h}_0^\top = [0, 0]$. This graph is unrelated to the communication graph or even the position of the agents. It depends only on the size of the swarm N and relies essentially on geometric considerations. Indeed, the edges set $\mathcal{E}_h$ gathers all couples of nodes in $\mathcal{V}_h$ (i.e., offsets to be assigned) that have adjacent Voronoi cells. The Voronoi cell associated to a given node is the region consisting of all points closer to that node than to any other. In our application, closeness of points is evaluated using the Euclidean distance. Figure 4.5 represents the Voronoi cells (dotted lines) of a given sets of nodes (bold point) and the resulting edges (solid lines).

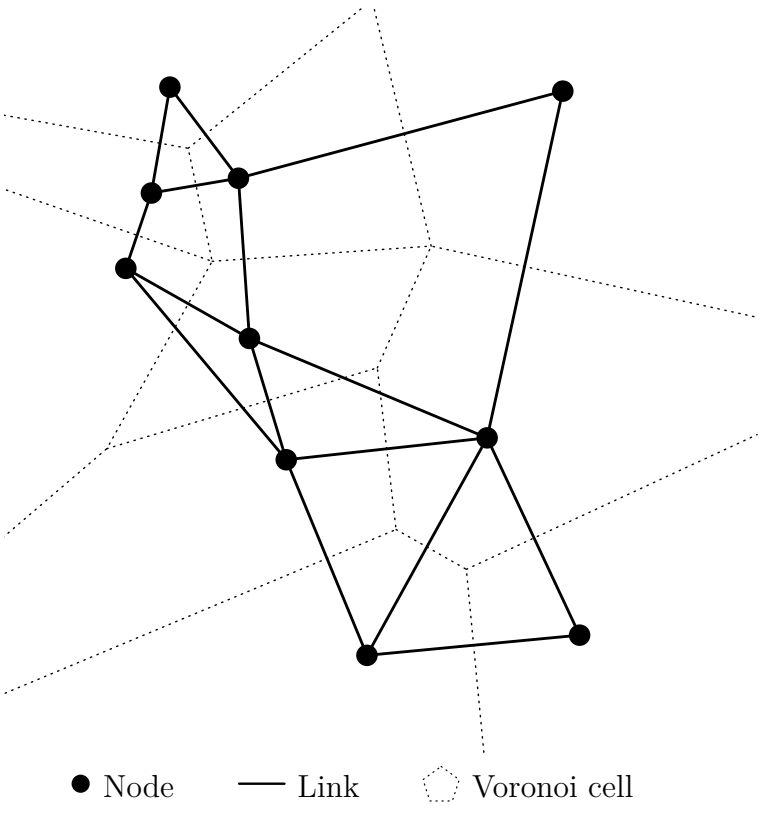
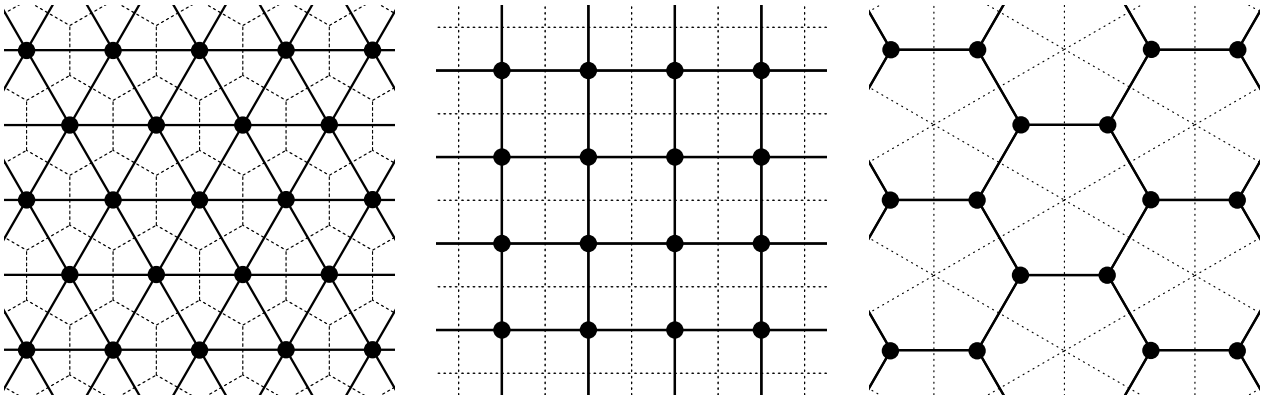


Figure 4.5 Voronoi diagram

As a first criterion, we want to find a formation pattern such that all nodes $i \in \mathcal{V}_h$ are at the same distance d_r from all their neighbors $j \in \mathcal{N}_{h,i}$. This is true if the formation graph $\mathcal{G}_h$ fits a regular tessellation in two dimensions (i.e., graph nodes are located at the corners of the tiles). Indeed, a regular tessellation in two dimensions is the tiling of a plane using identical regular polygons with no overlaps and no gaps. As displayed in Figure 4.6, there are only three polygons that can form such regular tessellations: the equilateral triangle, square, and regular hexagon. As a second criterion, we want to maximize the compactness of the formation. In other terms, we want to maximize the degree of each node (i.e., the number of neighbors) . Therefore, we chose the equilateral triangle, which implies a maximal degree of 6, whereas the square and the regular hexagon give maximal degrees of 4 and 3, respectively. For the same reason, we set the formation contour to a regular hexagon, which enforces a minimal degree of 3.



(a) (b) (c)

Figure 4.6 (a) Triangular (b) Square (c) Hexagonal regular tessellations with Voronoi cells

Note that on Figure 4.6 the Voronoi diagram (dotted lines) associated with such a formation graph draws an hexagonal grid. Specifically, all cells are regular hexagons with side length equals to $\frac{1}{2}d_r$. Then, the formation can be efficiently described using a hexagonal coordinate system. We introduce an implementation of the *axial coordinate system*, based on the work presented in [67]. Let $\{\mathtt{h}\}$ denote the hexagonal axial frame attached to the leader gravity center with axis $\{\mathbf{x}_\mathtt{h}, \mathbf{y}_\mathtt{h}\}$ defined as

$$\begin{cases} \mathbf{x}_\mathtt{h} = d_r \mathbf{R}_\alpha \mathbf{x}_\mathtt{o}, \\ \mathbf{y}_\mathtt{h} = \dfrac{d_r}{2} \mathbf{R}_\alpha \left(\mathbf{x}_\mathtt{o} + \sqrt{3} \mathbf{y}_\mathtt{o} \right), \end{cases} \qquad \text{with} \qquad \mathbf{R}_\alpha = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix}, \tag{4.3}$$

where $\mathbf{x}_\mathtt{o}$ and $\mathbf{y}_\mathtt{o}$ are respectively the North and East vectors of the NED frame and α is the yaw angle of the formation.

Then, for any $\mathbf{r} \in \mathbb{R}^2$, we can switch between its hexagonal coordinates $\mathbf{r}^{\mathtt{h}}$ in frame $\{\mathtt{h}\}$ and its Cartesian coordinates $\mathbf{r}^{\mathtt{o}}$ in frame $\{\mathtt{o}\}$ using

$$\mathbf{r}^{\mathtt{o}} = \mathbf{T}^{\mathtt{o}}_{\mathtt{h}} \mathbf{r}^{\mathtt{h}}, \qquad\qquad \mathbf{r}^{\mathtt{h}} = \mathbf{T}^{\mathtt{h}}_{\mathtt{o}} \mathbf{r}^{\mathtt{o}}, \tag{4.4}$$

where

$$\mathbf{T}^{\mathtt{o}}_{\mathtt{h}} = d_r \begin{bmatrix} \cos\alpha & \cos\left(\alpha + \frac{\pi}{3}\right) \\ \sin\alpha & \sin\left(\alpha + \frac{\pi}{3}\right) \end{bmatrix}, \qquad \mathbf{T}^{\mathtt{h}}_{\mathtt{o}} = \frac{2}{\sqrt{3} d_r} \begin{bmatrix} \sin\left(\alpha + \frac{\pi}{3}\right) & -\cos\left(\alpha + \frac{\pi}{3}\right) \\ -\sin\alpha & \cos\alpha \end{bmatrix}. \tag{4.5}$$

Using this coordinate system, we have $\mathbf{h}^{\mathtt{h}}_i \in \mathbb{Z}^2$ for all nodes i in $\mathcal{V}_h$. Figure 4.7 shows a hexagonal grid with unit cell coordinates and both reference frames.

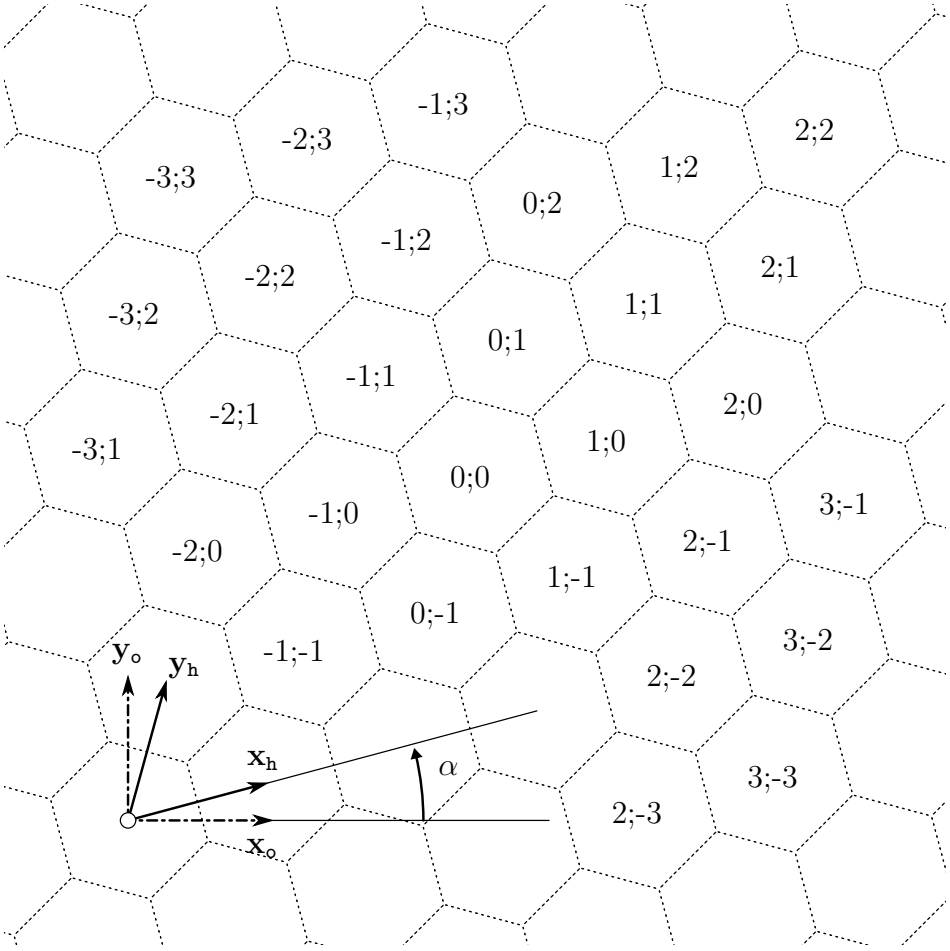


Figure 4.7 Hexagonal coordinate system

The defined hexagonal frame $\{\mathtt{h}\}$ also provides a fast way to determine the topological distance $\text{dist}(i,j)$ between two nodes $i,j \in \mathcal{V}_h$ such that

$$\text{dist}(i,j) = \frac{1}{2}\left(\left|(\mathbf{h}_j - \mathbf{h}_i)^\top \tilde{\mathbf{x}}_\mathtt{h}\right| + \left|(\mathbf{h}_j - \mathbf{h}_i)^\top \tilde{\mathbf{y}}_\mathtt{h}\right| + \left|(\mathbf{h}_j - \mathbf{h}_i)^\top (\tilde{\mathbf{x}}_\mathtt{h} + \tilde{\mathbf{y}}_\mathtt{h})\right|\right), \tag{4.6}$$

where $\tilde{\mathbf{x}}_\mathtt{h}$ and $\tilde{\mathbf{y}}_\mathtt{h}$ are the dual basis of $\mathbf{x}_\mathtt{h}$ and $\mathbf{y}_\mathtt{h}$ defined by

$$\tilde{\mathbf{x}}_\mathtt{h} = \frac{\mathbf{S}\mathbf{y}_\mathtt{h}}{\mathbf{x}_\mathtt{h}^\top \mathbf{S}\mathbf{y}_\mathtt{h}}, \qquad \tilde{\mathbf{y}}_\mathtt{h} = \frac{-\mathbf{S}\mathbf{x}_\mathtt{h}}{\mathbf{x}_\mathtt{h}^\top \mathbf{S}\mathbf{y}_\mathtt{h}}, \qquad \text{with} \qquad \mathbf{S} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}. \tag{4.7}$$

In particular, we define the radius $\rho_i \in \mathbb{N}^+$ of nodes $\mathbf{h}_i$ as the distance to the leader's node 0 so that $\rho_i = \text{dist}(i,0)$.

As a third criterion, we want to minimize the sum of distances of nodes $0 < i \leq M$ toward the leader node $\sum_{i=1}^{M} \|\mathbf{h}_i\|$. This aims to limit the communication delay between the followers and the leader. This last criterion is trivially satisfied when the formation is centered on the leader as shown in Figure 4.8. Thus, knowing the size of the swarm N, we can compute the maximal radius ρ_{max} for all nodes in $\mathcal{V}_h$ with

$$\rho_{max} = \left\lceil \sqrt{\frac{1}{4} + \frac{N-1}{3}} - \frac{1}{2} \right\rceil, \tag{4.8}$$

where $\lceil \cdot \rceil$ is the ceiling function that returns the least integer greater than or equal to its argument. Taking all nodes i with $\rho_i \leq \rho_{max}$, we obtain a first set of M_{max} offsets that completely fill the hexagonal formation contour where

$$M_{max} = 3\rho_{max}\left(\rho_{max} + 1\right). \tag{4.9}$$

Then, we refine the formation by removing the nodes i that are the farthest from the leader (i.e., biggest $\|\mathbf{h}_i\|$) while ensuring $M \geq N$. This is achieved by removing the nodes i with radius $\rho_i = \rho_{max}$ that do not satisfy

$$\kappa_i \geq \chi, \tag{4.10}$$

with

$$\kappa_i = \min\left(\left|\mathbf{h}_i^\top \tilde{\mathbf{x}}_\mathtt{h}\right|, \left|\mathbf{h}_i^\top \tilde{\mathbf{y}}_\mathtt{h}\right|, \left|\mathbf{h}_i^\top (\tilde{\mathbf{x}}_\mathtt{h} + \tilde{\mathbf{y}}_\mathtt{h})\right|\right), \qquad \chi = \left\lfloor \frac{M_{max} - N + 7}{12} \right\rfloor, \tag{4.11}$$

where $\lfloor \cdot \rfloor$ is the floor function that returns the greatest integer less than or equal to its argument. Note that this enforces $0 \leq M - N + 1 \leq 11$.

Note that κ_i equals 0 for nodes located at the corner of the hexagonal formation contour, increases by 1 when moving away to the next nodes and reaches $\left\lfloor \frac{\rho_{max}}{2} \right\rfloor$ for nodes located at the center of the edge. Therefore, the process described above removes unnecessary nodes, starting by the ones closest to the corners, that are the farthest from the center of the hexagonal formation contour (i.e., the leader node). Figure 4.8 illustrates a formation graph obtained for $N = 40$ and a given assignment of the generated offset. The first $M_{max} = 60$ cells have dashed lines, the $M = 42$ cells of the final formation graph have plain lines, the leader's cell is hatched and the cells assigned to an agent are identified by dots.

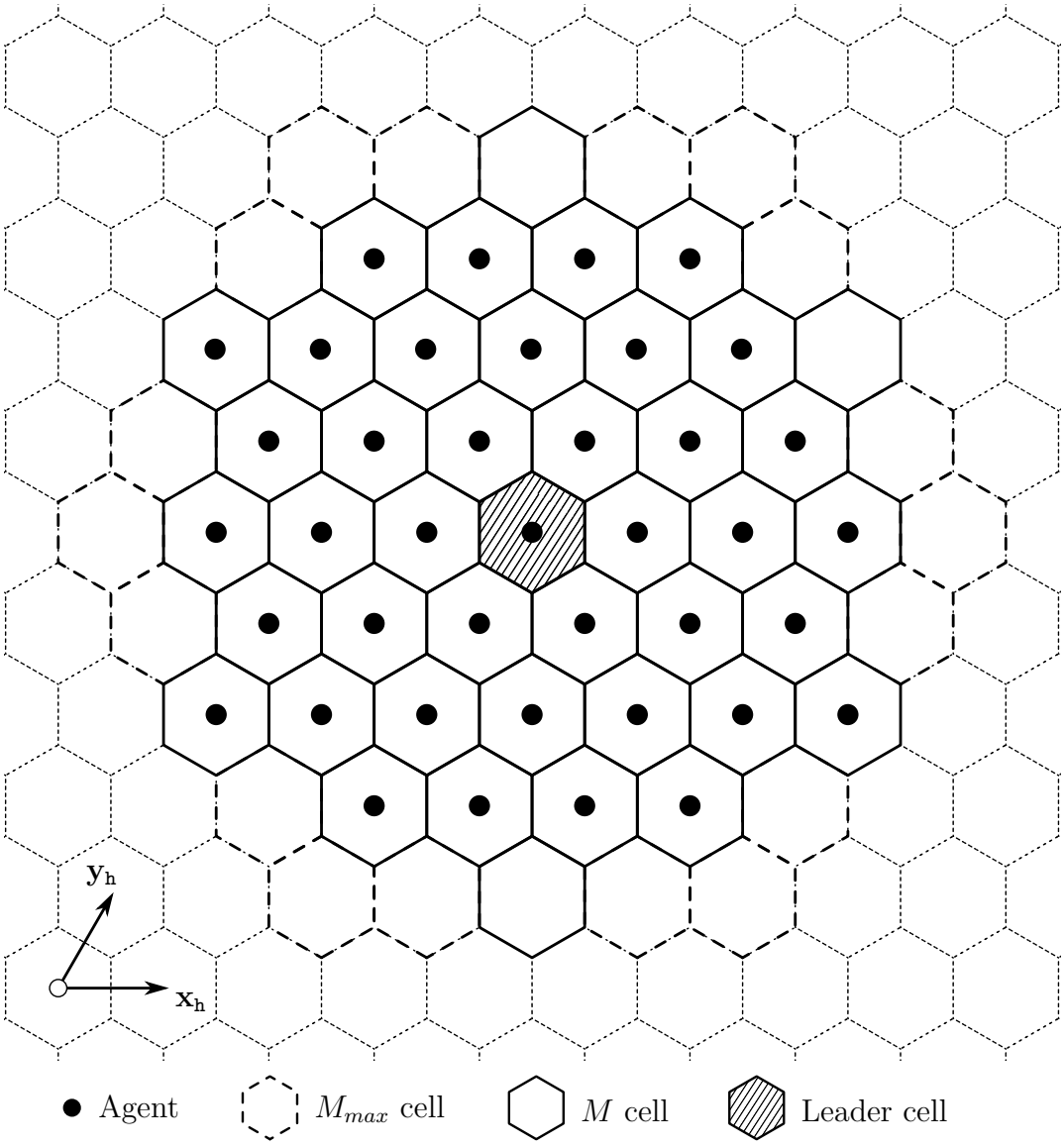


Figure 4.8 Formation graph for $N = 40$

Assignment Algorithm

Assuming we have a set of M formation offsets, we want to assign those to each $N-1$ followers in the swarm in order to minimize the overall distance traveled by all followers to reach their position in formation. This goal has two motivations. First, it minimizes the time and energy required for the swarm to reach the assigned formation. Second, it strongly limits collision hazards. Indeed, if the minimization goal is satisfied, then none of the paths from followers' initial positions to assigned positions intersect with each other. This can be observed in Figure 4.9 where a given set of followers (dots) are linked (line) to their assigned offsets (circles). Also, note that the minimization goal is expressed in terms of relative measurements. Therefore, the algorithm is able to perform in a decentralized manner. Discrepancies between several local assignment instances of different followers may arise if two followers are at the exact same minimal distance from two positions in formation (i.e., they form a square where corners alternate between the actual positions and the assigned positions). To avoid this issue, the followers are ordered alphabetically by `agent_id` before proceeding to the assignment and the distances are rounded up to the nearest decimetre.

Moreover, note that $M \geq N-1$, which means some offsets may remain unassigned in the end. This is commonly known as an unbalanced Linear Assignment Problem (LAP) because the two sets to be matched have different sizes. But let us consider the balanced case first where $M = N-1$ and we will generalize to the unbalanced case later.

A brute force solution would be to consider every possible assignment. The latter implies a time complexity of $\mathcal{O}(M!)$ and is obviously not an option unless M is really small. But back in 1955, Harold Kuhn introduced in [68] a method that can solve LAP in polynomial time: the *Hungarian Method*. This optimization algorithm was named after Dénes König and Jenö Egerváry, two Hungarian mathematicians whose works largely inspired Harold Khun.

A mathematical statement of the problem follows. Let c_{ij} be the positive cost resulting from the assignment of agent i to offset j (e.g., the distance between the actual position and position in formation). Then, solving the LAP is equivalent to choosing a set of M independent elements of the matrix $\mathbf{C} = (c_{ij})$ such that the sum of these elements is minimum. Here, a set of elements of a matrix are said to be independent if no two of them lie in the same line where "line" means either "rows" or "columns". Then, Khun invokes two results that structure his algorithm. First, it is readily seen that if a number is added to all of the elements of any line of a cost matrix, then a minimal-cost assignment for the resulting cost matrix is also a minimal-cost assignment for the original cost matrix. Second, the König-Egeváry theorem states: if $\mathbf{C}$ is a matrix, and k is the maximum number of independent zero elements of $\mathbf{C}$, then there are k lines which contain all the zero elements of $\mathbf{C}$.

From here, the *Hungarian Method* can be outlined as follows:

Step A Subtract from each row of $\mathbf{C}$ its smallest element.

Step B Subtract from each column of $\mathbf{C}$ its smallest element.

Step C Find a minimum set of k lines that contains all the zero elements.

Step D If the minimum number of covering lines k equals the matrix size M, then the König-Egeváry theorem concludes that a minimal-cost assignment of zeros is possible and we can proceed to **Step F**. Otherwise, proceed to **Step E**.

Step E Determine the smallest element not covered by any line. Subtract this element from each uncovered row, and then add it to each covered column. Return to **Step C**.

Step F Select a matching by choosing a set of independent zeros so that each row or column has only one selected. This matching has minimal cost and the LAP is solved.

A simple trick to handle unbalanced LAP with the very same algorithm is to add necessary rows or columns to the original matrix in order to obtain a square matrix. The dummy lines added are usually filled with zeros. Of course, this solution is not optimal in terms of time complexity, especially if $M \gg N$. Nevertheless, it could be acceptable in our case because from Section 4.3.2 we have $0 \leq M - N + 1 \leq 11$. Also note that if $M > N - 1$, **Step B** has no effect and could be skipped.

In our particular case, we want to assign a set of M offsets in formation $\mathbf{h}_j \in \mathbb{R}^2$ to the $N-1$ followers in the swarm. To simplify the notations in this section, $\mathbf{r}_i \in \mathbb{R}^2$ denotes the position of follower $i < N$ in the horizontal plane (i.e., the altitude is ignored by the assignment algorithm). We introduce the rectangular cost matrix $\tilde{\mathbf{C}} \in \mathbb{R}^{(N-1)\times M}$ with coefficients $\tilde{c}_{ij}$ such that

$$\tilde{c}_{ij} = \|\mathbf{h}_j - \mathbf{h}_0 + \mathbf{r}_N - \mathbf{r}_i\| + \gamma_j \rho_j d_r, \qquad \text{with} \qquad \gamma_j = \begin{cases} 1, & \text{if } \rho_j = \rho_{max}, \\ 0, & \text{otherwise.} \end{cases} \tag{4.12}$$

The term $\gamma_j \rho_j d_r$ is introduced to penalize the offsets on the periphery of the formation. Finally, the square cost matrix $\mathbf{C} \in \mathbb{R}^{M\times M}$ is obtained by adding $M - N + 1$ dummy rows

$$c_{ij} = \begin{cases} \tilde{c}_{ij}, & \text{if } 1 \leq i \leq N-1, \\ 0, & \text{if } N \leq i \leq M. \end{cases} \tag{4.13}$$

We now illustrate the *Hungarian Method* with a simple applied example. Consider the unbalanced LAP with $M = 5$, $N = 5$ and a given square cost matrix where the last row is obviously a dummy line:

$$\mathbf{C} = \begin{bmatrix} 10 & 10 & 13 & 12 & 14 \\ 19 & 18 & 16 & 19 & 17 \\ 8 & 7 & 9 & 8 & 10 \\ 15 & 17 & 14 & 18 & 19 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{4.14}$$

1. We subtract the row minimum from each row (**Step A**).

$$\begin{bmatrix} 0 & 0 & 3 & 2 & 4 \\ 3 & 2 & 0 & 3 & 1 \\ 1 & 0 & 2 & 1 & 3 \\ 1 & 3 & 0 & 4 & 5 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.15}$$

2. Because $M > N - 1$, subtracting column minima has no effect (**Step B**).

$$\begin{bmatrix} 0 & 0 & 3 & 2 & 4 \\ 3 & 2 & 0 & 3 & 1 \\ 1 & 0 & 2 & 1 & 3 \\ 1 & 3 & 0 & 4 & 5 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.16}$$

3. There are $k = 4$ lines (bold elements) required to cover all zeros (**Step C**). The number of lines is smaller than $M = 5$ so we proceed to **Step E**.

$$\begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{3} & \mathbf{2} & \mathbf{4} \\ 3 & 2 & \mathbf{0} & 3 & 1 \\ \mathbf{1} & \mathbf{0} & \mathbf{2} & \mathbf{1} & \mathbf{3} \\ 1 & 3 & \mathbf{0} & 4 & 5 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix} \tag{4.17}$$

4. The smallest uncovered number is 1. We subtract this number from all uncovered rows and add it to all covered columns (**Step E**).

$$\begin{bmatrix} 0 & 0 & 4 & 2 & 4 \\ 2 & 1 & 0 & 2 & 0 \\ 1 & 0 & 3 & 1 & 3 \\ 0 & 2 & 0 & 3 & 4 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{4.18}$$

5. There are now 5 lines required to cover all zeros (**Step C**). We go to **Step F**.

$$\begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{4} & \mathbf{2} & \mathbf{4} \\ \mathbf{2} & \mathbf{1} & \mathbf{0} & \mathbf{2} & \mathbf{0} \\ \mathbf{1} & \mathbf{0} & \mathbf{3} & \mathbf{1} & \mathbf{3} \\ \mathbf{0} & \mathbf{2} & \mathbf{0} & \mathbf{3} & \mathbf{4} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} \end{bmatrix} \tag{4.19}$$

6. The matching represented by the bold zeros have minimal cost (**Step F**).

$$\begin{bmatrix} \mathbf{0} & 0 & 4 & 2 & 4 \\ 2 & 1 & 0 & 2 & \mathbf{0} \\ 1 & \mathbf{0} & 3 & 1 & 3 \\ 0 & 2 & \mathbf{0} & 3 & 4 \\ 0 & 0 & 1 & \mathbf{0} & 0 \end{bmatrix} \tag{4.20}$$

The former implementation in [68] has time complexity of $\mathcal{O}(M^4)$, but James Munkres has provided a $\mathcal{O}(M^3)$ implementation in [69] a few years later. Remark that the LAP can also be tackled using graph formalism rather than matrix formalism. Graph formalism solves the LAP by finding a minimum-weight perfect matching of a bipartite complete graph. In other words, we build a graph where nodes belong to two disjoint and independent sets (e.g., formation positions and actual followers positions) and all nodes from the first set are linked to all nodes from the second by weighted edges (e.g., the weights equal the distance between formation positions and actual positions). Then, we use graph properties to find an independent edge set such that every node is incident to exactly one edge of this set (i.e., a perfect matching) and the sum of all edge's weights is minimized (i.e., minimum-weight). For instance, the LAPJV named after Roy Jonker and Ton Volgenant who proposed it in [70] is a widely used variant of the *Hungarian Method* relying on graph theory formalism. While the latter has the same $\mathcal{O}(M^3)$ time complexity as Munkres' algorithm, it presents lower computation times and lower sensitivity to the problem's inputs.

In practice, we could have adapted the C++ library in [71], itself based on the Pascal implementation in [70]. Nevertheless, we used the C library `libhungarian-0.3` in [72] based on [68] instead. Indeed, the latter can directly handle unbalanced problem with a time complexity of $\mathcal{O}((N-1)M^2)$ and already provides acceptable computation times.

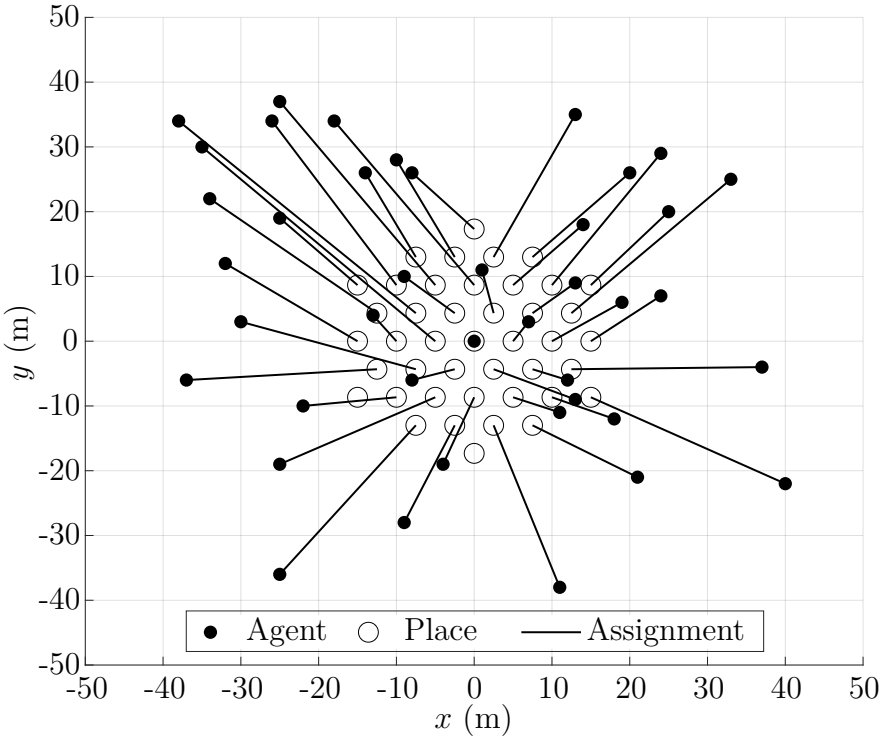


Figure 4.9 Assignment map for $N = 40$, $d_r = 5\,\mathrm{m}$ and $\alpha = 0°$

4.3.3 APF Navigation

We have seen that the position assignment discussed in Section 4.3.2 ensures that followers will not cross each other's path, while moving toward their assigned positions. Although this interesting property effectively limits the collision hazards during both the formation and stop sequences, it does not ensure collision avoidance. Indeed, this does not account for the span of quadrotors and followers may collide with each other even though their respective paths do not intersect.

To deal with collision avoidance, we propose a solution inspired by the APF method used for path planning, hence the name "APF navigation". In fact, a more accurate term would be "physicomimetic navigation". Here, all agents are viewed as positively charged moving particles while the targets (e.g., the assigned positions) would be negatively charged static particles. Then, the velocity control reference that moves one agent toward its target, while avoiding other agents, can be inferred from the resultant electromagnetic forces applied to that agent in the physicomimetic representation. Note that a system block may arise from local minima when using APF method in general. However, in our particular case, this cannot happens because agents' paths do not intersect and there are no obstacles.

Actually, only the horizontal velocity reference input is computed using APF navigation. As for the altitude control, agents are assigned a flight level different than the one of the leader and agents that have already reached their targets. This stratagem makes use of the third dimension in order to reduce the number of avoidance maneuvers required and therefore decrease the time needed for the swarm to reach the formation. This altitude is maintained with the position control of Section 3.2.4 while the horizontal control relies on the velocity control of Section 3.2.3. As a result, the altitude reference input for the agent i using APF navigation is set to $\bar{r}_{z,i} = \mathbf{t}_i^\top \mathbf{z}_\circ - d_c$ where $\mathbf{t}_i \in \mathbb{R}^3$ is the target position of agent i that satisfy $\mathbf{t}_i^\top \mathbf{z}_\circ = \mathbf{t}_j^\top \mathbf{z}_\circ$ for all agents i and j in the swarm. Let $\mathcal{F}_i$ denote the set of agents j that satisfy $\left|(\mathbf{r}_j - \mathbf{r}_i)^\top \mathbf{z}_\circ\right| < \frac{2}{3} d_c$ (i.e., at the same flight level as agent i).

Then, we define the horizontal velocity reference input $\bar{\mathbf{v}}_i \in \mathbb{R}^2$ for the agent i as follows

$$\bar{\mathbf{v}}_i = \begin{cases} \dfrac{\bar{\mathbf{v}}_{\infty,i}}{\|\bar{\mathbf{v}}_{\infty,i}\|} v_{sat}, & \text{if } \|\bar{\mathbf{v}}_{\infty,i}\| > v_{sat}, \\ \bar{\mathbf{v}}_{\infty,i}, & \text{otherwise.} \end{cases} \tag{4.21}$$

By definition, $\|\bar{\mathbf{v}}_i\|$ is bounded by $v_{sat} < v_{max}$ to account for communication delays. In practice, we set $v_{sat} = \frac{1}{10}v_{max}$ while we estimate $v_{max} = \frac{d_c}{\tau_{com}}$. The "unbounded" reference input $\bar{\mathbf{v}}_{\infty,i}$ is the sum of two components enforcing each desired behavior

$$\bar{\mathbf{v}}_{\infty,i} = \mathbf{v}_i^{col} + \mathbf{v}_i^{tar}. \tag{4.22}$$

The first component $\mathbf{v}_i^{col}$ enforce collision avoidance behavior. Hence, $\mathbf{v}_i^{col}$ is built so that agent i is moved away from its neighbors. Therefore, we define $\mathbf{v}_i^{col}$ as follows

$$\mathbf{v}_i^{col} = \sum_{j \in \mathcal{F}_i(\mathcal{G}_\sigma)} \psi_{col}(\|\mathbf{r}_j - \mathbf{r}_i\|) \frac{\mathbf{r}_j - \mathbf{r}_i}{\|\mathbf{r}_j - \mathbf{r}_i\|}, \tag{4.23}$$

with

$$\psi_{col}(z) = \begin{cases} v_{sat}\left(\dfrac{z}{d_c} - 2\right), & \text{if } z < 2d_c, \\ 0, & \text{otherwise.} \end{cases} \tag{4.24}$$

The second component $\mathbf{v}_i^{tar}$ ensures target attraction so that agent i moves horizontally toward its target $\mathbf{t}_i$. $\mathbf{v}_i^{tar}$ is defined as follows

$$\mathbf{v}_i^{tar} = \psi_{tar}(\|\mathbf{H}\mathbf{t}_i - \mathbf{r}_i\|) \frac{\mathbf{H}\mathbf{t}_i - \mathbf{r}_i}{\|\mathbf{H}\mathbf{t}_i - \mathbf{r}_i\|}, \tag{4.25}$$

with

$$\psi_{tar}(z) = \begin{cases} v_{sat}\dfrac{z}{d_c}, & \text{if } z < d_c, \\ v_{sat}, & \text{otherwise,} \end{cases} \qquad \text{and} \qquad \mathbf{H} = \begin{bmatrix} \mathbf{x}_\circ & \mathbf{y}_\circ \end{bmatrix}^\top. \tag{4.26}$$

On the one hand, it appears clearly on Figure 4.10 that $\psi_{col}(z)$ is a negative continuous function for $0 \leq z$ with $\psi_{col}(z) \leq -v_{sat}$ for $z \leq d_c$. On the other hand, it can be seen that $\psi_{tar}(z)$ is a positive continuous function for $0 \leq z$ where $0 \leq \psi_{tar}(z) \leq v_{sat}$. As a result, the collision avoidance component $\mathbf{v}_i^{col}$ can effectively compensate the target attraction component $\mathbf{v}_i^{tar}$. In other words the collision avoidance behavior can override the target attraction behavior if necessary. Moreover, note that $\psi_{tar}(z) = v_{sat}$ for $d_c \leq z$ and $\psi_{col}(z) = 0$ for $2d_c \leq z$. This implies that the agent moves at full speed toward its target as long as there is no risk of collision.

Once the agent is close enough to its target, APF navigation switches to position control on the three dimensions with reference input $\bar{\mathbf{r}}_i = \mathbf{t}_i$. Here, agent i is said to be "close enough" when he satisfies

$$\|\mathbf{H}(\mathbf{t}_i - \mathbf{r}_i)\| < \frac{d_c}{2}. \tag{4.27}$$

As described in Figure 4.3, the APF navigation procedure ends when the agent finally reaches its target (i.e., condition "arrived on target" is true) , that is when

$$\|\mathbf{t}_i - \mathbf{r}_i\| < \varepsilon_r, \tag{4.28}$$

where ε_r is a parameter that influences the navigation precision. In practice, we take $\varepsilon_r = \frac{d_c}{10}$.

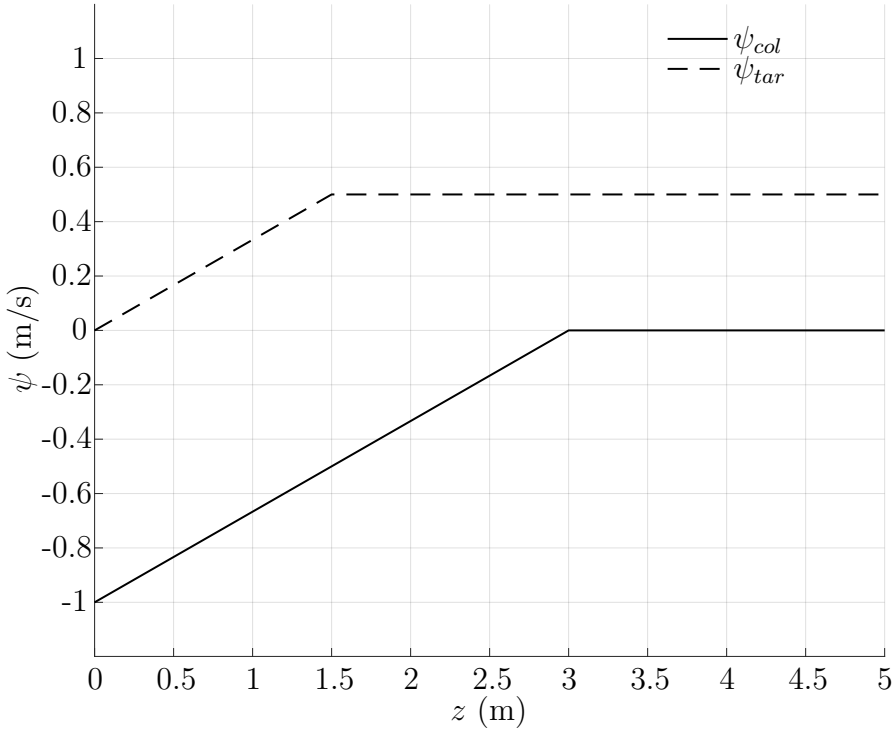


Figure 4.10 $\psi_{col}(z)$ and $\psi_{tar}(z)$ for $d_c = 1.5\,\mathrm{m}$ and $v_{sat} = 0.5\,\mathrm{m/s}$

4.4 Software Design

4.4.1 Development Environment

ROS

The embedded software for quadrotors, under development at Humanitas Solutions, largely relies on ROS, more precisely on the Kinetic Kame distribution. Contrary to what its name might suggest, ROS is not a proper Operating System (OS), but an open-source middleware supported by Open Robotics and running on Unix-based platforms (essentially, Ubuntu and Mac OS X). Although ROS is not an operating system, it provides services such as hardware abstraction and low-level device control. It is also worth noting that ROS is not a realtime framework (even less a Real-Time Operating System (RTOS)), though it is possible to integrate ROS with realtime code. On their web page [73], Open Robotics summarizes the philosophy of ROS in these terms: "The primary goal of ROS is to support code reuse in robotics research and development. ROS is a distributed framework of processes (i.e., nodes) that enables executables to be individually designed and loosely coupled at runtime. These processes can be grouped into Packages and Stacks, which can be easily shared and distributed." The "distributed framework of processes" evoked in the latter is actually represented as a graph: the "computation graph" as depicted in Figure 4.11. The remainder of this section outlines some basic computation graph concepts of ROS, namely nodes, Master, messages, and topics. Note that ROS also features a specific file system that will not be viewed here. For more details on ROS, one can visit `http://wiki.ros.org`.

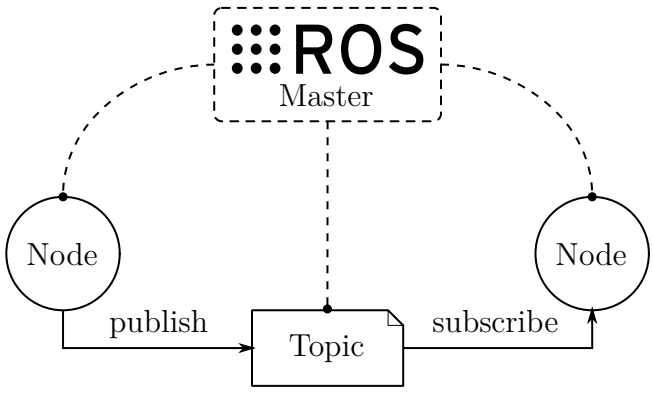


Figure 4.11 ROS computation graph

A node represents a single process in the computation graph. There is no limit to the complexity of the computation handled by a node, but the potential of ROS is best exploited when nodes are kept to be minimal functional units. This enables a certain versatility and re-usability of developed nodes. A robot control system usually comprises many nodes. For instance, one node controls the propellers' motors, one node performs localization, one node performs path planning, one node handles communication, and so on. A ROS node can either be written in C++ or in Python with the use of a ROS client library, respectively `roscpp` or `rospy`. The ROS Master supervises all running nodes. For instance, it provides name registration of nodes and set up node-to-node communication. Without the Master, nodes would not be able to find each other and exchange messages. In ROS, messages are simply data structures, defined by a given set of typed fields. Standard primitive types (integer, floating-point, boolean, etc.), arrays of primitive types and even nested structures are supported. Messages are routed via a transport system with publish/subscribe semantics. For instance, nodes send out a message by publishing it to a given topic. Then, nodes subscribed to this particular topic will receive the message.

Humanitas Ecosystem

Over the years, Humanitas Solutions has built a complete ecosystem to support the deployment of UAV swarms in a distributed fashion. As depicted in Figure 4.12, the Humanitas ecosystem is a distributed heterogeneous MAS where agents can be categorized into two groups: users and robots. Users, equipped with digital tablets, can monitor all the other agents and control the robots. All agents are connected via wireless links over HEAVEN, a distributed network solution developed by Humanitas Solutions. Among the supported wireless interfaces let us mention Wi-Fi (in both AP, STA and IBSS modes), Bluetooth and xBee 900 MHz. With Wi-Fi interfaces, the bandwidth achieved can reach 500 Mbps, while the range depends on the type of antenna and can vary from ten of meters to 15 km. At network level, the single-hop round-trip-time for devices over ad hoc Wi-Fi has been measured to be around a millisecond. The communication delay also depends on the refresh rates of the embedded processes. As a result, the effective single-hop one-trip-time τ_{com} experienced at agent level is often greater than a millisecond.

The hardware architecture of the quadrotors developed at Humanitas Solutions is summarized in Figure 4.13. Quadrotors embed four kinds of functional units: power units, actuators, sensors, and processing units. All embedded electronic components are powered by a Lithium Polymer (LiPo) battery through a power module that ensures isolated and reliable voltage supplies. As for actuators, the quadrotor simply has four Direct Current (DC) brushless

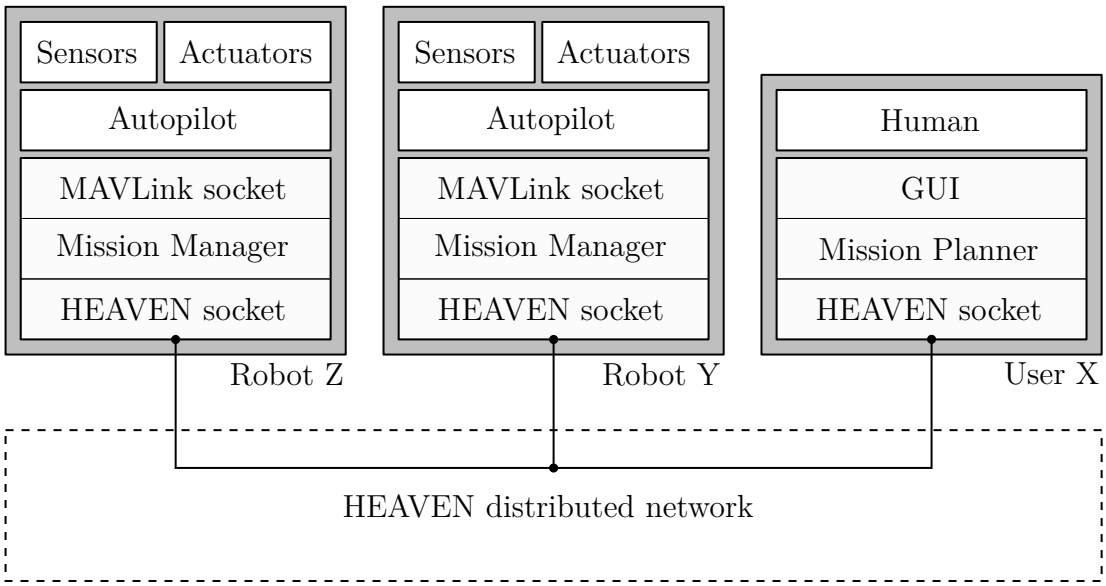


Figure 4.12 Humanitas ecosystem

motors paired with a 4in1 Electronic Speed Control (ESC) (Electronic Speed Controls). The ESC has BLHeli-S DShot firmware installed. The sensors consist of a monocular camera, a GNSS module, a barometer and a nine-axis IMU (itself including one three-axis gyroscope, one three-axis accelerometer, and one three-axis compass). Finally, Humanitas quadrotors are equipped with two processing units. The autopilot handles sensor fusion, basic localization, and low-level control, while the companion computer deals with high-level control, advanced localization, navigation, and communication. In other words, the autopilot handles time-critical tasks, while the companion computer can run complex and time-consuming processes. The autopilot board runs PX4, an open-source autopilot software, on NuttX RTOS. The companion computer has Linux Ubuntu 16.04 installed with ROS Kinetic Kame distribution. It runs ROS nodes along with the HEAVEN application developed by Humanitas Solutions. Besides the Wi-Fi module connected to the companion computer to handle HEAVEN communication, the autopilot board is equipped with a radio receiver that enables direct control of the quadrotor with an Radio Controlled (RC) controller. This feature is particularly helpful when testing. Note that the IMU, the barometer and an additional compass are built-in the Pixracer autopilot board. Moreover, other long range wireless interfaces can be added when needed.

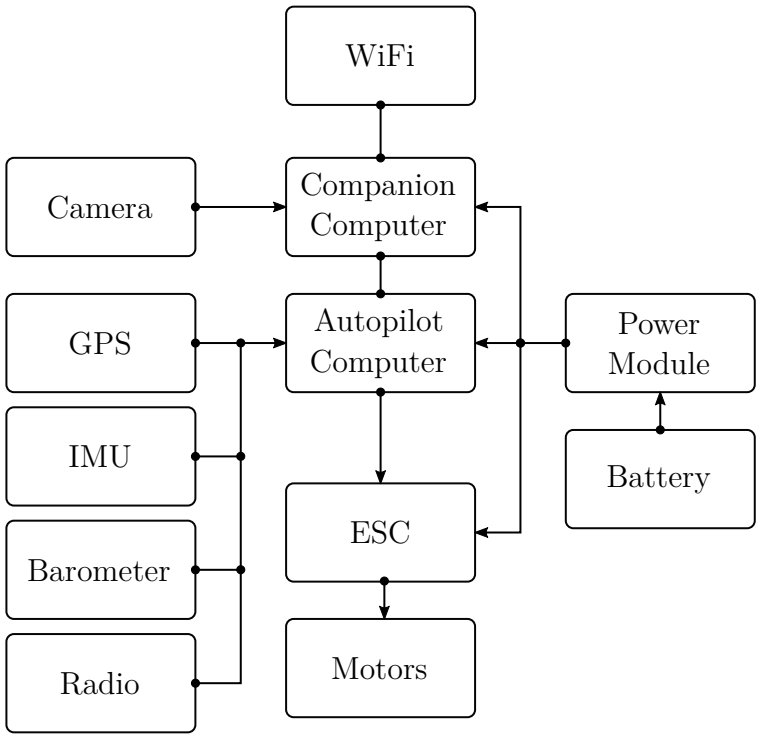


Figure 4.13 Humanitas quadrotor architecture

4.4.2 Embedded Software

According to the description of the development environment in the previous section, the swarming algorithm developed in Section 4.3 runs on the companion computer. At software level, it is integrated into the Mission Manager. As displayed in Figure 4.12, the Mission Manager communicates with the autopilot through the MAVLink (where MAV stand for Micro Air vehicle) socket and with other agents through the HEAVEN socket. As a result, the remainder of this section first focuses on the Mission Manager and then briefly covers HEAVEN and MAVLink sockets.

Mission Manager

The Mission Manager is an early version of Humanitas swarming engine that was used to test the algorithms developed in this project. Its main purpose is to handle both the distribution and the execution of the tasks sent to robots by users. When a user sends a task to the swarm, the mission manager usually assigns the task to a unique agent with an auction-based distributed method that will not be covered here. Indeed, in our particular case, the task (i.e., the role in the swarm) is directly assigned by the user.

MAVLink Socket

MAVLink is a very lightweight messaging protocol for communicating with UAVs and between onboard components. The MAVLink socket actually consists of one node providing ROS topics and service servers to let the mission manager communicate with the autopilot. The MAVLink package also defines a certain set of standardized messages/types readable by the Px4 autopilot software (e.g., `GlobalPositionInt`, `LocalPositionNED`, etc.).

HEAVEN Socket

The HEAVEN socket essentially provides ROS topics to let the mission manager node share information with other agents through HEAVEN network. It consists of two nodes in series. On the one hand, the node on the side of the mission manager serializes/deserializes the messages sent/received.On the other hand, the node on the side of the HEAVEN application handles, amongst other things, port opening and peer scanning.

4.5 Simulation

4.5.1 Test Setup

To asses the validity of the developed solution, we ran SITL simulations using the Humanitas simulator to model the quadrotor aerodynamics, the dynamics of the actuators and generate the sensor feeds accordingly. Figure 4.14 summarizes the test setup.

The first computer runs several instances of the quadrotor onboard software and exchanges data through a local network with the second computer that runs the simulator. Note that the main specifications of the computers used for the test are available in Table 4.1.

Several software instances are actually encapsulated in several Docker containers. A Docker container is a standardized and isolated unit that wraps up a piece of software and its dependencies for development purposes. This mimics the kind of isolation obtained with Virtual Machines (VMs) but it saves the kernel space of an OS and does not need to run hardware virtualization. In practice, the containers are all instances of the same Docker image. As depicted in Figure 4.16, the HEAVEN application, the mission manager software, and the autopilot firmware are all installed on the image.

The Humanitas simulator has been developed with Unreal Engine, a source-available commercial game engine with a royalty model for commercial use. Typically, this software includes a physics engine and a rendering engine for 3D graphics. Figure 4.15 show a desaturated preview of the graphics rendered by the simulator.

In its current version, the simulator models quadrotor aerodynamics and actuator dynamics and ideal sensor values (i.e., the estimators are directly fed with true values, not synthetic measurements). Note that the simulator models the AR Drone 2.0 equipped with its indoor hull, a quadrotor distributed by Parrot.

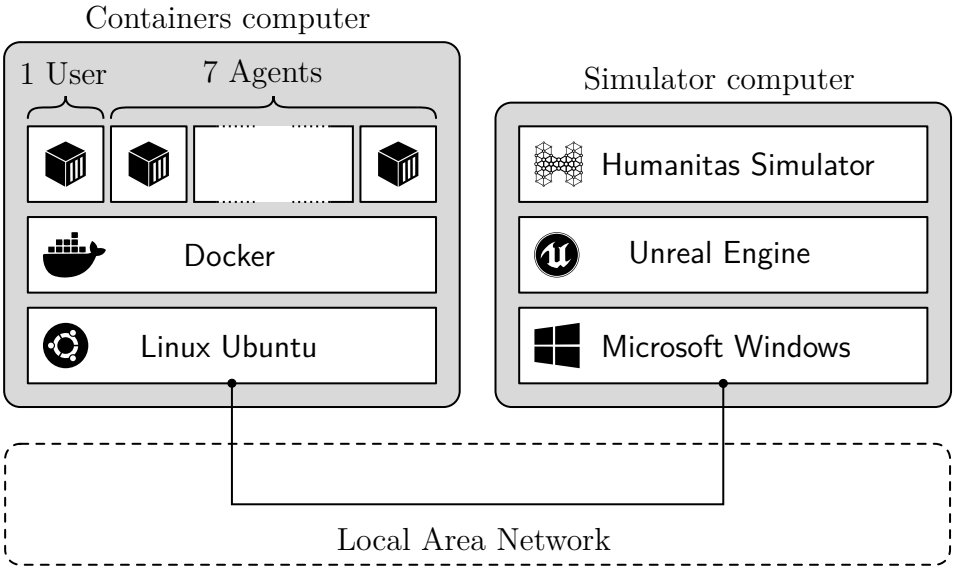


Figure 4.14 Test setup

Figure 4.15 Simulator visualization

Table 4.1 Test computers specifications

Containers computer	
Description:	Notebook
Vendor:	Lenovo
Version:	ThinkPad T420
Processor:	Intel(R) Core(TM) i5-2520M CPU @ 2.50GHz 2 Cores
Memory:	10GB
Graphics:	Integrated Intel HD Graphics 3000
OS:	Linux Ubuntu 16.04.6 LTS (64bits)

Simulator computer	
Description:	Notebook
Vendor:	Lenovo
Version:	Ideapad 330-15ICH-81FK
Processor:	Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz 6 Cores
Memory:	16GB
Graphics:	NVIDIA GeForce GTX 1050 2GB
OS:	Microsoft Windows 10 Home (64bits)

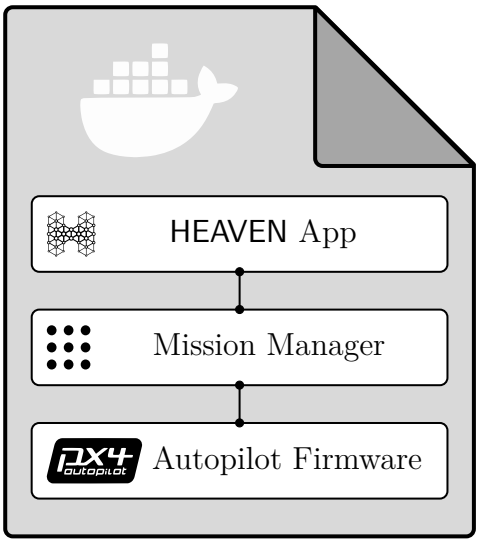


Figure 4.16 Docker image

4.5.2 Test Design

During the test, a swarm of $N = 8$ agents will have to make a formation, visit five waypoints forming a square and then go back home safely. In particular, the first and the last waypoint are roughly placed above the home position of the swarm. Global and local coordinates of the waypoints are given in Table 4.2.

More precisely, global coordinates are expressed in the World Geodetic System 84 (WGS84) and local coordinates are given in the initial leader NED frame. The initial positions (i.e., the home position) of agents are defined by design in the simulator and cannot be modified. Actually, the simulator places the quadrotors along the East axis of the NED frame, evenly spaced by $d_i \approx 1.5\,\mathrm{m}$.

For test automation purposes, we designed an automated user node widely based on the software onboard quadrotors. That node identifies the agents at power-up and then generates and sends a task to newly identified agents. Actually, the first identified agent will be the leader and all other agents will be followers. As a result, the leader usually happens to be the first or the second quadrotor created by the simulator. Note that this is a "hard" initial state for our swarming algorithm. Indeed, all agents are aligned, the leader is far from the center of the swarm and agents are spaced by less than the set collision distance $d_c = 1.65\,\mathrm{m}$.

All parameter definitions and values used for this test are summarized in Table 4.3. Nevertheless, it has to be noted that the results discussed in the next section have been obtained with an early iteration of the solution presented in previous sections. For instance, this version features slightly different controller structures and gains (e.g., no integral action and no velocity limiter). This leads to diminished performance with respect to the current version but does not invalidate the discussion on results in the next section. One can observe two other differences with the current version. First, once the leader reaches a waypoint, it waits for all followers to be precisely at their assigned offset which slows down the whole process. Second, in the end, the leader waits for all followers to go home before going home himself. This also tends to slow down the mission.

Table 4.2 Test path

Waypoint	**Global Coordinates** (DD,DD,m)	**Local Coordinates** (m,m,m)
1	45.4583 , −73.6346 , 59.0	0.00 , −0.00 , 4.0
2	45.4584 , −73.6346 , 59.0	11.0 , −0.00 , 4.0
3	45.4584 , −73.6348 , 59.0	11.0 , −15.5 , 4.0
4	45.4583 , −73.6348 , 59.0	0.00 , −15.5 , 4.0
5	45.4583 , −73.6346 , 59.0	0.00 , −0.00 , 4.0

Table 4.3 Test parameters

Definition	**Symbol**	**Value**
Sampling period	T_s	100 ms
Number of agents	N	7
Quadrotor diameter	d_q	55 cm
Initial spacing	d_i	150 cm
Collision-free distance	d_c	165 cm
Formation spacing	d_r	500 cm
Formation yaw	α	0°
Command saturation	u_{sat}	2 m/s^2
Velocity saturation	v_{sat}	0.5 m/s
Control tolerance	ε_r	50 cm
Catch-up threshold	$\Delta_{\mathcal{W},th}$	200 cm
Data storage time	T_{msg}	2 s
Initialization delay	T_{ini}	5 s
Follower delay	T_{fol}	10 s
Leader delay	T_{led}	10 s
Catch-up delay	T_{del}	5 s
Formation delay	T_{for}	40 s
End delay	T_{end}	10 s

4.5.3 Results

The "raw" material obtained from simulations is presented here. Figures 4.17-4.19 presents the states of the leader and a given follower throughout the test. In addition, Figure 4.20 displays the position of agents in the horizontal plane at three instants of interest: before the swarm starts forming, when the swarm has reached the defined formation, and when the follower start moving, a few moments after the leader has done the same. A full animated version of Figure 4.20 is available at `https://youtu.be/hSyYxYcHMxk` and the video rendered by the simulator is available at `https://youtu.be/V_h7y1PrMFk`.

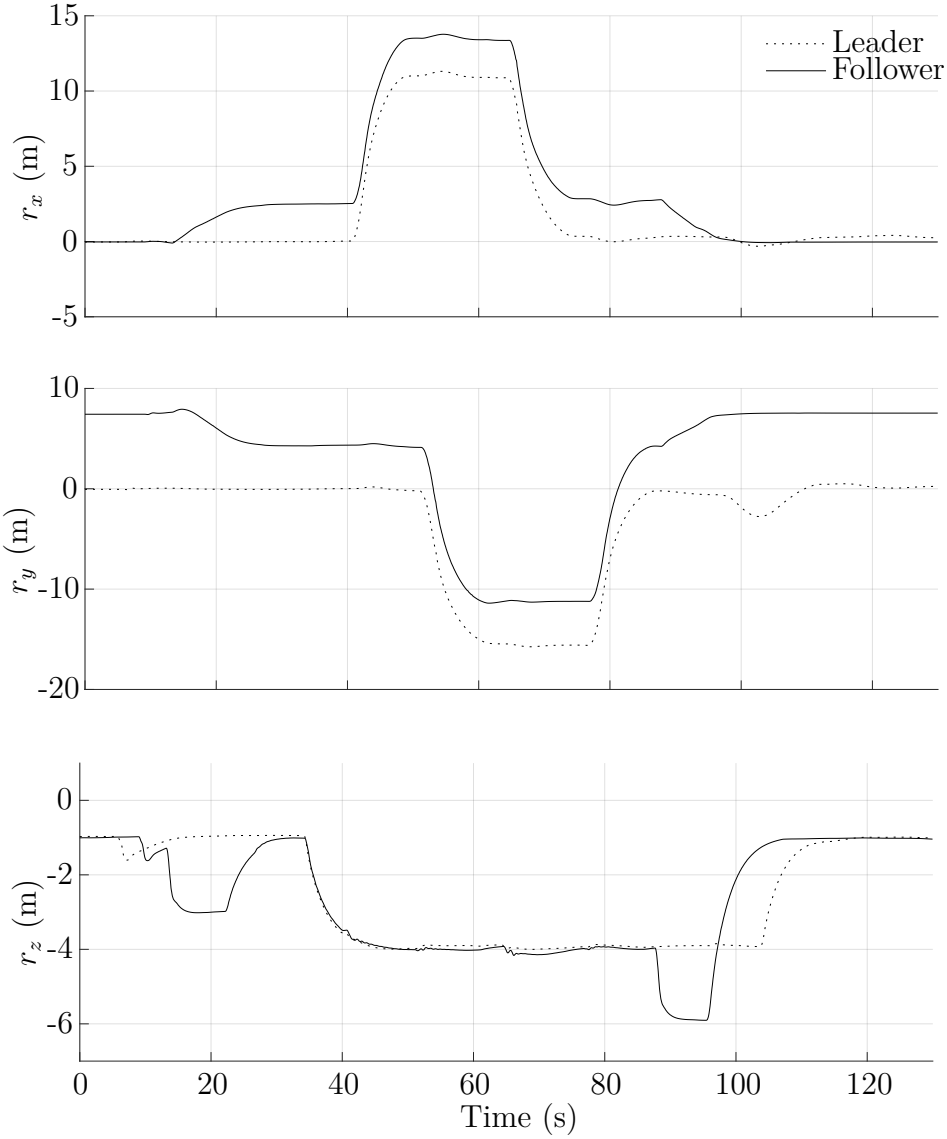


Figure 4.17 Position of the leader and a follower

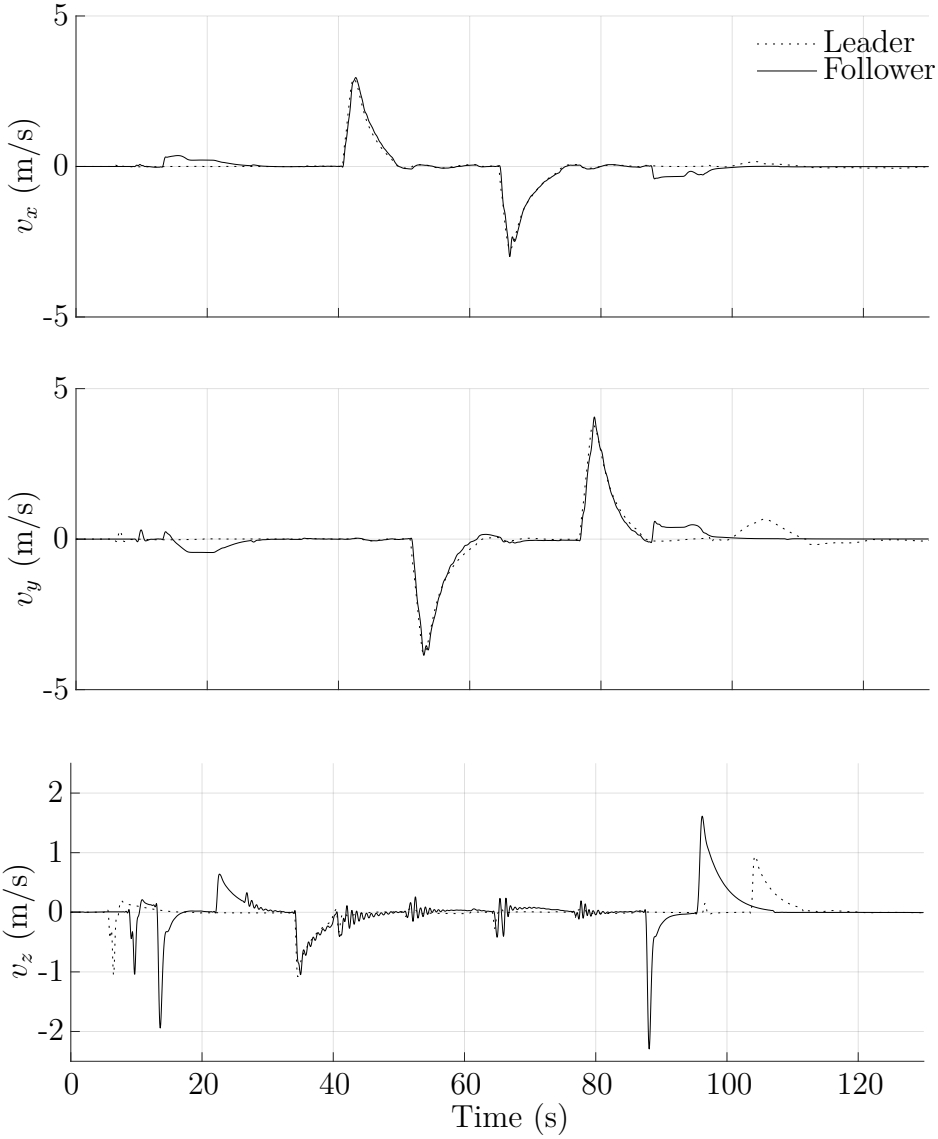


Figure 4.18 Velocity of the leader and a follower

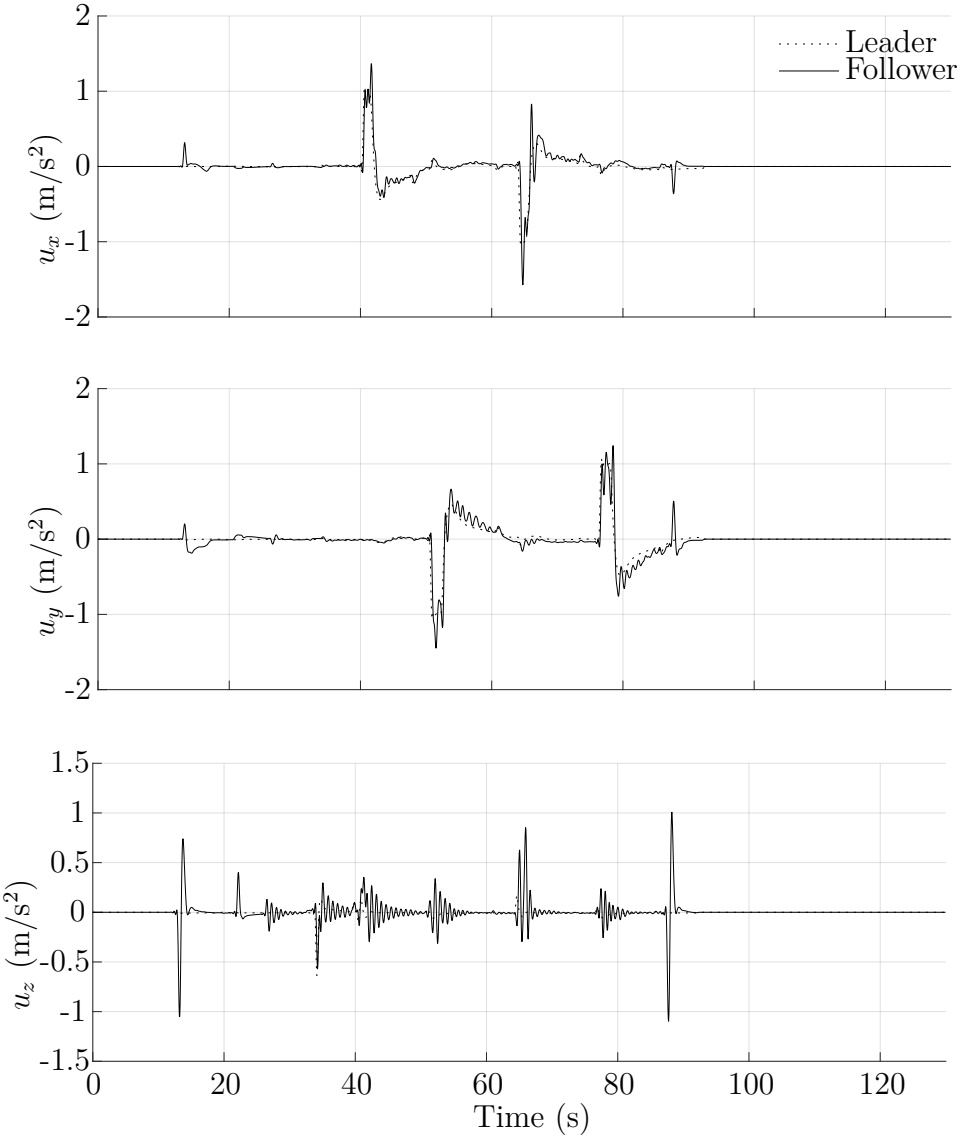


Figure 4.19 Command input of the leader and a follower

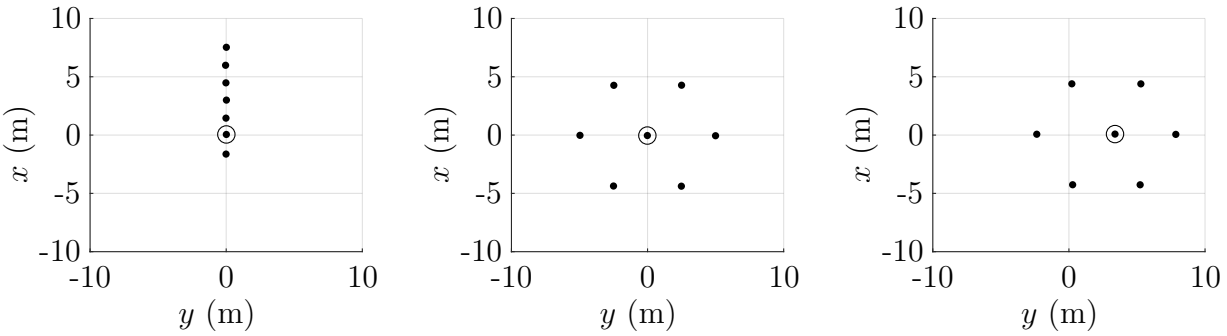


Figure 4.20 Snapshots of the swarm in initial, forming and following sequences

One can read on Figures 4.17-4.19 the workflow defined in Section 4.3.1. For instance, at $t = 10\,\text{s}$, the follower receives the task from the user and targets its home position. Specificities of the early controller design explain the bounce in altitude at that moment. Then, between $t = 13\,\text{s}$ and $t = 22\,\text{s}$, it uses the APF navigation algorithm to go above its position in formation. At $t = 32\,\text{s}$, the follower finally reaches his position in formation. Two seconds later, the leader starts moving toward the first waypoint. Then it visits waypoints 1,2,3,4 and 5 at $t = 40, 51, 64, 76, 87\,\text{s}$ respectively. Once it reaches the last waypoint, the leader sends the stop signal and the follower uses the APF navigation algorithm once again to go back home. The latter reaches home at $t = 110s$. Once all followers are home, the leader finally goes home and arrives there at $t = 120s$. The mission is completed.

On the one hand, it can bee seen on Figure 4.21 that during the following sequence, all agents manage to keep an average distance $d_n = 5\,\text{m}$ from their neighbors. More over, that distance is always greater than $4.2\,\text{m}$ and never exceeds $5.8\,\text{m}$. On the other hand, Figure 4.22 shows that the proposed solution enforce efficient collision avoidance during forming sequence, even with an initial state that does not validate the collision avoidance condition (i.e., $d_i < d_c$). In addition, Figure 4.24 shows that the average tracking error, i.e., distance between the actual position and the position in formation, remains below $0.8\,\text{m}$. This is acceptable with respect to the collision-free distance $d_c = 1.65\,\text{m}$. Finally, Figure 4.23 provides an insight on the communication delay. It can actually be inferred from that plot that the single-hop communication delay is $\tau_{com} \approx 0.2\,\text{s}$ which respects the condition $\tau_{com} \leq 3T_s$.

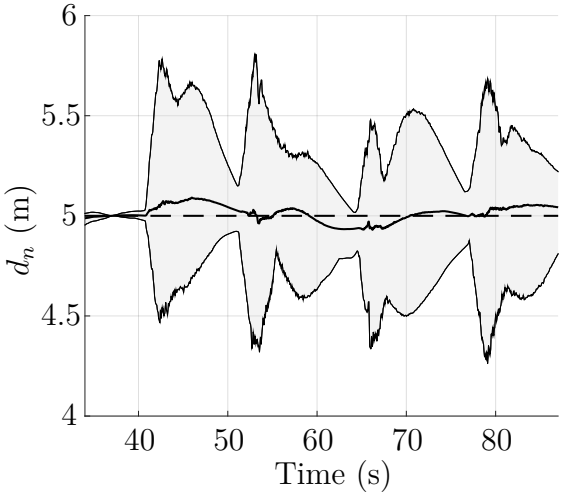


Figure 4.21 Average neighbors distance

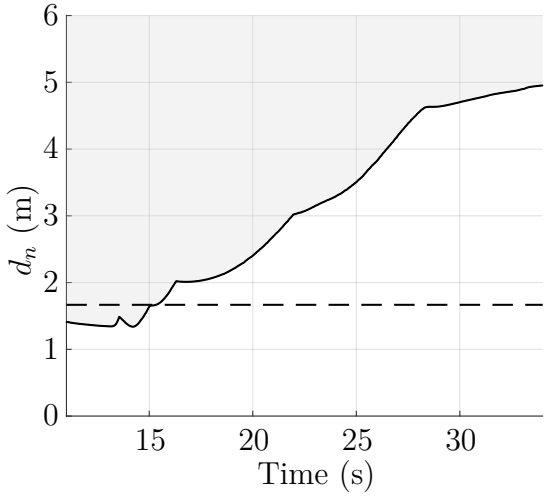


Figure 4.22 Minimal neighbors distance

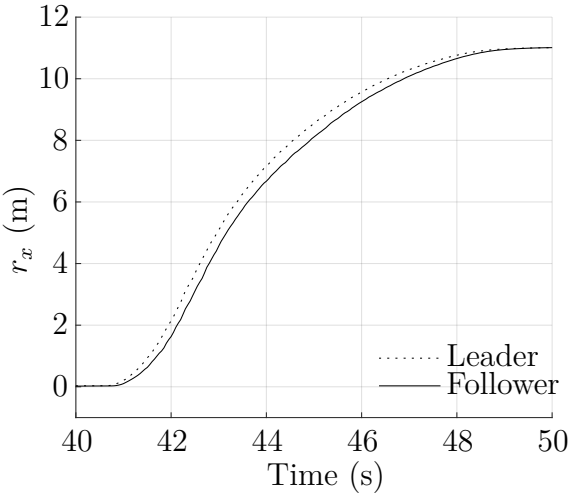


Figure 4.23 Zoom on following delay

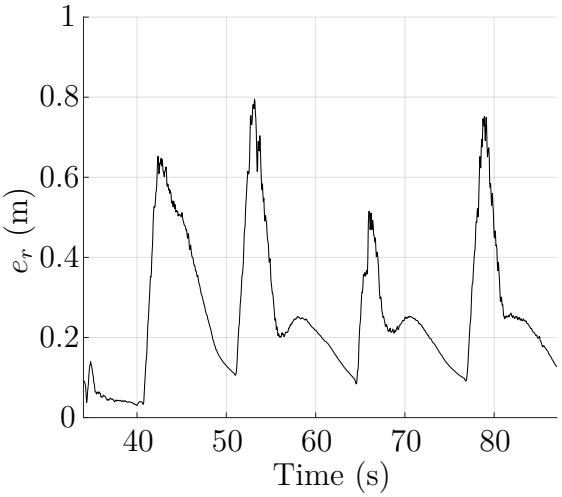


Figure 4.24 Average tracking error

CHAPTER 5 INDOOR SWARMING

5.1 Introduction

To the best of our knowledge, most of the swarming solutions in the literature involve a rigid formation where each agent is assigned to a certain position within the swarm. This implies a certain level of centralization which limits the scalability and the modularity of the swarm. Moreover, rigid formations could be interesting for outdoor applications but are not best suited for indoor navigation as obstacle avoidance becomes a major concern. Also, one has to keep in mind that the final aim of this project is to implement the swarming solution on nano-quadrotors. Hence, the swarming algorithm has to be sufficiently simple to be able to run on board in real-time.

In that regard, Sakai et al. [5] have proposed a simple control method for leader-follower swarming in cluttered indoor environments. They demonstrate their solution performance through experiments operating seven two-wheel differential drive robots. Strictly speaking, leader-follower approaches enforce a centralized architecture. In their paper, a hybrid solution fusing the behavior-based approach and the leader-follower approach is developed. An implicit leader tracks a predefined trajectory while the rest of the swarm follows. This approach differs from classical leader-follower strategies as the followers will simply maintain a certain distance from their neighbors and not track a predefined offset position from the leader. In fact, the followers are only given a certain set of simple local rules that lead to the emergence of global behavior. Moreover, they introduce a simple rule for changing the network topology depending on environments, giving the swarm the ability to pass through narrow spaces.

Following this framework, we identified two major problems that need to be addressed. On one hand, a swarming algorithm has to be designed to maintain formation and connectivity while avoiding collisions and obstacles. On the other hand, agents have to be able to deactivate/activate links with their neighbors in order to pass obstacles, while maintaining global connectivity among the swarm.

This chapter aims to present the problem setting as described in [5]. Then, a new distributed swarming algorithm is developed based on the solution proposed in [5]. Finally, simulations are performed to demonstrate solution effectiveness.

5.2 Problem Setting

5.2.1 Assumptions

Swarm control and indoor navigation are complex problems. For the purpose of focusing only on the formation control, simplifying assumptions have been made:

- *Single integrator dynamics* - The agents embed a robust low-level position controller such that their dynamics can be assimilated to an ideal 2D single integrator.
- *Point mass model* - The agents are considered as point masses. As such, orientation of the agent will not be explicitly considered.
- *Full-state estimate* - The agents can estimate their full state using the onboard computer and sensors (i.e., position and velocity in 2D).
- *Detection range* - The agents are able to detect obstacles and other agents in direct LOS within a certain range and in every direction.
- *Communication range* - The agents can communicate with all detected agents. However, only one-hop communication is authorized
- *Real-time constraints* - The low-level control, the estimator, detection and communication algorithms perform at higher rates than the swarming algorithm. Communication delays are sufficiently low.

5.2.2 Dynamic Model

We consider a swarm of N agents in a 2D environment with obstacles. We assume that a target path is given to only one of the N agents, called the *leader*, whose index is set as N without loss of generality. Other agents $i \in [1, N-1]$ are called *followers*. According to the above assumptions, the dynamics of the ith agent ($i = 1, 2, \ldots, N$) can be described by the following discrete-time state equation [5]

$$\mathbf{r}_i(k+1) = \mathbf{r}_i(k) + \mathbf{u}_i(k), \quad \|\mathbf{u}_i(k)\| \leq u_{max}, \tag{5.1}$$

where $\mathbf{r}_i(k) \in \mathbb{R}^2$ and $\mathbf{u}_i(k) \in \mathbb{R}^2$ are respectively the position and the control input of agent i at time step $k \in \mathbb{N}^+$. The input bound u_{max} represents the hardware limitation and depends on the sampling period T_s.

5.2.3 Collision Model

Even though the simplification of the dynamic model has led us to consider point mass agents for the dynamic model, we need to assign a certain footprint to agents in order to model the collision and obstacle avoidance problem. Thus, a sufficient condition for collision avoidance for agent i is given as follows

$$\|\mathbf{r}_i - \mathbf{r}_j\| \geq d_c, \quad \forall j \in \mathcal{V}\backslash\{i\}, \tag{5.2}$$

where $\mathcal{V} := \{1, 2, \ldots, N\}$ is the set of indices of all agents. The distance d_c represents roughly the diameter of the agents. We also assume that a sufficient condition for obstacle avoidance is given by

$$\|\mathbf{r}_i - \mathbf{r}_o\| \geq d_o, \quad \forall \mathbf{r}_o \in \mathcal{O}, \tag{5.3}$$

where $\mathcal{O}$ is the set of all points on obstacles in the environment.

5.2.4 Sensing Model

According to the assumption made, agents have a limited sensing range. Moreover, they can only detect elements in direct LOS. To begin, let us define the line segment joining agents i and j as

$$\mathcal{L}_{ij} = \{(1-\lambda)\mathbf{r}_i + \lambda\mathbf{r}_j, \forall\lambda \in [0,1]\}, \tag{5.4}$$

and the ray from agent i passing by obstacle o

$$\bar{\mathcal{L}}_{io} = \{(1-\lambda)\mathbf{r}_i + \lambda\mathbf{r}_o, \forall\lambda \geq 0\}. \tag{5.5}$$

Then, we assume that agent i can retrieve agent j's relative position

$$\mathbf{r}_{ji} = \mathbf{r}_j - \mathbf{r}_i, \tag{5.6}$$

if the following conditions are both satisfied

$$\|\mathbf{r}_j - \mathbf{r}_i\| \leq d_s, \tag{5.7}$$

$$\|\mathbf{q} - \mathbf{r}_o\| \geq d_l, \quad \forall \mathbf{q} \in \mathcal{L}_{ij}, \quad \forall \mathbf{r}_o \in \mathcal{O}, \tag{5.8}$$

where the distance d_s defines the maximum sensing range and d_l describes a minimum clearance distance between the segment joining the agents and each obstacle. Figure 5.1 presents an example of sensing range.

In the same fashion, agent i is able to detect a point on an obstacle $\mathbf{r}_o \in \mathcal{O}$, if

$$\|\mathbf{r}_o - \mathbf{r}_i\| \leq d_s, \tag{5.9}$$

$$\|\mathbf{r}_o - \mathbf{r}_i\| \leq \|\mathbf{q} - \mathbf{r}_i\|, \quad \forall \mathbf{q} \in \bar{\mathcal{L}}_{io} \cap \mathcal{O}. \tag{5.10}$$

The second condition means that an obstacle cannot be detected if another one is hiding it from the agent. However, we assume that an agent can be detected even if a third agent interrupts the LOS.

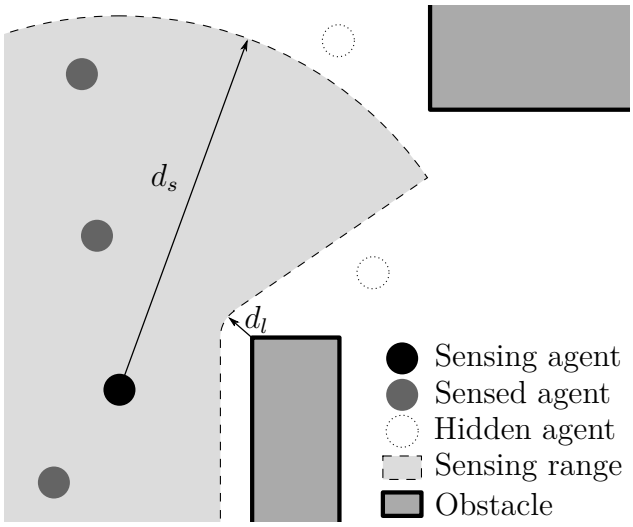


Figure 5.1 Sensing range

5.2.5 Network Topology

A common way to model and study swarm organization is using *graph theory*. In short, a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a finite set of *nodes* (or *vertices*) $\mathcal{V}$ associated with a finite set of *edges* (or *links*) $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$. For instance, the nodes represent the agents and the edges represent the interactions and communications between agents. The directed edge from nodes i to j is denoted $e = (i, j)$. In this work, we assume that agents can perform two-way communication. Therefore, we will only consider the case of *simple undirected graphs*, where respectively, $(i, i) \notin \mathcal{E}$ and $(i, j) \in \mathcal{E} \Leftrightarrow (j, i) \in \mathcal{E}$. A graph $\mathcal{G}_{sub} = (\mathcal{V}, \mathcal{E}_{sub})$ is a *subgraph* of graph $\mathcal{G}$, denoted by $\mathcal{G}_{sub} \subseteq \mathcal{G}$, if $\mathcal{E}_{sub} \subseteq \mathcal{E}$. For each node $i \in \mathcal{V}$, we define the set of *neighbors* $\mathcal{N}_i(\mathcal{G}) = \{j \in \mathcal{V} \,|\, (i, j) \in \mathcal{E}\}$. A path on $\mathcal{G}$ is a finite sequence of unique nodes such that every two consecutive nodes are neighbors (i.e., connected by an edge). The number of nodes in the path is the length of the path. A graph is *connected* if and only if for each pair of distinct nodes there exists a path that contains both nodes. By definition, if the subgraph

$\mathcal{G}_{sub}$ is connected, then the graph $\mathcal{G}$ is connected. The topological distance of two nodes i and j of a connected graph, denoted by $\text{dist}(i,j)$, is equal to the length of the shortest path in the graph that contains both nodes.

In terms of the sensing described above, let us define the sensing graph $\mathcal{G}_s(\mathbf{r}(k)) = (\mathcal{V}, \mathcal{E}_s(\mathbf{r}(k)))$ where $\mathbf{r} = \begin{bmatrix}\mathbf{r}_1^\top & \mathbf{r}_2^\top & \dots & \mathbf{r}_N^\top\end{bmatrix}^\top$. The edge set $\mathcal{E}_s$ gathers all couples of agents that are able to detect each other at time k (i.e., conditions (5.7) and (5.8) are verified). Note that $\mathcal{G}_s$ has to be connected at all times, and especially at initial time $k = 0$ in order to be able to perform formation control. That condition on connectivity maintenance also implies that the distance between the leader and each follower is kept under $(N-1)d_s$ as the maximum length of each edge in $\mathcal{G}_s$ is kept under d_s. In order to simplify the network topology, reduce the number of edges and subsequently maintain the communication between agents to a minimum level, we will rather consider the following subgraph of $\mathcal{G}_s$:

$$\mathcal{G}_n(\mathbf{r}(k)) = (\mathcal{V}, \mathcal{E}_n(\mathbf{r}(k))), \tag{5.11}$$

$$\mathcal{E}_n(\mathbf{r}(k)) = \{(i,j) \in \mathcal{E}_s(\mathbf{r}(k)) \,|\, \|\mathbf{r}_j - \mathbf{r}_i\| \leq d_n\}, \tag{5.12}$$

where $d_c < d_n < d_s$. By definition, we know that if $\mathcal{G}_n$ is connected, then $\mathcal{G}_s$ is connected as well. Therefore our goal will be to maintain the connectivity of $\mathcal{G}_n$ in the following.

5.3 Swarming Algorithm

5.3.1 Edge Management

We assume that $\mathcal{G}_n$ is connected at the initial time $k = 0$. Then, the easiest way to maintain connectivity would be to control the agents so that the edges in $\mathcal{E}_n$ are preserved for all $k > 0$. However, in order to be able to move in cluttered environments, some edges might have to be deactivated so the formation can squeeze through narrow spaces. This finally implies that edges will have to be reactivated as soon as possible to enable cohesive formation control. In order to deal with the edge management, Sakai et al. [5] introduces a subgraph of $\mathcal{G}_n$

$$\mathcal{G}_\sigma(\mathbf{r}(k)) = (\mathcal{V}, \mathcal{E}_\sigma(\mathbf{r}(k))), \tag{5.13}$$

$$\mathcal{E}_\sigma(\mathbf{r}(k)) = \{(i,j) \in \mathcal{E}_n(\mathbf{r}(k)) \,|\, \sigma_{ij}(k) = 1\}, \tag{5.14}$$

where $\sigma_{ij}(k) = \sigma_{ji}(k) \in \{0,1\}$ is a binary symmetric function. Thus, if we design σ such that $\mathcal{G}_\sigma$ is connected, then the connectivity of $\mathcal{G}_n$ is maintained as long as the swarming algorithm preserves all the edges of $\mathcal{G}_\sigma$. Figure 5.2 summarizes $\mathcal{G}_s$, $\mathcal{G}_n$ and $\mathcal{G}_\sigma$ definitions.

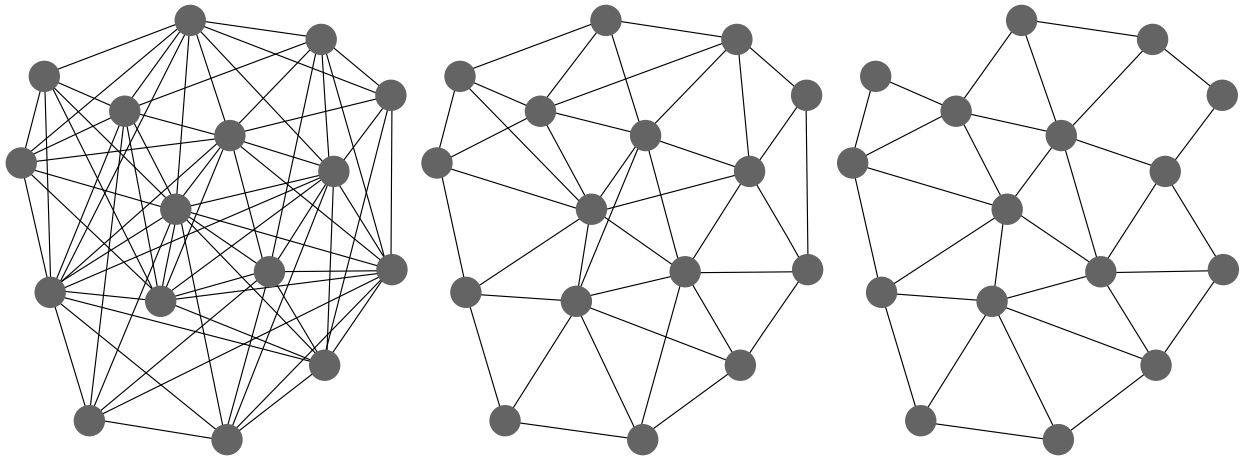


Figure 5.2 Graphs (a) $\mathcal{G}_s$, (b) $\mathcal{G}_n$ and (c) $\mathcal{G}_\sigma$

In their paper, Sakai et al. [5] proposed a set of two rules to determine the value of σ. While we have observed that those rules performed well for a small swarm, simulations with swarms bigger that 10 agents have shown that they are too restrictive (i.e., too many edges were preserved) and enforce too much rigidity in the formation. Moreover, the verification of those rules at each time step represents a relatively heavy computational cost. Therefore, we propose in the following a set of two other modified rules to deal with edge deactivation.

In this work, there are two cases where an edge $(i, j) \in \mathcal{E}_n$ is not included in $\mathcal{E}_\sigma$ (i.e., $\sigma_{ij}(k) = 0$). To describe the first case, let us consider a minimal example with only three agents as shown in Figure 5.3. In open spaces (i.e., no obstacle in the agents' neighborhood), the swarming algorithm moves the agent such that the distances between each agent are the same. Therefore, the agents form an equilateral triangle at rest. Then, when the agents encounter an obstacle, the formation will be distorted. In particular, when passing through a narrow gap, the formation will have to squeeze along the direction of the gap. As the triangle formation flattens out and the three agents tend to align, we can deactivate the longer edges without endangering the global connectivity of the swarm. Now, we describe this case in terms of a distributed deactivation rule. Let $\mathcal{A}_{ij}$ and $\mathcal{B}_{ij}$ be the sets of agents so that

$$\mathcal{A}_{ij} = \{m \in \mathcal{N}_i(\mathcal{G}_n) \backslash \{j\} \,|\, \|\mathbf{r}_{mi}\| < \|\mathbf{r}_{ji}\|\}, \quad \forall j \in \mathcal{N}_i(\mathcal{G}_n), \tag{5.15}$$

$$\mathcal{B}_{ij} = \{m \in \mathcal{N}_i(\mathcal{G}_n) \backslash \{j\} \,|\, \|\mathbf{r}_{mj}\| < \|\mathbf{r}_{ji}\|\}, \quad \forall j \in \mathcal{N}_i(\mathcal{G}_n). \tag{5.16}$$

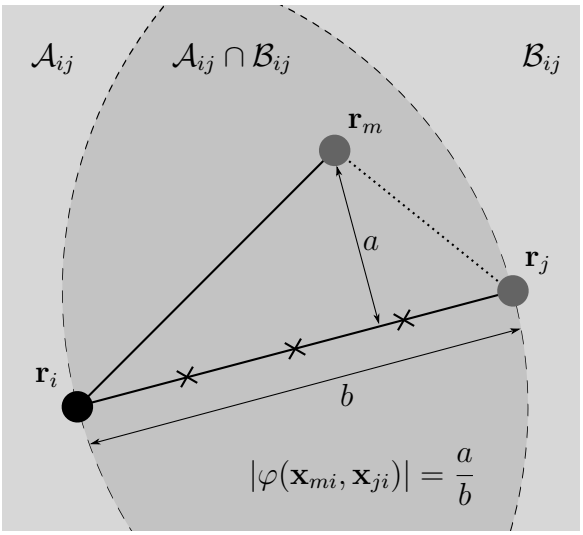


Figure 5.3 Deactivation case #1

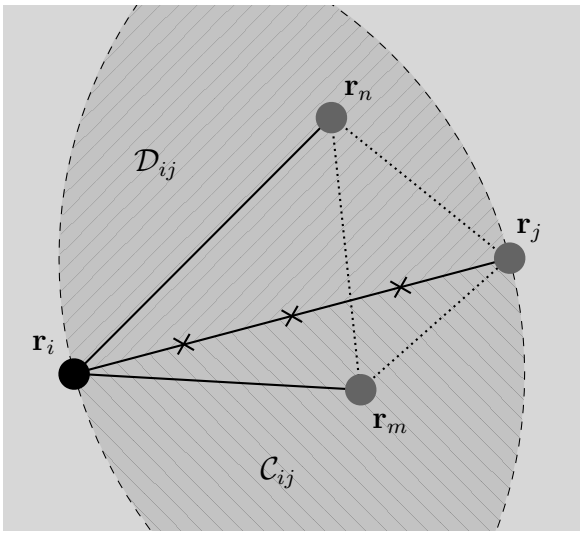


Figure 5.4 Deactivation case #2

We also define

$$\varphi(\mathbf{p}, \mathbf{q}) = \frac{\mathbf{p}^\top \mathbf{H} \mathbf{q}}{\mathbf{q}^\top \mathbf{q}}, \quad \mathbf{H} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}, \quad \forall \mathbf{p}, \mathbf{q} \in \mathbb{R}^2, \tag{5.17}$$

so that $|\varphi(\mathbf{r}_{mi}, \mathbf{r}_{ji})|$ equals the ratio of the distance between agent m and the edge (i, j) over the length of the edge (i, j). In other words, $|\varphi(\mathbf{r}_{mi}, \mathbf{r}_{ji})|$ quantifies the "flatness" of the triangle in m. And finally, the edge (i, j) is deactivated if

$$\exists m \in \mathcal{A}_{ij} \cap \mathcal{B}_{ij}, \quad |\varphi(\mathbf{r}_{mi}, \mathbf{r}_{ji})| < \varphi_d, \tag{5.18}$$

where $0 < \varphi_d < \frac{1}{\sqrt{2}}$ is a tuning parameter.

To describe the second case, we consider a minimal example with four agents as shown in Figure 5.4. Let us assume that the four agents are sufficiently close so that they are connected to all three other agents. In this case, there are two edges crossing in the middle of the formation. Simulations have shown that such configurations bring some unexpected rigidity to the formation as the position of each agent is too constrained. While none of the crossing edges are critical to global connectivity, we simply propose to deactivate the longest of the crossing edges. Now, we develop a distributed deactivation rule to deal with this case. Let $\mathcal{C}_{ij}$ and $\mathcal{D}_{ij}$ be the sets of agents so that

$$\mathcal{C}_{ij} = \{m \in \mathcal{A}_{ij} \cap \mathcal{B}_{ij} \backslash \{j\} \,|\, \varphi(\mathbf{r}_{mi}, \mathbf{r}_{ji}) < 0\}, \quad \forall j \in \mathcal{N}_i(\mathcal{G}_n), \tag{5.19}$$

$$\mathcal{D}_{ij} = \{m \in \mathcal{A}_{ij} \cap \mathcal{B}_{ij} \backslash \{j\} \,|\, \varphi(\mathbf{r}_{mi}, \mathbf{r}_{ji}) > 0\}, \quad \forall j \in \mathcal{N}_i(\mathcal{G}_n), \tag{5.20}$$

where the sign of $\varphi(\mathbf{r}_{mi}, \mathbf{r}_{ji})$ determines on which side of the edge (i, j) the agent m is located.

Then, the edge (i, j) is deactivated if

$$\exists (m, n) \in \mathcal{C}_{ij} \times \mathcal{D}_{ij}, \quad \|\mathbf{r}_{nm}\| < \|\mathbf{r}_{ji}\| . \tag{5.21}$$

Note that the deactivation rules developed above are distributed and symmetric such that if agent i determines that edge (i, j) must be deactivated, then agent j necessarily comes to the same conclusion and edge (i, j) is globally deactivated.

To sum up, we define σ_{ij} as follows :

$$\sigma_{ij} = \begin{cases} 0, & \text{if (5.18) or (5.21),} \\ 1, & \text{otherwise.} \end{cases} \tag{5.22}$$

5.3.2 Ranking System

In their paper, Sakai et al. [5] proposed a control scheme where followers do not need to identify the leader. This is a great advantage as they managed to operate a small swarm without the need for communication between agents. Unfortunately, such a control scheme has been observed to be inefficient in the case of a bigger swarm. Indeed, we can see the swarm like a damper that connects the leader to the farthest follower. Then, the leader applies a traction force on the rest of the swarm in order to move it along the defined path. But, as the swarm stretches and the distance between the leader and the farthest follower grows, the damping of the system increases such that the force is totally absorbed before reaching the last follower. This eventually results in the swarm being stuck as the leader is unable to drag the entire swarm.

In order to give some sense of the leader position to the followers, a new feature relying on minimal data transmission and simple computation has been introduced. We define the rank ρ_i of agent i as the distance to the leader N so that

$$\rho_i = \text{dist}(i, N), \tag{5.23}$$

where $\text{dist}(\cdot, \cdot)$ is the distance in the sense of graph theory. By definition, the leader rank will always be $\rho_N(k) = 0$. Figure 5.5 illustrates the system of ranks. Now, let us assume that each agent can retrieve the rank of its neighbors. Thus, each agent can estimate in what direction it should move to actually follow the leader and the problem is solved.

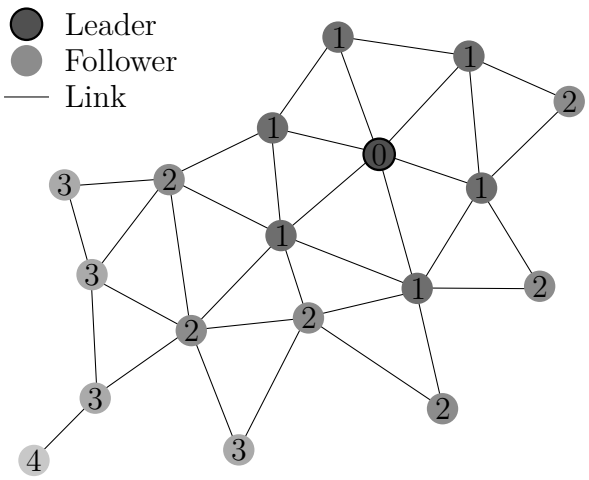


Figure 5.5 Rank system

For that purpose, we define the "virtual leader" position $\mathbf{l}_i$ as seen by agent i

$$\mathbf{l}_i = \frac{1}{|\mathcal{S}_i|} \sum_{j \in \mathcal{S}_i} \mathbf{r}_j, \tag{5.24}$$

where $\mathcal{S}_i$ is the set of all "surrogate leaders" as seen by agent i defined as

$$\mathcal{S}_i = \left\{ j \in \mathcal{N}_i(\mathcal{G}_\sigma) \,\middle|\, \rho_j = \max_{m \in \mathcal{N}_i(\mathcal{G}_\sigma)} \rho_m \right\}. \tag{5.25}$$

Also, each follower i can easily compute an estimate of its own rank $\bar{\rho}_i$ with

$$\bar{\rho}_i(k+1) = \min_{j \in \mathcal{N}_i(\mathcal{G}_\sigma)} (\bar{\rho}_j(k)) + 1, \quad \bar{\rho}_i(0) = N - 1. \tag{5.26}$$

Remark that this estimate may take some time before converging to the actual rank of the agent. For instance, in the ideal case where agents do not change rank, it can be inferred from (5.26) that

$$\forall k < \rho_i, \quad \bar{\rho}_i(k) > \rho_i, \tag{5.27}$$

$$\forall k \geq \rho_i, \quad \bar{\rho}_i(k) = \rho_i. \tag{5.28}$$

Even though the convergence time can become significant as it depends on the number of agents and the topology of the swarm, it cannot endanger the connectivity of the swarm. Indeed, (5.26) also implies that

$$\bar{\rho}_i(k) < \bar{\rho}_j(k) \;\Rightarrow\; \rho_i(k) < \rho_j(k), \quad \forall i, j \in \mathcal{V}. \tag{5.29}$$

In other words, the followers either sense the true direction of the leader, or does not sense it at all, but they cannot sense the leader in the opposite direction.

5.3.3 Follower Control

A common way to deal with autonomous navigation in an environment featuring obstacles is using the Artificial Potential Field (APF) method. To solve the path planning problem, for instance, the agent builds a potential field where regions to avoid are characterized by high potential values and the goal is to have the lowest potential value. Then a safe path to the goal can be inferred by sliding along the potential field valley toward the lowest potential. In their paper, Sakai et al. [5] used the APF method to generate the control input of followers. The final artificial potential field is constructed by superposition of several potential functions that enforce different behaviors, namely collision avoidance, obstacle avoidance, cohesion, and even LOS maintenance. Then, the direction of control input is derived from the APF and its magnitude is determined so as to ensure collision avoidance, connectivity maintenance and prevent actuator saturation. Note that the local minima issue that might arise when using APF for path planning does not occur in the case of formation control as each agent constructs a local APF at each time step.

In the following, we propose a modified version of the method described above. First, note that in the case of formation control, there is no need for constructing an actual APF as we only need to compute the derivative of the field at one point, not the entire path. Therefore, we choose to directly construct the control input without building an APF. Indeed, we would have to numerically compute the gradient of the APF at each time step otherwise. Second, extensive simulations have shown that the LOS maintenance behavior often leads to deadlock in a realistic environment. Therefore, we simply decide to ignore it. Third, apart from actuator saturation prevention, all other behaviors are already enforced by the APF definition. Then, the part where the control input magnitude is determined is redundant in our opinion and will not be considered here. As a result, the solution presented in the following does not require as many computational resources as the one in [5].

We define the control input $\mathbf{u}_i$ for the follower $i \in \mathcal{V}\backslash\{N\}$ as follows:

$$\mathbf{u}_i = \begin{cases} \dfrac{\mathbf{w}_i}{\|\mathbf{w}_i\|} u_{sat}, & \text{if } \|\mathbf{w}_i\| > u_{sat}, \\ \mathbf{w}_i, & \text{otherwise.} \end{cases} \tag{5.30}$$

By definition, $\|\mathbf{u}_i\|$ is bounded by $u_{sat} < u_{max}$ to prevent actuator saturation. Then, we describe the unbounded control input $\mathbf{w}_i$ as the sum of components enforcing each desired behavior so that

$$\mathbf{w}_i = \mathbf{w}_i^{obs} + \mathbf{w}_i^{col} + \mathbf{w}_i^{con} + \mathbf{w}_i^{com} + \mathbf{w}_i^{led}. \tag{5.31}$$

The first component $\mathbf{w}_i^{obs}$ is introduced for obstacle avoidance. In other words, $\mathbf{w}_i^{obs}$ is constructed so that it moves agent i away from the closest obstacle detected. Let $\mathcal{O}_i$ be the set of points on obstacles detected by agent i (i.e., all points that verify conditions (5.9) and (5.10)). Then the closest obstacle point $\mathbf{o}_i$ is defined as

$$\mathbf{o}_i = \underset{\mathbf{r}_o \in \mathcal{O}_i}{\operatorname{argmin}} \|\mathbf{r}_o - \mathbf{r}_i\| . \tag{5.32}$$

And $\mathbf{w}_i^{obs}$ can be computed as follows :

$$\mathbf{w}_i^{obs} = \begin{cases} 0, & \text{if } \mathcal{O}_i = \varnothing, \\ \psi_{obs}(\|\mathbf{o}_i - \mathbf{r}_i\|) \dfrac{\mathbf{o}_i - \mathbf{r}_i}{\|\mathbf{o}_i - \mathbf{r}_i\|}, & \text{otherwise}, \end{cases} \tag{5.33}$$

with

$$\psi_{obs}(z) = \begin{cases} -K_{obs} \left(\dfrac{1 + \varepsilon_{obs}}{\dfrac{z - d_o}{l_o - d_o} + \varepsilon_{obs}} - 1 \right) \varepsilon_{obs}, & \text{if } d_o \leq z \leq l_o, \\ 0, & \text{otherwise}, \end{cases} \tag{5.34}$$

where $0 < K_{obs}$, $0 < \varepsilon_{obs} \ll 1$ and $d_o < l_o$ are three tuning parameters. As shown on Figure 5.6, K_{obs} represents the maximum strength of the repulsive force generated by the obstacle on the agent (i.e., $\psi_{obs}(d_o) = -K_{obs}$). l_o defines the maximum range of the force around the obstacle (i.e $\psi_{obs}(l_o) = 0$). ε_{obs} allows us to adjust how the force varies with respect to the distance from the obstacle.

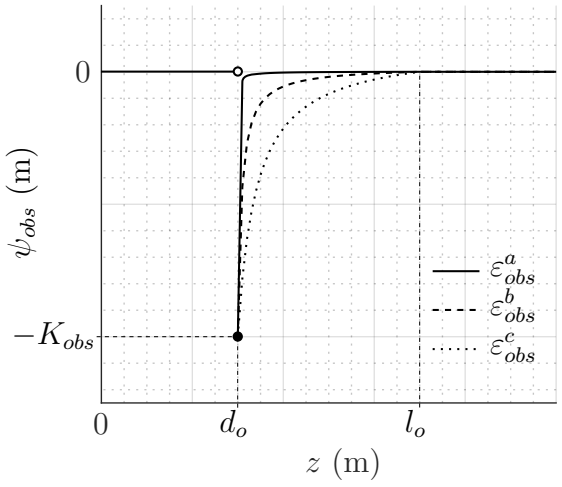


Figure 5.6 $\psi_{obs}(z)$ for $\varepsilon_{obs}^a < \varepsilon_{obs}^b < \varepsilon_{obs}^c$

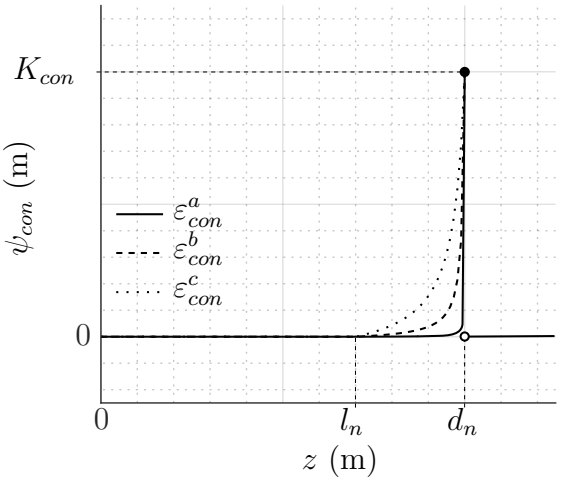


Figure 5.7 $\psi_{con}(z)$ for $\varepsilon_{obs}^a < \varepsilon_{obs}^b < \varepsilon_{obs}^c$

In the same fashion as for obstacle avoidance, the second component $\mathbf{w}_i^{col}$ enables collision avoidance behavior. In this case, $\mathbf{w}_i^{col}$ is built so that agent i is moved away from its neighbors. Therefore, we define $\mathbf{w}_i^{col}$ as follows

$$\mathbf{w}_i^{col} = \sum_{j \in \mathcal{N}_i(\mathcal{G}_\sigma)} \psi_{col}(\|\mathbf{r}_j - \mathbf{r}_i\|) \frac{\mathbf{r}_j - \mathbf{r}_i}{\|\mathbf{r}_j - \mathbf{r}_i\|}, \tag{5.35}$$

with

$$\psi_{col}(z) = \begin{cases} -K_{col} \left(\dfrac{1 + \varepsilon_{col}}{\dfrac{z - d_c}{l_c - d_c} + \varepsilon_{col}} - 1 \right) \varepsilon_{col}, & \text{if } d_c \leq z \leq l_c, \\ 0, & \text{otherwise,} \end{cases} \tag{5.36}$$

where $0 < K_{col}$, $0 < \varepsilon_{col} \ll 1$ and $d_c < l_c$ are three tuning parameters. Similarly to what can be seen on Figure 5.6, K_{col} represents the maximum strength of the repulsive force generated by one neighbor on the agent (i.e., $\psi_{col}(d_c) = -K_{col}$). l_c defines the maximum range of the force around a neighbor (i.e $\psi_{col}(l_c) = 0$). ε_{col} allows us to adjust how the force varies with respect to the distance from neighbors.

The third component $\mathbf{w}_i^{con}$ is introduced to ensure connectivity maintenance. $\mathbf{w}_i^{con}$ has the opposite effect than $\mathbf{w}_i^{col}$, moving the agent toward its neighbors when the distance between them becomes too big and the edge between them is endangered. Then, we describe $\mathbf{w}_i^{con}$ as follows:

$$\mathbf{w}_i^{con} = \sum_{j \in \mathcal{N}_i(\mathcal{G}_\sigma)} \psi_{con}(\|\mathbf{r}_j - \mathbf{r}_i\|) \frac{\mathbf{r}_j - \mathbf{r}_i}{\|\mathbf{r}_j - \mathbf{r}_i\|}, \tag{5.37}$$

with

$$\psi_{con}(z) = \begin{cases} K_{con} \left(\dfrac{1 + \varepsilon_{con}}{\dfrac{z - d_n}{l_n - d_n} + \varepsilon_{con}} - 1 \right) \varepsilon_{con}, & \text{if } l_n \leq z \leq d_n, \\ 0, & \text{otherwise,} \end{cases} \tag{5.38}$$

where $0 < K_{con}$, $0 < \varepsilon_{con} \ll 1$ and $l_n < d_n$ are three tuning parameters. As shown on Figure 5.7, K_{con} represents the maximum strength of the attractive force generated by one neighbor on the agent (i.e., $\psi_{con}(d_n) = -K_{con}$). l_n defines the minimum distance between agents for the connectivity maintenance behavior to enter into effect (i.e $\psi_{con}(l_n) = 0$). ε_{con} allows us to adjust how the force varies with respect to the distance from neighbors.

One would like to be able to adjust the compactness of the formation (i.e., the average distance between agents). For that purpose, we introduced a fourth component $\mathbf{w}_i^{com}$ defined as follows:

$$\mathbf{w}_i^{com} = \sum_{j \in \mathcal{N}_i(\mathcal{G}_\sigma)} \psi_{com}(\|\mathbf{r}_j - \mathbf{r}_i\|) \frac{\mathbf{r}_j - \mathbf{r}_i}{\|\mathbf{r}_j - \mathbf{r}_i\|}, \tag{5.39}$$

with

$$\psi_{com}(z) = \begin{cases} K_{com} \left(\dfrac{z - d_c}{\alpha_{com}(d_n - d_c)} - 1 \right), & \text{if } d_c \leq z \leq d_n, \\ 0, & \text{otherwise,} \end{cases} \tag{5.40}$$

where $0 < K_{com}$ and $0 < \alpha_{com} < 1$ are two tuning parameters. As shown on Figure 5.8, K_{com} sets the strength of the force generated in order to achieve the desired level of compactness. α_{com} is a normalized parameter that allows us to adjust the compactness of the swarm. For instance, the swarm will spread as α_{com} increases.

Additionally, a fifth component $\mathbf{w}_i^{led}$ ensures that agent i in the swarm is driven to the leader by the means of its surrogate leader. We describe $\mathbf{w}_i^{led}$ as follows

$$\mathbf{w}_i^{led} = \begin{cases} 0, & \text{if } \mathcal{N}_i(\mathcal{G}_\sigma) = \varnothing, \\ \psi_{led}(\|\mathbf{l}_i - \mathbf{r}_i\|) \dfrac{\mathbf{l}_i - \mathbf{r}_i}{\|\mathbf{l}_i - \mathbf{r}_i\|}, & \text{otherwise,} \end{cases} \tag{5.41}$$

with

$$\psi_{led}(z) = K_{led} z, \tag{5.42}$$

where $0 < K_{led}$ sets the strength of the traction force of the leader as can be seen on Figure 5.9. Note that the control law described above does not require absolute positioning, only relative positions with respect to the agent of interest are needed to compute the control input.

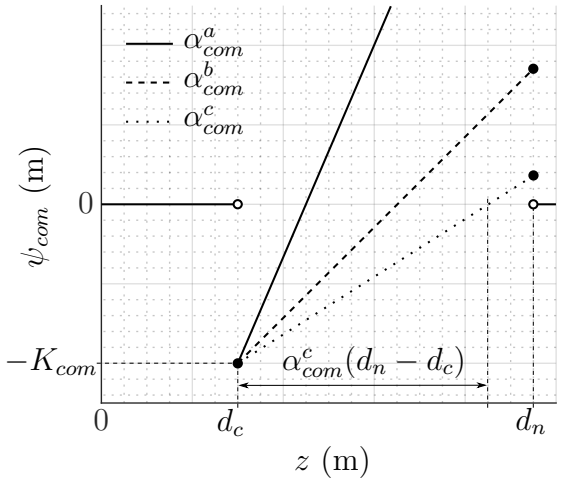


Figure 5.8 $\psi_{com}(z)$ for $\alpha^a_{com} < \alpha^b_{com} < \alpha^c_{com}$

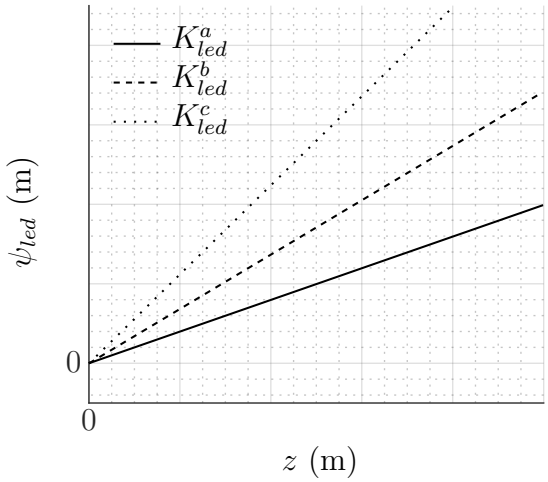


Figure 5.9 $\psi_{led}(z)$ for $K^a_{led} < K^b_{led} < K^c_{led}$

5.3.4 Leader Control

The goal of the leader ($i = N$) is to follow the target path without outrunning the rest of the swarm. Let the path be a sequence of way-points $\mathcal{P} = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n\}$ in the environment, then the direction of the control input of the leader $\mathbf{w}_N$ is defined such that it always points towards the next way-point. A simple way to compute $\mathbf{w}_N$ is presented in Algorithm 1. The path generation will not be discussed in this work. Nevertheless, note that for the algorithm to work properly, the path has to be sufficiently smooth and the way-points have to be sufficiently spaced.

Algorithm 1: Leader control input direction update

Input: The absolute position $\mathbf{r}_N$ and m the index of the last target way-point

Output: The control input direction $\mathbf{w}_N$ and m the index of the new target way-point

Data: The path $\mathcal{P} = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n\}$

while $0 \leq (\mathbf{p}_{m+1} - \mathbf{p}_m)^\top (\mathbf{r}_N - \mathbf{p}_m)$ **and** $m < n$ **do**

  $m \leftarrow m + 1$

end

$$\mathbf{w}_N \leftarrow \frac{\mathbf{p}_m - \mathbf{r}_N}{\|\mathbf{p}_m - \mathbf{r}_N\|}$$

return $\mathbf{w}_N$,m

Once the direction is found, the magnitude of the control input is determined so that the leader does not lose connectivity with the rest of the swarm. Thus, we introduce u_{lsat} and u_{lcon}, two upper bounds for $\mathbf{u}_N$ such that

$$u_{lsat} = \alpha_{lsat} u_{sat}, \qquad \text{and} \qquad u_{lcon} = \max\left\{0, \alpha_{lcon} d_n - \min_{j \in \mathcal{N}_i(\mathcal{G}_\sigma)} \|\mathbf{r}_j - \mathbf{r}_N\|\right\}, \tag{5.43}$$

where $0 < \alpha_{lsat} < 1$ and $0 < \alpha_{lcon} < 1$ are two tuning parameters. On one hand, u_{lsat} prevents actuator saturation and improves the cohesion of the swarm while the maximum speed of the leader is lower than the maximum speed of the followers. On the second hand, u_{lcon} ensures connectivity maintenance by limiting the maximum distance between the leader and the closest follower. Finally, $\mathbf{u}_N$ is defined as follows:

$$\mathbf{u}_N = \mathbf{w}_N \min\{u_{lsat}, u_{lcon}\}. \tag{5.44}$$

Figure 5.10 illustrates how the leader control input is generated from the path and the position of followers.

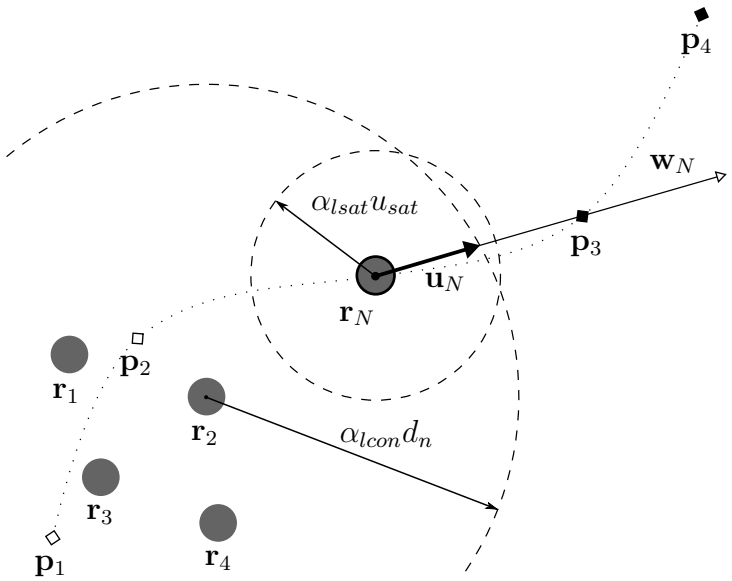


Figure 5.10 Leader control input

5.4 Simulation

In order to demonstrate the effectiveness of the proposed solution, we run several simulations in MATLAB 2018a. Simulation parameter values are inferred from dimensions and estimated performance of a nano-quadrotor. The parameter definitions and values are summarized in Table 5.1.

Table 5.1 Simulation parameters

Definition	**Symbol**	**Value**
Sampling period	T_s	10ms
Number of agents	N	30
Maximum input	u_{max}	0.025m
Collision-free distance	d_c	0.3m
Obstacle-free distance	d_o	0.1m
Sensing range	d_s	2m
LOS-clearance distance	d_l	0.05m
Neighbor maximum distance	d_n	1m
Edge deactivation threshold	ϕ_d	0.2
Input saturation level	u_{sat}	0.02m
Obstacle avoidance gain	K_{obs}	200m
Obstacle avoidance slope	ε_{obs}	0.0001
Obstacle avoidance range	l_o	0.7m
Collision avoidance gain	K_{col}	200m
Collision avoidance slope	ε_{col}	0.0001
Collision avoidance range	l_c	0.7m
Connectivity maintenance gain	K_{con}	200m
Connectivity maintenance slope	ε_{con}	0.0001
Connectivity maintenance range	l_n	0.7m
Compactness gain	K_{com}	0.1m
Compactness level	α_{com}	0.8
Leader traction gain	K_{led}	0.06m
Leader input saturation level	α_{lsat}	0.7
Leader connectivity level	α_{lcon}	0.95

As displayed in Figure 5.11, the first test environment consists of a long straight corridor with the leader path going from one extremity to another. This minimal test case allows us to easily examine the effectiveness of the edge deactivation strategy when it comes to going through narrow spaces. Consequently, we set the corridor length to $l = 10$ m and performed simulations for different corridor widths $w = \{0.5, 0.6, 0.7, 0.8, 0.9\}$ m. For each

value of w, we ran 30 simulations with random eligible initial agent positions (i.e., the swarm is connected in the initial state). Note that this also means that the leader is not necessarily at the front of the swarm in the initial state.

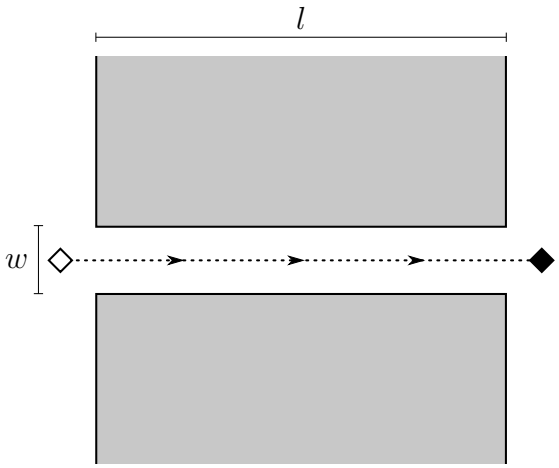


Figure 5.11 Straight corridor environment

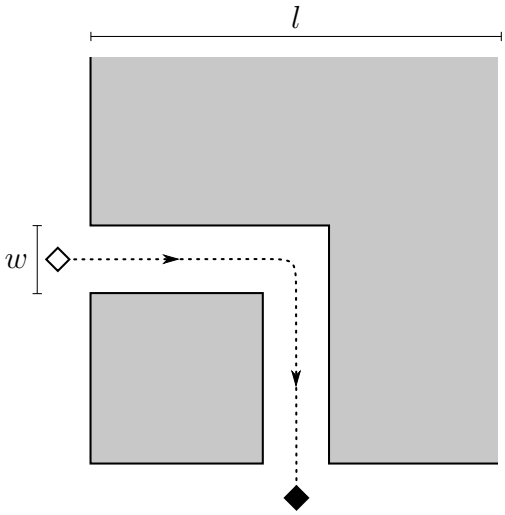


Figure 5.12 L-shaped corridor environment

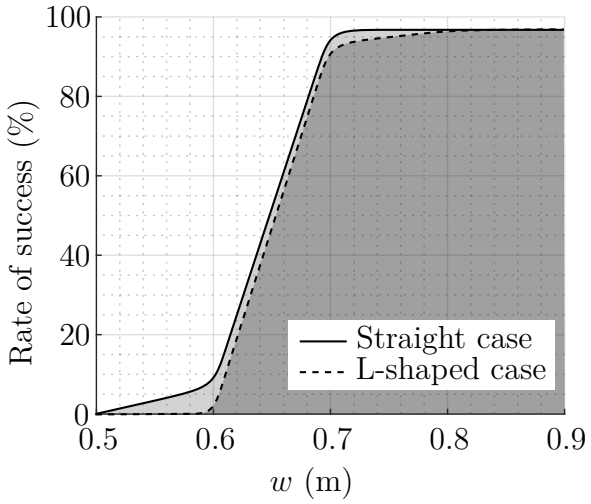


Figure 5.13 Rate of mission success for different corridor widths

Figure 5.13 shows the rate of success for each value of w. We observed that for widths larger than 0.7m, the proposed solution was effective. Indeed, all the agents managed to reach the other side of the corridor while maintaining connectivity as the maximum rank ρ_i among agents remains below N. On the contrary, for widths smaller than 0.7 m, LOS may be interrupted by the wall as the swarm enters the corridor, leading to the loss of global connectivity. Even though this may represent a strong limitation, we assume that corridors

smaller than 0.7 m are rarely seen in a real indoor environment. Furthermore, the conditions for collision and obstacle avoidance were satisfied in all simulations. Indeed, minimum inter-agent distance and agent-obstacle distance were always greater than their allowable minimum value, d_c and d_o respectively. When comparing with the results presented in [5], we can see that they obtain better performances as their solution can handle corridor widths as small as 0.25 m with success. Nevertheless, their test environment is built using only circle obstacles of radius 0.5 m which is not realistic and makes it easier for the swarm to enter the corridor. As a result, when we test their solution in our environment, the swarm is unable to enter the corridor without losing connectivity. Also, remark that failures manifest themselves in different ways for the two solutions. On one hand, in [5], a failure consists in a deadlock of the swarm along the path when the LOS-maintenance rule prevents the swarm from moving. On the other hand, our solution does not present any LOS-maintenance rules and a failure consists in global connectivity loss, with one part of the swarm following the leader and the other remaining roughly motionless.

We have seen in the first test that LOS interruption when going near a corner wall may result in connectivity failure. Therefore, the second test investigated the behavior of the swarm when the path goes through an L-shaped corridor as shown on Figure 5.12. Once again, we set the total corridor length to $l = 10$ m and performed 30 simulations for different corridor widths $w = \{0.5, 0.6, 0.7, 0.8, 0.9\}$ m. In this case, we can see on Figure 5.13 that the proposed solution was effective for corridors larger than 0.7 m. Therefore, this test shows that once the swarm has managed to enter the corridor, the proposed solution is able to move the swarm to the end of the corridor with success in most cases, no matter the shape of the corridor. In contrast, the results in [5] show that corners along the path significantly degrade the rate of success. Still, they report slightly better performance as their solution can handle corridor widths ranging to 0.5 m. But once again, when we test their solution in our environment featuring sharp obstacles, the swarm is unable to enter the corridor without losing connectivity.

Now, recall that the main limitation observed in the work of Sakai et al. [5] was the size of the swarm. As a matter of fact, they only present results for $N = 10$ and we observed in simulations that their solution was not able to move a bigger swarm. In this third test, we demonstrate the scalability of our solution in a realistic scenario and the effectiveness of the ranking system for leader following in big swarms. To do so, we reproduce the map of an existing building and design a path that goes around the building. Then, we perform 10 simulations with randomly selected initial states for a swarm of 100 agents, i.e., $N = 100$. Figures in Annex A show snapshots of the swarm for different simulation times. For all initial states, the swarm managed to successfully follow the path. The condition on connectivity

maintenance is satisfied as the maximum rank ρ_i among agents remains below N as seen on Figure 5.16. Furthermore, Figure 5.14 and Figure 5.15 respectively show that the conditions on collision avoidance and obstacle avoidance are also satisfied. Note that simulations with swarms bigger than 100 agents have not been performed as they involve prohibitive simulation time. Nevertheless, the proposed solution can virtually handle any number of agents because it is fully distributed. In short, we can conclude that our solution is able to safely operate any number of agents in a realistic environment. Moreover, the ranking system enables good cohesion among the swarm all along the path as the followers will tend to catch up with their virtual leader. This feature can be considered an improvement with respect to the solution in [5] where the swarm can only start regrouping once the last agent clears the obstacle area. Also, remark that the distributed swarming algorithm proposed here leads to the emergence of a formation fitting a triangular tiling in which agents have an average of 6 neighbors. However, on Figure 5.17 we can see that the average number of agents tends to 5 rather than 6. That can be explained by the case of agents at the periphery of the swarm that cannot have neighbors in every direction.

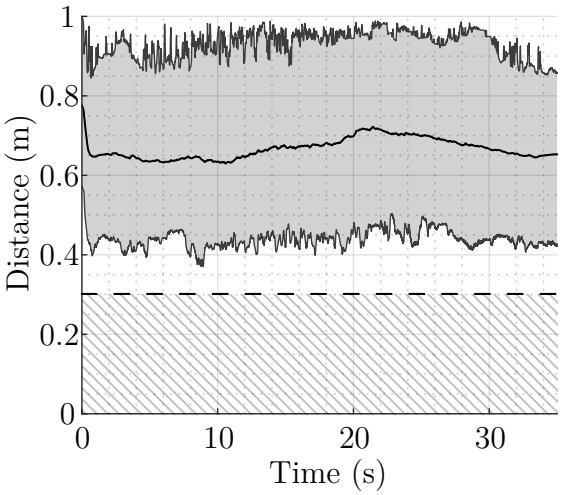


Figure 5.14 Average neighbors distance

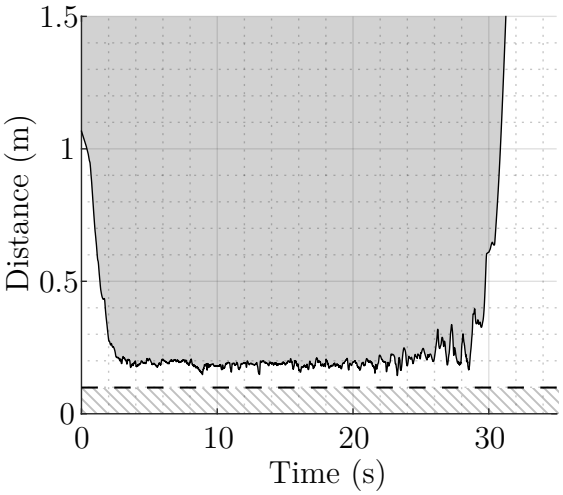


Figure 5.15 Distance from closest obstacle

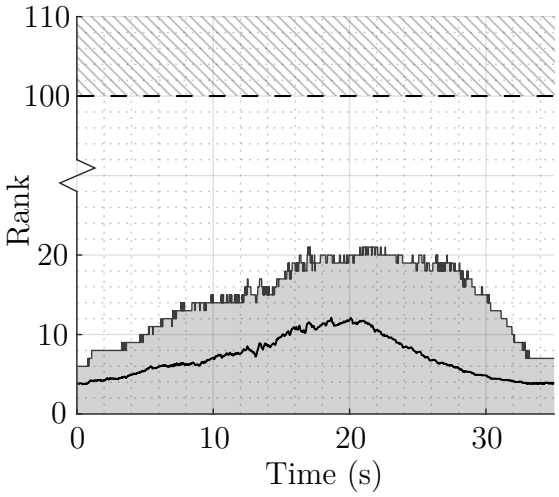


Figure 5.16 Average rank of agents

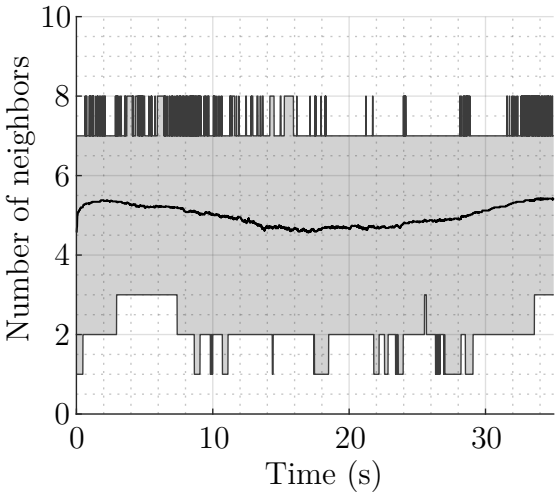


Figure 5.17 Average number of neighbors

CHAPTER 6 CONCLUSION

6.1 Summary of Works

This work aimed to explore the field of quadrotor swarm control from an SAR perspective. As an introduction to the subject, a wide literature review on the subject was produced in Chapter 2. Then, a mathematical model of the quadrotor dynamics based on the literature was presented in Chapter 3. In the same chapter, several digital control laws were designed in order to provide control modes useful for swarm control. Those control laws had classical PID-like structures. The remainder of this work consisted of two original and complimentary swarming solutions based on the leader-follower paradigm.

Firstly, a swarming solution for outdoor navigation (i.e., no obstacle avoidance and unlimited communication capabilities) was developed in Chapter 4. It accounted for specific design constraints observed in the material under development at Humanitas Solutions. Unlike most of the work in the literature, the solution proposed here handles not only the maintenance of the formation but also the initialization of the formation. In particular, a formation pattern is defined, then places are optimally assigned and, finally, the agents move to their assigned place while avoiding others. This solution has been integrated into Humanitas' quadrotor onboard software and tested with SITL simulations.

Secondly, a swarming solution for indoor navigation (i.e., obstacle avoidance and limited communication capabilities) was investigated. Based on previous work by Sakai et al. [5], a new distributed algorithm for indoor navigation was developed as detailed in Section 5.3. In particular, the algorithm enforces a certain set of expected individual simple behaviors such that a large swarm can follow a leader through cluttered environments relying only on local information. Simulation results discussed in Section 5.4 show that it is possible to operate a swarm of a hundred quadrotors with the algorithm proposed. The swarm is able to follow the leader, maintain connectivity, avoid collisions with other agents, avoid obstacles, and even squeeze to pass through narrow spaces. Overall, the proposed solution outperforms the solution in [5] in simulations.

6.2 Limitations & Future Research

The designed solution however has some limitations. The outdoor navigation solution proposed here enforces an undesirable centralization on the leader as each agent tracks the leader's position. Even though the distributed nature of the communication network allows

the followers to track the leader without a direct connection, this represents a strong limitation for the solution. Indeed, in cases of leader failure, the mission will simply be aborted. Thus, future improvements would have to investigate a way to replace the leader in case of failure. This could also let the swarm select automatically its leader rather than letting the user decide it which reduces the formation time. Also, the use of a virtual leader could be a path worth exploring.

On a software level, the low-level control of the quadrotor could be completely relegated to the autopilot. This may improve control performance and relax the mission manager workload. Moreover, the mission manager node should be split into several nodes to match ROS philosophy and provide more modularity and reusability.

Due to material limitations, we were not able to simulate more than seven quadrotors. Also, no metrics on fault tolerance are given in this work. Finally, the simulation model used does not account for the actual quadrotor specifications and does not model measurement errors (e.g., noises, bias, and scale factor error). Therefore, extensive simulation and experimental tests would have to be carried out to properly assess the solution performance and robustness. Those would have to be run under both normal and faulty conditions with a large number of agents.

As for the indoor environment, the proposed solution needs further testing that accounts for quadrotor dynamics. As a matter of fact, it remains a simple thought experiment in the current state. Therefore, future works should include experimental validation of the results of the simulation operating a swarm of a dozen nano-quadrotors. The Crazyflies 2.1 distributed by Bitcraze appear to be an affordable platform for such a test. As a first step, an external motion capture system such as Vicon could be used to provide each agent the position of the nearby obstacles and agents. In that regard, the Crazyswarm project [74] provides a solution to fly a swarm of Crazyflies with Vicon. Then, extensive work on inter-agent communication, obstacle detection, and neighbors relative localization, such as presented in [75], would have to be pursued in order to operate the swarm without the help of a motion capture system. Note that the solution developed here required high-rate communication between agents contrary to what is proposed in [5]. Furthermore, as seen in Section 5.4, the solution proposed fails to maintain connectivity when entering a narrow corridor. Therefore, working on preventing LOS interruption in such cases would greatly improve the resilience of the swarm.

REFERENCES

[1] I. Shaw, *Predator Empire: Drone Warfare and Full Spectrum Dominance*. University of Minnesota Press, 2016. [Online]. Available: https://books.google.ca/books?id=x0DZjgEACAAJ

[2] F. Kendoul, "Survey of advances in guidance, navigation, and control of unmanned rotorcraft systems," *Journal of Field Robotics*, vol. 29, no. 2, pp. 315–378, 4 2012. [Online]. Available: http:https://doi.org/10.1002/rob.20414

[3] X. H. Ge, F. W. Yang, and Q. L. Han, "Distributed networked control systems: A brief overview," *Information Sciences*, vol. 380, pp. 117–131, 2017. [Online]. Available: <GotoISI>://WOS:000390501500008

[4] C. T. Recchiuto and A. Sgorbissa, "Post-disaster assessment with unmanned aerial vehicles: A survey on practical implementations and research approaches," *Journal of Field Robotics*, vol. 35, no. 4, pp. 459–490, 2018. [Online]. Available: <GotoISI>://WOS:000434130700003

[5] D. Sakai, H. Fukushima, and F. Matsuno, "Leader-follower navigation in obstacle environments while preserving connectivity without data transmission," *Ieee Transactions on Control Systems Technology*, vol. 26, no. 4, pp. 1233–1248, 2018. [Online]. Available: <GotoISI>://WOS:000435195200007

[6] J. T. Qi *et al.*, "Search and rescue rotary-wing uav and its application to the lushan ms 7.0 earthquake," *Journal of Field Robotics*, vol. 33, no. 3, pp. 290–321, 2016. [Online]. Available: <GotoISI>://WOS:000374846000003

[7] V. San Juan, M. Santos, and J. M. Andujar, "Intelligent uav map generation and discrete path planning for search and rescue operations," *Complexity*, p. 17, 2018. [Online]. Available: <GotoISI>://WOS:000431548600001

[8] G. S. C. Avellar *et al.*, "Multi-uav routing for area coverage and remote sensing with minimum time," *Sensors*, vol. 15, no. 11, pp. 27 783–27 803, 2015. [Online]. Available: <GotoISI>://WOS:000365686400017

[9] P. Dames and V. Kumar, "Autonomous localization of an unknown number of targets without data association using teams of mobile sensors," *Ieee Transactions*

on Automation Science and Engineering, vol. 12, no. 3, pp. 850–864, 2015. [Online]. Available: <GotoISI>://WOS:000358585200008

[10] X. Ji *et al.*, "Cooperative search by multiple unmanned aerial vehicles in a nonconvex environment," *Mathematical Problems in Engineering*, vol. 2015, 2015.

[11] P. Li and H. B. Duan, "A potential game approach to multiple uav cooperative search and surveillance," *Aerospace Science and Technology*, vol. 68, pp. 403–415, 2017. [Online]. Available: <GotoISI>://WOS:000407185700037

[12] S. Hayat *et al.*, "Multi-objective uav path planning for search and rescue," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, Conference Proceedings, pp. 5569–5574.

[13] X. W. Dong *et al.*, "Time-varying formation control for unmanned aerial vehicles: Theories and applications," *Ieee Transactions on Control Systems Technology*, vol. 23, no. 1, pp. 340–348, 2015. [Online]. Available: <GotoISI>://WOS:000346794600030

[14] X. H. Ge, Q. L. Han, and X. M. Zhang, "Achieving cluster formation of multi-agent systems under aperiodic sampling and communication delays," *Ieee Transactions on Industrial Electronics*, vol. 65, no. 4, pp. 3417–3426, 2018. [Online]. Available: <GotoISI>://WOS:000422806300056

[15] W. L. He *et al.*, "Leader-following consensus of nonlinear multiagent systems with stochastic sampling," *Ieee Transactions on Cybernetics*, vol. 47, no. 2, pp. 327–338, 2017. [Online]. Available: <GotoISI>://WOS:000395476200006

[16] H. Q. Li *et al.*, "High-performance consensus control in networked systems with limited bandwidth communication and time-varying directed topologies," *Ieee Transactions on Neural Networks and Learning Systems*, vol. 28, no. 5, pp. 1043–1054, 2017. [Online]. Available: <GotoISI>://WOS:000401981800003

[17] Z. C. Hou, I. Fantoni, and Ieee, "Distributed leader-follower formation control for multiple quadrotors with weighted topology," *2015 10th System of Systems Engineering Conference (SoSE)*, pp. 256–261, 2015. [Online]. Available: <GotoISI>://WOS:000380516100023

[18] C. Altafini, "Consensus problems on networks with antagonistic interactions," *IEEE Transactions on Automatic Control*, vol. 58, no. 4, pp. 935–946, 2013.

[19] H. Y. Sun *et al.*, "Energy-optimized consensus formation control for the time-delayed bilateral teleoperation system of uavs," *International Journal of Aerospace Engineering*, p. 22, 2018. [Online]. Available: <GotoISI>://WOS:000434182900001

[20] R. W. Beard, J. Lawton, and F. Y. Hadaegh, "A coordination architecture for spacecraft formation control," *IEEE Transactions on control systems technology*, vol. 9, no. 6, pp. 777–790, 2001.

[21] A. Clark *et al.*, "Minimizing convergence error in multi-agent systems via leader selection: A supermodular optimization approach," *Ieee Transactions on Automatic Control*, vol. 59, no. 6, pp. 1480–1494, 2014. [Online]. Available: <GotoISI>://WOS:000337134000007

[22] L. Sabattini *et al.*, "Distributed control of multirobot systems with global connectivity maintenance," *Ieee Transactions on Robotics*, vol. 29, no. 5, pp. 1326–1332, 2013.

[23] C. W. Reynolds, "Flocks, Herds and Schools: A Distributed Behavioral Model," in *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '87. New York, NY, USA: ACM, 1987, pp. 25–34. [Online]. Available: http://doi.acm.org/10.1145/37401.37406

[24] X. B. Xiang *et al.*, "On decentralized adaptive full-order sliding mode control of multiple uavs," *Isa Transactions*, vol. 71, pp. 196–205, 2017. [Online]. Available: <GotoISI>://WOS:000416189100002

[25] W. Jasim and D. B. Gu, "Robust team formation control for quadrotors," *Ieee Transactions on Control Systems Technology*, vol. 26, no. 4, pp. 1516–1523, 2018. [Online]. Available: <GotoISI>://WOS:000435195200033

[26] S. Bandyopadhyay, S. J. Chung, and F. Y. Hadaegh, "Probabilistic and distributed control of a large-scale swarm of autonomous agents," *Ieee Transactions on Robotics*, vol. 33, no. 5, pp. 1103–1123, 2017. [Online]. Available: <GotoISI>://WOS:000412235700007

[27] X. H. Wang, V. Yadav, and S. N. Balakrishnan, "Cooperative uav formation flying with obstacle/collision avoidance," *Ieee Transactions on Control Systems Technology*, vol. 15, no. 4, pp. 672–679, 2007. [Online]. Available: <GotoISI>://WOS:000247645500007

[28] M. d. Berg *et al.*, *Computational geometry: algorithms and applications*. Springer-Verlag TELOS, 2008.

[29] Y. C. Zhao *et al.*, "Uav formation control with obstacle avoidance using improved artificial potential fields," in *36th Chinese Control Conference (CCC)*, ser. Chinese Control Conference. NEW YORK: Ieee, 2017, Conference Proceedings, pp. 6219–6224. [Online]. Available: <GotoISI>://WOS:000432015500072

[30] P. D. Nguyen, C. T. Recchiuto, and A. Sgorbissa, "Real-time path generation for multicopters in environments with obstacles," in *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*. IEEE, 2016, pp. 1582–1588.

[31] X. H. Ge *et al.*, "A survey on recent advances in distributed sampled-data cooperative control of multi-agent systems," *Neurocomputing*, vol. 275, pp. 1684–1701, 2018. [Online]. Available: <GotoISI>://WOS:000418370200158

[32] J. H. Qin *et al.*, "Recent advances in consensus of multi-agent systems: A brief survey," *Ieee Transactions on Industrial Electronics*, vol. 64, no. 6, pp. 4972–4983, 2017. [Online]. Available: <GotoISI>://WOS:000401328500066

[33] Z.-W. Liu *et al.*, "Pulse-modulated intermittent control in consensus of multiagent systems," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 47, no. 5, pp. 783–793, 2017.

[34] J. Barraquand, B. Langlois, and J.-C. Latombe, "Numerical potential field techniques for robot path planning," *IEEE transactions on systems, man, and cybernetics*, vol. 22, no. 2, pp. 224–241, 1992.

[35] D. Mellinger, A. Kushleyev, and V. Kumar, "Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 477–483.

[36] T. Zhang *et al.*, "Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search," in *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2016, pp. 528–535.

[37] D. Conover, "Trajectory generation for a quadrotor unmanned aerial vehicle," Ph.D. dissertation, École Polytechnique de Montréal, 2018.

[38] M. Faessler *et al.*, "Autonomous, vision-based flight and live dense 3d mapping with a quadrotor micro aerial vehicle," *Journal of Field Robotics*, vol. 33, no. 4, pp. 431–450, 2016. [Online]. Available: <GotoISI>://WOS:000380102800003

[39] Z. Q. Miao *et al.*, "Formation control of quadrotor uavs without linear velocity measurements," *2017 18th International Conference on Advanced Robotics (Icar)*, pp. 179–184, 2017. [Online]. Available: <GotoISI>://WOS:000426976400028

[40] W. R. Hamilton, "Xi. on quaternions; or on a new system of imaginaries in algebra," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 33, no. 219, pp. 58–60, 1848.

[41] R. Mahony, V. Kumar, and P. Corke, "Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor," *IEEE Robotics Automation Magazine*, vol. 19, no. 3, pp. 20–32, Sept 2012.

[42] Z. Jia *et al.*, "Integral backstepping sliding mode control for quadrotor helicopter under external uncertain disturbances," *Aerospace Science and Technology*, vol. 68, pp. 299–307, 2017.

[43] A. Ataei, I. C. Paschalidis, and Ieee, "Quadrotor deployment for emergency response in smart cities: A robust mpc approach," in *54th IEEE Conference on Decision and Control (CDC)*, ser. IEEE Conference on Decision and Control. NEW YORK: Ieee, 2015, Conference Proceedings, pp. 5130–5135. [Online]. Available: <GotoISI>://WOS:000381554505053

[44] V. Dabin, "Commande d'un quadricoptère par rejet actif de perturbations," Ph.D. dissertation, École Polytechnique de Montréal, 2018.

[45] M. Ranjbaran and K. Khorasani, "Fault recovery of an under-actuated quadrotor aerial vehicle," in *Decision and Control (CDC), 2010 49th IEEE Conference on.* IEEE, 2010, pp. 4385–4392.

[46] X. H. Ge and Q. L. Han, "Consensus of multiagent systems subject to partially accessible and overlapping markovian network topologies," *Ieee Transactions on Cybernetics*, vol. 47, no. 8, pp. 1807–1819, 2017. [Online]. Available: <GotoISI>://WOS:000405458200001

[47] T. Hamel and R. Mahony, "Attitude estimation on so [3] based on direct inertial measurements," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* IEEE, 2006, pp. 2170–2175.

[48] T. Tomic *et al.*, "Toward a fully autonomous uav research platform for indoor and outdoor urban search and rescue," *Ieee Robotics and Automation Magazine*, vol. 19, no. 3, pp. 46–56, 2012. [Online]. Available: <GotoISI>://WOS:000309058000010

[49] M. Zimmermann and W. Sulzer, "High bandwidth orientation measurement and control based on complementary filtering," in *Robot Control 1991*. Elsevier, 1992, pp. 525–530.

[50] E. J. Lefferts, F. L. Markley, and M. D. Shuster, "Kalman filtering for spacecraft attitude estimation," *Journal of Guidance, Control, and Dynamics*, vol. 5, no. 5, pp. 417–429, 1982.

[51] A. M. Sabatini, "Quaternion-based extended kalman filter for determining orientation by inertial and magnetic sensing," *IEEE Transactions on Biomedical Engineering*, vol. 53, no. 7, pp. 1346–1356, 2006.

[52] S. J. Julier, J. K. Uhlmann, and H. F. Durrant-Whyte, "A new approach for filtering nonlinear systems," in *American Control Conference, Proceedings of the 1995*, vol. 3. IEEE, 1995, pp. 1628–1632.

[53] J. L. Crassidis and F. L. Markley, "Unscented filtering for spacecraft attitude estimation," *Journal of guidance, control, and dynamics*, vol. 26, no. 4, pp. 536–542, 2003.

[54] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part i," *IEEE robotics and automation magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[55] S. Verykokou *et al.*, "Uav-based 3d modelling of disaster scenes for urban search and rescue," in *IEEE International Conference on Imaging Systems and Techniques (IST) / IEEE International School on Imaging*, ser. IEEE International Conference on Imaging Systems and Techniques. NEW YORK: Ieee, 2016, Conference Proceedings, pp. 106–111. [Online]. Available: <GotoISI>://WOS:000388735200018

[56] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[57] B. D. Lucas, T. Kanade *et al.*, "An iterative image registration technique with an application to stereo vision," in *Artificial Intelligence (IJCAI), 1981, International Joint Conference on)*. IJCAI, 1981, pp. 674–679.

[58] A. Hast, J. Nysjö, and A. Marchetti, "Optimal ransac-towards a repeatable algorithm for finding the optimal set," *Journal of WSCG*, vol. 21, no. 1, pp. 21–30, 2013.

[59] C. Forster, M. Pizzoli, and D. Scaramuzza, "Svo: Fast semi-direct monocular visual odometry," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 15–22.

[60] S. Lynen *et al.*, "A robust and modular multi-sensor fusion approach applied to mav navigation," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE, 2013, pp. 3923–3929.

[61] M. Pizzoli, C. Forster, and D. Scaramuzza, "Remode: Probabilistic, monocular dense reconstruction in real time," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 2609–2616.

[62] T. Hamel *et al.*, "Dynamic modelling and configuration stabilization for an x4-flyer." *IFAC Proceedings Volumes*, vol. 35, no. 1, pp. 217–222, 2002.

[63] P. Pounds, R. Mahony, and P. Corke, "Modelling and control of a large quadrotor robot," *Control Engineering Practice*, vol. 18, no. 7, pp. 691–699, Jul. 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0967066110000456

[64] D. Mellinger, N. Michael, and V. Kumar, "Trajectory generation and control for precise aggressive maneuvers with quadrotors," *The International Journal of Robotics Research*, vol. 31, no. 5, pp. 664–674, Apr. 2012. [Online]. Available: https://doi.org/10.1177/0278364911434236

[65] T. Lee, M. Leok, and N. H. McClamroch, "Geometric tracking control of a quadrotor UAV on SE(3)," in *49th IEEE Conference on Decision and Control (CDC)*, Dec. 2010, pp. 5420–5425, 00545.

[66] M. Turpin, N. Michael, and V. Kumar, "Trajectory design and control for aggressive formation flight with quadrotors," *Autonomous Robots*, vol. 33, no. 1, pp. 143–156, Aug. 2012. [Online]. Available: https://doi.org/10.1007/s10514-012-9279-y

[67] E. Luczak and A. Rosenfeld, "Distance on a hexagonal grid," *IEEE Transactions on Computers*, vol. 25, no. 5, pp. 532–533, may 1976.

[68] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval Research Logistics*, vol. 52, no. 1, pp. 7–21, Feb. 1955, 08558. [Online]. Available: http://doi.wiley.com/10.1002/nav.20053

[69] J. Munkres, "Algorithms for the assignment and transportation problems," *Journal of the society for industrial and applied mathematics*, vol. 5, no. 1, pp. 32–38, 1957, 03117.

[70] R. Jonker and A. Volgenant, "A shortest augmenting path algorithm for dense and sparse linear assignment problems," *Computing*, vol. 38, no. 4, pp. 325–340, Dec. 1987. [Online]. Available: http://link.springer.com/10.1007/BF02278710

[71] T. Kazmar, "Linear Assignment Problem solver (LAPJV/LAPMOD)," Oct. 2019, 00000 original-date: 2016-04-12T09:47:17Z. [Online]. Available: https://github.com/gatagat/lap

[72] B. Gerkey, "C implementation of the Hungarian Method," 2003, 00018. [Online]. Available: http://robotics.stanford.edu/~gerkey/tools/hungarian.html

[73] "ROS - Introduction," Open Robotics. [Online]. Available: http://wiki.ros.org/ROS/Introduction

[74] J. A. Preiss *et al.*, "Crazyswarm: A large nano-quadcopter swarm," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. Singapore, Singapore: IEEE, May 2017, pp. 3299–3304, 00054.

[75] A. C. Luque, "Relative positioning system for UAVs in swarming applications," Master's thesis, Technical University of Denmark, 2018.

APPENDIX A INDOOR SIMULATION SNAPSHOTS

For visualization purposes, the following figures present an upper view of a swarm of 100 agents that follows a path in a realistic environment inspired by an existing building plan. The obstacles (i.e. walls in this case) are depicted using solid lines and the leader target path is represented with a dotted line. Agents are represented by dots that get darker as their rank decreases (e.g. the leader is black with a white aureole). Connected agents are linked by solid thin lines. At last, the leader is identified by a white halo. The full animation is available at `https://youtu.be/Q2PtnOUx4Gs`.

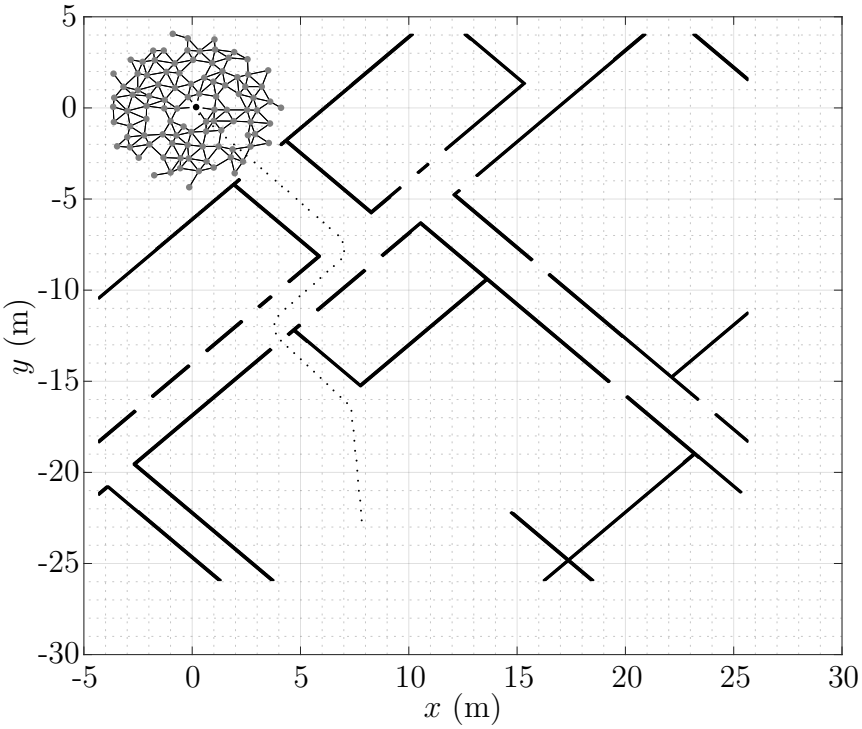


Figure A.1 Snapshot of a simulation for $N = 100$ at $t = 0$s

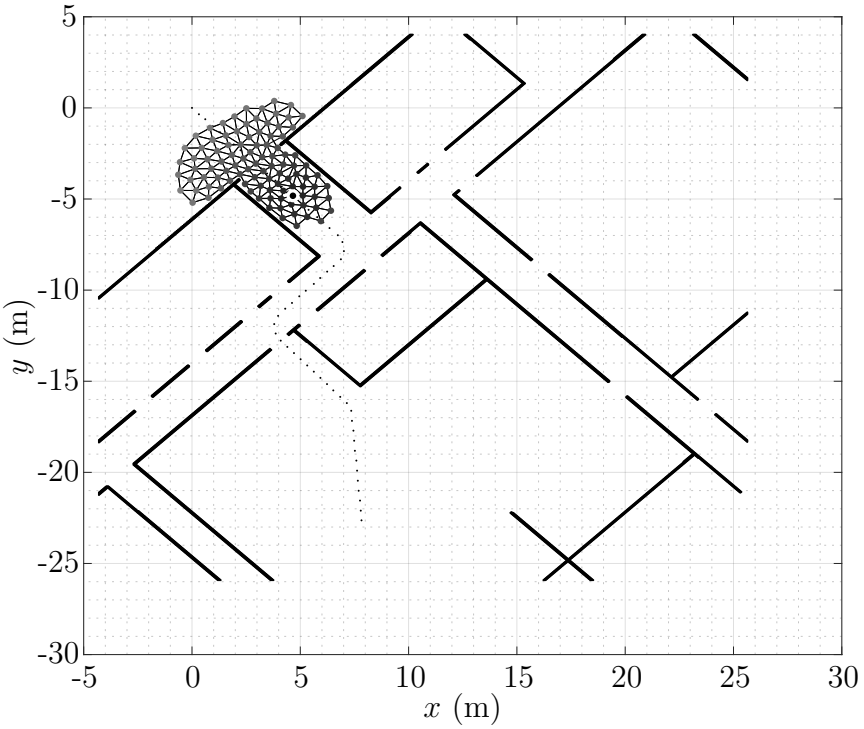


Figure A.2 Snapshot of a simulation for $N = 100$ at $t = 5$s

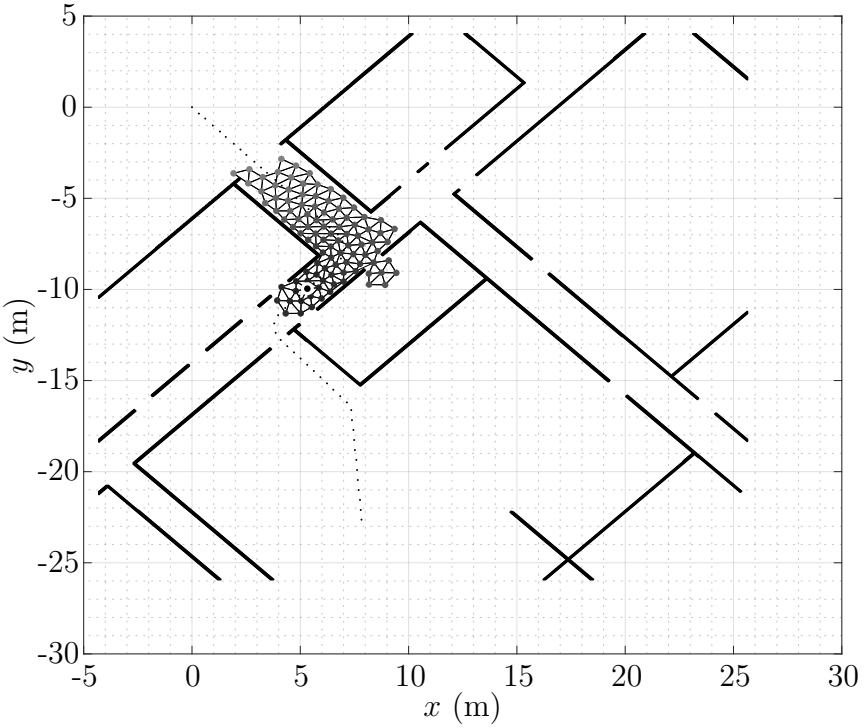


Figure A.3 Snapshot of a simulation for $N = 100$ at $t = 10$s

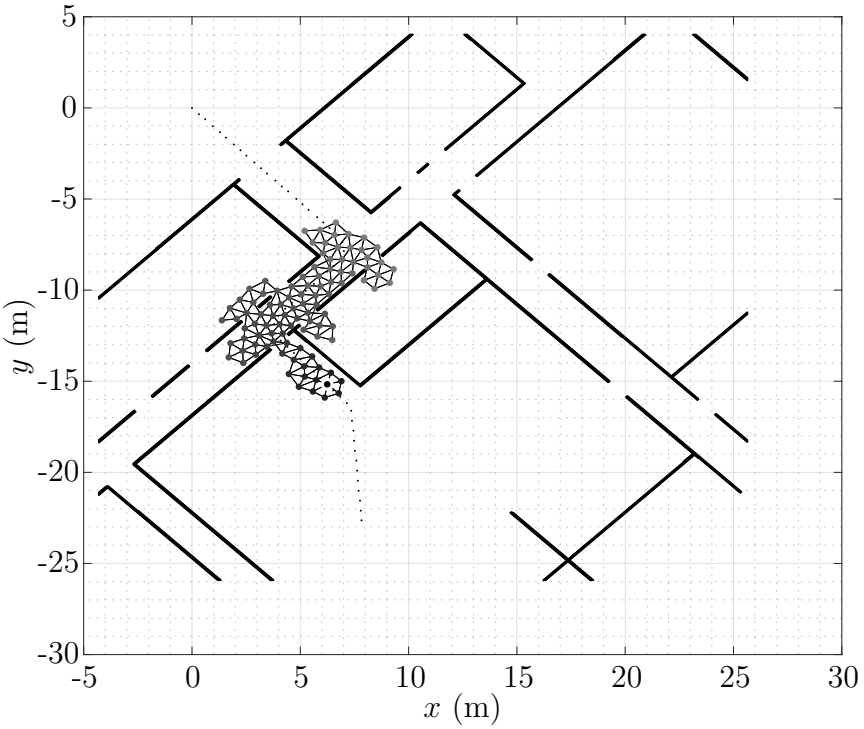


Figure A.4 Snapshot of a simulation for $N = 100$ at $t = 15$s

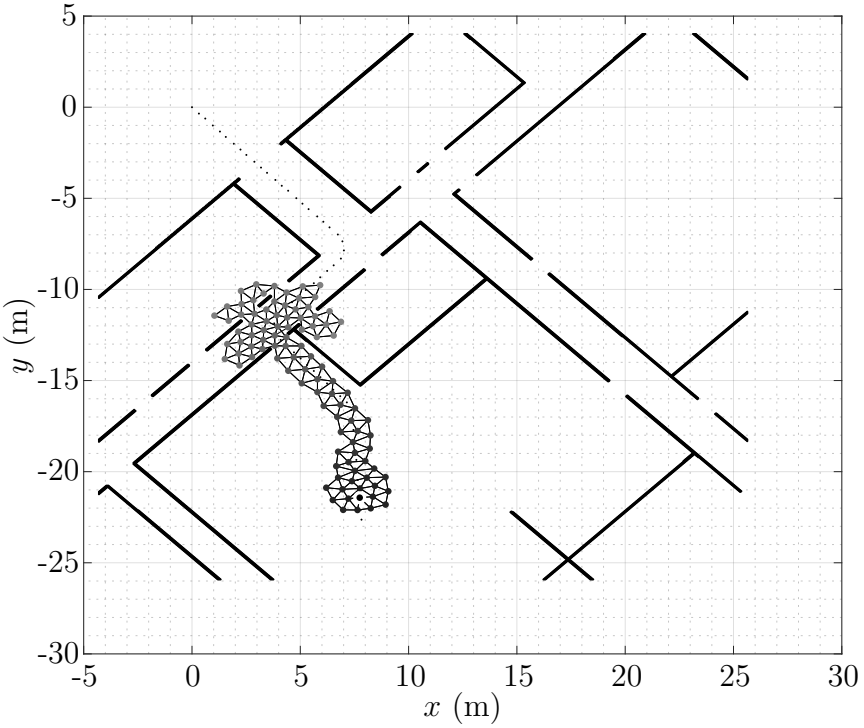


Figure A.5 Snapshot of a simulation for $N = 100$ at $t = 20$s

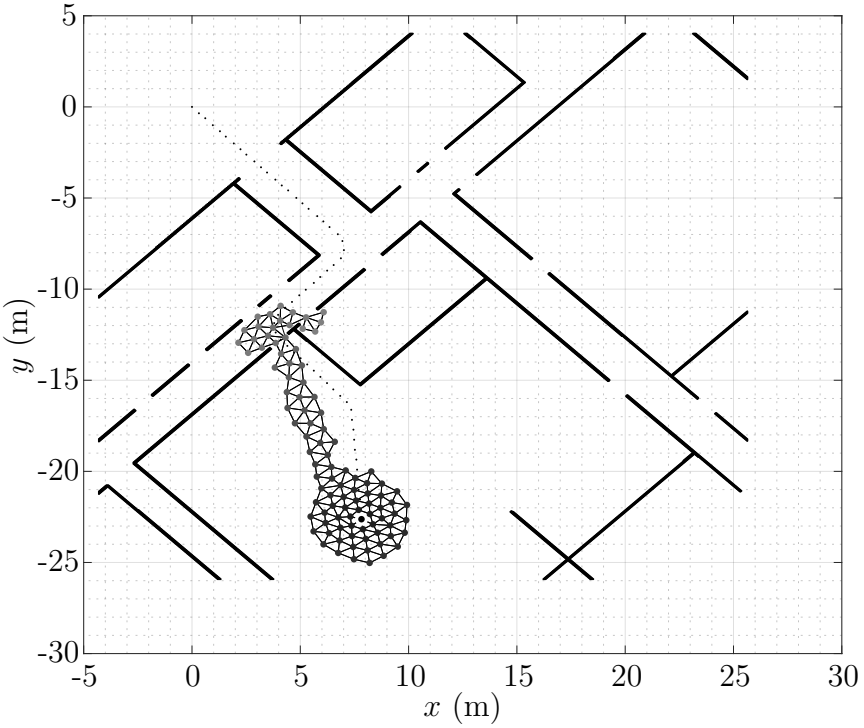


Figure A.6 Snapshot of a simulation for $N = 100$ at $t = 25$s

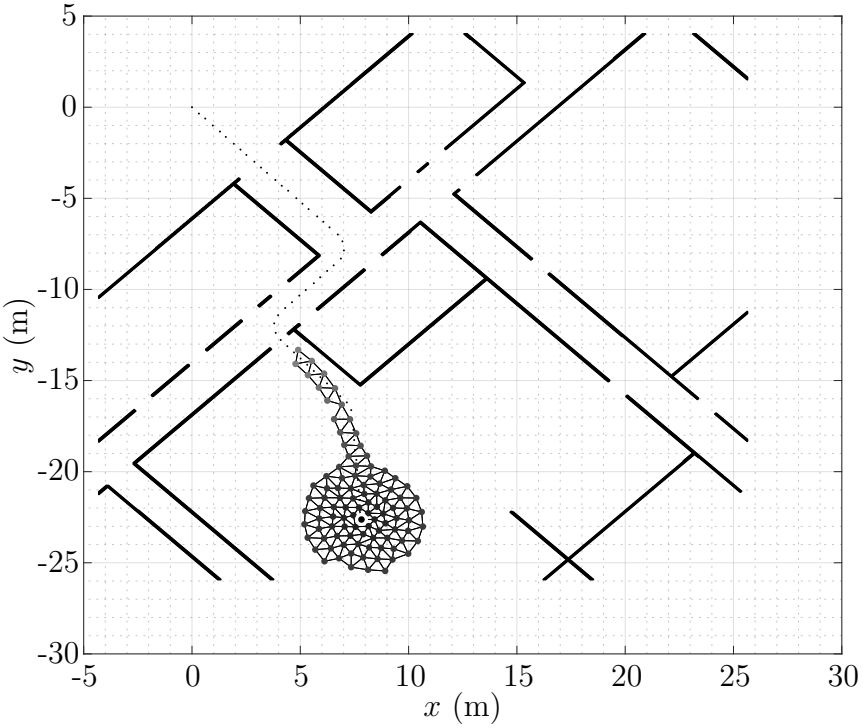


Figure A.7 Snapshot of a simulation for $N = 100$ at $t = 30$s

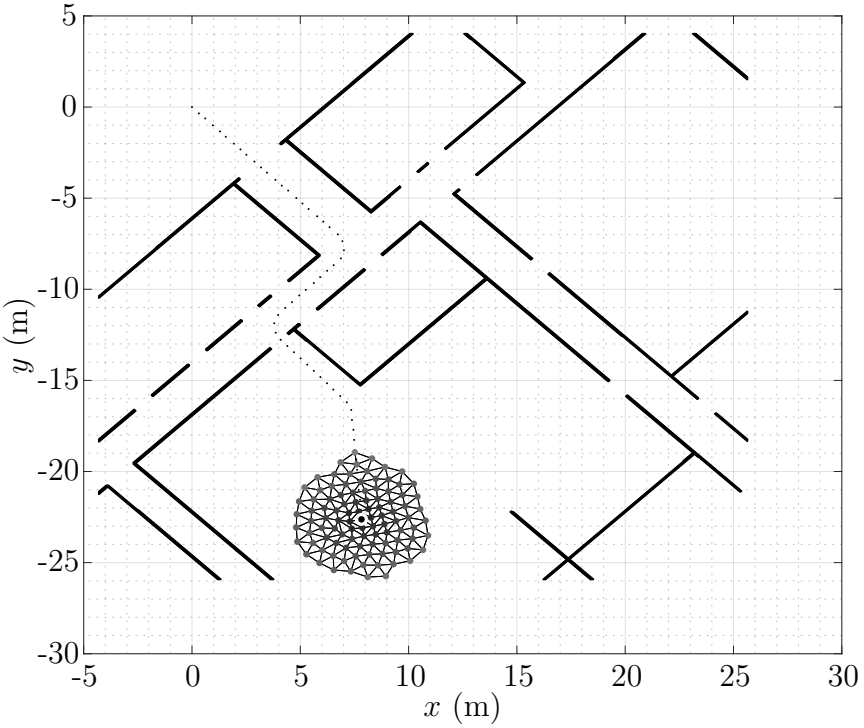


Figure A.8 Snapshot of a simulation for $N = 100$ at $t = 35$s