Automatic coordination of a quadcopters fleet using ad hoc communications

Omar Shrit

université
PARIS-SACLAY

Automatic coordination of a quadcopters fleet using ad hoc communications

Coordination automatique d'une flotte de quadcopters à l'aide de communications ad hoc

Thèse de doctorat de l'université Paris-Saclay

École doctorale n° 580 sciences et technologies de l'information et de la communication (STIC)
Spécialité de doctorat : Réseaux, information et communications
Graduate School : Informatique et sciences du numérique
Référent : Faculté des sciences d'Orsay

Thèse préparée dans l'unité de recherche **LISN (Université Paris-Saclay, CNRS)**,
sous la direction de **Steven MARTIN**
Professeur

Thèse soutenue à Paris-Saclay, le 15 décembre 2021, par

Omar SHRIT

Composition du jury

Nadjib AIT SAADI Professeur, Université Paris Saclay / UVSQ Campus	Président
Emmanuel CHAPUT Professeur, École nationale supérieure d'électrotechnique, d'électronique, d'informatique, d'hydraulique et des télécommunications	Rapporteur & Examinateur
Razvan STANICA Maître de Conférences (HDR), Institut National des Sciences Appliquées	Rapporteur & Examinateur
Steven MARTIN Professeur, Université Paris Saclay	Directeur de thèse

THESE DE DOCTORAT

NNT : 2021UPASG108

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : Coordination automatique d'une flotte de quadrotors l'aide de communications ad hoc.
Mots clés : Apprentissage itératif, apprentissage par imitation, contrôleur décentralisé, flotte de quadrotors, leader suiveur, conception de contrôleur.

Résumé : Dans cette thèse, nous étudions la conception d'un contrôleur décentralisé pour un ensemble de quadrotors. Les quadrotors sont organisés en leader et suiveurs. Le leader est piloté par l'homme, tandis que les suiveurs utilisent le contrôleur décentralisé pour suivre le leader. Les suiveurs sont autonomes et n'ont pas conscience du comportement du leader. La nouveauté de cette thèse est de s'appuyer sur des capteurs peu coûteux tels que des modules WiFi pour estimer les distances vers les quadrotors voisins. Afin de concevoir le contrôleur décentralisé, l'apprentissage itératif est utilisé et combiné à un apprentissage supervisé et par imitation, à travers plusieurs phases, notamment la collecte de journaux, la formation de modèles avancés et la conception d'un contrôleur sur celui-ci. Ensuite, le contrôleur est intégré dans les suiveurs, les rendant autonomes. Le principal avantage des méthodes d'apprentissage est de déplacer le fardeau de l'optimisation de l'étape des tests en ligne à l'étape de la collecte des données. Par conséquent, cette approche convient aux robots Commerical Off The Shelf (COTS) tels que les micro et nano quadrotors qui ne disposent pas de ressources de calcul considérables à bord. Nos méthodes ont été validées à l'aide de MagicFlock, un framework développé au laboratoire pour essaim de quadrotors qui étend RotorS, un framework de simulation Software In The Loop (SITL) construit sur le simulateur basé sur la physique Gazebo. Nos résultats ont démontré que le comportement de l'essaim est obtenu lorsqu'il est intégré à un ensemble de quadrotors à l'intérieur de Gazebo en utilisant les méthodes d'apprentissage itératif proposées avec une performance similaire à un modèle de essaim qui utilise les positions absolues des robots.

Title : Automatic coordination of a fleet of quadrotors using ad hoc communications

Keywords : Quadrotors swarm, leader-follower, iterative learning, imitation learning, decentralized controller, controller desgin.

Abstract : In this thesis, we study designing a decentralized controller for a set of quadrotors. The quadrotors are organized as a leader and followers. The leader is human-piloted, while the followers use the decentralized controller to follow the leader. The followers are autonomous and not aware of the leader's behavior. The novelty of this thesis is to rely on inexpensive sensors such as WiFi modules to estimate the distances toward neighbors' quadrotors. In order to design the decentralized controller, iterative learning is used and combined with supervised and imitation learning, through several phases, including logs gathering, training forward models, and designing a controller upon it. Then the controller is embedded in the followers, rendering them autonomous. The main advantage of learning methods is to shift the burden of optimization from the online tests step to the data gathering step. Therefore, making this approach is suitable for Commerical Off The Shelf (COTS) robots such as micro and nano quadrotors that do not have considerable computational resources on board. Our methods have been validated using MagicFlock, a home build framework for quadrotors swarm that extends RotorS, a Software In The Loop (SITL) simulation framework built on the top of the physics-based simulator Gazebo. Our results demonstrated that the swarm behavior is achieved when embedded on a set of quadrotors inside Gazebo using the proposed iterative learning methods with a performance similar to a flocking model that uses the absolute positions of the robots.

Coordination automatique d'une flotte de quadcopters à l'aide de communications ad hoc

---

Omar Shrit

Abstract

In this thesis, we study the design of a decentralized controller for a set of robots to realize the swarming behavior. The swarming behavior of robots is essential to achieve various applications in entertainment, agriculture, communications, and construction. However, current swarming controllers require heavy sensors and operate in limited indoor environments. The novelty of this thesis is to rely on inexpensive sensors such as WiFi modules to estimate the distances toward neighbors' robots (as opposed to positioning systems or cameras). The robots are quadrotors, organized as a leader and followers. The leader quadrotor is human-piloted, while the followers use the decentralized controller to follow the leader. The followers are autonomous and unaware of the leader's behavior. In this thesis, two approaches have been proposed to design the decentralized swarm controller. The first approach inspired by model predictive control aims to maintain the geometrical configuration for a set of quadrotors during the entire flight. The user defines the geometrical configuration before the flight, and the decentralized controller maintains these configurations. This approach is extended by a second approach inspired by imitation learning to realize complex policies such as flocking behavior. These propositions rely on an iterative learning strategy combined with supervised and imitation learning. Iterative learning requires several steps, including log gathering, training forward models, and designing a controller. Once the controller is designed, it is embedded in the followers for testing. The testing step is essential to assess the controller's performance and gather a new log for the next iteration. The controller improves iteratively, allowing to make the quadrotor fully autonomous. The main advantage of learning methods in comparison to traditional control methods is to shift the burden of optimization from the online tests steps to the data gathering step. Therefore, making this approach is suitable for Commerical Off The Shelf (COTS) robots such as micro and nano quadrotors that do not have considerable computational resources on board. Our methods have been validated using MagicFlock, a home build framework for quadrotors swarm that extends RotorS. MagicFlock uses the Software In The Loop (SITL) concept to benefit from simulation with a small reality gap. MagicFlock framework is also built on top of the physics-based simulator Gazebo. Our results demonstrated that the swarm behavior is achieved when embedded on a set of quadrotors inside Gazebo using the proposed iterative learning methods with a performance similar to a flocking model that uses the absolute positions of the robots.

Synthèse

Dans cette thèse, nous étudions la conception d'un contrôleur décentralisé pour un ensemble de robots pour réaliser le comportement d'essaimage. Le comportement d'essaimage des robots est essentiel pour réaliser diverses applications dans le divertissement, l'agriculture, les communications et la construction. Cependant, les contrôleurs d'essaimage actuels nécessitent des capteurs lourds et fonctionnent dans des environnements intérieurs limités. La nouveauté de cette thèse est de s'appuyer sur des capteurs peu coûteux tels que des modules WiFi pour estimer les distances vers les robots des voisins (par opposition aux systèmes de positionnement ou aux caméras). Les robots sont des quadrotors, organisés en leader et suiveurs. Le leader est piloté par l'homme, tandis que les suiveurs utilisent le contrôleur décentralisé pour suivre le leader. Les suiveurs sont autonomes et inconscients du comportement du leader. Dans cette thèse, deux approches ont été proposées pour concevoir le contrôleur d'essaim décentralisé. La première approche inspirée de la commande prédictive de modèles vise à maintenir la configuration géométrique d'un ensemble de quadrotor pendant tout le vol. L'utilisateur définit la configuration géométrique avant le vol et le contrôleur décentralisé maintient ces configurations. Cette approche est prolongée par une seconde approche inspirée de l'apprentissage par imitation pour réaliser des politiques complexes telles que le comportement d'essaim. Ces propositions reposent sur une stratégie d'apprentissage itérative combinée à un apprentissage par imitation. L'apprentissage itératif nécessite plusieurs étapes, notamment la collecte de données, la formation de modèles avancés et la conception d'un contrôleur. Une fois le contrôleur conçu, il est intégré dans les suiveurs pour être testé. L'étape de test est essentielle pour évaluer les performances du contrôleur et rassembler des nouvelles données pour la prochaine itération. Le contrôleur s'améliore de manière itérative, permettant de rendre le quadrotor totalement autonome. Le principal avantage des méthodes d'apprentissage par rapport aux méthodes de contrôle traditionnelles est de déplacer la charge d'optimisation depuis les étapes de tests en ligne jusqu'à l'étape de collecte de données. Par conséquent, cette approche convient pour "Commerical Off The Shelf (COTS)" robots tels que les micros et nano quadrotors qui n'ont pas ressources de calcul considérables à bord. Nos méthodes ont été validées à l'aide de MagicFlock, un framework développé au laboratoire pour essaim de quadrotors qui étend RotorS. MagicFlock utilise le concept Software In The Loop (SITL) pour bénéficier d'une simulation avec un petit écart de réalité. Le logiciel MagicFlock est également construit au-dessus du simulateur basé sur la physique Gazebo. Nos résultats ont démontré que le comportement en essaim est obtenu lorsqu'il est intégré à un ensemble de quadrotors à l'intérieur de Gazebo en utilisant les méthodes d'apprentissage itératives proposées avec une performance similaire à un modèle de flocage qui utilise les positions absolues des robots.

Acknowledgement

J'ai passé six ans dans le Laboratoire de Recherche en Informatique (LRI) qui est devenu récemment le Laboratoire Interdisciplinaire de Science du Numérique (LISN). Deux ans comme ingénieur d'étude et stagiaire et quatre ans comme un doctorant. Pendant cette période, j'ai eu la chance de rencontrer beaucoup de personnes de cette unité de recherche ainsi que des personnes d'autres unités de recherche. J'ai eu la chance de participer à des projets intéressants qui ont enrichi mes connaissances en Réseau, Intelligence Artificielle et Robotique. La fin de cette thèse ferme un chapitre dense d'événements de ma vie qui est aussi inoubliable.

Je commence par remercier les membres de mon jury de thèse : Emmanuel Chaput et Razvan Stanica pour leurs retours pertinents sur le manuscrit de thèse et pendant la soutenance. Je remercie surtout le président de jury Nadjib Ait Saadi qui a organisé la soutenance de la thèse ainsi toutes les démarches administratives liées à la fin de ma quatrième année de thèse. Je remercie aussi l'école doctorale pour l'organisation de ma mi-soutenance et la soutenance finale.

Je remercie mon directeur de thèse Steven Martin de m'avoir supervisé pendant cette thèse. Je remercie aussi Dominique Quadri et Nicola Roberto Zema pour leur aide pendant cette thèse et les discussions que nous avons partagées. Ainsi je remercie tout les membres d'équipe ROCS pour les bons moments que nous avons vécus.

Je remercie aussi les membres du comité de mi-soutenance : Guillaume Lozenguez et David Filliat pour leurs retours sur ma thèse. Je tiens aussi à remercier David Filliat qui a accepté d'être mon encadrant pour la troisième année de thèse.

Je tiens à remercier chaleureusement Emmanuelle Rio et Frédéric Baudin qui m'ont guidé administrativement pendant cette thèse et qui m'ont accompagné face aux difficultés en donnant des conseils sur les démarches de thèse. C'est grâce à leur support que cette thèse a vu le jour et abouti.

Je remercie finalement Michèle Sebag pour son encadrement précieux pendant cette thèse. Elle a trouvé le temps de répondre à mes questions avec patience et rigueur et en partageant son large savoir expérimental et théorique. Michèle a répondu à toutes mes questions en m'accordant du temps et en me donnant des suggestions et des conseils sur ma thèse et aussi sur la vie. Cela m'a permis d'explorer le domaine d'Intelligence Artificielle que je n'ai pas eu ni la chance ni

l'opportunité d'apprendre auparavant. C'est grâce à Michèle que ces contributions de thèse ont eu lieu. Je remercie Marc Schoenauer, Guillaume Charpiat, et Herilalaina Rakotoarison pour sa bonne humeur et son humour, ainsi que tous les membres de l'équipe TAO.

Finalement je remercie la directrice de notre laboratoire Johanne Cohen ainsi que les membres de l'équipe SAMI de support technique pour avoir répondu à toutes mes questions et mes mails pendant ces six ans. Je remercie aussi le directeur de FabLab Romain di Vozzo de m'avoir aidé à apprendre beaucoup de techniques sur la fabrication numérique cela m'ouvert les yeux sur un domaine qu'on n'apprend pas à l'université et qui est aussi très important dans la Robotique.

Merci surtout à ma compagne qui partage ma vie et qui me soutient. Merci à ma mère et à mes frères pour leur présence.

Contents

Acronyms

k-NN k nearest neighbor. 14, 68–70, 72, 110, 136

AI Artificial Intelligence. 5–7

AoA Angle of Arrival. 86, 97

AODV Ad hoc On-Demand Distance Vector. 98

API Application Programming Interface. 50, 51, 105–108, 111, 138

ARC Aerial Robotic Construction. 11

BLDC Brushless Direct Current. 30

C-CAPT Centralized-Concurrent Assignment And planning of Trajectories. 41

CAPT Concurrent Assignment and Planning of Trajectories. 36, 41

CDF Cumulative Distribution Function. 69, 70

CMA-ES Covariance matrix adaptation evolution strategy. 38, 97

COTS Commercial of-the-shelf. 13, 43, 44, 46, 57, 86, 97, 113

CSI Channel State Information. 97

D-CAPT Decentralized-Concurrent Assignment And planning of Trajectories. 41

DAGGER Dataset Aggregation. 14, 84, 85, 91, 114

dBm decibel-milliwatt. 46, 50

DC Direct Current. 30

DDPG Deep deterministic Policy Gradient. 39

DMPC Decentralized Model Predictive Control. 23, 37

EA Evolutionary Algorithms. iv, 29, 38, 97

GB Giga Bytes. 110

GNSS Global Navigation Satellite System. 12, 20–22, 32, 35, 41, 43, 49, 80, 81, 97

GPU Graphical Processing Unit. 86, 110

I2SL Iterative Imitation Supervised Learning. v, 8, 14, 79, 81, 83–85, 87–91, 96, 98, 110, 114, 138

IL4MRC Iterative Learning for Model Reactive Control. iv, v, 13, 14, 57–74, 76, 79, 80, 84, 85, 110, 113, 114, 136, 137

IMU Inertial Measurement Unit. 20, 31

IRL Inverse Reinforcement Learning. iii, 7, 26, 27, 59, 81

ISM Industrial, Scientific, and Medical. 45, 46

ITU International Telecommunication Union. 45

LIDAR Light detection and ranging. 21, 22, 76, 108

LOS Line-of-sight. 9, 45

MAS Multi-agent System. 81, 102, 114

MAV Micro Aerial Vehicles. 8, 30, 116

MAVLINK Micro Aerial Vehicles LINK. 105–107, 138

MAVSDK Micro Aerial Vehicles Software Development Kit. vi, 101, 105–107, 138

MDP Markov Decision Process. 25

MILP Mixed-Integer Linear Programming. 36

MIQP Mixed-Integer Quadratic Programming. 41

MPC Model Predictive Control. iii, iv, 13, 19, 22, 23, 29, 37, 41, 59, 60, 113

MUSIC MUltiple SIgnal Classification. 97

mW milliwatt. 46

NLOS Non-line-of-sight. 45

NN Neural Network. 65

NTC Negative Temperature Coefficient. 20

OCP Optimal Control Problem. 41

OFDM Orthogonal Frequency Division Multiplexing. 97

OLSR Optimized Link State Routing. 98

OOP Object-Oriented Programming. 106

PID Proportional, Integral, Derivative. 39, 40

PPO Proximal Policy Optimization. 39, 40

PSO Particle Swarm Optimization. 38

PTC Positive Temperature Coefficient. 20

PVT Position, Velocity, and Time. 21

RADAR Radio detection and ranging. 21

RAM Random Access Memory. 110

RFID Radio Frequency Identification. 41

RGB-D Red Green Blue Depth. 20, 21, 76

RL Reinforcement Learning. iii, 7, 25, 26, 39, 59, 81, 108

ROS Robot Operating System. 103, 104

RSS received signal strength. 46–52, 54, 62, 86, 97, 108, 113

RSSI Received Signal Strength Indicator. 46, 54, 135

RTK Real Time Kinematic. 12

SAR Search And Rescue. 10, 12

SCP Sequential Convex Programming. 41

SDF Simulation Description Format. 109

SITL Software In The Loop. 8–10, 14, 31, 41, 69, 90, 98, 103, 106, 114, 115

SLAM Simultaneous Localization and Mapping. 21, 22

SoC System on Chip. 41, 51

SONAR Sound Navigation and Ranging. 21

SVM Support Vector Machine. 80

UAV Unmanned Aerial Vehicles. iii, 7, 8, 10–12, 27, 30, 36, 103, 106

UDP User Datagram Protocol. 98

UGV Unmanned Ground Vehicles. 37

UWB Ultra-Wideband. 91

VIO Visual Inertial Odometry. 36

VTOL Vertical Take-Off and Landing. 30

XML eXtensible Markup Language. 109

ZMP Zero Moment Point. 7

Introduction

1

In ancient Mesopotamia, around 3500 BC, Sumerians developed an entire system to extract information from their brain to store it on a set of clay tablets (Robinson, 2009). The system is known as *writing* and is used to treat information that the human brain is not able to manipulate efficiently, such as numbers and mathematical calculations (Robinson, 2003). Developing a writing system was critical during that period, especially to transfer agricultural information from one individual to another or to store a thousand bytes of data related to taxes and payments. It allowed humans to do complex high-level tasks that require a decent amount of abstraction, such as poetry or philosophy. For example, it allowed the pioneering philosopher Aristotle around 350 BCE to describe the traditional logic (Lear, 1980; Smith, 2000) through his work on reasoning and inferences. Although, the aim of traditional logic is not to prove new facts but instead to *automate* the process of inference with the help of writing. In the 19th century, after two thousand years of Aristotle logic, George Boole investigated the work of Aristotle extended it to cover a wide set of applications in his work *The laws of Thoughts* (Boole, 1854, 1957). Bools also founded Boolean algebra in his book *The Mathematical Analysis of Logic* (Boole, 1847) which allowed describing the logical operation in a binary form. Boole's works on algebra and logic were fundamental for Artificial Intelligence (AI) one hundred years later.

Artificial Intelligence was founded in 1956, during a workshop organized by Marvin Minsky and John McCarthy (McCarthy et al., 2006) at Dartmouth College. They defined AI as creating intelligent machines that resolve complex problems for humans to solve, with the capacity of the machine or the robot to improve itself over time. At that period, AI research was focused on logic and solving problems such as playing chess or prove mathematical theories. Researchers predicted that in a couple of decades, AI would resolve all complex issues with the help of intelligent robots that can think like humans. However, they underestimated the number of efforts required to resolve these tasks. In addition, there was an over expectation of the impact of AI embedded technologies in daily life aspects. All of these led to what is known as the AI winter (Schank, 1991).

Hans Moravec, Rodney Brookes, and their colleagues analyzed issues that led to the AI winter, and why AI was successfully able to resolve complex issues, yet unable to understand simple ones. Their conclusion was related to understanding and representation of human intelligence,

as in that period, solving mathematical challenges was considered difficult. However, tasks that were done effortlessly by humans were not considered as requiring intelligence. Hans Moravec described this paradox in his book *Mind children: The future of robot and human intelligence* (Moravec, 1988) as follows:

> Encoded in the large, highly evolved sensory and motor portions of the human brain is a billion years of experience about the nature of the world and how to survive in it. The deliberate process we call reasoning is, I believe, the thinnest veneer of human thought, effective only because it is supported by this much older and much more powerful, though usually unconscious, sensorimotor knowledge. We are all prodigious Olympians in perceptual and motor areas, so good that we make the difficult look easy. Abstract thought, though, is a new trick, perhaps less than 100 thousand years old. We have not yet mastered it. It is not all that intrinsically difficult; it just seems so when we do it.

Moravec's paradox shows that humans' brains were not evolved to store a considerable amount of data or make a complex logical deduction, but instead, they evolved to store visual information that represents the patterns and shapes of surrounding objects. Thus, humans' capacity for perception and control progressed due to billions of years of evolution. For example, a one-year-old child can build a tower of cubes quickly. The *unconscious* baby brain needs to execute two steps to build the tower. The first step is related to *perception*, in which the brain needs to analyze visual input in order to identify the cube. The second step is related to *control* since the child needs to pick the cube and place it at the top of the tower. Contrary to humans, perception and control challenges are difficult to resolve by robots since they do not benefit from a brain resulting from biological evolution. Nor do they have a sufficient amount of computational capacities onboard (Moravec, 1988) .

Today, building autonomous robots that do not require external intervention seems challenging, as the perception and control problems are complex to resolve. Therefore, there have been two visions to resolve the challenge. On the one hand, one might think of programming the robot manually to execute one specific task bounded by a broad set of constraints to improve efficiency. In this case, the design is learned by engineers as they will imitate the biological evolution in order to reverse engineer each task. In addition, several successive versions of engineering are required to have a good design. This method has been used by several companies such as Boston Dynamics (*Boston Dynamics* 2021). They are keen on using control theory on their robots, as stated by their engineers. For instance, the Zero Moment Point (ZMP) algorithm can be used to

allow legged robots to work. Also, software packages such as Drake (*Drake* 2021) can be used to design model-based robots.

On the other hand, AI can be used to resolve the problem of perception and control. In this case, the design is not learned by engineers but instead is learned by the machine itself. This process is known as Machine Learning and is defined as allowing machines to learn how to execute a task, similar to how biological entities learn. For example, humans learn particular tasks by employing the trial and error principle or by imitating others. In addition, the learning process involves recording the task visually or registering it using the writing system. Similarly, when a robot is trying to learn a task, it needs access to the registered experiences that represent the task. Then in order for the robot to learn effectively, it needs to use a learning algorithm. In the last 30 years, there has been extensive research in this domain in order to learn from registered experiences (Kober et al., 2013; Mosavi and Varkonyi, 2017), such as Reinforcement Learning (RL) and imitation learning, and Inverse Reinforcement Learning. These methods are more suitable for robotics and control than traditional machine learning algorithms due to their capacity to predict sequences of optimal motor actions.

There has been a tremendous effort to develop robots in the past two decades, particularly telerobots and autonomous robots such as unmanned vehicles, either by programming them or using machine learning algorithms. The acceleration of this development is due to the miniaturization of the size of electronic and micro-electro-mechanical components allowed the manufacturing process of microcontrollers and sensors to become faster and more affordable. *Unmanned Vehicles* are defined as any moving vehicle that does not have a human pilot onboard, whether it is an aircraft, ground, or underwater vehicle. At first, they attracted military attention (Glade, 2000) in order to protect soldiers during warfare, they later showed a broader set of applications in the civil sector (Herrick, 2000; Shakhatreh et al., 2019). In this thesis, we are interested in teleoperated aircraft, known as Unmanned Aerial Vehicles (UAV). The UAVs used in the civil domain usually consists of simple components allowing the user to modify these robots easily by adding or removing a specific sensor according to the executed mission. This capacity of adaptation allowed to open a new set of applications that were not feasible in the past. UAVs can be used in different domains either individually or collectively and can be combined with other robots to accomplish a specific mission.

Due to the above advantages related to UAV in terms of mechanical and electronic simplicity, size, affordability, and applications, this robot presents a perfect candidate to study the problem of perception and control either by programming the robot or by using machine learning algorithms. The past decade has shown extensive research in this domain for single autonomous UAV (Al-Kaff et al., 2018). In this thesis, we focus on studying the same problem for a set of UAV instead of a

single robot. Our objective is to allow for these UAVs to realize together complex behavior such as swarming (Chung et al., 2018). Our preliminary results achieved before the start of this thesis were based on programming the UAV to do basic swarming behavior when the robots have a low perception level of their neighbors. While in this thesis, we focused on using machine learning to learn this behavior to make these robots fully autonomous. The following section describes more in detail the problem statement and the objectives defined in this thesis.

1.1 Thesis objective and problem statement

This thesis aims to study the design of a decentralized controller using learning algorithms for a set of UAVs in particular Micro Aerial Vehicles (MAV) under a set of constraints related to their perception of the environment. The MAV type is a quadrotor [1]. The quadrotors are homogeneous, dynamic, deployed in a simulation environment having little computational resources.

The objective of this thesis is as follows: Firstly, we propose a learned controller that maintains a set of UAVs in a specific formation during the entire flight. Secondly, we propose an improved controller allowing quadrotors to swarm. Both controllers should be decentralized and render the UAV autonomous. Finally, the controller should be validated using Gazebo, a high-fidelity robotic simulator based on a Software In The Loop (SITL) simulation. The concept of I2SL is used in order to reduce the reality gap since we do not realize any test on real UAVs.

1.1.1 Problem statement

Pilot in the loop The problem we are investigating in this thesis is described as follows: consider a pilot with a set of quadrotors. The pilot can handle only one quadrotor at a time for a specific mission, conditioned by the fact that the pilot has a direct line of sight toward this quadrotor. The novelty in this thesis is to include the pilot in the swarm of quadrotor, allowing the operator to fly several quadrotors simultaneously in a similar manner to one quadrotor.

In this thesis, we investigate: *if Would it be possible for a pilot to handle a swarm of quadrotors as it is the case for one quadrotor?*

[1] see 3.2 for more details about this type of UAV

1.1.2 Proposed solutions

Learn decentralized controllers We propose a simple leader-follower principle to be used entirely during the thesis. The leader quadrotor is human-piloted. While all the remaining robots will be autonomous followers. The pilot will operate only the leader. The followers should be fully autonomous and aware of each other and the leader. In most countries, the pilot is conditioned by the Line-of-sight (LOS) rule toward the leader. Due to this condition, we suppose that no collision is possible with any static/dynamic object since the operator can see where the leader is going. Each follower will embed a decentralized controller onboard, allowing it to follow the leader smoothly and achieve cooperative behavior between the agents.

In this thesis, we learn the controller progressively. First, a decentralized learned controller is proposed using an iterative approach. It addresses primary swarm control for a set of quadrotors given a specific geometric pattern that must be respected during the entire flight. Second, a more complex swarm behavior is addressed, building on the controller learned previously. We extract more information from the wireless sensor and apply an iterative approach combined with agile imitation and supervised learning to learn more complex policies. As a result, the controller learns and improves the complex behaviors iteration by iteration. We demonstrate that complex behavior such as swarming can be achieved even with minimal perception of the environment.

As said, the challenges faced in this thesis are related to the complexity of multi-agent control, since it is severely increased compared to the control of a single-robot (Wray and Zilberstein, 2019): it differs from the centralized control setting [2], where the state and action space are completely observed. Finally, such a decentralized controller does not have to be manually designed. The state and action space is multiplied by the number of agents and the complexity of such a system increases exponentially as the number of agents increases. We demonstrate that a *learned* controller iteratively inside a simulator that is based on cheap sensors can behave as an oracle with full the omniscience of the environment.

Software In The Loop (SITL) The final challenge is to validate the decentralized controllers on a set of quadrotors. However, because of the lack of necessary material such as quadrotor hardware and the testbed to provide the ground truth, we have decided to validate the controller inside the simulator. The simulation aims to be as close as possible to reality in order to validate the results. The proposed simulation is known as Software In The Loop has been developed in order to reduce the cost of testing autopilot software on quadrotors hardware to validate features and

[2] more details in section 3.4

functionalities. Thus, reducing the cost of the robots. The autopilot software is tested inside a compliant physics simulator such as Gazebo. In this work, we extend an existing SITL framework known as RotorS, to be compatible with a set of quadrotors instead of one robot. This framework we developed called MagicFlock allows us to realize experiments in order to validate the designed controller.

1.2 Unmanned Aerial Vehicles applications domain

In order to show the significance of the conducted research, we present in this section the applications domain for UAV. They are currently being used in traffic monitoring, disaster and wildfire management, package delivery, Search And Rescue (SAR) missions, provide internet and network access, cinematography, monitoring archeological sites, or observing environmental and climate changes. In this thesis, we were able to identify a set of applications that can employ our swarms composed of autonomous followers and piloted leaders. Thus, the following discusses, proposes, and foresees civil applications for these swarms:

Agriculture : UAV usage might be ideal for monitoring soil and crops. In the past, farmers used satellite imagery to monitor field variation. Yet, precision agriculture was minimal. According to (Whipker and Akridge, 2008), only 30 percent of farmers in the USA have access to geospatial techniques. This low rate is due to the lack of high image resolution, weather conditions, sensor limitation, and the high cost of accessing and processing these images. Today, small UAVs provide an ideal alternative. Their ability to provide high-resolution images at low flying altitude, makes them perfect for crop management (growth and yield), monitoring weeds and insects. State-of-the-art reviews (Carbone et al., 2018; Cerro et al., 2021; Mukherjee et al., 2019) discuss this subject deeply, as using multiple UAVs together is becoming crucial. For instance, In Australia and China, farm management is quite a dilemma. Farm size could reach several million acres, with several thousand cattle. Thus, several UAV swarms could be deployed rather than helicopters to deal with the situation and reduce the cost to the tenth order.

Building structures : The main objective when designing an UAV is the possibility to carry a payload and deliver it safely. It is possible to use a team of quadrotors in order to construct buildings and structures (Lindsey et al., 2011a, 2012; Santos et al., 2018). This idea helped in creating a new architectural field is known as Aerial Robotic Construction (ARC) (Willmann et al., 2012), in which the construction process relies heavily on a team of robots during the entire

pipeline starting from material fabrication, transportation until the positioning of the object. Such a proposal can extensively reduce construction costs, which are unfortunately highly dependable on heavy vehicles, machinery, and manual labor.

Cinematography : UAVs with embedded cameras were shown a great interest in shooting videos and scenarios (e.g., public gatherings, sports events). However, using these robots directly in the cinematography industry might be a little more complicated. Several researchers have proposed methods (Passalis et al., 2018) to improve control of cameras or to automate trajectory generation, specifically to smooth the transition between camera shots (Galvane et al., 2017). A team of autonomous UAV can cooperate to improve lighting conditions from different angles in a specific scene (Kratky et al., 2021), or to improve shots quality (Alcántara et al., 2021; Torres-González et al., 2017)

Communication infrastructure : Several research projects proposed the usage of a set of UAV in order to improve communication infrastructure (Bithas et al., 2019) in remote areas (Wei et al., 2014; Xu et al., 2018) or to provide communication in a specific region in the case of post-disaster scenarios (Mohamed et al., 2020; Saif et al., 2020). At the same time, several major industrial companies attempted to use UAV to improve communication. For instance, Google started the loon project (Katikala, 2014; Nagpal and Samdani, 2017) in which they sent balloons into stratosphere within the objective is to provide a free internet connection to remote areas, while Facebook designed a solar-powered UAV called Aquila (Zuckerberg, 2016) that intended to provide internet connection to the earth and rely upon the information to other UAV using laser beams. Both of these projects were discounted later for financial and technical reasons. However, they have already achieved several milestones and demonstrated feasible use cases of using UAV for communication.

Ecological conservation : The usage of UAVs have shown a great interest in protecting the environment (Cárdenas et al., 2005). These robots detect gas leaking (Berman et al., 2012; Yang et al., 2018) by embedding a compact gas analyzer explicitly designed for flying robots. Similarly, these robots can monitor the air quality (Malaver et al., 2015) in cities in order to determine in real-time the pollution level and the percentage of greenhouse gases (Allen et al., 2019). In addition to air pollution, a set of UAVs can cooperate in monitoring the status of the ice in the Arctic, the Antarctic, and other ice sheets in the earth, which are known as the cryosphere (Bollard-Breen et al., 2015; Goebel et al., 2015; Turner et al., 2014).

Entertainment : Quadrotors appear to be good athletes (Brescianini et al., 2013; Müller et al., 2011; Ritz et al., 2012), their capacity to realize low flight speed and high maneuver at the same time provide them with acrobatics ability. Several videos [3] [4], show quadrotors react as professional competitors and as good as skilled dancers. The possibility to have a group of quadrotors dance together has attracted several business activities. For example, in 2016, Intel has realized a swarm of 100 quadrotors outdoor. The goal was to accompany the musical rhythm of an orchestra; later, they realized the same show with 500 quadrotors (*Intel drones light show* 2019). In 2018, they developed small quadrotors and conducted the same show in an indoor environment. Nowadays, quadrotors show have become very popular in international events (*Drone display* 2021; *Ehang* 2019). However, knowing that the above systems have several constraints, they use a Real Time Kinematic (RTK) differential GNSS to localize robots. While tolerating collisions among robots due to the added frame that protects each one.

Search And Rescue : Due to the capacity of UAV of flying at low altitude, following a predefined trajectory, and capturing a real-time video, a set of UAV can be used in post-natural disaster for Search And Rescue missions. For instance, it can be deployed to detect survivors upon an earthquake (Qi et al., 2016) or it can save workers in offshore oil rigs (Dol, 2020; Eid and Sham Dol, 2019). In addition, these robots can search for someone missing in wild locations (Adams et al., 2009; Goodrich et al., 2009) and realize damage assessment mission after a hurricane (Murphy et al., 2006).

Transportation : It is possible to use a set of quadrotors to lift heavy loads (Bacelar et al., 2019). For example, several quadrotors attached to these loads can help in transporting them from one place to another (Loianno and Kumar, 2018; Mellinger et al., 2010; Wang et al., 2018) Or in a swarm scenario, each quadrotor can be attached to a part of a big arena (Cardona et al., 2019) that can be used in order to transport several heavy payloads. This payload could be bricks, in which a swarm of UAVs could collaborate in order to carry out building constructions.

1.3 Statement of contributions

In Chapter 2, we present a general review of literature in the domain of perception and control for robots. The perception part provides a review and definitions of standard sensors used in most

[3] https://www.youtube.com/watch?v=XxFZ-VStApo
[4] https://www.youtube.com/watch?v=2N_wKXQ6MXA

robots. While the control parts focus on methods related to the work presented in this thesis, such as Model Predictive Control (MPC) and learning algorithms.

In Chapter 3, we present a review of literature in the domain of quadrotors swarm as it is the main topic focused on in this thesis. An introduction of the robot is presented, followed by a mathematical description of flocking models developed from observation of the animal kingdom. In addition, we review the design of the swarm controller for quadrotors. The controller can be centralized or decentralized, manually designed or learned. We focus on reviewing only recent methods related to the quadrotor controller and how they are applied in trajectory planning and formation control for a set of quadrotors. Finally, we review the state-of-the-art method in imitation learning and reinforcement learning and their usage in the controller design for quadrotors.

In Chapter 4 we present initial preliminary results achieved shortly before the thesis. A manually designed decentralized controller for a set of followers quadrotors. This method relies on Commercial of-the-shelf (COTS) robot and wireless sensors such as WiFi. The sensor allows the estimation of the distance to neighbor quadrotors by analyzing the received signal strength from the transmitter. At first, we aim to design a simple policy that relies on only wireless sensor data. Such a policy can be described by allowing the pilot to control the leader on one axis only (for example, pitching forward or backward). This assumption of reduced action spaces allows demonstrating a primary proof of concept to examine if such a controller can be possible. We have realized a direct test on a real robot known as the Bebop quadrotor manufactured by Parrot to validate this approach. We show a correlation between the variation in signal strength and the quadrotor speed.

In Chapter 5, we present Iterative Learning for Model Reactive Control (IL4MRC) as a decentralized controller-based entirely on iterative learning approach. The proposed method is inspired by MPC and is intended to be used on small-sized quadrotors that have low computational resources onboard. Similar to chapter 4, IL4MRC relies on neighbor distance estimation using a Commercial of-the-shelf wireless sensors. The aim of the IL4MRC controller is to preserve the geometric pattern of the swarm that is defined at the initial state. The leader quadrotor is provided with a random controller to simulate the human pilot, while the followers embed an IL4MRC controller. IL4MRC tries to find the optimal action to execute in the next time step to satisfy the initial property of the system. The flight continues as the system is preserved. IL4MRC is validated using the MagicFlock simulation framework and compared to several baselines such as a random controller or a k nearest neighbor controller.

In Chapter 6, we present Iterative Imitation Supervised Learning (I2SL) as a decentralized controller for a set of quadrotors using an Iterative Imitation Supervised Learning approach. The objective of this controller is to achieve more complex policies than the one achieved in IL4MRC such as the flocking behavior using only the same cheap wireless sensor embedded on each follower quadrotor. This approach differs by extracting more information from the wireless sensors, such as the signal's angle of arrival from a specific transmitter. In addition to the signal strength, these data enrich the state information for each quadrotor. I2SL differs from IL4MRC by using a reduced imitation learning technique known as Dataset Aggregation (DAGGER) in which the learned policy is alternated with other policies to improve learning. To validate this method, we have executed the experiments in MagicFlock. We demonstrate that I2SL can achieve a performance similar to the one executed by the Oracle (Flocking model).

In Chapter 7 we present MagicFlock simulation framework that is developed during the thesis which used to obtain results in chapters 5, and 6. The framework allows accurate simulations using the Gazebo simulator and SITL concept. This framework builds on state-of-the-art RotorS simulator. It differs from RotorS by allowing the simulation of several quadrotors simultaneously. This framework is designed to be simple and allow fast prototyping by hiding complexities related to quadrotors from the user. In addition, this framework is not only designed for machine learning approaches. Users can use MagicFlock to design controllers manually or control a set of quadrotors to simulate a specific mission. For instance, users can allow an entire set of quadrotors to take off by calling the `swarm.takeoff()` function. MagicFlock automatically registers logs and data sets of the flight mission executed by each quadrotor. A set of examples is provided on GitHub, where the framework is published.

In Chapter 8 we conclude this thesis by recalling the contribution presented in chapters 5, 6, 7 and proposing a set of perspectives that can facilitate the transition from academic sector to industry. Finally, a personal point of view considering quadrotors swarm and autonomous vehicles is discussed.

List of publications

Publications in international conferences

- **Omar Shrit**, Michèle Sebag. Learn how to swarm autonomous quadrotors using Iterative Imitation Supervised Learning. In EPIA Conference on Artificial Intelligence 2021, Springer, Cham.
- **Omar Shrit**, David Filliat, Michèle Sebag. Iterative Learning for Model Reactive Control: Application to autonomous multi-agent control. International Conference on Automation, Robotics, and Applications (ICARA 2021) Feb 2021, Prague, Czech Republic.

Publications published before the thesis

- **Omar Shrit**, Steven Martin, Khaldoun Al Agha, Guy Pujolle. A new approach to realize drone swarm using ad-hoc network. 2017 16th Annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net), Jun 2017, Budva, Montenegro. IEEE, 2017.

Collaboration not presented in this thesis

- Nicola Roberto Zema, Dominique Quadri, Steven Martin, **Omar Shrit**. Formation control of a mono-operated UAV fleet through ad-hoc communications: a Q-learning approach. 2019 16th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON).
- Nicola Roberto Zema, Mirwaisse Djanbaz, Dominique Quadri, Steven Martin, Enrico Natalizio, **Omar Shrit**. Contrôle de formation d'un réseau de drones à base d'apprentissage par renforcement. Rencontres Francophones sur la Conception de Protocoles, l'Évaluation de Performance et l'Expérimentation des Réseaux de Communication, Jun 2019, Narbonne, France

Software

- Omar Shrit (2018-2021). MagicFlock, A simulation framework for quadrotors swarm GitHub repository used in Chapters 5, 6, and 7 can be found here `https://github.com/shrit/MagicFlock`

Part I

Literature Review

Generalities about perception and control

2

This chapter provides a general state-of-the-art overview in the domain of perception and control. The perception part provides a short and general review of sensors used in the robotics domain. While the control section provides a review that is related to methods that are used or investigated in this thesis.

Contents

2.1 General perception in robots

In order to perceive the environment, robots are embedded with a set of onboard devices known as sensors (Everett, 1995). Most of the available sensors have characteristics such as field of view, accuracy, range, and power consumption. These characteristics vary from one sensor to another according to the quality of the design and the material used. Generally speaking, most robots types have a set of essential sensors that are embedded onboard; we review part of them in the following:

2.1.1 Motion sensors

Inertial Measurement Unit (IMU) (Ahmad et al., 2013; Hazry et al., 2009) are used to estimate the orientation and the angular rate of the robot. It usually consists of an accelerometer and a gyroscope. In some cases, a magnetometer can be included to measure the magnetic field of the earth. Inertial Measurement Unit (IMU) sensors, which usually require calibration (Skog and Händel, 2006) are essential when measuring the velocity and the position of the robot. In addition, a barometer can be added to measure the atmospheric pressure to deduce the altitude. Additionally, Global Navigation Satellite System (GNSS) devices might be included to improve positioning with the IMU.

2.1.2 Temperature sensors

They are a special type of resistances usually known as Thermistors. Their resistances change according to the current temperature. If the sensor type is Positive Temperature Coefficient (PTC) then the resistance increases if the temperature increases, otherwise the sensor type is Negative Temperature Coefficient (NTC) (Rai, 2007).

2.1.3 Vision sensors

RGB-D cameras (Jing et al., 2017; Litomisky, 2012) known also as 3D-depth cameras, they have usually one lens (monocular) and used to provide vision and depth information. These RGB-D cameras can capture images up to 30 frames per second, along with per-pixel distances to each object in the images.

Stereo cameras (Neukum and Jaumann, 2004) similar to RGB-D cameras, in this case, several cameras (usually two) are used simultaneously, either RGB or gray-scale, they are mounted with a specific distance from one to another. The objective is to simulate the exact mechanism used by the human brain to estimate the depth of the objects. Some stereo cameras can include additional sensors to improve accuracy, such as infrared, or others uses only one camera and a bi-prism (Lee et al., 1999; Lee and Kweon, 2000)

2.1.4 Range sensors

Light detection and ranging (LIDAR) (Adams, 2000) is a laser scanner that is used to determine distances to the surrounding objects by sending concentrated light beams toward it. Then, by calculating the time of flight of the laser beam, the sensor can estimate the distance to the object. A different method is known as coherent detection, uses low-speed frequencies. LIDAR sensors can have additional parts. For instance, some of them have mechanical motors to scan the environment in 360 degrees instead of having only one specific field of view; others have a phased optical array (Poulton, 2016) inside to reduce the costs.

Radio detection and ranging (RADAR) is very similar to LIDAR, however, it uses a high portion of the electro-magnetic spectrum instead of using a small portion of it (light) (Levanon, 1988). An interesting application of RADAR the synthetic-aperture which is used for 2D-3D imaging (Curlander and McDonough, 1991).

Sound Navigation and Ranging (SONAR) (Dimitrov and Minchev, 2016; Zhmud et al., 2018) is a sensor that use a detection technics similar to LIDAR and RADAR. The only difference in this case is the medium. SONAR uses sound or ultrasound wave to detect object.

Radio antennas When radio antennas receive signals from transmitters, they can measure the strength of the received signals. As signals propagate in the air, they lose a considerable amount of power. This phenomenon is known as path loss, and it is related to several environmental factors. Propagation models (Almers et al., 2007; Sarkar et al., 2003) allowed us to estimate the direct distance to the transmitters.

2.1.5 Positioning sensors

Global Navigation Satellite System (GNSS) is an outdoor positioning system that relies on satellites. GNSS sensors process the time of flight of messages received from 4 or more satellites using trilateration, allowing to provide Position, Velocity, and Time (PVT) of the receiver (Gebre-Egziabher and Gleason, 2009).

2.1.6 Discussion on sensor's choice

A key issue exists concerning the agent sensors. Indeed, powerful sensors (e.g. RGB-D cameras) are required for an agent to perceive its environment and to be autonomous. In this case, agents equipped with such sensors use mapping methods such as Simultaneous Localization

and Mapping (SLAM) (Kerl et al., 2013; Sturm et al., 2012) to reconstruct their environment, allowing them to have proper navigation and collision avoidance. However, powerful sensors increase the required computational and algorithmic resources in order to take advantage of the perceptual information in real-time (Kuzmin, 2018). Other sensors are cheap, easy to use, and do not require many computational resources, such as ranging sensors. However, they are usually less accurate, provide much less data, and their resolution decreases typically with long distances. This type of sensor substantially reduces the capacity of the robot to perceive the environment.

Another solution to this conundrum is to use sensor fusion techniques (Elmenreich, 2002; Kam et al., 1997; Sasiadek, 2002). These algorithms fuse raw data from several sensing devices to reduce uncertainty and improve accuracy allowing to take advantage of several devices with different characteristics. Sensor fusion is commonly used in autonomous vehicles to enhance lane detection (Jeong et al., 2019; Shin et al., 2018) by using cameras, GNSS and LIDAR. It relies on current and past sensor reading and uses probabilistic approaches such as Kalman filter (Sasiadek and Hartana, 2000), maximum likelihood (Perlovsky and McManus, 1991) or Bayesian theory (Murphy, 1998).

Finally, sensors provide only raw data that need to be accompanied by a set of algorithms to achieve navigation. As observed, the sensor choice is essential for building robots and providing a control algorithm.

2.2 General control in robots

2.2.1 Model Predictive Control (MPC)

Model Predictive Control (Garcia et al., 1989; Rawlings and Mayne, 2009) has been developed during the late 70s and 80s to provide simple and accurate feedback control for industrial systems (Morari and Lee, 1999). It is used to find the optimal control input of the next time steps according to the last output, constraints, and noise. The objective of the feedback loop is to minimize the disturbance added to the input. The prediction model used is specified explicitly, and in some cases, the disturbance model is defined too. The prediction model is capable of predicting the output (state of the system) for several defined time steps. Suppose we have a system model:

$$y(t+1) = Ay(t) + Bu(t) \tag{2.1}$$

Let the current state of the system to be y_0 at instant $t = 0$ and the current input u_0. The system model allows us to compute the next state y_1, when providing the next control input u_1 at time $t = 1$. Therefore, knowing the initial state x_0 of the system and providing the input $(u_0, u_1, ..., u_{n-1})$, we can calculate $(y_1, y_2, ..., y_n)$, knowing that the system is subject to two constraints y_{max} and u_{max}.

Subject to:

$$|y(t)| < y_{max} \tag{2.2}$$

$$|u(t)| < u_{max} \tag{2.3}$$

The objective of MPC is to find the optimal control input horizon $(u_0^*, u_1^*, ..., u_{n-1}^*)$ that leads to the best output of the system is based on the system model and two inputs: the current state y and the control input, u with no need for past information.

$$u_{1:T}^* = \arg\min_u \{\sum_{t=1}^{T}(y(t) - y_{ref}) + \sum_{t=0}^{T}(u(t) - u_{ref})\} \tag{2.4}$$

The MPC controller uses the system model to predict the possible future output in a finite time horizon, while the MPC optimizer tries to find the best input that satisfies the objective function. The optimizer tries to compute the best input horizon at each time step, which increases the computational cost. In addition, MPC is only used with linear systems, but there exist several methods to allow using MPC with non-linear systems.

The centralized approach consists of using MPC directly in a separated control unit that solves the trajectory online using an optimizer, MPC is known to require a considerable computation as the optimizer (including the operating constraints) needs to solve at each time step the trajectory, this approach scales poorly when the number of the agents increases as the required computations become significantly important. The solution for this is to use Decentralized Model Predictive Control (DMPC) (Bemporad and Barcelli, 2010; Dai et al., 2017), in which the optimization problem is divided into a set of feasible sub-problems concerning local constraints. In this case, each agent solves in real-time the assigned sub-problem.

The major drawback for these two schemes is the computational constraints of the agents themselves. Suppose that we have an actual number of agents in the team and each agent has low computational resources, as is the case in most real-world applications, none of the above methods can be used. Thus, a different approach should be considered. As agents interact in the environment, agents can collect information from these interactions and then apply trial and error

methods to classify this information. In other words, the agent can learn by itself the consensus algorithm through interaction with its neighbors. The main interest of the learning agent or is the capacity of the model to see or forecast stochastic and random situations.

2.2.2 Evolutionary algorithms

Evolutionary algorithms are a set of methods inspired from the Darwinian principle of Evolution. Charles Darwin has observed in nature (Darwin, 2016) what is known as natural selection. The idea is that each species has several offspring with different traits and genetic variations. Only the fittest offspring will survive when they interact with the environment and pass their traits to descendants. This method can be applied to agents such as robots. For instance, to create an autonomous robot, a population of random controllers can be created, each one embedded through different artificial chromosomes. Each controller will be evaluated using a fitness function to determine their performance at a specific task. For instance, Floreano, Mondada, et al., 1994 applied a genetic algorithm (Golberg, 1989) that evolves a neural controller to render the agent autonomous. Similarly, the same principle can be applied to a set of robots rather than one to achieve cooperative behavior (Mondada et al., 2004; Quinn et al., 2003) or even develop a communication system (Marocco and Nolfi, 2006; Quinn, 2001).

2.2.3 Learning algorithms

In the following sections, we introduce learning algorithms that have been used or explored during this thesis. The following algorithms are closely related to control theory algorithms such as dynamic programming, optimal control, and stochastic programming. However, the main difference is the full knowledge of the behavioral system model in optimal control, while learning algorithms operate on the data set recovered from the real world or a simulation environment.

2.2.3.1. Imitation learning

Consider having access to an oracle demonstrator that can execute the perfect trajectories in a specific environment (real-world / simulator). When executing these trajectories, the oracle generates a dataset $\mathcal{D}$ that can be used to train a policy π_θ that mimics the demonstrator. The dataset generated by the demonstrator π^* is represented as a set of state s, action $a = \pi^*(s)$ pairs $D = \{\tau := (s, \pi^*(s)\}$. During the training, the objective is to minimize a loss function $\mathcal{L}$

that reduce the error between the executed trajectory by the policy being trained, and the expert trajectory executed by the oracle as follows:

$$\arg\min_{\theta} \mathop{\mathbb{E}}_{s\sim P(s|\theta)} |\mathcal{L}(\pi^*(s), \pi_\theta(s)| \tag{2.5}$$

Unlike supervised learning, in which the general objective is to map the input to the output using labeled data, imitation learning aims to predict a sequence of actions according to the provided states. Making the method suitable perfectly to resolve sequential decision-making problems. The work of Stephan Ross (Ross, 2013) developed the framework on imitation learning.

2.2.3.2. Reinforcement Learning

Reinforcement Learning (RL) (Sutton and Barto, 1998; Szepesvári, 2010) is a general framework of machine learning in which a specific agent modeled as a set of states s learns to execute a specific action a by interacting with the environment that is represented as Markov Decision Process (MDP) framework. The agent aims to learn a specific policy to maximize a scalar discounted value named as a reward $r(s,a)$. The agent uses the policy to find the optimal actions to execute. Usually, the training configurations are divided into a set of episodes in which the agent execute actions in step-settings. The agent tries to maximize the reward in each episode as follows:

$$G^\pi = \sum_{t=0}^{T} \gamma^t R_{t+1} \tag{2.6}$$

Where the return G is the sum of the discounted reward R, the reward is discounted by $0 < \gamma < 1$ in order to valorize the current reward rather than the future one. However, one scalar value does not represent how well the agent is doing in each state when executing a specific action, thus, Reinforcement Learning defines a value function as follows:

$$v(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{T} \gamma^t R_{t+1} | s_t = s \right] s \in S \tag{2.7}$$

$$q(s,a) = \mathbb{E}_\pi \left[\sum_{t=0}^{T} \gamma^t R_{t+1} | s_t = s, a_t = a \right] s \in S \tag{2.8}$$

The objective of Reinforcement Learning is to find the optimal solution, since it maximize the return over the run. Therefore, a specific policy π is said to be optimal if it maximizes the value function better than any other policy, as follows:

$$v^*(s) \doteq \max_{\pi} v_\pi(s) \tag{2.9}$$

$$q^*(s,a) \doteq \max_{\pi} q_\pi(s,a) \tag{2.10}$$

Reinforcement Learning differs from imitation learning by 1) the absence of the oracle demonstrator, and 2) the existence of a reward function that need to be maximized. In reinforcement learning settings, the agent has to *explore* the entire environment to find the optimal policy, this adds inevitable collision with the environment since the RL involve trial and error. When the agent is a quadrotor, collision have considerable costs in real testbed or inside a simulation, especially if the recovering inside the simulator requires human intervention.

2.2.3.3. Inverse Reinforcement Learning (IRL)

Inverse Reinforcement Learning (IRL) (Abbeel and Ng, 2004) is a combination of Reinforcement Learning and imitation learning usually used when the design of the reward function is too complex. It is inspired by the learning methodology used by humans when they try to comprehend and master new technics. For instance, when a child tries to learn to play piano, the oracle *demonstrator* in this case is the teacher, will perform the new technics in front of the child. However, at this level, the child is only knowledgeable of the technics, while still mastering them requires hours of *trial and error* of training to execute the technics successfully. The advantage of IRL over RL is that the initial state starts with a demonstration of perfect policy, removing the need for the exploration step. IRL can be formalized as follows:

$$\pi^* = \arg\max_{\pi \in \omega} \mathbb{E}_\pi[r^*(s,a)] \tag{2.11}$$

The objective of this function is to learn the optimal reward function r^* that allows us to recover the optimal policy π^*. The process of learning the reward function is run over a set of states and actions that are executed by the optimal policy. The reward function can be a simple linear function (Syed and Schapire, 2008) or approximated via a neural network (Finn et al., 2016). Once the policy is learned through the maximization of the reward function, we calculate the loss

of the learned and expert policy. The objective is to iteratively improve the learned policy by finding the best reward function.

IRL have been used for robot navigation (Kretzschmar et al., 2016; Vasquez et al., 2014; Xia and El Kamel, 2016), grasping objects (Christopoulos and Schrater, 2010; Xie et al., 2019), trajectory tracking and control of UAV (Choi et al., 2017; Fu, 2016).

2.3 Discussion

In this chapter, we have introduced a general theoretical framework of methods and algorithms in perception and control. Of course, we can not cover the entire state of the art. Therefore, we have only presented methods that are related to this thesis. In the next chapter, we show more in detail how these control methods are used with a single or a set of robots, in particular in the case of Unmanned Aerial Vehicles (UAV)s.

Perception and control in quadrotors swarms

3

This chapter provides a literature overview in the domain of quadrotors swarm, this covers flocking models, centralized control algorithms, and learning algorithms.

Contents

3.1 Motivation

A swarm is a group of similar agents interacting to create an emergent behavior at the group level. The primary source to imitate this behavior is nature. The collective motion has been well observed in the animal kingdom; bacterial colonies tend to produce a motion pattern (Cisneros et al., 2010). While the collective motion is more developed in insects by creating a set of colonies

that contain millions of members. The colonies provide protection and survival information. Insects have also evolved a complex communication system (Wenseleers, 2010) allowing to create a swarm of millions of individuals (Gotwald Jr et al., 1995). Fishes tend to have two terms to describe the collective motion known as shoal and schools of fishes. In shoals (Pitcher, 1983), fishes are close to each other and swim within the social group but not in the same direction. However, in schools (Hemelrijk and Hildenbrandt, 2008), fishes are swimming in the same direction, relying on velocity matching to achieve this behavior (Katz et al., 2011). Birds are known to achieve flocking using a hierarchy of leaders (Nagy et al., 2010). In mammals, coordination of motion is similar to birds where a hierarchy of individuals is observed (Sarova et al., 2010). As a result, the highest grade individual has the most decisive influence on the herd motion.

Researchers have tried to fly several robots at the same time, creating a form of a swarm. They choose quadrotors over other types of MAVs due to their ability to realize vertical takeoff, as well as being stationary in the air. They are also highly agile and have a high thrust-to-weight ratio. Their sizes have decreased drastically, allowing for Vertical Take-Off and Landing (VTOL) in general and quadrotors, in particular, to be used in indoor navigation due to their ability to fly at low speed. These characteristics have favored the usage of these flying robots. In the decentralized setting, the controller operates at the agent level, making each agent fully autonomous.

3.2 Agent model: quadrotor

The quadrotor (Carrillo et al., 2013) is a type of Unmanned Aerial Vehicles, with six degrees of freedom allowing motion in three dimensions x, y, z. In aerospace convention, these axes are called the roll, pitch, and yaw, respectively. A quadrotor needs to rotate around these axes to move from one point to another. For example, pitching with an angle θ allows a quadrotor to move forward, or in other words, to create a trans-lateral movement on the pitch axes. This motion is represented physically by increasing the speed of the rear motors. The same concept is applied to other axes to generate another movement. To illustrate the basic functionality of a quadrotor, each one of these flying robots has four DC motors, usually Brushless Direct Current (BLDC), arranged in a cross configuration. Two pairs of propellers are mounted. One is clockwise, and the other is counter-clockwise. These motors are connected to a speed and flight controller situated at the center. The latter is an onboard computer with embedded sensors, such as gyroscopes, accelerometers, and barometers. While all the above are purely hardware parts necessary to construct a flying machine, it would be impossible to take off without an autopilot. Autopilots are software that decodes pilot commands into electrical signals executed

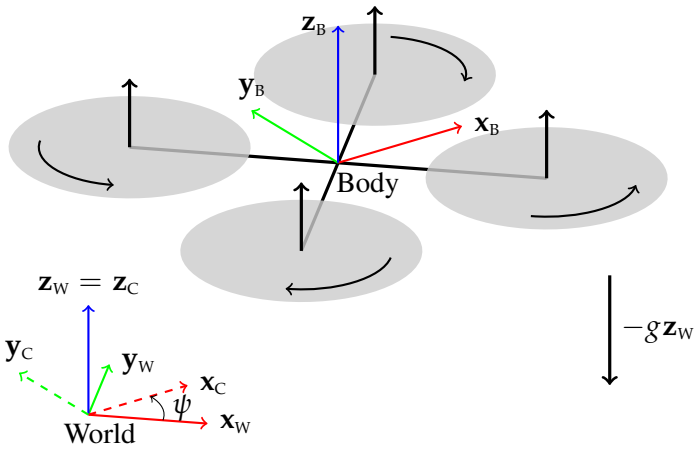


Figure 3.1.: shows a schematic draw for a quadrotor with four propellers, two propellers are mounted clockwise while the remaining two are mount counter-clockwise. The center body of the quadrotor usually has an integrated electronics part and flight controllers. the quadrotor takeoff and stabilize by applying positive force on the z-axis inversed to the gravity. The world frame is figured on the left part.

by airframe actuators. However, to successfully decode the flight command, autopilots use the Inertial Measurement Unit (IMU) sensors to estimate the attitude and effectively measure its roll and pitch. Also, most commercial autopilots today provide a wide range of functionalities such as camera control, fail-safe feature, and programming interface for camera vision. Communication with these autopilots is done using telemetry or WiFi. In addition to a commercial one, open-source autopilots developed by academics and research institutions are available. PX4 (Meier et al., 2015) developed by ETH Zurich is entirely a multi-threaded modular robotics framework; it is designed to work entirely on an embedded system using publish/subscribe design pattern and has support for Software In The Loop (SITL); it also supports a large number of peripherals and sensors. Other autopilots such as APM (*Ardupilot* 2018) or paprazzi (*Paparazzi Project* 2018) have restricted licenses, and their development has slowed down. For a complete list of autopilot software, please consult the list in the following survey (Lim et al., 2012)

The concept of a quadrotor can be traced back to 1907, when Breguet-Richet (Leishman, 2002) built Gyroplane No.1, the first quadrotor helicopter (YOUNG, 1982). A later improvement in the design led to the Gyroplane No.2 (YOUNG, 1982). Since then, the US army showed an interest in this domain and developed several quadrotors in the next years, such as Oemichen No.2 (*Archives Centrale-Histoire* 2019), Flying Octopus built by George de Bothezat (Bothezat, 1920), and the Curtiss-Wright VZ-7 (*FLIGHT* 2019) built by Curtiss-Wright corporation. These developments progressed during the twentieth century, especially in the military sector (Anderson, 1981), and continued in the twenty-one century to evolved from the military sector to the civilian

one. Quadrotors are inexpensive compared to quadruped and humanoid robots, and can realize a sophisticated achievement such as constructing buildings (Lindsey et al., 2011b), delivering packages [1], monitoring agriculture (Zhang and Kovacs, 2012) or executing search and rescuing missions (Valavanis and Vachtsevanos, 2014). In the case of small aerial robots, quadrotors are appreciated due to their ability to realize vertical takeoff, fly at low speed, or be stationary in the air.

3.3 Flocking model for dynamic system

The main challenge in multi-agent robotic systems is to design the individual controller or behavioral law such that the global behavior emerging from the set of individual behaviors achieves the target task. Several behaviors can be considered, such as cooperative or competitive behaviors between agents. In this thesis, we are interested in the collective motion of agents, known as the swarming behavior.

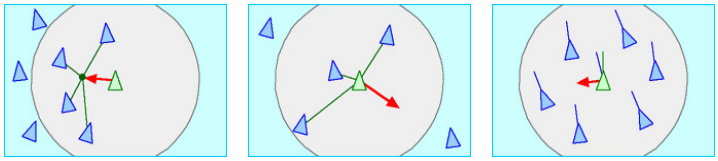
The observation of the animal kingdom allowed us to collect data using embedded sensors (e.g., Global Navigation Satellite System (GNSS)) on these animals. These observations have been analyzed, allowing the creation of several mathematical models that can be applied to robotic agents. Shimoyama et al., 1996 used a centralized algorithm, inspired by the biological organism, the aim of these models is to provide collective motion. Levine et al., 2000 proposed a flocking model that allows achieving the rotating movement in the same direction of a set of particles starting from the random initial condition. Mogilner and Edelstein-Keshet, 1995, 1996 proposed similar models to study alignment phenomena, as well as the pattern formation phenomena studied by Topaz and Bertozzi, 2004.

The main flocking models we review in detail are the major ones used in quadrotors swarm Reynolds model, Standard Viscek model, and Olfati-Saber model.

3.3.1 Reynolds flocking model

Reynolds, 1987, was the first to model the flock of birds in a computer simulation. He was the first to consider studying the simulation of complex natural phenomenons. His flocking model, known as the Reynolds model, consists of three steering elements: cohesion, alignment, and separation.

[1] https://www.x.company/projects/wing/



(a) Cohesion (b) Separation (c) Alignment

Figure 3.2.: Flocking rules knowing as steering behaviors applied to a simple Boids as represented by Reynolds. The rules knowing as cohesion, separation, and alignment are applied to each boid individually. The boid has full knowledge of the velocity, position, and heading of its neighbors in a certain range in order to compute its velocity and heading.

To illustrate successful computer simulations, Reynolds used the same particle system used in computer graphics to represent his boids (*Boids* 2021) (bird object). Each particle has several indispensable elements (velocity, position, shape, size, etc.). In addition to the above elements, Reynolds added several configurations to his boids such as local coordinate system and polygonal figure. The models is given as follows:

$$\ddot{x}_i = \frac{k_a}{|\mathcal{N}_i|}\sum_{j\in N_i}(\dot{x}_j - \dot{x}_i) + \frac{k_c}{|\mathcal{N}_i|}\sum_{j\in N_i}(x_j - x_i) - \frac{k_s}{|\mathcal{N}_i|}\sum_{j\in N_i}\frac{x_j - x_i}{\|x_j - x_i\|^2} \tag{3.1}$$

$\ddot{x}_i$, $\dot{x}_i$, x_i are respectively the acceleration, velocity and the position vector of the quadrotor i. k_a, k_c, k_s are the gains multiplied by their respective term alignment, cohesion and separation respectively. While $calN_i$ is the neighborhood for the robot i.

3.3.2 Viscek flocking model

Vicsek et al., 1995 studied the motion of agents in the 2D environment which is different from the Reynolds model, by taking into consideration, the orientation of the agent and tries to achieve alignment instead of flocking. It starts at time $t = 0$ by distributing agents randomly in the environment, each agent starts with random orientation and moves at constant velocity v. Note that, each agent interacts with its neighbors within a radius r. The model is as follows:

$$x_i(t+1) = x_i(t) + v_i(t)\Delta t \tag{3.2}$$

$$\theta_i(t+1) = \frac{1}{card(N_i)}\sum_{j\in N_i}\theta_i(t) + \Delta\theta_i(t) \tag{3.3}$$

The orientation of the agent i is calculated from the mean directions and velocities of agents in the neighborhood N_i. They also define $\Delta\theta_i(t)$ as a noise which is a random number chosen from a uniform distribution within the interval $[-\eta, +\eta]$.

3.3.3 Olfati-Saber flocking model

Olfati-Saber, 2006 proposes a theoretical flocking model that is divided into three parts (or algorithms). The first algorithm is similar to Reynolds algorithm shows frequent fragmentation in the flock, while the second algorithm fixes this fragmentation. The third one is used to avoid obstacles. This is done in order to make flocking scalable according to the environment constraints. In this section, we present only the third algorithm presented in equations 3.4, 3.5, and 3.6 since it implies the first and the second one. The olfati-Saber flocking models is giving as follows:

$$\ddot{x}_i^{\alpha} = c_1^{\alpha} \sum_{k \in \mathcal{N}_i^{\alpha}} \phi_{\alpha}\left(\frac{1}{\epsilon}\left(\sqrt{1 + \left\|x_j - x_i\right\|^2} - 1\right)\right) \frac{x_j - x_i}{\sqrt{1 + \epsilon \left\|x_j - x_i\right\|^2}} + \sum_{j \in \mathcal{N}_i^{\alpha}} a_{ij}(q)(\dot{x}_j - \dot{x}_i) \quad (3.4)$$

$$\ddot{x}_i^{\beta} = c_2^{\beta} \sum_{k \in \mathcal{N}_i^{\beta}} \phi_{\beta}\left(\frac{1}{\epsilon}\left(\sqrt{1 + \left\|x_j - x_i\right\|^2} - 1\right)\right) \frac{\hat{x}_{j,k} - x_i}{\sqrt{1 + \epsilon \left\|\hat{x}_{j,k} - x_i\right\|^2}} + \sum_{j \in \mathcal{N}_i^{\beta}} a_{ij}(q)(\hat{\dot{x}}_j - \dot{x}_i) \quad (3.5)$$

$$\ddot{x}_i^{\gamma} = c_3^{\gamma} \frac{x_i - x_r}{\sqrt{1 + \left\|x_i - x_r\right\|^2}} - c_2^{\gamma}(\dot{x}_i - \dot{x}_r) \quad (3.6)$$

$$\ddot{x}_i = \ddot{x}_i^{\alpha} + \ddot{x}_i^{\beta} + \ddot{x}_i^{\gamma} \quad (3.7)$$

Where $\ddot{x}_i^{\alpha}$ is the acceleration vector applied to the quadrotor i from the interaction with its neighbors. While $\ddot{x}_i^{\beta}$ is the acceleration applied to quadrotor i from the interaction between the quadrotor and the static/dynamic objects. The last term is the navigational feedback. The sum of the tree terms constitute the acceleration applied to the quadrotor i in order to create the flocking behavior. c_n are constants, as well as ϵ. Note that $a_{ij}(q)$ is the spatial adjacency matrix.

$$a_{ij}(q) = \rho_h\left(\frac{1}{\epsilon}\left(\sqrt{1 + \left\|x_j - x_i\right\|^2} - 1\right)/r_{\alpha}\right) \quad (3.8)$$

$\rho_h(z)$ is bump function. The objective of this function is to create a smooth potential function since it varies smoothly between 0 and 1.

$$\rho_h(z) = \begin{cases} 1, & z \in [0, h) \\ 0.5(1 + \cos(\pi \frac{z-h}{1-h})), & z \in [h, 1] \\ 0 & otherwise. \end{cases} \tag{3.9}$$

The last part is $\phi_\alpha(z) = \rho_h(z/r_\alpha \phi(z - d_\alpha))$ the action function for agents. While $\phi_\beta(z) = \rho_h(z/d_\beta)(z - d_\beta/\sqrt{1 + z^2})$ the repulsive action functions from the obstacle.

The advantages of the Olfati-Saber model are related to the convergence of the robots due to the use of the smooth collective potential functions. The algorithm presented above shows that one collective behavior with no fragmentation is possible for the swarm even when several obstacles are included.

3.3.4 Viragh flocking model

Virágh et al., 2013 presented a realistic flocking model. The model has similar rules to those proposed by Reynolds and Vicsek. However, it takes into account several features in order to enhance the behavioral law for quadrotors. For instance, the model depends on several parameters such as inertia, inner noise of sensors, time delay, and communication constraints. In addition to the basic rules, it proposes a migration term in which quadrotors use it to migrate smoothly. The authors have embedded the model into a set of quadrotors (Vásárhelyi et al., 2014). The test realized in an outdoor environment required GNSS sensors to localize neighbors and the position of the destination.

3.4 Centralized controller for aerial robots

3.4.1 Testbeds

Over the years, a large set of test-beds have been established (Hoffmann et al., 2011; Lupashin et al., 2014; Michael et al., 2010; Preiss et al., 2017; Schmittle et al., 2018; Stirling et al., 2012; Stirling et al., 2010). These test-beds allowed testing algorithms in an indoor/outdoor

environments. Using different types of Unmanned Aerial Vehicles (quadrotors, helicopters, fixed-wing). Most of these tests have required an expensive precise positioning systems such as indoor capture motion system (*Vicon* 2019), which consists of several cameras tracking the quadrotors in real-time, or a grid of interconnected infrared sensors (Roberts et al., 2012; Roberts et al., 2009), in addition, a separated control unit that generates the trajectory for each of the quadrotors in terms of their relative positions. Control commands are sent to each of the quadrotors using ZigBee wireless allowing quadrotors to be uncoupled, which mean each robot has limited or even non-existent knowledge about the environment.

3.4.2 Centralized control algorithms

For formation control problems (e.g., transition from a specific formation shape to another, circular swarm), using a centralized controller to generate real-time trajectories seems to be the dominant solution. Kushleyev et al., 2013 proposed such a controller for a set of quadrotors in an indoor environment. By using a set of predefined waypoints, the controller generates trajectories for a set of quadrotors by resolving a real-time optimization program. To reduce the complexity of the system, the same generated trajectory is sent to each quadrotor with a time interval allowing each robot to follow one another without a collision. Their method is based on a Mixed-Integer Linear Programming (MILP) for trajectory generation proposed by Schouwenaars et al., 2006; Schouwenaars et al., 2001. A centralized algorithm such as Concurrent Assignment and Planning of Trajectories (CAPT) proposed by Turpin et al., 2014 considers the problem of path planning for a set of quadrotors. The algorithm guarantees quadrotors to reach different destinations without collision with one another. For instance, it has been applied to a set of quadrotors to achieve formation control in indoor environment using a set of quadrotors with smartphones that are used to perceive the environment (Loianno et al., 2016), they improved their work to rely on Visual Inertial Odometry (VIO) process allowing to achieve formation control in indoor/outdoor environment (Weinstein et al., 2018). Each quadrotor uses an optical flow camera (Honegger et al., 2013) installed beneath the quadrotor to track the markers placed on the ground (Landmarks) (Wagner and Schmalstieg, 2007), allowing a quadrotor to position itself uniquely in the space. The size of the marker indicates the proper height for the quadrotor and the geometrical shape indicates its position relative to its neighbors. Du et al., 2019 used a similar optimization method to generate trajectories for quadrotors, in addition to trajectories, they propose a motion synchronization method and provide a generic framework to design swarm choreographic patterns (e.g., wave patterns, rotation patterns) that provides a smooth and safe transition between patterns. They validated their results by creating a choreographic dance of 25 quadrotors in an indoor environment.

3.5 Swarm controller using Model Predictive Control (MPC)

Researchers used MPC (Alexis et al., 2012) to control quadrotors or to resolve even more complex problems such as formation control in quadrotors. For example, Augugliaro et al., 2012 used MPC to generate a trajectory for a fleet of quadrotors. While Kamel et al., 2017 used non-linear MPC to achieve robust collision avoidance. Similarly, Saska et al., 2014b used MPC to control a fleet of quadrotors enabling obstacle avoidance and motion planning until the destination using vision sensors. They demonstrated (Saska et al., 2016a) that MPC can be used in a leader-follower scheme in order to follow a virtual leader that is executing a complex trajectory. In addition, they extended their research in order to support a fleet of a heterogeneous swarm of quadrotors and Unmanned Ground Vehicles (Saska et al., 2013; Saska et al., 2014c; Spurny et al., 2016). MPC is used in order to keep the formation shape and to plan the trajectory for followers of the virtual leader.

In the case of quadrotor fleets, the state of the system is well-known and shared with other quadrotors. The issue with MPC or centralized MPC is the computation cost required to resolve the state estimation, especially when the environment is complex and containing several agents. To resolve the computational issues, the authors (Cheng and Savkin, 2011; Dai et al., 2017; Richards and How, 2004; Shanbi et al., 2012) have worked on a Decentralized Model Predictive Control (DMPC), the method is intended to be used on each quadrotor individually; in such a case, quadrotors need to have a sufficient onboard computational capacity. For example, Baca et al., 2016 proposed an embedded system that is capable of executing quadratic MPC onboard with constraints.

3.6 Swarm controller using Evolutionary Algorithms

Vásárhelyi et al., 2018 used CMA-ES evolutionary algorithm to train a set of order parameters and to increase the rate of their convergence algorithm. They build on their previous result in (Vásárhelyi et al., 2014) using the Viscek flocking model. Their work addresses several important issues in quadrotors swarms, such as model validation and reality gap.

Saska et al., 2014a; Saska et al., 2016b used Particle Swarm Optimization (PSO) method in order to tackle the problem of cooperative surveillance with a set of quadrotors. The PSO algorithm generates a trajectory for each quadrotor in order to cover a predefined area of interest with its neighbors. They validated their results in simulation and an outdoor scenario with a set of quadrotors.

3.7 Single and swarm controller using learning algorithms

3.7.1 Imitation learning

Abbeel et al., 2006 applied imitation learning technique with reinforcement learning to learn how to fly a helicopter. Ross et al., 2012 applied their framework on one quadrotor that is trained using only one monocular camera; the quadrotors learn how to avoid obstacles in a forest environment. The objective is to make autonomous quadrotors that are capable of learning to fly as human experts. The authors validate their results by doing a real flight test for over 3.4 km. DroNet (Loquercio et al., 2018) is another platform that uses a convolutional neural network to analyze images captured by a quadrotor following a human-driven car. They used the public (*Udacity data set* 2019) and the images captured by the quadrotor.

To our best knowledge, the only known vision-based swarm using imitation learning is detailed in (Schilling et al., 2018) in which authors use the Reynolds flocking model as an oracle demonstrator to generate data set. Then they predict the velocity command based only on visual information to match the command law from the flocking model as possible. Their work is to eliminate the need for position knowledge of neighbors. Note that they mounted six cameras on each quadrotor to provide an omnidirectional vision. The method proposed by the authors has been extended and improved to be tested on real quadrotors swarm (Schilling et al., 2019). The authors realized a circle experiment in which the follower tries to keep with the leader to validate the cohesion rules, and a pull-push experiment to validate the separation rules between the leader and the follower.

3.7.2 Reinforcement Learning

To our knowledge, reinforcement learning has not been used in the quadrotors swarm domain but has been applied successfully on a single agent robot. The reason for this is related to the increased complexity to interact with the environment when other agents *constitute* part of the same environment rather than *sharing* the environment. This turns the design of the reward function into a complex dilemma since it has to consider all neighbors in the group. In addition, a reward function needs to consider the convergence factors of the swarms, such as the velocity matching, alignment, and collision-separation level between agents. Huttenrauch et al., 2019, 2017 describes the only framework that addressed the agent swarming challenge using reinforcement learning. In their first work, they address the formation control and target localization problem using a similar algorithm to Deep deterministic Policy Gradient (DDPG) algorithm. They extend their work to address the issue of cohesion between agents, where they

change the method of state representation to remove the dependency on the number of the agent as the number changes over time. To address this issue, they use the mean feature embeddings to represent the state vector. Long et al., 2018 address the challenge of collision avoidance for a set of 2d robots equipped with a laser range finder sensor. They use a centralized multi-stage training process with decentralized execution, the first stage contains only a small set of robots with no obstacles while the second stage adds a richer environment of obstacles and robots. They use Proximal Policy Optimization (PPO) algorithm to train a policy on the observations recovered from all robots. They validate their results in simulation and on a test-bed of ground robots.

Reinforcement learning has been used on a single agent quadrotor to learn a single policy, either a low-level policy to learn how to fly from scratch to eliminate the need for a manually designed and tuned Proportional, Integral, Derivative (PID) autopilot controller, or to achieve a high-level autonomous flight.

The use of Reinforcement Learning on low-level flight policies is well established in the literature. The first research dates to 2001 when Bagnell and Schneider, 2001 used RL to find optimal control policy to learn how to fly a helicopter, with more research add to the subject by Kim et al., 2004 that used the Pegasus (Ng and Jordan, 2013) Reinforcement Learning algorithm to achieve high stability helicopter controller better than a human pilot followed by aerobatic helicopter presented by Abbeel et al., 2006 using RL and imitation learning. Similar research proposed by Hwangbo et al., 2017 also uses reinforcement learning algorithms to create a flight controller for one quadrotor, they demonstrated that the quadrotors stabilize even under a very harsh initialization environment.

(Koch et al., 2018) used reinforcement learning algorithms to allow for a quadrotor to learn how to fly. The authors created a GYMFC simulation environment that replaced PID autopilot by the PPO algorithm. The authors demonstrated that the RL controller outperforms the PID controller in every metric in a different environment. The trained controller called Neuroflight (Koch et al., 2019) is tested on real quadrotor hardware, allowing to achieve stable flights.

Considering autonomous flight, Sadeghi and Levine, 2016 presented CAD^2RL a deep reinforcement learning method used in a simulated environment to train vision-based navigation policy for a flying robot with a monocular camera; the policy is extended to the real world to avoid obstacles and realize a collision-free flight. The authors solve the reality gap issue by preparing the simulation environment. First, they create synthetic data that contains various hallway geometries and textures that describe the environment. Second, they highly randomize the texture inside the domain during the simulation. The states are the input images. Each image is divided into a grid of bins, and each bin is an action. The robots execute an action by moving

towards the bin. The result is either collision or noncollision. In the end, all actions are evaluated, which generates the collision probability function for each state. A pre-trained convolutional neural network is used to predict the best action in each state for indoor flight.

Gandhi et al., 2017 went even further to crash the quadrotor in order to generate a data set that can be used to create a robust navigation policy. The policy is used only on single quadrotors.

3.8 Discussion

This chapter reviews state-of-the-art control methods and algorithms used to realize a swarm of quadrotors. The above methods do not directly answer the problem proposed in this thesis. Therefore, the following table establishes a summary of methods mentioned earlier and provides a comparison in terms of perception and control and whether if they are decentralized or not.

Authors	Control algorithm	Sensor Dependency	Decentralized	Advantages	Disadvantages
Michael et al., 2010 Kushleyev et al., 2013	Trajectory generation using MIQP	Vicon	No	Precise formation transition & swarms	Arena dependent
Augugliaro et al., 2012	Trajectory planning using SCP	Vicon	No	Fast & Easy formation transition	Arena dependent
Virágh et al., 2013	Viscek model	GNSS	Yes	Easy to deploy	Only outdoor
Saska et al., 2014d	Reynolds model & MPC	Vision sensor	Yes	Easy to deploy	Neither SITL nor real test
Turpin et al., 2014	C-CAPT/ D-CAPT	Vicon	No/Yes	Collision free trajectories	Arena dependent
Loianno et al., 2016	CAPT	Onboard SoC & cameras	Yes	Autonomous robots, work indoor/outdoor	Require a control center, computation resources.
Saska et al., 2016a, 2014b	MPC & virtual leader	Vision & RFID sensor	No	Work both indoor and outdoor	Require computational resources
Baca et al., 2016	Embedded MPC	Onboard sensors	Yes	Work in indoor/outdoor	Require computational resources
Kamel et al., 2017	Non-linear MPC & OCP	Onboard SoC & Vicon	Yes	A unified framework for robust collision avoidance	Arena dependent
Weinstein et al., 2018	CAPT	Monocular camera	No	Scalable swarm	Require control center
Vásárhelyi et al., 2018	Evolutionary algorithm & Viscek model	GNSS	Yes	Large swarm, high velocities,	Require inter-agent communication, Only outdoor
Schilling et al., 2018, 2019	Reynolds model & imitation learning	6 cameras	Yes	Require light computation	Simulation dependent
Du et al., 2019	Trajectory planner & optimizer	Vicon	No	Complex swarm patterns and fast transition	Arena dependent

Table 3.1.: Comparison between algorithms and methods that propose a quadrotors swarms

Preliminary results: manually designed controller using the distances between the quadrotors

4

Contents

4.1 Motivation

A swarm of Commercial of-the-shelf (COTS) quadrotors provides users with a wide set of applications that can not be executed using a single robot. For instance, several quadrotors can collaborate to maximize observation in a specific area, or realize a search and rescue missions (Bernard et al., 2011). However, the complexity of a swarm system is increased when compared to a single robot in terms of perception and control. From the perception side, several sophisticated sensors are required, but the cost of these sensors is a key issue for a quadrotor swarm in both terms of energy consumption (e.g., to carry heavy cameras or GNSS) and algorithmic complexity (to exploit fine-grained information). From the control side, most existing solutions allow orchestrating and plan trajectories for several quadrotors using a control center (Turpin et al., 2014), these solutions reduce the range of applications that can be executed by the robot

and render the mission complex; a preferred solution is to make each robot fully autonomous to handle its trajectory.

To avoid complexities related to perception and control, a different solution can be considered if the number of robots is too small, which is embedding a predefined trajectory on each quadrotor as set points. This solution does not take into account collision among quadrotors or with static or dynamic objects. Another basic solution is to consider using the wireless channel (communication) provided by the radio sensors. This method might work when the number of quadrotors is small. However, when the number of agents increases, the communication channel tends to saturate, with a high loss of the packets sent between the transmitter and receiver. A possible solution would be re-transmission, but this might increase the communication delay. In both cases, the power consumption increases due to the frequent treatment of the received packets.

The key idea behind this study is to reduce the cost required by quadrotor swarm systems by using a COTS quadrotors with their onboard sensors in order to make swarms affordable to ordinary users. However, inexpensive robots usually have limited sensing capabilities and have a small frame. The capacity of these sensors to perceive the environment is minimal. One might think about using the vision sensor usually represented as the front camera. However, the computations required to analyze images are beyond the capacity of such a robot, therefore eliminating the robot from being a potential candidate for swarming.

Our contributions are the following; first, we propose a decentralized controller that can be embedded on inexpensive COTS quadrotors. Second, the controller is designed manually and converts the perceived sensor information to a velocity vector. The controller is capable of filtering the sensor noise. The main advantage of this controller is its *simplicity* as it is based on cheap, lightweight, low consumption sensors (both in energy and in computation), making it suitable for a wide set of quadrotors.

4.2 Overview of the decentralized controller

4.2.1 Background related to WiFi sensor

Wireless signals are a specific case of radio waves that travel the environment at the speed of the light with the capacity to pass throughout the environment, even the vacuum. During the propagation through these environments, radio waves experience several phenomena such as diffraction, refraction, reflection, or absorption. These phenomena are applied differently to radio

waves according to the length of the wave. Therefore, choosing the wavelength or the frequency is a crucial part when it comes to radio communication systems. In such a system, electrical signals are modulated and transported using the radio waves from the transmitter to the receiver. Radio communications have been existing since the end of the 19th century. The set of radio frequencies used in the communication domain ranges from 3 kHz to 300 GHz. These frequencies are considered scarce resources, and governments worldwide manage access to these resources by licensing these frequencies to a specific operator for a specific amount of fees. However, the International Telecommunication Union (ITU) has provided a portion of this spectrum for unlicensed usage initially known as Industrial, Scientific, and Medical (ISM) band. These bands are used for various applications such as audio, video devices, baby monitors, microwave ovens, toys, radars, and cordless phones.

For radio signals to be sensed by a specific antenna receiver, the attenuation level that the signal suffered from should be at a specific level. The attenuation knowing as the reduction of the power density of the signals, depends on several factors such as 1) frequency, 2) the number of obstacles and 3) the path in which the signals travel. For example, radio signals do not suffer from attenuation in outer space since they propagate in a vacuum. However, on earth, the above factors directly affect the signals. Therefore, to study the propagation of radio signals, several empirical models have been formulated based on collected data from observation of the level of signals in several conditions. These models aim at predicting the attenuation of signals or known as path loss. The International Telecommunication Union regularly updates the propagation models. The last model published by ITU (Union, 2019) used to estimate the signal power at the receiver is given as follows:

$$L(d,f) = 10\alpha \log_{10}(d) + \beta + 10\gamma \log_{10}(f) + N(0,\sigma) \tag{4.1}$$

While L is the transmission loss given in decibel (dB), d is the 3D direct distance from the transmitter to the receiver measured in meters, and f is the operating frequency measure in GHz. α, γ are coefficients that are related respectively to the distance and frequency. β is the offset value, and N is the zero-mean Gaussian random variable with σ as the standard deviation given in decibel as well. This model is valid for frequencies that range between 0.8 and 73 GHz; the values of coefficients depend on the site configurations, whether the receiver is on a direct path with the transmitter known as Line-of-sight (LOS) or non-direct path is known as Non-line-of-sight (NLOS). In this work, we do not assume any obstacles between the quadrotors. Thus, the configuration should always be considered as LOS. The values of coefficients are given as follows: $\alpha = 2.12$ $\beta = 29.2$ $\gamma = 2.11$ $\sigma = 5.06$

The above equation only gives the transmission loss, in order to calculate the signal value at the receiver, this adds on the above formula:

$$S_r = S_t + G_t + G_r - L(d, f) \tag{4.2}$$

In which S_r is the signal power at the perceived by the receiver known as the received signal strength (RSS) given in decibel-milliwatt (dBm), S_t is the transmitter power, G_t, G_r are the gain of the transmitter and receiver antenna respectively. The signal power is measured in dBm in which 0 dBm is equal to 1 milliwatt (mW), conversion from mW to dBm is given by this formula:

$$S = 10 \log_{10}(P) \tag{4.3}$$

S is power in dBm and P is power in mW.

Most wireless home appliances that use ISM bands with 2.4 GHz can measure the Received Signal Strength Indicator even COTS devices that implement communication protocols such as WiFi, Bluetooth, ZigBee can analyze received signals from several transmitters. Due to these capacities, these technologies have been used beyond the traditional usage of communication and data transmission. Several researchers have used WiFi in order to build an indoor localization systems by using methods such as triangulation techniques (Bahl and Padmanabhan, 2000; Chintalapudi et al., 2010; Ferris et al., 2007; Lim et al., 2006; Wu et al., 2012) or fingerprinting techniques (Nandakumar et al., 2012; Sen et al., 2012; Youssef and Agrawala, 2005)

Our objective is not to conceive a localization system for quadrotors, but instead is to use the variation of signal strength as a method to detect the change of distance between the transmitter and the receiver and to build upon that a simple controller to make follower quadrotor fully autonomous.

4.2.2 Proposed controller

Since WiFi rely on 2.4 GHz frequency, it is usually the basic communication chipset used in most commercial quadrotors. In this chapter, since we do not use the communication channel to send any information on it, we refer to the WiFi chipset as a *Wireless sensor* rather than a communication module.

Let **s** be the state vector of the quadrotor that contains the received signal strength from neighbors quadrotors and the leader quadrotor. Let $v = (x, y, z)$ be the velocity vector for the robot defined on the tree axis.

Definition 1 *Let $\mathcal{N}$ be the set of quadrotors. Each quadrotor has a priority order in which determines the subsequent follower quadrotor. Consider the that quadrotors are ordered $(0, 1, 2, ..., n)$. Each quadrotor is assigned a number as a label that defines its order. The leader quadrotor has the highest priority $i = 0$ followed by the follower $j = 1$ as the second priority. Therefore, let the quadrotors $k \in \mathcal{N}_j$ be in the set of neighbors of the quadrotor j. k is said to have a lower priority than its neighbor j if and only if the value of $j < k$.*

Assumption 1 *The action space is limited to only x-axis, making the quadrotor capable of pitching forward or backward $v_x = +/-c$. This limitation is imposed only to demonstrate a proof of concept of the proposed solution.*

Assumption 2 *Each quadrotor embeds only one WiFi sensor with no information relative to the controller is shared on the communication channel.*

Under these definitions and assumptions, we define our controller as a 4 steps process. In each step, depending on the observed signal, the controller behaves as follows:

1. Sequences of state are recorded as a set of signal strength $\eta = (s_t, t = 1...T)$, each follower quadrotor monitor specifically the signal strength received from the quadrotor with a higher priority and apply the extended moving average filter on s as follows:

$$f(x) = \begin{cases} x_0 & : t = 0 \\ \alpha x_t + (1-\alpha) x_{t-1}^f & : t > 0 \end{cases} \tag{4.4}$$

Where $f(x)$ is a first-order infinite impulse filter, $0 < \alpha < 1$ is the smoothing coefficient, x_t is the value to filter, and x_{t-1}^f is the previously filtered value.

2. The received signal strength from the quadrotor are compared to a specific threshold, once the threshold is crossed, the quadrotor takes off

$$s < T_{takeoff} \Longrightarrow v_z = c; \tag{4.5}$$

Where $T_{takeoff}$ is the takeoff threshold, and c is the takeoff speed that is constant during the takeoff. The takeoff threshold is already defined by a far distance between the leader and the pilot, allowing the follower quadrotor to takeoff.

3. Once the quadrotor has taken off, the velocity vector is reset $v = (0,0,0)$ and the controller is defined as:

$$v_x = \begin{cases} 0 & : \gamma > -(f(s_j) - s_0) \\ -(f(s_j) - s_0) & : \gamma < -(f(s_j) - s_0) \end{cases} \tag{4.6}$$

Where $f(x)$ is the filter defined in the first step, $s_0 \in \mathbb{R}^-$ is a scalar that defines the satisfying property of the system that allows a decent degree of cohesion and separation between the quadrotor without any possible collision. γ is the offset reactivity coefficient.

4. This step is only executed at the end of the flight when the leader quadrotor starts returning to the pilot, the quadrotor with the lowest priority

$$s > T_{land} \Longrightarrow v_z = -c; \tag{4.7}$$

Compared to state-of-the-art methods, the advantages of the presented controller can be seen as follows: First, the controller is simple and requires only received signal strength value, which allows easy deployment. Second, the controller requires a low level of computation. The objective of this controller is to completely automate the control process of the followers' quadrotors, making them entirely autonomous under the above assumptions. Once this controller is embedded on followers, each quadrotor will take off automatically and position itself between the leader and the pilot. By applying the equation 4.6 the follower keeps an appropriate distance between any pair of quadrotors. The controller provides the velocity vector on the x-axis making the robots only pitching forward or backward. Finally, the last step allows landing the quadrotors close to the operator with the lowest priority.

Part of this controller is embedded within the operator module itself. The objective is to decentralize the tasks completely. For example, the operator module will automatically manage followers' takes off and landings. More precisely, this module will also monitor the received signal strength from the leader and the last follower, and according to the equations 4.5 and 4.7 it will decide what action to trigger next (takeoff, landing). While the other part of the controller is embedded in each one of the followers. Once the robot has realized a takeoff, it will monitor periodically the received signal strength from its neighbor with the higher priority. The robot executes the equation 4.6 to know in which direction it should move (forward, backward) or even hover at the same place if the neighbor robot is not moving.

This controller can be considered as a modified version of the Reynolds flocking models since:

- This method lifts the need for external positioning systems such as GNSS or a relative one based on heavy vision sensors.
- This method adds a leader-follower principle since it does not constitute a part of the Reynolds flocking models.
- It relies on a simple sensor and shows an extreme and reactive controller on the testbed. As shown with flocking models, the quadrotors are autonomous and do not collide with one another.

Figure 4.1 illustrates how the controller works, First, the operator with remote control (on the left) commands the leader (on the right). During the flight, the distance between the operator and the leader increases, allowing for the first follower (the quadrotor with the highest priority after the leader) to take off automatically; another follower will take off as the leader goes further is followed automatically by the leader. Note that distances between quadrotors are not

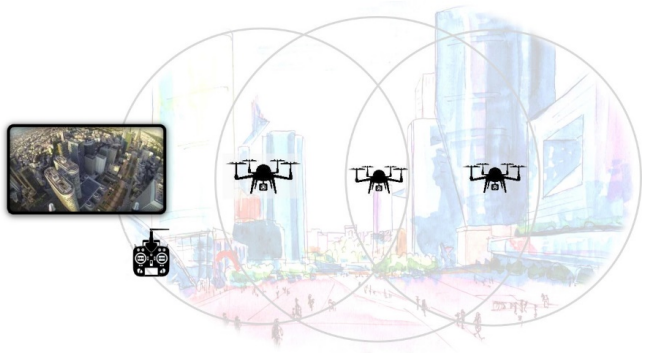


Figure 4.1.: A proposed scenario with a basic line formation

predetermined. They are defined from the received signal strength in dBm (circles in the figure 4.1).

4.2.3 Experimental settings

4.3 Test bed platform

The robot used in the experiment is a bebop2 quadrotor designed by Parrot (*PARROT* 2021). The Bebop2 has a relatively small size (28x32 cm) and weight (500 g), with an onboard ARM processor and flight time up to 25 minutes. In addition, the quadrotor has a fail-safe technique such as instant motor stops if the propellers hit an object allowing to reduce damages to external objects, humans, and the robot itself. In addition, Bebop2 can stabilize in the air, even in an indoor environment, thanks to a small optical flow camera. Finally, the quadrotor can be human-piloted through an external WiFi controller. The quadrotor software offers an Application Programming Interface (API) to take control through external/internal software modules. The WiFi sensor that is embedded in the bebop has limited power to 20 dBm (100 mW) giving about no more than several tens of meters for the follower quadrotors to sense their neighbors. The sensor provides two antennas attached to the body frame. Considering sensing limitations, one might suggest using robust WiFi sensors with a set of high gain directive antennas to reach further. However, this modification is out of the scope of our work.

The operator is any System on Chip (SoC) system that can communicate and send a control command to the leader (e.g., a laptop, a Raspberry Pi, or a Joystick). In this scenario, the operator module was a laptop, chosen for simplicity and integrating external software.

Figure 4.2.: shows the Bebop 2 quadrotor fabricated by Parrot that is used in the real world test experiment, the quadrotor have an API allowing to write a software to control the quadrotor from external device. In addition, the quadrotor has embedded internal hidden ports allowing to add additional software on-board (inside) the quadrotor.

4.4 Software module

The implementation of the controller is required to be embedded only on the follower quadrotors. The controller software is implemented in C++ programming language using the Boost library (*Boost C++ Libraries* 2021) and Bebop2 Parrot external control API. We divided the controller into two software modules. The first one is a cross-compiled onboard module that is embedded inside the Bebop2 quadrotor. The onboard module communicates with the autopilot software to provide control commands and autonomy based on the received signal strength periodically from the neighbors.

The second module is off-board and provides an extension to the operator control center functionalities by adding high-level commands such as automatic takeoffs and landings for followers. In addition, this module monitors in real-time the RSS from the follower with the lowest priority to determine if a new one has to be launched or not. Concerning security, the module also has a fail-safe feature that offers the possibility to provide real-time manual control of any follower in the formation in the case of drifting or system failure. Finally, quadrotors also benefit from feature provided generic features allowing any possible extension for a considerable set of applications such as video capture and information relaying from the leader to the operator.

4.5 Experimental validation

4.5.1 Basic line scenario

In order to validate the decentralized controller that is designed in section 4.2.2 a real flight test has been executed using the Bebop2 quadrotors. We have embedded our decentralized controller on each of these followers. The experiment required three quadrotors, one leader, and two followers. The test scenario was rolled out in an outdoor environment in the vicinity of the laboratory building. I was the operator of the leader quadrotor, with the help of two engineers to interrupt the test in case of emergency or to recover quadrotors in case of an occasional drifting since the controller does not act on the y-axis. Figure 4.3 shows time-lapse images captured by the first follower quadrotor; the flight lasted in a total of 154 seconds, equal to 2 minutes and 34 seconds. In the first image, the first follower is still on the ground and preparing to take off; it immediately follows the leader. In the first part of the experiment, mainly between (1-50) seconds, the operator pushes the leader forward to see if the follower pursues the leader and keeps an appropriate separation distance; during this time, another follower quadrotor (not figured in

(a) t = 1 (b) t = 39

(c) t = 48 (d) t = 65

(e) t = 88 (f) t = 100

(g) t = 133 (h) t = 154

Figure 4.3.: shows a time-lapse from the outdoor test outside the laboratory with 3 quadrotors. The test have been executed on an asphalt non-completed road with no cars or public infrastructure. The images are captured by the second quadrotors.

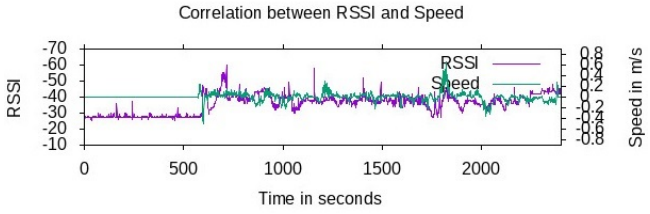


Figure 4.4.: shows the correlation between the Received Signal Strength Indicator (RSSI) and the velocity of the first follower quadrotor. The follower measure the RSSI from the leader and convert it according to the designed controller into one scalar value to be used in the velocity vector. The objective of the follower is to stay in the vicinity of the leader.

the images) takes off to follow the quadrotor with higher priority (first follower). Starting from the second minute, the operator stops moving the leader and verifies if the embedded controller stabilizes the two followers as they hover in their places. The operators start moving the leader backward from the second 88 to test if the two followers follow the same behavior. Figures in seconds 100 and 133 confirm this behavior. Finally, all followers land in the second 154 when they are close to the operator while the leader is still on the fly.

During the entire flight, no collision has been observed among follower quadrotors, neither with the leader. This experiment allowed us to i) validate the proposed approach, ii) verifying the autonomy of the followers, iii) their capacity to realize automatic takeoff and landing and pursue the leader.

Knowing that the quadrotors register entirely all data acquired during the flight, we have exploited these data using the logging feature provided by our software. The flight data 4.4 shows a correlation between the velocity vector of the follower quadrotor and the received signal strength from the leader, which means that the velocity vector increases when the received signal strength decreases. According to the figure, we can see that the received signal strength values remain relatively stable. In addition, results obtained from the second follower not showed here are comparable to the first one.

4.6 Conclusion

In this chapter, we have presented a simple decentralized controller for a set of quadrotors; the controller renders the follower autonomous, allowing to reduce the effort made by the operator to control the swarm by controlling only the leader. One might question the utility of such an approach by questioning real-life applications for this swarm. Therefore, among the

colossal set of applications for quadrotors, we have found an application related to video imaging, cinematography, and observation. For instance, with several sets of follower quadrotors in the air, the leader can capture a video and *relay* the video to the operator in real-time by using the followers as relaying points. In addition, each follower can capture its video and either rely upon the operator or store it internally. These videos help in increasing the coverage in the observed area by the quadrotor allowing to have more information per flight.

These applications are *only* related to quadrotors due to their characteristics, other unmanned aerial vehicles such as fixed-wing aircraft can not execute such a mission nor embed such a controller. Therefore, as we have mentioned in chapter 1, the totality of this work will be focused only on this robot. In the next chapter, we improve this controller and extend it into 3D action space using iterative learning techniques and supervised learning.

Part II

Decentralized controllers

Iterative Learning for Model Reactive Control (IL4MRC)

5

Contents

5.1 Motivation

In the last chapter, we presented a real-time decentralized controller for a set of quadrotors, following the leader-follower principle, the results have been validated using a real flight testbed that consists of several COTS Bebop2 quadrotor. The main drawback of that controller is its limitation on executing actions on only one dimension (pitching forward or backward), in which the swarm has only one degree of freedom with no possibility of movement on other dimensions.

In this chapter, we are interested in extending our previous work to cover a full action space in three dimensions, allowing the follower robots to be fully autonomous.

Starting from this chapter, we no longer consider our testbed, but instead, we use MagicFlock a home-made simulation environment that is based on RotorS (Furrer et al., 2016) and the high fidelity physics simulator (Gazebo) to simulate our quadrotors, more details about MagicFlock can be found in chapter 7. There are several reasons for this transition: it accelerates the testing of the controller, reduce the hardware cost in a case of outdoor collision and remove the need for testing engineers.

Building such a controller manually that allows robots to execute actions in 3 dimensions can be extremely complex given that all the robots have a severe limitation at the perception level. Similar to chapter 4, each follower is capable of sensing only distances to their neighbors. As the distance is a scalar value, it is rotationally invariant. This means that the distance value does not reflect the change of neighbors' locations. Therefore, several constraints need to be defined, such as the location of neighbors' quadrotors, the shape of the desired formation, the number of the neighbors, and the precise value of distance toward them to detect the changing dynamics and direction of leaders. In addition, the controller needs to be redesigned mathematically and programmed on each robot when one of the constraints changes.

When designing a controller, the design needs to consider the environment since it is stochastic by definition (e.g., wind, obstacles factors). Also, the complexity of the design might increase rapidly if several robots are present. In addition, the design should deal with sensors' impurity as they are subject to noise because of the vibration of the motors, resulting in a drifting behavior that is very common with quadrotors. This behavior has been observed in our outdoor testbed in chapter 4 during the testing of our controller.

Finally, if a programmed controller is designed successfully, it should also consider the computational resources available on each robot since the optimization problem needs to be resolved online during the flight. Such resources are scarce; if they are not appropriately managed, this will increase the battery drain, reducing the flight time substantially and making the robot less reactive.

5.1.1 Machine learning

A different approach can be addressed in this context, allowing the machine to design the controller to elevate these complexities related to constraints and environments. When using a machine learning approach, empirical data needs to be acquired. The data represents the system's

behavior. For instance, a robot needs to generate logs that represent the trajectories executed in the environment. Second, supervised machine learning (e.g., neural nets) is used to approximate the general behavioral law (e.g., controller) from these empirical data. Finally, the controller is refined gradually with subsequent iterations.

Another approach might build on different methods, such as using Reinforcement Learning to build such a behavioral law. However, RL requires a considerable amount of exploration and exploitation of the model. In addition, it is hardly complex to design a reward function that handles several agents interacting with each and with the environment. In this case, imitation learning is a better alternative due to the absence of a reward function, but it can not be used in this scenario since we do not have an oracle demonstrator. Therefore, in this chapter, we propose Iterative Learning for Model Reactive Control (IL4MRC) an agile alternative that is inspired by Inverse Reinforcement Learning with much lower sampling complexities.

5.1.2 Discussion

The shortcomings of a programmed controller can be analyzed as follows: 1) The model requires generations of trials and error by the engineers to be very accurate. 2) If the model is complex, it requires a hard optimization landscape that needs to be executed online to find the optimal action to execute.

From a machine learning perspective, the difficulty is twofold: First, some critical regions can hardly be identified *a priori*; secondly and most importantly, such critical regions might be hard to sample (e.g., have a minimal measure). When this is the case, significant amounts of empirical data are needed to gather sufficient evidence.

5.2 An iterative learning strategy

In this chapter, we present Iterative Learning for Model Reactive Control (IL4MRC), a machine learning approach is specifically conceived to address the challenge of the design of decentralized controllers with limited perception capacities. The empirical validation of IL4MRC is conducted in the context of quadrotors inside the MagicFlock simulation framework (described in chapter 7).

IL4MRC is inspired from Inverse Reinforcement Learning, imitation learning, and MPC. The aim is to learn a decentralized controller through 3 steps supervised learning iterative process.

In the first iteration, we need to build the first decentralized controller in 4 steps; the first one, logs are collected from each quadrotor embedded with a random controller. Inside these logs, each quadrotor reports the executed random trajectory as a vector composed of a set of states and actions. In the second step, a policy and a forward model are learned using the logs collected in the first step. In the third and fourth steps, the model is used similarly to MPC in order to deduce the controller.

In the second and further iterations, the controller built in the previous iteration is used to generate logs instead of the initial random controller. The logs are merged with the ones gathered in the previous iterations. The remaining steps are similar to the one executed previously, within the objective to refine the controller.

The proposed IL4MRC approach addresses the challenge under the following assumptions:

Assumption 3 *Assuming that the property is satisfied in the initial state of the system. The goal is to preserve the property of the system. Under this assumption, the desired command law boils down to: "In each time step, select the action most appropriate to avoid violating the property."*
This first assumption implies that the control problem can be tackled in reaction (instead of planning).

Assumption 4 *The action space noted $\mathcal{A}$ is discrete.*
This assumption makes it easier to verify if the control problem can be solved in all possible destinations of the robot.

Note that these assumptions hold for this specific application: the property of the system to be preserved is the initial geometric shape. Likewise, the action space of the robot is all the possible direction of the robot represented as a discrete set (e.g., Forward, Backward, Left, Right, Up, Down).

Under these assumptions, IL4MRC proceeds along an iterative 4-step process, starting from an initial random controller $a^{(0)}$. At the i-iteration:

1. trajectories defined as sequence of state and actions $\{(s_t, a^{(i)}(s_t)), t = 1 \dots T\}$ are recorded, where $a^{(i)}(s_t)$ is the action selected in state s_t by the controller learned in iteration $i-1$ (see below);

2. training set $\mathcal{E}_i$ is built, with

$$\mathcal{E}_i = \{(s_t, a^{(i)}(s_t)), t = 1, \dots T\} \tag{5.1}$$

3. model $\mathcal{F}_i$ is learned from $\mathcal{E}_i$. Denoting $\mathcal{Y}$ (respectively $\mathcal{A}$) the response or state space (resp. the action space),

$$\mathcal{F}_i : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{R} \tag{5.2}$$

   where $\mathcal{F}_i(s,a)$ is expected to be positive if selecting the action a in the state s leads to satisfying the sought property.

4. controller $a^{(i+1)}$ is defined as:

$$a^{(i+1)}(s_t) = \arg\min_{a \in \mathcal{A}} \{\mathcal{F}_i(s_t, a)\} \tag{5.3}$$

Compared to state of the art, the originality of the IL4MRC approach is twofold. On the one hand, the command law based on Eq. (5.3) is simple and computationally frugal, with complexity $\mathcal{O}(|\mathcal{A}|)$. On the other hand, the stress is put on visiting the (state, action) space based on the current model and gradually learning where this model needs to be refined. The complexity thus is shifted from the optimization task to the learning task and from the learning task to the data acquisition task. The main benefit is that the model trained from the data reflects by construction various sources of uncertainty and biases of the task at hand, either due to command imprecision, or to non-deterministic aspects of the system, or related to the inaccuracies of the current model.

5.3 A proof of principle of IL4MRC

This section presents a proof of principle of IL4MRC, applied to the control of a set of quadrotors. After describing the position of the problem, the algorithmic pipeline (data acquisition phase, training of the model, exploitation of the model) is detailed.

5.3.1 Position of the problem

Following the same problem described in chapter 1, we consider a set of quadrotors. In this case, two leaders and several followers have been used through different settings. The goal is to build an independent controller for each follower gradually. Each quadrotor is only endowed with a WiFi sensor, allowing them to have a minimal perception level of their neighbors. The sensor measures the RSS values from its neighbors that can be translated to distance using a propagation model (Union, 2019).

In this chapter, the learned controller should enable the follower to autonomously follow the leaders, such that all the quadrotors can collectively preserve their initial geometric pattern. That is said, this controller is not intended to achieve swarming or flocking behavior (such a complex policy is studied in chapter 6). Instead, this chapter establishes proof of principle for a controller that preserves the geometrical configurations.

One might think that maintaining the initial pattern is an algorithmically easy task given the WiFi sensors' information. Indeed, our first attempt was to directly write the algorithm similarly as done in chapter 4. However, several experiments described in 5.1 show that the distance traveled by the quadrotor varies significantly depending on its direction. Several lessons have been learned from these experiments. First, programming such a policy based on geometric reasoning and *in situ* sensor data is a deceptive task. Second, creating a data set synthetically using hand-written algorithms or generated from models instead of the simulation environment does not capture the system's actual behavior.

To address this problem, we have defined two settings to test the approach's scalability. The first set is composed of 3 quadrotors and described in 5.1, while the second one is composed of 4 quadrotors and described in 5.2.

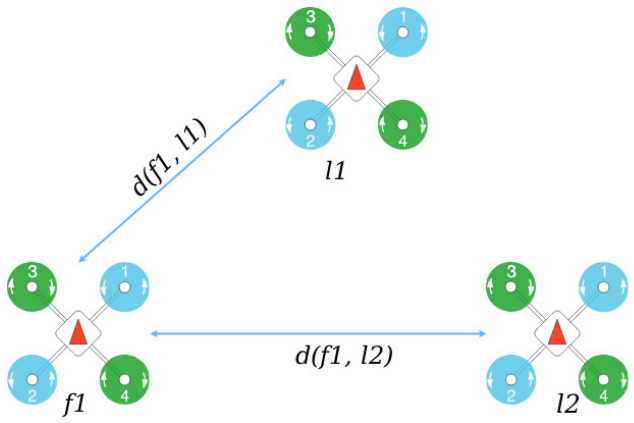


Figure 5.1.: 3-quadrotor settings: in this setting, the formation includes two *leaders*, labeled $l1$, $l2$, and positioned in the north and right side of the figure. Both of the leaders are remotely operated using the same random controller to simulate a human pilot. The formation include in addition one *follower* labeled $f1$ on the left side. The goal is to train the embedded follower controller with the objective to maintain the initial shape of the formation that is set to an equilateral triangle in these settings.

5.3.2 Data acquisition

During the data set acquisition step, the trajectory of each follower is recorded as a set of states and actions along with a set of episodes. Each episode starts with quadrotors taking off in the initial shape pattern according to the settings, either equilateral triangle in 3-quadrotor settings or rhombus in 4-quadrotor settings. The episode ends when all the quadrotors land if the property condition is heavily under satisfied; for instance, the geometric pattern of the formation is too far from the initial one.

The action space $\mathcal{A}$ is composed of 7 actions that the quadrotor can execute in this study: Forward, Backward, Left, Right, Up, Down, NoMove. The action *NoMove* is to stop the quadrotor from doing an action if the preserving property is satisfied.

In each episode, the takeoff process is monitored automatically to ensure that all quadrotors have takeoff successfully. Once this is done, the time steps are initialized, and at time $t = 0$, the leaders uniformly select the same action a_t^* in $\mathcal{A}$ and keep it constant for 10-time steps; another action is selected at time $t = 10$ and kept for 10-time steps, and so forth until the end of the episode. The idea is to simulate the behavior of a human pilot that moves the quadrotor is a specific direction. In the first iteration, each follower selects uniformly and independently an action a_t in $\mathcal{A}$ at each time step $t \geq 0$.

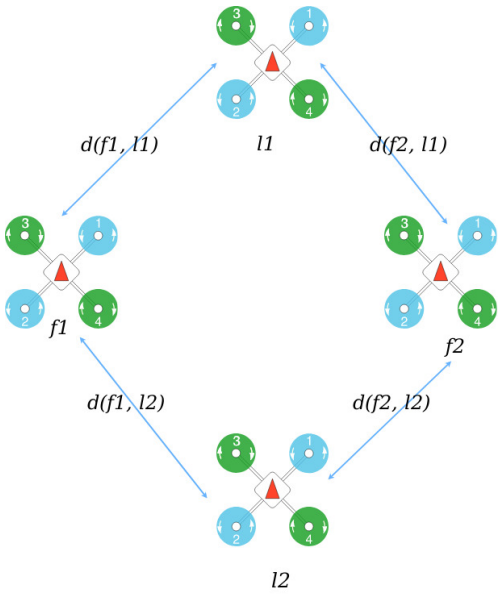


Figure 5.2.: 4-quadrotor settings: Similar to the previous setting, the formation includes two *leaders* labeled in the same manner but positioned differently. The leaders are situated in the north and the south of the figure. They embed the same random controller as in the previous settings. In addition to the leaders, there are two independent *followers* that are added. They are labeled $f1$ and $f2$ and positioned in the west and east of the figure. The goal is to train an embedded controller for each follower, enabling them to maintain the initial shape of the formation with the leaders, which is set to a parallelogram (rhombus).

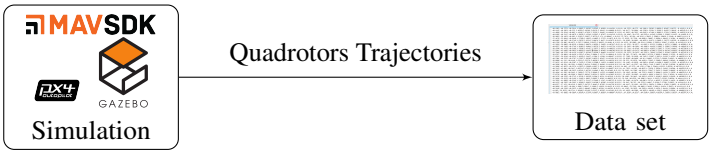


Figure 5.3.: Data set is registered from the trajectories that the quadrotor execute inside MagicFlock environment. All datasets are registered in real-time and then they are analyzed offline.

For each follower, the state $\mathbf{s}_t$ at time t is a 2-dimensional vector composed of its distance to each one of the two leaders. Eventually, the data set attached to each follower reports 1,500 episodes, where each episode is a varying length trajectory represented as a sequence as follows:

$$(\mathbf{s}_0, a_0, \mathbf{s}_1, \ldots, \mathbf{s}_t) \tag{5.4}$$

The sequence is ended with a terminal $\mathbf{s}_t$ iff:

$$\|\mathbf{s}_t - \mathbf{s}_0\| < \epsilon \tag{5.5}$$

with $\epsilon > 0$ a tolerance parameter. In this study the average of 1,500 episodes correspond to 7,000 pairs $a_t, \mathbf{s}_{t+1}$.

5.3.3 Forward model

Once the data set gathering phase is ended, the data set then is exploited to learn a forward model. The model aims to estimate the next state of the quadrotor based on its two last states and actions. Formally, the data set is organized as a set of pairs as follows:

$$(X = (\mathbf{s}_{t-1}, a_{t-1}, \mathbf{s}_t, a_t); Z = \mathbf{s}_{t+1}) \tag{5.6}$$

A Neural Network is used in the experiments as the mainstream supervised learning algorithm to learn the forward model. The architectural details of the Neural Network is detailed in section 5.4.4 and figure 5.6. They learn a function $\mathcal{F}$ such that $\mathcal{F}(X) = Z$ based on 80% of the data set. The remaining 20% part of the dataset is used to verify the quality of the learned model $\mathcal{F}$.

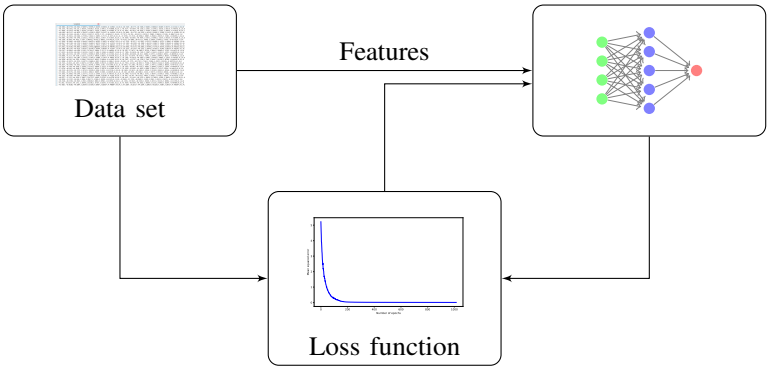


Figure 5.4.: shows the traditional supervised learning training process. Each line from the data set is considered as a feature vector composed from states and actions. The features are feed into a forward neural network that is used to train the model. The objective is to minimize a loss function in order to be able to predict the next state. The models are validated on a part of the dataset that is intended for this purpose

5.3.4 IL4MRC controller

Once the model $\mathcal{F}$ is learned, it can be used to support a decentralized controller as follows:

$$a^*_{t+1} = \arg\min_{a \in \mathcal{A}} \{\|\mathbf{s}_0 - \mathcal{F}(\mathbf{s}_{t-1}, a_{t-1}, \mathbf{s}_t, a)\|\} \tag{5.7}$$

The objective of the IL4MRC controller is to preserve the geometric figure of the quadrotor formation. Therefore, it needs to give the best action that is more amenable to bring it back to the initial state. The quadrotor uses its past state, its past action, its current state, and all actions from the action space in the forward model $\mathcal{F}$ to predict the next state vector. In addition, the quadrotor uses its initial state $\mathbf{s}_0$ to define the target property to be preserved. Once these elements are estimated, the controller can provide the best action that brings the quadrotor to the target state. The goal of choosing an independent forward model for each quadrotor, including the controller, is based on the data set collected separately. The strategy is to address the different behavior that each quadrotor might experience (e.g., drifting).

Once the controller is deployed on the quadrotors, it will follow a set of experiences as described in the next section 5.4. The test episodes are operated similarly to the episodes executed in the data set acquisition phase. Each episode starts with the quadrotors taking off. The leaders use the random controller to select a random trajectory to simulate a human pilot. They follow the same trajectory (set of actions) for ten consecutive time steps. In the meanwhile, each of the followers executes the trajectory that is generated by its IL4MRC controller. The test episodes end when

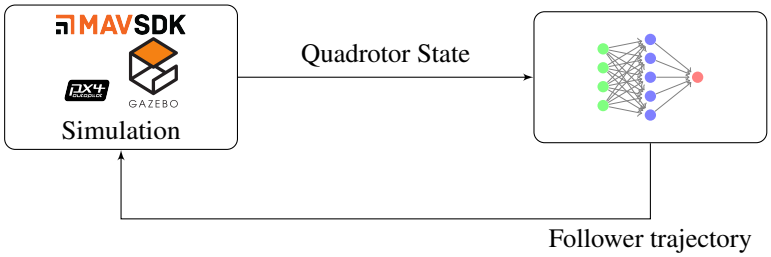


Figure 5.5.: Shows the test process, after the training and the validation steps, the model is tested directly on each follower quadrotor. The simulation environment provides the model with the state of each quadrotor and the model predict all possible next states according to action space, IL4MRC is used to deduce the best action to execute

the geometric pattern of the formation (triangle, rhombus) is very different from the target one, that is, when at least one follower is too far from its target state.

5.4 Experimental setting

In the following sections, we start a set of experiences in order to test and validate the IL4MRC controller. IL4MRC performance is compared with a set of baselines.

5.4.1 Goals of experiments

As stated earlier, the objective of the IL4MRC controller is to preserve the geometric configuration of the quadrotors during the flight. In order to have a performance indicator about the proposed controller, we propose to use the average number of the time steps that the follower quadrotors execute the geometric pattern is preserved up to the tolerance ϵ. However, high variability in the behavior was observed during the execution of several episodes run. These variabilities were analyzed and related to several factors:

- The continuous subsequent random moves of the leaders make them lose altitude and sink slightly, in some rare cases reaching the ground and ending the episode [1].
- Each one of the quadrotors has been identified drifting, especially when they are not moving.

[1] The issue has been investigated and identified as an autopilot issue. The bug has been signaled and sent to the PX4 development team for further investigation: `https://github.com/PX4/PX4-Autopilot/issues/12206`

Actions	forward	backward	left	right	up	down
mean	0.699932	0.713274	0.694962	0.704504	1.02292	1.00244
std dev.	0.00236473	0.00247612	0.00238012	0.00234223	0.00319365	0.00318693

Table 5.1.: Average traveled distance and standard deviation depending on the move direction, operated for 1 second with speed 1 m/s. Most unexpectedly, the average traveled distance depends on the move direction, and their distributions are statistically significantly different.

- The traveled distance is variable according to the action executed by each quadrotor see table 5.1 for more information.

We refer to these issues as the fatigue effect that the robots are suffering from during the experiments. Because of this fatigue, a more reliable performance indicator needs to be retained instead of the mean average. Therefore, several run of the episode have been executed to get the histogram and the cumulative distribution of the length of the episodes, reporting for each number of time steps t the fraction $z(t)$ of the episodes with a length less than t.

5.4.2 Baselines

Two baselines have been used in order to assess the performance of the IL4MRC controller. The first simple baseline is based on the random controller we used during the data set generation phase. The second refined baseline is based on k nearest neighbor (k-NN). This baseline exploits the exact same data set $\mathcal{E}$ as the one used to train the forward model. To each triplet $(\mathbf{s}_{t-1}, a_{t-1}, \mathbf{s}_t)$ is associated its k-nearest neighbors (with $k = 4$ in the experiments), where the distance is set to the Euclidean distance on the state space and the Hamming distance on the action space. Letting $(\mathbf{s}_{t-1}^{(i)}, a_{t-1}^{(i)}, \mathbf{s}_t^{(i)}, a_t^{(i)}, \mathbf{s}_{t+1}^{(i)})$ respectively denote the fragments of trajectories including these nearest neighbors, the selected action is $a_t^{(i)}$ such that it brings the quadrotor in state $\mathbf{s}_{t+1}^{(i)}$) as close as possible to the target state:

$$a_t = \arg\min_{i=1\ldots4} \left\{ \|\mathbf{s}_0 - \mathbf{s}_{t+1}^{(i)}\| \right\} \tag{5.8}$$

5.4.3 Simulation platform

In both, first and second settings, the system includes the same type of the robots, the IRIS quadrotor designed by 3DR[2], more details about the quadrotor type can be found in 7.4.1.

[2] https://3dr.com/

The quadrotors are simulated using MagicFlock, a home-made SITL platform for a swarm of quadrotors that is based on RotorS (Furrer et al., 2016). More details about MagicFlock can be found in chapter 7.

Robots velocity is fixed and set to 1m/s; the one-time step duration and actions are set to 1 second. The takeoff altitude is set during this chapter to 25 m. The tolerance ϵ on the deviation from the target pattern is 2 meters. For example, the pattern is broken, and the episode is ended if one of the quadrotors stays motionless for 3-time steps while the leaders move (e.g., forward).

5.4.4 Learning of the forward model

As said, the training data set records 1,500 episodes, totaling circa 7,000 time steps on average. The forward model is implemented as a neural net represented in figure 5.6, using mlpack (Curtin et al., 2013). The neural architecture is a 2-hidden layers, with 200 neurons on each layer, with Leaky ReLU as an activation function (Nair and Hinton, 2010). The training uses Glorot initialization (Glorot and Bengio, 2010), with .5 Dropout and batch size 32; the hyper-parameters are adjusted using Adam (Kingma and Ba, 2014) with $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-8}$, and initial learning rate $\alpha = 0.001$.

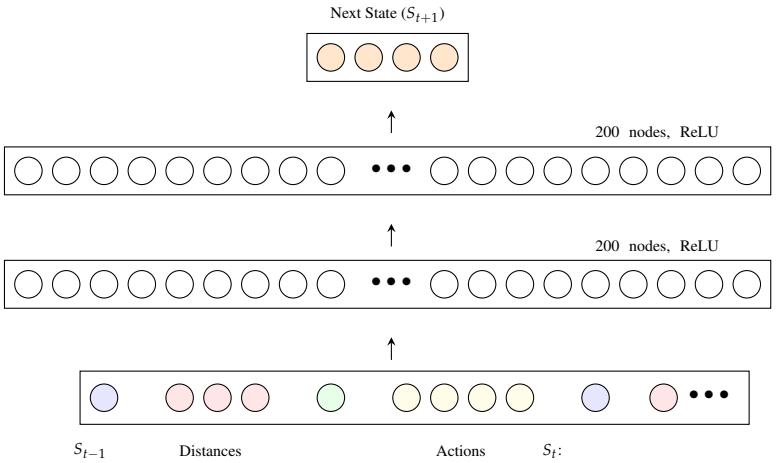
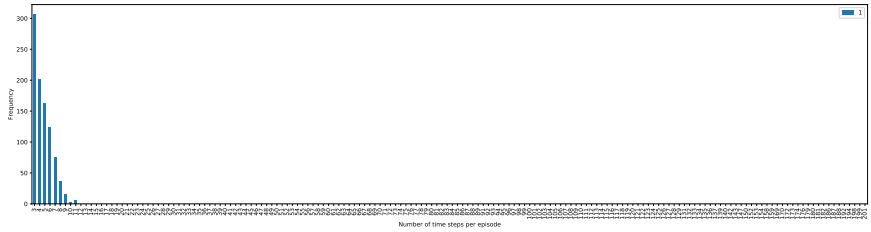
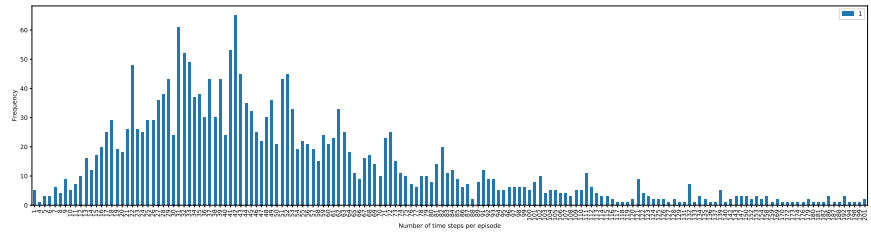


Figure 5.6.: shows the neural network used in order to train the model. The network is composed of an input layer, output layer, and two hidden layers. The input layer takes the states and actions and predicts the next states according to the all-provided action space.



(a) Histogram showing the performance of random controller for 3 quadrotors settings



(b) Histogram showing the performance of IL4MRC controller for 3 quadrotors settings

Figure 5.7.: shows the histogram for random controller vs IL4MRC controller when used on 3 quadrotors settings. The x-axis represents the number of time steps executed by the follower quadrotors during a specific episode, while the y-axis represents the corresponding frequency of episodes. We can see that the random controller never passes the 11-time steps for around a thousand episodes of runs, while IL4MRC controller can maintain the follower up to 200-time steps.

5.5 Empirical validation

To assess the performance of IL4MRC, we have run experiments described in subsection 5.4. The results are described as a set of histograms for the random and IL4MRC controller and as a Cumulative Distribution Function (CDF) of the random, the k-NN, and the IL4MRC controller.

Figure 5.7 reports the comparison of the random controller and the IL4MRC controller in the 3 quadrotors settings. Each bin represents the frequency of the episodes that run for a specific number of time steps in the following.

Since histograms show the performance for each quadrotor independently, it will be tough to have a performance comparison for all the controllers. Thus, we have reported the Cumulative Distribution Function (CDF) for all of the controllers combined. Figure 5.8 reports the Cumulative Distribution Function of the episode lengths in the 3-quadrotor and 4-quadrotor (button) settings. The difficulty of the problems is evident as the random controller loses track of leaders after 10-time steps on circa 90% of episodes in the 3-quadrotor setting, and almost 100% of episodes

in the 4-quadrotor setting. The k-NN controller significantly demonstrate a better performance compared to the random controller: circa 10% of the episodes last more than 20-time steps. Surprisingly, the performances of the k-NN controllers are quite similar in both settings since this is related to the size of the data set. One might propose to increase the size of the data in order to improve performance. However, this will affect the controller adversely since it will increase the necessary latency to search inside the data set and reduce the speed.

IL4MRC significantly improves on the k-NN controller: 60% (respectively 30%) of the trajectories last more than 40-time steps in the 3-quadrotor (resp. the 4-quadrotor) setting. The enhanced performance of IL4MRC compared to the k-NN can be easily understood from the generalization effect: the k-NN model can not provide reliable estimates in regions that have not been visited in the training set, and, more generally, the size of the data set limits its accuracy. On the opposite, under the assumption that the target behavior is sufficiently smooth, the IL4MRC model can estimate the behavior of the quadrotor in regions which have rarely been visited in the training set.

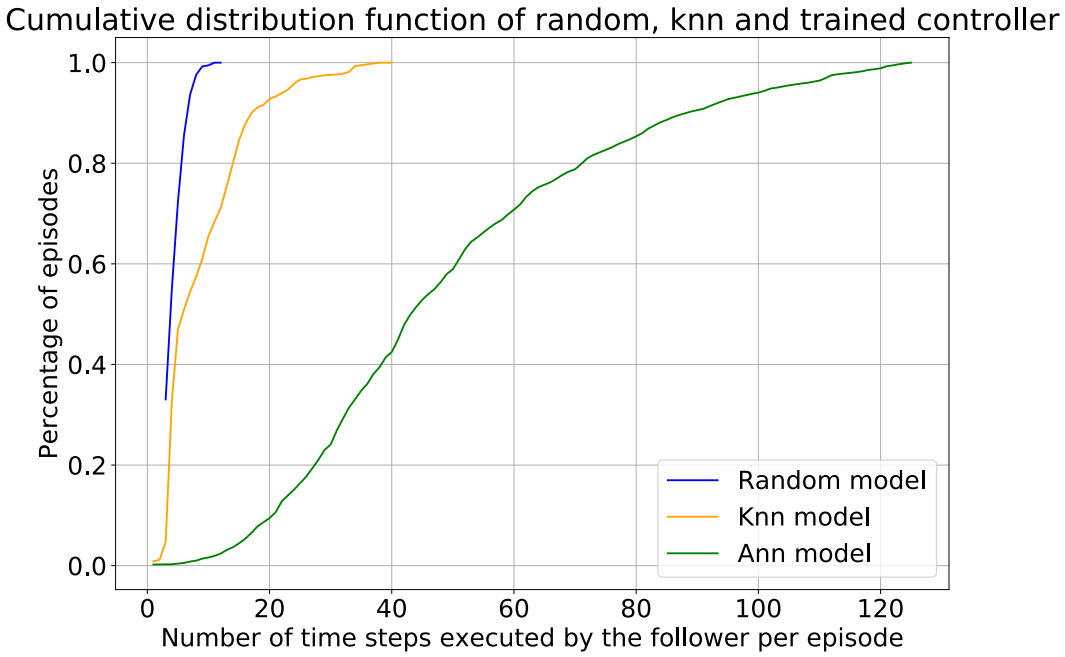
The performance of the next iterations has been evaluated to see if the controller improves over time. Our results described in figure 5.9 have showed a slight improvement over time. However, because of the fatigue effects described in section 5.4.1. The quadrotors were not able to maintain the geometric configuration over time. This result has proven the concept of IL4MRC controller. In the next chapter, we will focus on improving the controller iteratively.

Finally, a set of images that have been captured from the experimental simulations are joined in this chapter. The first figure 5.10 shows the three quadrotors settings. The leaders are on the right side, and the follower is on the left side. The leaders start executing its action one step before the follower. The follower then observe the change in its state and execute the action provided by IL4MRC controller. The second figure 5.11 shows the four quadrotors setting in action. Where leaders are in north and south, and followers are on the left and the right side.

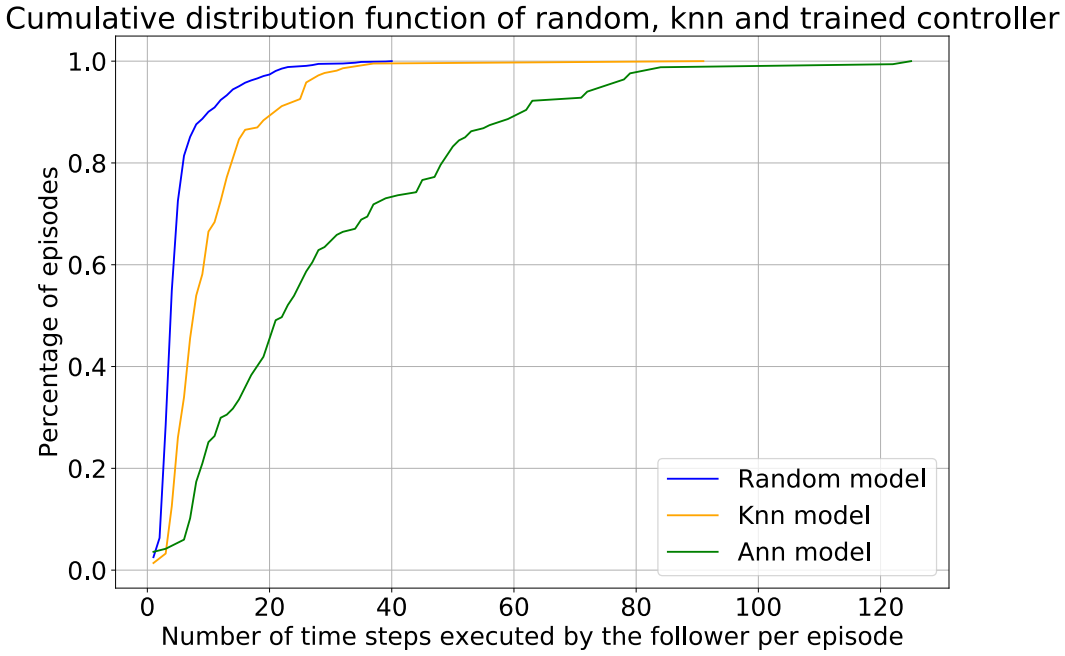
The simulation runs for about 24 hours, making it impossible to register the entire simulation. Therefore, we have captured a set of videos that demonstrated a normal behavior of IL4MRC and special behavior cases. All the videos are captured *before* the end of the episodes to reduce the video size and timing. The simulation videos are available through this link:

`https://www.dropbox.com/sh/97ixrp0ayejvim0/AADdIRkRWxtexlCP2C-nFGQEa?dl=0`

Here are a set of comments that describe each of the following video available online:



(a) Performance of IL4MRC for 3 quadrotors settings



(b) Performance of IL4MRC for 4 quadrotors settings

Figure 5.8.: Performance of IL4MRC on the 3-quadrotor setting (top) and the 4-quadrotor setting (bottom), compared to the random and the k nearest neighbor baselines. For each controller and each time step t on the horizontal axis, is indicated the fraction $z(t)$ of the episodes terminated before t time steps.

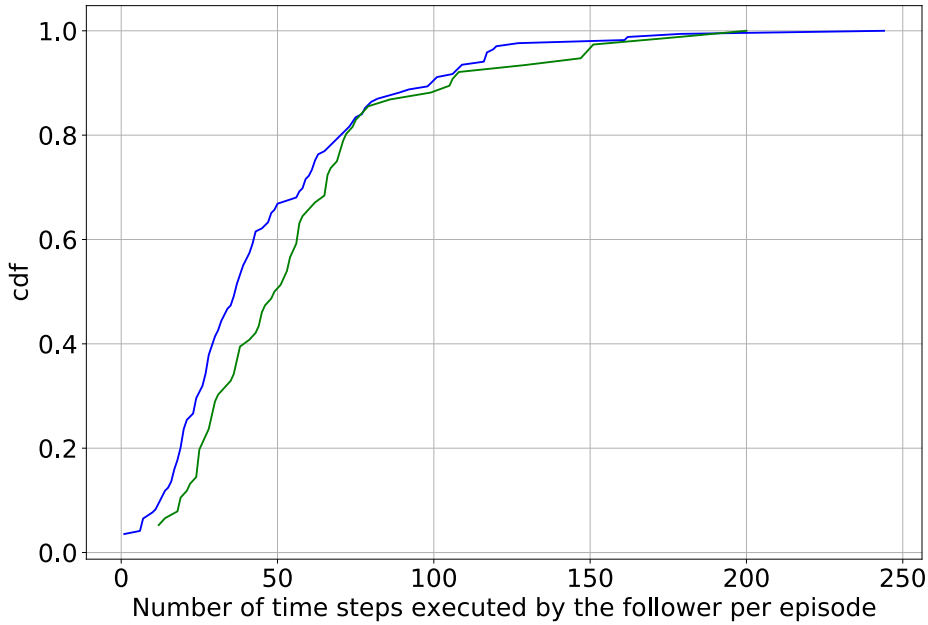
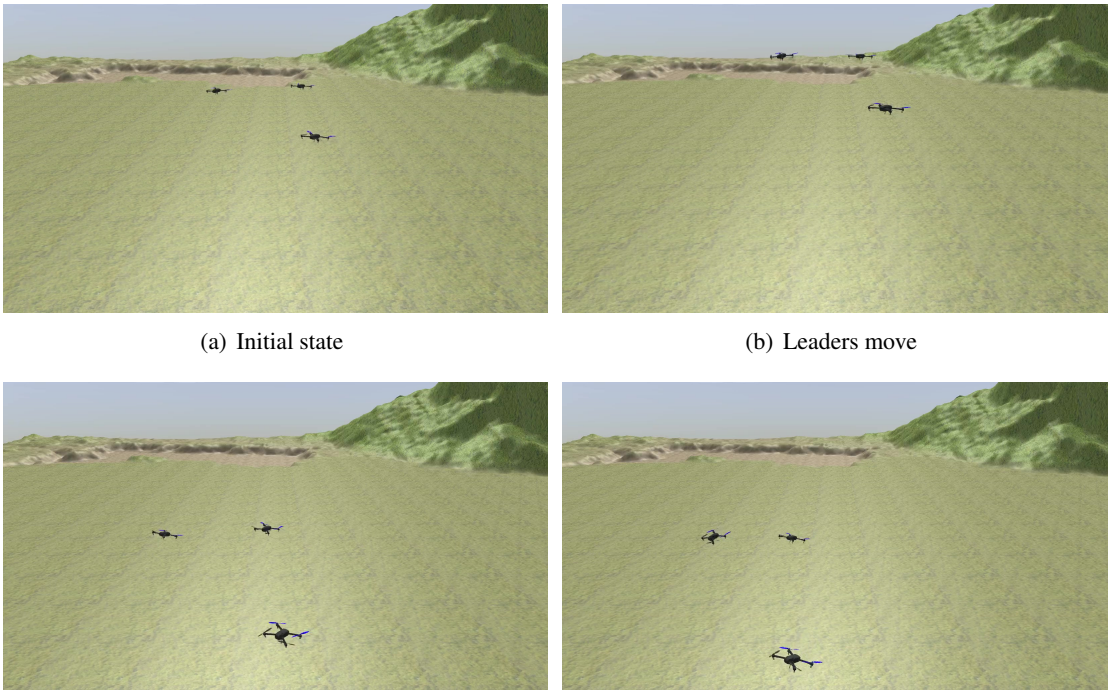


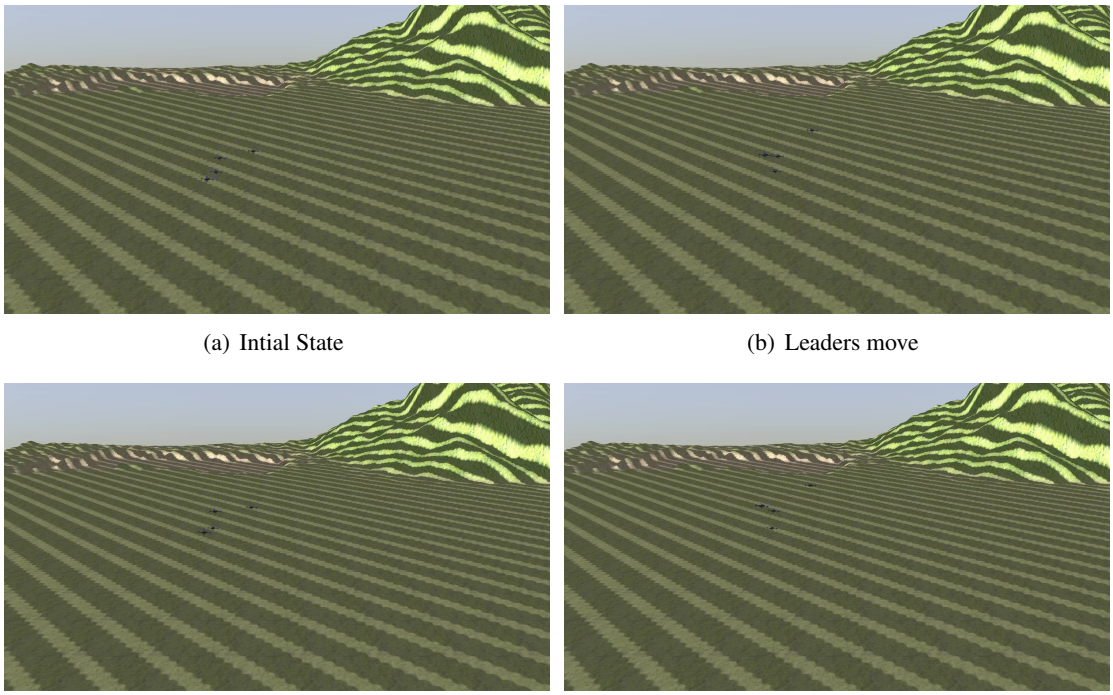
Figure 5.9.: shows the performance of IL4MRC on the 4-quadrotors settings compared to the second iteration, in the second iteration we have used IL4MRC controller from the first iteration to generate data set, the performance of the second iteration (green) is slightly better than the first one (blue), but does not improve further. This limitation is related to the drifting effect faced by the leader quadrotors, and other issues described in section 5.4.1. These difficulties are resolved in the next chapter showing a considerable improvement in learning between iterations.



(a) Initial state (b) Leaders move

(c) Follower moves, initial state (d) Leaders move and so on...

Figure 5.10.: shows a set of images captured from a simulation video. In the first image, all the quadrotors are in the initial state where the geometric figure is perfectly conserved. In the time step shown in the image, the leaders move backward toward the follower. The follower moves backward in the third image, and the triangle is back to its original state. The flight continues in the last image as leaders go backward.



(a) Intial State (b) Leaders move

(c) Followers move, initial state (d) Leaders move and so on...

Figure 5.11.: Similar to the above figure with a set of 4 images captured from simulation video. The first image is the initial state, in the second one the leader moves up, in the third one, the followers execute the same action and we are in the original state, and finally the process repeat. The time difference between images is equal to one step.

- Videos number 1, 2, 3, and 5 shows normal behavior of the IL4MRC controller.
- Video number 4 shows a rare case where the geometrical pattern has been inverted, and the IL4MRC controller continued to work.
- Videos number 6, 7 show the drifting effect of the leader or the follower.
- Video number 8 tests the reactivity of the controller, showing the leaders changing their action each time step instead of 10-time steps. This test was only to show if the Iterative Learning for Model Reactive Control (IL4MRC) controller could stay behind the leader in the case of aggressive pilot behavior.

5.6 Conclusion

In this chapter, we presented Iterative Learning for Model Reactive Control (IL4MRC) a new approach for decentralized controller design that is based on machine learning. IL4MRC works under two assumptions: First, the target behavior is based on a property-preserving task, Second, the action space is discrete.

A proof of principle is presented that illustrates the approach, showing the possibility to maintain a set of quadrotors equipped with minimal WiFi sensor instead of sophisticated LIDAR or RGB-D cameras. A significant advantage of the IL4MRC controller is to move the computational load from the online flight phase to the data acquisition phase. This enables to learn a short-sighted forward model that permits fast optimization of the following action, and handles seamlessly the uncertainty about the actual current state of the robot.

Using the empirical evidences demonstrated in section 5.5, this method suggests that sophisticated behavior is possible and can be achieved using offline learning in addition to exploiting the online forward model. In the next chapter, we extend this method to achieve more complex policies (e.g., flocking behavior) inspired by imitation learning.

Autonomous quadrotors swarming using Imitation learning

6

Contents

6.1 Motivation

In the previous chapter, we presented IL4MRC, an iterative learning method to learn a decentralized controller for a set of quadrotors. When the controller is embedded on followers, it is allowed only to achieve a simple policy, such as preserving the geometric configuration during

the flight. In this chapter, we extend the IL4MRC to build a controller that can achieve more complex policies (e.g., flocking behavior). Similar to our previous work, we assume that each quadrotor is embedded only with one WiFi sensor.

In chapter 3 we have presented a set of flocking models that were inspired by the animal kingdom. This chapter aims to make the follower quadrotors learn how to swarm to behave similarly to birds or fishes. This implies that each quadrotor has to stay in the vicinity of the neighbors to avoid collision and separation. We recall our problem statement illustrated in chapter 1 on having a human operator that pilots a leader quadrotors, and a set of followers are pursuing the leader.

It could be very advantageous to integrate flocking models on a set of quadrotors. On the one hand, they are mathematically simple, allowing easier understanding and manipulation. In addition, they are modular, allowing for each model to be changed, improved, or adapted for each use case. For instance, Schilling et al., 2019 adapted the Reynolds flocking model to add a migration rule in order to allow for a set of quadrotors to move from one point to another. Alternatively, they can be modified to avoid obstacles, as presented by Olfati-Saber, 2006. Virágh et al., 2013 adapted Vicsek model to become more realistic for real-world scenarios.

On the other hand, It can be more complicated to embed these models directly on real-world robots without tedious adaptation. Knowing that for the flocking model to behave perfectly, each robot requires access to the precise position of its neighbors. This can be possible only in simulation. In reality, outdoor global localization systems such as GNSS system provides a position with a certain amount of errors (Karaim et al., 2018) that depends on several factors such as clock drifting issues in satellites and receivers (El-Rabbany, 2002), troposphere errors (Hofmann-Wellenhof et al., 2007), the amount of noise of the receiver sensors (Han and Wang, 2010), not to mention intentional errors are imposed (e.g., by operators) to reduce the localization accuracy (Curran, 2017; Hofmann-Wellenhof et al., 2007), in addition to these errors, there are those related to radio signals (reflection, diffraction, multi-path, etc.) (Meguro et al., 2009). Because of these factors, the approximate position can vary, such a variation in position error values can not allow swarming a team of small-sized quadrotors since it increases the probabilities of collisions. One might propose to correct these errors, such as using Support Vector Machine (SVM) to detect multipath signal error (Bassma, Tayeb, et al., 2018; Hsu, 2017), or using Bayesian filtering methods in order to improve the accuracy (Yozevitch, 2017).

As detailed, there are several limitations of the GNSS system; in addition to its exclusive usage in an outdoor environment, one might require to add several mechanisms to improve accuracy. In indoor, this system does not work. Indeed, there are several alternatives in an indoor environment, with a similar precision limitation of GNSS. However, whether indoor or outdoor, a complete

controller redesign is required, increasing the limitations and dependency on the environment, not to mention the expensive sensors required when using localization systems (indoor, outdoor).

To alleviate the complexity related to these localization systems, one solution is to remove the need for such a system by replacing the absolute position information with reference local information. Each quadrotor would receive two pieces of information from its neighbors, the distance and the angle toward each robot in the neighborhood. The objective is to collect the necessary data to learn the flocking behavior.

When the followers execute the flocking model, the collected data correspond to the trajectories executed by these robots. In order to learn from these trajectories, a specific set of machine learning algorithms can be considered, such as Reinforcement Learning, Inverse Reinforcement Learning and imitation learning [1]. The particularity of these methods is their ability to learn from trajectories and predict the next sequences of actions. These methods may give a similar output, but they diverge in how they carry out their learning process. For instance, Reinforcement Learning requires to design a reward function. While imitation learning requires an oracle demonstrator that generates the trajectories in the environment (e.g., simulator, a real testbed). More importantly, several researchers (Bhattacharyya et al., 2018; Le et al., 2017) have demonstrated that imitation learning can be combined successfully in the case of Multi-agent System to learn a specific policy.

In the scenario presented in this chapter, the flocking model act as the perfect oracle demonstrator to execute the policy repeatedly inside the simulator to gather the number of required trajectories executed by each one of the follower quadrotors.

6.2 Overview of Iterative Imitation Supervised Learning

Before introducing I2SL, we start by introducing our adaption of the Reynolds flocking model which is used as an oracle during the learning process. Our adaptation builds on the modified flocking model of Schilling et al., 2018

6.2.1 Flocking algorithm

In chapter 3, we presented the basic Reynolds flocking model that handles the separation and cohesion rules between swarming agents. Schilling et al., 2018 added a migration rule to these

[1] more details can be found in chapter 3, section 3.7

models, which allow all quadrotors to migrate toward a specific predefined point in the space concerning cohesion and separation.

In our case, there is no need for migration term, as the considered swarm comprises a set of autonomous follower quadrotors and one human-operated leader. Rather, we insist that the followers migrate toward the leader permanently. Note that the leader quadrotor is unaware of the followers. Therefore let us consider that the migration point is the leader, not a fixed point but a dynamic one.

This flocking model involves three terms, the *cohesion*, *separation*, and a leader *migration* term. Let $\mathcal{N}_i$ denotes to the set of neighbor follower quadrotors of the quadrotor i, with

$$\mathcal{N}_i = (follower\ j : j \neq i \wedge \left\|\mathbf{r_{ij}}\right\| < r^{max}) \tag{6.1}$$

where $\|.\|$ is the euclidean norm. $\mathbf{r_{ij}} \in \mathbb{R}^3$ is the relative position of the quadrotor j with respect to follower i is given as follows:

$$\mathbf{r_{ij}} = \mathbf{p}_j - \mathbf{p}_i \tag{6.2}$$

There is only one leader quadrotor in this swarm; therefore, the follower quadrotor can not divide themselves into several sets. Formally, the swarm is said to be valid as long as the distance between any robots in the swarm is $d_{ij} < 30$ meters.

The three terms cohesion, separation, and migration work together to produce the flocking behavior; the separation term pushes away close agents to each other to avoid a collision; inversely, the cohesion term moves far away quadrotors toward their nearest neighbors. Both terms work together in order to provide a consistent swarm behavior.

The separation term prevent agents from collision by pushing them away from each other. The minimum distance allowed between the quadrotor is modeled by changing the separation gain k^{sep}. The separation velocity is given as follows:

$$\mathbf{v_i}^{sep} = -\frac{k^{sep}}{\mathcal{N}_i} \sum_{j \in \mathcal{N}_i} \frac{\mathbf{r}_{ij}}{\left\|\mathbf{r_{ij}}\right\|^2} \tag{6.3}$$

The cohesion term prevents the agents from dispersion by moving far agents toward their nearest neighbors. Similarly to the separation gain the cohesion gain k^{coh} modulate the cohesion distances. The cohesion velocity is given as follows:

$$\mathbf{v_i}^{coh} = \frac{k^{coh}}{\mathcal{N}_i} \sum_{j \in \mathcal{N}_i} \mathbf{r}_{ij} \tag{6.4}$$

The two terms work together to provide a consistent swarm behavior, therefore the sum of two velocities produce the basic Reynolds velocity:

$$\mathbf{v_i}^{rey} = \mathbf{v_i}^{sep} + \mathbf{v_i}^{coh} \tag{6.5}$$

In addition to the above terms, the migration term allows the followers quadrotors to move toward the leader quadrotor. As described earlier, the migration point is not fixed: it is the position of the leader itself. The k^{mig} is the migration gain and the migration velocity of the followers is given by:

$$\mathbf{v_i}^{mig} = k^{mig} \frac{\mathbf{r}_{ij}}{\left\|\mathbf{r_{ij}}\right\|} \tag{6.6}$$

Where $\mathbf{r}_{ij} \in \mathbb{R}^3$ is the relative position of the migration point w.r.t. the i-the quadrotor is given by

$$\mathbf{r}_i^{mig} = \mathbf{p}^{leader} - \mathbf{p}_i^{follower} \tag{6.7}$$

In order to achieve the flocking behavior, the controller embedded on each follower uses the sum of the three velocity commands:

$$\mathbf{v_i} = \mathbf{v_i}^{sep} + \mathbf{v_i}^{coh} + \mathbf{v_i}^{mig} \tag{6.8}$$

The leader is normally operated by a human pilot, with a limited velocity (racing tasks are not considered in the following). In simulation, the maximum speed of the flocking model is bounded to a maximum final velocity command, set to $v_{max} = 1m/s$, and the velocity of each follower $\mathbf{v}_i$ is accordingly bounded as:

$$\mathbf{v}_i = \min(\|\mathbf{v}_i\|, v_{max}) \tag{6.9}$$

6.2.2 Iterative Imitation Supervised Learning (I2SL)

We present Iterative Imitation Supervised Learning (I2SL), an iterative approach based on imitation supervised learning that is inspired from Ross et al., 2011; Schilling et al., 2018; Shrit et al., 2021. The presented work extends the IL4MRC method (Shrit et al., 2021), which likewise aims to achieve a decentralized controller with cheap embedded wireless sensors. The contribution is based on the combination of IL4MRC with Dataset Aggregation (DAGGER), along with an iterative approach, gradually refining the controller learned in the former iteration, thereby exploring only the necessary information rather than exploring the entire environment.

Formally, I2SL run as follows: In the first iteration, the leader is assigned a random model to simulate a human pilot, while the followers are using a flocking model, and each quadrotor generates logs describing its state as a vector of sensor values. The controller learned from these logs immediately enforces the following of the leader, i.e., it supports the *migration* function but does not enforce the *Cohesion* and *Separation* rules in order to avoid one another. In the second iteration, the former controller is used in alternation with the flocking policy following the DAGGER approach. The obtained trajectories thus alternate between avoiding any possible collision and following the leader. After several iterations, while each quadrotor is controlled from its embedded controller, they collectively *swarm* around the (remotely controlled) leader, i.e., they satisfy the main swarm properties: i) following the leader, ii) avoiding collision and preventing separation of the neighbor followers.

The contributions of I2SL can be described in two perspectives:

- From the control perspective: I2SL addresses the challenge of the design of a decentralized swarm controller using agile imitation learning.
- From the perception perspective: the goal is to learn a decentralized swarm controller using only one wireless sensor on each quadrotor, with very limited computation onboard.

I2SL aims to relieve the simplifying assumptions of discrete action space and a good initial condition of the swarm imposed by IL4MRC in chapter 5. We show that using the flocking policy to generate the logs that will serve the imitation-based controller learning alleviates the need for such simplifying assumptions. Formally, the proposed iterative imitation approach uses a 3-step process inspired by the DAGGER algorithm (Ross et al., 2011).

At the iteration i:

1. The trajectory executed by each quadrotor is logged, defined as a sequence of states s_t and actions a_t. This trajectory is generated after controller π_i using the DAGGER mechanism:

$$\pi_i = \beta_i \pi^* + (1 - \beta_i)\hat{\pi}_{i-1} \tag{6.10}$$

   Where π^* is the flocking policy, and $\hat{\pi}_{i-1}$ is the policy learned in the last iteration. β is decayed exponentially from 1 to 0 over time as $\beta = e^{-\lambda i}$ where $\lambda = 0.69314$ is a constant.

2. The trajectories generated in the above are logged to form a training dataset $\mathcal{E}_i$:

$$\mathcal{E}_i = \{(s_t, a_t), t = 1, \dots T\} \tag{6.11}$$

   and the datasets are stacked:

$$\mathcal{E} \leftarrow \mathcal{E}_i + \mathcal{E}_{i-1} \tag{6.12}$$

3. The model $\mathcal{F}_i$ is trained from $\mathcal{E}$ to learn the best action to execute based on the sequence of the last states:

$$\mathcal{A} = \mathcal{F}_i(\mathcal{S}) \tag{6.13}$$

This section presents a proof of principle of I2SL,[2] applied to the control of a set of quadrotors. After describing the position of the problem, the algorithmic pipeline (data acquisition phase, training of the model, exploitation of the model) is detailed.

6.2.3 Position of the problem

Following the problem statement in chapter 1, we recall the problem addressed in this chapter considering a set of quadrotors with one leader and several followers; the goal is to build an independent controller gradually for each follower, knowing that each follower has very minimal sensing capabilities, such as measuring the distance, the azimuth, and elevation angles to its neighbors. The objective of this controller is to achieve the flocking behavior known as swarming for the followers' quadrotors, where a human pilot manually controls the leader quadrotor.

[2] https://github.com/shrit/MagicFlock

Following (Shrit et al., 2021), the main goal of the proposed approach is to achieve some trade-off between efficiency and computational resources. On the one hand, the cost of sophisticated sensors is an issue. On the other hand, quadrotors consume a considerable amount of energy when carrying heavy sensors, not to mention the algorithmic complexity to analyze the perceived data from the environment.

In the simulation, Gazebo provides a generic wireless sensor that can be added to each quadrotor; we have integrated three antennas on each robot. The wireless sensor considers obstacles in the nearby robots which affect the value of the received signal strength according to the number and density of the obstacles. The proposed method does not require sharing information using the wireless channel, which means there are no direct communications between the agents. In addition, for the only sake of simulation, we add a ray sensor on each quadrotor to provide angle estimation (azimuth and elevation) to neighbors. During simulation, as provided by the link in the abstract to experiment videos, we have turned off ray visualization to remove heavy computations from GPU. However, in a real-life scenario, one can use COTS quadrotors that have an embedded WiFi card such as Intel 5300 with 3 antennas allowing to estimate the AoA of signals and the RSS values from neighbors.

Each robot is capable of mapping distances and angles to its neighbors. One might argue that robots can create a polar coordinate system, thus gradually constructing a relative localization system. Indeed, this system can be used directly by the flocking model and remove the need for imitation learning. Arguably, this is true. However, there are two disadvantages to this method. First, this will require more computation from each robot since it needs to calculate the relative position of neighbors in each time step and then apply the calculations related flocking model. Second, the sensor noise needs to be estimated before the flight, and proper modifications have to be applied to the flocking model accordingly. However, imitation learning alleviates the need for calibration, as the noise is embedded inside the data. The learned controller has a better estimation of its neighbors, allowing it to perform as well as the oracle flocking model as demonstrated in section 6.4.

quadrotors settings we have defined one setting of quadrotors to train and to test the models on the quadrotors. We have a set of seven quadrotors, in which there are one leader and six followers. The goal is to embed the **same** trained controller on all the followers to *follow* the leader without having any collision with the neighbor quadrotors.

6.2.4 Data acquisition

During the data acquisition, the state of each follower is recorded along with a set of episodes. The state is defined as a vector of the signal strength, the azimuth, and the elevation angles perceived from neighbors $i = 1, ..., n$. Therefore, at each time step t the state of the quadrotor j is given as:

$$s_t^j = (rss_1, \phi_1, \theta_1, rss_2, \phi_2, \theta_2, ..., rss_n, \phi_n, \theta_n) \tag{6.14}$$

While the action a_t^j is the velocity vector $\mathbf{v}$.

Each follower registers two data sets simultaneously. The first data set contains the states of the leaders along with the migration velocity as given by the flocking model. The second data set contains the state of the other followers' neighbors along with the Reynolds velocity as given by the flocking model.

First iteration Each episode starts with quadrotors taking off. Once the taking off has finished, the leader chooses a direction randomly and moves in this direction for 80 seconds. The follower quadrotors use the flocking model and start following the leader. The max velocity of the leader is equal to 0.7 meters per second which is slightly lower than the followers, equal to 1.0 meters per second. This slight variation allows the followers to catch up with the leader. The episode ends once the followers are close to the leader and there is no longer any distance change. When the quadrotors land, they are reset into a new position to ensure the diversity of the collected data set, allowing each quadrotor to have a different set of neighbors.

Second iteration Similar to the first iteration, each episode starts by taking off, and then the leader moves before the followers. The main difference is that the value of β is reduced from 1 to 0.5, allowing alteration between the flocking model and model trained in the first iteration.

Third and further iterations The following iterations follow the same principle in the second iteration while continuing to reduce β.

6.2.5 Forward model

The data set is exploited to learn a forward model. The forward model uses the last states to predict the action to execute at this time step. Formally, the data set is decomposed as a set of the last five states that are trained to predict the action a_t.

$$(X = (s_{t-4}, s_{t-3}, s_{t-2}, s_{t-1}, s_t); Z = a_t) \tag{6.15}$$

We use a mainstream supervised learning algorithm to train a function $\mathcal{F}$ such that $\mathcal{F}(X) = Z$ from 80% of the data. During the training, the quality of the model $\mathcal{F}$ is estimated by applying the model on the validation set, which comprises 20% of the data set.

6.2.6 I2SL controller

At production time, the decentralized controller is embedded on each quadrotor. The controller is composed of two forward models, the first model $\mathcal{F}^1$ is trained on the data set that is based on sensor value received from the leader while the second model $\mathcal{F}^2$ is trained on data set received from the neighbors followers.

$$a_t^{(1)} = \mathcal{F}^1(s_{t-4}, s_{t-3}, s_{t-2}, s_{t-1}, s_t) \tag{6.16}$$

$$a_t^{(2)} = \mathcal{F}^2(s_{t-4}, s_{t-3}, s_{t-2}, s_{t-1}, s_t) \tag{6.17}$$

$$a_t^* = a_t^{(1)} + a_t^{(2)} \tag{6.18}$$

The quadrotors are operated at production time very similarly as in the data acquisition phase. In each episode, the quadrotors take off, the leader is randomly operated. The leader starts moving 5 seconds before the followers to allow the followers to accumulate a decent amount of states from the sensor to predict the excellent action. The episode runs as long as the flocking behavior is maintained, and neither collision nor separation is noticed. As said, we consider that the flocking is maintained as long as the distance between two quadrotors $d_{ij} < 30$ meters.

6.3 Experimental setting

This section describes the goal of experiments and the experimental setting used to validate the I2SL approach.

6.3.1 Goals of experiments

As said, an episode starts with the swarm taking off. Once the taking off has finished, the leader starts moving 5 seconds before the followers. We create a ZigZag experiment that allows testing if the learned models are capable of imitating the flocking behavior, and how the followers are behaving when the leader changes its direction from time to time.

Such experiment aims to simulate a human pilot flying the leader quadrotor through a specific trajectory while the followers keep appropriate distances among all of them.

The straightforward performance indicator is to measure the minimum and the maximum distance among the follower quadrotors during the flight; these indicators reflect the consistency of the learned flocking behavior. The distance metric is given by:

$$d^{min} = \min_{i,j \in \mathcal{N}} \left\| \mathbf{r}_{ij} \right\| \tag{6.19}$$

$$d^{max} = \max_{i,j \in \mathcal{N}} \left\| \mathbf{r}_{ij} \right\| \tag{6.20}$$

Where $\mathcal{N}$ if the set of follower quadrotors and d^{min}, d^{max} is respectively the minimum and the maximum distance observed in the follower quadrotors swarm. In addition to indicators, we show the trajectory executed by the leader and followers for each iteration and experiment.

6.3.2 Baseline

To assess the performance of I2SL we used the flocking model described in 6.2.1 that uses the absolute positioning system. The flocking model (oracle) delivers the perfect flocking behavior when knowing the exact position of all the neighbor followers. Of course, we have chosen the gain of the flocking model carefully since they modulate the strength of the cohesion and the separation of the swarm. The behavior in each iteration is compared with the behavior obtained in the former iteration, and with the flocking model.

6.3.3 Simulation platform

In the experiments, the system includes seven robots of the same type, the IRIS quadrotor designed by 3DR[3], more details about the quadrotor type can be found in 7.4.1. The quadrotors

[3] https://3dr.com/

are simulated using MagicFlock, a home-made SITL platform for a swarm of quadrotors that is based on RotorS (Furrer et al., 2016). More details about MagicFlock can be found in chapter 7.

The maximum velocity that a quadrotor can reach is 1m/s, the takeoff altitude is 45 m.

6.3.4 Learning of the flocking model

The total training data set for all the iterations records at least 24 hours of flying time. The I2SL controller is implemented as a neural net, using mlpack (Curtin et al., 2013), while the linear algebra library is Armadillo (Sanderson, 2014). The neural architecture is a 2-hidden layers, with 256 neurons on each layer and Sigmoid as an activation function. The training uses Glorot initialization (Glorot and Bengio, 2010), with .5 Dropout and batch size 32; the hyper-parameters are adjusted using Adam (Kingma and Ba, 2014) with $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-8}$, and initial learning rate $\alpha = 0.001$.

6.4 Empirical validation

6.4.1 Zigzag experiment

The zigzag experiment aims to assess whether the followers can imitate the flocking behavior by following a complex trajectory that the pilot might execute. The experiment runs in two steps: first, all the quadrotors take off, the leader follows a random trajectory for 5 seconds, then the leader follows the zigzag trajectory that is already embedded in the leader. Second, The followers use the I2SL controller to follow the leader and avoid collision and dispersion. Note that the followers do not have any previous knowledge about their leader's trajectory. The experiment is exactly repeated in each iteration, allowing us to validate the performance I2SL controller in each iteration.

We compare the first and the second iteration in figures 6.1 6.2. In both iterations, the trained controller uses the data perceived by the wireless sensor. The first iteration uses the classic imitation learning algorithm, while the second iteration applies the DAGGER approach. The result 6.1 (left) shows the trajectory executed by each quadrotor, while the inter-quadrotor distances is shown in figure 6.2 (left). In the first iteration, we observe that the quadrotors learn how to follow the leader. However, they do not learn how to respect distances among them, resulting in several minor collisions between the quadrotors and distortion in the executed trajectory; this is confirmed in figure 6.2 for this experiment. Luckily, no aggressive collision has been

observed, which allowed us to complete the test without interruption. In the second iteration (right), quadrotors learn to avoid each other, but their behavior is very aggressive and not refined yet to be similar to the flocking model. Also, in this iteration, no collision was observed between the followers.

We continue to train the controller iteratively, resulting in a third iteration with a performance similar to the flocking model in figure 6.3. The controller uses the wireless sensor data, while the flocking model using the absolute position acting as ground truth for swarming behavior. We observe that the followers' quadrotors respect distances among each other similarly compared to the flocking model in figure 6.4.

The simulation video of the above experiments are available online through this link:

`https://tinyurl.com/7b2f7mcz`

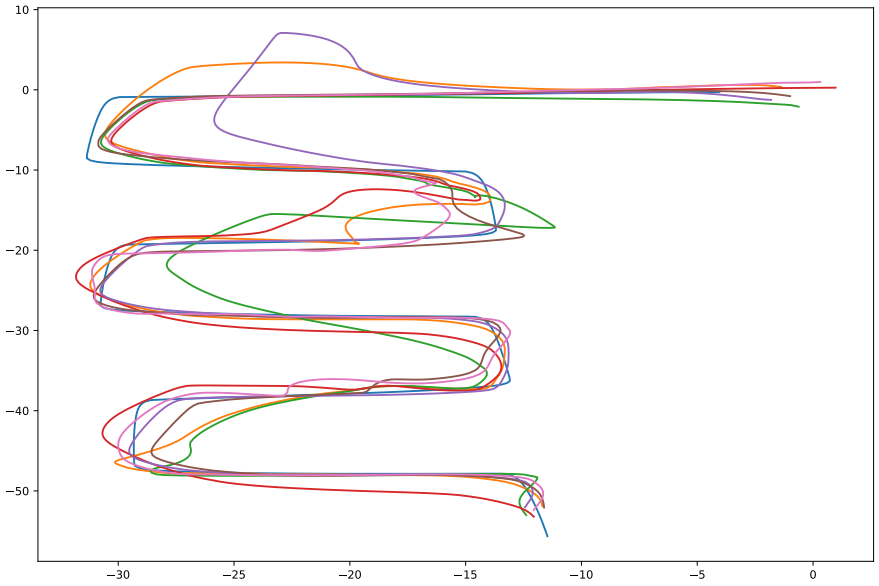
There are 4 videos, the first video `Flock_Zigzag` shows the swarm executing the flocking model 6.2.1 based on the absolute position. The second video `ZigZag_exp_iteration_1` shows the quadrotors using the basic I2SL controller in the first iteration. The third and fourth video `ZigZag_exp_iteration_2` (respectively. `ZigZag_exp_iteration_3`) shows the quadrotors using an improved version of I2SL in the second (resp. third) iteration.

6.5 Discussion

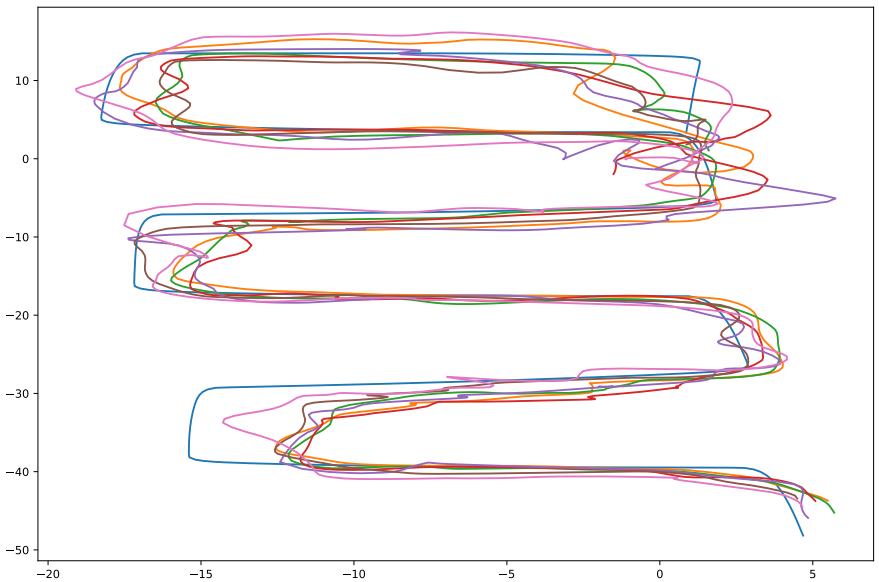
6.5.1 Wireless sensor

The wireless module that can be used is not limited to WiFi, but to any wireless devices that can be used with these robots such as Bluetooth, Zigbee, UWB. The device needs only to estimate the signal strength between the transmitter and the receiver robot successfully. Among these wireless tools, we have considered using WiFi for several reasons. First, most commercial quadrotors are already embedded with WiFi antennas, removing the cost of additional sensors on each robot. Second, the WiFi protocol is well known, and several available devices have open-source drivers such as Intel 5300, Qualcomm Atheros, or Esp32, allowing for easy manipulation and data extraction from the module.

The WiFi cards required to embed our controller on the robots are described as follows: controllers in chapter 4 and 5 requires only any basic integrated WiFi devices that can estimate the signal strength. For the controller presented in this chapter, the WiFi device needs to estimate the Angle

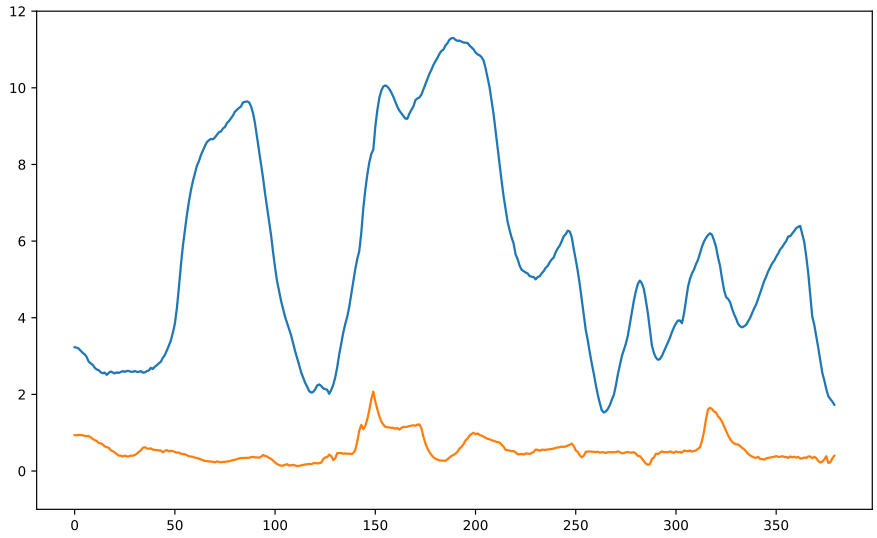


(a) Iteration 1 (sensor based): trajectories executed by all the quadrotors

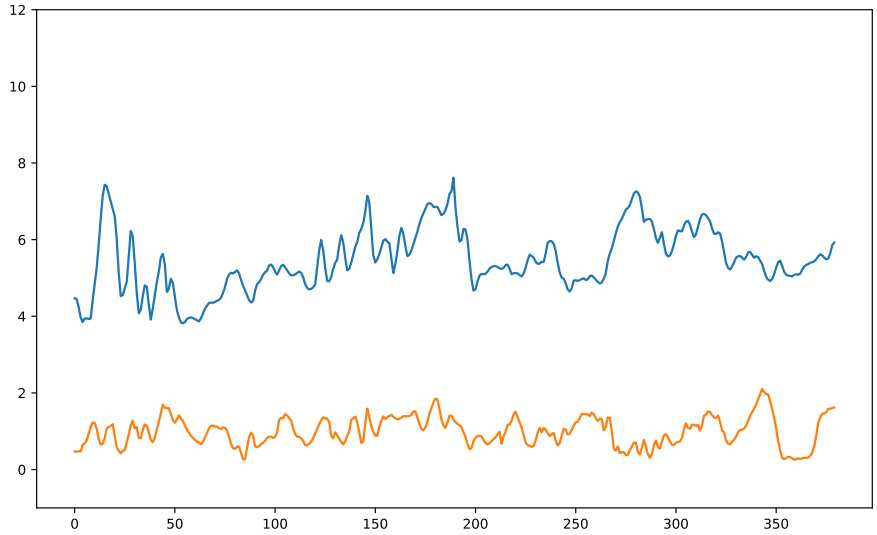


(b) Iteration 2 (sensor-based): trajectories executed by all the quadrotors

Figure 6.1.: compares the trajectories executed by quadrotors during the zigzag experiments in the first and second iteration. Quadrotors take off from the (0,0) coordinates and they land (at (11, -55), respectively (5, -44)) in the first (resp. second) iteration. The leader labeled in blue has integrated an embedded trajectory to simulate a human pilot, while all the followers use the learned controller. This figure shows an improved trajectory in the second iteration. This is due to the usage of the iterative learning over the basic imitation learning represented in the first iteration.

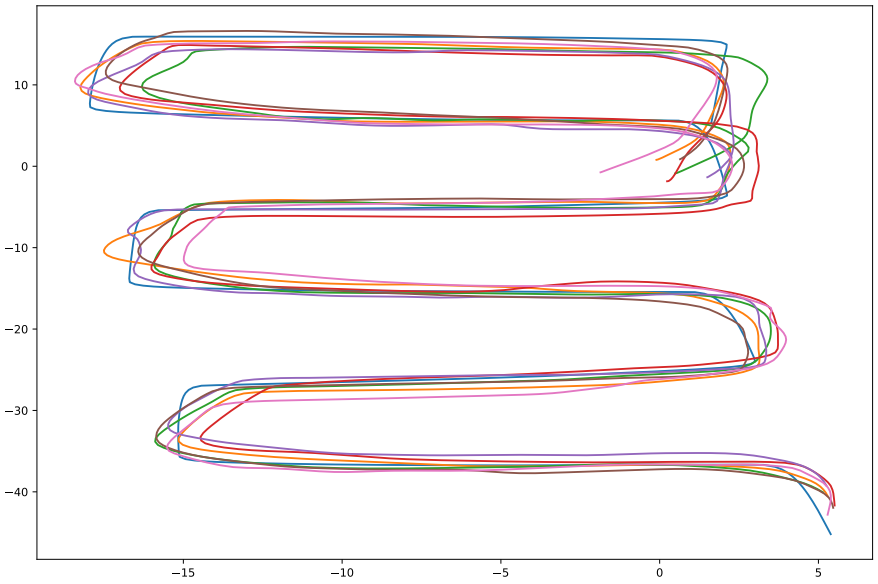


(a) Iteration 1 (sensors based): max and min inter-quadrotor distances

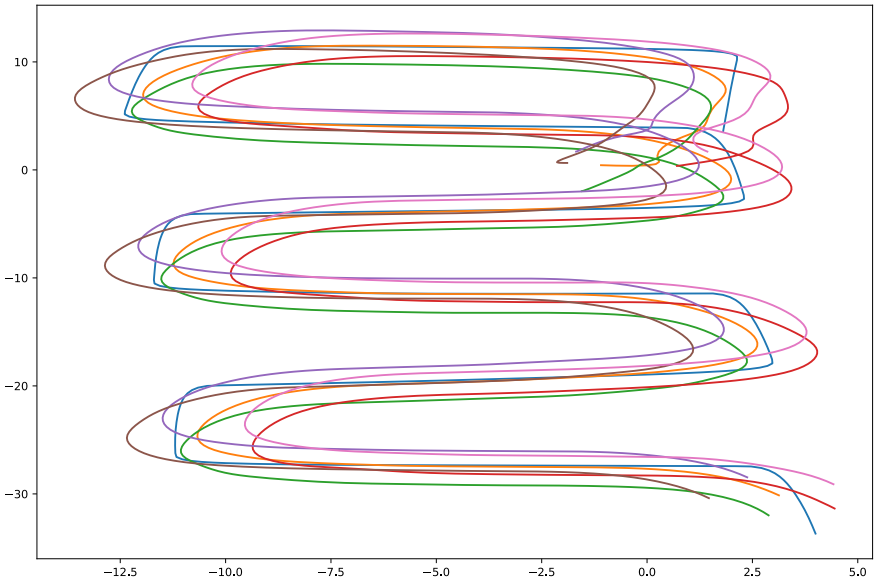


(b) Iteration 2 (sensor based): max and min inter-quadrotor distances

Figure 6.2.: shows the inter-quadrotors distances among the followers for both trajectories executed in iteration 1 (up) and iteration 2 (down). The blue line shows the maximum inter-quadrotor distances, while the orange one shows the minimum distance. We observe a considerable improvement in the second iteration compared to the first one, as the decentralized controller has learned the cohesion and separation policy in the second iteration. The quadrotors remain collision-free in the second iteration and do not disperse. Knowing that in the first iteration, we observe minor collisions, but none of these collisions were not critical, allowing us to complete the experiment.

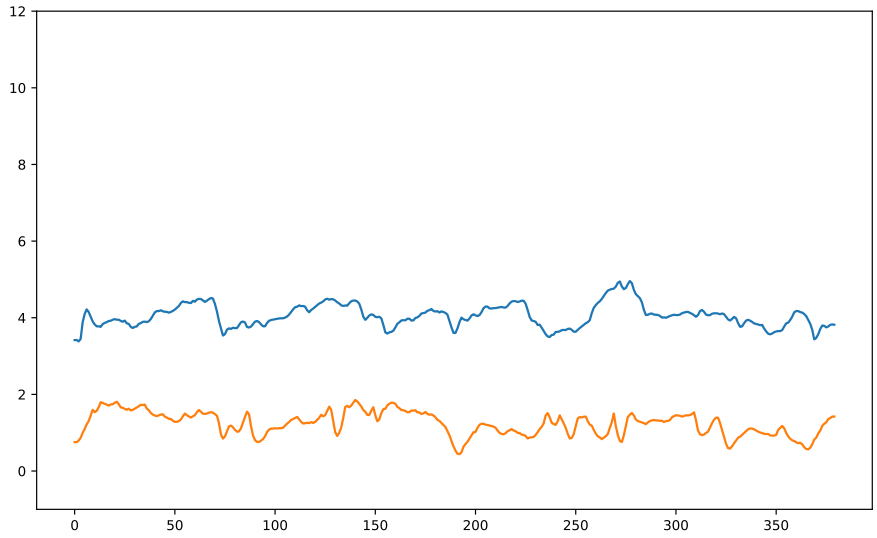


(a) Iteration 3 (sensor based): trajectories executed by all the quadrotors

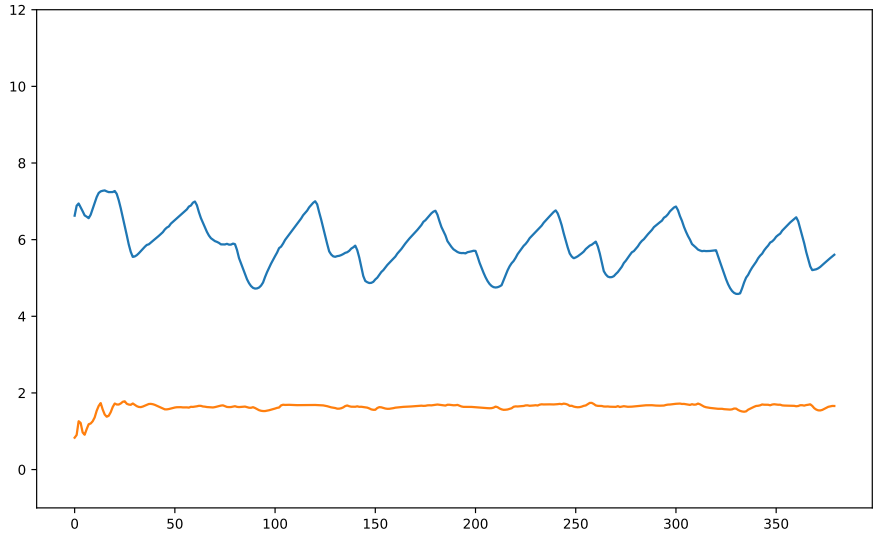


(b) Flocking model (position-based): trajectories executed by all the quadrotors

Figure 6.3.: shows the trajectory executed by the quadrotors in the third iteration (up) compared to the adapted Reynolds flocking model (down). The quadrotors start their trajectory at coordination (0,0) and end at (5, -44). The leader is labeled in blue in both cases. By comparing the two trajectories, we can observe a similar performance between the flocking model (position-based) and the third iteration of the learned controller (wireless sensors-based).

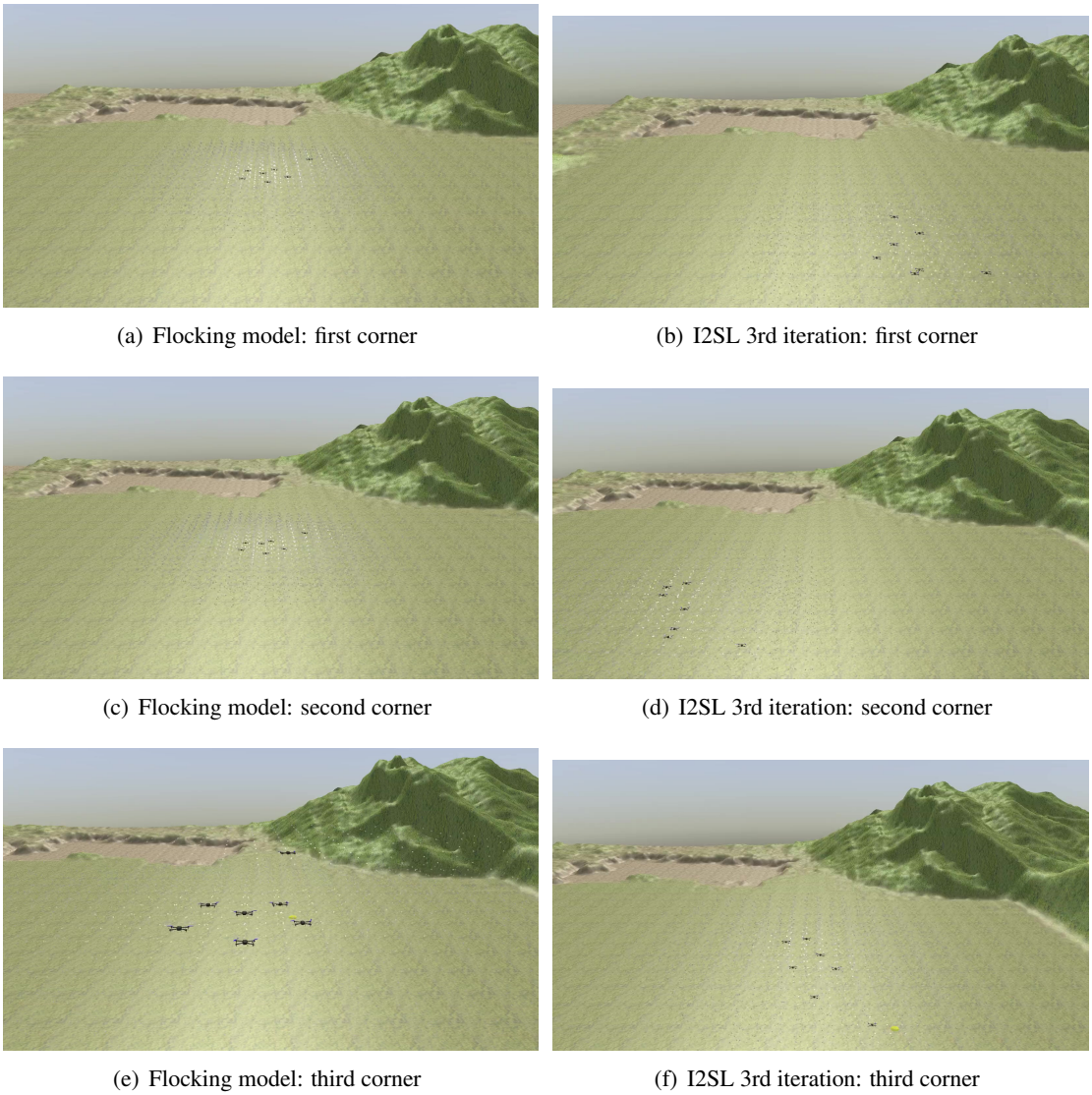


(a) Iteration 3 (sensor based): max and min inter-quadrotor distances



(b) Flocking model (position-based): max and min inter-quadrotor distances

Figure 6.4.: results of inter-agent distances when executing the zigzag trajectory by the quadrotors. The controller from the third iteration shows a similar performance compared to the flocking model. The quadrotors do not disperse nor collide with one another. These results show that the controller can be improved iteratively to achieve a performance compared to the flocking model.



(a) Flocking model: first corner

(b) I2SL 3rd iteration: first corner

(c) Flocking model: second corner

(d) I2SL 3rd iteration: second corner

(e) Flocking model: third corner

(f) I2SL 3rd iteration: third corner

Figure 6.5.: shows a set of images captured during the experiment of I2SL in the third iteration and the flocking model. The images are captured when the swarm traverse the southern corners. The leader is the quadrotor that is the most on the right side in all images. Note that while showing a different shape when using the I2SL controller the minimum and maximum distances between robots are very similar in both case.

of Arrival of the emitter (Niculescu and Nath, 2004; Paulraj et al., 1986) transmitter in addition to the signal strength.

Most generally, the method for AoA estimation on COTS WiFi cards are well detailed. First, we need to extract the Channel State Information (CSI) Halperin et al., 2011 since it contains the phase information and the signal strength for all Orthogonal Frequency Division Multiplexing (OFDM) subcarriers. Second, we need to analyze the signal phase since it suffers a shift and attenuation when the signal propagates in the environment. Finally, by analyzing this shift over all of these subcarriers using, for instance, the MUltiple SIgnal Classification (MUSIC) algorithm (Schmidt, 1979), one can easily deduce the AoA of the signal. This method can be used with a commodity WiFi card such as Intel 5300 since it has several antennas arranged to a uniform array.

6.6 Real-world communication on quadrotors

Several questions related to wireless communications might arise, especially about the received signal strength whether all quadrotors will be able to sense these signals from their neighbors or how much connection timeout can be observed. As we mentioned earlier, we did not realize an outdoor test scenario because of the lack of quadrotors hardware. However, a recent study (Vásárhelyi et al., 2018) that used Evolutionary Algorithms (EA) combined with flocking model on quadrotors. They have pushed their research beyond the simulation, successfully implementing their model on real outdoor quadrotors. Their method consists of using optimized parameters of Vicsek flocking model using CMA-ES (Hansen et al., 2003), then the model is embedded onboard and uses the position of neighbors provided by the GNSS sensors. Since it is necessary to exchange the position of neighbors. They installed two wireless modules on each robot. The first module is a Zigbee with a low transmitting rate and higher range, while the second module is a WiFi dongle with a higher rate and shorter range. Both of the modules set up an ad-hoc network and used it to share the GNSS information with everyone.

During the outdoor tests, the authors observed a communication shortage with far quadrotor (> 50) meters, while a near-perfect communication for close quadrotor (< 20) meters. Even when communications are perfect, they observed high packet loss between 40% to 80% at a small rate known that the transmission rate was low 10 Hz.

Indeed, the configuration of the authors (Vásárhelyi et al., 2018) does induce the high rate of packet loss for several reasons. First, all quadrotors are broadcasting User Datagram Protocol (UDP) packets on a WiFi medium simultaneously (Rohner et al., 2005) (Sheth et al., 2007).

Second, there is interference with the Zigbee module that is using the same frequency, (Musaloiu-E and Terzis, 2008) not to mention possible interference with the surrounding environment. Third, the robots are equipped with Odroid WiFi module 0 with a small range and a transmitting power. Several improvements can be applied to their system. The first and the second issues can be overcome by using a classical routing protocol related to ad-hoc networks such as OLSR (Clausen et al., 2003), or AODV (Perkins and Royer, 1999) in order to optimize message propagation throughout the network. The third issue can be resolved by using a high gain antenna connected to a WiFi card (Abedi et al., 2015) instead of using a dongle. This improvement reduces the interference generated by the Zigbee and substitutes the need for the module.

This study provides a real-world use case of combining ad-hoc networks with quadrotor swarms. We have discussed all limitations related to wireless networking, range, signal quality interference, and packet loss that the authors faced. We have also proposed a set of practical solutions that can improve communication. From a communication point of view, the authors did notice a minor reality gap between the simulation and reality, showing that real-life scenarios are more stochastic than observed in simulation. Therefore, in this thesis, we have followed the principles in terms of simulations. Each quadrotor broadcasts empty packets at a rate of 10 Hz to neighbors' quadrotors. The objective of these packets is to allows other quadrotors to analyze the signal strength of the neighbors.

6.7 Conclusion

In this chapter, we presented Iterative Imitation Supervised Learning (I2SL), an Iterative method used to resolve the challenge of decentralized controller design for a set of quadrotors with no computational power and endowed with a single wireless sensor. This method aims to resolve the optimization issue offline rather than during the flight and to learn the flocking behavior for follower quadrotors while following the remotely controlled leader. This approach demonstrated the feasibility of a leader-followers swarm using MagicFlock, a SITL simulation framework that is based on RotorS.

In the next chapter, we will present MagicFlock, the home-made simulation framework that we have used and conceived during this thesis.

MagicFlock simulation framework

7

Contents

7.1 Introduction

In the last decade, quadrotors have become major robots used by several industries, with a wide range of applications that ranges from search and rescue (Tomic et al., 2012), navigation (indoor, and outdoor) (Liu et al., 2016), cinematography (Joubert et al., 2016, 2015), entertainment (Ohta, 2017), to building construction (Lindsey et al., 2011b). Most of these applications can be better achieved collectively by using several quadrotors simultaneously to do a specific task (Gassner et al., 2017; Pizetta et al., 2016). This solution is possible today as quadrotors hardware is becoming more and more affordable to everyone. However, a significant issue is the development of high-level algorithms such as coordination of several autonomous quadrotors to achieve

Figure 7.1.: A screenshot of the simulation environment from Gazebo simulator when spawning several quadrotors. The robots are initiated randomly in the environment.

the desired behavior. To develop these algorithms, researchers and engineers can set up real quadrotors' hardware and test these algorithms directly (Michael et al., 2010). However, the effort required by these engineers can be time-consuming. Engineers need to be trained and have a specific pilot certificate to fly a quadrotor, not to mention the potentially damaging results in hardware and infrastructure. In addition, researchers working closely with real hardware need to follow specific safety protocols (Mulgaonkar, 2019; Pounds and Mahony, 2009) each time a test is initiated. Following these steps in day-to-day life can be counterproductive. Therefore, simulation can be a better choice for rapid prototyping and development for high-level algorithms. The validation of these algorithms is ensured as long as the simulation framework tries to reduce the reality gap as possible (Golemo et al., 2018; Pitonakova et al., 2018).

In this work, we propose MagicFlock a simulation framework for several quadrotors. This framework developed gradually during the thesis to obtain research results observed in chapter 5 and 6. The objective of this framework is to address the issue of reality gap simulation of Multi-agent System. Contrary to existing simulation frameworks, the objective of MagicFlock is to achieve accurate simulation for multi-agent systems while keeping a simple interface for non-experienced users.

7.2 Related work

Most of existing 3D robotics simulators such as Webots (Michel, 2004), Gazebo (Koenig and Howard, 2004) and AirSim (Shah et al., 2017) simulate the physiques and the dynamics of mechanical structures. These simulators support several physiques engines such as Bullet (Coumans and Bai, 2016–2019) or Dart (Lee et al., 2018), and uses 3D realistic rendering engines. The objective of these simulators is to avoid the cost of real robot tests by making simulations as close as possible to reality. These simulators provide ready-to-use robotics models, such as quadrotors, rovers, UAV, humanoid robots, or autonomous cars. We chose Gazebo for several reasons; first, Gazebo exists as an open-source project for two decades, contrary to WeBot. It provides extensive documentation and tutorials. Also, Gazebo uses Ogre as a rendering engine, which requires less computational power than AirSim since it uses RealEngine. Second, Gazebo allows the integration of external modules through their plugin system. This feature allowed the development of RotorS, a Software In The Loop (SITL) simulation framework proposed by (Furrer et al., 2016) for multirotor. Their research aims to validate their quadrotor's flight controller software behavior, such as flight dynamics in the Gazebo simulator. This method simulates diverse aspects such as onboard sensors or complex environments such as winds, obstacles, and dynamic objects. The last version of rotorS [1] does not have a dependency on Robot Operating System (ROS) allowing for a more straightforward installation process.

One interesting framework developed by Meyer et al., 2012 uses Gazebo and ROS to allow the simulation of flight dynamics and sensors. However, this platform allows the simulation for only one quadrotor at a time. Also, it has a dependency on ROS making it harder for users to install and maintain.

Most of the existing solutions for one quadrotor simulation uses Matlab or Simulink (Bouabdallah and Siegwart, 2007; Mester, 2011), these frameworks usually support basic simulation for flight dynamics, but does not support the simulation of sensors and require users to learn the Matlab programming language. The only existing solution created by Soria et al., 2020 allows the simulation of several quadrotors using MatLab with no physiques-enabled environments.

Our simulation framework extends the work in (Furrer et al., 2016). The objective of our work is to use the SITL method, Gazebo simulator, and to extend it to support the simulation of several quadrotors at the same time, by providing the average user an easy-to-use interface.

[1] https://github.com/ethz-asl/rotors_simulator

7.3 Package overview

MagicFlock has several features and methods that are available to the users, with a set of examples and documentation to ensure a good understanding of the library:

- Flocking models such as Reynolds model, Adapted Reynolds model, Random model.
- IL4MRC algorithm and its components such as state, action predictors.
- Machine learning interface using mlpack, and linear algebra interface through Armadillo.
- Automatic startup for Gazebo and spawn for quadrotors.
- Automatic dataset generation, random quadrotor spawning using circle packing.
- True terminal logging and registered logging to ease debugging.
- Python script to generate plots.
- Easy to use interface to add and remove sensors.
- Control of a specific quadrotor using a keyboard, Joystick input.

MagicFlock is entirely open source. The source code is published on GitHub with a separate repository for documentation that is open source too. GitHub allows tracking bugs using the issues interface and features adding using pull requests. Currently, MagicFlock has no major release yet, and it is only available on Linux. The installation process requires the compilation of the source code. There is no package available directly to be included in the mainstream Linux distributions package manager.

7.4 Software architecture

MagicFlock is written in C++ for several reasons. Firstly, C++ is widely popular and taught in most universities. It is used in research laboratories and industry, thus allowing higher chances of adaptation, contributions, and bug fixes. Secondly, most if not all the simulators, physics engines, flight controllers, Robot Operating System (ROS), are written in C++, allowing the possibility for the extension of MagicFlock by incorporating features from different frameworks or integrating MagicFlock directly into other frameworks. Thirdly, MagicFlock uses C++17 has an extensive set of features that facilitate the use of the language.

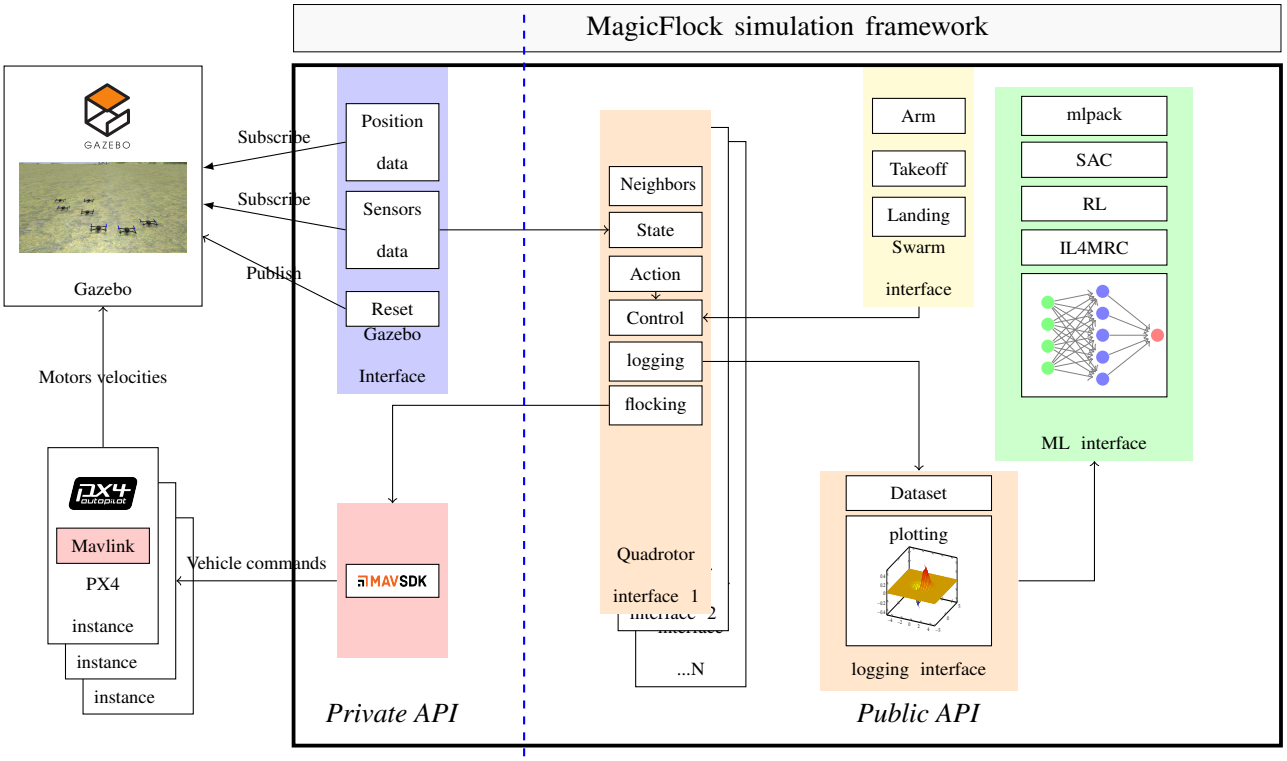


Figure 7.2.: This figure represents the software architecture of MagicFlock. On the left side, the autopilot controller (PX4) is started by RotorS. RotorS manages the entire communications between the Gazebo simulator and autopilots. MagicFlock connects to autopilots by using MAVSDK a wrapper library for the MAVLINK protocol and communicates with Gazebo through their publish-subscribe system. MagicFlock has a public, stable, easy-to-use API, allowing the users to use it directly to create simulation examples and tests. It hides the complexities from the users by creating a wrapper around the private API, which is used to manage the communication parts from inside and outside MagicFlock.

MagicFlock uses the Object-Oriented Programming (OOP) paradigm the entire software is divided into a public Application Programming Interface (API), and a private API. Public API classes provide the user with easy-to-use functions, examples, abstract classes, callbacks, and header-only functions. While the objective of the private API is to hide the implementation complexity of the users.

MagicFlock uses generic and meta-programming techniques; this allows the users to add interior features to the library without the need for a major change of the public API. The templates are used in order to provide a significant abstraction. For instance, if the user wants to use a different flight controller or a different communication protocol for quadrotors, the only requirement is to create a simple class that communicates with the simulation entity. The same principle is used with the simulator; users can use Webot, AirSim without any need for significant modifications. In addition, templates usage increases the speed of the software since the instantiation is done at compilation time rather than the run time.

7.4.1 Agent robot

MagicFlock currently supports quadrotors UAV, anyone can add or change the robot type by modifying the model files. The quadrotor we used in the simulation is the default robot provided by RotorS. It is the IRIS quadrotor and described in figure 7.3 designed by 3DR[2], with height 0.10 m, width 0.47 m, and weight 1.3 kg. It has a payload capacity of 0.4 kg and 0.55 m of the motor to motor dimension. The motors have a constant velocity of 920 kV and 10 inches propellers with a pitch of 4.7 inches, with a maximum velocity of 11 m/s.

The robot designed in the simulation has an identical specification to the real robot including the body colors, sensor placement, and camera mounting if required.

7.4.2 Micro Aerial Vehicles Software Development Kit (MAVSDK)

In order to communicate with quadrotors inside the Gazebo simulator, MagicFlock rely on MAVSDK, a simple library designed to provide communication with quadrotors that implements the Micro Aerial Vehicles LINK (MAVLINK) protocol. MAVSDK has several features and allows blocking and nonblocking functions to be called by implementing callbacks. Since we are relying on the SITL concept, it is possible to use MAVSDK locally and communicating with the flight controllers. To reduce complexity, we are using the same flight controller PX4 used

[2] https://3dr.com/

Figure 7.3.: The IRIS quadrotor fabricated by 3DR and used in MagicFlock simulation platform. The open-source design of the quadrotor and the usage of open-source hardware and software allowed the possible integration of this robot in Gazebo simulator.

by RotorS, even though, MAVSDK can be used to communicate with any vehicle that sends and receive MAVLINK packets whether if it has the PX4 autopilot onboard or not. MagicFlock provides a wrapper class that allows access to most of MAVSDK functions. The objective of this wrapper is to have a compatible API with the Quadrotor class, allowing the users to use a different autopilot that does not implement the MAVLINK protocol by adding another wrapper that needs to be compatible with quadrotors class API.

7.4.3 Quadrotor

The Quadrotor class is the main point of control and communication with robots. This class provides the user with several parameters such as:

- ID, name, and a label for each quadrotor.
- Generic interface to state and action spaces, collect data set, and noise and filter type.
- Generic interface to access any available autopilot.
- High-level sampling functions, and swarming models.

During the instantiation of the quadrotor class, the user needs to define the above parameters one by one, either through a template interface or using a constructor signature. The entire set of quadrotors can be assigned to a swarm class that has a set of standard high-level functionalities for the swarm such as (`arm, takeoff, land, disarm`)

7.4.4 Register trajectories as States and Actions

Since MagicFlock is intended to be used with imitation and Reinforcement Learning algorithms, we have developed the State class to be easily compatible with these methods, the state of quadrotors is not specified in advance, instead the users can define any class locally and use it as a state for quadrotors. For example, In this work, we usually defined the state as the received signal strength from other quadrotors. We have added wireless modules on each quadrotor using Gazebo, and then recover these values and register them locally as states. For the actions, we define two classes: `DiscretActions`, and `ContinuousActions`, The `DiscretActions` are the 7 actions, while the `ContinuousActions` are the velocity vector on (x, y, z)

7.4.5 Examples

In order to use MagicFlock a set of examples are described in the directory `examples/` in the source code. However, there are two major classes from the public API that are required to make the simulation usable. The quadrotor class and the swarm class. The second class provides a wrapper on high-level functionalities that can be executed collectively on a set of quadrotors, such as takeoff or landing. Users can define the number of quadrotors required to do the simulation on command lines, and Gazebo will generate these quadrotors without manual intervention. The quadrotors are generated on one line and separated by a distance of 1 meter. Users can then create any user-defined class that implements the desired functionalities to run and test on quadrotors. In the example directory, we provide a `generate_data_set` example, which gives an idea for the user how to use MagicFlock library to generate data set. Similarly, the `iterative_learning` example describes how to use the MagicFlock algorithm on the followers. The flocking example (`flocking_model`) describes how to run any flocking model on quadrotors with or without a leader.

7.4.6 Sensor information

Senors allow quadrotors to perceive their environments. MagicFlock uses Gazebo publish and subscribe systems to recover sensor values to communicate with the simulator in real-time. Gazebo has a set of sensors such as rays that can be used to construct higher-level ones such as LIDAR, ultrasound, and wireless sensors. If none of the available can accomplish the objective, users can implement their sensors in Gazebo by using the plugin systems. In MagicFlock we already use the provided sensors; this can be done by including the relative information of each

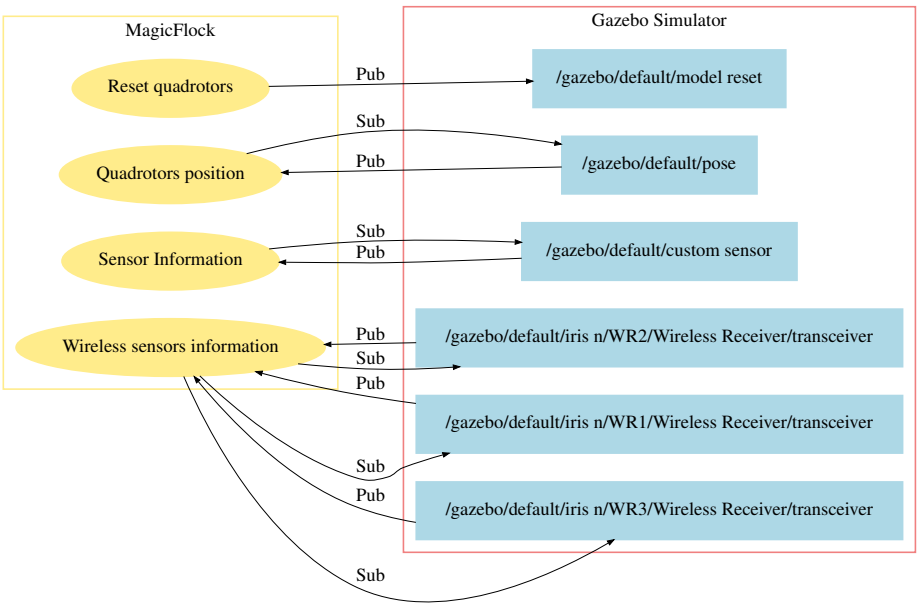


Figure 7.4.: shows the publish-subscribe architecture between MagicFlock and Gazebo. MagicFlock subscribe to topic published periodically by the simulator in order to receive robot's state in real time such as position of the robot, and sensor information. In addition, it is possible to send direct commands to Gazebo using plugins. For instance, the home-made reset plugin allow the entire swarm to their initial place.

sensor in the descriptive model file. This file is written in Gazebo format to describe a robot which is called Simulation Description Format (SDF). SDF provides an easy way to describe robots since it is based on eXtensible Markup Language (XML). Gazebo uses it to describe worlds, robots, sensors, and joints used to connect robots with sensors and other mechanical parts. Gazebo website provides extensive documentation about SDF and how to use it to create your world. Once sensors have been joined to robots, they start to publish their information into a message queue. MagicFlock subscribes to each of the sensors' channels and recovers the message; the following figure 7.4 describes the publish-subscribe process between Gazebo and MagicFlock.

MagicFlock created a Gazebo plugin called the ResetPlugin to provide the reset functionalities for all the quadrotors at the end of each episode. This plugin allows resetting all the quadrotors either to their original places or to random places generated randomly using the circle packing algorithm to avoid any possible overlap between quadrotors. This feature is exciting in reinforcement learning since the reset functionality is an essential step at the end of each episode.

7.5 Flocking and swarm algorithms

MagicFlock contains several implementations of flocking models; we have implemented Reynolds model (Reynolds, 1987), another adaptation from this work (Schilling et al., 2018), and extended by our leader-follower extension described in chapter 6. The extension adds a leader-follower principle and considers a dynamic migration point which is the leader, while followers use the Reynolds model to follow the leader.

7.6 Machine learning

Using mlpack (Curtin et al., 2013), MagicFlock provides an interface to use machine learning algorithms on a swarm of quadrotors. mlpack is entirely compatible with MagicFlock . It uses generic programming techniques and requires a small number of dependencies during the installation. In addition, mlpack is fast and compatible with embedded systems, allowing users to transfer the learned model from the simulator to real-world quadrotors without re-training. In previous work (Shrit et al., 2021), we have integrated the neural network model and k nearest neighbor models into a set of quadrotors. Each quadrotor has its model and uses it to predict the following states or actions to achieve the flocking behaviors.

7.7 Computational parameters

In this thesis, we have demonstrated some algorithms implemented in MagicFlock we recall the tests scenarios with a specific number of quadrotors in the swarm. In a first scenario, we presented IL4MRC controller applied directly on the entire swarm, two settings were used, with three (respectively. four) robots were used in the first (reps. second settings). In a second scenario, we present the same Reynolds flocking model using the leader-follower principle in which the followers' quadrotors have a specific destination which is the leader, and the I2SL. In both settings, 7 quadrotors were used.

We run several simulation tests with a set of different numbers of quadrotors in the swarm. The simulations are executed on a Dell Precision Tower with 1.7 GHz Intel Xeon E5-2609 with 16 cores inside and 32 GB of RAM, the GPU is an Nvidia Quadro k420. The real-time factor is determined by Gazebo, which is equal to 1.

7.8 Future Plans and conclusion

MagicFlock is far from being completed. Several flocking algorithms can be added, such as Olfati-Saber algorithms or Viscek model. This adding can be a nice feature to add either by contributors or the open-source community of users.

Further, by making MagicFlock open sources, new users can add features and fix bugs In the future, we are intended to have the first release of MagicFlock this is particularly possible when the public API is stable and requires rare changes.

Conclusion and Perspectives

8

Contents

8.1 Conclusion on thesis contributions

In this thesis, we have proposed a learning-based approach to the problem of decentralized swarm controller for cheap COTS quadrotors. Initially, a general state-of-the-art investigation was proposed related to perception and control (Chapter 2) and a more in-depth literature study related to quadrotors swarms (Chapter 3). In addition, we presented preliminary results related to the basic swarm controller (Chapter 4). In the following, we recall on thesis contributions presented in Part II:

Iterative Learning for Model Reactive Control (IL4MRC) is an iterative learning-based approach that is specifically designed to address the challenge of the decentralized controller. The objective of IL4MRC is to preserve the initial swarm property of the system by finding the optimal action to execute in the next time step. IL4MRC uses only the distances between the quadrotors that are converted from the WiFi received signal strength. IL4MRC uses a random model to generate a data set in the simulator. The data set is used to train a forward model, then use this model to find the optimal sequence of actions. The advantage of IL4MRC over traditional methods such as MPC is to shift the burden of optimization from the online real-time test flight to the generation of data set.

Iterative Imitation Supervised Learning (I2SL) is an iterative learning approach based on imitation learning that presents an improvement over IL4MRC in terms of the executed policy. I2SL allows a set of followers quadrotor to swarm around the leader by using the same decentralized swarm controller that finds the optimal action to execute. Similarly to IL4MRC, I2SL improves the swarm controller iteratively in each iteration in order to reach the same behavior of the oracle demonstrator (flocking model). The first iteration I2SL uses the flocking model on each follower to generate a data set while following the leader. The dataset is used to train a controller that allows enforcing following the leader effectively. In the second iteration, the controller learned recently is used and altered with the oracle demonstrator following the DAGGER approach to generate the data set. The data set is merged with the former and used to train an improved controller that enforces the flocking behavior. Iteratively, the second iteration is repeated until having a decentralized controller that behaves like the flocking model. I2SL presents similar advantages to IL4MRC by reducing the required computation for the swarming behavior and having only one WiFi module that is used to perceive the neighbors. In addition to providing a proof of concept that imitation learning methods can be applied effectively on a Multi-agent System.

MagicFlock is a home-based simulation framework that is based on RotorS which provides a Software In The Loop (SITL) simulation for a set of quadrotors instead of one robot. The quadrotors autopilot software is used and integrated with a high-fidelity physics' simulator (Gazebo). The objective of SITL is to provide the same system dynamic for robots inside the simulator as the one observed in reality in order to cover the shortage of robots' hardware. Both of the proposed methods IL4MRC and I2SL have been designed and validated using MagicFlock. MagicFlock implementation is currently open-source and published in GitHub through this link: `https://github.com/shrit/MagicFlock`. The primary advantage of MagicFlock is to allow researchers to accelerate the development of quadrotor swarms inside the Gazebo simulator, knowing that the simulated behavior has a small gap with reality.

8.2 Perspective

The thesis subject can lead directly to real-world applications. In this section, we discuss issues and barriers that can be alleviated in the future the ease the transition from research to the industry, since it is much harder due to the complexity of real-world scenarios. In the following, we discuss these points as follows:

- **Use of Generalized feature embedding instead of fixed size state vector.**

Currently, the trained models in chapters 5 and 6 depend on the fixed size of the state vector, as the state of each quadrotor is related to the number of neighbors in the swarm. However, if the number of neighbors changes, we need to restart the entire process from the generation of the data set, training to the validation of the model. However, by using the generalized feature embedding, the state of each follower quadrotor will no longer depend on the number of neighbors but will be already is converted to a histogram. This improvement will allow us to generate a vast number of data sets even if the swarm's number of quadrotors is changing. The model will be updated, but it will be the same independent of the size of the swarm.

- **Add obstacle avoidance to quadrotors.**

This will only be possible if all the obstacles have an embedded wireless sensor that allows them to calculate the distance and angle. The obstacles can be static or dynamic. In this scenario, Olfati-Saber can be used to generate data set instead of Reynolds model since it considers the possibilities to avoid both types of obstacles. By recovering a considerable set of training data, this approach should achieve obstacle avoidance when using only one wireless sensor, allowing the swarms to be compatible with indoor and outdoor flights.

- **Transfer the learned models from simulation to real robots.**

The training set is collected from a high fidelity physics simulator (e.g., Gazebo), which means that the robot's behavior achieved inside the simulator should be easily transferable to a real-world situation. However, there are several parameters to take into consideration, such as the robot type (frame size, memory, sensors noise, or computation power), the size footprint of trained models, the wireless interference that can have a higher effect in reality than in simulation. Since we have provided a real-world demonstration in chapter 4 for a basic one dimension scenario, we are keen on transferring these models into a real robots and provide at least a proof of concept.

- **Valorize MagicFlock as a state-of-the-art simulation for quadrotors swarms.**

Our results in chapters 5, and 6 have been validated due to the simulation framework that we have developed during this thesis. The framework allows simulating a set of quadrotors using SITL with a simple set of instructions that any researcher can execute. However, MagicFlock has been publicly available only starting from 2021 making it practically unknown to the robotic community. Therefore, we have decided to write a scientific paper to be published in an international conference or a journal dedicated to simulation software to increase the visibility of this

framework. In addition, several improvements and implementations can be incorporated inside MagicFlock to enhance the user experience, such as interactive documentation and a dedicated website.

8.3 Final subjective point of view

No one can foresee the future, prediction always misses the reality, but it helps to shape it. Nowadays, quadrotors swarm is still an emergent field, and most approaches and methods are within the research domain and moving slowly to industrial applications. Currently, the transition towards these applications is slow because of legislation and public acceptance issues. For example, cars evolved to become more and more autonomous. However, when related to their usage, we can observe many refrains from the global community of users and citizens pointing out a set of conundrums. For instance, when it comes to safety, users regularly bring up the Trolley problem (Foot, 1967), in which the autonomous car has to make an ethical and psychological decision instead of the driver. The dilemma consists of choosing between one person or several persons found on the road, and the car has to choose to collide and kill either one or several people.

When it comes to drones, MAVs, and quadrotors. Users often bring up privacy issues related to surveillance and private aerial space. It is simple to issue a law that authorizes non-flight zone overall private properties. However, how to make sure that an autonomous quadrotor is going to respect all these areas?

This question becomes more complex when it comes to swarms; it is not only one robot that needs to be autonomous but instead a fleet of autonomous robots that collaborate to resolve a mission. Therefore, legislation might require time to evolve toward this direction and resolve all moral questions that are still currently open.

Bibliography

Abbeel, P., A. Coates, M. Quigley, and A. Ng (2006). "An Application of Reinforcement Learning to Aerobatic Helicopter Flight." In: *NIPS* (cit. on pp. 38, 39).

Abbeel, Pieter and Andrew Y Ng (2004). "Apprenticeship learning via inverse reinforcement learning." In: *Proceedings of the twenty-first international conference on Machine learning*, p. 1 (cit. on p. 26).

Abedi, Naeim, Ashish Bhaskar, Edward Chung, and Marc Miska (2015). "Assessment of antenna characteristic effects on pedestrian and cyclists travel-time estimation based on Bluetooth and WiFi MAC addresses." In: *Transportation Research Part C: Emerging Technologies* 60, pp. 124–141 (cit. on p. 98).

Adams, Julie A, Curtis M Humphrey, Michael A Goodrich, et al. (2009). "Cognitive task analysis for developing unmanned aerial vehicle wilderness search support." In: *Journal of cognitive engineering and decision making* 3.1, pp. 1–26 (cit. on p. 12).

Adams, Martin D (2000). "Lidar design, use, and calibration concepts for correct environmental detection." In: *IEEE Transactions on Robotics and Automation* 16.6, pp. 753–761 (cit. on p. 21).

Ahmad, Norhafizan, Raja Ariffin Raja Ghazilla, Nazirah M Khairi, and Vijayabaskar Kasi (2013). "Reviews on various inertial measurement unit (IMU) sensor applications." In: *International Journal of Signal Processing Systems* 1.2, pp. 256–262 (cit. on p. 20).

Al-Kaff, Abdulla, David Martin, Fernando Garcia, Arturo de la Escalera, and José María Armingol (2018). "Survey of computer vision algorithms and applications for unmanned aerial vehicles." In: *Expert Systems with Applications* 92, pp. 447–463 (cit. on p. 7).

Alcántara, Alfonso, Jesús Capitán, Rita Cunha, and Aníbal Ollero (2021). "Optimal trajectory planning for cinematography with multiple unmanned aerial vehicles." In: *Robotics and Autonomous Systems* 140, p. 103778 (cit. on p. 11).

Alexis, K., G. Nikolakopoulos, and A. Tzes (2012). "Model predictive quadrotor control: attitude, altitude and position experimental studies." In: *IET Control Theory Applications* 6.12, pp. 1812–1827 (cit. on p. 37).

Allen, Grant, Peter Hollingsworth, Khristopher Kabbabe, et al. (2019). "The development and trial of an unmanned aerial system for the measurement of methane flux from landfill and greenhouse gas emission hotspots." In: *Waste Management* 87, pp. 883–892 (cit. on p. 11).

Almers, Peter, Ernst Bonek, A Burr, et al. (2007). "Survey of channel and radio propagation models for wireless MIMO systems." In: *EURASIP Journal on Wireless Communications and Networking* 2007, pp. 1–19 (cit. on p. 21).

Anderson, Seth B. (1981). *Historical Overview of V/STOL Aircraft Technology*. Tech. rep. NASA (cit. on p. 31).

Archives Centrale-Histoire (Jan. 2019). http://archives-histoire.centraliens.net/ (cit. on p. 31).

Ardupilot (2018). http://ardupilot.org// (cit. on p. 31).

Augugliaro, F., A. P. Schoellig, and R. D'Andrea (2012). "Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach." In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1917–1922 (cit. on pp. 37, 41).

Baca, T., G. Loianno, and M. Saska (2016). "Embedded model predictive control of unmanned micro aerial vehicles." In: *2016 21st International Conference on Methods and Models in Automation and Robotics (MMAR)*, pp. 992–997 (cit. on pp. 37, 41).

Bacelar, T., C. Cardeira, and P. Oliveira (2019). "Cooperative Load Transportation with Quadrotors." In: *2019 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pp. 1–6 (cit. on p. 12).

Bagnell, J Andrew and Jeff G Schneider (2001). "Autonomous helicopter control using reinforcement learning policy search methods." In: *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164)*. Vol. 2. IEEE, pp. 1615–1620 (cit. on p. 39).

Bahl, P. and V. N. Padmanabhan (2000). "RADAR: an in-building RF-based user location and tracking system." In: *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064)*. Vol. 2, 775–784 vol.2 (cit. on p. 46).

Bassma, Guermah, Sadiki Tayeb, et al. (2018). "Support vector machines for improving vehicle localization in urban canyons." In: *MATEC Web of Conferences*. Vol. 200. EDP Sciences, p. 00004 (cit. on p. 80).

Bemporad, Alberto and Davide Barcelli (2010). "Decentralized model predictive control." In: *Networked control systems*. Springer, pp. 149–178 (cit. on p. 23).

Berman, Elena SF, Matthew Fladeland, Jimmy Liem, Richard Kolyer, and Manish Gupta (2012). "Greenhouse gas analyzer for measurements of carbon dioxide, methane, and water vapor aboard an unmanned aerial vehicle." In: *Sensors and Actuators B: Chemical* 169, pp. 128–135 (cit. on p. 11).

Bernard, Markus, Konstantin Kondak, Ivan Maza, and Anibal Ollero (2011). "Autonomous transportation and deployment with aerial robots for search and rescue missions." In: *Journal of Field Robotics* 28.6, pp. 914–931 (cit. on p. 43).

Bhattacharyya, Raunak P, Derek J Phillips, Blake Wulfe, et al. (2018). "Multi-agent imitation learning for driving simulation." In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 1534–1539 (cit. on p. 81).

Bithas, P., E. T. Michailidis, Nikolaos Nomikos, D. Vouyioukas, and A. Kanatas (2019). "A Survey on Machine-Learning Techniques for UAV-Based Communications." In: *Sensors (Basel, Switzerland)* 19 (cit. on p. 11).

Boids (June 2021). https://www.red3d.com/cwr/boids/ (cit. on p. 33).

Bollard-Breen, Barbara, John D Brooks, Matthew RL Jones, et al. (2015). "Application of an unmanned aerial vehicle in spatial mapping of terrestrial biology and human disturbance in the McMurdo Dry Valleys, East Antarctica." In: *Polar biology* 38.4, pp. 573–578 (cit. on p. 11).

Boole, George (1854). *An investigation of the laws of thought: on which are founded the mathematical theories of logic and probabilities*. Vol. 2. Walton and Maberly (cit. on p. 5).

– (1957). *The Laws of Thought*. Dover, New York (original edition 1854) (cit. on p. 5).

– (1847). *The mathematical analysis of logic*. Philosophical Library (cit. on p. 5).

Boost C++ Libraries (May 2021). `http://www.boost.org` (cit. on p. 52).

Boston Dynamics (June 2021). `https://www.bostondynamics.com` (cit. on p. 6).

Bothezat, George de (1920). *The General Theory of Blade and Screws*. Tech. rep. (cit. on p. 31).

Bouabdallah, S. and R. Siegwart (2007). "Full control of a quadrotor." In: *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 153–158 (cit. on p. 101).

Brescianini, D., M. Hehn, and R. D'Andrea (2013). "Quadrocopter pole acrobatics." In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3472–3479 (cit. on p. 12).

Carbone, C., Oscar D. Garibaldi, and Z. Kurt (2018). "Swarm Robotics as a Solution to Crops Inspection for Precision Agriculture." In: (cit. on p. 10).

Cárdenas, Elsa, Pedro Boschetti, Andréa Amerio, and Carlos Velasquez (2005). "Design of an unmanned aerial vehicle for ecological conservation." In: *Infotech@ Aerospace*, p. 7056 (cit. on p. 11).

Cardona, GA, D Tellez-Castro, and E Mojica-Nava (2019). "Cooperative Transportation of a Cable-Suspended Load by Multiple Quadrotors." In: *IFAC-PapersOnLine* 52.20, pp. 145–150 (cit. on p. 12).

Carrillo, Luis Rodolfo García, Alejandro Enrique Dzul López, Rogelio Lozano, and Claude Pégard (2013). "Modeling the quad-rotor mini-rotorcraft." In: *Quad Rotorcraft Control*. Springer, pp. 23–34 (cit. on p. 30).

Cerro, Jaime del, Christyan Cruz Ulloa, Antonio Barrientos, and Jorge de León Rivas (2021). "Unmanned Aerial Vehicles in Agriculture: A Survey." In: *Agronomy* 11.2, p. 203 (cit. on p. 10).

Cheng, Teddy M. and Andrey V. Savkin (2011). "Decentralized control of multi-agent systems for swarming with a given geometric pattern." In: 61.4, pp. 731 –744 (cit. on p. 37).

Chintalapudi, Krishna, Anand Padmanabha Iyer, and Venkata N. Padmanabhan (2010). "Indoor Localization without the Pain." In: *Proceedings of the Sixteenth Annual International Conference on Mobile Computing and Networking*. MobiCom '10. Association for Computing Machinery, 173–184 (cit. on p. 46).

Choi, Seungwon, Suseong Kim, and H Jin Kim (2017). "Inverse reinforcement learning control for trajectory tracking of a multirotor UAV." In: *International Journal of Control, Automation and Systems* 15.4, pp. 1826–1834 (cit. on p. 27).

Christopoulos, Vassilios and Paul Schrater (2010). “Learning reward functions in grasping objects with position uncertainty via inverse reinforcement learning.” In: *Journal of Vision* 10.7, pp. 1084–1084 (cit. on p. 27).

Chung, Soon-Jo, Aditya Avinash Paranjape, Philip Dames, Shaojie Shen, and Vijay Kumar (2018). “A Survey on Aerial Swarm Robotics.” In: *IEEE Transactions on Robotics* 34.4, pp. 837–855 (cit. on p. 8).

Cisneros, Luis, R. Cortez, C. Dombrowski, R.E. Goldstein, and John Kessler (Jan. 2010). “Fluid dynamics of self-propelled microorganisms, from individuals to concentrated populations.” In: *Animal Locomotion*, pp. 99–115 (cit. on p. 29).

Clausen, Thomas, Philippe Jacquet, Cédric Adjih, et al. (2003). “Optimized link state routing protocol (OLSR).” In: (cit. on p. 98).

Coumans, Erwin and Yunfei Bai (2016–2019). *PyBullet, a Python module for physics simulation for games, robotics and machine learning*. `http://pybullet.org` (cit. on p. 101).

Curlander, John C and Robert N McDonough (1991). *Synthetic aperture radar*. Vol. 11. Wiley, New York (cit. on p. 21).

Curran, J. (2017). “A Look at the Threat of Systematic Jamming of GNSS.” In: (cit. on p. 80).

Curtin, Ryan R., James R. Cline, N. P. Slagle, et al. (Mar. 2013). “MLPACK: A Scalable C++ Machine Learning Library.” In: *J. Mach. Learn. Res.* 14.1, pp. 801–805 (cit. on pp. 71, 90, 108).

Dai, Li, Qun Cao, Yuanqing Xia, and Yulong Gao (2017). “Distributed MPC for formation of multi-agent systems with collision avoidance and obstacle avoidance.” In: *Journal of the Franklin Institute* (cit. on pp. 23, 37).

Darwin, Charles (2016). *On the origin of species, 1859* (cit. on p. 24).

Dimitrov, Atanas and Dimitar Minchev (2016). “Ultrasonic sensor explorer.” In: *2016 19th International Symposium on Electrical Apparatus and Technologies (SIELA)*. IEEE, pp. 1–5 (cit. on p. 21).

Dol, Sharul Sham (2020). “Aerodynamic optimization of unmanned aerial vehicle for offshore search and rescue (SAR) operation.” In: *IOP conference series: materials science and engineering*. Vol. 715. 1. IOP Publishing, p. 012015 (cit. on p. 12).

Drake (June 2021). `https://github.com/RobotLocomotion/drake` (cit. on p. 7).

Drone display (May 2021). `https://en.wikipedia.org/wiki/Drone_display` (cit. on p. 12).

Du, Xintong, Carlos E. Luis, Marijan Vukosavljev, and Angela P. Schoellig (2019). *Fast and In Sync: Periodic Swarm Patterns for Quadrotors*. arXiv: `1810.03572 [cs.RO]` (cit. on pp. 36, 41).

Ehang (Jan. 2019). `http://www.ehang.com/formation/example/` (cit. on p. 12).

Eid, Saif Eldin and Sharul Sham Dol (2019). “Design and Development of Lightweight-High Endurance Unmanned Aerial Vehicle for Offshore Search and Rescue Operation.” In: *2019 Advances in Science and Engineering Technology International Conferences (ASET)*, pp. 1–5 (cit. on p. 12).

El-Rabbany, Ahmed (2002). *Introduction to GPS: the global positioning system*. Artech house (cit. on p. 80).

Elmenreich, Wilfried (2002). "An introduction to sensor fusion." In: *Vienna University of Technology, Austria* 502, pp. 1–28 (cit. on p. 22).

Everett, HR (1995). *Sensors for mobile robots*. CRC Press (cit. on p. 19).

Ferris, Brian, Dieter Fox, and Neil Lawrence (2007). "WiFi-SLAM Using Gaussian Process Latent Variable Models." In: *Proceedings of the 20th International Joint Conference on Artifical Intelligence*. IJCAI'07. Hyderabad, India: Morgan Kaufmann Publishers Inc., 2480–2485 (cit. on p. 46).

Finn, Chelsea, Sergey Levine, and Pieter Abbeel (2016). "Guided cost learning: Deep inverse optimal control via policy optimization." In: *International conference on machine learning*. PMLR, pp. 49–58 (cit. on p. 26).

FLIGHT (Jan. 2019). `https://www.flightglobal.com/pdfarchive/view/1960/1960%20-%201367.html` (cit. on p. 31).

Floreano, Dario, Francesco Mondada, et al. (1994). "Automatic creation of an autonomous agent: Genetic evolution of a neural-network driven robot." In: *From animals to animats* 3, pp. 421–430 (cit. on p. 24).

Foot, Philippa (1967). "The problem of abortion and the doctrine of the double effect." In: (cit. on p. 114).

Fu, Kuan-Hsiang (2016). "Using machine learning to learn from demonstration: application to the AR. Drone quadrotor control." PhD thesis (cit. on p. 27).

Furrer, Fadri, Michael Burri, Markus Achtelik, and Roland Siegwart (2016). "RotorS—A Modular Gazebo MAV Simulator Framework." In: *Robot Operating System (ROS): The Complete Reference (Volume 1)*. Cham: Springer International Publishing, pp. 595–625 (cit. on pp. 60, 71, 90, 101).

Galvane, Quentin, Julien Fleureau, Francois-Louis Tariolle, and Philippe Guillotel (2017). "Automated cinematography with unmanned aerial vehicles." In: *arXiv preprint arXiv:1712.04353* (cit. on p. 11).

Gandhi, Dhiraj, Lerrel Pinto, and Abhinav Gupta (2017). "Learning to Fly by Crashing." In: *CoRR* abs/1704.05588. arXiv: `1704.05588` (cit. on p. 40).

Garcia, Carlos E, David M Prett, and Manfred Morari (1989). "Model predictive control: Theory and practice—A survey." In: *Automatica* 25.3, pp. 335–348 (cit. on p. 22).

Gassner, Michael, Titus Cieslewski, and Davide Scaramuzza (2017). "Dynamic collaboration without communication: Vision-based cable-suspended load transport with two quadrotors." In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 5196–5202 (cit. on p. 99).

Gebre-Egziabher, Demoz and Scott Gleason (2009). *GNSS applications and methods*. Artech House (cit. on p. 21).

Glade, David (2000). *Unmanned aerial vehicles: Implications for military operations*. 16. Center for Strategy and Technology, Air War College, Air University (cit. on p. 7).

Glorot, Xavier and Yoshua Bengio (2010). "Understanding the difficulty of training deep feedforward neural networks." In: *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics* (cit. on pp. 71, 90).

Goebel, Michael E, Wayne L Perryman, Jefferson T Hinke, et al. (2015). "A small unmanned aerial system for estimating abundance and size of Antarctic predators." In: *Polar Biology* 38.5, pp. 619–630 (cit. on p. 11).

Golberg, David E (1989). "Genetic algorithms in search, optimization, and machine learning." In: *Addion wesley* 1989.102, p. 36 (cit. on p. 24).

Golemo, Florian, Adrien Ali Taiga, Aaron Courville, and Pierre-Yves Oudeyer (2018). "Sim-to-real transfer with neural-augmented robot simulation." In: *Conference on Robot Learning*. PMLR, pp. 817–828 (cit. on p. 100).

Goodrich, Michael A, Bryan S Morse, Cameron Engh, Joseph L Cooper, and Julie A Adams (2009). "Towards using unmanned aerial vehicles (UAVs) in wilderness search and rescue: Lessons from field trials." In: *Interaction Studies* 10.3, pp. 453–478 (cit. on p. 12).

Gotwald Jr, William H et al. (1995). *Army ants: the biology of social predation.* Cornell University Press (cit. on p. 30).

Halperin, Daniel, Wenjun Hu, Anmol Sheth, and David Wetherall (Jan. 2011). "Tool Release: Gathering 802.11n Traces with Channel State Information." In: *SIGCOMM Comput. Commun. Rev.* 41.1, p. 53 (cit. on p. 97).

Han, Songlai and Jinling Wang (2010). "Quantization and colored noises error modeling for inertial sensors for GPS/INS integration." In: *IEEE Sensors Journal* 11.6, pp. 1493–1503 (cit. on p. 80).

Hansen, Nikolaus, Sibylle D Müller, and Petros Koumoutsakos (2003). "Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES)." In: *Evolutionary computation* 11.1, pp. 1–18 (cit. on p. 97).

Hazry, Desa, Mohd Sofian, and Mohammad Rosbi (2009). "Study of inertial measurement unit sensor." In: (cit. on p. 20).

Hemelrijk, Charlotte K. and Hanno Hildenbrandt (2008). "Self-Organized Shape and Frontal Density of Fish Schools." In: *Ethology* 114.3, pp. 245–254. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1439-0310.2007.01459.x` (cit. on p. 30).

Herrick, Katrina (2000). "Development of the unmanned aerial vehicle market: forecasts and trends." In: *Air & Space Europe* 2.2, pp. 25–27 (cit. on p. 7).

Hoffmann, Gabriel M., Haomiao Huang, Steven L. Waslander, and Claire J. Tomlin (2011). "Precision flight control for a multi-vehicle quadrotor helicopter testbed." In: *Control Engineering Practice* 19.9. Special Section: DCDS'09 – The 2nd IFAC Workshop on Dependable Control of Discrete Systems, pp. 1023 –1036 (cit. on p. 35).

Hofmann-Wellenhof, Bernhard, Herbert Lichtenegger, and Elmar Wasle (2007). *GNSS–global navigation satellite systems: GPS, GLONASS, Galileo, and more*. Springer Science & Business Media (cit. on p. 80).

Honegger, D., L. Meier, P. Tanskanen, and M. Pollefeys (2013). "An open source and open hardware embedded metric optical flow CMOS camera for indoor and outdoor applications." In: *2013 IEEE International Conference on Robotics and Automation*, pp. 1736–1741 (cit. on p. 36).

Hsu, Li-Ta (2017). "GNSS multipath detection using a machine learning approach." In: *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pp. 1–6 (cit. on p. 80).

Huttenrauch, Maximilian, Adrian Sosic, and Gerhard Neumann (2019). "Deep Reinforcement Learning for Swarm Systems." In: arXiv: 1807.06613 [cs.MA] (cit. on p. 38).

– (2017). "Guided Deep Reinforcement Learning for Swarm Systems." In: arXiv: 1709.06011 [cs.MA] (cit. on p. 38).

Hwangbo, Jemin, Inkyu Sa, Roland Siegwart, and Marco Hutter (2017). "Control of a Quadrotor With Reinforcement Learning." In: *IEEE Robotics and Automation Letters* 2.4, 2096–2103 (cit. on p. 39).

Intel drones light show (Jan. 2019). https://iq.intel.com/500-drones-light-show-sets-record/ (cit. on p. 12).

Jeong, Jinyong, Younggun Cho, Young-Sik Shin, Hyunchul Roh, and Ayoung Kim (2019). "Complex urban dataset with multi-level sensors from highly diverse urban environments." In: *The International Journal of Robotics Research* 38.6, pp. 642–657 (cit. on p. 22).

Jing, Changjuan, Johan Potgieter, Frazer Noble, and Ruili Wang (2017). "A comparison and analysis of RGB-D cameras' depth performance for robotics application." In: *2017 24th International Conference on Mechatronics and Machine Vision in Practice (M2VIP)*. IEEE, pp. 1–6 (cit. on p. 20).

Joubert, Niels, Dan B Goldman, Floraine Berthouzoz, et al. (2016). "Towards a drone cinematographer: Guiding quadrotor cameras using visual composition principles." In: *arXiv preprint arXiv:1610.01691* (cit. on p. 99).

Joubert, Niels, Mike Roberts, Anh Truong, Floraine Berthouzoz, and Pat Hanrahan (2015). "An interactive tool for designing quadrotor camera shots." In: *ACM Transactions on Graphics (TOG)* 34.6, pp. 1–11 (cit. on p. 99).

Kam, Moshe, Xiaoxun Zhu, and Paul Kalata (1997). "Sensor fusion for mobile robot navigation." In: *Proceedings of the IEEE* 85.1, pp. 108–119 (cit. on p. 22).

Kamel, Mina, Javier Alonso-Mora, Roland Siegwart, and Juan Nieto (2017). *Nonlinear Model Predictive Control for Multi-Micro Aerial Vehicle Robust Collision Avoidance*. arXiv: 1703.01164 [cs.RO] (cit. on pp. 37, 41).

Karaim, Malek, Mohamed Elsheikh, Aboelmagd Noureldin, and RB Rustamov (2018). "GNSS error sources." In: *Multifunctional Operation and Application of GPS*, pp. 69–85 (cit. on p. 80).

Katikala, Soujanya (2014). "Google project loon." In: *InSight: Rivier Academic Journal* 10.2, pp. 1–6 (cit. on p. 11).

Katz, Yael, Kolbjørn Tunstrøm, Christos C. Ioannou, Cristián Huepe, and Iain D. Couzin (2011). "Inferring the structure and dynamics of interactions in schooling fish." In: *Proceedings of the National Academy of Sciences* 108.46, pp. 18720–18725 (cit. on p. 30).

Kerl, Christian, Jürgen Sturm, and Daniel Cremers (2013). "Dense visual SLAM for RGB-D cameras." In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, pp. 2100–2106 (cit. on p. 22).

Kim, H., Michael Jordan, Shankar Sastry, and Andrew Ng (2004). "Autonomous Helicopter Flight via Reinforcement Learning." In: *Advances in Neural Information Processing Systems*. Ed. by S. Thrun, L. Saul, and B. Schölkopf. Vol. 16. MIT Press (cit. on p. 39).

Kingma, Diederik P. and Jimmy Ba (2014). *Adam: A Method for Stochastic Optimization* (cit. on pp. 71, 90).

Kober, Jens, J Andrew Bagnell, and Jan Peters (2013). "Reinforcement learning in robotics: A survey." In: *The International Journal of Robotics Research* 32.11, pp. 1238–1274 (cit. on p. 7).

Koch, William, Renato Mancuso, and Azer Bestavros (2019). "Neuroflight: next generation flight control firmware." In: (cit. on p. 39).

Koch, William, Renato Mancuso, Richard West, and Azer Bestavros (2018). "Reinforcement Learning for UAV Attitude Control." In: arXiv: `1804.04154 [cs.RO]` (cit. on p. 39).

Koenig, N. and A. Howard (2004). "Design and use paradigms for Gazebo, an open-source multi-robot simulator." In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*. Vol. 3, 2149–2154 vol.3 (cit. on p. 101).

Kratky, Vit, Alfonso Alcantara, Jesus Capitan, et al. (2021). "Autonomous Aerial Filming with Distributed Lighting by a Team of Unmanned Aerial Vehicles." In: *IEEE Robotics and Automation Letters*, pp. 1–1 (cit. on p. 11).

Kretzschmar, Henrik, Markus Spies, Christoph Sprunk, and Wolfram Burgard (2016). "Socially compliant mobile robot navigation via inverse reinforcement learning." In: *The International Journal of Robotics Research* 35.11, pp. 1289–1307 (cit. on p. 27).

Kushleyev, Alex, Daniel Mellinger, Caitlin Powers, and Vijay Kumar (2013). "Towards a Swarm of Agile Micro Quadrotors." In: *Auton. Robots* 35.4, pp. 287–300 (cit. on pp. 36, 41).

Kuzmin, Maxim (2018). "Classification and comparison of the existing SLAM methods for groups of robots." In: *2018 22nd Conference of Open Innovations Association (FRUCT)*. IEEE, pp. 115–120 (cit. on p. 22).

Le, Hoang M, Yisong Yue, Peter Carr, and Patrick Lucey (2017). "Coordinated multi-agent imitation learning." In: *International Conference on Machine Learning*. PMLR, pp. 1995–2003 (cit. on p. 81).

Lear, Jonathan (1980). *Aristotle and logical theory*. CUP Archive (cit. on p. 5).

Lee, Doo Hyun, In So Kweon, and Roberto Cipolla (1999). "A biprism-stereo camera system." In: *Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149)*. Vol. 1. IEEE (cit. on p. 20).

Lee, DooHyun and InSo Kweon (2000). "A novel stereo camera system by a biprism." In: *IEEE Transactions on Robotics and Automation* 16.5, pp. 528–541 (cit. on p. 20).

Lee, Jeongseok, Michael X. Grey, Sehoon Ha, et al. (2018). "DART: Dynamic Animation and Robotics Toolkit." In: *Journal of Open Source Software* 3.22, p. 500 (cit. on p. 101).

Leishman, J Gordon (2002). "The breguet-richet quad-rotor helicopter of 1907." In: *Vertiflite* 47.3, pp. 58–60 (cit. on p. 31).

Levanon, Nadav (1988). "Radar principles." In: *New York* (cit. on p. 21).

Levine, Herbert, Wouter-Jan Rappel, and Inon Cohen (2000). "Self-organization in systems of self-propelled particles." In: *Phys. Rev. E* 63 (1), p. 017101 (cit. on p. 32).

Lim, H., L. . Kung, J. C. Hou, and H. Luo (2006). "Zero-Configuration, Robust Indoor Localization: Theory and Experimentation." In: *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*, pp. 1–12 (cit. on p. 46).

Lim, H., J. Park, D. Lee, and H. J. Kim (2012). "Build Your Own Quadrotor: Open-Source Projects on Unmanned Aerial Vehicles." In: *IEEE Robotics Automation Magazine* 19.3, pp. 33–45 (cit. on p. 31).

Lindsey, Quentin, Daniel Mellinger, and Vijay Kumar (2011a). "Construction of cubic structures with quadrotor teams." In: *Proc. Robotics: Science & Systems VII* (cit. on p. 10).

– (2011b). "Construction of Cubic Structures with Quadrotor Teams." In: *Robotics: Science and Systems*. Ed. by Hugh F. Durrant-Whyte, Nicholas Roy, and Pieter Abbeel (cit. on pp. 32, 99).

– (2012). "Construction with quadrotor teams." In: *Autonomous Robots* 33.3, pp. 323–336 (cit. on p. 10).

Litomisky, Krystof (2012). "Consumer rgb-d cameras and their applications." In: *Rapport technique, University of California* 20, p. 28 (cit. on p. 20).

Liu, Sikang, Michael Watterson, Sarah Tang, and Vijay Kumar (2016). "High speed navigation for quadrotors with limited onboard sensing." In: *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE, pp. 1484–1491 (cit. on p. 99).

Loianno, G. and V. Kumar (2018). "Cooperative Transportation Using Small Quadrotors Using Monocular Vision and Inertial Sensing." In: *IEEE Robotics and Automation Letters* 3.2, pp. 680–687 (cit. on p. 12).

Loianno, G., Y. Mulgaonkar, C. Brunner, et al. (2016). "A swarm of flying smartphones." In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1681–1688 (cit. on pp. 36, 41).

Long, Pinxin, Tingxiang Fan, Xinyi Liao, et al. (2018). *Towards Optimally Decentralized Multi-Robot Collision Avoidance via Deep Reinforcement Learning*. arXiv: `1709.10082 [cs.RO]` (cit. on p. 39).

Loquercio, Antonio, Ana Isabel Maqueda, Carlos R. Del Blanco, and Davide Scaramuzza (2018). "Dronet: Learning to Fly by Driving." In: *IEEE Robotics and Automation Letters* (cit. on p. 38).

Lupashin, Sergei, Markus Hehn, Mark W. Mueller, et al. (2014). "A platform for aerial robotics research and demonstration: The Flying Machine Arena." In: *Mechatronics* 24.1, pp. 41 –54 (cit. on p. 35).

Malaver, Alexander, Nunzio Motta, Peter Corke, and Felipe Gonzalez (2015). "Development and integration of a solar powered unmanned aerial vehicle and a wireless sensor network to monitor greenhouse gases." In: *Sensors* 15.2, pp. 4072–4096 (cit. on p. 11).

Marocco, Davide and Stefano Nolfi (2006). "Self-organization of communication in evolving robots." In: *Proceedings of the Conference on Artificial Life (ALIFE)*, pp. 178–184 (cit. on p. 24).

McCarthy, John, Marvin L Minsky, Nathaniel Rochester, and Claude E Shannon (2006). "A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955." In: *AI magazine* 27.4, pp. 12–12 (cit. on p. 5).

Meguro, Jun-ichi, Taishi Murata, Jun-ichi Takiguchi, Yoshiharu Amano, and Takumi Hashizume (2009). "GPS Multipath Mitigation for Urban Area Using Omnidirectional Infrared Camera." In: *IEEE Transactions on Intelligent Transportation Systems* 10.1, pp. 22–30 (cit. on p. 80).

Meier, L., D. Honegger, and M. Pollefeys (2015). "PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms." In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6235–6240 (cit. on p. 31).

Mellinger, Daniel, Michael Shomin, Nathan Michael, and Vijay Kumar (2010). "Cooperative Grasping and Transport Using Multiple Quadrotors." In: *DARS* (cit. on p. 12).

Mester, G. (2011). "The Modeling and Simulation of an Autonomous Quad-Rotor Microcopter in a Virtual Outdoor Scenario." In: (cit. on p. 101).

Meyer, J., A. Sendobry, S. Kohlbrecher, U. Klingauf, and O. V. Stryk (2012). "Comprehensive Simulation of Quadrotor UAVs Using ROS and Gazebo." In: *SIMPAR* (cit. on p. 101).

Michael, N., D. Mellinger, Q. Lindsey, and V. Kumar (2010). "The GRASP Multiple Micro-UAV Testbed." In: *IEEE Robotics Automation Magazine* 17.3, pp. 56–65 (cit. on pp. 35, 41, 100).

Michel, O. (2004). "Webots: Professional Mobile Robot Simulation." In: *Journal of Advanced Robotics Systems* 1.1, pp. 39–42 (cit. on p. 101).

Mogilner, Alex and Leah Edelstein-Keshet (1995). "Selecting a common direction." In: *Journal of mathematical biology* 33.6, pp. 619–660 (cit. on p. 32).

– (Jan. 1996). "Spatio-angular order in populations of self-aligning objects: Formation of oriented patches." In: *Physica D: Nonlinear Phenomena* 89, pp. 346–367 (cit. on p. 32).

Mohamed, Nader, Jameela Al-Jaroodi, Imad Jawhar, Ahmed Idries, and Farhan Mohammed (2020). "Unmanned aerial vehicles applications in future smart cities." In: *Technological Forecasting and Social Change* 153, p. 119293 (cit. on p. 11).

Mondada, Francesco, Giovanni C Pettinaro, Andre Guignard, et al. (2004). "SWARM-BOT: A new distributed robotic concept." In: *Autonomous robots* 17.2, pp. 193–221 (cit. on p. 24).

Morari, Manfred and Jay H Lee (1999). "Model predictive control: past, present and future." In: *Computers & Chemical Engineering* 23.4-5, pp. 667–682 (cit. on p. 22).

Moravec, Hans (1988). *Mind children: The future of robot and human intelligence*. Harvard University Press (cit. on p. 6).

Mosavi, Amir and Annamaria Varkonyi (2017). "Learning in robotics." In: *International Journal of Computer Applications* 157.1, pp. 8–11 (cit. on p. 7).

Mukherjee, A., S. Misra, and N. Raghuwanshi (2019). "A survey of unmanned aerial sensing solutions in precision agriculture." In: *J. Netw. Comput. Appl.* 148 (cit. on p. 10).

Mulgaonkar, Yash (2019). "Small, Safe Quadrotors for Autonomous Flight." PhD thesis. University of Pennsylvania (cit. on p. 100).

Murphy, Robin, Sam Stover, Kevin Pratt, and Chandler Griffin (2006). "Cooperative Damage Inspection with Unmanned Surface Vehicle and Micro Unmanned Aerial Vehicle at Hurricane Wilma." In: *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 9–9 (cit. on p. 12).

Murphy, R.R. (1998). "Dempster-Shafer theory for sensor fusion in autonomous mobile robots." In: *IEEE Transactions on Robotics and Automation* 14.2, pp. 197–206 (cit. on p. 22).

Musaloiu-E, Razvan and Andreas Terzis (2008). "Minimising the effect of WiFi interference in 802.15. 4 wireless sensor networks." In: *International Journal of Sensor Networks* 3.1, pp. 43–54 (cit. on p. 98).

Müller, M., S. Lupashin, and R. D'Andrea (2011). "Quadrocopter ball juggling." In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5113–5120 (cit. on p. 12).

Nagpal, Lavina and Krishna Samdani (2017). "Project Loon: Innovating the connectivity worldwide." In: *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT)*, pp. 1778–1784 (cit. on p. 11).

Nagy, M, Z Akos, D Biro, and T Vicsek (2010). "Hierarchical group dynamics in pigeon flocks." In: *Nature* 464.7290, pp. 890–893 (cit. on p. 30).

Nair, Vinod and Geoffrey E. Hinton (2010). "Rectified Linear Units Improve Restricted Boltzmann Machines." In: *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pp. 807–814 (cit. on p. 71).

Nandakumar, Rajalakshmi, Krishna Kant Chintalapudi, and Venkata N. Padmanabhan (2012). "Centaur: Locating Devices in an Office Environment." In: *Proceedings of the 18th Annual International Conference on Mobile Computing and Networking*. Mobicom '12. Istanbul, Turkey: Association for Computing Machinery, 281–292 (cit. on p. 46).

Neukum, Gerhard and Ralf Jaumann (2004). "HRSC: The high resolution stereo camera of Mars Express." In: *Mars Express: The Scientific Payload*. Vol. 1240, pp. 17–35 (cit. on p. 20).

Ng, Andrew Y. and Michael I. Jordan (2013). "PEGASUS: A Policy Search Method for Large MDPs and POMDPs." In: arXiv: `1301.3878 [cs.AI]` (cit. on p. 39).

Niculescu, Dragoş and Badri Nath (2004). "VOR Base Stations for Indoor 802.11 Positioning." In: *Proceedings of the 10th Annual International Conference on Mobile Computing and Networking*. MobiCom '04. Association for Computing Machinery, 58–69 (cit. on p. 97).

Ohta, Akio (2017). “Sky magic: Drone entertainment show.” In: *ACM SIGGRAPH 2017 Emerging Technologies*, pp. 1–2 (cit. on p. 99).

Olfati-Saber, R. (2006). “Flocking for multi-agent dynamic systems: algorithms and theory.” In: *IEEE Transactions on Automatic Control* 51.3, pp. 401–420 (cit. on pp. 34, 80).

Paparazzi Project (May 2018). `https://wiki.paparazziuav.org/wiki/Main_Page` (cit. on p. 31).

PARROT (May 2021). `http://www.parrot.com` (cit. on p. 50).

Passalis, Nikolaos, Anastasios Tefas, and Ioannis Pitas (2018). “Efficient camera control using 2D visual information for unmanned aerial vehicle-based cinematography.” In: *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, pp. 1–5 (cit. on p. 11).

Paulraj, A., V. U. Reddy, T. J. Shan, and T. Kailath (1986). “Performance Analysis of the Music Algorithm with Spatial Smoothing in the Presence of Coherent Sources.” In: *MILCOM 1986 - IEEE Military Communications Conference: Communications-Computers: Teamed for the 90’s*. Vol. 3, pp. 41.5.1–41.5.5 (cit. on p. 97).

Perkins, C.E. and E.M. Royer (1999). “Ad-hoc on-demand distance vector routing.” In: *Proceedings WMCSA’99. Second IEEE Workshop on Mobile Computing Systems and Applications*, pp. 90–100 (cit. on p. 98).

Perlovsky, Leonid I and Margaret M McManus (1991). “Maximum likelihood neural networks for sensor fusion and adaptive classification.” In: *Neural Networks* 4.1, pp. 89–102 (cit. on p. 22).

Pitcher, T.J. (1983). “Heuristic definitions of fish shoaling behaviour.” In: *Animal Behaviour* 31.2, pp. 611–613 (cit. on p. 30).

Pitonakova, Lenka, Manuel Giuliani, Anthony Pipe, and Alan Winfield (2018). “Feature and performance comparison of the V-REP, Gazebo and ARGoS robot simulators.” In: *Annual Conference Towards Autonomous Robotic Systems*. Springer, pp. 357–368 (cit. on p. 100).

Pizetta, Igor Henrique Beloti, Alexandre Santos Brandão, and Mario Sarcinelli-Filho (2016). “Cooperative quadrotors carrying a suspended load.” In: *2016 international conference on unmanned aircraft systems (ICUAS)*. IEEE, pp. 1049–1055 (cit. on p. 99).

Poulton, Christopher Vincent (2016). “Integrated LIDAR with optical phased arrays in silicon photonics.” PhD thesis. Massachusetts Institute of Technology (cit. on p. 21).

Pounds, Paul and Robert Mahony (2009). “Design principles of large quadrotors for practical applications.” In: *2009 IEEE International Conference on Robotics and Automation*. IEEE, pp. 3265–3270 (cit. on p. 100).

Preiss, J. A., W. Honig, G. S. Sukhatme, and N. Ayanian (2017). “Crazyswarm: A large nano-quadcopter swarm.” In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3299–3304 (cit. on p. 35).

Qi, Juntong, Dalei Song, Hong Shang, et al. (2016). “Search and Rescue Rotary-Wing UAV and Its Application to the Lushan Ms 7.0 Earthquake.” In: *J. Field Robotics* 33, pp. 290–321 (cit. on p. 12).

Quinn, Matt (2001). “Evolving communication without dedicated communication channels.” In: *European Conference on Artificial Life*. Springer, pp. 357–366 (cit. on p. 24).

Quinn, Matt, Lincoln Smith, Giles Mayley, and Phil Husbands (2003). “Evolving controllers for a homogeneous system of physical robots: Structured cooperation with minimal sensors.” In: *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* 361.1811, pp. 2321–2343 (cit. on p. 24).

Rai, Vineet Kumar (2007). “Temperature sensors and optical sensors.” In: *Applied Physics B* 88.2, pp. 297–303 (cit. on p. 20).

Rawlings, J and D.Q. Mayne (Jan. 2009). *Model Predictive Control: Theory and Design* (cit. on p. 22).

Reynolds, Craig W. (1987). “Flocks, Herds and Schools: A Distributed Behavioral Model.” In: *SIGGRAPH Comput. Graph.* 21.4, pp. 25–34 (cit. on pp. 32, 108).

Richards, Arthur and Jonathan How (2004). “Decentralized model predictive control of cooperating UAVs.” In: *2004 43rd IEEE Conference on Decision and Control (CDC)(IEEE Cat. No. 04CH37601)*. Vol. 4. IEEE, pp. 4286–4291 (cit. on p. 37).

Ritz, R., M. W. Müller, M. Hehn, and R. D’Andrea (2012). “Cooperative quadrocopter ball throwing and catching.” In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4972–4978 (cit. on p. 12).

Roberts, J., T. Stirling, J. Zufferey, and D. Floreano (2012). “3-D relative positioning sensor for indoor flying robots.” In: *Autonomous Robots* 33, pp. 5–20 (cit. on p. 36).

Roberts, J. F., T. S. Stirling, J. Zufferey, and D. Floreano (2009). “2.5D infrared range and bearing system for collective robotics.” In: *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3659–3664 (cit. on p. 36).

Robinson, Andrew (2003). *The origins of writing*. na (cit. on p. 5).

– (2009). *Writing and script: a very short introduction*. Vol. 208. Oxford University Press (cit. on p. 5).

Rohner, Christian, Erik Nordström, Per Gunningberg, and Christian Tschudin (2005). “Interactions Between TCP, UDP, and Routing Protocols in Wireless Multi-hop Ad Hoc Networks.” In: *Proc. IEEE ICPS Workshop on Multi-hop Ad hoc Networks: from theory to reality (REALMAN’05)* (cit. on p. 97).

Ross, Stéphane (2013). “Interactive Learning for Sequential Decisions and Predictions.” In: (cit. on p. 25).

Ross, Stephane, Geoffrey J. Gordon, and J. Andrew Bagnell (2011). “A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning.” In: arXiv: `1011.0686 [cs.LG]` (cit. on p. 84).

Ross, Stéphane, Narek Melik-Barkhudarov, Kumar Shaurya Shankar, et al. (2012). “Learning Monocular Reactive UAV Control in Cluttered Natural Environments.” In: *CoRR* abs/1211.1690. arXiv: `1211.1690` (cit. on p. 38).

Sadeghi, Fereshteh and Sergey Levine (2016). “(CAD)$ˆ2$RL: Real Single-Image Flight without a Single Real Image.” In: *CoRR* abs/1611.04201. arXiv: `1611.04201` (cit. on p. 39).

Saif, Abdu, Kaharudin Dimyati, Kamarul Ariffin Noordin, et al. (2020). “Unmanned Aerial Vehicles for Post-Disaster Communication Networks.” In: *2020 IEEE 10th International Conference on System Engineering and Technology (ICSET)*. IEEE, pp. 273–277 (cit. on p. 11).

Sanderson, C. (2014). “Armadillo: C++ template metaprogramming for compile-time optimization of linear algebra.” In: *Computational Statistics and Data Analysis* 71 (cit. on p. 90).

Santos, Sérgio R Barros dos, Sidney Givigi, Cairo L Nascimento, et al. (2018). “Iterative decentralized planning for collective construction tasks with quadrotors.” In: *Journal of Intelligent & Robotic Systems* 90.1, pp. 217–234 (cit. on p. 10).

Sarkar, Tapan K, Zhong Ji, Kyungjung Kim, Abdellatif Medouri, and Magdalena Salazar-Palma (2003). “A survey of various propagation models for mobile communication.” In: *IEEE Antennas and propagation Magazine* 45.3, pp. 51–82 (cit. on p. 21).

Sarova, Radka, Marek Spinka, Jose L. Arias Panama, and Petr Simecek (2010). “Graded leadership by dominant animals in a herd of female beef cattle on pasture.” In: *Animal Behaviour* 79.5, pp. 1037–1045 (cit. on p. 30).

Sasiadek, Jurek Z (2002). “Sensor fusion.” In: *Annual Reviews in Control* 26.2, pp. 203–228 (cit. on p. 22).

Sasiadek, JZ and P Hartana (2000). “Sensor data fusion using Kalman filter.” In: *Proceedings of the Third International Conference on Information Fusion*. Vol. 2. IEEE, WED5–19 (cit. on p. 22).

Saska, M., T. Baca, and D. Hert (2016a). “Formations of unmanned micro aerial vehicles led by migrating virtual leader.” In: *2016 14th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pp. 1–6 (cit. on pp. 37, 41).

Saska, M., J. Chudoba, Libor Precil, et al. (2014a). “Autonomous deployment of swarms of micro-aerial vehicles in cooperative surveillance.” In: *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 584–595 (cit. on p. 37).

Saska, M., Zdenek Kasl, and L. Preucil (2014b). “Motion planning and control of formations of micro aerial vehicles.” In: *IFAC Proceedings Volumes* 47, pp. 1228–1233 (cit. on pp. 37, 41).

Saska, M., T. Krajník, V. Vonásek, P. Vaněk, and L. Přeučil (2013). “Navigation, localization and stabilization of formations of unmanned aerial and ground vehicles.” In: *2013 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 831–840 (cit. on p. 37).

Saska, M., Vojtech Vonásek, T. Krajník, and L. Preucil (2014c). “Coordination and navigation of heterogeneous MAV–UGV formations localized by a ‘hawk-eye’-like approach under a model predictive control scheme.” In: *The International Journal of Robotics Research* 33, pp. 1393 –1412 (cit. on p. 37).

Saska, Martin, Jan Vakula, and Libor Přeućil (2014d). “Swarms of micro aerial vehicles stabilized under a visual relative localization.” In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3570–3575 (cit. on p. 41).

Saska, Martin, Vojtundefinedch Vonásek, Jan Chudoba, et al. (2016b). “Swarm Distribution and Deployment for Cooperative Surveillance by Micro-Aerial Vehicles.” In: *J. Intell. Robotics Syst.* 84.1–4, 469–492 (cit. on p. 37).

Schank, Roger C (1991). “Where’s the AI?” In: *AI magazine* 12.4, pp. 38–38 (cit. on p. 5).

Schilling, Fabian, Julien Lecoeur, Fabrizio Schiano, and Dario Floreano (2018). “Learning Vision-based Cohesive Flight in Drone Swarms.” In: *CoRR* abs/1809.00543. arXiv: 1809.00543 (cit. on pp. 38, 41, 81, 84, 108).

– (2019). “Learning Vision-based Flight in Drone Swarms by Imitation.” In: *CoRR* abs/1908.02999. arXiv: 1908.02999 (cit. on pp. 38, 41, 80).

Schmidt, R. (1979). “Multiple emitter location and signal Parameter estimation.” In: (cit. on p. 97).

Schmittle, Matthew, Anna Lukina, Lukas Vacek, et al. (Apr. 2018). “OpenUAV: A UAV Testbed for the CPS and Robotics Community.” In: pp. 130–139 (cit. on p. 35).

Schouwenaars, T., E. Feron, and J. How (2006). “Multi-vehicle path planning for non-line of sight communication.” In: *2006 American Control Conference*, 6 pp.– (cit. on p. 36).

Schouwenaars, Tom, Bart De Moor, Eric Feron, and Jonathan How (Jan. 2001). “Mixed integer programming for multi-vehicle path-planning.” In: *European Control Conf.* (cit. on p. 36).

Sen, Souvik, Božidar Radunovic, Romit Roy Choudhury, and Tom Minka (2012). “You Are Facing the Mona Lisa: Spot Localization Using PHY Layer Information.” In: *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*. MobiSys ’12. Low Wood Bay, Lake District, UK: Association for Computing Machinery, 183–196 (cit. on p. 46).

Shah, Shital, Debadeepta Dey, Chris Lovett, and Ashish Kapoor (2017). *AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles*. arXiv: 1705.05065 [cs.RO] (cit. on p. 101).

Shakhatreh, Hazim, Ahmad H Sawalmeh, Ala Al-Fuqaha, et al. (2019). “Unmanned aerial vehicles (UAVs): A survey on civil applications and key research challenges.” In: *Ieee Access* 7, pp. 48572–48634 (cit. on p. 7).

Shanbi, Wei, Chai Yi, and Penghua Li (Apr. 2012). “Distributed Model Predictive Control of the Multi-Agent Systems with Improving Control Performance.” In: *Journal of Control Science and Engineering* 2012 (cit. on p. 37).

Sheth, Anmol, Sergiu Nedevschi, Rabin Patra, et al. (2007). “Packet loss characterization in WiFi-based long distance networks.” In: *IEEE INFOCOM 2007-26th IEEE International Conference on Computer Communications*. IEEE, pp. 312–320 (cit. on p. 97).

Shimoyama, Sugawara, Mizuguchi, Hayakawa, and Sano (1996). “Collective motion in a system of motile elements.” In: *Physical review letters* 76 20, pp. 3870–3873 (cit. on p. 32).

Shin, Young-Sik, Yeong Sang Park, and Ayoung Kim (2018). “Direct visual slam using sparse depth for camera-lidar system.” In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 5144–5151 (cit. on p. 22).

Shrit, Omar, David Filliat, and Michele Sebag (Feb. 2021). “Iterative Learning for Model Reactive Control: Application to autonomous multi-agent control.” In: *ICARA*. Prague, Czech Republic (cit. on pp. 84, 86, 108).

Skog, Isaac and Peter Händel (2006). “Calibration of a MEMS inertial measurement unit.” In: *XVII IMEKO world congress*. Citeseer, pp. 1–6 (cit. on p. 20).

Smith, Robin (2000). “Aristotle’s logic.” In: (cit. on p. 5).

Soria, Enrica, Fabrizio Schiano, and Dario Floreano (2020). “SwarmLab: a Matlab Drone Swarm Simulator.” In: arXiv: 2005.02769 [cs.RO] (cit. on p. 101).

Spurny, V., T. Baca, and M. Saska (2016). “Complex manoeuvres of heterogeneous MAV-UGV formations using a model predictive control.” In: *2016 21st International Conference on Methods and Models in Automation and Robotics (MMAR)*, pp. 998–1003 (cit. on p. 37).

Stirling, T., J. Roberts, J. Zufferey, and D. Floreano (2012). “Indoor navigation with a swarm of flying robots.” In: *2012 IEEE International Conference on Robotics and Automation*, pp. 4641–4647 (cit. on p. 35).

Stirling, Timothy, Steffen Wischmann, and Dario Floreano (June 2010). “Energy-Efficient Indoor Search by Swarms of Simulated Flying Robots Without Global Information.” In: *Swarm Intelligence* 4 (cit. on p. 35).

Sturm, Jürgen, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers (2012). “A benchmark for the evaluation of RGB-D SLAM systems.” In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, pp. 573–580 (cit. on p. 22).

Sutton, Richard S. and Andrew G. Barto (1998). *Introduction to Reinforcement Learning*. 1st. Cambridge, MA, USA: MIT Press (cit. on p. 25).

Syed, Umar and Robert E Schapire (2008). “A Game-Theoretic Approach to Apprenticeship Learning.” In: *Advances in Neural Information Processing Systems*. Ed. by J. Platt, D. Koller, Y. Singer, and S. Roweis. Vol. 20. Curran Associates, Inc. (cit. on p. 26).

Szepesvári, Csaba (2010). “Algorithms for reinforcement learning.” In: *Synthesis lectures on artificial intelligence and machine learning* 4.1, pp. 1–103 (cit. on p. 25).

Tomic, Teodor, Korbinian Schmid, Philipp Lutz, et al. (2012). “Toward a fully autonomous UAV: Research platform for indoor and outdoor urban search and rescue.” In: *IEEE robotics & automation magazine* 19.3, pp. 46–56 (cit. on p. 99).

Topaz, Chad and Andrea Bertozzi (Jan. 2004). “Swarming Patterns in a Two-Dimensional Kinematic Model for Biological Groups.” In: *SIAM Journal of Applied Mathematics* 65, pp. 152–174 (cit. on p. 32).

Torres-González, Arturo, Jesús Capitán, Rita Cunha, Anibal Ollero, and Ioannis Mademlis (2017). “A multidrone approach for autonomous cinematography planning.” In: *Iberian robotics conference*. Springer, pp. 337–349 (cit. on p. 11).

Turner, Darren, Arko Lucieer, Zbyněk Malenovskỳ, Diana H King, and Sharon A Robinson (2014). “Spatial co-registration of ultra-high resolution visible, multispectral and thermal images acquired with a micro-UAV over Antarctic moss beds.” In: *Remote Sensing* 6.5, pp. 4003–4024 (cit. on p. 11).

Turpin, Matthew, Nathan Michael, and Vijay Kumar (2014). “Capt: Concurrent assignment and planning of trajectories for multiple robots.” In: *I. J. Robotics Res.* 33.1, pp. 98–112 (cit. on pp. 36, 41, 43).

Udacity data set (Jan. 2019). `https://github.com/udacity/self-driving-car` (cit. on p. 38).

Union, International Telecommunication (2019). *Propagation data and prediction methods for the planning of short-range outdoor radiocommunication systems and radio local area networks in the frequency range 300 MHz to 100 GHz*. Tech. rep. P.1411-10 (cit. on pp. 45, 64).

Valavanis, Kimon P. and George J. Vachtsevanos (2014). *Handbook of Unmanned Aerial Vehicles*. Springer Publishing Company, Incorporated (cit. on p. 32).

Vásárhelyi, Gábor, Csaba Virágh, Gergő Somorjai, et al. (2018). “Optimized flocking of autonomous drones in confined environments.” In: *Science Robotics* 3.20. eprint: `http://robotics.sciencemag.org/content/3/20/eaat3536.full.pdf` (cit. on pp. 37, 41, 97).

Vásárhelyi, Gábor, Csaba Virágh, Gergo Somorjai, et al. (2014). “Outdoor flocking and formation flight with autonomous aerial robots.” In: *CoRR* abs/1402.3588. arXiv: `1402.3588` (cit. on pp. 35, 37).

Vasquez, Dizan, Billy Okal, and Kai O Arras (2014). “Inverse reinforcement learning algorithms and features for robot navigation in crowds: an experimental comparison.” In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, pp. 1341–1346 (cit. on p. 27).

Vicon (Feb. 2019). `https://www.vicon.com/` (cit. on p. 36).

Vicsek, Tamás, András Czirók, Eshel Ben-Jacob, Inon Cohen, and Ofer Shochet (1995). “Novel type of phase transition in a system of self-driven particles.” In: *Physical Review Letters* 75.6, p. 1226 (cit. on p. 33).

Virágh, Csaba, Gábor Vásárhelyi, Norbert Tarcai, et al. (Oct. 2013). “Flocking algorithm for autonomous flying robots.” In: 9 (cit. on pp. 35, 41, 80).

Wagner, Daniel and Dieter Schmalstieg (2007). “Artoolkitplus for pose tracking on mobile devices.” In: (cit. on p. 36).

Wang, Z., S. Singh, M. Pavone, and M. Schwager (2018). “Cooperative Object Transport in 3D with Multiple Quadrotors Using No Peer Communication.” In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1064–1071 (cit. on p. 12).

Wei, Sixiao, Linqiang Ge, Wei Yu, et al. (2014). “Simulation study of unmanned aerial vehicle communication networks addressing bandwidth disruptions.” In: *Sensors and Systems for Space Applications VII*. Vol. 9085. International Society for Optics and Photonics, 90850O (cit. on p. 11).

Weinstein, A., A. Cho, G. Loianno, and V. Kumar (2018). “Visual Inertial Odometry Swarm: An Autonomous Swarm of Vision-Based Quadrotors.” In: *IEEE Robotics and Automation Letters* 3.3, pp. 1801–1807 (cit. on pp. 36, 41).

Wenseleers, Tom (Mar. 2010). “The Superorganism: The Beauty, Elegance, and Strangeness of Insect Societies.” In: *Quarterly Review of Biology QUART REV BIOL* 85, pp. 114–115 (cit. on p. 30).

Whipker, Linda D and Jay T Akridge (2008). *2008 Precision Agricultural Services Dealership Survey Results*. Tech. rep. (cit. on p. 10).

Willmann, Jan, Federico Augugliaro, Thomas Cadalbert, et al. (2012). "Aerial robotic construction towards a new field of architectural research." In: *International journal of architectural computing* 10.3, pp. 439–459 (cit. on p. 10).

Wray, K. H. and S. Zilberstein (2019). "Generalized Controllers in POMDP Decision-Making." In: *2019 (ICRA)*, pp. 7166–7172 (cit. on p. 9).

Wu, K., Jiang Xiao, Youwen Yi, Min Gao, and L. M. Ni (2012). "FILA: Fine-grained indoor localization." In: *2012 Proceedings IEEE INFOCOM*, pp. 2210–2218 (cit. on p. 46).

Xia, Chen and Abdelkader El Kamel (2016). "Neural inverse reinforcement learning in autonomous navigation." In: *Robotics and Autonomous Systems* 84, pp. 1–14 (cit. on p. 27).

Xie, Xu, Changyang Li, Chi Zhang, Yixin Zhu, and Song-Chun Zhu (2019). "Learning virtual grasp with failed demonstrations via bayesian inverse reinforcement learning." In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 1812–1817 (cit. on p. 27).

Xu, Wenbo, Shu Wang, Shu Yan, and Jianhua He (2018). "An efficient wideband spectrum sensing algorithm for unmanned aerial vehicle communication networks." In: *IEEE Internet of Things Journal* 6.2, pp. 1768–1780 (cit. on p. 11).

Yang, Shuting, Robert W Talbot, Michael B Frish, et al. (2018). "Natural gas fugitive leak detection using an unmanned aerial vehicle: Measurement system description and mass balance approach." In: *Atmosphere* 9.10, p. 383 (cit. on p. 11).

YOUNG, Warren R (1982). *The Helicopters. The Epic of Flight* (cit. on p. 31).

Youssef, Moustafa and Ashok Agrawala (2005). "The Horus WLAN Location Determination System." In: *Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services*. MobiSys '05. Seattle, Washington: Association for Computing Machinery, 205–218 (cit. on p. 46).

Yozevitch, Roi (Dec. 2017). "Bayesian Methods for GNSS Accuracy Improvement." PhD thesis (cit. on p. 80).

Zhang, Chunhua and John M. Kovacs (2012). "The application of small unmanned aerial systems for precision agriculture: a review." In: *Precision Agriculture* 13.6, pp. 693–712 (cit. on p. 32).

Zhmud, VA, NO Kondratiev, KA Kuznetsov, VG Trubin, and LV Dimitrov (2018). "Application of ultrasonic sensor for measuring distances in robotics." In: *Journal of Physics: Conference Series*. Vol. 1015. 3. IOP Publishing, p. 032189 (cit. on p. 21).

Zuckerberg, Mark (2016). "The technology behind Aquila." In: (cit. on p. 11).

List of Figures

List of Tables