Automatic Design of Verifiable Robot Swarms

Coppola, M.

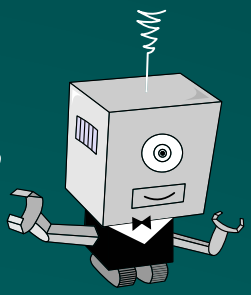
DOI
10.4233/uuid:b6ad7ddd-c660-4aab-8277-65f7a22a4a52

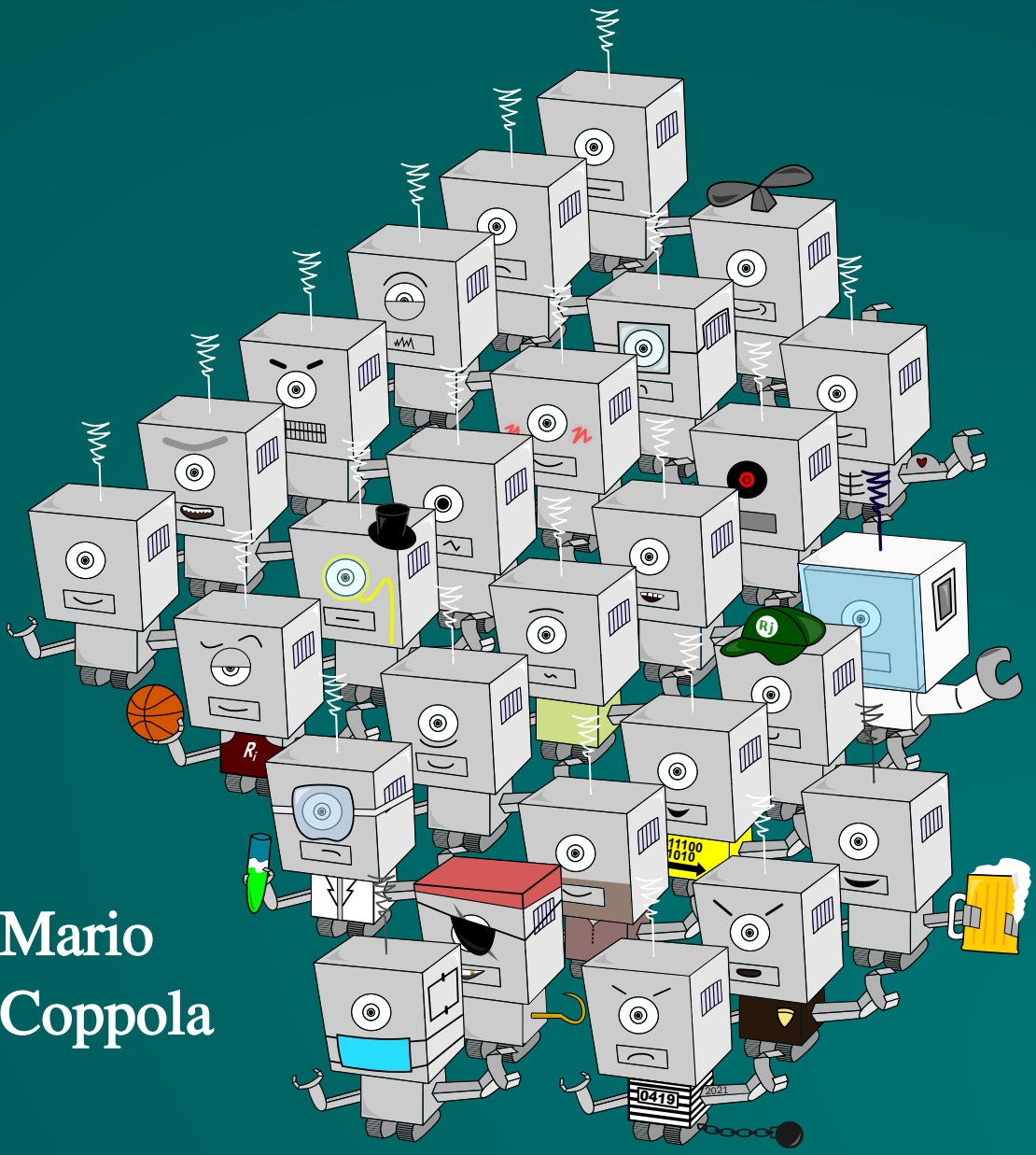
Publication date
2021

Document Version
Final published version

Citation (APA)
Coppola, M. (2021). *Automatic Design of Verifiable Robot Swarms*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:b6ad7ddd-c660-4aab-8277-65f7a22a4a52

Automatic Design of Verifiable Robot Swarms





Mario Coppola

AUTOMATIC DESIGN OF VERIFIABLE ROBOT SWARMS

AUTOMATIC DESIGN OF VERIFIABLE ROBOT SWARMS

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus Prof. dr. ir. T. H. J. J. van der Hagen,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op maandag 19 april 2021 om 15:00 uur

door

Mario COPPOLA

Ingenieur Luchtvaart en Ruimtevaart,
Technische Universiteit Delft, Nederland,
geboren te Grottaglie, Italië.

Dit proefschrift is goedgekeurd door de promotoren.

Samenstelling promotiecommissie bestaat uit:

Rector Magnificus,	voorzitter
Prof. dr. G. C. H. E. de Croon,	Technische Universiteit Delft, promotor
Prof. dr. E. K. A. Gill,	Technische Universiteit Delft, promotor
Dr. J. Guo,	Technische Universiteit Delft, promotor
Onafhankelijke leden:	
Prof. dr. M. Dorigo,	Université Libre de Bruxelles
Prof. dr. J. P. How,	Massachusetts Institute of Technology
Prof. dr. ir. T. Keviczky,	Technische Universiteit Delft
Dr. V. Trianni,	Consiglio Nazionale delle Ricerche
Prof. dr. ir. M. Wisse,	Technische Universiteit Delft, reservelid

Keywords: swarm intelligence, swarm robotics, distributed systems, robots, micro air vehicles, machine learning, verification

Printed by: Ipskamp printing, www.ipskampprinting.nl

Front & Back: Mario Coppola (with suggestions by Jun Hyeon Lee, Diana Olejnik, and Ren Smis)

ISBN 978-94-6421-287-7

An electronic version of this dissertation is available at
`http://repository.tudelft.nl/`.

You are never more than a half step away from a right note.

— Victor Wooten

Contents

SUMMARY

The paradigm of swarm robotics aims to enable several independent robots to collaborate together toward collective goals, acting as a "swarm". The distributed nature of a swarm, whereby each robot acts independently in accordance with its perceived environment, is expected to provide the system with a high degree of flexibility, robustness, and scalability. However, this comes at the cost of increased system complexity. This thesis explores how to automatically design a collective behavior in a way that is transparent and verifiable.

We begin by taking a step back and analyzing the design choices that need to be made when designing a swarm of robots. At the "local" level, the individual robots needs to be designed, specifying, for instance, their actuators, sensors, and computational capabilities. At the system level (or "global" level), the desired behavior of the collective system needs to be designed. Popular examples of collective behaviors are: aggregation (coming together into a group), pattern formation, area exploration, or foraging (gathering objects from the environment). Through an in-depth literature study, focusing on swarms of small drones as a case study, we found how sensor and actuator choices can create constraints for the swarm behavior that can be achieved, and how desired swarm behaviors can create requirements for the hardware design and local-level controllers. Coincidentally, we found a prominent example of this in our own research on relative localization sensors for swarms of tiny drones (performed in addition to the research in this thesis). We developed a communication-based relative localization approach that enabled teams of tiny drones to fly together in tight areas, the advantages being: omnidirectional sensing, independence from lighting conditions and/or visual clutter, low mass, and low computational costs. However, this solution also comes with the restriction of ensuring that robots never move parallel to each other, as this will present an unobservable situation. Based on such lessons, the remainder of the thesis then aims for a framework that is agnostic with respect to the robot and the task. The framework proposed in this thesis is centered around the following notion: a collective goal can be broken down into a set of locally observable objectives which the robots can sense, referred to as "desired" objectives. The robots then take actions in order to reach these desired objectives. When all robots achieve the desired objectives, then the global goal and/or collective behavior emerges.

This framework was first developed for the specific case study of pattern formation by cognitively limited robots, which could only sense the relative location of close-by neighbors. It was later generalized, and its use was demonstrated on other collective tasks, namely: aggregation, consensus, and foraging. Through a local model of agent transitions, it was possible to: 1) identify potential obstructions to achieving the collective goal, and 2) optimize the behavior of the robots so as to maximize the likelihood of achieving the desired objectives. The optimization is performed by an evolutionary algorithm that leverages the local model, whereby the fitness function maximizes the

probability of being in a desired local state. Using this approach, the policy evaluation only scales with the size of the local state space, and demands much less computation than swarm simulations would.

In the final stage of this research, a framework was developed to alleviate the need to manually define the desired objectives as well as the local models required for potential verification and/or optimization. The framework uses a data-driven approach to automatically extract two models: 1) a deep neural network that estimates the global performance of the swarm from the distribution of local sensor data, and 2) a probabilistic state transition model that explicitly models the local state transitions (i.e., transitions in observations from the perspective of a single robot in a swarm) given a policy. The framework can efficiently lead to effective controllers, as demonstrated via multiple case studies. It can also be used in combination with an evolutionary optimization process, leading to higher efficiency, or for heterogeneous online learning.

Overall, the methods and insights developed in this thesis propose a new way to approach the development of verifiable and understandable behaviors for swarms of robots, using models in order to perform analysis, verification, and optimization.

SAMENVATTING

Het paradigma van de zwermrobotica heeft als doel om verschillende onafhankelijke robots samen te laten werken aan collectieve doelen, soortgelijk aan een “zwerm”. Het gedistribueerde karakter van een zwerm, waarbij elke robot onafhankelijk handelt in overeenstemming met zijn waargenomen omgeving, biedt het system een hoge mate van flexibiliteit, robuustheid en schaabaarheid, naar verwachting. Dit gaat echter ten koste van de complexiteit van het systeem. In deze proefschrift wordt onderzocht hoe automatisch een collectief gedrag kan worden ontworpen op een manier die transparant en verifieerbaar is.

We beginnen met een stap terug te nemen en de ontwerpkeuzes te analyseren die gemaakt moeten worden bij het ontwerpen van een zwerm robots. Op het “lokale” niveau moeten de individuele robots worden ontworpen, waarbij bijvoorbeeld hun actuatoren, sensoren, en rekencapaciteiten worden gespecificeerd. Op systeemniveau (of “lobale” niveau) moet het gewenste gedrag van het collectieve systeem worden ontworpen. Populaire voorbeelden van collectief gedrag zijn: aggregatie (samenkomen in een groep), patroonvorming, gebiedsverkenning, of foerageren (voorwerpen uit de omgeving verzamelen). Via een diepgaande literatuurstudie, waarbij we ons concentreerden op zwermen van kleine drones als casestudy, ontdekten we hoe sensor- en actuatorkeuzes beperkingen kunnen creëren voor het zwermgedrag dat kan worden bereikt, en hoe gewenst zwermgedrag eisen kan stellen aan het hardwareontwerp en de regelaars op lokaal niveau. Toevallig vonden we een prominent voorbeeld hiervan in ons eigen onderzoek naar relatieve lokalisatiesensoren voor zwermen kleine drones (uitgevoerd naast het onderzoek in deze proefschrift). We ontwikkelden een communicatie–gebaseerde relatieve lokalisatie aanpak die teams van kleine drones in staat stelde om samen te vliegen in krappe gebieden, met als voordelen: omnidirectionele sensing, onafhankelijkheid van lichtcondities en/of visuele clutter, lage massa, en lage computerkosten. Deze oplossing heeft echter ook de beperking dat de robots nooit evenwijdig met elkaar mogen bewegen, omdat dit een onwaarneembare situatie zal opleveren. Op basis van deze lessen wordt in de rest van het proefschrift gestreefd naar een raamwerk dat agnostisch is met betrekking tot de robot en de taak. Het raamwerk dat in deze proefschrift wordt voorgesteld is gecentreerd rond het volgende begrip: een collectief doel kan worden opgesplitst in een verzameling van lokaal-waarneembare doelen die de robots kunnen observeren, aangeduid als “gewenste” doelen. De robots bewegen vervolgens door de omgeving om deze gewenste doelen te bereiken. Wanneer alle robots de gewenste doelen bereiken, dan ontstaat het globale doel en/of collectief gedrag.

Dit raamwerk werd eerst ontwikkeld voor het specifieke geval van patroonvorming door cognitief beperkte robots, die alleen de relatieve locatie van nabije buren konden waarnemen. Het werd later veralgemeend, en het gebruik ervan werd gedemonstreerd op andere collectieve taken, namelijk: aggregatie, consensus, en foerageren. Door middel van een lokaal model van agent-transities, was het mogelijk om: 1) potentiële belem-

meringen voor het bereiken van het collectieve doel te identificeren, en 2) het gedrag van de robots te optimaliseren om de kans op het bereiken van de gewenste doelen te maximaliseren. De optimalisatie wordt uitgevoerd door een evolutionair algoritme dat gebruik maakt van het lokale model, waarbij de fitnessfunctie de waarschijnlijkheid maximaliseert om in een gewenste lokale toestand te zijn. Door deze aanpak schaalt de beleidsevaluatie enkel met de grootte van de lokale toestandsruimte, en vergt ze veel minder rekenwerk dan zwermsimulaties zouden vergen.

In de laatste fase van dit onderzoek werd een raamwerk ontwikkeld om de noodzaak te verlichten van het handmatig definiëren van de gewenste doelstellingen, alsmede van de lokale modellen die nodig zijn voor eventuele verificatie en/of optimalisatie. Het raamwerk gebruikt een datagedreven benadering om automatisch twee modellen te extraheren: 1) een diep neuraal netwerk dat de globale prestaties van de zwerm schat uit de verdeling van lokale sensorgegevens, en 2) een probabilistisch toestandsovergangsmodel dat expliciet de lokale toestandsovergangen modelleert (d.w.z. overgangen in waarnemingen vanuit het perspectief van een enkele robot in een zwerm) gegeven een beleid. Het raamwerk kan efficiënt leiden tot effectieve controllers, zoals aangetoond via meerdere case studies. Het kan ook gebruikt worden in combinatie met een evolutionair optimalisatieproces, wat leidt tot een hogere efficiëntie, of voor heterogeen online leren.

Samenvattend stellen de methoden en inzichten die in dit proefschrift zijn ontwikkeld een nieuwe manier voor om de ontwikkeling van verifieerbaar en begrijpelijk gedrag voor zwermen robots te benaderen, waarbij modellen worden gebruikt om analyse, verificatie en optimalisatie uit te voeren.

1

INTRODUCTION

The goal of robotics is to create advanced machines that can physically interact with their environment. Nature is perhaps our greatest inspiration. From the simplest of robotic arms to full-scale humanoid robots or flapping wing drones, the animal kingdom never ceases to provide us with examples of how successful robots could be designed. Yet, one of the most interesting displays of nature is not necessarily observed when we study individual entities, like a particular ant or bird, but in the way that they collaborate with other members of their species. This collaboration can take several forms: from simple acts, such as a mother bringing food to her offspring, to more complex interactions, such as the construction of shelter by entire communities. A community, as a whole, is much more complex than the individuals that compose it, and can also achieve more complex and ambitious goals as a result. The examples are numerous. Ants help each other to find and carry food effectively. Fish work together to fend off and confuse their bigger predators. By enabling robots to collaborate together in a swarm-like fashion, we can hope to unlock new potentials in robotics. The challenge, however, lies in efficiently guiding a robotic swarm toward the successful completion of a common goal of our choice. This is especially challenging, because each individual robot, much like the individual animal, only has a very limited view of the environment as given by its onboard sensors. In this thesis, we will explore how we can control swarms of robots and enable them to collaborate successfully.

1.1. EMERGENCE, SWARM ROBOTICS, AND ITS HURDLES

Swarms are often said to display emergent properties. Across fields ranging from particle physics to social studies, the concept of emergence can be used to describe a process for which the outcome results from the complex (inter-)actions of its consitutent parts. It is then often said that "the whole is more than the sum of its parts", as it is only when the parts interact that the system, as a whole, achieves the emergent outcome. Emergent outcomes may not be directly obvious when observing the system's constituent elements, because the result emerges from the complex network of interactions between them, which may carry a certain degree of unpredictability. Although emergent processes are generally not well understood, their potential is greatly acknowledged.[1]

[1] Even the very definition of emergence is often the subject of debate. Several (re-)definitions have been attempted and contemplated, without a clear consensus (Deguet et al., 2006; Hamann, 2018).

If we could find a way to "engineer" emergence for swarms of robots, then we would be able to create collective multi-robot systems whose cumulative capabilities transcend the limitations of any of the individual robots that compose it. We would be able to use several simple and inexpensive robots, and still fulfill complex goals by exploiting their (inter-)actions. Such systems are envisioned to be robust to the loss of individual robots, flexible and adaptable to different tasks via reconfiguration, and scalable in the number of robots (Şahin et al., 2008). *This* is the ambition of *swarm robotics*.

The complexity of swarm robotics stems from the fact that each robot only has the information that can be measured within its direct environment, via its onboard sensors. We refer to this as the "local" information, or the "microscopic" view. Meanwhile, the actions of each robot can have "global" (or "macroscopic") repercussions on the whole swarm, something which the individual robot cannot directly observe. Further complexity stems from the fact that the actions of a robot may, in turn, cause another robot to take a different decision, and so on. This creates a non-deterministic dynamic environment for each robot to navigate through. The field of swarm robotics aims to tackle this challenge with the goal of achieving desired and predictable collective behaviors from simple robots and their interactions, so that we can use them in real-world tasks.

1.2. SWARM ROBOTICS: A BRIEF OVERVIEW

Swarm robotics is the application of swarm intelligence to robots. Several early works that studied swarms did so for purposes outside of robotics. Reynolds (1987) published a pioneering work on swarming in the context of computer graphics. The work showed that it was possible to visually simulate large bird-like flocks by only combining three very simple rules that each simulated entity should follow: 1) cohesion, 2) repulsion, and 3) alignment. In simpler terms, each simulated bird aimed to: 1) be close to its nearby neighbors, 2) avoid getting too close, and 3) align its direction of flight with them. Together, these rules resulted in behaviors that appeared very similar to bird flocks to the external observer.[2] Meanwhile, the idea of swarm intelligence had also found its way to optimization algorithms, such as the Ant Colony Optimization algorithm published by Dorigo et al. (1996), whereby simulated ants offered an effective way to find optimal paths. In the field of biology, there was also a drive to understand the rules that these complex systems exhibited in nature (Bonabeau et al., 1999). Thanks to all these studies, two notions had become clearly established:

1. Swarm intelligence can be a powerful paradigm to achieve complex goals effectively.
2. These complex goals can be achieved with simple individuals following simple rules.

Combined with the many concurrent improvements in robotics, applying swarm intelligence to robots became an attractive opportunity (Beni, 2005). This can enable robots to achieve more challenging tasks while keeping the individual unit cost low, lending itself to economies of scale. In addition, the system as a whole has the potential of becoming

[2] This approach is still very popular in modern applications, see for instance the works of Hauert et al. (2011) and Vásárhelyi et al. (2018).

more robust to the loss/malfunctioning of individual robots, flexible to different tasks (by changing the way in which the robots collaborate), and scalable (Brambilla et al., 2013; Hamann, 2018; Şahin, 2005).

Early works in swarm robotics include the works of Matarić et al. (1995), Matarić (1997), Hayes et al. (2001), and Lerman et al. (2001). In the last two decades, the research field has grown and several additional studies have been published, including dedicated real world demonstrations, such as Swarm-bot (Mondada et al., 2004) and Swarmanoid (Dorigo et al., 2013), which showcase practical use cases for robot swarms. Several applications can be unlocked by creating a distributed robotic system. Swarms have been developed to explore areas efficiently (by adequately spreading and covering more ground) (Duarte et al., 2016), move large and/or heavy loads by uniting forces (Dorigo et al., 2013), or form shapes and structures (Rubenstein et al., 2014; Werfel et al., 2014). The potential applications also extend to flying vehicles or satellites. For example, the OLFAR mission aimed to use a large swarm of satellites as a distributed radio interferometer (Engelen et al., 2014; Verhoeven et al., 2011). Similarly, with the rising popularity of drones in the last decade, a lot of swarm robotics research has also expanded into this novel domain (McGuire et al., 2019; Vásárhelyi et al., 2018). Overall, swarm robotics is now regarded as one of the main challenges in robotics (Yang et al., 2018).

The objective in swarm robotics is to engineer the correct local rules that lead to a desired goal reliably when deployed in the real world (Beni, 2005). It is also important that the swarm achieves the goal without exhibiting unexpected behaviors, as these could lead to safety hazards (Winfield et al., 2005a). Developing local rules and behaviors manually, although seemingly attractive and with many successful examples, has two general disadvantages: 1) it is limited by the ability of the designer to model and solve the problem at hand, and 2) it needs to be repeated ad hoc for each new task. Therefore, even if we were to design a perfect behavior, complete with a proof that the behavior will lead to the desired goal, this would only be specific to the particular system and task at hand. We can abstract away from this issue by using machine learning techniques to *automatically* find the correct behavior, a strategy that has been immensely successful in other branches of robotics and artificial intelligence, such as decision-making (Silver et al., 2016, 2017), computer vision (Redmon et al., 2016), or locomotion (Kolter and Ng, 2011), to name a few examples. The power of machine learning approaches has also been established in the swarming community through several works, many of which also bring the controllers outside of the simulation domain and into the real world (Duarte et al., 2016; Trianni et al., 2003, 2006). Among the machine learning paradigms used in swarm robotics, evolutionary robotics (Floreano and Nolfi, 2000) has taken the limelight over reinforcement learning, which, although highly effective, generally faces more issues with partially observable domains (Oliehoek and Amato, 2016; Sutton and Barto, 2018).

Despite many impressive developments on all fronts, which we will return to in Chapter 2, a mature methodology for the automatic development of local behaviors starting from a global goal is still to be established (Francesca and Birattari, 2016). In addition, although many current machine learning methodologies offer a way to find suitable controllers, the methods to non-empirically verify the functionality of these controllers remain limited for the general case. This is either because the controllers are not transparent enough for analysis (e.g., neural networks), or because of scalability issues in the

verification procedure when analyzing all potential repercussions of a microscopic behavior for the full swarm (Dixon et al., 2012). Even in the absence of verification procedures, transparency is a very desirable property, because it allows us to inspect and understand the behavior of the robot before it is implemented (Jones et al., 2019).

1.3. RESEARCH OBJECTIVE

In this thesis, we focus on two fundamental open challenges in swarm robotics:

1. The top-down automatic development of local rules from a global goal.
2. The bottom-up verification of whether the local rules will lead to the desired global goal.

The aim is to develop an end-to-end method to automatically extract robot behaviors that will lead to the desired result, with the inclusion of an automatic method to check whether the swarm will eventually achieve this desired result. We are guided by the following research question:

> **Guiding research question:** How can we automatically design a swarm of robots to systematically and efficiently produce a desired emergent result?

To provide a solid and irrefutable answer to the above would mean to solve one of the main questions in the field of swarm robotics, and I do not have sufficient hubris to claim that this was (or could ever be) achieved in the mere four years that I have dedicated to it. However, very much in line with a swarming philosophy, I hope that the contributions that have been made in this thesis provide original insights as well as new methods that can help the community move forward. The specific objective, exploratory in nature, is the following:

> **Research Objective:** To develop a novel methodology to automatically design the behavior of a swarm of robots such that they can efficiently and successfully achieve a cooperative goal, while taking into account their local constraints.

Such a goal is intended to provide an answer to the main research question to the best extent possible, the results of which are recounted in this thesis.

1.4. RESEARCH QUESTIONS AND THESIS OUTLINE

The main research objective was broken down into four main steps. This enabled an exploratory approach from which we could develop a final solution.

1.4.1. UNDERSTANDING THE CHALLENGE IN DEPTH

As a first step, we explored the practical issues with developing a real robot swarm — what exactly makes it so difficult?

Research Question 1: What are the challenges of developing a successful swarm of robots?

To answer this question with depth, we focused on the specific case study of aerial robotics, which features an interesting mix of challenges for swarm robotics (Kumar and Michael, 2012; Yang et al., 2018). For this particular case study, we delved into the entire design process in order to extract and analyze challenges that need to be solved at each stage. This analysis showed just how limited a real swarm can be, and that the primary issue of swarming lies in the sensory information available to the control system. Quite basic knowledge, such as knowing the relative location of neighboring robots, can be very difficult to obtain, and may even impose certain restrictions on the way in which each robot should behave. This creates a fundamental link between sensory information and the actions that a robot in the swarm can take. Additional components, such as effective intra-swarm communication, also face several challenges as the size of the swarm grows. The result of this survey, published in Coppola et al. (2020), can be found in Chapter 2. Mapping out all underlying relationships showed a clear need to better understand how a very limited local perception can be exploited.

1.4.2. A NOVEL METHOD FOR PROVABLE PATTERN FORMATION

Based on the insights from the first work, we set out to explore the relationship between a local objective and local sensory data. To define the scope without losing ourselves in generality, we focused on a specific case study: pattern formation. This was chosen because it is a fixed goal with a binary outcome — a desired pattern will either form, or fail to do so. We strongly limited the sensory knowledge of the robots such that they were only aware of their current closest neighborhood (and nothing else), and then aimed to devise a behavior that enabled the robots to generate the pattern based on local actions. Our research question was:

Research Question 2: How can a swarm of cognitively limited robots arrange in an arbitrary formation, in a way that provably leads to success?

The outcome of this research was a novel algorithm to achieve pattern formation based on an automatically generated probabilistic policy. The policy is generated based on three simple bio-inspired rules:

- Be “safe” (avoid collisions)
- Be “social” (avoid moving away from peers)
- Be “happy” (stay in *desired* local states)

The first two rules allow the swarm to reshuffle freely while never separating in multiple groups or experiencing collisions. The third rule tells robots that, if its local neighborhood matches one of the pre-specified *desired* local states, then it should stop moving. Here, a local state is a state describing the current observation of a single robot. Then, unbeknownst to the robots, the global desired pattern is achieved once all robots are “happy”. In addition, based on a model of local interactions and states, we could also

verify whether a pattern would always eventually be formed without endless reshuffling. The advantage of this verification procedure was that it was based on a local model, meaning that it did not scale with the size of the swarm, but with the (quite limited) size of the local state space. The outcome of the research was published in Coppola et al. (2019b) and is recounted in Chapter 3.

1.4.3. GENERALIZING AND OPTIMIZING THE BEHAVIOR OF SWARMS

The contribution above provided us with a methodology to lead swarms to achieve a global pattern and with a way to verify whether the outcome would always eventually happen. The methodology was to break down the pattern into desired local states, shared by all robots, which would then enable them to reconstruct the pattern in spite of their limitations. At this stage, we were faced with two subsequent challenges:

1. Abstracting the methodology to cooperative tasks other than pattern formation.
2. Optimizing the behavior of the robots so that they would achieve the global goal more efficiently, as opposed to moving around randomly until it was achieved.

These two challenges were synthesized in the research question below.

Research Question 3: How can a swarm of robots coordinate to achieve a global goal efficiently?

The main output of this research was a novel way to use a local model of the experiences of a robot in order to optimize the global performance of the swarm. Based on a model of the local state transitions, we optimized a controller so as to increase the probability that the robot would achieve a "happy" local states, which is a representation of the global goal from the local perspective of an agent. This approach allows for the controller to be optimized only based on a local model, thus avoiding several of the otherwise common scalability issues that arise in such optimization problems. When the controller is applied to the whole swarm, it shows major improvements in performance at the macroscopic level. This procedure was applied to the pattern formation method from (Coppola et al., 2019b), and was also generalized and applied to two new case studies: consensus and aggregation.[3] For the pattern formation, the previously developed local proof procedure could be included in the optimization procedure. The contribution of this work was published in Coppola et al. (2019a), and it is recounted in Chapter 4.

1.4.4. AN END-TO-END FRAMEWORK

In this final task, we aimed to complete the end-to-end framework that we set out to develop in the beginning. So far, in addition to our in-depth survey, we had achieved two main contributions: 1) a method to extract local states that represent the global goal, and 2) an approach to efficiently extract/optimize the swarm behavior. The final goal was to develop a framework where a global goal is translated into a local behavior. Our fourth and final research question was:

[3] Consensus refers to when several robots have to form an agreement. Aggregation refers to when several robots have to come together into one group.

Research Question 4: How can we design the behavior of a swarm of robots such that it reliably achieves a cooperative goal?

To achieve this objective, two new challenges needed to be overcome.

1. Automatically breaking down a global cooperative goal into desired local states, for the general case.
2. Automatically extracting a local model of the swarm, so as to provide a means to optimize the behavior (using the PageRank approach) and verifying whether the global goal can always eventually be achieved.

We showed how these two goals can be achieved using a data-driven model-based approach. By creating a data set we can train a deep neural network to correlate local states with the global fitness. In turn, despite having a data set of random behaviors with poor performance, we can extrapolate the trend and extract which combination of local states is expected to maximize the global fitness. Using the same data set, we can also extract a probabilistic state transition model of the system, which can be further refined if needed. Altogether, the framework can determine a transparent and efficient behavior for a swarming task. Thanks to the model, we can also analyze it to find potential pitfalls using quantifiable metrics and/or formal conditions. The output of this work can be found in Chapter 5.

This thesis ends in Chapter 6 with a summary of our main contributions as well as an explanation of their implications for future work in the field of swarm robotics.

2

ROBOT SWARMS: FUNDAMENTAL CHALLENGES AND CONSTRAINTS

This chapter presents an in-depth review and discussion of the challenges that must be solved in order to successfully develop swarms of robots. As a particular case study, we will focus on Micro Air Vehicles (MAVs) for real world operations. We will extract constraints and links that relate the local level MAV capabilities to the global operations of the swarm. It will be explained how these should be taken into account when designing swarm behaviors in order to maximize the utility of the group, which will become a foundational pillar in future chapters. At the lowest level, each MAV should operate safely. Robustness is often hailed as a pillar of swarm robotics, and a minimum level of local reliability is needed for it to propagate to the global level. At the swarm level, the final outcome is intrinsically influenced by the onboard abilities and sensors of the individual. The real-world behavior and operations of an MAV swarm intrinsically follow in a bottom-up fashion as a result of the local level limitations in cognition, relative knowledge, communication, power, and safety. Taking these local limitations into account when designing a global swarm behavior is key in order to take full advantage of the system, enabling local limitations to become true strengths of the swarm.

The contents of this chapter have been published in Coppola et al. (2020).

2.1. BACKGROUND

Micro Air Vehicles (MAVs), or "small drones", are becoming commonplace robots in the modern world. The term refers to small, light-weight, flying robots. Several MAV designs exist, including multi-rotors (Kumar and Michael, 2012), flapping wing (de Croon et al., 2016; Michelson and Reece, 1998; Wood et al., 2013), fixed wing (Green and Oh, 2006), morphing designs (Falanga et al., 2019b), or hybrid vehicles (Itasse et al., 2011). Of these, quadrotors have enjoyed the spotlight due to their high maneuverability, their ability to take off vertically (as opposed to most fixed wing MAVs, for instance), and their relative simplicity in design (Gupte et al., 2012; Kumar and Michael, 2012). MAVs can be used for surveillance and mapping (Mohr and Fitzpatrick, 2008; Saska et al., 2016b; Scaramuzza et al., 2014), infrastructure inspection (Sa and Corke, 2014), load transport and delivery (Palunko et al., 2012), or construction (Augugliaro et al., 2014; Lindsey et al., 2012). Such applications are particularly useful in areas that are not easily accessible by humans, like forests or disaster sites (Achtelik et al., 2012; Alexis et al., 2009). Smaller and lighter designs push the boundaries of their applications further. Aside from the asset of increased portability, smaller MAVs can also navigate through tighter spaces such as narrow indoor environments with higher agility (Mohr and Fitzpatrick, 2008). They also cause less damage to their surroundings (including people) in the event of a collision, making them intrinsically safer tools (Kushleyev et al., 2013).

Unfortunately, smaller size comes at the expense of more limited capabilities. The interplay between limited flight time, limited sensing, and limited power hinder an MAV from performing grander tasks on its own. This has created a strong interest in developing MAV *swarms* (Yang et al., 2018). The paradigm of swarm robotics aims to transcend the limitations of a single robot by enabling cooperation in larger teams. This is inspired by the animal kingdom, where animals and insects have been observed to unite forces toward a common goal that is otherwise too complex or challenging for the lone individual (Garnier et al., 2007). Using several robots at once can bring several advantages and possibilities such as: redundancy, faster task completion due to parallelization, or the execution of collaborative tasks (Martinoli and Easton, 2003; Nedjah and Junior, 2019; Trianni and Campo, 2015). The control of robotic swarms is envisioned to be fully distributed. The individual robots perceive and process their environment locally and then act accordingly without global awareness or direct awareness of the final goal of the swarm. Nevertheless, by means of collaboration, the robots can achieve an objective that they would not have been able to achieve by themselves. As they say: there is strength in numbers.

It is easy to imagine swarms of MAVs jointly carrying a load that is too heavy for a single one to lift, or persistently exploring an area without interruption. As is often the case, however, putting such visions into practice is another story altogether. Developing self-organizing swarms of MAVs in the real world is a multi-disciplinary challenge coarsely divided in two main aspects. One aspect is that of the individual MAV design, where the local abilities of a single MAV are defined. The second aspect is the swarm design, whereby we need to develop controllers with which the global goal can be efficiently achieved, autonomously, by the swarm. To make matters more complicated, the two are not decoupled. As we shall explore in this chapter, there exist fundamental links between the local limitations of an MAV and the behaviors that a swarm of MAVs could,

or should, execute as a result. Vice versa, in order to realize certain swarm behaviors, there are local requirements that the individual MAVs must meet. This bond between the local and the global cannot be ignored if MAV swarms are to be brought to the real world. In this chapter, we aim to reconcile these two aspects and present a discussion of the fundamental challenges and constraints linking local MAV properties and global swarm behaviors. In the grander context of this thesis, we use MAVs as a case study in order to better understand the intricacies of developing swarms of robots, as well as the general state of the art in the field.

2.2. CO-DEPENDENCE OF SWARM DESIGN AND DRONE DESIGN

Let us begin from the primary challenge of swarm robotics: to design local controllers that successfully lead to global swarm behaviors (Şahin et al., 2008). Concerning MAVs, these global behaviors include, but are not limited to: collaborative transport, collaborative construction, distributed sensing, collaborative object manipulation, and parallelized exploration and mapping of environments. Albeit the individual MAV may be limited in its ability to successfully perform these tasks (for instance, as areas get larger or loads get heavier), they can be tackled by collaborating in a swarm. Generally, swarms of robots are expected to feature the following inherent advantages (Brambilla et al., 2013; Şahin et al., 2008):

- **Robustness**. The swarm is robust to the loss or failure of individual robots.
- **Flexibility**. The swarm can reconfigure to tackle different tasks.
- **Scalability**. The swarm can grow and shrink in size depending on the needs of the global task.

When designing a swarm of MAVs, we must then ask ourselves: how can we design a swarm that is robust, flexible, and scalable? It is true that these properties pertain to the swarm rather than the individual, but if the swarm is composed of individual units, then it follows that they must also be present (although perhaps not always apparent) at the local level. We cannot use individual robots that are not robust and merely expect the swarm as a whole to be immune or tolerant to individual failures (Bjerknes and Winfield, 2013). If there is a high probability of errors at the local level, such as erroneous observations, poorly executed commands, or failure of a unit, then this may have a repercussion on the swarm's performance; an effect that Bjerknes and Winfield (2013) have shown can worsen with the number of robots in a swarm. There is a point after which the individual robots are too unreliable and the swarm can fail to achieve its goal, or it can be shown to be outperformed by smaller teams with more reliable units (Stancliff et al., 2006) or even by a single reliable system (Engelen et al., 2014). The further complication with MAVs is that local failures do not remain local, but are likely to cause collisions and damages to other nearby MAVs and/or objects. For some tasks, such as collective transport, the impact may be even more severe as the MAVs are mechanically attached to the load (Tagliabue et al., 2019). It thus follows that to develop a robust swarm for real world deployment, we must also ensure robustness at the local level.

Of equal importance is to make sure that the robots have the necessary tools and sensors to carry out their individual components of a global task. The more capable the sensors are, the more likely it is that the swarm can be flexible and adjust to different

tasks or unexpected changes. When performing pure swarm intelligence research, we can afford to abstract away from lower-level issues (Brutschy et al., 2015). For instance, in a study on making a decision about selecting a new location for a swarm's nest, one can abstract away from actually evaluating the quality of a nest location, and instead focus the analysis on a particular aspect of the system such as the decision making process. However, when dealing with real-world applications, this is not an option. If we want to develop nest selection capabilities for a swarm in the real world, each robot should be capable of: flying and operating safely, recognizing the existence of a site, evaluating the quality of a site with a certain reliability, exchanging this information with its neighbors, and more. All these lower-level requirements need to be appropriately realized for the global-level outcome to emerge, or otherwise need to be accepted as limitations of the system. The way in which they are implemented shapes the final behavior of the swarm.

Last but not least, unless properly accounted for, there are scalability problems that may also occur as the swarm grows in size. Examples of issues are: a congested airspace whereby the MAVs are unable to adhere to safety distances, a cluttered visual environment as a result of the presence of several MAVs (thus obstructing the task), or poor connectivity as a result of low-range communication capabilities. To achieve scalability, the MAV design must be such that the swarm's desired properties are appropriately accommodated, from the appropriate hardware design all the way to the higher-level controllers which make up the swarm behavior.

2.2.1. THE CHALLENGE OF LOCAL SENSING AND CONTROL

When flying several MAVs at once, the control architecture can be of two types: 1) centralized, or 2) decentralized. In the centralized case, all MAVs in a swarm are controlled by a single computer. This "omniscient" entity knows the relevant states of all MAVs and can (pre-)plan their actions accordingly. The planning can be done a priori and/or online. In the decentralized case, the MAVs make their decisions locally. A second dichotomy can also be defined for how the MAVs sense their environment: 1) using external position sensing, or 2) locally. External positioning is typically achieved with a Global Navigation Satellite System (GNSS) or with a Motion Capture System (MCS), depending on whether the MAVs are flying outdoors or indoors, respectively. Alternatively, the latter only relies on the sensors that are onboard of the MAV.

Currently, the combination of centralized architecture and external positioning have achieved the highest stage of maturity, allowing for flights with several MAVs. Kushleyev et al. (2013) showed a swarm of 20 micro quadrotors that could reorganize in several formations. Lindsey et al. (2012), Augugliaro et al. (2014), and Mirjan et al. (2016) developed impressive collaborative construction schemes using a team of MAVs. Preiss et al. (2017) showcased Crazyswarm, an indoor display of 49 small quadrotors flying together. The strategy of centralized planning and external positioning has also attracted large industry investments, leading to shows with record-breaking number of MAVs flying simultaneously. In 2015, Intel and Ars Electronica Futurelab first flew 100 MAVs, making a Guinness World Record (Swatman, 2016a). In 2016, Intel beat its own record by flying 500 MAVs simultaneously (Swatman, 2016b). In 2018, EHang claimed the record with 1,374 MAVs flying above the city of Xi'an, China (Cadell, 2018). Intel later reclaimed the title by flying 2,066 MAVs outdoors. In September 2020, the record was broken yet again by

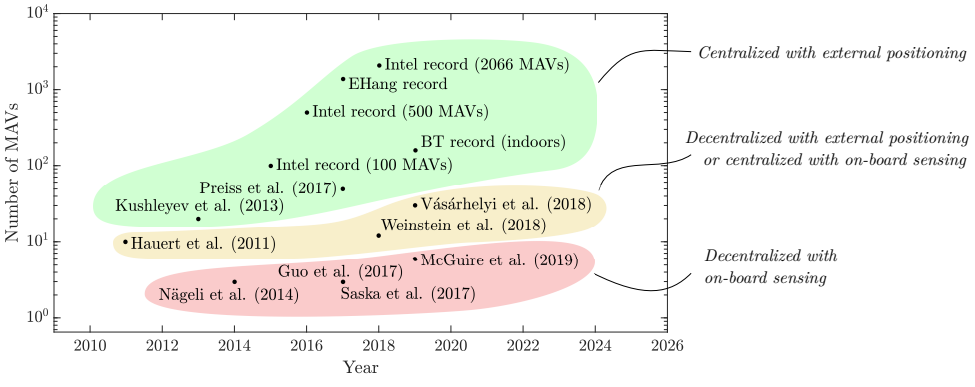


Figure 2.1 Scatter plot of the number of MAVs that have been flown in sampled state of the art studies discussed in this chapter. The combination of centralized planning/control with external positioning has allowed to fly significantly larger swarms. The numbers are lower for the works featuring decentralized control with external positioning, or centralized control with local sensing. The works that use both decentralized control and do not rely on external positioning can be seen to feature the fewest MAVs due to the increased complexity of the control and perception task.

Shenzhen Damoda Intelligent Control Technology Co., Ltd., in Zhuhai, China with 3,051 drones (Echo Zhan, 2020). Meanwhile, the record for the most MAVs flying indoors (from a single computer) was broken by BT with 160 MAVs in 2019 (Guinness World Records, 2019).

Without external positioning systems or centralized planning/control, the problem of flying several MAVs at once becomes more challenging. This is because: 1) the MAVs have to rely only on onboard perception, or 2) they have to make local decisions without the benefit of global planning, or 3) both. It is then not surprising that, as shown in Figure 2.1, the swarms that have been flown without external positioning and/or centralized control are significantly smaller. When the control is decentralized, but the MAVs benefit from an external positioning system, or vice versa, the largest swarms are in the dozens (Hauert et al., 2011; Vásárhelyi et al., 2018; Weinstein et al., 2018). For swarms featuring both local perception and distributed control, the highest numbers are currently in the single digits (Guo et al., 2017; McGuire et al., 2019; Nägeli et al., 2014; Saska et al., 2017). Despite the fact that these numbers have been increasing in the last few years, they are still lower, as the operations are shifted away from external system and toward onboard perception and control. If the past is any indication for the future, we expect that: 1) the numbers of drones will keep increasing for all cases, and 2) the industry will continue to break new records as the technologies for onboard decision making and perception become more mature.

Although we can fly a high number of MAVs when using centralized planning and external positioning, swarming is not just a numbers game. Flying with many MAVs does not automatically imply that we are achieving the benefits of swarm robotics (Hamann, 2018). A centralized system relies on a main computer to take all decisions. This means that a prompt online replanning by the main computer is needed in order to achieve robustness and flexibility in case of changes. This replanning grows in complexity with the

size of the swarm, making the system unscalable. Moreover, the central computer represents a single point of failure. Instead, a swarm adopts a distributed strategy whereby each robot takes decisions independently. The fact that each MAV needs to take its own decisions, and, additionally, if the MAVs do not rely on external infrastructure, introduces a new layer of difficulty. However, this is also what brings new advantages: redundancy, scalability, and adaptability to changes (Bonabeau and Theraulaz, 2008; Şahin, 2005).[1]

When we analyze swarms of MAVs with local onboard sensing and control, we can observe two trends: 1) As the size of the swarm increases, the *relative* knowledge that each MAV will have of its global environment, which includes the remainder of the swarm, decreases (Bouffanais, 2016); 2) as the individual MAV's size and/or mass decreases, its capability to sense its own local environment decreases (Kumar and Michael, 2012). This creates an interesting challenge. On the one hand, we aim to design smaller, lighter, cheaper, and more efficient MAVs. On the other hand, as we make these MAVs smaller, the gap between the microscopic and macroscopic widens further. Designing the swarm becomes a more challenging task because each MAV has less information about its environment and is also less capable to act on it. This can be generalized to other robotic platforms as well, but MAVs feature the increased difficulty of having a tightly bound relationship between their onboard capabilities, their dynamics, their processing power, and their sensing (Chung et al., 2018). This is sometimes referred to as the SWaP (Size, Weight, and Power) trade-off (Liu et al., 2018; Mahony et al., 2012). The relationship is often complex. For many MAVs, especially the smaller varieties, grams and milliwatts matter (de Croon et al., 2016). This makes the design of autonomous decentralized swarms of MAVs a more unique challenge.

2.2.2. OVERVIEW OF CHALLENGES THROUGHOUT THE DESIGN CHAIN

Throughout this chapter, we shall review the state of the art in MAV technology from the swarm robotics perspective. To facilitate our discussion, we will break down the challenges for the design and control of an MAV swarm in the following four levels, from *local* to *global*.

1. **MAV design**. This defines the processing power, flight time, dynamics, and capabilities of the single MAV. Most importantly from a swarm engineering perspective, it defines the sensory information available on each unit, from which it can establish its view of the world. This is discussed in Section 2.3.

2. **Local ego-state estimation and control**. At the lowest level, an MAV must be capable of controlling its motion with sufficient accuracy. This lower-level layer handles basic flight operations of the MAV. This includes attitude control, height control, and velocity estimation and control. Moreover, the MAV should be capable of safely navigating in its environment. Minimally, it should detect and avoid potential obstacles. The challenges and state of the art for these methods are discussed in Section 2.4.

[1] Of course, flying several MAVs with a centralized controller has its own challenges, which we do not mean to undermine. We only mean that it appears that these methods are at a more mature stage when compared to self-organized approaches, which is the focus of this work.

3. **Intra-swarm relative sensing and avoidance**. There are two key enabling technologies for swarming. The first is the knowledge on (the location of) nearby neighbors. This is particularly important for MAV swarms as it not only enables several higher-level swarming behaviors, but it also ensures that MAVs do not collide with one another in mid-air. The second enabling technology is communication between MAVs, such that they can share information and thus expand their knowledge of the environment via their neighborhood. These are is discussed in Section 2.5.

4. **Swarm behavior**. This is the higher-level control policy that the robots follow to generate the global swarm behavior. Examples of higher-level controllers in swarms range from attraction and repulsion forces for flocking (Gazi and Passino, 2002; Vásárhelyi et al., 2018) to neural networks for aggregation, dispersion, or homing (Duarte et al., 2016). We discuss how MAV swarm behaviors can be designed in Section 2.6.

Other similar taxonomies have been defined. Floreano and Wood (2015) describe three levels of robotic cognition: sensory-motor autonomy, reactive autonomy, and cognitive autonomy. Meanwhile, de Croon et al. (2016) divide the control process for autonomous flight into four levels: attitude control, height control, collision avoidance, and navigation. Although the taxonomies above are conceptually similar (generally going from low-level sensing and control to a higher level of cognition), the redefinition that we provide here is designed to better organize our discussion within the context of swarm robotics. Moreover, we also include the design of the MAV within the chain. As we will explain in this manuscript, this has a fundamental impact on the higher-level layers.

The four stages that have been defined have an increasing level of abstraction. The lower levels enable the robustness, flexibility, and scalability properties expected at the higher level, while the higher levels dictate, accommodate, and make the most out of the capabilities set at the lower level. From a systems engineering perspective, the MAV design poses constraints on what the higher-level controllers can expect to achieve, while the higher-level controllers create requirements that the MAV must be able to fulfill. A simplified view of the flow of requirements and constraints is shown in Figure 2.2.

Throughout the remainder of this chapter, as we discuss the state of the art at each level, we will highlight the major constraints that flow upward and the requirements that flow downward. Naturally, each sub-topic that we will treat features a plethora of solutions, challenges, and methods, each deserving of a review paper of its own. It is beyond the scope (and probably far beyond any acceptable word limit, too) to present an exhaustive review about each topic. Instead, we keep our focus to highlighting the main methodologies and how they can be used to design swarms of MAVs. Where possible, we will refer the reader to more in-depth reviews on a specific topic.

2.3. MICRO AIR VEHICLE DESIGN

The differentiating challenge faced by a flying robot, namely (and somewhat trivially) the fact that it has to carry its own mass around, creates a strong design driver toward minimalism. Despite battery mass consisting of up to 20% to 30% of the total system

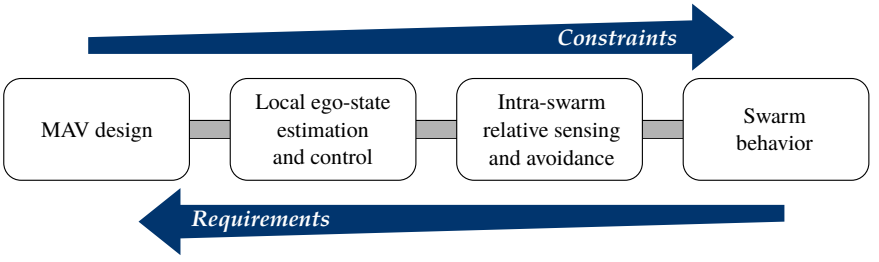


Figure 2.2 Generalized depiction of flow of requirements and constraints for the design of MAV swarms. The lower-level design choices create constraints on the higher-level properties of the swarm. Higher-level design choices create requirements for the lower levels, down to the physical design of the MAV. Note that, for the specific case, the flow of requirements and constraints is likely to be more intricate than this picture makes it out to be. However, the general idea remains.

mass, the flight time of quadrotor MAVs still remains limited to the order of magnitude of minutes (Kumar and Michael, 2012; Mulgaonkar et al., 2014; Oleynikova et al., 2015). To increase the carrying capabilities of an MAV, enabling it to carry more/better sensors, processors, or actuators, while keeping flight time constant, means that the size of the battery should also increase. In turn, this leads to a new increase in mass, and so on. This type of spiral, often referred to as the "snowball effect", is a well-known issue for the design of any flying vehicle, from MAVs to trans-Atlantic airliners (Lammering et al., 2012; Obert, 2009; Voskuijl et al., 2018). It then becomes paramount for an MAV design to be as minimalist as possible relative to its task, such that it may fulfill the mission requirements with a minimum mass (or, at the very least, there is a trade-off to be considered). This design driver has been taken to the extreme and has lead to the development of miniature MAV systems, popular examples of which include the Ladybird drone and the Crazyflie (Giernacki et al., 2017; Lehnert and Corke, 2013; Remes et al., 2014). These MAVs have a mass of less than 50 g, making them attractive due to their low cost and the fact that they are safer to operate around people. This makes them appealing for swarming, especially in indoor environments (Preiss et al., 2017).

A substantial body of literature already exists on single MAV design, the specifics of which largely vary depending on the type of MAV in question. We refer the reader to the works of Mulgaonkar et al. (2014) and Floreano and Wood (2015) and the sources therein for more details. From the swarming perspective, it is important to understand that, independently of the type of MAV in question, the following constraints are intertwined during the design phase: 1) flight time, 2) onboard sensing, 3) onboard processing power, and 4) dynamics. This means that the choice of MAV directly constrains the application as well as the swarming behavior that can be achieved (or, vice versa, a desired swarming behavior requires a specific type of MAV). For example, fixed wing MAVs benefit from longer autonomy. This makes them better candidates for long term operations, and also give the operators more time to launch an entire fleet and replace members with low batteries (Chung et al., 2016). However, fixed wing MAVs also have limited agility in comparison to quadrotors or flapping wing MAVs. The latter, for instance, can have a very high agility (Karásek et al., 2018), but also comes with more limited endurance and

payload constraints (Olejnik et al., 2019). The MAV design impacts the number and type of sensors that can be taken on board. It can also impact how these sensors are positioned and their eventual disturbances and noise. In turn, this affects the local sensing and control properties of the MAV and can also impact its ability to sense neighbors and operate in a team more effectively. We will return to these issues in the next sections of this chapter, whereby we discuss how an MAV can estimate and control its motion, sense its neighbors, and navigate in an environment together with the rest of the swarm.

A special note is made to designs that are *intended* for collaboration. Oung and D'Andrea (2011) introduced the Distributed Flight Array, a design whereby multiple single rotors can attach and detach from each other to form larger multi-rotors. More recently, Saldaña et al. (2018) introduced the ModQuad: a quadrotor with a magnetic frame designed for self-assembly with its neighbors. This design provides a solution for collaborative transport by creating a more powerful rigid structure with several drones. Gabrich et al. (2018) have shown how the ModQuad design can be used to form an aerial gripper. Because of the frame design, one of the difficulties of the ModQuad was in the disassembly back to individual quadrotors. This was tackled with a new frame design which enabled the quadrotors to disassemble by moving away from each other with a sufficiently high roll/pitch angle (Saldaña et al., 2019).

2.4. LOCAL EGO-STATE ESTIMATION AND CONTROL

The primary objective for a single MAV operating in a swarm is to remain in flight and perform higher-level tasks with a given accuracy. This requires a robust estimation of the onboard state as well as robust lower-level control, preferably while minimizing the size, power, and processing required. The design choices made here dictate the accuracy (i.e., noise, bias, and disturbances) with which each MAV will know its own state, as well as which variables the state is actually comprised of. In turn, this affects the type of maneuvers and actions that an MAV can execute. For instance, aggressive flight maneuvers likely require relatively accurate real-time state estimation (Bry et al., 2015). Of equal importance are the considerations for the processing power that remains for higher-level tasks. While it can be attractive to implement increasingly advanced algorithms to achieve a more reliable ego-state estimate, these can be too computationally expensive to run on board even by modern standards (Ghadiok et al., 2012; Schauwecker and Zell, 2014). This limits the MAV, as processing power is diverted from tasks at a higher-level of cognition. If not properly handled, it can lead to sub-optimal final performances by the MAVs and by the swarm.[2]

2.4.1. LOW-LEVEL STATE ESTIMATION AND CONTROL

This section outlines the main sensors and methods that can be used by MAVs to measure their onboard states, laying the foundations for our swarm-focused discussion in later sections. We organize the discussion by focusing on the following parameters: attitude, velocity and odometry, and height and altitude. Moreover, we restrict our overview

[2] When we relate this to nature, then low-level control and state-estimation seldom require large "computational" efforts by the individual animal. Rather, they eventually become second nature (Rasmussen, 1983). The real focus is directed to higher-level tasks.

to *onboard* sensing, as this is in line with the swarming philosophy and the relevant applications.

ATTITUDE

It is essential for an MAV to estimate and control its own attitude in order to control its flight (Beard, 2007; Bouabdallah and Siegwart, 2007). Angular rotation rates and accelerations are typically measured through the onboard Inertial Measurement Unit (IMU) sensor (Bouabdallah et al., 2004; Gupte et al., 2012). The IMU measurements can be fused together to both estimate and control the attitude of an MAV (Macdonald et al., 2014; Mulgaonkar et al., 2015; Schauwecker et al., 2012; Shen et al., 2011). Additionally to the IMU, MAVs equipped with cameras can also use it to infer the attitude with respect to certain reference features or planar surfaces, as in Schauwecker and Zell (2014). Thurrowgood et al. (2009), Dusha et al. (2011), de Croon et al. (2012a), and Carrio et al. (2018) estimate the roll and pitch angles of an MAV based on the horizon line (outdoors). The measurements from the IMU and vision can then be filtered together to improve the estimate as well as filter out the accumulating bias from the IMU (Martinelli, 2011). Once known, attitude control can be achieved with a variety of controllers. For a recent survey that treats the topic of attitude control in more detail, we refer the reader to the review by Nascimento and Saska (2019). Of particular interest to swarming are controllers that can provide robustness to disturbances or mishaps. One interesting example is the scheme devised by Faessler et al. (2015), which can automatically reinitialize the leveled flight of an MAV in mid-air.

Measuring and controlling the heading (for instance, with respect to north) is not strictly needed for basic flight. However, it can be an enabler for collective motion by providing a common reference that can be measured locally by all MAVs (Flocchini et al., 2008). Heading with respect to north can be measured with a magnetometer, which is a common component for MAVs (Beard, 2007). A main limitation of this sensor is that it is highly sensitive to disturbances in the environment (Afzal et al., 2011). The disturbances can be corrected for with the use of other attitude sensors. For example, Pascoal et al. (2000) fused gyroscope measurements with the magnetometer in order to filter out disturbances from the magnetometer while also reducing the noise from the gyroscope. Another sensor that has been explored is the celestial compass, which extracts the orientation based on the Sun (Dupeyroux et al., 2019; Jung et al., 2013). Although this sensor is not subject to electro-magnetic disturbances, it is limited to outdoor scenarios and performs best under a clear sky, which may also not always be the case.

VELOCITY AND ODOMETRY

A tuned sensor fusion filter with an accurate prediction model can estimate velocity just based on the IMU readings (Leishman et al., 2014). However, the use of additional and dedicated velocity sensors is commonly used to achieve a more robust system without bias. Fixed wing MAVs can be equipped with a pitot tube in order to measure airspeed (Chung et al., 2016). For other designs, such as quadrotors, a popular solution is to measure the optic flow, i.e., the motion of features in the environment, from which an MAV can extract its own velocity (Santamaria-Navarro et al., 2015). To observe velocity, the flow needs to be scaled with the help of a distance measurement such as height (albeit

this assumes that the ground is flat, which may be untrue in cluttered/outdoor environments). Optic flow can be measured with a camera or with dedicated sensors, such as PX4FLOW (Honegger et al., 2013) or the PixArt sensor.[3] Using optical mouse sensors, Briod et al. (2013) were able to make a 46 *g* quadrotor fly based on only inertial and optic flow sensors, even without the need to scale the flow by a distance measurement. This was achieved by only using the direction of the optic flow and disregarding its magnitude. In nature, optic flow has also been shown to be directly correlated with how insects control their velocity in an environment (Lecoeur et al., 2019; Portelli et al., 2011). Similar ideas have also been ported to the drone world, whereby the optic flow detection is directly correlated to a control input, without even necessarily extracting states from it (Zufferey et al., 2010). This can be an attractive property in order to create a natural correlation between a sensor and its control properties. State estimates improve when optic flow is fused with other sensors, such as IMU readings or pressure sensors (Kendoul et al., 2009a,b; Santamaria-Navarro et al., 2015), or with the control input of the drone (Ho et al., 2017). As opposed to optic flow sensors, a camera has the advantage that it can observe both optic flow as well as other features in the environment, thus enabling an MAV to get more out of a single sensor. Although this may be more computationally expensive, it also provides versatility.

The use of vision also enables the tracking of features in the environment, which a robot can use to estimate its odometry. Using Visual Odometry (VO), a robot integrates vision-based measurements during flight in order to estimate its motion. The inertial variant of VO, known as Visual Inertial Odometry (VIO), further fuses visual tracking together with IMU measurements. This makes it possible for an MAV to move accurately relative to an initial position (Scaramuzza and Zhang, 2019). VIO has been exploited for swarm-like behaviors, such as in the work by Weinstein et al. (2018), whereby twelve MAVs form patterns by flying pre-planned trajectories and use VIO to track their motion. A step beyond VO and its variants is to use Simultaneous Localization And Mapping (SLAM). The advantage of SLAM is that it can mitigate the integration drift of VO-based methods. When solving the full SLAM problem, a robot estimates its odometry in the environment and then corrects it by recognizing previously visited places and optimizing the result accordingly, so as to make a consistent map (Cadena et al., 2016; Cieslewski and Scaramuzza, 2017). Yousif et al. (2015) and Cadena et al. (2016) provide more in-depth reviews of VO and SLAM algorithms. Within the swarming context, a map can also be shared so as to make use of places and features that have been seen by other members of the swarm. One common drawback of VO and SLAM methods is that they are computationally intensive and thus reserved for larger MAVs (Ghadiok et al., 2012; Schauwecker and Zell, 2014). However, recent developments have also seen the introduction of more light-weight solutions such as Navion (Suleiman et al., 2018).

Odometry and SLAM are not limited to the use of vision. A viable alternative sensor is the LIDAR (LIght Detection And Ranging) scanner, more commonly referred to as *laser scanner*. LIDAR-based SLAM feature the same philosophy as the vision counterparts, but instead of a camera it uses LIDAR to measure depth information and build a map (Bachrach et al., 2011; Doer et al., 2017; Opromolla et al., 2016; Tripicchio et al., 2018). A LIDAR is generally less dependent on lighting conditions and needs less computations,

[3] See "PMW3901MB Product Datasheet" by PixArt Imaging Inc., June 2017.

but it is also heavier, more expensive, and consumes more onboard power (Opromolla et al., 2016). Vision and LIDAR can also be used together to further enhance the final estimates (López et al., 2016; Shi et al., 2016).

HEIGHT AND ALTITUDE

In an abstract sense, the ground represents an obstacle that the MAV must avoid, much like walls, objects, or other MAVs. It does not need to be explicitly known in order to control an MAV, as shown in the work of Beyeler et al. (2009). Unlike other obstacles, however, gravity continuously pulls the MAV toward the ground, meaning that measuring and controlling height and altitude often requires special attention.

Note that we differentiate here between height and altitude. Height is the distance to the ground surface, which can vary when there is a high building, a canyon, or a table. The height of an MAV can be measured with an ultrasonic range finder (or *sonar*). Sonar can provide more accurate data at the cost of power, mass, size, and a limited range. Its accuracy, however, made it a part of several designs (Abeywardena et al., 2013; Ghadiok et al., 2012; Krajník et al., 2011). Infrared or laser range finders have also been used as an alternative (Grzonka et al., 2009; Gupte et al., 2012). The advantage of an infrared sensor is that it can be very power efficient, albeit it is only reliable up to a limited range of a few meters, and on favorable light conditions (Laković et al., 2019)[4]. Altitude is the distance to a fixed reference point, such as sea level or a takeoff position. A pressure sensor is a common sensor to obtain this measurement (Beard, 2007), but it can be subject to large noise and disturbances in the short term, which can be reduced via low pass filters (Sabatini and Genovese, 2013; Shilov, 2014). If flying outdoors, a Global Navigation Satellite System (GNSS) can also be used to obtain altitude.

The choice of height/altitude sensor has an impact on the swarm behaviors that can be programmed. GNSS and pressure sensors provide a measurement of the altitude of the MAV with respect to a certain position. This is an attractive property, although, as previously noted, GNSS is limited to outdoor environments, while pressure sensors can be noisy. Moreover, all pressure sensors of all MAVs in the swarm should be equally calibrated. Unlike pressure sensors, ultrasonic sensors or laser range finders do not require this calibration step, since the measurement is made from the MAV to the nearest surface. However, one must then assume that the MAVs all fly on a flat plane with no objects (or other MAVs below them), which may turn out to not be a valid assumption. SLAM and VIO methods, previously discussed, can also estimate altitude/height as part of the odometry/mapping procedure provided that a downward facing camera is available.

Just as for the use of a common heading like north, the measurements of height and/or altitude can provide a common reference plane for a swarm of MAVs. If the vertical distance between the MAVs is sufficient, it can provide a relatively simple solution for intra-swarm collision avoidance (albeit with constraints — we return to this in Section 2.5.2). It can also enable self-organized behaviors, such as in the work of Chung et al. (2016), where the MAVs are made to follow the one with the highest altitude within their sub-swarm. In this way, the leader is automatically elected in a self-organized manner by the swarm. For example, should a current leader MAV need to land as a result of a

[4] See `www.st.com/en/imaging-and-photonics-solutions/vl53l0x.html`.

malfunction, a new leader can be automatically re-elected so that the rest of the swarm can keep operating.

2.4.2. ACHIEVING SAFE NAVIGATION

It is important that each MAV remains safe and that it does not collide with its surroundings, or that damages remain limited in case this happens. This safety requirement can be satisfied in two ways. The first, which is more passive and brings us back to MAV design, is to develop MAVs that are mechanically collision resilient. This allows the MAV to hit obstacles without risking significant damage to itself or its environment. With this rationale, Briod et al. (2012), Mulgaonkar et al. (2015, 2018), and Kornatowski et al. (2017) proposed protective cages to be placed around an MAV. However, the additional mass of a cage can negatively impact flight time and the cage can also introduce drag and controllability issues (Floreano et al., 2017). Instead, Mintchev et al. (2017) developed a flexible design for miniature quadrotors in order to be more collision resilient upon impact with walls. The use of airships has also been proposed as a more collision resilient solution (Melhuish and Welsby, 2002; Troub et al., 2017). The limitations of airships, however, are in their lower agility and restricted payload capacity. More recently, Chen et al. (2019) demonstrated insect-scale designs that use soft artificial muscles for flapping flight. The soft actuators, combined with the small scale of the MAV, are such that the MAVs can be physically robust to collisions with obstacles and with each other. Collision resistant designs can even be exploited to improve onboard state estimation, such as in the recent work by Lew et al. (2019), whereby collisions are used as pseudo velocity measurement under the assumption that the velocity perpendicular to an obstacle, at the time of impact, is null. The alternative, or complementary, solution to passive collision resistance is active obstacle sensing and avoidance, whereby an MAV uses its onboard sensors to identify and avoid obstacles in the environment.

Collision-free flight can be achieved via two main navigation philosophies: 1) map-based navigation, and 2) reactive navigation. With the former, a map of the environment can be used to create a collision-free trajectory (Ghadiok et al., 2012; Shen et al., 2011; Weiss et al., 2011). The map can be generated during flight (using SLAM) and/or, for known environments, it can be provided a priori. The advantage of a map-based approach is that obstacle avoidance can be directly integrated with higher-level swarming behaviors (Saska et al., 2016b). Instead, a reactive control strategy uses a different philosophy whereby the MAV only reacts to obstacles in real-time as they are measured, regardless of its absolute position within the environment. In this case, if an MAV detects an obstacle, it reacts with an avoidance maneuver without taking its higher-level goal into account. The trajectories pursued with a reactive controller may be less optimal, but the advantage of a reactive control strategy is that it naturally accounts for dynamic obstacles and it is not limited to a static map. The two can also operate in a hierarchical manner, such that the reactive controller takes over if there is a need to avoid an obstacle, and the MAV is otherwise controlled at a higher level by a path planning behavior. Regardless of the navigation philosophy in use, if the MAV needs to sense and avoid obstacles during flight, it will require sensors that can provide it with the right information in a timely manner.

Of all sensors, vision provides a vast amount of information from which an MAV can

interpret its direct environment. By using a stereo-camera, the disparity between two images gives depth information (Heng et al., 2011; Matthies et al., 2014; McGuire et al., 2017b; Oleynikova et al., 2015). Alternatively, a single camera can also be used. For example, the work of de Croon et al. (2012b) exploited the decrease in the variance of features when approaching obstacles. Ross et al. (2013) used a learning routine to map monocular camera images to a pilot command in order to teach obstacle avoidance by imitating a human pilot. Kong et al. (2014) proposed edge detection to detect the boundary of potential obstacles in an image. Saha et al. (2014) and Aguilar et al. (2017) used feature detection techniques in order to extract potential obstacles from images. Alvarez et al. (2016) used consecutive images to extract a depth map (a technique known as motion parallax), albeit the accuracy of this method is dependent on the ego-motion estimation of the quadrotor. Learning approaches have also been investigated in order to overcome the limitations of monocular vision. By exploiting the collision resistant design of a Parrot AR Drone, Gandhi et al. (2017) collected data from 11,500 crashes and used a self-supervised learning approach to teach the drone how to avoid obstacles from only a monocular camera. Self-supervised learning of distance from monocular images can also be accomplished without the need to crash, but with the aid of an additional sensor. Lamers et al. (2016) did this by exploiting an infrared range sensor, and van Hecke et al. (2018) applied this to see distances with one single camera by learning a behavior that used a stereo-camera. This is useful if the stereo-camera were to malfunction and suddenly become monocular. Alternative camera technologies have also been developed, providing new possibilities. RGB-D sensors are cameras that also provide a per-pixel depth map, a mainstream example of which is the Microsoft Kinect camera (Newcombe et al., 2011). This particular sensor augments one RGB camera with an IR camera and an IR projector, which together are capable of measuring depth (Smisek et al., 2013). RGB-D sensors have been used on MAVs to navigate in an environment and avoid obstacles (Huang et al., 2017; Odelga et al., 2016; Shen et al., 2014; Stegagno et al., 2014). One of the disadvantages of these RGB-D sensors over a stereo-camera setup (whereby depth is inferred from the disparity) is that RGB-D sensors can be more sensitive to natural light, and may thus perform less well in outdoor environments (Stegagno et al., 2014). Finally, in recent years, the introduction of Dynamic Vision Sensor (DVS) cameras has also enabled new possibilities for reactive obstacle sensing. A DVS camera only measures changes in the brightness, and can thus provide a higher data throughput. This enables a robot to quickly react to sudden changes in the environment, such as the appearance of a fast moving obstacle (Falanga et al., 2019a; Mueggler et al., 2015).

The capabilities of a vision algorithm depend on the resolution of the onboard cameras, the number of the onboard cameras, as well as the processing power on board. On very lightweight MAVs, such as flapping wings, even carrying a small stereo-camera can be challenging (Olejnik et al., 2019). A further known disadvantage of vision is the limited Field of View (FOV) of cameras. Omni-directional sensing can only be achieved with multiple sets of cameras (Floreano et al., 2013; Moore et al., 2014) at the cost of additional mass, the impact of which is dependent on the design of the MAV.

Although vision is a rich sensor, in that it can provide different types of information, other sensors also can be used for reactive collision avoidance. LIDAR, for instance, has the advantage that it is less dependent on lighting conditions and can provide more ac-

curate data for localization and navigation (Bachrach et al., 2011; Tripicchio et al., 2018). Alternatively, time-of-flight laser ranging sensors have also been proposed for reactive obstacle avoidance algorithms on small drones (Laković et al., 2019). These unidirectional sensors can sense whether an object appears along their line of sight (typically up to a few meters). Due to their small size and low power requirements, they can be used on tiny MAVs (Bitcraze AB, 2019).[5]

2.5. INTRA-SWARM RELATIVE SENSING AND COLLISION AVOIDANCE

Once we have an MAV design that can perform basic safe flight, we begin to expand its capabilities toward collaboration in a swarm. Two fundamental challenges need to be considered in this domain. The first is relative localization. This is not only required to ensure intra-swarm collision avoidance, which is a basic safety requirement, but also to enable several swarm behaviors (Bouffanais, 2016). The design choice used for intra-swarm relative localization defines and constrains the motion of the MAVs relative to one another, which affects the swarming behavior that can be implemented. The second challenge is intra-swarm communication. Much like knowing the position of neighbors, the exchange of information between MAVs can help the swarm to coordinate (Hamann, 2018; Valentini, 2017). In this section, we explore the state of the art for relative localization (Section 2.5.1), reactive collision avoidance maneuvers (Section 2.5.2), and we discuss intra-swarm communication technologies (Section 2.5.3).

2.5.1. RELATIVE LOCALIZATION

In outdoor environments, relative position can be obtained via a combination of GNSS and intra-swarm communication. Global position information obtained via GNSS is communicated between MAVs and then used to extract relative position information. This has enabled connected swarms that can operate in formations or flocks (Chung et al., 2016; Yuan et al., 2017). An impressive recent display of this in the real world was put into practice by Vásárhelyi et al. (2018), who programmed a swarm of 30 MAVs to flock. The same concept can be applied to indoor environments if pre-fitted with, for example: external markers (Pestana et al., 2014), motion-tracking cameras (Kushleyev et al., 2013), antenna beacons (Guo et al., 2016; Ledergerber et al., 2015), or ultra sound beacons (Vedder et al., 2015). However, this dependency on external infrastructure limits the swarm to being operable only in areas that have been properly fitted to the task. Several tasks, especially the ones that involve exploration, cannot rely on these methods. In order to remove the dependency on external infrastructure, there is a need for technologies that allows the MAVs themselves to obtain a direct MAV-to-MAV relative location estimate. This is still an open challenge, with several technologies and sensors currently being developed.

One of the earlier solutions for direct relative localization on flying robots proposed the use of infrared sensors (Roberts et al., 2012). However, since infrared sensors are unidirectional, this used an array of sensors (both emitting and receiving) placed around the MAV in order to approach omni-directionality, making for a relatively heavy system.

[5] See `www.bitcraze.io/multi-ranger-deck/`.

Alternatively, vision-based algorithms have once again been extensively explored. However, the robust visual detection of neighboring MAVs is not a simple task. The object needs to be recognized at different angles, positions, speeds, and sizes. Moreover, the image can be subject to blur or poor lighting conditions. One way to address this challenge is with the use of visual aids mounted on the MAVs, such as visual markers (Faigl et al., 2013; Krajník et al., 2014; Nägeli et al., 2014), colored balls (Epstein and Feldman, 2018; Roelofsen et al., 2015), or active markers such as infrared markers (Faessler et al., 2014; Teixeira et al., 2018)[6] or Ultra Violet (UV) markers (Walter et al., 2018, 2019). Visual aids simplify the task and improve the detection accuracy and reliability. However, they are not as easily feasible on all designs, such as flapping wing MAVs or smaller quadrotors. Markerless detection of other MAVs is very challenging, since other MAVs have to be detected against cluttered, possibly dynamic backgrounds while the detecting MAV is moving by itself as well. A successful current approach is to rely on stereo vision, where other drones can be detected because they "float" in the air unlike other objects like trees or buildings. Carrio et al. (2018) explored a deep learning algorithm for the detection of other MAVs in stereo-based disparity images. An alternative is to detect other MAVs in monocular still images. Like the detection in stereo disparity images, this removes the difficulty of interpreting complex motion fields between frames, but it introduces the difficulty of detecting other, potentially (seemingly) small MAVs against background clutter. To solve the challenge, Opromolla et al. (2019) used a machine learning framework that exploited the knowledge that the MAVs were supposed to fly in formation. Their scheme used the knowledge of the formation in order to predict the expected position of a neighboring MAV and focus the vision-based detection on the expected region, thus simplifying the task. Employing a more end-to-end learning technique, Schilling et al. (2019) used imitation learning to autonomously learn a flocking behavior from camera images. Following the attribution method by Selvaraju et al. (2017), Schilling et al. studied the influence that each pixel of an input image had on the predicted velocity. It was shown that the parts of the image whereby neighboring MAVs could be seen were more influential, demonstrating that the network had implicitly learned to localize its neighbors. Despite the promising preliminary results, it is yet to be seen how it can handle other MAVs sizes or more cluttered backgrounds. Finally, it is possible to use the optic flow field for detecting other MAVs. This approach could have the benefit of generality, but it would require the calculation and interpretation of a complex, dense optic flow field. To our knowledge, this method has not yet been investigated.

From a swarming perspective, it may also be desirable to know the ID of a neighbor. However, IDs may be difficult to detect using vision without the aid of markers. This issue was explored by Stegagno et al. (2011), Cognetti et al. (2012), and Franchi et al. (2013) with fusion filters that infer IDs over time with the aid of communication. Moreover, cameras have a limited FOV. This limits the behaviors that can be achieved by the swarm. For instance, it may be limiting for surveillance tasks where quadrotors may need to look away from each other, but can't or else they may collide or disperse. It can be addressed by placing several cameras around the MAVs (Schilling et al., 2019), but at the cost of additional mass, size, and power, which in turn creates new repercussions.

The use of vision is not only limited to directly recognizing other drones in the en-

[6]The solution by Teixeira et al. (2018) additionally uses communication between the MAVs.

vironment. With the aid of communication, two or more MAVs can also estimate their relative location indirectly by matching mutually observed features in the environment. The MAVs can compare their respective views and infer their relative location. In the most complete case, each MAV uses a SLAM algorithm to construct a map of its environment, which is then compared in full with the rest of the team (as discussed in Section 2.4, mapping can also be accomplished using other sensors such as LIDAR, so this approach is not only reserved for vision). Although SLAM is a computationally expensive task, more easily handled centrally (Achtelik et al., 2012; Forster et al., 2013), it can also be run in a distributed manner, making for an infrastructure free system (Cieslewski et al., 2018; Cunningham et al., 2013; Lajoie et al., 2019). For a survey of collaborative visual SLAM, we refer the reader to the paper by Zou et al. (2019) and the sources therein. An additional benefit of collective map generation is that the MAVs benefit from the observations of their teammates and can thus achieve a better collective map. However, if the desired objective is only to achieve relative localization, the computations can be simplified. Instead of computing and matching an entire map, the MAVs need only to concern themselves with the comparison of mutually observed features in order to extract their relative geometric pose (Achtelik et al., 2011; Montijano et al., 2016). This requires that the images compared by the MAVs have sufficient overlap and can be uniquely identified.

An alternative stream of research leverages communication between MAVs to achieve relative localization, while also using the antennas as relative range sensors. Here, we will refer to these methods as *communication-based ranging* localization. The advantage of this method is that it offers omni-directional information at a relatively low mass, power, and processing penalty, leveraging a technology that is likely available on even the smallest of MAVs. Szabo (2015) first proposed the use of signal strength to detect the presence of nearby MAVs and engage in avoidance maneuvers. Also for the purposes of collision avoidance, in Coppola et al. (2018) we implemented a beacon-less relative localization approach based on the signal strength between antennas, using the Bluetooth Low Energy connectivity already available on even the smaller drones. Guo et al. (2017) proposed a similar solution using UltraWide Band (UWB) antennas for relative ranging, which offer a higher resolution even at larger distances. However, this work used one of the drones as a reference beacon for the others. One commonality between our solution (Coppola et al., 2018) and the one by Guo et al. (2017) is that the MAVs are required to have a knowledge of north, which enables them to compare each other's velocities along the same global axis. However, in practice this is a significant limitation due to the difficulties of reliably measuring north, especially if indoors, as already discussed in Section 2.4.1. To tackle this, in van der Helm et al. (2020) we showed that, if using a high accuracy ranging antenna such as UWB, then it is not necessary for the MAVs to measure a common north. However, selecting this option creates fundamental constraints on the high-level behaviors of the swarm. This issue is there for the case where north is known and when it is not, albeit the requirement when north is not known are more stringent. If north is known, at least one of the MAVs must be moving relative to the other for the relative localization to remain theoretically observable. If north is not known, *all* MAVs must be moving. The MAVs remain bound to trajectories that excite the filter (van der Helm et al., 2020). For the case where north is known, Nguyen et al. (2019) proposed that

a portion of MAVs in the swarm should act as "observers" and perform trajectories that persistently excite the system.

Another solution is to use sound. Early research in this domain was performed by Tijs et al. (2010), who used a microphone to hear nearby MAVs. This was explored in more depth by Basiri (2015) using full microphone arrays for relative localization. A primary issue encountered was that the sound emitted by the listening quadrotor would mask the sound of the neighboring MAVs, which were also similar. This issue was addressed with the use of a "chirp" sound, which can then be more easily heard by neighbors (Basiri et al., 2014, 2016). In recent work, Cabrera-Ponce et al. (2019) proposed the use of a Convolutional Neural Network to detect the presence of nearby MAVs. This is done using a large scale microphone array (Ruiz-Espitia et al., 2018) featuring eight microphones based on the ManyEars framework (Grondin et al., 2013). Specific to sound sensors, the accuracy of the detection depends on how similar the sounds of other MAVs are. Moreover, the localization accuracy depends on the microphone setup. Most works use a microphone array, where the localization accuracy depends on the length of the baseline between microphones, which is inherently limited on small MAVs.

As it can be seen, several different techniques exist. Minimally, these technologies should enable neighboring MAVs to avoid collisions with one another. However, the particular choice of relative localization technology creates a fundamental constraint on the swarm behavior that can be achieved. For example, communication-based ranging methods have unobservable conditions depending on the MAVs' motion, and sound-based localization with microphone arrays will be less accurate when used on smaller MAVs. Similarly, certain swarm behaviors (e.g., one that requires known IDs, or long range distances) may place certain requirements on which technology is best to be used. In Table 2.1, we outline the major relative localization approaches with their advantages and disadvantages.

2.5.2. INTRA-SWARM COLLISION AVOIDANCE

Collision detection and avoidance of objects in the environment has already been discussed in Section 2.4.2. As MAVs operate in teams, relative intra-swarm collision avoidance becomes a safety-critical behavior that should be implemented. The complexity of this task is that it requires a collaborative maneuver between two or more MAVs.

MAVs operate in 3D space, and thus relative collision avoidance could be tackled by vertical separation. However, particularly in indoor environments where vertical space is limited, vertical avoidance maneuvers may cause undesirable aerodynamic interactions with other MAVs as well as other parts of the environment. For quadrotors, while aerodynamic influence is relatively negligible when flying side-by-side, flying above another will create a disturbance for the lower one (Michael et al., 2010; Powers et al., 2013). Furthermore, emergency vertical maneuvers could also cause a quadrotor to fly too close to the ground, which creates a ground effect and pushes it upward, or, if indoors, to fly too close to the ceiling, which creates a pulling effect toward the ceiling (Powers et al., 2013). Vertical avoidance may also corrupt the sensor readings of the MAV. For instance, height may be compromised if another MAV obstructs a sonar sensors. Overall, horizontal avoidance maneuvers are desired.

Table 2.1 Current technologies in the state of the art for relative localization between MAVs, with their main advantages and disadvantages.

Technology	*Sample references*	*Advantages*	*Disadvantages*
Vision (direct, passive)	Faigl et al. (2013) Krajník et al. (2014) Nägeli et al. (2014) Roelofsen et al. (2015) Carrio et al. (2018)	· Rich information · Passive sensor · Possible to extract ID (if using markers or having otherwise visually distinct MAVs) · Lightweight (depending on model) · Scalable (provided environment is not cluttered)	· Computationally expensive · Limited FOV · Dependent on lighting conditions · Dependent on visual clutter in environment · No IDs (if markerless) · Needs visual line of sight
Observation matching	Achtelik et al. (2011) Montijano et al. (2016)	· No direct line of sight is needed · Thrives in visually cluttered environments · Includes IDs (via communication)	· Active sensor, requires communication · Requires sufficient visual overlap · Computationally expensive · Dependent on lighting conditions
Communication-based ranging	Szabo (2015) Guo et al. (2017) Coppola et al. (2018) van der Helm et al. (2020) Nguyen et al. (2019)	· Low mass · Omni-directional · Possible to extract IDs · Can work in visually cluttered environments · Enables communication of additional data	· Active sensor, requires communication · Needs relative excitation maneuvers · Noisy (depending on sensor) · Difficult to scale due to communication interference
Sound	Basiri et al. (2014, 2016) Cabrera-Ponce et al. (2019)	· Scalable (provided that the sound environment is, or does not become, not cluttered) · Passive sensor · Omni-directional · Can work in visually cluttered environments	· Self-propeller noise · Limited range · Noise pollution (if using "chirps") · No IDs (if chirp-less) · Limited angular accuracy due to limited baseline between microphones in the array
Infrared (sensor array)	Roberts et al. (2012)	· Accurate · Computationally simple · Lower dependence on lighting conditions (Roberts et al. (2012) did not test outdoors) · Can work in visually cluttered environments	· Heavy · Needs visual line of sight · Many active sensors result in high energy expense
Vision (direct, with active markers)	Faessler et al. (2014) Teixeira et al. (2018) Walter et al. (2018, 2019)	· Accurate · Possible to extract IDs · Lower dependence on lighting conditions · Can work in visually cluttered environments	· Needs visual line of sight · Many active sensors result in high energy expense

A popular algorithm for obstacle avoidance, provided that the robots know their relative position and velocity, is the Velocity Obstacle (VO) method (Fiorini and Shiller, 1998). The core idea is for a robot to determine a set of all velocities that will lead to collisions with the obstacle (a collision cone), and then choose a velocity outside of that set, usually the one that requires minimum change from the current velocity. VO has stemmed a number of variants specifically designed to deal with multi-agent avoidance, such as Reciprocal Velocity Obstacle (RVO) (van den Berg et al., 2008, 2011), Hybrid Reciprocal Velocity Obstacle (HRVO) (Snape et al., 2009), and Optimal Reciprocal Collision Avoidance (ORCA) (Snape et al., 2011). These variants alter the set of forbidden velocities in order to address reciprocity, which may otherwise lead to oscillations in the behavior. These methods have been successfully applied on MAVs, both in a decentralized way as well as via centralized replanners. They accounted for uncertainties by artificially increasing the perceived radii of the robots. Alonso-Mora et al. (2015) showed the successful use of RVO on a team of MAVs such that they may adjust their trajectory with respect to a reference. This was done using an external MCS for (relative) positioning. In Coppola et al. (2018), we showed a collision cone scheme with onboard relative localization, introducing a method to adjust the cone angle in order to better account for uncertainties in the relative localization estimates. A disadvantage of VO methods and its derivatives is scalability. If the flying area is limited and the airspace becomes too crowded, then it may become difficult for MAVs to find safe flight directions (Coppola et al., 2018). Another avoidance algorithm, called Human-Like (HL), presents the advantage that the heading selection is decoupled from speed selection (Guzzi et al., 2013a; Guzzi et al., 2014), such that the MAVs only engage in a change in heading. HL has been found to be successful even when operating at relatively lower rates (Guzzi et al., 2013b). Although it has not been tested on MAVs, their tests also demonstrated generally better scalability properties.

Alternatively, attraction and repulsion forces between obstacles are also a valid algorithm for collision avoidance. This is a common technique which has been extensively studied in swarm research (Gazi and Passino, 2002; Gazi and Passino, 2004; Reynolds, 1987). If one wishes for the MAVs to flock, these attraction and repulsion forces can also be directly merged with the swarm controller (Vásárhelyi et al., 2018). One potential shortcoming of this approach is that it can lead to equilibrium states whereby the swarm remains in a fixed final formation, although this can also be seen as a positive property that can be exploited (Gazi and Passino, 2011).

In summary, multiple methods exist for intra-swarm collision avoidance. Given sufficiently accurate relative locations, these methods are very successful. The main challenges here are: 1) how to deal with uncertainties and unobservable conditions deriving from the localization mechanism used by the drones, and 2) how to keep guaranteeing successful collision avoidance when the swarm scales up to very large numbers.

2.5.3. INTRA-SWARM COMMUNICATION

Direct sharing of information between neighboring robots is an enabler for swarm behaviors as well as relative sensing (Hamann, 2018; Pitonakova et al., 2018; Valentini, 2017). To achieve the desired effect, it needs to be implemented with scalability, robustness, and flexibility in mind. Common problems that can otherwise arise are: 1) the

messaging rate between robots is too low (low scalability); 2) high packet loss (low robustness); 3) communication range is too low (low scalability and flexibility); 4) inability to adapt to a switching network topology (low flexibility) (Chamanbaz et al., 2017).

Solutions to the above depend on the application. With respect to hardware, the three main technologies in the state of the art are: Bluetooth, WiFi, and ZigBee (Bensky, 2019). All three operate in the 2.4 GHz band.[7] Bluetooth is energy efficient, but features a low maximum communication distances of ≈10-20 m (indoors, depending on the environment and version). This makes it more important to establish a network that can adapt to a switching topology, as it is very likely to change during operations. The latest version of the Bluetooth standard, Bluetooth 5, features a higher range and a higher data-rate despite keeping a low power consumption. It also has longer advertising messages, such that, without pairing, asynchronous network nodes can exchange messages of 255 bytes instead of 31 (Collotta et al., 2018). Bluetooth antennas were used in the previously discussed work of (Coppola et al., 2018) on a swarm of three MAVs to exchange data indoors and to measure their relative range. In comparison to Bluetooth, WiFi is known to be less energy efficient, but works more reliably at longer ranges and has a higher data throughput. Chung et al. (2016) used WiFi to enable a swarm of 50 MAVs to form an ad hoc network. WiFi was also used by Vásárhelyi et al. (2018) in combination with an XBee module[8] using a proprietary communication protocol. ZigBee's primary benefits are scalability (it can keep up to, theoretically, 64000 nodes) and low power, although it has a low data communication rate (Bensky, 2019).[9] Depending on the application, this may or may not be an issue depending on what the intra-swarm communication requirements are. Allred et al. (2007) used a ZigBee module to enable communication on a flock of fixed wing MAVs due to its combination of light energy consumption and long range (offering *"a range of over 1 mile at 60mW"*). For comparisons of technical details of these technologies we refer the reader to the detailed book by Bensky (2019), the MAV-focused review by Zufferey et al. (2013), as well as the earlier comparisons by Lee et al. (2007).

In addition to the technologies discussed above, there is also the possibility of enabling indirect communication via cellular networks. In the near future, 5G networks are expected to make it possible to have a reliable and high data throughput between several MAVs (Campion et al., 2018). Finally, the use of UWB can also gain more relevance in the future, especially because its additional capability to accurately measure the range between MAVs, as discussed in Section 2.5.1, can be very helpful for swarms. One technological challenge is that communication needs power, and while this may be near-negligible for the bigger MAVs, it is not so for the smaller designs (Petricca et al., 2011). From this perspective, the communication-based relative localization discussed in Section 2.5.1, which can also double as a communication device for MAVs, is an interesting solution if one desires a system that can achieve both goals simultaneously. However, using any relative localization approach that relies on communication means

[7] WiFi also operates at other frequency bands. The 5 GHz band, for instance, is typically known to feature a lower interference (Verma et al., 2013). ZigBee can also operate at the 868 MHz and 915 MHz frequency bands (Collotta et al., 2018).

[8] Not to be confused with ZigBee (Faludi, 2010).

[9] Note that Bluetooth Low Energy, a sub-version of the Bluetooth standard, also requires very little power. Tests by Collotta et al. (2018) return that Bluetooth 4.2 and 5.0 have a lower power consumption than ZigBee.

that having a stable connection among MAVs is an important requirement, and possibly a safety-critical one. Moreover, high messaging rates also become important in order to have a high update rate.

2.6. SWARM-LEVEL CONTROL

We finally arrive at the swarm part of this chapter. Once we have reliable MAVs that can safely fly in an environment, localize one another, and perhaps even communicate, we can begin to exploit them as a swarm. The complexity of this task stems from the fact that, due to the decentralized nature of the swarm, the local actions that a robot takes can have any number of repercussions at the global level. These cannot be known unless the system is fully observed and optimized for, which the individual robot cannot do.

This section discusses possible approaches to design MAV swarm behaviors. Prominent examples of behaviors are: flocking, formation flight, distributed sensing (e.g., mapping/surveillance), and collaborative transport and object manipulation.[10] Of these, formation flight receives significant attention. It can be useful for several applications such as surveillance, mapping, or cinematography, so as to collaboratively observe a scene (Mademlis et al., 2019). Additionally, it can also be used for collaborative transport (de Marina and Smeur, 2019), and it has even been shown that certain formations lead to energy efficient flight for groups (Weimerskirch et al., 2001). Flocking behaviors bear similar properties to formation flight, but with more "fluid" inter-agent behaviors that allow the swarm to reorganize according to their current neighborhood and the environment. Distributed sensing behaviors may require the swarm to travel in a formation or flock, but may also include behaviors in which the swarm distributes over pre-specified areas (Bähnemann et al., 2017) or disperses (McGuire et al., 2019). Collaborative transport and object manipulations take two forms. The first is that of MAVs individually foraging for different objects and bringing them to base (Bähnemann et al., 2017). The second is that of jointly carrying a load that is too heavy for the individual MAV to carry (Tagliabue et al., 2019). In order to achieve the behaviors above, and others, the MAVs can also engage in a number of more general swarm behaviors such as distributed task allocation or collective decision making. For all cases, the challenge is to endow the MAVs with a controller that achieves the desired swarm behavior while also avoiding undesired results (Winfield et al., 2005a, 2006).

Similarly to the review by Brambilla et al. (2013) (which the reader is referred to for a general overview of swarm robotics and engineering), we divide the design methods in two categories. The first, which we call "manual design methods", refers to handcrafted controllers that instigate a particular behavior in the swarm. These are discussed in Section 2.6.1, where we provide an overview of the state of the art for different swarm behaviors. The second, which we refer to as "automatic design methods", uses machine learning techniques in order to design and/or optimize the controller for an arbitrary goal. This is discussed in Section 2.6.2. We discuss the advantages and disadvantages between the two, from the perspective of designing swarms of MAVs, in Section 2.6.3.

[10] Note that this list is not exhaustive. Additionally, we will see that there may also be overlaps between these behaviors. For example, as explored in section Section 2.6.1, flocking behaviors may achieve fixed formations under certain equilibria.

2.6.1. MANUAL DESIGN METHODS

This is the "classical" strategy to control, whereby a swarm designer develops the controllers so as to achieve a desired global behavior. For swarm robotics, we differentiate between two approaches. One approach is to design local behaviors, analyze them, and then manually iterate until the swarm behaves as desired. Another approach is to make mathematical models of the robots and their interactions and then design a suitable controller that comes with a certain proof of convergence. The latter approach has some obvious advantages if one succeeds, but it makes the designer face the full complexity of swarm systems. Hence, such methods typically have limited applicability. For example, in the work of Izzo and Pettazzi (2007), the behavior is limited to only symmetrical formations of limited numbers of agents. The preferred approach is dependent on the swarm behavior that the designer wishes to achieve, under the constraints of the local properties of each MAV.

A large portion of methods focuses on formation control algorithms, whereby the goal is for the MAVs to form and/or keep a tight formation during flight. To hold a formation, the MAVs must hold a relative position or distance between given neighbors, such that they can move as one unit through space. See, for instance, the works of Quintero et al. (2013), Schiano et al. (2016), Yuan et al. (2017), de Marina et al. (2017), and de Marina and Smeur (2019). One advantage of flying in formation for MAV swarms is their predictability during operations. Several methods provide robust controllers with mathematical proofs that the formation can be achieved and maintained during flight. A review dedicated to formation control algorithms for MAVs is provided by Oh et al. (2015). Chung et al. (2018) also discuss different methods.

There are applications for which a rigid formation is sub-optimal, undesired, or unnecessary, and it is better for the MAVs to move through space in a flock. Flocking behaviors were originally synthesized from the motion of animals in nature (Aoki, 1982), and were most famously formalized by Reynolds (1987) with the intent of simulating swarms in computer animations. The behavior is typically characterized by a combination of simple local rules: attraction forces, repulsion forces, heading alignment with neighbors, speed agreement with neighbors. This behavior naturally incorporates collision avoidance via the repulsion rule, and it has also been explored as a means to collectively navigate in an environment with obstacles, whereby the obstacles provide additional repulsion fores (Saska, 2015; Saska et al., 2014). Alternatively, the local rules can also be exploited to achieve formations by making use of equilibrium points between attraction and repulsion forces (Gazi, 2005). Depending on the way in which the rules are used, they can be incorporated into an iterative approach, or they can be made part of a mathematical regime combined with the model of the robot. An early real-world demonstration of distributed flocking was achieved by Hauert et al. (2011) with a swarm of ten fixed wing MAVs. The more recent work by Vásárhelyi et al. (2018) demonstrated outdoor flocking for a swarm of 30 quadrotors.

Concerning behaviors such as distributed sensing, exploration, or mapping, there are several different types of solutions that have been developed specifically for MAVs. Typically, these are found to vary depending on the nature of the task, requiring the designer to make careful choices on the best algorithm to be used. Bähnemann et al. (2017) and Spurný et al. (2019), aided by GNSS for positioning, divided a search area

into multiple regions so that a team of three MAVs could efficiently explore it with a pre-planned trajectory. The recent work of McGuire et al. (2019) demonstrated a swarm of six Crazyflie MAVs performing an autonomous exploration task in an unknown indoor environment. Each MAV acted entirely locally based on a manually designed bug algorithm which enabled exploration as well as homing to a reference beacon.

2.6.2. AUTOMATIC METHODS FOR BEHAVIOR DESIGN AND OPTIMIZATION

In the last few decades, the increasing power of machine learning methods cannot be denied, with multiple examples in robotics, autonomous driving, smart homes, and more. Machine learning techniques are a way to automatically extract the local controller that can fulfill a task, relieving us from the need to design it ourselves. However, the problem shifts to devising algorithms that can efficiently and effectively discover the controllers. In this section, we discuss the possibilities based on two primary machine learning approaches in swarm intelligence research: Evolutionary Robotics (ER) and Reinforcement Learning (RL).

EVOLUTIONARY ROBOTICS

ER uses the concept of *survival of the fittest* in order to efficiently search through the design space for an effective controller (Nolfi, 2002).[11] It has been widely adopted in swarm robotics literature in order to evolve local robot controllers that optimize the performance of the swarm with respect to a global, swarm-level objective (Trianni, 2008). ER bypasses the analysis of the relation between the local controllers and the global behavior of the swarm. Instead, it optimizes the controllers "blindly" by means of several evaluations in an evolutionary process, which most often happens in simulation, but can also be performed in the real world (Eiben, 2014). Evolved solutions often exploit the robots' bodies and environment, including the behaviors of other swarm members. Moreover, thanks to the blind optimization, other factors can also be evolved, such as the communication between robots (Ampatzis et al., 2008). ER offers a generic approach to generate swarm controllers of different types, including, but not limited to: neural networks (Silva et al., 2015; Trianni et al., 2003), grammar rules (Ferrante et al., 2013), behavior trees (Jones et al., 2018, 2019; Scheper et al., 2016), and state machines (Francesca et al., 2014). Although neural network architectures can be very powerful, the advantage of the latter methods is that they can be better understood by a designer, which makes it easier to cross the *reality gap* between simulation and the real world when deploying the controllers on the real robots (Jones et al., 2019). Crossing the reality gap is a major challenge in the field of ER and many different approaches have been investigated, also for neural networks. See Scheper (2019) for a more extensive discussion on these methods.

A major challenge for the effective use of ER, especially for swarm robotics, is the design of the fitness functions to be optimized (Francesca and Birattari, 2016). This is usually left to the designer's ability to explicitly define the key elements that indicate the success of a behavior in a measurable and quantitative manner. It is not uncommon

[11] Looking at the complexity achieved by natural swarming systems, it also seems intuitive that such complexity could be achieved automatically by mimicking an evolutionary process (Bouffanais, 2016). It is not surprising that a closely related discipline to ER is that of Artificial Life (AL), dedicated to artificially representing life-like processes, albeit with generally more open-ended exploratory goals (Bedau, 2003; Trianni, 2014)

to see empirically defined parameters that represent certain desired elements, such as safety in the example of Duarte et al. (2016). As task complexity increases, so does the challenge of designing a fitness function. In the worst case, it may become *uninformative* or even *deceptive*, leading the algorithm to not finding the desired behavior (Silva et al., 2016). Different approaches have been proposed to tackle this issue, such as behavioral decomposition or incremental learning (Nelson et al., 2009). The risk with these strategies, however, is that the designer shapes the learning of the task too much, which may lead to sub-optimal performances. As an alternative strategy for learning complex tasks, Lehman and Stanley (2011) proposed novelty search, whereby the fitness is not defined by how well the task is performed, but by how novel a behavior is. This can lead to finding more unorthodox solutions, also for swarm robotics (Gomes et al., 2013). Potential drawbacks of this approach are that the search becomes less directed, and that the shaping shifts from defining a fitness function to defining what constitutes a novel behavior.

To conclude, the ER approach applied to swarming has the large advantage that it deals with complexity by actually bypassing it. However, this currently comes at the cost of needing many evaluations involving the simulation of not one but multiple robots, which leads to longer lasting evolutions. An additional problem of simulating a specific number of robots to evolve a swarm behavior is that the evolution may overfit the behaviors not only to the (simulation) environment, but also to the exact number of robots that were used during the evolution. A naive solution is to simulate different swarm sizes over the evolution, but this will take even more simulation time and, in any case, the number of robots will be limited, meaning that scalability is not guaranteed. Recent developments in this domain have seen the introduction of size-agnostic techniques (Coppola et al., 2019a). Finally, although there are studies on online evolutionary learning for swarm robotics (Bredeche et al., 2018), online evolutionary strategies have yet to be explored (in practice) for MAVs.

REINFORCEMENT LEARNING

With RL, a robot is made to learn by trial-and-error from interacting with its environment under a certain reward scheme. This approach teaches the robot an optimal mapping between a state and the action that it should take so as to maximize its final reward (Sutton and Barto, 2018). RL has been widely used in robotics, and it has thus also found its way to swarm robotics (Brambilla et al., 2013). The advantage of RL is that the robots can explore the environment and continuously adapt their behavior. Several techniques have been proposed over the years for multi-agent RL (Busoniu et al., 2008). However, within swarm robotics literature, it has generally received less attention than ER (Brambilla et al., 2013). A main difficulty with this approach is that, from the perspective of the individual robot, being in a swarm is a non-Markovian task, and each robot only has a partial observation of the full global state. A potential issue, for instance, is *state aliasing*, which refers to when multiple states appear to be the same from the perspective of the agent, even though they are not (McCallum, 1997). It has been demonstrated that ER can achieve better solutions for non-Markovian tasks (de Croon et al., 2005).

The solution to use RL with non-Markovian task leads to a Partially Observable Markov Decision Problem (POMDP). In this case, a robot keeps a history of its observations and thus extracts the most likely global state from them. RL can be applied to POMDPs (Ishii

et al., 2005), yet features scalability issues (the so called "state explosion"), especially when ported to the swarm domain because the global state of the swarm, which it tries to estimate, can take exponentially many forms (Parsons and Wooldridge, 2002). In recent work, Hüttenrauch et al. (2017) proposed to use mean feature embeddings which encode a mean distribution of the agents. Another known difficulty of RL with respect to ER is the credit assignment problem. This refers to the challenge of decomposing the global rewards into local rewards for each robot, as the individual contribution of a single robot to a global task may not always be clearly determined (Brambilla et al., 2013). The credit assignment problem is also manifested over time, as it is difficult to judge which prior action was most conducive.

In short, until now ER appears to be a more appropriate choice for learning control in swarms, as it allows robots to exploit non-Markovian properties of the problem (e.g., the states and behaviors of other robots). However, because of the reality gap, online learning methods may turn out very useful in the future, including RL methods.

2.6.3. MANUAL VS. AUTOMATIC METHODS

A primary advantage of manual design methods for MAV swarms is that the solutions are generally better understood, given that they have to be designed and programmed manually. The algorithms that are developed can be analyzed, and in certain cases it can even be assessed whether the system will converge to the desired properties and even be resilient to faults (Saldaña et al., 2017; Saulnier et al., 2017). This is a particularly attractive property for MAV applications, where safety and predictability are a primary concern. A second advantage is that they carry a clearer breakdown of the requirements. For these reasons, it is not surprising that, to the best of our knowledge and as confirmed by Chung et al. (2018), most real-world implementations of MAV swarms to date have relied on primarily manually designed swarming algorithms. These advantages have also been acknowledged by the automatic design community, which has brought a general interest in using automatic approach to develop explicit controllers such as state machines (Francesca et al., 2014, 2015) or behavior trees (Jones et al., 2019; Kuckling et al., 2018). In future work, the use of these methods could lead to a compromise between extracting an understandable controller and exploiting the power of automatic methods.

A challenge of designing an algorithm manually is in the need to ensure that it can work within the limitations of the system. For instance, if using a communication-based ranging relative localization system, the relative location estimate is only observable when both MAVs are moving in such a way that the system is excited (Nguyen et al., 2019). Alternatively, cameras can be limited by the FOV and be forced to keep a reference neighbor in the center (Nägeli et al., 2014). This may be undesirable for the final application of the swarm (e.g., surveillance), since the camera is kept pointing to other MAVs as opposed to interesting features in the environment. Examples such as these serve to show how a manually designed algorithm can either fail to regard certain elements, or may not exploit the environment optimally so as to best deal with the limitations. An automatic method, on the other hand, could extract a controller that best deals with the limitations, possibly finding solutions that cannot be easily designed manually. For instance, ER studies show that evolved robot controllers can find behaviors that tightly exploit the sensory and motor capabilities of the given robot (Nolfi, 2002) — this is called

sensory-motor coordination.

Despite their power, the application of automatic design methods to MAV swarms are relatively few. One of the first steps was done by Hauert et al. (2009) for the purposes of developing a flying communication network. In this case, the authors proposed to reverse engineer the behavior of an evolved neural network and subsequently program a similar behavior manually. This approach provided original and "creative" insights that enabled them to design a viable and flexible behavior. In later work, Szabo (2015) applied evolutionary behavior trees to a team of MAVs for the purposes of collision avoidance, exploiting the increased readability of behavior trees. The MAVs only knew each other's relative distance (not position) as measured by noisy Bluetooth signal strength, yet the evolved behavior was capable of reducing the number of collisions in a cluttered space. The automatically evolved behavior tree was not only simpler (fewer nodes/branches), but also performed better when compared to a manually designed one. Scheper and de Croon (2017) trained a neural network to form a triangle with a team of three MAVs, inspired by a similar task by Izzo et al. (2014). Although not aimed at MAVs, Izzo et al. (2014) had previously shown that an automatic method was able to extract a behavior with which homogeneous agents could self-organize into asymmetric patterns, whereas the previously developed manual approaches for the same system were limited to symmetric patterns (Izzo and Pettazzi, 2007). Scheper and de Croon (2017) additionally showed that evolving a controller at a higher level of abstraction does not necessarily compromise the ability of automatic methods to exploit an environment and sensory-motor relationships, yet helps to reduce the reality gap. The more recent work of Schilling et al. (2019) showed that it's possible to learn a flocking behavior directly from camera images using imitation learning. This was demonstrated in a real-world environment with two MAVs. This automatic approach was able to find a viable, collision-free behavior that could also localize neighbors.

The limited amount of works show that this field is still young. The extra challenge comes from the several constraints that flow from the lower levels as well as the additional cost and difficulty of real-world experimentation. Nevertheless, there are arguments to show that automatic methods may eventually provide a way to make the most out of the swarms (Francesca and Birattari, 2016). We expect that in the future, once both MAVs as well as automatic swarming design technologies become more mature, we will begin to see an increase of (experimental) works in this domain.

2.7. FURTHER CHALLENGES AND FUTURE DEVELOPMENTS

2.7.1. BATTERY RECHARGING AND SCHEDULING

As already discussed, flight time is a fundamental constraint for MAVs. Swarming can help to increase the flight time of the whole system, as a portion of MAVs can recharge while others are still in operation. This is subject to two main challenges. The first is the design of the combined MAV + recharging ecosystem, and the second is the distributed scheduling between drones. Research has already begun on this front, albeit to the best of our knowledge an automated and distributed recharging method for a swarm of MAVs has yet to be demonstrated outside of a controlled environment. Toksoz et al. (2011) and Lee et al. (2015) designed a battery swapping station to quickly exchange batteries on a

quadrotor. The advantage of such a system is that the battery can be changed quickly. However, it also requires an intricate design as well as highly accurate landing to ensure that the battery is properly replaced. Instead, a contact-based recharging station such as the one proposed by Leonard et al. (2014) offers a simpler system, albeit at the cost of a slower turnover. The authors investigated its use for a multi-robot system, whereby the MAVs queued their use of the charging stations via a prioritization function. Using a similar charging system, Mulgaonkar and Kumar (2014) demonstrated a system where three quadrotors take turns to surveil a target region, such that one operates while the other two recharge. Vasile and Belta (2014) and Leahy et al. (2016) proposed formal strategies based on temporal logic constraints to ensure that the MAVs would correctly queue for recharging. However, the experimental efforts focused on the case where only one MAV operates at a given time. Nowadays, commercial charging station are also available (Brommer et al., 2018). This will likely accelerate the research progress. Wireless charging, albeit slower, is also an attractive choice as it softens the requirement on precision landing (Choi et al., 2016; Junaid et al., 2017).

Flight time can also be increased at the MAV design level by designing MAVs with onboard recharging or longer endurance. The capability for long endurance would allow the swarm to be more flexible and take on a more diverse set of missions. One possible method to increase the flight time is to use solar cells. These have mostly been applied to fixed wing designs such as the Skysailor MAV (Noth and Siegwart, 2010), benefiting from efficient flight conditions and large wing areas. It can in fact be shown that the benefit of solar cells begins to have little effect on smaller platforms, due to the reduced surface area available (Bronz et al., 2009). This trend is even more prominent on quadrotors, which have higher energy requirements. As a solution, D'Sa et al. (2016) proposed an MAV design that can alternate between fixed wing and quadrotor mode, such that *"surplus energy collected and stored while in a fixed wing configuration is utilized while in a quadrotor configuration"*. Recently, Goh et al. (2019) demonstrated a fully solar-powered quadrotor. To meet the energy requirements, an area of 4 m^2 was required. A different solution is to use combustion engines (Nex and Remondino, 2014; Ross, 2014; Zufferey et al., 2013). They benefit from the high-energy density of fuel and can help to provide long endurance flight, although they are typically applied to larger drones in outdoor environments Alternatively, fuel cells have also been explored as a power source for long endurance flight, with increasingly promising results in the recent years (De Wagter et al., 2019; Gong and Verstraete, 2017; Pan et al., 2019).

2.7.2. SWARM-LEVEL ACTIVE FAULT DETECTION

Active and decentralized fault detection should also play a fundamental role for the realization of MAV swarms.[12] If not catered to, then there is a risk that the erroneous actions of one MAV hinder the entire swarm (Bjerknes and Winfield, 2013). Winfield and Nembrini (2006) applied the Failure Mode and Effect Analysis (FMEA) methodology to evaluate the reliability of an entire swarm based on its possible failure points. From such studies it can be evaluated whether, and to what extent, local failures can incapacitate

[12] We differentiate between fault *detection* and fault *tolerance*. Fault detection refers to the ability of the robots in the swarm to detect issues, and thus possibly also cope with them. Fault tolerance refers to the ability of the system to be robust to faults.

the swarm. The question is how such faults can be detected and dealt with during operations. Doing so would create a system that is more robust to failures.

Li and Parker (2007) developed the Sensor Analysis based Fault Detection (SAFDetection). In this approach, a clustering algorithm is used to learn a model of the robots' expected behavior. This model is then used to determine whether the behavior of a robot in the swarm can be considered "normal" (i.e., falls within the learned model), or "abnormal", in which case a likely fault has been detected. A distributed version of the algorithm has also been developed (Li and Parker, 2009), in which case each robot learns its own behavior model locally and then shares it. This strategy scales better with the size of the swarm, as it parallelizes the clustering computations. The works by Tarapore et al. (2013, 2015a,b) also propose a strategy for normal/abnormal behavior classification by synthesizing the behavior of neighbors within a binary feature vector. In more recent work, Tarapore et al. (2017) proposed the use of a consensus algorithm so that the robots can collectively reach a decision on whether the behavior of a team-member can be considered normal or abnormal. This was also tested on a real robotic system (Tarapore et al., 2019). Strobel et al. (2018) explored the use of Blockchain technology within robot swarms in order to identify and exclude faulty/malicious members, further implemented on real robots in Pacheco et al. (2020). Qin et al. (2014) wrote a review on this active area of research. Bringing these solutions to MAV swarms can largely improve the operational safety of the full system, which is paramount for deployment in the real world.

2.7.3. CONTROLLING AND SUPERVISING SWARMS

A control interface should enable an operator to provide commands to the swarm, such as takeoff and landing, the commencement of mission objectives, or the engagement of swarm-wide emergency procedures. All should be done in a direct and intuitive way to minimize the effort by the operator (Dousse et al., 2016; Fuchs et al., 2014). To this end, Nagi et al. (2014) explored the use of a gesture vocabulary which allows a human operator to instruct a team of MAVs. The human operator and the gestures are detected directly by the MAVs using their onboard camera. Thanks to their multiple viewpoints, they are able to discern the operator's commands in a distributed fashion. Tsykunov et al. (2018) explored how to use a haptic glove to control a team of drones as if they were all connected via a spring-damper system. Research has also focused on the development of gesture languages, as in the works of Soto-Gerrero and Ramrez-Torres (2016) and Couture et al. (2018). Virtual reality is also becoming an increasingly popular technology, and is beginning to be applied to the control of MAVs (Tsykunov and Tsetserukou, 2019; Vempati et al., 2019). Besides the above, a less technical, yet highly significant, challenge to overcome on this front is the (understandably) stringent legislation surrounding MAV flight, particularly in outdoor scenarios, often requiring at least one pilot per drone (the specifics vary based on the location) (Vincenzi et al., 2015). We refer the interested reader to Hocraffer and Nam (2017) and the sources therein for a more thorough overview of the challenges and the current technologies for human control of aerial swarms.

2.8. DISCUSSION: HOW FAR ARE WE?

Following the many topics discussed in this chapter comes the inevitable question: how far are we from large scale aerial swarms that can cooperatively explore areas, carry heavier objects, and autonomously complete complex tasks without low level human-in-the-loop control? Despite the large amount of research and development that has been done to tackle the topics within this grander scheme, the field of robotics and the field of swarm intelligence are both still relatively young, and there remain advances to be made. In this chapter, we discussed how the swarm behavior depends on the constraints set by lower-level properties, and vice versa. This interdependency and iterative nature of design means that, if we wish to bring full-fledged MAV swarms to the real world, there must be a mutual understanding between the design levels as to what is required and what can be achieved in reality.

One of the main technologies required to make the leap from flying a single MAV to flying a decentralized swarm is an accurate and reliable intra-swarm relative localization technology. Even for those applications where cooperation is limited and each member in the swarm acts mostly independently, relative localization is still needed to ensure relative collision avoidance, which is a safety-critical requirement. As we have shown throughout this chapter, several technologies are currently under exploration and it is still unclear which will prove most reliable and advantageous in the long run. As the choice of these systems very directly shapes the behavior of the swarm, the challenge of designing the swarm behavior needs to be tightly coupled to it, additionally to the way it is coupled to the design of the individual robots. As such, automatic design algorithms of swarm behaviors can provide a way to make the most out of the individual MAVs and their limitations, albeit at the potential cost of relying on less well understood controllers.

Additionally, the on-going standardization of tools is expected to help the field to reach a new level of maturity (Nedjah and Junior, 2019). Systems such as ROS (Quigley et al., 2009), Paparazzi (Brisset and Hattenberger, 2008; Mueller and Drouin, 2007), or PX4 (Meier et al., 2015) have now accelerated the process of prototyping and testing on real-world MAVs, and have also made it easier to share hardware/software advancements. Low cost programmable MAVs such as the Crazyflie are also available, making it more feasible to experiment with large numbers of MAVs. Additionally, dedicated standards such as MAVLink, which provides communication between software modules, are becoming increasingly popular (Dietrich et al., 2016), and full-stack frameworks have been developed to handle the entire pipeline (Millan-Romera et al., 2019; Sanchez-Lopez et al., 2016). The combination of these systems together with simulators, such as the well known Gazebo (Koenig and Howard, 2004), ARGoS (Pinciroli et al., 2012), or AirSim (Shah et al., 2018), further help to quickly prototype software in a realistic simulation environment. Combined with models and frameworks such as hector-quadrotor (Meyer et al., 2012) or RotorS (Furrer et al., 2016), simulation environments can significantly accelerate the development time (Johnson and Mishra, 2002). Mairaj et al. (2019) provides an extensive review of several simulators for this purpose. Dedicated swarm languages such as Buzz (Pinciroli and Beltrame, 2016) also provide a simpler prototyping framework dedicated to swarm robotics, which can also be applied to MAVs.

Finally, the prominent rise in popularity of MAVs in the last decade has brought about

several technology accelerators. MAV focused robotics competitions such as the Mohamed Bin Žayed International Robotics Challenge (MBZIRC) or the International Micro Air Vehicle (IMAV) competition have now also begun to integrate swarming or multi-robot elements (Bähnemann et al., 2017; Nieuwenhuisen et al., 2017; Pestana et al., 2014; Saska et al., 2016a; Spurný et al., 2019). This pushes researchers to take a technology out of the lab and into unknown environments, thereby increasing their robustness.

2.9. CHAPTER CONCLUSIONS

The challenges to solve before we can expect to see swarms of autonomous MAVs are many. They begin at the lowest level, forcing us to think of how the MAV design will impact the swarm behavior, and they end at the highest level, where we must design collective behaviors that best exploit our lower-level designs, controllers, and sensors. In the last decade, the field of swarm robotics and MAV design have started to merge more and more, leading to increasingly impressive achievements. To go further, the tight and complex relationship between the low level and the high level needs to be appreciated in order to break into a new era of truly autonomous and distributed swarms of MAVs.

Several of the principles that have been explored in this chapter also apply to other robotic systems. Overall, in order to achieve a reliable swarm of robots, we need to have a method that transparently relates the global goal to the local sensory data. Additionally, for the sytem to be safe to use, each robot should be capable of prioritizing long-term safety over achieving the goal.

3

PROVABLE SELF-ORGANIZING PATTERN FORMATION WITH LIMITED KNOWLEDGE

In the previous chapter, we studied the relationship between sensory observations and swarm design, and emphasized the need to establish transparent procedures in order to actualize real world swarms. In this chapter, we thus explore how to automatically design and verify the local behavior of robots with highly limited cognition, while ensuring safety. We will focus on the task of pattern formation as a case study (the method will be generalized in later chapters). All robots in this particular pattern formation task are: anonymous, homogeneous, non-communicating, memoryless, reactive, do not know their global position, do not have global state information, and operate by a local clock. They only know: 1) the relative location of their neighbors within a short range and 2) a common direction (north). We developed a procedure to generate a local behavior that allows the robots to self-organize into a desired global pattern despite their individual limitations. This is done while also avoiding collisions and keeping the coherence of the swarm at all times. The generated local behavior is a probabilistic local state-action map. The robots follow this stochastic policy to select an action based on their current observation, referred to in this work as their local state. It is this stochasticity, in fact, that allows the global pattern to eventually emerge. For a generated local behavior, we then present a set of conditions to verify whether the desired pattern will always eventually emerge from the local actions of the agents. The novelty of the verification procedure is that it is primarily local in nature and focuses on the local states of the robots and the global implications of their local actions. A local approach is of interest to reduce the computational effort as much as possible when verifying the emergence of larger patterns. Finally, we explore how the behavior could be implemented on real robots and investigate this with extensive simulations on a realistic robot (drone) model using ROS and Gazebo.

The contents of this chapter have been published in Coppola et al. (2019b).

3.1. BACKGROUND

The objective of swarm robotics is to enable several robots to collaborate toward a common goal. The goal of pattern formation, which is when the swarm must form a desired spatial configuration, has been a topic of significant attention with many applications for aerial robots (Achtelik et al., 2012; Saska et al., 2016b), underwater robots (Joordens and Jamshidi, 2010), satellites (Engelen et al., 2011; Verhoeven et al., 2011), and more. For safety reasons, the behavior should also ensure that collision paths are avoided and that the swarm remains coherent (i.e., the swarm does not break apart into multiple groups). Our principal interest in this chapter lies in developing a simple behavior to achieve pattern formation with a swarm of robots with extremely low levels of cognition.

One relevant example of an extremely limited robot is miniature quadrotors, henceforth referred to as Micro Air Vehicles (MAVs). They are characterized by low memory and processing capabilities due to their increasingly small size and mass (McGuire et al., 2016). When operating in closed environments, where Global Navigation Satellite Systems (GNSSs) may be unavailable, they should coordinate only using the relative position of their neighbors, of which they may also be unable to discern the identity, as for instance in the system studied by Faigl et al. (2013) or by Stegagno et al. (2016). Furthermore, intra-swarm communication may prove itself challenging to achieve in practice and is best kept at a minimum (Hamann, 2018). For example, our recent experiments showed how a small group of three MAVs can already begin to suffer from relatively limited rate of communication and growing interference (Coppola et al., 2018; van der Helm et al., 2020). Finally, in our pursuit of a minimalist swarm, we also expect all MAVs to be functionally homogeneous without individually pre-allocated tasks. Mesbahi and Egerstedt (2010) refer to this as *assignment free*. Accepting all these limitations leads us to robots that have no knowledge of their surroundings except (in what we assume to be a minimal requirement for collaboration) the current relative location of their closest neighbors. The motivation behind this work was thus to determine a local behavior with which a swarm of robots with such minimal knowledge could nevertheless be able to both handle safety critical goals (i.e., collision avoidance and swarm coherence) as well as systematically self-organize into a pattern. Moreover, we aimed for a simple reactive behavior that could be concisely stored and processed even by the least capable of robots.

As discussed in the introduction, there are two fundamental challenges in the development of swarm behavior for such limited robots:

1) The top-down automatic development of local rules from a global goal.
2) The bottom-up verification of whether the local rules will lead to the desired global goal.

The two main contributions in this chapter directly address these two challenges for the domain at hand. For our very limited robots, we automatically define the local rules that they must follow in order to form a pattern. As it will be seen, these rules are presented as a probabilistic state-action map that can be automatically generated with a few steps. This is the first main contribution. We then provide a method to automatically verify whether the swarm will always eventually form the pattern, or whether certain other spurious results may occur. The proof procedure has the novel aspect that it

largely focuses on the analysis of local states of the agents, rather than all global states of the swarm, in order to determine the successful formation of the global desired pattern from any other initial pattern. This allows for computation tractability and constitutes the second main contribution.

The generated local behavior of the robots is defined by a probabilistic local state-action map. The local state of a robot is simply a discretized view of its current neighborhood, and the actions are directions that it can move toward. This local state-action map can easily be developed to simultaneously handle collision avoidance, avoidance of swarm separation, and formation of a desired pattern. The swarm acts entirely stochastically only based on this. All robots have the same state-action map. As the robots operate using local clocks, any robot can move at any time. When it does, it uses the probabilistic state-action map to stochastically select its next action out of the available options (with equal probability, optimizing the probabilities will be addressed in Chapter 4). The global pattern emerges from this stochastic process once all robots find themselves in local states in which they cannot select any action to move anymore. This stochastic behavior means that the same pattern will be formed in several different ways even when starting from the same initial conditions, and how the pattern is formed is left to the robots. However, although it may not necessarily be important *how* the goal is reached, it is important that *it is* reached. This is the reason that we present an automatic verification procedure to verify whether the local behaviors will always eventually lead to the intended higher-level behavior.

This chapter is organized as follows. We define the problem in Section 3.2. In Section 3.3, we review other solutions to pattern formation and we explain the context and novelty of our contributions. The methodology is then detailed in Section 3.4. Here, we explain how to generate the probabilistic state-action map and we present the proof procedure to check whether the desired pattern will always eventually emerge. We then perform extensive simulations of an increasing level of fidelity. In this way, we explore different aspects of the behavior, from the more fundamental to the more practical. Specifically, we start with an idealized system operating on a discrete grid in discrete time steps in Section 3.5. We then move on to accelerated particles in continuous space and to simulated MAVs with a realistic quadrotor model and sensor noise in Section 3.6. The insights gathered are further discussed in Section 3.7. Finally, Section 3.8 provides concluding remarks and summarizes future research directions.

3.2. PROBLEM DEFINITION, CONSTRAINTS, AND ASSUMPTIONS

The problem tackled in this chapter is for a swarm of robots to reshuffle into a pattern while avoiding collisions and group separation. In this work, a pattern P is an anonymous spatial configuration of robots on a 2D plane with specific relative positions to one another.[1] Let P_{des} be the desired final pattern that the swarm settles in. Considering our interest in robotics, P_{des} must be achieved while also avoiding collision paths and swarm separation. More formally, we are interested in achieving a behavior that can ensure that the swarm is **safe** (Definition 3.2.1) and **live** (Definition 3.2.2).

[1] This definition of pattern is adapted from the definition used in the context of cellular automata by Sapin (2010).

Definition 3.2.1. The swarm is **safe** if neither of the following events occurs: 1) a collision between two or more robots, 2) the swarm disconnects into two or more groups.

Definition 3.2.2. The swarm is **live** if, starting from any initial pattern $P_0 \neq P_{des}$, it will always eventually form the desired pattern P_{des}, where the only restriction on P_0 and P_{des} is that the swarm has a connected sensing topology.[2]

The robots have the following constraints:

C1: The robots are homogeneous (all entirely identical).
C2: The robots are anonymous (they cannot sense each other's identity).
C3: The robots are reactive (they only select an action based on their current state).
C4: The robots are memoryless (they do not remember past states).
C5: No robot can be a leader or seed.
C6: The robots cannot communicate with each other.
C7: The robots only have access to their local state.
C8: The robots do not know their global position.
C9: The robots exist in an unbounded space.[3]
C10: Each robot can only sense the relative location of its neighbors up to a short range.

The following assumptions are made:

A1: The robots all have knowledge of a common direction (i.e., north).
A2: The robots operate on a 2D plane.
A3: When a robot senses the relative location of a neighbor, it can sense it with enough accuracy and update frequency to establish if a neighbor is moving or standing still (e.g., hovering).
A4: P_0, the initial pattern formed by the robots, has a connected sensing topology.

The rationale behind each assumption is:

- Assumption A1 is a typical assumption in several swarm designs (Ji and Egerstedt, 2007; Shiell and Vardy, 2016). On real robots, a common direction can be known using onboard sensors such as, but not limited to, a magnetic sensor and/or a gyroscope (Conroy et al., 2005; Oh et al., 2015).
- Assumption A2 is representative of ground robots, or MAVs flying at approximately the same height.
- Assumption A3 deserves a more in-depth analysis. For general robotic platforms, relative localization is deemed a fundamental tool for collision avoidance and coordination. Concerning MAVs, for instance, a sufficiently accurate relative localization technology is required if collision avoidance (a basic behavior needed for them to swarm safely) is required. As was discussed in depth in Section 2.5.1, there exist several technologies to achieve relative localization. Pugh et al. (2009) and Roberts et al. (2012) used technology based on infrared (IR) signals. Basiri et al.

[2] A connected graph is one that features a path from any node to any other node. In this case, the robots in the swarm are the nodes, and the edges of the graph represent how the robots sense one another.
[3] This is listed as a constraint because it means that the robots cannot exploit the environment, such as the walls of an arena, to complete their task.

(2014) introduced an audio-based solution with a microphone array. Faigl et al. (2013) and Roelofsen et al. (2015) proposed vision based methods relying solely on (one or more) onboard cameras. Coppola et al. (2018) and Guo et al. (2017) explored relative localization sensors based on signal ranging. In this chapter, we will show that fulfilling Assumption A3 up to a certain extent is paramount to provide safe behavior in spite of all other constraints. In our final simulations, to be found in Section 3.6.3, we will show that in practice the swarm can also function even when the robots are only able to detect movements beyond a certain threshold velocity, rather than if adhering perfectly to the assumption.

- Assumption A4 is needed for the entire swarm to begin acting as a collective. If Assumption A4 were violated (and, for instance, the swarm was to begin while separated into two groups that cannot sense each other), then it could not ever be expected for the separate groups to find each other in an unbounded space.

3.3. RELATED WORKS AND RESEARCH CONTEXT

Pattern formation is a well studied problem in robotics. A review of existing solutions is presented in Section 3.3.1. The swarm treated in this work sets itself apart by its minimalist nature, constraining the knowledge of the robots to only the relative location of near-by neighbors and a north direction. We discuss our contributions and their context in Section 3.3.2.

3.3.1. REVIEW OF APPROACHES TO PATTERN FORMATION BY A SWARM OF ROBOTS

The solutions to pattern formation found in the literature rightfully vary depending on the sensing capabilities of the robots. In this section, we review solutions present in the literature, starting from cases where the robots are more knowledgeable of their surroundings to increasingly more minimalist cases more similar to our own (as introduced in Section 3.2).

Several solutions are based on the assumption that each robot in the swarm can directly sense every other robot. In this case, the topology of the swarm is said to be *fully connected* or *complete*. This endows each robot with a global view of the swarm. This type of swarm is found to self-stabilize to an equilibrium only by means of attraction and repulsion forces (Gazi and Passino, 2004). Izzo and Pettazzi (2005, 2007) showed how the attraction and repulsion forces alone could be tuned such that the swarm stabilizes into a desired pattern. However, the results had two limitations: 1) the swarm can unpredictably form spurious patterns depending on the initial conditions due to the presence of spurious equilibria, and 2) they were limited to symmetric patterns. Asymmetry is difficult for a homogeneous non-communicating swarm to resolve, and it was tackled with the use of neural networks in later work (Izzo et al., 2014; Scheper and de Croon, 2016). Formation control algorithms have also been proposed, whereby the robots are allocated positions/distances to achieve and maintain with the other robots (de Marina, 2016; Pereira and Hsu, 2008). With this strategy, the swarm will quickly form the desired pattern. However, it is required for one to specify the necessary inter-robot distances/locations without anonymity.

To address that the swarm may not always begin in a fully connected topology, Ji and Egerstedt (2007) and Mesbahi and Egerstedt (2010) proposed the use of a gathering algorithm so that all robots come together prior to initiating the pattern formation task. In several scenarios, however, being in a *fully* connected topology is simply not viable, and we must accept that the topology of the system is just connected, and not fully connected. For instance, if robots sense each other using onboard cameras or IR sensors, as could likely be the case for either MAVs or ground robots, they will be unable to see behind other robots or beyond a certain distance.[4] Tanner (2004) and Rahmani et al. (2009) showed how to control swarms with a static connected topology, yet when the robots can only sense their closest neighbors, the topology of the swarm will not be static but it will change depending on the current relative positions. Falconi et al. (2010) showed how to combine local positioning information together with a communication protocol in a consensus algorithm. Similarly to formation control, however, this algorithm requires specifying the formation parameters without anonymity. Another popular solution found in the literature is to use seed robots. These are robots in the swarm that do not move and act as a reference to the other robots. Rubenstein et al. (2014) used this to enable an impressively large swarm of simple robots (up to 1,024) to form shapes. Four seed robots were manually placed in a cross formation, and the other robots then circled around them and "filled up" the shape. Wessnitzer et al. (2001) used seed robots to build up patterns in a chain-like fashion, starting from a seed robot that recruits other robots. A seed was also used for a system of self-arranging blocks by Grushin and Reggia (2008, 2010). Here, a static seed block acted as a reference for others to determine their correct relative position (through communication with neighbors), virtually providing them with a global reference albeit while still only making use of local communication. Bonabeau et al. (2000) also studied the rules for the construction of a structure by robots. The robots would begin by placing blocks next to a seed block according to specific rule sets, whereby the blocks could no longer be moved once a robot had placed them. This created a slowly evolving construction. More recently, Werfel and Nagpal (2008) and Werfel et al. (2014) developed and implemented an algorithm in order to coordinate the construction task for a team of robots. This algorithm also relied on the use of a seed block, which the robots could use as a unique shared reference to determine where to place the other blocks. However, in general, the use of a reference (which for pattern formation would be a seed robot) requires that other robots can identify it, which is not the case here given that the robots are all anonymous. Moreover, when they are all functionally homogeneous, no robot can be assigned as the seed. Without communication, then they cannot elect one themselves either, as otherwise explored by Yamauchi and Yamashita (2014), Derakhshandeh et al. (2016), and Di Luna et al. (2017), where a swarm could self-elect a leader/seed robot.

We now move to even simpler systems. For homogeneous and anonymous robots with no seeds, Klavins (2002) proposed to encode a pattern as a graph and a collection of its sub-graphs. This technique set the way for the use of graph grammars, later devel-

[4] As also discussed in Section 2.5.1, there is a vast amount of solutions for relative localization in swarm robotics, and it is also a separate topic of exploration in our own current research (Coppola et al., 2018; Li et al., 2020; van der Helm et al., 2020). In this chapter, however, we declare the challenge outside of the scope of this work and we deem it sufficient to assume that the robots are endowed with the necessary sensors to sense neighboring robots within a short omni-directional range.

oped in Klavins (2007) for self-assembly by a team of robots. The robots randomly drifted in a confined environment and could latch together upon encounter. Once latched, they could communicate their state and determine whether the connection formed a part of the total graph, in which case they would remain attached. Otherwise, they would detach and continue drifting. Using this approach, the pattern would slowly assemble. Similar strategies were studied by Smith et al. (2009), Arbuckle and Requicha (2010), Arbuckle and Requicha (2012), Fox and Shamma (2015). In more recent work, Haghighat and Martinoli (2017) proposed an algorithm for the automatic encoding of such rules for rotationally symmetric modules. However, the local rules used in these studies do not incorporate the additional fundamental constraints of the robots that are studied in this work, namely that the robots cannot: collide, latch together, randomly drift apart, or (most importantly for these algorithms to work) communicate. Without communication it is not possible for the assembly to grow, because the robots are not capable of knowing more than their local state at any point and thus require a different decision making process on the level of the individual agent.

Intra-swarm communication is a very powerful tool. It allows robots to share their intentions and their perspectives. It was used in several works that we already discussed and more, including consensus algorithms (Falconi et al., 2010, 2011, 2015), leader election algorithms (Di Luna et al., 2017), or bidding algorithms for task allocation (Gerkey and Matarić, 2004). More recently, Slavkov et al. (2018) studied how to use a communication architecture to diffuse activation values across the swarm. The swarm could then rearrange itself so as to protrude in regions of high activation values, creating emergent morphologies. Communication can also double as a sensor. Nembrini et al. (2002) and Winfield et al. (2008) used communication to enable a swarm to remain connected even in the presence of obstacles by repeatedly checking for connectivity with the neighbors through a broadcast and listening protocol. In Winfield and Nembrini (2012), the robots communicate their adjacency matrix to one another in order to extend their knowledge beyond what their sensors allow, which is found to increase the coherence performance.

Despite its advantages, considering the difficulties in ensuring a high throughput and reliable intra-swarm wireless communication (Coppola et al., 2018; Hamann, 2018), we have taken an interest in establishing a behavior that also does not natively require communication, such that it can work even when such hardware is not available. Once even communication is removed, few works, to the best of our knowledge, explore the coordination of a swarm of robots that is as limited as the one presented in this work. Krishnanand and Ghose (2005) developed alignment behaviors by which they could form non-finite grids and lines. Flocchini et al. (2005) explored the gathering problem, whereby all robots must aggregate together as much as possible. Yamauchi and Yamashita (2013) examined the formation power of very limited agents, but a behavior to achieve the patterns was not developed. This leaves a knowledge gap in the field of minimalist swarming.

3.3.2. CONTRIBUTIONS AND RESEARCH CONTEXT

There are two principal scientific contributions in this chapter:

1. An automatic procedure to extract the local behavior so that a swarm of robots with extremely limited cognition and no communication can form a desired pat-

tern, while also avoiding collisions and keeping the swarm in a connected sensing topology.

2. An automatic proof procedure to verify whether the set of local rules will always eventually cause the swarm to generate the pattern. We present a primarily *local* analysis of the behavior which allows to verify that the *global* pattern can be achieved from any initial pattern P_0. The large advantage of such a local analysis is that it limits the computational explosion of global proof methods.

Automatic procedures to generate local rules that create high-level functions are already present in the literature. Two notable recent works in this domain are from Rubenstein et al. (2014) and Werfel et al. (2014). Both systems demonstrate an efficient distributed behavior. Looked at from above, we see that the global goal is slowly reached by the robots. The difference with our work stems from the limitations of our robots, which do not (and cannot, in light of their limited cognition) rely on a reference. As a result, their behavior is fully dictated by their local environment without any global context. Furthermore, unlike the system tackled by Grushin and Reggia (2008, 2010), our robots also cannot see far, meaning that they do not know what they will find when they move. Therefore, they cannot knowingly move toward local target locations. This is why they must rely on a probabilistic scheme.

The final pattern is automatically encoded from the larger pattern within the state-action map under this rule: if a robot finds itself in a local state that *may* constitute the global desired pattern, it will stay still. This eventually gives rise to the pattern once all robots end up in such states. Conceptually, the breakdown of a large pattern into smaller parts resembles graph grammar approaches, as for instance used by Klavins (2007) or Haghighat and Martinoli (2017). In our case, however, the robots cannot communicate and must only use the knowledge that a neighbor is (or is not) there in order to decide their next action. Furthermore, the robots cannot detach and drift freely, which restricts how the swarm can evolve. Overall, this means that the pattern does not slowly assemble, but rather forms by the stochastic (inter-)actions of the robots. The phenomenon can only be detected at the macroscopic scale and not by the robots themselves. This behavior is characteristic to emergent processes (Bonabeau and Dessalles, 1997), and its complexity is the reason that we also need to verify that our desired pattern is the sole emergent result.

Our verification of the emergent property (i.e., the final pattern) is based on a formal analysis of the swarm, inspired by Winfield et al. (2005b). Dixon et al. (2012) and Gjondrekaj et al. (2012) applied this with the use of model checking and demonstrated its potential. However, an issue with model checking is that it performs an exhaustive search of all global states (Clarke, Jr. et al., 1999) and it is subject to a computational explosion as the size of the swarm grows. Konur et al. (2012) tackled this using macroscopic swarm models. These models efficiently describe the evolution of the swarm by means of one finite state machine (Winfield et al., 2008). However, macroscopic models typically assume that robots are uniformly distributed, or, in general, make probabilistic assumptions about the presence of robots in a given area (Lerman et al., 2001; Prorok et al., 2011). These assumptions may be suitable for more abstract spatial goals, such as aggregation, exploration, or coherence, but they do not apply to pattern formation, which by definition has a strict requirement on the spatial arrangement. To be able to

verify the emergent property yet keep the computations low, we focus on a local analysis of the behavior. With this novel analysis, we provide a set of local conditions that, if met, guarantee that the swarm will always eventually self-organize into the desired pattern. Unlike the macroscopic models discussed above, this analysis means that we do not merely *assume* that there is enough free movement/motion in the swarm, but use the conditions to check that this is in fact the case. With this, we limit the global analysis only to the discovery of spurious patterns. However, this search only needs to be executed on a very restricted subspace, for which we provide a methodology to identify the candidates.

3.4. DESIGNING AND VERIFYING THE BEHAVIOR OF THE ROBOTS

This section describes the design and verification of the probabilistic local state-action map that dictates the behavior of the robots. We detail how the state-action map can be crafted such that the swarm will remain safe (Definition 3.2.1) and (possibly) also live (Definition 3.2.2). As we are dealing with robots with extremely limited knowledge, it can be expected that it is not always the case that both properties can be achieved at the same time. Safety is a hard requirement, but it will naturally restrict the ways in which the swarm can evolve. This could lead the swarm to a **livelock**.

Definition 3.4.1. A **livelock** is a situation in which the swarm will endlessly transition through a set of patterns (e.g., $P_0 \rightarrow P_1 \rightarrow P_2 \rightarrow P_0 \rightarrow P_1 \rightarrow P_2 \rightarrow P_0 \ldots$) and cannot transition to any other patterns.

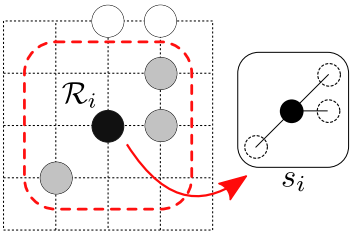
Furthermore, the limited view that the robots have of their surroundings limits the knowledge that they have of the structure, which may cause other (perhaps undesired) patterns to form. We will refer to this situation as **deadlock**.

Definition 3.4.2. A **deadlock** is a situation in which the swarm forms an undesired pattern $P \neq P_{des}$, where no robot in the swarm can take action.

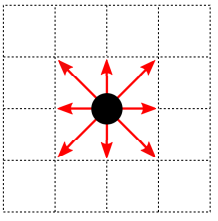
We have developed proof procedures to verify that livelocks or deadlocks will not happen. We will provide a set of conditions and checks that, if fulfilled, guarantee that the state-action map constructed for a given pattern is such that livelocks and deadlocks do not occur, and thus imply that the swarm is safe *and* live. The state-action map is developed and verified in a formal domain, assuming robots to be idealized agents existing on a 2D grid and operating in discrete time. Although this may seem restrictive, we will show in Section 3.6 how it can be used on robots operating in a realistic setting. The idealized framework is described in Section 3.4.1, and the method to design the probabilistic state-action map is detailed in Section 3.4.2. The conditions to prove whether a state-action map is safe, free of livelocks, and free of deadlocks are provided in sections 3.4.3, 3.4.4, and 3.4.5, respectively.

3.4.1. THE FORMALIZED FRAMEWORK

Consider N agents (idealized robots) that exist in an unbounded discrete 2D grid. Each robot is endowed with short range omni-directional relative sensors and knowledge of



(a) Example of an agent (black circle) in a local state s_i, given by the relative positions of its neighbors (white circles).



(b) Possible actions that an agent can take. It can move omnidirectionally in the grid.

Figure 3.1 Depictions of local state and the actions that an agent can take as used in this chapter.

north. Here we will focus our attention to robots with omni-directional sensing and motion capabilities, albeit the concepts presented hold for other state spaces and action spaces as well.

In the idealized case, each agent $\mathcal{R}_i$ can sense the location of its neighbors in the eight grid points that surround it (Figure 3.1a). Let s_i be the current state of agent $\mathcal{R}_i$, and let $\mathcal{S}$ be the local state space of the agents. It follows that $|S| = 2^8$, as it represents all local combinations of neighbors that could be sensed. To represent omni-directional motion, the agents are also able to move to any of the eight grid points surrounding it, as depicted in Figure 3.1b. This forms the action space of the agents, denoted $\mathcal{A}$. Note that other discretizations of $\mathcal{S}$ or $\mathcal{A}$ could also apply depending on the sensors and motors available on the robot of interest.

At time step $k = 0$, we assume the swarm begins in an arbitrary pattern P_0 on the grid. The only restriction on P_0 is that it has a connected sensing topology (Assumption A4). At each discrete time step, a random agent in the swarm takes an action and moves to a new location on the grid.[5]

3.4.2. DEVELOPING THE PROBABILISTIC STATE-ACTION MAP

In analogy to biological systems, the behavior that we will design replicates these three rules:

1. *be careful* (do not take actions that are in collision course with others),
2. *be social* (do not take actions whereby the swarm might locally break apart),
3. *be happy* (when in a desired local state, do not move).

Let us begin with the full state-action map, given by $\Pi \leftarrow \mathcal{S} \times \mathcal{A}$. With Π, any agent $\mathcal{R}_i$ in any state $s_i \in \mathcal{S}$ can stochastically take any action in $\mathcal{A}$. Naturally, this can readily

[5] At first sight, this seems rigid and difficult to implement on real robots. It can be in part justified under the intuition that the probability that two robots with different internal clocks begin to move at *exactly* the same time is small. A similar assumption was also suggested by Winfield et al. (2005b) as a method to model random concurrency in the swarm. In Section 3.6 we will show that, if robots are able to sense whether their neighbors are taking an action (assumption A3 from Section 3.2), then it can be exported to real robots. Multiple robots within the swarm will move, yet locally only one neighbor will move on a first-come first-served basis. In the idealized system, this is simplified to only one robot moving at one time step.

cause both collisions and/or group separation, which we want to avoid (if the swarm separates, then there is a chance that the two groups will never find each other, since they are operating in an unbounded environment). Therefore, we scan through Π to identify all state-action pairs that:

a) **are in the direction of a neighbor.**
These state-action pairs will lead to collisions (two agents occupying the same grid point). They form the set $\Pi_{collision}$.

b) **may cause the swarm to become disconnected.**
These actions will break the local connectivity of the agents (the local neighborhood splits into two or more groups). They form the set $\Pi_{separation}$.

We then define Π_{safe}:

$$\Pi_{safe} = \Pi - (\Pi_{collision} \cup \Pi_{separation}). \tag{3.1}$$

If the agents follow Π_{safe}, we can guarantee that the swarm remains safe while randomly reshuffling. The proofs for this are provided in Section 3.4.3.

Π_{safe} can be further modified to also make a desired pattern form. To do this, let us extract the set of local states that the agents are in when the desired pattern P_{des} is achieved. This forms a set of local *desired* states, denoted $\mathcal{S}_{des}$, examples of which are shown in Figure 3.2 for different patterns. If an agent $\mathcal{R}_i$ finds itself in a state $s_i \in \mathcal{S}_{des}$, then it should not move. The rationale behind this is that, from its perspective, the goal has been achieved (although this may or may not be the case at the global level, the robot does not know this). Therefore, for these states, we exclude all possible actions. The state-action map to form a given pattern P_{des} is:

$$\Pi_f = \Pi_{safe} - (\mathcal{S}_{des} \times \mathcal{A}). \tag{3.2}$$

With Π_f, the robots are capable of moving around until the swarm self-organizes into the desired pattern. Sections 3.4.4 and 3.4.5 provide the procedures to prove whether Π_f is such that the desired pattern always eventually forms from any initial pattern P_0.

The states in $\mathcal{S}$ can be divided into three groups:

Desired: When in these states, the agent should not move. Π_f does not map these states to any action. Desired states are grouped in the set $\mathcal{S}_{des}$.
Blocked: These are all states in $\mathcal{S} - \mathcal{S}_{des}$ where the agent cannot move because all actions are unsafe. Π_f does not map these states to any action. We group these states in the set $\mathcal{S}_{blocked}$.
Active: These are states that Π_f maps to one or more actions in $\mathcal{A}$. We group these states in the set $\mathcal{S}_{active}$.

Functionally speaking, $\mathcal{S}_{blocked}$ and $\mathcal{S}_{des}$ are equivalent. In either case, the agent will not move. Based on this, we also define the superset $\mathcal{S}_{static} = \mathcal{S}_{des} \cup \mathcal{S}_{blocked}$. Overall, the local behavior of an agent is summarized by the Finite State Machine (FSM) in Figure 3.3. Two examples of blocked states are shown in Figure 3.4a and 3.4b.

Additionally to the taxonomy above, we also define a set of states as **simplicial**.

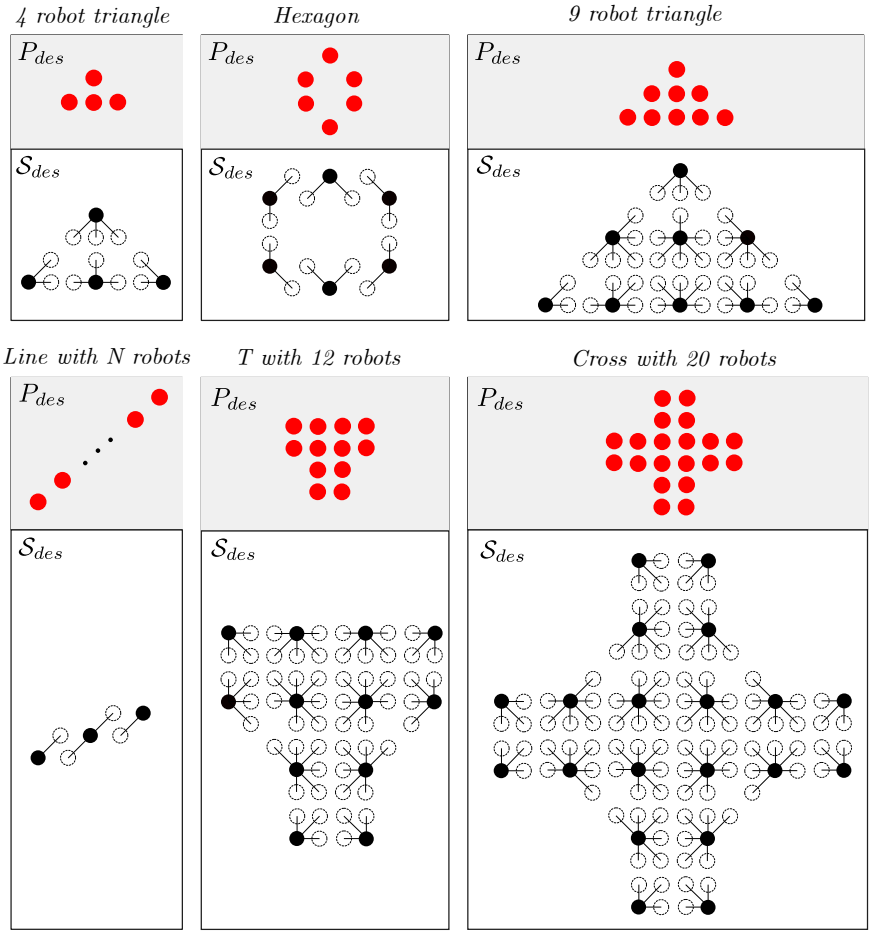


Figure 3.2 Examples of patterns and their respective desired states $\mathcal{S}_{des}$. The set $\mathcal{S}_{des}$ can be intuitively extracted from a pattern P_{des}, making it easy for a designer to define the local behavior of the robots.

Definition 3.4.3. A **simplicial** state is a state $s \in \mathcal{S} - \mathcal{S}_{blocked}$ for which its neighbors form only one clique.

Definition 3.4.4. A **clique** is a connected set of an agent's neighbors.

These definitions are borrowed from, but not equivalent to, the typical definitions of *simplicial node* and *clique* (van Steen, 2010). In standard graph theory, a simplicial node is a node whose neighboring nodes are *fully* connected among each other, not just connected. Similarly, a clique is a fully connected set of neighbors, whereas in our case it is just a connected set.

Simplicial states are grouped under the set $\mathcal{S}_{simplicial}$. All states in $\mathcal{S}$ that are not simplicial are denoted $\mathcal{S}_{\neg simplicial}$. From this, it follows that $\mathcal{S}_{blocked} \subseteq \mathcal{S}_{\neg simplicial}$. An example of a state that is both simplicial and active is shown in Figure 3.4c. By contrast,

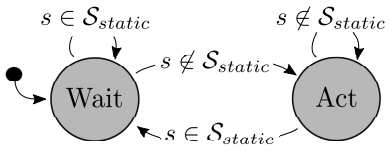


Figure 3.3 FSM of agent behavior.

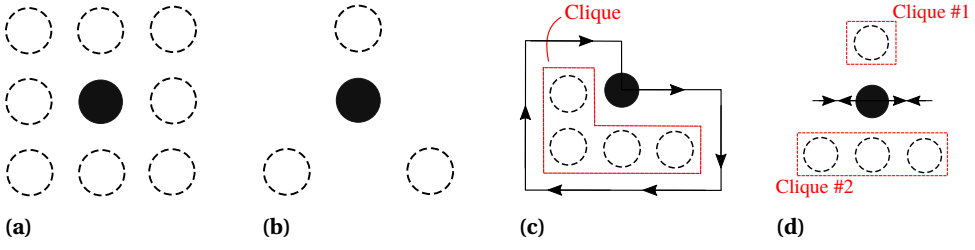


Figure 3.4 Examples of an agent (black circle) in different states depending on the relative positions of its neighbors (white circles). Specifically: **(a)** a state $s \in \mathcal{S}_{blocked}$, all actions will cause a collision; **(b)** a state $s \in \mathcal{S}_{blocked}$, all actions will either cause a collision or the local topology to disconnect; **(c)** a state $s \in \mathcal{S}_{active} \cap \mathcal{S}_{simplicial}$, its neighbors form one clique, which allows it to (potentially) travel freely away from or around its neighborhood; **(d)** a state $s \in \mathcal{S}_{active} \cap \mathcal{S}_{\neg simplicial}$, its neighbors form two cliques, the agent can move, but it cannot leave its neighborhood.

a non-simplicial active state is shown in Figure 3.4d. An agent in a simplicial state could potentially move without risking that the swarm ceases to be in a connected topology, unlike the non-simplicial case. Intuitively, agents who happen to be in a simplicial state thus have the potential to travel freely across the swarm and break livelocks. For this reason, simplicial states are going to be an important element to the local proof procedure to determine whether the swarm is free of livelocks, which can be found in Section 3.4.4.

3.4.3. VERIFYING SAFETY

Our swarm consists of several agents that can choose to take actions at any point in time. Safety can be guaranteed when agents do not simultaneously perform conflicting actions. To formalize this, we bring forward Proposition 3.4.1.

Proposition 3.4.1. *If the swarm never features more than one agent moving at the same time, then the swarm can remain safe.*

Proof. Consider a connected swarm organized into an arbitrary pattern P. At a given time $t = t_1$, agent $\mathcal{R}_i$ decides to take an action based on action space $\mathcal{A}$. This action should last until $t = t_2$. However, at time $t_1 < t < t_2$, an unsafe event takes place. It follows that the event must have been the fault of agent $\mathcal{R}_i$, because it was the only agent that moved. Therefore, if agent $\mathcal{R}_i$ could select only from safe actions, this would be sufficient to guarantee that the swarm is safe at time $t = t_2$. ■

Proposition 3.4.1 only applies to the idealized system and cannot be implemented on the real system where robots use local clocks. This explains the importance for Assump-

tion A3 from Section 3.2: an agent must know whether its neighbors are executing an action. If then a robot does not move whenever one of its neighbors is moving (on a first-come-first-served basis), then the swarm can locally approach the formal requirement of Proposition 3.4.1 even if several robots may be moving in different neighborhoods. We will return to this in Section 3.6.

Under the assumption that the conditions of Proposition 3.4.1, if Π_{safe} meets the conditions in Propositions 3.4.2 and 3.4.3, then the swarm is safe.

Proposition 3.4.2. *If an agent is the only agent moving in the entire swarm, and Π_{safe} is such that the agent can only select actions in directions that can be sensed by its onboard sensors, then no collisions will occur in the swarm.*

Proof. Consider an agent $\mathcal{R}_i$ in a swarm. Following Proposition 3.4.1, we know that the agent will be the only agent to move. The agent moves in the environment according to the action space $\mathcal{A}$. If all actions in $\mathcal{A}$ lead to a location that is already sensed, then agent $\mathcal{R}_i$ can establish whether the action will cause a collision, and it can choose against performing these actions. ■

Proposition 3.4.3. *If an agent is the only agent moving in the entire swarm, and Π_{safe} is such that the agent can only select actions where, at its next location, all its prior neighbors and itself remain connected, then the whole swarm will remain connected.*

Proof. Consider a connected swarm of N agents. The graph of the swarm is connected if any node (agent) $\mathcal{R}_i$ features a path to any other node (agent) $\mathcal{R}_j$. Consider the case where agent $\mathcal{R}_i$ takes an action. If, following the action, agent $\mathcal{R}_i$ is still connected to all its original neighbors, then the connectivity of the graph was not affected. If agent $\mathcal{R}_i$ only selects actions where, at its final position, this principle is respected, then it will be able to move while guaranteeing that the swarm remains connected. ■

3.4.4. VERIFYING AGAINST THE PRESENCE OF LIVELOCKS

We now provide the proof procedure to check that the system can form the patterns and will do so without ending up in livelocks. Let us begin at the global level and define a directed graph $G_P = (V_P, E_P)$. The vertices V_P represent all possible patterns that the swarm could generate. The edges E_P represent all global pattern transitions that could take place whenever one agent in the swarm executes an action from Π_f. Our final objective is to establish whether Π_f is such that G_P always features a path from any vertex (i.e. an arbitrary initial pattern P_0) to the global desired pattern P_{des}. If this is the case, then it is proven that livelocks will not occur.

This problem could be tackled by directly inspecting G_P, but an exhaustive computation of G_P quickly becomes intractable (Dixon et al., 2012). Otherwise, livelocks (if existent) could be found using heuristic search algorithms, as done by Sapin (2010) to find loops (gliders) for Game of Life Cellular Automata. However, should we not find any, then it is not guaranteed that livelocks do not exist. It only means that the heuristic search did not find them. We thus take a different route and extract local conditions that, if respected, *also guarantee the global property*. Although this comes at the cost of imposing certain local restrictions that may not necessarily be required at the global level,

it bears the advantage that they can be verified at the local level and thus independently of the number of robots in the swarm.

In the following analysis, it is assumed that P_0 always has a connected sensing topology (Assumption A4) and that it has N_{des} agents, where N_{des} is the number of agents required to form P_{des}. We also assume that deadlocks are not present. This is not required, and is merely done for simplicity. The absence of deadlocks can be verified independently by the methodology in Section 3.4.5.

ENSURING MOTION

We begin by showing that, if no deadlocks are present, then any pattern $P \neq P_{des}$ will always have at least one agent in an active state, as per Lemma 3.4.1.

Lemma 3.4.1. *For a swarm of N_{des} agents, if $\mathcal{S}_{static}$ is such that the desired pattern P_{des} is unique (i.e., no deadlocks can occur), any arbitrary pattern $P \neq P_{des}$ will feature at least* 1 *agent with a state $s \in \mathcal{S}_{active}$.*

Proof. By definition: $\mathcal{S}_{static} \cap \mathcal{S}_{active} = \emptyset$ and $\mathcal{S}_{static} \cup \mathcal{S}_{active} = \mathcal{S}$. For a swarm of N_{des} agents that can be in states $s \in \mathcal{S}$, N_{des} instances of states $s \in \mathcal{S}_{static}$ can only coexist into P_{des}, which is known to be the unique outcome. Therefore, it follows that any other pattern must feature at least one agent that is in a state $s \notin \mathcal{S}_{static}$, meaning that it is in a state $s \in \mathcal{S}_{active}$. ■

Lemma 3.4.1 says that if the swarm cannot be in a deadlock then it must always have at least one agent that is active, unless P_{des} forms. Therefore, if we can establish that no livelocks can occur, then we know that the swarm will always eventually self-organize into P_{des}. To do this, we need to analyze the local state transitions that an agent can experience over time.

THE LOCAL STATE TRANSITION GRAPHS

To conduct a local analysis, let us look at Π_f and define its role from the perspective of an agent. When an agent in the swarm experiences a transition from state s to a state s', this can be due to three events:

> **Event 1** The agent was in a state $s \in \mathcal{S}_{active}$ and computed an action in Π_f. When this happens, some neighbors may disappear from view, while new neighbors may come into view.
> **Event 2** The agent did not move, but one of its neighbors did. In this case, the neighbor may also have moved away from view.
> **Event 3** The agent did not move, but some other agent which was previously not in view has moved into view and has become a new neighbor.

Based on the above, let $G_{\mathcal{S}} = (V_{\mathcal{S}}, E_{\mathcal{S}})$ be a directed graph where each vertex $V_{\mathcal{S}}$ represents a different local state $s \in \mathcal{S}$, such that $V_{\mathcal{S}} = \mathcal{S}$, and the edges $E_{\mathcal{S}}$ represent all local state transitions that an agent could experience. More specifically, let us define $E_{\mathcal{S}} = E_1 \cup E_2 \cup E_3$, where E_1 are all edges describing Event 1, E_2 are all edges describing Event 2, and E_3 are all edges describing Event 3. Similarly, $G^1_{\mathcal{S}} = (V_{\mathcal{S}}, E_1)$, $G^2_{\mathcal{S}} = (V_{\mathcal{S}}, E_2)$, $G^3_{\mathcal{S}} = (V_{\mathcal{S}}, E_3)$. The graphs G^1_S, G^2_S, and G^3_S are illustrated in Figure 3.5.

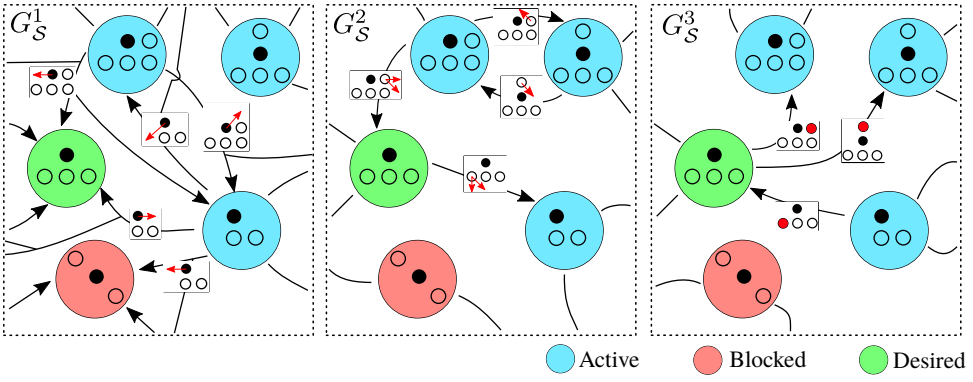


Figure 3.5 Exemplary depiction of portions of $G^1_{\mathcal{S}} = (V_{\mathcal{S}}, E_1)$, $G^2_{\mathcal{S}} = (V_{\mathcal{S}}, E_2)$, and $G^3_{\mathcal{S}} = (V_{\mathcal{S}}, E_3)$ (from left to right). Green nodes indicate a desired state, blue nodes indicate an active state, and red nodes indicate a blocked state. The states are visually depicted within each node, showing the agent (in black) and its neighbors. In $G^1_{\mathcal{S}}$ the edges E_1 represent transitions where the agent itself executes an action (shown by the arrows), from which it may probabilistically end up in several local states depending on what it finds after it has moved (notice the bifurcations in the arrows). Note how in $G^1_{\mathcal{S}}$ both the green node (desired) and the red node (blocked) act as sinks, because in these states the agent will not take actions. In $G^2_{\mathcal{S}}$ the edges E_2 represent state transitions experienced by the agent when a neighbor of the agent executes an action. This is shown by one of the neighbors (in white) taking an action. Finally, in $G^3_{\mathcal{S}}$ the edges E_3 represent state transitions that occur when another agent moves into view and becomes a new neighbor. This is shown by the red agents in the transitions.

LOCAL ACHIEVABILITY OF DESIRED STATES

As a prerequisite for a pattern to form, we require that Π_f ensures that any local state can experience a local transition to a desired local state. If this is the case, we will say that the pattern is **achievable**, as defined by Definition 3.4.5.

Definition 3.4.5. A pattern P_{des} is **achievable** if all local states $\mathcal{S}_{des}$ can be reached starting from any local state in $\mathcal{S}$.

If a pattern is achievable, then there are no restrictions on the local states that can be present in P_0, else there might be certain starting patterns with agents in local states that are unable to transition to certain desired states. This is proven by Lemma 3.4.2.

Lemma 3.4.2. *If the digraph $G^1_{\mathcal{S}} \cup G^2_{\mathcal{S}}$ shows that each state in $\mathcal{S}$ features a path to each state in $\mathcal{S}_{des}$, then P_{des} is achievable independently of the local states that compose P_0.*

Proof. P_{des} is formed if and only if all agents have a state $s \in \mathcal{S}_{des}$, where $\mathcal{S}_{des} \subseteq \mathcal{S}$. Consider an arbitrary initial pattern P_0 for which the local states of the agents form an arbitrary set $\mathcal{S}_0$. Via Lemma 3.4.1 we know that there is at least one agent in the swarm that is active for any pattern $P_0 \neq P_{des}$, and in turn any set of states $S_0 \neq S_{des}$. As the active agents move, they will experience transitions described by $G^1_{\mathcal{S}}$, and their neighbors will experience transitions described by $G^2_{\mathcal{S}}$. By the unified graph $G^1_{\mathcal{S}} \cup G^2_{\mathcal{S}}$ we describe the local transitions that an agent experiences as it moves and as its neighbors move. Consider a state $s \in \mathcal{S}_0$ that is incapable (either by its own actions or by the actions of its potential neighbors) to transition to a state in $\mathcal{S}_{des}$. It follows that having this state in S_0 may mean that a state in S_{des} cannot be achieved, and in turn that P_{des} cannot be

realized. However, if it is possible for any state in $\mathcal{S}$ to experience local transitions such that it may reach any state $\mathcal{S}_{des}$, it follows that P_{des} is achievable independently of the local states that compose P_0 (i.e, the set $\mathcal{S}_0$), because there is no state $s \in \mathcal{S}_0$ that is incapable of experiencing the necessary transitions that would lead it to be in a state $\mathcal{S}_{des}$. By purposely ignoring the role of $G^3_{\mathcal{S}}$, we restrict the analysis such that:

1. Any state s that has too few links for a desired state will have to be active and move to a position where it is surrounded by enough agents. It cannot wait for a local desired state to arise by other agents moving in from outside of its neighborhood.
2. Any state $s \in \mathcal{S}_{blocked}$ can only become active by the actions of a neighbor.
3. The transitions that occur must occur because of changes in the local neighborhood.

This additional restriction ensures that the system can rely on the actions of an agent and/or its neighbors. ■

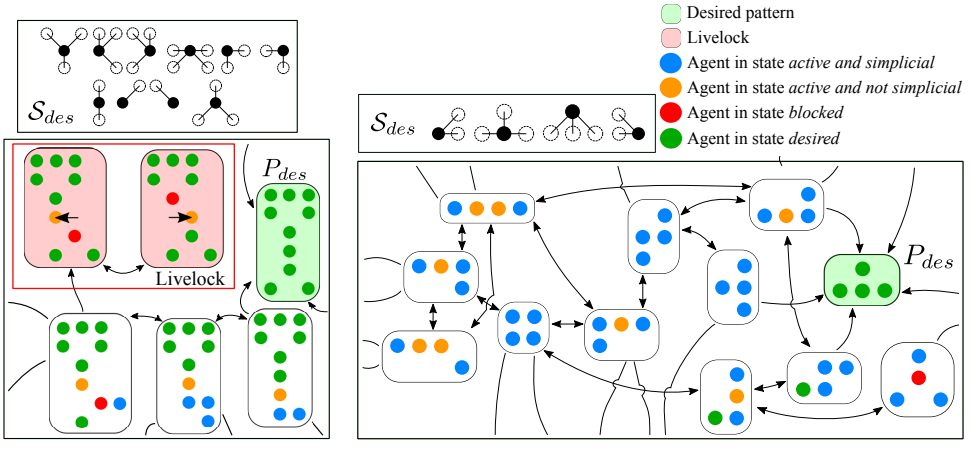
By fulfilling the condition of Lemma 3.4.2, we ensure that any initial state could potentially turn into a desired state and avoid placing local level restrictions on P_0. However, this is still only a local property, and it does not yet fully confirm that, at the global level, P_{des} will always eventually form from any initial pattern P_0, which is the property that we wish to verify. We continue our analysis in the text below.

ENSURING THE PRESENCE OF AGENTS WITH SIMPLICIAL STATES

In Section 3.4.2 we have already discussed that an agent in a state $s \in \mathcal{S}_{simplicial} \cap \mathcal{S}_{active}$ can potentially move away from its neighborhood. This is an important property. Intuitively, an agent in this state has sufficient freedom for the swarm to escape any livelock. To exemplify this, let us once again consider the global graph G_P as introduced at the beginning of Section 3.4.4, and consider the example in Figure 3.6a. When the global pattern formed by the swarm is such that no agent is in a simplicial state, then the swarm is unable to exit the livelock. There is an agent in the swarm that can move, but, because Π_f is designed to keep the swarm safe, it cannot leave its neighborhood and can only move left and right. The result is that the swarm cycles endlessly between the two patterns. By contrast, the patterns in Figure 3.6b always have an agent in a simplicial state and no livelocks occur. In this section, we introduce the local conditions necessary such that any vertex (pattern) in G_P always eventually transitions to a pattern with at least one agent with a state that is both active and simplicial (unless P_{des} is reached). This will be an important stepping stone to the final verification in Theorem 3.4.1.

Let P_{AS} be the set of all patterns where one or more agents are in a state $s \in \mathcal{S}_{simplicial} \cap \mathcal{S}_{active}$ (the subscript AS stands for Active and Simplicial). We wish to ensure a pattern $P \in P_{AS} \cup P_{des}$ will be reached from any other pattern. This is verified via Lemma 3.4.3. In this Lemma we also make use of a graph $G^{2r}_{\mathcal{S}} \subseteq G^2_{\mathcal{S}}$, which only considers the transitions in $G^2_{\mathcal{S}}$ that do not feature a neighbor leaving the neighborhood when moving, but only holds transitions about the agent. We also single out a special state in $\mathcal{S}_{blocked}$, which is the one that is fully surrounded by neighbors as in Figure 3.4a. We refer to this state as $s_{surrounded}$.

Lemma 3.4.3. *If the following conditions are satisfied:*



(a) Example of a livelock. The livelock cannot be exited because only the robots with an active state will move (shown in orange).

(b) Pattern where a livelock is not possible.

Figure 3.6 Illustrations of how a swarm can transition between different patterns, based on movements of the agents that are in active states. More specifically, the figure shows a portion of G_P for two possible desired patterns. The arrows between the nodes are swarm transitions that happen as a result of one of the robots taking an action. Notice that the livelock in (a) does not feature any agents with a state that is both active and simplicial. There is an agent in an active state (in the middle), but because it is not simplicial it cannot escape its neighborhood and repeatedly moves right and left, causing the livelock.

1. *for all states* $s \in \mathcal{S}_{static} \cap \mathcal{S}_{\neg simplicial} - s_{surrounded}$, *none of the cliques of each state can be formed only by agents that are in a state* $s \in \mathcal{S}_{des} \cap \mathcal{S}_{simplicial}$,
2. $G_{\mathcal{S}}^{2r}$ *shows that all static states with two neighbors will directly transition to an active state,*

then a pattern in $P \in P_{AS} \cup P_{des}$ *will always be reached from any other pattern* $P \notin P_{AS} \cup P_{des}$.

Proof. Consider an agent $\mathcal{R}_i$ with state $s_i \in \mathcal{S}_{static} \cap \mathcal{S}_{\neg simplicial}$. By definition, s_i must have more than one clique, unless $s_i = s_{surrounded}$. If $s_i = s_{surrounded}$ and $P \neq P_{des}$ then one of $\mathcal{R}_i$'s neighbors must be in a state $s \in \mathcal{S}_{active} \cap \mathcal{S}_{simplicial}$, or else there must exist other agents beyond $\mathcal{R}_i$'s direct neighborhood. If $s_i \neq s_{surrounded}$, then the neighbors of agent $\mathcal{R}_i$ form two or more cliques. In all cases, the pattern $P \neq P_{des}$ extends in two or more directions that stem from agent $\mathcal{R}_i$. If we trace any branch, because only a finite number of agents N_{des} exists, we have the two following possible situations:

1. The branch eventually features an agent $\mathcal{R}_j$ with state $s_j \in \mathcal{S}_{simplicial}$. In the extreme, this is a leaf on the edge of the pattern. Here, we can have two situations:
   (a) $s_j \in \mathcal{S}_{des} \cap \mathcal{S}_{simplicial}$. If this exists, then the simplicial agent is also static. Therefore, it is possible that the entire pattern does not feature *any* active and simplicial agent.

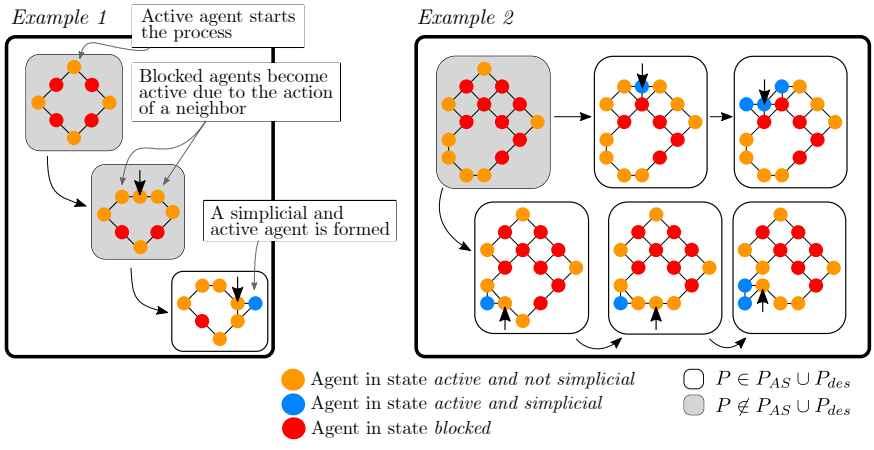


Figure 3.7 Illustration of two exemplary loops that "collapse". Notice that the active states present at the borders cause a chain reaction until eventually a simplicial active agent is present. This is a property that can be determined by inspecting $G^{2r}_{\mathcal{S}}$, which will show that the static agents will become active and propel the chain reaction.

(b) If $s_j \notin \mathcal{S}_{des} \cap \mathcal{S}_{simplicial}$, then $s_j \in \mathcal{S}_{active} \cap \mathcal{S}_{simplicial}$ and so we are done.

If, by design, states $s \in \mathcal{S}_{des} \cap \mathcal{S}_{simplicial}$ cannot be combined to form the clique of a state in $\mathcal{S}_{static} \cap \mathcal{S}_{\neg simplicial} - s_{surrounded}$, then it is guaranteed that $s_j \notin \mathcal{S}_{des} \cap \mathcal{S}_{simplicial}$. Therefore, we can *locally* impose that situation (b) always occurs, that situation (a) never occurs, and we thus guarantee that $s_j \in \mathcal{S}_{active} \cap \mathcal{S}_{simplicial}$. This is the first condition of this Lemma.

2. If all branches from agent $\mathcal{R}_i$ only feature non-simplicial states, then this is only the case if the branches form loops, otherwise at least one leaf would be present as in situation 1. However, it can be ensured that a loop will always collapse and feature one simplicial active agent. In a loop, all agents have two cliques, each formed by one neighbor. $G^{2r}_{\mathcal{S}}$ tells whether any static agent with two neighbors, by the action of its neighbors, will become active. This is the second condition of this Lemma. If this is the case for all states, then we know that the action of any neighbor will cause a chain reaction about the loop. This will eventually cause the loop to collapse about one corner point and create a simplicial leaf, unless P_{des} forms. In either case, we reach a pattern $P \in P_{AS} \cup P_{des}$. The collapse of two exemplary loops is depicted in Figure 3.7.

In summary, by creating the conditions such that situation 1(a) never occurs, we restrict the possible patterns that can exist outside of $P_{AS} \cup P_{des}$ to patterns with only loops (situation 2). If P_0 is a loop, then through $G^{2r}_{\mathcal{S}}$ we know that loop patterns will collapse into a pattern that exists within $P_{AS} \cup P_{des}$. Else, P_0 already exists within $P_{AS} \cup P_{des}$. This means that any pattern P_0 will either exist within $P_{AS} \cup P_{des}$, or will transition into $P_{AS} \cup P_{des}$. ■

LOCAL PROOF CONDITIONS TO GUARANTEE THAT LIVELOCKS DO NOT OCCUR

With the conditions from Lemma 3.4.3 we ensure that a simplicial active agent will always be present regardless of P_0. We can now introduce Theorem 3.4.1, which we use to determine that P_{des} will eventually form from P_0 without livelocks.

Theorem 3.4.1. *If the following conditions are satisfied:*

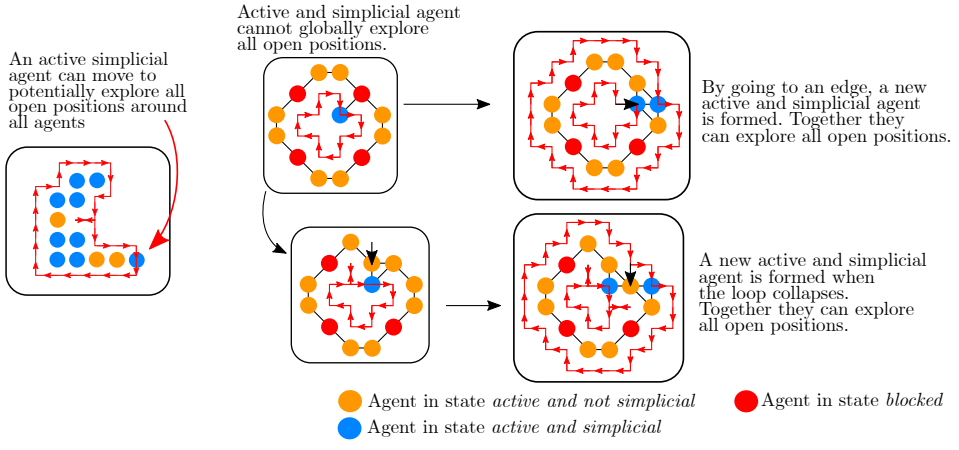
1. *P_{des} is achievable,*
2. *a pattern in $P \in P_{AS} \cup P_{des}$ will always be reached from any other pattern $P \notin P_{AS} \cup P_{des}$,*
3. *$G^1_{\mathcal{S}}$ shows that any agent in any state $s \in \mathcal{S}_{active} \cap \mathcal{S}_{simplicial}$ can move to explore all open positions surrounding its neighbors (with the exception of when a loop is formed or when it enters a state $s \in \mathcal{S}_{static}$),*
4. *in $G^3_{\mathcal{S}}$, any agent in any state $s \in \mathcal{S}_{static}$ only has outward edges toward states $s \in \mathcal{S}_{active}$ (with the exception of a state that is fully surrounded along two or more perpendicular directions),*

then P_{des} will always eventually be reached from any initial pattern P_0.

Proof. Consider a swarm of N_{des} agents arranged in a pattern P_0. If P_{des} is achievable, via Lemma 3.4.2, P_0 can be composed of any combination of local states without impacting the local ability of the agents to transition into the states $\mathcal{S}_{des}$ (this is the first condition in this theorem). Then, through Lemma 3.4.3 we know that if $P_0 \notin P_{AS} \cup P_{des}$, then it will always eventually form a pattern $P \in P_{AS} \cup P_{des}$ (this is the second condition in this theorem). In the following, we will show that any pattern $P \in P_{AS} \cup P_{des}$ will keep transitioning until it forms P_{des}. We observe the case where at least one agent, agent $\mathcal{R}_i$, exists with state $s_i \in \mathcal{S}_{active} \cap \mathcal{S}_{simplicial}$. As agent $\mathcal{R}_i$ moves, one of the following events can happen:

1. Agent $\mathcal{R}_i$ enters a state $s'_i \notin \mathcal{S}_{simplicial}$. Via Lemma 3.4.3, at least one other agent is (or will be) in state $s \in \mathcal{S}_{active} \cap \mathcal{S}_{simplicial}$, taking us to point 3 in this list.
2. Agent $\mathcal{R}_i$ enters a state $s'_i \in \mathcal{S}_{static}$. If P_{des} is not yet achieved, then at least one other agent in the swarm is in an active state (Lemma 3.4.1). If the active agent(s) are in state $s \in \mathcal{S}_{active} \cap \mathcal{S}_{\neg simplicial}$, then this takes us back to point 1 in this list. If the active agent(s) are in state $s \in \mathcal{S}_{active} \cap \mathcal{S}_{simplicial}$, this takes us to point 3 in this list.
3. Agent $\mathcal{R}_i$, and/or the agent(s) taking over, keeps moving and each time enters a state $s'_i \in \mathcal{S}_{active} \cap \mathcal{S}_{simplicial}$. Via $G^1_{\mathcal{S}}$ we know that it can potentially explore all open positions surrounding all its neighbors (this is the third condition of this theorem). As it moves, its neighbors also change, such that it always can potentially explore all open positions around all agents, and thus *all open positions in the pattern* (see Figure 3.8a for a depiction). This means that the swarm can evolve toward a pattern that is closer to the desired one.

Any situation will always develop into the situation of point 3. This is free of livelocks, as all possible livelock situations are mitigated:



(a) Simplicial agent that can travel to all open positions in the pattern.

(b) Two possibilities for how, should the agent be globally surrounded in a loop and unable to travel to all open positions, then a new simplicial and active agent will take over.

Figure 3.8 Illustration of how an agent with a state that is active and simplicial can travel to all open positions in the structure.

1. It may happen that a simplicial and active agent cannot actually visit all open positions in the swarm because, at the global level, it is enclosed in a loop by the other agents. Alternatively, it may happen that it itself creates a loop while moving (this is the first exception to condition 3 of this theorem). By Lemma 3.4.3, the loop will always collapse, meaning that a new agent will enter a state $s \in \mathcal{S}_{active} \cap \mathcal{S}_{simplicial}$. The new agent will be able to travel to all positions external to the loop, avoiding a livelock. This resolution is depicted in Figure 3.8b.
2. Agent $\mathcal{R}_i$ can travel about all open positions in the swarm. Let us assume the extreme case in which $\mathcal{R}_i$ is the only agent that can potentially do this in the entire swarm. Via $G^3_{\mathcal{S}}$, we can verify that, unless P_{des} forms, this must eventually cause at least one static agent to become active (following the fourth condition of this theorem). Consider a static agent $\mathcal{R}_j$ which becomes active when $\mathcal{R}_i$ becomes its neighbor. This may lead to one of the following developments, all of which avoid livelocks.
   (a) Agent $\mathcal{R}_i$ remains in state $s'_i \in \mathcal{S}_{active} \cap \mathcal{S}_{simplicial}$. The pattern can keep evolving further. A livelock is avoided.
   (b) Agent $\mathcal{R}_i$ enters a state $s'_i \in \mathcal{S}_{active} \cap \mathcal{S}_{\neg simplicial}$. By Lemma 3, another simplicial and active agent will be present elsewhere in the swarm. A livelock is avoided.
   (c) As per the second exception to condition 3 of this theorem, agent $\mathcal{R}_i$ enters a state $s \in \mathcal{S}_{static}$ upon becoming a neighbor of agent $\mathcal{R}_j$, before agent $\mathcal{R}_j$ moves. In this case, the departure of agent $\mathcal{R}_j$ will bring it back to a state

$s_i \in \mathcal{S}_{active}$ taking us back to points 2(a) or 2(b) in this list.

(d) Agent $\mathcal{R}_i$ enters a state $s \in \mathcal{S}_{static}$ upon becoming a neighbor of agent $\mathcal{R}_j$, after agent $\mathcal{R}_j$ moves. At this point, either $\mathcal{R}_j$ will move back to its original position and agent $\mathcal{R}_i$ will return to to a state $s_i \in \mathcal{S}_{active} \cap \mathcal{S}_{simplicial}$ and keep moving, or $\mathcal{R}_j$ will continue to move elsewhere. In either case, when agent $\mathcal{R}_j$ moves, it will also cause other neighbors to become active. In turn, these will move, and $\mathcal{R}_i$, who also neighbors them, will then return to being in an active state, bringing us back to to points 2(a) or 2(b) in this list.

(e) Agent $\mathcal{R}_i$, after agent $\mathcal{R}_j$ has moved, enters the position (and state) that was originally taken by agent $\mathcal{R}_j$. As in point 2(d) in this list, it is not possible that agent $\mathcal{R}_j$ will always only free $\mathcal{R}_i$ in exactly the same way that agent $\mathcal{R}_i$ freed agent $\mathcal{R}_j$, because $G^3_{\mathcal{S}}$ shows that motions of agent $\mathcal{R}_j$ will free any static agent in the neighborhood, and not just agent $\mathcal{R}_i$.

There is an exception to the fourth condition of this theorem, which is the static state that is fully surrounded by other agents along two perpendicular axes. In this case, $G^3_{\mathcal{S}}$ may show that it will not directly become active. However, it is trivially impossible (since there is a finite number of agents) for the swarm to only feature agents that are surrounded. A situation where *all* agents are *all* surrounded cannot occur; at least one agent will not be surrounded. This justifies the exception to the fourth condition in this theorem.

With the above it is confirmed that 1) any open position in the pattern can potentially be filled, and 2) no livelocks will arise. This means that the swarm will evolve into all patterns in $P_{AS} \cup P_{des}$. Therefore, P_{des} will always eventually be formed starting from any pattern P_0. ■

We thus conclude the proof procedure to check that livelocks will not occur. We showed that by fulfilling a set of local conditions we can determine that the pattern will be achieved from any initial configuration of the swarm. These conditions, being local in nature, are more strict than it is potentially required at the global level. It can be seen that it is actually the agents' ability to stochastically select from a pool of actions that endows them with the potential to keep exploring new neighborhoods and ensure that the swarm keeps evolving without livelocks. A primary condition is the important presence of agents in simplicial active states, which brings interesting insights. Here, we note the following:

- Any desired state with only one neighbor violates the first condition of Lemma 3.4.3. This is because this desired state can form the clique of a blocked state on its own. If this occurs, the local conditions are too restrictive to formally guarantee that the swarm will not run into livelocks.
- Removing a dependency on north (Assumption A1) may lead to a violation of the first condition of Lemma 3.4.3. This is because desired states become rotation invariant.

3.4.5. VERIFYING AGAINST THE PRESENCE OF DEADLOCKS

We now have means to verify that no livelocks will occur, but to know that the swarm will always self-organize into the desired pattern, we must also show that no deadlocks

can form. That is, there can be no pattern other than the desired pattern P_{des} where none of the agents can take an action. Let us begin, once again, with G_P as introduced in Section 3.4.4. Similarly as to the livelock, we could search exhaustively though G_P for possible nodes with no outgoing edges. Alternatively, we could repeatedly simulate the swarm and experimentally check whether any other pattern forms, but this would not strictly ensure that other patterns cannot manifest.[6] In this work, we still choose to search through G_P. However, to counter the computation explosion, we show that if no livelock exists then it is only necessary to search through a small subset of G_P, and we also provide a method to quickly scan through the remaining subspace (alternatively, if livelocks may exist, then there is technically also no reason to search for deadlocks since we already know that the swarm may evolve undesirably).

RESTRICTING THE SEARCH SPACE

By definition, deadlocks are patterns $P \neq P_{des}$ where all agents are in a state $s \in \mathcal{S}_{static} = \mathcal{S}_{des} \cup \mathcal{S}_{blocked}$. By Proposition 3.4.4 the search space is restricted to patterns that contain at least one agent with state $s \in \mathcal{S}_{des}$.

Proposition 3.4.4. *A deadlock cannot consist only of agents with state $s \in \mathcal{S}_{blocked}$.*

Proof. Following the same reasoning in Lemma 3.4.3, any finite pattern, at its edges, features one of the following:

1. an agent with state $\mathcal{S}_{simplicial}$. By definition, however, $\mathcal{S}_{blocked} \cap \mathcal{S}_{simplicial} = \emptyset$,
2. agents with a state $\mathcal{S}_{\neg simplicial}$ forming a loop boundary. Then, at least one agent must be in a state $\mathcal{S}_{des}$, else it would be in a state $s \in \mathcal{S}_{active}$, which we are not concerned with.

Therefore, in both occurrences, there must be at least one agent with state $s \notin \mathcal{S}_{blocked}$. ■

Then, for a certain class of patterns, it can be shown that *all* agents must be in a state $s \in \mathcal{S}_{des}$, as per Proposition 3.4.5.

Proposition 3.4.5. *If the conditions of Lemma 3.4.3 hold and $\mathcal{S}_{des} \subseteq \mathcal{S}_{simplicial} \cup s_{surrounded}$, then all agents in a deadlock must be in a state $s \in S_{des}$.*

Proof. If $\mathcal{S}_{des} \subseteq \mathcal{S}_{simplicial} \cup s_{surrounded}$, then all states in $\mathcal{S}_{des}$ are either simplicial or $s_{surrounded}$. By the first condition of Lemma 3.4.3, none of the states in $\mathcal{S}_{des}$ can satisfy the cliques of any state $\mathcal{S}_{static} \cap \mathcal{S}_{\neg simplicial} - s_{surrounded}$. This means that they cannot ever coexist in the same pattern. By Proposition 3.4.4, however, at least one agent must exist with state $s \in \mathcal{S}_{des}$. Therefore, all agents in the spurious pattern must be in a state $s \in \mathcal{S}_{des}$. Alternatively, this proposition can also be verified by a local inspection. ■

Therefore, if a pattern is such that $\mathcal{S}_{des} \subseteq \mathcal{S}_{simplicial} \cup s_{surrounded}$, we can further restrict our search to patterns that only have agents in $\mathcal{S}_{des}$. The patterns shown in Figure 3.2, with the exception of the hexagon and the line, meet this condition (the line, however, also does not meet Lemma 3.4.3).

[6] Considering that the self-organization of the pattern resembles an emergent property, Darley (1994) argues that this would be more efficient.

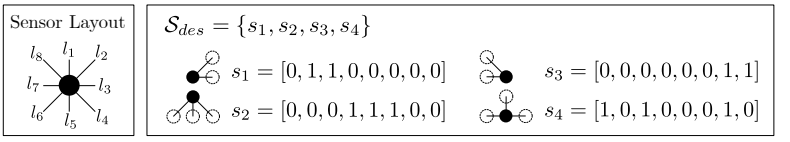


Figure 3.9 Example of a set S_{des} for a triangle with 4 agents. $l_i, i = 1,\ldots,8$ represent the eight directions where a neighbor is expected. In a binary representation, if $l_i = 0$ then a neighbor is not expected in that direction, and if $l_i = 1$ then a neighbor is expected. The states $s_1,\ldots,s_4$ are realizations of this.

3

FINDING SPURIOUS PATTERNS

In this section we detail our implementation to find spurious patterns for an arbitrary set $\mathcal{S}_{des}$. To sort through the possibilities more efficiently, we analyze state combinations to determine whether they could potentially make a pattern. By first analyzing combinations we need not concern ourselves with the spatial arrangement but only determine whether the states could potentially be combined together independently of order. It is only if such a combination is found that we explore its spatial arrangement, which is done by the use of spanning tree graphs.

Preliminaries Consider a set $\mathcal{S}_{des}$. Because the agents can sense each other omnidirectionally, then any two states "match" when two neighbors could have those two states and be neighbors. We introduce two tools to summarize how the states in an arbitrary set $\mathcal{S}_{des}$ match:

- **Match-Direction matrix**, denoted D, is a square matrix ($d \times d$) that holds the directions along which any two states in $\mathcal{S}_{des}$ are reciprocal to each other.
- **Match-Count matrix**, denoted M, is a square matrix ($d \times d$) that holds the *number of* directions along which any two states in $\mathcal{S}_{des}$ match. M is symmetric. Intuitively, this is because if agent $\mathcal{R}_i$ is a neighbor of agent $\mathcal{R}_j$, then agent $\mathcal{R}_j$ is a neighbor of agent $\mathcal{R}_i$.

For example, consider the set $\mathcal{S}_{des} = \{s_1, s_2, s_3, s_4\}$ in Figure 3.9. For this set:

$$D(\mathcal{S}_{des}) = \begin{Bmatrix} - & [l_2] & - & [l_3] \\ [l_6] & - & [l_4] & [l_5] \\ - & [l_8] & - & [l_7] \\ [l_7] & [l_1] & [l_3] & - \end{Bmatrix} \qquad M(\mathcal{S}_{des}) = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

All entries with 0 in $M(\mathcal{S}_{des})$ correspond to empty entries in $D(\mathcal{S}_{des})$. From $M(\mathcal{S}_{des})$ we can quickly extract that state s_1 can never connect to itself or to s_3, but it can connect to states s_2 and s_4.[7] With $D(\mathcal{S}_{des})$ we can see that s_1 can match with s_2 along l_2 and with s_4

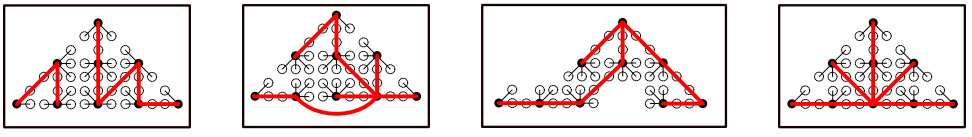
[7] If analyzed visually, s_1 cannot connect to itself because it expects a neighbor to its right (l_3) and top-right (l_2), yet if it were to connect to itself, then the robot next to it would have a neighbor to its left, which thus cannot be s_1. Also, s_1 cannot connect to s_3 because s_1 does not expect a neighbor to be right above itself (at l_1), whereas s_3 would expect a neighbor to be there because it expects a neighbor on its top-left (at l_8), and vice versa.

along l_3. Note that $D(\mathcal{S}_{des})$, although not strictly symmetric, also has a symmetry to it: each link always features, at its symmetry position, a link along the opposite direction. For example, if s_1 matches with s_2 along direction l_2, then s_2 matches with s_1 along l_6. Therefore, the two matrices essentially provide a local summary of which states can be neighbors and which cannot. This will be used in the following analysis.

Combination analysis A combination of local states should meet a set of conditions independently of how they are arranged. Using these conditions, it is possible to quickly restrict the search space without performing a more computationally expensive spatial analysis. The conditions are:

1. **The topology graph is finite and undirected.** For any finite undirected graph $G = (V, E)$, the sum of the vertex degrees must be equal to twice the amount of edges (Ismail et al., 2009; van Steen, 2010). As a consequence, the graph will always feature an even amount of vertices with an odd degree. This is known as the handshaking theorem (Ismail et al., 2009). In our context, this translates to the fact that any valid combination should feature an even amount of states that expect an odd number of neighbors.
2. **The neighbor expectations are reciprocal.** In a combination, each state that expects a neighbor in one direction should have at least another state expecting a neighbor in the opposite direction.
3. **The pattern is finite.** For each direction, there should be at least one state in a combination that does not expect a neighbor along that direction. Else, the pattern cannot be finite.
4. **The pattern has edges.** For each direction, there must be at least one state in the combination that expects a neighbor in that direction, but not in the opposite direction. Otherwise, no state in the combination should expect any neighbor along either direction.
5. **The states can match with each other along all expected directions.** Each state in a combination should be capable of being potentially matched (i.e., be a neighbor of) to the other states in a combination sufficiently to cover its expected neighborhood. This information is provided by $M(\mathcal{S}_{des})$ and $D(\mathcal{S}_{des})$. The reasoning is best explained via an example. Consider a swarm of four agents with $\mathcal{S}_{des}$ as in Figure 3.9 and a potential combination $C_i = \{s_1, s_1, s_2, s_3\}$. Using $M(\mathcal{S}_{des})$, we observe pair-wise matches that are possible between the states in C_i. $M(\mathcal{S}_{des})$ tells us that s_1 only matches with s_2 in one direction. In $D(\mathcal{S}_{des})$ we can see that this is direction l_2 from the perspective of s_1, and l_6 from the perspective of s_2. However, C_i features *two* instances of s_1 and *only one* instance of s_2. This means that one instance of s_1 can never be satisfied — the combination cannot exist. This can be checked for all states.

Spanning trees analysis Combinations that have the potential to form a pattern are analyzed further. We do this by composing spanning tree graphs. Let $T_i(C_k)$ represent an arbitrary spanning tree generated from a combination C_k. The nodes of T_i are the states in C_k, and the edges of T_i are one of the connections between the states. A representative spanning tree must meet the conditions below.



(a) **(b)** **(c)** **(d)**

Figure 3.10 Examples of: **(a)** an invalid spanning tree, because the graph is not connected; **(b)** an impossible spanning tree, because one agent is expected to have more neighbors than it can support; **(c)** an impossible spanning tree, because some states end up with unfulfilled neighbor expectations; **(d)** a possible spanning tree.

- It is acyclic.
- It is simple (no duplicate edges).
- The edges must at least meet the match conditions in $M(\mathcal{S}_{des})$, or else we know that the edges are impossible because the two states can never be neighboring states.
- It is connected. If operating by Π_f, then the swarm is connected. This means that it can be represented by a connected spanning tree. If $T_i(C_k)$ is not connected, as in the example in Figure 3.10a, then it is invalid.
- The degree of each state should be less than or equal to the number of neighbors that an agent in that state expects. If the degree of a node in $T_i(C_k)$ is larger than the degree of the state, then $T_i(C_k)$ is invalid and the spanning tree is discarded, as for the example in Figure 3.10b.
- The spatial arrangement must be feasible. All other conditions above depend on the properties of the spanning tree graph and are not (directly) dependent on the spatial arrangement of the states. In this last condition, we analyze the spatial arrangement of the graph to see if all neighboring states match without lose ends (i.e., "unfulfilled neighbors"), or loops where two states are eventually expected to occupy the same positions. For instance, Figure 3.10c shows a spanning tree that fails this test. $D(\mathcal{S}_{des})$ can be used to quickly generate the full pattern.

If a possible spanning tree is found, as in Figure 3.10d, then a possible pattern has been identified and it can be checked to determine whether it is equivalent to P_{des} or whether it is spurious. A variety of methods can be used to do so automatically (Loncaric, 1998).

3.5. EVALUATION OF THE IDEALIZED SYSTEM

We begin by evaluating the performance of the idealized swarm as described in Section 3.4.1. This allows us to investigate more fundamental properties and gain initial high level insights. We also explore how further tuning of Π_f could affect the statistical performance of the swarm in forming a desired pattern more quickly. The latter leads to insights on possible optimization strategies, which we discuss further in Section 3.7.3.

The simulation environment used in this section replicates the idealized framework from Section 3.4.1. We simulated idealized agents on a discrete 2D grid world operating in discrete time. At each time step, one random agent with state $s \in \mathcal{S}_{active}$ executes an

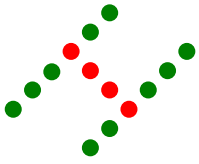


Figure 3.11 Example of a spurious pattern with the slanted line, forming a slanted "H" due to the combination of desired states (in green) and static states (depicted in red), which can form a bridge between two lines.

action based on Π_f. All tests begin by initializing the agents in a random pattern P_0 and end when all agents are in a state $s \in \mathcal{S}_{static}$.

We evaluated the formation of the patterns from Figure 3.2. All patterns were successfully achieved, with no collisions or separation ever occurring. This also happened for the slanted line, which did not pass the proof and was additionally also prone to spurious patterns, as the one that is for instance depicted in Figure 3.11. Generally, as the complexity of the pattern and size of the swarm grew, the cumulative actions taken by the swarm to go from P_0 to P_{des} also grew significantly. The swarm is successfully safe and forms the desired patterns, even though (as expected due to the low cognition of the robots) it can take a significant amount of steps before the swarm self-organizes into the pattern. This can be appreciated in the histograms of the results shown in Figure 3.12, split in two graphs to address the difference in scale. Note that the line with 50 robots performed better than the T with only 12 robots. When we also analyze the mean number of actions per agent, we see that the histograms of the line with 50 robots and the triangle with nine robots are comparable. This implies that there is a deeper correlation with shape complexity that should be explored further.

Motivated by the increasingly low performance of larger and/or more complex patterns, we explored certain alterations of the behavior in order to investigate whether it was possible to achieve the pattern faster than in the baseline tests above. We tested this for the triangles with four and nine robots and the hexagon, for which the expected number of actions were fewer and the differences could be better investigated. We explored the following alterations:

- *Alteration 1 (ALT1)*: same as baseline; however, when an agent moves at time step k, the same agent will not move at time step $k+1$ (unless it is the only active agent).
- *Alteration 2 (ALT2)*: same as ALT1; additionally, all states with more than five neighbors are now not mapped to any actions.
- *Alteration 3 (ALT3)*: same as ALT2; additionally, all actions must ensure that all agents in the neighborhood, following the action, have at least one neighbor at north, south, east, or west, else the state-action pair is discarded from Π_f. For the triangle with nine agents, we made one exception to this, and it is the state $s = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$ (following the layout in Figure 3.9) for which otherwise a spurious pattern could also form.
- *Alteration 4 (ALT4)*: same as ALT3; additionally, all states with more than four neighbors are now not mapped to any actions.

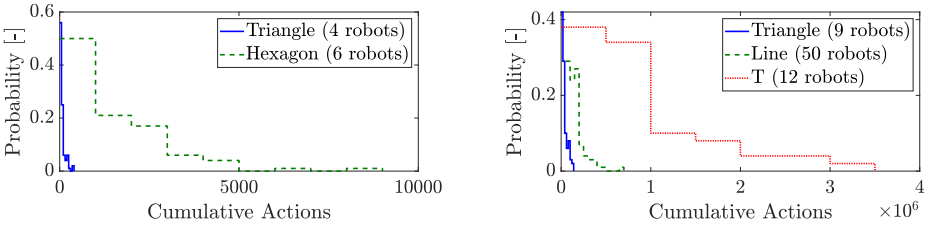
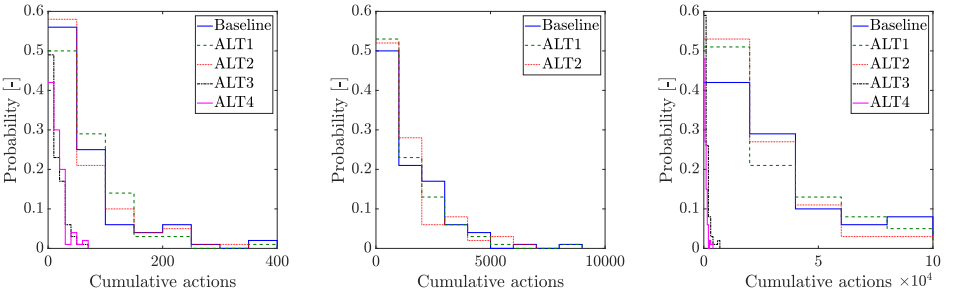


Figure 3.12 Normalized histograms of the actions taken before the pattern is achieved for the different patterns tested. The plots are separated in two for scale differences. The bin width was adjusted to show the overall trend for each pattern. The cross with 20 robots is excluded for scale reasons, with the lowest amount of steps measured being $\approx 2.5 \cdot 10^5$.



(a) Triangle with four robots **(b)** Hexagon **(c)** Triangle with nine robots

Figure 3.13 Normalized histograms of actions to completion by different alternations of the state-action spaces for three patterns. The bin width was adjusted to show the overall trend for each case.

ALT3 and ALT4 stem from the intuition to let agents "cut corners" and have fewer functionally active states. In turn, however, ALT3 and ALT4 do no meet the conditions of Lemma 3.4.2 for the hexagon of six robots, and do not meet Condition 3 of Theorem 3.4.1 for all patterns tested with it. This is because some states in $\mathcal{S}_{simplicial}$ lost their property of enabling the agent to potentially move freely around its neighborhood. Functionally, they behaved like states in the set $\mathcal{S}_{\neg simplicial}$, and a few even like states in $\mathcal{S}_{blocked}$. Normalized distributions for the number of steps to completion using ALT1-ALT4 are shown in Figure 3.13a, Figure 3.13b, Figure 3.13c for the triangle with four agents, the hexagon, and the triangle with nine agents, respectively. For ALT1 and ALT2 the final pattern is achieved in all cases. As the size of the pattern grows, ALT1 and ALT2 are seen to provide a marginally better performance, but not significantly so. The real improvement is seen with ALT3 and ALT4. By blocking more local states and cutting corners, the swarm is less chaotic and forms the pattern orders of magnitude faster. As expected through Lemma 3.4.2, however, ALT3 and ALT4 prevented the hexagon from forming. Instead, failing condition 3 of Theorem 3.4.1 did not stop ALT3 and ALT4 from achieving the triangles with four and nine robots. This could imply that Theorem 3.4.1, by nature of featuring local conditions, becomes more restrictive than necessary for some global patterns. This was also the case for the line with 50 agents, because the line also does not

meet the condition. Alternatively, it could also be possible that the robots were simply "lucky" to not encounter deadlock situations during any of our simulations.

3.6. IMPLEMENTING THE BEHAVIOR ON ROBOTS

Until now, we have dealt with idealized agents on a 2D grid. In this section, we describe how the behavior can be brought to real robots operating in continuous time and space and using local clocks. We test the behavior in two stages of fidelity: 1) accelerated particles, and 2) simulated MAV flights, showing that the behavior is also robust to noise.

3.6.1. ROBOT BEHAVIOR

The robots can sense omni-directionally all their neighbors within a range ρ_{sensor} and can determine whether their neighbors are computing an action (Assumption A3). A robot $\mathcal{R}_i$ determines its discrete local state $s_i \in \mathcal{S}$ following the bearing based discretization in Figure 3.14a.

All robots act following the FSM in Figure 3.14b. This FSM locally enforces that only one robot in the neighborhood can move at any time. Following this FSM, a robot will initiate and pursue an action from Π_f if and only if no other robot in a neighborhood is sensed to be already doing so, which locally recreates the conditions of the idealized system. Therefore, even though multiple robots around the swarm can take actions at the same time, this does not occur at the local level. If two robots who are not neighbors become neighbors while both are executing an action, the actions will interrupt, ensuring safety. Using $t_{adj} > 0$ and $t_{wait} > 0$ the robots have allocated time to execute attraction, repulsion, and alignment behaviors. As these alignments maneuvers are minimal, they are not sensed by neighbors as actions and therefore create natural time windows whereby robots take turns in taking actions.

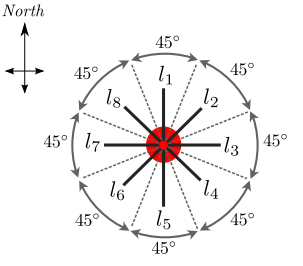
We have designed a unified attraction, repulsion, and alignment behavior that allows the robots to naturally arrange in a grid structure whenever not executing an action. Consider a robot $\mathcal{R}_i$ and its neighbor $\mathcal{R}_j$. The robots are controlled according to a northeast (NE) frame of reference. The commanded velocity of $\mathcal{R}_i$ along north (and, equivalently, east) is given by:

$$v_{N_{cmd_{ij}}} = \underbrace{(v_{r_{ij}} + v_{b_{ij}})\cos(\beta_{ij})}_{\text{Attraction and repulsion}} - \underbrace{v_{b_{ij}}\cos(2\beta_{des} - \beta_{ij})}_{\text{Alignment}}. \tag{3.3}$$

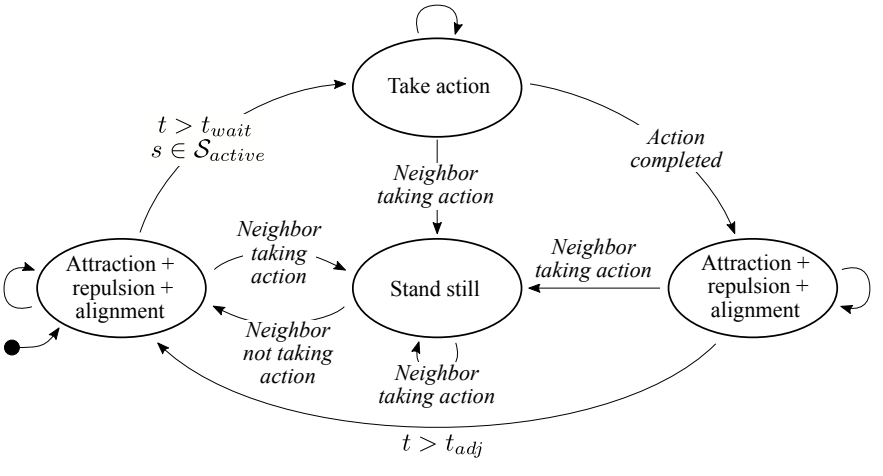
The first term handles attraction and repulsion. The second term aligns $\mathcal{R}_i$ at a bearing β_{des} to $\mathcal{R}_j$. β_{ij} is the bearing of $\mathcal{R}_i$ to $\mathcal{R}_j$ with respect to north. $v_{b_{ij}}$ is the desired radial velocity. The attraction-repulsion velocity $v_{r_{ij}}$ is:

$$v_{r_{ij}} = -k_r \frac{1}{|\rho_{ij}|} + \frac{1}{1 + e^{-k_a(|\rho_{ij}| - \rho_s)}}, \tag{3.4}$$

where $k_r \geq 0$ is the repulsion gain, $k_a \geq 0$ is the attraction gain, ρ_{ij} is the range between $\mathcal{R}_i$ and $\mathcal{R}_j$, ρ_s is a shift in the attraction term used to tune the equilibrium distance to ρ_{des}. Equation 3.4 has Lyapunov stability (Gazi and Passino, 2002). For given ρ_{des}, k_r, and k_a, one can extract ρ_s such that $v_{r_{ij}} = 0$. The two robots are in equilibrium



(a) Rounding method used by the robots to discretize their local state



(b) FSM of robot behavior

Figure 3.14 State discretization and FSM of robot behavior

($v_{N_{cmd_{ij}}} = v_{E_{cmd_{ij}}} = v_{N_{cmd_{ji}}} = v_{E_{cmd_{ji}}} = 0$) when $\beta_{ij} = \beta_{des}$, $\beta_{ji} = \beta_{des} \pm \pi$, and $v_{r_{ij}} = v_{r_{ji}} = 0$. Note that Equation 3.3 is reciprocal. For each β_{des}, there exists a corresponding equilibrium point at $\beta_{des} \pm \pi$. This is due to the identities $\sin(\beta+\pi) = -\sin(\beta)$ and $\cos(\beta+\pi) = -\cos(\beta)$. Furthermore, multiple desired bearings β_{des} can be defined, such that each robot can gravitate to the one that is closest to its current β. We provided the robots with $\beta_{des} = \{0, \pi/4, \pi/2, 3\pi/4\}$, making them adjust at all the eight bearings to each other that match the idealized grid. For $\beta_{des} = \pi/4$ and $\beta_{des} = 3\pi/4$, then we define $\rho_{des} = \sqrt{2}\ m$ instead of $\rho_{des} = 1\ m$. For a robot $\mathcal{R}_i$ which senses m neighbors, the complete command along north is $v_{N_{cmd_i}} = \sum_{j=1}^{m} v_{N_{cmd_{ij}}}$, and the equivalent for east. This is unless the closest neighbor is at a distance $\rho < \rho_{safe}$, in which case only the closest neighbor is considered.

3.6.2. SIMULATION TESTS WITH ACCELERATED PARTICLES

We begin by testing the behavior from Section 3.6.1 on accelerated particles in an unbounded 2D space. This allows us to quickly test the performance of large swarms while remaining independent of the dynamics of any particular robot.

The simulations in these sections have been executed on an in-house simulator called *Swarmulator*. Swarmulator is a light-weight swarm simulator designed to quickly develop and prototype spatial swarm behavior.[8] Swarmulator's simplicity and emphasis on quick prototyping is the reason that it was chosen for this phase. Each robot is simulated as a point in an unbounded 2D space by a detached C++ thread, thus simulating a random asynchronicity and minimizing the simulation artifact that would otherwise stem from simulating the swarm in a loop. Swarmulator is described in more detail in Chapter A. To further reduce simulation artifacts, the robots initiate the behavior with a random local time $0 < t < t_{wait}$. Other detached threads handle animation and logging, allowing automatic checks of global properties. In the simulations: $\rho_{sensor} = 1.6\ m$, $\rho_{des} = 1\ m$, $\rho_{safe} = 0.5\ m$, $t_{adj} = 1.8\ s$, $t_{wait} = 3.6\ s$, $k_r = 1$, $k_a = 5$, $v_{action} = 1\ m/s$, $v_b = 10\ m/s$. The state-action map Π_f was as in ALT4 from Section 3.5.

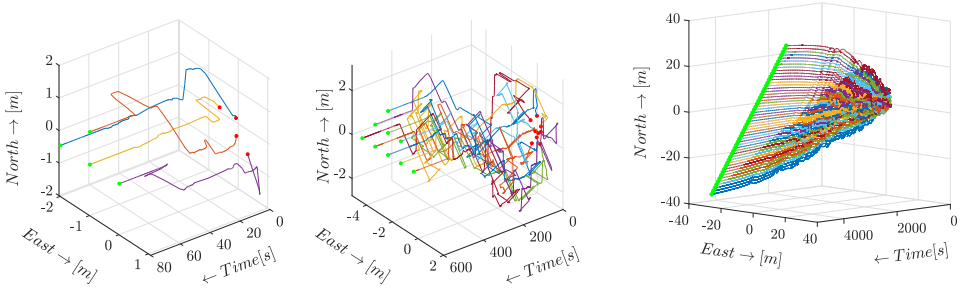
Results The results for the triangles with four and nine agents from Section 3.5, using the controller from ALT4, were validated in this continuous setting. Figure 3.15a and Figure 3.15b shows sample trajectories over time.[9] We can see that the agents reshuffle until the desired pattern is achieved. All simulations were repeated 50 times. The triangle with four agents was achieved successfully in 50 out of 50 trials, with generally fast convergence times (within 100 seconds of simulated time). The triangle with nine agents was achieved successfully in 49 out of 50 trials. Only one trial experienced a separation. This happened as two non-neighboring agents chose to perform an action at approximately the same time, came into each other's view, but the alignment maneuvers that followed were such that two agents (who were the link between two parts of the swarm) momentarily moved further than 1.6 m apart, which was the limit of the sensor. Although we could be more lenient and accept the fact that the swarm quickly reconnects, as done by Winfield and Nembrini (2012), the issue is noted and should be tackled in future work to further guarantee safety even in a continuous setting. Nevertheless, this was *the only* "unsafe" event that took place out of thousands of maneuvers executed over all 50 trials. We also successfully simulated the behavior of the swarm with large groups tasked with making a line with 50 robots, for which a sample trajectory is shown in Figure 3.15c. Here, it is interesting to see how the line slowly forms as robots all over the swarm begin to align themselves as required.

3.6.3. MICRO AIR VEHICLE SIMULATIONS

Having developed and tested a behavior that can be used in a continuous domain, we now explore whether it can be used when robots have more realistic dynamics and reaction times. This section provides a proof of concept and shows how the selected al-

[8] The source code can be found at `https://github.com/coppolam/swarmulator/tree/SI_PatternFormation`

[9] Videos of other sample runs are available at `https://www.youtube.com/playlist?list=PL_KSX9GOn2P8BYpwA-_WfXmtb7CRnVhC3`



(a) Triangle with four robots **(b)** Triangle with nine robots **(c)** Line with 50 robots

Figure 3.15 Simulated trajectories to the desired patterns

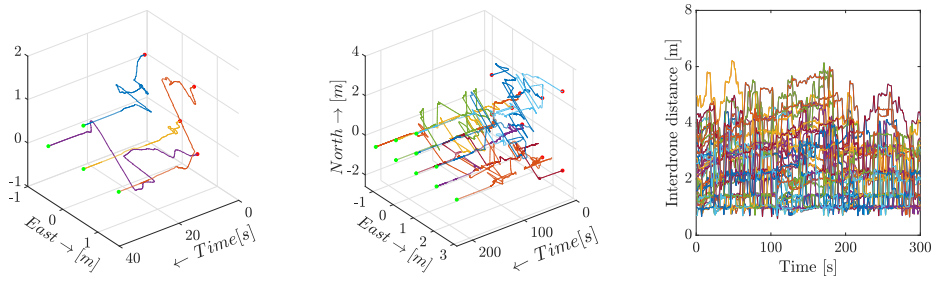
gorithm can work on a team of real MAVs with the relevant dynamic constraints and perturbations.

The simulations were executed using using Robotics Operating System (ROS) (Quigley et al., 2009) and the Gazebo physics engine (Koenig and Howard, 2004). The hector-quadrotor model provided by Meyer et al. (2012) simulates the dynamics of a quadrotor MAV. Each MAV is simulated on a separate module and runs independently, with the higher level controller running at 10 Hz. The same simulation environment was used in both Coppola et al. (2018) and McGuire et al. (2017a) with successful replication of the controllers on real-world MAVs, and it was chosen for this reason. We assumed that the MAVs could measure the position of their nearest neighbors up to 1.6 m, and that they could then sense whenever a neighbor was moving at more than 0.1 m/s, which they would interpret as the neighbor taking an action. All other control parameters were kept the same as in the Swarmulator trials, with some minimal tuning to suit the new dynamics (namely: $v_b = 2\ m/s$, $t_{adj} = 1.5\ s$, $t_{wait} = 3\ s$).

Results The results of Section 3.6 were successfully replicated using this set-up. We show two sample trajectories of flights in Figure 3.16a and Figure 3.16b.[10] As for the accelerated particles, the triangle with four MAVs was generally reached within only 100 s of flight, and in 48 out of 50 cases it was completed before the final simulation time of 500 s. As expected based on our idealized simulation, the flight time was not enough for such a high success rate also with the significantly more complex triangle with nine MAVs. 20 out of 50 cases finished the triangle within the maximum simulation time of 5000 s for these simulations. Nevertheless, the MAVs never collided with each other and the swarm never separated in any of the trials, showing that the idealized rules translate well to realistic dynamics.

Simulation results with sensor noise Additionally, we explored the performance of the behavior under the influence of noise in the relative position readings of neighbors by

[10] Videos are available at `https://www.youtube.com/playlist?list=PL_KSX9GOn2P8BYpwA-_WfXmtb7CRnVhC3`



(a) Triangle with four robots **(b)** Triangle with nine robots **(c)** Simulated flight of nine MAVs with sensor noise

Figure 3.16 Exemplary results of ROS simulations

applying Guassian noise with standard deviation of 0.1 m and 0.1 rad for relative range and bearing, respectively. The only change was that the MAVs could see up to 2 m instead of 1.6 m in order to restrict false negatives. The results were robust to the noise. Consider, for instance, the 300 s flight with nine MAVs shown in Figure 3.16c. It can be seen that the swarm distances are kept, while the swarm still reshuffles, and no collisions occur. The discretization imposed by the state-action map is such that the behavior is robust to sensor noise. The behavior is robust even when the same set-up from the noiseless case is used, without any filtering of the Gaussian noise (e.g., using a Kalman filter or a low pass filter), which would otherwise drastically improve the results further.

3.7. DISCUSSION

3.7.1. INTUITIVE AND VERIFIABLE DESIGN OF COMPLEX BEHAVIORS

The approach presented in this chapter allows a swarm designer to intuitively define local behavior of cognitively limited robots faced with a global task. It is merely necessary to divide the global task into the locally observable constituents and incorporate this into the state-action map of the robots. Doing so provides the robots with a behavior that forms the pattern, even though the robots are incapable of locally knowing when/if this ever occurs.

Having such an intuitive method allows us to form patterns that (for systems with similarly limited capabilities) had previously not been achieved using an explicit design. We showed six patterns as examples, but the limits of the algorithm do not stop there. Izzo et al. (2014) and Scheper and de Croon (2016), for instance, both proposed neural networks to tackle the formation of an asymmetric triangle, whereby the difficulty was that three non-communicating homogeneous robots could not resolve the asymmetry. However, using the approach presented in this chapter, it becomes easy to form any asymmetric triangle. The desired states to develop Π_f are readily extracted, as in Figure 3.17a and the dimensions of the triangle can be tuned by adjusting the attraction and repulsion forces along north and east. The asymmetric triangle is then obtained as exemplified in Figure 3.17b and Figure 3.17c.

In this work we focused on pattern formation, but we postulate that this framework

(a) Desired pattern and related desired states

(b) Asymmetric triangle in ROS simulation

(c) Flight path

Figure 3.17 Development of an asymmetric triangle and test on ROS

could also be extended to other global tasks such as organized navigation or task allocation. In future work, we aim to investigate how the framework can be generalized.

3.7.2. GENERATING ARBITRARY PATTERNS WITHOUT LIVELOCKS AND DEADLOCKS

Section 3.4.2 showed how Π_f can be readily computed for any pattern. However, because of how limited the robots are, it is not necessary that the swarm is able to reach this pattern from any initial condition while being free of livelocks or deadlocks. Deadlocks and livelocks, however, stem from the limited knowledge that is available to the robots. If the robots could see further, or remember past states, or communicate, they would be able to form more complex patterns and would be able to move more freely. Theoretically then, any pattern can be formed provided that the state space is sufficiently detailed to uniquely represent the desired goal and allow enough freedom to the robots. In line with the goals of generalizing the scheme that was presented here, we also wish to determine how providing the agents with some extra capabilities can allow more complex goals to emerge. This is while resting on the knowledge that the swarm can also operate when these extra capabilities malfunction. Furthermore, the proof conditions have been shown to be more restrictive than it can turn out to be in the real swarm. The advantage of using local properties are that we do not need to analyze the global states of the swarm, yet this comes at the cost of possibly being more strict than required from the global perspective. At this moment, however, we have seen that patterns that do not respect some of the conditions still form in our simulations, such as the line pattern. Indeed, it may be that the subset of global states that represent a deadlock or livelock is very small compared to the total state space, making such failures possible, yet extremely unlikely. More focused investigations should be conducted in order to understand when it is possible to be more lenient on some conditions while still ensuring that livelocks and deadlocks do not arise.

3.7.3. TIME FOR SELF-ORGANIZATION

In Section 3.5, generally speaking, it was found that as the size of the swarm and the intricacy of the pattern grow, the pattern could form only after a possibly unrealistic number of actions by the robots. This property was expected in light of all the limitations of the robots, as it becomes increasingly unlikely that the agents' random actions will lead to the desired global pattern, given that essentially the swarm randomly transitions between possible global patterns.[11] However, there are two important things to note:

1. In real robot swarms, several actions will be taking place at the same time, so the time to completion will be faster than expected. For instance, in Figure 3.15c the line is seen to slowly form across the entire swarm, whereas this is not the case for the idealized system.
2. Our investigations in Section 3.5 showed that it is possible to improve performance by several orders of magnitude by further altering the local state-action map.

The latter leads to questions about how to best alter a local state-action map. The alterations in this work were done manually, using intuition, for exploratory purposes. The problem could be solved more optimally using machine learning methodologies such as reinforcement learning or evolutionary robotics. The objective would be to alter Π_f such that, statistically, the time for the robots to self-organize into a desired pattern is minimized. Here, the local proofs would allow us to verify that the alterations are such that the system is still guaranteed, at all times, to always eventually reach the pattern. Policy optimization will be explored in Chapter 4.

3.7.4. TOWARD REAL-WORLD IMPLEMENTATIONS AND APPLICATIONS

The simulations using ROS in Section 3.6.3 provide a large degree of confidence in the possibility to implement the system on real MAVs (or other robots). Provided that the necessary sensory information is available, then they are able to follow the behavior even when behaving by their own internal clock and in the presence of sensor noise, and this is without the aid of any additional filtering. We then find that the local behavior can also be used simply to guarantee collision avoidance and swarm coherence in spite of all limitations of the robots. This has several applications of its own. For instance, it can be used to preemptively guarantee that a robotic sensor network never separates in multiple groups.

3.7.5. SCALABILITY OF PROOF PROCEDURE

Our proof procedure focused as much as possible on the local level, making it largely independent of the number of agents in the swarm, and thus able to mitigate state explosion issues. Most notably, we are able to prove, only by a local level analysis, that livelocks will not exist when starting from any initial pattern. A key element of this proof was an analysis of the simplicial states and the intuition that they could help the swarm to resolve livelocks. Nevertheless, the complete proof still requires us to verify that deadlock patterns will not occur, and this part is still done using an ultimately global analysis. We have shown how to mitigate the computational explosion by looking at a limited subset of state combinations and using a procedure to quickly sort through the possibilities,

[11] In popular adage, one might say that there is no such thing as a free lunch.

yet the issue is not yet fully eliminated. In future research, there should be efforts to further mitigate its effects for finite patterns. Here, we expect that the match matrices introduced will be a fundamental tool to analyze local connections between the robots.

For now, three solution directions have been identified in order to mitigate the computational explosion. The first is to focus on the agents at the border of the structure, assuming that all other agents will be enclosed by these agents. The second avenue is to use repeating sub-patterns. The local states could be made such that the agents can arrange into infinitely repeating patterns (e.g., infinitely connecting hexagons) and create a large complex structure without defining or checking the larger structure in full. This we actually already did, in part, for the line pattern. The third solution, perhaps most trivial, is to allow robots that have been blocked for a long time to temporarily perform partially unsafe maneuvers, which might set the system free from deadlocks (but may come at other costs).

3.8. CHAPTER CONCLUSIONS

In this chapter we introduced a method to design the local behavior of robots in a swarm so as to form desired global patterns in spite of extremely limited cognitive abilities. Because the robots only know the relative location of their closest neighbors and have no memory of the past, they cannot take "purposeful" actions. Therefore, a mechanism has been designed that makes the global pattern emerge from the local, stochastic behaviors of the agents. Approaching the problem from top-down, the method simply requires one to identify the local states that build the desired global pattern in order to design the behavior. Then, to close the loop, we presented a proof procedure to verify whether the desired pattern will always eventually emerge from the random interaction of the agents. An important insight from these proofs is the crucial roles that simplicial states play in helping the swarm to avoid livelocks and minimizing the possibility of deadlocks. It is important to note here, however, that should we find that livelocks and deadlocks are possible, then this tells us that the robots have an insufficient sensory knowledge for the desired global goal to always eventually happen, which is equally valuable information when designing a robotic swarm. Despite developing the behavior for idealized agents on a grid world, we have presented very promising results that show that it can potentially be successfully reproduced by robots operating in continuous time and space, with local clocks, even in the presence of noise. In follow-up work, Ripoll Sanchez (2019) also explored the possibility of using this algorithm on swarms of satellites. In this context, it could be especially useful as a fail-safe mechanism in case the communication between satellites fails.

The methodology presented here has been used for pattern formation. At its core, however, it is based on the more general idea of synthesizing a global goal into a probabilistic state-action map executed by the robots, and the verification of the global property by ensuring that the swarm features agents with a state that empowers them to help the swarm evolve (i.e., simplicial states). With a modified mapping, we expect this strategy to also be applicable to systems with significantly different state and action spaces. A further challenge that must be solved is to use an optimization procedure to enable larger and more complex patterns to form faster. The next chapter tackles these challenges. We will introduce a model-based optimization algorithm, and we will abstract

our methodology and apply it to two more tasks: consensus and aggregation.

CODE

- The simulation code used in this chapter can be downloaded at `https://github.com/coppolam/swarmulator/tree/SI_PatternFormation`.
- The ROS simulator can be downloaded at `https://github.com/coppolam/ros_mav_swarm_simulator`.
- All data used in this chapter can be downloaded at: `https://surfdrive.surf.nl/files/index.php/s/ZNM2mtKFdWHnRH9`.

4

PageRank centrality as a measure to optimize swarm behaviors

In the prior chapter, we have established a procedure to break down a global goal into local states, and enabling a swarm to achieve this goal. In particular, we focused on the goal of pattern formation. In this chapter, we generalize the method to additional tasks, and introduce a novel method, based on PageRank centrality, that enables us to optimize the behavior of the swarms using the concept of desired states. PageRank is a graph centrality measure that assesses the importance of nodes based on how likely they are to be reached when traversing a graph. We relate this to a random robot in a swarm that transitions through local states by executing local actions, using PageRank to evaluate how likely it is, given a local policy, for a robot in the swarm to visit each local state. This is used to optimize a stochastic policy such that a robot is most likely to reach the local states that are desirable, based on the swarm's global goal. The optimization is performed by an evolutionary algorithm, whereby the fitness function maximizes the PageRank score of these local states. The calculation of the PageRank score only scales with the size of the local state space, and demands much less computation than swarm simulations would. The approach is applied to a consensus task, a pattern formation task (as seen in the previous chapter), and an aggregation task. For each task, when all robots in the swarm execute the evolved policy, the swarm significantly outperforms a swarm that uses the baseline policy. When compared to globally optimized policies, the final performance achieved by the swarm is also shown to be comparable. As this new approach is based on a local model, it natively produces controllers that are flexible and robust to global parameters such as the number of robots in the swarm, the environment, and the initial conditions. Furthermore, as the wall-clock time to evaluate the fitness function does not scale with the size of the swarm, it is possible to optimize for larger swarms at no additional computational expense.

The contents of this chapter have been published in Coppola et al. (2019a).

4.1. BACKGROUND

Machine learning techniques are a powerful approach to develop swarm behaviors. Evolutionary algorithms, for instance, can efficiently explore a solution space and extract viable local behaviors that fulfill a desired global goal (Francesca and Birattari, 2016; Nolfi, 2002). They have been used on numerous architectures, including: neural networks (Duarte et al., 2016; Izzo et al., 2014), state machines (Francesca et al., 2015), behavior trees (Jones et al., 2018; Scheper et al., 2016), and grammar rules (Ferrante et al., 2013). A bottleneck of these algorithms is in the need to evaluate how the whole swarm performs against each controller that they generate. Because of the complexity of swarms and the difficulty in predicting the global outcome, a full simulation of the entire swarm is carried out each time. This is subject to scalability issues as the size of the swarm increases, for example:

1. The computational load required to execute the simulations increases with the size of the swarm.
2. It may take longer for the desired behavior to emerge, requiring a longer simulation time for each evaluation trial, especially in the initial stages of the evolution.
3. The evolved policy may be over-fitted to the global parameters used during the simulation, such as the number of robots, the initial conditions, or the environment.
4. A solution needs to be simulated multiple times in order to reliably assess the expected performance of a given behavior (Trianni et al., 2006). Avoiding re-evaluation may result in poor behaviors being erroneously assigned a higher fitness thanks to one lucky run, which may ultimately result in a performance drop (Di Mario et al., 2015a,b).[1]
5. The evolution may be subject to bootstrap issues (Gomes et al., 2013; Silva et al., 2016).

In order to tackle these scalability problems, we introduce a new approach to the field of swarm robotics: PageRank (Brin and Page, 1998; Page et al., 1999). PageRank is a graph centrality and node ranking algorithm. It was originally developed by Sergey Brin and Larry Page as part of Google™. Its objective was to rank the importance of Web pages based on the hyperlink structure of the World Wide Web. PageRank's philosophy was to model the browsing behavior of a user who surfs the Web by randomly clicking through hyperlinks, and to measure the value of Web pages based on how likely it would be for this user to visit them. In this chapter, we port this idea to the world of swarm robotics. Here, a robot in a swarm becomes analogous to a Web surfer. The robot moves through local states by taking actions, much like a Web surfer navigates through Web pages by clicking hyperlinks. With PageRank centrality, we can then evaluate the relative likelihood with which the robot will end up in the local states. Then, with the knowledge that a desired global goal is more likely to be achieved when the robots are (or pass through) a given set of local states, we can efficiently quantify the global performance of the swarm in achieving its goal. More specifically, we propose a fitness function, based

[1] An alternative to re-evaluation is to vary other parameters. For instance, one could simulate once but for a longer time (Di Mario and Martinoli, 2014), although this is applicable to continuing task and not to tasks with a definite global goal.

on PageRank, that can assess the *global* performance of a swarm while only evaluating the *local* model of a single robot in the swarm. This micro-macro link frees us from the need to simulate the swarm. Due to the local nature of this approach, the evaluation is independent from global parameters such as the size of the swarm, the initial condition, the environment, or lower-level controllers. The introduction of this method is the main contribution of this chapter. We will showcase its potential by applying it to optimize the local behavior for three different swarming tasks: 1) consensus agreement, 2) pattern formation, and 3) aggregation.

We begin by discussing related work in Section 4.2. Here, we place our contribution within the context of other solutions found in the literature which also had the aim of tackling scalability issues. We further compare our use of a PageRank-based microscopic model to other swarm modeling approaches. In Section 4.3, we then detail how PageRank works and explain how it can be applied to model, assess, and optimize the performance of a robotic swarm. The approach is then directly applied to optimize the behavior of three swarming tasks, as follows.

- *Consensus agreement (Section 4.4).* In this task we optimize the behavior of a swarm that must achieve consensus between multiple options. Each robot can sense the opinion of its neighbors and, based on a stochastic policy, decide whether it should change its opinion (and, if so, what to change its opinion to). Using PageRank, we optimize the stochastic policy so as to help the swarm achieve consensus as quickly as possible. The policy is optimized independently of the size of the swarm or its spatial configuration. We further optimize a more limited variant of this task whereby robots cannot sense the opinion of their neighbors, but can only sense whether they are in agreement with their neighborhood or not.
- *Pattern formation (Section 4.5).* For this task, we optimize the performance of a swarm of robots with the global goal of arranging into a desired spatial configuration. The robots in the swarm have very limited knowledge of their surroundings. This section directly extends the work from Chapter 3, published in Coppola et al. (2019b), as well as the work published in Coppola and de Croon (2018). In the latter, we attempted to optimize the behavior of a swarm in a pattern formation task, yet quickly encountered scalability problems for larger swarms/patterns. The scalability problems were a result of the fact that the swarm had to be simulated in order to assess the efficacy of a controller. This was infeasible for larger swarms, especially in early generations where performance is poor. Using the PageRank algorithm we can now tackle these scalability issues, as it is no longer needed to simulate the swarm in order to assess the fitness of a behavior.
- *Aggregation (Section 4.6).* In this task, we study a swarm of robots in a closed arena which should aggregate in groups of three or more. In comparison with the other two tasks, this optimization tunes a higher-level policy featuring two sub-behaviors (*random walk* and *stop*) with a probability that is dependent on the number of neighbors that a robot in the swarm can sense. The final achieved policy allows the swarm as a whole to aggregate successfully.

In Section 4.7 we discuss our general findings, including an analysis of the strengths and the current limitations of using PageRank centrality as a behavior optimization tool for swarm robotics. Section 4.8 provides concluding remarks.

4.2. RELATED WORKS AND RESEARCH CONTEXT

In state of the art, the problems of scalability discussed in Section 4.1 have mostly been tackled in two ways. First, there are methods that try to deal with the broad solution space that comes as the number of robots increases. For example, Gomes et al. (2012) used novelty search to encourage a broader exploration of the solution space. The second way is to use global insights to aid the evolutionary process. For example, Duarte et al. (2016) partitioned complex swarm behavior into simpler sub-behaviors. Hüttenrauch et al. (2017), with a focus on deep reinforcement learning, used global information to guide the learning process toward a solution. Alternatively, Trianni et al. (2006) and Ericksen et al. (2017) explored whether evolved behaviors for smaller swarms could generalize to larger swarms. In all cases, the need to simulate the swarm remains a bottleneck for both evaluating the behavior and generalizing the behavior beyond the parameters used in simulation. Here, we offer a different solution which discards simulation and exploits a micro-macro link based on evaluating the relative PageRank score between local states using only a local model of a single robot in the swarm. It extracts performance parameters without simulation or propagation from an initial condition and only relies on an analysis of the model for a given policy.

This approach differs from other swarm modeling and evaluation solutions found in the literature, such as the multi-level modeling framework introduced by Lerman et al. (2001), Martinoli and Easton (2003), and Martinoli et al. (2004). There, the idea is to model the evolution of a swarm via probabilistic finite state machines, propagating the flow between states over time. At the microscopic level, one probabilistic finite state machine is propagated from an initial condition for each robot in the swarm. At the macroscopic level, which can be readily abstracted from the microscopic, the finite state machine describes the mean transition of robots between states. The macroscopic model probabilistically describes, at the global level, how many robots are in each state at a given point in time. This can also be expressed in terms of rate equations. For each level, the relevant transition rates between states are extrapolated from an analysis of the policy, as well as from geometric reasoning (e.g., based on the size of the arena and the expected robot density that ensues) (Martinoli et al., 2004) or from empirical data (Berman et al., 2007). To predict the global evolution of the swarm, the models are propagated in time from an initial condition, essentially simulating the swarm at an abstract level. For certain tasks, these models have been shown to be highly effective in predicting the general behavior of a swarm (Lerman et al., 2005). However, their accuracy and applicability are limited by the validity of the global assumptions which define the transition rates. For instance, to estimate the probability of interaction between any two robots, one assumption is that the system is "well mixed", meaning that the robots are all equally distributed within an area at all times and always have the same probability of encountering any neighbor. For reasons such as this, their use has been largely limited to the evaluation of tasks with few states and by swarms in bounded arenas. Examples where these models have been used are the modeling of collective decision making (Hamann et al., 2014; Reina et al., 2015), area exploration/foraging (Campo and Dorigo,

2007; Correll and Martinoli, 2006), or keeping an aggregate (Winfield et al., 2008).[2] For other tasks where the level of detail required is higher and/or where the global goal is achieved by a combination of several states, rather than all robots being in one state, this approach does not provide a sufficient level of detail. One such example is pattern formation, whereby the global goal is to achieve a particular spatial arrangement which cannot be described by such models.

Another class of macroscopic models, focused on spatial movements, models a swarm by using a diffusion model based on the Fokker-Planck equation (Hamann and Wörn, 2008). This approach macroscopically models the general motion of a swarm under the assumption of Brownian motion. The Fokker-Planck equation provides an estimate for the density of robots in the environment as a result of the robots' motion. Prorok et al. (2011) explored the use of rate equations together with a diffusion model. This made it possible to study swarms in more complex environments where the swarm may not be well mixed, which otherwise causes drift errors (Correll and Martinoli, 2006). However, there were still limitations on how to approximate more complex behaviors by the robots, which, by the nature of the assumption of Brownian motion, were limited to random walks.

The use of macroscopic models to optimize the behavior of robots in a swarm was explored by Berman et al. (2007, 2009, 2011) using models similar to those of the rate equations by Martinoli and Easton (2003). In these models, the fraction of robots in each state is modeled together with the rates at which the robots will transition between these states. It is then possible to optimize the transition rates such that the macroscopic model, on average, shows that the swarm settles in certain states of interest. In (Berman et al., 2009) this is a done for the problem of task allocation, where it is also proven that all robots in the swarm will settle to a desired global equilibrium. However, as for rate equations, this approach is limited by the global level assumptions that are being taken in order to describe the mean of the swarm (Berman et al., 2011). Moreover, the outcome from such approaches can also be dependent on the initial condition of the swarm (Hsieh et al., 2008).

Macroscopic approaches thus make it possible to approximate how a swarm can evolve in time. They purposely refrain from incorporating the detailed experience of a single robot and rather approximate the mean evolution of the entire swarm. This makes them effective prediction tools, but they are limited in their ability to describe interactions to a higher-level of detail. In contrast to multi-level models, the PageRank framework captures state transitions probabilistically, rather than temporally. We can apply this to analyze the impact of the actions by a single robot in a dynamic environment. This makes it able to tackle more specific tasks such as consensus agreement or pattern formation, with results that are found to be scalable, flexible, and robust to initial conditions or the number of robots.

[2] In the example by Winfield et al. (2008), the swarm operated in an unbounded arena. Instead, it was assumed that neighbors of each robot would be equally dispersed within the robot's sensing and communication range. This was shown to be reliable up to a certain extent, in part thanks to the behavior that was implemented.

4.3. PAGERANK CENTRALITY AS A MICRO-MACRO LINK

Centrality measures assess the relative importance of nodes in a graph based on the graph's topology. Several of these measures exist, which capture centrality from different perspectives. For example, *degree* centrality computes the importance of nodes based on the number of edges connected to them. Alternatively, *closeness* centrality measures the average shortest path between a node and all other nodes.[3] This chapter deals with *PageRank* centrality. This is a graph centrality measure for directed graphs that measures the importance of each node recursively. This means that the PageRank centrality of a node is a function of the centrality of the nodes pointing to it, the centrality of the nodes pointing to those nodes, and so forth. This recursiveness indirectly accounts for the topology of the entire network, and it models how likely the node is to be reached when traversing the graph.

In this section, we detail how the PageRank centrality algorithm works (Section 4.3.1) and then explain how it can be used to microscopically model the possible state transitions of a robot in a swarm (Section 4.3.2). In Section 4.3.3, we then introduce a PageRank-based fitness function. This fitness function assesses, at a local level, a swarm's ability to achieve a desired global goal. It will be used to optimize the behavior for all tasks treated in this chapter.

4.3.1. A REVIEW OF PAGERANK CENTRALITY

Consider an arbitrary graph $G = (V, E)$ with nodes V and edges E. Let $u \in V$ be an arbitrary node in the graph. Following Page et al. (1999), a simplified expression for the PageRank $R(u)$ of a node $u \in V$ can be expressed as

$$R(u) = \sum_{v \in B_u} \frac{R(v)}{N_v}, \tag{4.1}$$

where: B_u is the set of all nodes pointing to u, N_v is the number of outgoing edges of node v, and $R(v)$ is the PageRank of node v. Equation 4.1 serves to show the basic concept behind PageRank: the PageRank of a node is a function of the PageRank of the nodes pointing to it. This means that being pointed to by a more important node will provide a node with a higher PageRank. In the case of the World Wide Web, for instance, this reflects how being linked to by a popular Web page (whereby its popularity is also established in the same way) will then be evaluated as being more valuable than being linked to by a niche Web page.

PageRank can be calculated simultaneously for all nodes using an iterative procedure. Here, we briefly review its key elements.[4] Let $\mathbf{R}$ be a vector that holds the PageRank of all nodes in V.

$$\mathbf{R}_{k+1}^{\intercal} = \mathbf{R}_{k}^{\intercal}\mathbf{G} \tag{4.2}$$

$\mathbf{R}$ is obtained once Equation 4.2 converges such that $|\mathbf{R}_{k+1}| - |\mathbf{R}_k| \leq \varepsilon$, where k is the iteration step and ε is a threshold (in this work we used $\varepsilon = 10^{-8}$ and $\mathbf{R}_0$ was a unit vector

[3] The interested reader is referred to the work of Fornito et al. (2016), where a summary and comparison of several centrality measures are provided.

[4] For more details, including sample code and even some humorous anecdotes, we refer the reader to the book *"Google's PageRank and Beyond: The Science of Search Engine Rankings"* by Langville and Meyer (2006).

with uniform entries). Equation 4.2 can be shown to converge quickly provided that the matrix **G** is stochastic and primitive (Langville and Meyer, 2006).[5] **G** is known as the *Google matrix*. In its full form, it is defined as

$$\mathbf{G} = \alpha(\mathbf{H} + \mathbf{D}) + (1 - \alpha)\mathbf{E}, \tag{4.3}$$

where:

- **H** is the normalized adjacency matrix of graph G. **H** is a sub-stochastic transition probability matrix that describes the probability of transitioning between the nodes of graph G. For the example of the World Wide Web, **H** holds the hyperlink structure of the Web and it models how a Web surfer can navigate the Web by using hyperlinks.
- **D** is the *dangling node matrix*. The dangling nodes of graph G are nodes with no outgoing edges (e.g., no hyperlinks), which result in empty rows in **H**. For the World Wide Web, if a user reaches a Web page with no hyperlinks, then the user will resort to writing the name of a Web page in the address bar. **D** describes these transitions and the probabilities thereof. The combined matrix $\mathbf{S} = \mathbf{H} + \mathbf{D}$ is a stochastic matrix.
- **E** is known as the *teleportation matrix*. This is an additional element that describes random transitions between nodes that are not captured by the topology of the graph G. For the World Wide Web, these transitions model how a Web surfer may choose, at any moment and on any Web page, to manually type the address of a Web page in the address bar instead of clicking through hyperlinks. In this case, the user will "teleport" to another Web page regardless of whether a hyperlink to that Web page exists. Note that matrices **D** and **E** have a non-null intersection. The information in **D** is included in **E**. Both matrices describe transitions that occur as a result of a user typing the name of a Web page in the address bar, except that **D** only holds those transitions for the cases where the user, having reached a dangling node, has no other option than to teleport.
- α is known as the *expansion factor*, where $0 \leq \alpha \leq 1$, which models the probability that a user follows hyperlinks as opposed to accessing a Web page directly via the address bar. Note how Equation 4.3 consists of two complementary terms. The first term models the transitions via hyperlinks (unless no hyperlinks exist as in the case of dangling nodes). The second term models "teleportation", described via **E**. α describes the relative probability between these two transition types. If $\alpha = 1$, the user always only follows hyperlinks (when they are available). If $\alpha = 0$, the user never follows hyperlinks and only teleports through the Web by typing Web sites in the address bar. Brin and Page (1998) advised to set $\alpha = 0.85$. Note that the simplified version of PageRank from Equation 4.1 featured $\alpha = 1$.

In summary, the matrix **G** models how a user navigates the Web. **H** models the user's use of hyperlinks. **D** models what the user does when a Web page with no hyperlinks

[5] A *stochastic matrix* holds the probability of transitioning between nodes as it would be described by Markov chains. It follows that the sum of each row must be equal to 1. A matrix **A** is *primitive* if $\exists k \forall (i,j) : A^k_{ij} > 0$ (i.e., the k^{th} power is positive). A primitive matrix is both irreducible and aperiodic. Irreducible means that any state in a Markov chain is reachable from any other state. Aperiodic means that there is no set period for returning to any given state. These properties allow the iterative algorithm to converge.

is reached. **E** models the transitions that take place when the user chooses to not use hyperlinks, but directly go to a Web page of choice. The matrices **H**, **D**, **E**, and the parameter α can then be tailored to a user's Web surfing behavior in order to produce more personalized results once the PageRank vector **R** is evaluated.

4.3.2. USING PAGERANK TO MODEL SWARMS

Just like Brin and Page used the Google matrix from Equation 4.3 to model the behavior of a Web surfer, we can use it to model the behavior of a robot in the swarm. To do so, we must correlate the local experiences of a robot to a graph structure. We thus begin by defining the following discrete sets.

- Let $\mathcal{S}$ be the local state space of a robot. This is the set of local, discretized states that a robot can observe using its onboard sensors. The local states are going to be analogous to Web pages.
- Let $\mathcal{A}$ be the set of all discrete local actions that a robot can take.
- Let Π be a stochastic policy (a stochastic map between states $\mathcal{S}$ and actions $\mathcal{A}$) that the robot follows. This stochastic policy is analogous to the hyperlinks in the Web, as it allows robots to travel through local states.

Now consider a graph $G_{\mathcal{S}} = (V, E)$. The graph $G_{\mathcal{S}}$ is our microscopic local model of the robot in the swarm. The nodes of $G_{\mathcal{S}}$ are the local states that the robots can be in, such that $V = \mathcal{S}$. The edges of $G_{\mathcal{S}}$ are all transitions between local states that could take place. The local transitions can be of two types: "active" or "passive". That is, they can either happen as a result of an action by the robot (active), or they can happen because the environment around the robot changes (passive). More formally, $G_{\mathcal{S}}$ is the union of two subgraphs: $G^a_{\mathcal{S}}$ and $G^p_{\mathcal{S}}$.

- $G^a_{\mathcal{S}}$, whereby the superscript a stands for *active*, holds all state transitions that a robot could go through by an action of its own based on the stochastic policy Π. The edges of this graph are weighted based on the relative probabilities in the stochastic policy Π, and thus represent how likely it is that the robot will take the action when in a given local state.
- $G^p_{\mathcal{S}}$, whereby the superscript p stands for *passive*, holds all state transitions that a robot could go through because of changes in its environment, independently from Π.

The graphs $G^a_{\mathcal{S}}$ and $G^p_{\mathcal{S}}$ model the robot and the effects of the environment on the robot, respectively. These models are specific to the type of robot that is being used, its state space, its action space, and the nature of the task.[6] For each of the three tasks that we explore in this manuscript (consensus, pattern formation, and aggregation) we will show how to define $G^a_{\mathcal{S}}$ and $G^p_{\mathcal{S}}$ accordingly. Once properly defined, the graphs $G^a_{\mathcal{S}}$ and $G^p_{\mathcal{S}}$ can then be expressed as the matrices **H**, **E**, and **D** first introduced in Section 4.3.1.

[6] The attentive reader may have noticed the similarity of $G^a_{\mathcal{S}}$ and $G^p_{\mathcal{S}}$, explained in this section, with $G^1_{\mathcal{S}}$, $G^2_{\mathcal{S}}$ and $G^3_{\mathcal{S}}$, from Chapter 3. This similarity will be addressed in more detail when we use the PageRank measure for the pattern formation task, in Section 4.5.

- **H** shall model how a robot navigates in its environment by its own actions, describing the possible local state transitions that may take place when the robot executes these actions. This is analogous to a user navigating through Web pages via hyperlinks. We thus define:

$$\mathbf{H} = \text{adj}(G^a_{\mathcal{S}}) \tag{4.4}$$

  where adj(G) denotes the weighted adjacency matrix of a graph G.
- **E** models the "environment". These are state transitions that can happen to the robot even when it is not taking an action. For example, a neighbor could move away and cease to be a neighbor. This is analogous to a Web user who, instead of following hyperlinks through the Web, navigates to another Web page via the address bar. In this case, the user instigates a state transition that is not described by the hyperlink structure (i.e., the policy) of the World Wide Web. The matrix **E** is thus given by:

$$\mathbf{E} = \text{adj}(G^p_{\mathcal{S}}) \tag{4.5}$$

- **D** shall model what can happen to a robot as a result of the changing environment around the robot, *if and only if the robot reaches a state wherein it cannot (or will not) take any actions*. These states are states that the policy Π does not map to any actions. We group these states in the set $\mathcal{S}_{static} \subseteq \mathcal{S}$. Reaching such a state is analogous to reaching a Web page with no hyperlinks. We define **D** as:

$$\mathbf{D} = \text{adj}(G^p_{\mathcal{S}}(S_{static})) \tag{4.6}$$

Note that the matrices **H**, **E**, and **D** must be row normalized in order for the iterative procedure of Equation 4.2 to converge.

The expansion factor α remains to be defined, which models the probability that a robot will take an action over being subject to changes in the environment, as modeled by **E**. One may choose to keep α constant, as originally suggested by Page et al. (1999). For instance, α could be a constant value that is a function of the robot density. However, this would be a global parameter, and would thus not be representative of the local experience of the robots, unlike the remainder of the model. Instead, we propose to make α a function of the local state that the robot is in. For instance, this may reflect that, if the robot is in a local state whereby it is surrounded by several neighbors, the probability that it undergoes passive transitions due to actions of its neighbors may be higher than if it is in a state whereby it is not surrounded by neighbors at all. While exploring the three tasks analyzed in this chapter, we will define different approaches to express α in more detail.

4.3.3. USING PAGERANK TO EVALUATE THE PERFORMANCE OF THE SWARM

Once a microscopic model of the swarm has been constructed using the framework described in Section 4.3.2, we can evaluate the PageRank centrality of all states in $\mathcal{S}$. This will inform us on how likely it is for a robot in the swarm to reach each local state. In turn, this will be used to evaluate the performance of the whole swarm.

This is where we devise the micro-macro link based on PageRank centrality. The true metric that we wish to optimize is the efficiency of the swarm in achieving a given

global goal. However, directly assessing this parameter would require us to simulate the swarm (which may lead to the scalability problems introduced at the beginning of this chapter), or use a macroscopic model of the swarm (which may fail to capture the details of the policy). Therefore, instead of directly optimizing the stochastic policy Π against the global performance metric, we optimize the likelihood that a robot in the swarm will end up in certain local states of interest, i.e., local states that we know should be reached for the global goal to happen. For example, in the aggregation task, we will wish to achieve (and stay in) local states with neighbors over local states without neighbors. Alternatively, in the consensus task, the robot will aim to be in a state of agreement with its neighbors. Because all robots in the swarm are trying to achieve these desired local states as efficiently as possible, the global effect will then be that the swarm achieves the global goal more efficiently as well.

This concept can be formalized into a fitness function. Let $\mathcal{S}_{des} \subseteq \mathcal{S}$ be the set of local desired states pertaining to a global goal, and let $R(s)$ be the PageRank centrality of a state $s \in \mathcal{S}$. Based on this, we propose the following fitness function in order to evaluate the performance of the swarm:

$$F = \frac{\sum_{s \in \mathcal{S}_{des}} R(s)/|\mathcal{S}_{des}|}{\sum_{s \in \mathcal{S}} R(s)/|\mathcal{S}|} \tag{4.7}$$

where $R(s)$ is the PageRank of state s. It is extracted following the calculation of the PageRank vector $\mathbf{R}$ from Equation 4.2 with the model of Section 4.3.2. This fitness function expresses the average PageRank of the states in $\mathcal{S}_{des}$ in relation to the average PageRank of all states $\mathcal{S}$. When used within an optimization strategy, our objective will be to alter Π so as to maximize F. This will maximize the average "popularity" of the states in $\mathcal{S}_{des}$ and increase the likelihood for the individual robot to be in one of the states. At the global level, we expect that if all robots act such that they are likely to enter a state $s \in \mathcal{S}_{des}$, then the final global goal will also be more likely to emerge. Defining the set of desired states for a given task may appear troublesome. However, throughout this chapter we present three different tasks and show that the desired states can often be intuitively extracted from a global goal. We will return to this discussion in Section 4.7. Furthermore, in Chapter 5, we will also propose a method to automatically extract the desired states.

4.4. SAMPLE TASK 1: CONSENSUS AGREEMENT

In this first task, a swarm of robots needs to achieve a consensus between a certain set of options. We will use the fitness function described in Section 4.3.3 to optimize the probability with which a robot should change its opinion such that the entire swarm achieves consensus as quickly as possible.

4.4.1. TASK DESCRIPTION AND SETTING

Consider a set of N robots that must collectively choose between M options. As an example, these options could be possible sites to visit together with the rest of the swarm. We focus on the case where all options are of equal value, meaning that the robots do not have a preference for any particular option. Let $\mathcal{C}$ be the set of M options that the swarm

considers. The global goal of the swarm is for all robots to settle on a choice $c \in \mathcal{C}$. We shall assume that the robots are in a static arbitrary connected configuration P.

The state s_i of a robot $\mathcal{R}_i$ holds the opinion of robot $\mathcal{R}_i$ and the number of neighbors with each of the other opinions. As an example, consider a swarm that must choose between $M = 2$ options, whereby $\mathcal{C} = \{A, B\}$. The local state space of a robot $\mathcal{R}_i$ is then described by all possible combinations of these three variables:

1. The internal opinion of $\mathcal{R}_i$, denoted c_i.
2. The number of neighbors of $\mathcal{R}_i$ with opinion A, denoted n_{iA}.
3. The number of neighbors of $\mathcal{R}_i$ with opinion B, denoted n_{iB}.

As each robot can only sense a maximum number of neighbors $N_{\max}$, there is a constraint $n_{iA} + n_{iB} \leq N_{\max}$. In reference to our setup, we shall set $N_{\max} = 8$. All possible combinations of the above form the local state space $\mathcal{S}$ for this task. If $M > 2$, then the $\mathcal{S}$ can be expanded accordingly to accommodate all other relevant combinations.

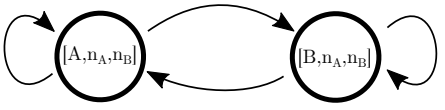
Based on its local state s_i, a robot can choose to keep its opinion or change it. We set the action space $\mathcal{A}$ to $\mathcal{A} = \mathcal{C}$. Each robot follows a stochastic policy Π, dictating the probability of choosing an opinion $c \in \mathcal{C}$ (including its current one) for each state $s \in \mathcal{S}$. We then define the set $\mathcal{S}_{des}$ to be all states wherein the robot has the same opinion as all of its neighbors, for which we know a priori that there is no need to change opinion, as all robots are locally in agreement. These states are excluded from the stochastic policy Π, creating a global convergence point for the swarm. If all robots are in one of these local state, then it follows that consensus (which is the global goal) has been achieved. The size of the stochastic policy is $|\Pi| = (|\mathcal{S}| - |\mathcal{S}_{des}|) \cdot |\mathcal{A}|$.

4.4.2. PAGERANK MODEL

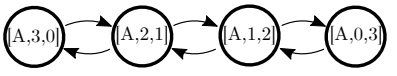
To define the PageRank model, we need to define $G^a_{\mathcal{S}}$, which denotes the state transitions whenever a given robot takes an action, and $G^p_{\mathcal{S}}$, which models the state transitions whenever the neighbors of the robot take actions.

Let us begin by defining $G^a_{\mathcal{S}}$. In this case, whenever a robot $\mathcal{R}_i$ changes its opinion between the M options that are available, only its internal choice changes, while the choice that its neighbors hold is not (directly) impacted by the action. Thus, for the example of $M = 2$, the only parameter of its state that changes is c_i, while n_{iA} and n_{iB} remain constant. $G^a_{\mathcal{S}}$ is thus formed by a set of disconnected subgraphs, each where the robot is only capable of changing its own internal state, but all other parameters stay constant. A sample subgraph of $G^a_{\mathcal{S}}$ for $M = 2$ is depicted in Figure 4.1a.

$G^p_{\mathcal{S}}$ follows the same philosophy as $G^a_{\mathcal{S}}$, but from the opposite perspective. The internal choice c_i of a robot $\mathcal{R}_i$ does not change when its neighbors change opinions. What changes is the number of neighbors with a certain opinion. The model assumes that the robot will always notice every time one of its neighbors changes opinion. $G^p_{\mathcal{S}}$ thus also takes the form of several disconnected subgraphs. An example of a subgraph for $M = 2$ is depicted in Figure 4.1b for a robot with three neighbors. The matrices **H**, **E**, and **D** are extracted from $G^a_{\mathcal{S}}$ and $G^p_{\mathcal{S}}$ using Equation 4.4, Equation 4.5, and Equation 4.6, respectively.



(a) Depiction of a subgraph of $G^a_{\mathcal{S}}$ for a robot which can change its opinion from A to B and vice versa. Its neighbors do not change opinion, hence n_A and n_B remain constant.



(b) Depiction of a subgraph of $G^p_{\mathcal{S}}$ for a robot with opinion A and three neighbors. All transitions have equal probability.

Figure 4.1 Representation of subgraphs of $G^a_{\mathcal{S}}$ and $G^p_{\mathcal{S}}$ for a consensus task with $M = 2$ options, $C = \{A, B\}$.

Finally, we define the parameter α. For this task, we can reason that, if the robot is surrounded by more neighbors, then the likelihood that it will change opinion before one of its neighbors decreases. We can include this in our local model via a redefinition of α. To do so, we define a vector $\alpha_{\mathbf{v}}$ that holds a different value of α for each state $s \in \mathcal{S}$, such that the equation for the Google matrix is modified to:

$$\mathbf{G} = \mathbf{D}_{\alpha_{\mathbf{v}}}(\mathbf{H} + \mathbf{D}) + \left(\mathbf{I} - \mathbf{D}_{\alpha_{\mathbf{v}}}\right)\mathbf{E} \tag{4.8}$$

where $\mathbf{D}_{\alpha_{\mathbf{v}}}$ is a diagonal matrix holding the vector $\alpha_{\mathbf{v}}$. The entries of $\alpha_{\mathbf{v}}$ are $\alpha_i \leq 1$ for $i = 1, \ldots, |\mathcal{S}|$. In this work, we model α_i for a given state $s_i \in \mathcal{S}$ using the following general definition:

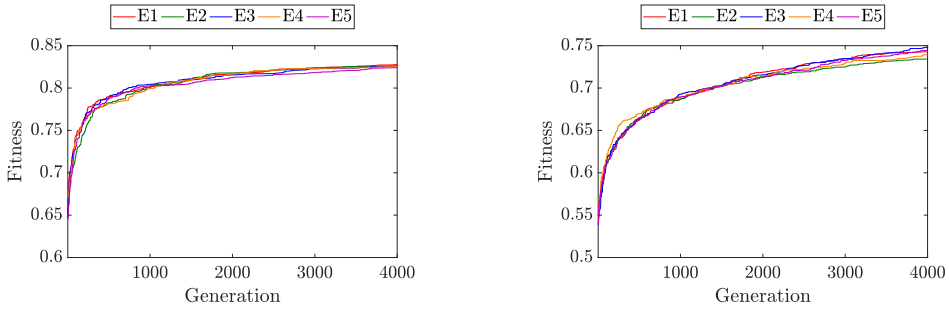
$$\alpha_i = p_{action}(s_i) \cdot \frac{1}{n_{neighbors}(s_i) + 1}, \tag{4.9}$$

where $p_{action}(s_i) = \sum_{a \in \mathcal{A}} P(a|s_i)$. Thus, $p_{action}(s_i)$ is the cumulative probability of taking an action (any action) from the stochastic policy Π when in state s_i. All states $s_i \in \mathcal{S}_{active}$ feature $0 \leq p_{action}(s_i) \leq 1$. If $p_{action}(s_i) < 1$, then there is also a probability that, when in state s_i, the robot will not take an action but remain idle. The parameter $n_{neighbors}(s_i)$ is the number of neighbors at state s_i. As this parameter increases, α_i decreases. This same definition α will also be used for the pattern formation task in Section 4.5.

With the above, we have now fully defined a microscopic model of the swarm from the perspective of an individual robot while using PageRank's framework. We now proceed to optimize Π in order to maximize the fitness function expressed in Equation 4.7.

4.4.3. GENETIC ALGORITHM SETUP AND RESULTS

The optimization in this work is done via a Genetic Algorithm (GA). The GA features a population of ten scalar genomes, where the size of each genome is equal to Π. Each gene in a genome holds a value $0 \leq p \leq 1$, indicating the probability of executing the corresponding state-action pair from Π. Each new generation is produced by elite members (30%), offspring (40%), and mutated members (30%). Offspring are generated from averaging two parents' genomes. Mutation replaces 10% of a genome's genes with random values from a uniform distribution. The initial generation was created by assigning random values between 0 and 1 to all genes in each genome, following a uniform distribution. Note that the cumulative probability of taking any of the actions when in a given state is always normalized, since all choices are part of the policy.



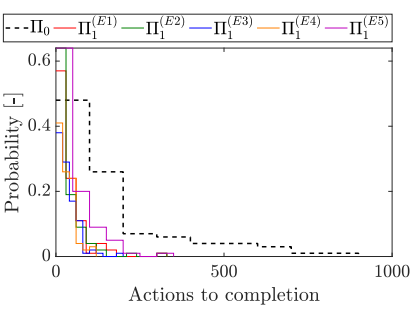
(a) $M = 2$ **(b)** $M = 3$

Figure 4.2 Evolution of policy for consensus agreement task with two choices ($M = 2$) and three choices ($M = 3$).

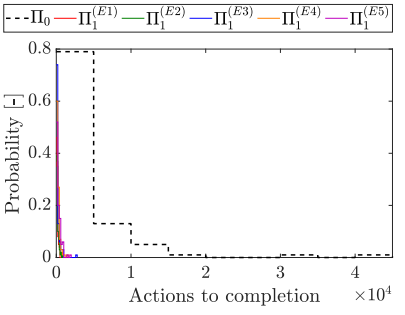
With this setup, we evolved behaviors for the case where $M = 2$ and $M = 3$. The results of the fitness increase over five evolutionary runs are shown in Figure 4.2, where we show the best genome that was evolved for both the case where $M = 2$ (Figure 4.2a) and the case where $M = 3$ (Figure 4.2b).

We now test the results of the evolved behavior to examine the magnitude by which the global performance of the swarm has improved. To do so, the swarm is simulated in discrete time whereby, at each time step, a random robot in the swarm changes its opinion. This models random concurrency in the swarm. For simplicity, we assume that the robots are placed on a grid world. Each robot $\mathcal{R}_i$ is capable of sensing the opinion of any robot $\mathcal{R}_j$ that happens to be in the eight grid points that surround it. We measure the performance of the swarm as the cumulative number of times that the robots in the swarm take an action before consensus is achieved, whereby an action is whenever a robot renews its opinion. This is indicative of how quickly the swarm is capable of reaching a consensus to one of the options. The results are compared to a basic baseline behavior where the robots choose between options with equal probability (except for the states $\mathcal{S}_{des}$, in which case they are in full agreement with their neighborhood and remain idle). This provides a reference for the improvement that the optimized policy brights to the system. In the following, the baseline behavior is denoted Π_0, and the evolved behaviors are denoted Π_1.

The tests were performed for swarms of ten robots and 20 robots. Each swarm was evaluated 100 times, in random configurations and from random starting conditions. The results for a swarm that must choose between $M = 2$ options are shown in Figure 4.3. The results for a swarm that must choose between $M = 3$ options are shown in Figure 4.4. In all cases, the swarm is capable of achieving a consensus substantially faster than with the baseline behavior. Moreover, we see that the performance is robust to the number of robots in the swarm. These results show that the evolutionary procedure was capable of finding a local behavior that provides an efficient outcome at the global level and also adapts well to the number of robots, the initial conditions, and the spatial configuration of the robots.

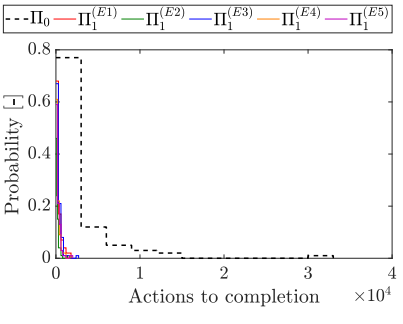


(a) Swarm of ten robots

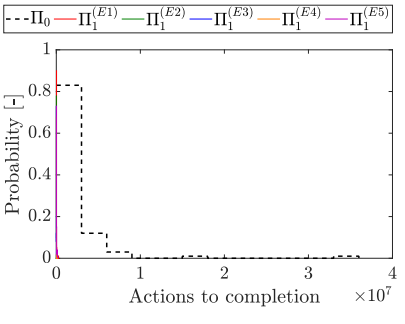


(b) Swarm of 20 robots

Figure 4.3 Performance of consensus agreement task where $M = 2$, meaning that the robots must settle between two choices. The global performance is measured by the number of times that, cumulatively, the robots change their decision before a consensus is achieved.



(a) Swarm of ten robots



(b) Swarm of 20 robots

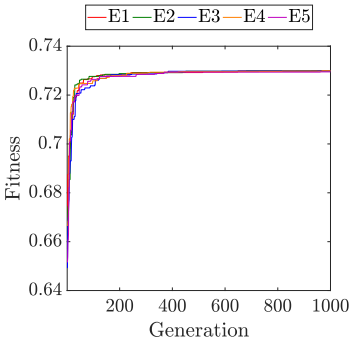
Figure 4.4 Performance of consensus agreement task where $M = 3$, meaning that the robots must settle between three choices. The global performance is measured by the number of times that, cumulatively, the robots change their decision before a consensus is achieved.

4.4.4. VARIANT WITH LIMITED BINARY COGNITION

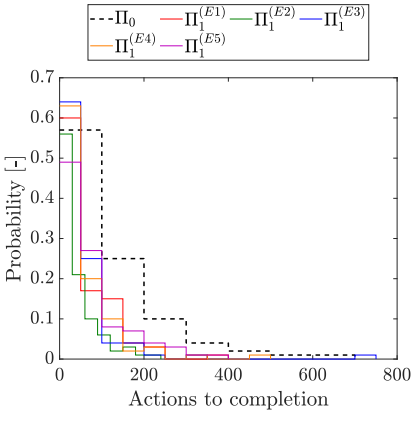
To gain further insight into the adaptability of the framework, we consider a limited variant of the consensus task where robots are endowed with binary sensors. Each robot is only capable of sensing whether *all* of its neighbors agree with it or not, but is unable to discern the opinions of individual neighbors. Such a case reflects the impact of high noise or poor sensing abilities on the part of the robots.

Consider the case where robots must choose between $M = 2$ options, with choices $\mathcal{C} = \{A, B\}$. It follows that each robot in the swarm can be in one of four local states:

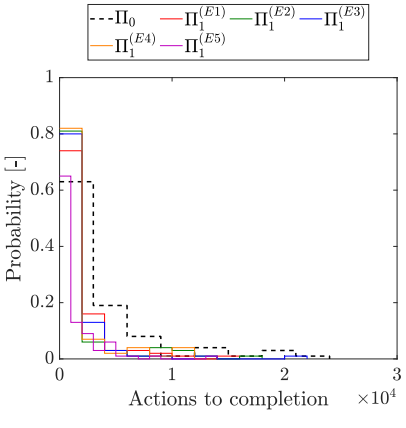
s_{A0}: Internal opinion A, but the robot is not in agreement with all neighbors.
s_{A1}: Internal opinion A, and the robot is in agreement with all neighbors.
s_{B0}: Internal opinion B, but the robot is not in agreement with all neighbors.
s_{B1}: Internal opinion B, and the robot is in agreement with all neighbors.



(a) The best fitness over five evolutionary runs



(b) Performance improvements with a swarm of ten robots of the evolved policies versus the baseline policy



(c) Performance improvements with a swarm of 20 robots of the evolved policies versus the baseline policy

Figure 4.5 Evolution and performance of the policy for the binary variant of the consensus task. The baseline policy is as seen in Table 4.1a and the evolved policies, for all cases, are similar to the one in Table 4.1b.

The local state space is $\mathcal{S} = \{s_{A0}, s_{A1}, s_{B0}, s_{B1}\}$, and $\mathcal{S}_{static} = \mathcal{S}_{des} = \{s_{A1}, s_{B1}\}$. If in a state $s \notin \mathcal{S}_{des}$, a robot can choose between two actions.

a_A: Select opinion A
a_B: Select opinion B

The policy is evolved following the same evolutionary setup as in Section 4.4.3. The only difference is that, as the robots are now incapable of sensing the number of robots in their neighborhood, we set $\alpha = 0.3$ for all states, instead of using Equation 4.9. The results of five evolutionary runs are shown in Figure 4.5a. As expected, all evolutionary runs evolve to similar results. The performance improvement over a baseline case (whereby the robots alternate between states with equal probability) is shown in Figure 4.5b and Figure 4.5c for swarms of ten robots and 20 robots, respectively.

4.4.5. ANALYSIS

The small size of the solution space for the variant of the consensus task studied in Section 4.4.4 provides an opportunity to analyze the evolved policy and the global fitness landscape in detail. The baseline policy and the evolved policies are given in Table 4.1a and Table 4.1b, respectively. In Table 4.1b it can be seen that our evolutionary procedure determined that the best strategy would be for the robots to, almost deterministically, switch their opinion.[7] We investigated whether this was an optimum solution by simulating the system for different probabilities of switching opinion. This was done using the same simulation setup as all evaluations in this section, whereby each setting was evaluated and averaged over 100 runs.

A first set of results for these simulations is shown in Figure 4.6a, which shows the performance of the swarm in relation to the probability of switching. Here, it is confirmed that the policy is correct. As the probability of switching choices increases, the swarm is capable of achieving consensus more efficiently. It can also be seen that the evolved policy scales well with the size of the swarm.

We further investigated the entire solution space for a swarm of ten robots. Here, it is revealed that the evolved policy Π_1, as given in Table 4.1b, is not the global optimum, but the unbiased global optimum. By unbiased, we mean that the robots are equally probable of selecting option A as they are of selecting option B. However, more optimal options exist if the robots become biased toward one of the two options. This result is actually reflected in the PageRank-based fitness function with which the behavior was evolved, as we had provided the set $\mathcal{S}_{des} = \{s_{A1}, s_{B1}\}$ and gave equal importance to both options. It then follows that this brought us to the unbiased optimum.

To test whether our framework could adapt to this, we also evolved the policy for the cases where $\mathcal{S}_{des} = \{s_{A1}\}$ and where $\mathcal{S}_{des} = \{s_{B1}\}$. Note that we kept $\mathcal{S}_{static} = \{s_{A1}, s_{B1}\}$, because we still want the robots to stop if they are in agreement with their neighborhood. The results of these two evolutions are given in Table 4.2a and Table 4.2b for $\mathcal{S}_{des} = \{s_{A1}\}$ and $\mathcal{S}_{des} = \{s_{B1}\}$, respectively. The evolution discovered the biased optimum for both options. This shows a flexibility on the part of the algorithm to adapt to the desired goals. Originally, the algorithm had evolved the unbiased policy because it had been told that

[7] We attribute the fact that the solution is nearly deterministic, and not fully deterministic, to our GA implementation.

Table 4.1 Baseline policy and evolved policy for the binary variant of the consensus task for the case of $M = 2$. In the original policy (denoted Π_0), the robots have equal probability of selecting either choice. In the evolved policy (denoted Π_1), the robots always switch their opinion whenever their neighborhood is not in agreement with their choice. For Π_1, the reason that the values are not exactly 1 and 0 is likely because of the mutation strategy that was used during the evolution.

(a) Baseline policy Π_0

Π_0	a_A	a_B
s_{A0}	0.5	0.5
s_{B0}	0.5	0.5

(b) Evolved unbiased policy Π_1

Π_1	a_A	a_B
s_{A0}	0.002	0.998
s_{B0}	0.999	0.001

Table 4.2 Evolved policies with a bias toward options A or B. For both policies, the reason that the values are not exactly 1 and 0 is likely because of the mutation strategy that was used during the evolution.

(a) Policy Π_{bA} with bias to option A

Π_{bA}	a_A	a_B
s_{A0}	0.9982	0.0018
s_{B0}	0.9982	0.0018

(b) Policy Π_{bB} with bias to option B

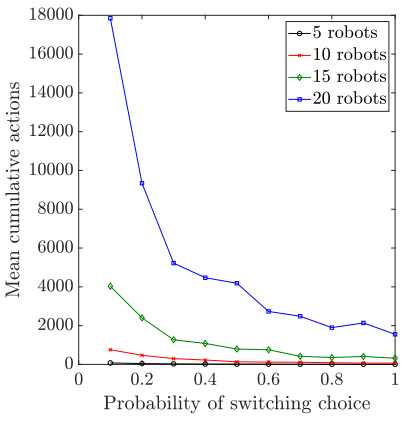
Π_{bB}	a_A	a_B
s_{A0}	0.0004	0.9996
s_{B0}	0.0042	0.9958

both options had equal importance. When this constraint was altered to a new desired goal, the algorithm evolved a policy which reflected the new goal.

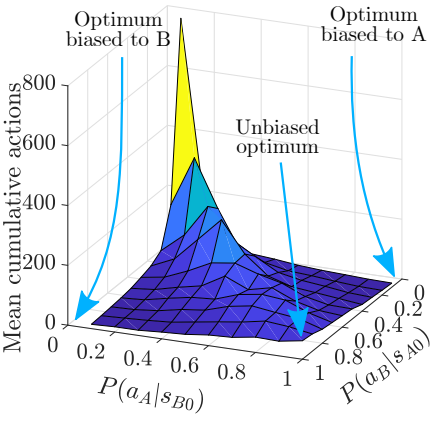
Additionally to the analysis above, the scalability of the evolved policies was analyzed in order to better understand their performance as swarm size increases. The comparison results showing the average performance of the policies are shown in Figure 4.7a and Figure 4.7b. This comparison is based on the results shown in the previous sections. It can be seen that the evolved policies scale well with the size of the swarm. The policy that was evolved with knowledge of the neighborhood scales gracefully, with only a minimal increase in the number of actions taken (per agent) as swarm size increases. A reason for this increase is the fact that, in a larger swarm, a robot is more likely to be surrounded by more neighbors, thus experiencing more uncertainty in its neighborhood. Furthermore, note that when a robot chooses to stay with its current choice, then this is also counted as an action. As expected, the limited variant evolved in Section 4.4.4, by nature of the fact that the robots have less data on their local surroundings, begins to struggle more as the swarm size increases.

4.5. SAMPLE TASK 2: PATTERN FORMATION

This task deals with a swarm of robots with low awareness of their surroundings that must arrange into a desired spatial configuration. The robots are homogeneous (identical and without hierarchy), anonymous (did not have identities), reactive (memoryless), cannot communicate, do not have global position information, do not know the goal of the swarm, and operate asynchronously (i.e., by local clocks) in an unbounded space. The only knowledge available to the robots is: 1) a common heading direction, such as north, and 2) the relative location of their neighbors within a maximum range. In the work in Chapter 3 (Coppola et al., 2019b), we have developed a local behavior with which such limited robots can always eventually arrange into a global desired pattern. More-



(a) Correlation between the global performance of the swarm and the probability of switching opinion for swarms of different sizes.



(b) Correlation between the global performance of the swarm and the probability of switching opinion to choices A or B, for a swarm of ten robots.

Figure 4.6 Depiction of global policy analysis for the probability of switching between two options A and B in the limited variant of the consensus task.

over, we provided a set of conditions to verify whether a given behavior would always eventually lead to the emergence of the desired pattern.

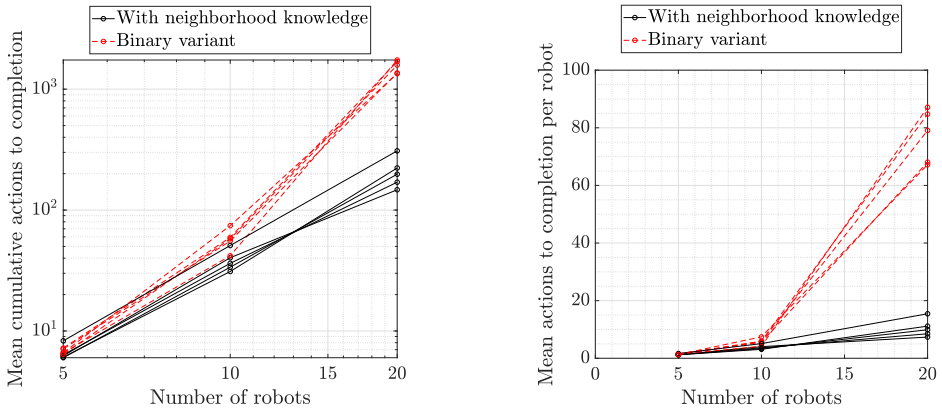
One issue that remained to be solved was that even simple patterns were found to take up to hundreds of actions (cumulatively by all robots) to emerge, and this number appeared to grow with the complexity of the pattern and the size of the swarm. Although this is to be expected, in light of the limited knowledge on the part of the robots, it is an issue that cannot be ignored if the system is to be used on real robots with limited battery life and other real-world time constraints. Solving this via classical approaches, wherein the swarm had to be ultimately simulated in order to find an optimum policy, proved unscalable (Coppola and de Croon, 2018). This was because of all the issues listed in Section 4.1. We now show how we can circumvent the scalability problem by tackling the optimization using the PageRank algorithm.

4.5.1. DESCRIPTION OF APPROACH TO PATTERN FORMATION

Such that this manuscript may be self-contained, this section summarizes the pattern formation methodology from Coppola et al. (2019b), wherein a more detailed explanation can be found. For the sake of brevity, in this work we will assume that the swarm operates in a grid world and in discrete time. However, as demonstrated in Coppola et al. (2019b), the behavior can also be used in continuous time and space with robots operating with local clocks.

SETTING

Consider N robots that exist in an unbounded discrete grid world and operate in discrete time. In the case studied here, each robot $\mathcal{R}_i$ can sense the location of its neighbors in



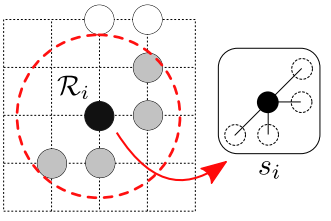
(a) Mean cumulative actions taken by the swarm

(b) Mean number of actions taken per robot

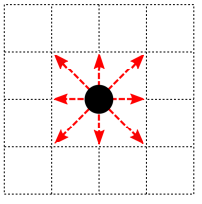
Figure 4.7 Comparison of mean performance of the evolved policies for both variants of the consensus task for the case where $M = 2$.

the eight grid points that surround it, as depicted in Figure 4.8a. This is the local state s_i of the robot, which is all the information that it has. The local state space $\mathcal{S}$ consists of all combinations of neighbors that it could sense, such that $|\mathcal{S}| = 2^8$. At time step $k = 0$, we assume the swarm begins in a connected topology forming an arbitrary pattern P_0. At each time step, one random robot in the swarm takes an action, whereby it is able to move to any of the eight grid points surrounding it, as depicted in Figure 4.8b. This is the action space of the robots, denoted $\mathcal{A}$. If a robot takes an action at one time step, then it will not take an action at the next time step. This models the fact that a real robot would need some time to settle after each action and reassess its situation, leaving a time window for its neighbors to move.

The goal of the swarm is to rearrange from its initial arbitrary pattern P_0 into a desired pattern P_{des}. This is achieved using the following principle. The local states that the robots are in when P_{des} is formed are extracted, and form a set of local desired states $\mathcal{S}_{des} \subseteq \mathcal{S}$, as depicted by the examples in Figure 4.9. These local desired states are extracted as the observations of the robots once the pattern is formed, similarly to how puzzle pieces form a puzzle. If robot $\mathcal{R}_i$ finds itself in any state $s_i \in \mathcal{S}_{des}$, then it is instructed to not move, because, from its perspective, the goal has been achieved. Given a P_{des} and the corresponding $\mathcal{S}_{des}$, it can be automatically (and sometimes even intuitively) verified whether the local desired states will uniquely form P_{des}, or whether they can also can give rise to spurious global patterns. If spurious global patterns are not possible, then, until P_{des} is formed, at least one robot will be in a state $s \notin \mathcal{S}_{des}$ and will seek to amend the situation. The swarm will then keep reshuffling unless P_{des} forms (Coppola et al., 2019b).



(a) Example of a local state $s_i \in \mathcal{S}$ of a robot $\mathcal{R}_i$. The robot is shown in black and its neighbors within sensing range are shown in gray. It is assumed that a robot can always sense neighbors in the eight grid points that surround it.



(b) Possible actions that a robot can take. The robot is capable of moving omnidirectionally to any of the eight grid points around its current position. These eight actions form the actions space $\mathcal{A}$.

Figure 4.8 Depictions of a local state and the actions that a robot in the swarm can take.

BASELINE BEHAVIOR OF THE ROBOTS

When a robot $\mathcal{R}_i$ is in a state $s_i \notin \mathcal{S}_{des}$, it should execute an action from $\mathcal{A}$. From the state space and action space, we can formulate a stochastic policy $\Pi = \mathcal{S} \times \mathcal{A}$. However, not all actions should be allowed. The actions that: a) cause collisions and b) cause local separation of the swarm are eliminated from Π, because they are not "safe". Moreover, as previously explained, all states $s \in \mathcal{S}_{des}$ do not take any actions. From this, we extract a final stochastic policy Π_0, where $\Pi_0 \subseteq \Pi$. When a robot is in a state s, it will use the policy Π_0 to randomly select one possible action from the available state-action pairings. In Coppola et al. (2019b), the final action was chosen from the available options based on a uniform probability distribution.

It is important to note that, from this pruning process, there also emerge additional local states that cannot take any actions. A robot in such a state will not be able to move or else it will either collide with other robots or possibly cause separation of the swarm. We refer to such states as *blocked* states. The set of blocked states is denoted $\mathcal{S}_{blocked}$. The states in $\mathcal{S}_{des}$ and $\mathcal{S}_{blocked}$ are functionally equivalent. In either case, a robot will not take any action. Together, they form the umbrella set $\mathcal{S}_{static} = \mathcal{S}_{des} \cup \mathcal{S}_{blocked}$.

Furthermore, conceptually in contrast to static states, there are states where a robot will be capable of moving away from and/or around its neighborhood without issues. We call these states *simplicial*. Simplicial states are characterized by only having one *clique*, where we define a clique as a connected set of neighbors.[8] The set of simplicial states is denoted $\mathcal{S}_{simplicial}$. Figure 4.10 shows examples of blocked states (Figure 4.10a and Figure 4.10b), a simplicial state (Figure 4.10c), and a non-simplicial state (Figure 4.10d).

MODELING THE LOCAL EXPERIENCES OF THE ROBOTS

Let us now construct the graph $G_{\mathcal{S}} = (V, E)$ that models the local experiences of the robots. The nodes of $G_{\mathcal{S}}$ are the local states that the robots can be in, such that $V = \mathcal{S}$. The edges of $G_{\mathcal{S}}$ are all possible transitions between local states.

[8] If the neighbors of a robot form only one clique, then this means that, if this robot were to disappear, it is guaranteed that its neighbors would all remain connected among each other.

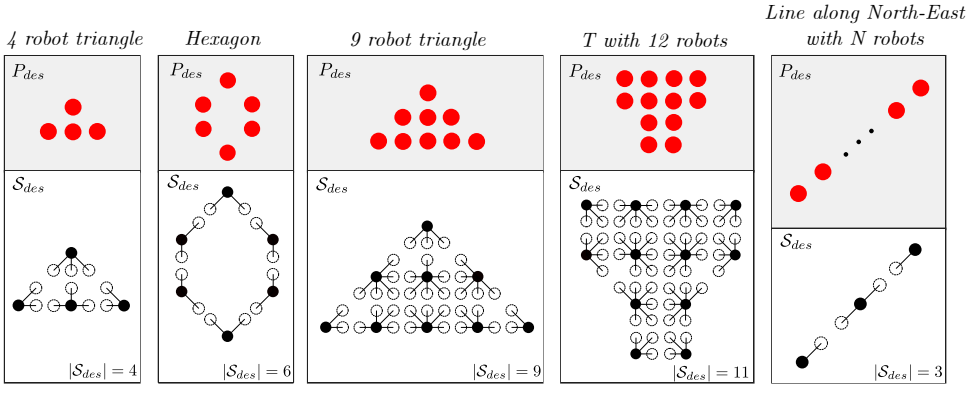


Figure 4.9 Set of desired states $\mathcal{S}_{des}$ for the exemplary patterns treated in this chapter, featuring patterns of increasing complexity and/or size. The set $\mathcal{S}_{des}$ can be intuitively extracted for each pattern as the "puzzle pieces" that compose it. Note that the T with 12 robots has 11 states in $\mathcal{S}_{des}$. This is because the top state repeats. Also note that the line always only has three states in $\mathcal{S}_{des}$, because the center state can repeat indefinitely.

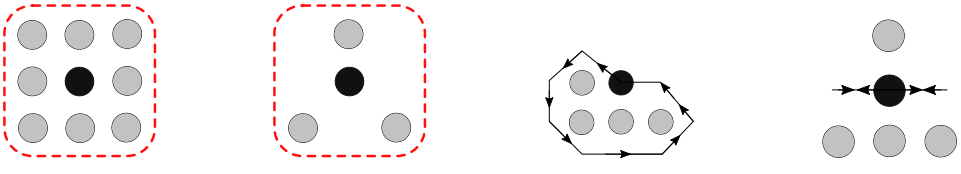


Figure 4.10 Examples of: **(a)** a state $s \in \mathcal{S}_{blocked}$, due to it being surrounded; **(b)** a state $s \in \mathcal{S}_{blocked}$, because any motion will cause the swarm to locally disconnect; **(c)** a state $s \in \mathcal{S}_{active} \cap \mathcal{S}_{simplicial}$, because it can travel around all its neighbors; **(d)** a state $s \in \mathcal{S}_{active}$ but $s \notin \mathcal{S}_{simplicial}$, because it can move but it cannot travel around all its neighbors or else it might disconnect the swarm.

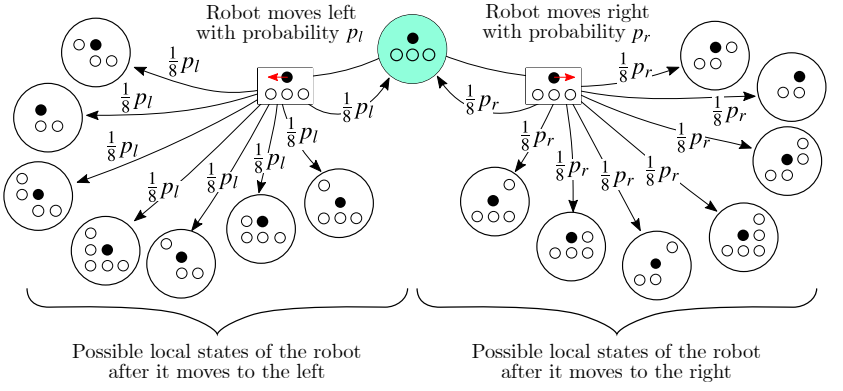
For this task, we break down $G_{\mathcal{S}}$ in three subgraphs.

- $G^1_{\mathcal{S}}$ indicates all state transitions that a robot could go through by an action of its own, based on Π_0.
- $G^2_{\mathcal{S}}$ indicates all state transitions that a robot could go through by an action of its neighbors, which could also move out of view.
- $G^3_{\mathcal{S}}$ indicates all state transitions that a robot could go through if another robot, previously out of view, were to move into view and become a new neighbor.

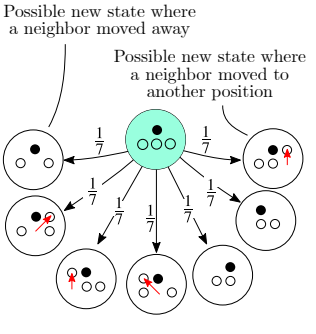
Additionally, let $G^{2r}_{\mathcal{S}}$ be a subgraph of $G^2_{\mathcal{S}}$. $G^{2r}_{\mathcal{S}}$ only holds the state transitions in $G^2_{\mathcal{S}}$ where neighbors stay in view, and does not hold the ones where neighbors fall out of view.

To help visualize $G^1_{\mathcal{S}}$, $G^2_{\mathcal{S}}$, and $G^3_{\mathcal{S}}$, Figure 4.11 shows a node and its successors for each graph. Figure 4.11a shows a node and its successors for $G^1_{\mathcal{S}}$ following an action based on a sample policy Π_0. Figure 4.11b and Figure 4.11c show the same node in graphs $G^2_{\mathcal{S}}$ and $G^3_{\mathcal{S}}$, respectively. As it can be seen, these three graphs bear a strong re-

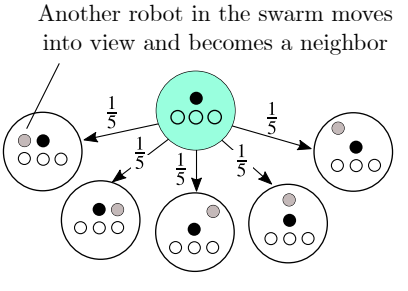
semblance to the graphs $G^a_{\mathcal{S}}$ and $G^p_{\mathcal{S}}$ needed to define PageRank. We shall return to this in Section 4.5.2, where the microscopic PageRank model for this task is defined.



(a) Example of a state (i.e., node) $s \in \mathcal{S}$ in graph $G^1_{\mathcal{S}}$ and its successor nodes. The successor nodes are extracted based on the possible actions to which the state s is mapped in Π_0. As the robot moves, it may end up in several states (including its original state) depending on the type of neighborhood that it finds after its action. Because it cannot see ahead, all options are possible until the action is executed. Therefore, all successor nodes following an action can be reached with equal probability. For this example, p_l indicates the probability of taking an action to the left, and p_r indicates the probability of taking an action to the right, according to the policy, when in state s.



(b) Example of a state (node) in graph $G^2_{\mathcal{S}}$ and its successor nodes. These are all possible states that the robot may be in after one of its neighbors takes an action. Because the robot does not know what states its neighbors are in, it is assumed that neighbors can take any action from $\mathcal{A}$ with equal probability. All state transitions have the same probability.



(c) Example of the state (node) in graph $G^3_{\mathcal{S}}$ and its successors. State transitions are caused by a new neighbor arriving at an empty grid point. All state transitions have the same probability of being reached.

Figure 4.11 Examples of a node and its successor nodes from graphs $G^1_{\mathcal{S}}$, $G^2_{\mathcal{S}}$, and $G^3_{\mathcal{S}}$.

VERIFYING THAT THE PATTERN WILL ALWAYS EVENTUALLY EMERGE

In Coppola et al. (2019b), we showed that by analyzing certain properties of $G^1_{\mathcal{S}}$, $G^2_{\mathcal{S}}$, and $G^3_{\mathcal{S}}$, it can be verified that the pattern P_{des} will eventually form starting from any initial pattern P_0. This way, we can assess whether a policy is such that the final pattern will always eventually emerge, or whether it may cause the swarm to reshuffle endlessly without ever settling into the pattern (we refer to this situation as a livelock).

The conditions are repeated here because, once we optimize the policy using PageRank, they shall be used as constraints in order to ensure that the final policy always eventually achieves the pattern. Specifically, the following conditions need to be met:

1. $G^1_{\mathcal{S}} \cup G^2_{\mathcal{S}}$ shows that each state in $\mathcal{S}$ features a path to each state in $\mathcal{S}_{des}$.
2. For all states $s \in \mathcal{S}_{static} \cap \mathcal{S}_{\neg simplicial} - s_{surrounded}$, none of the cliques of each state can be formed only by robots that are in a state $s \in \mathcal{S}_{des} \cap \mathcal{S}_{simplicial}$. Here, $s_{surrounded}$ is the state that is surrounded by neighbors along all directions.
3. $G^{2r}_{\mathcal{S}}$ shows that all static states with two neighbors can directly transition to an active state.
4. $G^1_{\mathcal{S}}$ shows that any robot in state $s \in \mathcal{S}_{active} \cap \mathcal{S}_{simplicial}$ can travel around all its local neighbors, as exemplified in Figure 4.10c (with the exception of when a loop is formed or when it enters a state $s \in \mathcal{S}_{static}$).
5. In $G^3_{\mathcal{S}}$, any state $s \in \mathcal{S}_{static}$ only has outward edges toward states $s \in \mathcal{S}_{active}$ (with the exception of a state that is fully surrounded along two or more perpendicular directions).

The detailed motivations behind these conditions can be found in the previous chapter. In summary, they ensure that all robots will keep moving around with sufficient freedom for the swarm to reshuffle until the pattern is achieved and that this will happen. Condition 1 checks that it is possible to reach the final local desired states independently of the initial local states. Conditions 2 and 3 check that there is always at least one robot in the swarm that has the potential to move with sufficient freedom. Conditions 4 and 5 check that the free robot(s) is (or are) capable of sufficient exploration.

These conditions are local in nature. They analyze the local states of a robot based on its limited sensing range and the actions that the robot could take as a result. The advantage of this is that checking whether the conditions are met is independent of the size of the swarm, avoiding the combinatorial explosion that would otherwise ensue. Note that the conditions are *sufficient*, but not *necessary*. Fulfilling them means that the pattern will be achieved, but not fulfilling them does not mean the pattern will not be achieved.[9] The first four patterns in Figure 4.9 pass the proof conditions. The line, instead, does not fulfill all conditions. Moreover, it can be subject to spurious patterns (for example, a slanted H) as a result of $\mathcal{S}_{static}$. A more thorough discussion on the patterns that can be generated using this approach can be found in the original paper.

This verification procedure, combined with the fact that the behavior deals with very limited robots (anonymous, homogeneous, memoryless, with limited range sensing,

[9] To this point, Condition 4 was made slightly more lenient than from what is expressed in (Coppola et al., 2019b) and Chapter 3. The original version checks that a robot can travel to *all* open positions surrounding *all* of its neighbors. This is a strongly restrictive rule that heavily limits optimization. Therefore, we limited to checking that a robot could travel about all its neighbors, but not to *all* open grid positions around *all* its neighbors.

and without needing any communication, global knowledge, or seed robots) moving in space, sets the work in Coppola et al. (2019b) apart from other works such as the ones of Klavins (2007), Yamins and Nagpal (2008), or Rubenstein et al. (2014). Furthermore, we remind the reader that here we are dealing with robots that should move in space such that the swarm arranges into a desired spatial configuration. This is different in nature from altering the internal states of the robots into a particular repeating pattern, such as in the work of Yamins and Nagpal (2008).

4.5.2. PAGERANK MODEL

To define the PageRank framework for this task, we need to define graphs $G^a_{\mathcal{S}}$ and $G^p_{\mathcal{S}}$. This can be readily done based on the graphs $G^1_{\mathcal{S}}$, $G^2_{\mathcal{S}}$ and $G^3_{\mathcal{S}}$, previously introduced. For $G^a_{\mathcal{S}}$, the graph holds the local transitions that a robot will experience based on actions it takes from a stochastic policy. This is exactly the same as $G^1_{\mathcal{S}}$, therefore:

$$G^a_{\mathcal{S}} = G^1_{\mathcal{S}}.$$

For $G^p_{\mathcal{S}}$, the graph holds all the local transitions that happen whenever the environment causes a state transition (i.e., other robots take an action). This can be the result of actions by neighbors, as described by $G^2_{\mathcal{S}}$, or actions of other robots which become neighbors, as described by $G^3_{\mathcal{S}}$. Therefore, it follows that:

$$G^p_{\mathcal{S}} = G^2_{\mathcal{S}} \cup G^3_{\mathcal{S}}.$$

The matrices **H**, **E**, and **D** are extracted from $G^a_{\mathcal{S}}$ and $G^p_{\mathcal{S}}$ using Equation 4.4, Equation 4.5, and Equation 4.6, respectively.

Concerning α, we follow the same definition as for the consensus task (see Equation 4.9), whereby we define it based on the number of neighbors that a robot senses in a given state. This models how, with more neighbors, the robot is more likely to be subject to its environment. We once again have $0 \leq p_{action}(s) \leq 1$, where $p_{action}(s)$ is the cumulative probability of taking an action when in state s. If $p_{action}(s) < 1$, then there is also a probability that the robot will not take an action and that it will remain idle. In the example shown in Figure 4.11a, for instance, we would have $p_{action}(s_i) = p_l + p_r \leq 1$. Here, p_l is the probability of taking an action to the left, and p_r is the probability of taking an action to the right, according to the policy that is being used.

4.5.3. OPTIMIZATION STRATEGY

For this task, we wish to take care that the optimization procedure does not violate the conditions that tell us that the pattern will be formed. For this reason, we have divided the optimization process in two phases. In the first phase, we only perform the removal of state-action pairs from Π_0 while keeping the previoulsy stated verification conditions as constraints. From this phase, we will extract a policy $\Pi_1 \subseteq \Pi_0$. After this is done, we remove the constraints and freely alter the probability of taking actions in Π_1, except that we always keep a nonzero probability for all state-action pairs. We would like to make it clear to the reader that Phase 1 is not required, and it is possible to directly go to Phase 2 (we provide an example of that in Section 4.5.7). However, Phase 1 allows us to quickly

reduce the solution space prior to Phase 2, while still ensuring that the pattern of interest can be formed. In both cases, we use the PageRank-based fitness function proposed in Equation 4.7.

- **Phase 1: State-action pair removal from Π_0**
  In Phase 1, state-action pairs are eliminated from the stochastic policy Π_0 with the goal of maximizing F. The input to this phase is the baseline policy that is extracted following the methods described in Section 4.5.1. The output of this first phase is a stochastic policy $\Pi_1 \subseteq \Pi_0$, whereby state-action pairs in Π_0 that do not help to achieve the pattern efficiently are automatically removed, while maximizing F as per Equation 4.7. The optimization of Phase 1 is subject to the following two constraints:

  1. *The verification conditions must be respected.*
     This constraint checks that the pattern will always eventually be formed from any initial configuration. The conditions can be checked based from the local graphs $G^1_{\mathcal{S}}$, $G^2_{\mathcal{S}}$, and $G^3_{\mathcal{S}}$, and therefore the time required to check them scales with the local state space size, and not the size of the swarm.
  2. *The final pattern must remain the unique emergent pattern.*
     As state-action pairs are removed from Π_0, it may be that additional states behave like states in $\mathcal{S}_{static}$ and will not move. However, an important axiom needed to guarantee that P_{des} will always form is that, for a swarm of N robots, N instances of the local states in $\mathcal{S}_{static}$, with repetition, must uniquely rearrange into P_{des}. If this is not the case, another pattern could emerge where all robots are in a state $\mathcal{S}_{static}$ and do not move. It must therefore be verified that Π_0 is not affected in such a way that this can happen. The method to verify this is described at the end of this subsection.

- **Phase 2: Probability optimization**
  In Phase 2 the probability of executing individual the state-action pairs in Π_1 is altered so as to maximize the fitness F. This phase parallels the final optimization step of Coppola and de Croon (2018), with the key difference being that we now evaluate the performance of swarm using the PageRank-based fitness function in Equation 4.7, rather than by simulating the swarm. The output of this second phase is a stochastic policy Π_2.

Procedure to check the first constraint of Phase 1: Because we are already at the optimization stage, we consider a starting point whereby the original $\mathcal{S}_{static}$ already guarantees that P_{des} is unique. We then only need to check that adding new states to $\mathcal{S}_{static}$ does not affect this property, which we can do at the local level. Consider a state $s \in \mathcal{S}_{active}$ which has become a candidate to be moved to $\mathcal{S}_{static}$. For s, we locally check whether it could be fully surrounded by robots with a state within $\mathcal{S}_{static}$. If this is not possible, because s is such that at least one of its neighbors would be in an active state, then we can add s to $\mathcal{S}_{static}$. This is because we know that, if this state s were static, there would always be one active robot somewhere next to it anyway, so P_{des} still remains the

Table 4.3 Genome size during Phase 1 for the four patterns tested. The genomes size is equal to the size of the original policy Π_0 extracted as described in Section 4.5.1. Following Phase 1, which could remove state-action pairs, the size of the policy could be reduced. These smaller policies can then be optimized in Phase 2.

Pattern	Triangle (four robots)	Hexagon	Triangle (nine robots)	T (12 robots)
$\lvert\Pi_0\rvert$	187	486	531	525
$\lvert\Pi_1\rvert$	66	174	176	276

only unique pattern that can be formed by static states. Furthermore, when this active neighbor moves, the local state of the robot will also change and it will also no longer be static. Note that, because of the importance of states in the $\mathcal{S}_{simplicial}$, these states are not allowed to transition into the set $\mathcal{S}_{static}$.

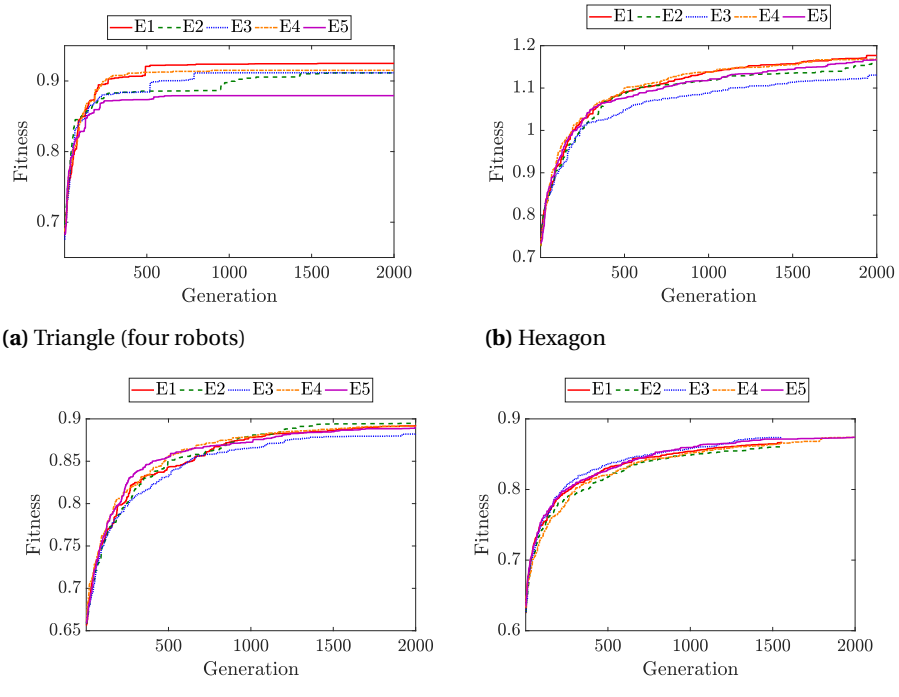
4.5.4. PHASE 1: GENETIC ALGORITHM SETUP AND RESULTS

For Phase 1, we set up a GA with a population of ten binary genomes. Each gene in a genome represented a state-action pair in Π_0, with a 1 indicating that the state-action pair was kept and a 0 indicating that it was eliminated. Each generation consisted of: elite members (30%), new offspring (40%), and mutated members (30%). Offspring genomes were the result of an `AND` operation between two parent genomes. Offspring were only kept if they complied with the constraints, else the parents were forced to look for new mates. On each generation round, mutation was applied to a random portion of the population, for which the `NOT` operator was randomly applied to 10% of each selected member's genome (thus changing ones to zeros and vice versa). Similarly as to the offspring, a mutation was kept if and only if it returned a genome for which the constraints were met, else it was discarded and a new mutation was attempted until a successful one was found. This way it was guaranteed that the population always consisted of genomes that respected the constraints. Each valid genome was evaluated assigned a fitness F to as per Equation 4.7. The genomes of the initial population were generated by performing the mutation procedure on a fully positive genome (all 1s). The genomes size for the four patterns was equal to $|\Pi_0|$ for each pattern, as indicated in Table 4.3.

With this setup, five evolutionary runs were performed for the first four patterns of Figure 4.9.[10] The results of the evolutionary runs are shown in Figure 4.12. Within 2000 generations, the fitness of all patterns approached convergence for all evolutionary runs, characterized by a steep increase in the first generations.

The performance of all the evolved policies was evaluated in simulation. The simulator was a direct implementation of the swarm on a grid as described in Section 4.5.1. Each policy was evaluated 100 times with the swarm starting from random initial configurations, with the only requirement that they would begin in a connected topology. The global metric of performance was the amount of actions taken cumulatively by all robots before the desired final pattern emerged. This was compared to the original unoptimized policy Π_0 of each pattern. The results are shown as histograms in Figure 4.13.

[10] As explained in Section 4.5.1, the line does not pass the verification procedure. It thus cannot be optimized in Phase 1. For this reason, it will be directly optimized using the GA of Phase 2, which does not directly enforce any constraints. The results for the line are shown separately in Section 4.5.6.



(a) Triangle (four robots)

(b) Hexagon

(c) Triangle (nine robots)

(d) T with 12 robots

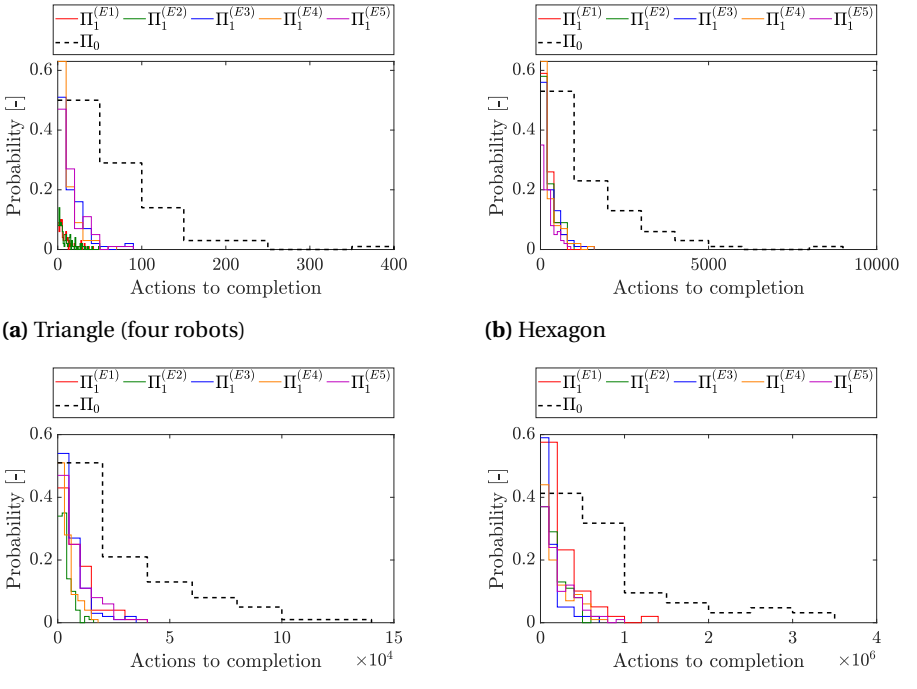
Figure 4.12 Phase 1 evolutions for all patterns tested, showing the best fitness from each of the five evolutionary runs

It can be seen that all policies are able to achieve the pattern and significantly outperform the original unoptimized policy by approximately one order of magnitude, showing a successful application of PageRank as an optimization parameter. The difference of the evolved performance from the original was confirmed by a bootstrap test with 10,000 repetitions, using the sample mean and the ratio of the variances as the reference statistics. All distributions were declared different over the 95% confidence interval. We now continue to Phase 2 in order to further improve the performance of the swarm.

4.5.5. PHASE 2: GENETIC ALGORITHM SETUP AND RESULTS

Phase 1 can achieve a stochastic policy $\Pi_1 \subseteq \Pi_0$ which can be seen to provide a global performance that is one order of magnitude faster than Π_0. However, Π_1 still dictates that, when in a given state, a robot chooses from the available actions using a uniform distribution. This limitation is surpassed in Phase 2. The objective of Phase 2 is to optimize the probability of executing the state-action pairs in the policy, as was the case for the GA of the consensus task. The GA setup of Phase 2 is similar to the one of Phase 1, with two main differences:

1. The GA optimizes the probability of executing the state-action pairs from a stochas-



(a) Triangle (four robots)

(b) Hexagon

(c) Triangle (nine robots)

(d) T with 12 robots

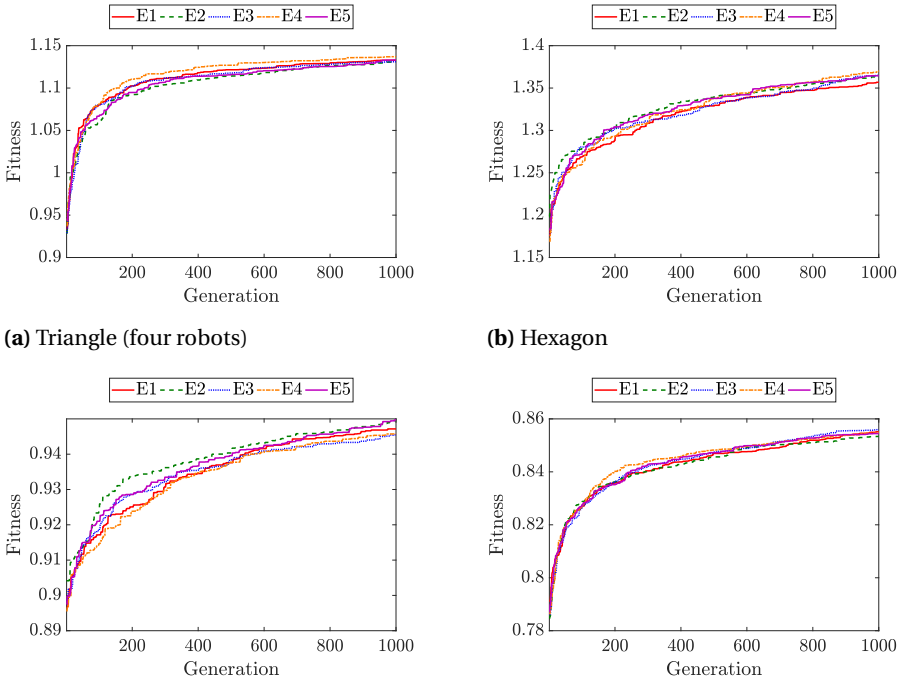
Figure 4.13 Normalized histograms showing the cumulative number of actions required before the global goal is achieved for the evolved policies $\Pi_1 \subseteq \Pi_0$, extracted from the five evolutionary runs of Phase 1 (colored lines), compared to a baseline unoptimized performance of Π_0 for each pattern (black line).

tic policy Π_1, rather than determining which state-action pairs could be removed.

2. The constraints are no longer enforced (but, as none of the state-action pairs are removed, the provability is preserved).

For Phase 2, we set up a GA with a population of 20 scalar genomes. Each gene in a genome holds a value $0 < p \leq 1$, indicating the probability of executing the corresponding state-action in Π_1. Naturally, for each state, the cumulative probability of taking one of the actions in Π_1, denoted $p_{action}(s)$, cannot be larger than 1. If this happens, then the probabilities of the relevant state-action pairs in the genome are appropriately normalized. Each new generation was produced by elite members (30%), offspring (40%), and mutated members (30%). Offspring were generated from averaging two parents' genomes. Mutation was applied to random genomes, for which 10% of their genes were replaced by random values of $0 < p \leq 1$ from a uniform distribution. The members of the initial population were produced by applying the mutation procedure above to a genome with uniform probability distributions, as extracted from Phase 1. The genome size is detailed in Table 4.3.

Five evolutionary runs were conducted for each pattern for 1000 generations. The policy to be optimized was the best performing policy from Phase 1, for each pattern,



(a) Triangle (four robots)

(b) Hexagon

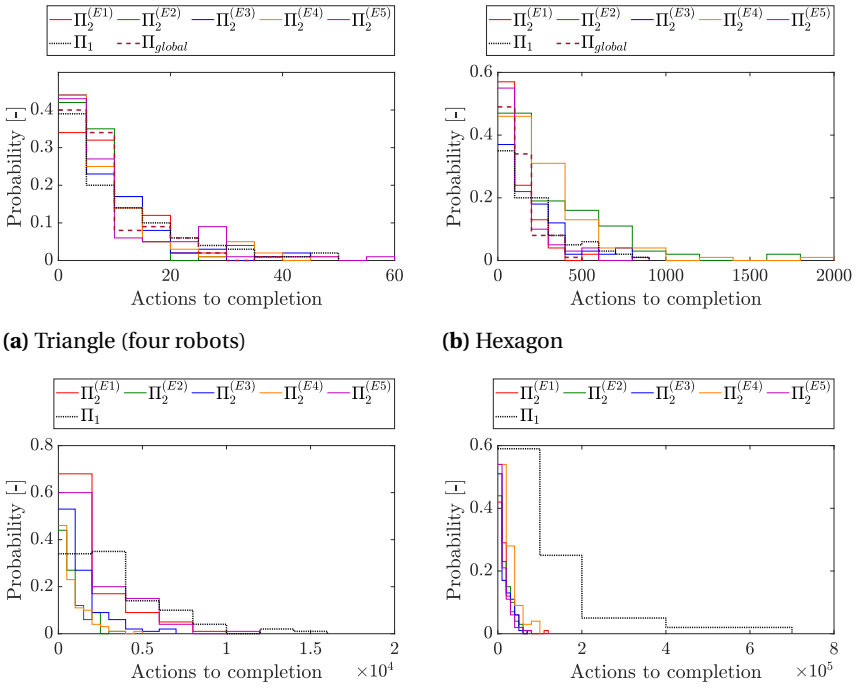
(c) Triangle (nine robots)

(d) T with 12 robots

Figure 4.14 Phase 2 evolutions for all patterns tested, showing the best fitness from each of the five evolutionary runs.

based on the results of Phase 1 shown in Figure 4.13. The results of the evolution of Phase 2 are shown in Figure 4.14. The global performance of the final policies Π_2 was evaluated 100 times as in Phase 1. Once again, we measure the global performance by the cumulative number of actions that the robots in the swarm take before the desired pattern emerges. The results are shown in Figure 4.15, where they are compared to the best policy obtained at the end Phase 1 for each pattern. For the triangle with four robots and the hexagon, we also compare these results to the performance of the optimized behavior from Coppola and de Croon (2018), which used simulation in order to directly optimize for the cumulative number of actions taken by the swarm. Sample trajectories of simulations are shown in Figure 4.16.

The improvements achieved for the triangle with four robots and the hexagon appear marginal. For two policies evolved for triangle with four robots, a bootstrap test (based on the difference in sample means) could not reject the hypothesis that the distribution of the final performance is equal to the performance at the end of Phase 1. This was the case for also one of the policies evolved for the hexagon. However, this is a justified result, because for both the triangle with four robots and the hexagon, the performance after Phase 1 already approached the performance of a policy which was optimized with the fitness being evaluated through simulations, which we take to be a near-limit perfor-



(a) Triangle (four robots)

(b) Hexagon

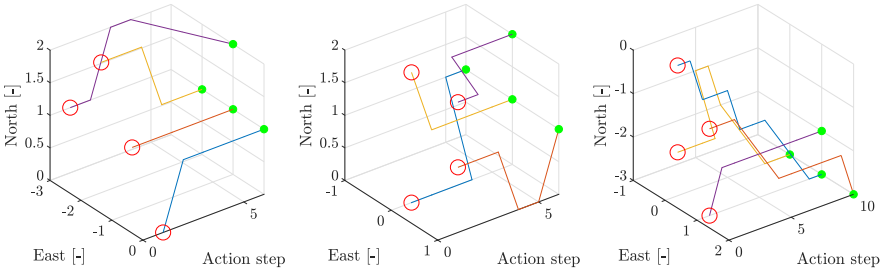
(c) Triangle (nine robots)

(d) T with 12 robots

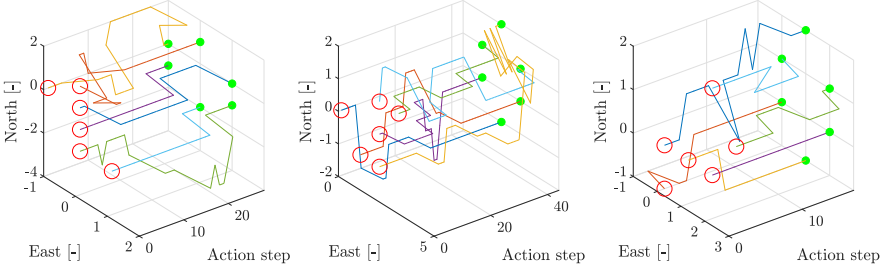
Figure 4.15 Normalized histograms showing the cumulative number of actions required before the global goal is achieved for the evolved policies from Phase 2, in comparison with the best performing policy of Π_1, and the globally optimized performance from Coppola and de Croon (2018). The latter is available only for the first two cases, the triangle with four robots and the hexagon, as extracting it for larger patterns leads to scalability issues).

mance. The evolutions of Phase 2 achieved policies that performed equivalently and, in some cases, seemingly even slightly better than the ones that were achieved in Coppola and de Croon (2018). In Figure 4.15 it can be seen that a higher fraction of runs take fewer than ten actions and fewer than 100 actions for the triangle with four robots and the hexagon, respectively. These results show the power of PageRank's framework in being able to describe the swarm up to the point that the achieved performance matches the performance of results that are achieved with a global evaluation procedure.

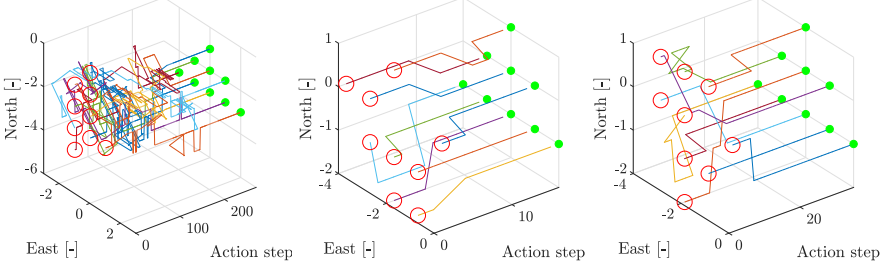
For the triangle with nine robots and the T with 12 robots, Phase 2 instead shows significant improvements over Phase 1. These are most noticeable for the T with 12 robots, which reaches an improvement in performance of one order of magnitude compared to the policies from Phase 1. Overall, this means that the policies following Phase 2 outperform the unoptimized policy by two orders of magnitude. We remind the reader that both of these patterns could never have been tackled if, instead of using PageRank, we had used simulation in order to assess the performance of the swarm. Simulations would have been infeasible in light of scalability issues, as was indeed the case in our previous



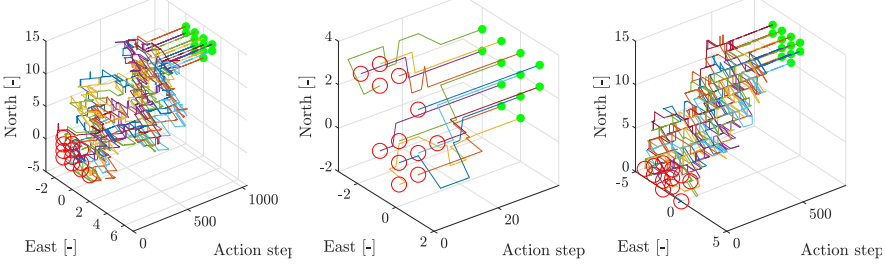
(a) Triangle (four robots)



(b) Hexagon



(c) Triangle (nine robots)



(d) T with 12 robots

Figure 4.16 Exemplary simulations showing the evolution of the swarm with policy Π_2 for each pattern studied, given random initial conditions. The red dots indicate starting positions, and the green dots indicate final positions.

work (Coppola and de Croon, 2018). Instead, these results have been achieved with the use of an entirely local evaluation.

4.5.6. ANALYSIS

The results of Phase 1 and Phase 2 both show that the evolutions successfully lead to policies with high performance boosts, which even match the performance of policies that were optimized using a fitness function based on a global simulation of the swarm. In this section we shall now aim to gain more insight into how the PageRank-based fitness function was correlated with the global performance of the swarm. As we are now dealing with a large policy with a large number of parameters (see Table 4.3), the entire fitness landscape cannot be studied. Instead, we evaluated the global performance at various stages of the Phase 1 and Phase 2 evolutions to show how the performance improvements are correlated with the PageRank-based fitness.

For Phase 1, the results of this evaluation are shown in Figure 4.17. It can be seen that the performance of the swarm improves quickly in the beginning. This matches the fast rise in fitness observed in Figure 4.12. Moreover, for the triangle with four robots and the hexagon, we see that the performance even begins to approach the one of the final global optimization from Coppola and de Croon (2018). Overall, Phase 1 shows a successful correlation between the PageRank-based fitness function and the global performance of the swarm.[11] For Phase 2, the results of the same analysis for the triangle with four robots, the hexagon, and the triangle with nine robots are shown in Figure 4.18. Once again, it is possible to observe a general overall improvement in the performance. In all cases, over the first generations, the performance of the swarm improves.

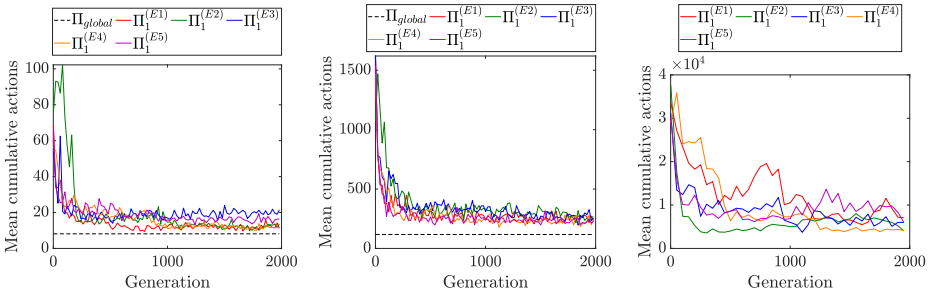
It is in Phase 2 that, however, we also notice certain limitations. It appears that, beyond a certain point, some evolutionary runs begin performing worse as the evolution continues, which is not reflected in the fitness function. The effect is most clearly noticeable on:

- the hexagon, starting from around the 100th generation, for all runs.
- the triangle with nine robots, where one of the evolutionary runs features a drastic spike in performance at approximately the 400th generation (after which it seems to recover).

These effects are likely the result of reaching a limitation of PageRank's microscopic model once the constraints from Phase 1 are removed. Eventually, there comes a point where the evolution over-fits to the microscopic model at the expense of the global performance. There are also a number of other effects at play:

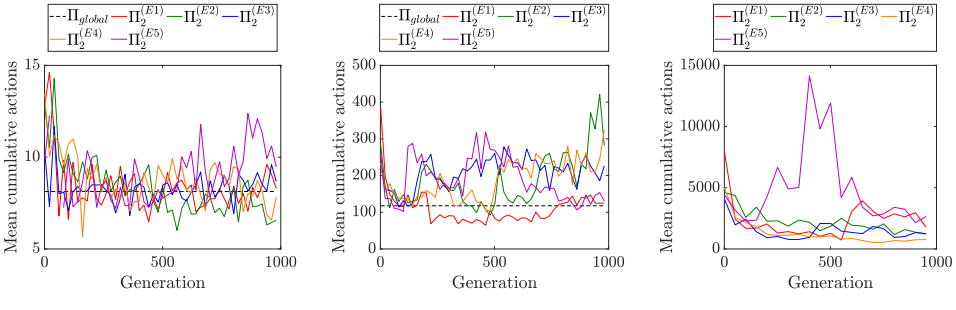
1. The change in fitness is low compared to Phase 1 evolutions; hence, the effects of noise are more clearly noticeable.

[11] Unfortunately, we cannot show this analysis for the T with 12 robots. This is because evaluating early generations in simulation is too computationally expensive. This serves as another reminder of how infeasible it would have been to try and optimize the behavior of the swarm using simulations to assess the fitness. Only the results before and after the evolutions have been evaluated, which can be appreciated in Figure 4.13d and Figure 4.15d.



(a) Triangle (four robots) **(b)** Hexagon **(c)** Triangle (nine robots)

Figure 4.17 Evaluation of the average performance improvement during Phase 1 for all evolutionary runs. The performance is measured as the average number of actions required cumulatively by the swarm in order to achieve the pattern. This is based on 100 simulations for the highest fitness at each generation, sampled every 20 generations for the triangle with four robots and the hexagon, and every 50 generations for the triangle with nine robots. The dotted line shows the mean performance of the policy achieved by a global optimization in Coppola and de Croon (2018), only available for the first two patterns.



(a) Triangle (four robots) **(b)** Hexagon **(c)** Triangle (nine robots)

Figure 4.18 Evaluation of performance improvement during Phase 2 for all evolutionary runs. The performance is measured as the average number of actions required cumulatively by the swarm in order to achieve the pattern. This is based on 100 simulations for the highest fitness at each generation, sampled every 20 generations for the triangle with four robots and the hexagon, and every 50 generations for the triangle with nine robots. The dotted line shows the mean performance of the policy achieved by a global optimization in Coppola and de Croon (2018), only available for the first two patterns.

2. The impact of changing probabilities in a stochastic policy is less direct than completely removing a state-action pair from the policy. Therefore, the changes have less impact on the results.

4.5.7. FURTHER INVESTIGATIONS

We conclude this section with a further investigation, whereby we explore whether placing more importance on certain desired states can be beneficial or destructive to the pattern formation task. This is relevant to patterns that feature a repeating desired state.

Consider, for example, the line along northeast as shown in Figure 4.9. The line only requires three states in the set $\mathcal{S}_{des}$ (bottom edge, middle piece, and top edge), yet can be generated with any number of robots by nature of a repeating middle state. We thus investigated whether, for long lines, placing more importance on the repeating middle state would improve the performance of the optimized policy. This was done with an altered version of the fitness function which calculated a weighted average of the desired states, rather than a normal average, as in Equation 4.10.

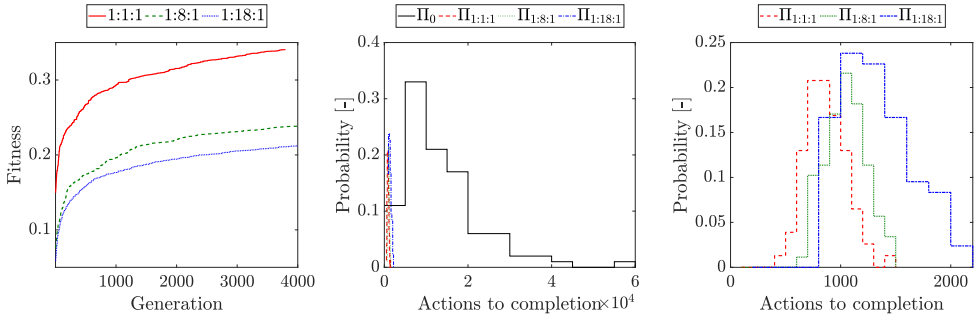
$$F_w = \frac{\sum_{s\in\mathcal{S}_{des}} w(s)\cdot R(s)/|\mathcal{S}_{des}|}{\sum_{s\in\mathcal{S}} R(s)/|\mathcal{S}|}, \tag{4.10}$$

where $w(s)$ indicates the importance of a state $s \in \mathcal{S}_{des}$. We performed three different optimization runs with weight ratios 1:1:1, 1:8:1, and 1:18:1, relating to the weight of the bottom edge, middle piece, and top edge, respectively. For these optimizations, we directly used the optimization setup of Phase 2, with the fitness function of Equation 4.10. The results of the evolution are shown in Figure 4.19a. When the performance of the obtained policies was evaluated for a swarm of 20 robots, it was found that the policy that was optimized with a weight ratio of 1:1:1 featured the best performance, followed by 1:8:1, and finally by 1:18:1. This is shown in and Figure 4.19b and Figure 4.19c.[12] These results lead to the insight that, at least for this type of task, it appears equally important for robots to reach all desired states simultaneously, and preference toward one may be detrimental.

4.6. SAMPLE TASK 3: AGGREGATION

In this task, robots move within a closed arena with the goal of forming aggregates. Each robot can sense the location of neighbors within a range. This task is similar in nature to that of Correll and Martinoli (2011). Using PageRank, we optimize a stochastic policy that tells a robot whether it should perform a random walk or remain idle based on the number of neighbors that it can sense. The final policy is optimized without taking into account the lower-level implementation of the random walk, the number of robots in the swarm, the size of the environment, or the initial condition. Applying the framework to optimize a higher-level policy such as this one provides further insight into the applicability and flexibility of the approach.

[12] Note: creating a line along northeast is subject to both spurious patterns and livelocks. These, which happened on occasion, have been removed from the results shown in these figures, as they are outside the scope of this discussion.



(a) Results of evolutions of ≈4000 generations

(b) Performance compared to original unoptimized performance when generating a line with 20 robots

(c) Relative performance between three evolved policies

Figure 4.19 Study on the impact of changing the relative importance of states in $\mathcal{S}_{des}$. Even when a state repeats, the performance remains more optimal when all desired states are weighted equally in the fitness function.

4.6.1. TASK DESCRIPTION

Consider N robots in a closed arena. The robots operate in continuous space (rather than a grid, as in previous tasks), and in continuous time, using local clocks. Each robot can sense the number of neighbors, m, within a limited range $\rho_{\max}$. We assume that each robot is capable of sensing a maximum of M neighbors, after which the sensor saturates. Therefore, the local state space $\mathcal{S}$ for this task is given by all possible numbers of neighbors that can be sensed by the robot, namely $\mathcal{S} = \{0, 1, \ldots, M\}$. The sensor updates its reading with a time interval t_c.

Based on the number of neighbors sensed, a robot can choose to engage in one of the following two sub-behaviors:

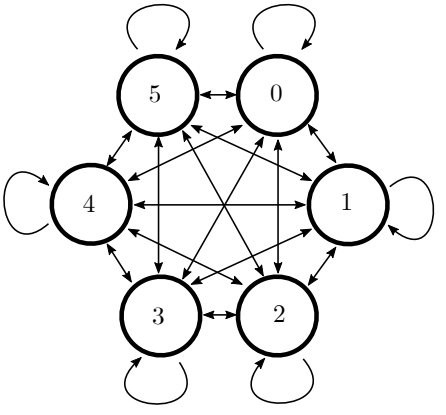
1. Perform/resume a random walk behavior. We denote this action as a_{walk}.
2. Stop and remain idle. We denote this action as a_{idle}.

These form the action space $\mathcal{A} = \{a_{walk}, a_{idle}\}$. The policy Π, to be optimized using the PageRank algorithm, is the stochastic policy mapping the state space $\mathcal{S}$ to the actions space $\mathcal{A}$. The set $\mathcal{S}_{des} = \{m_{\min}, m_{\min}+1, \ldots, M\}$ is the set of desired states, where $m_{\min}$ is the minimum amount of neighbors that we wish for a robot to observe.

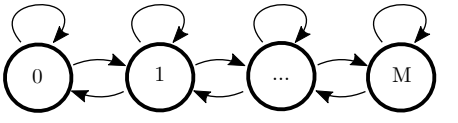
4.6.2. PAGERANK MODEL

To set up the PageRank framework, we begin by defining the active graph $G^a_{\mathcal{S}}$, which describes the local state changes as a result of the actions of a robot.

The graph $G^a_{\mathcal{S}}$ is defined as a fully connected graph between all states, as exemplified in Figure 4.20a for the case where $M = 5$. If the robot remains idle, then it will always keep its number of neighbors (remember that all changes in the environment, including neighbors leaving, are described in the passive graph, $G^p_{\mathcal{S}}$). We thus set the weights for all



(a) Depiction of $G^a_{\mathcal{S}}$ for the sample case where $M = 5$.



(b) Depiction of $G^p_{\mathcal{S}}$ for an arbitrary value of M. All transitions from any node are equally probable. To model random concurrency, it is assumed that two robots will never leave (or join) a neighborhood at exactly the same time.

Figure 4.20 Depictions of $G^a_{\mathcal{S}}$ and $G^p_{\mathcal{S}}$ models for the aggregation task.

transitions to the same state to be equal to $\Pi(s_i, a_{idle})$. All other edges are given weights according to the following:

$$w_{i,m} = \Pi(s_i, a_{walk}) \cdot p(m) \tag{4.11}$$

Where $p(m)$ is the probability of transitioning to a state with m neighbors. In our work we set: $p(m) = 1/(m+1)^2$, albeit we have found that the same final policy is evolved also for other definitions, such as $p(m) = 1/(m+1)$. Therefore, we can see that it is sufficient to coarsely model the trend that the probability of finding larger groups decreases with the size of the group. The swarm designer does not need to concern oneself with the exact specification of how likely it is to encounter larger group sizes, which eases the design task.

Graph $G^p_{\mathcal{S}}$ holds the transitions that a robot can experience regardless of its own choice to be idle or not. In this case, the robot can be in any state $s \in \mathcal{S}$, at which point it could lose a neighbor or gain a neighbor at any point. The graph $G^p_{\mathcal{S}}$ takes the form shown in Figure 4.20b.

As done for the previous tasks, α is set as a function of the total probability of selecting a new action and the number of neighbors, as in Equation 4.9. We also note here, however, that our evolutions returned the same final behavior also for simpler static definitions of α, such as $\alpha = 0.5$, which would simply model that at all times the robot is equally subject to its decisions as it is to its environment. This is likely because the behavior that is being optimized is too high level for such details to make a real difference.

4.6.3. GENETIC ALGORITHM SETUP

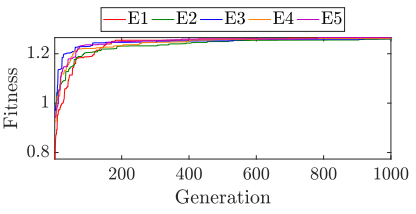
We used the same GA setup as used for the consensus task in order to tune the probabilities of the policy Π for this task. We considered the case where $m_{\min} = 2$, meaning that the robots have the goal of forming aggregates of three or more. Unlike other cases, we did not set the policy values for the states $\mathcal{S}_{des}$ to zero. Note that this is not necessary and actually makes the problem harder to solve, since more freedom is given to the optimizer with more parameters to be optimized. If we had done it, we would have automatically forced a relative basic solution (i.e., move when not in a group) upon the behavior. Thus, we instead took this as an opportunity to see whether PageRank would be able to tune its weights to also inform the robots to remain in the states of $\mathcal{S}_{des}$, without us providing this knowledge a priori.

The results of five evolutionary runs, showing the best genome for each generation, are shown in Figure 4.21a. The evolved behavior was the same in all cases. The behavior we extracted from one evolutionary run is shown in Figure 4.21b. Here we can clearly see that the final evolved behavior is simple, yet very sensible. The extracted policy approaches a deterministic policy whereby if the number of neighbors is less than $m_{\min} = 2$, then the robot always chooses to move and explore the environment in the hope of finding more neighbors. If the number of neighbors is sufficient, then the robot remains idle.

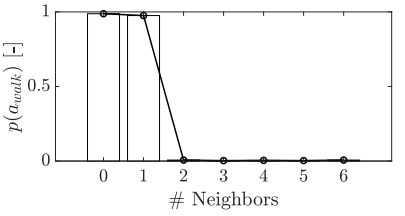
4.6.4. SIMULATION ENVIRONMENT AND EVALUATION RESULTS

The performance of the evolved behavior was tested using a simulation environment called *Swarmulator*. Swarmulator is a C++ based simulator designed to efficiently simulate and prototype the behavior of swarms in continuous time and space. Each robot (together with its dynamics) is simulated and controlled by an independent detached thread, which limits simulation artifacts which may otherwise stem, for instance, from the robots being simulated always in the same order. Swarmulator is described in more detail in Chapter A. In our simulations, the robots had the same dynamics as accelerated particles, allowing them to move omnidirectionally. The details of the simulated behavior were the following:

- When initiating a random walk, a robot could randomly select a direction θ from a uniform distribution [0, 2π] and pursue it with a reference speed v. The decision to continue the random walk or stop, commanded by the high level policy, was reassessed with an interval t_c. If the robot chose to remain in a random walk, then it altered its current direction θ by an angle θ_d. The angle θ_d was extracted from a normal distribution with zero mean and a standard deviation σ_d. In our implementation, we set $v = 0.5\ m/s$, $t_c = 2\ s$, and $\sigma_\theta = 0.3\ rad$. The robots could sense their neighbors omnidirectionally up to a distance $\rho_{\max} = 2\ m$.
- If a robot was at a distance d_w from a wall, it would set a velocity command away from the wall for a time t_w. In our implementation, we set $d_w = \rho = 2\ m$ and $t_w = 2t_c$. Setting $d_w = \rho_{\max}$ simulates the fact that the same sensor is used to sense both neighbors and the walls.
- At all times, in direct addition to the velocity commands from the random walk behavior and wall avoidance, the robots were governed by attraction and repulsion forces. This handled collision avoidance. In our implementation, the attrac-



(a) Fitness over five evolutionary runs, showing the best fitness for each run.



(b) A final evolved policy, showing the probability of moving with respect to the number of neighbors that are sensed. The robots almost always move if the number of neighbors found is less than the desired amount, else they always remain idle.

Figure 4.21 Details of the evolution of the policy for the aggregation task, showing the increase in fitness a final evolved policy.

tion and repulsion behavior was tuned such that the equilibrium distance between robots was 1.5 m.

The system above was tested 50 times for a swarm of 20 robots in a 30 $m \times 30$ m square arena. In all cases, the swarm successfully forms aggregates of three or more robots within the arena. Two sample runs are shown in Figure 4.22.

4.6.5. ANALYSIS

As for the variant of the consensus task from Section 4.4.4 and Section 4.4.5, we took advantage of the small size of this policy in order to map and investigate the fitness landscape. The investigation was focused on testing the impact of the probability of moving within the arena when the robots did not have any neighbors and when the robot had one neighbor. Since the robots had the goal of being with two or more neighbors, these were the two main parameters of interest.

The simulation environment from Section 4.6.4 was used to evaluate the performance of the swarm against different policies, in order to construct the fitness landscape with respect to our global parameter of interest. The results were obtained by averaging 50 simulations for each setting, and repeated for swarms of 20 robots and 50 robots in a 30 $m \times 30$ m arena. The maximum simulated time allowed for the task was set to 1000 s. The results are shown in Figure 4.23a and Figure 4.23c, respectively.

This fitness landscape, which is the global metric that we wanted to optimize for in reality, is compared to the fitness landscape shaped by the PageRank-based fitness function with which the behavior was actually evolved. The fitness landscape, shown in Figure 4.23c, has a clear correlation with the global performance landscapes shown in the remainder of Figure 4.23. The result of this analysis show that the behavior that was evolved approaches an optimum that is also found at the global level.

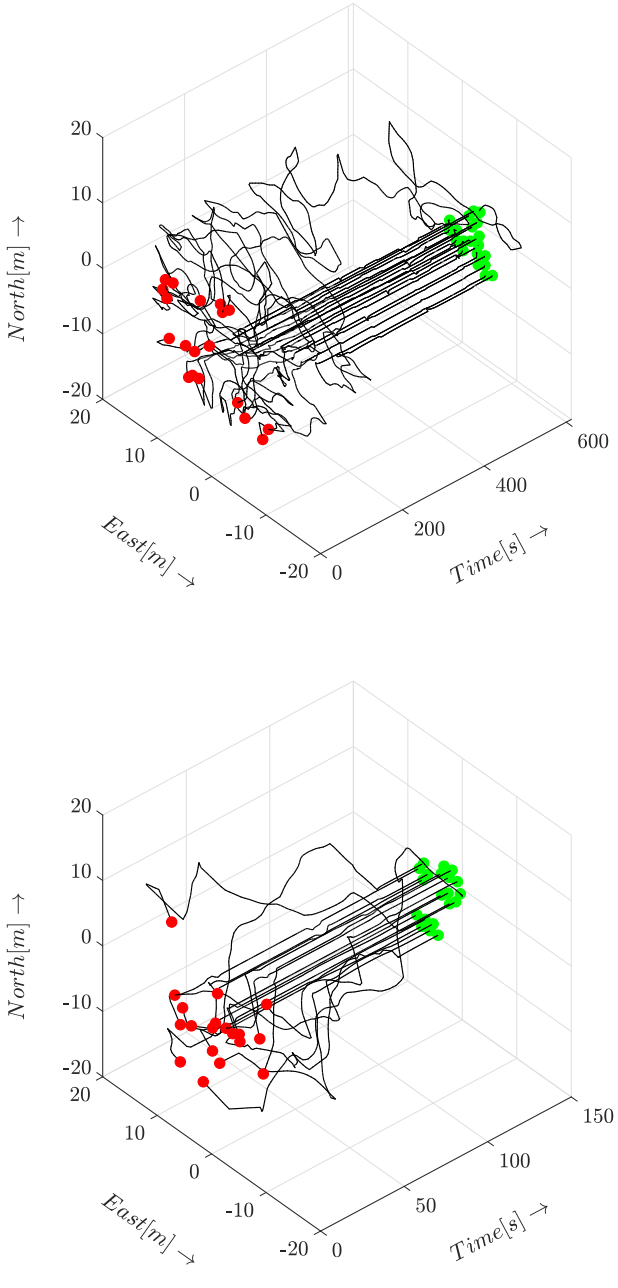
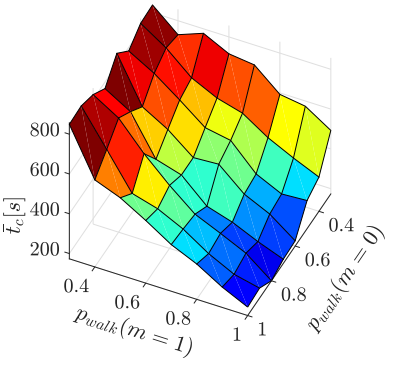
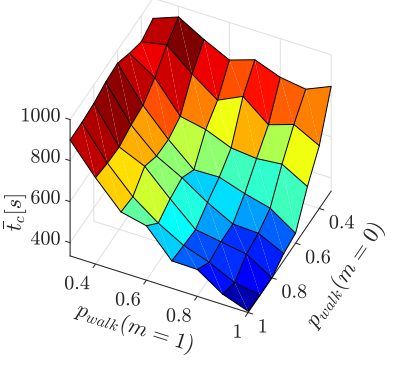


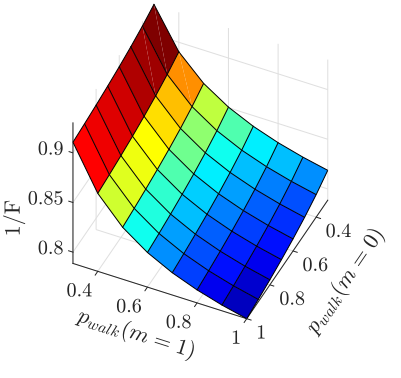
Figure 4.22 Depiction of two simulations of the aggregation task using the evolved policy together with the simulation environment described in Section 4.6.4. On the top figure, the robots assemble into one cluster. On the bottom figure, the robots assemble into multiple clusters. Multiple clusters may happen due to the low sensing range of the robots in comparison with the size of the arena (Correll and Martinoli, 2011).



(a) Global performance of the solution space for a swarm of 20 robots in a 30 $m \times 30$ m arena. $\bar{t}_c$ is the mean time taken before the task is completed.



(b) Global performance of the solution space for a swarm of 50 robots in a 30 $m \times 30$ m arena. $\bar{t}_c$ is the mean time taken before the task is completed.



(c) PageRank-based fitness landscape, where $1/F$ is the inverse of the fitness evaluated through Equation 4.7.

Figure 4.23 Global mapping of the solution space for the aggregation policy for different probabilities of motion with zero and one neighbor(s). $p_{walk}(m)$ is the probability of pursuing a random walk when surrounded by m neighbors.

4.7. DISCUSSION

In Section 4.5, Section 4.4, and Section 4.6 the PageRank framework has been applied to optimize the stochastic policy of robots in a swarm for three separate tasks. With the knowledge gained, this section summarizes the advantages of the proposed approach (Section 4.7.1), its current limitations, and possible solutions (Section 4.7.2) and proposes research directions that could lead to interesting extensions (Section 4.7.3).

4.7.1. ADVANTAGES AND PROPERTIES OF THE PROPOSED APPROACH

The primary motivation for the development of the approach proposed in this chapter was to find a method for which the time required to evaluate a behavior does not scale with the size of the swarm. Taking as example the tasks tested in this chapter, if the fitness of the controller were to have been tested in simulation, then the time to be evaluated (wall-clock time) would have been dependent on a number of parameters, such as:

- The number of robots in the swarm, which affects both the processing time required to simulate it and the time before which the global goal is achieved.
- The size of the environment, which would alter the rate at which events occur. This would have been relevant for a task such as the aggregation task.

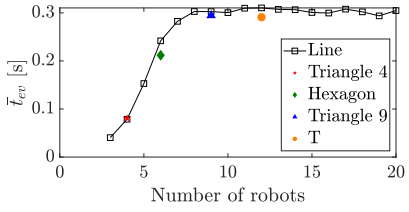
Furthermore, the final evolved solution would have been fitted to the following:

- The initial conditions from which the swarm is simulated. To avoid over fitting to a particular initial condition, each controller would have had to be tested from several random initial conditions.
- The number of robots used in the simulations. To avoid over fitting to swarms of a specific size, each controller would have had to be tested on swarms of different sizes.
- The global environment. If evolving within a specific environment, the final solution may implicitly be fitted to it.

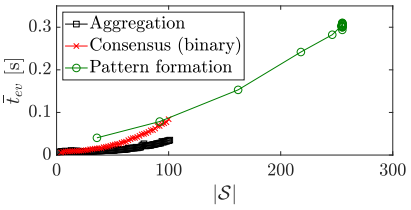
Certain methods may help to mitigate the issues above when simulation is used, such as evolving for fewer robots and then testing on larger swarms (Trianni et al., 2006), or capping the simulation time to a maximum value (Scheper and de Croon, 2016). Nevertheless, such solutions present new challenges of their own. The first, for instance, does not guarantee that the controller will be scalable. The latter, on the other hand, may limit the ability of the evolutionary algorithm to properly search the solution space. If the time limit is too low, then few genomes, if any, will succeed in the goal. Moreover, this means that the fitness is no longer assessed by whether, and how efficiently, the goal is achieved, but must be assessed based on how much of the goal has been achieved. This may require a different fitness function, a task which can likely cause additional difficulties and may even shape the final outcome (Nelson et al., 2009).

Using PageRank, the evaluation of the fitness function is based only on parameters extracted from a microscopic model. This has the following beneficial effects, which can be found back in the results of this chapter:

- **The wall-clock time does not scale with the size of the swarm.** This can be appreciated directly from the results in this manuscript, such as for the pattern formation task, where we have managed to optimize the policy for patterns that we could not tackle using simulation to evaluate the fitness (Coppola and de Croon, 2018). We have plotted the wall-clock time required to evaluate the PageRank-based fitness for the pattern formation task in Figure 4.24a. Each data point is averaged over 100 evaluations of PageRank for random instances of a stochastic policy. It can be seen that, once the maximum neighborhood size of eight is fulfilled, the wall-clock time stops scaling and it remains constant even if the number of robots in the swarm increases. Instead of scaling with global parameters, the wall-clock time scales with the number of local states $\mathcal{S}$ and the possible transitions between them, as modeled by $G^a_{\mathcal{S}}$ and $G^p_{\mathcal{S}}$. To test the practical impact of this, we evaluated the wall-clock time of the PageRank-based fitness function when varying the size of $\mathcal{S}$ (which also bring about more state transitions) for the separate tasks tested in this chapter, averaged over 100 random policy iterations. The results are shown in Figure 4.24b. It can be seen that the wall-clock time grows with the size of $\mathcal{S}$. In comparison, the fitness wall-clock time for simulations directly grows with the size of the swarm. This is shown for the pattern formation task from Figure 4.25. Note that this is despite using the simple grid simulation environment as was used for the simulations in Section 4.5.4, Section 4.5.5, and Section 4.5.6. The performance of higher fidelity simulators would be worse.
- **The evolved policy scales well with the number of robots in the swarm.** The policy is evaluated from a graph for which the number of robots in the swarm is not modeled, only the probability that certain events in the neighborhood of a specific robot may take place. For example, for the consensus task, the same exact policy could be applied to swarms of different sizes (we tested with swarms of ten and 20 robots), from random initial configurations, and consistently provided efficient results orders of magnitude better than the baseline case.
- **The evolved policy is not tuned to any particular initial condition of the swarm.** The proposed PageRank-based fitness function (Equation 4.7) evaluates the likelihood of achieving certain local states. However, as PageRank is evaluated for the entire graph simultaneously using the iterative procedure described in Section 4.3, the score does not assume any initial condition by the robot, and, by extension, the swarm. This makes it robust to the initial conditions. In all results shown in this chapter, there was no initial condition for which the evolved solution made it impossible to achieve the final global goal or for which the performance did not improve.
- **The evolved policy is not tuned to any particular lower-level controller.** In all instances, a policy was evolved without any specification of lower-level parameters. This can be best appreciated for the aggregation task. Here, we evolved a high level policy without specifying the type of random walk that was going to be implemented, but only specifying the relative probability of encountering neighbors when a robot is moving. Had such a behavior been tuned in simulation, then it would have inherently adapted to the particular implementation of search behavior (e.g., wall-following, Lévy Flight, etc.), whereas this is not the case for our



(a) Change in wall-clock time with the number of robots for the pattern formation task. Once the swarm reaches nine or more robots (such that the local neighborhood of a robot could be entirely filled with eight other robots) the wall-clock time does not scale.



(b) Change in wall-clock time as the size of the local state space increases

Figure 4.24 Trends in wall-clock time of PageRank-based fitness evaluation, denoted $t_e v$. When using the PageRank centrality measure, the wall-clock time scales with the size of the local state space. $\bar{t}_{ev}$ denotes the average wall-clock time over 100 random policy evaluations. These have been computed using a Matlab script on a Dell Latitude 7490 laptop with Intel®Core™i7-6600U CPU @ 2.60GHz × 4, Intel®HD Graphics 520 (Skylake GT2), 8GB RAM, equipped with SSD, and running Ubuntu 16.04.

results, providing a more general controller. The relative probability of encountering neighbors subsumes a larger set of specific lower-level behaviors. Similarly, for the consensus task and the pattern formation task, the behavior is evolved without any particular specification of communication time, decision time, or motion controller. A simulation-based optimization might have adapted the probabilities to implicitly account for these specifications. Instead, the only concern for the PageRank-based optimization is with which probability it should transition between states.

- **The evolved policy is not tuned to any particular global environment.** PageRank's framework only models the possible transitions that may happen in the environment, but not the entire environment itself. Therefore, as long as the possible local transitions that may take place in a global environment are sufficiently well represented, then the evolved policy remains robust to different environments. In some instances, the global environment may also be excluded altogether. This was the case for the consensus task and the pattern formation task.

4.7.2. CURRENT LIMITATIONS AND POTENTIAL SOLUTIONS

We have identified a set of limitations that pertain to the current implementation of the proposed approach. These are discussed below. When relevant, possible solutions or extensions are proposed.

- **Discrete state space and discrete action space.** The current version of the approach relies on a discrete set of local states and a discrete set of actions. This may be limiting for applications with continuous states. The limitation comes from the fact that we are using a tabular policy which maps discrete states to discrete

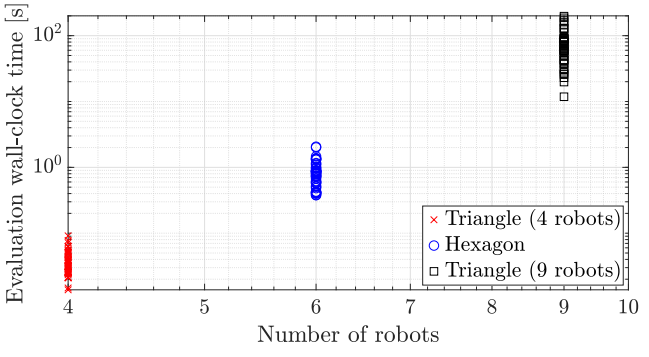


Figure 4.25 Trends in wall-clock time to evaluate the performance of a policy in a simulation environment for the pattern formation task, evaluated for 50 randomly generated policies for which the wall-clock time was averaged over ten iterations. Note that the plot features a log scale on both axes. The simulator used was a simple grid simulation implemented in Matlab. The machine details are the same as used for the results in Figure 4.24.

actions, which then allows us to construct a graph with discrete nodes and edges. Although this is a limitation of the current form, future work can explore the possibilities of extending the approach to deal with continuous state spaces. A research direction that would be interesting to follow is the possibility to use basis functions to interpolate over continuous state spaces. This type of solution has been proposed in the past for other approaches that also natively rely on a tabular policy, such as Q-learning, in order to accommodate continuous state spaces and/or action spaces (Busoniu et al., 2010; Sutton and Barto, 2018).

- **"Missing out" on the global optimum.** Although the benefits of swarming are often credited as flexibility, robustness, and scalability (Hamann, 2018), some applications may benefit from using policies which are tuned to a specific settings. For instance, if it is known a priori that a swarm of robots will always exclusively operate within a specific environment and be composed of a fixed number of robots, then it may be desired to find the global optimum for this particular setup. As an example, consider an aggregation task with a swarm of three robots. An optimal policy might feature that it is beneficial for two robots to wait for the third to come into their neighborhood. The number of robots would thus be implicitly encoded into the final policy. Instead, the locally evolved policy would not consider this option, as the number of robots would not be known nor modeled explicitly throughout the evolution. Although this means that the solution is likely more scalable, flexible, or robust, it does mean that it is less globally optimal for the specific case. Moreover, as the approach is local in nature, it is not possible to directly evaluate global swarm parameters such as the ideal number of robots for a given task.
- **The need to define desired local states.** A complexity and potential limitation of using our approach is the need to define the set $\mathcal{S}_{des}$. This is the set of locally desired states that form the global goal. For the three tasks shown in this chap-

ter, $\mathcal{S}_{des}$ was extracted with ease using intuition. Nevertheless, as task complexity grows, $\mathcal{S}_{des}$ may become more difficult to define. A potential solution to this problem could be to use an automatic method which is capable of inferring the local desired states by observing a global behavior. This is discussed further in Section 4.7.3.

- **Assumption of successful state transitions.** The microscopic model used to evaluate PageRank assumes that state transitions are successful in executing the policy. In reality, this may not always be the case in light of imperfect actuation or noisy sensors, which may cause unexpected state transitions that are not modeled. This limitation can be addressed at the robot level via proper filtering of sensory data as well as accurate lower-level control. From the policy optimization side, it would be interesting to see whether this issue can be tackled via an online refinement of the model, allowing the policy to adapt itself to the imperfections of the real world during deployment. We continue this discussion in Section 4.7.3.

4.7.3. FURTHER POTENTIAL EXTENSIONS

- **Extension to online learning.** The current implementation uses an offline learning process, whereby the graphs $G^a_{\mathcal{S}}$ and $G^p_{\mathcal{S}}$ are devised based on a model of the robot and the task. There is also the possibility to extend the approach to online learning. The graphs $G^a_{\mathcal{S}}$ and $G^p_{\mathcal{S}}$ are based on the expected local states and transitions that the robot can experience. All these local states and all these transitions can be measured by the local sensors present on the robot. Therefore, this means that it is also possible for a robot to construct, expand, and/or tune the graphs $G^a_{\mathcal{S}}$ and $G^p_{\mathcal{S}}$ online directly based on its experiences during operation. In turn, it could then re-optimize its controller to suit these new graphs. For example, this could be done in order to automatically tune to the failure probability of certain actions using an online refinement of the transition rates. It can also be used to tune against "noisy" actuators which fail (with a given measurable probability) to follow through on an action, and either do not complete the action or create a different state transition than planned. Alternatively, it could adapt to systematic events in its environment, placing a higher probability on those transitions in the graph $G^p_{\mathcal{S}}$. The parameter vector $\alpha_{\mathbf{v}}$ could also be estimated or refined online. Thanks to the generally low computational efforts required to run the optimization (notice the wall-clock times in Figure 4.24), the optimization could also be performed on board of a robot without requiring excessive computational power.
- **Alternative fitness functions.** The fitness function proposed in this work (Equation 4.7) focused on maximizing the probability that a robot in the swarm would be in a desired state. The hypothesis that this would lead to global level improvements was found to be successful for each of the tested scenarios. Further investigations led us to the discovery that there may also be correlations to other parameters stemming from PageRank. For instance, using the results of the analysis from the pattern formation task from Section 4.5.6, we have found that the variance of the centrality also appears to show a correlation with the global performance of the swarm. In future work, it would be interesting to see if, and how, alternative fitness functions could be constructed in order to provide even larger global level

benefits.

- **Extension to heterogeneous swarms.** This chapter has tested the algorithm on homogeneous swarms of robots with one global goal. However, we wish to note that swarm homogeneity is not a direct limitation of the proposed algorithm. In fact, the passive graph $G^p_{\mathcal{S}}$, which models the environment, already does not assume that the neighboring robots operate with the same policy (rather, it assumes that all possible changes in the environment are equally probable). For a swarm of functionally heterogeneous robots, we anticipate the main difficulty to be in defining concurrent sets of desired states for each type of robot, which all combine toward a greater global goal.
- **Including transitions with a probability of failure.** An assumption that was taken for the tasks in this work was that, if a robot decides to perform an action, then it will always succeed. In reality, however, this may not always be the case, as certain state-action pairs may have a lower success rate than anticipated in the real world. However, this failure probability can be directly factored in within the transition probabilities of the graphs $G^a_{\mathcal{S}}$ and $G^p_{\mathcal{S}}$. On this note, it could also be possible for a robot to do this online while performing the task.
- **Automatic extraction of desired states.** As stated in Section 4.7.2, a limitation of the approach is the need to explicitly define the locally desired states. It can be postulated that this may not always be intuitively done by a swarm designer, particularly as task complexity increases. However, it can be argued that most (if not all) tasks require that some local states are more desired than others, and that these must be locally observable by the robot (even if not apparent, or understood, by the designer) — the challenge is to find them. In the next chapter, this challenge is solved by training a neural network that learns the relationship between local states and global fitness. From this, we are able to infer which combination of local states are most likely to maximize the global fitness.

4.8. CHAPTER CONCLUSIONS

The framework proposed in this chapter is based on the idea to microscopically model the local experiences of a robot in a swarm by means of graphs. It then becomes possible to evaluate the PageRank centrality of the local states, and use this as a measure of the global behavior of the swarm by evaluating the relative PageRank of local states of interest. This can be used within an optimization procedure in order to optimize the global behavior of swarms while only requiring a local analysis.

This technique provides several advantages which natively include flexibility, robustness, and scalability, which are three key properties of swarm robotics. We have applied it to a pattern formation task, a consensus task, and an aggregation task. The time needed to perform the optimization did not scale with the size of the swarm, which allowed us to tackle swarm sizes that could not be tackled if simulation had been our means of assessment. Moreover, the results obtained were natively independent of global parameters such as the initial starting condition, the environment, or the number of robots in the swarm. The new insights presented in this chapter offer a new strategy to evaluate the final behavior of groups from the perspective of the individual. We expect that the approach can generalize to several other tasks, whereby a robot in the swarm

and its environment is modeled in analogy to how a user surfs the World Wide Web.

Two challenges that now need to be overcome toward an end-to-end approach are:

1. Automatically extracting the local desired states.
2. Automatically extracting a local model of the swarm.

These two challenges are tackled in the next chapter, where we use a training set of random policies in order to simultaneously extract a both parameters from empirical data, showing that this can lead to a transparent end-to-end process.

CODE

- The main code used in this chapter can be downloaded at `https://github.com/coppolam/SI_pagerank` (Matlab). A script is also available to automatically download the data as needed.
- The aggregation simulator can be downloaded at `https://github.com/coppolam/swarmulator/tree/SI_aggregation`.
- The data used in this chapter can be downloaded at: `https://surfdrive.surf.nl/files/index.php/s/xrkOTVP30KI5Hle`.

5

A MODEL-BASED FRAMEWORK FOR LEARNING TRANSPARENT SWARM BEHAVIORS

This chapter proposes a model-based framework to automatically and efficiently design understandable and verifiable behaviors for swarms of robots. The framework is based on the automatic extraction of two distinct models: 1) a neural network model trained to estimate the relationship between the robots' sensor readings and the global performance of the swarm, and 2) a probabilistic state transition model that explicitly models the local state transitions (i.e., transitions in observations from the perspective of a single robot in the swarm) given a policy. The models can be trained from a data set of simulated runs featuring random policies. The first model is used to extract a set of local states that are expected to maximize the global performance. In the previous two chapters, these local states have been referred to as desired *local states, and were either defined manually or through an ad hoc procedure, rather than automatically as will be done here. The second model is used to optimize a stochastic policy so as to increase the probability that the robots in the swarm observe one of the desired local states. Following these steps, the framework proposed in this work can efficiently lead to effective controllers. This is tested on four case studies, featuring aggregation and foraging tasks. Importantly, thanks to the models, the framework allows us to understand and inspect a swarm's behavior. To this end, we propose verification checks to identify some potential issues that may prevent the swarm from achieving the desired global objective. In addition, we explore how the framework can be used in combination with a "standard" evolutionary robotics strategy (i.e., where performance is measured via simulation), or with online learning.*

5.1. BACKGROUND

The goal of swarm robotics is to design behaviors that enable several relatively simple robots to collaborate toward a common objective. The complexity of this paradigm stems from the fact that each robot can only sense and act according to local information, yet the goals are only observable at the global level. The complexity of swarming makes the manual design of successful controllers difficult. Machine learning approaches, however, offer an attractive way to do so automatically. State of the art techniques in the field primarily adopt evolutionary algorithms that optimize a policy with respect to a centrally measured objective (Brambilla et al., 2013; Francesca and Birattari, 2016). Other learning methods, such as reinforcement learning, have received less attention. This is because of issues such as low sample efficiency or credit assignment problems, whereby it is not clear how to relate a swarm's global performance with the local states and actions of its individual robots. Alternatively, with evolutionary strategies, the global performance of a population of controllers is assessed across multiple generations in order to find a suitable candidate. The approach has been used to optimize multiple controller architectures, including neural networks (Duarte et al., 2016) and behavior trees (Jones et al., 2018, 2019). Neural network controllers can approximate complex functions efficiently. However, their "black box" architecture can be a disadvantage in this context, particularly for safety-critical control. They are difficult to interpret and understand without resorting to experimentation on the swarm. Behavior trees and other explicit controllers are more transparent. Behavior trees, in particular, feature an attractive mixture of expressiveness and understandability (Colledanchise and Ögren, 2017). They are human-readable and can be understood, inspected, and even fine-tuned. In prior research on single robots, this has also given the ability to modify a behavior by hand to better transfer from simulation to the real world (Scheper et al., 2016). Nevertheless, understanding a behavior tree within the context of a swarm of robots also requires careful qualitative and experimental analysis (Jones et al., 2019). This is because the local state transitions experienced by a robot are not modeled. The local objectives of the robots, and how these correlate with the global performance, are also not explicitly known.

This chapter proposes a novel model-based framework in order to automatically design the behavior of a swarm of robots. To achieve these, the proposed framework makes use of two separate models: 1) a neural network model that maps the local states of the robots in the swarm to the global performance value, and 2) a probabilistic transition model that describes the local transitions as experienced by a single robot in the swarm. Both models are generated automatically from a data set of random behaviors. The first model is used to extract the local states that contribute to maximizing the swarm's global performance. We refer to these local states as *desired* local states. In prior chapters, the desired local states were defined manually or through an an ad hoc process. In this chapter, they are extracted automatically, reducing a global goal to locally observable constituents. Then, the second model is used to determine a policy that maximizes the probability that a robot in the swarm transitions to any one of the desired local states. The transition model enables us to examine the behavior of the swarm, given a policy. A set of conditions is proposed in order to identify, based on the generated models, potential issues that could prevent the swarm from achieving a collective goal. Overall,

the proposed framework has two main advantages: 1) the extraction of understandable and verifiable controllers, and 2) increased evaluation and optimization efficiency through the models. The framework can also be combined with a standard evolutionary algorithm (i.e., one which uses simulations to measure the swarm's performance) for increased efficiency. It can also be used online, such that each robot in the swarm locally optimizes its behavior according to a model of its experiences that it generates onboard. Implementations of the framework in both of these contexts are also provided in this chapter.

This chapter is structured as follows. In Section 5.2, we place our contribution in the context of recent swarm robotics and machine learning research. In Section 5.3, the model-based framework is explained. In Section 5.4, its performance is analyzed via four case studies, involving aggregation and foraging. It is also shown how the proposed model-based framework can be used in: 1) a hybrid model-based evolutionary algorithm, and 2) a model-based online learning approach. Section 5.5 discusses the advantages and limitations of the framework, highlighting potential future extensions and research. Section 5.6 provides concluding remarks.

5.2. RELATED WORKS AND RESEARCH CONTEXT

This work explores how to automatically design understandable and verifiable swarm behaviors. This is required in order to examine whether a swarm of robots, given a policy, will achieve the intended performance once deployed. The recent works by Jones et al. (2018, 2019) studied how to automatically evolve behavior trees for a similar purpose. However, albeit behavior trees are human-readable, they do not model the effects of local actions and their impact. In this work, Probabilistic Finite State Machines (PFSMs) are used to dictate the behavior. PFSMs are transparent architectures that offer one significant advantage for this context: they probabilistically model the local state transitions due to actions. This is a considerable benefit when it comes to understanding and analyzing a behavior. For instance, the works by Martinoli and Easton (2003), Correll and Martinoli (2006), and Liu and Winfield (2010) showed how a PFSM, directly extracted from the Finite State Machine used by each robot, can be used to predict a swarm's global behavior over time. In Chapter 3 of this thesis (Coppola et al., 2019b), a PFSM model was used to verify whether a swarm will eventually achieve its goal. PFSM models have also been used to optimize the policy of a swarm (Berman et al., 2009, 2011; Coppola et al., 2019a). In this chapter, following the optimization methodology presented in Chapter 4 of this thesis and published in Coppola et al. (2019a), the policy will be optimized by using a probabilistic state machine that models the local state transitions of a single robot in a swarm. However, the probabilistic model is now automatically extracted from experience data.

To automatically learn a swarm behavior, it is necessary to have a metric that measures the performance of the whole swarm. In state of the art automatic design methods, this metric is typically measured centrally, and directly used to assess the performance of a specific policy. However, this means that the learning procedure must either be performed offline (Duarte et al., 2016), or be reliant on a centralized sensor (e.g. an overhead camera) in a controlled environment (Jones et al., 2019). The robots cannot continue learning once deployed without this central sensor, which reduces flexibility to

untrained environments and novel situations. In this work, we thus take a different approach: a feed-forward neural network is trained to estimate the global performance of the swarm from the local states of its robots. This allows us to extract the local objectives of the individual robots, such that, if the robots achieve these local objectives, the global performance will benefit. This is a transparent step that results in the explicit representation of the local goals of robots. With this, the robot's policy can then be analyzed in relation to its fulfillment of local objectives. The model can be trained offline and the desired local states that are extracted from it can then even be used during online learning, which liberates the need to centrally assess the performance of the swarm during operation. To the best of our knowledge, this is the first work where a function is trained to explicitly relate microscopic (i.e., local) states with macroscopic (i.e., global) performance, and additionally use this to optimize the behavior of the swarm, also online.

The challenge of developing a function that synthesizes a global performance metric into measurements that are observable by onboard sensors is not unique to the swarming regime. It applies to any situation whereby the objective can only be partially measured by an agent. We briefly discuss some related literature on this topic from outside of the swarming domain, but that aim to solve this challenge for Partially Observable Markov Decision Problems (POMDPs). Recently, Faust et al. (2019) and Chiang et al. (2019) proposed AutoRL. The proposed solution is to complement the reinforcement learning process with an evolutionary procedure that optimizes the parameters of an explicit reward function. For swarming, however, this additional learning layer could be computationally problematic and potentially subject to over-fitting. Other solutions to similar problems have been proposed. In the works by Florensa et al. (2017) and Ivanovic et al. (2019), a robot learns to reach a final goal by learning backward from start states that are progressively far away. This strategy is also not easily suitable to swarming, which is expected to feature a prohibitive set of starting states that scales with the size of the swarm, whereas we aim for a scalable solution. In addition, swarm robotics often does not deal with a specific final state, but rather with a general performance metric. Ng and Russell (2000) proposed Inverse RL (IRL) to learn a reward function from a known optimal policy. Šošić et al. (2017) also applied IRL to the swarming domain to extract local controllers. However, these works assume a policy to be known, which is most often not the case. In a similar vein, demonstrations can be used to represent the actions of an optimal policy. Li et al. (2016) used demonstrations in a competitive evolutionary setup to learn to replicate the behavior of a swarm. However, this practice is not applicable when the optimal policy is not known, which is most often the case. Recently, Brown et al. (2019) ranked sub-optimal demonstrations in order to extract the underlying rules, leading to results that exceed the demonstrations. At a conceptual level, we will apply a similar strategy, as we extract local objectives from random swarm behaviors and optimizing the policy accordingly.

5.3. FRAMEWORK DESCRIPTION

The framework proposed in this chapter is depicted in the diagram in Figure 5.1. It is based on three core steps:

1. Learn the micro-macro relationship (Model 1) and use it to extract a set of desired

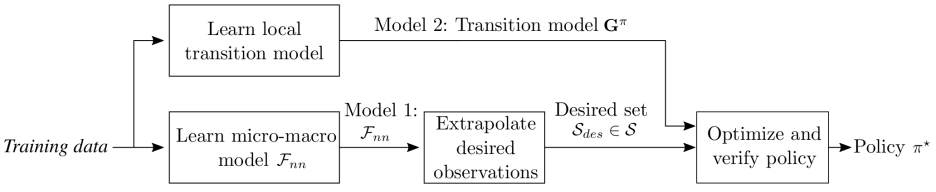


Figure 5.1 Overview of framework proposed in this chapter.

local states, denoted $\mathcal{S}_{des}$.
2. Extract the explicit local transition model (Model 2).
3. Use the above to optimize and verify a policy.

The three steps are detailed in Section 5.3.1, Section 5.3.2, and Section 5.3.3, respectively. Model 1, the "micro-macro" model, is a function that relates the local states of the robots in the swarm to the global performance. In this work, the function is modeled with a neural network. Training this function requires centralized data of the performance of the swarm during sample runs. However, once trained, the model can estimate the global performance of the swarm from the local states of the robots. Model 2 models the local state transitions experienced by a robot in the swarm. It can be learned offline or it can be directly estimated/tuned onboard by the robots during a task, enabling them to adapt their policy accordingly. Model 2 is a microscopic probabilistic transition model, which allows us to analyze it. To this end, Section 5.3.4 introduces verification conditions.

5.3.1. MODEL 1: MICRO-MACRO NEURAL NETWORK MODEL

For an arbitrary swarming task, let $F_g(t)$ be a global performance metric for the performance of the swarm at time $t \geq 0$, which needs to be maximized within a time $t \leq T$. Let $\mathcal{S} = \{s_1, s_2, \ldots, s_N\}$ be a discrete set of N local states that a robot in the swarm can observe using its onboard sensors. At a given time t, a robot in the swarm will be in a local state $s \in \mathcal{S}$. The robot cannot measure $F_g(t)$, since this requires knowledge that is not measurable by its onboard sensors. For example, as in Chapter 3, a swarm may be tasked with self-organizing into a pattern while the robots can only sense their nearby neighbors, and not the whole pattern. Therefore, instead of guiding the robots to optimize $F_g(t)$, we will instead guide them to be in local states such that $F_g(t)$ is maximized. The set of local states that maximize $F_g(t)$ are grouped in the set $\mathcal{S}_{des} \in \mathcal{S}$, which, as in Chapter 3 and Chapter 4, is referred to as the set of desired local states.[1]

To automatically extract the set $\mathcal{S}_{des}$ for an arbitrary swarm, we need to model the relationship between the local states of the robots and the global performance. This can be a complex nonlinear relationship, and so a neural network is used to automatically learn this micro-macro relationship from training data, limiting the a priori knowledge

[1] Since we are the first to propose this kind of approach, we make a number of simplifications, such as discrete local states and action spaces and binary desirability of states. Generalization beyond such simplifications is discussed in Section 5.5.

required. The neural network is represented by the function $\mathcal{F}_{nn}$ in Equation 5.1.

$$\hat{F_g}(t) = \mathcal{F}_{nn}(\mathbf{P_s}(t)) \tag{5.1}$$

In Equation 5.1, $\hat{F_g}(t)$ is the estimated $F_g(t)$, and $\mathbf{P_s}(t)$ is a vector of length N indicating the distribution of local states in a swarm at time t. Based on a training data set, the network $\mathcal{F}_{nn}$ will be trained to estimate $F_g(t)$ for unseen scenarios. $\mathcal{F}_{nn}$ has N input neurons and has one output.

Once the micro-macro model is trained, it can be used to extrapolate the set of desired local states $\mathcal{S}_{des}$ that are expected to maximize F_g. We use an optimizer to find an input vector $\mathbf{B}$ such that $\hat{F_g}(t)$ is maximized. The input vector is a binary vector $\mathbf{B}$ of size N. This allows us to extract the local states that aid to maximize $\hat{F_g}$, without bias for any single local state that may have been more prevalent according to the data. These local states will compose the set $\mathcal{S}_{des}$. In our particular implementation for this chapter, the optimization was performed using an evolutionary optimizer in order to avoid local maxima. Specifically, we used the evolutionary toolbox provided by the Distributed Evolutionary Algorithms in Python (DEAP) package (Fortin et al., 2012).

As an example, consider a swarm of robots that needs to reach consensus. $F_g(t)$ measures the highest fraction of robots in the swarm that is in agreement at time t. The robots, however, can only observe whether they are in agreement with their direct neighbors or not, i.e. $\mathcal{S} = \{s_{agree}, \neg s_{agree}\}$. $\mathcal{F}_{nn}$ would indicate that being in agreement with your direct neighborhood is correlated with a high $\hat{F_g}(t)$. Therefore, we can then find that an input $\mathbf{B} = \{1,0\}$ maximizes $\hat{F_g}(t)$, meaning that $\mathcal{S}_{des} = \{s_{agree}\}$.

Remark 5.3.1. *In the currently proposed form, $\mathcal{F}_{nn}$ is a feed-forward network and it does not use information from previous time steps for its estimate. This means that its estimate is independent of the actions of the robots, their dynamics, past information, and the number of robots in the swarm (albeit a bias may still exist as a result of the training data). Although this may create a disadvantage for the case where the performance metric has a temporal aspect, there are also advantages to it for certain contexts. Namely, the results of this procedure can be transferred across different environments and across different swarms, provided that the local states (i.e., observations using the onboard sensors) and the performance metric remain unchanged. This will be empirically evaluated in Section 5.4. Its further potential is discussed in Section 5.5.*

5.3.2. MODEL 2: TRANSITION MODEL

Following the procedure in Section 5.3.1, a set of desired local states correlated with a high global performance is automatically extracted. This reduces the problem to a local variant, wherein the policy of a robot needs to be such that its probability of transitioning to a local state $s \in \mathcal{S}_{des}$ is maximized.

A robot in the swarm can perform actions from an action set $\mathcal{A} = \{a_1, a_2, \ldots, a_M\}$ of size M. Let π be a tabular probabilistic policy of size $N \times M$ that, for a given local state $s \in \mathcal{S}$, indicates the probability of taking an action $a \in \mathcal{A}$. The transition model can be used to understand, quantify, and potentially verify the policy and its impact on the swarm. As in Chapter 4 (Coppola et al., 2019a), an optimum policy can be found efficiently using PageRank centrality as a fitness measure. The main advantage of this approach is that

the policy optimization is performed without simulation. The layout of the model is inspired by the random surfer model by Brin and Page (1998), which can be summarized by the "Google" matrix $\mathbf{G}_\pi$:

$$\mathbf{G}_\pi = \alpha \mathbf{H}_\pi + (\mathbf{I} - \alpha)\mathbf{E}. \tag{5.2}$$

$\mathbf{G}_\pi$ is a cumulative model composed of two parts:

- $\mathbf{H}_\pi$ is a stochastic matrix of size $N \times N$ holding the probabilities of local state transitions that result from the actions taken by an arbitrary robot in the swarm, following policy π.
- $\mathbf{E}$ is a stochastic matrix of size $N \times N$ holding the probabilities of local state transitions that occur when the robot has not taken an action. These transitions are thus attributed to the environment (which includes the other robots in the swarm).

The diagonal matrix α balances the probability of local state transitions in $\mathbf{H}_\pi$ and $\mathbf{E}$ for each row. In other words, each diagonal entry in α, denoted $\alpha(s_i)$ is the ratio of likelihood of occurrence between $\mathbf{H}_\pi$ and $\mathbf{E}$ when in a local state s_i. Cumulatively, $\mathbf{G}_\pi$ models all transitions that can occur to a robot in the swarm, either as a result of its policy or as a result of its environment.

To model the expected transition due to a specific action, $\mathbf{H}_\pi$ can be decomposed into a series of sub-models, each showing the impact of one action from the action set $\mathcal{A}$. Let $\mathbf{A}$ be a set of M sparse stochastic matrices of size $N \times N$, such that $\mathbf{A} = \{\mathbf{A_1}, \mathbf{A_2}, \ldots, \mathbf{A_M}\}$. The elements of a matrix $\mathbf{A}_k$ hold the probability of a local state transition from $s_i \in \mathcal{S}$ to $s_j \in \mathcal{S}$ given an action $a_k \in \mathcal{A}$, for $k = 1, \ldots, M$. Specifically:

$$\mathbf{A_k}(s_i, s_j) = P(s_{i,j}|a_k), \tag{5.3}$$

where $s_{i,j}$ indicates a transition event from $s_i \in \mathcal{S}$ to $s_j \in \mathcal{S}$. It follows that $\mathbf{A}$ can be combined into a cumulative model of all transitions as a result of a policy π, which is the matrix $\mathbf{H}_\pi$. An element in $\mathbf{H}_\pi$ is given by:

$$\mathbf{H}_\pi(s_i, s_j) = P((s_{i,j} \cap a_1) \cup (s_{i,j} \cap a_2) \cup \ldots \cup (s_{i,j} \cap a_M)) = \sum_{k=1}^{M} P(s_{i,j} \cap a_k)_\pi \tag{5.4}$$

where:

$$P(s_{i,j} \cap a_k)_\pi = P(s_{i,j}|a_k) \cdot \pi(s_i, a_k) = \mathbf{A_k}(s_i, s_j) \cdot \pi(s_i, a_k) \tag{5.5}$$

In the above, $\pi(s_i, a_k)$ is the probability of taking a_k under local state s_i. This format separates the transition of an action into separate models, which serves to predict the effect of a different policy. The transition models can be numerically estimated from experience data as a maximum likelihood model (Sutton, 1990). Any time a transition happens when taking an action a_k, its transition is assigned to the corresponding model $\mathbf{A_k}$. Any time a transition happens when no action is taken, its transition is assigned to $\mathbf{E}$. The diagonal elements of the diagonal matrix α are estimated by the ratio of active events to environment events.

Remark 5.3.2. *One limitation of the transition model used in this paper is that it requires the discretization of both the local state space and action space into the discrete sets $\mathcal{S}$ and $\mathcal{A}$. This is done to enable the model-based optimization and analysis explained in Section 5.3.3 and Section 5.3.4. However, the use of alternative models could be explored in future work. This is further discussed in Section 5.5.*

5.3.3. MODEL-BASED POLICY OPTIMIZATION

As in Chapter 4, an optimum policy $\pi^\star$ is found by maximizing the probability that a robot in the swarm transitions to a local state $s \in \mathcal{S}_{des}$. This is evaluated by the metric F_{pr},

$$F_{pr}(\pi) = \frac{\sum_{\forall s \in \mathcal{S}_{des}} PR_\pi(s)/|S_{des}|}{\sum_{\forall s \in \mathcal{S}} PR_\pi(s)/|S|}, \tag{5.6}$$

where $PR_\pi(s)$ indicates the PageRank centrality of local state s given a model $\mathbf{G}_\pi$. PageRank centrality is a measure for the probability of transitioning to a given node in a graph that represents the transition model $\mathbf{G}_\pi$. Conceptually, Equation 5.6 serves to maximize the average PageRank of the local states in the set $\mathcal{S}_{des}$ over the set of all local states $\mathcal{S}$. This optimizes the policy based on the transition model efficiently and without 1) a more computationally expensive procedure based on simulated experience, or 2) trial and error.

5.3.4. MODEL-BASED ANALYSIS AND VERIFICATION

This section explains how the transition model and the desired local states can be used in order to check properties of the swarm's behavior. A set of task-agnostic conditions are presented to determine the swarm's ability to achieve its goal, whereby the goal is defined as a state where all robots achieved a desired local state.

Consider a swarm of m robots. Let $\mathcal{P}$ be the set of all global states that the swarm can be in. Let $\mathcal{P}_{des} \in \mathcal{P}$ be a set of all global states for which all robots in the swarm have a local state $s \in \mathcal{S}_{des}$, and let $p(t)$ be the global state of the swarm at time t. It is assumed here that $\mathcal{P}_{des} \neq \emptyset$. The swarm is defined as **successful** as per Definition 5.3.1, which is indicative of a good performance by the swarm as obtained through the procedure based on Model 1 from Section 5.3.1.

Definition 5.3.1 (Successful)**.** A swarm with a global state $p(t)$ is **successful** if $p(t) \in \mathcal{P}_{des}$.

We wish to determine whether the swarm will eventually be successful when starting from any arbitrary initial global state $p(t = 0)$. One way to determine this would be to analyze all possible global states of the swarm and transitions between them. However, such an analysis can become intractable as the size of the swarm, and its global state space, increases (Dixon et al., 2012). We thus adopt a different strategy that uses the local transition model (Model 2) to identify two types of potential counter-examples: **deadlocks** and **livelocks**. In this chapter, we define them in Definition 5.3.2 and 5.3.3, respectively. If no livelocks and deadlocks exist, then the swarm can keep transitioning and potentially achieve all global states, thus including the successful states contained in $\mathcal{P}_{des}$. Note, however, that the local checks shown in this chapter are not guaranteed,

for any arbitrary task, to identify all possible livelocks and deadlocks. For a given task, however, more detailed and ad hoc checks can be constructed, as was for instance done for the pattern formation task presented in our previous work in Chapter 3 (Coppola et al., 2019b).

Definition 5.3.2 (Deadlock)**.** A **deadlock** is a state $p(t) \notin \mathcal{P}_{des}$ whereby the swarm does not transition to a different state, i.e., $p(t_2) = p(t_1) \,\forall\, t_2 > t_1$.

Definition 5.3.3 (Livelock)**.** Consider an arbitrary subset of global states $\mathcal{P}_{lock} \in \mathcal{P}$, where $\mathcal{P}_{lock} \cap \mathcal{P}_{des} = \emptyset$ and $1 < |\mathcal{P}_{lock}| < |\mathcal{P}|$. At time $t > t_0$ a swarm is in a **livelock** if it can only transition between states that are within the set $\mathcal{P}_{lock}$, i.e., $p(t > t_0) \in \mathcal{P}_{lock}$.

IDENTIFYING DEADLOCKS FROM LOCAL STATES

Deadlocks indicate that the swarm has stopped from changing global state. Potential deadlocks can be identified with the help of the transition model (Model 2). Let $\mathcal{S}_{static}$ be a set of local states for which a robot does not transition to any new local state via its policy. $\mathcal{S}_{static}$ can be determined from the empty rows of the model $\mathbf{H}_\pi$, indicating that no state transitions are possible.

$$\mathcal{S}_{static} = \{s \in \mathcal{S} : \sum_{j}^{N} \mathbf{H}_\pi(s, s_j) = 0\} \tag{5.7}$$

The following three outcomes are possible:

- If $\mathcal{S}_{static} = \emptyset$, then a deadlock is not present (according to the model), because all robots can move and have a nonzero probability of transition.
- If $\mathcal{S}_{static} \cup \mathcal{S}_{des} = \mathcal{S}_{des}$, then the swarm may cease to transition. However, this is only for global states $p(t) \in \mathcal{P}_{des}$, which does therefore not constitute a deadlock under Definition 5.3.2.
- If $\mathcal{S}_{static} \cup \mathcal{S}_{des} \neq \mathcal{S}_{des}$, then a deadlock may be present. This is caused by the local states in $\mathcal{S}_{static}$ that are not part of $\mathcal{S}_{des}$.

To determine the deadlocks that may occur based on the above, it is necessary to analyze the local states in $\mathcal{S}_{static}$ and whether (and, if desired, how) they can coexist at the global level. This is a global level check that however only requires analysis of a subset of global states. In addition, the check does not necessarily scale with the size of the swarm. For any task where the swarm is not expected to exist in one connected cluster, then local connected deadlocks may be identified for small clusters and it is not necessary to examine with the swarm size m. During policy optimization, if a deadlock is caused by a new policy, causing $\mathcal{S}_{static} \cup \mathcal{S}_{des} \neq \mathcal{S}_{des}$, then the policy of the robots may be adapted to resolve this.

LOCALLY IDENTIFYING LIVELOCKS

In a livelock, the swarm transitions between global states that do not include or lead to success. In this section, we propose checks to identify the presence of livelocks using the local transition model. Note that the checks proposed in this section are general and thus not guaranteed to detect *all* possible livelocks for an arbitrary task. However, more

detailed and ad hoc checks can be constructed with this type of model, as was done for the pattern formation task presented in our previous work (Coppola et al., 2019b), and in future work we encourage the development of further checks.

Let $\mathcal{S}_t$ be the set of the local states of the robots in the swarm at an arbitrary time t. By the condition in Proposition 5.3.1 it can be established whether there is any set of initial local states $\mathcal{S}_{t=0}$ that can potentially prevent the swarm from achieving a global state $p \in \mathcal{P}_{des}$. This can stem from the fact that local states exist which endlessly cycle between states $s \notin S_{des}$, thus preventing the swarm from achieving a successful global state as defined in Definition 5.3.1. In the following, let $G_{\mathcal{S}}^{H_\pi}$ be a weighted graph made from the transition model $\mathbf{H}_\pi$, and let $G_{\mathcal{S}}^{E}$ be a weighted graph made from the transition model $\mathbf{E}$.

Proposition 5.3.1. *If in $G_{\mathcal{S}}^{H_\pi}$ there is a path from all local states $s \notin \mathcal{S}_{des}$ to all local states $s \in \mathcal{S}_{des}$, then a swarm will be successful independently of $\mathcal{S}_{t=0}$.*

Proof. The model $G_{\mathcal{S}}^{H_\pi}$ only considers the local state transitions that the robots make themselves. Consider a robot i in the swarm with an arbitrary local state $s_i \in \mathcal{S}$. If the condition holds, then, by following the policy π, the robot has a probability $\rho > 0$ of transitioning to any and all local states in the set $\mathcal{S}_{des}$. Applying this to all robots in the swarm, it follows that all robots will have a nonzero probability of eventually transitioning to a local state $s \in \mathcal{S}_{des}$, according to the model. Therefore, a desired global state $p \in \mathcal{P}_{des}$ can be achieved independently of $\mathcal{S}_{t=0}$. As robots can transition to all local states in $\mathcal{S}_{des}$, they are not bounded to any specific desired local state. ■

If the condition in Proposition 5.3.1 is true, then, if an initial global state $p(0)$ exists for which the swarm cannot be successful, then this is not due to any local state in $\mathcal{S}_{t=0}$. A case where the condition in Proposition 5.3.1 fails is if static local states exist which are not desired, i.e., $\mathcal{S}_{static} \cup \mathcal{S}_{des} \neq \mathcal{S}_{des}$, where $\mathcal{S}_{static}$ is defined as per Equation 5.7. In this situation, a robot with a local state $s \in \mathcal{S}_{static} - \mathcal{S}_{des}$ may not transition to a local state $s \in \mathcal{S}_{des}$. We can then check whether the environment is such that the robot could transition to non-static local states, and subsequently transition to a desired local state. This is addressed by Proposition 5.3.2. Note that in the following we assume that all diagonal entries of α are between 0 and 1, i.e., $0.0 < \alpha(s) < 1.0 \; \forall \; s \in \mathcal{S}$.

Proposition 5.3.2. *If the following two conditions apply:*

1. *In $G_{\mathcal{S}}^{E}$, all local states $s \in \mathcal{S}_{static} - \mathcal{S}_{des}$ have a direct transition to any local state $s \notin \mathcal{S}_{static}$.*
2. *In $G_{\mathcal{S}}^{H_\pi}$, there is a path from all local states $s \notin \mathcal{S}_{static}$ to all local states $s \in \mathcal{S}$.*

then a global state $\mathcal{P}_{des}$ can be achieved independently from $\mathcal{S}_{t=0}$.

Proof. Consider a robot with a local state s where $s \in \mathcal{S}_{static} - \mathcal{S}_{des}$. If the first condition is fulfilled, then the robot has a probability ρ_1 of transitioning to a local state $s \notin \mathcal{S}_{static}$, where $\rho_1 > 0$. If now the robot has a local state $s \in \mathcal{S}_{des}$, then we are done. If, instead, the robot has a local state $s \notin \mathcal{S}_{static} \cup \mathcal{S}_{des}$, then, if $G_{\mathcal{S}}^{H_\pi}$ fulfills the second condition, the robot has a probability $\rho_2 > 0$ of transitioning to any local state. This includes any local state in the set S_{des}. If the robot transitions back to a local state $s \in \mathcal{S}_{static}$, then the process is reinitiated. ■

Remark 5.3.3. *The policy optimization procedure can alter the policy π in such a way as to alter the outcomes of the conditions. During optimization, this can be prevented by setting a constraint $\pi(s,a) > \varepsilon > 0 \ \forall \ s \in \mathcal{S}, a \in \mathcal{A}$, where ε is a minimum bound on the probability.*

5.4. PERFORMANCE ANALYSIS

In this section, the framework is evaluated through four case studies, introduced in Section 5.4.1, featuring aggregation and foraging scenarios. The case studies were designed to test different aspects of the framework, in its current implementation, in tasks of increasing and/or different complexity, with a global performance metric that is not measurable by the individual robots. The framework will first be analyzed in a standalone fashion. For this case, training data will be generated from the execution of random policies, which will be used to train the models and optimize the policy. Section 5.4.2 details the simulation environment and the data gathering methodology. The results are evaluated in Section 5.4.3 and Section 5.4.4. The swarms are analyzed, using the models, in Section 5.4.5. Following the standalone analysis, Section 5.4.6 and Section 5.4.7, explore its use for hybrid model-based evolutionary learning and model-based online learning, respectively.

5.4.1. DESCRIPTION OF CASE STUDIES

CASE STUDY A: AGGREGATION TASK WITH NON-DIRECTIONAL NEIGHBOR SENSORS

In this task, a swarm of n freely moving robots in an arena should aggregate as much as possible within a maximum time T. The robots behave like accelerated particles and use repulsion to avoid collisions with nearby neighbors. The robots also reflect off walls.

- *Goal:* The global fitness function to maximize is

$$F_g^A(t) = \frac{n}{c(t)}, \tag{5.8}$$

  where $c(t)$ is the number of connected clusters at time $0 \leq t \leq T$, and n is the total number of robots in the swarm. A connected cluster is defined as a group of robots that forms one group with a connected graph topology, whereby a single robot by itself also counts as one cluster. Note that $F_g^A(t)$ is subject to $1 \leq F_g^A(t) \leq n$. This creates a lower bound for a poorly performing swarm and a higher bound if a larger swarm aggregates, which is representative of a greater difficulty. A top view of the task is shown in Figure 5.2a.

- *Onboard sensors:* Each robot can sense how many neighbors are within a sensing range $r_{\max}$ from itself, for a maximum of $m_{\max}$ neighbors. The number of neighbors is denoted m, where $m \in \mathbb{Z}^+$ and $0 \leq m \leq m_{\max}$. Therefore, $\mathcal{S} = \{0, 1, \ldots, m_{\max}\}$. This set up is representative of an antenna that the robots use to sense each other, with a maximum range $r_{\max}$ and a maximum number of possible connection nodes $m_{\max}$. The sensory setup is depicted in Figure 5.2d.

- *Actions:* Each robot can choose whether it should a) move randomly, or b) stay still. It makes this choice based on policy π with frequency f_c, or if its current local state

changes. π is a vector of $(m_{\max}+1)\times 1$, where each entry indicates the probability of choosing "move", given $m \in \mathbb{Z}^+$ neighbors, where $0 \le m \le m_{\max}$. When "move" is selected, the robot moves in a randomly chosen direction.

In the results in this chapter, the following parameters were used: $f_c = 0.5\ Hz$, $m_{\max} = 7$, $T = 200\ s$.

CASE STUDY B1: AGGREGATION TASK WITH DIRECTIONAL NEIGHBOR SENSORS

In this task, a group of robots must form an aggregate as in case study A, yet the specifics of the robots are significantly different, thereby increasing the difficulty of the task. Each robot moves as a directed particle within its own frame of reference. According to the action space, it is always commanded to have a given forward speed, and can only dictate its turning rate. The robots avoid collisions using repulsion, which is added to the commanded velocity.

- *Goal:* The fitness function is the same as given by Equation 5.8 for case study A. A top view of this task is shown in Figure 5.2b.

- *Onboard sensors:* Each robot can sense the presence of neighbors within sectors, up to a range $r_{\max}$ away. This is representative of, for instance, infrared sensors. The local state space is $\mathcal{S} = \{s_1, \ldots, s_n\}$ where a state s_i describes the sectors in which neighbors are sensed. It follows that $|\mathcal{S}| = 2^q$, where q is the number of sectors. The sensory setup is depicted in Figure 5.2e for the case where $q = 4$, which we used in this work.

- *Actions:* At all times, a robot moves forward with a forward velocity v_{cmd} along its local body frame. It can control its turning rate ψ from the action set $\mathcal{A}$, where the following was defined:

$$\mathcal{A} = \{-1.0, -0.7, -0.3, -0.1, 0.1, 0.3, 0.7, 1.0\}.$$

  The policy π is a stochastic matrix of size $|\mathcal{S}| \times |\mathcal{A}|$. A robot selects a new action with a frequency f_n, or if its local state changes. It is not possible for the robots to not select an action from $\mathcal{A}$ (with the exception of wall avoidance within the arena, whereby the robots rotate away from the wall). This means that the robots are always prompted to move with a nonzero turning rate, which can create a challenge for aggregation. Collision avoidance commands are added to the commanded velocity vector from the action space.

The following parameters were used in the results: $f_n = 0.5\ Hz$, $q = 4$, $v_{cmd} = 0.5\ m/s$, $T = 200\ s$.

CASE STUDY B2: AGGREGATION TASK VARIANT WITH DIRECTIONAL NEIGHBOR SENSORS

This task is a modified version of case study B1, intended to further increase the difficulty. The goal and onboard sensors are the same as in B1. However, the actions that the robots take are different.

- *Goal:* The fitness function is the same as given by Equation 5.8 for case studies A and B1. A top view of this task is shown in Figure 5.2b.
- *Onboard sensors:* This is the same as in case study B1, depicted in Figure 5.2e.
- *Actions:* The action space used is the same as B1:

$$\mathcal{A} = \{-1.0, -0.7, -0.3, -0.1, 0.1, 0.3, 0.7, 1.0\}.$$

  The definition of the probabilistic policy π is also the same. However, the following is different. The forward speed is $v_{cmd} = v_{\text{mean}} \cdot |a_k|$, where a_k is the currently selected value from $\mathcal{A}$. In addition, a_k is also the turning rate $\dot{\psi}$. When a robot selects a turning rate, the corresponding rate is applied for a time t_1, after which the robot moves straight until a time t_2 with velocity v_{cmd}, where $t_1 + t_2 = 1/f_n$, or until its local state changes.

The following parameters were used: $f_n = 0.5\ Hz$, $q = 4$, $v_{\text{mean}} = 0.5\ m/s$, $t_1 = 1\ s$, $t_2 = 1\ s$, $T = 200\ s$.

5

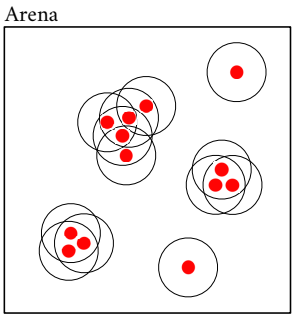
CASE STUDY C: FORAGING TASK

In this task, a swarm of n robots in an arena is tasked with maximizing the food at the nest at a final time T. Each robot at the nest consumes e_n food items per second. A robot that ventures out to forage for food eats e_f food items every t_c seconds that it spends searching. If a food item falls within its sensor range, then the robot collects it and returns to the nest. When collected, a new food item appears in a random location, such that the total number of food items in the arena remains constant. A robot cannot hold more than one food item at any given time.

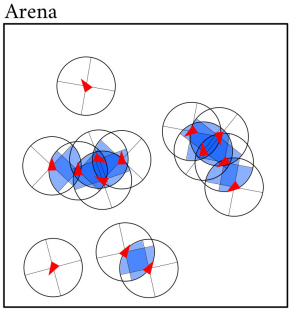
- *Goal:* The global fitness function is

$$F_g^C(t) = f(t) - f(0), \tag{5.9}$$

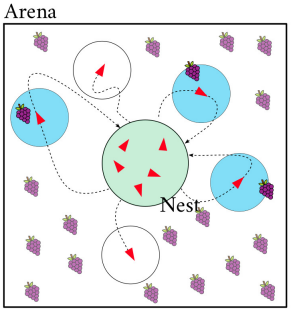
  where $f(t)$ is the total number of food items at the nest at time $0 \leq t \leq T$, and $f(0)$ is the food at the nest at the start of the task. A top view of the task is shown in Figure 5.2c.
- *Onboard sensors:* A foraging robot can sense food items within a maximum range $r_{\max}$ as shown in Figure 5.2f. Upon returning to the nest, the robot can sense the difference in food between when it left and when it returned. This is saturated by a value $f_{\max}$ and discretized evenly along $|\mathcal{S}|$ steps, i.e, $\mathcal{S} = \{-f_{\max}, \ldots, f_{\max}\}$. A robot cannot measure the global fitness $F_g^C(t)$ but only the difference, i.e., $f_g^C(t) = F_g^C(t_{\text{arrival}}) - F_g^C(t_{\text{departure}})$, where $-f_{\max} \leq f_g^C(t) \leq f_{\max}$. When at the nest, the robot measures the difference between its observations every time that it reiterates whether to explore or not, as below.
- *Actions:* When at the nest, a robot can make the choice to forage or to remain at the nest. The policy π is a vector of length $|\mathcal{S}| \times 1$, where each entry in π indicates the probability of choosing "explore" given the current local state. When "explore" is selected, the robot leaves the nest and does not come back until food is found. When at the nest, a robot reassess its choice with a frequency f_n.



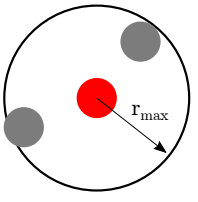
(a) Top view of case study A



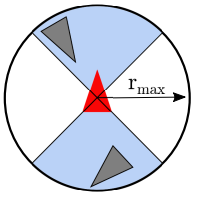
(b) Top view of case studies B1 and B2



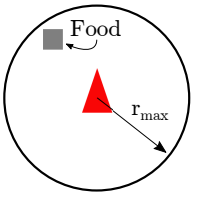
(c) Top view of case study C



(d) Sensor for case study A



(e) Sensor for case studies B1 and B2



(f) Sensor for case study C

Figure 5.2 Global views and sensor depictions for each task. Robots are indicated in red.

The following parameters were used: $f(0) = 15$, $f_n = 0.1\ Hz$, $T = 500\ s$, $e_f = 0.1$, $t_c = 10\ s$, $e_n = 0.02$, and $|\mathcal{S}| = 30$ with $f_{\max} = 5$.

5.4.2. TEST ENVIRONMENT AND DATA GATHERING

TEST ENVIRONMENT

All four tasks are evaluated using the C++ based simulation environment *Swarmulator*. This is the dedicated swarm simulator that was also used in prior chapters of this thesis and their accompanying publications (Coppola et al., 2019a,b). Each robot is simulated on an independent detached thread so that timing artifacts are avoided and automatically randomized. Swarmulator is described in more detail in Chapter A. A link to the relevant code is provided at the end of this chapter.

TRAINING DATA

The training data was gathered from 500 independent simulations, simulated for time T in arenas of size 20×20 meters. In each simulation run, a swarm of n robots follows a randomly generated policy π based on the policy spaces introduced in Section 5.4.1. Random swarm sizes were used with $1 \leq n \leq 30$, chosen from a discrete uniform distribution. For case study C, this was changed to $1 \leq n \leq 20$ to limit congestions at the nest. The global fitness $F_g(t)$ and the local states $s \in \mathcal{S}$ of each robot is logged with a frequency of 2 Hz with respect to the simulated time clock. This makes for $2T$ data points per sim-

Table 5.1 Summary table with properties for each case study.

Case study	$\|\mathbf{P}_s\|$	Hidden layers	Layer size	Learning Rate	T(s)
A	8	3	30	1e-5	200
B1	16	3	30	1e-5	200
B2	16	3	30	1e-5	200
C	30	3	100	1e-6	500

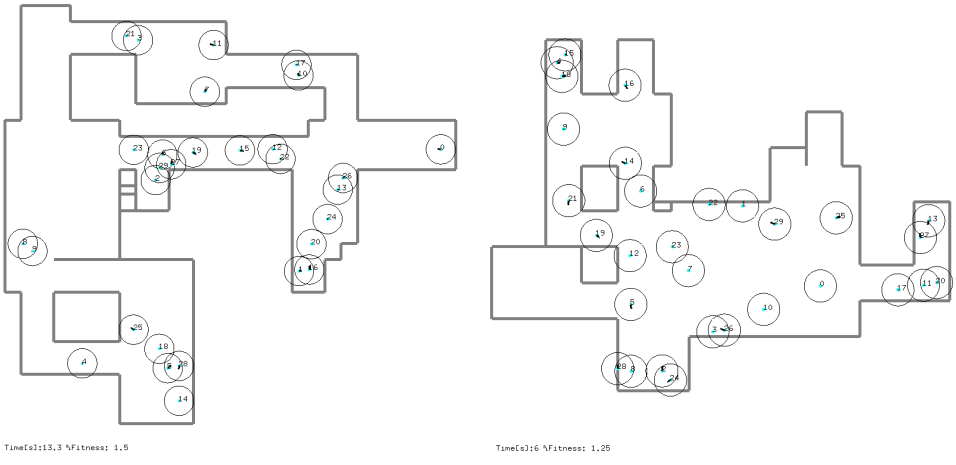


Figure 5.3 Top view of two examples of randomly generated arenas with 30 robots from case study A, as for VS 3. These screenshots are taken directly from the simulator.

ulation, and a training set size of $\approx 1000T$ to train Model 1 (the micro-macro model). In addition, all local state transitions by the robots during these simulations were logged and used to generate Model 2 (the transition model).

5.4.3. IMPLEMENTATION AND RESULTS OF MODEL TRAINING

MODEL 1: IMPLEMENTATION, TRAINING, AND ANALYSIS

For each task, a neural network $\mathcal{F}_{nn}$ is constructed with Rectified Linear Unit (ReLU) layers. All networks were trained using the Adam optimizer as implemented in PyTorch.[2] The specifics of each network are summarized in Table 5.1. Each neural network was trained using the training data generated as per Section 5.4.2. Then, three validation sets were made for each task:

- **Validation Set 1 (VS 1)**, generated by 100 runs using the same setup as for the training data set. As for the training set, each run featured a randomly generated policy and a random number of robots.
- **Validation Set 2 (VS 2)**, generated by 100 runs using the same setup as for the training data set, but with a smaller arena size of 10×10 meters. As for the training set,

[2] See Stevens et al. (2020) for more information.

each run featured a randomly generated policy and a random number of robots.

- **Validation Set 3 (VS 3)**, generated by 100 runs using the same setup as for the training data set, but with randomly generated arenas featuring multiple rooms and corridors. Each run featured a randomly generated policy, a random number of robots, and a random arena.

  Two examples of randomly generated arenas are shown in Figure 5.3.

The intent behind VS 2 and VS 3 is to determine how well a network can generalize to different environments. The performance of the networks is validated by assessing the mean correlation between the true F_g and its estimate $\hat{F}_g$ across a validation set. The correlation measures how valid $\mathcal{F}_{nn}$ is as a tool to extract the set of desired local states $\mathcal{S}_{des}$. A positive correlation indicates that the network has learned to predict when the global fitness increases and/or decreases. A correlation of 1.0 implies that the trend in F_g is perfectly estimated.

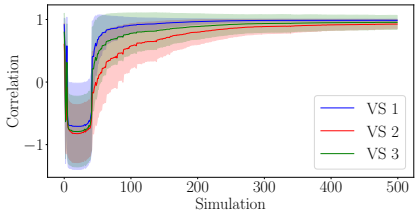
The results of this analysis are shown in Figure 5.4. In all cases, a positive correlation is achieved for the validation sets. For case study A, B1, and B2 the correlation is almost maximized (Figures 5.4a, 5.4b, and 5.4c, respectively). The sample runs in Figures 5.5a, 5.5b, and 5.5c show that the global fitness is accurately predicted. The correlation is lower for case study C, shown in Figure 5.4d. This is expected because of the temporal component of this task, which is not modeled by the neural network. Nevertheless, also for case study C, the general trend was still captured. This is exemplified by analyzing the prediction for the individual case, as shown in Figure 5.5d. We remark that it is likely that these results could be further improved with more dedicated hyperparameter tuning, which we encourage in future work. Two conclusions can be drawn from these results.

1. The networks can also generalize to validation sets generated with different arenas. This means that the trained network can be exported outside of the direct environment for which it has been trained, and that by adding more diverse training data we could also be able to generalize the predictions further. This is especially valid for the aggregation tasks, for which the fitness function does not feature temporal effects.

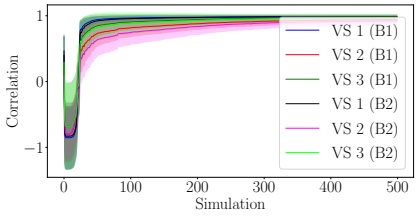
2. Even though the behavior of the robots in tasks B1 and B2 is different, the trained networks are almost interchangeably applicable. This can be appreciated in Figures 5.4b and 5.4c (green and black lines). This is because both tasks feature the same global fitness and the same local sensors, showing that the performance of $\mathcal{F}_{nn}$ is not dependent on the dynamics.

MODEL 2: TRAINING AND RESULTS

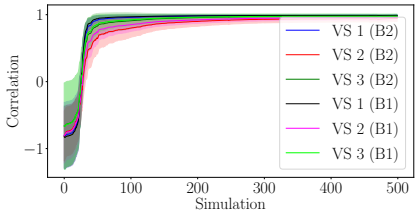
The transitions are estimated by the proportion of times that they occur over the simulations of the training data, as detailed in Section 5.3.2. Figure 5.6 shows the convergence of all transition probabilities in the action models in **A**, estimated over the training simulations. The figures shows the difference between the estimate at a given simulation and the final estimate after all simulations, providing a visualization of the convergence. It can be observed that more rare transitions follow a less graceful decline, providing only a



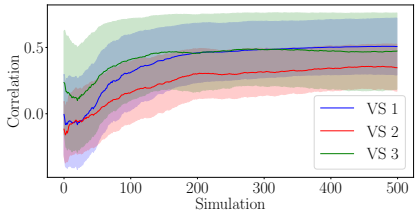
(a) Case study A



(b) Case study B1

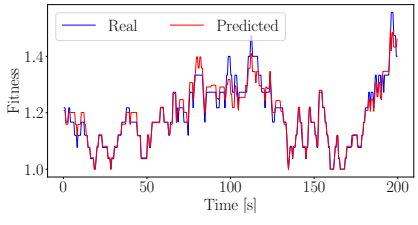


(c) Case study B2

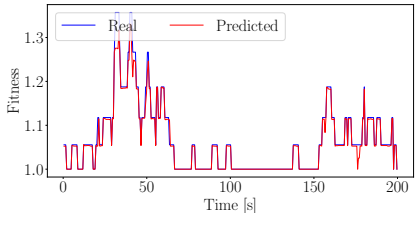


(d) Case study C

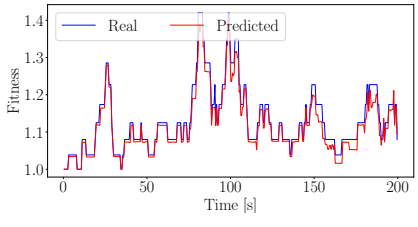
Figure 5.4 Correlation between predicted global fitness and real global fitness of validation set during learning.



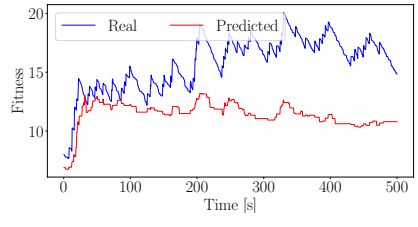
(a) Case study A



(b) Case study B1



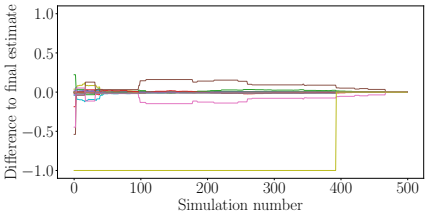
(c) Case study B2



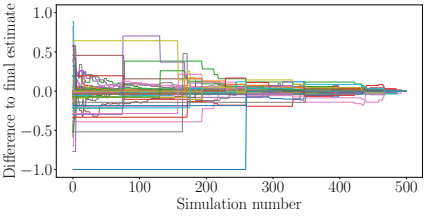
(d) Case study C

Figure 5.5 Sample global fitness estimates over one run, compared to the real fitness.

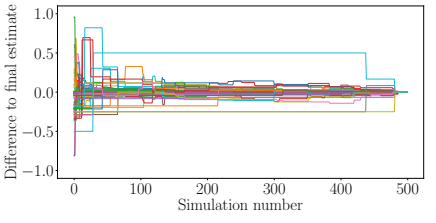
rough estimate of their true likelihood. In practice, this problem will be mitigated when this framework is not used in a standalone procedure, as will be explored in Section 5.4.6 and Section 5.4.7. For online use, it would be beneficial to adopt a pre-trained model to



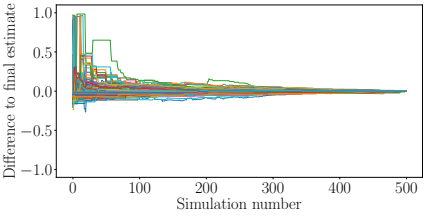
(a) Case study A



(b) Case study B1



(c) Case study B2



(d) Case study C

Figure 5.6 Convergence of transition model probabilities over 500 simulation runs for each task.

provide the robots with initial estimates.

5.4.4. SWARM PERFORMANCE EVALUATION OF OPTIMIZED POLICY

Once both models are generated, the set $\mathcal{S}_{des}$ can be extracted and the policies can be optimized accordingly. The policy optimization in these results was achieved using an evolutionary algorithm, implemented using DEAP (Fortin et al., 2012), in order to optimize the parameters according to the PageRank-based metric from Section 5.3.3. For each case study, the performance of the swarm is matched against: 1) an original baseline performance, evaluated using 100 random policies, and 2) the performance of a policy that was evolved using a centrally monitored evolutionary algorithm, as is state of the art practice in the literature. The performance was developed and tested in arenas of 20×20 meters.

PERFORMANCE FOR CASE STUDIES A, B1, AND B2 (AGGREGATION)

In all three cases, the set $\mathcal{S}_{des}$ included all local states featuring one or more neighbors that had been explored during the training runs. This is a sensible result: the algorithm finds out that the robots should arrive at these local states in order to maximize $\hat{F}_g$. In turn, the optimized policy directly aimed at maximizing the probability of ending up with a local state featuring one or more neighbors. The swarm's performance is evaluated with 100 simulations of 200 s. The performance of the optimized policies can be seen in Figures 5.7a, 5.7b, and 5.7c, where they are compared to a baseline performance and to an evolved performance. The mean performance over the evaluation run using the optimized policy can be appreciated in Figures 5.8a, 5.8b, and 5.8c. Case studies A and B1 match the policy evolved using a standard evolutionary algorithm. The perfor-

mance of the optimized policy for case study B2 did not match the true optimum evolved via evolution. This is because it could not exploit global properties and its optimized behavior was greedy in its exploitation of Model 2, which did not allow the robots for optimum exploration in the environment. Note that in all cases the optimized policy from our framework was achieved with 500 simulations, which is equivalent to the number of simulations in just *one* generation of the evolutionary algorithm that was employed for the comparison (the evolution used a population of 100 policies that was evaluated 5 times each). Case study A was evolved in 100 generations, and case studies B1 and B2 were evolved with ≈ 300 generations each. To achieve the best of both worlds, the increased efficiency of the model-based approach can be exploited within the evolutionary setup. This idea will be explored in Section 5.4.6.

A particularly interesting emergent result can be appreciated in the behavior of the robots in case study B1. This case study featured the particular difficulty that the robots were always told to move with forward speed $v_{\text{cmd}} = 0.5\ m/s$, purposely making aggregation difficult due to their inability to stop. The optimized policy, however, learned to fully exploit collision avoidance (i.e., repulsion) forces between robots, which actually created the ability to stop by balancing the commanded forward speed with the repulsion force. By exploiting the repulsion forces, the robots learned that it was possible to remain in equilibrium and readily form clusters. Other clusters were also formed by robots repeatedly moving around each other.

PERFORMANCE FOR CASE STUDY C (FORAGING)

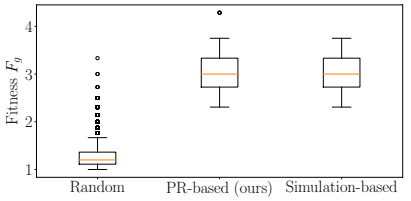
For the foraging task, we found that $\mathcal{S}_{des}$ was only composed of states where $f_g \geq -1.2069$. The swarm's performance is evaluated with 100 simulations of 200 s. The performance can be appreciated in Figure 5.7d and Figure 5.8d, showing that it matches the performance of an evolved policy. In all cases, the fitness improves directly from the beginning of a run and stabilizes after approximately 100 s. From these results we can establish that, even though the correlation of Model 1 was found to be lower than for the aggregation task, the extracted desired states and corresponding optimized policy can still match an evolved policy. This result may be attributed to the accuracy of Model 2, which in Figure 5.6d can be seen to more quickly converge to its final estimates.

5.4.5. UNDERSTANDABILITY AND VERIFICATION ANALYSIS

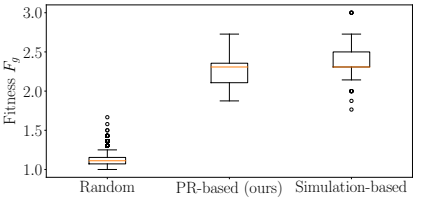
This section discusses how the two models help to better understand the behavior and performance of a swarm. The discussion is divided into four parts. First, we discuss the ability of Model 1 to find suitable desired local states. Then, the policy optimization procedure and its results are analyzed. This is followed by a discussion on the identified livelocks, and subsequently deadlocks, for each case study.

GLOBAL GOAL TO LOCAL OBJECTIVES

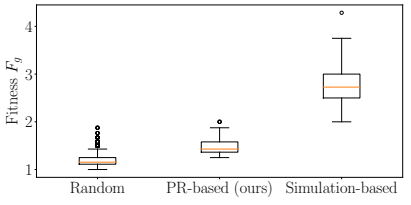
Model 1 is a function that relates the distribution of local states in the swarm to a global performance. When designing the swarm behavior, it is primarily used as a tool to extract the set of local states that are expected to lead to a high performance by the swarm. These local states provide direct insight into the individual goals of the robots and their goals within the swarm. Central to the swarming framework in this chapter is that the



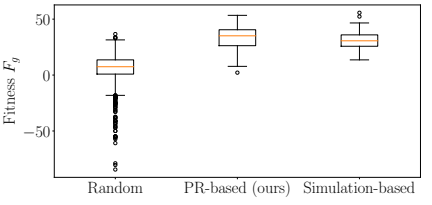
(a) Case study A



(b) Case study B1



(c) Case study B2



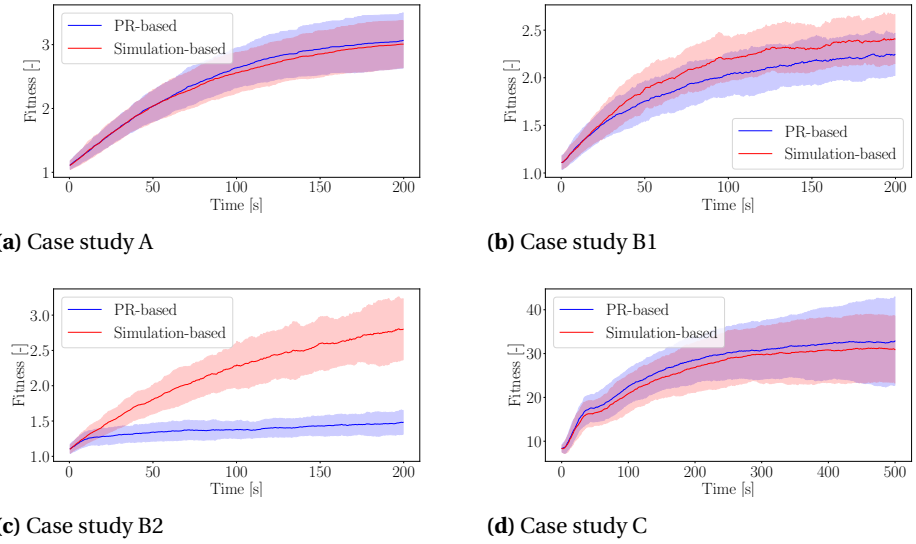
(d) Case study C

Figure 5.7 Box and whisker plots of the fitness achieved by the PageRank-optimized policy (denoted "PR-based") against the performance of random policies (denoted "Random") and optimized policies evolved using a conventional evolutionary algorithm that evaluates the performance via simulation (denoted "Simulation-based").

swarm is optimized by balancing a greedy local behavior with multiple "altruistic" local objectives that benefit the swarm's performance. The desired states for each case study can be seen in green in Figure 5.10. For the aggregation tasks, it is possible to see that all local states with one or more neighbors are regarded as desired. This is a reasonable result, as robots learn to aggregate in small clusters. This behavior is also observed by the policy optimized via standard evolution, due to the time constraint on the simulations. For the foraging task, the set of desired local states include all local states with $f_g > 0$, but also a few local states where $f_g < 0$. This shows that the robot's goal is primarily to maximize the observed local difference, but also that the robots are ready to accept a small (perceived) loss in food at the nest, as this may still be beneficial for the global performance.

POLICY OPTIMIZATION, ANALYSIS, AND INSPECTION

The transition model (Model 2) provides information about the relative likelihood of transitioning to each local state. Through this, we can study the estimated likelihood of reaching a particular local state. Figure 5.9 shows the PageRank score for each individual state, separated into the policy model ($\mathbf{H}_\pi$) and the environment model ($\mathbf{E}$). For the aggregation case studies (A, B1, and B2, in Figure 5.9a, Figure 5.9b, and Figure 5.9c, respectively) it is possible to see that, adopting random policies, robots are most likely to be without neighbors. They are progressively less likely to finish with local states with more neighbors. For the foraging case study, it can be seen that motion by the robots is most likely to cause a positive observation (local states 15-30), whereas the impact of the



(a) Case study A

(b) Case study B1

(c) Case study B2

(d) Case study C

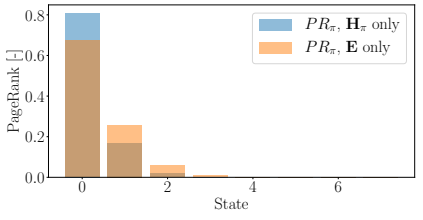
Figure 5.8 Fitness of 100 validation runs using the optimized policies, showing mean and standard deviation. As in Figure 5.7, "PR-based" relates to our PageRank-based optimization, and "Simulation-based" refers to a conventional evolutionary algorithm that evaluates the performance using simulations.

environment is most likely to cause a negative observation (local states 0-15). The estimated effects of the optimized policy on the PageRank score are shown in Figure 5.10 for all case studies. This PageRank analysis is useful to understand the rationale behind the optimized policies. Case study A (Figure 5.9a) is seen to benefit from the environment transitions rather than motion, particularly for states with one or more neighbors. Case studies B1 (Figure 5.9b) and B2 (Figure 5.9c) show that statistically the system is very unlikely to reach any local state with neighbors. This shows that the robot must take certain actions to increase its likelihood of having neighbors, which was observed when we analyzed the optimized behavior. Case study C (Figure 5.9d) shows a contrasting correlation between the action model and the environment model, implying that choosing to explore can increase the probability of having an observation with $f_g > 0$ (local states 15-30), with a significantly higher probability of being just above $f_g = 0$.

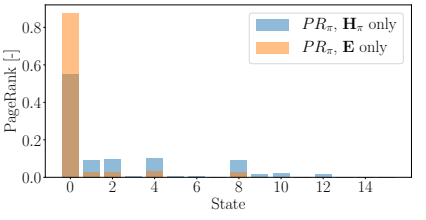
ANALYSIS OF POTENTIAL DEADLOCKS

Deadlocks can be identified by studying the set of static local states as detailed in Section 5.3.4, analyzed here for the four different case studies that have been designed.

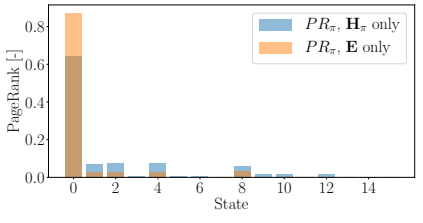
- For case study A, the static local state set is $\mathcal{S}_{static} = \{s_1, s_2, s_3, s_4\}$, corresponding to observations with one to four neighbors. This means that small aggregate clusters will form, due to the limitation of the robot's sensing. This can also be seen to be a likely outcome via Figure 5.9a. Note that a similarly greedy behavior was also observed in the policy that was evolved using a standard methodology, due to the time constraint of 200 seconds on each simulation.



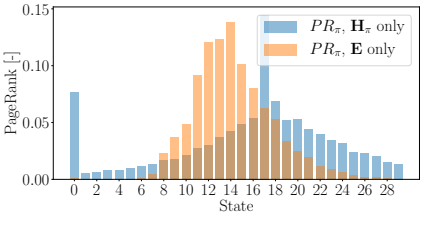
(a) Case study A



(b) Case study B1

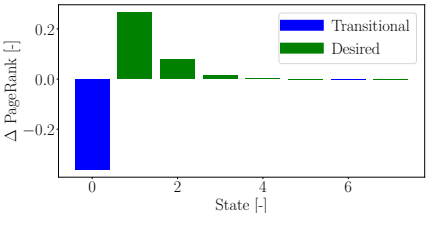


(c) Case study B2

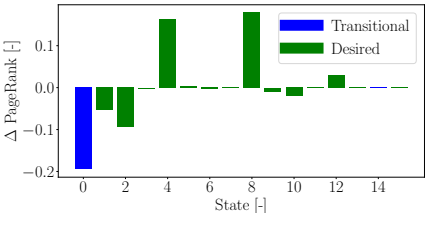


(d) Case study C

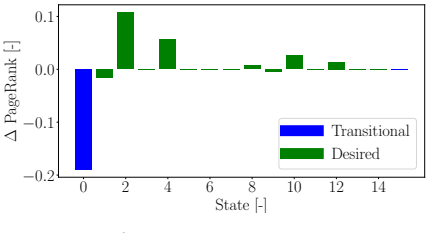
Figure 5.9 PageRank scores of the active model $G_{\mathcal{S}}^{H_\pi}$ and the environment model $G_{\mathcal{S}}^{E}$, from models built using the training data sets.



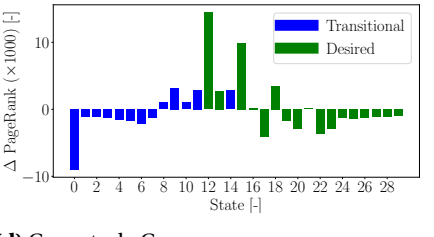
(a) Case study A



(b) Case study B1



(c) Case study B2



(d) Case study C

Figure 5.10 Difference in PageRank scores between the models learned using the training data sets (as in Figure 5.6), and the estimated model at the end of the policy optimization, given $\pi^\star$. We can observe how generally the likelihood for states in the set $\mathcal{S}_{des}$ ("desired" states) increases while the likelihood of the other states ("transitional" states) decreases.

- For case studies B1 and B2, no direct deadlock could be identified directly from the model as the set $\mathcal{S}_{static} = \emptyset$. However, the PageRank analysis of the estimated model reveals that the probability of creating small clusters is high in relation to the probability of creating larger clusters. This can be seen in Figure 5.9 and Figure 5.10, by the significantly larger (change in) PageRank score for those local states (local states s_1, s_2, s_4, and s_8, which only feature one filled sector) in relation to the rest.
- For case study C, the foraging task, deadlock conditions were found from the set $\mathcal{S}_{static} = \{s_0, s_2, s_3, s_4, s_6, s_9, s_{12}\}$. If all robots feature these local states, according to model tuned to the optimized policy, they will not take actions (i.e., explore). In practice, however, these local states are not truly static. They are mitigated by the environment, which will cause the local states to change as the robots reevaluate food at the nest. The only true identified deadlock is then the one where all robots have a local state s_0, and the swarm will not recover.

ANALYSIS OF POTENTIAL LIVELOCKS

Table 5.2 shows the result of checks on the conditions of Propositions 5.3.1 and 5.3.2 to determine potential livelocks from the transition model with policy $\pi^\star$. The results are presented in Table 5.2 and discussed in detail below.

- For case study A, the results of which are given in Table 5.2a, a potential livelock could be identified due to a missing path from local state s_0 (no neighbors) to local states s_3, s_4, and s_5 (three, four, and five neighbors, respectively). This means that, in the event that it is necessary for one or more robots in the swarms to achieve one of these local states in order to complete an aggregate, then this may not happen and the swarm may cycle between and/or through less successful global states.
- For case studies B1 and B2, no potential livelocks have been identified through the conditions. This is because all local states have the potential to transition sufficiently and provide sufficient motion in the swarm.
- For case study C, local states s_0, s_2, s_3, s_4, s_6, and s_9 do not feature paths to local states $s \in \mathcal{S}_{des}$ as a result of exploration outside of the nest. However, the environment changes are sufficient to avoid livelocks. The environment changes in two ways: 1) one or more other robots arrive at the nest with food, such that the robots waiting at the nest perceive a difference in food between their reassessment intervals, or 2) the robots at the nest consume food, which also leads to perceiving a difference in food between reassessments. Both have the potential of liberating the swarm from a livelock.

5.4.6. HYBRID APPROACH WITH AN EVOLUTIONARY ALGORITHM

In this section, the model-based framework is used *together* with an evolutionary algorithm that evaluates the performance of the swarm by means of simulation. Both models required in the framework are trained directly using the simulation runs that are used to evaluate the population of policies. At the end of each generation, the models are trained and a policy, optimized via the PageRank fitness, is generated using these models. One member of the population holds the model-based policy at all times. Using this hybrid

Table 5.2 Results of check for the conditions in Propositions 5.3.1 and 5.3.2 following the policy optimization. No bounds were used for the probabilities in the policy in order to avoid deadlocks and/or livelocks. P 1 refers to the condition of Proposition 5.3.1. P 2.1 and P 2.2 refer to the conditions of Proposition 5.3.2, in order.

Condition	Outcome	Counterexamples	Impact
P 1	False	Missing paths: (0, {3,4,5})	A robot cannot transition from having 0 neighbors to having 3 or more neighbors. This means that, in the event that this is necessary to achieve an equilibrium state at the global level (where having one or two neighbors is not possible), there may be a livelock situation.
P 2.1	True	-	-
P 2.2	False	Missing paths: (0, {3,4,5})	Robots can transition to active local states, but a robot with a local state s_0 will not transition to local states with 3 or more neighbors. As per the result of P 1, this has the potential to create a livelock at the global level.

(a) Case study A, $\mathcal{S}_{static} = \{s_1, s_2, s_3, s_4\}$, $\mathcal{S}_{des} = \{s_1, s_2, s_3, s_4, s_5\}$. No information on s_6 and s_7.

Condition	Outcome
P 1	True
P 2.1	True
P 2.2	True

(b) Case study B1, $\mathcal{S}_{static} = \emptyset$, $\mathcal{S}_{des} = \{s_{1-15}\}$.

Condition	Outcome
P1	True
P2.1	True
P2.2	True

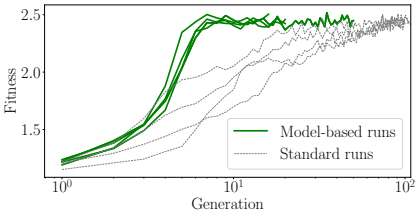
(c) Case study B2, $\mathcal{S}_{static} = \emptyset$, $\mathcal{S}_{des} = \{s_{1-14}\}$.

Condition	Outcome	Counterexamples	Impact
P 1	False	Missing paths: (0, {12,13,15-29}), (2, {12,13,15-29}), (3, {12,13,15-29}), (4, {12,13,15-29}), (6, {12,13,15-29}), (9, {12,13,15-29})	Local states s_0, s_2, s_3, s_4, s_6, and s_9 do not feature paths to $s \in \mathcal{S}_{des}$ as a result of exploration outside of the nest.
P 2.1	True	-	-
P 2.2	True	-	-

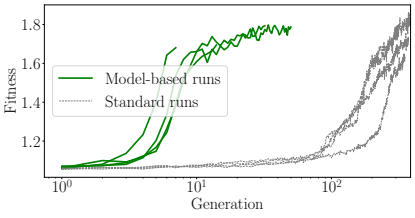
(d) Case study C, $\mathcal{S}_{static} = \{s_0, s_2, s_3, s_4, s_6, s_9, s_{12}\}$, $\mathcal{S}_{des} = \{s_{12}, s_{13}, s_{15-29}\}$.

model-augmented approach, the evolution is found to be more efficient. In addition, the models that are obtained can be a useful side product to investigate and analyze the behavior of the swarm.

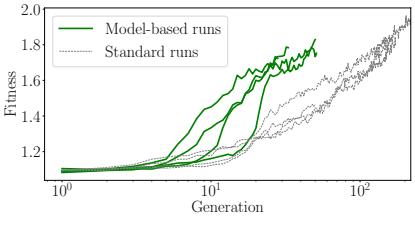
The performance of the hybrid setup is benchmarked against a standard evolution-



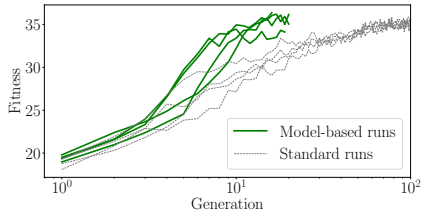
(a) Case study A



(b) Case study B1



(c) Case study B2



(d) Case study C

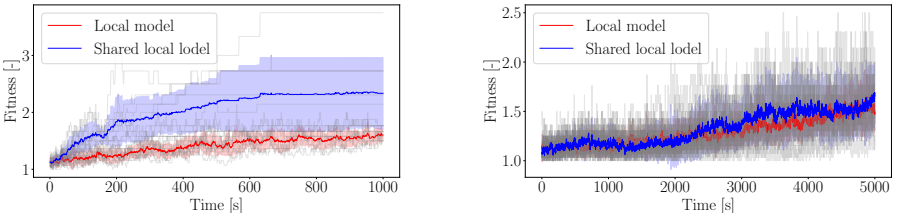
Figure 5.11 Evolution of optimal policies using a standard evolutionary approach and a model-based hybrid evolutionary approach described in Section 5.4.6. In all cases, the model-based evolutionary approach outperforms the learning rate of the standard approach, all other parameters being equal. Note that the horizontal axis is logarithmic.

ary algorithm. All evolutions are implemented using the DEAP package, using a population of 100, each evaluated 5 times, in order to assess the mean performance. The only difference between the two is the inclusion of the model-based approach as described in the paragraph above. The results for case studies A, B1, B2, and C are shown in Figure 5.11a, Figure 5.11b, Figure 5.11c, and Figure 5.11d, respectively. For all case studies, it can be see that the model-based version approaches the optimal performance more quickly than evolution alone. The model-based approach is most impactful in the early stages. Note that this is also the case for case study B2, for which the standalone policy optimization results were found to be less successful.

5.4.7. FRAMEWORK IMPLEMENTATION WITH ONLINE LEARNING

This section briefly explores how the proposed framework can also be combined with online learning. The transfer learning properties of Model 1 allow the extraction of an adequate set $\mathcal{S}_{des}$ for a given task. Given $\mathcal{S}_{des}$, during the online learning phase, the robots only need to estimate the local state transfer model (Model 2) and optimize their policy accordingly. As the transition model is local to the robots, it can be estimated online by each robot. Two variations are explored:

1. *Fully local heterogeneous models.* In this case, each robot estimates the transition model based on its own experiences. In turn, this leads to heterogeneous policies where each robot attempts to achieve the best performance according to its own transition model.



(a) Case study A **(b)** Case study B1

Figure 5.12 Global performance with online learning during a single run, with no initial transition model. The mean performance is shown by the colored lines, with the margins indicating the standard deviation. The surrounding gray lines show individual runs. For reference, as it can also be seen in Figures 5.7 and 5.8, an evolved run of Study Case A reached, on average, a $F_g(t) \approx 3.0$ at $t = 200$ s, and an optimized run of Study Case B1 reached, on average, a $F_g(t) \approx 2.5$ at $t = 200$ s. Note, however, that the results in this figure are from stand-alone runs, each with a blank model at time $t = 0$ s.

5

2. *Shared local model.* In this case, all robots combine their experiences into a shared transition model. This leads to homogeneous behaviors as all robots base their policy on the same model.

The approach was implemented and studied for case studies A and B1. In all cases, no pre-trained version of Model 2 was used. Model 2 was estimated entirely online. The performance of the strategy was tested for case studies A and B1 over ten independent runs. The results are shown in Figures 5.12a and 5.12b. It can be seen that the robots in the swarm learn better policies that lead to a higher performance over time, as the model improves. As is particularly noticeable for case study A, this is most effective for the shared local model, given that it can converge toward reliable estimates in less time.

5.5. DISCUSSION ON THE PROPOSED FRAMEWORK

This section discusses the advantages and limitations of the model-based framework proposed in this chapter. This is done in Section 5.5.1 and Section 5.5.2, respectively. Where relevant, ideas and avenues for future research in the context of swarm robotics research are provided.

5.5.1. ADVANTAGES AND INSIGHTS

- **Learned micro-macro model.** To the best of our knowledge, this is the first time that a model of the swarm's micro-macro relationship has been learned from data, and then used as a tool to automatically design a swarm behavior. The advantages of this are: 1) it is possible to predict the swarm's global performance from the (distribution of) local states, and 2) it is possible to extrapolate the local states that increase the global performance. In particular, a deep neural network can model a complex relationship directly from data, without a priori knowledge. The micro-macro function has shown to be scalable to different swarm sizes, and transferable across swarms and environments. This makes it possible to reuse or transfer the model without having to train new models every time. Such properties could mo-

tivate a larger endeavor by the research community to produce highly accurate micro-macro models and/or training datasets for common swarming tasks. For common tasks, datasets can be constructed that could become useful across the research community for purposes extending beyond the one in this work. Similar community-wide efforts have been successful in other fields, such as computer vision, which now holds a plethora of publicly available and high quality models and datasets.

- **Transparent transition model.** The transition model was used to model the expected local state transitions experienced by the robots. A PFSM architecture was chosen specifically to increase transparency and traceability. It was readily used for policy optimization and analysis. Its transparency enabled us to inspect and predict elements of a swarm's behavior, without solely relying on empirical testing.
- **Sample efficient policy optimization.** The number of simulations required to train Model 1 and Model 2 were comparable with the ones required by a single generation of a standard (i.e., simulation-based) evolutionary approach. The reason for this sample efficiency is that the model-based framework exploits all the events in a simulation run, rather than only assessing its cumulative/final performance. The hybrid setup introduced in Section 5.4.6 provides an interesting way to combine the benefits of both approaches.
- **Online model-based learning.** Online learning and adaptability are complex challenges for swarm robotics. Using the approach in this chapter, the robots can generate a model of their experiences and adapt their behavior accordingly, without any a priori knowledge. Although the robots act greedily with respect to their own local model (as no global information is available) their common local objectives can guide them toward a high global performance. One primary issue is the balance between exploration and exploitation, as exploration implies slower convergence, yet exploitation implies that the solution will likely be subpar. In practice, online learning may be best used as a fine-tuning approach for the purposes of adaptability, whereby the robots are loaded with a transition model that is fine-tuned online.

5.5.2. LIMITATIONS AND POTENTIAL SOLUTIONS

- **Difficulties with delayed/temporal effects.** The micro-macro model implemented in this work faced issues with temporal effects. These were apparent with case study C (foraging). The temporal effects in case study C were the results of the following aspects.
  1. The value of the global fitness $F_g(t)$, given the same vector $\mathbf{P_s}(t)$, depends on prior actions/global states.
  2. As food is replenished in the environment, a bias forms toward areas that are further from the nest and/or harder to reach. This bias increases over time, as the robots first find the food near the nest before venturing out further.

  These effects could not be captured by the feed-forward setup of Model 1. Temporal properties could be represented using a recurrent neural network instead of a feed-forward one. However, this would also require changes in the remainder of

the framework.

- **Inaccurate transition model.** An inaccurate transition model can be responsible for an inaccurate assessment of the system as well as a policy with poor performance. One solution is to improve the model via additional simulations in order to improve the estimates, potentially also building on other models of similar systems, if available. In addition, it is possible for the robots to fine-tune the model online, and then re-optimize their policy accordingly. Fine-tuning a model online can also help to overcome situations that were not simulated or modeled. Alternative model structures should also be investigated aside from the tabular model used here.
- **Sub-optimal policy optimization.** Not all case studies matched the performance of the standard evolutionary procedure. This is because the performance achieved by the policies is inherently sub-optimal as a result of a policy optimization based on Model 2. Whereas a global optimizer can tune to the specific environment, this cannot be directly done when optimizing based on a local model. The policy optimization, which is based on a local model, prompts greedy behaviors by the individual robots. However, the common local objectives, which maximize the global performance, can guide this behavior to a behavior that benefits the swarm as a whole. In addition, temporal aspects pertaining to the performance are not modeled. This means that the robots aim to achieve their objectives as quickly as possible, whereas a global optimizer may optimize the behavior for the time given to the task. Finally, the transition model assumes that the other robots in the swarm behave in a certain way, which is an assumption that is violated once the policy is optimized and the behavior of all robots changes.
- **State space and action space scalability in the transition model.** The discretized nature of PFSMs has two limitations: 1) it prevents continuous states from being used, and 2) it is subject to scalability issues as the size of the local state space and action space increases, which increases the PageRank evaluation time. Approximate model architectures can help to mitigate this issue and should be the subject of future research.

5.6. CHAPTER CONCLUSIONS

This chapter proposed a model-based framework to extract local behaviors for robotic swarms. This framework uses a neural network model to estimate the effect of local states on the global performance. As this model is a direct mapping of local states to global performance, it is not dependent on the dynamics of the swarm and it can be potentially transferred across different swarms and environments. The model is used to find desired local states, i.e., local states that are expected to maximize the performance of the swarm. A second model, a probabilistic model of the local state transitions that a robot in the swarm can experience, is then used to optimize and analyze the behavior. The complexity of this second step does not increase with the number of robots in the swarm, which is an attractive property. Overall, the behavior can be found very efficiently. Together with the experience necessary to learn the models, it approximately requires the equivalent of a single generation in an evolutionary algorithm. This model-based framework provides a way to develop behaviors that are understandable and to

which we can also apply verification checks in order to determine possible pitfalls.

Future work should be focused on experimenting with alternative model architectures, potentially solving problems pertaining to temporal aspects as well as discretization. This, however, should be done while still keeping transparency and verifiability in mind. Additionally, we encourage further investigations into how the framework can be integrated within current practices, building on the setups explored in Section 5.4.6 and Section 5.4.7. In the hybrid evolutionary setup, for example, the models allowed a performance improvements with marginal computational increase, as the framework directly uses the simulation logs used to evaluate a controller in order to construct and use the models. Further evaluations need to be performed in order to better understand the limitations and advantages of the framework within these contexts. Finally, we believe that a community-wide effort to build high quality models and datasets for swarms (robotic, but also natural) could be useful across the research community, for purposes extending beyond the scope of this work.

CODE

- The code used in this chapter can be downloaded at `https://www.github.com/coppolam/SI_framework`. This also includes scripts that will automatically download and setup the simulator and the data.
- The simulator can be downloaded separated at `https://github.com/coppolam/swarmulator/tree/SI_framework`
- All data used in this chapter can be downloaded at: `https://surfdrive.surf.nl/files/index.php/s/DtU5rW7za4DeNb5`.

6

CONCLUSION

6.1. SUMMARY

The objective of this work was to develop a novel methodology to automatically design understandable, verifiable, and efficient swarm behaviors. A focused literature review in Chapter 2 revealed multiple links that relate low level design choices, including hardware design, to the swarm behaviors that can be performed as a result. The links go beyond defining the local sensory knowledge that is available to each robot, but can also dictate primitives about what the robots should be doing. Coincidentally, we found a prominent example of this in our own work (Coppola et al., 2018; Li et al., 2020; van der Helm et al., 2020), performed in auxiliary to the work in this PhD thesis, where we developed a communication-based relative localization approach to enable swarm behaviors for multiple tiny drones. Based on these insights, our strategy then became to develop a systematic framework that would allow us to break down a global goal into local constituents, aiming for a general framework that can be applied across several swarm behaviors. This strategy resulted in three novel developments:

1. A systematic framework to break down a global goal into locally observable constituents, developed for the purpose of pattern formation in Chapter 3.
2. A model-based approach to optimize a swarm behavior based on PageRank centrality by maximizing the probability of robots to be in a set of desired local states, presented in Chapter 4.
3. A data-driven model-based framework to automatically extract local desired states, local state transition models, and learn transparent swarm behaviors, presented in Chapter 5.

These developments were primarily centered around the following concept: that a global goal can be broken down into a set of local states, and that, if all robots aim to be in any of these local states, then the global goal is achieved. This idea was generalized to multiple tasks and also used in reverse: from the local states, it was possible to make predictions about the global result. This could be done either through formal checks, as in Chapter 3, or approximated by deep neural networks, as in Chapter 5. In the remainder of this conclusion chapter we review the outcomes of the research questions introduced in Chapter 1 and we provide insights and strategic recommendations for future research.

6.2. ANSWERS TO RESEARCH QUESTIONS

Research Question 1: What are the challenges of developing a successful swarm of robots?

Several challenges are encountered when developing a swarm of robots, as low level design choices will influence the capabilities of the swarm. It is important that each robot is designed such that it can navigate by itself, according to specifications, and without causing damages to itself or the environment. It is also important to ensure that it has the necessary sensors to perform the expected functions. Next, the robots must be capable of sensing each other. This is necessary for intra-swarm collision avoidance, but also to potentially enable desired collective behaviors. The method/sensor chosen may dictate the behavior of the robots, and as such it may impose limitations on the higher level swarm behavior that can be achieved. Finally, developing the collective swarm behavior can be done using either a *manual* or an *automatic* approach. The latter can absolve the designer from having to thoroughly understand the intricacies of the swarm's interactions. Machine learning, most commonly evolutionary learning, can be used to find a suitable swarm behavior. However, this creates the challenge of understanding what the machine learning algorithm has learned and validating its performance.

Research Question 2: How can a swarm of cognitively limited robots arrange in an arbitrary formation, in a way that provably leads to success?

A global formation can be broken into local observations of relative locations, named *desired* local states. These local states are a partial observation of the global goal. In this case, they are local neighbor configurations that, placed together, will reconstruct the desired global pattern (analogous to pieces of a puzzle). In analogy to biological systems, the robots in the swarm can be made to follow three rules: 1) be "safe" (avoid collisions), 2) be "social" (do not leave the group), and 3) be "happy" (remain in one of the desired local states). By following these three rules, the pattern eventually forms from random initial configurations. Based on a model of transitions, it is possible to verify whether the swarm will eventually transition into the final pattern, or whether it is possible that it will endlessly reshuffle. Based on the assembly of the local states, it is also possible to find whether other spurious patterns may form. When this is the case, it means that the onboard sensors are insufficient to uniquely form the desired goal, and a redesign must be performed.

Research Question 3: How can a swarm of robots coordinate to achieve a global goal efficiently?

In the previous solution, it was found that a global goal can be achieved by guiding all robots to be in a set of desired local states. This behavior can be optimized efficiently using a model-based approach. Our specific solution was inspired by the PageRank framework, originally used by Google to rank Web pages. This framework is particularly attrac-

tive because it offers the ability to model the local experiences of a robot in the swarm, and probabilistically analyze the impact of a policy. The PageRank model has a bipartite structure which allows us to differentiate between local state transitions due to a robot's own policy (analogous to following hyperlinks), and local transitions due to the environment (analogous to directly going to a specific Web page without the use of hyperlinks). Combined with an evolutionary framework, it is possible to also explore a more complex solution space that includes constraints on the behavior.

Research Question 4: How can we design the behavior of a swarm of robots such that it reliably achieves a cooperative goal?

Breaking down a global goal into desired local states can be achieved using a model-based deep learning approach, where the model is automatically trained from simulated and/or real-world data. Using this approach, it is possible to reconstruct the global performance trends from a distribution of local states. In addition, a local transition model can also be explicitly learned. Overall, these models provide transparent building blocks to 1) automatically determine the desired local states for a swarming task, and 2) automatically optimize the probability of the robots to be in these desired states. This can be shown to lead to efficient cooperative behavior on diverse tasks and setups. The models also provide transparency and the ability to inspect properties of the swarm. This work shows that it is possible, in different scenarios, to use deep learning to learn to map the complex relationship between local sensory data and global goal. The model-based framework can be used in multiple architectures, such as during evolutionary learning or during online learning.

6.3. CONCLUSIONS AND INSIGHTS

The work performed in this thesis leads to multiple conclusions and insights that help to answer the guiding research question, restated here.

Guiding research question: How can we automatically design a swarm of robots to systematically and efficiently produce a desired emergent result?

One of the main insights has been how to break down global goals into desired local states, showing that this can be achieved in a structured and intuitive way, and even using deep learning approaches. Thanks to this, it is possible to simplify or even automate the process of finding successful behaviors, while also keeping a clear understanding of what the local objectives of the robots are. A second insight is that to ensure that the goal will always eventually be achieved requires maintaining a degree of randomness in the swarm. A nonrandom behavior is highly likely to end up in livelocks or deadlocks, which is not desired. Because of this randomness, there is also some inherent inefficiency that robot swarms must accept. Efficiency is traded for scalability, flexibility, and robustness to initial conditions or unexpected events. Finally, a model-based approach, based on a local model of local state transitions, is a powerful tool that allows us to optimize, inspect, and verify the behavior of a swarm. This model can be automatically extracted

and used as a behavior optimization tool. Combining all the tools above, it is possible to achieve efficient and transparent swarm behaviors with a clear mapping between local controllers and global behaviors.

6.4. OUTLOOK

The challenges of swarm robotics primarily stem from the complex relationships between the robots' onboard sensors, the robots' capabilities, and the swarm's global objective. Following the findings in this thesis, we have identified the following main directions for future research in the field.

EXPLORING THE DESIRED STATES FRAMEWORK

The idea that a global goal can be represented as a set of desired local states has been central to this work. It was shown that it can also be used to optimize the behavior of a swarm and verify global properties. We recommend the further exploration of this framework in future research. It provides an intuitive and logical breakdown for homogeneous robotic swarms. It does not necessarily need to be used for development, but it can also be used as a tool to analyze an existing behavior. There are still steps to be taken in order to generalize the framework further. For example, swarming tasks may require (or benefit from) a temporal specification of desired local states. This has not been considered in the current approach, but it can unlock additional application domains.

FURTHER STUDIES WITH MODEL-BASED APPROACHES

Model-based approaches have shown to be valuable for understanding, optimizing, and verifying swarm behaviors. When possible, the models can be made by hand, as we have done in Chapter 3 and Chapter 4. However, it required a relatively deep insight into the problem and making certain system level assumptions. The strategy that we finally adopted in Chapter 5 was to use data in order to automatically extract the models. This has proven to be a more general strategy. Future works should focus on tackling the scalability issues that can arise with the transition models without losing the understandability and verifiability properties. In addition, model-based approaches can also go beyond tabular models as we have done in this research. Although these are less explainable, they offer significant advantages in terms of continuous function approximation, and we encourage the community to explore them. In recent work, building on the ideas presented in this thesis, we also explored the use of a model-based approach (with a learned generative world model) in order to improve the sample efficiency of multi-robot reinforcement learning (Willemsen et al., 2021). Using the generative model, the proposed reinforcement learning algorithm was capable of learning behaviors more efficiently than its model-free counterpart. Finally, further studies with deep learning models predicting macroscopic swarm performance from local sensor data are also encouraged, including the development of the accompanying data sets to train such models.

MORE INCORPORATION OF REINFORCEMENT LEARNING TECHNIQUES

Reinforcement learning can be a computationally more efficient strategy than evolution as it can extract rewards from each single time step, rather than at the end of a simulation run. However, one of the reasons that reinforcement learning is less often used in

swarming is because of the difficulty in correlating a global goal to local rewards, specifically for the case of decentralized learning. In this work, we have developed a framework that can automatically perform this breakdown and extrapolate goals from poorly performing behaviors. By adopting this strategy, we expect that reinforcement learning can be guided toward achieving the global objective. Hybrid approaches, which combine reinforcement learning with evolutionary learning, should also be considered, combining a local behavior optimization with a global optimizer, for higher efficiency and adaptability.

PERFECTING THE "BALANCE" BETWEEN RANDOMNESS AND EFFICIENCY

Swarm robotics requires a certain degree of randomness. For example, in Chapter 3, we showed that a level of randomness is required to avoid livelocks and deadlocks. By optimizing a behavior, we inherently trade in this randomness for a statistically more efficient behavior. A viable strategy to raise efficiency while keeping flexibility is to incorporate online learning, thus allowing the robots to adapt their behavior to the current situation. A model-based approach could prove to be more efficient by optimizing a behavior based on a pre-trained probabilistic model of the environment which is then updated online. The framework that has been proposed in this paper provides the tools to allow this to happen in a transparent way, while also providing explicit means to put boundaries on the optimization process (e.g. verification conditions) such that sufficient randomness remains in the swarm in order to be scalable, flexible, and robust.

A

APPENDIX: SWARMULATOR

Swarmulator is a lightweight C++ simulator designed to prototype, test, and analyze the behavior of a swarm of robots. The simulator can be downloaded, compiled, and executed on Linux platforms. Swarmulator addresses the need for a lightweight C++ simulator that, while being higher level, can also be easily customized. Users can enter their own dynamics and controllers as desired. Its multi-threaded architecture enables the simulated robots to act as independent agents, each handled by a dedicated detached thread. Being lightweight, it is also possible to run multiple simulations at the same time. The code can be accessed at `https://coppolam.github.io/swarmulator/`.

A

A.1. BACKGROUND

Simulation plays a key role in understanding, devising, verifying, and even validating swarm behaviors. Simulation is also key for machine learning tools such as evolutionary robotics or reinforcement learning, enabling them to evaluate a system's performance and find optimal policies. Repeated simulations are often needed in order to quantify the performance of a controller without over-fitting or bootstrapping.

Swarmulator is a lightweight simulator, built with C++, designed to efficiently prototype and test swarm behaviors. The driving requirements behind Swarmulator were: 1) it should be able to simulate each robot independently and quickly, and 2) it should be easy to customize the controllers, dynamics, sensors, and environment. A screenshot of Swarmulator's animation window is shown in Figure A.1. It was a valuable prototyping and testing tool throughout a large part of the research performed in this thesis. A C++ simulator also makes it easy to transfer code to and/or from robots, using the simulation as a verification and pre-validation tool. This strategy was useful during some of our communication-based relative localization works discussed in Chapter 2.

In this appendix chapter, we introduce Swarmulator and its main properties. Section A.2 briefly reviews swarm simulators in the state of the art and explains the niche covered by Swarmulator. Section A.3 introduces the higher level architecture of the simulator. Section A.4 explains, at a higher level, how one can use Swarmulator as a prototyping tool. Section A.5 gives benchmarks of the expected performance on a conventional laptop.

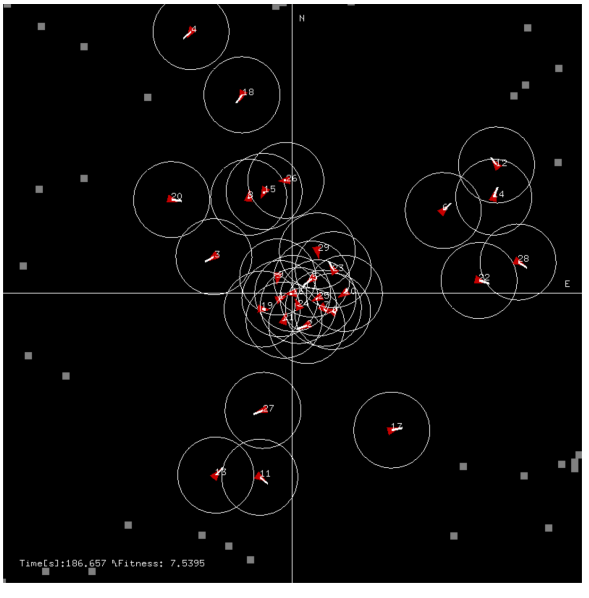


Figure A.1 Screenshot of Swarmulator during a foraging simulation. The red triangles are the robots and the white circles indicate their sensing range. The gray squares are food items. The nest is located at the center of the arena. A user can interact with the window by adding additional robots, additional walls, changing the real-time simulation factor, and more.

A.2. RELATED WORKS

Robotics Operating System (ROS) (Quigley et al., 2009) is one of the leading frameworks for interfacing with robots. It provides a modular architecture that can be easily inspected and altered, provided that input-output messages (topics) across its modules (nodes) are well maintained. This also means that it can be easily interfaced with dedicated robotics simulators, a popular example of which is Gazebo (Koenig and Howard, 2004). This setup, or other high fidelity alternatives, can be attractive for single robots, yet can be limiting for multi-robot domains, primarily because of the computational load required. In certain cases, such as for full end-to-end learning where we wish to map visual inputs directly to controller parameters, then high fidelity simulation environments may be desired. However, when learning higher level swarming behaviors with abstractions from the sensors, as can often be the case (Brambilla et al., 2013), then we can adopt lower fidelity solutions to achieve our results more efficiently.

Within the swarming field, ARGoS (Pinciroli et al., 2012) is among the best known simulation environments. Similarly to Swarmulator, it has also been designed with swarming in mind, albeit at a higher fidelity level. With Swarmulator, we aimed at a very small codebase that would still allow customization for complex dynamics and controllers. A simulator like Roborobo (Bredeche et al., 2013) offers a similar concept as Swarmulator. It is C++ based and designed to be simple and efficient. However, unlike Swarmulator, it is not modular and constrains the user to an e-puck or Khepera robot. Recently, Soria et al. (2020) deployed Swarmlab. This is a swarm simulator, specifically for drone swarms, based on Matlab. However, this may make it more difficult to transfer code to real-world robots. Another attractive feature of Swarmulator is that it allows the user to interact with the simulator in real time. The user can, for instance: control one of the robots, add robots to the swarm, or create new walls. This is often useful to better understand a swarm behavior.

A.3. DESCRIPTION

Swarmulator is based on C++ code, which has notable advantages:

- It can be easily transferred to/from real robots;
- Higher computational speeds;
- Modularity via the object oriented architecture.

The simulation operates using detached threads, as depicted in Figure A.2. There are three high level threads (simulation, animation, and logging). In addition, the simulation thread launches additional detached threads, each simulating one robot in the swarm. The detached thread architecture means that each robot operates according to its own thread clock without an enforced synchronization or order. There is no hierarchy between threads. The threads can access relevant simulation data using shared memory mutexes, according to the C++17 standard. Shared mutexes are read-only and can thus be owned simultaneously by multiple threads.

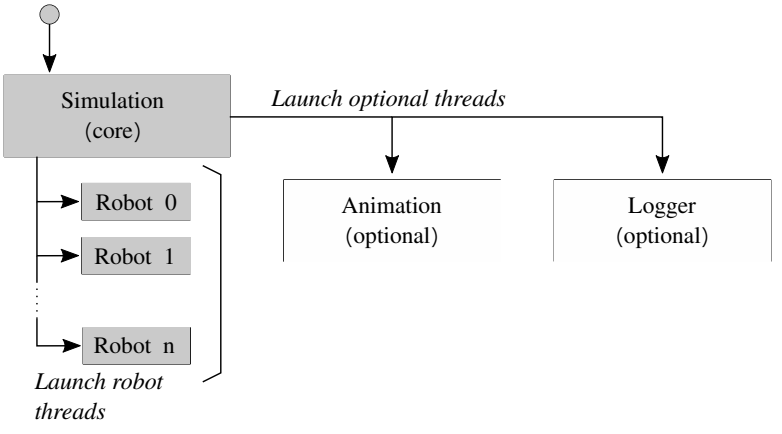


Figure A.2 Breakdown of Swarmulator threads. A simulation thread acts as a core thread and launches separate detached threads, one for each robot in the swarm. Optionally, two additional threads can be launched — one that handles the animation and one that handles logging.

A.3.1. SIMULATION THREAD (CORE THREAD)

This thread handles the simulation. It sets up the environment, launches all other threads, and handles the program termination. For example, if a maximum simulation time is specified, then the simulation thread will write the final fitness to a First-In First-Out (FIFO) pipe prior to automatically quitting the simulation. This allows Swarmulator to communicate its performance with external programs. In Section A.4.4, a Python Application Programming Interface (API) is presented which can launch a simulation and receive the final fitness from the FIFO pipe. This can be used by an external Python program, such as an evolutionary algorithm, to develop the behavior while using Swarmulator as the evaluation tool. Each simulation has a unique ID, meaning that multiple simulations can be launched simultaneously from the API.

A.3.2. ROBOT THREADS

Each robot is simulated by an independent thread. These threads are launched at the beginning by the core simulation thread, but they can also be launched by a user via the animation window. Each thread is responsible for the simulation of one robot in the swarm. The thread updates the dynamics with a specified frequency until the core thread quits. The dynamics and control are fully customizable, as detailed in Section A.4.

A.3.3. ANIMATION THREAD

The animation thread serves two functions: 1) real-time depiction of the swarm, and 2) user interaction. The real-time depiction, currently implemented via OpenGL, displays the swarm in a separate window. Standard visual user interaction functions such as drag and zoom are implemented. In addition, the user can use this window to interact with the simulation. For example, in the current implementation, the user can launch a new robot at a desired location, create walls in the environment at desired locations, change the real-time factor of the simulation, and pause or restart the simulation. These user

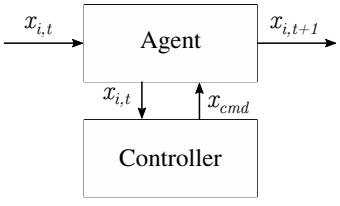


Figure A.3 In this figure, $x_{i,t}$ is the state of robot i at time t, x_{cmd} is the command given by the controller, $x_{i,t+1}$ is the state of robot i at the next time step. Both the controller and the agent are customizable.

interaction functions are designed to allow the user to understand and explore a swarm's behavior. The user can easily modify the environment to see how a swarm will react, or can scale up the swarm to see how the swarm will behave as more robots are introduced. Changing the real-time factor or restarting a simulation also provide a way to quickly gather insight over the behavior. Launching the animation thread is optional.

A.3.4. LOGGER THREAD

The logger thread logs data to a text file at a specified frequency, specified at launch time. In the default implementation, the position of each robot is logged so that a simulation can be reconstructed. Launching the logger thread is optional.

A.4. PROTOTYPING

Swarmulator is primarily designed for easy prototyping. A user can define the desired robot dynamics and the controller by creating child classes for two main parent classes: "agent" and "controller". Figure A.3 depicts the basic setup.

A.4.1. AGENT CLASS

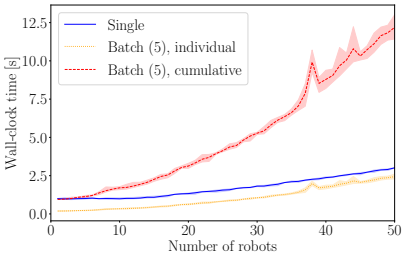
An agent class holds the dynamics of a robot. It runs two core functions:

1. A function to update the dynamics (including potential lower level controls)
2. A function to draw the robot according to the user's taste.

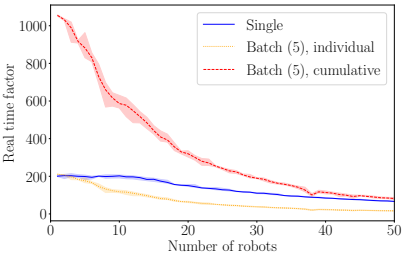
The user needs to define both functions in the child class. The first function specifies the discrete dynamics of the robot. This function can launch the controller's function to receive a desired command x_{cmd}, and then updates the dynamics accordingly. The second function is adopted by the animation class to draw the robot. By default, if using the helper bash script to construct the controller child class, a robot will be drawn as a circle. The second function can also be left empty if animation is not required or desired.

A.4.2. CONTROLLER CLASS

A controller class has one main function that processes the behavior of the robot. This includes any higher level behavior, including obstacle avoidance and intra-swarm avoidance. The function takes in references to the desired values and updates them accordingly to the desired value. The controller class can access all state variables of the robots



(a) Speed benchmark showing the evaluation time of simulations for increasing number of robots, comparing single simulations to batch simulations (5 simulations in parallel).



(b) Real-time factor of simulations launched from the Python API, comparing single simulations to batch simulations (5 simulations in parallel).

Figure A.4 Simulation speed reference benchmarks, simulating the dynamics at 25 Hz, with a maximum real-time factor of 200 for 200 s of simulated time. For each line, the average and standard deviation over 5 runs is shown. The results were obtained on a Dell Latitude 7400 laptop with Intel®Core™i7-8664U CPU @ 1.9 GHz × 8, Intel®HD Graphics 620 (WHL GT2), 8 GB RAM, equipped with SSD, and running Ubuntu 18.04.

and the environment. This allows sensor modeling by only using the information that one desires to be sensed.

A.4.3. RUNTIME PARAMETERS

Swarmulator uses an XML file (`parameters.xml`) to import a set of parameters at runtime. These parameters enable a user to launch Swarmulator with different setting without building the code again. This was inspired by the launch files used by ROS. The file can be customized to include any parameter that may be useful to a particular implementation of a controller and/or agent. Furthermore, XML parameters can be readily modified from the command line, which allows easy interfacing using an external API from external software that wants to launch Swarmulator.

A.4.4. PYTHON API

Interfacing with a Python API is an auxiliary yet key part of Swarmulator. The API can launch (multiple) simulations and receive their performance and/or logs. Then, these can be natively used by a Python program for purposes of machine learning, benchmarking, analysis, and more. This was used in Chapter 5.

A.5. PERFORMANCE BENCHMARKS

For reference, we tested the current performance of Swarmulator using a basic attraction, repulsion, and aggregation controller in an enclosed arena (including wall avoidance). The results, featuring increasing number of robots, can be seen in Figure A.4a and Figure A.4b.

REFERENCES

D. Abeywardena, S. Kodagoda, G. Dissanayake, and R. Munasinghe. Improved state estimation in quadrotor MAVs: A novel drift-free velocity estimator. *IEEE Robotics Automation Magazine*, 20(4):32–39, 2013.

M. Achtelik, S. Weiss, M. Chli, F. Dellaerty, and R. Siegwart. Collaborative stereo. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2242–2248, San Francisco, CA, USA, 2011.

M. (Markus) Achtelik, M. (Michael) Achtelik, Y. Brunet, M. Chli, S. Chatzichristofis, J. Decotignie, K. Doth, F. Fraundorfer, L. Kneip, D. Gurdan, L. Heng, E. Kosmatopoulos, L. Doitsidis, G. H. Lee, S. Lynen, A. Martinelli, L. Meier, M. Pollefeys, D. Piguet, A. Renzaglia, D. Scaramuzza, R. Siegwart, J. Stumpf, P. Tanskanen, C. Troiani, and S. Weiss. SFly: Swarm of micro flying robots. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2649–2650, Vilamoura, Portugal, 2012.

M. H. Afzal, V. Renaudin, and G. Lachapelle. Magnetic field based heading estimation for pedestrian navigation environments. In *2011 International Conference on Indoor Positioning and Indoor Navigation*, pages 1–10, Guimaraes, Portugal, 2011.

W. G. Aguilar, V. P. Casaliglla, and J. L. Pólit. Obstacle avoidance for low-cost UAVs. In *2017 IEEE 11th International Conference on Semantic Computing (ICSC)*, pages 503–508, San Diego, CA, USA, 2017.

K. Alexis, G. Nikolakopoulos, A. Tzes, and L. Dritsas. Coordination of helicopter UAVs for aerial forest-fire surveillance. In K. P. Valavanis, editor, *Applications of Intelligent Control to Engineering Systems*, pages 169–193. Springer Netherlands, Dordrecht, 2009.

J. Allred, A. B. Hasan, S. Panichsakul, W. Pisano, P. Gray, J. Huang, R. Han, D. Lawrence, and K. Mohseni. Sensorflock: An airborne wireless sensor network of micro-air vehicles. In *Proceedings of the 5th International Conference on Embedded Networked Sensor Systems*, SenSys '07, page 117–129, New York, NY, USA, 2007. Association for Computing Machinery.

J. Alonso-Mora, T. Nägeli, R. Siegwart, and P. Beardsley. Collision avoidance for aerial vehicles in multi-agent scenarios. *Autonomous Robots*, 39(1):101–121, 2015.

H. Alvarez, L. M. Paz, J. Sturm, and D. Cremers. *Collision Avoidance for Quadrotors with a Monocular Camera*, pages 195–209. Springer International Publishing, Cham, Switzerland, 2016.

C. Ampatzis, E. Tuci, V. Trianni, and M. Dorigo. Evolution of signaling in a multi-robot system: Categorization and communication. *Adaptive Behavior*, 16(1):5–26, 2008.

I. Aoki. A simulation study on the schooling mechanism in fish. *Nippon Suisan Gakkaishi*, 48(8):1081–1088, 1982.

D. J. Arbuckle and A. A. G. Requicha. Self-assembly and self-repair of arbitrary shapes by a swarm of reactive robots: Algorithms and simulations. *Autonomous Robots*, 28(2): 197–211, 2010.

D. J. Arbuckle and A. A. G. Requicha. Issues in self-repairing robotic self-assembly. In R. Doursat, H. Sayama, and O. Michel, editors, *Morphogenetic Engineering: Toward Programmable Complex Systems*, pages 141–155. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

F. Augugliaro, S. Lupashin, M. Hamer, C. Male, M. Hehn, M. W. Mueller, J. S. Willmann, F. Gramazio, M. Kohler, and R. D'Andrea. The flight assembled architecture installation: Cooperative construction with flying machines. *IEEE Control Systems Magazine*, 34(4):46–64, 2014.

A. Bachrach, S. Prentice, R. He, and N. Roy. RANGE–Robust Autonomous Navigation in GPS-denied Environments. *Journal of Field Robotics*, 28(5):644–666, 2011.

M. Basiri. *Audio-based Positioning and Target Localization for Swarms of Micro Aerial Vehicles*. PhD thesis, Lausanne, France, 2015.

M. Basiri, F. Schill, D. Floreano, and P. U. Lima. Audio-based localization for swarms of micro air vehicles. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4729–4734, Hong Kong, China, 2014.

M. Basiri, F. Schill, P. Lima, and D. Floreano. On-board relative bearing estimation for teams of drones using sound. *IEEE Robotics and Automation Letters*, 1(2):820–827, 2016.

R. W. Beard. State estimation for micro air vehicles. In J. S. Chahl, L. C. Jain, A. Mizutani, and M. Sato-Ilic, editors, *Innovations in Intelligent Machines - 1*, pages 173–199. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

M. A. Bedau. Artificial life: Organization, adaptation and complexity from the bottom up. *Trends in Cognitive Sciences*, 7(11):505 – 512, 2003.

G. Beni. From swarm intelligence to swarm robotics. In E. Şahin and W. M. Spears, editors, *Swarm Robotics*, pages 1–9, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

A. Bensky. *Short-range wireless communication*. Newnes, Cambridge, MA, USA, 2019.

S. Berman, Á. Halász, V. Kumar, and S. Pratt. Algorithms for the analysis and synthesis of a bio-inspired swarm robotic system. In E. Şahin, W. M. Spears, and A. F. T. Winfield, editors, *Swarm Robotics*, pages 56–70, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

S. Berman, Á. Halász, M. A. Hsieh, and V. Kumar. Optimized stochastic policies for task allocation in swarms of robots. *IEEE Transactions on Robotics*, 25(4):927–937, 2009.

S. Berman, R. Nagpal, and Á. Halász. Optimization of stochastic strategies for spatially inhomogeneous robot swarms: A case study in commercial pollination. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3923–3930, San Francisco, CA, USA, 2011.

A. Beyeler, J.-C. Zufferey, and D. Floreano. Vision-based control of near-obstacle flight. *Autonomous Robots*, 27(3):201, 2009.

Bitcraze AB. Multi-ranger deck. 2019. www.bitcraze.io/multi-ranger-deck/.

J. D. Bjerknes and A. F. T. Winfield. *On Fault Tolerance and Scalability of Swarm Robotic Systems*, pages 431–444. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

E. Bonabeau and J. Dessalles. Detection and emergence. *Intellectica*, 2(25):85–94, 1997.

E. Bonabeau and G. Theraulaz. Swarm smarts. *Scientific American*, 18(1):40–47, 2008.

E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm intelligence: from natural to artificial systems*. Number 1. Oxford university press, 1999.

E. Bonabeau, S. Guérin, D. Snyers, P. Kuntz, and G. Theraulaz. Three-dimensional architectures grown by simple 'stigmergic' agents. *Biosystems*, 56(1):13 – 32, 2000.

S. Bouabdallah and R. Siegwart. Full control of a quadrotor. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 153–158, San Diego, CA, USA, 2007.

S. Bouabdallah, P. Murrieri, and R. Siegwart. Design and control of an indoor micro quadrotor. In *2004 IEEE International Conference on Robotics and Automation (ICRA)*, volume 5, pages 4393–4398, New Orleans, LA, USA, 2004.

R. Bouffanais. *Design and control of swarm dynamics*. Springer Singapore, 2016.

M. Brambilla, E. Ferrante, M. Birattari, and M. Dorigo. Swarm robotics: A review from the swarm engineering perspective. *Swarm Intelligence*, 7(1):1–41, 2013.

N. Bredeche, J. Montanier, B. Weel, and E. Haasdijk. Roborobo! a fast robot simulator for swarm and collective robotics. *arXiv preprint arXiv:1304.2888*, 2013.

N. Bredeche, E. Haasdijk, and A. Prieto. Embodied evolution in collective robotics: A review. *Frontiers in Robotics and AI*, 5:12, 2018.

S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *Seventh International World-Wide Web Conference (WWW 1998)*, 1998.

A. Briod, A. Klaptocz, J.-C. Zufferey, and D. Floreano. The AirBurr: A flying robot that can exploit collisions. In *2012 ICME International Conference on Complex Medical Engineering (CME)*, pages 569–574, Kobe, Japan, 2012.

A. Briod, J.-C. Zufferey, and D. Floreano. Optic-flow based control of a 46g quadrotor. In *Workshop on Vision-based Closed-Loop Control and Navigation of Micro Helicopters in GPS-denied Environments, IROS 2013, Tokyo, Japan, November 7, 2013*, 2013.

P. Brisset and G. Hattenberger. Multi-UAV control with the paparazzi system. In *Conference on Humans Operating Unmanned Systems (HUMOUS)*, Brest, France, 2008.

C. Brommer, D. Malyuta, D. Hentzen, and R. Brockers. Long-duration autonomy for small rotorcraft uas including recharging. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7252–7258, Madrid, Spain, 2018.

M. Bronz, J. M. Moschetta, P. Brisset, and M. Gorraz. Towards a long endurance MAV. *International Journal of Micro Air Vehicles*, 1(4):241–254, 2009.

D. S. Brown, W. Goo, P. Nagarajan, and S. Niekum. Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations. *arXiv preprint arXiv:1904.06387*, 2019.

A. Brutschy, L. Garattoni, M. Brambilla, G. Francesca, G. Pini, M. Dorigo, and M. Birattari. The TAM: Abstracting complex tasks in swarm robotics research. *Swarm Intelligence*, 9(1):1–22, 2015.

A. Bry, C. Richter, A. Bachrach, and N. Roy. Aggressive flight of fixed-wing and quadrotor aircraft in dense indoor environments. *The International Journal of Robotics Research*, 34(7):969–1002, 2015.

L. Busoniu, R. Babuška, and B. De Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(2):156–172, 2008.

L. Busoniu, R. Babuška, B. De Schutter, and D. Ernst. *Reinforcement Learning and Dynamic Programming Using Function Approximators*. CRC Press, Inc., Boca Raton, FL, USA, 2010.

R. Bähnemann, D. Schindler, M. Kamel, R. Siegwart, and J. Nieto. A decentralized multiagent unmanned aerial system to search, pick up, and relocate objects. In *2017 IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*, pages 123–128, Shanghai, China, 2017.

A. A. Cabrera-Ponce, J. Martinez-Carranza, and C. Rascon. Detection of nearby UAVs using CNN and spectrograms. In *International Micro Air Vehicle Conference and Competition (IMAV)*, Madrid, Spain, 2019.

C. Cadell. Flight of imagination: Chinese firm breaks record with 1,374 dancing drones. *Reuters*, 2018. www.reuters.com/article/us-china-drones/flight-of-imagination-chinese-firm-breaks-record-with-1374-dancing-drones-idUSKBN1I3189.

C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332, 2016.

M. Campion, P. Ranganathan, and S. Faruque. A review and future directions of UAV swarm communication architectures. In *2018 IEEE International Conference on Electro/Information Technology (EIT)*, pages 0903–0908, Rochester, MI, USA, 2018.

A. Campo and M. Dorigo. Efficient multi-foraging in swarm robotics. In F. Almeida e Costa, L. M. Rocha, E. Costa, I. Harvey, and A. Coutinho, editors, *Advances in Artificial Life*, pages 696–705, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

A. Carrio, H. Bavle, and P. Campoy. Attitude estimation using horizon detection in thermal images. *International Journal of Micro Air Vehicles*, 10(4):352–361, 2018.

A. Carrio, S. Vemprala, A. Ripoll, S. Saripalli, and P. Campoy. Drone detection using depth maps. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1034–1037, Madrid, Spain, 2018.

M. Chamanbaz, D. Mateo, B. M. Zoss, G. Tokić, E. Wilhelm, R. Bouffanais, and D. K. P. Yue. Swarm-enabling technology for multi-robot systems. *Frontiers in Robotics and AI*, 4:12, 2017.

Y. Chen, H. Zhao, J. Mao, P. Chirarattananon, E. F. Helbling, N. Patrick Hyun, D. R. Clarke, and R. J. Wood. Controlled flight of a microrobot powered by soft artificial muscles. *Nature*, 2019.

H. L. Chiang, A. Faust, M. Fiser, and A. Francis. Learning navigation behaviors end-to-end with autorl. *IEEE Robotics and Automation Letters*, 4(2):2007–2014, 2019.

C. H. Choi, H. J. Jang, S. G. Lim, H. C. Lim, S. H. Cho, and I. Gaponov. Automatic wireless drone charging station creating essential environment for continuous drone operation. In *2016 International Conference on Control, Automation and Information Sciences (ICCAIS)*, pages 132–136, Ansan, South Korea, 2016.

S. Chung, A. A. Paranjape, P. Dames, S. Shen, and V. Kumar. A survey on aerial swarm robotics. *IEEE Transactions on Robotics*, 34(4):837–855, 2018.

T. H. Chung, M. R. Clement, M. A. Day, K. D. Jones, D. Davis, and M. Jones. Live-fly, large-scale field experimentation for large numbers of fixed-wing UAVs. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1255–1262, Stockholm, Sweden, 2016.

T. Cieslewski and D. Scaramuzza. Efficient decentralized visual place recognition from full-image descriptors. In *2017 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*, pages 78–82, Los Angeles, CA, USA, 2017.

T. Cieslewski, S. Choudhary, and D. Scaramuzza. Data-efficient decentralized visual SLAM. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2466–2473, Brisbane, QLD, Australia, 2018.

E. M. Clarke, Jr., O. Grumberg, D. Kroening, D. Peled, and H. Veith. *Model Checking*. MIT Press, Cambridge, MA, USA, 1999.

M. Cognetti, P. Stegagno, A. Franchi, G. Oriolo, and H. H. Bülthoff. 3-D mutual localization with anonymous bearing measurements. In *2012 IEEE International Conference on Robotics and Automation (ICRA)*, pages 791–798, Saint Paul, MN, USA, 2012.

M. Colledanchise and P. Ögren. How behavior trees modularize hybrid control systems and generalize sequential behavior compositions, the subsumption architecture, and decision trees. *IEEE Transactions on Robotics*, 33(2):372–389, 2017.

M. Collotta, G. Pau, T. Talty, and O. K. Tonguz. Bluetooth 5: A concrete step forward toward the IoT. *IEEE Communications Magazine*, 56(7):125–131, 2018.

J. Conroy, P. Samuel, and D. Pines. Development of an MAV control and navigation system. In *Infotech@ Aerospace, AIAA 2005, Arlington, Virginia*, page 7065, 2005.

M. Coppola and G. C. H. E. de Croon. Optimization of swarm behavior assisted by an automatic local proof for a pattern formation task. In M. Dorigo, M. Birattari, C. Blum, A. L. Christensen, A. Reina, and V. Trianni, editors, *Swarm Intelligence — 11th International Conference, ANTS 2018, Lecture Notes in Computer Science, volume 11172*, pages 123–134, Cham, Switzerland, 2018. Springer International Publishing.

M. Coppola, K. N. McGuire, K. Y. W. Scheper, and G. C. H. E. de Croon. On-board communication-based relative localization for collision avoidance in micro air vehicle teams. *Autonomous Robots*, 42(8):1787–1805, 2018.

M. Coppola, J. Guo, E. Gill, and G. C. H. E. de Croon. The PageRank algorithm as a method to optimize swarm behavior through local analysis. *Swarm Intelligence*, 13 (3):277–319, 2019a.

M. Coppola, J. Guo, E. Gill, and G. C. H. E. de Croon. Provable self-organizing pattern formation by a swarm of robots with limited knowledge. *Swarm Intelligence*, 13(1): 59–94, 2019b.

M. Coppola, K. N. McGuire, C. De Wagter, and G. C. H. E. de Croon. A survey on swarming with micro air vehicles: Fundamental challenges and constraints. *Frontiers in Robotics and AI*, 7:18, 2020.

N. Correll and A. Martinoli. Collective inspection of regular structures using a swarm of miniature robots. In M. H. Ang and O. Khatib, editors, *Experimental Robotics IX: The 9th International Symposium on Experimental Robotics*, pages 375–386, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

N. Correll and A. Martinoli. Modeling and designing self-organized aggregation in a swarm of miniature robots. *The International Journal of Robotics Research*, 30(5):615–626, 2011.

N. Couture, S. Bottecchia, S. Chaumette, M. Cecconello, J. Rekalde, and M. Desainte-Catherine. Using the soundpainting language to fly a swarm of drones. In J. Chen, editor, *Advances in Human Factors in Robots and Unmanned Systems*, pages 39–51, Cham, Switzerland, 2018. Springer International Publishing.

A. Cunningham, V. Indelman, and F. Dellaert. Ddf-sam 2.0: Consistent distributed smoothing and mapping. In *2013 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5220–5227, Karlsruhe, Germany, 2013.

V. Darley. Emergent phenomena and complexity. *Artificial Life*, 4:411–416, 1994.

G. C. H. E. de Croon, M. F. van Dartel, and E. O. Postma. Evolutionary learning outperforms reinforcement learning on non-markovian tasks. In *Workshop on Memory and Learning Mechanisms in Autonomous Robots, 8th European Conference on Artificial Life, Canterbury, Kent, UK*, 2005.

G. C. H. E. de Croon, C. De Wagter, B. D. W. Remes, and R. Ruijsink. Sub-sampling: Real-time vision for micro air vehicles. *Robotics and Autonomous Systems*, 60(2):167 – 181, 2012a.

G. C. H. E. de Croon, E. de Weerdt, C. De Wagter, B. D. W. Remes, and R. Ruijsink. The appearance variation cue for obstacle avoidance. *IEEE Transactions on Robotics*, 28 (2):529–534, 2012b.

G. C. H. E. de Croon, M. Perçin, B. D. W. Remes, R. Ruijsink, and C. De Wagter. *The DelFly*. Springer Netherlands, Dordrecht, Netherlands, 2016.

H. G. de Marina. *Distributed formation control for autonomous robots*. PhD thesis, 2016.

H. G. de Marina and E. Smeur. Flexible collaborative transportation by a team of rotorcraft. In *2019 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1074–1080, Montreal, QC, Canada, 2019.

H. G. de Marina, Z. Sun, M. Bronz, and G. Hattenberger. Circular formation control of fixed-wing UAVs with constant speeds. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5298–5303, Vancouver, BC, Canada, 2017.

C. De Wagter, B. D. W. Remes, R. Ruisink, F. van Tienen, and E. van der Horst. Design and testing of a vertical take-off and landing UAV optimized for carrying a hydrogen fuel-cell with pressure tank. In *International Micro Air Vehicle Conference and Competition (IMAV)*, Madrid, Spain, 2019.

J. Deguet, Y. Demazeau, and L. Magnin. Elements about the emergence issue: A survey of emergence definitions. *ComPlexUs*, 3(1-3):24–31, 2006.

Z. Derakhshandeh, R. Gmyr, A. W. Richa, C. Scheideler, and T. Strothmann. Universal shape formation for programmable matter. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '16)*, pages 289–299, New York, NY, USA, 2016. ACM.

G. A. Di Luna, P. Flocchini, N. Santoro, G. Viglietta, and Y. Yamauchi. Shape formation by programmable particles. *arXiv preprint arXiv:1705.03538*, 2017.

E. Di Mario and A. Martinoli. Distributed particle swarm optimization for limited-time adaptation with real robots. *Robotica*, 32(2):193–208, 2014.

E. Di Mario, I. Navarro, and A. Martinoli. Distributed particle swarm optimization using optimal computing budget allocation for multi-robot learning. In *2015 IEEE Congress on Evolutionary Computation (CEC)*, pages 566–572, 2015a.

E. Di Mario, I. Navarro, and A. Martinoli. A distributed noise-resistant particle swarm optimization algorithm for high-dimensional multi-robot learning. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5970–5976, 2015b.

T. Dietrich, O. Andryeyev, A. Zimmermann, and A. Mitschele-Thiel. Towards a unified decentralized swarm management and maintenance coordination based on MAVLink. In *2016 International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 124–129, Braganca, Portugal, 2016.

C. Dixon, A. F. T. Winfield, M. Fisher, and C. Zeng. Towards temporal verification of swarm robotic systems. *Robotics and Autonomous Systems*, 60(11):1429 – 1441, 2012. Towards Autonomous Robotic Systems 2011.

C. Doer, G. Scholz, and G. F. Trommer. Indoor laser-based SLAM for micro aerial vehicles. *Gyroscopy and Navigation*, 8(3):181–189, 2017.

M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(1):29–41, 1996.

M. Dorigo, D. Floreano, L. M. Gambardella, F. Mondada, S. Nolfi, T. Baaboura, M. Birattari, M. Bonani, M. Brambilla, A. Brutschy, D. Burnier, A. Campo, A. L. Christensen, A. Decugniere, G. Di Caro, F. Ducatelle, E. Ferrante, A. Forster, J. M. Gonzales, J. Guzzi, V. Longchamp, S. Magnenat, N. Mathews, M. Montes de Oca, R. O'Grady, C. Pinciroli, G. Pini, P. Retornaz, J. Roberts, V. Sperati, T. Stirling, A. Stranieri, T. Stutzle, V. Trianni, E. Tuci, A. E. Turgut, and F. Vaussard. Swarmanoid: A novel concept for the study of heterogeneous robotic swarms. *IEEE Robotics Automation Magazine*, 20(4):60–71, 2013.

N. Dousse, G. Heitz, and D. Floreano. Extension of a ground control interface for swarms of small drones. *Artificial Life and Robotics*, 21(3):308–316, 2016.

R. D'Sa, D. Jenson, T. Henderson, J. Kilian, B. Schulz, M. Calvert, T. Heller, and N. Papanikolopoulos. SUAV:Q - an improved design for a transformable solar-powered UAV. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1609–1615, Daejeon, South Korea, 2016.

M. Duarte, V. Costa, J. Gomes, T. Rodrigues, F. Silva, S. M. Oliveira, and A. L. Christensen. Evolution of collective behaviors for a real swarm of aquatic surface robots. *PLOS ONE*, 11(3):1–25, 2016.

J. Dupeyroux, J. R. Serres, and S. Viollet. AntBot: A six-legged walking robot able to home like desert ants in outdoor environments. *Science Robotics*, 4(27), 2019.

D. Dusha, L. Mejias, and R. Walker. Fixed-wing attitude estimation using temporal tracking of the horizon and optical flow. *Journal of Field Robotics*, 28(3):355–372, 2011.

Echo Zhan. 3,051 drones create spectacular record-breaking light show in China. *Guinness World Records*, 2020. https://www.guinnessworldrecords.com/news/

commercial/2020/10/3051-drones-create-spectacular-record-breaking-light-show-in-china.

A. E. Eiben. Grand challenges for evolutionary robotics. *Frontiers in Robotics and AI*, 1: 4, 2014.

S. Engelen, E. Gill, and C. J. M. Verhoeven. Systems engineering challenges for satellite swarms. In *2011 Aerospace Conference*, AERO '11, pages 1–8, Washington, DC, USA, 2011. IEEE Computer Society.

S. Engelen, E. Gill, and C. Verhoeven. On the reliability, availability, and throughput of satellite swarms. *IEEE Transactions on Aerospace and Electronic Systems*, 50(2):1027–1037, 2014.

D. Epstein and D. Feldman. Quadcopter tracks quadcopter via real-time shape fitting. *IEEE Robotics and Automation Letters*, 3(1):544–550, 2018.

J. Ericksen, M. Moses, and S. Forrest. Automatically evolving a general controller for robot swarms. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8, 2017.

M. Faessler, E. Mueggler, K. Schwabe, and D. Scaramuzza. A monocular pose estimation system based on infrared leds. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 907–913, Hong Kong, China, 2014.

M. Faessler, F. Fontana, C. Forster, and D. Scaramuzza. Automatic re-initialization and failure recovery for aggressive flight with a monocular vision-based quadrotor. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1722–1729, Seattle, WA, USA, 2015.

J. Faigl, T. Krajník, J. Chudoba, L. Přeučil, and M. Saska. Low-cost embedded system for relative localization in robotic swarms. In *2013 IEEE International Conference on Robotics and Automation (ICRA)*, pages 993–998, Karlsruhe, Germany, 2013.

D. Falanga, S. Kim, and D. Scaramuzza. How fast is too fast? the role of perception latency in high-speed sense and avoid. *IEEE Robotics and Automation Letters*, 4(2):1884–1891, 2019a.

D. Falanga, K. Kleber, S. Mintchev, D. Floreano, and D. Scaramuzza. The foldable drone: A morphing quadrotor that can squeeze and fly. *IEEE Robotics and Automation Letters*, 4(2):209–216, 2019b.

R. Falconi, S. Gowal, and A. Martinoli. Graph based distributed control of non-holonomic vehicles endowed with local positioning information engaged in escorting missions. In *2010 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3207–3214, Anchorage, AK, USA, 2010. IEEE Press.

R. Falconi, L. Sabattini, C. Secchi, C. Fantuzzi, and C. Melchiorri. A graph-based collision-free distributed formation control strategy. *IFAC Proceedings Volumes*, 44 (1):6011 – 6016, 2011. 18th IFAC World Congress.

R. Falconi, L. Sabattini, C. Secchi, C. Fantuzzi, and C. Melchiorri. Edge-weighted consensus-based formation control strategy with collision avoidance. *Robotica*, 33 (2):332–347, 2015.

R. Faludi. *Building wireless sensor networks: with ZigBee, XBee, arduino, and processing*. O'Reilly Media, Inc., Sebastopol, CA, USA, 2010.

A. Faust, A. Francis, and D. Mehta. Evolving rewards to automate reinforcement learning. In *6th ICML Workshop on Automated Machine Learning*, 2019.

E. Ferrante, E. Duéñez Guzmán, A. E. Turgut, and T. Wenseleers. GESwarm: Grammatical evolution for the automatic synthesis of collective behaviors in swarm robotics. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, GECCO '13, pages 17–24, New York, NY, USA, 2013. ACM.

P. Fiorini and Z. Shiller. Motion planning in dynamic environments using velocity obstacles. *The International Journal of Robotics Research*, 17(7):760–772, 1998.

P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Gathering of asynchronous robots with limited visibility. *Theoretical Computer Science*, 337(1):147 – 168, 2005.

P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Arbitrary pattern formation by asynchronous, anonymous, oblivious robots. *Theoretical Computer Science*, 407(1): 412 – 447, 2008.

D. Floreano and S. Nolfi. *Evolutionary robotics: The biology, intelligence, and technology of self-organizing machines*. MIT press, 2000.

D. Floreano and R. J. Wood. Science, technology and the future of small autonomous drones. *Nature*, 521(7553):460, 2015.

D. Floreano, R. Pericet-Camara, S. Viollet, F. Ruffier, A. Brückner, R. Leitel, W. Buss, M. Menouni, F. Expert, R. Juston, M. K. Dobrzynski, G. L'Eplattenier, F. Recktenwald, H. A. Mallot, and N. Franceschini. Miniature curved artificial compound eyes. *Proceedings of the National Academy of Sciences*, 110(23):9267–9272, 2013.

D. Floreano, S. Mintchev, and J. Shintake. Foldable drones: From biology to technology. In M. Knez, A. Lakhtakia, and R. J. Martín-Palma, editors, *Bioinspiration, Biomimetics, and Bioreplication 2017*, volume 10162, pages 1 – 6, Portland, OR, USA, 2017. International Society for Optics and Photonics, SPIE.

C. Florensa, D. Held, M. Wulfmeier, M. Zhang, and P. Abbeel. Reverse curriculum generation for reinforcement learning. In S. Levine, V. Vanhoucke, and K. Goldberg, editors, *Proceedings of the 1st Annual Conference on Robot Learning*, volume 78 of *Proceedings of Machine Learning Research*, pages 482–495. PMLR, 2017.

A. Fornito, A. Zalesky, and E. T. Bullmore. *Centrality and Hubs*, pages 137 – 161. Academic Press, San Diego, 2016.

C. Forster, S. Lynen, L. Kneip, and D. Scaramuzza. Collaborative monocular SLAM with multiple micro aerial vehicles. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3962–3970, Tokyo, Japan, 2013.

F. Fortin, F. De Rainville, M. Gardner, M. Parizeau, and C. Gagné. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13:2171–2175, 2012.

M. J. Fox and J. S. Shamma. Probabilistic performance guarantees for distributed self-assembly. *IEEE Transactions on Automatic Control*, 60(12):3180–3194, 2015.

G. Francesca and M. Birattari. Automatic design of robot swarms: Achievements and challenges. *Frontiers in Robotics and AI*, 3:29, 2016.

G. Francesca, M. Brambilla, A. Brutschy, V. Trianni, and M. Birattari. Automode: A novel approach to the automatic design of control software for robot swarms. *Swarm Intelligence*, 8(2):89–112, 2014.

G. Francesca, M. Brambilla, A. Brutschy, L. Garattoni, R. Miletitch, G. Podevijn, A. Reina, T. Soleymani, M. Salvaro, C. Pinciroli, F. Mascia, V. Trianni, and M. Birattari. Automode-chocolate: Automatic design of control software for robot swarms. *Swarm Intelligence*, 9(2):125–152, 2015.

A. Franchi, G. Oriolo, and P. Stegagno. Mutual localization in multi-robot systems using anonymous relative measurements. *The International Journal of Robotics Research*, 32 (11):1302–1322, 2013.

C. Fuchs, C. Borst, G. C. H. E. de Croon, M. M. van Paassen, and M. Mulder. An ecological approach to the supervisory control of UAV swarms. *International Journal of Micro Air Vehicles*, 6(4):211–229, 2014.

F. Furrer, M. Burri, M. Achtelik, and R. Siegwart. *RotorS—A Modular Gazebo MAV Simulator Framework*, pages 595–625. Springer International Publishing, Cham, Switzerland, 2016.

B. Gabrich, D. Saldaña, V. Kumar, and M. Yim. A flying gripper based on cuboid modular robots. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7024–7030, Brisbane, QLD, Australia, 2018.

D. Gandhi, L. Pinto, and A. Gupta. Learning to fly by crashing. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3948–3955, Vancouver, BC, Canada, 2017.

S. Garnier, J. Gautrais, and G. Theraulaz. The biological principles of swarm intelligence. *Swarm Intelligence*, 1(1):3–31, 2007.

V. Gazi. Swarm aggregations using artificial potentials and sliding-mode control. *IEEE Transactions on Robotics*, 21(6):1208–1214, 2005.

V. Gazi and K. M. Passino. A class of attraction/repulsion functions for stable swarm aggregations. In *Proceedings of the 41st IEEE Conference on Decision and Control, 2002*, volume 3, pages 2842–2847, Las Vegas, NV, USA, 2002.

V. Gazi and K. M. Passino. Stability analysis of social foraging swarms. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(1):539–557, 2004.

V. Gazi and K. M. Passino. *Swarm stability and optimization.* Springer Berlin Heidelberg, Heidelberg, Germany, 2011.

B. P. Gerkey and M. J. Matarić. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research*, 23(9):939–954, 2004.

V. Ghadiok, J. Goldin, and W. Ren. On the design and development of attitude stabilization, vision-based navigation, and aerial gripping for a low-cost quadrotor. *Autonomous Robots*, 33(1):41–68, 2012.

W. Giernacki, M. Skwierczyński, W. Witwicki, P. Wroński, and P. Kozierski. Crazyflie 2.0 quadrotor as a platform for research and education in robotics and control engineering. In *22nd International Conference on Methods and Models in Automation and Robotics (MMAR)*, pages 37–42, Miedzyzdroje, Poland, 2017.

E. Gjondrekaj, M. Loreti, R. Pugliese, F. Tiezzi, C. Pinciroli, M. Brambilla, M. Birattari, and M. Dorigo. Towards a formal verification methodology for collective robotic systems. In T. Aoki and K. Taguchi, editors, *Formal Methods and Software Engineering: 14th International Conference on Formal Engineering Methods (ICFEM), Kyoto, Japan, November 12-16, 2012. Proceedings*, pages 54–70, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

C. S. Goh, J. R. Kuan, J. H. Yeo, B. S. Teo, and A. Danner. A fully solar-powered quadcopter able to achieve controlled flight out of the ground effect. *Progress in Photovoltaics: Research and Applications*, 27(10):869–878, 2019.

J. Gomes, P. Urbano, and A. L. Christensen. Introducing novelty search in evolutionary swarm robotics. In M. Dorigo, M. Birattari, C. Blum, A. L. Christensen, A. P. Engelbrecht, R. Groß, and T. Stützle, editors, *Swarm Intelligence — 8th International Conference, ANTS 2012, Lecture Notes in Computer Science, volume 7461*, pages 85–96, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

J. Gomes, P. Urbano, and A. L. Christensen. Evolution of swarm robotics systems with novelty search. *Swarm Intelligence*, 7(2):115–144, 2013.

A. Gong and D. Verstraete. Fuel cell propulsion in small fixed-wing unmanned aerial vehicles: Current status and research needs. *International Journal of Hydrogen Energy*, 42(33):21311 – 21333, 2017.

W. E. Green and P. Y. Oh. Autonomous hovering of a fixed-wing micro air vehicle. In *2006 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2164–2169, Orlando, FL, USA, 2006.

F. Grondin, D. Létourneau, F. Ferland, V. Rousseau, and F. Michaud. The ManyEars open framework. *Autonomous Robots*, 34(3):217–232, 2013.

A. Grushin and J. A. Reggia. Automated design of distributed control rules for the self-assembly of prespecified artificial structures. *Robotics and Autonomous Systems*, 56 (4):334 – 359, 2008.

A. Grushin and J. A. Reggia. Parsimonious rule generation for a nature-inspired approach to self-assembly. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 5 (3):12:1–12:24, 2010.

S. Grzonka, G. Grisetti, and W. Burgard. Towards a navigation system for autonomous indoor flying. In *2009 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2878–2883, Kobe, Japan, 2009.

Guinness World Records. Most unmanned aerial vehicles (UAVs) airborne simultaneously from a single computer (indoors). 2019. www.guinnessworldrecords.com/world-records/507534-most-uavs-controlled-from-a-single-computer.

K. Guo, Z. Qiu, C. Miao, A. H. Zaini, C. Chen, W. Meng, and L. Xie. Ultra-wideband-based localization for quadcopter navigation. *Unmanned Systems*, 04(01):23–34, 2016.

K. Guo, Z. Qiu, W. Meng, L. Xie, and R. Teo. Ultra-wideband based cooperative relative localization algorithm and experiments for multiple unmanned aerial vehicles in GPS denied environments. *International Journal of Micro Air Vehicles*, 9(3):169–186, 2017.

S. Gupte, P. I. T. Mohandas, and J. M. Conrad. A survey of quadrotor unmanned aerial vehicles. In *Proceedings of IEEE Southeastcon*, pages 1–6, Orlando, FL, USA, 2012. IEEE.

J. Guzzi, A. Giusti, L. M. Gambardella, and G. A. Di Caro. Local reactive robot navigation: A comparison between reciprocal velocity obstacle variants and human-like behavior. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2622–2629, Tokyo, Japan, 2013a.

J. Guzzi, A. Giusti, L. M. Gambardella, G. Theraulaz, and G. A. Di Caro. Human-friendly robot navigation in dynamic environments. In *2013 IEEE International Conference on Robotics and Automation (ICRA)*, pages 423–430, Karlsruhe, Germany, 2013b.

J. Guzzi, A. Giusti, L. M. Gambardella, and G. A. Di Caro. Bioinspired obstacle avoidance algorithms for robot swarms. In G. A. Di Caro and G. Theraulaz, editors, *Bio-Inspired Models of Network, Information, and Computing Systems*, pages 120–134. Springer International Publishing, Cham, Switzerland, 2014.

B. Haghighat and A. Martinoli. Automatic synthesis of rulesets for programmable stochastic self-assembly of rotationally symmetric robotic modules. *Swarm Intelligence*, 11(3):243–270, 2017.

H. Hamann. *Swarm robotics: A formal approach*. Springer International Publishing, Cham, Switzerland, 2018.

H. Hamann and H. Wörn. A framework of space–time continuous models for algorithm design in swarm robotics. *Swarm Intelligence*, 2(2):209–239, 2008.

H. Hamann, G. Valentini, Y. Khaluf, and M. Dorigo. Derivation of a micro-macro link for collective decision-making systems. In T. Bartz-Beielstein, J. Branke, B. Filipič, and J. Smith, editors, *Parallel Problem Solving from Nature – PPSN XIII*, pages 181–190, Cham, Switzerland, 2014. Springer International Publishing.

S. Hauert, J.-C. Zufferey, and D. Floreano. Evolved swarming without positioning information: An application in aerial communication relay. *Autonomous Robots*, 26(1): 21–32, 2009.

S. Hauert, S. Leven, M. Varga, F. Ruini, A. Cangelosi, J.-C. Zufferey, and D. Floreano. Reynolds flocking in reality with fixed-wing robots: Communication range vs. maximum turning rate. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5015–5020, San Francisco, CA, USA, 2011.

A. T. Hayes, A. Martinoli, and R. M. Goodman. Swarm robotic odor localization. In *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the Next Millennium (Cat. No.01CH37180)*, volume 2, pages 1073–1078, 2001.

L. Heng, L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys. Autonomous obstacle avoidance and maneuvering on a vision-guided MAV using on-board processing. In *2011 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2472–2477, Shanghai, China, 2011.

H. W. Ho, G. C. H. E. de Croon, and Q. Chu. Distance and velocity estimation using optical flow from a monocular camera. *International Journal of Micro Air Vehicles*, 9 (3):198–208, 2017.

A. Hocraffer and C. S. Nam. A meta-analysis of human-system interfaces in unmanned aerial vehicle (UAV) swarm management. *Applied Ergonomics*, 58:66 – 80, 2017.

D. Honegger, L. Meier, P. Tanskanen, and M. Pollefeys. 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 (ICRA)*, pages 1736–1741, Karlsruhe, Germany, 2013.

M. A. Hsieh, Á. Halász, S. Berman, and V. Kumar. Biologically inspired redistribution of a swarm of robots among multiple sites. *Swarm Intelligence*, 2(2):121–141, 2008.

A. S. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox, and N. Roy. *Visual Odometry and Mapping for Autonomous Flight Using an RGB-D Camera*, pages 235–252. Springer International Publishing, Cham, Switzerland, 2017.

M. Hüttenrauch, A. Šošić, and G. Neumann. Guided deep reinforcement learning for swarm systems. *arXiv preprint arXiv:1709.06011*, 2017.

S. Ishii, H. Fujita, M. Mitsutake, T. Yamazaki, J. Matsuda, and Y. Matsuno. A reinforcement learning scheme for a partially-observable multi-agent game. *Machine Learning*, 59(1):31–54, 2005.

A. S. Ismail, R. Hasni, and K. G. Subramanian. Some applications of eulerian graphs. *International Journal of Mathematical Science Education*, 2(2):1–10, 2009.

M. Itasse, J. M. Moschetta, Y. Ameho, and R. Carr. Equilibrium transition study for a hybrid MAV. *International Journal of Micro Air Vehicles*, 3(4):229–245, 2011.

B. Ivanovic, J. Harrison, A. Sharma, M. Chen, and M. Pavone. Barc: Backward reachability curriculum for robotic reinforcement learning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 15–21. IEEE, 2019.

D. Izzo and L. Pettazzi. Equilibrium shaping: Distributed motion planning for satellite swarm. In *Proceedings of the 8th International Symposium on Artificial Intelligence, Robotics and Automation in space*, Munich, Germany, 2005.

D. Izzo and L. Pettazzi. Autonomous and distributed motion planning for satellite swarm. *Journal of Guidance, Control, and Dynamics*, 30(2):449–459, 2007.

D. Izzo, L. F. Simões, and G. C. H. E. de Croon. An evolutionary robotics approach for the distributed control of satellite formations. *Evolutionary Intelligence*, 7(2):107–118, 2014.

M. Ji and M. Egerstedt. Distributed coordination control of multiagent systems while preserving connectedness. *IEEE Transactions on Robotics*, 23(4):693–703, 2007.

E. Johnson and S. Mishra. Flight simulation for the development of an experimental UAV. In *AIAA Modeling and Simulation Technologies Conference and Exhibit*, Monterey, California, 2002.

S. Jones, M. Studley, S. Hauert, and A. F. T. Winfield. *Evolving Behaviour Trees for Swarm Robotics*, pages 487–501. Springer International Publishing, Cham, Switzerland, 2018.

S. Jones, A. F. T. Winfield, S. Hauert, and M. Studley. Onboard evolution of understandable swarm behaviors. *Advanced Intelligent Systems*, 1(6):1900031, 2019.

M. A. Joordens and M. Jamshidi. Consensus control for a system of underwater swarm robots. *IEEE Systems Journal*, 4(1):65–73, 2010.

A. B. Junaid, A. Konoiko, Y. Zweiri, M. N. Sahinkaya, and L. Seneviratne. Autonomous wireless self-charging for multi-rotor unmanned aerial vehicles. *Energies*, 10(6), 2017.

S. Jung, C. Liu, and K. B. Ariyur. Absolute orientation for a UAV using celestial objects. In *AIAA Infotech@Aerospace (I@A) Conference*, Boston, MA, USA, 2013.

M. Karásek, F. T. Muijres, C. De Wagter, B. D. W. Remes, and G. C. H. E. de Croon. A tailless aerial robotic flapper reveals that flies use torque coupling in rapid banked turns. *Science*, 361(6407):1089–1094, 2018.

F. Kendoul, I. Fantoni, and K. Nonami. Optic flow-based vision system for autonomous 3D localization and control of small aerial vehicles. *Robotics and Autonomous Systems*, 57(6):591 – 602, 2009a.

F. Kendoul, K. Nonami, I. Fantoni, and R. Lozano. An adaptive vision-based autopilot for mini flying machines guidance, navigation and control. *Autonomous Robots*, 27(3): 165–188, 2009b.

E. Klavins. Automatic synthesis of controllers for distributed assembly and formation forming. In *2002 IEEE International Conference on Robotics and Automation (ICRA)*, volume 3, pages 3296–3302. IEEE Press, 2002.

E. Klavins. Programmable self-assembly. *IEEE Control Systems*, 27(4):43–56, 2007.

N. Koenig and A. Howard. Design and use paradigms for Gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 2149–2154 volume 3, Sendai, Japan, 2004.

J. Zico Kolter and A. Y. Ng. The stanford littledog: A learning and rapid replanning approach to quadruped locomotion. *The International Journal of Robotics Research*, 30 (2):150–174, 2011.

L. Kong, J. Sheng, and A. Teredesai. Basic micro-aerial vehicles (MAVs) obstacles avoidance using monocular computer vision. In *2014 13th International Conference on Control Automation Robotics Vision (ICARCV)*, pages 1051–1056, Singapore, 2014.

S. Konur, C. Dixon, and M. Fisher. Analysing robot swarm behaviour via probabilistic model checking. *Robotics and Autonomous Systems*, 60(2):199 – 213, 2012.

P. M. Kornatowski, S. Mintchev, and D. Floreano. An origami-inspired cargo drone. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6855–6862, Vancouver, BC, Canada, 2017.

T. Krajník, V. Vonásek, D. Fišer, and J. Faigl. AR-Drone as a platform for robotic research and education. In D. Obdržálek and A. Gottscheber, editors, *Research and Education in Robotics - EUROBOT 2011*, pages 172–186, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

T. Krajník, M. Nitsche, J. Faigl, P. Vaněk, M. Saska, L. Přeučil, T. Duckett, and M. Mejail. A practical multirobot localization system. *Journal of Intelligent & Robotic Systems*, 76 (3):539–562, 2014.

K. N. Krishnanand and D. Ghose. Formations of minimalist mobile robots using local-templates and spatially distributed interactions. *Robotics and Autonomous Systems*, 53(3):194 – 213, 2005.

J. Kuckling, A. Ligot, D. Bozhinoski, and M. Birattari. Behavior trees as a control architecture in the automatic modular design of robot swarms. In M. Dorigo, M. Birattari, C. Blum, A. L. Christensen, A. Reina, and V. Trianni, editors, *Swarm Intelligence — 11th International Conference, ANTS 2018, Lecture Notes in Computer Science, volume 11172*, pages 30–43, Cham, Switzerland, 2018. Springer International Publishing.

V. Kumar and N. Michael. Opportunities and challenges with autonomous micro aerial vehicles. *The International Journal of Robotics Research*, 31(11):1279–1291, 2012.

A. Kushleyev, D. Mellinger, C. Powers, and V. Kumar. Towards a swarm of agile micro quadrotors. *Autonomous Robots*, 35(4):287–300, 2013.

P. Lajoie, B. Ramtoula, Y. Chang, L. Carlone, and G. Beltrame. DOOR-SLAM: Distributed, online, and outlier resilient SLAM for robotic teams. *arXiv preprint arXiv:1909.12198v2*, 2019.

N. Laković, M. Brkić, B. Batinić, J. Bajić, V. Rajs, and N. Kulundžić. Application of low-cost VL53L0X ToF sensor for robot environment detection. In *2019 18th International Symposium INFOTEH-JAHORINA (INFOTEH)*, pages 1–4, East Sarajevo, Bosnia and Herzegovina, 2019.

K. Lamers, S. Tijmons, C. De Wagter, and G. de Croon. Self-supervised monocular distance learning on a lightweight micro air vehicle. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1779–1784, Daejeon, South Korea, 2016.

T. Lammering, E. Anton, and R. Henke. Technology assessment on aircraft-level: Modeling of innovative aircraft systems in conceptual aircraft design. In *10th AIAA Aviation Technology, Integration, and Operations Conference (ATIO)*, Fort Worth, Texas, 2012.

A. N. Langville and C. D. Meyer. *Google's PageRank and beyond: The science of search engine rankings*. Princeton University Press, 2006.

K. Leahy, D. Zhou, C. Vasile, K. Oikonomopoulos, M. Schwager, and C. Belta. Persistent surveillance for unmanned aerial vehicles subject to charging and temporal logic constraints. *Autonomous Robots*, 40(8):1363–1378, 2016.

J. Lecoeur, M. Dacke, D. Floreano, and E. Baird. The role of optic flow pooling in insect flight control in cluttered environments. *Scientific Reports*, 9(1):7707, 2019.

A. Ledergerber, M. Hamer, and R. D'Andrea. A robot self-localization system using one-way ultra-wideband communication. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3131–3137, Hamburg, Germany, 2015.

D. Lee, J. Zhou, and W. T. Lin. Autonomous battery swapping system for quadcopter. In *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*, Denver, CO, USA, 2015.

J. Lee, Y. Su, and C. Shen. A comparative study of wireless protocols: Bluetooth, uwb, zigbee, and wi-fi. In *IECON 2007 - 33rd Annual Conference of the IEEE Industrial Electronics Society*, pages 46–51, Taipei, Taiwan, 2007.

J. Lehman and K. O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2):189–223, 2011. PMID: 20868264.

C. Lehnert and P. Corke. μav - design and implementation of an open source micro quadrotor. In J. Katupitiya, J. Guivant, and R. Eaton, editors, *Australasian Conference on Robotics and Automation (ACRA2013)*, pages 1–8, University of New South Wales, Sydney, NSW, Australia, 2013. Australian Robotics & Automation Association.

R. C. Leishman, J. C. Macdonald, R. W. Beard, and T. W. McLain. Quadrotors and accelerometers: State estimation with an improved dynamic model. *IEEE Control Systems Magazine*, 34(1):28–41, 2014.

J. Leonard, A. Savvaris, and A. Tsourdos. Energy management in swarm of unmanned aerial vehicles. *Journal of Intelligent & Robotic Systems*, 74(1):233–250, 2014.

K. Lerman, A. Galstyan, A. Martinoli, and A. Ijspeert. A macroscopic analytical model of collaboration in distributed robotic systems. *Artificial Life*, 7(4):375–393, 2001.

K. Lerman, A. Martinoli, and A. Galstyan. A review of probabilistic macroscopic models for swarm robotic systems. In E. Şahin and W. M. Spears, editors, *Swarm Robotics*, pages 143–152, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

T. Lew, T. Emmei, D. D. Fan, T. Bartlett, A. Santamaria-Navarro, R. Thakker, and A. Aghamohammadi. Contact inertial odometry: Collisions are your friend. In *International Symposium on Robotics Research (ISRR)*, Hanoi, Vietnam, 2019.

S. Li, M. Coppola, C. De Wagter, and G. C. H. E. de Croon. An autonomous swarm of micro flying robots with range-based relative localization. *arXiv preprint arXiv:2003.05853*, 2020.

W. Li, M. Gauci, and R. Groß. Turing learning: A metric-free approach to inferring behavior and its application to swarms. *Swarm Intelligence*, 10(3):211–243, 2016.

X. Li and L. E. Parker. Sensor analysis for fault detection in tightly-coupled multi-robot team tasks. In *2007 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3269–3276, Rome, Italy, 2007.

X. Li and L. E. Parker. Distributed sensor analysis for fault detection in tightly-coupled multi-robot team tasks. In *2009 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3103–3110, Kobe, Japan, 2009.

Q. Lindsey, D. Mellinger, and V. Kumar. Construction with quadrotor teams. *Autonomous Robots*, 33(3):323–336, 2012.

W. Liu and A. F. T. Winfield. Modeling and optimization of adaptive foraging in swarm robotic systems. *The International Journal of Robotics Research*, 29(14):1743–1760, 2010.

W. Liu, G. Loianno, K. Mohta, K. Daniilidis, and V. Kumar. Semi-dense visual-inertial odometry and mapping for quadrotors with swap constraints. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3904–3909, Brisbane, QLD, Australia, 2018.

S. Loncaric. A survey of shape analysis techniques. *Pattern Recognition*, 31(8):983 – 1001, 1998.

E. López, R. Barea, A. Gómez, Á. Saltos, L. M. Bergasa, E. J. Molinos, and A. Nemra. Indoor SLAM for micro aerial vehicles using visual and laser sensor fusion. In L. P. Reis, A. P. Moreira, P. U. Lima, L. Montano, and V. Muñoz-Martinez, editors, *Robot 2015: Second Iberian Robotics Conference*, pages 531–542, Cham, Switzerland, 2016. Springer International Publishing.

J. Macdonald, R. Leishman, R. W. Beard, and T. McLain. Analysis of an improved IMU-based observer for multirotor helicopters. *Journal of Intelligent & Robotic Systems*, 74 (3):1049–1061, 2014.

I. Mademlis, N. Nikolaidis, A. Tefas, I. Pitas, T. Wagner, and A. Messina. Autonomous UAV cinematography: A tutorial and a formalized shot-type taxonomy. *ACM Comput. Surv.*, 52(5):105:1–105:33, 2019.

R. Mahony, V. Kumar, and P. Corke. Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor. *IEEE Robotics Automation Magazine*, 19(3):20–32, 2012.

A. Mairaj, A. I. Baba, and A. Y. Javaid. Application specific drone simulators: Recent advances and challenges. *Simulation Modelling Practice and Theory*, 94:100 – 117, 2019.

A. Martinelli. Closed-form solution for attitude and speed determination by fusing monocular vision and inertial sensor measurements. In *2011 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4538–4545, Shanghai, China, 2011.

A. Martinoli and K. Easton. Modeling swarm robotic systems. In B. Siciliano and P. Dario, editors, *Experimental Robotics VIII*, pages 297–306, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

A. Martinoli, K. Easton, and W. Agassounon. Modeling swarm robotic systems: A case study in collaborative distributed manipulation. *The International Journal of Robotics Research*, 23(4-5):415–436, 2004.

M. J. Matarić. *Reinforcement Learning in the Multi-Robot Domain*, pages 73–83. Springer US, Boston, MA, 1997.

M. J. Matarić, M. Nilsson, and K. T. Simsarin. Cooperative multi-robot box-pushing. In *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots*, volume 3, pages 556–561 vol 3, 1995.

L. Matthies, R. Brockers, Y. Kuwata, and S. Weiss. Stereo vision-based obstacle avoidance for micro air vehicles using disparity space. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3242–3249, Hong Kong, China, 2014.

R. McCallum. *Reinforcement learning with selective perception and hidden state*. PhD thesis, University of Rochester, 1997.

K. N. McGuire, G. C. H. E. de Croon, C. De Wagter, B. D. W. Remes, K. Tuyls, and H. Kappen. Local histogram matching for efficient optical flow computation applied to velocity estimation on pocket drones. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3255–3260. IEEE Press, 2016.

K. N. McGuire, M. Coppola, C. De Wagter, and G. C. H. E. de Croon. Towards autonomous navigation of multiple pocket-drones in real-world environments. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 244–249, Vancouver, BC, Canada, 2017a.

K. N. McGuire, G. C. H. E. de Croon, C. De Wagter, K. Tuyls, and H. Kappen. Efficient optical flow and stereo vision for velocity estimation and obstacle avoidance on an autonomous pocket drone. *IEEE Robotics and Automation Letters*, 2(2):1070–1076, 2017b.

K. N. McGuire, C. De Wagter, K. Tuyls, H. J. Kappen, and G. C. H. E. de Croon. Minimal navigation solution for a swarm of tiny flying robots to explore an unknown environment. *Science Robotics*, 4(35), 2019.

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

C. Melhuish and J. Welsby. Gradient ascent with a group of minimalist real robots: Implementing secondary swarming. In *IEEE International Conference on Systems, Man and Cybernetics*, volume 2, pages 509–514, Yasmine Hammamet, Tunisia, 2002.

M. Mesbahi and M. Egerstedt. *Graph theoretic methods in multiagent networks*, volume 33. Princeton University Press, 2010.

J. Meyer, A. Sendobry, S. Kohlbrecher, U. Klingauf, and O. von Stryk. Comprehensive simulation of quadrotor UAVs using ROS and Gazebo. In I. Noda, N. Ando, D. Brugali, and J. J. Kuffner, editors, *Simulation, Modeling, and Programming for Autonomous Robots*, pages 400–411. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

N. Michael, D. Mellinger, Q. Lindsey, and V. Kumar. The GRASP multiple micro-UAV testbed. *IEEE Robotics Automation Magazine*, 17(3):56–65, 2010.

R. C. Michelson and S. Reece. Update on flapping wing micro air vehicle research-ongoing work to develop a flapping wing, crawling entomopter. In *13th Bristol International RPV/UAV Systems Conference Proceedings, Bristol England*, volume 30, pages 30–1, 1998.

J. A. Millan-Romera, H. Perez-Leon, A. Castillejo-Calle, I. Maza, and A. Ollero. ROS-MAGNA, a ROS-based framework for the definition and management of multi-UAS cooperative missions. In *International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 1477–1486, Atlanta, GA, USA, 2019.

S. Mintchev, S. de Rivaz, and D. Floreano. Insect-inspired mechanical resilience for multicopters. *IEEE Robotics and Automation Letters*, 2(3):1248–1255, 2017.

A. Mirjan, F. Augugliaro, R. D'Andrea, F. Gramazio, and M. Kohler. *Building a Bridge with Flying Robots*, pages 34–47. Springer International Publishing, Cham, Switzerland, 2016.

B. B. Mohr and D. L. Fitzpatrick. Micro air vehicle navigation system. *IEEE Aerospace and Electronic Systems Magazine*, 23(4):19–24, 2008.

F. Mondada, G. C. Pettinaro, A. Guignard, I. W. Kwee, D. Floreano, J. Deneubourg, S. Nolfi, L. Maria Gambardella, and M. Dorigo. Swarm-bot: A new distributed robotic concept. *Autonomous robots*, 17(2-3):193–221, 2004.

E. Montijano, E. Cristofalo, D. Zhou, M. Schwager, and C. Sagüés. Vision-based distributed formation control without an external positioning system. *IEEE Transactions on Robotics*, 32(2):339–351, 2016.

R. J. D. Moore, K. Dantu, G. L. Barrows, and R. Nagpal. Autonomous MAV guidance with a lightweight omnidirectional vision sensor. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3856–3861, Hong Kong, China, 2014.

E. Mueggler, N. Baumli, F. Fontana, and D. Scaramuzza. Towards evasive maneuvers with quadrotors using dynamic vision sensors. In *2015 European Conference on Mobile Robots (ECMR)*, pages 1–8, Lincoln, UK, 2015.

M. Mueller and A. Drouin. Paparazzi - The Free Autopilot. Build Your Own UAV. In *24th Chaos Communication Congress*, pages 27–30, Berlin, Germany, 2007.

Y. Mulgaonkar and V. Kumar. Autonomous charging to enable long-endurance missions for small aerial robots. In T. George, M. Saif Islam, and A. K. Dutta, editors, *Micro- and Nanotechnology Sensors, Systems, and Applications VI*, volume 9083, pages 404 – 418, Baltimore, MD, USA, 2014. International Society for Optics and Photonics, SPIE.

Y. Mulgaonkar, M. Whitzer, B. Morgan, C. M. Kroninger, A. M. Harrington, and V. Kumar. Power and weight considerations in small, agile quadrotors. In T. George, M. Saif Islam, and A. K. Dutta, editors, *Micro- and Nanotechnology Sensors, Systems, and Applications VI*, volume 9083, pages 376 – 391, Baltimore, MD, USA, 2014. International Society for Optics and Photonics, SPIE.

Y. Mulgaonkar, G. Cross, and V. Kumar. Design of small, safe and robust quadrotor swarms. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2208–2215, Seattle, WA, USA, 2015.

Y. Mulgaonkar, A. Makineni, L. Guerrero-Bonilla, and V. Kumar. Robust aerial robot swarms without collision avoidance. *IEEE Robotics and Automation Letters*, 3(1):596–603, 2018.

J. Nagi, A. Giusti, L. M. Gambardella, and G. A. Di Caro. Human-swarm interaction using spatial gestures. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3834–3841, Chicago, IL, USA, 2014.

T. P. Nascimento and M. Saska. Position and attitude control of multi-rotor aerial vehicles: A survey. *Annual Reviews in Control*, 2019.

N. Nedjah and L. Silva Junior. Review of methodologies and tasks in swarm robotics towards standardization. *Swarm and Evolutionary Computation*, 50:100565, 2019.

A. L. Nelson, G. J. Barlow, and L. Doitsidis. Fitness functions in evolutionary robotics: A survey and analysis. *Robotics and Autonomous Systems*, 57(4):345 – 370, 2009.

J. Nembrini, A. F. T. Winfield, and C. Melhuish. Minimalist coherent swarming of wireless networked autonomous mobile robots. In B. Hallam, D. Floreano, J. Hallam, G. Hayes, and J. Meyer, editors, *From Animals to Animats 7: Proceedings of the Seventh International Conference on Simulation of Adaptive Behavior*, ICSAB, pages 373–382, Cambridge, MA, USA, 2002. MIT Press.

R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, pages 127–136, Basel, Switzerland, 2011.

F. Nex and F. Remondino. UAV for 3D mapping applications: A review. *Applied Geomatics*, 6(1):1–15, 2014.

A. Y. Ng and S. J. Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICLM)*, volume 1, page 2, 2000.

T. Nguyen, Z. Qiu, T. H. Nguyen, M. Cao, and L. Xie. Distance-based cooperative relative localization for leader-following control of MAVs. *IEEE Robotics and Automation Letters*, 4(4):3641–3648, 2019.

M. Nieuwenhuisen, M. Beul, R. A. Rosu, J. Quenzel, D. Pavlichenko, S. Houben, and S. Behnke. Collaborative object picking and delivery with a team of micro aerial vehicles at MBZIRC. In *2017 European Conference on Mobile Robots (ECMR)*, pages 1–6, Paris, France, 2017.

S. Nolfi. Power and the limits of reactive agents. *Neurocomputing*, 42(1):119 – 145, 2002.

A. Noth and R. Siegwart. *Solar-Powered Micro-air Vehicles and Challenges in Downscaling*, pages 285–297. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

T. Nägeli, C. Conte, A. Domahidi, M. Morari, and O. Hilliges. Environment-independent formation flight for micro aerial vehicles. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1141–1146, Chicago, IL, USA, 2014.

E. Obert. *Aerodynamic design of transport aircraft*. IOS press, Amsterdam, Netherlands, 2009.

M. Odelga, P. Stegagno, and H. H. Bülthoff. Obstacle detection, tracking and avoidance for a teleoperated UAV. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2984–2990, Stockholm, Sweden, 2016.

K.-K. Oh, M.-C. Park, and H.-S. Ahn. A survey of multi-agent formation control. *Automatica*, 53:424 – 440, 2015.

D. A. Olejnik, B. P. Duisterhof, M. Karásek, K. Y. W. Scheper, T. van Dijk, and G. C. H. E. de Croon. A tailless flapping wing MAV performing monocular visual servoing tasks. In *International Micro Air Vehicle Conference and Competition (IMAV)*, Madrid, Spain, 2019.

H. Oleynikova, D. Honegger, and M. Pollefeys. Reactive avoidance using embedded stereo vision for MAV flight. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 50–56, Seattle, WA, USA, 2015.

F. A. Oliehoek and C. Amato. *A concise introduction to decentralized POMDPs*, volume 1. Springer, 2016.

R. Opromolla, G. Fasano, G. Rufino, M. Grassi, and A. Savvaris. LIDAR-inertial integration for UAV localization and mapping in complex environments. In *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 649–656, Arlington, VA, USA, 2016.

R. Opromolla, G. Inchingolo, and G. Fasano. Airborne visual detection and tracking of cooperative UAVs exploiting deep learning. *Sensors*, 19(19):4332, 2019.

R. Oung and R. D'Andrea. The distributed flight array. *Mechatronics*, 21(6):908 – 917, 2011.

A. Pacheco, V. Strobel, and M. Dorigo. A blockchain-controlled physical robot swarm communicating via an ad-hoc network. In M. Dorigo, T. Stützle, M. J. Blesa, C. Blum, H. Hamann, M. K. Heinrich, and V. Strobel, editors, *Swarm Intelligence*, pages 3–15, Cham, 2020. Springer International Publishing. ISBN 978-3-030-60376-2.

L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, 1999.

I. Palunko, R. Fierro, and P. Cruz. Trajectory generation for swing-free maneuvers of a quadrotor with suspended payload: A dynamic programming approach. In *2012 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2691–2697, Saint Paul, MN, USA, 2012.

Z. F. Pan, L. An, and C. Y. Wen. Recent advances in fuel cells based propulsion systems for unmanned aerial vehicles. *Applied Energy*, 240:473 – 485, 2019.

S. Parsons and M. Wooldridge. Game theory and decision theory in multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 5(3):243–254, 2002.

A. Pascoal, I. Kaminer, and P. Oliveira. Navigation system design using time-varying complementary filters. *IEEE Transactions on Aerospace and Electronic Systems*, 36(4):1099–1114, 2000.

A. R. Pereira and L. Hsu. Adaptive formation control using artificial potentials for euler-lagrange agents. *IFAC Proceedings Volumes*, 41(2):10788 – 10793, 2008.

J. Pestana, J. L. Sanchez-Lopez, P. de la Puente, A. Carrio, and P. Campoy. A vision-based quadrotor swarm for the participation in the 2013 international micro air vehicle competition. In *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 617–622, Orlando, FL, USA, 2014.

L. Petricca, P. Ohlckers, and C. Grinde. Micro-and nano-air vehicles: State of the art. *International journal of aerospace engineering*, 2011(214549), 2011.

C. Pinciroli and G. Beltrame. Buzz: An extensible programming language for heterogeneous swarm robotics. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3794–3800, Daejeon, South Korea, 2016.

C. Pinciroli, V. Trianni, R. O'Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. Di Caro, F. Ducatelle, M. Birattari, L. M. Gambardella, and M. Dorigo. ARGoS: A modular, parallel, multi-engine simulator for multi-robot systems. *Swarm Intelligence*, 6(4):271–295, 2012.

L. Pitonakova, R. Crowder, and S. Bullock. Information exchange design patterns for robot swarm foraging and their application in robot control algorithms. *Frontiers in Robotics and AI*, 5:47, 2018.

G. Portelli, F. Ruffier, F. L. Roubieu, and N. Franceschini. Honeybees' speed depends on dorsal as well as lateral, ventral and frontal optic flows. *PLOS ONE*, 6(5):1–10, 2011.

C. Powers, D. Mellinger, A. Kushleyev, B. Kothmann, and V. Kumar. Influence of aerodynamics and proximity effects in quadrotor flight. In J. P. Desai, G. Dudek, O. Khatib, and V. Kumar, editors, *Experimental Robotics: The 13th International Symposium on Experimental Robotics*, pages 289–302, Heidelberg, Germany, 2013. Springer International Publishing.

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

A. Prorok, N. Correll, and A. Martinoli. Multi-level spatial modeling for stochastic distributed robotic systems. *The International Journal of Robotics Research*, 30(5):574–589, 2011.

J. Pugh, X. Raemy, C. Favre, R. Falconi, and A. Martinoli. A fast onboard relative positioning module for multirobot systems. *IEEE/ASME Transactions on Mechatronics*, 14(2): 151–162, 2009.

L. Qin, X. He, and D. H. Zhou. A survey of fault diagnosis for swarm systems. *Systems Science & Control Engineering*, 2(1):13–23, 2014.

M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. ROS: An open-source Robot Operating System. In *ICRA workshop on open source software*, volume 3, page 5, Kobe, Japan, 2009.

S. A. P. Quintero, G. E. Collins, and J. P. Hespanha. Flocking with fixed-wing UAVs for distributed sensing: A stochastic optimal control approach. In *2013 American Control Conference*, pages 2025–2031, Washington, DC, USA, 2013.

A. Rahmani, M. Ji, M. Mesbahi, and M. Egerstedt. Controllability of multi-agent systems from a graph-theoretic perspective. *SIAM Journal on Control and Optimization*, 48(1): 162–186, 2009.

J. Rasmussen. Skills, rules, and knowledge; signals, signs, and symbols, and other distinctions in human performance models. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(3):257–266, 1983.

J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

A. Reina, R. Miletitch, M. Dorigo, and V. Trianni. A quantitative micro–macro link for collective decisions: The shortest path discovery/selection example. *Swarm Intelligence*, 9(2):75–102, 2015.

B. D. W. Remes, P. Esden-Tempski, F. van Tienen, E. J. J. Smeur, C. De Wagter, and G. C. H. E. de Croon. Lisa-S 2.8 g autopilot for GPS-based flight of MAVs. In *International Micro Air Vehicle Conference and Competition (IMAV)*, Delft, Netherlands, 2014.

C. W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '87, pages 25–34, New York, NY, USA, 1987. ACM.

A. Ripoll Sanchez. Guidance and control of a spacecraft swarm with limited knowledge. Master's thesis, Delft University of Technology, 2019.

J. F. Roberts, T. Stirling, J.-C. Zufferey, and Dario Floreano. 3-D relative positioning sensor for indoor flying robots. *Autonomous Robots*, 33(1):5–20, 2012.

S. Roelofsen, D. Gillet, and A. Martinoli. Reciprocal collision avoidance for quadrotors using on-board visual detection. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4810–4817, Hamburg, Germany, 2015.

P. E. Ross. Open-source drones for fun and profit. *IEEE Spectrum*, 51(3):54–59, 2014.

S. Ross, N. Melik-Barkhudarov, K. S. Shankar, A. Wendel, D. Dey, J. A. Bagnell, and M. Hebert. Learning monocular reactive UAV control in cluttered natural environments. In *2013 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1765–1772, Karlsruhe, Germany, 2013.

M. Rubenstein, A. Cornejo, and R. Nagpal. Programmable self-assembly in a thousand-robot swarm. *Science*, 345(6198):795–799, 2014.

O. Ruiz-Espitia, J. Martinez-Carranza, and C. Rascon. AIRA-UAS: an evaluation corpus for audio processing in unmanned aerial system. In *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 836–845, Dallas, TX, USA, 2018.

I. Sa and P. Corke. Vertical infrastructure inspection using a quadcopter and shared autonomy control. In K. Yoshida and S. Tadokoro, editors, *Field and Service Robotics: Results of the 8th International Conference*, pages 219–232, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

A. M. Sabatini and V. Genovese. A stochastic approach to noise modeling for barometric altimeters. *Sensors*, 13(11):15692–15707, 2013.

S. Saha, A. Natraj, and S. Waharte. A real-time monocular vision-based frontal obstacle detection and avoidance for low cost UAVs in GPS denied environment. In *2014 IEEE International Conference on Aerospace Electronics and Remote Sensing Technology*, pages 189–195, Yogyakarta, Indonesia, 2014.

E. Şahin. Swarm robotics: From sources of inspiration to domains of application. In E. Şahin and W. M. Spears, editors, *Swarm Robotics*, pages 10–20, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

E. Şahin, S. Girgin, L. Bayindir, and A. E. Turgut. *Swarm Robotics*, pages 87–100. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

D. Saldaña, A. Prorok, S. Sundaram, M. F. M. Campos, and V. Kumar. Resilient consensus for time-varying networks of dynamic agents. In *2017 American Control Conference (ACC)*, pages 252–258, Seattle, WA, USA, 2017.

D. Saldaña, B. Gabrich, G. Li, M. Yim, and V. Kumar. ModQuad: The flying modular structure that self-assembles in midair. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 691–698, Brisbane, QLD, Australia, 2018.

D. Saldaña, P. M. Gupta, and V. Kumar. Design and control of aerial modules for inflight self-disassembly. *IEEE Robotics and Automation Letters*, 4(4):3410–3417, 2019.

J. L. Sanchez-Lopez, R. A. S. Fernández, H. Bavle, C. Sampedro, M. Molina, J. Pestana, and P. Campoy. AEROSTACK: An architecture and open-source software framework for aerial robotics. In *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 332–341, Arlington, VA, USA, 2016.

A. Santamaria-Navarro, J. Solà, and J. Andrade-Cetto. High-frequency MAV state estimation using low-cost inertial and optical flow measurement units. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1864–1871, Hamburg, Germany, 2015.

E. Sapin. Gliders and glider guns discovery in cellular automata. In A. Adamatzky, editor, *Game of Life Cellular Automata*, pages 135–165. Springer London, London, 2010.

M. Saska. MAV-swarms: Unmanned aerial vehicles stabilized along a given path using onboard relative localization. In *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 894–903, Denver, CO, USA, 2015.

M. Saska, J. Vakula, and L. Přeućil. Swarms of micro aerial vehicles stabilized under a visual relative localization. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3570–3575, Hong Kong, China, 2014.

M. Saska, T. Baca, V. Spurnỳ, G. Loianno, J. Thomas, T. Krajník, P. Stepan, and V. Kumar. Vision-based high-speed autonomous landing and cooperative objects grasping—towards the MBZIRC competition. In *Proceedings of the 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems—Vision-based High Speed Autonomous Navigation of UAVs (Workshop)*, pages 9–14, Daejeon, South Korea, 2016a.

M. Saska, V. Vonásek, J. Chudoba, J. Thomas, G. Loianno, and V. Kumar. Swarm distribution and deployment for cooperative surveillance by micro-aerial vehicles. *Journal of Intelligent & Robotic Systems*, pages 1–24, 2016b.

M. Saska, T. Baca, J. Thomas, J. Chudoba, L. Preucil, T. Krajnik, J. Faigl, G. Loianno, and V. Kumar. System for deployment of groups of unmanned micro aerial vehicles in GPS-denied environments using onboard visual relative localization. *Autonomous Robots*, 41(4):919–944, 2017.

K. Saulnier, D. Saldaña, A. Prorok, G. J. Pappas, and V. Kumar. Resilient flocking for mobile robot teams. *IEEE Robotics and Automation Letters*, 2(2):1039–1046, 2017.

D. Scaramuzza and Z. Zhang. Visual-inertial odometry of aerial robots. *arXiv preprint arXiv:1906.03289*, 2019.

D. Scaramuzza, M. C. Achtelik, L. Doitsidis, F. Friedrich, E. Kosmatopoulos, A. Martinelli, M. W. Achtelik, M. Chli, S. Chatzichristofis, L. Kneip, D. Gurdan, L. Heng, G. H. Lee, S. Lynen, M. Pollefeys, A. Renzaglia, R. Siegwart, J. C. Stumpf, P. Tanskanen, C. Troiani, S. Weiss, and L. Meier. Vision-controlled micro flying robots: From system design to autonomous navigation and mapping in GPS-denied environments. *IEEE Robotics Automation Magazine*, 21(3):26–40, 2014.

K. Schauwecker and A. Zell. On-board dual-stereo-vision for the navigation of an autonomous MAV. *Journal of Intelligent & Robotic Systems*, 74(1):1–16, 2014.

K. Schauwecker, N. R. Ke, S. A. Scherer, and A. Zell. Markerless visual control of a quadrotor micro aerial vehicle by means of on-board stereo processing. In P. Levi, O. Zweigle, K. Häußermann, and B. Eckstein, editors, *Autonomous Mobile Systems 2012*, pages 11–20, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

K. Y. W. Scheper. *Abstraction as a Tool to Bridge the Reality Gap in Evolutionary Robotics*. PhD thesis, Delft University of Technology, 2019.

K. Y. W. Scheper and G. C. H. E. de Croon. Abstraction as a mechanism to cross the reality gap in evolutionary robotics. In E. Tuci, A. Giagkos, M. Wilson, and J. Hallam, editors,

From Animals to Animats 14: 14th International Conference on Simulation of Adaptive Behavior, SAB 2016, Aberystwyth, UK, August 23-26, 2016, Proceedings, pages 280–292, Cham, Switzerland, 2016. Springer International Publishing.

K. Y. W. Scheper and G. C. H. E. de Croon. Abstraction, sensory-motor coordination, and the reality gap in evolutionary robotics. *Artificial Life*, 23(2):124–141, 2017. PMID: 28513202.

K. Y. W. Scheper, S. Tijmons, C. C. de Visser, and G. C. H. E. de Croon. Behavior trees for evolutionary robotics. *Artificial Life*, 22(1):23–48, 2016. PMID: 26606468.

F. Schiano, A. Franchi, D. Zelazo, and P. R. Giordano. A rigidity-based decentralized bearing formation controller for groups of quadrotor UAVs. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5099–5106, Daejeon, South Korea, 2016.

F. Schilling, J. Lecoeur, F. Schiano, and D. Floreano. Learning vision-based flight in drone swarms by imitation. *IEEE Robotics and Automation Letters*, 4(4):4523–4530, 2019.

R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. Grad-CAM: Visual explanations from deep networks via gradient-based localization. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 618–626, Venice, Italy, 2017.

S. Shah, D. Dey, C. Lovett, and A. Kapoor. AirSim: High-fidelity visual and physical simulation for autonomous vehicles. In M. Hutter and R. Siegwart, editors, *Field and Service Robotics*, pages 621–635, Cham, Switzerland, 2018. Springer International Publishing.

S. Shen, N. Michael, and V. Kumar. Autonomous multi-floor indoor navigation with a computationally constrained MAV. In *2011 IEEE International Conference on Robotics and Automation (ICRA)*, pages 20–25, Shanghai, China, 2011.

S. Shen, Y. Mulgaonkar, N. Michael, and V. Kumar. Multi-sensor fusion for robust autonomous flight in indoor and outdoor environments with a rotorcraft MAV. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4974–4981, Hong Kong, China, 2014.

J. Shi, B. He, L. Zhang, and J. Zhang. Vision-based real-time 3D mapping for UAV with laser sensor. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4524–4529, Daejeon, South Korea, 2016.

N. Shiell and A. Vardy. A bearing-only pattern formation algorithm for swarm robotics. In M. Dorigo, M. Birattari, X. Li, M. López-Ibáñez, K. Ohkura, C. Pinciroli, and T. Stützle, editors, *Swarm Intelligence — 10th International Conference, ANTS 2016, Lecture Notes in Computer Science, volume 9882*, pages 3–14, Cham, Switzerland, 2016. Springer International Publishing.

K. Shilov. The next generation design of autonomous mav flight control system SMARTAP. In *International Micro Air Vehicle Conference and Competition (IMAV)*, Delft, Netherlands, 2014.

F. Silva, P. Urbano, L. Correia, and A. L. Christensen. odNEAT: An algorithm for decentralised online evolution of robotic controllers. *Evolutionary Computation*, 23(3):421–449, 2015.

F. Silva, M. Duarte, L. Correia, S. M. Oliveira, and A. L. Christensen. Open issues in evolutionary robotics. *Evolutionary Computation*, 24(2):205–236, 2016. PMID: 26581015.

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359, 2017.

I. Slavkov, D. Carrillo-Zapata, N. Carranza, X. Diego, F. Jansson, J. Kaandorp, S. Hauert, and J. Sharpe. Morphogenesis in robot swarms. *Science Robotics*, 3(25), 2018.

J. Smisek, M. Jancosek, and T. Pajdla. *3D with Kinect*, pages 3–25. Springer London, London, 2013.

B. Smith, A. Howard, J. M. McNew, J. Wang, and Magnus Egerstedt. Multi-robot deployment and coordination with embedded graph grammars. *Autonomous Robots*, 26(1): 79–98, 2009.

J. Snape, J. van den Berg, S. J. Guy, and D. Manocha. Independent navigation of multiple mobile robots with hybrid reciprocal velocity obstacles. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5917–5922, St. Louis, MO, USA, 2009.

J. Snape, J. van den Berg, S. J. Guy, and D. Manocha. The hybrid reciprocal velocity obstacle. *IEEE Transactions on Robotics*, 27(4):696–706, 2011.

E. Soria, F. Schiano, and D. Floreano. SwarmLab: A Matlab drone swarm simulator. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8005–8011, 2020.

D. Soto-Gerrero and J.-G. Ramrez-Torres. A human-machine interface with unmanned aerial vehicles. In S. Zeghloul, M. A. Laribi, and J.-P. Gazeau, editors, *Robotics and Mechatronics*, pages 233–242, Cham, Switzerland, 2016. Springer International Publishing.

V. Spurný, T. Báča, M. Saska, R. Pěnička, T. Krajník, J. Thomas, D. Thakur, G. Loianno, and V. Kumar. Cooperative autonomous search, grasping, and delivering in a treasure hunt scenario by a team of unmanned aerial vehicles. *Journal of Field Robotics*, 36(1): 125–148, 2019.

S. B. Stancliff, J. M. Dolan, and A. Trebi-Ollennu. Mission reliability estimation for multirobot team design. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2206–2211, Beijing, China, 2006.

P. Stegagno, M. Cognetti, A. Franchi, and G. Oriolo. Mutual localization using anonymous bearing measurements. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 469–474, San Francisco, CA, USA, 2011.

P. Stegagno, M. Basile, H. H. Bülthoff, and A. Franchi. A semi-autonomous UAV platform for indoor remote operation with visual and haptic feedback. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3862–3869, Hong Kong, China, 2014.

P. Stegagno, M. Cognetti, G. Oriolo, H. H. Bülthoff, and A. Franchi. Ground and aerial mutual localization using anonymous relative-bearing measurements. *IEEE Transactions on Robotics*, 32(5):1133–1151, 2016.

E. Stevens, L. Antiga, and T. Viehmann. *Deep Learning with PyTorch*. Manning Publications, 2020.

V. Strobel, E. Castelló Ferrer, and M. Dorigo. Managing byzantine robots via blockchain technology in a swarm robotics collective decision making scenario. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '18, page 541–549, Richland, SC, 2018. International Foundation for Autonomous Agents and Multiagent Systems.

A. Suleiman, Z. Zhang, L. Carlone, S. Karaman, and V. Sze. Navion: A fully integrated energy-efficient visual-inertial odometry accelerator for autonomous navigation of nano drones. In *2018 IEEE Symposium on VLSI Circuits*, pages 133–134, Honolulu, HI, USA, 2018.

R. S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine learning proceedings 1990*, pages 216–224. Elsevier, 1990.

R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction (Second Edition)*. MIT press, Cambridge, MA, 2018.

R. Swatman. Intel stuns during CES keynote with record for most drones airborne simultaneously - watch incredible footage. *Guinness World Records*, 2016a. https://www.guinnessworldrecords.com/news/brand-or-agency/2016/1/intel-stuns-during-ces-keynote-with-record-for-most-drones-airborne-simultaneousl-411677.

R. Swatman. Intel launches 500 drones into sky and breaks world record in spectacular style. *Guinness World Records*, 2016b. https://www.guinnessworldrecords.com/news/2016/11/intel-launches-500-drones-into-sky-and-breaks-world-record-in-spectacular-style-449886.

T. Szabo. Autonomous collision avoidance for swarms of MAVs: Based solely on RSSI measurements. Master's thesis, Delft University of Technology, 2015.

A. Tagliabue, M. Kamel, R. Siegwart, and J. Nieto. Robust collaborative object transportation using multiple MAVs. *The International Journal of Robotics Research*, 38(9):1020–1044, 2019.

H. G. Tanner. On the controllability of nearest neighbor interconnections. In *2004 43rd IEEE Conference on Decision and Control (CDC)*, volume 3, pages 2467–2472, Nassau, Bahamas, 2004.

D. Tarapore, A. L. Christensen, P. U. Lima, and J. Carneiro. Abnormality detection in multiagent systems inspired by the adaptive immune system. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems*, AAMAS '13, pages 23–30, Richland, SC, 2013. International Foundation for Autonomous Agents and Multiagent Systems.

D. Tarapore, A. L. Christensen, and J. Timmis. Abnormality detection in robots exhibiting composite swarm behaviours. In *Artificial Life Conference Proceedings*, number 27, pages 406–413, York, UK, 2015a.

D. Tarapore, P. U. Lima, J. Carneiro, and A. L. Christensen. To err is robotic, to tolerate immunological: Fault detection in multirobot systems. *Bioinspiration & Biomimetics*, 10(1):016014, 2015b.

D. Tarapore, A. L. Christensen, and J. Timmis. Generic, scalable and decentralized fault detection for robot swarms. *PLOS ONE*, 12(8):1–29, 2017.

D. Tarapore, J. Timmis, and A. L. Christensen. Fault detection in a swarm of physical robots based on behavioral outlier detection. *IEEE Transactions on Robotics*, pages 1–7, 2019.

L. Teixeira, F. Maffra, M. Moos, and M. Chli. Vi-rpe: Visual-inertial relative pose estimation for aerial vehicles. *IEEE Robotics and Automation Letters*, 3(4):2770–2777, 2018.

S. Thurrowgood, D. Soccol, R. J. D. Moore, D. Bland, and M. V. Srinivasan. A vision based system for attitude estimation of UAVS. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5725–5730, St. Louis, MO, USA, 2009.

E. Tijs, G. C. H. E. de Croon, J. Wind, B. D. W. Remes, C. De Wagter, H. E. de Bree, and R. Ruijsink. Hear-and-avoid for micro air vehicles. In *International Micro Air Vehicle Conference and Competition (IMAV)*, volume 69, Braunschweig, Germany, 2010.

T. Toksoz, J. Redding, M. Michini, B. Michini, J. P. How, M. Vavrina, and J. Vian. Automated battery swap and recharge to enable persistent UAV missions. In *Infotech@Aerospace 2011*, volume 21, St. Louis, MO, USA, 2011.

V. Trianni. *Evolutionary swarm robotics: Evolving self-organising behaviours in groups of autonomous robots*, volume 108. Springer Berlin Heidelberg, Heidelberg, Germany, 2008.

V. Trianni. Evolutionary robotics: Model or design? *Frontiers in Robotics and AI*, 1:13, 2014.

V. Trianni and A. Campo. *Fundamental Collective Behaviors in Swarm Robotics*, pages 1377–1394. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.

V. Trianni, R. Groß, T. H. Labella, E. Şahin, and M. Dorigo. Evolving aggregation behaviors in a swarm of robots. In W. Banzhaf, J. Ziegler, T. Christaller, Peter Dittrich, and J. T. Kim, editors, *Advances in Artificial Life*, pages 865–874, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

V. Trianni, S. Nolfi, and M. Dorigo. Cooperative hole avoidance in a swarm-bot. *Robotics and Autonomous Systems*, 54(2):97 – 103, 2006.

P. Tripicchio, M. Satler, M. Unetti, and C. A. Avizzano. Confined spaces industrial inspection with micro aerial vehicles and laser range finder localization. *International Journal of Micro Air Vehicles*, 10(2):207–224, 2018.

B. Troub, B. DePineuil, and C. Montalvo. Simulation analysis of a collision-tolerant micro-airship fleet. *International Journal of Micro Air Vehicles*, 9(4):297–305, 2017.

E. Tsykunov and D. Tsetserukou. WiredSwarm: High resolution haptic feedback provided by a swarm of drones to the user's fingers for vr interaction. In *25th ACM Symposium on Virtual Reality Software and Technology*, VRST '19, New York, NY, USA, 2019. Association for Computing Machinery.

E. Tsykunov, L. Labazanova, A. Tleugazy, and D. Tsetserukou. Swarmtouch: Tactile interaction of human with impedance controlled swarm of nano-quadrotors. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4204–4209, Madrid, Spain, 2018.

G. Valentini. *Achieving Consensus in Robot Swarms: Design and Analysis of Strategies for the best-of-n Problem*, volume 706. Springer International Publishing, Cham, Switzerland, 2017.

J. van den Berg, M. Lin, and D. Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *2008 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1928–1935, Pasadena, CA, USA, 2008.

J. van den Berg, S. J. Guy, M. Lin, and D. Manocha. Reciprocal n-body collision avoidance. In C. Pradalier, R. Siegwart, and G. Hirzinger, editors, *Robotics Research*, pages 3–19, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

S. van der Helm, M. Coppola, K. N. McGuire, and G. C. H. E. de Croon. On-board range-based relative localization for micro air vehicles in indoor leader–follower flight. 44 (3):415–441, 2020.

K. van Hecke, G. C. H. E. de Croon, L. van der Maaten, D. Hennes, and D. Izzo. Persistent self-supervised learning: From stereo to monocular vision for obstacle avoidance. *International Journal of Micro Air Vehicles*, 10(2):186–206, 2018.

M. van Steen. *Graph theory and Complex Networks: An introduction*. M. van Steen, 2010.

G. Vásárhelyi, C. Virágh, G. Somorjai, T. Nepusz, A. E. Eiben, and T. Vicsek. Optimized flocking of autonomous drones in confined environments. *Science Robotics*, 3(20), 2018.

C. I. Vasile and C. Belta. An automata-theoretic approach to the vehicle routing problem. In *Robotics: Science and Systems*, Berkeley, CA, USA, 2014.

B. Vedder, H. Eriksson, D. Skarin, J. Vinter, and M. Jonsson. Towards collision avoidance for commodity hardware quadcopters with ultrasound localization. In *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 193–203, Denver, CO, USA, 2015.

A. S. Vempati, H. Khurana, V. Kabelka, S. Flueckiger, R. Siegwart, and P. Beardsley. A virtual reality interface for an autonomous spray painting UAV. *IEEE Robotics and Automation Letters*, 4(3):2870–2877, 2019.

C. J. M. Verhoeven, M. J. Bentum, G. L. E. Monna, J. Rotteveel, and J. Guo. On the origin of satellite swarms. *Acta Astronautica*, 68(7):1392 – 1395, 2011.

L. Verma, M. Fakharzadeh, and S. Choi. Wifi on steroids: 802.11ac and 802.11ad. *IEEE Wireless Communications*, 20(6):30–35, 2013.

D. A. Vincenzi, B. A. Terwilliger, and D. C. Ison. Unmanned aerial system (UAS) human-machine interfaces: New paradigms in command and control. *Procedia Manufacturing*, 3:920 – 927, 2015. 6th International Conference on Applied Human Factors and Ergonomics (AHFE 2015) and the Affiliated Conferences, AHFE 2015.

M. Voskuijl, J. van Bogaert, and A. G. Rao. Analysis and design of hybrid electric regional turboprop aircraft. *CEAS Aeronautical Journal*, 9(1):15–25, 2018.

A. ŠoŠić, W. R. KhudaBukhsh, A. M. Zoubir, and H. Koeppl. Inverse reinforcement learning in swarm systems. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '17, pages 1413–1421, Richland, SC, 2017. International Foundation for Autonomous Agents and Multiagent Systems.

V. Walter, N. Staub, M. Saska, and A. Franchi. Mutual localization of UAVs based on blinking ultraviolet markers and 3D time-position hough transform. In *IEEE 14th International Conference on Automation Science and Engineering (CASE)*, pages 298–303, Munich, Germany, 2018.

V. Walter, N. Staub, A. Franchi, and M. Saska. UVDAR system for visual relative localization with application to leader-follower formations of multirotor UAVs. *IEEE Robotics and Automation Letters*, 4(3):2637–2644, 2019.

H. Weimerskirch, J. Martin, Y. Clerquin, P. Alexandre, and S. Jiraskova. Energy saving in flight formation. *Nature*, 413(6857):697–698, 2001.

A. Weinstein, A. Cho, G. Loianno, and V. Kumar. Visual inertial odometry swarm: An autonomous swarm of vision-based quadrotors. *IEEE Robotics and Automation Letters*, 3(3):1801–1807, 2018.

S. Weiss, M. Achtelik, L. Kneip, D. Scaramuzza, and R. Siegwart. Intuitive 3D maps for MAV terrain exploration and obstacle avoidance. *Journal of Intelligent & Robotic Systems*, 61(1):473–493, 2011.

J. Werfel and R. Nagpal. Three-dimensional construction with mobile robots and modular blocks. *International Journal of Robotics Research*, 27(3-4):463–479, 2008.

J. Werfel, K. Petersen, and R. Nagpal. Designing collective behavior in a termite-inspired robot construction team. *Science*, 343(6172):754–758, 2014.

J. Wessnitzer, A. Adamatzky, and C. Melhuish. Towards self-organising structure formations: A decentralized approach. In J. Kelemen and P. Sosík, editors, *Advances in Artificial Life*, pages 573–581, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

D. Willemsen, M. Coppola, and G. C. H. E. de Croon. MAMBPO: Sample-efficient multi-robot reinforcement learning using learned world models. *arXiv preprint arXiv:2103.03662*, 2021.

A. F. T. Winfield and J. Nembrini. Safety in numbers: Fault-tolerance in robot swarms. *International Journal of Modelling, Identification and Control*, 1(1):30, 2006.

A. F. T. Winfield and J. Nembrini. Emergent swarm morphology control of wireless networked mobile robots. In R. Doursat, H. Sayama, and O. Michel, editors, *Morphogenetic Engineering: Toward Programmable Complex Systems*, pages 239–271. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

A. F. T. Winfield, C. J. Harper, and J. Nembrini. *Towards Dependable Swarms and a New Discipline of Swarm Engineering*, pages 126–142. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005a.

A. F. T. Winfield, J. Sa, M. C. Fernández-Gago, C. Dixon, and M. Fisher. On formal specification of emergent behaviours in swarm robotic systems. *International Journal of Advanced Robotic Systems*, 2(4):39, 2005b.

A. F. T. Winfield, C. F. Harper, and J. Nembrini. Towards the application of swarm intelligence in safety critical systems. In *The First Institution of Engineering and Technology International Conference on System Safety*, pages 7 pp.–, London, UK, 2006.

A. F. T. Winfield, W. Liu, J. Nembrini, and A. Martinoli. Modelling a wireless connected swarm of mobile robots. *Swarm Intelligence*, 2(2):241–266, 2008.

R. J. Wood, R. Nagpal, and G.-Y. Wei. Flight of the robobees. *Scientific American*, 308(3): 60–65, 2013.

Y. Yamauchi and M. Yamashita. Pattern formation by mobile robots with limited visibility. In T. Moscibroda and A. A. Rescigno, editors, *Structural Information and Communication Complexity: 20th International Colloquium, SIROCCO 2013, Ischia, Italy, July 1-3, 2013, Revised Selected Papers*, pages 201–212, Cham, Switzerland, 2013. Springer International Publishing.

Y. Yamauchi and M. Yamashita. Randomized pattern formation algorithm for asynchronous oblivious mobile robots. In F. Kuhn, editor, *Distributed Computing*, pages 137–151, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

D. Yamins and R. Nagpal. Automated global-to-local programming in 1-D spatial multi-agent systems. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS) - Volume 2*, pages 615–622, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems.

G.-Z. Yang, J. Bellingham, P. E. Dupont, P. Fischer, L. Floridi, R. Full, N. Jacobstein, V. Kumar, M. McNutt, R. Merrifield, B. J. Nelson, B. Scassellati, M. Taddeo, R. Taylor, M. Veloso, Z. L. Wang, and R. J. Wood. The grand challenges of science robotics. *Science Robotics*, 3(14), 2018.

K. Yousif, A. Bab-Hadiashar, and R. Hoseinnezhad. An overview to visual odometry and visual SLAM: Applications to mobile robotics. *Intelligent Industrial Systems*, 1(4):289–311, 2015.

Q. Yuan, J. Zhan, and X. Li. Outdoor flocking of quadcopter drones with decentralized model predictive control. *ISA Transactions*, 71:84 – 92, 2017. Special issue on Distributed Coordination Control for Multi-Agent Systems in Engineering Applications.

D. Zou, P. Tan, and W. Yu. Collaborative visual SLAM for multiple agents: A brief survey. *Virtual Reality & Intelligent Hardware*, 1(5):461 – 482, 2019.

J.-C. Zufferey, A. Beyeler, and D. Floreano. Autonomous flight at low altitude using light sensors and little computational power. *International Journal of Micro Air Vehicles*, 2 (2):107–117, 2010.

J.-C. Zufferey, S. Hauert, T. Stirling, S. Leven, J. Roberts, and D. Floreano. Aerial collective systems. In *Handbook of Collective Robotics*, pages 609–659. Pan Stanford, New York, NY, USA, 2013.

CURRICULUM VITÆ

Mario COPPOLA

4 August 1992	Born in Grottaglie, Italy

EDUCATION

2016–2021	Ph.D. in Aerospace Engineering Delft University of Technology, Delft, the Netherlands *Thesis:* Automatic design of verifiable robot swarms *Promotors:* Prof. Dr. G. C. H. E. de Croon, Prof. Dr. E. K. A. Gill, and Dr. J. Guo
2013–2016	M.Sc. in Aerospace Engineering (with honors certificate) Delft University of Technology, Delft, the Netherlands *Thesis:* Relative localization for collision avoidance in micro air vehicle teams
Fall 2012	Minor in Robotics (exchange) Nanyang Technological University, Singapore
2010–2013	B.Sc. in Aerospace Engineering Delft University of Technology, Delft, the Netherlands
2003–2010	International Baccalaureate (IB) International School Eindhoven, Eindhoven, the Netherlands

EXPERIENCE

09/2016–08/2020	PhD Candidate, Delft University of Technology Delft, the Netherlands
02/2015–05/2015	Researcher (intern), Max Planck Institute for Biological Cybernetics Tübingen, Germany
07/2014–01/2015	R&D Scientist (intern), Honeywell Aerospace Brno, Czech Republic

ACKNOWLEDGEMENTS

I believe that there is a lot more to a PhD program than just sitting down, doing some research, and writing about it. A PhD is a creative long-term endeavor, and much like any creative endeavor it is deeply impacted by its environment. As we all know, an environment is only as good as its people. We have all heard the saying: "people make the place", and so far I have always found that to be true. Throughout my PhD, I have been incredibly fortunate to have always been surrounded by intelligent, kind, thoughtful, and overall pretty swell people. These people have supported me, laughed with me, and, most importantly for my personal and professional growth, were also willing to challenge me when necessary. This thesis, and also yours truly, would not be the same without them (I mean that in a good way, obviously), and for that I am grateful.

First and foremost, I would like to thank my promotors: Guido de Croon, Eberhard Gill, and Jian Guo. Back in 2015 I sent Guido a short email to explore the possibility of doing an MSc thesis at the MAVLab. Little did I know that I would enjoy the thesis so much that I would eventually choose to stay on for a PhD. Guido, your energy, intelligence, wide knowledge, and your ability to maintain a light-hearted atmosphere at all times have always been inspiring. I always enjoyed our meetings and discussions throughout these years, which were always also very helpful — thank you. Eberhard, your vast experience as an engineer and as a professor was a great source of knowledge for me, and I always appreciated your willingness to share it with me. Thank you for all the feedback and insights throughout these years (and more, if you consider that I took some of your courses during my BSc!). Jian, you never failed to provide great insight and to make time for me when I needed it, guiding me along this process. Thank you for all the constant and valuable support from the very beginning.

I did my PhD in a joint collaboration between two groups in the Aerospace Engineering faculty: the MAVLab (from the Control & Simulation section), and the Space Systems Engineering section. Both were incredibly welcoming from day one, and I thank all of you for all the amazing times; from the innumerous coffee breaks to all our various outings, dinners, drinks, conferences, sport events, trips, and more, that we've had since then.

Starting with the MAVLab, thanks to: Diana Olejnik, Federico Paredes Vallés, Tom van Dijk, Shushuai Li, Yingfu Xu, Shuo Li, Sven Pfeiffer, Kimberly McGuire, Kirk Scheper, Sjoerd Tijmons, Matěj Karásek. Hann Woei Ho, Sunyou Hwang, Ziqing Ma, Ewoud Smeur, Nilay Sheth, Julien Dupeyroux, Erik van der Horst, Christophe De Wagter, Bart Remes, Kevin van Hecke, Freek van Tienen, Jesse Hagenaars, Roland Meertens, Michaël Ozo, and also many others that came and went. I have always enjoyed the strong team spirit in the MAVLab. Thanks to all of you for all the good times, in and out of the office, and also for all the help during these last years. A special thanks to Kimberly and Kirk, who helped me immensely during my MSc thesis and inspired me to stay on for "a little longer", and who I have also had the pleasure of publishing papers with. Also a special thanks to

Christophe and Shushuai, who I am also proud to have written papers with, and to Erik, for always making time to help even on the shortest of notices.

In addition, thanks to all the other C&S PhDs during my years here. Whether it was by the coffee machine, or at a barbecue, or at a dinner, or during drinks, or sailing, or during flight practicals, you all made the SIMONA office an amazing place. Thanks to: Dirk van Baelen, Jaime Junell, Malik Doole, Noor Nabi, Sihao Sun, Daniel Friesen, Jelmer Reitsma, Bo Sun, Jerom Maas, Isabel Metz, Sarah Barendswaard, Paolo Scaramuzzino, Dyah Jatiningrum, Tommaso Mannucci, Sophie Armanini, Annemarie Landman, Ivan Miletović, Junzi Sun (now staff!), Kasper van der El, Jan Smisek, Ye Zhang, Emmanuel Sunil, Wei Fu, Ye Zhou, Ezgi Akel, Julia Rudnyk, Rolf Klomp, and Xuerui Wang. And also thanks to the staff: Max Mulder, Olaf Stroosma, Daan Pool, Jacco Hoekstra, Erik-Jan van Kampen, Marilena Pavel, Bob Mulder, Alexander in 't Veld, Joost Ellerbroek, Hans Mulder, Coen de Visser, Anahita Jamshidnejad, Bertine Markus, Clark Borst, Qiping Chu, Andries Muis, Ferdinand Postema, Menno Klaassen, Olaf Grevenstuk, Harold Thung, and the many more who also joined over the years. Also thanks to Lorenzo Martini. A special thanks to Max for his help and support during my MSc and during the early phases of my PhD. Max, your guidance was always very helpful, going back all the way to when I started my MSc and I was wondering what internships I could do. With your help I managed to get an internship at the MPI in Tübingen, leading me to first discover my passion for robotics research, which in turn led me to the MAVLab upon my return, and now here we are. A special thanks also to Bertine for helping me arrange all sorts of things over these last years.

Let's now move on to the Space Systems Engineering section, where I would like to start by thanking all the amazing fellow 8th floor PhDs. Thank you for the outings, coffees, drinks (yeah, I know, I should have joined the drinks more often), ice-karting (because that deserves its own spot on the list), and more. Most importantly, thank you for always keeping the spirits up, even during these crazy times. Thanks to: Johan Carvajal Godínez, Fiona Leverone, Victor Villalba Corbacho, Stefano Casini, Stefano Di Mascio, Ruipeng Liu, Linyu Zhu, Pengyu Wang, Erdem Turan, Marsil de Athayde Costa e Silva, Daduí Cordeiro Guerrieri, Adolfo Chaves Jiménez, Lorenzo Pasqualetto Cassinis, Zixuan Zheng, Minghe Shan, and Dennis Dolkens. Also, a big thank you to the staff: Debby van der Sande, Mariëlle Hoefakker, Prem Sundaramoorthy, Chris Verhoeven, Stefano Speretta, Alessandra Menicucci, Barry Zandbergen, Mehmet Uludağ, Angelo Cervone, Vidhya Pallichadath, Barry Zandbergen, Hans Kuiper, Jasper Bouwmeester, Botchu Jyoti, and others that came and went, even if for a short while. An additional thanks to Debby and Mariëlle for always having been ready to help me arrange everything that I needed. Also a special thanks to Chris for his valuable advise in the early stages of my PhD, which prepared me for what was to come.

During these years I also had the pleasure of guiding some MSc students, working together with them to develop or test new ideas, and getting to very interesting results and findings. For that, thanks to Steven van der Helm, Thomas Fijen, Andrés Ripoll Sánchez, Eduardo Lourenço, and Daniël Willemsen for all the amazing work and dedication.

To all the past and current members of the PhD council, which I've had the pleasure of joining for more than a year: keep up all the great work. Organizing and keeping the PhD spirit up is more important now than ever, and I am proud to have joined you during

my time here (and thanks to Dirk for encouraging me to do so).

During my time as a PhD I have also published multiple papers, each of which went through a careful peer-review process. I would like to thank all the editors and reviewers that took part. Your insightful and thoughtful feedback was key to improving the quality of my research as well as the clarity of my explanations — thank you.

It goes without saying (but I'll say it anyway) that this PhD and my time in Delft as a whole would also not have been what they were if it wasn't for some other amazing people that I had the pleasure of meeting along the way, starting from when I arrived here as a fresh-out-of-high-school-how-do-I-adult 18-year-old. I will always cherish the special bonds that I have made during my years here, which have inspired me to be a better person, encouraged me to try new crazy things, supported me when I needed it, helped me grow, were always down for a good joke, and ultimately never failed to put a big stupid smile on my face. I owe you so much. Thank you for everything.

Last but most definitely not least, vorrei ringraziare la mia famiglia. Mamma, Papà, e Pier, tutti e tre siete sempre stati incredibilmente di supporto durante tutti i possibili ups and downs, e non avrei potuto chiedere di più da nessuno di voi — questo PhD è anche vostro. Anche un grazie immenso ai miei nonni, zii, zie, e cugini. Anche se purtroppo le varie distanze (e una pandemia) ci hanno portato a non vederci spesso come vorremmo, vi ho sempre sentiti vicino, e questo mi ha sempre dato forza.

Delft, March 2021

LIST OF PUBLICATIONS

JOURNAL PUBLICATIONS

5. **Mario Coppola**, Kimberly N. McGuire, Christophe De Wagter, and Guido C. H. E. de Croon. A survey on swarming with micro air vehicles: Fundamental challenges and constraints. *Frontiers in Robotics and AI*, 7:18, 2020.

4. Steven van der Helm, **Mario Coppola**, Kimberly N. McGuire, and Guido C. H. E. de Croon. On-board range-based relative localization for micro air vehicles in indoor leader-follower flight. *Autonomous Robots*, 44(3):415–441, 2020.

3. **Mario Coppola**, Jian Guo, Eberhard Gill, and Guido C. H. E. de Croon. The PageRank algorithm as a method to optimize swarm behavior through local analysis. *Swarm Intelligence*, 13(3):277–319, 2019.

2. **Mario Coppola**, Jian Guo, Eberhard Gill, and Guido C. H. E. de Croon. Provable self-organizing pattern formation by a swarm of robots with limited knowledge. *Swarm Intelligence*, 13(1):59–94, 2019.

1. **Mario Coppola**, Kimberly N. McGuire, Kirk Y. W. Scheper, and Guido C. H. E. de Croon. On-board communication-based relative localization for collision avoidance in micro air vehicle teams. *Autonomous Robots*, 42(8):1787–1805, 2018.

CONFERENCE PUBLICATIONS (PEER-REVIEWED)

2. **Mario Coppola** and Guido C. H. E. de Croon. Optimization of swarm behavior assisted by an automatic local proof for a pattern formation task. In Marco Dorigo, Mauro Birattari, Christian Blum, Anders L. Christensen, Andreagiovanni Reina, and Vito Trianni, editors, *Swarm Intelligence — 11th International Conference, ANTS 2018, Rome, Italy, Lecture Notes in Computer Science, volume 11172*, pages 123–134, Cham, Switzerland, 2018.

1. Kimberly N. McGuire, **Mario Coppola**, Christophe De Wagter, and Guido C. H. E. de Croon. Towards autonomous navigation of multiple pocket-drones in real-world environments. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 244–249, Vancouver, BC, Canada, 2017.

PREPRINTS

3. **Mario Coppola**, Jian Guo, Eberhard Gill, and Guido C. H. E. de Croon. A model-based framework for learning transparent swarm behaviors. *arXiv preprint arXiv: 2103.05343*, 2021.

2. Daniël Willemsen, **Mario Coppola**, and Guido C. H. E. de Croon. MAMBPO: Sample-efficient multi-robot reinforcement learning using learned world models. *arXiv pre-print arXiv:2103.03662*, 2021. [Submitted to *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2021*, currently in review]

1. Shushuai Li, **Mario Coppola**, Christophe De Wagter, and Guido C. H. E. de Croon. An autonomous swarm of micro flying robots with range-based relative localization. *arXiv preprint arXiv:2003.05853*, 2020. [Updated version submitted to *IEEE Transactions on Robotics*, currently in review]

ISBN: 978-94-6421-287-7

Propositions

accompanying the dissertation

AUTOMATIC DESIGN OF VERIFIABLE ROBOT SWARMS

by

Mario COPPOLA

1. Before applying swarm intelligence to a system, one needs to think about whether the swarm's flexibility is at odds with the other system requirements (this thesis, Chapter 2).
2. Randomness and sub-optimality are fundamental limitations of swarming, but they are needed to provide scalability, flexibility, and robustness (this thesis, Chapter 3).
3. A community-wide effort to develop high quality and open access swarm datasets can accelerate the understanding and the development of the swarm research field (this thesis, Chapter 5).
4. Evolutionary robotics and reinforcement learning should not be treated as competing paradigms by the swarm robotics community (this thesis, Chapter 5).
5. The assumption that something is irreducible (i.e., that it cannot be explained by its parts) limits the ability to study it, regardless of whether it is in fact irreducible.
6. We are not in control of the properties of our creations. Following the initial conception, they can only be discovered.
7. Subjective notions should not be used to define emergence.
8. The main lesson that a PhD program teaches is the value of taking small daily steps, sometimes even in the wrong direction, in order to achieve a long-term goal.
9. The digital age requires a revamp of scientific publishing so as to maximize the opportunities provided by media other than print.
10. Setting aside time for physical exercise alongside research is beneficial to the final research outcome and should be encouraged.

These propositions are regarded as opposable and defendable, and have been approved as such by the promotors Prof. Dr. G. C. H. E. de Croon, Prof. Dr. E. K. A. Gill, and Dr. J. Guo.

Stellingen

behorende bij het proefschrift

AUTOMATIC DESIGN OF VERIFIABLE ROBOT SWARMS

door

Mario COPPOLA

1. Voordat men zwermintelligentie toepast op een systeem, moet men zich afvragen of de flexibiliteit van de zwerm niet op gespannen voet staat met de andere systeemeisen (dit proefschrift, hoofdstuk 2).
2. Willekeurigheid en sub-optimaliteit zijn fundamentele beperkingen van zwermen, maar ze zijn nodig om schaalbaarheid, flexibiliteit en robuustheid te bieden (dit proefschrift, hoofdstuk 3).
3. Een gemeenschapsbrede inspanning om hoge kwaliteit en open toegang swarm datasets te ontwikkelen kan het begrip en de ontwikkeling van het onderzoeksveld versnellen (dit proefschrift, hoofdstuk 5).
4. Evolutionaire robotica en reinforcement learning moeten door de zwermrobotica gemeenschap niet als concurrerende paradigma's worden behandeld (dit proefschrift, hoofdstuk 5).
5. De aanname dat iets onherleidbaar is (d.w.z. dat het niet verklaard kan worden uit zijn onderdelen) beperkt de mogelijkheid om het te bestuderen, ongeacht of het in feite onherleidbaar is.
6. Wij hebben geen controle over de eigenschappen van onze scheppingen. Na de eerste conceptie kunnen ze alleen nog worden ontdekt.
7. Subjectieve begrippen mogen niet worden gebruikt om de opkomst te definiëren.
8. De belangrijkste les die een doctoraatsprogramma ons leert, is de waarde van het nemen van kleine dagelijkse stappen, soms zelfs in de verkeerde richting, om een doel op lange termijn te bereiken.
9. Het digitale tijdperk vereist een vernieuwing van het wetenschappelijk publiceren om de mogelijkheden van andere media dan drukwerk maximaal te benutten.
10. Tijd uittrekken voor lichaamsbeweging naast het onderzoek is bevorderlijk voor het uiteindelijke onderzoeksresultaat en moet worden aangemoedigd.

Deze stellingen worden opponeerbaar en verdedigbaar geacht en zijn als zodanig goedgekeurd door de promotors prof. dr. G. C. H. E. de Croon, prof. dr. E. K. A. Gill, en dr. J. Guo.