Formation Control Algorithms With Limited or No Communication

Master of Science Thesis
University of Turku
Department of Future Technologies
Turku Intelligent Embedded & Robotic Systems
2020
Cassandra McCord

UNIVERSITY OF TURKU
Department of Future Technologies

CASSANDRA MCCORD: Formation Control Algorithms With Limited or No Communication

Master of Science Thesis, 39 p.
Turku Intelligent Embedded & Robotic Systems
July 2020

Formation control refers to a collective behaviour of multi-agent systems where individual agents come together to form a pattern, often geometric. These formations can enable multi-agent systems to function more effectively in a broad range of applications. Many formation control algorithms require centralized decision making, communication between agents or a centralized decision maker and other factors that increase per-agent cost and reduce the robustness and scalability of multi-agent systems. To this end, we introduce two algorithms that operate using local decision making and limited or no communication. The first algorithm is a communication-free and index-free algorithm based on polar indicator distributions. The second is a progressive assignment algorithm using limited, situated communication that deterministically assigns agents a position in the objective formation along a convex spiral directed path graph. We also present an extension of the second algorithm for 3-dimensional formation definitions. The first algorithm is demonstrated in a physical experiment using ground-based agents while the second one is simulated using micro air vehicles (MAVs) in a physics-based simulator.

Keywords: Multi-Agent Systems,Mobile Robots, Formation Control, Distributed Control, Progressive Formation Control, Robot Operating System, Pattern Configuration, Index-Free Control

Contents

List of Figures

1 Introduction

Formation control refers to a collective behaviour of multi-agent systems where individual agents come together to form a pattern, often geometric. These formations can enable multi-agent systems to function more effectively in a broad range of applications including satellites and search-and-rescue [1]. Formation control systems can also be used to model collective behaviours found in nature, such as insect swarms and bird formations.

1.1 Motivation

The primary goal of this thesis is to explore formation control algorithms that operate under limited communication situations and allow for arbitrary patterns to be achieved. Since for many applications of formation control all agents involved are equivalent, or anonymous, distributed algorithms are of particular interest.

Multi-agent systems by definition consist of multiple agents. The costs, both monetarily and in terms of power usage, of each agent is magnified across the entire system. Thus, there is incentive to minimize the per-agent costs. Communication hardware can have a significant effect on these costs, so minimizing communication between agents reduces costs. Moreover, even though ubiquitous connectivity is becoming a trend across multiple domains, including robotics, interaction models based on local behaviour observation can bring increased robustness and scalability to multi-agent systems.

To this end we introduce two algorithms that operate under conditions with reduced the absence of communication between agents. The first one, described in chapter 4, describes an algorithm where agents do not communicate at all. The second, described in chapter 5 and modified in chapter 6, uses one way broadcast communication between agents.

1.2 Structure

The structure of this thesis, flowing from abstract to concrete, is as follows

Chapter 2 and 3 give an overview of the field of Formation Control and the common types of algorithms used therein. We also explore hardware and software considerations, in particular with respect to the choices used for our own project.

Chapters 4 to 6 offer an in depth formulation of our proposed algorithms, starting with a entirely communication-free algorithm, followed by a progressive assignment based algorithm with limited communication in two dimensions, and finally an extension of the progressive assignment algorithm into three dimensions.

Chapter 7 covers the implementation and results of our proposed algorithms. We discuss aspects of implementation including difficulties encountered and suggestions for future improvements. We also provide rough comparisons of the algorithms presented.

2 Formation Control

Many applications either require or benefit from the cooperation of multiple agents.

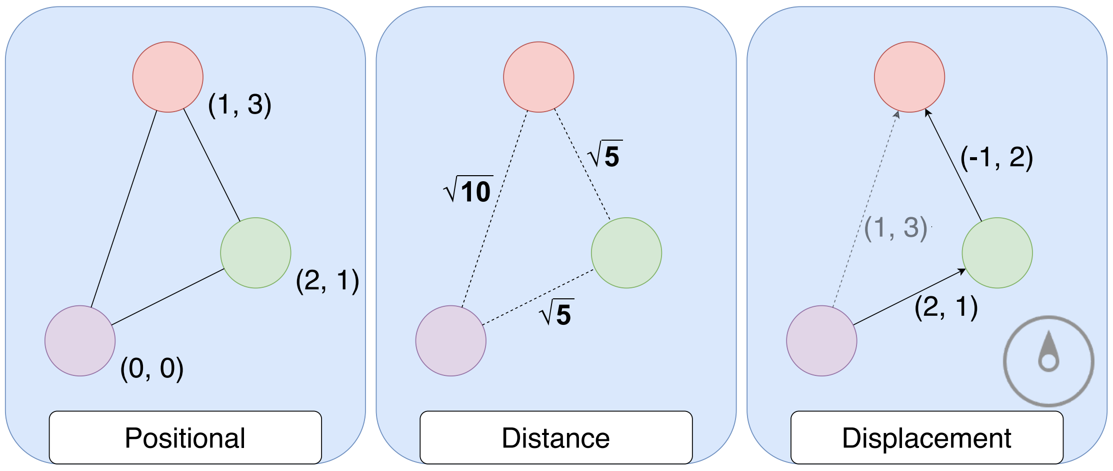
We are primarily concerned with static formation for the purposes of this thesis, rather than formation tracking. Static formation refers to achieving some objective formation. Once the objective formation has been achieved it is considered a success. Formation tracking instead focuses on maintaining a formation as it moves through space, although it may implicitly include the initial achievement of the formation.

We can classify formation control algorithms based on a number of parameters [2]: how the objective formation is defined, how much or how little agents are allowed to communicate, where decisions related to the formation are made, and whether the agents are indexed or not. Each algorithm can then be described as a combination of these parameters. For instance, chapter 4 describes a communication-free, anonymous (index-free), decentralized formation control algorithm, whereas chapters 5 and 6 describe a displacement-based, limited communication, decentralized and indexed algorithm.

2.1 Objective Formation Definition

The objective formation definition refers to how the overall structure of the formation is specified. The three main approaches to this are position-, distance- and displacement-based [2].

This list is not exhaustive and sometimes it can be difficult to categorize an algorithm strictly into one of these categories [3]. For instance, bearing-based algorithms, which



(a) Positional-based (b) Distance-based (c) Displacement-based

Figure 2.1: Objective formation definition types

can be used to form regular polygons by maintaining a constant angle between neighbors, could be categorized as displacement-based except that there is no requirement on how close or far the agents are.

Position-based By far the most flexible of the approaches. Each agent is assigned an absolute position in the formation. Necessitates that all agents have a common frame of reference for both position and orientation.

Distance-based Each agent tries to maintain distance to nearby agents. No common frame of reference is necessary. It can be used to model flocking behavior. Requires at distances defined for at least 2 neighbors to be stable in 2 dimensions and 3 in 3 dimensions.

Displacement-based Each agent attempts to maintain a particular displacement from its neighbors [1]. Agents require a common frame of reference for orientation but not position. Each agent may have one or more displacements defined or displacements may be symmetrically defined.

2.2 Level of Communication

Another factor to consider when selecting an algorithm or designing your agents is how much communication between agents is necessary or feasible. Communication hardware increases the cost of each agent and also increases power consumption. However, using more communication generally makes achieving more complex behavior easier.

Communication-Free Agents do not directly communicate with each other at all. They may still sense neighbors.

Limited Communication Agents can communicate through limited means. Communication may rely on short-range signals such as situated communication [4] or through visual means. Communication may be one-way, such as only being able to broadcast.

Full Communication Any agent is able to communicate with any other agent.

2.3 Decision Making

Similarly, many formation control algorithms require making decisions, such as which position in the objective formation an agent should be assigned to. Choosing to have a single decision-maker can simplify algorithms but is also more susceptible to failure. Decentralization often complicates algorithm design but is more robust under failure.

Centralized One agent or an outside source of truth. Makes decisions for all agents in play. Requires full communication.

Decentralized No single source of truth. Each agent makes decisions for itself. May still coordinate with other agents. Consensus-based with higher communication or local-only decision making in the absence of communication.

2.4 Indexing

Anonymous or index-free formation control algorithms are well suited to groups of homogeneous agents, that is that each agent is functionally equivalent to each other and can successfully fulfill any role in the formation. In nature, homogeneous swarms are common, such as schools of fish or flocks of birds. This is contrasted with heterogeneous formation, where each agent is not necessarily considered equivalent to every other agent. An example of a heterogeneous swarm might be a mother duck leading her ducklings. Indexed algorithms, however, make no assumption about the equivalency of agents. They can be applied to both homogeneous and heterogeneous groups of agents. Agents might be assigned a particular role or position based on some context, such as having extra capabilities or proximity to the desired position. Indices can be preassigned or assigned in real-time.

Most current work has focused on indexed algorithms [2], [5], [6]. Some work has been done using permutation-invariance, that is algorithms that function identically even when agent indices are shuffled [7]–[9]. One of the first truly index-free formation control algorithms uses indicator distributions [10].

Progressive formation control algorithms are an interesting combination of index-free and indexed formation control [11]. Initially, all agents start out anonymous but then, through some process, the agents receive an ID and are added to the formation. The algorithm presented in chapters 5 and 6 is a progressive formation control algorithm.

Anonymous Also called index-free. Each agent is equivalent and can fulfill any role in the formation.

Indexed Agents are assigned specific roles in the formation.

2.5 Considerations

When selecting an algorithm there are a number of factors to consider. Certain constraints might rule out certain classes of algorithms. Among the factors to consider are processing power, availability of sensors and application requirements. In an ideal situation with unlimited processing power and perfect information, any objective formation can be achieved. However, processing power, sensors, and communication hardware all cost money and energy. In many embedded applications, the availability of energy is limited and must be carefully rationed and when a large number of agents are needed, the cost of an individual agent becomes more significant.

3 Robotics Hardware and Software

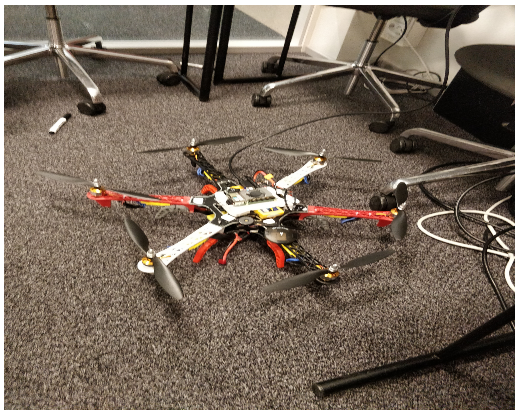
3.1 Robotics Hardware

This is only a very brief overview of robotics hardware with a narrow focus on hardware related to formation control and in particular hardware used throughout the development of this thesis. Actuators and other methods of interacting with the environment, for instance, are not discussed at all. We broadly discuss types of vehicles used, onboard processing and the necessary sensors. Each of these is a component of an agent but no single component defines the agent itself.

3.1.1 Vehicles

Each agent generally needs some method of moving around in the environment, otherwise moving to achieve an objective formation would not be possible.

Drones Drones are unmanned aerial vehicles. They are generally divided into two categories: fixed-wing and rotary-wing. Fixed-wing drones are similar to commercial jets and fly by thrusting forward and relying on aerodynamics to generate lift. They require constant motion to stay aerial. Rotary-wing drones, such as helicopters, generate lift directly by rotating large blades. They are capable of maintaining a constant position in space. Rotary-wing drones with four or six rotors, often called quad- or hexacopters respectively, are commonly used in embedded applications due to their stability, maneuverability, and simpler control compared to fixed-wing.


(a) Example drone

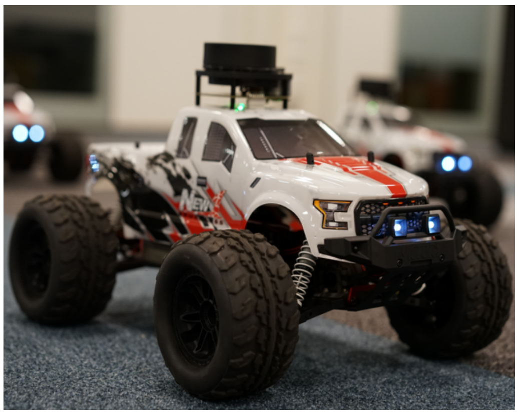

(b) Example ground vehicle

Figure 3.1: Example autonomous vehicles. 3.1b shows an agent used in section 7.1

Ground Vehicles Unmanned ground vehicles are also incredibly varied in terms of size and function. Small radio controlled cars can be adapted into an autonomous agent by replacing the radio control hardware. Self-driving cars are another example of unmanned ground vehicles. Similarly, robotic vacuum cleaners are also unmanned ground vehicles. Not all ground vehicles necessarily need wheels or follow the same kinematics as normal cars.

3.1.2 Processing

Autonomous vehicles require processing to function. At the lowest level, they must process sensor data and control the vehicle's systems. On top of that, processing power is needed to perform whatever task is necessary. Certain processes might be real-time, requiring computation to be completed by strict deadlines continuously, such as the flight controller of a drone. Failure of the flight controller may result in the vehicle catastrophically failing and crashing. For this reason, low-level control of the vehicle is sometimes relegated to separate hardware, such as the Pixhawk [12]. This allows for a better separation of concerns between the high-level application and the low-level details of controlling the vehicle.

The rise of single-board computers such as the Raspberry Pi have simplified programming of embedded systems, developers are no longer restricted to C/C++ and assembly. A single-board computer is a small embedded device with strong capabilities. They usually run some flavor of Linux and allow for a large range of toolsets and applications to be used.

Still, sometimes certain calculations can benefit from older approaches. Field programmable gate arrays, or FPGAs, allow custom logic to be implemented in hardware resulting in faster, reliable execution and lower power consumption than general computing devices. Similarly, single-board computers running Linux have more overhead that can interfere with real-time systems and consumes extra power.

3.1.3 Sensors

Finally, autonomous vehicles need some way of perceiving their surrounding environment. In terms of formation control algorithms, two primary forms of sensing are needed: vision, the ability to see the shape of the environment and sense other agents located nearby, and localization, the ability to determine where the agent itself is located in the environment. Both forms are complex problems in their own right and are the subject of rigorous study [13]–[16].

Localization is the problem of determining where an agent is located. This localization can be with respect to some specific, locally defined, coordinate frame or a global coordinate frame. Localization methods are often combined to improve accuracy [16]–[18]. One common form of localization is the use of a GNSS (global navigation satellite system) sensor. The most popular among these, GPS (Global Positioning System), is a system of global positioning using satellites in orbit of earth. Since the signals used for localization, in this case, come from space, the accuracy of this method is dependent on certain factors of the environment [19]. Additionally, the resolution of the location, how precisely an agent can determine its position, can vary from by an order of magnitude, from tens of centimeters to tens of meters.

Another form of localization comes from odometry [20]. Odometry is the process of using sensor data to estimate movement over time. It requires a known initial position from which it can estimate the current location. Odometry systems are often subject to an increasingly larger error in the actual position versus projected position called drift. However, over short spans they can be very accurate.

Vision is the perception of an environment by an agent using images. Cameras are an obvious choice as a sensor, but LIDAR and other depth perceiving sensors are also very useful. Using object detection algorithms, an agent can approximate the position of nearby agents [21]. It can be used for the purpose of localization as well, using either an *a priori* map or one built on the fly, as in Simultaneous Localization and Mapping [22]–[26]. The local environment of the agent is sensed and then compared to the map to determine where the agent is.

3.2 Software and Simulation tools

3.2.1 ROS

The Robot Operating System (ROS) is an open-source framework for writing software to control robots. While not an operating system in the true sense, such as Windows or Linux, it provides a cross-language communication layer that allows robotic systems to be built in a modular and reusable way [27]. The system broadly follows the publish-subscribe architecture pattern.

ROS is based on a few key concepts from which the rest of the system falls into place. These concepts are messages, topics, nodes and services.

Messages Messages are essentially structured containers of data. They are predefined and consist of a few basic types as building blocks. More complex messages can be composited from other message types. They also generally have rich interaction in

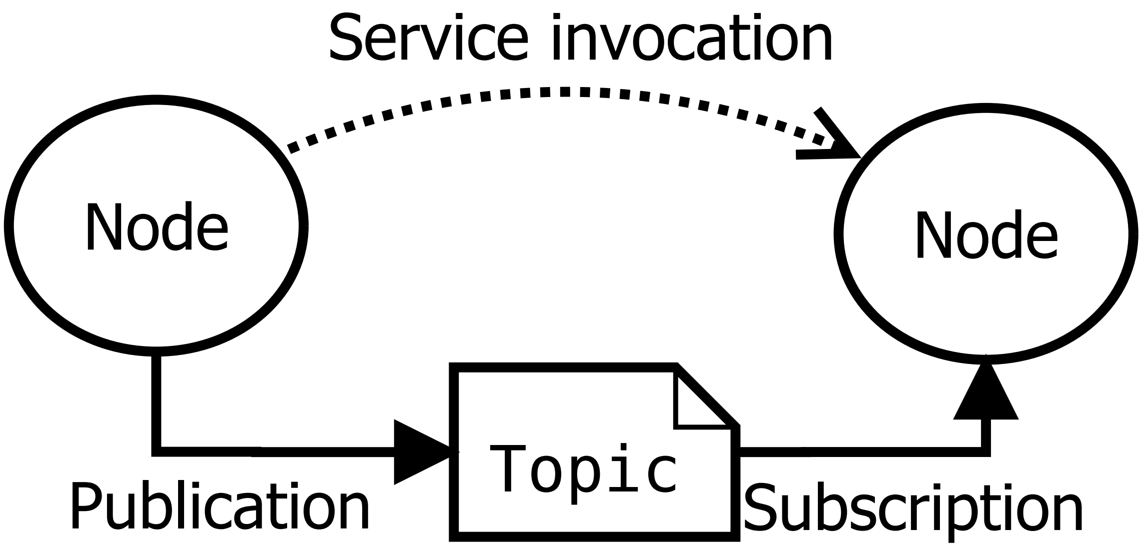


Figure 3.2: Reproduced diagram of ROS basic concepts [28]

each language. For instance, in C++ the messages are represented by native objects. These messages are the messages that get passed around from node to node through topics.

Topics Under the publish-subscribe paradigm, topics represent the various channels that messages pass through. Each topic is identified by a name. A message is then sent, or published, to a particular topic, and any node listening, or subscribed, to that topic can then receive the message. Each topic is strongly typed, that is associated with exactly one message type. It is important to note that when a node sends a message to a topic, it does not specify any recipients. Rather, any node subscribed to the topic will receive it. Conversely, any node subscribed to a topic has no information about the source of the message.

Nodes Nodes are where the bulk of the work is done. They can communicate with other nodes by sending messages to a topic or subscribing to a topic. They may call upon services offered by other nodes or offer services of their own. Nodes usually perform some specified task. They may directly control some specific hardware, collect sensor data or run calculations on data.

Services Services are defined by a pair of message types and advertised by a node. They differ from topics in two main ways. First, the service is associated with a specific node as opposed to topics which are not associated with nodes. Second, they send information back to the node that called the service.

Services resemble remote procedure calls. Some data is sent to the node called the request analogous to parameters and then some data is sent back called the response. This request/response pair, along with a name are what define the service. Unlike topics, which are many to many, only one node is allowed to advertise a given service.

3.2.2 PX4

The PX4 Drone Autopilot is a software stack designed to allow for remote control of multi-rotor UAVs. The system handles the low-level details of flying a UAV as well as managing sensor data. It can receive high level instructions, such as positional or velocity-based setpoints and transform them into a low level instructions to the motors controlling each rotor. Additionally, the software can handle fail-safes, such as low power or loss of communication with the base station, in a configurable and robust way. It can communicate wirelessly with a base station or remote control, or with a separate onboard micro-controller through, for example, a commonly used messaging protocol called MAVLink.

3.2.3 Gazebo

Gazebo is a 3D physics simulator with a focus on robotics [29]. It supports complex interactions between multiple agents in a simulated environment and is highly extensible. It also models vision based sensors such as cameras and LIDAR, allowing rapid development and testing of robotics algorithms. A large number of commercially available robots and autonomous vehicles have simulation packages and models available for Gazebo.

Although Gazebo works as a standalone simulator, it is often used in conjunction with ROS. A high level of interoperability allows developers to take advantage of existing ROS tools and libraries when developing applications using Gazebo. Gazebo can also be used for automated testing of complex applications in realistic scenarios. The ability to simulate multiple agents simultaneously is vital for formation control research. By leveraging the Gazebo simulator and ROS integration, the same codebase can be used in both simulation and practice.

RotorS

RotorS is a multi-rotor UAV simulation module for Gazebo [30]. On top of a more realistic simulation, it offers a set of multi-rotor models for various drones and a suite of tools for designing and simulating custom multi-rotor UAVs. The PX4 firmware can operate on top of RotorS as a Software in the Loop (SITL) simulation.

4 Communication-free Formation Control

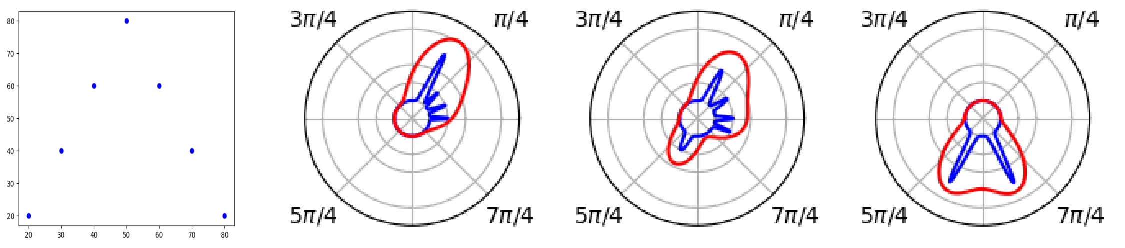
Portions of text and figures reproduced from author's previous work [31].

4.1 Overview

The algorithm presented in this chapter is both communication free and index-free. It requires agents to share a global orientation frame (e.g., through a compass) but does not require a common global reference frame. Broadly speaking, each agent has a set of polar distributions (equation (4.1)) called the objective positions, and one for its own current position. The agent then selects the best candidate among the objective position distributions based on its own distribution and then moves to minimize the difference, or error, between them.

4.2 Motivation

The motivation behind these polar distributions is that they account for errors or noise in the agents' sensors. Rather than utilizing individual points in space to define the positions of all other agents, which then would have to be matched one-to-one with positions in the objective formation configuration, the polar distributions represent a sort of probability distribution of what an agent sees around it in terms of *free* and *occupied* space.



(a) Objective pattern (b) $-\psi_{0|\tau=0}(\theta)$, $-\psi_{0|\tau>0}(\theta)$ (c) $-\psi_{1|\tau=0}(\theta)$, $-\psi_{1|\tau>0}(\theta)$ (d) $-\psi_{3|\tau=0}(\theta)$, $-\psi_{3|\tau>0}(\theta)$

Figure 4.1: Illustration of the polar indicator distribution. Figure (a) shows a wedge configuration of 7 agents. Figures (b), (c) and (d) show the polar indicator distribution of the first, second and fourth (top) agents, if indexed from left to right, for $\tau = 0$ and $\tau > 0$.

4.3 Formulation

We will only deal with the 2D case here. The full 3D formulation is found in our previous work [31].

Definition 4.1 (Polar Indicator Distribution)**.** Given a set of N agents with positions $(x_i, y_i) \in \mathbb{R}^2\ \forall\, i = 1, \dots, N$, let N_i be the set of agent i's neighbors. Their positions are measured from the perspective of agent i's local frame of reference and represented by $(x^i_j, y^i_j) \equiv (r^i_j, \theta^j_i)\ \forall j \in N_i$ in Cartesian coordinates or polar coordinates, respectively. Then we define agent i's polar indicator distribution by

$$\psi_i(\theta) = \alpha \sum_{j \in N_i} w(r^i_j)(\theta) * \delta(\theta - \theta^i_j) \tag{4.1}$$

where $\delta(\theta)$ is the Dirac delta function defined in in the subset $[0, 2\pi) \subset \mathbb{R}$, $\alpha > 0$ is a constant and $w(r)(\theta)$ is defined as $w(r)(\theta) = \exp\left(-\theta Q(r)(\theta)\right)$ where $Q(r) > 0$.

The Dirac delta function represents the bearing of a specified neighbor and $Q(r)$ modifies it as a function of distance. Because the distribution is defined over a finite range of $\mathbb{R}$, we can easily discretize it and represent it using an array in our actual implementation.

$Q(r)$ is defined as

$$Q(r) = \beta r^2 + \tau \tag{4.2}$$

for $\beta, \tau \in \mathbb{R},\ \beta, \tau > 0$ where β varies the width of $w(r)$ based on distance and τ introduces a constant minimum width. With this definition we can expand the definition of an agents distribution into equation (4.3) as illustrated in figure 4.1

$$\psi_i(\theta) = \alpha \sum_{j \in N_i} \exp\left(-\left(\beta {r_j^i}^2 + \tau\right)\left(\theta - \theta_j^i\right)^2\right) \tag{4.3}$$

4.4 Position Assignment

Each agent must autonomously assign itself a position in the formation. In some formation control algorithms, such as bearing and distance based ones, all positions are inherently equivalent and thus position assignment does not make sense. In others, positions are either explicitly assigned ahead of time, or are negotiated through some form of communication [2].

Definition 4.2. In order to achieve this, we define a cost function for achieving a desired position ψ_j^* for an agent in position ψ_i as

$$D_{\psi_i}(\psi_j^*) = \int_0^{2\pi} \left(\psi_i(\theta) - \psi_j^*(\theta)\right)^2 \, d\theta \tag{4.4}$$

from which an agent decides its objective position by minimizing

$$\psi_i^{obj} = \psi_j^* : j = \underset{j=1,\ldots,N}{\operatorname{argmin}} D_{\psi_i}(\psi_j^*),\ \ D_{\psi_i}^{obj} := D_{\psi_i}(\psi_i^{obj}) \tag{4.5}$$

Definition 4.3 (Formation objective)**.** Given a pattern configuration defined by a set of N positions, from which we can calculate their individual spherical distributions, the desired formation shape of a group of agents is represented by the set $\mathbb{E}_\psi = \{\psi_1^*, \ldots, \psi_N^*\}$ of desired spherical indicator distributions. Therefore, the formation objective is to have a permutation $\sigma \in S_N$ such that $D_{\psi_i}(\psi^*_{\sigma(i)}) < \delta$ for a constant $\delta \in \mathbb{R}$, $\delta > 0$. The value of δ is the minimum error allowed to assume a successful convergence to the desired position. The set $\{\psi_i(\theta)\}$ represents the agents' positions and $\{\psi^*_j(\theta)\}$ the desired positions.

Definition 4.2 does not ensure that there exists a permutation σ mapping the set of agents into the set of formation positions. This is an initial approach to the problem and therefore we operate under the assumption that the initial distribution of agents ensures the existence of σ.

Our next step is to minimize the cost function of achieving convergence as defined in definition 4.3. Here is where the value of τ comes in to play. Because of our definition of $\psi_i(\theta)$ from equation (4.3), if τ is too small then the minimization problem can become non-convex. $\psi_i(\theta)$ becomes comprised of several peaks with values close to zero between them. This means that a local minimum exists where several peaks line up but others do not. The value of τ can be adjusted to minimize this risk.

4.5 Collision Avoidance

The algorithm presented above does not consider collision avoidance. For example, when $r_j \to 0$, the contribution of j to ψ_i is reduced to a constant, and in particular can lead to a collision. This is especially true when neighbors are distributed densely over θ and ψ_i becomes almost constant.

The following modification to equation (4.1) introduces collision avoidance into the algorithm. We define two thresholds r_s and r_d representing the minimum distance considered safe and the minimum distance where the danger collision is likely.

$$\psi_i(\theta) = \sum_{j\in N_i} \alpha_j \exp\left(-\left(\beta {r_j^i}^2 + \tau\right)\left(\theta - \theta_j^i\right)^2\right), \alpha_j = \begin{cases} \alpha & if \;\; r_s < r_j \\ r_j/(r_j - r_d) & if \;\; r_d < r_j \le r_s \\ \infty & if \;\; r_d \le r_j \end{cases} \tag{4.6}$$

4.6 Control Inputs

Next, we define the control input for our algorithm. We first consider a single-integrator model (where input is given in terms of velocity) and then later expand it for double-integrator models (acceleration based).

For a single-integrator model each agent i's dynamics are given by $p_i[k+1] = p_i[k] + T_s u_i[k]$ where $p_i[k], u_i[k] \in \mathbb{R}^2$ denoting position and velocity respectively at time k with sampling period T_s. $u_i[k]$ is defined in terms of radial and angular inputs $(u_{i_r}[k], u_{i_\theta}[k])$ representing speed and direction respectively. We can calculate $r_j^i[k+1]$ and $\theta_j^i[k+1]$ for agent i using trigonometry to define a cost function.

$$r_j^i[k+1]^2 = r_j^i[k]^2 + T_s^2 u_{i_r}[k]^2 - 2T_s r_j^i[k] u_{i_r}[k]\cos\left(\theta_j^i[k] - u_{i_\theta}\right) \tag{4.7}$$

$$\theta_j^i[k+1] = \arccos\left(\frac{r_j^i[k]\cos\left(\theta_j^i[k]\right) - T_s u_{i_r}[k]\cos\left(u_{i_\theta}[k]\right)}{r_j^i[k+1]}\right) \tag{4.8}$$

Definition 4.4 (Cost function)**.** Given an agent with dynamics described by a single integrator, its position represented by $\psi_i(\theta)$, and its position objective by $\psi_i^{obj}(\theta)$, then we define its cost function at a time step k by

$$J_i[k] = \int_0^{2\pi} \left(\psi_i - \psi_i^{obj}\right)^2 d\theta + \gamma||u_i||^2 = D_{\psi_i}^{obj}[k] + \gamma u_{i_r}[k] \tag{4.9}$$

where $\gamma > 0$ controls the weight of the closed loop control.

From this this we propose a control law to minimize (equation (4.9))

$$u_{i_\theta}[k] = \widetilde{\theta} : \widetilde{\theta} = \underset{\widetilde{\theta}\in[0,2\pi)}{\operatorname{argmin}} \int_0^{2\pi} \left(\psi_i \left[\text{k+1} \mid u_{i_\theta}[k] = \widetilde{\theta}\right] - \psi_i^{obj}[k]\right)^2 d\theta \tag{4.10}$$

$$u_{i_r}[k] = \nu \frac{D_{\psi_i}^{obj}}{10\delta + D_{\psi_i}^{obj}} \tag{4.11}$$

where ψ is the maximum allowable error for convergence defined in definition 4.3. The factor 10 is to encourage faster convergence of $u_{i_r}^2$ with respect to the error term from equation (4.9).

Similarly for a double-integrator model each agent i's dynamics are given by $p_i[k+] = p_i[k]T_s q_i[k], q_i[k+1] = q_i[k] + T_s u_i[k]$ where $p_i[k]$,$q_i[k]$ and $u_i[k]$ denote position, velocity and acceleration respectively. Our control input is now $u_i[k]$ or acceleration with components u_{i_r} and u_{i_θ} representing radial and angular acceleration. We must modify the cost function (equation (4.9)) to account for velocity into $J_i[k] = D_{\psi_i}^{obj}[k] + \gamma_r u_{i_r}[k] + \gamma_\theta \left|u_{i_\theta}[k]\right|$. Now we can adjust our desired angular speed from (equation (4.10)) into

$$q_{i_\theta}[k] = \widetilde{\theta} : \widetilde{\theta} = \underset{\widetilde{\theta}\in\left[q_{i_\theta} - u_{i_\theta^{max}}, q_{i_\theta} + u_{i_\theta^{max}}\right)}{\operatorname{argmin}} \int_0^{2\pi} \left(\psi_i \left[k+1 \mid u_{i_\theta}[k] = \widetilde{\theta}\right] - \psi_i^{obj}[k]\right)^2 d\theta \tag{4.12}$$

where we change the minimization interval to account for our maximum radial acceleration. Our control inputs then simply become $u_{i_\theta} = q_{i_\theta}[k] - q_{i_\theta}[k-1]$ and $u_{i_r} = q_{i_r}[k] - q_{i_r}[k-1]$.

5 2D Minimal Communication Formation Control

Portions of text and figures reproduced from author's previous work [32].

5.1 Overview

The algorithm presented in this chapter uses minimal communication between agents. It is a progressive algorithm whereby agents start as anonymous or index-free and self assign an index. It requires agents to share a global orientation frame. Broadly speaking, the algorithm is divided into two phases. The first phase is position assignment. Each agent self assigns a position in turn in a deterministic manner based on a locally convex spiral. The second phase involves each agent following its parent, the previous agent on the spiral, while maintaining a displacement to achieve the final formation.

5.2 Motivation

This algorithm was partially inspired by Pinciroli's progressive formation algorithm [11]. Compared to their work, our algorithm requires only one-way communication and no negotiation between agents, allowing for lower latency during the assignment phase. Our algorithm introduces a constraint on the available objective formation configurations based on the sensing characteristics of each agent. Additionally, we are able to easily extend our algorithm into three dimensions as explained in chapter 6.

5.3 Formulation

In this chapter the following notation is used. $[N] = \{k \in \mathbb{Z}^+ : k \leq N\}$ to denote the set of the first N positive integers. Given a set of vectors $x_1, \ldots, x_N \in \mathbb{R}^n$, $\mathbf{x} = [x_1^T, \ldots, x_N^T] \in \mathbb{R}^{nN}$ denotes the stacking of vectors. $Conv(\mathbf{q})$ denotes the convex hull of $\mathbf{q}$ and by $\delta Conv(\mathbf{q})$ its boundary, which is a convex polygon.

We define the formation as a set of points $\mathbf{q} = [q_1, \ldots, q_N] \in \mathbb{R}^{2N}$. Given a set of N agents with positions $\mathbf{p}(t) = [p_1(t), \ldots, p_N(t)] \in \mathbb{R}^{2N}$, our goal is to achieve a spatial distribution equivalent to $\mathbf{q}$ under translation.

Definition 5.1 (Formation Objective)**.** Given an objective point set $\mathbf{q}$ and a set of agents represented by their positions $\mathbf{p}(t)$, we consider that the formation has been achieved at a time $t = t'$ if a permutation $\sigma : [N] \to [N]$ exists such that

$$\|p_i(t') - p_0(t') + q_{\sigma(i)} - q_{\sigma(0)}\| < \varepsilon \tag{5.1a}$$

$$\|\dot{p}_i(t')\| < \delta \tag{5.1b}$$

for constants $\varepsilon, \delta > 0$ that represent the maximum error allowed for positions and speed. We assume that agents are able to measure the position of any other agent in line-of-sight up to a maximum sensing distance δ_s.

Thus we have two disjoint problems to solve. First, find a permutation $\sigma : [N] \to [N]$ such that equation (5.1) can be fulfilled. Second, define a control law such that we can converge to the desired formation within the limit set by ε with $t = t' < \infty$. We propose a progressive assignment algorithm that enables agents to automatically assign themselves a unique position in the objective formation. Additionally, a fairly standard control law derived from previous work on leader-follower formation control algorithms with collision avoidance is proposed.

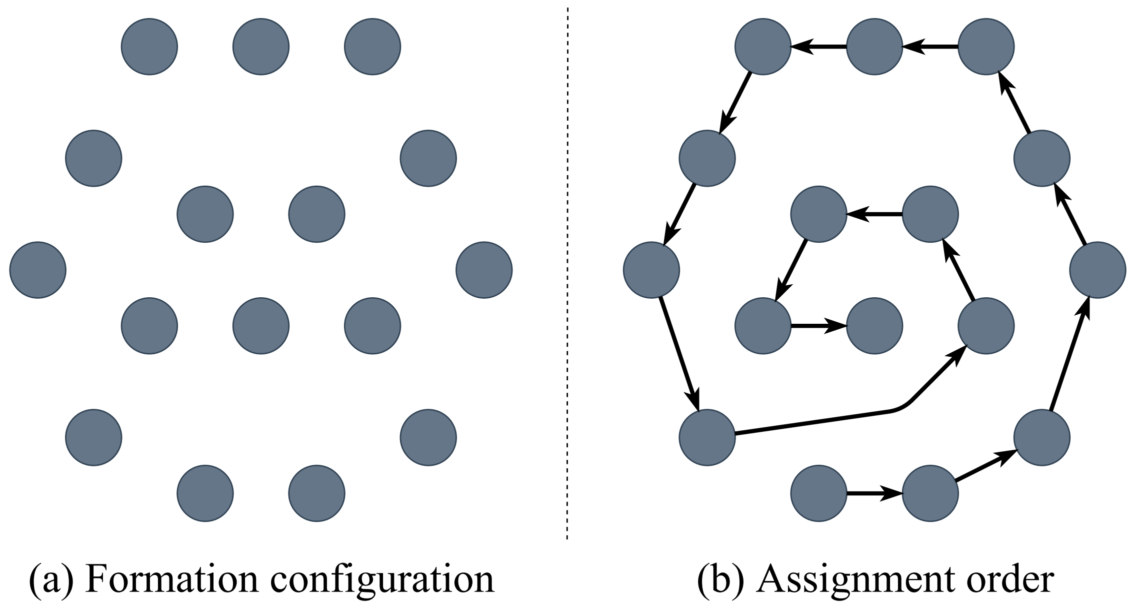


Figure 5.1: Illustration of the position assignment via a locally convex path.

5.4 Locally Convex Directed Path Graph

A locally convex directed path graph, or convex spiral, can be defined for an objective formation configuration $\mathbf{q}$ that guarantees a unique assignment of identifiers. We call a directed path graph locally convex if each point in the path, excepting the first and last, form a convex angle with its neighbors. Figure 5.1 shows an example of a convex spiral generated from a set of points. The convex spiral is generated using a trivial modification to the Jarvis march method of computing the convex hull in the plane [33].

First we chose a node as the graph root called p_{root}. Any node that lies on $\delta Conv(\mathbf{q})$ is suitable, for instance the leftmost node. We define two sets $P = \{p_{root}\}$ and $Q = \mathbf{q} \setminus P$ where P denotes nodes that are already part of our convex spiral and Q denotes nodes still to be considered. $p_{last} \in P$ denotes the last node added to P and is initially p_{root}.

Then while $|R| > 0$ select a node $r_i \in R$ such that $\forall r_k \neq r_i \in R : r_k$ lies to the right of the line between p_{last} and r_i. We then move r_i from R to P and p_{last} becomes r_i. That is $R = R \cap \{r_i\}, P = P \cup \{r_i\}, p_{last} = r_i$.

5.5 Position Assignment

Next, we describe an algorithm for position assignment requiring only one-way communication between agents. Suppose we have a set of N agents where $\mathbf{p}(t) = [p_1^T(t), \ldots, p_N^T(t)] \in \mathbb{R}^{2N}$ represents position and δ_s represents an agents maximum sensing range. The objective formation is defined as $\mathbf{q} = [q_1^T, \ldots, q_N^T] \in \mathbb{R}^{2N}$. We assume the following condition holds:

Assumption 5.1 (Distances in convex spiral)**.** Let $\mathbf{p}(0)$ represent an initial distribution of agents. Without any loss of generality, we assume the boundary of its convex hull is the set $\delta Conv(\mathbf{p}(0)) = \{p_i(0), \ldots, p_{h_0}(0)\}$ where h_0 represents the number of agents in $\delta Conv(\mathbf{p}(0))$. We assume that any agent in the convex hull boundary is able to sense its two immediate neighbors, i.e., $\|q_i - q_{i+1 \mod h_0}\| < \delta_s \; \forall i \leq h_0$, where δ_s is a lower limit of the agents' sensing range.

Each agent maintains a state that is either *unassigned* or the identifier of the position they have been assigned to. All agents are initially *unassigned.* The agents periodically broadcast their state. Situated communication can be used to locate the position of other agents based on these broadcasts. Next, an agent must assign itself as root. This agent must lie on $\delta Conv(\mathbf{p}(0))$. Since we assume a global reference frame, we can define the root as the leftmost agent and then the agents must simply check to see if there are any agents to their left or $p_{root} = \operatorname{argmin} p_{i_x}, p_i \in \mathbf{p}(0)$. Once the root node has assigned itself, it starts broadcasting its identifier. We still need one more assumption in order to guarantee each node on the convex spiral can self-assign.

Assumption 5.2 (Existence of a locally convex path)**.** Let $\mathbf{p}(0)$ be the position of N agents after deployment. Given an agent in $\delta Conv(\mathbf{p}(0))$ identified as the root of the directed path graph, and an assignment direction that uniquely identifies the second node in the graph, then these two agents define a unique locally convex directed path graph in which all agents are included exactly once. We assume that any two consecutive agents in the path are able to sense each other.

Assuming Assumption 2 holds, then all agents are able to self-assign a unique position in the formation.

5.6 Collision Avoidance and Control Inputs

Given that the assignment process essentially creates a chain of leader-follower pairs, we can easily adapt established leader-follower control laws from the literature. The downside to this is that small errors in the chain will compound. Additionally, convergence for agents near the end of the chain takes significantly longer than agents near the beginning of the chain. Another issue that arises is that we assume each agent except for the root can sense its leader agent. If using line-of-sight based sensing then it is possible for other agents to break line-of-sight while moving. Various solutions have been proposed, such as using a non line-of-sight based sensing method or reducing the speed of the leader as the follower approaches the sensing range limit or when line-of-sight is lost.

A potential is used for collision avoidance [34]:

$$V_{ij}(t) = \begin{cases} \left(\min\left\{0, \dfrac{\|p_j^i(t)\|^2 - R^2}{\|p_j^i(t)\|^2 - r^2}\right\}\right)^2 & \|p_j^i(t)\| > r \\ \infty & \|p_j^i(t)\| < r \end{cases} \tag{5.2}$$

where R, r represent the *warning* and *danger* distance, respectively. These constants are defined in a way such that an agent actively tries to avoid another agent when the distance that separates them is smaller than the warning distance, and it must never be below or equal to the danger distance.

6 3D Minimal Communication Formation Control

Portions of text and figures reproduced from author's previous work [32].

6.1 Overview

The extension of the previous algorithm into three dimensions relies on finding a mapping between the convex spiral in two dimensions and some ordering or path through the objective formation in three dimensions. However, given the increase in complexity in the possible shapes in three dimensions, we have not attempted to prove that such a path exist.

6.2 Formulation

The following section is built on top of the ideas described in chapter 5. Specifically, the position assignment process is identical. A few changes must be made to extend the formation into three dimensions. Objective formation **q** must be redefined in terms of $\mathbb{R}^{3N}$. We assume the agents are initially distributed in a plane, or, that when projected into a plane, no agents occupy the same position. This allows us to use the position assignment process without any significant changes.

Assumption 6.1 (Ordering)**.** The position assignment process requires an ordering of the objective formation but the concept of a convex spiral does not easily extend into three dimensions we must choose another, independent, method to specify an ordering. Suppose we define σ as permutation of **q** such that $\sigma = (\sigma(q_1), \sigma(q_2), \ldots \sigma(q_n))$. We define $LOS(p_1, p_2)$ as true if, and only if, there is a clear line-of-sight between p_1 and p_2. We assume σ is a valid ordering if it satisfies

$$\|q_{\sigma_i} - q_{\sigma_{i+1}}\| < \delta_s, \tag{6.1a}$$

$$LOS(q_{\sigma_i}, q_{\sigma_{i+1}}) \tag{6.1b}$$

where $\delta_s > 0$ represents the maximum sensing distance of an agent.

Position assignment then follows section 5.5. The same collision avoidance and control inputs can also be used since they work irrespective of the dimension of the input vectors. We do not provide a methodology for finding a valid ordering in the general case. In the case that the objective formation consists of a single three-dimensional convex hull then if the triangulation of the hull boundary contains a Hamiltonian path, it is a sufficient ordering. All platonic solids contain a Hamiltonian cycle, which implies the existence of a Hamiltonian path [35]. Thus all graphs that are isomorphic to Platonic solids contain a Hamiltonian cycle, however not all such graphs necessarily fulfill the sensing requirements of this algorithm.

7 Implementation and Results

Sections 7.1 to 7.3 includes figures and some text from author's previous work [31], [32].

7.1 Communication-free Formation Control

7.1.1 Implementation

In order to implement the algorithm, we needed to build several agents capable of acting autonomously. They needed to be able to sense the other agents in the environment. To this end, we built several autonomous vehicles based on *1:10 Elektro-Monstertruck "NEW1" BL*, an example of such is shown in figure 7.1a. We replaced the radio control receiver with an Arduino and Raspberry Pi connected to each other via serial. The car is controlled via two servo motors, one for turning and one for speed. Turning is limited to $\pm 0.35 rad$ or $\approx \pm 20 deg$. The Arduino takes care of generating the control signals for the servos and receives instructions from the Raspberry Pi via serial. The Raspberry Pi runs the actual algorithm. Each iteration of the algorithm is somewhat quick but there is a bit of overhead from reading and processing the lidar data as well as a web server we run for monitoring and visualization. Because of this, we run the algorithm in steps rather than real-time and then move the car some distance after each step proportional to the velocity control input.

To sense the other agents we included a *RPLiDAR A1M8* lidar component mounted on top of each truck. This lidar has a range of 12m and a 360-degree view of the en-

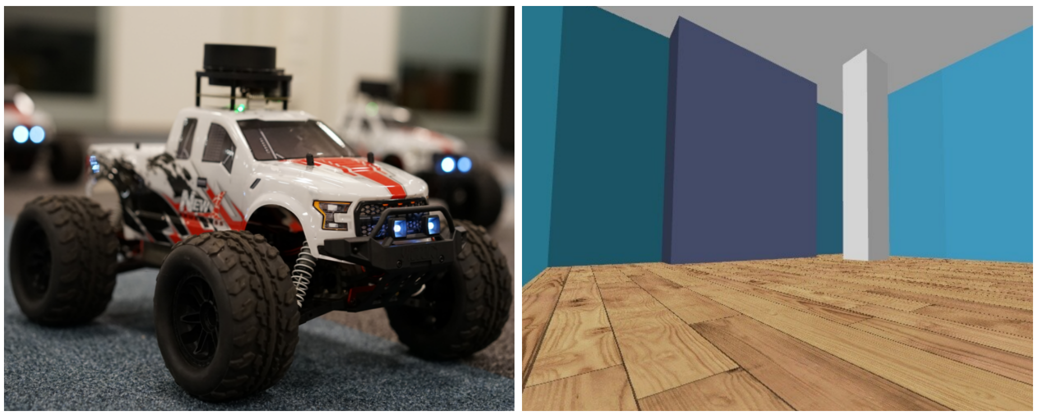

(a) Example agent (b) Lab environment

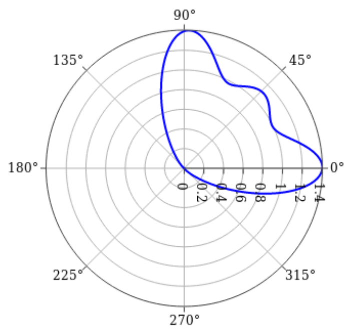
Figure 7.1: (a) Autonomous agent used during experiment and (b) Illustration of lab environment where the white column is used for orienting reference

vironment in a plane. Since we require a global orientation we use knowledge of the environment's geometry rather than using a compass or magnetometer for orientation. The testing environment is a typical office room with furniture shown in figure 7.1b. It features a column and we assigned a north and east direction to the walls adjacent to the column. This was sufficient for our purposes, though a more robust orienting method would be recommended for field use.

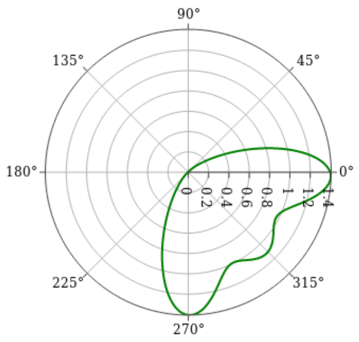
Owing to the kinematic properties of the agents we constructed and the relatively small size of the testing environment, we opted for a relatively simple objective formation with a rather large allowed error. We use 4 agents with a square pattern as the objective formation.

7.1.2 Results

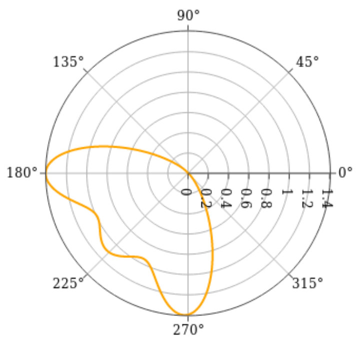
The results of the experiment are presented in figure 7.2. The polar indicator distributions for 3 of the positions in the objective formation are presented in figures 7.2a to 7.2c. A rough convergence was achieved in just 5 iterations. While this test has few agents and a relatively simple formation definition, it shows that convergence is possible in realistic



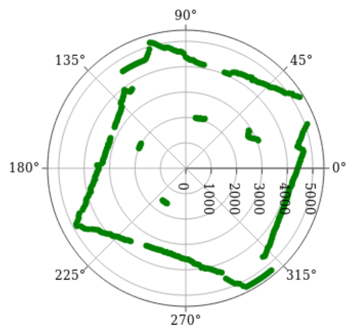
(a) Position 0



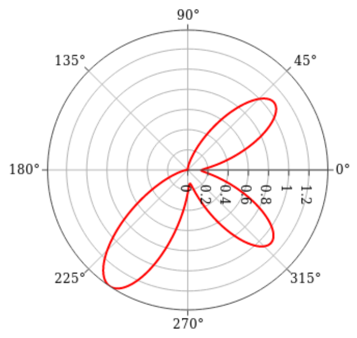
(b) Position 2



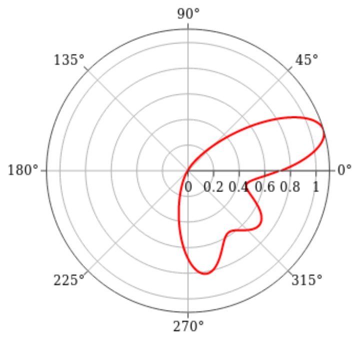
(c) Position 3



(d) Raw lidar output



(e) Initial vision



(f) Final vision

Figure 7.2: (a-c) Desired vision for bottom left, top left, and top right agents respectively (d) Raw output of the lidar. (e) Initial view from an agent. (f) Final view from an agent after convergence with respect to δ

scenarios. More complex formations have been achieved in simulation.

7.2 2D Minimal Communication Formation Control

7.2.1 Implementation

We performed two simulations to test the formation control algorithm. In the first one, we used a single-integrator simulator written in python. For the second we decided to focus on having a more realistic simulation using Robot Operating System (ROS) with the Gazebo simulator and RotorS for simulating the flight controller.

Figure 7.3 show the simulation environment for the Gazebo simulation. It consists of the ROS master process, the Gazebo simulator, a formation broadcaster and a set of processes for each drone in the simulation. Each drone runs a PX4 software in the loop simulator (SITL), MAVROS for communicating with the PX4 and the formation control process. The formation control process consists of two threads, one for communicating with the PX4 through MAVROS and the other for high level movement planning. PX4 is an opens source flight controller [12]. It can be used for simulation in Gazebo on top of RotorS (a drone simulator plugin for Gazebo) [30]. In realistic scenarios, it runs on separate dedicated hardware. The formation broadcaster broadcasts the desired formation to each drone through ROS. It is responsible for ordering the objective configuration which may be computationally expensive. In a real implementation, the formation definition could be embedded if the desired formation is static, or broadcast using other means. A few quirks of the implementation are discussed in section 7.3.

7.2.2 Results

We have compared the performance of the single integrator simulation and results obtained with ROS/Gazebo in terms of convergence time and agent paths. In order to be

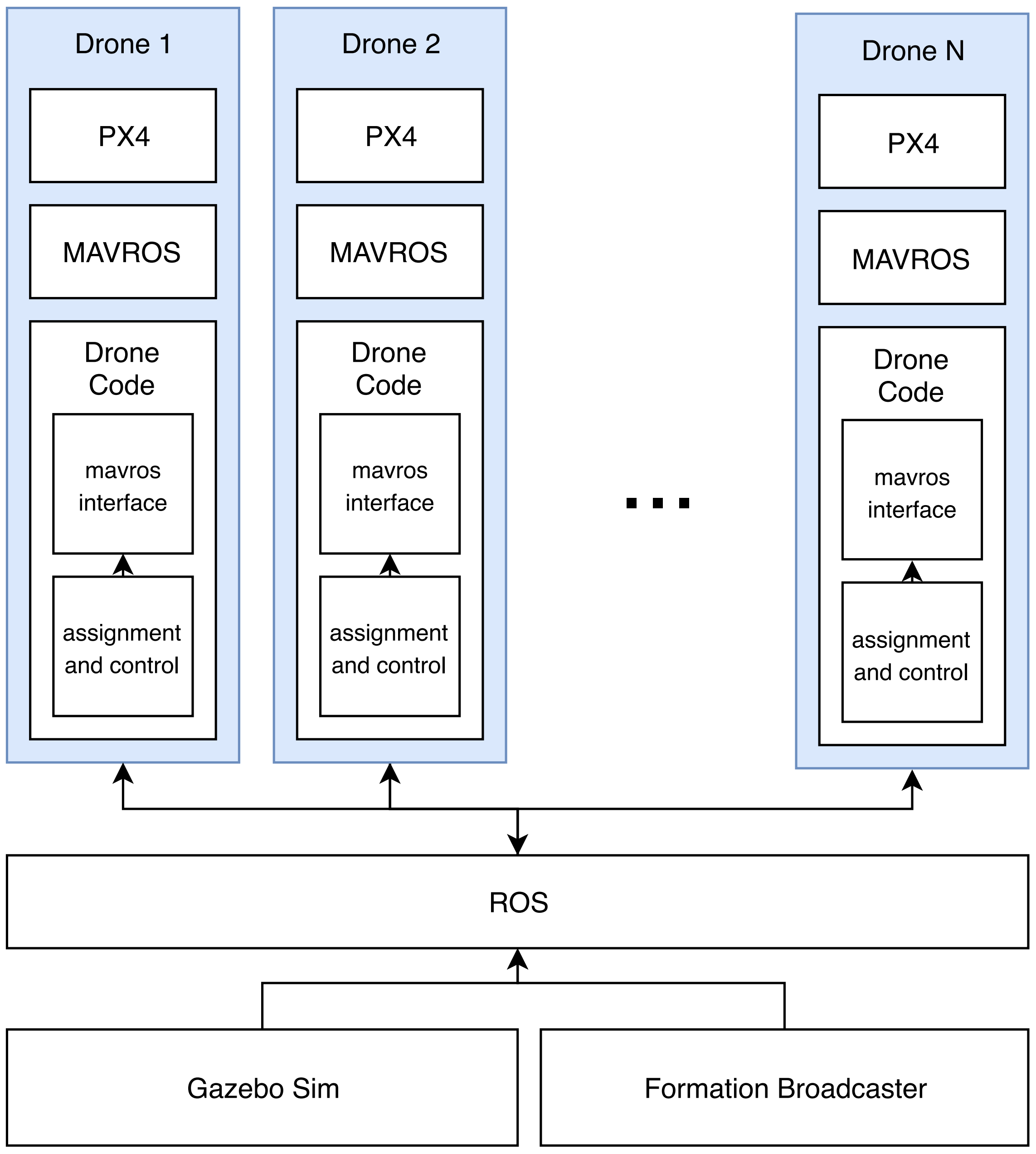


Figure 7.3: ROS/Gazebo implementation architecture.

able to compare the single integrator implementation in Python with the ROS implementation, we have adjusted the time steps and speeds to match both simulations.

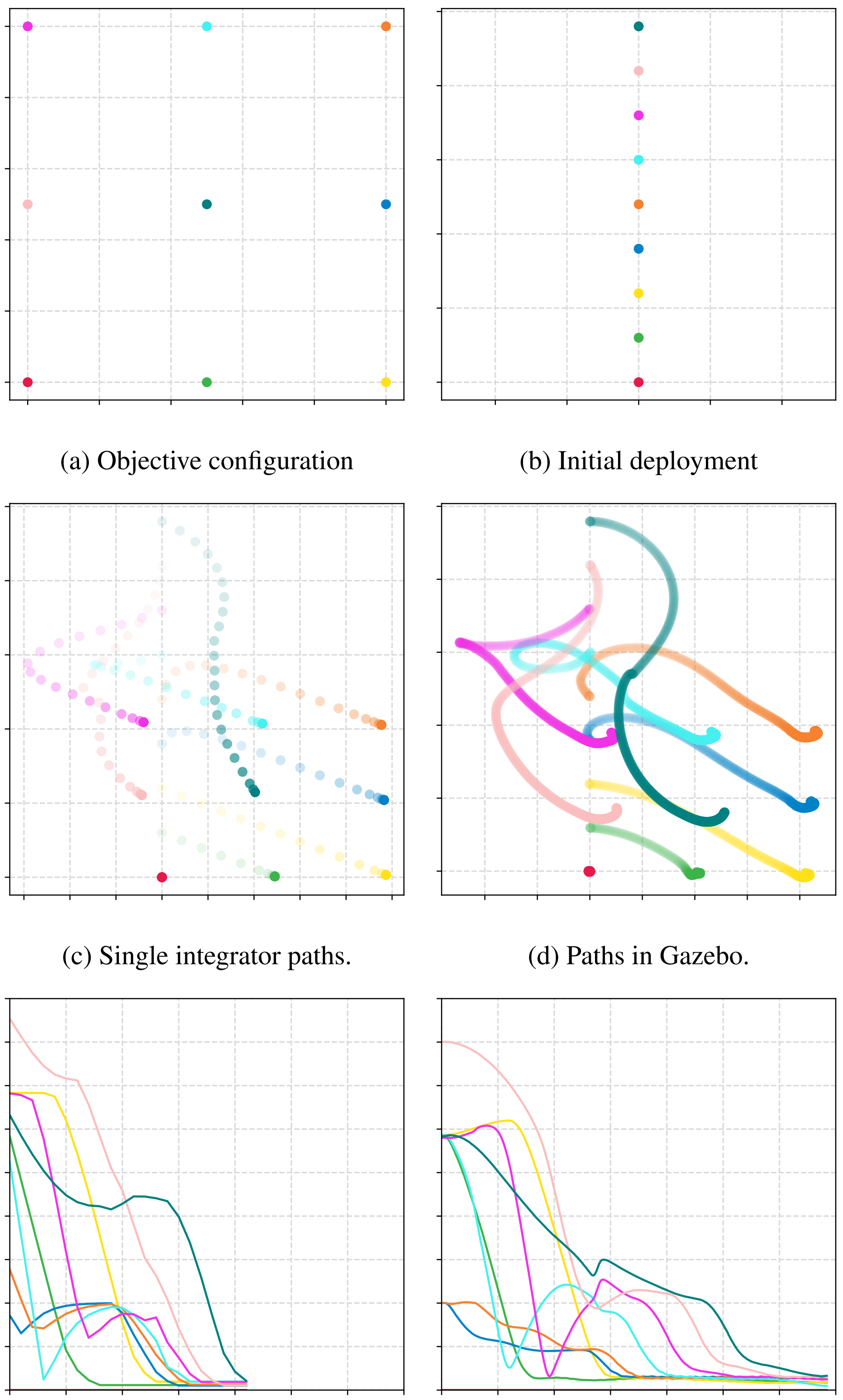
Figures 7.4a and 7.4b show the objective configuration and initial distribution of the agents. We chose a line along the y-axis as the initial pattern and a 3x3 grid for the objective configuration. Rather than a random initial distribution, we have chosen a line in order to show the efficacy of our method in a disadvantageous initial distribution. The paths taken by both the single integrator simulation and the Gazebo drone simulation are similar, with moments where collisions were being avoided being more clear in the Gazebo simulation. The single integrator converges much faster, owing to the simpler model used. This is shown in figures 7.4c and 7.4d, where instantaneous errors of individual agents are illustrated. These errors are calculated as the norm of the difference of the current leader-follower displacement and the objective one.

Drones have an approximate size of $55\ cm$ by $55\ cm$, and the size of the area displayed in figure 7.4a is $9\ m \times 9\ m$ and 7.4b is $4\ m \times 25\ m$. For figures 7.4c and 7.4d the size of the area displayed is $20\ m \times 30\ m$. Although, our control input to the drones is velocity which the drone is not capable of instantaneously achieving. The maximum speed was $1\ m/s$. The Gazebo simulation took 35 seconds to converge, which could be improved with better control. The positions of the drones were sampled at 1Hz for the single integrator simulation and 10Hz for the Gazebo simulation. Figure 7.5 shows the final results in Gazebo. For the collision avoidance potential, we have utilized $r = 1.5\ m$ and $R = 0.75\ m$.

7.3 3D Minimal Communication Formation Control

7.3.1 Implementation

Since this is an extension of section 7.2 the implementation remains largely identical with a few modifications to support specifying the objective formation in three dimensions. We



(a) Objective configuration

(b) Initial deployment

(c) Single integrator paths.

(d) Paths in Gazebo.

(e) Single integrator errors.

(f) Errors in Gazebo.

Figure 7.4: Simulation of a 3x3 grid formation in both

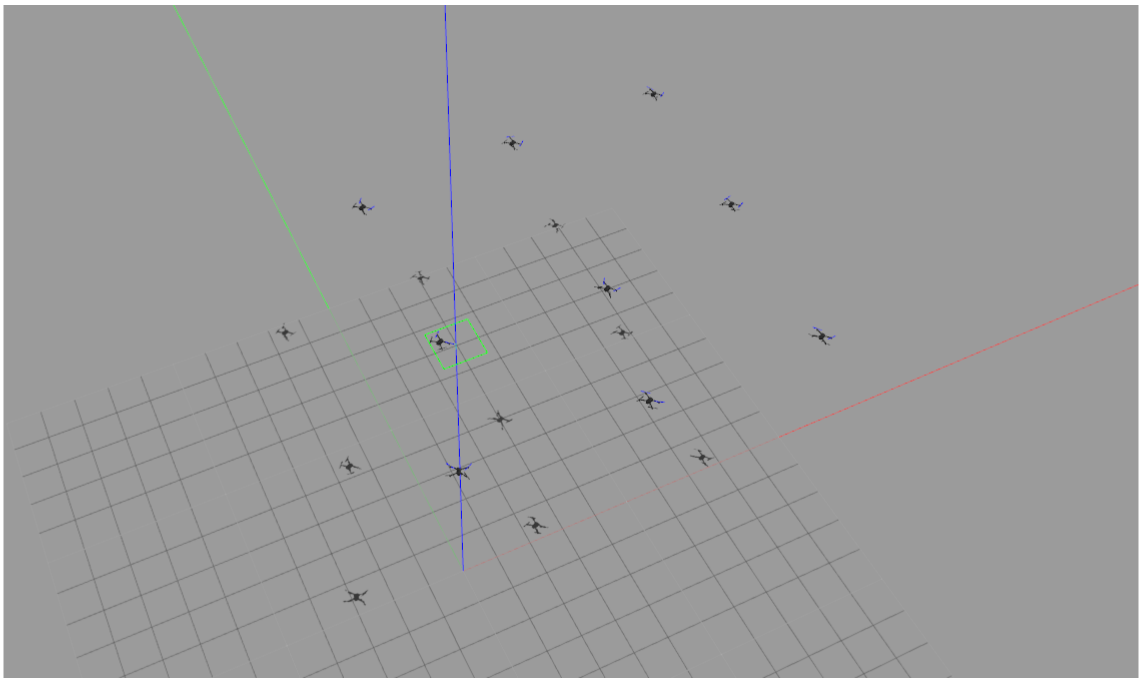


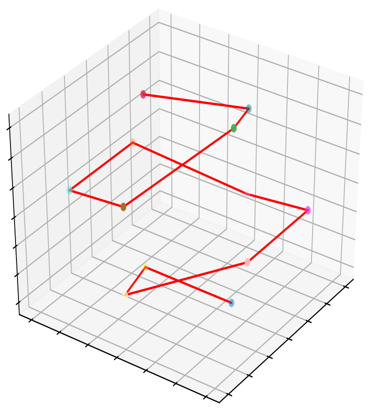
Figure 7.5: Final 2D configuration in Gazebo

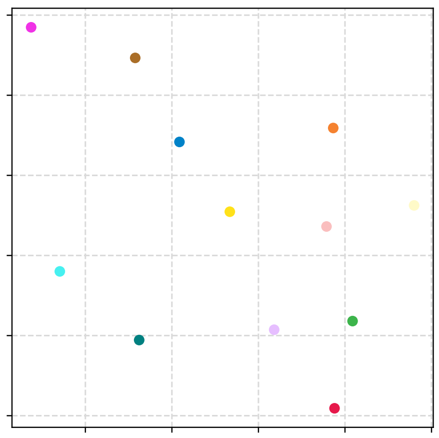
use the same process architecture specified in figure 7.3. In reality the implementations are identical except for the Formation Broadcaster component. In the 2d case it broadcasts the formation with a static z coordinate assigned for all positions. This means that due to rounding errors and the properties of the collision avoidance potential, it would be possible for two agents to pass over each other using the z axis. However this did not happen in any of the simulations we ran.

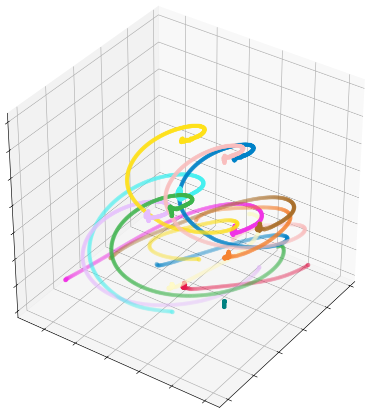
7.3.2 Results

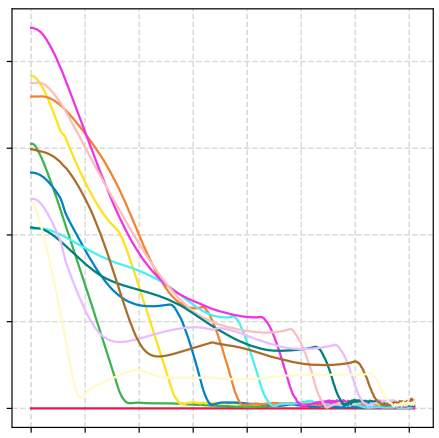
To demonstrate the validity of our algorithm for deploying drones into three-dimensional configurations, we chose a pseudo-icosahedron as our objective configuration, which is illustrated in figure 7.6a. Our pseudo-icosahedron is generated using vertices at $(0, \pm 1, \pm 2)$ rather than $(0, \pm 1, \pm \phi)$ where ϕ is the golden ratio. The pseudo-icosahedron is scaled by a factor of $20m$ so that the objective positions are farther than our collision avoidance parameter r. The ordered path was generated manually. Additionally since the pseudo-icosahedron is an irregular platonic solid it demonstrates the flexibility of our algorithm. The initial configuration was random as seen in figure 7.6b.

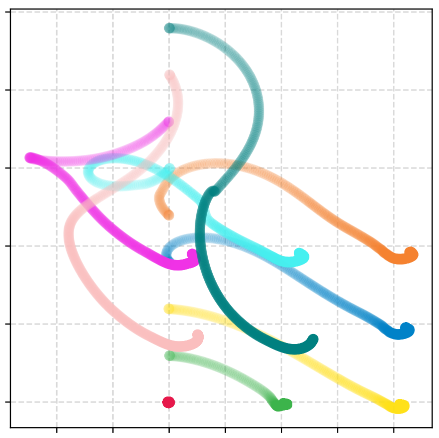
It took roughly 70 seconds to converge, with the same agent speed and collision avoidance parameters used in the two-dimensional case. The 2D case converged in roughly 35 seconds but overall had shorter distance to travel and fewer agents so comparison is not very useful. As mentioned in the 2D implementation, a better control algorithm could reduce the time to convergence. Figure 7.7 shows a side view of the final configuration in the Gazebo simulator. We used tho same parameters for the collision avoidance potential as with the 2D case, $r = 1.5\ m$ and $R = 0.75\ m$.


(a) Objective configuration


(b) Initial deployment


(c) Paths for 3D.


(d) Errors for 3D.


(e) Paths for 2D.

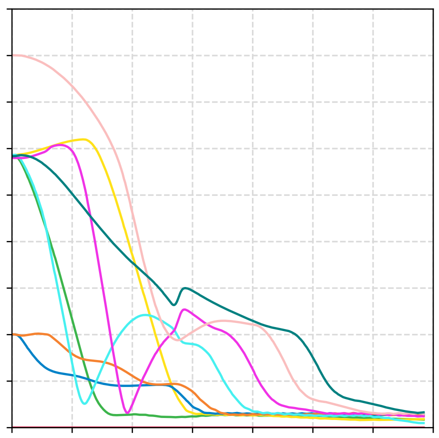

(f) Errors for 2D.

Figure 7.6: Simulation of an pseudo-icosahedron formation in both

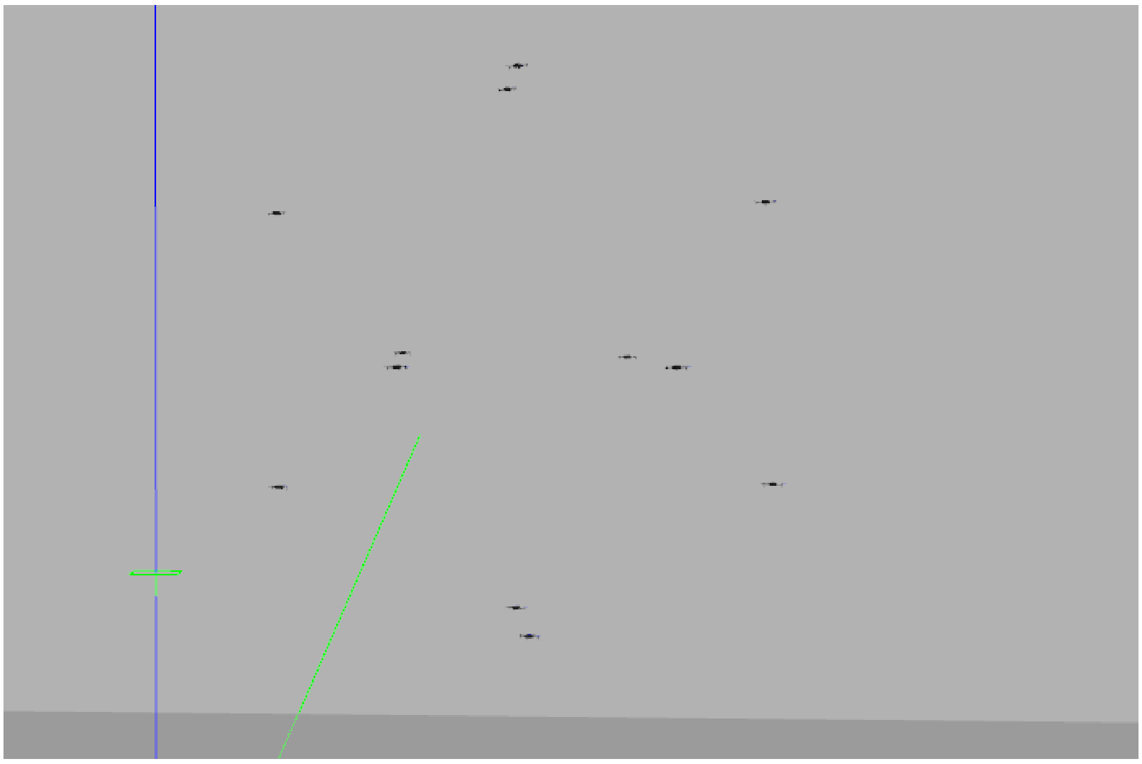


Figure 7.7: Final 3D configuration in Gazebo

8 Conclusion

Our goal was explore formation control algorithms that allow for arbitrary patterns to be achieved and operate with limited or no communication between anonymous agents. To this end, we introduced two algorithms plus a modification showing that arbitrary formations can be achieved under the given constraints. Future work could focus on improving the drawbacks of the presented approaches, including focusing on optimizing convergence exploring alternative structures for use during assignment for the limited communication algorithm and refining the 3d extension of the limited communication algorithm.

References

[1] Z. Yang, Q. Zhang, and Z. Chen, "Formation control of multi-agent systems with region constraint", *Complexity*, vol. 2019, p. 8 481 060, Dec. 2019, ISSN: 1076-2787. DOI: 10.1155/2019/8481060.

[2] K. K. Oh, M. C. Park, and H. S. Ahn, "A survey of multi-agent formation control", *Automatica*, vol. 53, pp. 424–440, Mar. 2015, ISSN: 00051098. DOI: 10.1016/j.automatica.2014.10.022.

[3] C. K. Verginis, A. Nikou, and D. V. Dimarogonas, "Position and orientation based formation control of multiple rigid bodies with collision avoidance and connectivity maintenance", in *2017 IEEE 56th Annual Conference on Decision and Control, CDC 2017*, vol. 2018-Janua, 2018, pp. 411–416, ISBN: 9781509028733. DOI: 10.1109/CDC.2017.8263699. arXiv: 1703.08217v3.

[4] K. Støy, "Using situated communication in distributed autonomous mobile robotics", in *Proceedings of the Seventh Scandinavian Conference on Artificial Intelligence*, ser. SCAI '01, IOS Press, 2001, pp. 44–52, ISBN: 1-58603-161-9.

[5] M. C. Park, Z. Sun, M. H. Trinh, B. D. O. Anderson, and H. S. Ahn, "Distance-based control of k4 formation with almost global convergence", in *2016 IEEE 55th Conference on Decision and Control (CDC)*, Dec. 2016, pp. 904–909.

[6] S. A. Barogh and H. Werner, "Cascaded formation control using angle and distance between agents with orientation control (part 1 and part 2)", in *2016 UKACC 11th International Conference on Control (CONTROL)*, Aug. 2016, pp. 1–6.

[7] N. P. Hyun *et al.*, “Collision free and permutation invariant formation control using the root locus principle”, in *2016 ACC*, Jul. 2016.

[8] S. Kloder *et al.*, “A configuration space for permutation-invariant multi-robot formations”, in *2004 ICRA*, vol. 3, Apr. 2004.

[9] M. M. Zavlanos *et al.*, “Distributed formation control with permutation symmetries”, in *2007 46th IEEE CDC*, Dec. 2007.

[10] P. Kingston and M. Egerstedt, “Index-free multi-agent systems: An eulerian approach”, *IFAC Proceedings Volumes*, vol. 43, no. 19, pp. 215–220, 2010, 2nd IFAC Workshop on Distributed Estimation and Control in Networked Systems, ISSN: 1474-6670. DOI: `https://doi.org/10.3182/20100913-2-FR-4014.00064`.

[11] C. Pinciroli *et al.*, “Decentralized progressive shape formation with robot swarms”, in *Distributed Autonomous Robotic Systems: The 13th International Symposium*, Springer, 2018, pp. 433–445.

[12] L. Meier, D. Honegger, and M. Pollefeys, “PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms”, in *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2015-June, IEEE, May 2015, pp. 6235–6240, ISBN: 978-1-4799-6923-4. DOI: `10.1109/ICRA.2015.7140074`.

[13] F. Zafari, A. Gkelias, and K. K. Leung, “A survey of indoor localization systems and technologies”, *IEEE Communications Surveys Tutorials*, vol. 21, no. 3, pp. 2568–2599, 2019.

[14] P. Zhang, J. Lu, Y. Wang, and Q. Wang, “Cooperative localization in 5g networks: A survey”, *ICT Express*, vol. 3, no. 1, pp. 27–32, 2017, ISSN: 2405-9595. DOI: `https://doi.org/10.1016/j.icte.2017.03.005`.

[15] E. von Puttkamer, G. Weiss, and T. Edlinger, “Localization and On-Line Map Building for an Autonomous Mobile Robot”, in, Springer, Berlin, Heidelberg, 1999, pp. 36–48. DOI: `10.1007/10705474_3`.

[16] E. Hu, Z. Deng, Q. Xu, L. Yin, and W. Liu, *Relative entropy-based Kalman filter for seamless indoor/outdoor multi-source fusion positioning with INS/TC-OFDM/GNSS*, Jan. 2018. DOI: `10.1007/s10586-018-1803-1`.

[17] D. Nada, M. Bousbia-Salah, and M. Bettayeb, *Multi-sensor data fusion for wheelchair position estimation with unscented Kalman filter*, Apr. 2017. DOI: `10.1007/s11633-017-1065-z`.

[18] T. Koshizen, “Improved sensor selection technique by integrating sensor fusion in robot position estimation”, *Journal of Intelligent and Robotic Systems: Theory and Applications*, vol. 29, no. 1, pp. 79–92, 2000, ISSN: 09210296. DOI: `10.1023/A:1008123508778`.

[19] J. Breßler, P. Reisdorf, M. Obst, and G. Wanielik, “Gnss positioning in non-line-of-sight context—a survey”, in *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, 2016, pp. 1147–1154.

[20] S. A. S. Mohamed, M. Haghbayan, T. Westerlund, J. Heikkonen, H. Tenhunen, and J. Plosila, “A survey on odometry for autonomous navigation systems”, *IEEE Access*, vol. 7, pp. 97 466–97 486, 2019.

[21] Z. Zou, Z. Shi, Y. Guo, and J. Ye, *Object detection in 20 years: A survey*, 2019. arXiv: `1905.05055 [cs.CV]`.

[22] L. Pan, J. Cheng, W. Feng, and X. Ji, “A robust RGB-D image-based SLAM system”, in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10528 LNCS, Springer, Cham, Jul. 2017, pp. 120–130, ISBN: 9783319683447. DOI: `10.1007/978-3-319-68345-4_11`.

[23] Z. Wang, S. Huang, and G. Dissanayake, “D-SLAM: Decoupled localization and mapping for autonomous robots”, *International Symposium of Robotics Research ISRR 05*, vol. 26, no. 2, pp. 203–213, 2005, ISSN: 16107438. DOI: 10.1.1.88.5371.

[24] S. Jung, J. Kim, and S. Kim, “Simultaneous localization and mapping of a wheel-based autonomous vehicle with ultrasonic sensors”, *Artificial Life and Robotics*, vol. 14, no. 2, pp. 186–190, Nov. 2009, ISSN: 14335298. DOI: 10.1007/s10015-009-0650-9.

[25] Y. Zhuang, M. W. Gu, W. Wang, and H. Y. Yu, “Multi-robot cooperative localization based on autonomous motion state estimation and laser data interaction”, *Science China Information Sciences*, vol. 53, no. 11, pp. 2240–2250, Nov. 2010, ISSN: 1674733X. DOI: 10.1007/s11432-010-4096-4.

[26] M. Alpen, K. Frick, and J. Horn, “A real-time on-board orthogonal SLAM for an indoor UAV”, in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7508 LNAI, Springer, Berlin, Heidelberg, 2012, pp. 542–551, ISBN: 9783642335020. DOI: 10.1007/978-3-642-33503-7_53.

[27] 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”, *ICRA workshop on open source system*, 2009.

[28] KenConley. (2009). “attatchment:ROS_basic_concepts.dia of ROS/Concepts - ROS Wiki”. Creative Commons Attribution 3.0 http://creativecommons.org/licenses/by/3.0/ exported to pdf.

[29] 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) (IEEE Cat. No.04CH37566), vol. 3, IEEE, 2005, pp. 2149–2154, ISBN: 0-7803-8463-6. DOI: `10.1109/iros.2004.1389727`.

[30] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart, "RotorS—A modular gazebo MAV simulator framework", *Studies in Computational Intelligence*, vol. 625, pp. 595–625, Feb. 2016, ISSN: 1860949X. DOI: `10.1007/978-3-319-26054-9_23`.

[31] J. Peña Queralta, C. McCord, T. N. Gia, H. Tenhunen, and T. Westerlund, "Communication-free and Index-free Distributed Formation Control Algorithm for Multi-robot Systems", *Procedia Computer Science*, vol. 151, pp. 431–438, 2019, ISSN: 18770509. DOI: `10.1016/j.procs.2019.04.059`.

[32] C. McCord, J. Peña Queralta, T. Nguyen gia, and T. Westerlund, "Distributed progressive formation control for multi-agent systems: 2d and 3d deployment of uavs in ros/gazebo with rotors", Sep. 2019.

[33] R. A. Jarvis, "On the identification of the convex hull of a finite set of points in the plane", *Inf. Process. Lett.*, vol. 2, pp. 18–21, 1973.

[34] S. Ahmadi Barogh *et al.*, "Formation control of non-holonomic agents with collision avoidance", in *American Control Conference (ACC)*, 2015. DOI: `10.1109/ACC.2015.7170825`.

[35] M. Gardner, *Mathematical Games: About the Remarkable Similarity between the Icosian Game and the Towers of Hanoi*. May 1957, vol. 196, pp. 150–156.