UNIVERSITÀ DI BOLOGNA

School of Engineering

Master Degree in Automation Engineering

Distributed Autonomous Systems 2024-2025

Course Project

Professors:
Giuseppe Notarstefano
Ivano Notarnicola

Students:
Francesca Bocconcelli
Luca Fantini

Academic year 2024/2025

Abstract

In recent years, distributed optimization has emerged as a powerful framework to coordinate large-scale systems of autonomous agents without relying on centralized control. This report explores how such techniques can be applied to multi-robot systems, focusing on both consensus and aggregative optimization problems. Our investigation begins with the implementation of a Distributed Gradient Tracking algorithm, which allows a team of robots to cooperatively estimate shared variables by relying only on local information and neighbor-to-neighbor communication. Through simulations, we observe accurate convergence to the global optimum, even in the presence of noisy measurements, demonstrating the effectiveness of distributed estimation strategies.

We then shift our attention to a more complex form of coordination, where each robot aims to balance its own objectives with the collective behavior of the team. This leads us to consider Aggregative Optimization problems, in which each local cost depends not only on the robot's individual decision but also on an aggregate quantity, such as the average position of all agents. To address this in a decentralized setting, we design a distributed algorithm based on a two-time-scale structure: while the decision variables evolve slowly through gradient steps, each agent concurrently tracks the aggregate quantities and their gradients via fast dynamic consensus schemes. This tracking mechanism enables each agent to locally approximate global information that would otherwise be inaccessible, effectively bridging the gap between centralized and distributed implementations. The proposed method proves robust and scalable, maintaining performance across different communication topologies and behavioral scenarios.

To ensure safe navigation, we integrate a low-level safety controller based on control barrier functions, which modifies the agents' actions in real time to prevent collisions. Finally, the aggregative tracking framework is also deployed using the ROS2 middleware, enabling a fully distributed architecture that mirrors the theoretical assumptions of the algorithms. The resulting system is capable of achieving cooperative behavior in real-time, while preserving both privacy and safety. This work highlights how distributed optimization, enriched with safety and implementation considerations, can serve as a foundation for intelligent, autonomous multi-robot systems.

Contents

Introduction

Motivations

Recent advancements in embedded electronics have enabled the integration of computation and communication capabilities into a wide range of devices, spanning various domains such as factories, farms, buildings, energy grids, and cities. This interconnectedness among devices has paved the way for the transformation of individual smart devices into smart cooperating systems. This emerging paradigm, characterized by its interconnection and complexity, where significant advantages arise from exploiting such structure, is commonly referred to as cyber-physical networks.
Within this context, a novel peer-to-peer distributed computational framework is gaining prominence. In such a framework, processors operate as peers, communicating over a network to collaboratively solve tasks, without relying on a single central authority that possesses all the data. [1]

Distributed Optimization Many challenges within cyber-physical networks such as estimation, decision-making, learning, and control can be cast as optimization problems. Solving these large-scale, structured problems using classical centralized algorithms is often impractical, as the problem data is distributed across the network, and collecting all data at a central node is either undesirable or infeasible.
In a distributed setup, the objective is to the design of algorithms where agents, acting as peers without a central coordinator, cooperatively solve an optimization task. Often, these systems impose privacy constraints, requiring that local data not be shared with other agents. These challenges have led to the growth of a new area of research called distributed optimization.

Communication model

In the distributed scenario we consider, the system is composed of $N \in \mathbb{N}$ agents (also referred to as robots or processors), each equipped with both computation and communication capabilities. The interactions among agents are modeled using graph theory.
At each discrete time instant $t \in \mathbb{N}$, the communication network is described by a directed graph:

$$\mathcal{G}_t = (\mathcal{I}, \mathcal{E}_t)$$

where $\mathcal{I} = \{1, \ldots, N\}$ denotes the set of agents, and $\mathcal{E}_t \subseteq \mathcal{I} \times \mathcal{I}$ represents the set of directed communication links at time t. A directed edge $(j, i) \in \mathcal{E}_t$ indicates that agent j is able to send information to agent i at time t.

For each agent $i \in \mathcal{I}$, we define the set of in-neighbors at time t as:

$$\mathcal{N}_i^t := \{j \in \mathcal{I} \mid (j, i) \in \mathcal{E}_t\}$$

These are the agents from which i can receive information at time t. Similarly, the set of out-neighbors is defined as:

$$\mathcal{N}_i^{t\ OUT} := \{j \in \mathcal{I} \mid (i, j) \in \mathcal{E}_t\}$$

corresponding to the agents to which i can send information.

The graph $\mathcal{G}_t$ is said to be fixed if the edge set remains constant over time, i.e., $\mathcal{E}_t \equiv \mathcal{E}$ for all t. Otherwise, the graph is said to be time-varying. If for every pair of nodes $i, j \in \mathcal{I}$, the existence of an edge $(i, j) \in \mathcal{E}_t$ implies that also $(j, i) \in \mathcal{E}_t$, the graph is undirected.
The communication structure at each time t can be associated with a weighted adjacency matrix $A_t \in \mathbb{R}^{N \times N}$, where the entry $a_{ij}^t > 0$ if $(j, i) \in \mathcal{E}_t$, and $a_{ij}^t = 0$ otherwise. This matrix captures how information is aggregated across the network. A matrix is column-stochastic if it has non-negative entries and each column sums to one. It is doubly-stochastic if, in addition, each row also sums to one. Doubly-stochastic matrices are useful in consensus algorithms as they ensure balanced and averaging behavior across the network.

Graph connectivity plays a critical role in the design and analysis of distributed algorithms. A fixed directed graph is said to be strongly connected if there exists a directed path between every pair of agents, meaning that information can propagate from any node to any other by following the edge directions. In contrast, a graph is weakly connected (also referred to as simply connected) if there exists an undirected path between every pair of nodes i.e. the connectivity holds when edge directions are ignored. Notably, when the graph is undirected, the notions of strong and weak connectivity coincide.
In the time-varying case, joint strong connectivity (or T-strong connectivity) ensures that the union of the communication graphs over a window of T consecutive time steps remains strongly connected, allowing information to eventually propagate throughout the network.

In the context of distributed algorithms, each agent starts from its own local state and performs an iterative procedure that alternates local computation with communication from its in-neighbors. Agents typically follow the same update rule and rely solely on locally accessible information. The underlying communication graph plays a crucial role in convergence properties. A graph is said to be periodic if there exists an integer $k > 1$ (called the period) that divides the length of every cycle in the graph. Conversely, a graph is aperiodic if no such k exists. Notably, a graph with at least one self-loop is aperiodic, which in distributed settings corresponds to agents incorporating their own state during updates.

Structure of the report

To guide the reader, the structure of the report is organized as follows. In Chapter 1, we focus on the consensus optimization problem and present the implementation of the Gradient Tracking algorithm, with applications to cooperative multi-target localization. Then, in Chapter 2, we address the aggregative optimization problem, detailing the design and implementation of a distributed control strategy based on Aggregative Tracking. Each section includes theoretical modeling, algorithmic implementation, and simulations results.

Chapter 1

Task 1 - Multi-Robot Target Localization

In the last decade, the optimization community has developed a novel theoretical framework to solve optimization problems over networks of communicating agents. In this computational setting, agents (e.g., processors or robots) co-operate with the aim of optimizing a common performance index without resorting to a central computing unit. The key assumption is that each agent is aware of only a small part of the optimization problem, and can communicate only with a few neighbors.
In the so called Distributed Cost-Coupled Optimization Problem, the objective is to minimize the sum of N local functions $\ell_i : \mathbb{R}^d \to \mathbb{R}$, each one depending on a common global variable $z \in Z \subseteq \mathbb{R}^d$:

$$\min_{z \in Z} \sum_{i=1}^{N} \ell_i(z)$$

In this setup, robots usually update local solution estimates $z_i^k \in \mathbb{R}^d$ that eventually converge to a common optimal solution $z^* \in \mathbb{R}^d$. This setup is particularly suited for decision systems performing data analytics in which data are private and the consensus has to be reached on a common set of parameters characterizing a global classifier or estimator (e.g., neural network) based on all data. In cooperative robotics, this scenario can be interesting to reach a consensus on a common quantity, e.g., target estimation. Although Distributed Cost-Coupled Optimization provides the most general problem formulation; algorithms designed for such a framework require storing, sharing, and reaching consensus on a decision variable that includes the stack of all robots' estimates. [2] For this reason this is also called Consensus Optimization.

Centralized setting. In the centralized formulation of the problem, we assume access to the entire collection of cost functions $\ell_i(z)$ for $i = 1, \dots, N$, which allows us to define the global objective function as $\ell(z) = \sum_{i=1}^{N} \ell_i(z)$. A widely used method in this context is the Gradient Method, an iterative optimization algorithm whose update rule can be expressed as:

$$\begin{aligned} z^{k+1} &= z^k + \alpha^k d^k \\ &= z^k - \alpha^k \nabla \ell(z^k), \end{aligned}$$

where z^k denotes the current estimate of the optimal solution at iteration k, α^k is the step size (also referred to as the learning rate), $\nabla \ell(z^k)$ is the gradient of the global cost function at z^k and $d^k = -\nabla \ell(z^k)$ is the steepest descent direction for minimizing the overall objective function.

Distributed setting. In a distributed setting, each agent i has access only to its local cost function $\ell_i(z)$ and communicates with a subset of other agents, typically those in its neighborhood. Instead of relying solely on local information, as in methods like the Incremental Gradient Method, distributed algorithms exploit inter-agent communication to improve convergence.
Figure 1.1 illustrates the evolution of local estimates z_i^k and z_j^k. The agents iteratively update their estimates, which evolve over time within a two-dimensional cost landscape.

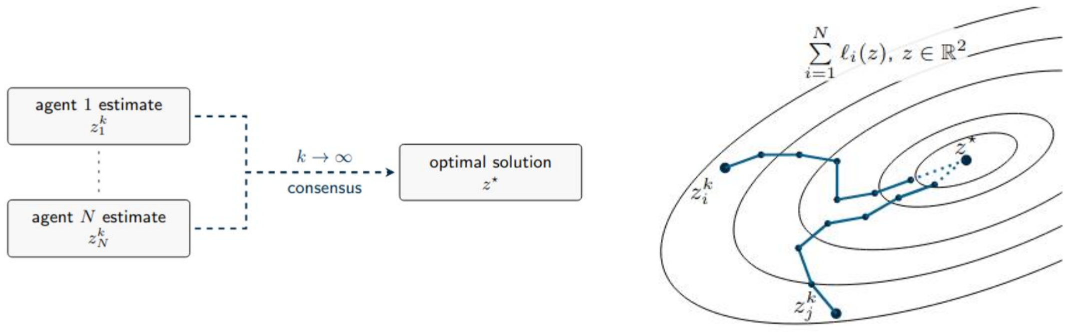


Figure 1.1: It shows the evolution of local estimates z_i^k and z_j^k in a distributed consensus optimization process. Through local communication and gradient updates, agents align their estimates over time and converge to the global optimum z^* of the objective $\sum_{i=1}^{N} \ell_i(z)$, with $z \in \mathbb{R}^2$.

A typical update rule in a Distributed Gradient Method takes the form:

$$z_i^{k+1} = \left(\sum_{j \in N_i} a_{ij} z_j^k \right) - \alpha^k \nabla \ell_i \left(\sum_{j \in N_i} a_{ij} z_j^k \right),$$

where z_i^k is agent i's estimate at iteration k, N_i denotes the set of neighbors of agent i, and a_{ij} are weights representing how much agent i is getting influenced by the estimate of neighbor j. This formulation combines information from the neighborhood to approximate the global direction of descent.
However, the direction computed as $d_i^k = -\nabla \ell_i \left(\sum_{j \in N_i} a_{ij} z_j^k \right)$ does not correspond to the true gradient of the global function $\ell(z)$, as in the centralized case. Instead, it can be seen as an approximation that, on average, points toward the optimal

solution. Due to this inaccuracy, especially near the optimum z^*, the method introduces a persistent error term, i.e. the estimate solution will "walk" within a ball with radius depending on α instead of converging.
To mitigate this issue, a possible approach is to adopt a diminishing step size α^k that gradually decreases over the iterations and inhibits the second term. Under such conditions, it can be shown that the method converges asymptotically to the optimal solution.
Using a constant stepsize $\alpha^k = \alpha > 0 \; \forall k$ not only simplifies the distributed implementation by removing the need for synchronization among agents on the iteration index or step value, but also can lead to improved convergence rates. Each agent can operate independently without global coordination, which is particularly beneficial in asynchronous or delay-prone networks, while still achieving faster convergence under suitable conditions.
To overcome the limitation of approximate convergence caused by the use of a constant stepsize, we aim to improve the local descent direction by reconstructing the true global gradient. Ideally, we would like each agent to compute a direction satisfying:

$$d_i^k \xrightarrow[k\to\infty]{} -\frac{1}{N}\sum_{h=1}^{N} \nabla \ell_h(z_h^k),$$

which corresponds to the average gradient of the global cost function. However, this quantity is not directly accessible in a distributed setting. For this reason we introduce a proxy variable that allows each agent to track the global gradient in a decentralized manner. We first introduce the core mechanism underlying this approach, and then show how it can be integrated into our specific setting.

Dynamic Average Consensus. The goal of the Dynamic Average Consensus (DAC) algorithm is to allow each agent to track, in a distributed manner, the time-varying average of local signals. Consider N agents, each observing a time-varying local signal $r_i^k \in \mathbb{R}$ at time k. The global quantity of interest is the average:

$$\bar{r}^k = \frac{1}{N}\sum_{i=1}^{N} r_i^k,$$

which is not globally available due to the distributed nature of the system. Each agent maintains a local estimate s_i^k and updates it via:

$$s_i^{k+1} = \sum_{j\in N_i} a_{ij} s_j^k + \left(r_i^{k+1} - r_i^k\right),$$

where the first term is a consensus step (a "mixing" step, a weighted average over neighbors), and the second is a local innovation that incorporates changes in the observed signal.
Under the assumption that the signal variation is bounded, it can be shown that the tracking error remains bounded; in the special case where the signal variations vanish asymptotically, i.e., $\|r_i^{k+1} - r_i^k\| \to 0$, the algorithm guarantees perfect tracking:

$$\lim_{k\to\infty} \|s_i^k - \bar{r}^k\| = 0.$$

The idea behind Gradient Tracking is to apply the dynamic average consensus mechanism to track the average gradient of the global objective function $\ell(z) = \sum_{i=1}^{N} \ell_i(z)$.
To this end, each agent uses its local gradient $\nabla \ell_i(z_i^k)$ as the input signal:

$$r_i^k = \nabla \ell_i(z_i^k),$$

and applies the update:

$$s_i^{k+1} = \sum_{j \in N_i} a_{ij} s_j^k + \left(\nabla \ell_i(z_i^{k+1}) - \nabla \ell_i(z_i^k)\right), \quad s_i^0 = \nabla \ell_i(z_i^0).$$

The descent direction at each agent is then set to:

$$d_i^k = -s_i^k.$$

If the gradient signals $\nabla \ell_i(z_i^k)$ converge (which is linked to the convergence of the decision variables z_i^k), the DAC mechanism ensures that each s_i^k will converge to the true average gradient:

$$s_i^k \xrightarrow[k \to \infty]{} \frac{1}{N} \sum_{h=1}^{N} \nabla \ell_h(z_h^k).$$

This strategy enables each agent to approximate the centralized gradient direction in a fully distributed way, significantly improving the convergence properties compared to standard distributed gradient descent.
The Gradient Tracking algorithm reads as:

$$\begin{aligned} z_i^{k+1} &= \sum_{j \in N_i} a_{ij} z_j^k - \alpha s_i^k \\ s_i^{k+1} &= \sum_{j \in N_i} a_{ij} s_j^k + (\nabla \ell_i(z_i^{k+1}) - \nabla \ell_i(z_i^k)) \end{aligned}$$

Gradient Tracking: Assumptions Let's make the following assumptions before providing the theorem results:

1. [`network`] Let a_{ij} with $i, j \in$ 1,..., N be non-negative entries of a weighted adjacency matrix A associated to the undirected and connected graph G, with $a_{ii} > 0$ and A doubly stochastic
2. [`cost`] For all $i \in$ 1, ..., N each cost function $\ell_i : \mathbb{R}^d \to \mathbb{R}$ satisfies the following conditions:
   - it is strongly convex with coefficient $\mu > 0$
   - it has Lipschitz continuous gradient with constant $L > 0$

Gradient Tracking: Theorem Let Assumptions 1 and 2 hold true. Then, there exists $\alpha^* > 0$ such that for all $\alpha \in (0, \alpha^*)$ the sequences of local solution estimates $\{z_i^k\}_{k \in \mathbb{N}}$, $i = 1, ..., N$, generated by the Gradient Tracking algorithm, asymptotically converge to a (consensual) solution z^* of the problem, i.e., for all $i = 1, ..., N$, it holds:

$$\lim_{k\to\infty} ||z_i^k - z^*|| = 0$$

Moreover, the convergence rate is linear/the stability is exponential, i.e., for all $i = 1, ..., N$ there exists $\rho \in (0, 1)$ such that

$$||z_i^k - z^*|| \le \rho||z_i^{k-1} - z^*|| \le \rho^k||z_i^0 - z^*||$$

for all $k \in \mathbb{N}$.

1.1 Distributed Consensus Optimization

We have implemented the Gradient Tracking algorithm as described, and verified its correctness through a set of simulations. In particular, we first tested the algorithm on a sum of quadratic functions, for which the optimal solution can be derived in closed form. This allowed us to compare the distributed solution with the centralized one and validate the convergence behavior.
In Task 1.2, we extended this analysis by comparing the performance of Gradient Tracking with the Centralized Gradient Method in the target localization scenario.

Assumption Checks. Certain structural properties of the communication graph are required to guarantee convergence of the distributed algorithm. In our setup, these assumptions are explicitly verified:

- *Aperiodicity.* The presence of self-loops in the graph ensures that it is aperiodic. From the perspective of distributed algorithms, this implies that each agent i incorporates its own local information during each iteration.
- *Strong connectivity.* We verified strong connectivity by computing the N-th power of the adjacency matrix A, where N is the number of agents. The entry (i, j) of A^k indicates the existence of a path of length k from node i to node j. If all entries of A^N are nonzero, this implies that every agent is reachable from every other agent within at most N steps, confirming that the graph is strongly connected.
- *Doubly stochastic weights.* The communication graph is generated as an undirected graph. Then, the weights are assigned according to the Metropolis-Hastings algorithm, which guarantees that the resulting weight matrix is doubly stochastic by construction.
- *Cost function properties.* The cost functions must satisfy certain regularity conditions. Quadratic or quartic cost functions fulfill these assumptions and admit a unique global minimum.

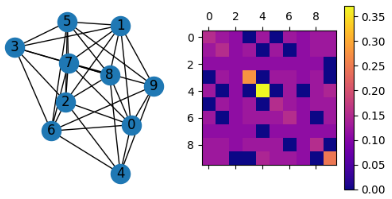


Figure 1.2: Experiment 1: $N = 10$, $\alpha = 0.05$, `GraphType.ERDOS_RENYI` with $p = 0.65$

Experimental Results. As shown in Figure 1.2, we report the results of a simulation using an Erdős–Rényi communication graph with edge probability $p = 0.65$, $N = 10$ nodes, and Metropolis-Hastings weights for the adjacency matrix. The local cost functions are quadratic, of the form:

$$\ell_i(z) = \frac{1}{2} z^\top Q_i z + r_i^\top z,$$

where each Q_i is a diagonal positive-definite matrix and r_i is a randomly generated vector. The gradient is therefore given by $\nabla \ell_i(z) = Q_i z + r_i$.
The global optimum z^* can be computed in closed form as:

$$z^* = -\bar{Q}^{-1}\bar{r},$$

where $\bar{Q} = \sum_{i=1}^{N} Q_i$ and $\bar{r} = \sum_{i=1}^{N} r_i$. In this specific configuration, the optimal cost is approximately -3.262.

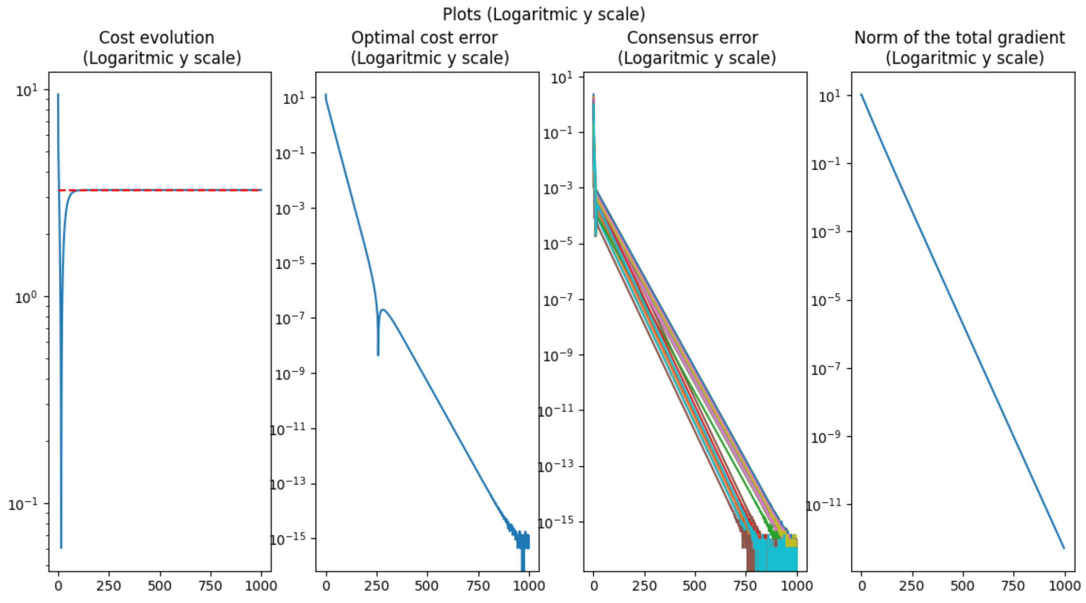


Figure 1.3: Experiment 1: $N = 10$, $\alpha = 0.05$, `GraphType.ERDOS_RENYI` with $p = 0.65$

In the first plot (far left) of the Figure 1.3, we show the evolution of the absolute value of the cost function over time, in logarithmic scale. The cost starts from a positive value, then becomes negative, and finally stabilizes close to the optimal value (red dashed line).
The second plot displays the optimal cost error, defined as the difference between the current cost and the optimal one computed centrally. The convergence is linear, with the error rapidly decreasing to around 10^{-15} in 1000 iterations.
The third plot shows the consensus error, defined as the distance between each local estimate z_i^k and the average $\frac{1}{N}\sum_{j=1}^{N} z_j^k$. The consensus error of each agent converges to zero with a linear rate.
Finally, the fourth plot reports the norm of the total gradient. As expected, the gradient norm approaches zero, indicating that the algorithm has reached a stationary point. Due to the strong convexity of the cost functions, this stationary point also corresponds to the unique global optimum.

1.2 Cooperative Multi-Robot Target Localization

As previously introduced, one relevant application of distributed optimization is cooperative target localization. Consider a team of $N \in \mathbb{N}$ mobile robots aiming to estimate the positions of one or more targets based on local, noisy measurements. As illustrated in Figure 1.4, each robot is equipped with sensors that allow it to observe the position of a target within a limited sensing range.

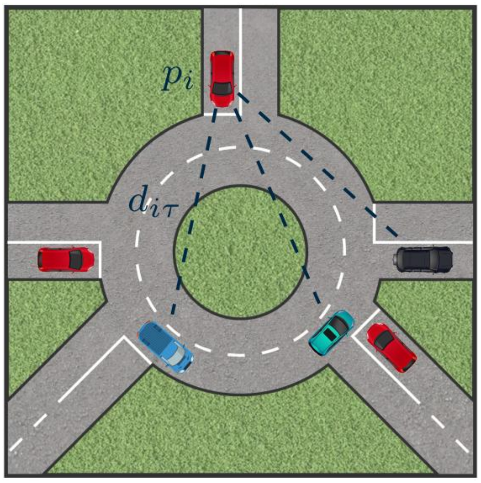


Figure 1.4: A team of red vehicles observing three targets (black, blue, and green vehicles). The blue, dashed lines represent the noisy measurements made one vehicle in the team.

In our simulation, we randomly generated N robot positions, denoted by $p_i \in \mathbb{R}^d$, and $N_T \in \mathbb{N}$ target positions. For each robot i and each target $\tau \in \{1, \dots, N_T\}$, we computed the true Euclidean distance and then perturbed it by adding Gaussian noise with fixed mean μ and variance σ^2. This yielded a set of noisy measurements $d_{i\tau} \in \mathbb{R}_{\geq 0}$ representing the observed distance between robot i and target τ.
In this simplified scenario, we assume that each robot can measure its distance to

every target.
We define the local cost function for agent i as:

$$\ell_i(z) := \sum_{\tau=1}^{N_T} \left(d_{i\tau}^2 - \|z_\tau - p_i\|^2\right)^2$$

where $z = \text{col}(z_1, \ldots, z_{N_T}) \in \mathbb{R}^{dN_T}$ is the optimization variable, i.e. the stacked vector of estimated target positions, with τ-th block component $z_\tau \in \mathbb{R}^d$ representing the estimate for target τ, and $p_i \in \mathbb{R}^d$ is the known position of robot i.
This is a least-squares formulation where each robot tries to minimize the difference between the squared distance measurement and the squared Euclidean distance between the estimated target and its own position.

Gradient computation of $\ell_i(z)$ We observe that the function $\ell_i(z)$ is a sum of squared residuals, where each term only depends on the corresponding block variable z_τ. We define the residual:

$$r_{i\tau}(z_\tau) := d_{i\tau}^2 - \|z_\tau - p_i\|^2$$

Then, the cost can be rewritten as:

$$\ell_i(z) = \sum_{\tau=1}^{N_T} r_{i\tau}(z_\tau)^2$$

We compute the gradient of ℓ_i with respect to each block $z_\tau \in \mathbb{R}^d$ using the chain rule:

$$\nabla_{z_\tau} \ell_i(z) = 2 \cdot r_{i\tau}(z_\tau) \cdot \nabla_{z_\tau} r_{i\tau}(z_\tau)$$

It remains to compute the gradient of the residual wrt. z_τ:

$$\nabla_{z_\tau} r_{i\tau}(z_\tau) = \nabla_{z_\tau} \left(d_{i\tau}^2 - \|z_\tau - p_i\|^2\right) = -2(z_\tau - p_i)$$

Therefore, we obtain:

$$\nabla_{z_\tau} \ell_i(z) = -4\left(d_{i\tau}^2 - \|z_\tau - p_i\|^2\right)(z_\tau - p_i)$$

Finally, the full gradient $\nabla \ell_i(z) \in \mathbb{R}^{dN_T}$ is given by stacking the partial gradients for each target estimate:

$$\nabla \ell_i(z) = \begin{bmatrix} -4\left(d_{i1}^2 - \|z_1 - p_i\|^2\right)(z_1 - p_i) \\ -4\left(d_{i2}^2 - \|z_2 - p_i\|^2\right)(z_2 - p_i) \\ \vdots \\ -4\left(d_{iN_T}^2 - \|z_{N_T} - p_i\|^2\right)(z_{N_T} - p_i) \end{bmatrix}$$

Simulations We begin by validating the correctness of our implementation using noise-free distance measurements.
In Figure 1.5, the configuration for the subsequent simulations is illustrated. The map displays red dots that signify the agents, while the blue crosses indicate the target positions to be estimated. Both the positions of the agents and the positions of the targets are generated randomly within a world-size 10×10.

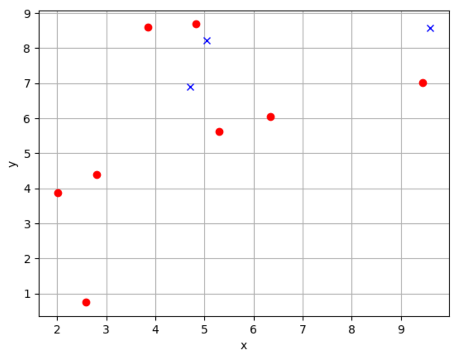


Figure 1.5: The map features red dots representing the agents and blue crosses indicating the target positions to be estimated.

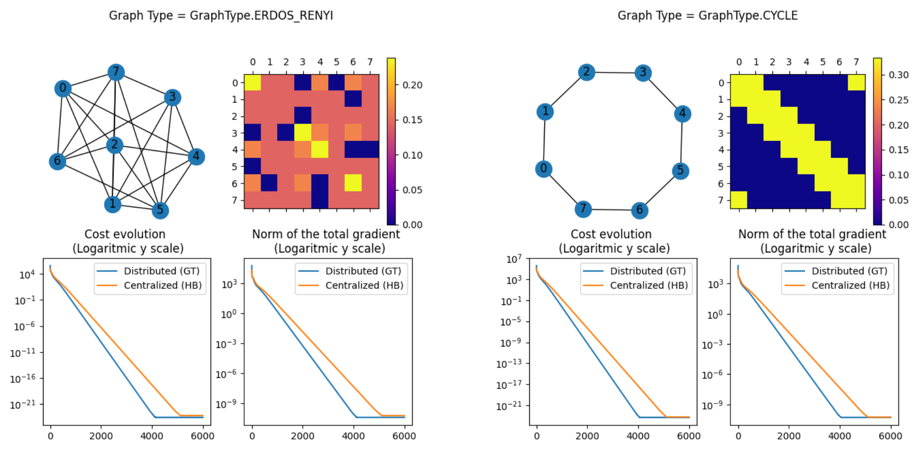


Figure 1.6: Comparison between two different communication topologies: `GraphType.ERDOS_RENYI` and `GraphType.CYCLE`. Parameters used: $N = 8$ agents and $N_T = 3$ targets, with noise-free distances.

To validate the implementation, we tested the algorithm on different network topologies. In the plots shown in Figure 1.6, we report two representative cases: one based on a randomly generated Erdős–Rényi graph and the other on a cycle graph. In both cases, the cost function and the norm of the gradient converge to zero at a linear rate. Since the optimal value of the least-squares problem in the noise-free setting is zero, the decreasing cost confirms the correctness of the algorithm.
We now move to a more realistic scenario, where measurements are affected by

Gaussian noise. In the plots shown in Figure 1.7, we observe that the gradient norm converges to zero at a linear rate, while the cost function stabilizes at a value strictly greater than zero. This behavior is due to the presence of noise in the measurements, which prevents the cost from reaching the theoretical minimum.

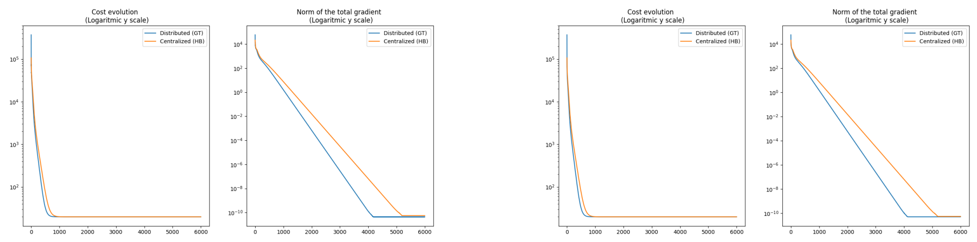


Figure 1.7: In this figure we can see two graph types respectively `GraphType.ERDOS_RENYI` e `GraphType.CYCLE`. Parameters used: $N = 8$ agents and $N_T = 3$ targets, with noisy distances.

Chapter 2

Task 2 - Aggregative Optimization for Multi-Robot Systems

2.1 Distributed Aggregative Optimization

In this chapter, we address a coordination problem in which the goal is no longer to achieve uniform consensus, but rather to enable the emergence of a collective behavior that minimizes a global cost function. Unlike classical consensus schemes, where all agents asymptotically agree on a common state, here each agent evolves toward individual states that, together, optimize an objective function defined over the entire network.
A common and effective approach to this class of problems is provided by aggregative optimization, where each agent minimizes a local objective function that depends both on its own decision variable and on an aggregate quantity (typically the average) of all agents' decisions. The structure of the aggregate term allows for the development of scalable, decentralized, and privacy-preserving algorithms, often built on average-tracking dynamics or modified consensus schemes.

This framework is particularly suitable for cooperative robotics applications. For instance, consider N robots in the plane, each optimizing its position $z_i \in \mathbb{R}^d$, for $i = 1, \ldots, N$, to perform multi-robot surveillance tasks.
Let $r_0 \in \mathbb{R}^d$ denote a global target to protect, and $r_i \in \mathbb{R}^d$ represent an intruder associated with robot i. Define the barycenter of the robots as

$$\sigma(z) = \frac{1}{N} \sum_{i=1}^{N} z_i,$$

and let $\gamma_i > 0$ be a tradeoff parameter that assigns different weights to components of each robot's cost function.
Each robot aims to achieve two goals simultaneously: remain close to the global target r_0, and stay close to its associated intruder r_i. A simple way to express this is by considering the squared Euclidean distances between each robot and its intruder, and between the barycenter of all robots and the global target.

Accordingly, we define the local cost function for robot i as

$$\ell_i(z_i, \sigma(z)) = \gamma_i \|z_i - r_i\|^2 + \|\sigma(z) - r_0\|^2,$$

where the first term corresponds to a private objective of staying close to the intruder (private opponent), and the second term captures a global objective of maintaining proximity to the overall target.
In this context, the vector $z \in \mathbb{R}^{dN}$, formed by stacking the positions $z_1, \ldots, z_N$, plays a different role compared to the Gradient Tracking algorithm. Here, z represents the collective positions of all robots, which collaboratively seek to minimize the total cost function by converging to an optimal configuration.

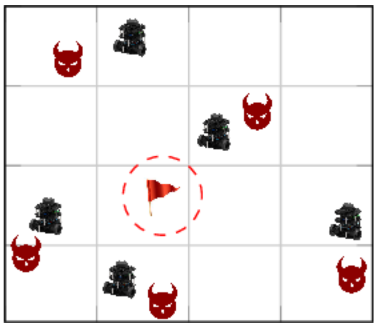


Figure 2.1: Multi- robot surveillance scenario - Robot icons denote agents, devil icons denote intruders, while the flag is the target to be protected. [3]

Consider aggregative optimization problems formulated as

$$\min_{z_1, \ldots, z_N} \quad \sum_{i=1}^{N} \ell_i(z_i, \sigma(z))$$

where the aggregative variable $\sigma(z)$ is defined as

$$\sigma(z) = \frac{1}{N} \sum_{i=1}^{N} \phi_i(z_i)$$

where $z = (z_1, \ldots, z_N)$, with each $z_i \in \mathbb{R}^{n_i}$, for all $i = 1, \ldots, N$, and $\ell_i : \mathbb{R}^{n_i} \times \mathbb{R}^d \to \mathbb{R}$ and $\phi_i : \mathbb{R}^{n_i} \to \mathbb{R}^d$, for all $i = 1, \ldots, N$. As introduced in the previous example, the problem can be described with $\phi_i(z_i) = z_i$ representing the equally weighted barycenter. In general, the aggregative function $\sigma(z)$ does not have to be the barycenter specifically; rather, it must have the average form $\sigma(z) = \frac{1}{N} \sum_{i=1}^{N} \phi_i(z_i)$.
This form ensures that each agent contributes equally to the aggregate, enabling distributed computation and preserving symmetry among agents. The barycenter is commonly used because it offers a simple, intuitive, and fair representation of the collective influence, while maintaining the problem's traceability.

For completeness, we present both the Centralized and Distributed Aggregative Optimization frameworks, and their comparison will be further detailed in the implementation section.

Centralized setting. In the centralized setting, a central coordinator has access to all agents' information and solves the global aggregative optimization problem collectively.
Consider the decision variables $z_i \in \mathbb{R}^{n_i}$ for $i = 1, \dots, N$. The centralized gradient descent method at iteration k updates each z_i as

$$z_i^{k+1} = z_i^k - \alpha \left[\nabla_{z_i} \sum_{j=1}^{N} \ell_j(z_j, \sigma(z_1, \dots, z_N)) \right] \Bigg|_{z_1 = z_1^k, \dots, z_N = z_N^k}$$

where $\alpha > 0$ is the step size, and the notation $\nabla_{z_i} \ell_j(\cdot)$ denotes the gradient of ℓ_j with respect to the variable z_i.
To explicitly compute the gradient, we apply the chain rule to the composite function $\ell_j(z_j, \sigma(z))$. The gradient at iteration k becomes

$$\begin{aligned} &\left[\nabla_{z_i} \textstyle\sum_{j=1}^{N} \ell_j(z_j, \sigma(z_1, \dots, z_N))\right] \Bigg|_{z_1 = z_1^k, \dots, z_N = z_N^k} \in \mathbb{R}^{n_i} \\ &= \nabla_1 \ell_i(z_i, \sigma) \Bigg|_{z_i = z_i^k, \sigma = \frac{1}{N} \sum_{j=1}^{N} \phi_j(z_j^k)} + \tfrac{1}{N} \nabla \phi_i(z_i) \Bigg|_{z_i = z_i^k} \cdot \\ &\quad \textstyle\sum_{j=1}^{N} \nabla_2 \ell_j(z_j, \sigma) \Bigg|_{z_j = z_j^k, \sigma = \frac{1}{N} \sum_{j=1}^{N} \phi_j(z_j^k)} \end{aligned}$$

where $\nabla_1 \ell_i$ is the gradient of ℓ_i with respect to its first argument z_i, and $\nabla_2 \ell_j$ is the gradient with respect to the second argument σ.
From this formulation, it is evident that some terms, such as $\nabla_1 \ell_i(z_i^k, \sigma^k)$ and $\nabla \phi_i(z_i^k)$, can be computed locally by each agent. However, the aggregative variable σ^k and the summation $\sum_{j=1}^{N} \nabla_2 \ell_j(z_j^k, \sigma^k)$ require global knowledge, which motivates the need for approximation techniques in distributed implementations.
Thus, the centralized update rule can be written as

$$z_i^{k+1} = z_i^k - \alpha \left(\nabla_1 \ell_i(z_i^k, \sigma^k) + \frac{1}{N} \nabla \phi_i(z_i^k) \sum_{j=1}^{N} \nabla_2 \ell_j(z_j^k, \sigma^k) \right).$$

Let us now derive the distributed version of this algorithm.

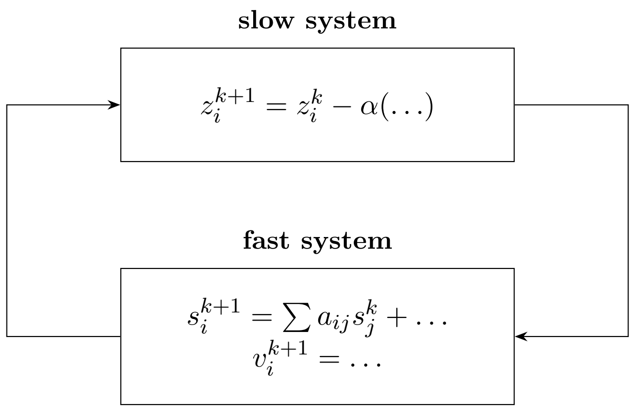
Distributed setting. In a distributed setting, each agent i has access only to its local functions ℓ_i and ϕ_i, and maintains an estimate z_i^k of its optimal decision variable z_i^*.
A key challenge in this context is that the gradient computation required for the update cannot be performed locally by a single agent, as it depends on global quantities that aggregate information from all agents.
Specifically, the quantities

$$\sigma = \frac{1}{N} \sum_{j=1}^{N} \phi_j(z_j^k) \quad \text{and} \quad \frac{1}{N} \sum_{j=1}^{N} \nabla_2 \ell_j(z_j^k, \sigma)$$

are global and thus unknown to individual agents.
To overcome this, each agent maintains local estimates s_i^k and v_i^k that serve as proxies for these global quantities, respectively. These proxies are dynamically updated through local communication with neighboring agents to track the true global values over time. This approach draws inspiration from the gradient tracking technique, exploiting the fact that both σ and the sum of gradients can be interpreted as averages, which can be tracked using dynamic consensus algorithms.
The distributed algorithm proceeds as follows: agent i updates its local variables according to:

$$\begin{aligned}
z_i^{k+1} &= z_i^k - \alpha \left(\nabla_1 \ell_i(z_i^k, s_i^k) + \nabla \phi_i(z_i^k) v_i^k \right), && z_i^0 \in \mathbb{R}^{n_i} \\
s_i^{k+1} &= \sum_{j \in \mathcal{N}_i} a_{ij} s_j^k + \phi_i(z_i^{k+1}) - \phi_i(z_i^k), && s_i^0 = \phi_i(z_i^0) \\
v_i^{k+1} &= \sum_{j \in \mathcal{N}_i} a_{ij} v_j^k + \nabla_2 \ell_i(z_i^{k+1}, s_i^{k+1}) - \nabla_2 \ell_i(z_i^k, s_i^k), && v_i^0 = \nabla_2 \ell_i(z_i^0, s_i^0)
\end{aligned}$$

where the weight coefficients a_{ij} correspond to a consensus matrix consistent with the communication graph $\mathcal{N}_i$ of agent i.
The proxies s_i^k and v_i^k thus perform a dynamic consensus to track the evolving aggregate quantities σ and $\frac{1}{N}\sum_j \nabla_2 \ell_j$, allowing each agent to approximate the global gradient information necessary for convergence while relying only on local information and neighbor communication.
A key aspect of this distributed algorithm is the separation of time scales between the consensus variables (s_i^k, v_i^k) and the decision variables z_i^k. Specifically, the consensus dynamics evolve on a fast time scale, while the updates of the decision variables happen on a slow time scale regulated by the step size $\alpha > 0$.


slow system
$z_i^{k+1} = z_i^k - \alpha(\dots)$
fast system
$s_i^{k+1} = \sum a_{ij} s_j^k + \dots$
$v_i^{k+1} = \dots$

The consensus updates, involving weighted averages of neighbors' estimates combined with correction terms, enable the proxies to rapidly track the true aggregate quantities $\sigma(z^k)$ and $\frac{1}{N}\sum_j \nabla_2 \ell_j(z_j^k, \sigma(z^k))$. Since these updates do not depend on the step size α, they can converge much faster than the decision variables z_i^k, which are updated incrementally with step size α.
The step size α controls how quickly each agent adjusts its decision variable z_i^k in response to gradient information. A smaller α causes the decision variables to evolve more slowly, effectively allowing the consensus variables s_i^k and v_i^k to track

the aggregate quantities accurately before significant changes in z_i^k occur. This slow evolution relative to the fast consensus ensures that gradient computations use up-to-date global information, preventing divergence due to stale or inaccurate estimates.
Mathematically, this can be viewed as a two-time-scale dynamical system, where the fast dynamics correspond to consensus on s_i^k and v_i^k, and the slow dynamics correspond to gradient descent steps on z_i^k. Under suitable assumptions on the communication graph and smoothness and convexity properties of the cost functions, this separation enables the distributed algorithm to closely approximate the centralized gradient descent, guaranteeing convergence to the global optimum at a linear rate for sufficiently small α. [4]

Theorem: Convergence of the Algorithm Let's us consider a communication graph G that is strongly connected and aperiodic, with an associated consensus matrix A that is doubly stochastic. Suppose the function $\sum_{i=1}^{N} \ell_i(\cdot, \sigma(\cdot))$ is strongly convex, each $\phi_i(\cdot)$ is differentiable and Lipschitz continuous, and the gradients $\nabla_1 \ell_i(\cdot, \cdot)$, $\nabla_2 \ell_i(\cdot, \cdot)$, and $\nabla \phi_i(\cdot) \nabla_2 \ell_i(\cdot, \cdot)$ are Lipschitz continuous. Under these conditions, there exists a step size $\alpha^* > 0$ such that for all $\alpha \in (0, \alpha^*)$, the sequences of local solution estimates $\{z_1^k, \dots, z_N^k\}_{k \in \mathbb{N}}$ generated by the Distributed Aggregative Tracking algorithm satisfy

$$\lim_{k \to \infty} \|z_i^k - z_i^\star\| = 0,$$

at a linear rate for all agents $i = 1, \dots, N$.

2.2 Implementation of a Distributed Aggregative Optimization Algorithm

Consider a team of N mobile robots operating in a two-dimensional environment, so that each robot's position lies in $\mathbb{R}^d$, with $d = 2$. Let $z_i^k \in \mathbb{R}^d$ denote the position of robot i at iteration $k \in \mathbb{N}$, and let $z^k \in \mathbb{R}^{dN}$ be the stacked vector containing the positions of all robots at time k.
The objective is to design a distributed control algorithm that enables each robot to move towards a designated private target, while maintaining cohesion within the team due to communication constraints. It is possible to see an illustrative example of this setup in Figure 2.2.

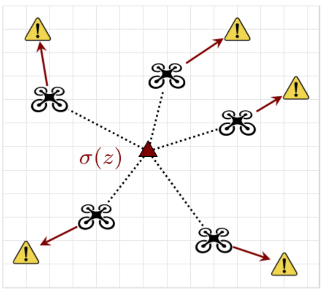


Figure 2.2: Example of the aggregative optimization scenario. Each robot aims to reach a private target while keeping the formation cohesive.

This scenario naturally fits within the framework of aggregative optimization, as it shares key features with the multi-robot surveillance problem introduced earlier. In this context, we define the global aggregative quantity $\sigma(z)$ as the barycenter of the entire team, i.e.,

$$\sigma(z) = \frac{1}{N}\sum_{j=1}^{N} \phi_j(z_j), \quad \text{with } \phi_j(z_j) = z_j,\ \forall j = 1, \dots, N.$$

Cost function definition. To solve this problem, the first step is to define a local cost function that captures the dual objective of staying close to the assigned target, while preserving the spatial coherence of the team. By drawing a parallel with the multi-robot surveillance scenario introduced earlier, we can interpret each robot's private target as analogous to the intruder assigned to that robot, while the tendency to aggregate toward the barycenter reflects the need to maintain coverage or provide collective protection around a specific area, similar to the global protection objective in the surveillance setting.
As a first, simplified implementation, we model the local cost function as a weighted combination of two terms:

- the squared Euclidean distance between the robot and its private target, encouraging convergence toward the target,
- the squared Euclidean distance between the robot and the team's barycenter, promoting group cohesion.

Let $r_i \in \mathbb{R}^d$ denote the position of the target associated with robot i, and let $\gamma_i, \overline{\gamma_i} > 0$ be weights that balance the trade-off between tracking the private target and maintaining cohesion within the group. The resulting local cost function for agent i is given by:

$$\ell_i(z_i, \sigma(z)) = \gamma_i \|z_i - r_i\|^2 + \overline{\gamma_i}\|z_i - \sigma(z)\|^2.$$

In Section 2.2.1, we will present an enhanced version of the algorithm that incorporates collision avoidance capabilities. Specifically, we introduce a low-level safety controller that modifies the planned trajectories to ensure inter-robot collision avoidance, while the high-level distributed aggregative optimization framework governs the global coordination behavior.

Centralized setting. Following the theoretical framework, we first decided to implement the centralized algorithm before moving on to the distributed one. As introduced in the previous section, the centralized update rule is given by

$$\begin{aligned} z_i^{k+1} &= z_i^k - \alpha \left[\nabla_{z_i} \sum_{j=1}^{N} \ell_j(z_j, \sigma(z_1, \dots, z_N))\right]\Bigg|_{z_1 = z_1^k, \dots, z_N = z_N^k} \\ &= z_i^k - \alpha \left(\nabla_1 \ell_i(z_i^k, \sigma^k) + \frac{1}{N}\nabla\phi_i(z_i^k)\sum_{j=1}^{N}\nabla_2 \ell_j(z_j^k, \sigma^k)\right), \end{aligned}$$

where ∇_1 denotes the partial gradient with respect to the first argument of ℓ_i (i.e., z_i), and ∇_2 the partial gradient with respect to the second argument (the aggregative variable σ).
We start by computing $\nabla_1 \ell_i(z_i^k, \sigma(z^k))$. From the definition of the local cost function:

$$\ell_i(z_i, \sigma(z)) = \gamma_i \|z_i - r_i\|^2 + \overline{\gamma_i}\|z_i - \sigma(z)\|^2,$$

we have

$$\begin{aligned}
\nabla_1 \ell_i(z_i, \sigma(z)) &= \nabla_{z_i}\left(\gamma_i \|z_i - r_i\|^2 + \overline{\gamma_i}\|z_i - \sigma(z)\|^2\right) \\
&= 2\gamma_i(z_i - r_i) + 2\overline{\gamma_i}(z_i - \sigma(z)) \cdot \nabla_{z_i}(z_i - \sigma(z)) \\
&= \begin{bmatrix} 2\gamma_i(z_{i_1} - r_{i_1}) + 2\overline{\gamma_i}(z_{i_1} - \sigma_1(z)) \cdot \frac{\partial}{\partial z_{i_1}}(z_{i_1} - \sigma_1(z)) \\ \dots \\ 2\gamma_i(z_{i_d} - r_{i_d}) + 2\overline{\gamma_i}(z_{i_d} - \sigma_d(z)) \cdot \frac{\partial}{\partial z_{i_d}}(z_{i_d} - \sigma_d(z)) \end{bmatrix}
\end{aligned}$$

Expressing $\sigma(z)$ explicitly as the barycenter of the team,

$$\sigma(z) = \frac{1}{N}\sum_{j=1}^{N} z_j = \begin{bmatrix} \frac{1}{N}\sum_{j=1}^{N} z_{j_1} \\ \dots \\ \frac{1}{N}\sum_{j=1}^{N} z_{j_d} \end{bmatrix},$$

we note that its derivative with respect to z_i is the following Jacobian matrix:

$$\nabla_{z_i}\sigma(z) = \begin{bmatrix} \frac{1}{N} & 0 & \cdots & 0 \\ 0 & \frac{1}{N} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{1}{N} \end{bmatrix} = \frac{1}{N} I_d$$

where I_d is the d-by-d identity matrix.
Therefore,

$$\nabla_{z_i}(z_i - \sigma(z)) = I_d - \frac{1}{N} I_d = \left(1 - \frac{1}{N}\right) I_d.$$

Using this, the gradient with respect to z_i becomes

$$\nabla_1 \ell_i(z_i, \sigma(z)) = 2\gamma_i(z_i - r_i) + 2\overline{\gamma_i}\left(1 - \frac{1}{N}\right)(z_i - \sigma(z)).$$

Next, we compute $\nabla_2 \ell_i(z_i, \sigma(z))$, the gradient with respect to the aggregative variable σ:

$$\begin{aligned}
\nabla_2 \ell_i(z_i, \sigma(z)) &= \nabla_{\sigma(z)}\left(\gamma_i \|z_i - r_i\|^2 + \overline{\gamma_i}\|z_i - \sigma(z)\|^2\right) \\
&= -2\overline{\gamma_i}(z_i - \sigma(z)).
\end{aligned}$$

Finally, since $\phi_i(z_i) = z_i$, we have

$$\nabla \phi_i(z_i) = I_d.$$

At this point, we have computed all the necessary components to implement the centralized version of the aggregative optimization algorithm. The result of a simulation with four agents is shown in the figure below.

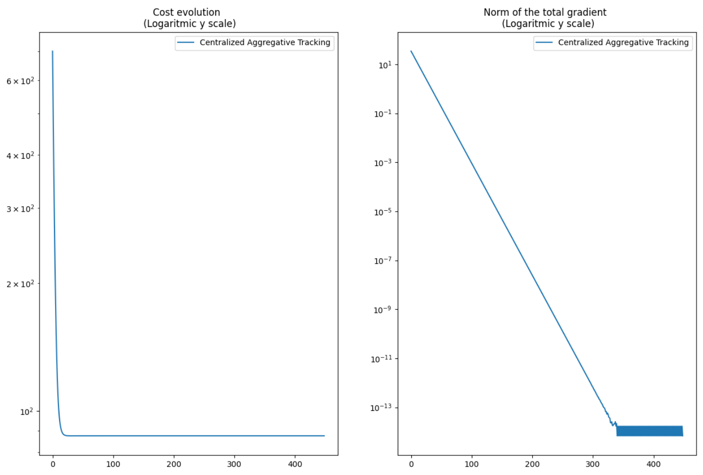


Figure 2.3: Centralized aggregative optimization applied to a team of four agents. The figure illustrates fast consensus convergence and the linear decrease of the gradient norm toward zero.

As illustrated, the centralized algorithm behaves as expected: the agents quickly reach consensus on a common configuration, and the norm of the gradients converges linearly to zero.

Distributed setting. Let us now proceed with the distributed algorithm. As mentioned in the previous section, in distributed scenarios each agent lacks access to the full network data, making it impossible to compute the gradient exactly as in the centralized case. To overcome this, we rely on local proxy variables s_i^k and v_i^k that track the global aggregative quantities in a decentralized manner.
Recall that the distributed aggregative tracking optimization algorithm is defined by the following update rules:

$$\begin{aligned}
z_i^{k+1} &= z_i^k - \alpha\left(\nabla_1 \ell_i(z_i^k, s_i^k) + \nabla\phi_i(z_i^k)v_i^k\right), && z_i^0 \in \mathbb{R}^{n_i} \\
s_i^{k+1} &= \sum_{j\in\mathcal{N}_i} a_{ij}s_j^k + \phi_i(z_i^{k+1}) - \phi_i(z_i^k), && s_i^0 = \phi_i(z_i^0) \\
v_i^{k+1} &= \sum_{j\in\mathcal{N}_i} a_{ij}v_j^k + \nabla_2\ell_i(z_i^{k+1}, s_i^{k+1}) - \nabla_2\ell_i(z_i^k, s_i^k), && v_i^0 = \nabla_2\ell_i(z_i^0, s_i^0)
\end{aligned}$$

Here, each agent i updates its estimate z_i based on local gradients evaluated at the proxy variables, while s_i and v_i evolve through weighted consensus steps over neighbors $\mathcal{N}_i$, combined with local corrections to track the global aggregative quantities and their gradients.
The following figure (Figure 2.4) shows the same scenario as the centralized aggregative optimization simulation, but now the agents communicate over a random graph.

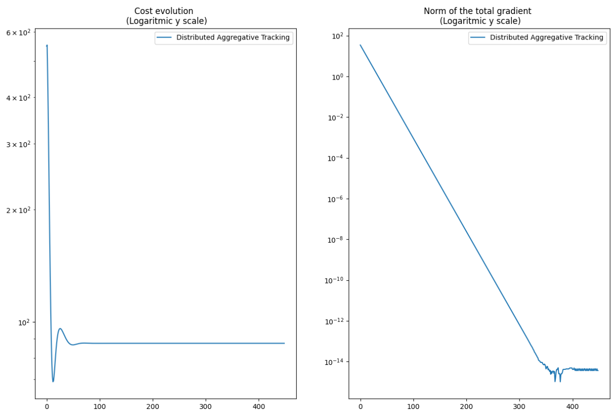
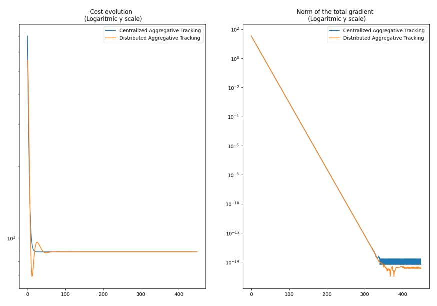


Figure 2.4: Results of the Distributed Aggregative Optimization Algorithm

Also in this case, the convergence theorem holds, and the norm of the gradient converges to zero at a linear rate.



The plot compares the convergence behavior of both the centralized and distributed algorithms. As expected, both satisfy the convergence theorem. The distributed algorithm experiences slower convergence due to the limited local communication among agents-each agent only communicates with its neighbors. However, after a sufficient number of iterations, consensus is reached across the network.

Figure 2.5: Comparison of Convergence between Distributed and Centralized Aggregative Optimization Algorithms

Simulations To better understand the effectiveness of our implementation, we tested the same algorithm under different conditions by varying both the graph topology and the agents' behaviors. In particular, we considered three scenarios that highlight different trade-offs between individual objectives and aggregative terms.
In the first scenario (Figure 2.6) , we introduced a "VIP" agent that does not follow a specific target, but is constrained to remain close to the barycenter of all agents. This can be interpreted as a situation where the rest of the agents must coordinate to "protect" the VIP.

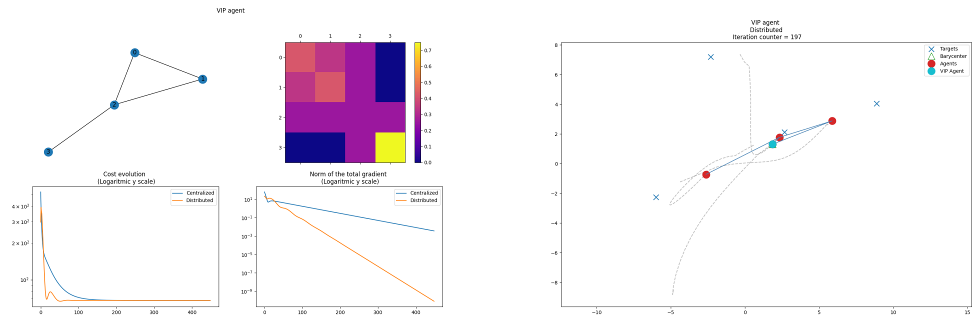


Figure 2.6: Aggregative tracking with VIP agent fixed at the barycenter.

In the second scenario (Figure 2.7), we increased the weight of the aggregative term in each cost function. In this case, the agents prioritize staying close to one another rather than to their individual targets. This promotes group cohesion and leads to tightly clustered trajectories, often at the expense of accuracy in target tracking.

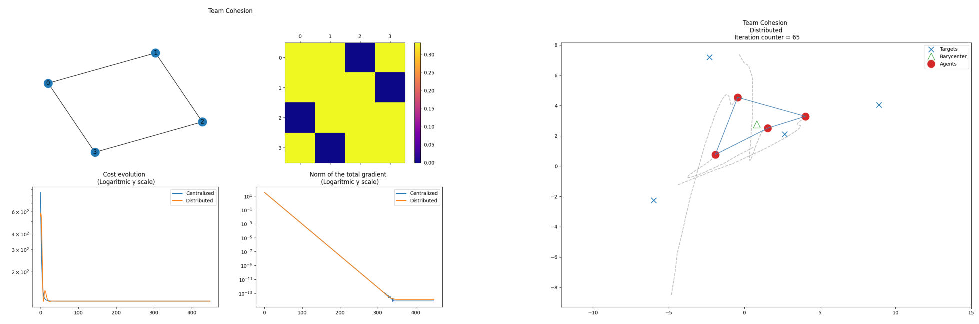


Figure 2.7: Aggregative tracking with stronger team cohesion.

Finally, we considered the opposite setting, where agents are driven more strongly by their private targets and less influenced by the aggregative term (Figure 2.8). This results in trajectories that more closely follow the desired target paths, while still maintaining some degree of coordination.

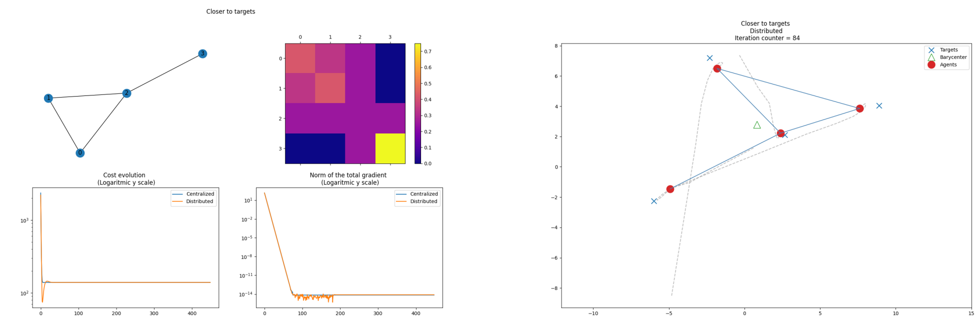
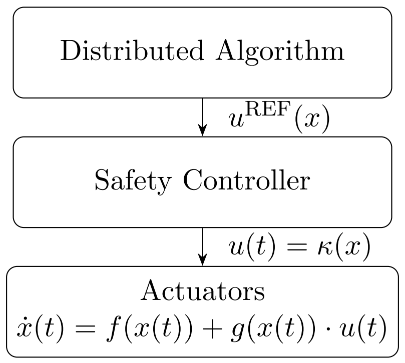


Figure 2.8: Aggregative tracking with stronger focus on individual targets.

2.2.1 Aggregative Tracking Distributed Optimization Algorithm with a Low-Level Multi-Robot Safety Controller

The cooperative aspect of multi-robot control corresponds to a high-level task that focuses on achieving collective objectives, such as formation control, coverage, or task allocation. This is fundamentally different from obstacle avoidance, which is considered a lower-level task concerned primarily with ensuring the safety of each individual robot by preventing collisions with obstacles or other robots.
In this context, a safety controller used on a distributed algorithm can be interpreted as a safety filter. Its role is to adjust or modify the control inputs generated by the high-level cooperative controller in real time, so as to guarantee collision avoidance and maintain safe operation of the robotic team. By filtering the reference inputs, the safety controller ensures that safety constraints are respected without compromising the overall cooperative behavior, thereby smoothly integrating high-level coordination with low-level safety.



Safety Control for Multi-Robot Systems The main idea of the safety control is to separate the safe and unsafe areas via a single function, which differs only in sign:

$$V^s(x) \geq 0 \quad \text{(safe)}, \qquad V^s(x) < 0 \quad \text{(unsafe)}.$$

Our goal is to design a control law such that if the system state is in the safe set initially, it remains there for all future time (forward invariance).
Formally, if we consider a control-affine nonlinear dynamical system:

$$\dot{x}(t) = f(x(t)) + g(x(t))u(t), \quad t \geq 0,$$

with state $x \in \mathbb{R}^n$ and control input $u \in U \subset \mathbb{R}^m$.
Define the safe state set

$$\mathcal{X}^s := \{x \in \mathcal{X} \subset \mathbb{R}^n \mid V^s(x) \geq 0\}$$

for some sufficiently regular function

$$V^s : \mathcal{X} \subset \mathbb{R}^n \to \mathbb{R}.$$

The goal is to design a feedback control law $\kappa^s : \mathcal{X} \to \mathbb{R}^m$ such that $\mathcal{X}^s$ is forward invariant, i.e., any trajectory starting in $\mathcal{X}^s$ at $t = 0$ remains in $\mathcal{X}^s$ for all $t \geq 0$.

Let's now focus on the case of multi-robot collision avoidance.
Consider a team of N robots modelled as single integrators, i.e., for $i = 1, \dots, N$,

$$\dot{x}_i = u_i,$$

with $x_i \in \mathbb{R}^d$ the state (position) of robot i and $u_i \in U \subset \mathbb{R}^d$ the control input. Out goal is to keep the robots at a safe distance $\Delta > 0$ from each other. Consider the time derivative of $V^s(x(t))$ along the system trajectories

$$\begin{aligned}\frac{d}{dt} V^s(x(t)) &= \nabla V^s(x(t))^\top f(x(t)) + \sum_{h=1}^{m} \nabla V^s(x(t))^\top g_h(x(t)) u_h(t) \\ &= L_f V^s(x(t)) + L_g V^s(x(t)) u(t),\end{aligned}$$

For the single integrator system, we have

$$f(x(t)) = 0, \quad g(x(t)) = I,$$

thus

$$\frac{d}{dt} V^s(x(t)) = \nabla V^s(x(t))^\top u(t) = L_g V^s(x(t)) u(t)$$

We say that V^s is a Control Barrier Function if there exists a continuous, strictly increasing function $\gamma : \mathbb{R} \to \mathbb{R}$ with $\gamma(0) = 0$ such that

$$\sup_{u \in U} \left[L_f V^s(x) + L_g V^s(x) u + \gamma\big(V^s(x)\big) \right] \geq 0.$$

The above inequality induced by a Control Barrier Function is called a Control Barrier Certificate.
Note that the condition allows the time derivative of $V^s(x)$ to be negative of a certain amount depending on the distance from the boundary defined by $V^s(x) = 0$. The crucial aspect is that the function γ is strictly increasing: this ensures that, if the system starts in the unsafe region (i.e., $V^s(x) < 0$), the dynamics are driven toward the safe set $\{x \mid V^s(x) \geq 0\}$; once the state reaches the safe set (i.e., $V^s(x) \geq 0$), the condition guarantees forward invariance.
Going back to multi-robot collision avoidance problem, let's consider a pair-wise local CBF defined as

$$V^s_{i,j}(x_i, x_j) = \|x_i - x_j\|^2 - \Delta^2,$$

where Δ denotes the safety distance between robots i and j.
The gradients are

$$\nabla_1 V^s_{i,j}(x_i, x_j) = 2(x_i - x_j), \quad \nabla_2 V^s_{i,j}(x_i, x_j) = 2(x_j - x_i),$$

where $\nabla_1 V^s_{i,j}(\cdot, \cdot)$ represents the gradient of $V^s_{i,j}$ with respect to the first variable and $\nabla_2 V^s_{i,j}(\cdot, \cdot)$ represents the gradient of $V^s_{i,j}$ with respect to the second variable. Gradient w.r.t. all other states is zero.
Let $u \in \mathbb{R}^{dN}$ be the stacked input vector for all N robots.
The admissible control space for collision avoidance is obtained by intersecting all pair-wise control barrier certificate constraints:

$$\begin{aligned}\mathcal{U}^s(x) = \Big\{ & u \in \mathbb{R}^{dN} \mid -\nabla_1 V^s_{i,j}(x_i, x_j)^\top u_i - \nabla_2 V^s_{j,i}(x_j, x_i)^\top u_j - \gamma V^s_{i,j}(x_i, x_j) \leq 0, \\ & \forall i \in \{1, \dots, N\}, \quad \forall j \in \mathcal{N}_i \Big\}.\end{aligned}$$

where $\mathcal{N}_i$ includes all robots j that may collide with robot i.
It is worth noting that, in this case, the strictly increasing function γ is chosen as a linear function, $\gamma(x) = \gamma \cdot x$. The parameter $\gamma > 0$ governs the conservativeness of the resulting constraint: a larger value of γ allows $V^s(x)$ to attain more negative values even near the boundary $V^s(x) = 0$, thereby permitting the agents to pass closer to each other. Conversely, a smaller value of γ results in more conservative behavior, maintaining a greater safety margin.

Let's now define the safety controller for multi-robot collision avoidance in the distributed scenarios. As we already know, in distributed multi-robot scenarios, it is desirable to avoid a central coordinator, so we will need a more constrained problem formulation.
Let $u_{\text{ref}}(x) \in \mathbb{R}^d$ be a (possibly unsafe) reference input, e.g., from a higher-level controller. The safety controller is designed to be minimally invasive, i.e., to modify the reference input as little as possible.
So, the safe control policy $u_i = \kappa_i^s(x)$ can be designed as:

$$\kappa_i^s(x) = \arg\min_{u_i \in \mathbb{R}^d} \|u_i - u_i^{\text{ref}}\|^2$$

subject to

$$\begin{aligned} -\nabla_1 V_{i,j}^s(x_i, x_j)^\top u_i - \tfrac{1}{2}\gamma(V_{i,j}^s(x_i, x_j)) \leq 0, \\ \|u_i\| \leq u_i^{\max}, \quad \forall j \in \mathcal{N}_i. \end{aligned}$$

The $\frac{1}{2}$ is introduced to symmetrically split the degree of freedom between the two agents involved, i and j.

Our Implementation of Safety Controller

In this section, we explore a practical approach to safety control in distributed multi-robot scenarios, focusing on how agents locally detect and avoid collisions while maintaining coordinated behavior.

First of all, we need to define a function that identifies the physical neighbors of each agent. As an implementation choice, we compute this information in a semi-centralized manner: every time an agent needs to determine its next state, a function is called to compute the distances to all other agents and return only those that fall within a predefined radius r, thus identifying its actual neighbors. While this approach simplifies the implementation, it is not suitable for real-world distributed systems, where no centralized unit is available to compute global distances. However, it can be interpreted in a distributed context as each agent being equipped with sensors such as LiDAR that provide real-time distance measurements to nearby agents and obstacles.

Once the distances from neighbors are obtained, the agent computes its next state using a safety controller. This controller takes as input the desired control action computed by the high-level distributed aggregative algorithm. Since the system is modeled with single-integrator dynamics, the output of the aggregative controller is first processed by a proportional (P) controller, which generates a control signal compatible with the assumed dynamics. The safety controller then solves a constrained optimization problem to ensure safety and sends the resulting safe control input to the actuators. The outcome is also fed back to the high-level controller to maintain an accurate estimate of the agent's state, effectively closing the loop with a feedback mechanism.

In certain scenarios, particularly when the environment becomes too crowded, we observed that the optimization problem might become infeasible due to excessive constraints. To address this, we implemented a simple fallback mechanism: if the optimization problem cannot be solved, the agent stops. This behavior introduces a form of implicit priority or "right-of-way" mechanism, where agents wait until it is safe to move again, preventing collisions while still allowing eventual progress.

Once the agents have resolved their local interactions and reached a safe configuration, the safety controller allows the control input from the high-level controller to pass through to the actuators in a linear fashion. As a result, the system retains the same convergence properties as the original aggregative distributed algorithm in the absence of safety concerns.

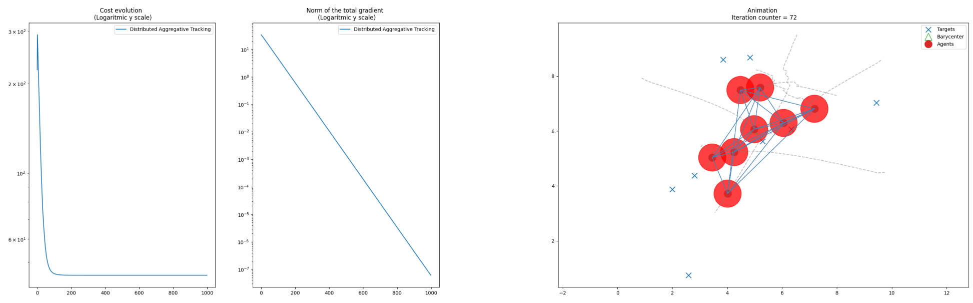


Figure 2.9: Safety controller performance in multi-robot collision avoidance. In the animation (image on the right), it is possible to see a red circle which represents the unsafe collision area of each agent.

This structure is particularly interesting because it clearly acts as a low-level safety mechanism: the higher-level controller is unaware of its presence, and yet the safety layer operates seamlessly in the background for each agent.

While the proposed approach provides an effective and practical method for implementing a safety controller in multi-robot systems, several limitations may arise in more complex scenarios. In particular, accounting for dynamic constraints and input bounds can lead to infeasibility of the optimization problem, making it impossible to guarantee safety under all conditions. Furthermore, when the objectives of multiple agents conflict with the safety barrier constraints, the system may experience deadlocks. These challenges highlight the need for more sophisticated control strategies that ensure both safety and feasibility in distributed multi-agent systems.

2.3 Aggregative Tracking Implemented with ROS2 Framework

Robot Operating System (ROS) is an open-source framework designed to support the development, integration, and management of software for robotic systems. It simplifies how different components of a robot such as sensors, actuators, and control modules communicate and coordinate, even when these components are distributed across multiple machines.
In ROS, each functional unit is implemented as an independent node, which performs specific tasks like reading sensor data, computing control commands, or logging information. Communication between nodes occurs through named data channels called topics. Nodes publish data to topics or subscribe to receive data from them. This publish-subscribe model supports anonymous and decoupled communication: nodes do not require direct knowledge of one another, but instead communicate indirectly through the shared topic namespace.
This modular and distributed architecture is particularly well-suited to multi-agent systems. In the context of distributed optimization, the publish-subscribe model naturally maps to a communication graph, where each agent exchanges data only with its immediate neighbors. For example, if an agent subscribes to a topic published by another agent, the publisher is considered its in-neighbor. Conversely, if it publishes data read by others, it is an out-neighbor. This mapping allows each agent to maintain private local data while interacting only with nearby agents, following the typical assumptions of distributed algorithms and enabling scalable, decentralized coordination.
Each agent begins by extracting its configuration parameters, such as the communication graph structure and adjacency matrix, from a predefined launch file. These parameters are not computed dynamically, but are instead initialized before execution, and can be considered centralized information that is locally known by each agent at runtime.
Once initialized, the agent operates in a fully distributed manner. It publishes two types of information: its internal state, defined by the variables z, s, and v; and auxiliary data used for logging and visualization. Simultaneously, it subscribes to the state topics of its neighbors, receiving only the information required by the algorithm. This setup ensures that each agent communicates only with its local neighborhood. Moreover, because the communication graph includes self-loops, each agent also subscribes to its own topic, processing its data through the same interface as data from others.
To maintain temporal consistency and synchronization, each agent uses a timer-based callback mechanism triggered every 0.1 seconds. During each callback, the agent checks whether it has received the latest state data from all of its neighbors. Once all the necessary data have been collected in a queue, the agent proceeds to update its estimate of z and the auxiliary variables s and v. This ensures that updates occur only when all dependencies are satisfied, providing a synchronized and consistent evolution of the distributed algorithm.
The distributed algorithm implemented is an aggregative tracking scheme, as described in Section 2.1. This algorithm enables each agent to update its local state based only on the data received from its immediate neighbors. As a result, the

overall system behaves in a fully distributed manner, where coordination emerges from local interactions without any central controller.

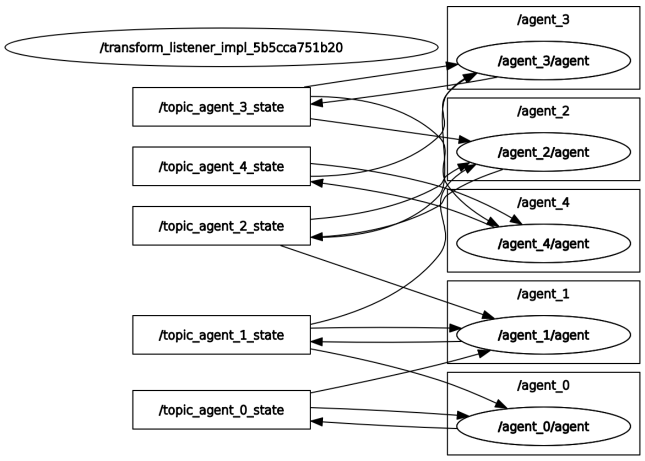


Figure 2.10: ROS graph structure showing the topic connections between agents, the subscriptions are determined using a Erdős–Rényi communication graph with edge probability $p = 0.65$. Note: the self-loops are modelled with self-subscriptions.

Simultaneously, a dedicated visualizer node is responsible for aggregating and rendering the system state in real time using RViz. RViz is a three-dimensional visualization tool designed for robotic applications. It can display robot models, sensor data, and environmental elements using a rich set of plugins.
The visualizer subscribes to two topics for each agent: one containing the agent's position and internal state (topic_agent_i_state), and the other containing performance metrics such as the cost and gradient norm (topic_agent_i_plot_info). Incoming messages are buffered in FIFO queues to enable synchronized data processing.
As with the agents, a timer callback is executed periodically. At each time step, the visualizer checks for synchronized data from all agents. Once data synchronization is ensured, the visualizer executes several tasks. It publishes the positions of all targets, computes and publishes the total cost and the norm of the aggregated gradient, and renders the positions of all agents along with their barycenter. These values are visualized in RViz to provide a real-time, intuitive representation of the system's evolution and performance.

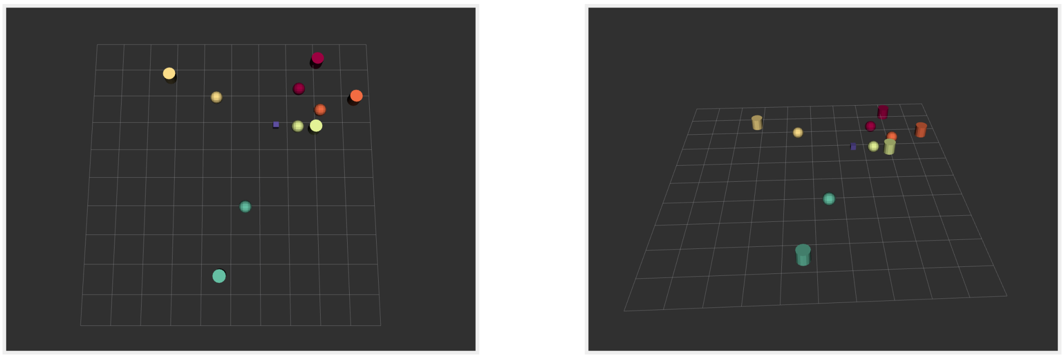


Figure 2.11: Visualization of agent positions in RViz.

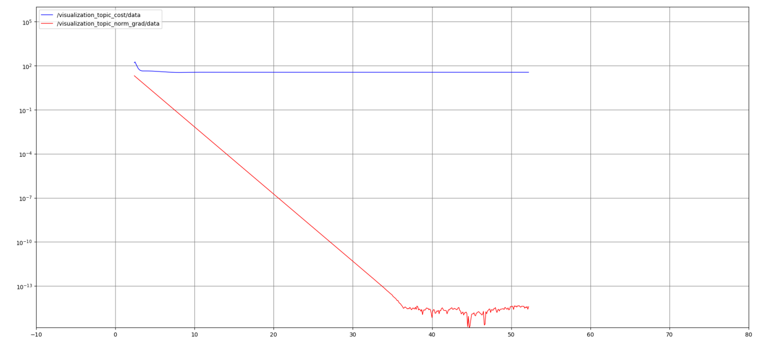


Figure 2.12: Convergence of the gradient norm over time

Figure 2.11 illustrates the spatial behavior of the agents as visualized in RViz. In addition, Figure 2.12 shows the convergence of the distributed optimization algorithm. The norm of the gradient progressively decreases to zero, confirming the linear convergence predicted by Distributed Aggregative Tracking theorem.

Conclusions

In this project, we explored distributed optimization algorithms as a flexible and powerful framework for the coordination of multi-agent systems. Through the implementation and testing of both consensus-based and aggregative optimization strategies, we demonstrated that global objectives can be effectively achieved through fully decentralized computations and neighbor-to-neighbor communication.

The results obtained across all scenarios were highly satisfactory. In the context of cooperative localization (Chapter 1), the Gradient Tracking algorithm proved to be robust and efficient, achieving accurate convergence even under noisy measurement conditions. In Task 2 (Chapter 2), we extended the analysis to aggregative optimization, where each agent balances its own objective with a global aggregative term. The proposed algorithm showed reliable convergence and good scalability, confirming its theoretical guarantees in both centralized and distributed configurations.
Moreover, we validated the approach in several specific scenarios, including the presence of VIP agents, increased cohesion among agents, and stronger individual target tracking. These experiments revealed how the proposed framework can flexibly adapt to heterogeneous behaviors and priorities among agents, while still maintaining stable and coordinated evolution.

To let the agents work in a more realistic scenario, it was integrated a low-level safety controller based on Control Barrier Functions. This mechanism ensures inter-agent collision avoidance in real time, while preserving the coordination induced by the high-level distributed algorithm. Although simple, this solution demonstrated good performance in simulation and provided a valuable safety layer.

While the results are promising, several directions for improvement remain. First, the current method for detecting physical neighbors is only semi-distributed. A fully distributed implementation would require each agent to autonomously sense nearby peers using local sensors, such as LiDAR or cameras. Additionally, the communication model assumes synchronous and reliable message exchange. Introducing asynchronous updates and handling communication delays or packet loss would enhance the algorithm's applicability in real-world networks.
Finally, while the safety controller worked well in moderately crowded environments, its scalability to highly dynamic or dense scenarios remains limited. Future work could include the integration of predictive or learning-based safety mechanisms, allowing for more adaptive and reactive behaviors while preserving formal guarantees on collision avoidance.

Bibliography

[1] Giuseppe Notarstefano, Ivano Notarnicola, and Andrea Camisa. Distributed Optimization for Smart Cyber-Physical Networks. 2019.

[2] Andrea Testa, Guido Carnevale, and Giuseppe Notarstefano. A tutorial on distributed optimization for cooperative robotics: From setups and algorithms to toolboxes and research directions. Proceedings of the IEEE, 113(1):40–65, 2025.

[3] Guido Carnevale, Andrea Camisa, and Giuseppe Notarstefano. Distributed online aggregative optimization for dynamic multirobot coordination. IEEE Transactions on Automatic Control, 68:3736–3743, 2021.

[4] Guido Carnevale, Nicola Mimmo, and Giuseppe Notarstefano. A unifying system theory framework for distributed optimization and games, 2025.