UNIVERSITÀ DI BOLOGNA

School of Engineering

Master Degree in Automation Engineering

Distributed Autonomous Systems

DAS Project

Professors:
Giuseppe Notarstefano
Ivano Notarnicola

Students:
Davide Zarabini
Vincenzo Lombardi
Vittorio Caputo

Academic year 2024/2025

Abstract

This project develops **distributed optimization algorithms** for multi-robot systems, focusing on two main tasks. **Task 1** solves a **distributed consensus optimization problem** for multi-robot target localization, where robots estimate target positions through noisy distance measurements using the **Gradient Tracking Algorithm**. **Task 2** addresses an **aggregative optimization problem** where robots balance between reaching private targets and maintaining swarm cohesion through the **Aggregative Tracking Method**.

The project combines theoretical problem formulation, algorithm development, and practical implementation using both Python and ROS 2. Simulations and real-time visualizations validate the effectiveness of the proposed methods, with performance metrics such as total gradient norm, total cost and estimation accuracy.

Contents

List of Figures

List of Tables

Introduction

Introduction

This report presents the development and implementation of **distributed optimization algorithms** for **multi-robot systems**, focusing on two key tasks: **distributed consensus optimization** for target localization and **aggregative optimization**. These tasks are explored within the broader context of enabling decentralized coordination in robotic networks, where agents must operate with limited information and communication capabilities.

In **Task 1**, we address the problem of **cooperative target localization**, where a team of robots estimates the positions of static targets using noisy distance measurements. The **Gradient Tracking Algorithm** is employed to solve this distributed consensus optimization problem, ensuring that robots collaboratively converge to an accurate estimate of the target positions. This task highlights fundamental challenges in multi-agent estimation under uncertainty, emphasizing the importance of efficient information fusion and convergence guarantees in distributed systems.

In **Task 2**, we tackle an **aggregative optimization problem**, where robots must balance between reaching their **private targets** and maintaining swarm **cohesion** to satisfy communication constraints. The **Aggregative Tracking Method** is implemented to solve this problem, enabling each robot to dynamically adjust its trajectory based on local cost functions and estimates of the team barycenter. This task showcases how trade-offs between individual objectives and global coordination can be managed through algorithmic design.

The project encompasses **theoretical formulation**, **algorithm design**, and practical **implementation** in both **Python** and **ROS 2**, providing a comprehensive evaluation of the proposed solutions. Simulation environments and real-time visualization tools are employed to monitor the behavior of the robotic team, analyze convergence trends, and assess algorithm performance. Key contributions include the development of efficient distributed algorithms, simulation-based validation, and real-time visualization of multi-robot coordination. The results demonstrate the effectiveness of the proposed methods in achieving distributed optimization goals, highlighting their potential for real-world applications in autonomous systems, such as surveillance, exploration, and environmental monitoring.

The remainder of this report is structured as follows:

- **Chapter 1** details Task 1, including problem formulation, the Gradient Tracking Algorithm, Python implementation and results.
- **Chapter 2** covers Task 2, including problem formulation, the Aggregative Tracking Method, Python and ROS 2 implementation and results.
- **Conclusions** summarize the findings and discuss future directions.

- **Appendix** contains basic definitions and properties of graph theory.

Contributions

This work makes the following key contributions to distributed multi-robot systems:

- **Implementation and validation** of two distributed optimization algorithms:
  - Gradient Tracking for consensus-based target localization (Task 1)
  - Aggregative Tracking for agents coordination (Task 2)
- **Comprehensive evaluation framework** including:
  - Cost function convergence analysis
  - Gradient norm reduction metrics
  - Consensus error measurements
  - Real-time visualization tools
- **Practical implementation** in both Python and ROS 2 environments
- **Theoretical-practical synthesis** connecting algorithm design with:
  - Graph theory fundamentals
  - Convergence Results

Chapter 1

Task 1: Multi-Robot Target Localization

1.1 Introduction

Task 1 focuses on distributed consensus optimization and is composed of two subtasks. **Task 1.1** addresses the **distributed consensus optimization** of a **quadratic function** as a preliminary step. **Task 1.2** involves **cooperative target localization**, where robots collaboratively estimate the positions of static targets based on noisy distance measurements. In particular, we consider a team of N robots with positions $p_i \in \mathbb{R}^d$ $(i = 1, \ldots, N)$ collaborates to estimate the positions of N_T static targets $z_\tau \in \mathbb{R}^d$ $(\tau = 1, \ldots, N_T)$. Each robot i obtains noisy distance measurements $d_{i\tau} \geq 0$ to target τ.

1.2 Distributed Consensus Optimization Problem

1.2.1 Problem Setting

The **Task 1.1** scenario can be formalized as follows:

$$\min_{z \in \mathbb{R}^d} \sum_{i=1}^{N} \ell_i(z),$$

where each local cost function $\ell_i(z)$ is a strongly convex quadratic function defined as:

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

with $Q_i \in \mathbb{R}^{d \times d}$ a symmetric **positive definit** matrix, and $b_i \in \mathbb{R}^d$ a vector. These properties ensure that each $\ell_i(z)$ has a **unique minimizer** and is differentiable, with gradient:

$$\nabla \ell_i(z) = Q_i z + b_i.$$

In practice, the matrix Q_i is constructed as:

$$Q_i = A_i^\top A_i + \lambda \cdot I_d,$$

where $A_i \in \mathbb{R}^{d \times d}$ is a randomly generated matrix, I_d is the $d \times d$ identity matrix, and $\lambda > 0$ is a small scalar added to ensure that Q_i is positive definite. This construction ensures that Q_i is symmetric and positive definite. The vector b_i is also randomly sampled, typically from a Gaussian distribution. Each robot i has access only to its own local cost function $\ell_i(z)$, and the goal is to collaboratively compute a common decision variable z that minimizes the global

objective.

On the other hand, the **Task 1.2** scenario addresses the cooperative localization of multiple static targets by a team of robots. It can be formalized as the following optimization problem:

$$\min_{z\in\mathbb{R}^{dN_T}} \sum_{i=1}^{N} \ell_i(z),$$

where each local cost function $\ell_i(z)$ models the discrepancy between the actual and predicted distance measurements of robot i to the targets:

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

and the local gradient is given by:

$$\nabla\ell_i(z) = [-4(d_{i1}^2 - \|z_1 - p_i\|^2)(z_1 - p_i), ..., -4(d_{iN_T}^2 - \|z_{N_T} - p_i\|^2)(z_{N_T} - p_i)]^T$$

The term $\|z_\tau - p_i\|^2$ in the cost represents the squared Euclidean distance between robot i and target τ, and each robot aims to minimize the squared error between the measured squared distance $d_{i\tau}^2$ and the predicted squared distance. Each robot i knows only its own position p_i and its local distance measurements $\{d_{i\tau}\}_{\tau=1}^{N_T}$ to the targets. Consequently, each robot can compute only its own local cost function $\ell_i(z)$ and does not have access to the full objective. Therefore, the robots must collaborate in a distributed manner to collectively estimate the global variable z, which encodes the positions of all targets.

1.2.2 Gradient Tracking Algorithm

In order to solve the described Distributed Consensus Optimization Problems in a distributed way, we can use the Gradient Tracking Method:

$$\begin{aligned} z_i^{k+1} &= \sum_{j\in\mathcal{N}_i} a_{ij} z_j^k - \alpha s_i^k \quad z_i^0 \in \mathbb{R}^d, \\ s_i^{k+1} &= \sum_{j\in\mathcal{N}_i} a_{ij} s_j^k + \nabla\ell_i(z_i^{k+1}) - \nabla\ell_i(z_i^k) \quad s_i^0 = \nabla\ell_i(z_i^0). \end{aligned}$$

where:

- z_i^k is the solution estimate of agent i at iteration k,
- s_i^k is the gradient tracking variable of agent i at iteration k, estimate of the true total gradient $\frac{1}{N}\sum_{h=1}^{N}\nabla\ell_h(z_h^k)$,
- $\mathcal{N}_i$ is the set of neighbors of agent i in the **strongly connected** (or connected if undirected) and **aperiodic** communication graph $\mathcal{G}$,
- a_{ij} are the weights of the **doubly stochastic** adjacency matrix A,
- $\alpha > 0$ is the step-size,
- $\nabla\ell_i$ is the gradient of the local cost function ℓ_i.

The Gradient Tracking (GT) can be considered an improvement of the Distributed Gradient Method (DG). In fact, to update z_i^k, an average of neighbor states is considered as in the DG, while the update direction, that in DG is always a wrong direction with respect the true total

gradient direction, in GT converges to the true total gradient direction. Each agent has a local estimates s_i^k of the true gradient direction that is updated according to dynamic average consensus protocol that takes into account the average of neighbor estimates s_j^k and an innovation term that considers the variation in time of the local gradient. Important to notice that the inizialization for s_i is not arbitrary and it's important to set $s_i^0 = \nabla \ell_i(z_i^0)$ for convergence.

Theorem 1.2.1 (Convergence of Gradient Tracking Method)**.** *Consider the Gradient Tracking Method applied to problem* $\min_z \sum_{i=1}^N \ell_i(z)$ *under the following assumptions:*

- *Communication graph* $\mathcal{G}$ *is strongly connected and aperiodic*
- *Weight matrix* $A = [a_{ij}]$ *is doubly stochastic with* $a_{ii} > 0$
- *Each* $\ell_i : \mathbb{R}^d \to \mathbb{R}$ *is* μ*-strongly convex*
- *Each* $\nabla \ell_i$ *is L-Lipschitz continuous*

Then, there exists $\alpha^* > 0$ *such that for all step-sizes* $\alpha \in (0, \alpha^*)$*, the algorithm achieves:*

1. *Exact convergence to the optimal solution:*

$$\lim_{k \to \infty} \|z_i^k - z^*\| = 0 \quad \forall i = 1, \dots, N$$

2. *Linear convergence rate:*

$$\|z_i^k - z^*\| \leq \rho^k \|z_i^0 - z^*\| \quad \forall i, \exists \rho \in (0, 1)$$

where z^* *is the unique minimizer of* $\sum_{i=1}^N \ell_i(z)$*.*

1.3 Python Implementation

The Python implementation simulates a multi-robot target localization system performing distributed optimization using a gradient tracking algorithm.

The main elements are described below:

TLSimulation Class

- **Primary Function:** It creates the setting for task 1.2 target localization
- **Key Features:**
  - **Robots and targets generation:** It creates random positions p_i for each of the N agents according to uniform distribution and it generates noise distances $d_{i\tau}$ between each agent i and target τ considering a Gaussian noise added to the true distance
  - **Communication Graph Generation:** Creates **undirected**, **connected** and **aperiodic** graphs (RGG, Erdős–Rényi, cycle, star, path, or complete) with **Metropolis-Hastings** weights for **doubly stochastic** weighted adjacency matrices to guarantee convergence. In particular, the Random Geometric Graph (RGG) is a graph based on distances among agents. Agent j is a neighbor of agent i only if the distance between them is less or equal than a certain defined communication radius
  - **Local Cost Functions:** Provides individual cost functions for each agent based on squared error between measured and estimated distances and returns also the related gradient (there's also another method to generate random quadratic cost functions for task 1.1 as described in subsection 1.2.1)

– **Initialization:** Generates initial guesses for the local estimates z_i of target positions for each agent. Considering that $z_i = [z_{i1}, ..., z_{iN_T}]^T$, all elements of zi are initialized to the position p_i of agent i

Gradient Tracking Method
It implements the Gradient Tracking Method with a certain number of defined maximum iterations and a stopping criterion based on the norm of the total gradient (sum of local gradients) to be below a defined threshold (10^{-7}) with proper initialization.

Visualization

- **Total Cost:** Plots the evolution of the sum of all local cost functions over iterations. Uses logarithmic scale for task 1.2 and natural scale for task 1.1 to visualize convergence behavior.
- **Total Gradient Norm:** Displays the 2-norm of the sum of all local gradients in logarithmic scale, providing insight into the convergence rate and stopping criterion satisfaction.
- **Total Consensus Error:** Shows the sum of distances from each agent's state to the average state across all agents in logarithmic scale, measuring how well agents reach consensus.
- **Estimates Evolution:** For each target and each coordinate dimension, plots individual agent estimates over time, average estimate evolution, and true target position to visualize convergence to a consensual optimal solution.
- **Communication Graph:** Renders the network topology using NetworkX with nodes representing agents, edges showing communication links for visualization of the distributed system structure.
- **Spatial Map (2D/3D):** In task 1.2, it creates a spatial visualization showing agent locations, true target positions, estimated target positions, and error vectors connecting true and estimated positions.

1.4 Results

In this section are reported the results of a set of simulations for task 1.1 and 1.2 with different communication graphs, number of agents, number of targets etc. to showcase the effectiveness of the implementation. For all simulations, the main plots, including total cost, gradient norm, and consensus error over iterations are presented. Additional plots are provided selectively for certain simulations to maintain brevity and avoid redundancy. In the table below, the main parameters of each simulation are summarized:

Simulation	N (Agents)	N_T (Targets)	d (Dimension)	Graph Type
Sim 1 (task 1.1)	12	-	2	Erdős–Rényi
Sim 2 (task 1.1)	17	-	3	Path
Sim 3 (task 1.2)	15	3	2	Cycle
Sim 4 (task 1.2)	17	3	3	Star
Sim 5 (task 1.2)	22	5	3	RGG

Table 1.1: Simulation parameters for task 1

Note that d is the dimension of the decision variable z in task 1.1, while it's the dimension of each target position z_τ.

1.4.1 Task 1.1 Plots

Simulation 1

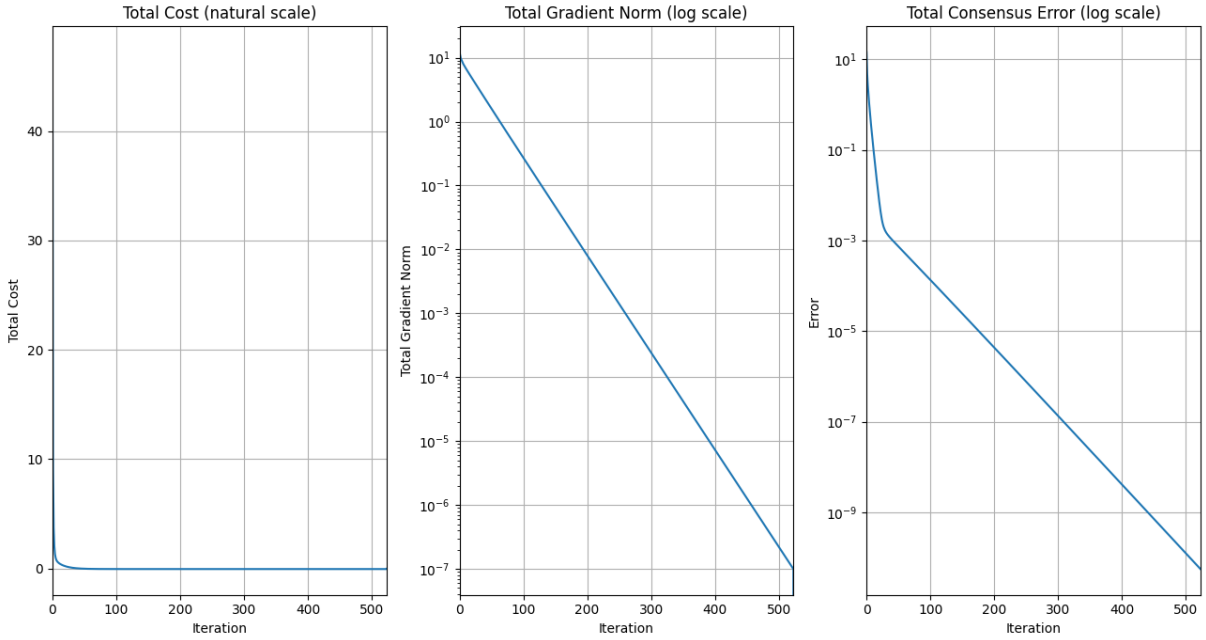

Figure 1.1: Total Cost, Gradient Norm, Consensus Error over Iterations (sim 1)

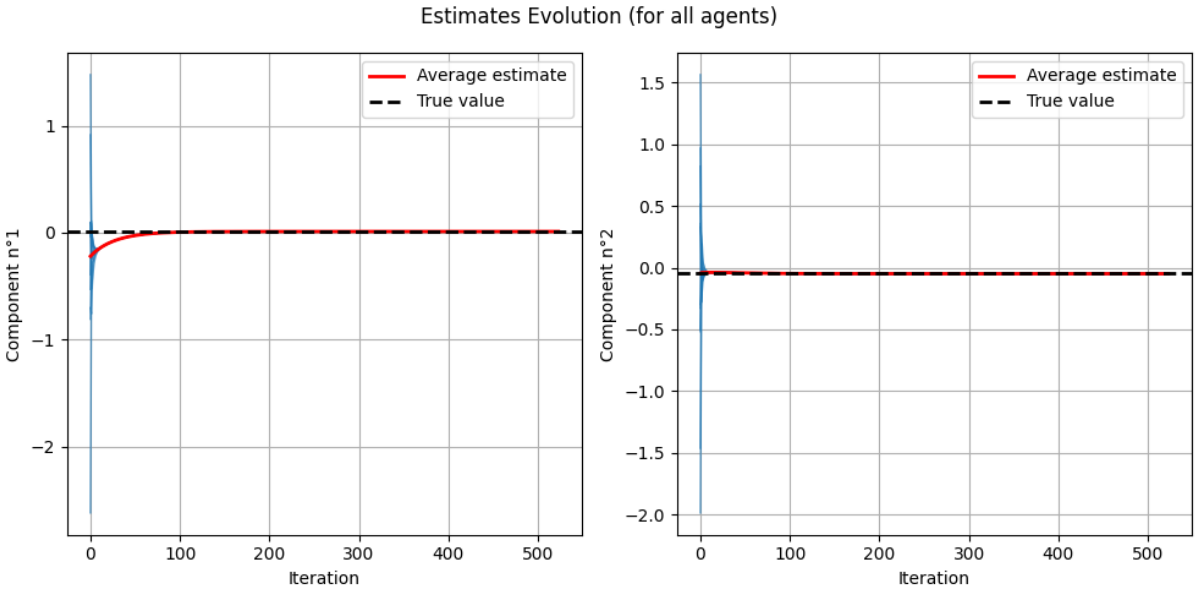

Figure 1.2: Estimates Evolution over Iterations (sim 1)

Simulation 2

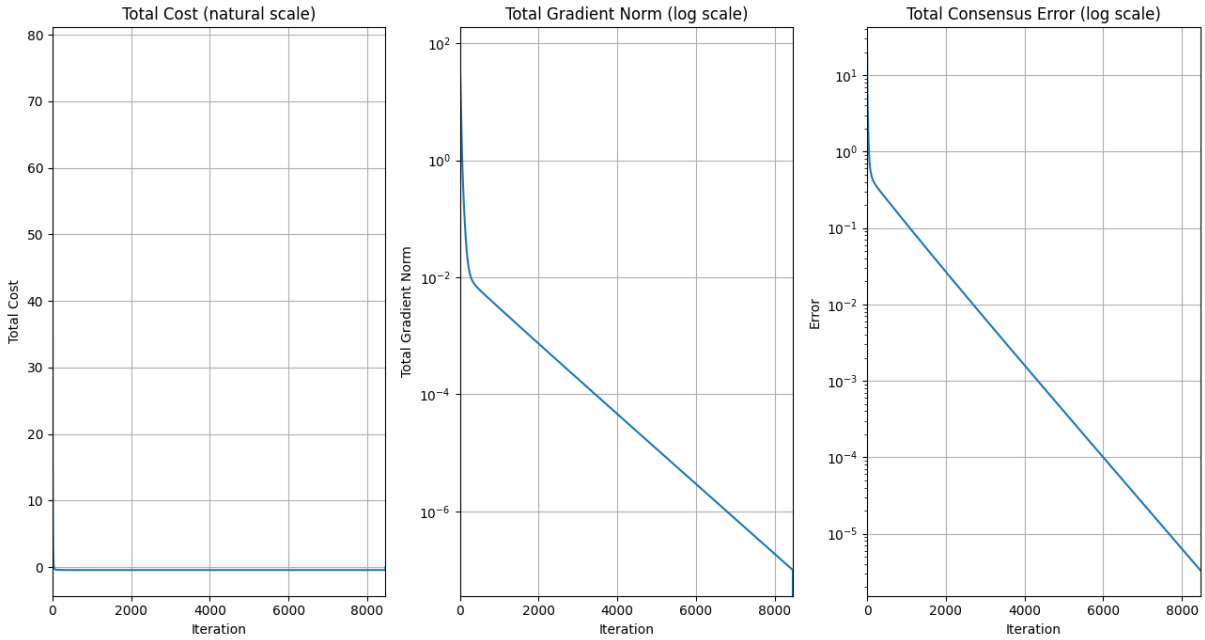

Figure 1.3: Total Cost, Gradient Norm, Consensus Error over Iterations (sim 2)

1.4.2 Task 1.2 Plots

Simulation 3

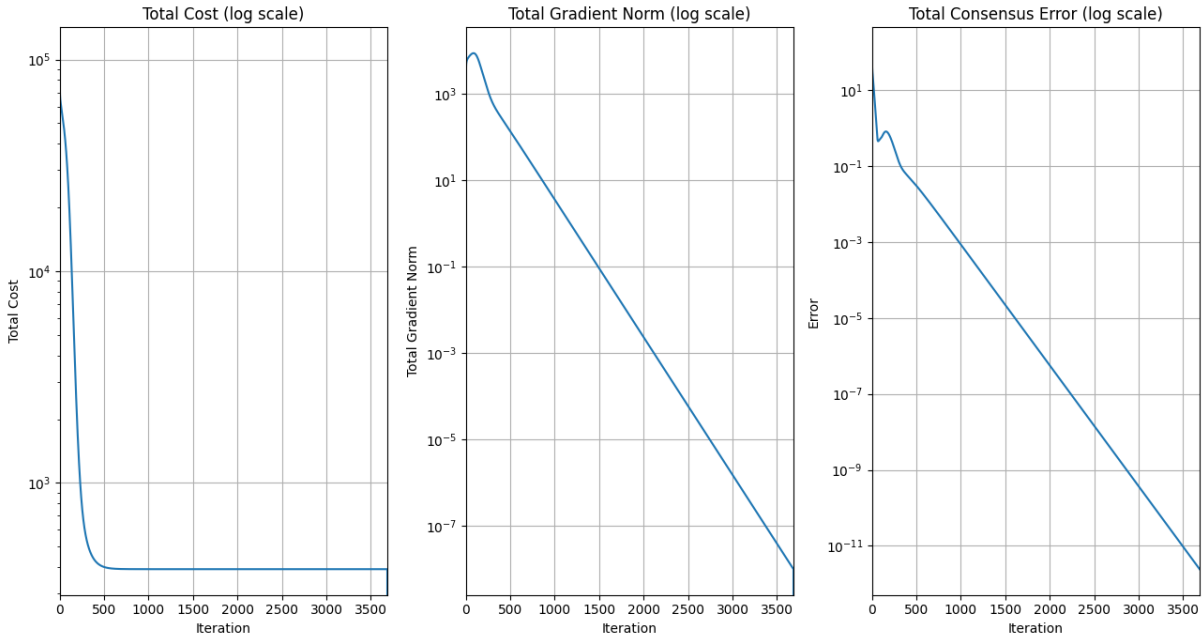

Figure 1.4: Total Cost, Gradient Norm, Consensus Error over Iterations (sim 3)

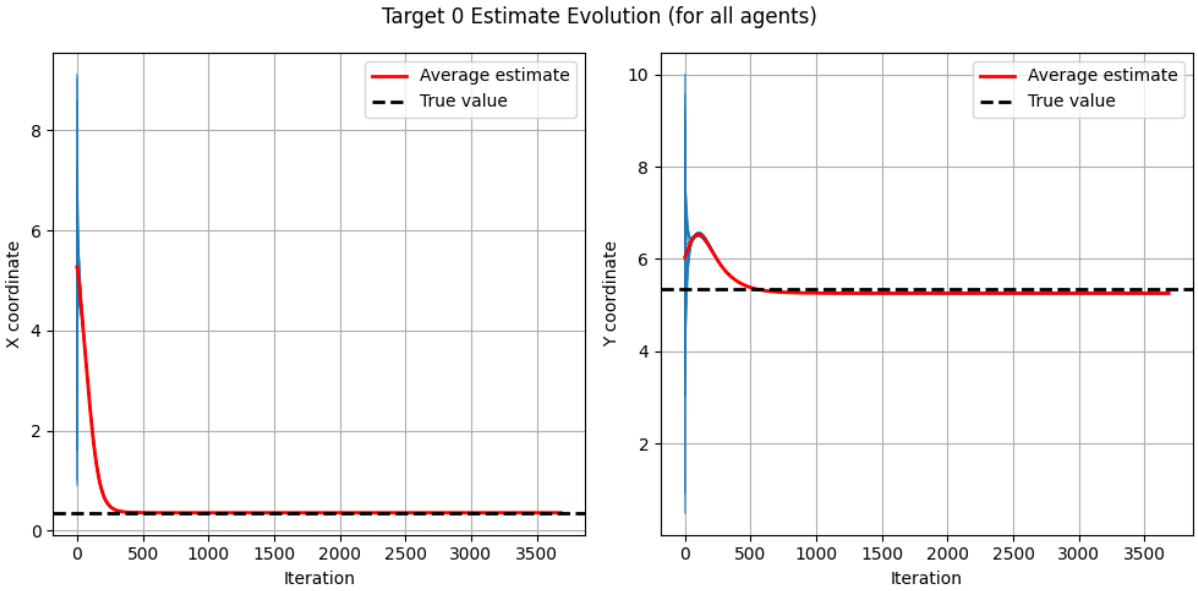

Figure 1.5: Position Estimate of Target 0 over Iterations (sim 3)

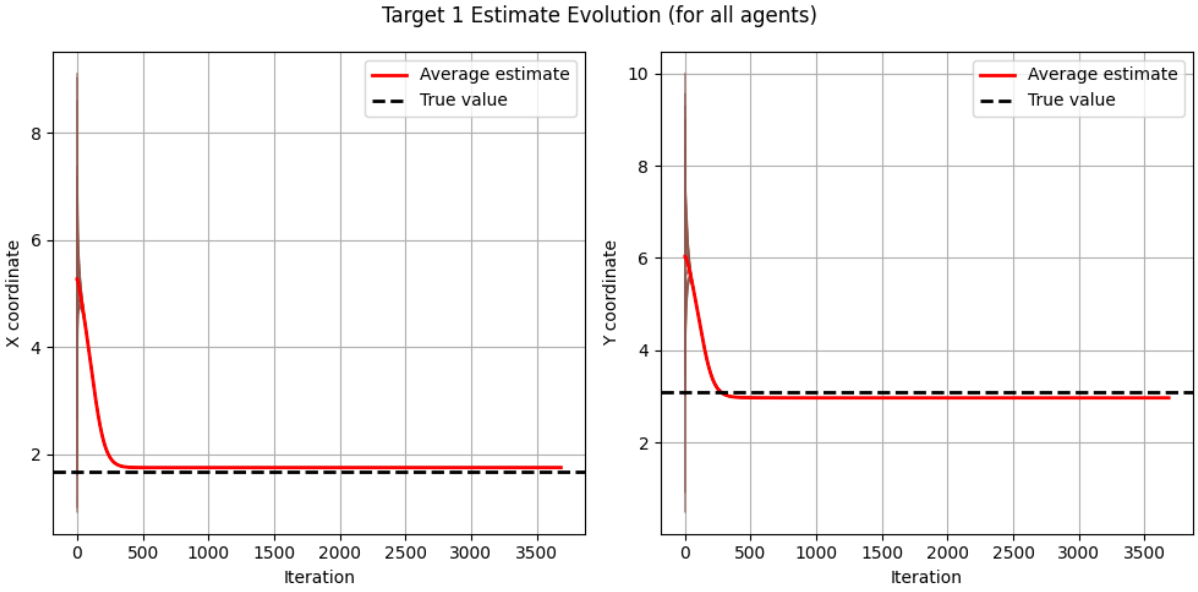

Figure 1.6: Position Estimate of Target 1 over Iterations (sim 3)

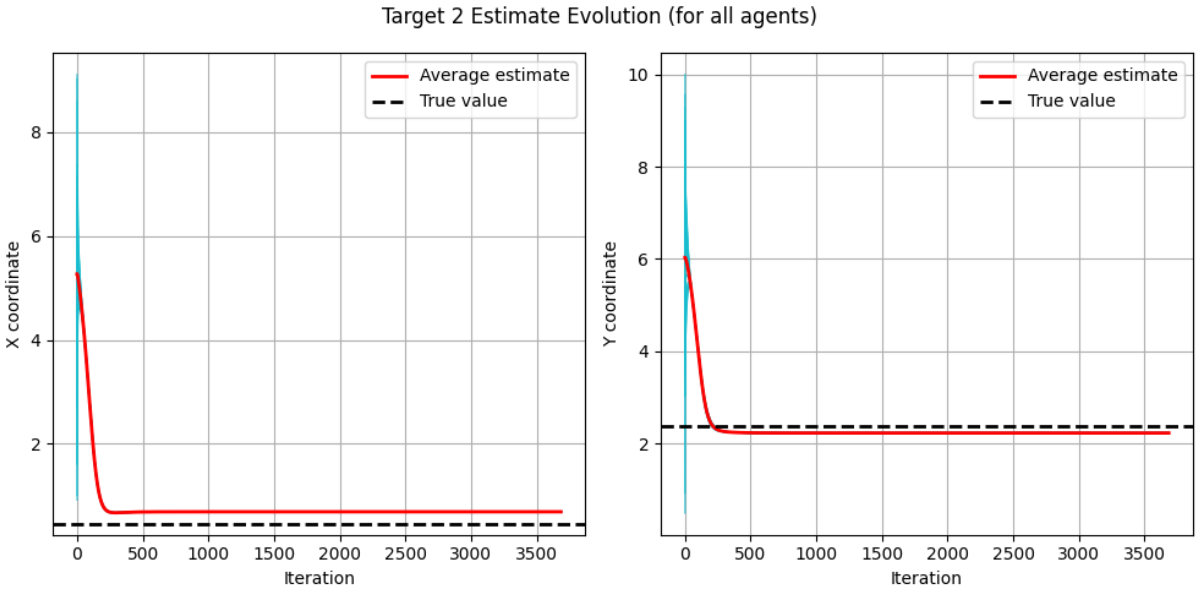

Figure 1.7: Position Estimate of Target 2 over Iterations (sim 3)

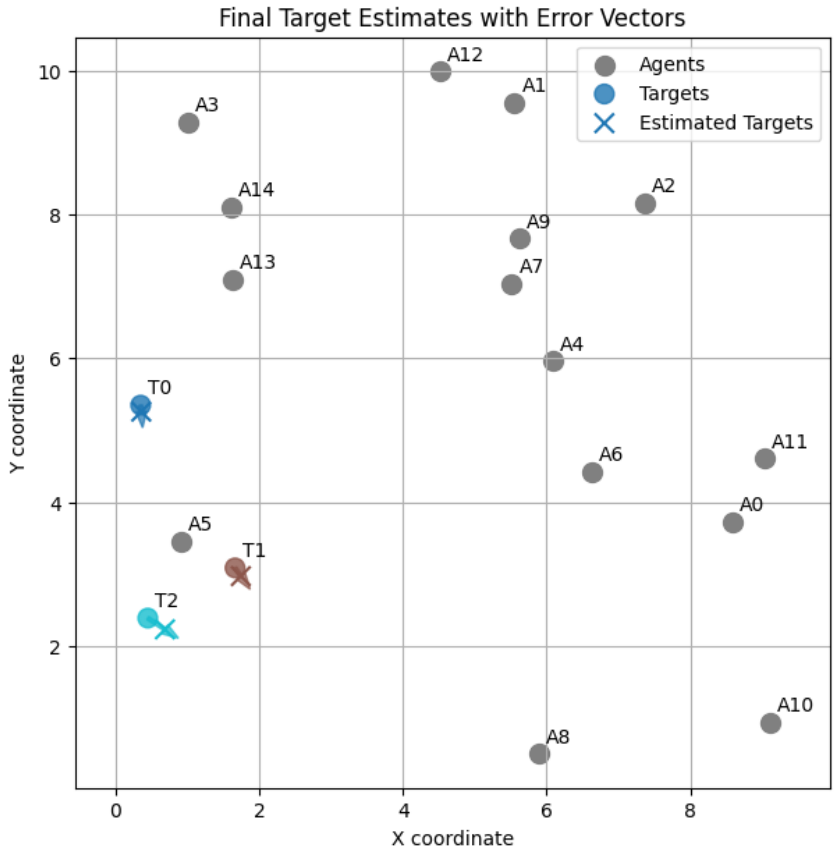

Figure 1.8: Spatial Plot of Agents and Targets (sim 3)

Simulation 4

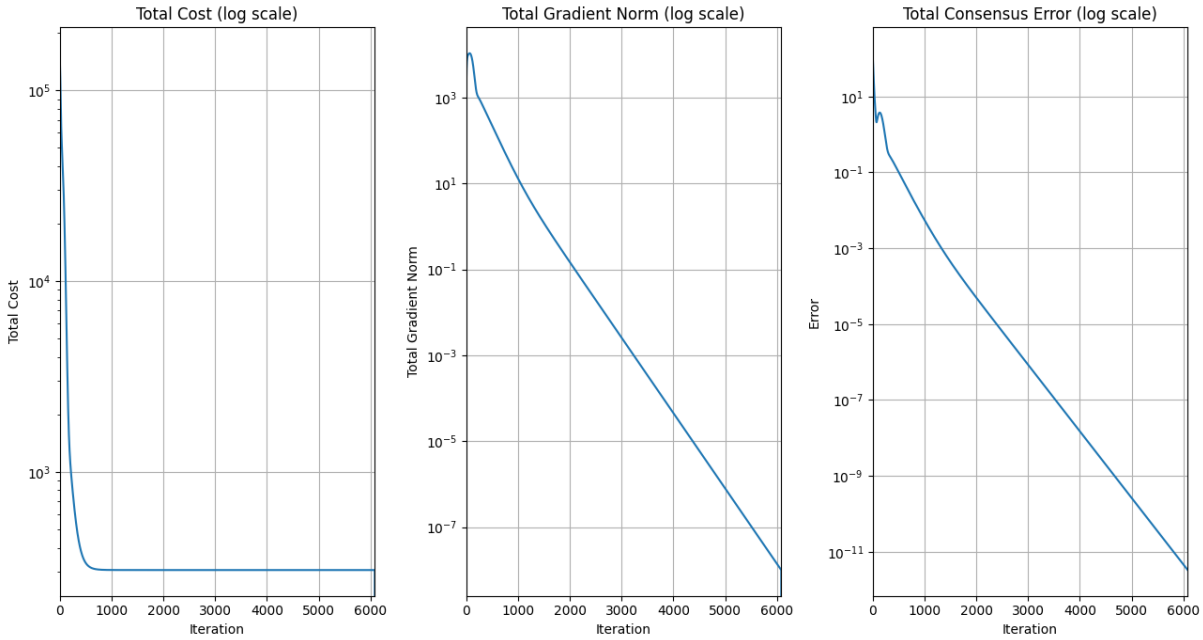

Figure 1.9: Total Cost, Gradient Norm, Consensus Error over Iterations (sim 4)

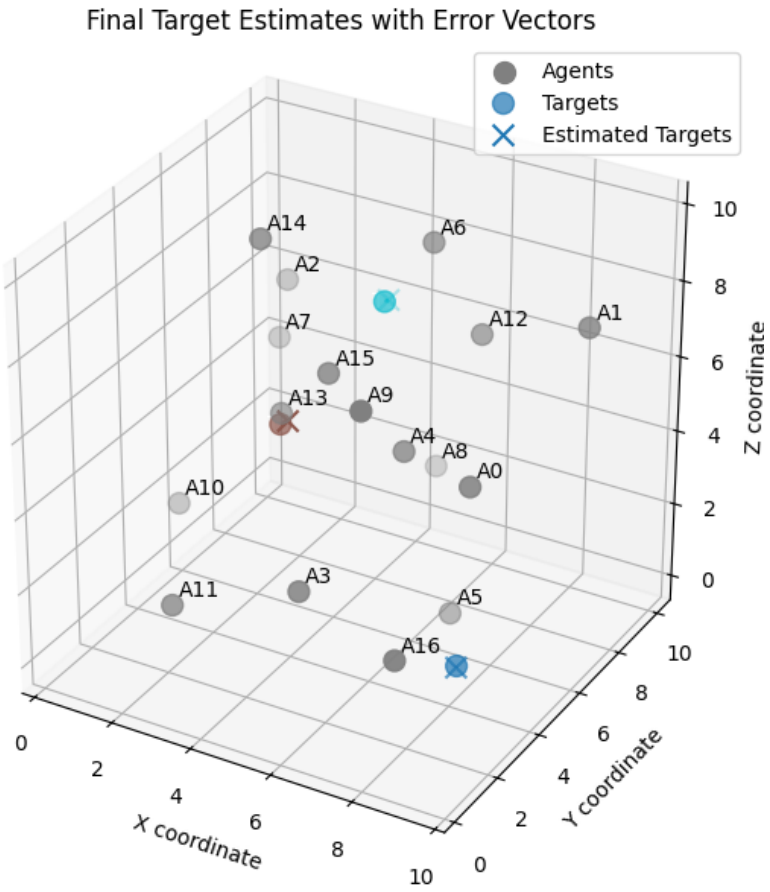

Figure 1.10: Spatial Plot of Agents and Targets (sim 4)

Simulation 5

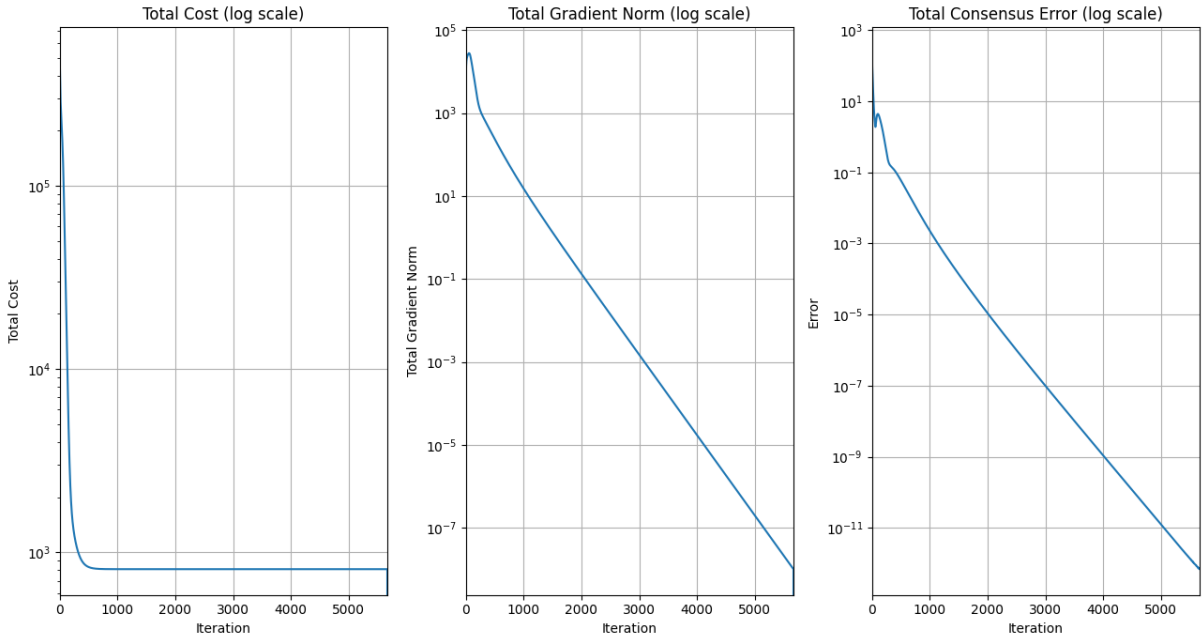

Figure 1.11: Total Cost, Gradient Norm, Consensus Error over Iterations (sim 5)

Chapter 2

Task 2: Aggregative Optimization for Multi-Robot Systems

2.1 Introduction

In **Task 2**, we consider a setting involving a team of N robots operating in a two-dimensional environment ($d = 2$). Each robot i is characterized by a position $z_i^k \in \mathbb{R}^2$ at iteration k. The collective state of the team is represented by the stacked vector $z^k = \text{col}(z_1^k, \dots, z_N^k) \in \mathbb{R}^{2N}$.

The goal of this task is to implement a **distributed control algorithm** that allows to:

- Enable each robot to remain as **close as possible** to its own **private target**.
- Keep the **fleet tight** due to communication constraints.

This chapter covers the following aspects in detail: **algorithm design**, **cost function formulation**, **Python implementation**, **ROS 2 integration**, and **experimental results**.

2.2 Aggregative Optimization Problem

2.2.1 Problem Setting

The task 2 scenario can be formalized as an **aggregative optimization problem** of the form:

$$\min_{z \in \mathbb{R}^{2N}} \sum_{i=1}^{N} \ell_i(z_i, \sigma(z)), \quad \text{where} \quad \sigma(z) = \frac{1}{N} \sum_{i=1}^{N} \phi_i(z_i).$$

Here, $\ell_i : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}$ is the **local cost function** of robot i, and $\phi_i : \mathbb{R}^2 \to \mathbb{R}^2$ defines its contribution to the **aggregative variable** $\sigma(z)$, which represents the **team's barycenter** (i.e., $\phi_i(z_i) = z_i$).

2.2.2 Local Cost Function Design

In order to accomplish the task 2, we considered the following **local cost function**:

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

The cost function consists of two terms: the first term encourages each robot to remain as **close as possible** to its own **private target**, while the second promotes robot **aggregation** toward their **barycenter** to keep the **feet tight**. A **trade-off** parameter γ_i is introduced to balance the influence of individual goal tracking against the desire to maintain cohesion.

2.2.3 Aggregative Tracking Algorithm

In order to solve the described Aggregative Optimization Problem in a distributed way, we can use the **Aggregative Tracking Method**, that in brief is an extension of the Centralized Gradient Method through **Dynamic Average Consensus**:

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

where:

- $z_i^k \in \mathbb{R}^{n_i}$ is the local state estimate of agent i at iteration k,
- $s_i^k \in \mathbb{R}^d$ is the local estimate of the aggregative variable $\sigma(z^k) = \frac{1}{N} \sum_{j=1}^N \phi_j(z_j^k)$ with $\phi_i : \mathbb{R}^{n_i} \to \mathbb{R}^d$,
- $v_i^k \in \mathbb{R}^d$ is the local estimate of the average gradient $\frac{1}{N} \sum_{j=1}^N \nabla_2 \ell_j(z_j^k, \sigma(z^k))$,
- $\mathcal{N}_i$ is the set of neighbors of agent i in the **strongly connected** (or connected if undirected) and **aperiodic** communication graph $\mathcal{G}$,
- a_{ij} are the weights of the **doubly stochastic** adjacency matrix A,
- $\alpha > 0$ is the constant step-size,
- $\nabla_1 \ell_i$ is the gradient of ℓ_i with respect to its first argument (z_i),
- $\nabla_2 \ell_i$ is the gradient of ℓ_i with respect to its second argument (σ),
- $\nabla \phi_i$ is the gradient of ϕ_i.

The Aggregative Tracking method is a distributed implementation of the Centralized Gradient Method (component-wise):

$$
z_i^{k+1} = z_i^k - \alpha \left[\nabla \ell(z^k) \right]_i
$$

where:

$$
[\nabla \ell(z^k)]_i = \nabla_1 \ell_i(z_i^k, \sigma(z^k)) + \frac{1}{N} \nabla \phi_i(z_i^k) \sum_{j=1}^N \nabla_2 \ell_j(z_j^k, \sigma(z^k)),
$$

where, according to the chosen cost function, the gradients are:

- $\nabla_1 \ell_i(z_i^k, \sigma(z^k)) = 2\gamma_i(z_i^k - r_i) + 2(z_i^k - \sigma(z^k))$
- $\nabla_2 \ell_j(z_j^k, \sigma(z^k)) = 2(\sigma(z^k) - z_j^k)$
- $\nabla \phi_i(z_i) = \mathbf{I}_{n_i}$

The Aggregative Tracking Method simply replaces the global terms $\sigma(z^k)$ and $\frac{1}{N} \sum_{j=1}^N \nabla_2 \ell_j(z_j^k, \sigma(z^k))$ with the respective estimates s_i^k and v_i^k through dynamic average consensus: average of neighbor estimates plus an innovation term. Moreover, the initializations $s_i^0 = \phi_i(z_i^0)$ and $v_i^0 = \nabla_2 \ell_i(z_i^0, s_i^0)$ are not arbitrary and necessary for convergence.

It's important to remark that in a distributed context, each agent i:

- knows only ℓ_i and ϕ_i
- maintains an estimate z_i^k of z_i^*
- maintains an estimate s_i^k of $\sigma(z^k) = \frac{1}{N}\sum_{j=1}^N \phi_j(z_j^k)$
- maintains an estimate v_i^k of $\frac{1}{N}\sum_{j=1}^N \nabla_2 \ell_j(z_j^k, \sigma(z^k))$

Theorem 2.2.1 (Convergence of Aggregative Tracking Method)**.** *Consider the Aggregative Tracking Method applied to problem* $\min_z \sum_{i=1}^N \ell_i(z_i, \sigma(z))$ *with* $\sigma(z) = \frac{1}{N}\sum_{j=1}^N \phi_j(z_j)$*. Under the following assumptions:*

- *Communication graph* $\mathcal{G}$ *is strongly connected and aperiodic*
- *Weight matrix* $A = [a_{ij}]$ *is doubly stochastic*
- *Global cost* $\sum_{i=1}^N \ell_i(\cdot, \sigma(\cdot))$ *is strongly convex*
- $\phi_i(\cdot)$ *are differentiable with Lipschitz continuous gradients*
- *Gradients* $\nabla_1 \ell_i$*,* $\nabla_2 \ell_i$*, and* $\nabla \phi_i \nabla_2 \ell_i$ *are Lipschitz continuous*

Then, there exists $\alpha^* > 0$ *such that for all step-sizes* $\alpha \in (0, \alpha^*)$*, the algorithm achieves:*

$$\lim_{k\to\infty} \|z_i^k - z_i^*\| = 0 \quad \textit{(linear rate)}$$

for all agents $i = 1, \ldots, N$*, where* $z^* = (z_1^*, ..., z_N^*)$ *is the optimal solution.*

2.3 Python Implementation

The Python implementation simulates a multi-robot system performing distributed optimization using an aggregative tracking algorithm.

The main elements are described below:

Agent Class
Represents individual robot with some attributes, such as:

- Current state position (z_i^k) and history across iterations
- Private target positions (r_i)
- Current barycenter estimate (s_i^k) and $\frac{1}{N}\sum_{j=1}^N \nabla_2 \ell_j(z_j^k, \sigma(z^k))$ estimate (v_i^k)
- Tradeoff parameter (γ_i) for local cost function

AggregativeOptimizer Class
Implements the distributed optimization algorithm with:

- **Communication Graph Generation**: Creates **undirected**, **connected** and **aperiodic** graphs (Erdős–Rényi, cycle, star, path, or complete) with **Metropolis-Hastings** weights for **doubly stochastic** weighted adjacency matrices to guarantee convergence
- **Aggregative Tracking Algorithm**: Allows each agent i to converge to component z_i^* of an optimum $z^* = [z_1^*, ..., z_N^*]^\top$ of the total cost function $\sum_{i=1}^N \ell_i(z_i, \sigma(z))$. In particular, we considered an early **stopping criterion** of the method based on a threshold (10^{-7}) applied to the **total gradient norm**. Additionally, Each agent's **initial state** is sampled **uniformly at random** from a square area, as:

$$z_i^0 \sim \mathcal{U}(0, \text{area\_size})^2$$

where:

- $z_i^0 \in \mathbb{R}^2$ is the initial position of agent i,
- $\mathcal{U}(0, \text{area\_size})$ denotes a continuous uniform distribution over the interval $[0, \text{area\_size}]$

The same was done for generating target positions.

Visualization

- **Total Cost**: Semi-logarithmic plot showing the sum of all agents' local cost functions $\sum_i \ell_i(z_i^k, \sigma(z^k))$ over iterations
- **Total Gradient Norm**: Defined as the Euclidean norm of the gradient of the global objective $\ell(z)$, where the full gradient is constructed by stacking the N local components:

$$\nabla \ell(z^k) = \text{col}\left([\nabla \ell(z^k)]_1, \ldots, [\nabla \ell(z^k)]_N\right)$$

Each local component is given by:

$$[\nabla \ell(z^k)]_i = \nabla_1 \ell_i(z_i^k, \sigma(z^k)) + \frac{1}{N} \nabla \phi_i(z_i^k) \sum_{j=1}^{N} \nabla_2 \ell_j(z_j^k, \sigma(z^k)),$$

and the total gradient norm is:

$$\|\nabla \ell(z^k)\| = \left\|\text{col}\left([\nabla \ell(z^k)]_1, \ldots, [\nabla \ell(z^k)]_N\right)\right\|.$$

- **Total Sigma Estimation Error**: Semi-logarithmic plot showing the total estimation error $\sum_i \|s_i^k - \sigma(z^k)\|$ across all agents over iterations
- **Sigma Component Evolution Plots**: The implementation generates separate plots for each spatial dimension (X and Y components) of the barycenter estimate of each agent to show consensus
- **Animation of the multi-robot system**: Animation of agents across iterations with private targets and barycenter visualization

2.4 ROS 2 Implementation

The **ROS 2** implementation follows the structure of the previous pure Python implementation.

The main elements are described below:

Launch File (`aggregative_optimization_launch.py`)

- **Primary Function:** Configures the distributed multi-agent optimization system
- **Key Features:**
  - Contains `Agent_Config` and `AggregativeSetup` classes for system configuration
  - Configures **communication topology** (Erdős–Rényi, cycle, star, path, complete graphs)

- Generates **Metropolis-Hastings** weights for **doubly stochastic** adjacency matrix
- Initializes agent parameters (positions, targets, neighbors, weights etc.)
- Launches multiple agent nodes, visualization nodes and **RViz** simultaneously

Agent Node (`the_agent.py`)

- **Primary Function:** Implements individual optimization agent that communicate in a distributed network
- **Key Features:**
  - Performs **aggregative tracking** algorithm with proper synchronization with other agents
  - It is characterized by some attributes such as current state position z_i^k, private target position r_i, current σ estimate s_i^k
  - Publishes optimization data on its own topic (`/topic_{agent_id}`)
  - Subscribes to neighbor agents' topics for the distributed optimization algorithm
  - Sends visualization data to `/visualization_data` topic
- **Agents' Synchronization**
  The synchronization among agents in the distributed optimization algorithm works as follows: agent i checks whether all messages from its neighbors related to iteration $k-1$ (required for the optimization step) have been received. If all messages are available, it proceeds with the optimization step. Otherwise, it exits the timer callback and increments a waiting counter. If the waiting counter exceeds a predefined threshold, an exception or warning is triggered to prevent the agent from continuing with outdated information, which could negatively affect convergence.

Visualization Node (`visualization_node.py`)

- **Primary Function:** Collects and processes data from all agents for plotting
- **Key Features:**
  - Subscribes to `/visualization_data` topic
  - Generates matplotlib plots total cost, total gradient norm, total sigma estimation error over iterations and an animation of the multi-robot system

Generic Visualizer (`the_visualizer.py`)

- **Primary Function:** Provides real-time markers to RViz for animation
- **Key Features:**
  - Subscribes to `/visualization_data` topic
  - Publishes RViz markers on `/visualization_markers` topic

Rviz

- **Primary Function:** Creates a real-time animation of the multi-robot system
- **Key Features:**
  - Subscribes to `/visualization_markers` topic
  - Displays agents as red spheres, targets as blue spheres, and the barycenter as a blue cube

2.5 Results

In this section are reported the results of a set of simulations for task 2 with different communication graphs, number of agents, γ_i etc. to showcase the effectiveness of the implementation. For all simulations, the main plots, including total cost, gradient norm and sigma estimation error over iterations are presented.

Simulation	N (Agents)	γ_i	Graph Type
Sim 1	8	1	Cycle
Sim 2	11	0.6	Erdős–Rényi

Table 2.1: Simulation parameters for task 2

2.5.1 Task 2.1 Plots

Simulation 1

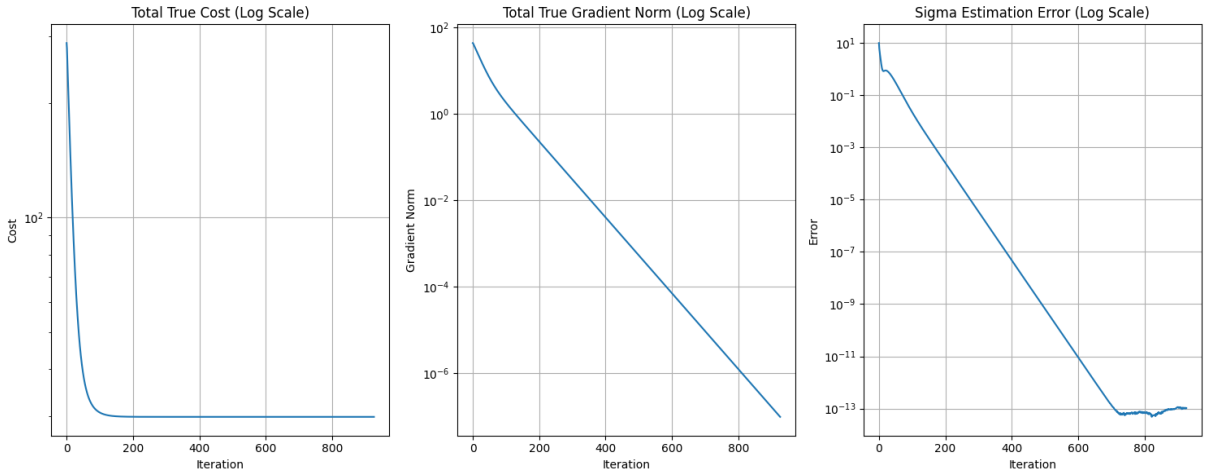

Figure 2.1: Total Cost, Gradient Norm, Sigma Estimation Error over Iterations (sim 1)

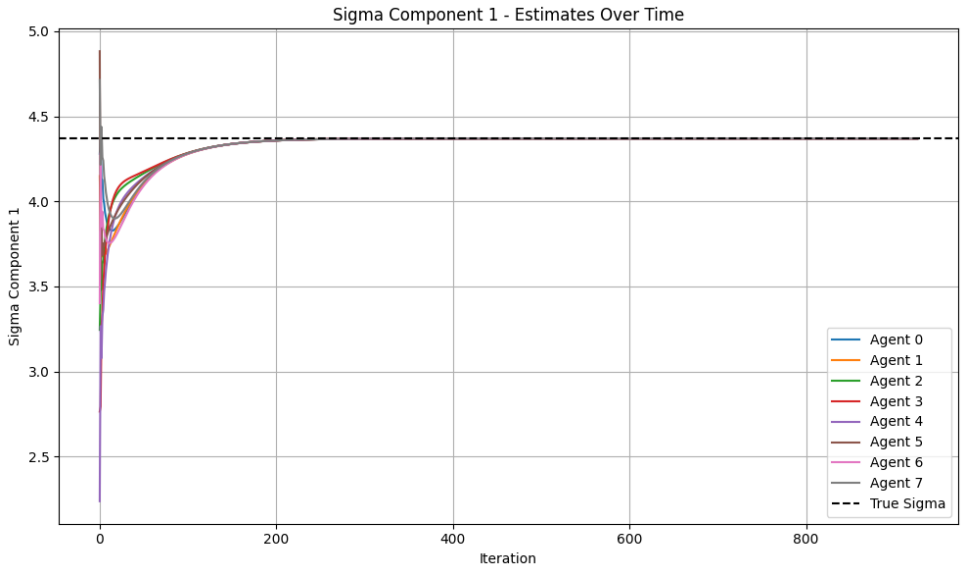

Figure 2.2: Barycenter Estimate Component 1 (sim 1)

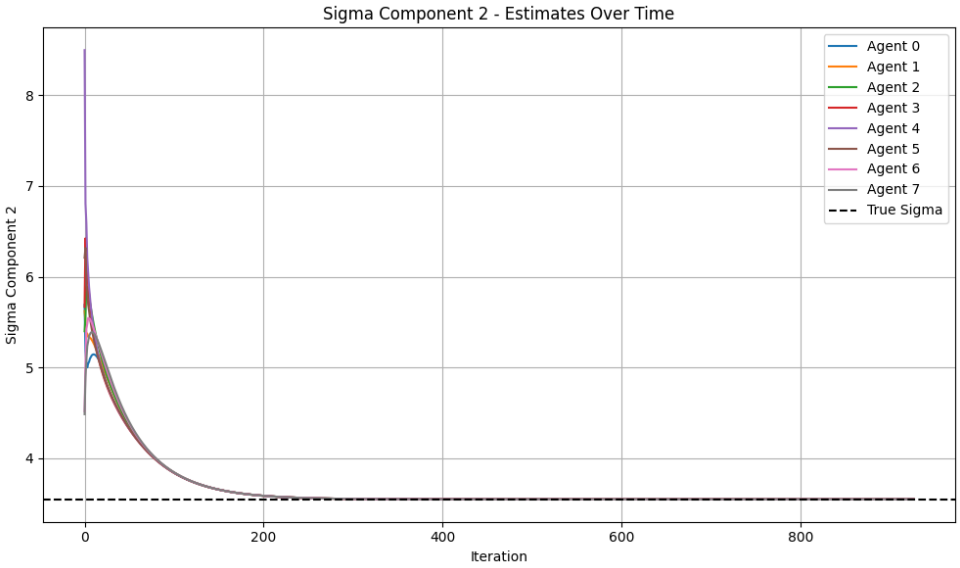

Figure 2.3: Barycenter Estimate Component 2 (sim 1)

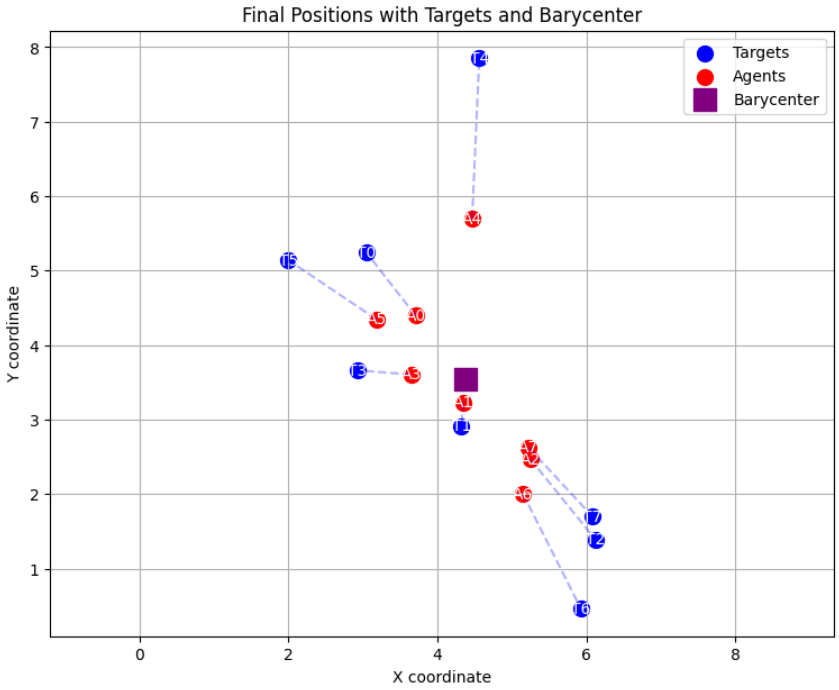

Figure 2.4: Spatial Plot Agents, Targets and Barycenter (sim 1)

Simulation 2

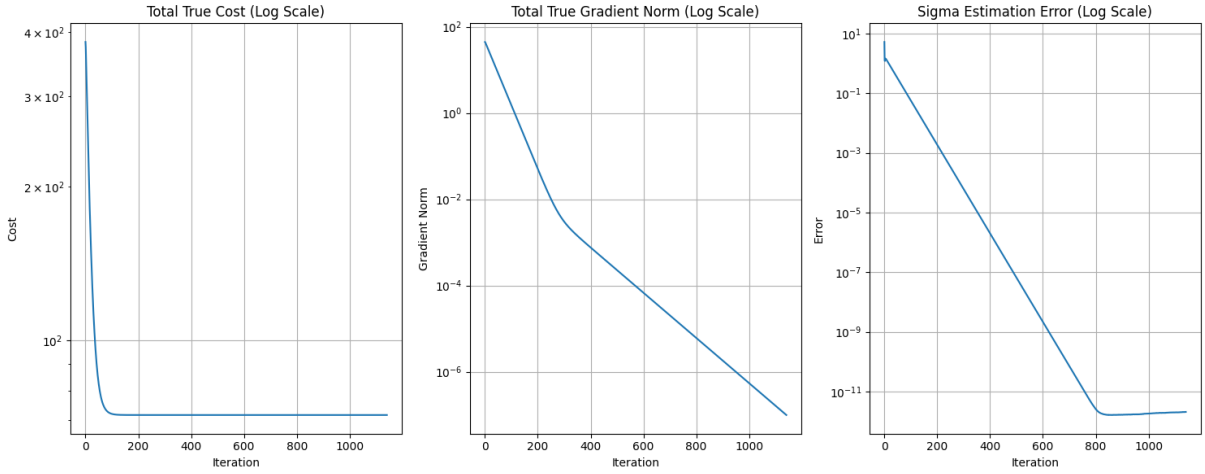

Figure 2.5: Total Cost, Gradient Norm, Sigma Estimation Error over Iterations (sim 2)

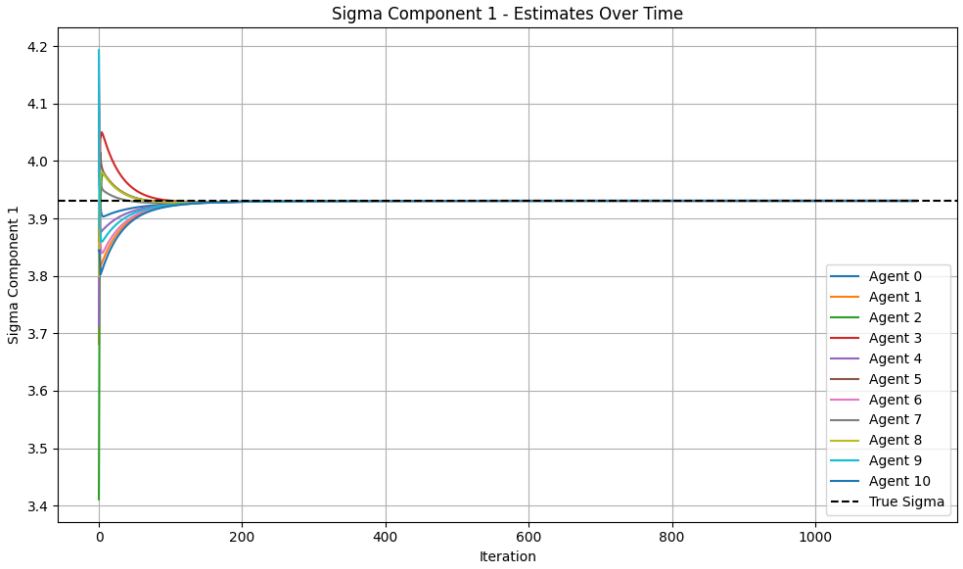

Figure 2.6: Barycenter Estimate Component 1 (sim 2)

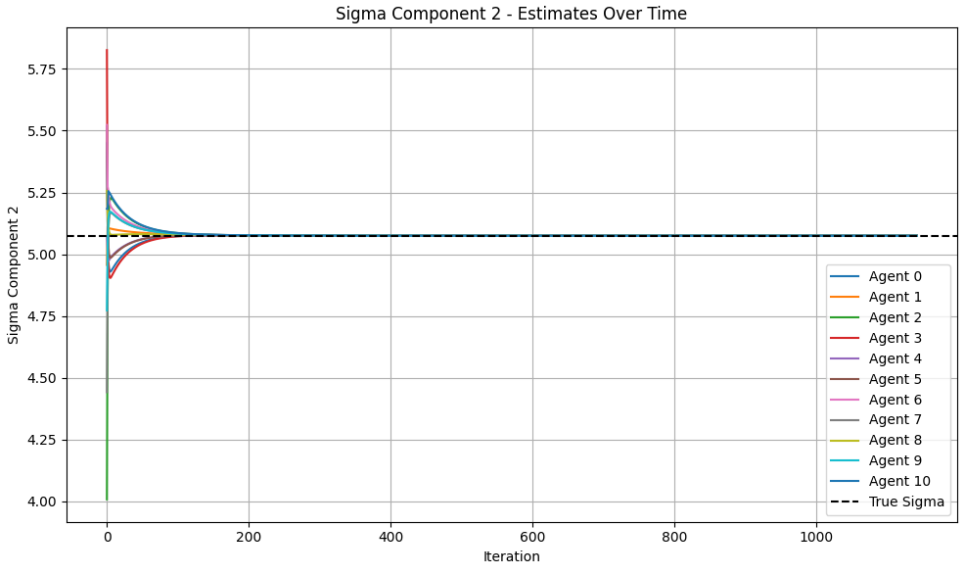

Figure 2.7: Barycenter Estimate Component 2 (sim 2)

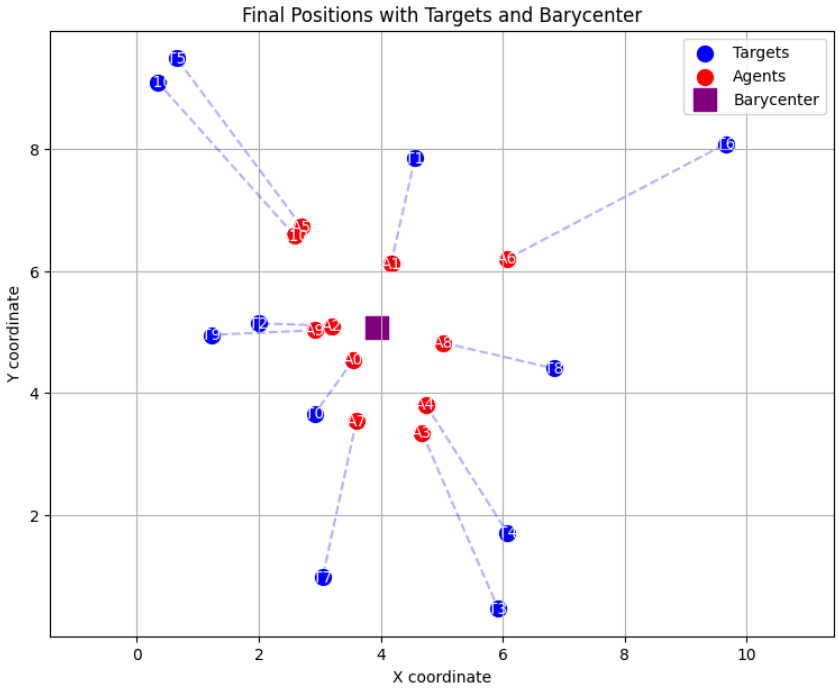

Figure 2.8: Spatial Plot Agents, Targets and Barycenter (sim 2)

2.5.2 Task 2.2 Plots

Appendix A

Graph Theory: Main Definitions and Properties

This appendix provides key definitions and properties from graph theory, particularly those relevant to Distributed Consensus and Aggregative Optimization discussed in the report.

A.1 Basic Graph Definitions

Definition A.1.1 (Digraph)**.** *A directed graph (digraph) is a pair $G = (\mathcal{I}, \mathcal{E})$, where:*

- $\mathcal{I} = \{1, \dots, N\}$ *is a set of nodes (agents).*
- $\mathcal{E} \subset \mathcal{I} \times \mathcal{I}$ *is a set of ordered edges representing communication links.*

An edge (i, j) denotes a link from node i to node j. A self-loop is an edge (i, i).

Definition A.1.2 (Undirected Graph)**.** *A digraph is undirected if for every edge $(i, j) \in \mathcal{E}$, the reverse edge $(j, i) \in \mathcal{E}$. Such graphs can also be defined with unordered node pairs.*

A.2 Connectivity and Periodicity

Definition A.2.1 (Strongly Connected Digraph)**.** *A digraph is strongly connected if there exists a directed path from any node to any other node.*

Definition A.2.2 (Connected Undirected Graph)**.** *An undirected graph is connected if there exists a path between any two nodes.*

Definition A.2.3 (Aperiodic Graph)**.** *A digraph is aperiodic if the greatest common divisor of the lengths of all its cycles is 1. A graph with at least one self-loop is always aperiodic.*

A.3 Weighted Graphs and Matrices

Definition A.3.1 (Weighted Digraph)**.** *A weighted digraph is a triplet $G = (\mathcal{I}, \mathcal{E}, \{a_{ij}\}_{(i,j)\in\mathcal{E}})$, where $(\mathcal{I}, \mathcal{E})$ is a digraph and each $a_{ij} > 0$ is a weight for edge (i, j). Unweighted graphs can be treated as weighted graphs with all $a_{ij} = 1$.*

Definition A.3.2 (Adjacency Matrix)**.** *The weighted adjacency matrix A of a digraph is defined as:*

$$A_{ij} = \begin{cases} a_{ij} > 0 & \text{if } (i,j) \in \mathcal{E} \\ 0 & \text{otherwise} \end{cases}$$

For unweighted graphs, $A_{ij} = 1$ if $(i, j) \in \mathcal{E}$.

A.4 Stochastic Matrices

Definition A.4.1 (Row Stochastic Matrix)**.** *A non-negative matrix $A \in \mathbb{R}^{N\times N}$ is row stochastic if each row sums to 1, i.e., $A\mathbf{1} = \mathbf{1}$ where $\mathbf{1}$ is the vector of all ones.*

Definition A.4.2 (Column Stochastic Matrix)**.** *A non-negative matrix $A \in \mathbb{R}^{N\times N}$ is column stochastic if each column sums to 1, i.e., $A^\top \mathbf{1} = \mathbf{1}$.*

Definition A.4.3 (Doubly Stochastic Matrix)**.** *A matrix A is doubly stochastic if it is both row and column stochastic.*

Theorem A.4.1 (Metropolis-Hastings Weight Matrix)**.** *For an undirected graph $G = (\mathcal{I}, \mathcal{E})$, the Metropolis-Hastings weights construct a doubly stochastic and symmetric weighted adjacency matrix A with entries:*

$$a_{ij} = \begin{cases} \frac{1}{1+\max\{d_i, d_j\}} & \text{if } (i,j) \in \mathcal{E} \text{ and } i \neq j \\ 1 - \sum\limits_{h \in \mathcal{N}_i \setminus \{i\}} a_{ih} & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

where $d_i = |\mathcal{N}_i|$ is the degree of node i.

Lemma A.4.1 (Properties of Stochastic Matrices)**.** *Let A be a row-stochastic matrix with associated digraph G. If G is strongly connected and aperiodic, then:*

- *The eigenvalue $\lambda = 1$ is simple*
- *All other eigenvalues μ satisfy $|\mu| < 1$*

A.5 Averaging Systems

Definition A.5.1 (Discrete-Time Averaging System)**.** *Given a digraph $G = (\mathcal{I}, \mathcal{E})$ with self-loops, the linear averaging algorithm has two formulations:*

- ***In-Neighbor Version**:*
$$x_i^{k+1} = \sum_{j \in \mathcal{N}_i^{in}} a_{ij} x_j^k$$
where A represents the adjacency matrix of the reverse communication graph $G^{comm,rev}$
- ***Out-Neighbor Version**:*
$$x_i^{k+1} = \sum_{j \in \mathcal{N}_i^{out}} a_{ij} x_j^k$$
where A represents the adjacency matrix of the sensing graph G^{sens}

Both formulations can be compactly written as $x^{k+1} = Ax^k$.

A.6 Consensus Results

Theorem A.6.1 (Consensus for Averaging Systems)**.** *Consider a discrete-time averaging system $x^{k+1} = Ax^k$ with associated digraph G strongly connected and aperiodic and row-stochastic matrix A. then:*

1. *There exists a positive left eigenvector $w > 0$ related to eigenvalue 1 such that:*
$$\lim_{k\to\infty} x^k = \mathbf{1}\left(\frac{w^\top x^0}{w^\top \mathbf{1}}\right)$$

2. If A is doubly stochastic, the system reaches average consensus:

$$\lim_{k\to\infty} x^k = \mathbf{1}\left(\frac{1}{N}\sum_{i=1}^{N} x_i^0\right)$$

Remark A.6.1. *The consensus value is a weighted average of initial conditions when A is row-stochastic, and the exact average when A is doubly stochastic.*

Conclusions

This report demonstrated the successful implementation of **distributed optimization algorithms** for multi-robot systems, focusing on **consensus-based target localization** (Task 1) and **aggregative optimization** (Task 2). The effectiveness was verified through:

- **Cost function convergence** to optimal solutions
- **Gradient norm reduction** below predefined thresholds
- **Consensus achievement** across agents
- **Real-time visualizations** of multi robot system behavior

The results confirm the algorithms' **robustness** and **scalability**. Future improvements could explore:

- **Aggregative feedback optimization** to consider the dynamics of robots
- **Moving targets** in Task 2 for dynamic environments
- **Time varying graphs** for both tasks

Bibliography

[1] F. Bullo, *Lectures on Network Systems*, 1st ed., Kindle Direct Publishing, 2019.

[2] G. Notarstefano, I. Notarnicola, and A. Camisa, "Distributed Optimization for Cyber-Physical Networks: Tutorial and Overview," *Foundations and Trends® in Systems and Control*, vol. 8, no. 4, pp. 253-356, 2021.

[3] A. Testa, G. Carnevale, and G. Notarstefano, "A Tutorial on Distributed Optimization for Cooperative Robotics: From Setups and Algorithms to Toolboxes and Research Directions," *IEEE Robotics and Automation Magazine*, vol. 30, no. 1, pp. 65-81, March 2023.