UNIVERSITÀ DI BOLOGNA

School of Engineering

Master Degree in Artificial Intelligence

Distributed Autonomous Systems

TITLE

Professors:
Giuseppe Notarstefano
Ivano Notarnicola

Students:
Valerio Costa
Tian Cheng Xia

Academic year 2024/2025

Abstract

Targets localization and aggregative multi-robot positioning are tasks that can be tackled in a distributed fashion. In this project, we solve both of them and experiment with their capabilities: the former is implemented in Python while the latter both in Python and ROS2. The first task can be solved using the gradient tracking algorithm, while the second one as an aggregative optimization problem. For the tracking task, our experimental results showed that communication graph connectivity and number of agents have significant effects in reaching convergence. Also, higher amounts of noise negatively affects the final results. For the aggregative tasks, we observed that all graph patterns behave similarly, independently of the number of robots. Moreover, experiments with different loss hyperparameters showed that the movement of the agents is coherent with the weights assigned to the components of the loss function.

Contents

Introduction **1**

1 Gradient tracking with quadratic functions **2**
1.1 Problem definition 2
1.2 Code structure 2
1.3 Experiments 3
1.3.1 Comparison between different graph patterns 3
1.3.2 Comparison with centralized gradient 6

2 Cooperative multi-robot target localization **8**
2.1 Problem definition 8
2.2 Code structure 9
2.3 Experiments 9
2.3.1 Comparison between different graph patterns 9
2.3.2 Comparison with centralized gradient 12
2.3.3 Different noises 13

3 Aggregative Optimization for Multi-Robot Systems **15**
3.1 Problem definition 15
3.2 Code structure 16
3.3 Experiments 16
3.3.1 Comparison with different graph patterns 17
3.3.2 Comparison with different loss configurations 18

Conclusions **20**

Introduction

Motivations

The project aims at solving and implementing distributed algorithms to solve two specific tasks. The first one, which can be solved using the gradient tracking algorithm, involves the problem of distributed target localization where some tracking robots want to estimate the position of some targets for which only noisy measurements are known. The second one, which is an aggregative optimization problem, consists of positioning robots balancing the requirements of being close to private targets while keeping the whole fleet tight. For both tasks, we experiment the resulting implementation to assess their results in terms of correctness, convergence, and scalability.

The rest of the report is structured as follows: in Chapter 1, we implement the gradient tracking algorithm for quadratic functions. In Chapter 2, we solve and show the results for the task of target localization. In Chapter 3, we present the solution and the experiments for the task of robots positioning.

Contributions

This project provides a small comparison benchmark of some distributed algorithms. These results, although limited, show the effectiveness of these algorithms in solving some tasks for which taking a centralized approach is not realistic.

In practical terms, most of the work has been done in pair programming and both group members have contributed equally to the implementation.

Chapter 1

Gradient tracking with quadratic functions

1.1 Problem definition

The first task aim is to solve a multi-robot target localization problem. Indeed, we consider a fleet of $N \in \mathbb{N}$ agents and a collection of $N_T \in \mathbb{N}$, in which each agent makes noisy measurements of its distance from every target. The objective is to design a distributed algorithm, in order to minimize the loss between every target estimated position and its ground truth (as formulated below), with a particular focus on trying to reach a consensus within the fleet.

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

The first part of the task consists of implementing the gradient tracking algorithm generalized in $\mathbb{R}^d$ and then experiment with the implementation using quadratic functions, which we define in the usual way as:

$$f(\boldsymbol{z}) = \frac{1}{2}\boldsymbol{z}^T \boldsymbol{Q}\boldsymbol{z} + \boldsymbol{r}^T \boldsymbol{z} \quad \nabla f(\boldsymbol{z}) = \boldsymbol{Q}\boldsymbol{z} + \boldsymbol{r}$$

where $\boldsymbol{z} \in \mathbb{R}^d$, $\boldsymbol{Q} \in \mathbb{R}^{d \times d}$ is positive definite, and $\boldsymbol{r} \in \mathbb{R}^d$.

1.2 Code structure

The code provided is structured with the following main modules:

`algorithm.py` Two functions, allowing to run the distributed gradient tracking algorithm and its centralized version.

`loss.py` Class definition of the quadratic loss function.

`plot.py` All the functions used to plot the loss, gradient norm and distance to optimum evolution over iterations.

`scenarios.py` Two functions, one aiming to create the graph G and the relative adjacency matrix A, while the other designed to initialize the parameters and the quadratic loss functions.

`utils.py` The definition of the functions for computing the average estimate error for each agent with respect to its ground truth, and the average consensus error meaning the average distance of each agent estimate with the consensus.

In practice all the experiments can be executed from the script `main_quadratic.py`.

1.3 Experiments

We analyzed the behavior with quadratic functions through the definition of different problems with different kinds of graph patterns (in particular complete, binomial, cycle, star, and path graph). The configurations we tested are the following:

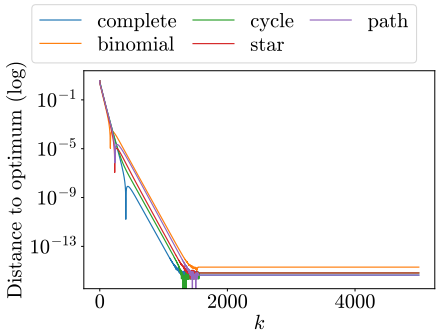
- A small problem (5 agents in $\mathbb{R}^3$),
- A problem with higher dimensionality (5 agents in $\mathbb{R}^{15}$),
- A problem with many agents (15 and 30 agents in $\mathbb{R}^3$),
- A problem with many agents in higher dimensionality (15 and 30 agents in $\mathbb{R}^{15}$).

In addition, we performed a comparison between the distributed gradient tracking algorithm and the centralized one. For compactness in the discussion, in the rest of this report we show the results with a single initialization seed and, if not specified, it indicates that the results are consistent across different initializations. Also, for readability, for quadratic functions we report the distance to the optimum in semi-logarithmic scale instead of the cost itself which can be negative.

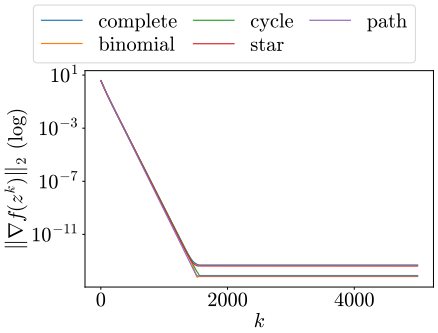
1.3.1 Comparison between different graph patterns

For the starting small problem, we can observe from Figure 1.1 a relatively smooth improvement of the cost function and an exponentially decreasing gradient in all cases. Moreover, a result that can be expected and is consistently persistent in all the other experiments is that consensus is reached slightly faster with a complete graph. Next, by experimenting with higher dimensionality, we can observe from Figure 1.2 that the behavior of both the cost and its gradient are very similar to the previous case with the only

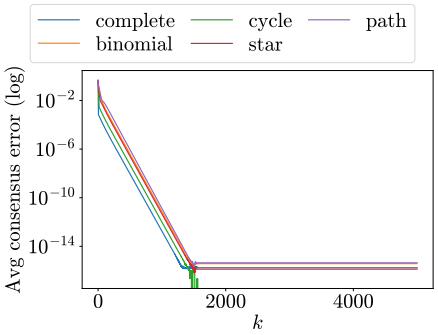
difference that more iterations are required to reach full convergence, indicating that the dimensionality is marginal in changing the difficulty of the problem.



(a) Distance to optimum

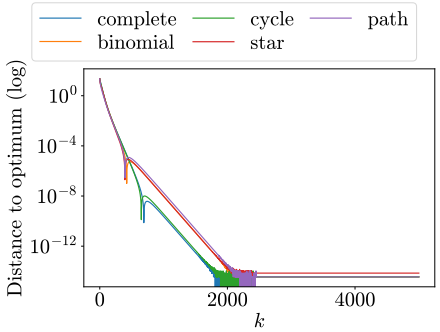


(b) Gradient norm evolution

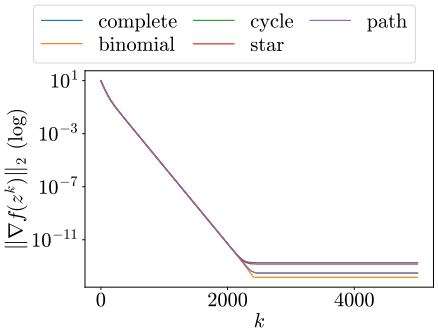


(c) Consensus error

Figure 1.1: Quadratic function minimization with 5 agents in $\mathbb{R}^3$



(a) Distance to optimum

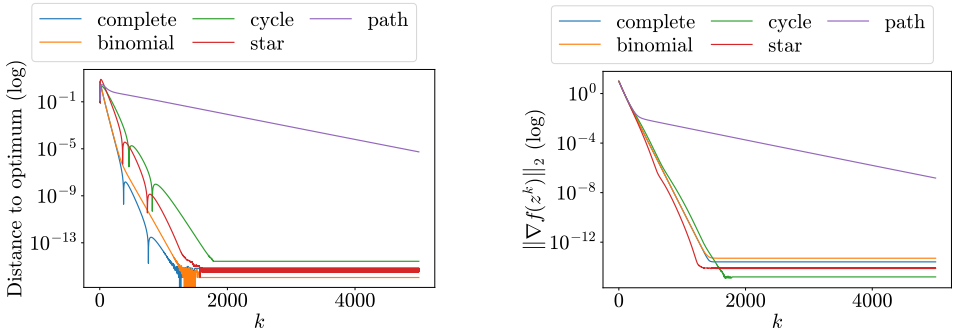


(b) Gradient norm evolution

Figure 1.2: Quadratic function minimization with 5 agents in $\mathbb{R}^{15}$

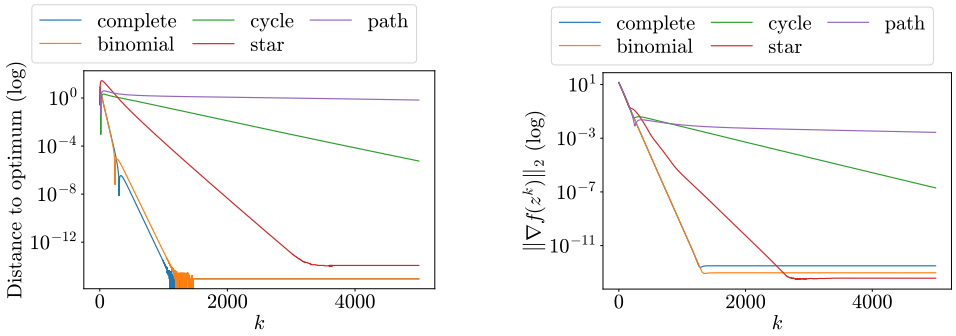
In the case of many agents with lower dimensionality, we can observe

from Figure 1.3 and Figure 1.4 that the cost function does not reach the optimum within the given number of iterations with the configuration using the path graph, indicating that connectivity is important for larger numbers of agents. This can be explained by the fact that, by adding more agents, the overall problem includes more local losses and becomes more difficult to solve in a distributed way. From Figure 1.5 and Figure 1.6, we observe the same convergence behavior as in the previous case and also confirm that, with higher dimensionality, the problem is not significantly affected.



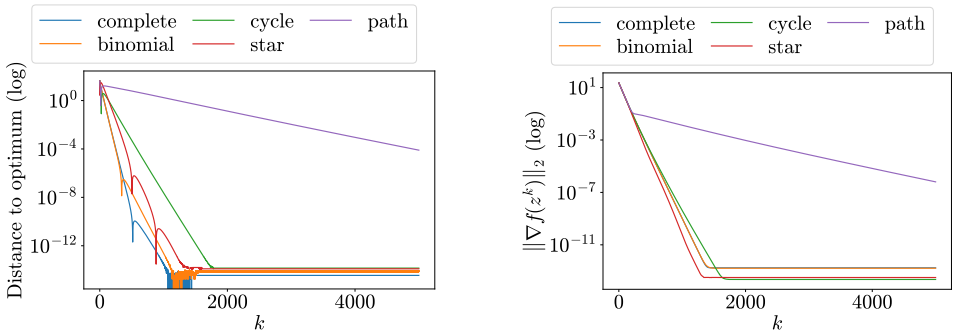
(a) Distance to optimum (b) Gradient norm evolution

Figure 1.3: Quadratic function minimization with 15 agents in $\mathbb{R}^3$



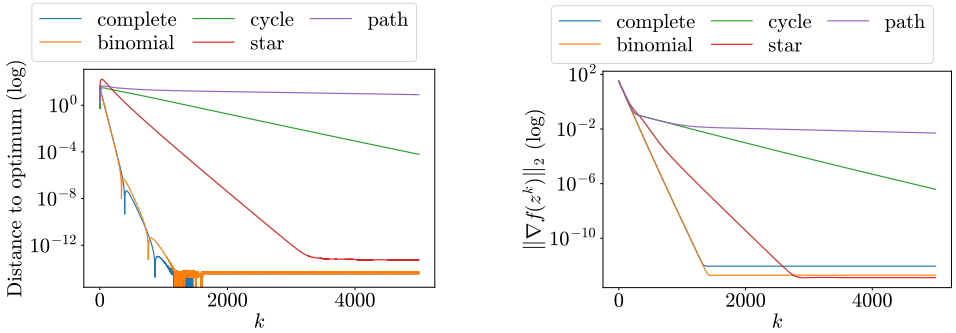
(a) Distance to optimum (b) Gradient norm evolution

Figure 1.4: Quadratic function minimization with 30 agents in $\mathbb{R}^3$



(a) Distance to optimum (b) Gradient norm evolution

Figure 1.5: Quadratic function minimization with 15 agents in $\mathbb{R}^{15}$

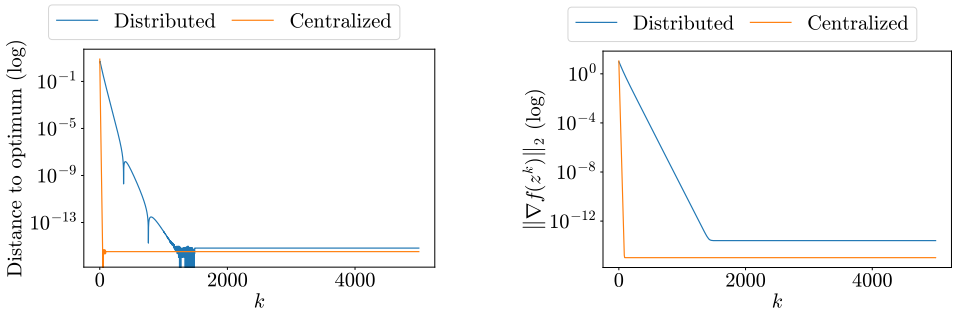


(a) Distance to optimum (b) Gradient norm evolution

Figure 1.6: Quadratic function minimization with 30 agents in $\mathbb{R}^{15}$

1.3.2 Comparison with centralized gradient

Following the previous results, we select the configuration using the complete graph for the comparison with the centralized gradient method. In Figure 1.7, we can observe the results with 15 agents, but the overall behavior is the same for all configurations. It can be seen that, as one could expect, the centralized gradient method is faster to converge compared to a distributed algorithm as it has available all the global information and does not rely on estimates and information exchange with the neighbors.



(a) Distance to optimum

(b) Gradient norm evolution

Figure 1.7: Quadratic function minimization with 15 agents in $\mathbb{R}^3$ compared to centralized gradient

Chapter 2

Cooperative multi-robot target localization

2.1 Problem definition

The problem in the second part of the first task involves a similar context as the previous case. The fundamental difference is that the local losses each agent has to minimize are designed to solve the problem of target localization. More specifically, this problem requires the implementation of a gradient tracking algorithm for estimating the position of N_T fixed targets in a distributed way through N_R tracking robots. Each robot is located at position $\boldsymbol{p}_i \in \mathbb{R}^2$ and it is assumed that the distance measured from each robot is noisy. Given the positions $\boldsymbol{p}_i$ and $\boldsymbol{p}_\tau$ of the i-th robot and the τ-th target, respectively, we model the measured noisy distance $d_{i,\tau}$ as follows:

$$d_{i,\tau} = \|\boldsymbol{p}_i - \boldsymbol{p}_\tau\| + \varepsilon \cdot \texttt{noise}$$

where $\texttt{noise} \sim P$ is drawn from some distribution P and $\varepsilon \in \mathbb{R}$ is the noise rate.

The local loss each robot i uses is the following:

$$l_i(\boldsymbol{z}) = \sum_{\tau=1}^{N_T} \left(d_{i,\tau}^2 - \|\boldsymbol{z}_\tau - \boldsymbol{p}_i\|^2\right)^2 \quad \nabla l_i(\boldsymbol{z}) = (\nabla l_{i,1}(\boldsymbol{z}_1), \ldots, \nabla l_{i,N_T}(\boldsymbol{z}_{N_T}))$$

$$\nabla l_{i,j}(\boldsymbol{z}_j) = -4\left(d_{i,j}^2 - \|\boldsymbol{z}_j - \boldsymbol{p}_i\|^2\right)(\boldsymbol{z}_j - \boldsymbol{p}_i)$$

where $\boldsymbol{z} = (\boldsymbol{z}_{\tau_1}, \ldots, \boldsymbol{z}_{\tau_{N_T}}) \in \mathbb{R}^{2N_T}$ is the stack of decision variables of robot i containing the estimated positions of the targets $\boldsymbol{z}_\tau \in \mathbb{R}^2$ and $\nabla l_i(\boldsymbol{z}) \in \mathbb{R}^{2N_T}$ is the concatenation of the gradients computed with respect to each target.

2.2 Code structure

The code provided is structured with the following main modules (considering only the ones which have differences with the previous case):

`loss.py` Class definition of the target localization loss function.

`plot.py` All the functions used to plot the loss, gradient norm evolution over iterations.

`scenarios.py` Two functions, one aiming to create the graph G and the relative adjacency matrix A, while the other is designed to initialize the parameters (e.g., the ground truth target positions, initial robot positions, and noisy estimated target distances) and the target localization loss functions.

In practice all the experiments can be executed from the script `main_tracking.py`.

2.3 Experiments

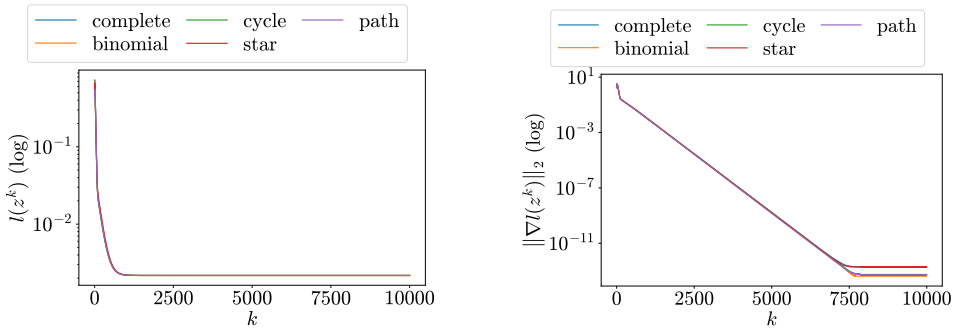
We approach the experimentation of such algorithm by trying different graph patterns and comparing with the centralized gradient method, similarly to the previous case. At first, we evaluated the performance of the algorithm in the following cases, all with the same type of noise:

- Network of 5 robots and 1 target,
- Network of 5 robots and 3 targets,
- Network of 15 robots and 3 targets,
- Network of 30 robots and 3 targets.

Then, the focus switched to observe how much the performance changes in terms of noise. By fixing the problem configuration, we experimented with varying Gaussian noises, Poisson noises, and noise rates.

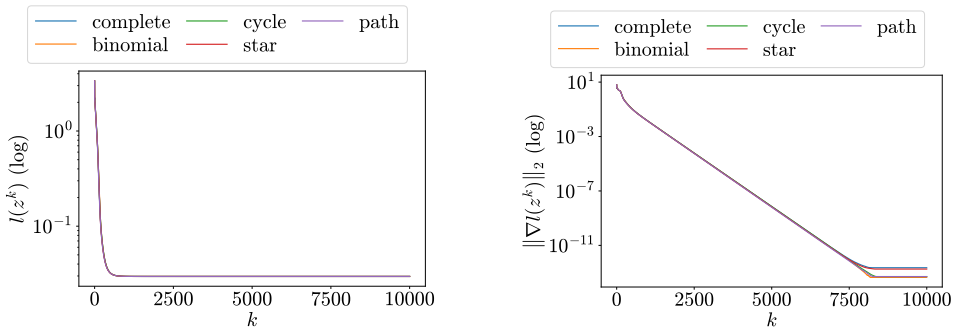
2.3.1 Comparison between different graph patterns

In terms of graph pattern, we can observe from Figure 2.1 and Figure 2.2 that with the same number of robots and increasing number of targets, the number of iterations required to converge is roughly the same. This makes sense as each target is independent to the others and can be tracked in parallel.



(a) Loss evolution (b) Gradient norm evolution

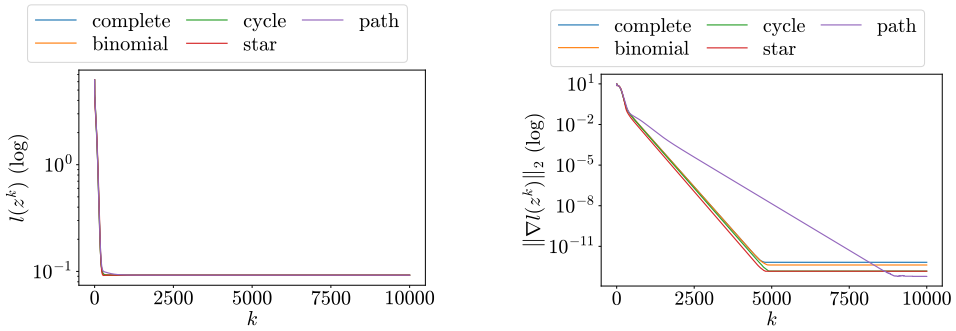
Figure 2.1: Tracking with 5 robots and 1 target



(a) Loss evolution (b) Gradient norm evolution

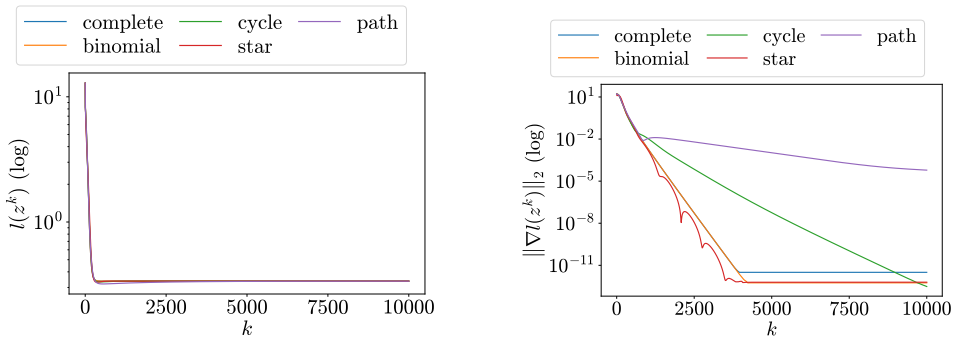
Figure 2.2: Tracking with 5 robots and 3 targets

Instead, by increasing the number of tracking robots, we can see from Figure 2.3 and Figure 2.4 that the plateau is reached in fewer number of iteration, which intuitively means that more tracking robots help in reaching a faster convergence. However, we can also observe from these figures that with less connected graphs, convergence is affected negatively.



(a) Loss evolution (b) Gradient norm evolution

Figure 2.3: Tracking with 15 robots and 3 targets



(a) Loss evolution (b) Gradient norm evolution

Figure 2.4: Tracking with 30 robots and 3 targets

Moreover, as the total loss is a summation, we must note that the overall loss is higher in the case of more agents or targets, but this does not indicate worse tracking results. We report in Figure 2.5 the average distance between the estimated and real target positions for a fixed configuration with varying number of robots. It can be seen, as intuition would suggest, that on average the tracking error becomes smaller by increasing the number of tracking robots.

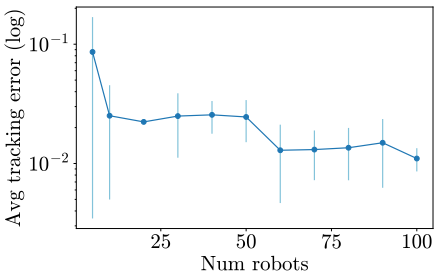


Figure 2.5: Average tracking error for different number of robots (average of 5 runs). Vertical bars represent the standard deviation.

Finally, an observation consistent in all experiments is that, as shown in Figure 2.6, the overall behavior of this system is to reach an approximate consensus (i.e., consensus error around 10^{-4}) in the first few iterations and then optimize the loss while improving and preserving consensus. This can also be observed in Figure 2.7 where a few frames of the animation of the scenario are shown.

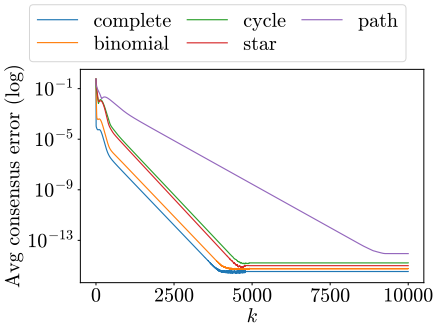


Figure 2.6: Tracking with 15 robots and 3 targets

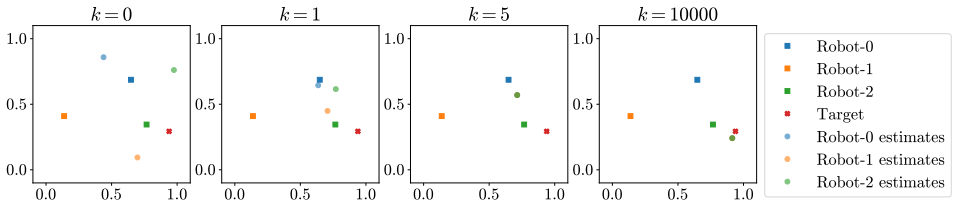
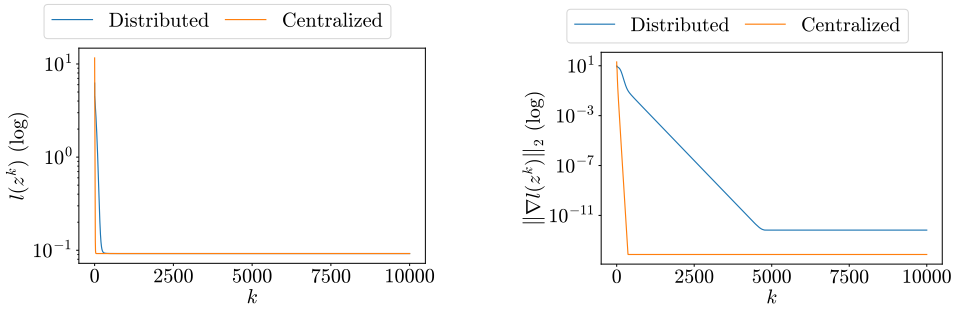


Figure 2.7: Tracking animation with 3 robots and 1 target

2.3.2 Comparison with centralized gradient

Compared with the centralized gradient algorithm, the plots in Figure 2.8 confirm what we observed before in the case of quadratic functions. The

convergence speed is faster and more accurate in a centralized approach, which also results in a lower average tracking error.

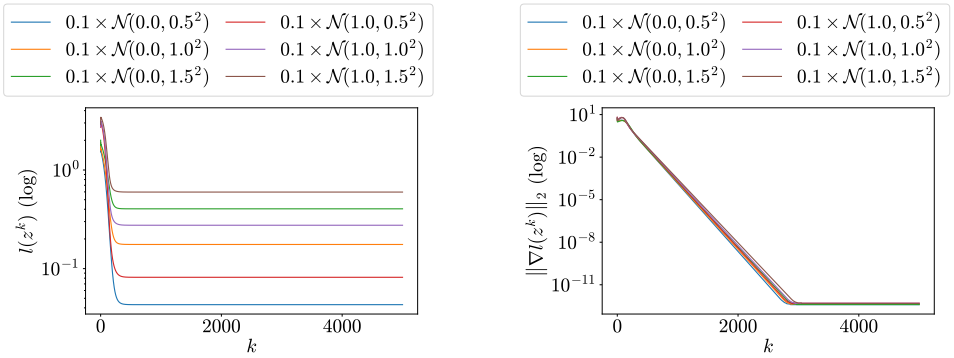


(a) Loss evolution (b) Gradient norm evolution

Figure 2.8: Tracking with 15 robots and 3 targets with centralized gradient

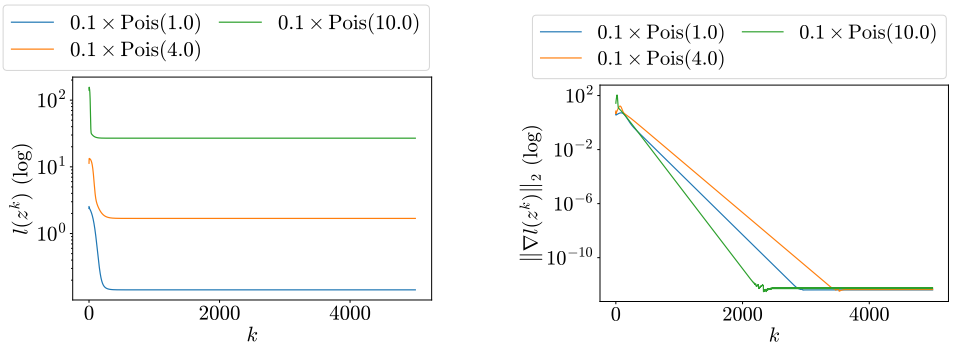
2.3.3 Different noises

From the experiments with different noises, we observed a worsening in performance that is proportional to the amount of noise injected into the distance measurement, implying as one could expect that more noise leads to worse results. This behavior is consistent with noise drawn from different distributions as in Figure 2.9 and Figure 2.10, and also when the noise rate is increased as in Figure 2.11.



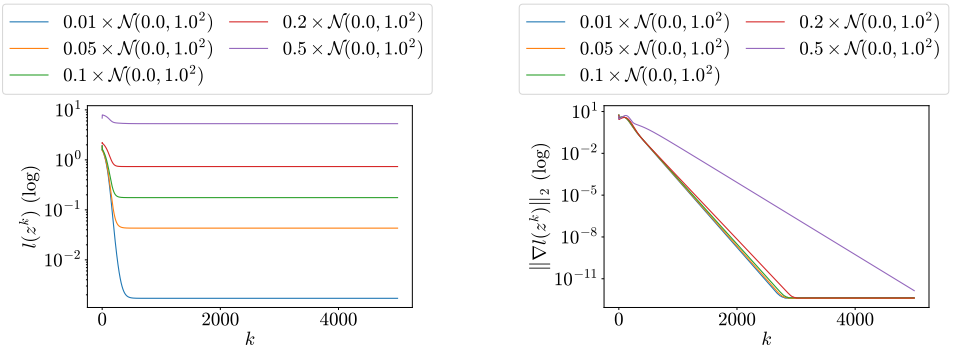
(a) Loss evolution (b) Gradient norm evolution

Figure 2.9: Tracking with 15 robots and 3 targets with noise drawn from Gaussian distributions



(a) Loss evolution (b) Gradient norm evolution

Figure 2.10: Tracking with 15 robots and 3 targets with noise drawn from Poisson distributions



(a) Loss evolution (b) Gradient norm evolution

Figure 2.11: Tracking with 15 robots and 3 targets with different rates of Gaussian noise

Chapter 3

Aggregative Optimization for Multi-Robot Systems

3.1 Problem definition

This task consists of solving the problem of moving $N \in \mathbb{N}$ agents with positions $\boldsymbol{z}_i \in R^2$ in such a way that they are close to their own private target $\boldsymbol{r}_i \in \mathbb{R}^2$ while staying tight to the fleet. This problem can be tackled using aggregative optimization by solving the following problem:

$$\min_{\boldsymbol{z} \in \mathbb{R}^{2N}} \sum_{i=1}^{N} l_i(\boldsymbol{z}_i, \sigma(\boldsymbol{z}))$$

$$\text{with } \sigma(\boldsymbol{z}) = \frac{1}{N} \sum_{i=1}^{N} \phi_i(\boldsymbol{z}_i),$$

where $\boldsymbol{z} = (\boldsymbol{z}_1, \dots, \boldsymbol{z}_N)$ is the stack of the agents' positions, $l_i : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}$ is the local losses of agent i and $\sigma : \mathbb{R}^{2N} \to \mathbb{R}^2$ is an aggregation function.

We formulate the local loss of an agent i as the following function:

$$l_i(\boldsymbol{z}_i, \sigma(\boldsymbol{z})) = \gamma_1 \frac{1}{2} \|\boldsymbol{z}_i - \boldsymbol{r}_i\|^2 + \gamma_2 \frac{1}{2} \|\boldsymbol{z}_i - \sigma(\boldsymbol{z})\|^2,$$

where the first term models the vicinity to the private targets while the second one represents the fleet tightness. To add more flexibility, the hyper-parameters $\gamma_1 \in \mathbb{R}$ and $\gamma_2 \in \mathbb{R}$ have been introduced to allow weighing the two requirements differently. The gradients of the loss with respect to the first and second arguments, respectively, are the following:

$$\nabla_1 l_i(\boldsymbol{z}_i, \sigma(\boldsymbol{z})) = \gamma_1(\boldsymbol{z}_i - \boldsymbol{r}_i) + \gamma_2(\boldsymbol{z}_i - \sigma(\boldsymbol{z}))$$
$$\nabla_2 l_i(\boldsymbol{z}_i, \sigma(\boldsymbol{z})) = -\gamma_2(\boldsymbol{z}_i - \sigma(\boldsymbol{z})),$$

In terms of aggregation function $\sigma(\boldsymbol{z})$, the local function ϕ_i associated to each agent i is defined as follows:

$$\phi_i(\boldsymbol{z}_i) = \alpha_i \boldsymbol{z}_i,$$

where $\alpha_i \in R$ is a hyperparameter. With $\alpha_i = 1$ for all i, we obtain the standard formulation of the problem in which $\sigma(\boldsymbol{z})$ represents the barycenter of the fleet. With different α_i (with $\sum_i^N \alpha_i = N$), we obtain a $\sigma(\boldsymbol{z})$ that is a weighted average and can be interpreted as a barycenter that is biased towards specified agents.

3.2 Code structure

The code provided is structured with the following main modules:

`algorithm.py` Function implementing the aggregative optimization algorithm.

`loss.py` Class definition of the Agent and its private functions, nominally the aggregative loss and the linear function needed to compute the barycenter.

`plot.py` All the functions used to plot the loss, gradient norm evolution over iterations, but also an animation of the system and its single frames.

`scenarios.py` Two functions, one aiming to create the graph G and the relative adjacency matrix A, while the other designed to initialize the parameters for the problem (e.g., targets' positions and list of Agents).

In practice all the experiments can be executed from the `main_tracking.py` script.

The agent in the ROS2 version of the project is implemented using the node defined in `Agent.py`. Each agent has a topic in which it publishes its states and listens from the topics of its neighbors. A polling timer is used to determine the communication frequency. Update steps are performed using the same functions defined in `algorithm.py`. For visualization, we implemented a node that acts as a middleware for RVIZ that reads from the topics of all agents and published some new topics for the specific markers.

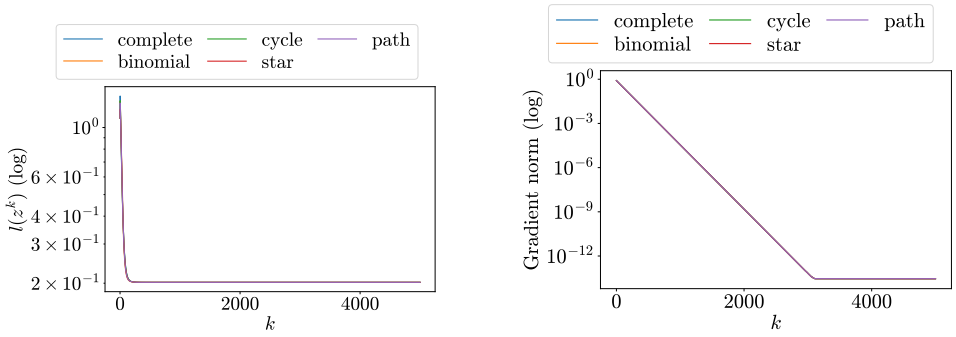
3.3 Experiments

As in the previous task, we first test the effectiveness of our implementation with different graph patterns for the communication graph and different number of agents. We test with configurations consisting of 5 robots, 15 robots, and 30 robots. Then, to assess the actual results, we visually check the coherence of the positioning of the agents at convergence and experiment with different loss hyperparameters to favor specific agents, and target and barycenter vicinity.

3.3.1 Comparison with different graph patterns

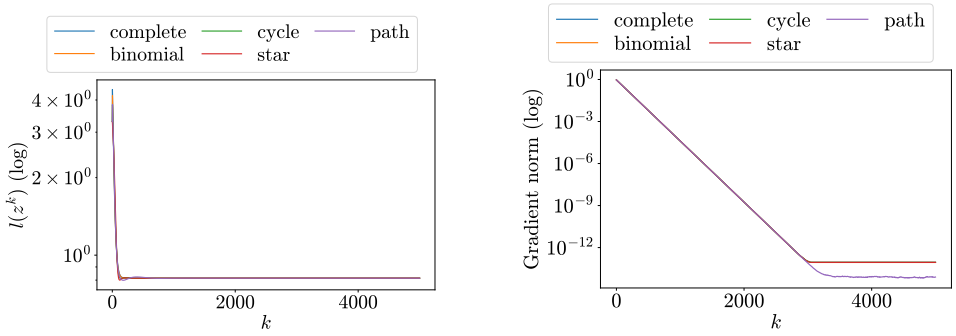
We first test the implementation with 5 agents. From the plots in Figure 3.1, we can observe an identical trend in terms of loss and gradient for every graph pattern we have experimented with. In all cases, we can observe that the algorithm converges, and also, visually, we can see that the final positions of the agents tends to reach an expected behavior.

By considering a scenario with 15 and 30 agents, as shown in Figure 3.2 and Figure 3.3, we cannot detect any relevant changes in behavior as they all reach convergence in the same way as in the experiment with 5 agents. The only thing we can underline is that the trend for less connected graphs have a little variation at convergence, most likely due to numerical instability.



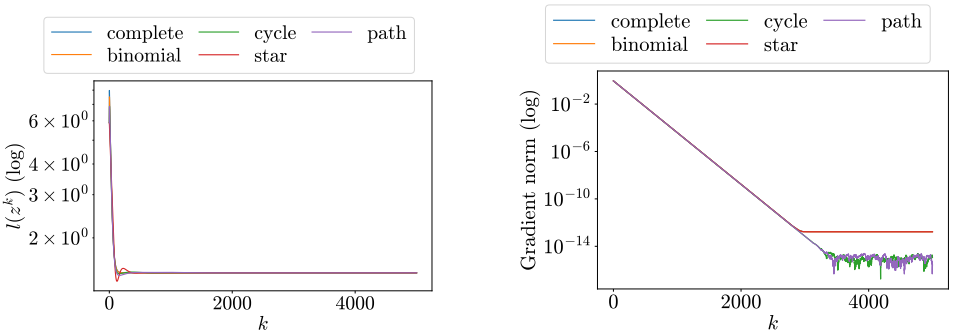
(a) Loss evolution (b) Gradient norm evolution

Figure 3.1: Positioning with 5 robots



(a) Loss evolution (b) Gradient norm evolution

Figure 3.2: Positioning with 15 robots



(a) Loss evolution

(b) Gradient norm evolution

Figure 3.3: Positioning with 30 robots

3.3.2 Comparison with different loss configurations

To test the visual results, we first check the results in the plain version with equally weighted loss components and equal agents' importance. Results are presented in Figure 3.4 in Python and Figure 3.5 in ROS2. As one could expect, with the two components of the loss balanced, the agents converge to a position that is midway between the barycenter and their private target.

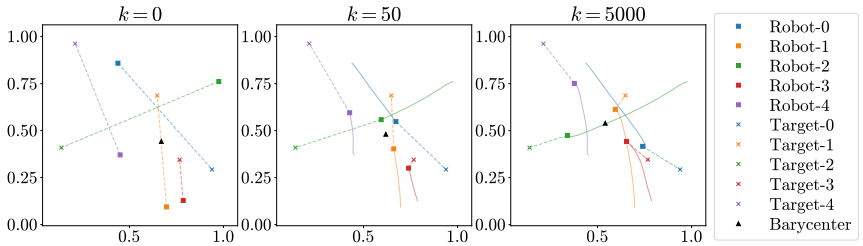


Figure 3.4: Frames with 5 robots and balanced loss. Dashed lines connect robots to the targets and solid lines are the trajectories.

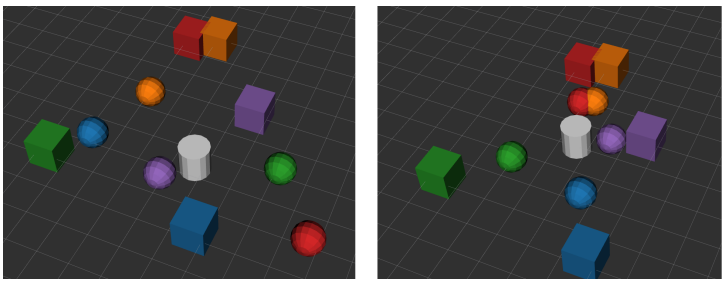


Figure 3.5: Frames with 5 robots and barycenter that prioritizes robot 0. Spheres are agents, cubes are targets, and the cylinder is the barycenter.

Then, we continue the experiments by checking the results in terms of different loss hyperparameters. We experiment our formulation with different weights to prioritize target vicinity (higher γ_1), barycenter vicinity (higher γ_2), and different agents' importance (different α_i). Results are presented in Figure 3.6, Figure 3.7, and Figure 3.8, respectively. In all cases the positions at convergence are intuitively the expected ones, showing that the algorithm allows many degrees of freedom.

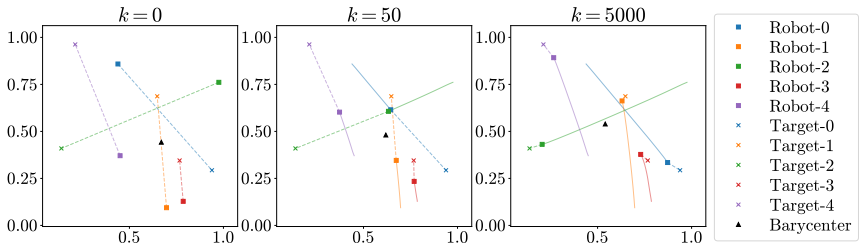


Figure 3.6: Frames with 5 robots and loss that prioritizes the private targets. Dashed lines connect robots to the targets and solid lines are the trajectories.

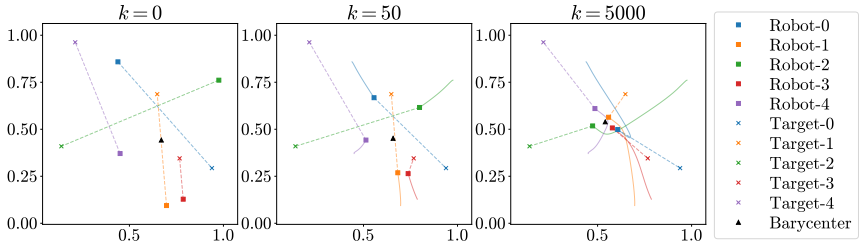


Figure 3.7: Frames with 5 robots and loss that prioritizes the barycenter. Dashed lines connect robots to the targets and solid lines are the trajectories.

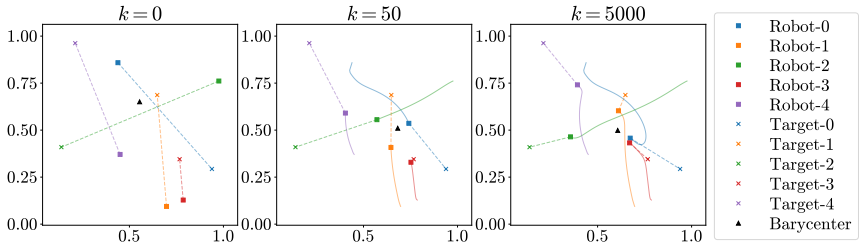


Figure 3.8: Frames with 5 robots and barycenter that prioritizes robot 0. Dashed lines connect robots to the targets and solid lines are the trajectories.

Conclusions

We tackled both problems and performed a set of experiments for each task. Regarding the multi-robot target localization problem, we tested the variation in performance under different graph patterns, number of agents, and dimensionality of the state variables. These tests showed that all graph patterns converge when using quadratic functions and higher connectivity helps in reaching it faster, which is more noticeable in case of higher numbers of agents. Instead, in the case of the target tracking loss, the most important observation is that convergence is faster with a higher number of agents and more connected graph patterns. Other tests showed that under different noise distributions and intensities, tracking accuracy, as expected, worsens for stronger amounts of noise.

Regarding the aggregative optimization problem, we first observed how the loss and gradient norm evolution changes based on different graph patterns and number of agents. The outcome showed a nearly identical trend for all the configurations, independently of the number of robots. Additional experiments were performed by changing the loss hyperparameters affecting agents' importance, and targets and barycenter weights. Results showed that the agents' movement and the final positions were intuitively the expected ones.