UNIVERSITÀ DI BOLOGNA

School of Engineering

Master Degree in Artificial Intelligence

Distributed Autonomous Systems

Multi-Robot Distributed Optimization

Professors:
Giuseppe Notarstefano
Ivano Notarnicola

Students:
Valerio Costa
Tian Cheng Xia

Academic year 2024/2025

Abstract

Multi-robot target localization and multi-robot positioning are two 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 show that communication graph connectivity and number of agents have significant effects in convergence speed and accuracy. For the aggregative task, we observed that all graph patterns behave similarly, independently of the number of robots. Moreover, experiments with different loss hyperparameters show that the constraints defined in the loss function can be easily tuned and the movement of the robots is coherent with the assigned weights.

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 **7**
2.1 Problem definition 7
2.2 Code structure 8
2.3 Experiments 8
2.3.1 Comparison between different graph patterns 8
2.3.2 Comparison with centralized gradient 11
2.3.3 Comparison with different noises 11

3 Aggregative Optimization for Multi-Robot Systems **13**
3.1 Problem definition 13
3.2 Code structure 14
3.3 Experiments 14
3.3.1 Comparison with different graph patterns 15
3.3.2 Comparison with different loss configurations 16

Conclusions **18**

Introduction

Motivations

The project aims at 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 and keeping the whole fleet tight. For both tasks, we experiment with the 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 part of the first task consists of implementing the gradient tracking algorithm generalized in $\mathbb{R}^d$ and experimenting 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 $\boldsymbol{z} \in \mathbb{R}^d$ are the parameters, and $\boldsymbol{Q} \in \mathbb{R}^{d\times d}$ (positive definite) and $\boldsymbol{r} \in \mathbb{R}^d$ are the coefficients of the function. More in details, given $N \in \mathbb{N}$ agents where each has associated a private quadratic function f_i with its own coefficients $\boldsymbol{Q}_i$ and $\boldsymbol{r}_i$, the goal is to use the gradient tracking algorithm to solve the following problem:

$$\min_{\boldsymbol{z}=(\boldsymbol{z}_1,\dots,\boldsymbol{z}_N)} \sum_{i=1}^{N} f_i(\boldsymbol{z}_i)$$

where $\boldsymbol{z}$ is the stack of estimates of the agents.

1.2 Code structure

The code provided is structured with the following modules:

`algorithm.py` contains the functions to run the distributed gradient tracking algorithm and the centralized gradient descent.

`loss.py` defines the quadratic function implemented as a Python class.

`plot.py` provides all the functions used to plot the cost, the gradient norm, and the distance to the optimum.

`scenarios.py` with the functions to create the communication graph and to initialize the problem of minimizing quadratic functions.

`utils.py` implements the utility metric to compute the average consensus error.

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

1.3 Experiments

We analyze the behavior with quadratic functions through the definition of different scenarios with different kinds of graph patterns (in particular: complete, binomial with edge probability of 0.3, cycle, star, and path graph). The configurations we test 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 $\mathbb{R}^{15}$).

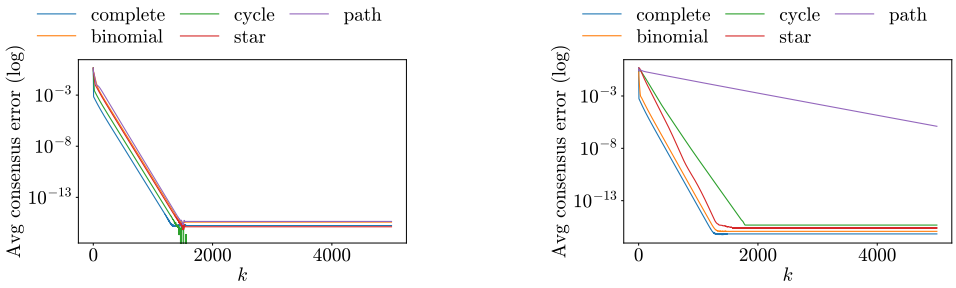
In addition, we perform a comparison between the distributed gradient tracking algorithm and the centralized one.

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

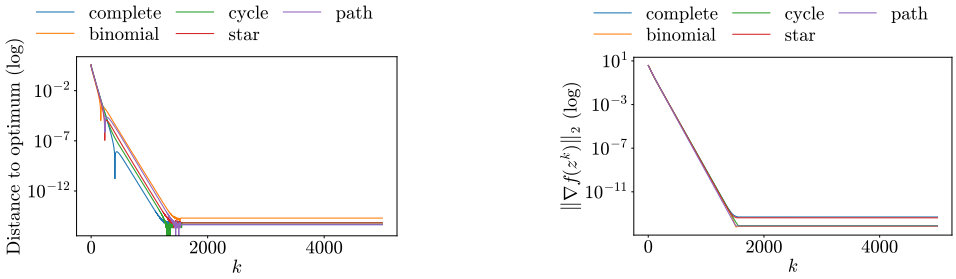
We start the analysis by first inspecting the consensus error of the algorithm. In Figure 1.1, we report the average consensus error in the cases of 5 and 15 agents. The general behavior that can be observed is that in all cases consensus is reached, in line with the theoretical guarantees of the algorithm. However, it can also be seen that consensus is reached faster when the communication graph has more connectivity, indicating that, as intuition would suggest, being able to communicate with more agents helps in this aspect. Also, although not reported for compactness, these observations also hold for the other scenarios.

In terms of cost function instead, for the small problem we can observe from Figure 1.2 a relatively smooth improvement of the cost function and an



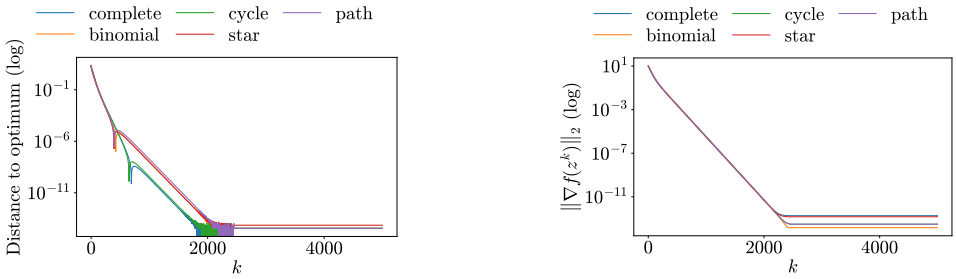
(a) 5 agents in $\mathbb{R}^3$ (b) 15 agents in $\mathbb{R}^3$

Figure 1.1: Consensus error for quadratic function minimization



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

Figure 1.2: Quadratic function minimization with 5 agents in $\mathbb{R}^3$

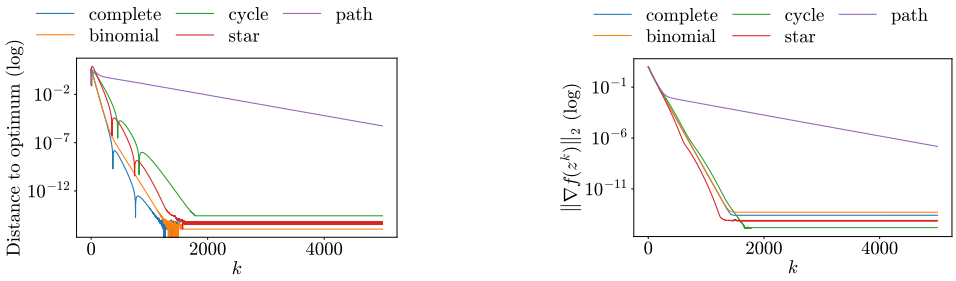


(a) Distance to optimum (b) Gradient norm evolution

Figure 1.3: Quadratic function minimization with 5 agents in $\mathbb{R}^{15}$

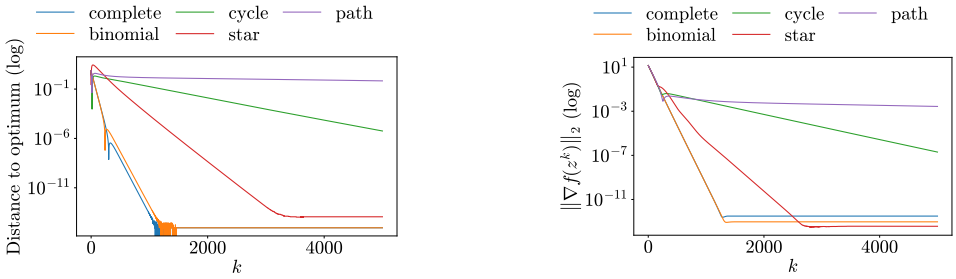
exponentially decreasing gradient in all cases, as expected from theoretical results. By experimenting with variables of higher dimensionality, we can observe from Figure 1.3 that the behavior of both the cost and its gradient are very similar to the previous case with the only difference that more iterations are required to reach full convergence, indicating that dimensionality is marginal in changing the difficulty of the problem.

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



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

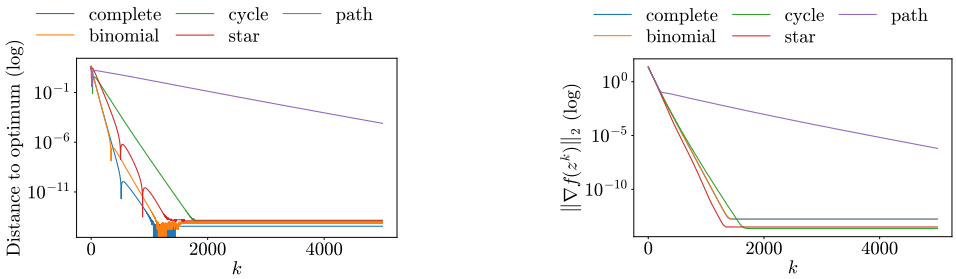
Figure 1.4: Quadratic function minimization with 15 agents in $\mathbb{R}^3$



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

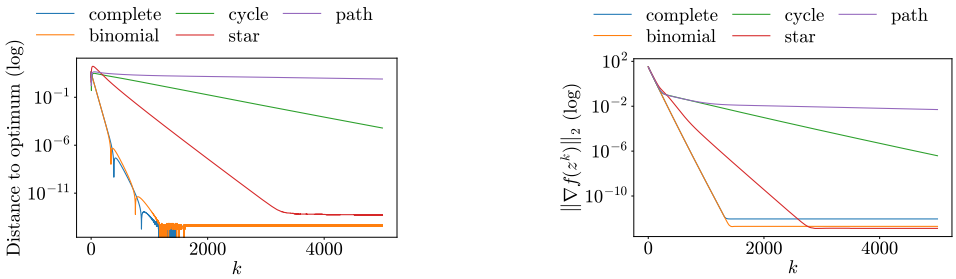
Figure 1.5: Quadratic function minimization with 30 agents in $\mathbb{R}^3$

from Figures 1.4 and 1.5 that the cost function does not reach the optimum within the given number of iterations with the configurations using poorly connected communication graphs, indicating that connectivity is important with 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.



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

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



(a) Distance to optimum

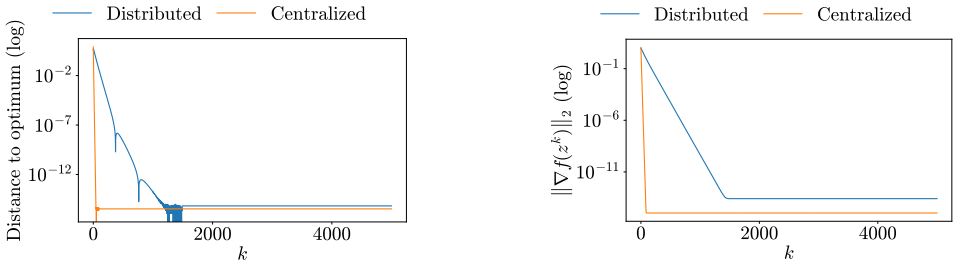
(b) Gradient norm evolution

Figure 1.7: Quadratic function minimization with 30 agents in $\mathbb{R}^{15}$

Lastly, by increasing variables dimensionality, from Figures 1.6 and 1.7 we can observe the same convergence behavior as in the case of low dimensional variables, confirming that on this aspect the problem is not significantly affected.

1.3.2 Comparison with centralized gradient

Following the previous results, we select the best performing configuration and compare it with the centralized gradient method. In Figure 1.8 we can observe the results with 15 agents and a complete communication graph, 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 and it is slightly more precise compared to a distributed algorithm as it has available all the global information without relying on estimates obtained from neighbors and therefore it accumulates less numerical errors.



(a) Distance to optimum

(b) Gradient norm evolution

Figure 1.8: 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 to estimate the position of $N_T \in \mathbb{N}$ fixed targets in a distributed way through $N_R \in \mathbb{N}$ 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 $d_{i,\tau}$ from robot i 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 that controls how much noise is injected into the measurement.

The local loss each robot i uses is the following:

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

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

where $\boldsymbol{z}_i = (\boldsymbol{z}_{i,1}, \ldots, \boldsymbol{z}_{i,N_T}) \in \mathbb{R}^{2N_T}$ is the stack of decision variables of robot i containing the estimated positions of the targets $\boldsymbol{z}_{i,\tau} \in \mathbb{R}^2$ and $\nabla l_i(\boldsymbol{z}_i) \in \mathbb{R}^{2N_T}$ is the concatenation of the gradients computed with respect to each target.

2.2 Code structure

The code provided is structured with the following modules (considering only the differences with Section 1.2):

`loss.py` contains the class definition of the target localization loss.

`plot.py` defines all the functions used to plot the loss and the gradient norm.

`scenarios.py` implements the function to initialize the parameters (i.e., the ground truth target positions, initial robot positions, and noisy estimated target distances) and the target localization loss of the problem.

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

2.3 Experiments

We approach the experimentation of this problem by trying different graph patterns and comparing with the centralized gradient method, similarly to Chapter 1. At first, we evaluate the performance of the algorithm in the following cases, all with the same type of noise:

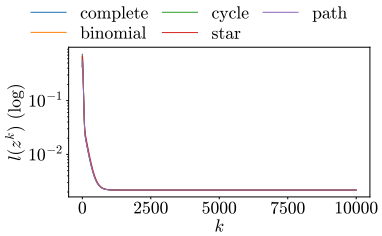
- A few robots (5) with a single target,
- A few robots (5) with multiple targets (3),
- Many robots (15 and 30) with multiple targets (3).

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

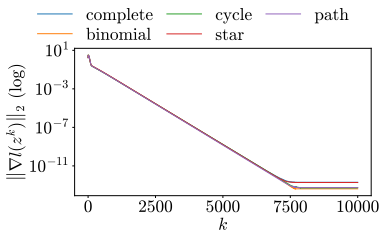
2.3.1 Comparison between different graph patterns

In terms of graph patterns, we can observe from Figures 2.1 and 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.

Instead, by increasing the number of tracking robots, we can see from Figures 2.3 and 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. Moreover, we can also observe from these figures that, as we already mentioned, with less connected graphs convergence is affected negatively.

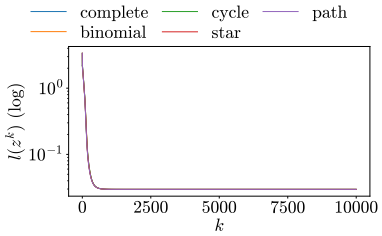


(a) Loss evolution

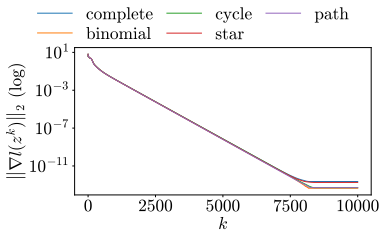


(b) Gradient norm evolution

Figure 2.1: Tracking with 5 robots and 1 target

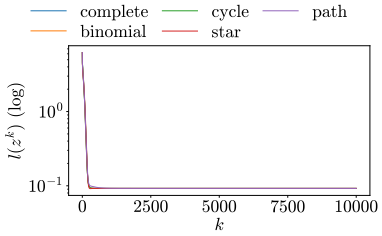


(a) Loss evolution

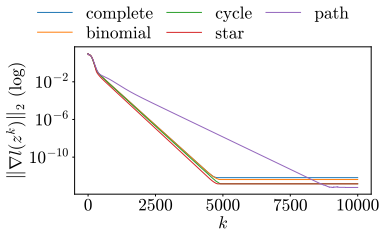


(b) Gradient norm evolution

Figure 2.2: Tracking with 5 robots and 3 targets

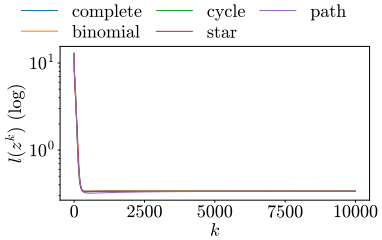


(a) Loss evolution

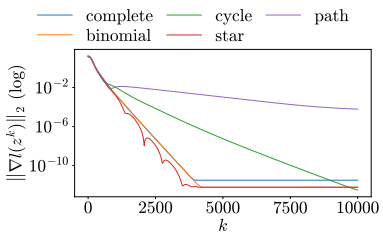


(b) Gradient norm evolution

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 over multiple runs 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.

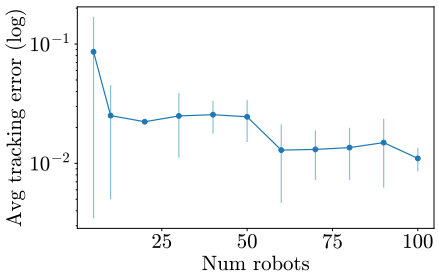


Figure 2.5: Average tracking error for different number of robots (average of 5 runs). Vertical bars represent the standard deviation.

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 some frames of the animation of the scenario are shown.

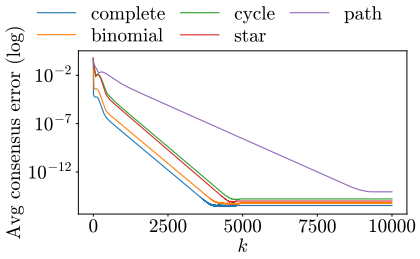


Figure 2.6: Tracking with 15 robots and 3 targets

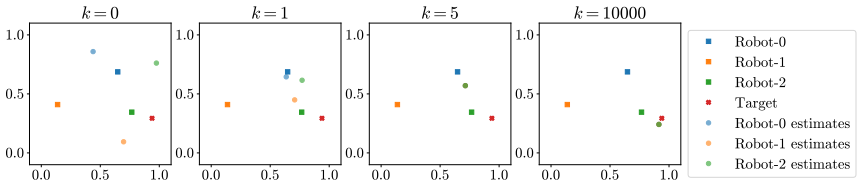
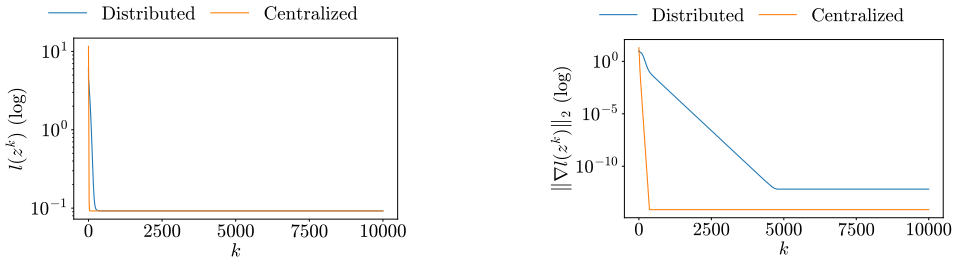


Figure 2.7: Tracking animation with 3 robots and 1 target

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.



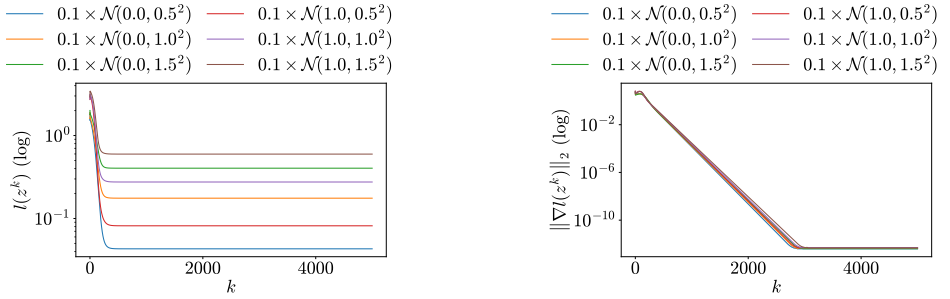
(a) Loss evolution

(b) Gradient norm evolution

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

2.3.3 Comparison with different noises

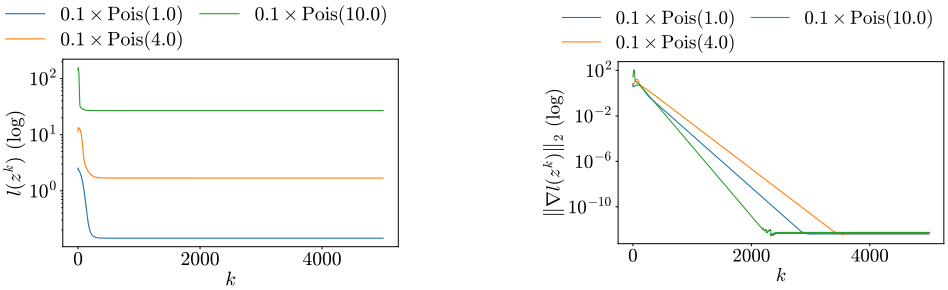
From the experiments with different noises, we observed a worsening in performance that is proportional to the amount of noise injected into the distance measurement, implying as one could expect that more noise leads to worse results. This behavior is consistent with noise drawn from different distributions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



(a) Loss evolution

(b) Gradient norm evolution

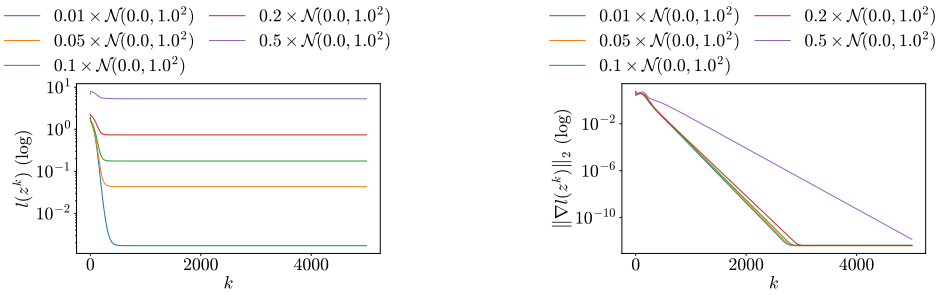
Figure 2.9: Tracking with 15 robots and 3 targets with noise drawn from Gaussian distributions



(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 positioning $N \in \mathbb{N}$ robots with positions $\boldsymbol{z}_i \in R^2$, for $i = 1, \ldots, N$, 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 $\boldsymbol{z} = (\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N) \in \mathbb{R}^{2N}$ is the stack of the robots' positions, $l_i : \mathbb{R}^2 \times \mathbb{R}^{2N} \to \mathbb{R}$ is the local loss 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 agent i is defined as follows:

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

where $\alpha_i \in R$ is a hyperparameter that can be different for each agent. 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 (subject to $\sum_i^N \alpha_i = N$), 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

For the Python version, the code is structured with the following modules:

`algorithm.py` implements the aggregative optimization algorithm.

`loss.py` contains the definition of the loss and the aggregative function.

`plot.py` with the functions used to plot the loss, the gradient norm, and to animate the scenario.

`scenarios.py` provides the functions to to create the communication graph and to initialize the parameters for the problem.

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

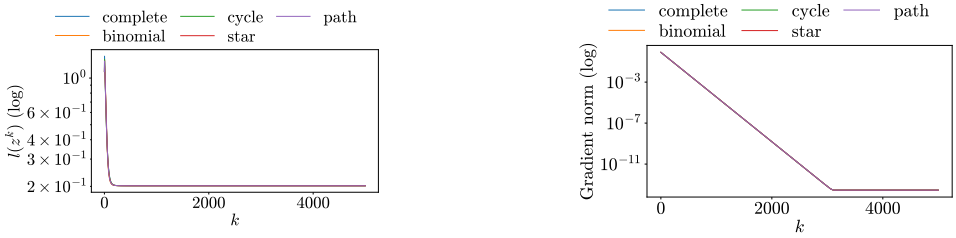
For the ROS2 version, the agent is implemented using the node defined in `Agent.py`. Each agent has a topic in which it publishes its states and listens from the topics of its neighbors. A polling timer is used to control the communication frequency while the update steps are performed using the same 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 that are visualized in RViz.

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, 15, 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, target vicinity or fleet tightness.

3.3.1 Comparison with different graph patterns

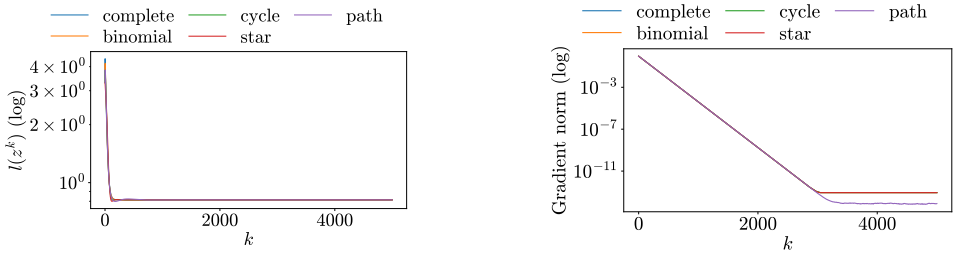
We first test the implementation with 5 agents. From the plots in Figure 3.1, we can observe an identical trend in terms of loss and gradient for every graph pattern we have experimented with. In all cases, we can observe that, as expected from theoretical results, the algorithm converges.



(a) Loss evolution

(b) Gradient norm evolution

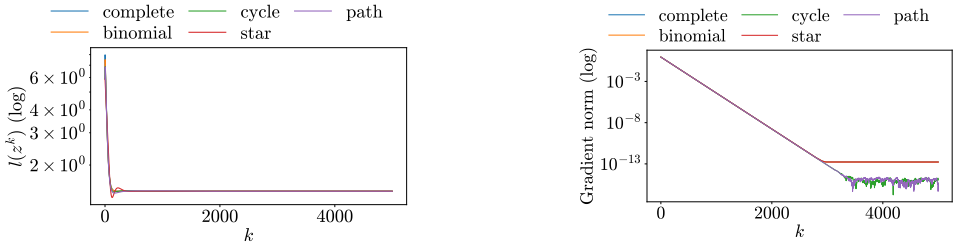
Figure 3.1: Positioning with 5 robots



(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

By considering a scenario with 15 and 30 agents, as shown in Figures 3.2 and 3.3, we cannot detect any relevant changes in behavior as they all reach convergence in a similar 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.

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, neither are favored and the agents converge to a position that is midway between the barycenter and their private target.

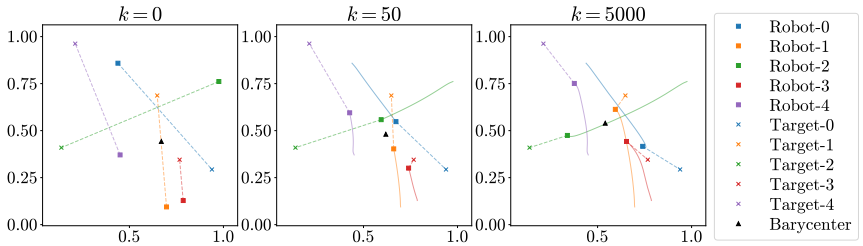


Figure 3.4: Frames with 5 robots and balanced loss. Dashed lines connect robots to the targets and solid lines are the trajectories.

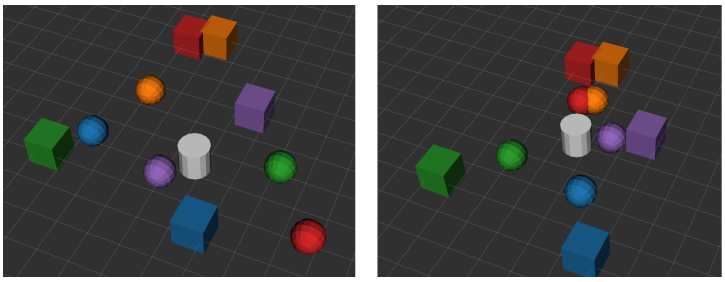


Figure 3.5: Frames with 5 robots and barycenter that prioritizes robot 0. Spheres are agents, cubes are targets, and the cylinder is the barycenter.

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.

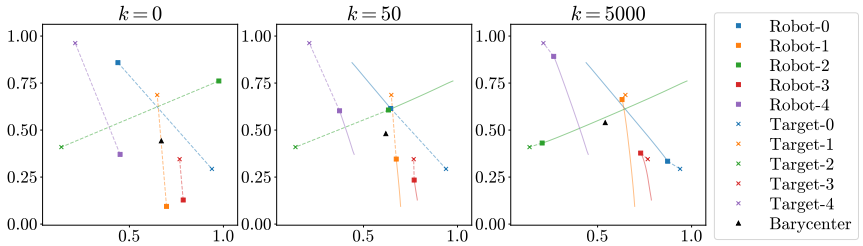


Figure 3.6: Frames with 5 robots and loss that prioritizes the private targets. Dashed lines connect robots to the targets and solid lines are the trajectories.

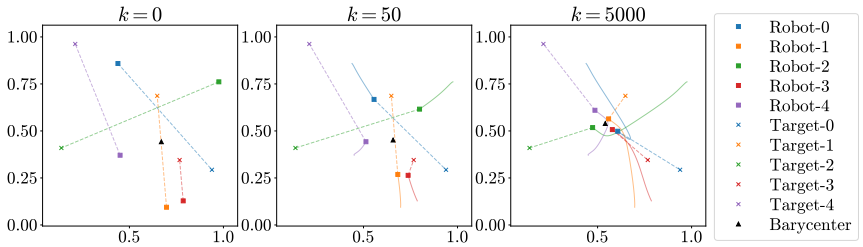


Figure 3.7: Frames with 5 robots and loss that prioritizes the barycenter. Dashed lines connect robots to the targets and solid lines are the trajectories.

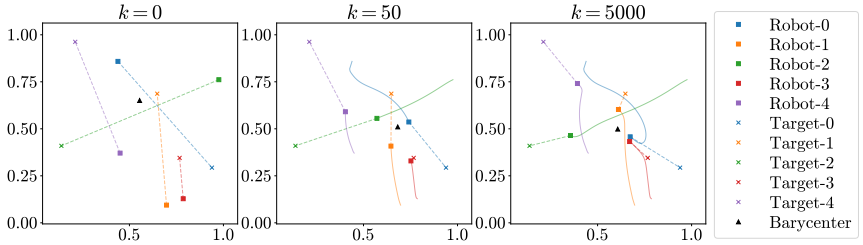


Figure 3.8: Frames with 5 robots and barycenter that prioritizes robot 0. Dashed lines connect robots to the targets and solid lines are the trajectories.

Conclusions

In this project, we tackled and experimented with the problem of multi-robot target localization and positioning. 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 and higher graph connectivity helps in reaching it faster, which is more noticeable in case of larger numbers of agents. Other tests showed that under different noise distributions and intensities, tracking accuracy, as expected, is higher with more agents and worsens for stronger amounts of noise.

Regarding the multi-robot positioning problem, we first observed similar trends for all the configurations in terms of loss and gradient behavior, independently of the number of robots or the communication graph pattern. Additional experiments were performed and assessed visually by changing the loss hyperparameters affecting robots' importance, target vicinity and fleet tightness. Results showed that the loss function provides flexibility in tuning the constraints and the robots' final positions were the expected ones.