UNIVERSITÀ DI BOLOGNA

School of Engineering

Master Degree in Automation Engineering

Distributed Autonomous Systems

Distributed Classification via Logistic Regression & Aggregative Optimization for Multi-Robot Systems

Professors:
Giuseppe Notarstefano
Ivano Notarnicola

Student: Agatino Ricciardi

Academic year 2023/2024

Abstract

In this report we explore the implementation, in Python and ROS2, of two main tasks: a data analitics application revolving around a distributed classification via logistic regression and an aggregative optimization algorithm for multi-robot systems.
In order to build up the first task we procede gradually, first developing a gradient tracking algorithm to solve a consensus optimization problem with multiple agents. Then we move on to the centralized classification problem, where we implement a gradient method to minimize a logistic regression function. Finally, we implement a distributed gradient tracking algorithm to classify the dataset cooperatively across multiple agents.
For the second task, we focus on aggregative optimization for a team of robots in a two-dimensional environment. The goal is to design the cost function such that the robots must remain in tight formation while trying to reach some private targets. Then we introduce a constraint, a corridor through which the team must pass in order to reach the targets. Once the Python implementation is completed, we can move on to the ROS2 environment, using its framework to enforce real-time communication among the robots.
Various simulations, with different parameters values, were run to verify and test the solutions quality for both tasks.

Contents

Introduction

This report explores the implementation of two significant tasks for distributed autonomous systems: distributed classification via logistic regression and aggregative optimization for multi-robot systems. The primary focus is on developing algorithms in Python and ROS2 environments to address these tasks effectively.

The first task revolves around distributed classification using logistic regression. The objective is to implement a gradient tracking algorithm to solve a consensus optimization problem, initially using a quadratic cost function. The algorithm is implemented in such a way to work independently from the number of agents in the network, albeit strong connectivity is ensured. Then, we setup a centralized classification problem, where a gradient method minimizes a logistic regression function. Two separating functions are tested for this task. The variables were trained on a training set (70% of the dataset size) randomly generated and then tested on a test set (30% of the dataset size). Subsequently, the approach is extended to a distributed framework, still using the logistic regression function, utilizing the gradient tracking algorithm to solve the classification problem, where the dataset are split among the agents in a randomic way.

The second task, which is divided into two sub-tasks, focuses on aggregative optimization for multi-robot systems. The first sub-task involves the implementation of an aggregative tracking algorithm in Python to ensure a team of robots maintains tight formation while reaching specific targets. The second sub-task introduces an additional constraint of navigating through a corridor, requiring the team to maintain formation and reach targets within this confined space. the proposed solution implements a barrier function term to the cost function, in order to constrain the robots to navigate through the corridor, without collision. The implementation is extended to the ROS2 environment, using RViz2 for visualizing the robots' behavior and ROS2 topics to model the communication inside this framework.

Chapter 1

Distributed Classification via Logistic Regression: Task 1.1 - Distributed Optimization

1.1 Problem Setup

The goal of this subtask was to implement a gradient tracking algorithm on a quadratic program, in order to solve a consensus optimization problem. The problem was defined as follows:

$$\min_{z} \sum_{i=1}^{N} \ell_i(z) \tag{1.1}$$

With $z \in \mathbb{R}^d$ and $\ell_i : \mathbb{R}^d \to \mathbb{R}$ being a quadratic function of the form:

$$\ell_i(z) = \frac{1}{2} z^T Q z + R^T z \qquad \forall i \in \{1, \dots, N\} \tag{1.2}$$

To test the effectiveness of the implementation, we need to understand the communication topology among our agents, thus te first step is to determine on which graph to evaluate our solution. The idea is to test the implementation on three different scenarios with three different graphs: a cycle, a star and a path graph. Using `networkx` library, we built a graph dictionary.

In order to ensure double stochasticity for the adjacency matrix, condition needed to reach average consensus, we implement a Python method that computes the matrix weights via Metropolis-Hastings technique, which is defined as follows:

$$A_{ij} = \begin{cases} \frac{1}{1+\max\{d_i, d_j\}} & \text{if } (i,j) \in E \text{ and } i \neq j \\ 1 - \sum_{h \in N_i \setminus \{i\}} A_{ih} & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{1.3}$$

Where d_i is the degree of node i and N_i is the set of neighbors of node i.

1.2 Gradient Tracking Algorithm

Once the problem is setup, we can define the gradient tracking algorithm as follows:

$$z_i^{k+1} = \sum_{j \in N_i} a_{ij} z_j^k - \alpha s_i^k \qquad z_i^0 \in \mathbb{R} \tag{1.4}$$

$$s_i^{k+1} = \sum_{j \in N_i} a_{ij} s_j^k + \nabla \ell_i(z_i^{k+1}) - \nabla \ell_i(z_i^k) \qquad s_i^0 = \nabla \ell_i(z_i^0) \tag{1.5}$$

Where z_i^k is the state of agent i at iteration k, s_i^k is the gradient of the cost function of agent i at iteration k, $\nabla \ell_i(z_i^k) : \mathbb{R}^d \to \mathbb{R}^d$ is the gradient of the cost function of agent i at z_i, N_i is the set of neighbors of agent i, a_{ij} is the adjacency matrix of the graph and α is the stepsize.

The following table describes the implemented gradient tracking algorithm step-by-step:

Algorithm 1 Gradient Tracking Algorithm

Initialization of $Z[0,i]$ and $S[0,i] = \nabla \ell(Z[0,i])$
for $k \leftarrow 1$ to iterations $- 1$ **do**
  for $i \leftarrow 1$ to N **do**
    $N_i \leftarrow \text{neighbors}(i)$
    $Z[k+1,i] \leftarrow A[i,i] \cdot Z[k,i]$
    $S[k+1,i] \leftarrow A[i,i] \cdot S[k,i]$
    for $j \in N_i$ **do**
      $Z[k+1,i] \leftarrow Z[k+1,i] + A[i,j] \cdot Z[k,j]$
      $S[k+1,i] \leftarrow S[k+1,i] + A[i,j] \cdot S[k,j]$
    end for
    $Z[k+1,i] \leftarrow Z[k+1,i] - \alpha \cdot S[k,i]$
    $\nabla \ell(Z[k+1,i]) \leftarrow Q_i \cdot Z[k+1,i] + R_i$
    $\nabla \ell(Z[k,i]) \leftarrow Q_i \cdot Z[k,i] + R_i$
    $S[k+1,i] \leftarrow S[k+1,i] + \nabla \ell(Z[k+1,i]) - \nabla \ell(Z[k,i])$
    $\ell[k] \leftarrow \ell[k] + 0.5 \cdot Z[k,i]^T \cdot Q_i \cdot Z[k,i] + R_i \cdot Z[k,i]$
  end for
end for
Storing the values of the cost and its gradient for plotting

1.3 Simulation and Results

In this section we show and comment the results achieved by running the gradient tracking algorithm on the three different graphs. We also show the

evolution of the cost function, the norm of the gradient of the cost function and the consensus plot across the iterations.
The parameters used for the simulations are defined in the following table:

Parameter	**Description**	**Value**
`NN`	Number of agents	10
`iterations`	Number of iterations	10000
`alpha`	Stepsize	0.01
`d`	State dimension	2
`QQ`	Quadratic term for the cost	random (uniform) positive definite
`R`	Linear term for the cost	random (uniform) generated
`graphs`	Graph dictionary	Cycle, Path, Star

1.3.1 Scenario: Cycle Graph

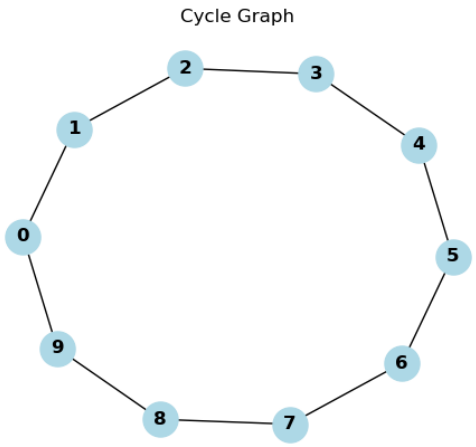
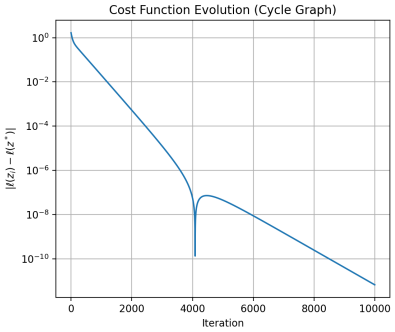
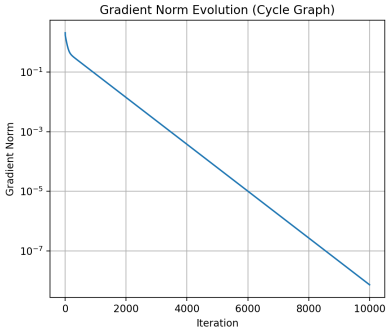


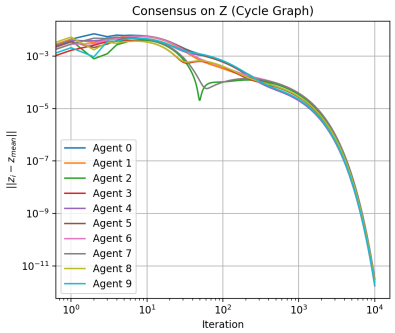
Figure 1.1: Cycle graph



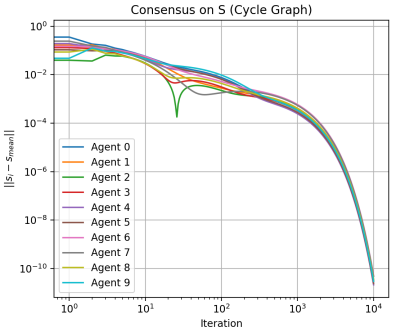
(a) Absolute value of the difference between the cost function and the optimal value across the iterations



(b) Evolution of the norm of the gradient across the iterations



(c) Norm of the difference between the states, for each agent, and the mean value across the iterations



(d) Norm of the difference between the gradient estimate, for each agent, and the mean value across the iterations

Figure 1.2: Results for the cycle graph

1.3.2 Scenario: Path Graph

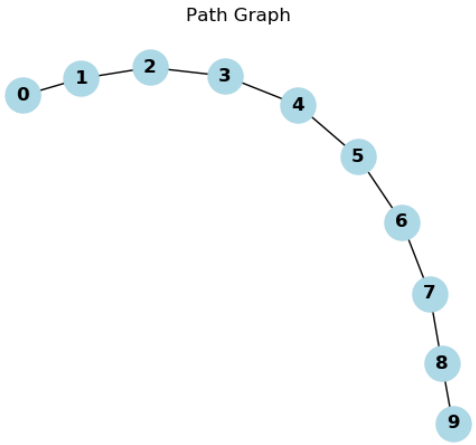
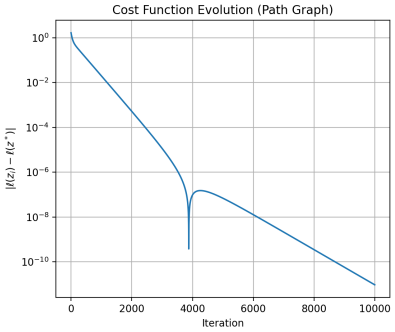
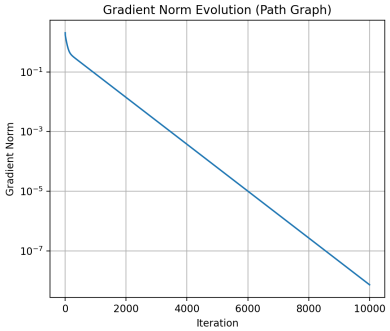


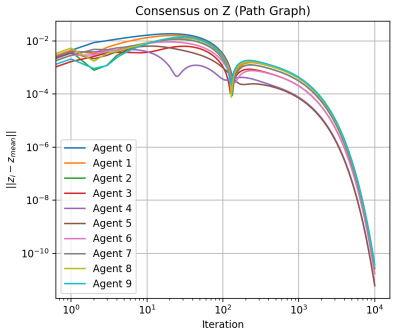
Figure 1.3: Path graph



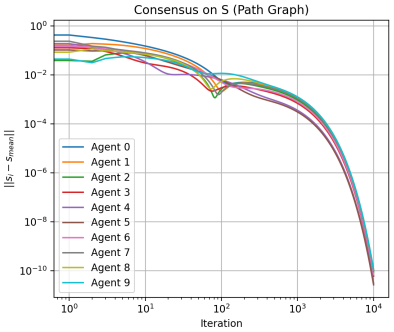
(a) Absolute value of the difference between the cost function and the optimal value across the iterations



(b) Evolution of the norm of the gradient across the iterations



(c) Norm of the difference between the states, for each agent, and the mean value across the iterations



(d) Norm of the difference between the gradient estimate, for each agent, and the mean value across the iterations

Figure 1.4: Results for the path graph

1.3.3 Scenario: Star Graph

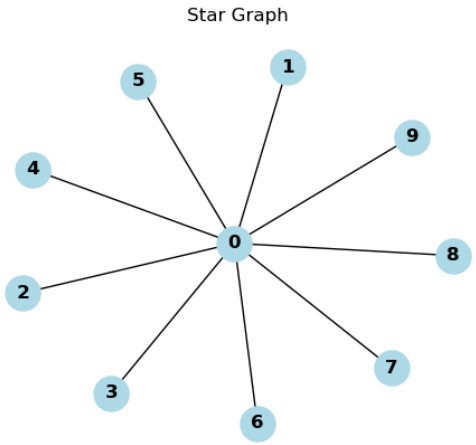
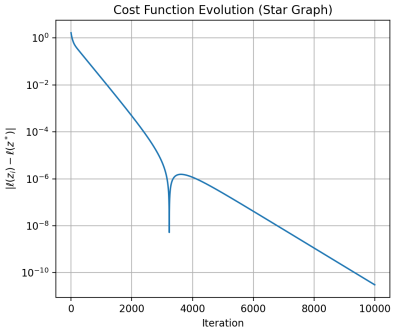
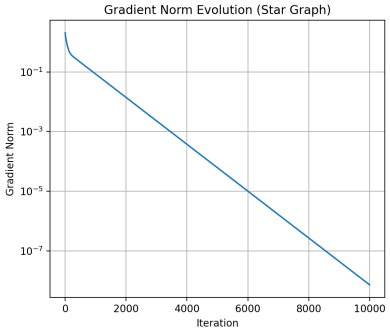


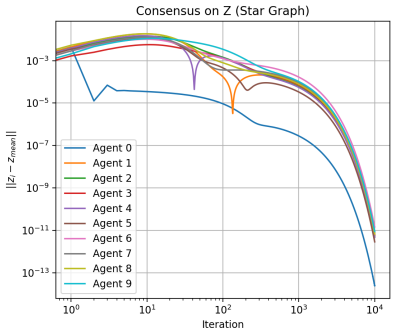
Figure 1.5: Star graph



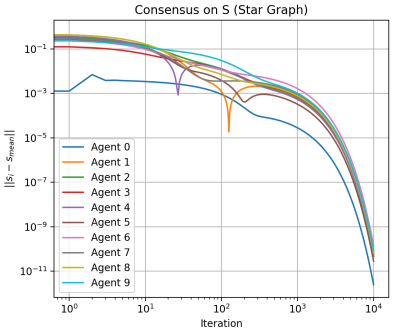
(a) Absolute value of the difference between the cost function and the optimal value across the iterations



(b) Evolution of the norm of the gradient across the iterations



(c) Norm of the difference between the states, for each agent, and the mean value across the iterations



(d) Norm of the difference between the gradient estimate, for each agent, and the mean value across the iterations

Figure 1.6: Results for the star graph

The results provided by the plots show consistency with the expectation for the three different scenarios. When plotting the cost ($|\nabla\ell(z_i) - \nabla\ell(z^*)|$) in a `semilog-y` scale, we can see that it quickly goes to zero. The optimal solution z^* is computed by exploiting the first-order necessary condition for optimality, which is given by setting the gradient to zero and computing analitically the value.

The plot of the norm of the gradient, also in a `semilog-y` scale, it keeps decreasing as expected. The consensus plot shows how the agents are able to reach a consensus on the optimal solution, as the states of the agents converge to the same value. The same happens for the gradients, which also converge to the same value.

Even if not explicitly required, we also plotted the evolution of the con-

sensus, with respect to the state and to the gradient, across the iterations. The results, plotted in a `loglog` scale for better visualization, show that after a while all the agents reach consensus. Even considering the consensus plots for the star graph, we can see that the agent that strays away from the consensus result, is the one connecting all other agents.

Chapter 2

Distributed Classification via Logistic Regression: Task 1.2 - Centralized Classification

2.1 Problem Setup

The goal of this subtask was to implement a centralized gradient method in order to minimize a logistic regression function.

2.1.1 Dataset Generation and Labeling

We start from the generation of a dataset $\mathcal{D}^m \in \mathbb{R}^d$ of $\mathcal{M} \in \mathbb{N}$ points, with $\mathcal{D}^m \in \mathbb{R}^d$ for all $m = 1, \ldots, \mathcal{M}$. We use the `generate_dataset` method, which generates a dataset of $\mathcal{M}$ points in a two-dimensional space with a normal standard distribution.

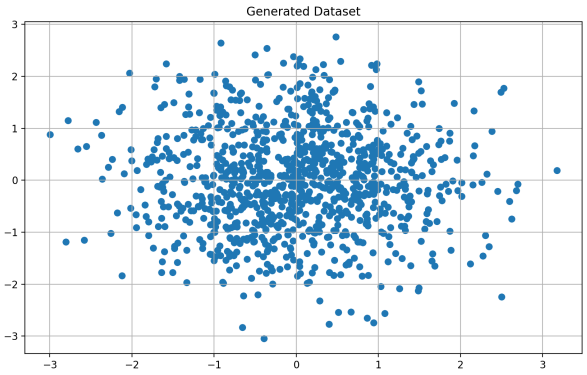


Figure 2.1: Randomly generated dataset

The goal now is to label the points with a binary label $p^m \in \{-1, 1\}$ for all $m = 1, \ldots, \mathcal{M}$. This operation is performed by the `label_points` method, which defines the following separating function:

$$\{x \in \mathbb{R}^d \mid w^\top \phi(x) + b = 0\} \tag{2.1}$$

where $\phi : \mathbb{R}^d \to \mathbb{R}^q$ is a nonlinear function and $w \in \mathbb{R}^q$, $b \in \mathbb{R}$ are our parameters. After establishing the structure of the separating function, the points are labeled as follows:

$$w^\top \phi(D^m) + b \geq 0, \quad \text{if } p^m = 1 \tag{2.2}$$

$$w^\top \phi(D^m) + b < 0, \quad \text{if } p^m = -1 \tag{2.3}$$

Now the idea is to implement a data points manipulation through our function ϕ. Two ideas are proposed:

- Linear transformation: $\phi(\mathcal{D}) = \mathcal{D}$
- Nonlinear transformation: $\phi(\mathcal{D}) = \begin{bmatrix} [\mathcal{D}]_1 & [\mathcal{D}]_2 & ([\mathcal{D}]_1)^2 & ([\mathcal{D}]_2)^2 \end{bmatrix}^\top$

In the nonlinear case, we assume the separating function to be an ellipse, with $\mathcal{D} = ([\mathcal{D}]_1, [\mathcal{D}]_2) \in \mathbb{R}^2$.

2.1.2 Logistic Regression

Once the generation and the labeling of the dataset is done, in order to test the effectiveness of our algorithm, we need to split our dataset into a training set (70% of the points) and a test set (30% of the points). The idea is to run the optimization on the training set, and used the data obtained on the test set to evaluate the performance of the algorithm.

The optimization problem to be minimized is then defined as:

$$\min_{w,b} \sum_{m=1}^{M} \log(1 + \exp(-p^m(w^\top \phi(\mathcal{D}^m) + b))) \tag{2.4}$$

Where $(w, b) \in \mathbb{R}^q \times \mathbb{R}$ is the optimization variable and $\phi(\mathcal{D}^m)$ is the transformation function. The computation of the logistic regression function is handled by the `logistic_regression_function` method.

2.2 Centralized Gradient Method

The goal is to implement a centralized gradient method to minimize the logistic regression function in order to classify correctly the dataset points. The choice fell on the batch gradient method, which is defined as follows:

$$z^{k+1} = z^k - \alpha \cdot \nabla \ell(z^k) \tag{2.5}$$

Where $\alpha > 0$ is the stepsize, $\nabla\ell(z^k) : \mathbb{R}^{q+1} \to \mathbb{R}^{q+1}$ and the decision vector z is defined as the stack vector $z = \begin{bmatrix} w & b \end{bmatrix}^\top \in \mathbb{R}^{q+1}$.

To solve this problem we rely on the `centralized_gradient_descent` method, that exploits the `compute_gradient` method to compute the gradient of the variables w and b.

The following table describes the implemented batch gradient algorithm step-by-step:

Algorithm 2 Batch Gradient Algorithm

Initialization of w and b
for $i \leftarrow 1$ to iterations **do**
$\nabla_w(\ell(z^k)) \leftarrow -p^m \cdot \phi(\mathcal{D}^m) \cdot \frac{\exp(-p^m(w^\top\phi(\mathcal{D}^m)+b))}{1+\exp(-p^m(w^\top\phi(\mathcal{D}^m)+b))}$
$\nabla_b(\ell(z^k)) \leftarrow -p^m \cdot \frac{\exp(-p^m(w^\top\phi(\mathcal{D}^m)+b))}{1+\exp(-p^m(w^\top\phi(\mathcal{D}^m)+b))}$
$w \leftarrow w - \alpha \cdot \nabla_w(\ell(z^k))$
$b \leftarrow b - \alpha \cdot \nabla_b(\ell(z^k))$
$\ell(z^k) = \log(1 + \exp(-p^m(w^\top\phi(\mathcal{D}^m) + b)))$
$\nabla(\ell(z^k)) \leftarrow [\nabla_w(\ell(z^k)), \nabla_b(\ell(z^k))]^T$
end for

2.3 Simulation and Results

In this section we show and comment the results achieved by running the centralized gradient method with two different separating function, a linear one and a nonlinear one. The `run_optimization` method is the one that handles the dataset setup, which separating function to use, the dataset split into training and test set and calls the `centralized_gradient_descent` method. We also show the evolution of the cost function and the norm of the gradient of the cost function across the iterations.

The parameters used for the simulations are defined in the following table:

Parameter	**Description**	**Linear**	**Ellipse**
$\mathcal{M}$	Number of points	1000	1000
`iterations`	Number of iterations	5000	5000
`alpha`	Stepsize	0.01	0.01
`d`	State dimension	2	2
`q`	Transformation dimension	2	4

The variables w and b are initialized randomly for each agent, from a standard normal distribution.

To evaluate the implementation, a `missclassification_rate` method was designed to check how many points were misclassified by the separating

function. This operation is performed after checking how many points are incorrectly classified and dividing these samples by the entirety of the used set.

2.3.1 Scenario: Linear Classifier

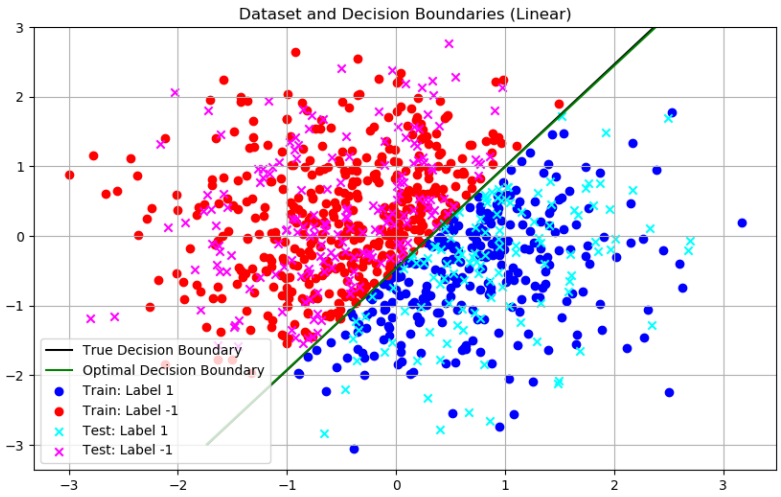
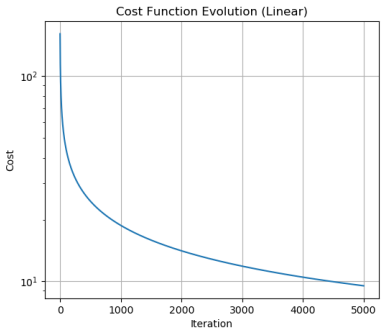
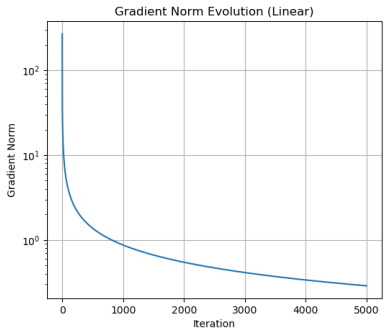


Figure 2.2: Linear classifier



(a) Evolution of the cost function across the iterations



(b) Evolution of the norm of the gradient across the iterations

Figure 2.3: Results for the linear classifier

Dataset	**Missclassification Rate**
Training Set	0.12 %
Test Set	0.34 %

2.3.2 Elliptic Classifier

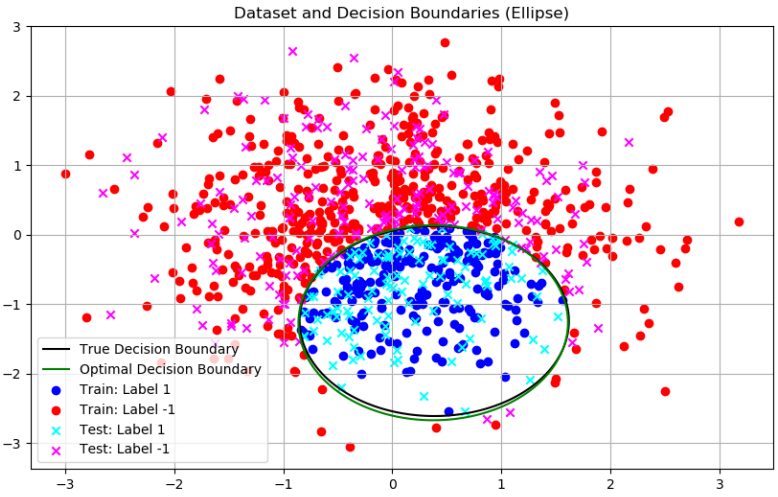
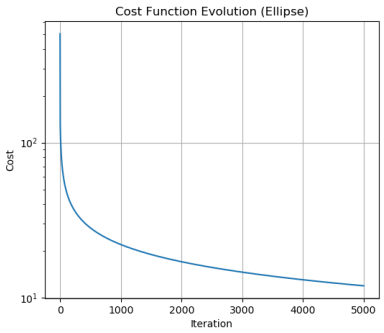
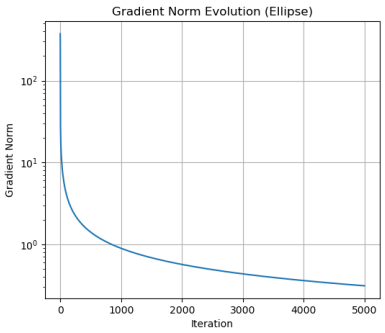


Figure 2.4: Elliptic classifier



(a) Evolution of the cost function across the iterations



(b) Evolution of the norm of the gradient across the iterations

Figure 2.5: Results for the elliptic classifier

Dataset	**Missclassification Rate**
Training Set	0.14%
Test Set	0.33%

With the linear classifier, the algorithm achieved a missclassification rate of 0.66% on the test set, and of 0.00% on the training set, which are satisfying results, considering that we are under the 1.00% error threshold. If

we look at the dataset plot, we can see the true decision boundary perfectly overlaps with the optimal one. Looking at the cost and the gradient evolutions, plotted in `semilog-y` scale, we can see that both decrease across the iterations, as expected, even if they don't reach the low values achieved in the previous task. This is easily explained by the fact that the cost function is not a quadratic function, but a logistic regression function, which is more complex to minimize.

With the elliptic classifier the algorithm achieved a missclassification rate of 0.33% on the test set and of 0.14% on the training set. Again, we obtained satisfactory results, since we are again under the 1.00% error threshold. The cost and the gradient evolutions, plotted in `semilog-y` scale, show a similar trend to the linear classifier, with the related comments that were previously made.

Chapter 3

Distributed Classification via Logistic Regression: Task 1.3 - Distributed Classification

3.1 Problem Setup

Now that we have the tools to implement a gradient tracking algorithm and to solve a classification problem, we can move on to the distributed version of the same problem. The goal of this subtask was to implement a distributed version of the gradient tracking algorithm to minimize a logistic regression function.
Since this task can be viewed as the merging of the two previous tasks, many of the concepts and methods used in the previous tasks are reused here. The main difference is that now we have to implement the algorithm in a distributed way, thus, we have to split the dataset into N subsets, one for each agent, and then implement the algorithm to classify the dataset in a distributed way.

Like done in task 1.1, we have to define the communication topology among the agents. The idea is to test the implementation on a simple cycle graph. The `metropolis_hastings_weights` method is used to compute the matrix weights via Metropolis-Hastings technique.

3.1.1 Dataset Generation and Labeling

Like done in task 1.2, we use the `generate_dataset` method, which generates a dataset of $\mathcal{M}$ points in a two-dimensional space with a normal standard distribution. Then again, we have to split the dataset into a training set (70% of the points) and a test set (30% of the points).

The `label_points` method is used to label the points with a binary label

$p^m \in \{-1, 1\}$ for all $m = 1, \ldots, \mathcal{M}$. We have to define the ϕ function, which is used to manipulate the data points. We can choose between a linear transformation and a nonlinear transformation. The linear transformation is defined as $\phi(\mathcal{D}) = \mathcal{D}$, while the nonlinear transformation is defined as $\phi(\mathcal{D}) = \begin{bmatrix} [\mathcal{D}]_1 & [\mathcal{D}]_2 & ([\mathcal{D}]_1)^2 & ([\mathcal{D}]_2)^2 \end{bmatrix}^\top$.

Once this is done we can proceed to split the dataset into N subsets, one associated to each agent. The `split_dataset` method is used to perform this operation.

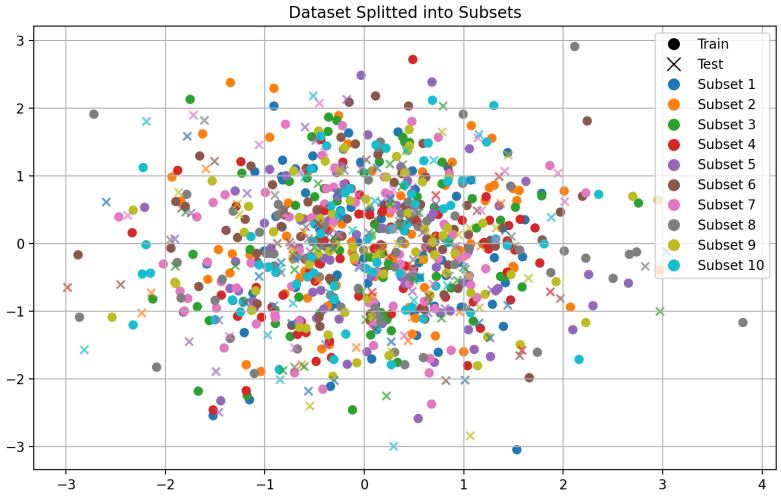


Figure 3.1: Dataset split into subsets

Different colors represent the different batches, each associated to a different agent, while different markers (crosses and circles) represent the different classes of set.

3.2 Distributed Gradient Tracking Algorithm

The idea now is to exploit the implementation of the algorithm of task 1.1 to solve the minimization problem of task 1.2, thus using the same logistic regression function. Doing it in a distributed way means that each agent will compute the cost function and its gradient on its own subset of the dataset, and then will run the gradient tracking algorithm.

3.3 Simulation and Results

In this section we show and comment the results achieved by running the distributed gradient tracking algorithm on the cycle graph, with two differ-

ent separating functions. The `run_optimization` method is the one that handles the dataset setup, which separating function to use, the dataset split into training and test set and the algorithm running in general. We also show the evolution of the cost function, of the norm of the gradient and the consensus plot across the iterations.

The parameters used for the simulations are defined in the following table:

Parameter	**Description**	**Linear**	**Ellipse**
`NN`	Number of agents	10	10
$\mathcal{M}$	Number of points	1000	1000
`iterations`	Number of iterations	10000	10000
`alpha`	Stepsize	0.001	0.001
`d`	State dimension	2	2
`q`	Transformation dimension	2	4
`G`	Graph object	Cycle	Cycle

To evaluate the implementation, the same `missclassification_rate` method was used to check how many points were misclassified by the separating function.

3.3.1 Scenario: Linear Classifier

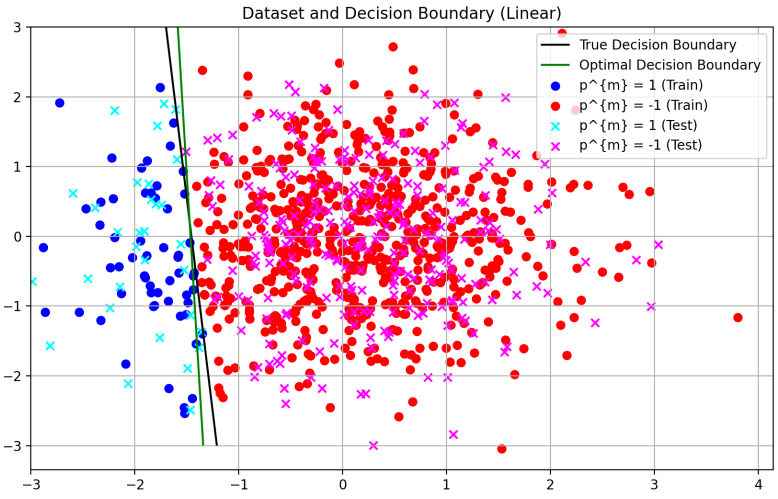
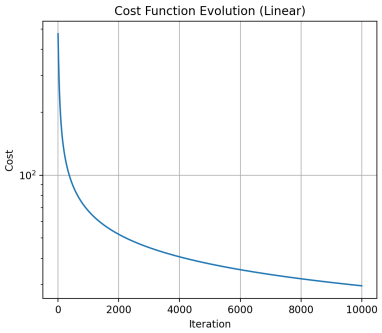
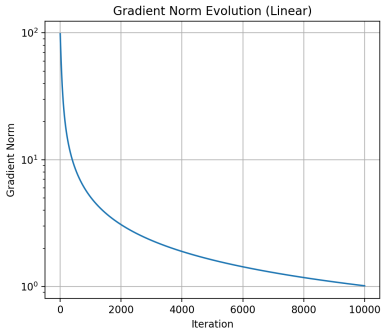


Figure 3.2: Dataset for the linear classifier

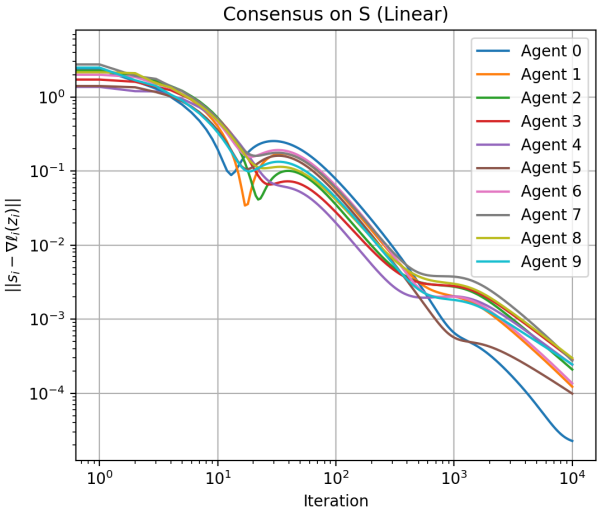
Dataset	**Missclassification Rate**
Training Set	0.43 %
Test Set	0.67 %



(a) Evolution of the cost function across the iterations



(b) Evolution of the norm of the gradient across the iterations



(c) Norm of the difference between the gradient estimate, for each agent, and the true gradient across the iterations

Figure 3.3: Results for the linear classifier

3.3.2 Scenario: Elliptic Classifier

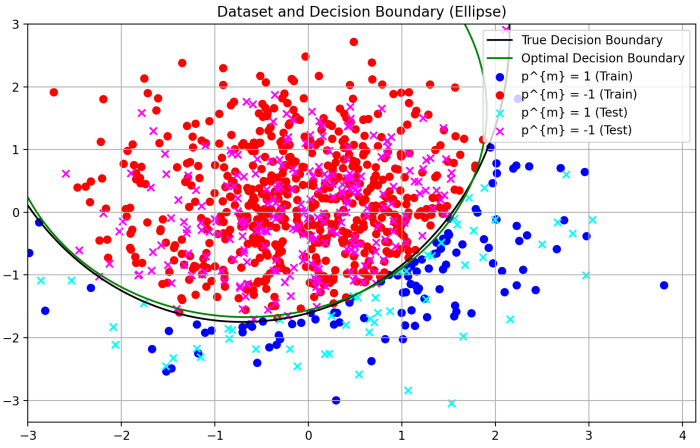
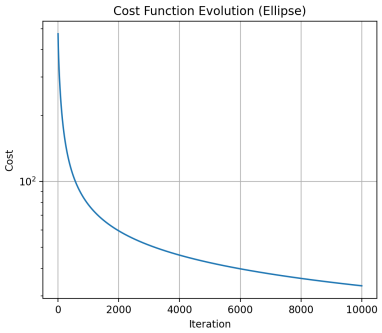
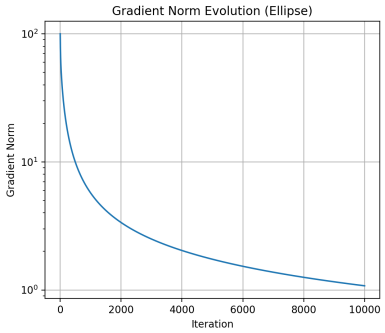


Figure 3.4: Dataset for the elliptic classifier

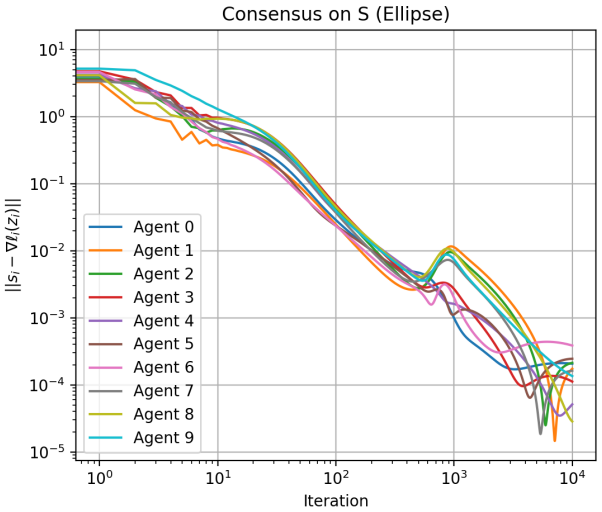
Dataset	**Missclassification Rate**
Training Set	0.47%
Test Set	0.81%



(a) Evolution of the cost function across the iterations



(b) Evolution of the norm of the gradient across the iterations



(c) Norm of the difference between the gradient estimate, for each agent, and the true gradient across the iterations

Figure 3.5: Results for the elliptic classifier

In terms of missclassification, the results are good and consistent. The algorithm achieved a missclassification rate of 0.67% on the test set and of 0.43% on the training set with the linear classifier, and of 0.81% on the test set and of 0.47% on the training set with the elliptic classifier. Since with both separating functions we are under the 1.00% error threshold, we can consider the results satisfactory.
The results provided by the plots show consistency with the expectation for the two different scenarios. The cost and the gradient evolutions, plotted in `semilog-y` scale, show that both decrease across the iterations, even though not as steeply as the plots of task 1.1. Again, since the cost function is not

a quadratic function, but a logistic regression function, it is more complex to minimize.
About the consensus plot on the gradient tracking variable, we are less happy with the results. Even though it decreases across the iterations, they don't reach a common consensus value. This could be due to the fact that there is no such thing as a unique classifier which perfectly splits two classes. An idea to improve the consensus result could be to introduce regularization terms that enforce consensus constraints more strongly, ensuring that nodes' estimates converge more rapidly.

Chapter 4

Aggregative Optimization for Multi-Robot Systems: Task 2.1 - Problem Setup

The goal of this task is to implement a distributed control algorithm that allows a team of N robots to keep a tight formation while, at the same time, to be close to some private targets. For the purposes of this task, we consider a two-dimensional environment, where $z_i^k \in \mathbb{R}^2$ is the position of robot i at iteration k. The problem can be formalized as the following aggregative optimization problem:

$$\min_{z \in \mathbb{R}^d} \sum_{i=1}^{N} \ell_i(z_i, \sigma(z)) \tag{4.1}$$

$$\sigma(z) \triangleq \frac{1}{N} \sum_{i=1}^{N} \varphi_i(z_i) \tag{4.2}$$

where $\ell_i : \mathbb{R}^2 \times \mathbb{R}^d \to \mathbb{R}$ and $\varphi_i : \mathbb{R}^2 \to \mathbb{R}^d$. In this scenario, agent i is aware only of the decision variable z_i and of the functions ℓ_i and φ_i. The global aggregative variable $\sigma(z)$ can be seen as the barycenter of the team, and, hence, $\varphi_i(z_i) = z_i$ for all $i = 1, \ldots, N$.

4.1 Cost Function

Designing a suitable cost function is of paramount importance, since the assignment requires that the robots keep a tight formation while being close to their private targets. The cost function is defined as:

$$\ell_i(z_i, \sigma) = \gamma_{\text{tar}} \cdot \|z_i - r_i\|^2 + \gamma_{\text{agg}} \cdot \|z_i - \sigma(z)\|^2 \tag{4.3}$$

With the first term being the distance between the robot and its private target, and the second term being the distance between the robot and the

barycenter of the team. The parameters γ_{tar} and γ_{agg} are used as weights to balance the two terms. We can define the gradient of the cost function as:

$$\nabla_1 \ell_i(z_i, \sigma(z)) = 2 \cdot \gamma_{\text{tar}} \cdot (z_i - r_i) + 2 \cdot \gamma_{\text{agg}} \cdot (z_i - \sigma(z)) \tag{4.4}$$

$$\nabla_2 \ell_i(z_i, \sigma(z)) = 2 \cdot \gamma_{\text{agg}} \cdot (\sigma(z) - z_i) \tag{4.5}$$

With $\nabla_1 \ell_i(z_i, \sigma(z)) : \mathbb{R}^d \to \mathbb{R}^d$ and $\nabla_2 \ell_i(z_i, \sigma(z)) : \mathbb{R}^d \to \mathbb{R}^d$. Since the function $\sigma(z)$, the barycenter of the team, is a global variable not known by the agents, we can substitute it with the barycenter estimation s_i, thus our cost function becomes:

$$\ell_i(z_i, s_i) = \gamma_{\text{tar}} \cdot \|z_i - r_i\|^2 + \gamma_{\text{agg}} \cdot \|z_i - s_i\|^2 \tag{4.6}$$

In my implementation, the computations of the cost function and its gradient are handled by the `cost_function` and `compute_gradient` methods respectively.

4.2 Aggregative Gradient Tracking Algorithm

In order to solve the problem, we implemented the aggregative gradient tracking algorithm, which is defined as follows:

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

Where z_i^k is the position of robot i at iteration k, s_i^k is the barycenter estimation of robot i at iteration k, and v_i^k is the gradient tracking variable of robot i at iteration k. As done in previous tasks, the adjacency matrix weights are computed via Metropolis-Hastings technique. The topology chosen for this scenario is the cycle graph.

In my code, the implementation of the algorithm is handled by the `aggregative_tracking_optimization` method.

The following table describes the implemented aggregative gradient tracking algorithm step-by-step:

Algorithm 3 Aggregative Gradient Tracking Algorithm

Initialization of $Z[0,i]$, $S[0,i] = \phi(Z[0,i])$ and $V[0,i] = \nabla_2 \ell(Z[0,i], S[0,i])$
for $k \leftarrow 1$ to iterations **do**
  for $i \leftarrow 1$ to N **do**
    $\nabla_1 \ell(Z[k,i], S[k,i]) \leftarrow 2 \cdot \gamma \cdot (Z[k,i] - r[k,i]) + 2 \cdot \gamma_{\text{agg}} \cdot (Z[k,i] - S[k,i])$
    $\nabla_2 \ell(Z[k,i], S[k,i]) \leftarrow 2 \cdot \gamma_{\text{agg}} \cdot (S[k,i] - Z[k,i])$
    $Z[k+1,i] \leftarrow Z[k,i] - \alpha \cdot (\nabla_1 \ell(Z[k,i], S[k,i]) + \nabla \phi(Z[k,i]) \cdot V[k,i])$
    for $j \leftarrow 1$ to N **do**
      $S[k+1,i] \leftarrow A[i,j] \cdot S[k,j] + \phi(Z[k+1,i]) - \phi(Z[k,i])$
    end for
    $\nabla_2 \ell(Z[k+1,i], S[k+1,i]) \leftarrow 2 \cdot \gamma_{\text{agg}} \cdot (S[k+1,i] - Z[k+1,i])$
    for $j \leftarrow 1$ to N **do**
      $V[k+1,i] \leftarrow A[i,j] \cdot V[k,j] + \nabla_2 \ell(Z[k+1,i], S[k+1,i]) - \nabla_2 \ell(Z[k,i], S[k,i])$
    end for
  end for
end for
Storing the values of the cost and its gradient for plotting

4.3 Simulation and Results

In this section we show and comment the results achieved by running the aggregative gradient tracking algorithm on the problem. We also show the evolution of the cost function, of the norm of the gradient and the consensus plot across the iterations. A further request was to generate an animation of the robots moving in the environment, this task is handled by the `animate_robots` method.

The parameters used for the simulations are defined in the following table:

Parameter	**Value**
`NN`	Number of agents
`iterations`	Number of iterations
`alpha`	Stepsize
`gamma_target`	List of target weights
`gamma_aggregative`	List of focus weights
`G`	Graph object
`radius`	Target spawn radius

In order to better visualize how the robots move in the environment and how they balance the tradeoff between target-tracking and tight formation, we decided to make the targets spawn on the circumference of a circle of radius $radius = 3$, while the robots are randomly initialized inside this circle.

4.3.1 Scenario: Tradeoff between target-tracking and tight formation

Parameter	Value
`NN`	6
`iterations`	2000
`alpha`	0.001
`gamma_target`	3
`gamma_aggregative`	3
`G`	Cycle
`radius`	3

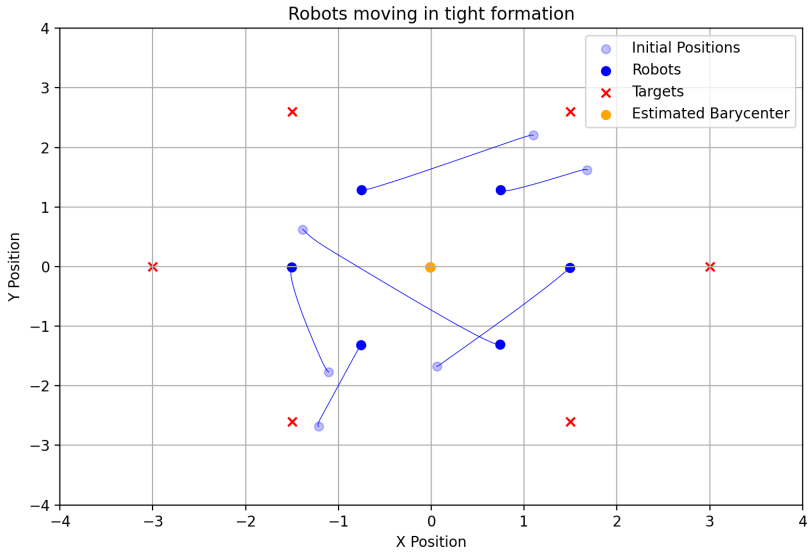
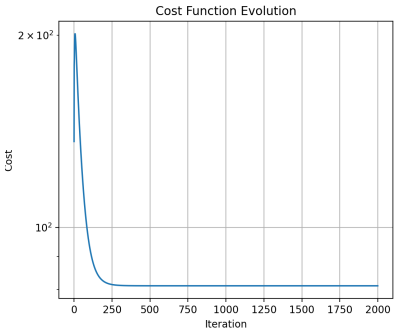
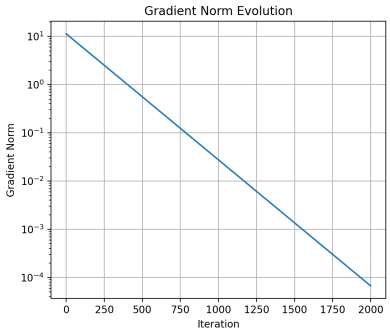


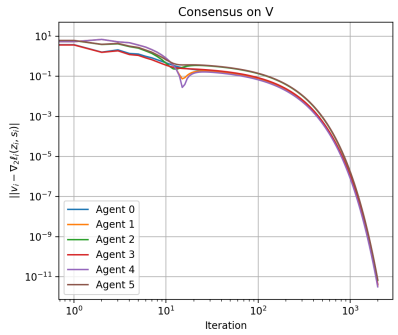
Figure 4.1: Balanced target-tracking and tight formation



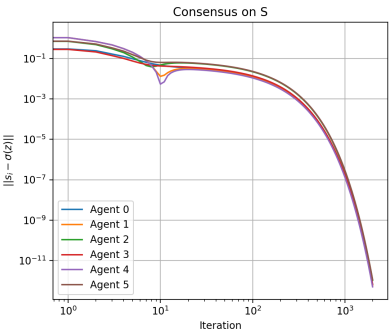
(a) Evolution of the cost function across the iterations



(b) Evolution of the norm of the gradient across the iterations



(c) Norm of the difference between the gradient estimate, for each agent, and the true gradient across the iterations



(d) Norm of the difference between the barycenter estimate, for each agent, and the true barycenter across the iterations

Figure 4.2: Results for the tradeoff between target-tracking and tight formation

4.3.2 Scenario: Focus on target-tracking

Parameter	Value
NN	6
iterations	2000
alpha	0.001
gamma_target	15
gamma_aggregative	3
G	Cycle
radius	3

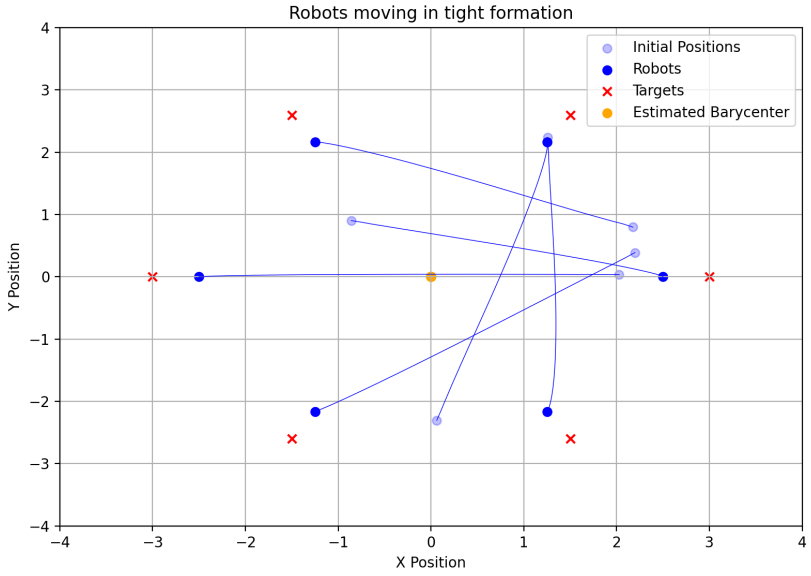
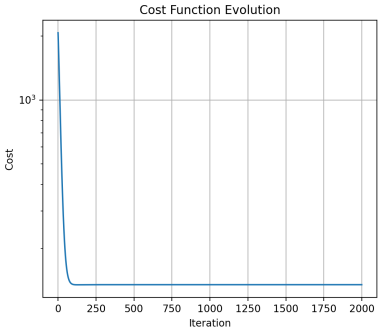
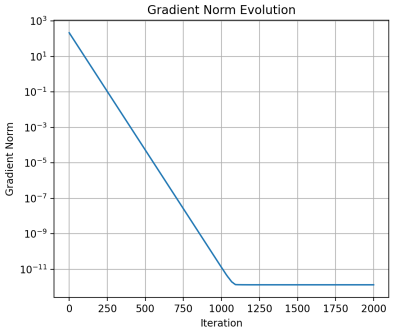


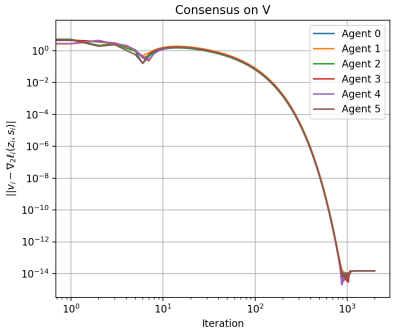
Figure 4.3: Focus on target-tracking



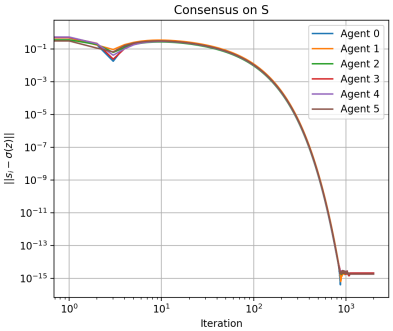
(a) Evolution of the cost function across the iterations



(b) Evolution of the norm of the gradient across the iterations



(c) Norm of the difference between the barycenter estimate, for each agent, and the true barycenter across the iterations



(d) Norm of the difference between the barycenter estimate, for each agent, and the true barycenter across the iterations

Figure 4.4: Results for the focus on target-tracking

4.3.3 Scenario: Focus on tight formation

Parameter	Value
`NN`	6
`iterations`	2000
`alpha`	0.001
`gamma_target`	3
`gamma_aggregative`	15
`G`	Cycle
`radius`	3

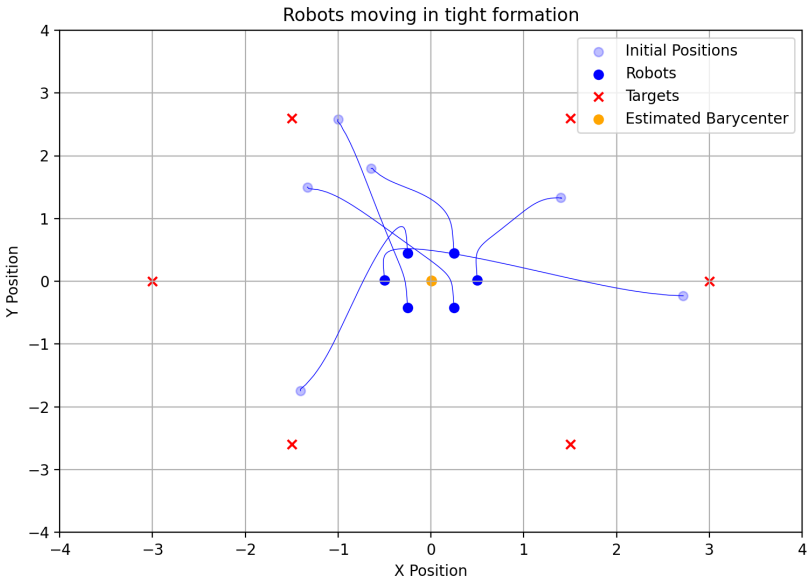
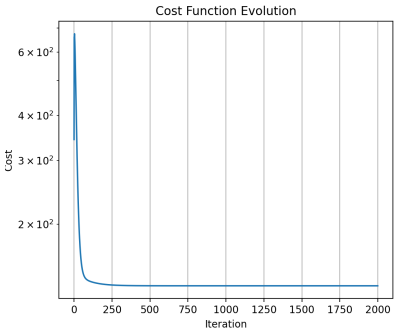
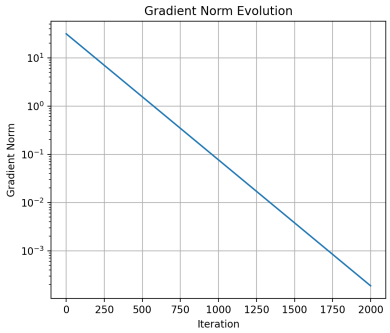


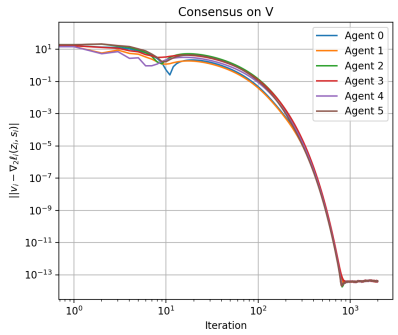
Figure 4.5: Focus on tight formation



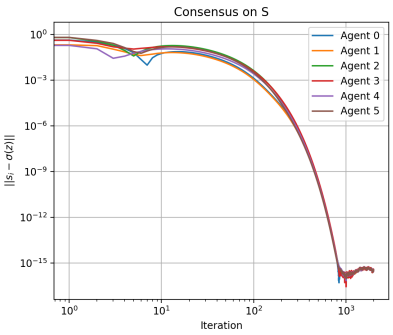
(a) Evolution of the cost function across the iterations



(b) Evolution of the norm of the gradient across the iterations



(c) Norm of the difference between the gradient estimate, for each agent, and the true gradient across the iterations



(d) Norm of the difference between the barycenter estimate, for each agent, and the true barycenter across the iterations

Figure 4.6: Results for the focus on tight formation

As shown in the plots above, the algorithm works correctly. In the first scenario, we chose an equal value for the gamma weights, showing that the robots are able to balance the tradeoff between target-tracking and tight formation. In the second scenario, we focused on target-tracking, choosing a γ_{tar} five times bigger than γ_{agg}, and the robots are way close to the targets. If we further increment the value of γ_{tar}, the robots will focus only on reaching the targets. In the third scenario, we focused on tight formation, choosing a γ_{agg} five times bigger than γ_{tar}, and the robots remain close to the expected barycenter of the team. The cost function in the first 100 iterations falls rapidly and stabilizes. The norm of the gradient decrease very steeply across the iterations. The consensus graphs, plotted in `loglog` scale for better visualization, show how the robots are able to reach consensus regarding the barycenter and the ∇_2 of the cost function.

Chapter 5

Aggregative Optimization for Multi-Robot Systems: Task 2.3 - Moving inside a corridor

This subtask is basically a direct upgrade of task 2.1. While maintaining the same original requests of tight formation and target-tracking, we must implement a new cost function that takes into account the presence of a costraint: a corridor through which the robots must move in order to reach the targets. Since everything else remains the same, we can avoid discussing the algorithm implementation and the problem setup, and focus solely on the new cost function.

5.1 Cost Function

In order to model the corridor constraints, we have to define two eight-order polynomials that represents the corridor boundaries:

$$y_{\text{up}}(x) = (ax + b)^8 + c \tag{5.1}$$

$$y_{\text{low}}(x) = -(ax + b)^8 - c \tag{5.2}$$

We can introduce the concept of barrier functions: a continous function whose value increases to infinity as soon as its agent approaches the boundary of the feasible region, while it has close to zero impact as long as the agent stays far away from the boundary. The term that will be added to our cost function is the sum of two logarithmic barrier functions, one for each boundary:

$$\text{cost\_barrier} = -\epsilon \cdot (\log(-g_{\text{up}}(z_i) - \text{t\_obs}) + \log(-g_{\text{low}}(z_i) - \text{t\_obs})) \tag{5.3}$$

Where $g_{\text{up}}(z_i) \leq 0$ and $g_{\text{low}}(z_i) \leq 0$ are the safe regions and t_obs is a quantity that represents the desired distance from the polynomial functions. The parameter ϵ is used to balance the cost function terms.

$$g_{\text{up}}(z_i) = y_i - y_{\text{up}}(x) \tag{5.4}$$
$$g_{\text{low}}(z_i) = -y_i + y_{\text{low}}(x) \tag{5.5}$$

We have finally designed our new cost function as:

$$\begin{aligned}\ell_i(z_i, s_i) = \gamma_{\text{tar}} \cdot \|z_i - r_i\|^2 + \gamma_{\text{agg}} \cdot \|z_i - s_i\|^2 + \\ - \epsilon \cdot (\log(-g_{\text{up}}(z_i) - \text{t\_obs}) + \log(-g_{\text{low}}(z_i) - \text{t\_obs}))\end{aligned} \tag{5.6}$$

We now can compute the gradients of the cost function:

$$\begin{aligned}\nabla_1 \ell_i(z_i, s_i) = 2 \cdot \gamma_{\text{tar}} \cdot (z_i - r_i) + 2 \cdot \gamma_{\text{agg}} \cdot (z_i - s_i) + \\ - \epsilon \cdot \left(\frac{\nabla g_{\text{up}}(z_i)}{-g_{\text{up}}(z_i) - \text{t\_obs}} + \frac{\nabla g_{\text{low}}(z_i)}{-g_{\text{low}}(z_i) - \text{t\_obs}} \right)\end{aligned} \tag{5.7}$$
$$\nabla_2 \ell_i(z_i, s_i) = 2 \cdot \gamma_{\text{agg}} \cdot (s_i - z_i) \tag{5.8}$$

With $\nabla_1 \ell_i(z_i, s_i) : \mathbb{R}^d \to \mathbb{R}^d$ and $\nabla_2 \ell_i(z_i, s_i) : \mathbb{R}^d \to \mathbb{R}^d$ and the gradients of the barrier functions being:

$$\nabla g_{\text{up}}(z_i) = \begin{bmatrix} -8 \cdot (a x_i + b)^7 \cdot a \\ 1 \end{bmatrix} \tag{5.9}$$
$$\nabla g_{\text{low}}(z_i) = \begin{bmatrix} -8 \cdot (a x_i + b)^7 \cdot a \\ -1 \end{bmatrix} \tag{5.10}$$

With the cost function and its gradient now defined, the only issue to be addressed are the spawn points of the robots and of the targets. Once we have found the polynomials coefficients a, b and c, defining the shape of our corridor, we can properly where the robots and the targets will spawn. The `generate_robots` method handles in which range the robots will spawn, while the `generate_targets` method handles the range in which the targets spawn. They are implemented in such a way that the robots will surely spawn to the left of the corridor, while the targets will spawn to its right. We won't further discuss the algorithm implementation, since it remains the same as in task 2.1.

5.2 Simulation and Results

In this section we show and comment the results achieved by running the aggregative gradient tracking algorithm on the problem. We also show the evolution of the cost function, of the norm of the gradient and the consensus plot across the iterations. Again, we had to generate an animation of the

robots moving in the environment, through the corridor, this task is handled by the `animate_robots` method. The parameters used for the simulations are defined in the following table:

Parameter	**Description**	**Value**
`NN`	Number of robots	5
`iterations`	Number of iterations	2000
`alpha`	Stepsize	0.001
`gamma_target`	List of target weights	5
`gamma_aggregative`	List of aggregative weights	5
`G`	Graph object	Cycle
`t_obs`	Desired distance from polynomial function	0.3
`epsilon`	Barrier term weight	30
`a`	Polynomial parameter	0.3
`b`	Polynomial parameter	-3
`c`	Polynomial parameter	0.5

By tuning the cost function parameters γ_{tar}, γ_{agg} and ϵ, we can force the robots to move through the corridor, while keeping a tight formation and tracking the targets. We are able to put more emphasis on a constraint by increasing that specific parameter.

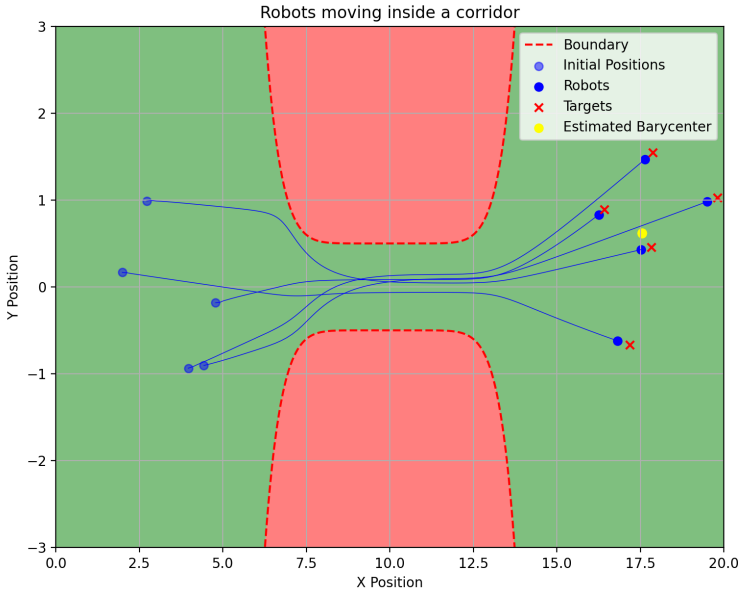
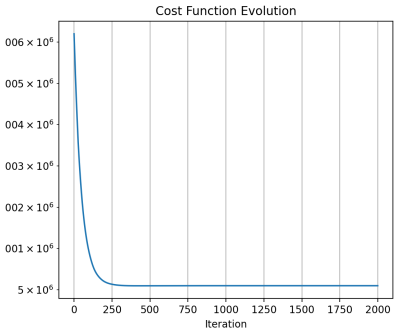
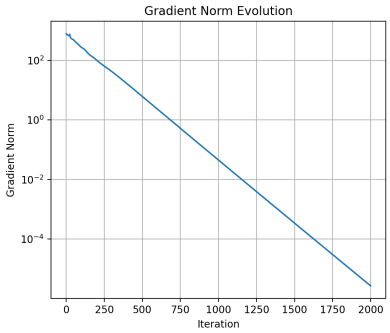


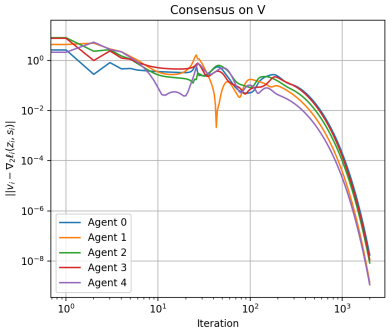
Figure 5.1: Team of robots moving through the corridor



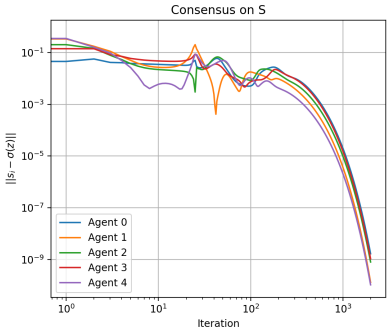
(a) Evolution of the cost function across the iterations



(b) Evolution of the norm of the gradient across the iterations



(c) Norm of the difference between the gradient estimate, for each agent, and the true gradient across the iterations



(d) Norm of the difference between the barycenter estimate, for each agent, and the true barycenter across the iterations

Figure 5.2: Results for the team of robots moving through the corridor

The results obtained are satisfactory. The animation shows that the robots are able to move through the corridor, avoiding collision, while keeping a tight formation and tracking the targets. The norm of the gradient decreases very steeply. The cost function has a pathological behaviour, since it doesn't decrease at all and mantains the same cost, this is due to how the barrier term is implemented inside the cost function. A way to improve this condition would've been to implement a projected gradient tracking algorithm, a more sophisticate solution that includes a projection step in which each node update its estimate by taking a step in the direction of the negative aggregate gradient (each node would project this new estimate inside the feasibile set). The consensus graphs, plotted in `loglog` scale for better visualization, show how the robots are able to reach consensus regarding the barycenter and the ∇_2 of the cost function.

Chapter 6

Aggregative Optimization for Multi-Robot Systems: Task 2.2 & Task 2.4 - ROS2 Implementation

6.1 Problem Setup

The goal of these tasks is to implement the same algorithm developed in Task 2.1 and Task 2.3 in a ROS2 environment. This is required if we want to run the algorithm in a realistic environment, where a real-time communication between the robots is established In order to do this, we have to setup nodes for optimization and visualization, and creating a launch file to run the system.

6.1.1 Optimization Node

The optimization node is responsible for executing the aggregative optimization algorithm. The node initializes basically any parameter needed for running the algorithm, computing the cost function and ensuring that robots and targets spawn in a safe zone.

The core function of the optimization node that implements the algorithm is the `aggregative_tracking_optimization` method, which iteratively updates the positions of the robots based on the cost function and its gradients. The node publishes the updated robot positions, target positions, barycenter estimates, and trajectories to corresponding topics. These data are then used by the visualization node to display the movements and interactions of the robots in real-time.

6.1.2 Visualization Node

The visualization node subscribes to the topics published by the optimization node. It listens for updates on robot positions, target positions, barycenter estimates, and trajectories. Upon receiving new data, it updates the markers in the RViz environment to reflect the current state of the robots and targets.

The node uses different types of markers to visualize the robots, targets, barycenter, and trajectories. Robots are shown as blue spheres, targets are indicated by red cross markers, the barycenter is displayed as a yellow sphere, and trajectories are shown as purple lines. This visualization helps in monitoring the robots' behavior and their path towards the targets while maintaining a tight formation.

6.1.3 Topic

The communication between the optimization and visualization nodes is done through ROS2 topics. The optimization node publishes the robot positions, target positions, barycenter estimates, and trajectories to corresponding topics. The visualization node subscribes to these topics to receive the updated information and update the markers in the RViz environment.

Topic	**Description**
`robot_positions`	Publishes the current positions of all robots
`target_positions`	Publishes the positions of all targets
`barycenter_position`	Publishes the estimated barycenter position of the robot team
`robot_trajectories`	Publishes the trajectories of the robots over time

6.1.4 Launch File

The launch file is a crucial component for running the ROS2 implementation. It defines the nodes to be launched and their configurations. In our setup, the launch file includes the optimization node and multiple instances of the visualization node, one for each robot.

The launch file also includes the configuration for RViz, specifying the RViz configuration file to use. This setup ensures that all necessary nodes are launched simultaneously, and the RViz environment is properly setup to visualize the robots and how they move in the environment.

6.2 Simulation and Results

In order to show the plots with the data obtained through ROS2, we implemented a Python script that fetches `files.txt` and reads the data from the files. These files are created by the optimization node and contain the data of the cost function, the norm of the gradient, the norm of the difference betweenn the barycenter estimate and the true barycenter and the norm of the difference between the gradient estimate and the true gradient. When running the appropriate command from the terminal, the script plots the data and shows the results obtained. Beware that when running this scripts, if there were previous data stored in the files, they will be overwritten. To run this scripts, open a terminal, travel to the workspace directory and run the following commands:

```
python3 src/task_2_2/scripts/plots.py
python3 src/task_2_4/scripts/plots.py
```

6.2.1 RViz Visualization

RViz is an essential tool in ROS2 for visualizing the state of the robots and their environment. In our implementation, RViz provides a graphical interface to observe the positions of the robots, their trajectories, the target locations, and the barycenter estimate in real-time. This visualization aids in monitoring the behavior and performance of the robots, ensuring they maintain formation while moving towards their targets. In order to show the appropriate topics, we built custom configurations for RViz for each task, the launch file will handle the loading of these configurations.

6.2.2 Scenario: Task2.2

The parameters used for the simulations are defined in the following table:

Parameter	Value
`NN`	6
`iterations`	1000
`alpha`	0.001
`gamma_target`	5
`gamma_aggregative`	5
`G`	Cycle
`radius`	5

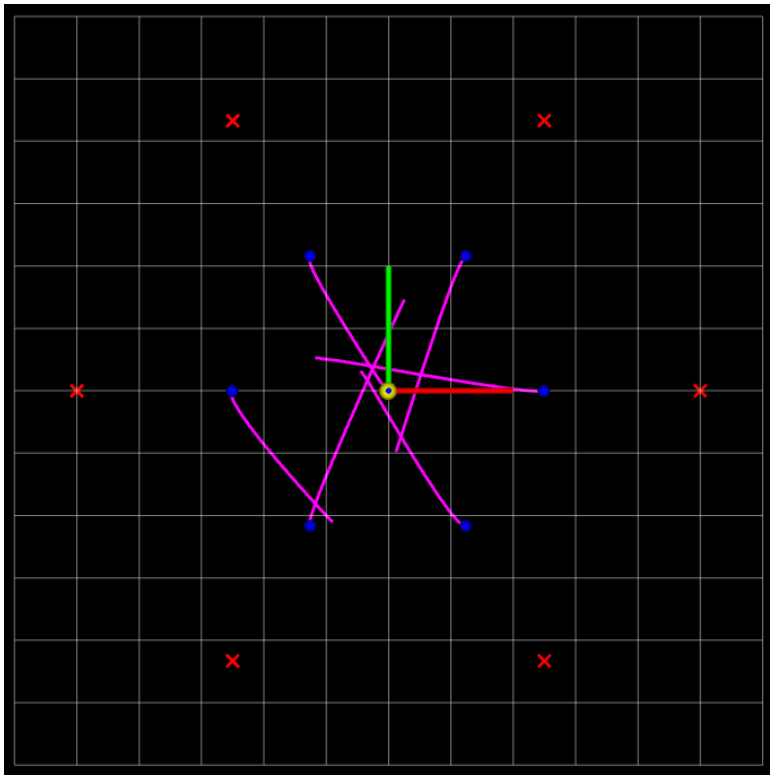
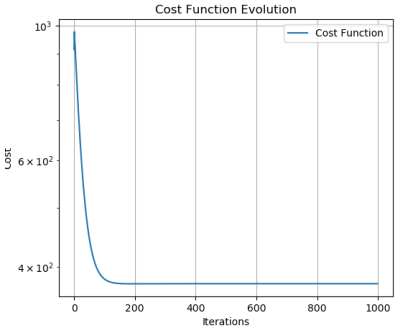
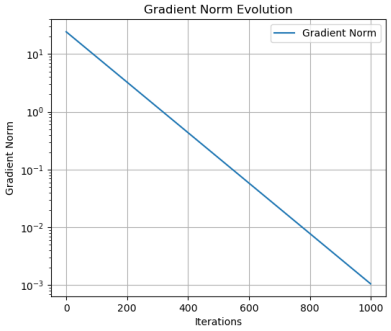


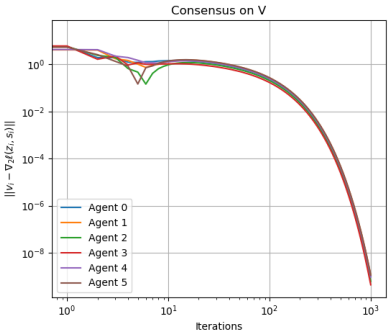
Figure 6.1: RViz visualization of the robots moving. Cost weights are chosen to balance the tradeoff between target-tracking and tight formation.



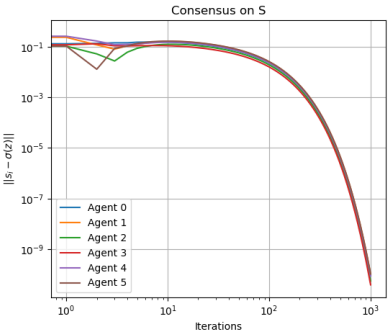
(a) Evolution of the cost function across the iterations



(b) Evolution of the norm of the gradient across the iterations



(c) Norm of the difference between the gradient estimate, for each agent, and the true gradient across the iterations



(d) Norm of the difference between the barycenter estimate, for each agent, and the true barycenter across the iterations

Figure 6.2: Results for the tradeoff between target-tracking and tight formation. Data stored from the ROS2 implementation

We focused again on balancing the target-tracking term and the aggregative term. By doing some testing, we ensure that if we change the gamma variables, we can force the robots to focus more on target-tracking or on tight formation. The plots demonstrate that the algorithm effectively minimizes the cost function, that after about 100 iterations, reaches its optimum, while the gradient norm keeps decreasing steadily over time. The consensus plots indicate that the agents successfully reach a consensus on their states and gradients, ensuring cohesive behavior among the robots.

6.2.3 Scenario: Task2.4

The parameters used for the simulations are defined in the following table:

Parameter	**Value**
`NN`	5
`iterations`	1000
`alpha`	0.001
`gamma_target`	5
`gamma_aggregative`	5
`G`	Cycle
`t_obs`	0.3
`epsilon`	30
`a`	0.3
`b`	-3
`c`	0.5

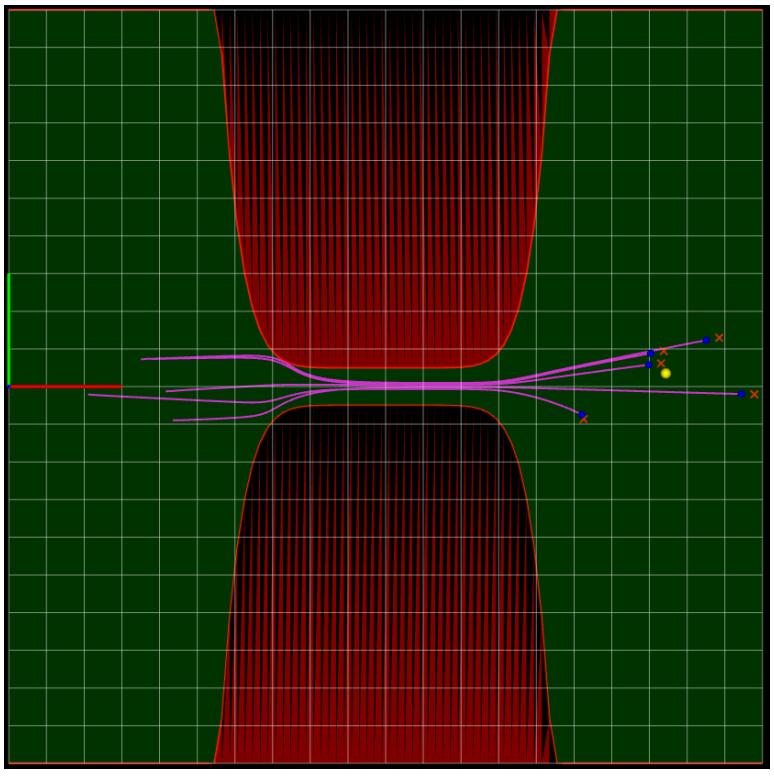
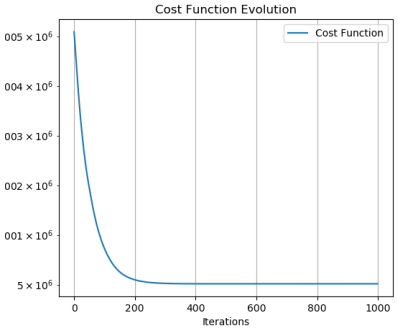
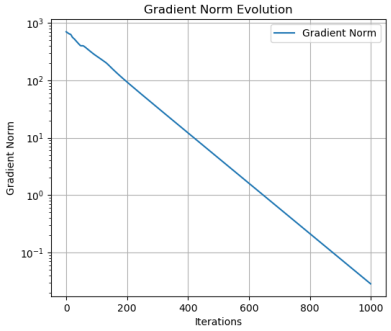


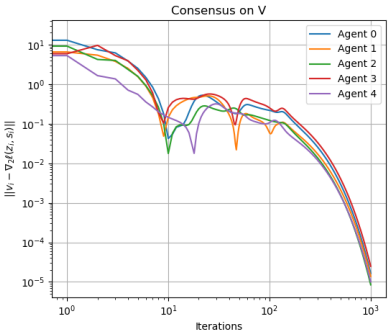
Figure 6.3: RViz visualization of the robots moving through the corridor



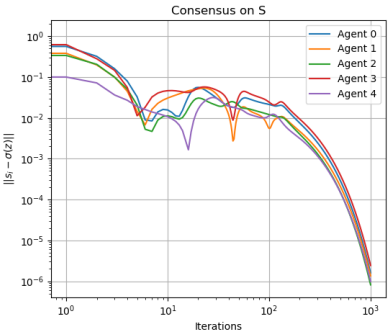
(a) Evolution of the cost function across the iterations



(b) Evolution of the norm of the gradient across the iterations



(c) Norm of the difference between the gradient estimate, for each agent, and the true gradient across the iterations



(d) Norm of the difference between the barycenter estimate, for each agent, and the true barycenter across the iterations

Figure 6.4: Results for the team of robots moving through the corridor. Data stored from the ROS2 implementation

In this scenario, the visualization node gets an upgrade, since in addition to visualizing robots, targets, barycenter estimate and trajectories, the node also publishes boundary markers to define the corridor through which robots must navigate. It generates points for the upper and lower boundaries of the corridor and creates markers for these boundaries, as well as the corridor itself and the red zones outside the corridor. The same comments made for the scenario described in task 2.3 apply here too.

Chapter 7

Conclusions

In conclusion, this report demonstrates the successful implementation of distributed algorithms for classification and optimization in multi-agent systems. We have shown the theory behind the algorithms and their implementation on several different scenarios.

Overall, the results produced by this report are satisfactory. The gradient tracking algorithm solves distributed classification problems with great precision and accuracy, while the aggregative gradient tracking ensures robustness when tracking targets and avoiding obstacles.

The extension of these algorithms to the ROS2 environment showcases their suitability in real-world scenarios, enabling real-time communication and visualization of multi-robot systems.