UNIVERSITÀ DI BOLOGNA

School of Engineering

Master Degree in Automation Engineering

Distributed Autonomous Systems

DISTRIBUTED CLASSIFIER & DISTRIBUTED AGGREGATIVE OPTIMIZATION

Professors:
Giuseppe Notarstefano
Ivano Notarnicola

Students:
Nicola Francesconi
Gian Paolo Savorani
Riccardo Mengozzi

Academic year 2023/2024

Introduction

This project aims to design distributed algorithms in order to accomplish different tasks. In particular two tasks have been taken in consideration: Non-Linear Distributed Classification & Distributed Aggregative Optimization. For the first task is required firstly to develop a distributed gradient tracking algorithm then test the efficiency and reliability using different type of graph, this algorithm is crucial in the third part of the first task in order to classify in a distributed way a given dataset. Then a centralized non-linear classification algorithm has been developed. Finally the first two parts have been merged in order to develop the distributed non-linear classifier. In the second task the goal is to implement a distributed control algorithm that allows the robots to keep the formation tight, and, at the same time, be close to some private targets. First of all a python script that implement the Aggregative Tracking algorithm has been developed. Then, once the first part has been accomplished, the ROS2 implementation of the algorithm has been developed in order to simulate in a better way the neighborhood communication. Finally a potential obstacle avoidance term has been added to the local cost function definition in order to emulate a corridor through which the agents have to pass in order to accomplish the task.

Contents

Chapter 1

Distributed Non-Linear Classification

1.1 Introduction

Suppose to have N agents that want to cooperatively determine a nonlinear classifier Fig:1.1 for a set of points in a given feature space. Each agent knows only a subset of the dataset and can communicate with the other agents according to a graph $G = (\{1, \ldots, N\}, E)$

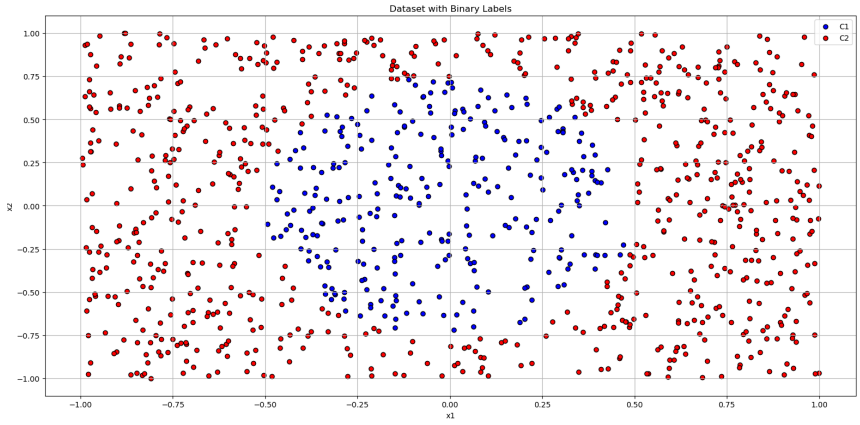


Figure 1.1: Non Linear Classification Task

As preliminary step a gradient tracking algorithm has been implemented and tested on quadratic optimization problem.

1.2 Gradient Tracking

Consider N agents that want to cooperate to solve the following minimization problem:

$$\min_z \sum_{i=1}^{N} \ell_i(z)$$

Each agent i knows only the private cost $\ell_i(z)$ where $\ell_i(z) : \mathbb{R}^d \to \mathbb{R}$ and want to reach consensus with the other agents on the value of z that minimize the sum of all the private cost functions.
For each agent we then defined it's private cost:

$$\ell_i(z_i) = z_i^T Q_i z_i + r_i^T z_i$$

Where $z_i \in \mathbb{R}^d$ denotes the state of each agent i, $Q_i = Q_i^T \in \mathbb{R}^{d\times d}$ is a positive definite matrix and $r_i \in \mathbb{R}^d$ is a linear term $\forall i \in \{1, \dots, N\}$.
Each Agent can communicate only with it's neighbours according to the graph G. Suppose G is connected and a global minimum of the problem exist. then the algorithm reads as:

$$z_i^{k+1} = \sum_{j\in\mathcal{N}_i} a_{ij} z_j^k - \alpha s_i^k \qquad z_i^0 = z_{init} \tag{1.1}$$

$$s_i^{k+1} = \sum_{j\in\mathcal{N}_i} a_{ij} s_j^k + \nabla\ell_i(z_i^{k+1}) - \nabla\ell_i(z_i^k) \qquad s_i^0 = \nabla\ell_i(z_i^0) \tag{1.2}$$

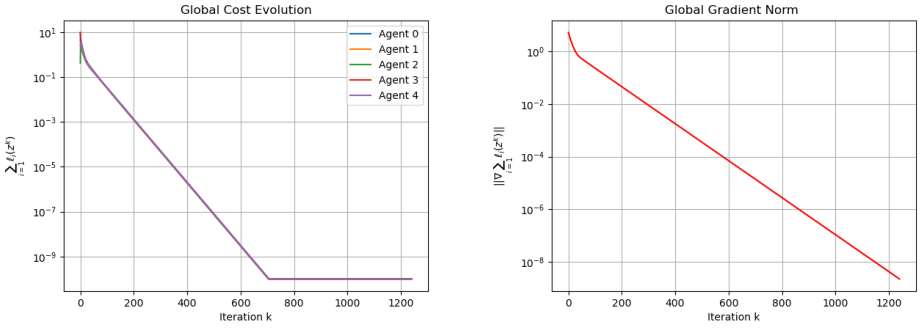
Where $a_{ij} > 0 \in \mathbb{R}$ is related to the weighted adjacency matrix of the graph G, $\alpha > 0 \in \mathbb{R}$ is the step-size and $s_i^k \in \mathbb{R}^d$ is the local estimate of the global gradient: $\nabla \sum_{i=1}^{N} \ell_i(z)$.

1.2.1 Results

The following plots show the performance of the algorithm with different number of agents and different graph: In each simulation shown how the global cost, the gradient norm and the error with respect to the optimal solution evolve.

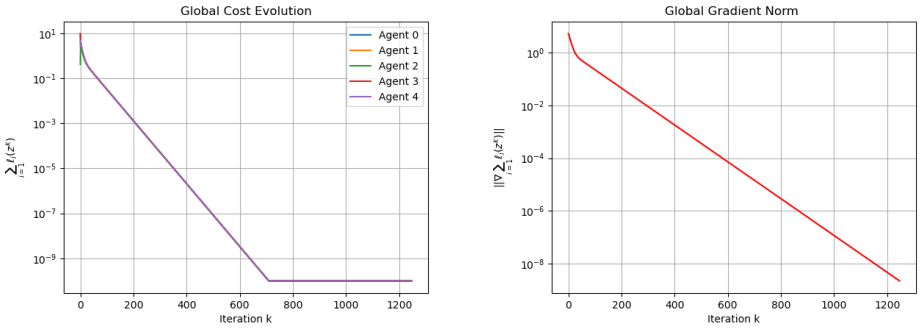
Agent	Q	r	z_init
0	$\begin{bmatrix}0.363 & 0.255 & 0.255 & 0.597\end{bmatrix}$	$\begin{bmatrix}0.438 & 0.892\end{bmatrix}$	$\begin{bmatrix}1.764 & 0.400\end{bmatrix}$
1	$\begin{bmatrix}0.000 & 0.000 & 0.000 & 0.030\end{bmatrix}$	$\begin{bmatrix}0.186 & 0.346\end{bmatrix}$	$\begin{bmatrix}1.624 & -0.612\end{bmatrix}$
2	$\begin{bmatrix}0.302 & 0.231 & 0.231 & 0.286\end{bmatrix}$	$\begin{bmatrix}0.205 & 0.619\end{bmatrix}$	$\begin{bmatrix}-0.417 & -0.056\end{bmatrix}$
3	$\begin{bmatrix}0.085 & 0.260 & 0.260 & 1.601\end{bmatrix}$	$\begin{bmatrix}0.126 & 0.207\end{bmatrix}$	$\begin{bmatrix}1.789 & 0.437\end{bmatrix}$
4	$\begin{bmatrix}0.946 & 0.679 & 0.679 & 0.534\end{bmatrix}$	$\begin{bmatrix}0.976 & 0.006\end{bmatrix}$	$\begin{bmatrix}0.051 & 0.500\end{bmatrix}$

Table 1.1: Values of Q, r, and z_init for each agent, $N = 5$



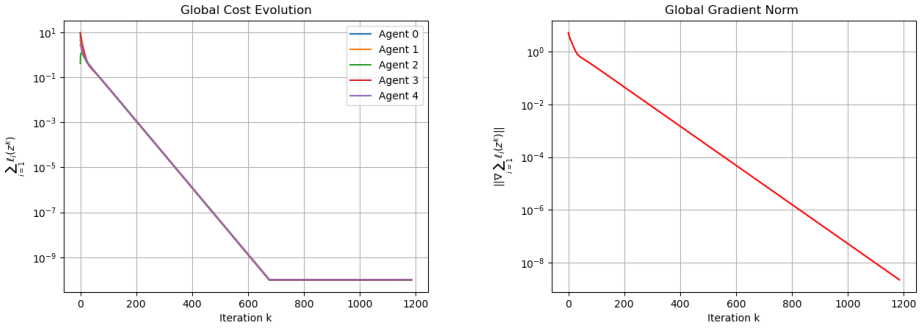
(a) Global Cost (b) Global Gradient Norm

Figure 1.2: G="Path Graph", $N = 5$



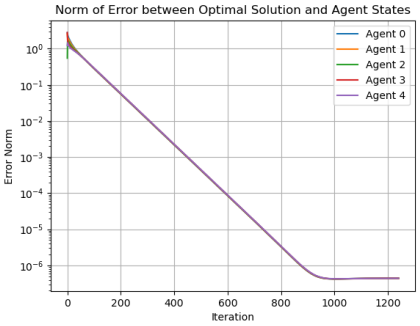
(a) Global Cost (b) Global Gradient Norm

Figure 1.3: G="Cycle Graph", $N = 5$

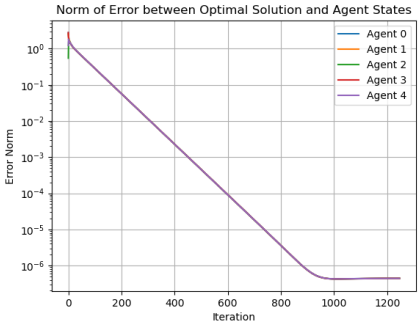


(a) Global Cost (b) Global Gradient Norm

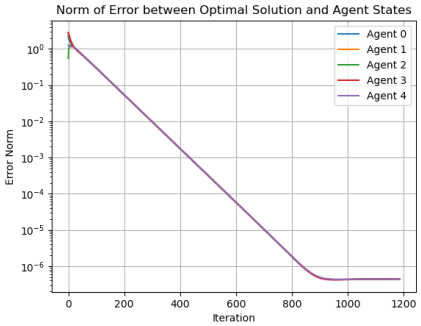
Figure 1.4: G="Star Graph", $N = 5$



(a) Path Graph



(b) Cycle Graph



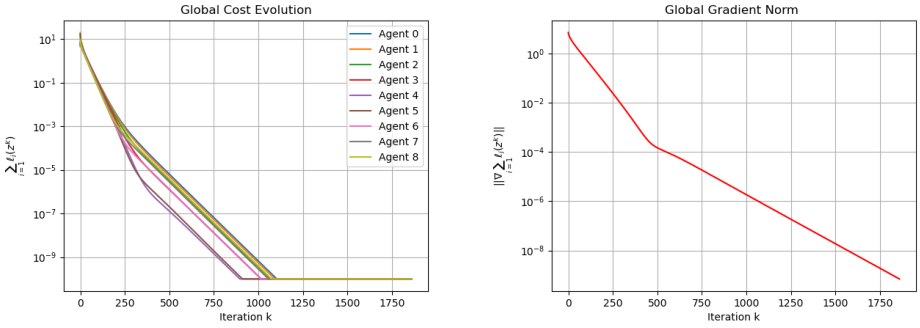
(c) Star Graph

Figure 1.5: fig:N=5 Consensus for different Graphs

The same random seed is used for the initialization of all the different test with same number of agents in order to compare the solutions evenly. It's possible to see from the previous figures how,for a small number of agents the algorithm reach consensus almost in the same way.

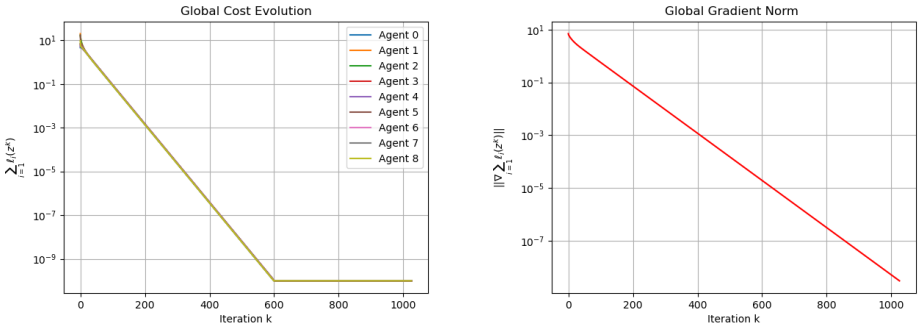
Agent	Q	r	z_init
0	$\begin{bmatrix}0.246 & 0.070 & 0.070 & 0.068\end{bmatrix}$	$\begin{bmatrix}0.419 & 0.248\end{bmatrix}$	$\begin{bmatrix}0.001 & -0.290\end{bmatrix}$
1	$\begin{bmatrix}0.249 & 0.099 & 0.099 & 0.618\end{bmatrix}$	$\begin{bmatrix}0.169 & 0.088\end{bmatrix}$	$\begin{bmatrix}1.332 & 0.715\end{bmatrix}$
2	$\begin{bmatrix}0.177 & 0.005 & 0.005 & 0.238\end{bmatrix}$	$\begin{bmatrix}0.942 & 0.851\end{bmatrix}$	$\begin{bmatrix}1.749 & -0.286\end{bmatrix}$
3	$\begin{bmatrix}0.069 & 0.004 & 0.004 & 0.844\end{bmatrix}$	$\begin{bmatrix}0.901 & 0.033\end{bmatrix}$	$\begin{bmatrix}0.473 & -0.681\end{bmatrix}$
4	$\begin{bmatrix}0.679 & 0.802 & 0.802 & 1.152\end{bmatrix}$	$\begin{bmatrix}0.609 & 0.776\end{bmatrix}$	$\begin{bmatrix}-0.712 & 0.754\end{bmatrix}$
5	$\begin{bmatrix}0.758 & 0.270 & 0.270 & 1.013\end{bmatrix}$	$\begin{bmatrix}0.513 & 0.318\end{bmatrix}$	$\begin{bmatrix}1.551 & 0.079\end{bmatrix}$
6	$\begin{bmatrix}0.003 & 0.015 & 0.015 & 0.357\end{bmatrix}$	$\begin{bmatrix}0.306 & 0.304\end{bmatrix}$	$\begin{bmatrix}-0.312 & 0.339\end{bmatrix}$
7	$\begin{bmatrix}0.303 & 0.199 & 0.199 & 0.180\end{bmatrix}$	$\begin{bmatrix}0.689 & 0.164\end{bmatrix}$	$\begin{bmatrix}0.128 & -1.528\end{bmatrix}$
8	$\begin{bmatrix}0.037 & 0.151 & 0.151 & 1.050\end{bmatrix}$	$\begin{bmatrix}0.638 & 0.576\end{bmatrix}$	$\begin{bmatrix}0.276 & -1.855\end{bmatrix}$

Table 1.2: Values of Q, r, and z_init for each agent N=9



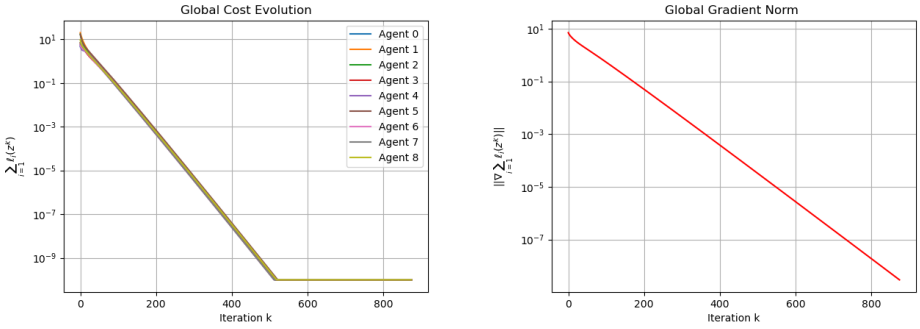
(a) Global Cost (b) Global Gradient Norm

Figure 1.6: G="Path Graph", $N = 9$



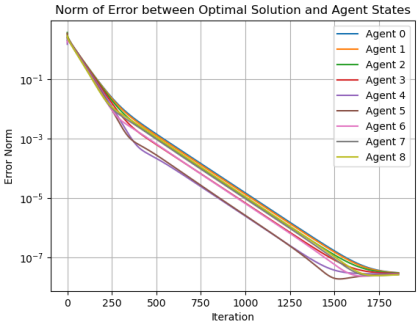
(a) Global Cost (b) Global Gradient Norm

Figure 1.7: G="Cycle Graph", $N = 9$

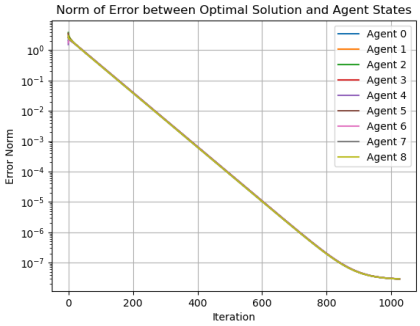


(a) Global Cost (b) Global Gradient Norm

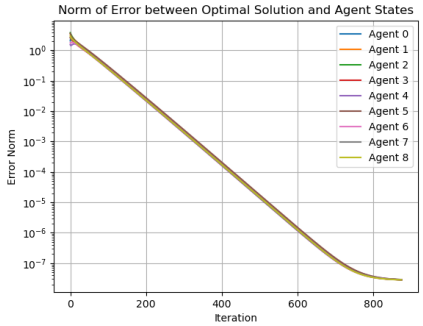
Figure 1.8: G="Star Graph", $N = 9$



(a) Path Graph



(b) Cycle Graph



(c) Star Graph

Figure 1.9: fig:N=9 Consensus for different Graphs

The algorithm for different types of graph and different number of agents always converged to the optimal solution of the problem, however it's possible to see how for higher number of agent the path graph has a slower convergence.

1.3 Centralized Non-Linear Classification

This task require to implement a non-linear classification algorithm. Suppose to have a dataset of $M \in \mathbb{N}$ points $D_m \in \mathbb{R}^d$, $m = 1, \ldots, M$. Each point is labeled with a binary label $p_m = \{-1, 1\}$ according to some ideal separating function:

$$w^T \phi(Dm) + b \geq 0 \rightarrow p_m = +1$$
$$w^T \phi(Dm) + b < 0 \rightarrow p_m = -1$$

where $w \in \mathbb{R}^q$ is the weights vector, $\phi : \mathbb{R}^d \rightarrow \mathbb{R}^q$ is the feature map and $b \in \mathbb{R}$ is the bias term. It's possible to augment both the weights vector and the map by one element embedding the bias:

$$w = [w_1, w_2, \ldots, w_q]^T \rightarrow w = [w_1, w_2, \ldots, w_q, w_{q+1}]^T$$

$$\phi = [\phi_1, \phi_2, \ldots, \phi_q]^T \rightarrow \phi = [\phi_1, \phi_2, \ldots, \phi_q, 1]^T$$

By doing that the last weight of the vector w is the bias and the separating function is given by:

$$w^T \phi(Dm) \geq 0 \rightarrow p_m = +1$$
$$w^T \phi(Dm) < 0 \rightarrow p_m = -1$$

And the task consist in solve the minimization problem:

$$\min_w \ell(w, D, p) = \sum_{m=1}^{M} \log(1 + \exp\left(-p_m w^T \phi(D_m)\right) \tag{1.3}$$

In order to solve this problem in a centralized way we developed a gradient method:

$$w^{k+1} = w^k - \alpha d^k \qquad , w^0 = w_{init}$$

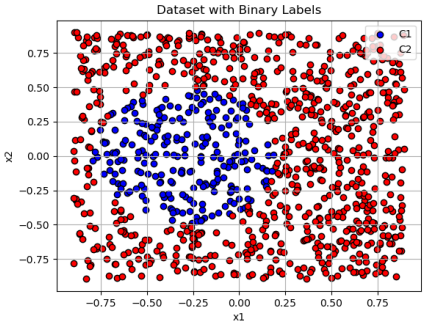
Where $d^k \in \mathbb{R}^{q+1}$ is the direction given by the gradient of the cost function with respect to w and $\alpha > 0 \in \mathbb{R}$ is the step-size. d^k is defined as:

$$d^k = \nabla_1 \ell(w, D, p) = \sum_{m=1}^{M} \frac{-p_m \phi(D_m) \exp(-p_m w^\top \phi(D_m))}{1 + \exp(-p_m w^\top \phi(D_m))} \tag{1.4}$$

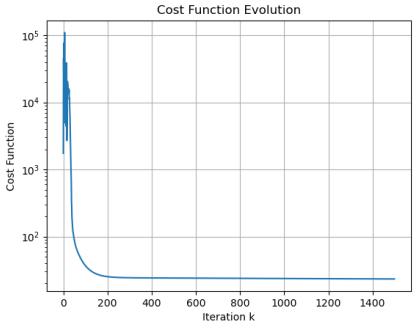
1.3.1 Results

The examples taken in consideration classify two dimensional data $Dm \in \mathbb{R}^2$, allowing us to visually examine patterns in the data and propose appropriate nonlinear decision boundary, however the code can be used also for higher dimensional data.

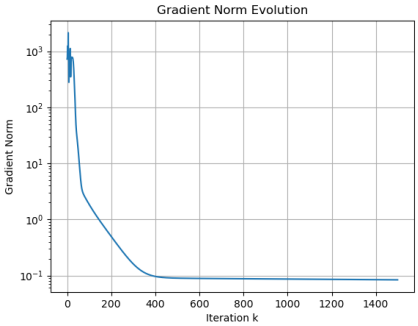
Each classifier has been trained with a set of $M_{train} \in \mathbb{N}$ points, then a new set of $M_{test} \in \mathbb{N}$ belonging to the same distribution has been generated and the has been used in order to test the performances of the classifier. The classifier has been trained both a deterministic distribution and a noisy distribution then is always tested on NON-noisy dataset.



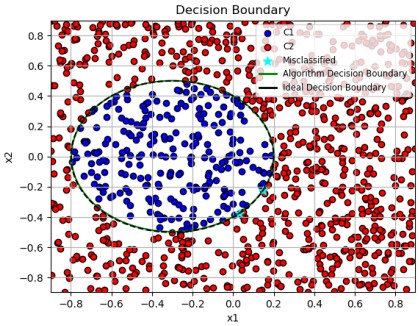
(a) Train Dataset



(b) Cost Evolution



(c) Gradient Norm Evolution

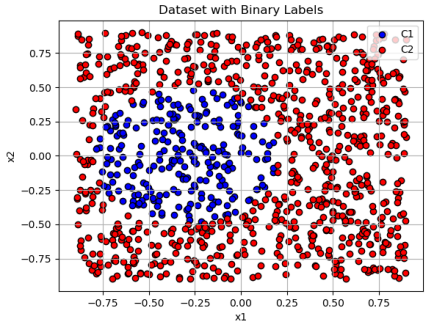


(d) Decision Boundary in Test Dataset

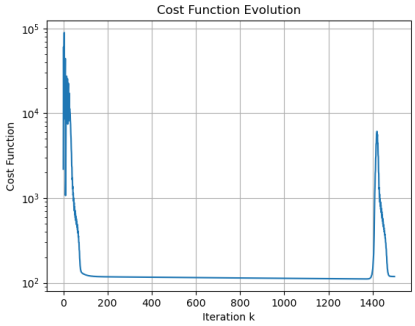
Figure 1.10: Elliptic Classifier without noise

Dataset	N_Data	Misclassified Points	Percentage
Training	3000	4	0.13%
Test	1000	2	0.20%

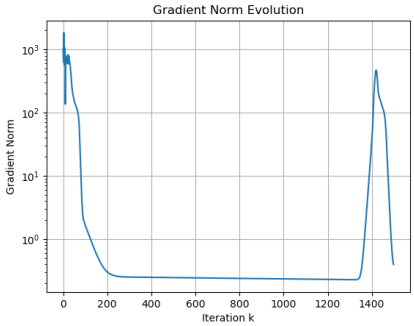
In figure:1.10 it's possible to see how the Algorithm classifier classify properly almost all the points in the test dataset. It's reasonable have the cost which doesn't reach an "absolute" zero due to the definition of the cost function (Eq:1.3) for which $w^T\phi(D_m)$ must goes to $+\text{inf}$ for each point.



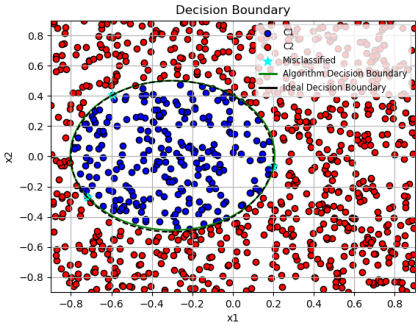
(a) Train Dataset



(b) Cost Evolution



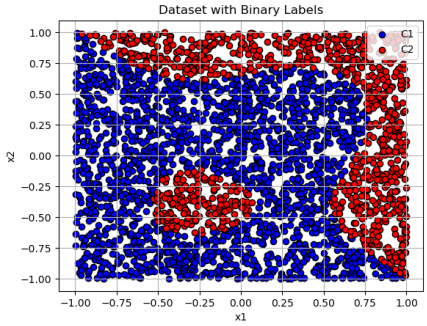
(c) Gradient Norm Evolution



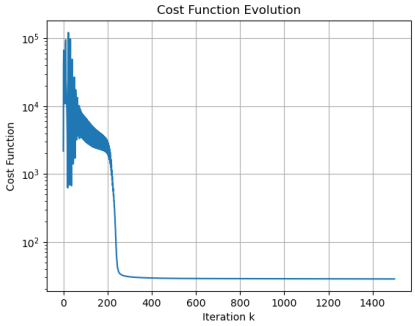
(d) Decision Boundary in Test Dataset

Figure 1.11: Elliptic Classifier with noise

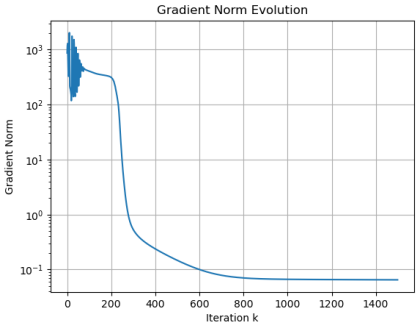
Dataset	N_Data	Misclassified Points	Percentage
Training	3000	49	1.63%
Test	1000	3	0.30%



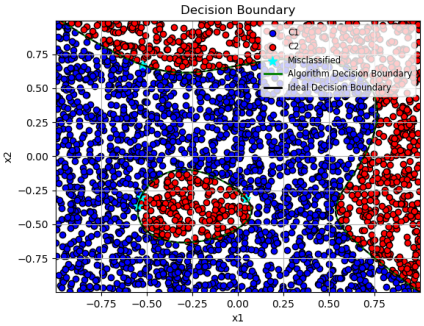
(a) Train Dataset



(b) Cost Evolution



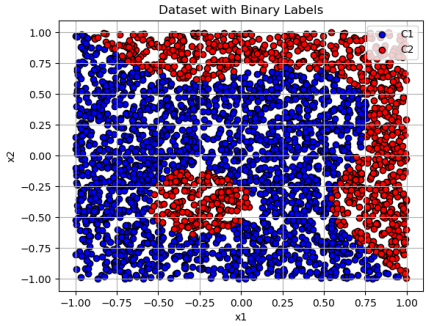
(c) Gradient Norm Evolution



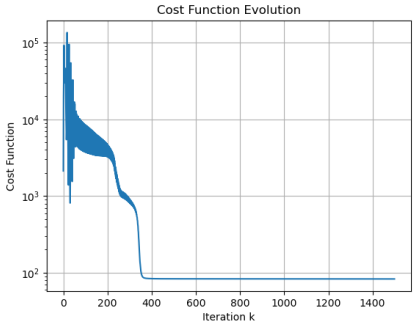
(d) Decision Boundary in Test Dataset

Figure 1.12: Polynomial Classifier without noise

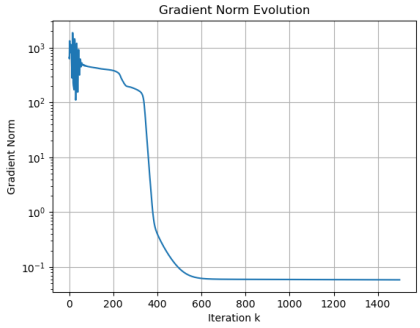
Dataset	N_Data	Misclassified Points	Percentage
Training	3000	2	0.07%
Test	1000	4	0.40%



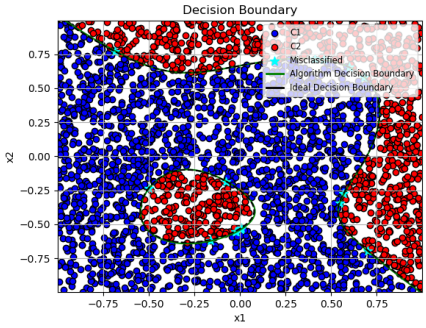
(a) Train Dataset



(b) Cost Evolution



(c) Gradient Norm Evolution



(d) Decision Boundary in Test Dataset

Figure 1.13: Polynomial Classifier with noise

Dataset	N_Data	Misclassified Points	Percentage
Training	3000	21	0.70%
Test	1000	16	1.60%

1.4 Distributed Non-Linear Classification

In this task the problem slightly different from the previous, each agent N knows only a subset of data and can communicate with other agents according to a graph G. By splitting the original dataset D into N subset $D_1, \ldots, D_N$ and assigning to each agent a different subset, the problem, as in the gradient tracking task, reads as:

$$\min_{w} \sum_{i=1}^{N} \ell_i(w, D_i, p_i)$$

Where:

$$\ell_i(w, D_i, p_i) = \sum_{m=1}^{M_i} \log(1 + \exp\left(-p_{im} w^T \phi(D_{im})\right) \qquad \forall i = 1, \ldots, N \qquad (1.5)$$

Then each agent update the local estimate w_i^k of w^* communicating with it's neighbours according to the graph G using as direction s_i^k which is a local estimate of the gradient of the global cost. The algorithm reads as:

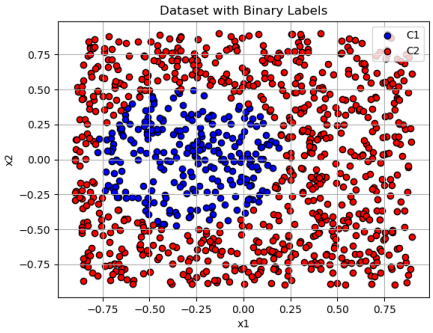
$$w_i^{k+1} = \sum_{j \in \mathcal{N}_i} a_{ij} w_j^k - \alpha s_i^k \qquad w_i^0 = w_{i_{init}} \tag{1.6}$$

$$s_i^{k+1} = \sum_{j \in \mathcal{N}_i} a_{ij} s_j^k + \nabla_1 \ell_i(w_i^{k+1}) - \nabla_1 \ell_i(w_i^k) \qquad s_i^0 = \nabla \ell_i(w_i^0) \tag{1.7}$$

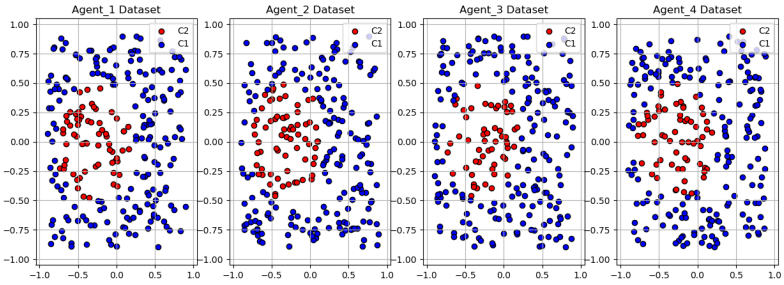
We tested the algorithm with different types of non linear classifier and also with different method in the dataset splitting.

1.4.1 Results

The following plots shows how the algorithm react to different shape of classification and different data splitting using path graph with self loops and $N = 4$ agents.

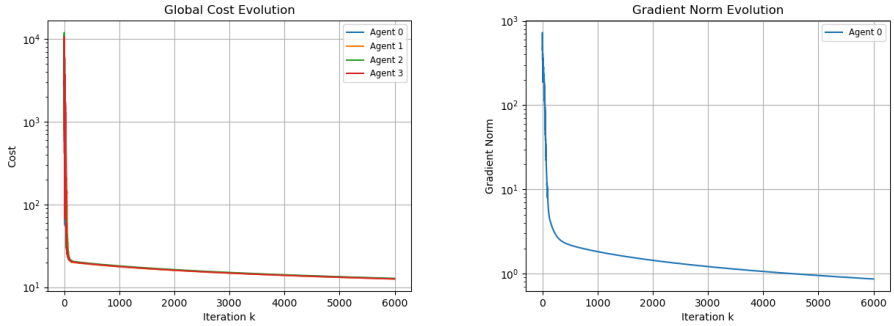


(a) Train Dataset



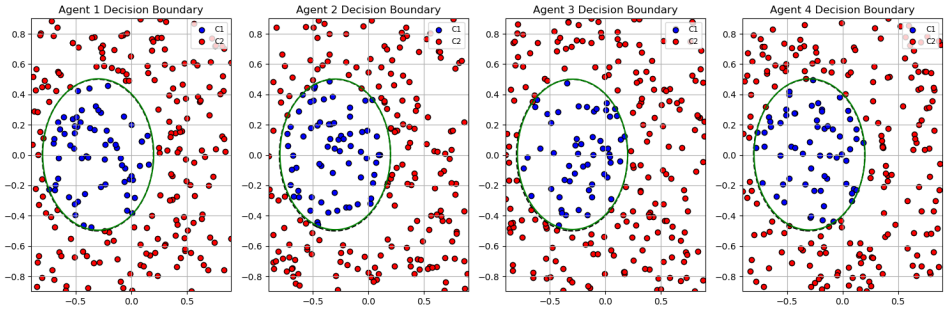
(b) Split Dataset

Figure 1.14: Random Split Elliptic Dataset



(a) Cost Evolution

(b) Gradient Norm Evolution



(c) Decision Boundary in Test Dataset

Figure 1.15: Elliptic Classifier without noise and random split dataset

Agent	Dataset	N_Data	Misclassified Points	Percentage
0	Training	500	8	1.60%
0	Test	1000	9	0.90%
1	Training	500	8	1.60%
1	Test	1000	6	0.60%
2	Training	500	10	2.00%
2	Test	1000	12	1.20%
3	Training	500	11	2.20%
3	Test	1000	7	0.70%

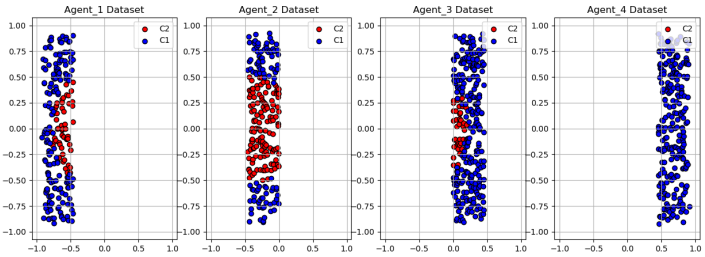
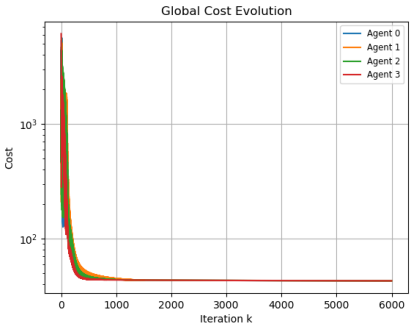
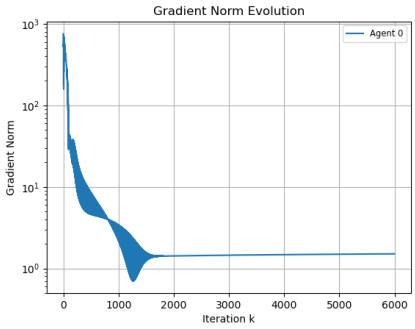


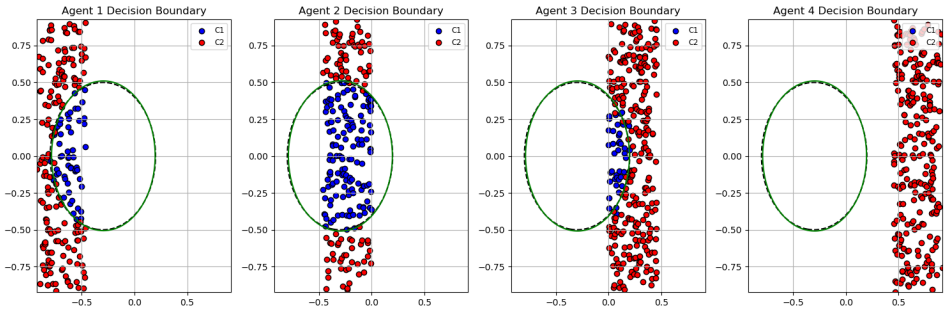
Figure 1.16: Column Split Elliptic Dataset



(a) Cost Evolution



(b) Gradient Norm Evolution



(c) Decision Boundary in Test Dataset

Figure 1.17: Elliptic Classifier with noise and column split dataset

Agent	Dataset	N_Data	Misclassified Points	Percentage
0	Training	499	18	0.90%
0	Test	1000	6	0.60%
1	Training	531	13	0.65%
1	Test	1000	1	0.10%
2	Training	512	11	0.55%
2	Test	1000	2	0.20%
3	Training	459	0	0.00%
3	Test	1000	0	0.00%

Chapter 2

Distributed Aggregative Optimization

2.1 Introduction

Consider N robots in the plane that want to optimize their positions $z_i \in \mathbb{R}^2$ in order to perform multi-robot surveillance

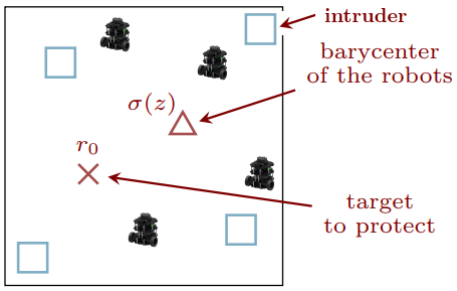


Figure 2.1: Aggregative Optimization Example

The problem reads as:

$$\min_{\mathbf{z}=z_1,\dots,z_N} \sum_{i=1}^{N} \ell_i(z_i, \sigma(\mathbf{z})) = \gamma_i \|z_i - r_i\|^2 + \|\sigma(\mathbf{z}) - r_0\|^2 \tag{2.1}$$

Where $\sigma(\mathbf{z}) \in \mathbb{R}^2$ is the barycenter of the robot, $r_i \in \mathbb{R}^2$ is the local intruder for each robot, $r_0 \in \mathbb{R}^2$ is the common target to protect and $\gamma_i \in \mathbb{R} > 0$ is a trade-off parameter that weights the penalty due to the distance of each agent from the local intruder.
Apply a gradient method in a centralized system would be:

$$z_i^{k+1} = z_i^k - \alpha \frac{\partial}{\partial z_i} \left[\sum_{j=1}^{N} \ell_j(z_j, \sigma(z_1, \dots, z_N)) \right] \Bigg|_{z_1=z_1^k,\dots,z_N=z_N^k} \tag{2.2}$$

However in a distributed system each robot can only receive information from its neighbour according to a graph G. The barycenter $\sigma(\mathbf{z})$ and the gradient of the cost function $\frac{\partial}{\partial \mathbf{z}_i}\sum_{j=1}^{N}\ell_j(\mathbf{z}_j,\sigma(\mathbf{z}))$ are therefore global information that are not available locally. For this reason a local estimate of the barycenter and of the gradient is needed.

2.2 Aggregative Tracking

The tracking idea of the gradient tracking is to applied into this context in order to estimate the global barycenter and the global gradient. Each agent therefore:

- Knows only it's local cost: $\ell_i(z_i, s_i)$
- Maintains an estimate z_i^k of $z_i^\star$
- Maintains an estimate s_i^k of the global barycenter $\sigma(\mathbf{z^k})$)
- Maintains an estimate v_i^k of the global gradient $\sum_{j=1}^{N}\nabla_2\ell_j\left(z_j^k,\sigma(\mathbf{z}^k)\right)$

For each agent the update rule is:

$$z_i^{k+1} = z_i^k - \alpha\left(\nabla_1\ell_i(z_i^k, s_i^k) + v_i^k\right), \qquad z_i^0 = z_{i_{init}} \tag{2.3}$$

$$s_i^{k+1} = \sum_{j\in\mathcal{N}_i} s_j^k + z_i^{k+1} - z_i^k \qquad s_i^0 = z_i^0 \tag{2.4}$$

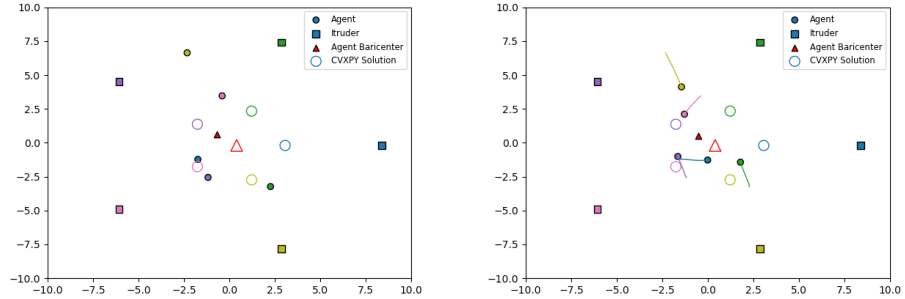
$$v_i^{k+1} = \sum_{j\in\mathcal{N}_i} v_j^k + \nabla_2\ell_i(z_i^{k+1}, s_i^{k+1}) - \nabla_2\ell_i(z_i^k, s_i^k), \quad v_i^0 = \nabla_2\ell_i(z_i^0, s_i^0) \tag{2.5}$$

In our implementation we just wanted the agents to keep the formation tight (close to the barycenter of the formation) while trying to reach the private intruders. In order to do that the agent private cost function has been defined as:

$$\ell_i(z_i, s_i) = \gamma\|z_i - r_i\|^2 + \|s_i - z_i\|^2 \tag{2.6}$$

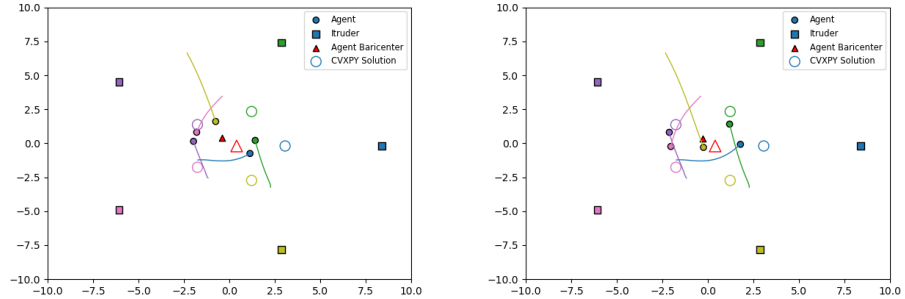
2.2.1 Results

We tested the algorithm with different size of agents changing also the trade-off parameter γ.



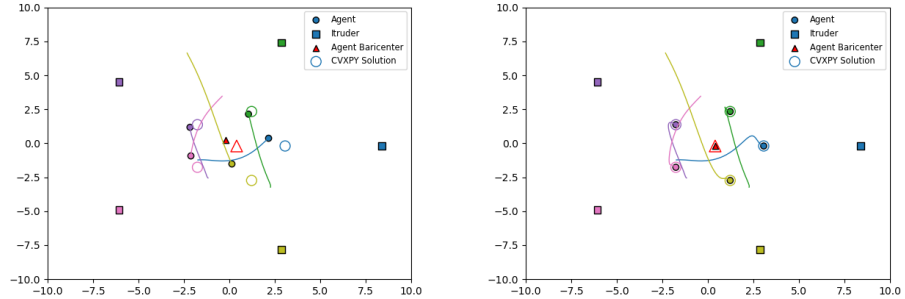
(a)

(b)



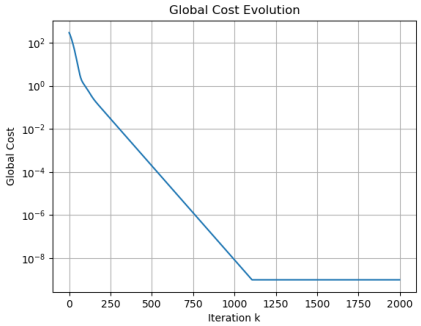
(c)

(d)

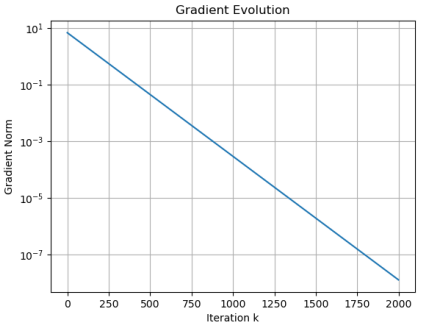


(e)

(f)

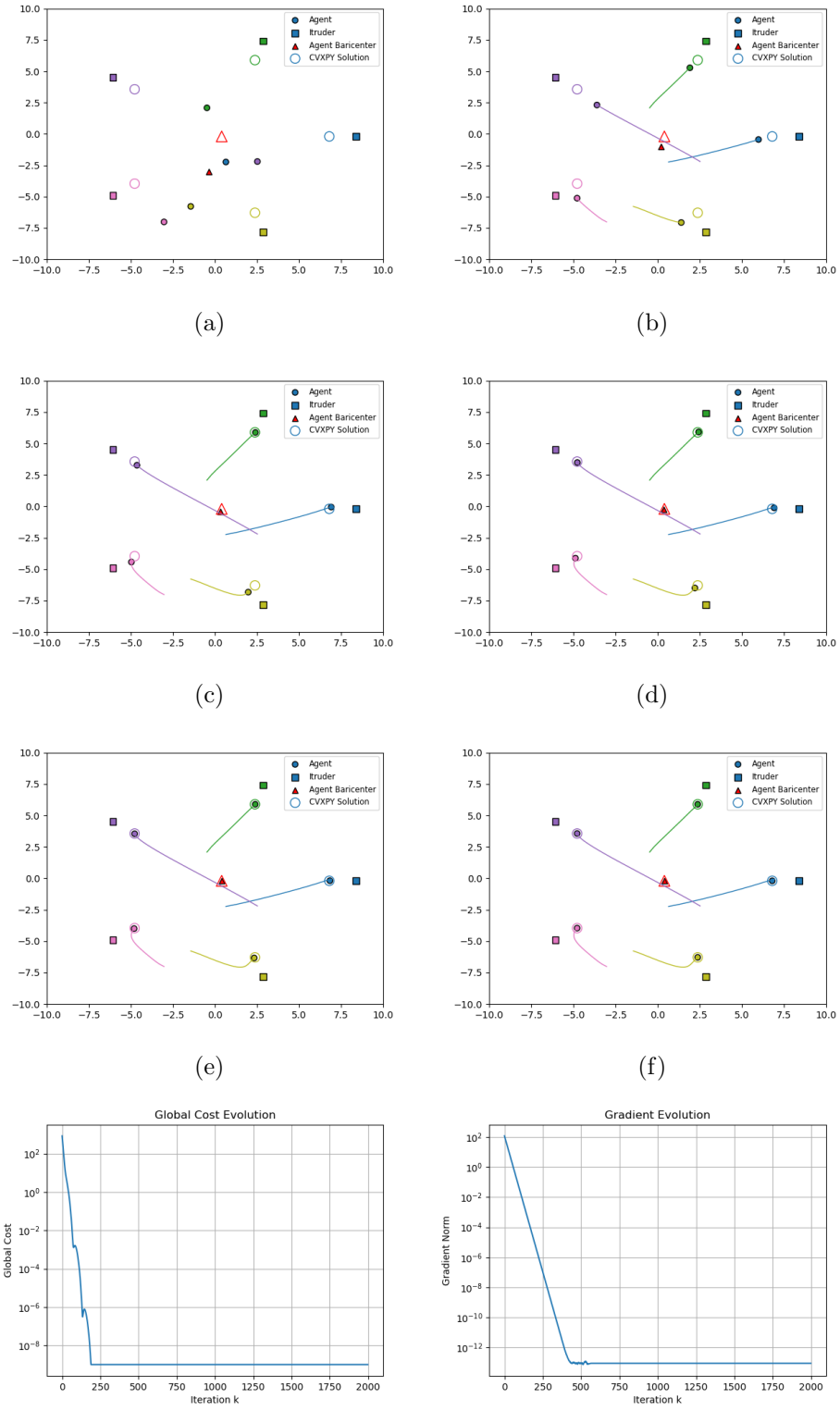


(g) Global Cost Evolution



(h) Gradient Norm

Figure 2.2: Aggregative Tracking $\gamma = 0.5$ $N = 5$



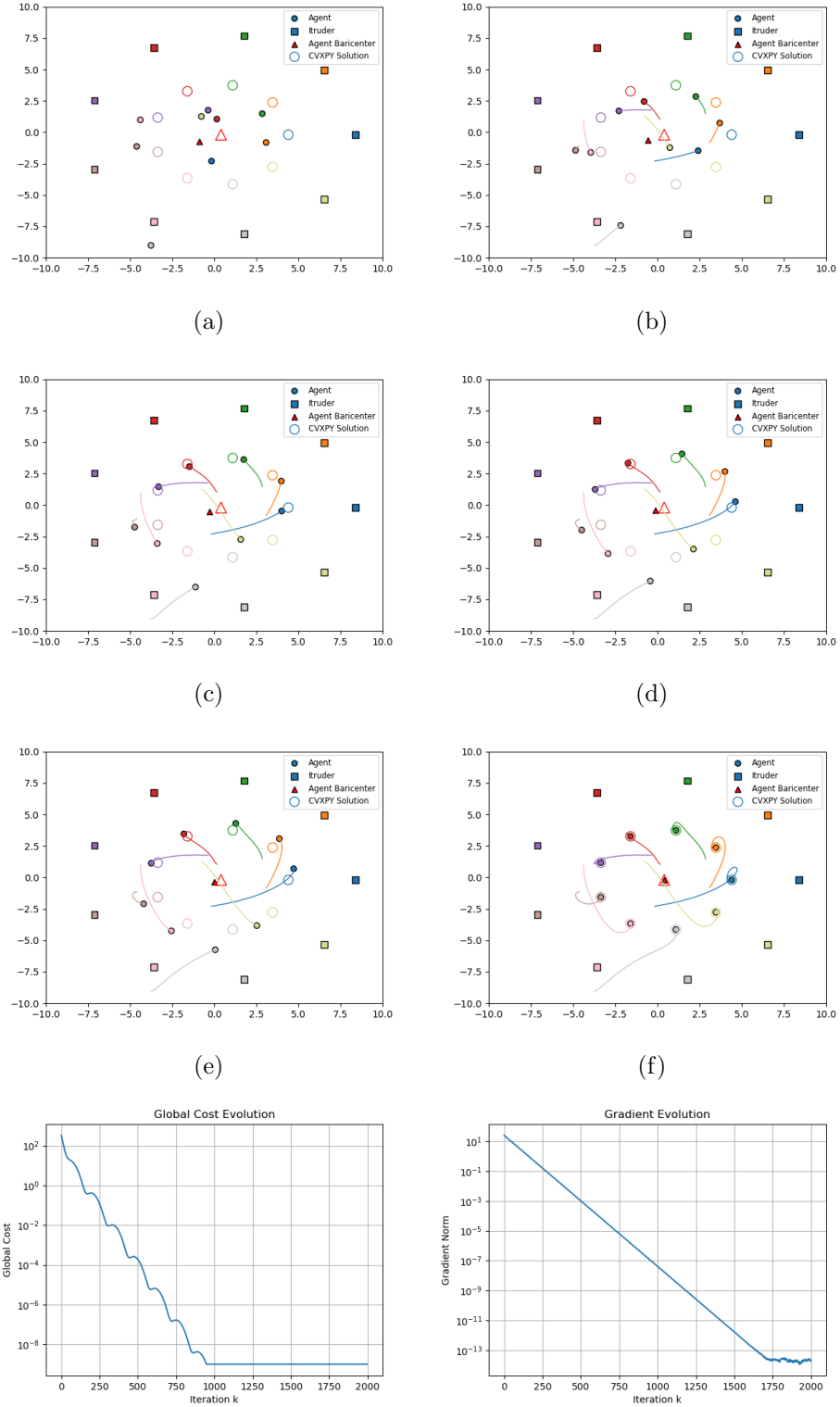
(a) (b)

(c) (d)

(e) (f)

(g) Global Cost Evolution (h) Gradient Norm

Figure 2.3: Aggregative Tracking $\gamma = 4$ $N = 5$



(a) (b)

(c) (d)

(e) (f)

(g) Global Cost Evolution (h) Gradient Norm

Figure 2.4: Aggregative Tracking $\gamma = 1$ $N = 9$

2.3 ROS Implementation

For the ROS2 implementation we defined first of all a ROS2 Custom message: "AggregativeOptimization.msg". This message is the one used for communication between the agents and contains the following informations:

- int32 agent_id
- int32 iteration
- geometry_msgs/Point s
- geometry_msgs/Point v

Then we created a node in order to emulate an agent. This node must be launched with the following parameters:

- "id": The id value of the Agent.
- "Nii": A list containing the ID of the agent neighbour.
- "Adj": A list containing the a_{ij} value for each neighbour.
- "z_init": The initial position of the agent.
- "r_init": The position of the intruder to control.
- "gamma": The trade-off parameter for the local cost function.
- "alpha": The step size for the update of the state.

Each agent create an AggregativeOptimization message publisher on a topic called "/information_agent_id" and create subcription to the same topic for all the id in the list of neighbours. Moreover each agent publish two Marker messages in order to visualize the position of the Agent and of the respective Intruder.. A secondary node has been created in order to collect all the markers from the agents and publish them on a single topic for RViz2 visualization.

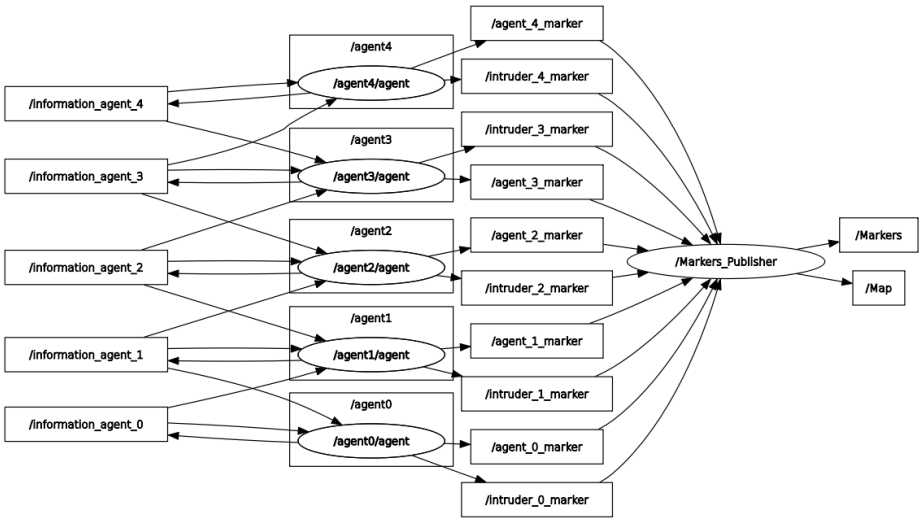
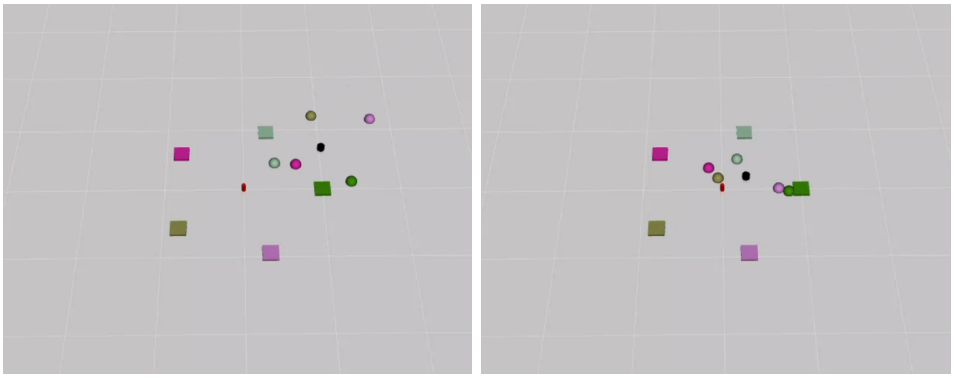


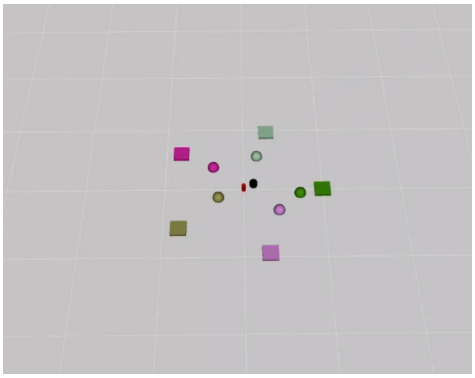
Figure 2.5: ROS2 connection with $N = 5$ Agents and G="Path Graph + Self Loops"

The algorithm is basically the same of the previous section. Each agent has a buffer dictionary where collects the neighbour's message and proceed with the a new iteration when it has available in the buffer all the neighbour messages related to the right iteration. In rviz2 visualization the agents are represented by sphere, intruder by cube, the target to protect by a thin red cylinder while the actual barycenter of the team is represented by a black cylinder.

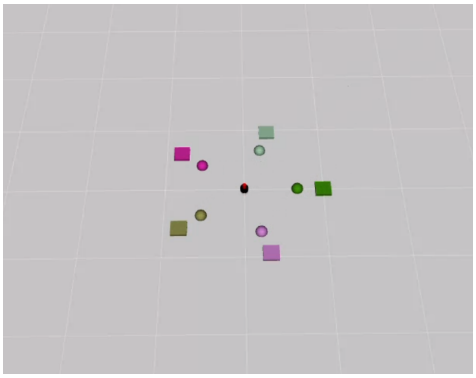


(a)

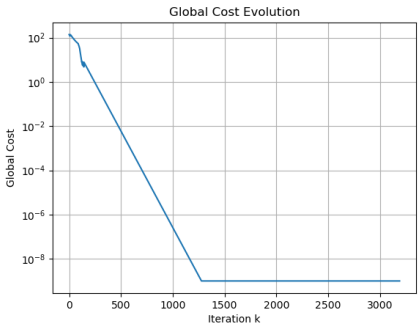
(b)



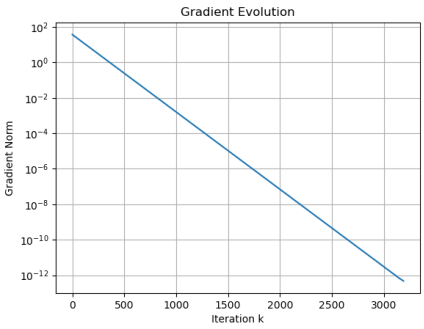
(c)



(d)



(e) Global Cost Evolution



(f) Gradient Norm Evolution

Figure 2.6: Rviz2 Visualization of the Algorithm

2.4 Navigation through a Corridor

In order to emulate a corridor we added a potential term to the local cost function. First of all we have defined a function that could be used in order

to represent the wall position. The candidate function is:

$$g\left(x,y\right)=y^2-0.5x^2-0.5\geq 0\quad\{-2<x<2\} \tag{2.7}$$

This function represents the regions where the wall exist, therefore the constraint for the minimization problem is given by:

$$g\left(x,y\right)=y^2-0.5x^2-0.5\leq 0\quad\{-2<x<2\} \tag{2.8}$$

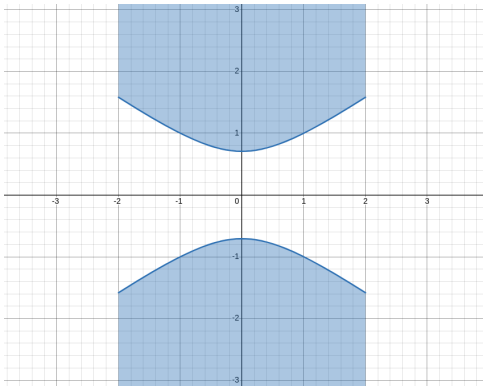


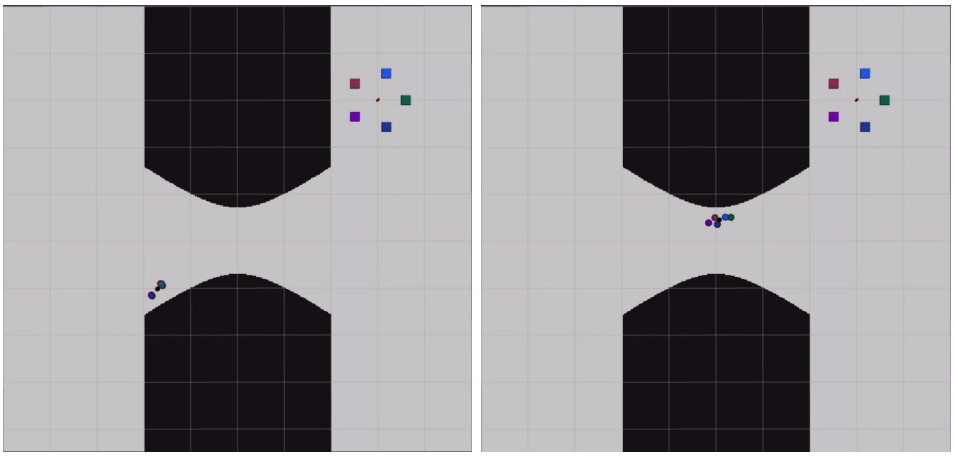
Figure 2.7: Wall Function Visualization: Blue = Wall, White = Free Space

In order to embed the constraint in the minimization problem and treat it as an unconstrained problem an extra term is added into the local cost function of each agent as a barrier function:

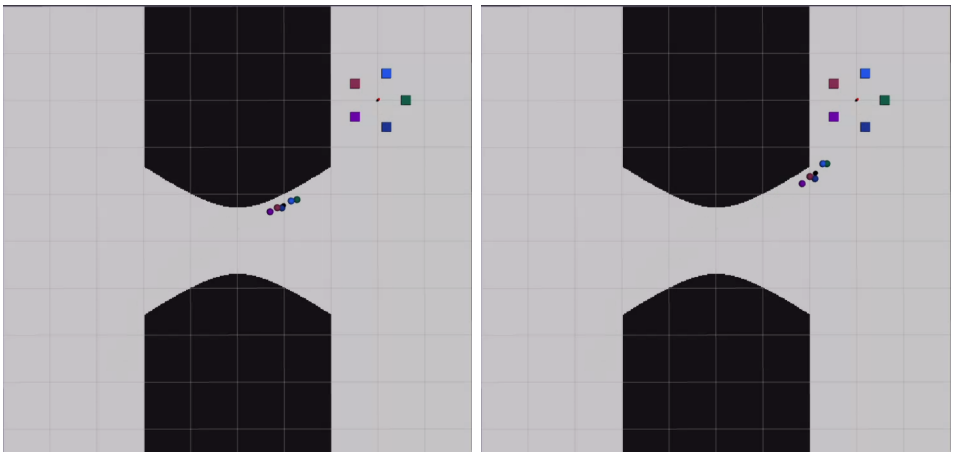
$$\ell_i\left(z_i,s_i\right)=\gamma_i\|z_i-r_i\|^2+\|s_i-r_0\|^2+2\|z_i-r_0\|^2-k\log(-g\left(x,y\right)) \tag{2.9}$$

The barrier function goes toward $+\inf$ when the agent is close to the boundary. In doing that, the gradient for the update will push far away the agent from the wall allowing to let the agent go through the corridor when they need to change room. In order to visualize the behaviour on RViz2, an OccupancyGrid map has been generated representing the corridor.

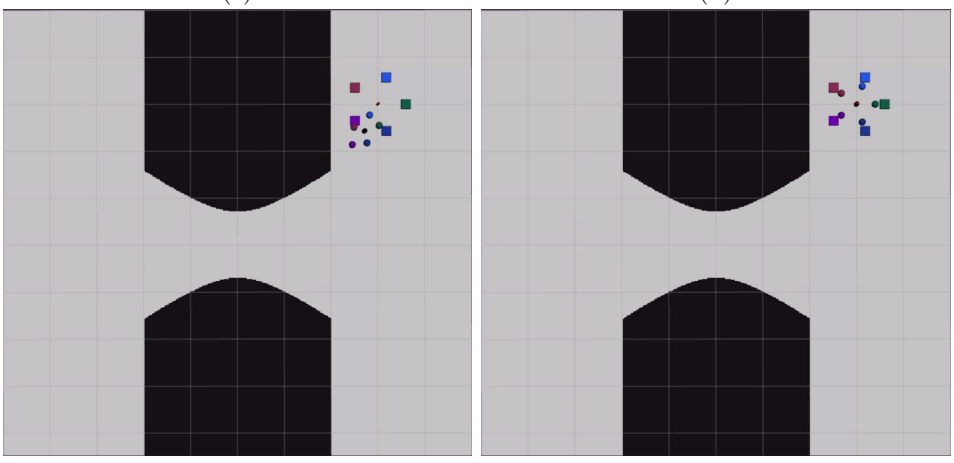
As in the previous visualization, the agents are represented by sphere, the intruder by squares while the target to protect and the baricenter of the team respectively by red and black cylinders.



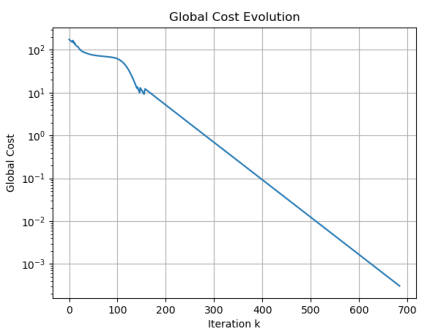
(a) (b)



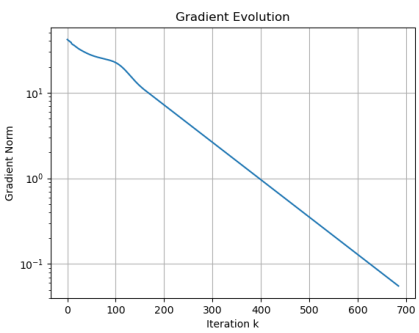
(c) (d)



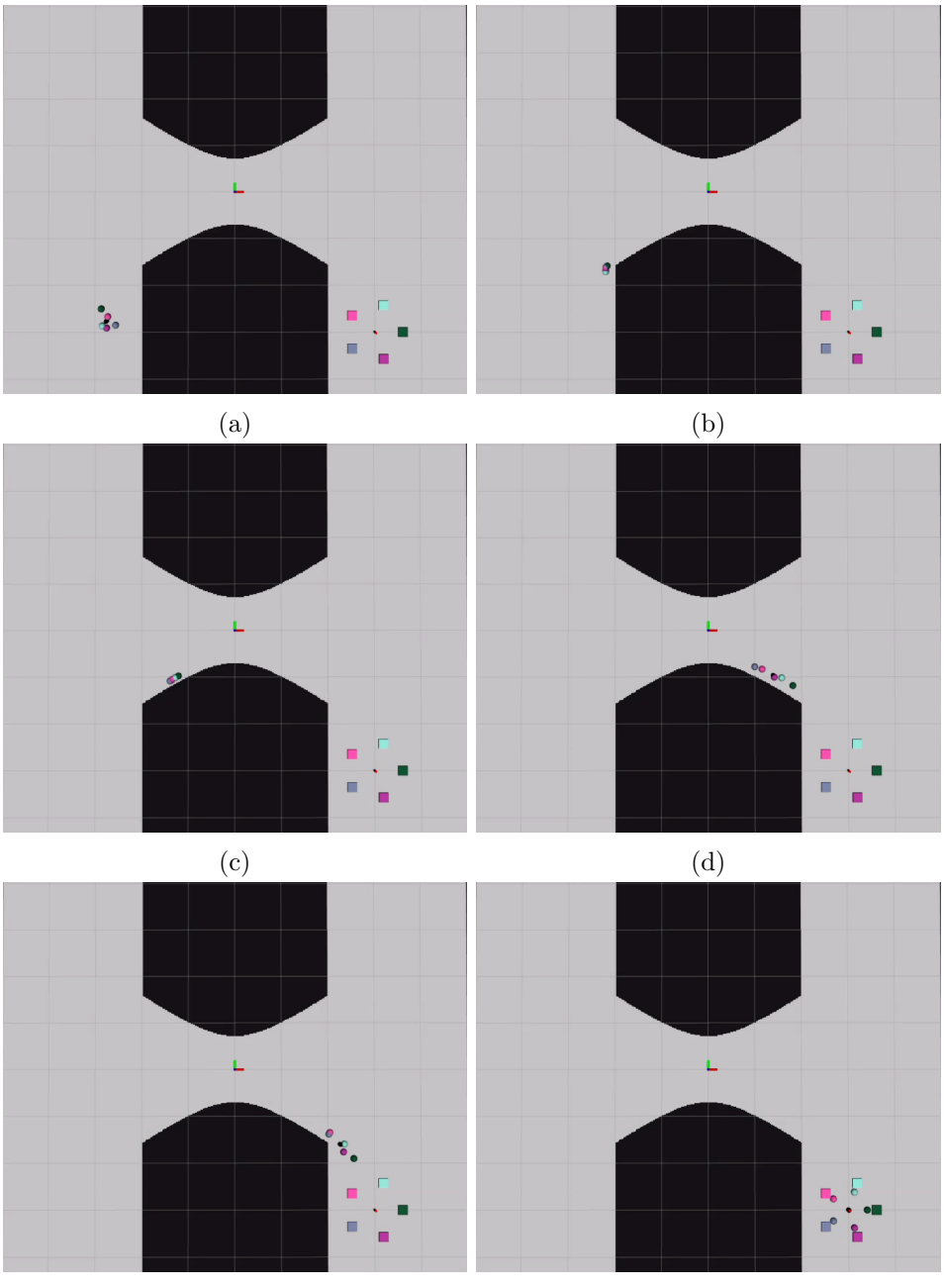
(e) (f)



(g) Global Cost Evolution



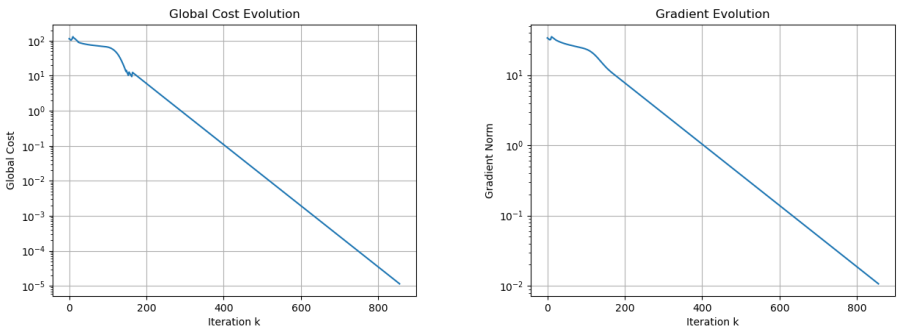
(h) Gradient Norm Evolution



(a) (b)

(c) (d)

(e) (f)



(g) Global Cost Evolution (h) Gradient Norm Evolution

Figure 2.9: Rviz2 Visualization of the Algorithm with barrier function cost

Conclusions

This project successfully designed and implemented distributed algorithms to tackle two distinct tasks: Non-Linear Distributed Classification and Distributed Aggregative Optimization. For the first task, we developed a distributed gradient tracking algorithm and evaluated its efficiency and reliability using various types of graphs. This algorithm proved essential for enabling the distributed classification of a given dataset. Additionally, we implemented a centralized non-linear classification algorithm, which was then integrated with the distributed approach to create a comprehensive distributed non-linear classifier. The results from these implementations indicate strong performance and reliability across different graph structures. In the second task, we aimed to implement a distributed control algorithm to maintain tight formation among robots while allowing them to stay close to their private targets. Initially, a Python script for the Aggregative Tracking algorithm was developed. This script laid the groundwork for the subsequent ROS2 implementation, which provided a more realistic simulation of neighborhood communication among agents. Moreover, the inclusion of an obstacle avoidance term in the local cost function successfully emulated a corridor navigation scenario, enhancing the robustness of the algorithm. Overall, the project achieved its objectives effectively. However, further improvements can be made, particularly in optimizing the code for the classification task to reduce computational effort. These enhancements will likely bolster the efficiency and scalability of the algorithms, making them more suitable for practical, real-world applications.