UNIVERSITÀ DI BOLOGNA

School of Engineering

Master Degree in Automation Engineering

Distributed Autonomous Systems

DISTRIBUTED CLASSIFICATION VIA NEURAL NETWORKS AND FORMATION CONTROL

Professors:
Giuseppe Notarstefano
Ivano Notarnicola

Students:
Simone Cenerini
Giulia Cutini
Riccardo Paolini

Academic year 2022/2023

Abstract

Contents

Introduction

Motivations

Contributions

Chapter 1

Distributed classification via Neural Network

In the first task we were asked to correctly classify a set of grayscale images. The dataset, downloaded from the Keras `mnist`, collects images of handwritten digits of 28×28 pixels each one, an example in figure 1.1. In order to perform our task we were asked to implement a distributed classification: given a predefined number of agents running each one a neural network with the same structure, we were asked to implement a Gradient Tracking algorithm to ensure consensus of the connection weights of the several neural networks runned separately by different agents.

1.1 Initial setup

The first step was to prepare the dataset by a reshape and a normalization of the images. We have flattened the 28×28 pixels grayscale images in order to obtain a $[784, 1]$ column vector and we have normalized the pixels' intensities by dividing each intensity by 255 in order to avoid a saturation of the activation function.

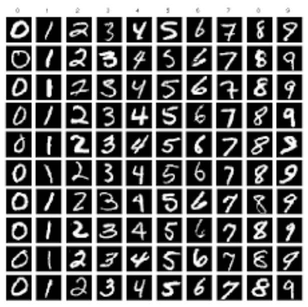


Figure 1.1: Example of images from the `mnist` dataset

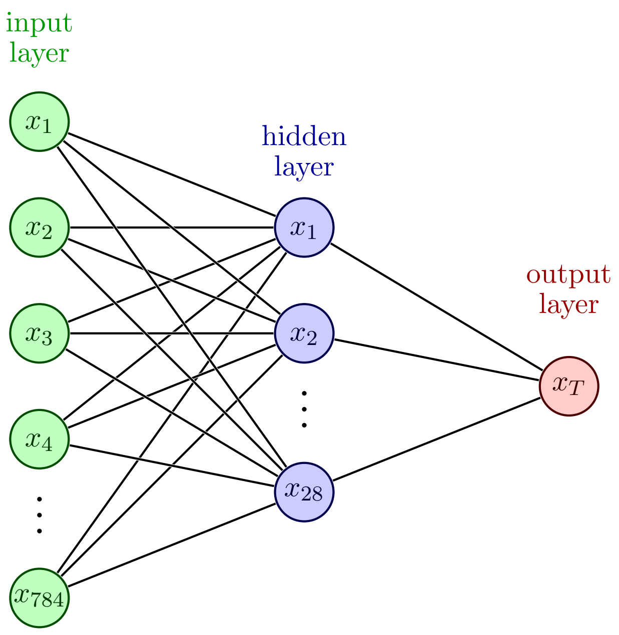


Figure 1.2: Scheme of the neural network for image classification

The classification we were asked to perform was a binary one: we have choose one among the ten digits (named as `target`) and assigned to it the label 1, while, to all the other images, we have assigned the label 0, in the following way:

$$y_i = \begin{cases} 1, & \text{if label} = \texttt{target} \\ 0, & \text{otherwise} \end{cases} \tag{1.1}$$

As a consequence, we have performed a **balancing of the dataset**: in order to efficiently train our neural network we have equalized the representation of both the classes in the dataset (50% of both since we have two classes). In order to do that we have undersampled the number of samples in the majority class.

1.2 Neural Network structure

The structure we have choose to develop for the neural network is a tapered one, represented in figure 1.2. The first layer is composed by 784 neurons, that is the dimension of the flattened image we give as input in the network. In this way each neurons of the first layer gets a single pixel as input. The second layer, the hidden one, has 28 neurons since an higher number of neurons was useless in order to accomplish our task, while it increases a lot the computational effort required to train the network. The last layer has just one single neuron since we are asked to perform a binary classification, and thus only a number between 0 and 1 is required as a prediction.

In this setting we obtain two sets of weights, one for the connections between the input and hidden layers, while the second one is for the connections between the hidden and output layers. In particular, each neuron of the generic layer L^t is connected with all the neurons of the previous layer L^{t-1} plus one connection for the bias, thus each set W^t contains $|L^t| * (|L^{t-1}| + 1)$ weights.

The activation function used to introduce non-linearities into the model is the **sigmoid** function

$$\sigma(x) = \frac{1}{1 + \exp^{-x}} \tag{1.2}$$

and its derivative is

$$\frac{d}{dx}\sigma(x) = \frac{\sigma(x)}{1 - \sigma(x)} \tag{1.3}$$

1.3 Training

The training happens as a sequence of epochs, every epoch contains multiple mini-batches that are small sets of images. For each mini-batch the computations are the following:

- Forward pass to compute the prediction;
- Loss and gradient evaluation;
- Backward pass to compute the derivative of the loss function with respect to each weight of the network.
- Weights update.

The loss function we used is the **binary cross-entropy**

$$L = -y_i \cdot log(\hat{y}) - (1 - y_i) \cdot log(1 - \hat{y_i}) \tag{1.4}$$

and its derivative with respect to the activations in the output layer, i.e. the prediction, is

$$\frac{\partial L}{\partial \hat{y}} = -\frac{y}{\hat{y}} + \frac{1 - y}{1 - \hat{y}} \tag{1.5}$$

The initial connection weights for our neural network have been randomly initialized very close to zero, while the bias term has been initialize at 0.5 since our prediction value will be $\in [0, 1]$

1.3.1 Gradient Tracking algorithm

The training we were required to implement was a distributed one: several agents run a neural network with the same structure, over a different training set, then thanks to the **Gradient Tracking algoritm** we make possible to them to reach consensus over the weights. To implement the Gradient Tracking algorithm we compute the new weights x^{k+1} before the forward pass, contrarily the auxiliary variable s^{k+1} is obtained after the backward pass since its computation requires the gradient computed with the new weights x^{k+1}.

$$\textit{implemented} \qquad \begin{aligned} x_i^{k+1} &= \sum_{j \in N_i} a_{ij} x_j^k - \alpha s_i^k \\ s_i^{k+1} &= \sum_{j \in N_i} a_{ij} s_j^k - \nabla f_i(x_i^{k+1}) - \nabla f_i(x_i^k) \end{aligned} \tag{1.6}$$

The adjacency matrix A from which we extract the coefficient a_{ij} is a non-negative matrix such that the entry (i, j) is $a_{ij} > 0$ if (i, j) is and edge of the graph, $a_{ij} = 0$ otherwise. In our implementation we have chose the adjacency matrix weight according to the *Metropolis Hastings* rule:

$$A_{ij} = \begin{cases} \frac{1}{\max\{d_i, d_j\}+1}, & \text{if } (i,j) \in E \text{and} i \neq j, \\ 1 - \sum_{h \in N_i} A_{ih}, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases} \tag{1.7}$$

1.4 Experiments and Results

For testing the network we use the test set we created without splitting it, in this way we can check whether the agents provide results that are equal, given that they should reach consensus to the same weights.

The performance have been evaluated using the accuracy as a metric. In addition, given that the predictions are not binary we threshold them at 0.5.

$$\hat{y}_i = \begin{cases} 1, & \hat{y}_i > 0.5 \\ 0, & \hat{y}_i \leq 0.5 \end{cases} \tag{1.8}$$

We made different experiments to verify the functioning of the network both as the hyper-parameters such as step-size change but also as the target class or graph changes.

1.5 First test

In the first experiment we want to inspect the convergence of the network. In this configuration we have 5 agents communicating according to a *cycle graph* 1.3, each of them is trained on 32 mini-batches containing 8 samples each for a total of 256 images.

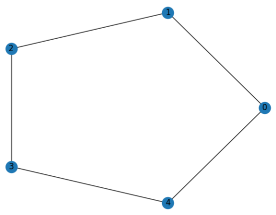
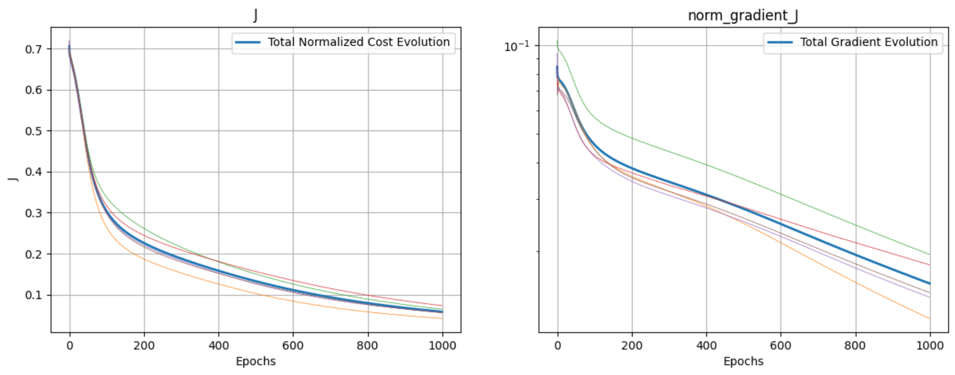


Figure 1.3: Communication graph

From the plots in 1.4 we can appreciate the evolution of the cost function and norm of the gradients, in particular it is possible to observe that the magnitude of the gradients decreases exponentially indicating a fast improvement of the networks.



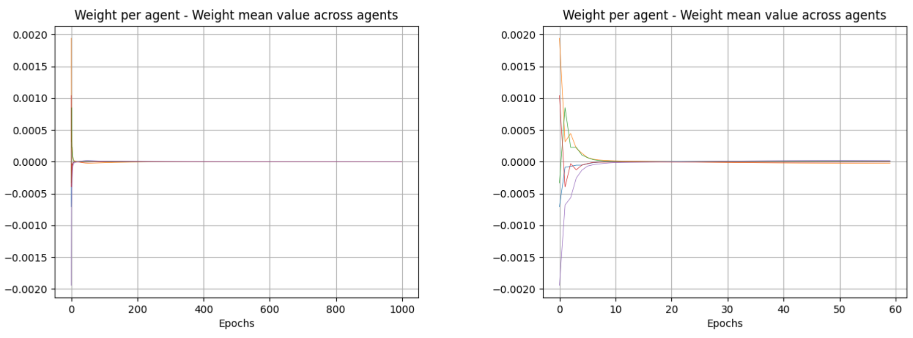
(a) Cost evolution

(b) Gradients evolution

Figure 1.4: Evolution of the cost function and norm of the gradients.

After that, to see if the network converges, we take a single weight from the networks and, for each node, we plot the difference from the mean value of the weight. If the network's weights converge we expect these values to be equal for different nodes, thus they will be equal to the mean and the difference goes to zero. Indeed, from the plot 1.5(a) we can see that these differences approaches the zero value, therefore the networks converge to a consensus. However, it can also be observed that in reaching this consensus the weights of the networks oscillate, but these oscillations have smaller and smaller amplitudes 1.5(b).

In this experiment we obtained an accuracy of 94.9% on the test set.

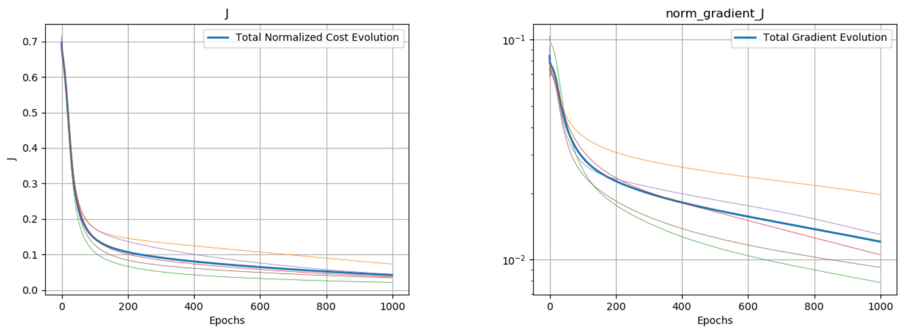


(a) Weight evolution (b) Weight evolution (first 60 epochs)

Figure 1.5: Evolution of a single weight.

1.6 Second test

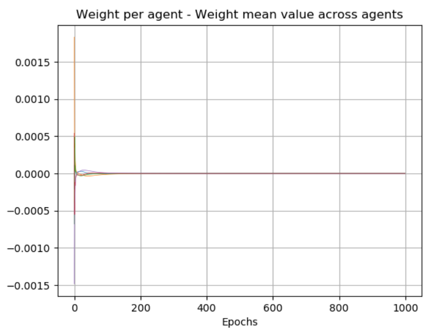
In the second experiment we check whether the target digit impacts the network behaviour, in particular we use the digit 7 as a target instead of the digit 8 used in the first experiment. From the following plots 1.6 and 1.7 we notice that the decrease both in the cost and gradients magnitude is very similar across these experiments, which indicates a good generalization ability of the network, i.e., that the network is capable of distinguishing different input digits.



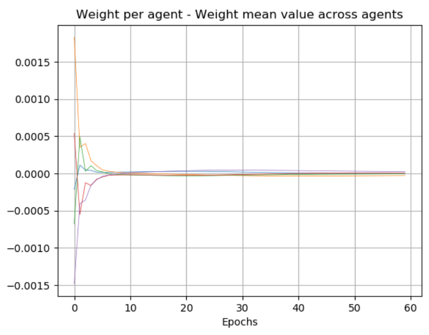
(a) Cost evolution (b) Gradients evolution

Figure 1.6: Evolution of the cost function and norm of the gradients.

We may also note a slightly slower convergence to consensus in this second experiment, in fact the amplitude of the first oscillations 1.7(b) is slightly bigger than in the first experiment.



(a) Weight evolution



(b) Weight evolution (first 60 epochs)

Figure 1.7: Evolution of a single weight.

We have repeated the test previously explained giving as target the different available digits: in the following we show the results obtained in terms of accuracy. The simulations have been obtained with 5 agents connected with a cycle graph, and a step-size of $1e^{-2}$.

Target digit	Accuracy
0	97.66%
1	96.88%
2	Row 3
3	Row 4
4	Row 5
5	Row 6
6	Row 7
7	Row 8
8	Row 9
9	Row 10

1.7 Third test

We have done, as last test, a check on how much the shape of the communication graph influences the velocity (in terms of epochs) required in order to reach the consensus. We have tested three different graphs:

- cycle graph
- star graph
- path graph

As expected the number of epochs required to reach consensus is smaller in case of a better connected graph such as the cicle, while it's bigger for the path graph. We can also observe that, for example in the star graph, the agent in the "middle" of the connection is the faster to reach consensus since it has a lot of neighbors with respect to the agents on the branches of the graph.

Chapter 2

Formation Control

In the second task we were asked to implement in ROS2 a discrete-time version of a **formation control law** for a team of N robots.

2.1 Distance-based formation

The aim of the task was to obtain a desired geometric formation among a group of N autonomous agents, by acting on the relative positions of agents.

The desired formation can be encoded in terms of an undirected graph, called *formation graph*, whose set of vertices is indexed by the team agents $\mathcal{N} = \{1, \cdots, N\}$, and whose set of edges $E = \{(i,j) \in \mathcal{N} \times \mathcal{N} | \, j \in \mathcal{N}_i\}$ contains pairs of agents. Each edge $(i,j) \in E$ is assigned a scalar parameter $d_{ij} = d_{ji} > 0$, representing the distance at which agents i, j should converge to.

An example could be a predefined formation with hexagon shape with the following *distances matrix*:

$$d_{ij} = \begin{bmatrix} 0 & l & 0 & d & h & l \\ l & 0 & l & 0 & d & 0 \\ 0 & l & 0 & l & 0 & d \\ d & 0 & l & 0 & l & 0 \\ h & d & 0 & l & 0 & l \\ l & 0 & d & 0 & l & 0 \end{bmatrix} \tag{2.1}$$

The desired distance value serve as reference value for the control law. The control law computes control signals for each agent based on its current position and the position of the neighboring agents, aiming to drive the agents towards the desired formation. The neighbors of a certain agent i can be extracted from the i-th row of the distances matrix, by taking the indexes of the elements with $d_j > 0$.

By denoting the position of agent $i \in \{1, \cdots, N\}$ at time $t > 0$ with $x_i(t) \in \mathbb{R}^3$, one way to approach distance-based formation control involves the usage of **potential functions**, similar to the following one:

$$V_{ij}(x) = \frac{1}{4}\bigg(\|x_i - x_j\|^2 - d_{ij}^2\bigg)^2 \tag{2.2}$$

This kind of potential function represents the energy associated with the relative positions of the agents. By minimizing this potential function, the agents can achieve and maintain the desired formation. As a consequence, the control law we have implemented for each agent i is the following:

$$\dot{x}_i(t) = f_i(x(t)) = -\sum_{j \in \mathcal{N}_i} \bigg(\|x_i - x_j\|^2 - d_{ij}^2\bigg)(x_i - x_j) \tag{2.3}$$

Since we were required to work in discrete time, we denote with $p_i^k \in \mathbb{R}^3$ the discretized version of $x_i(t)$ at iteration $k \in \mathbb{N}$ and, once we have computed the dynamics update for agent i we discretize it with Euler formula in the following way:

$$p_i^{k+1} = p_i^k + \Delta f_i(p^k) \tag{2.4}$$

where $\Delta > 0$ is the sampling period.

2.2 ROS2 implementation

The solution proposed in previous section has been implemented in ROS2 environment with a code written in Python language.

By means of a launch file (named `formation_control.launch.py`) we can generate the desired number of agents and give them the required informations in order to accomplish the target formation. In particular for each agent the launch file generates a ROS2 node that executes the file named `the_agent.py`. Each node gets the following main parameters from the launch file:

- id: a number from 0 to N associated to the agent's identity
- initial position, randomly generated
- distances: each agent gets only "its" row of the distances matrix. In this way it will be able to know who are its neighbors and the distances he has to keep from them in the desired configuration
- maximum number of dynamics update

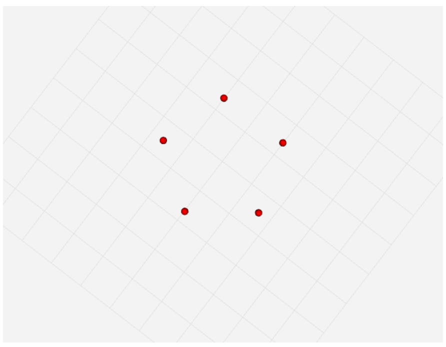
This file manages the dynamic evolution of the agent and the communication with the other ones (the neighbors' ones). For what regards the communication, we have used a publisher-subscriber protocol implemented in ROS2: each node communicates its position only to its neighbors. The communication between nodes is made using a topic for each edge defined in the formation graph. In order to write to a topic or to read from it, the node must know the name of the topic he want to access.

We have developed also a plotter agent (`the_plotter.py`) that can read from its topic the actual positions of all the other agents in order to visualize the evolution of the trajectories of the whole formation.

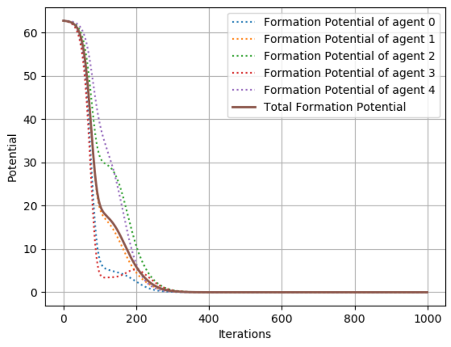
For what regards the simulation of the team behaviour we have used RViz.

2.2.1 Results

In the following we can see some results obtained during the simulation of the `formation_control` package. We have tested several formation-shapes that involve different number of agents.



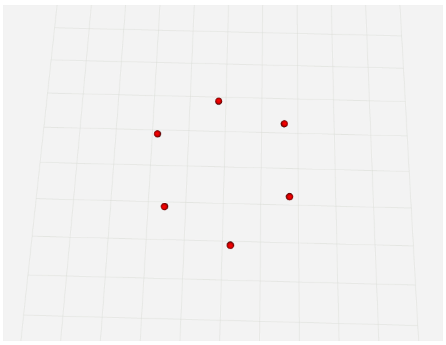
(a) RViz visualization



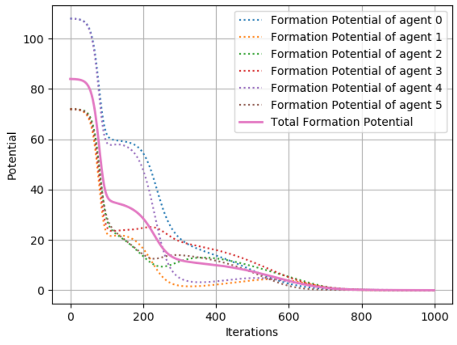
(b) Formation potential plot

Figure 2.1: Pentagon formation with 5 agents

First of all we have tried 2D-formations like a pentagon and an hexagon, the results are represented respectively in figure 2.1 and 2.2. We have initialized the agent's positions randomly and close to zero for the x and the y dimensions, while z position has been initialized equal to zero. Since our target formation is planar, the z remains always equal to zero during the whole evolution. We show both the final RViz visualization and the potential evolution during the iterations: for the single agents and for the whole formation. The potential diminishes along the iterations while the agents reach the desired configuration. For these formations we have also

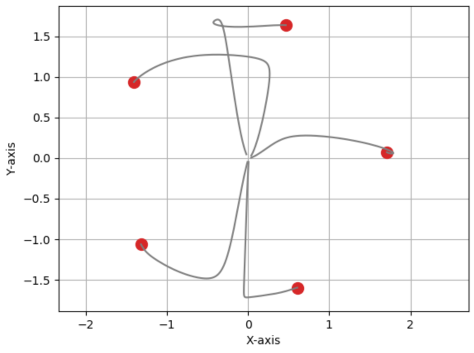


(a) RViz visualization

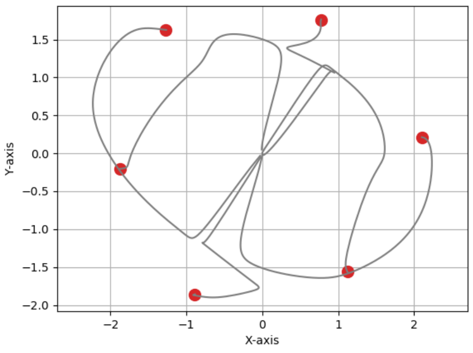


(b) Formation potential plot

Figure 2.2: Hexagon formation with 6 agents

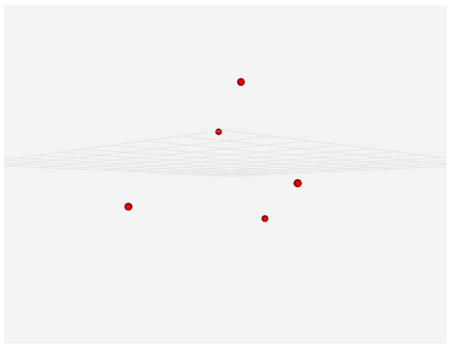


(a) Paths for pentagon formation

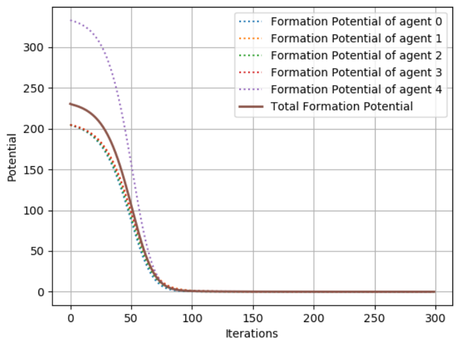


(b) Paths for hexagon formation

Figure 2.3: Paths followed by agents to reach formation



(a) RViz visualization



(b) Formation potential plot

Figure 2.4: Square-based piramid with 5 agents

represented in figure 2.3 the paths the single agents follow during the time evolution in order to accomplish the formation.

After that, we have tried a 3D-formation, a square-based piramid. In order to do that we have initialized all the x, y and z positions close to zero. In figure 2.4 we can see the results obtained with simulations. The plane the "base-agents" chose, changes from one simulation to the others since we don't have any constraint in this sense.

2.3 Collision avoidance

In formation control, the control laws typically consists of two components: attraction and repulsion. The attraction component pulls the agents towards their desired positions relative to their neighbors, while the repulsion component prevents collision by creating a repulsive force when the agents get too close to each other.

The control law that models the agent's dynamics up to now, had just one component, the one linked to the *formation potential*, formula 2.2, this was not enough in order to avoid collision among agents (as we can see in figure 2.5).

For this reason the **task 2.2** asked to introduce a second potential to avoid agent's collision. In order to do that the following proper barrier function has been used:

$$V_{ij}(x) = -\log(\|x_i - x_j\|^2) \tag{2.5}$$

As a result, the control law has a new term due to this potential that is

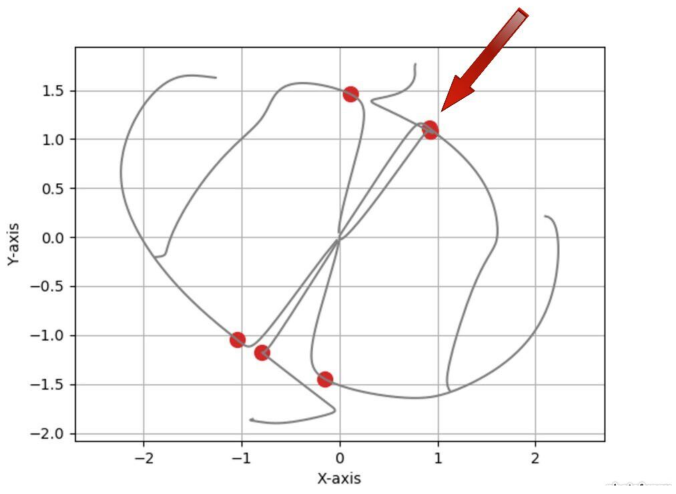


Figure 2.5: Collision among two agents during hexagon formation

the following one:

$$\dot{x}_i(t) = -2 \sum_{j \in \mathcal{N}_i} \frac{(x_i - x_j)}{\left(\|x_i - x_j\|^2 - d_{ij}^2 \right)} \tag{2.6}$$

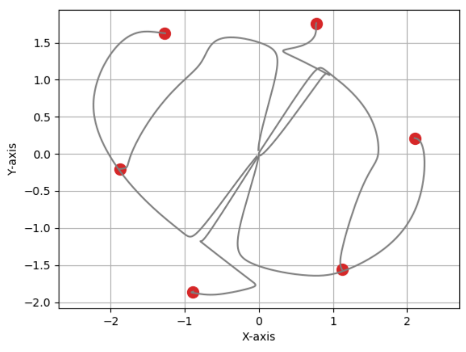
And the total control law becomes:

$$\dot{x}_i(t) = -\sum_{j \in \mathcal{N}_i} \left(\|x_i - x_j\|^2 - d_{ij}^2 \right)(x_i - x_j) - 2\frac{(x_i - x_j)}{\left(\|x_i - x_j\|^2 - d_{ij}^2 \right)} \tag{2.7}$$

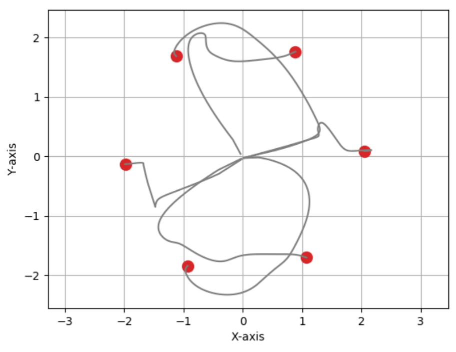
2.3.1 Results

In figure 2.6 we can see the different shape of the paths followed by the agents in order to reach the same hexagon formation. The plot on the left shows the paths due to the dynamic evolution expressed in formula 2.3 (the one without the collision avoidance term), while the plot on the right shows the behaviour due to the introduction of collision avoidance term (formula 2.7).

In figure 2.7 we can see the evolution of the two contributions of the overall potential function. The formation potential, the attractive one that pulls the agents toward the desired formation, diminishes but we can see a "flat area" when the agents goes too close to some neighbors. Accordingly, the barrier potential that repels the agents from neighbors, has a peak.



(a) Path followed by agents without collision avoidance term



(b) Path followed by agents with collision avoidance term

Figure 2.6: Comparison among paths with and without collision avoidance term

2.4 Moving formation and leaders control

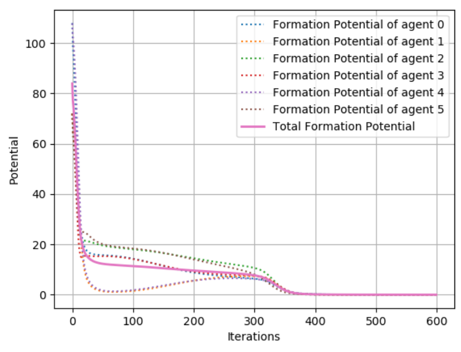
At this point robots are partitioned in two groups, namely leaders and followers. The objective of **task 2.3** is to control the translation and/or the rotation of the formation (by giving proper inputs to the leaders) while maintaining the chosen pattern. Distance constrains are invariant to both translation and rotation of the formation, so the distance-based approach can be applied to realize translational and rotational formation maneuvers.

We have modified the `formation_control.launch.py` in order to pass to the `the_agent.py` file the information about their identity (leader or follower) and about the motion the leaders have to accomplish.

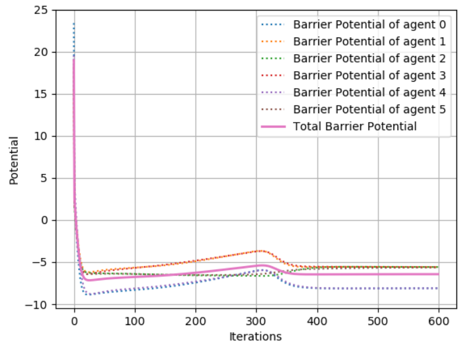
We use the `the_plotter` agent as a "supervisor" that tells all the agents when the target formation has been reached. Once all the agents are in formation the leaders start moving. As a trigger condition the supervisor checks when the potentials' derivatives of all the agents are smaller then a certain threshold. At this moment, he notifies all the agents the formation has been reached and so, the leaders can start moving. This threshold value depends on the formation shape we chose. In figure 2.8 the red line represents this notification criterion.

From this moment on, the dynamics evolution changes: the followers continue with the same motion, trying to maintain the formation, while the leader's dynamic has two contributions, the formation one and the motion we have decided them to follow. Both have also the collision avoidance term.

Figure 2.9 represents the motion of the formation along the x-axis due to a constant input assigned to leaders' dynamic. The leaders are represented



(a) Formation potential (attraction) term



(b) Barrier potential (repulsion), collision avoidance term

Figure 2.7: Potential contribution in hexagon formation with collision avoidance

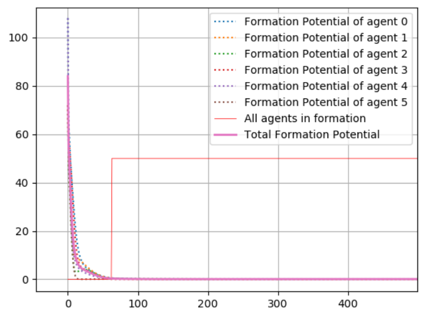


Figure 2.8: Check on the formation accomplishment

as the blue stars.

We have also tried to give the leaders a proper input in order to accomplish a circular motion during the number of iterations given. In figure 2.10 we can see the motion of the hexagon formation: in the left plot we have given as input a radius equal to 2 while in the right one the radius is equal to 5.

2.5 Obstacle avoidance

Task 2.4 asked to us to introduce obstacle avoidance in our `formation_control` package. Obstacle avoidance in formation control refers to the ability of a group of N coordinated agents to navigate through en environment while maintaining a desired formation and avoiding collision with fixed or moving obstacles.

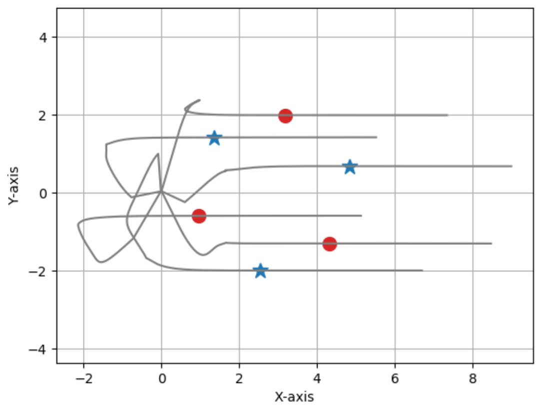
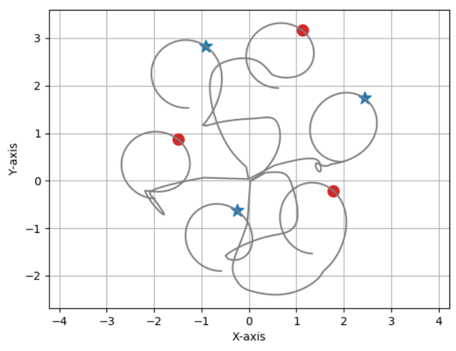
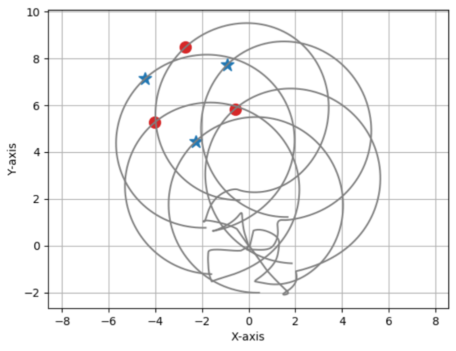


Figure 2.9: Motion of the formation along the x-axis



(a) Moving along a circle of radius 2



(b) Moving along a circle of radius 5

Figure 2.10: Motion of the formation along a circular path

In order to accomplish this task we have created a certain number of fixed obstacles as ROS2 nodes. The obstacles gets their position from the launch file and communicate it to the follower agents. In `the_agent.py` we have still two dynamics: the leaders move accordind to the assigned input while the followers move according to the formation and trying to avoid obstacles.

FARE TEST DANDO UNA TARGET POSITION

Conclusions