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 **5**

1 Distributed classification via Neural Network **6**
1.1 Initial setup 6
1.2 Neural Network structure 7

2 Formation Control **9**
2.1 Distance-based formation 9
2.2 ROS2 implementation 10
2.2.1 Results 11
2.3 Collision avoidance 13
2.3.1 Results 14
2.4 Moving formation and leaders control 15
2.5 Obstacle avoidance 16

Conclusions **18**

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.

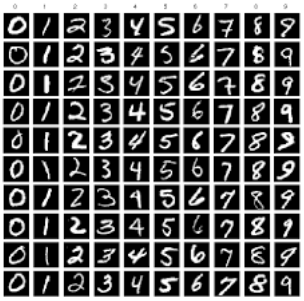


Figure 1.1: Example of images from the `mnist` dataset

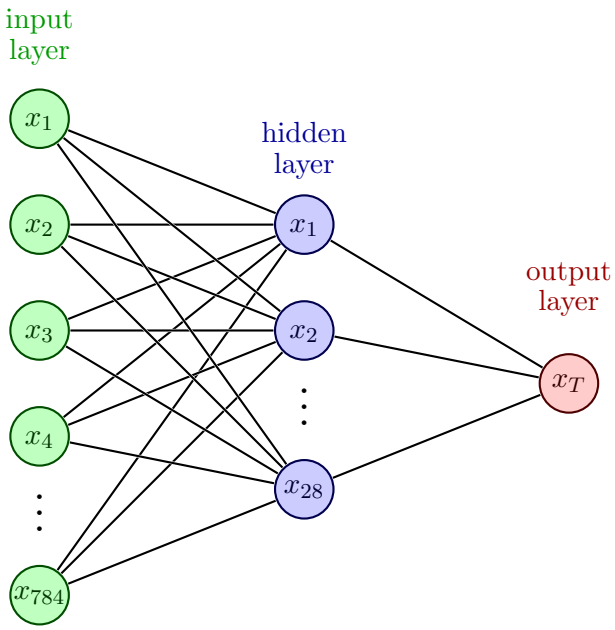


Figure 1.2: Scheme of the neural network for image classification

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 so just a number between 0 and 1 is required as prediction.

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:

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

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:

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

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 time k and, once we have computed the dynamics update for agent i we discretize 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

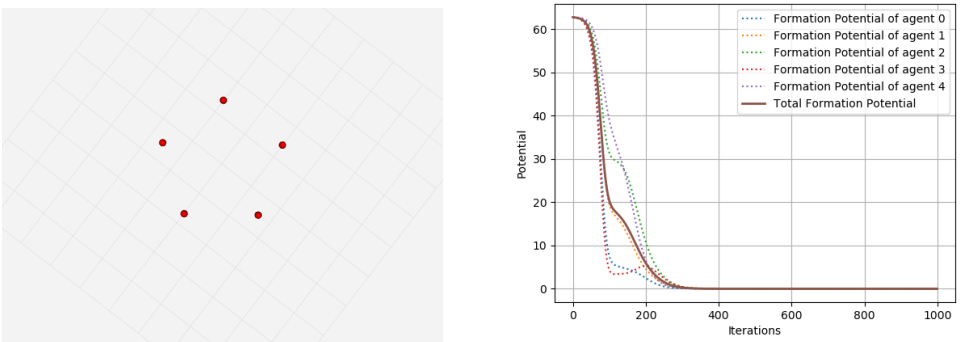
This file manages the dynamic evolution of the agent and the communication with the other ones (the neighbors' ones). For what regards the communication, we have used a publisher-subscriber protocol implemented in ROS2: each node communicates its position only to its neighbors. The communication between 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

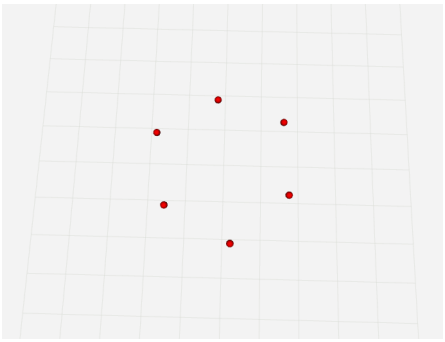
In the following we can see some results obtained during the simulation of the `formation_control` package. We have tested several formation-shapes that involve different number of agents.



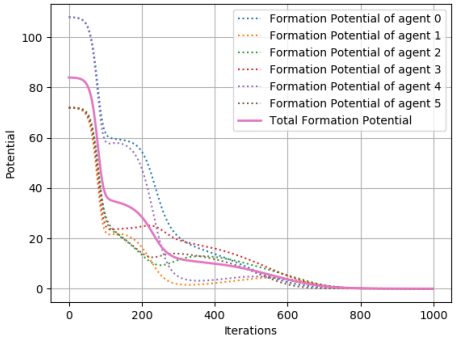
(a) RViz visualization (b) Formation potential plot

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 direction has been initialized equal to zero in such a way to remain 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 represented in figure 2.3 the paths the single agents follow during the time evolution in order to accomplish the formation.

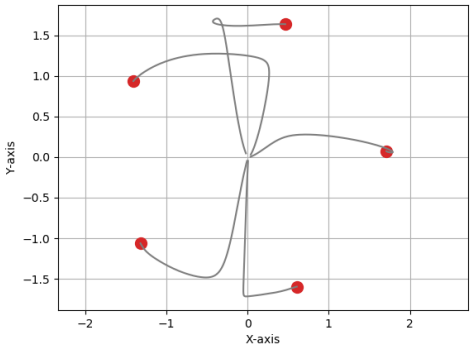


(a) RViz visualization

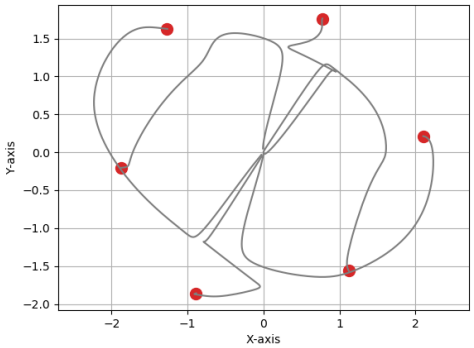


(b) Formation potential plot

Figure 2.2: Hexagon formation with 6 agents

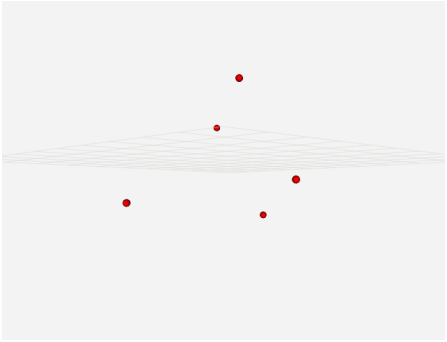


(a) Paths for pentagon formation

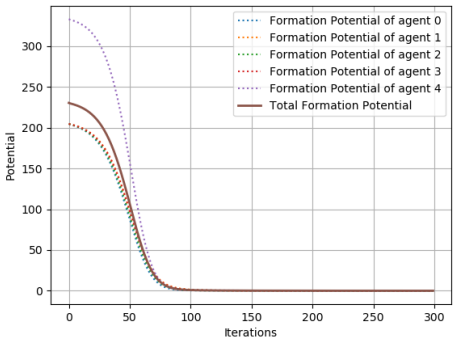


(b) Paths for hexagon formation

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

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 dimensions 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, he 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.

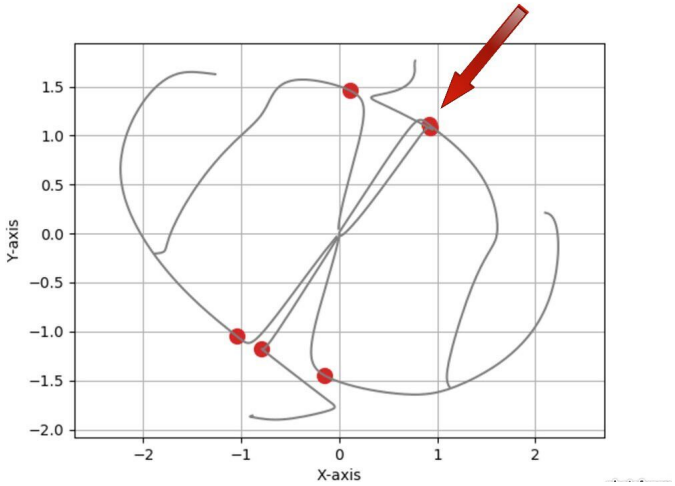


Figure 2.5: Collision among two agents during hexagon formation

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

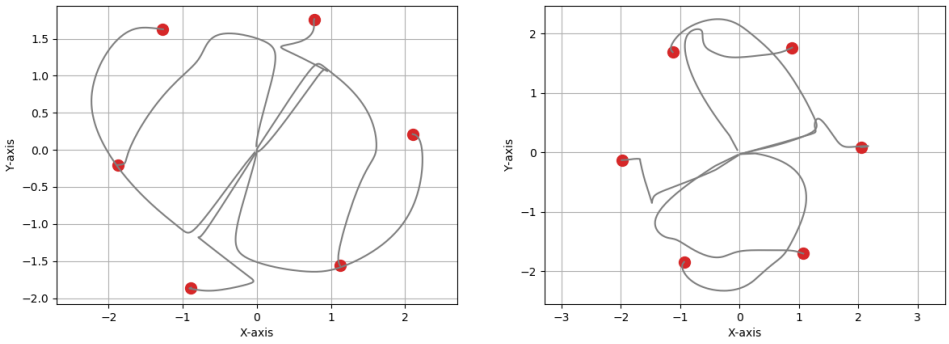
the following one:

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

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



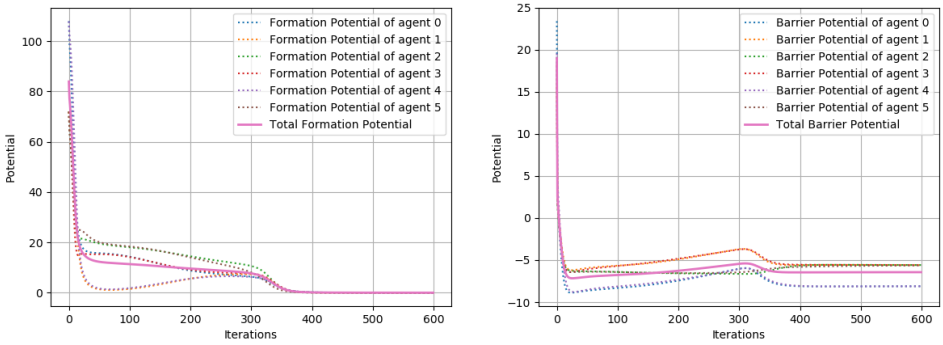
(a) Path followed by agents without collision avoidance term

(b) Path followed by agents with collision avoidance term

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

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 so some neighbors. Accordingly, the barrier potential that repels the agents from other agents, has a peak.



(a) Formation potential (attraction) term

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

Figure 2.7: Potential contribution in hexagon formation with collision avoidance

2.4 Moving formation and leaders control

At this point robots are partitioned in two groups, namely leaders and followers. The objective of this task 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.

In task 2.3 we are asked to apply a certain input to the leaders in order to move the whole formation in a predefined way. 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.

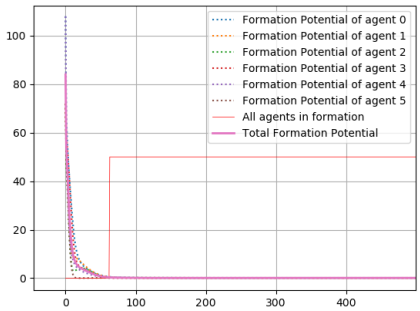


Figure 2.8: Check on the formation accomplishment

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 leader start moving. As a checking condition the supervisor notifies when the potentials of all the agents are less then a certain small threshold for 5 consecutive iterations, 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.

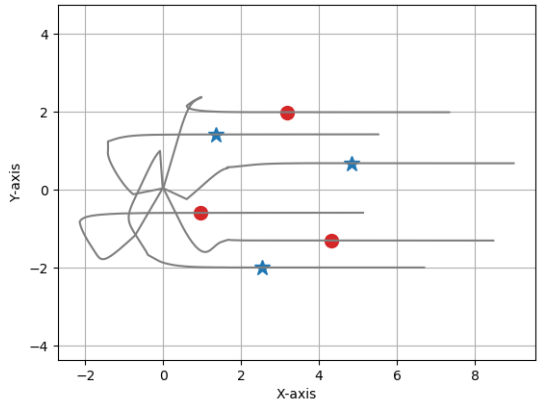


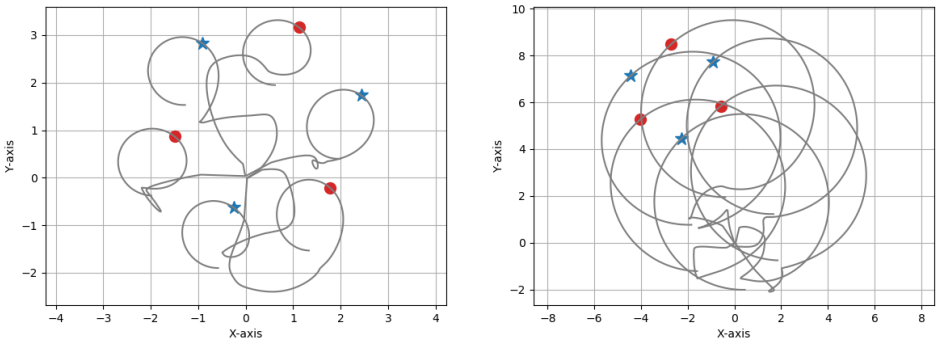
Figure 2.9: Motion of the formation along the x-axis

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 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.



(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 left 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