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:
Federico Cristallo
Federico Iadarola
Nicholas Baraghini

Academic year 2021/2022

Abstract

The report shows the development of two projects that aim to show the application of distributed control algorithm in different fields.

As a first project, a distributed network of supervised classifiers was developed to classify handwritten digits. By exploiting the supervised **MNIST dataset**, and dividing it by a predefined number of agents, a distributed information interchange network was then implemented to obtain convergence of the model of each agent of the network. As a solution approach, a Gradient tracking algorithm has been implemented to ensure consensus of the connection weights of the neural network implemented for each agent, by improving the local descent direction to track the correct gradient.
The algorithm has been implemented flexibly, guaranteeing the user's ability to change the network structure (number of layers, size of the layers) at will. Furthermore, both a binary classification and a multi-classification problem were analyzed in the development of the network.
As a final improvement of the Neural Network, an algorithm for calculating the optimal learning rate was also implemented, to improve the optimization of the classifier. The performance of the algorithm is then evaluated with the test set.
In the second task a Formation Control, implemented by a bearing-based approach, is created in order to control the translation of different formations while, possibly, maintaining the same pattern. This kind of approach relies on inter-neighbor bearings which are invariant with respect to the translation and scale of the formation. The aim of the project is to obtain a rigid formation that can perform predefined geometrical patterns or move from a formation pattern to another, in order to compose a letter of words one at a time. Considering N robotic agents modeled by double-integrator dynamics, whose state includes position and velocity, using followers and leaders approach, firstly some desired formation patterns are achieved by having constant dynamics (zero) for leaders. Then, the control law is augmented considering also the acceleration to allow time-varying leader velocities.
Plots and real-time simulations are provided to show the final results.

Contents

Task 1: Distributed Classification via Neural Networks

Introduction

In this task the goal is to build a binary distributed classifier that recognizes a selected digit from a supervised dataset of ten digits. The distributed gradient tracking is used to train the neural network and reach the convergence of the algorithm. Finally, the obtained solution is evaluated by computing the accuracy of the results on the test set.

Data-set

N agents were considered to cooperatively train a supervised model to classify images of handwritten digits. For this purpose **MNIST** dataset from the **Keras** Python Library, was used
A sample in the dataset consists in a image in greyscale, represented in matricial form as $[0, 255]^{28\times 28}$, and an associated label in the range $\{0, 1, ..., 9\}$.

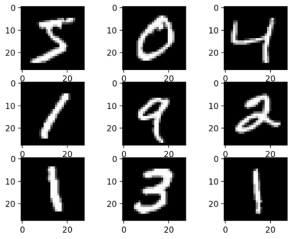

Figure 1: Examples of greyscale MNIST digits

Starting from this framework, we have to pre-process the dataset appropriately to address our classification problem. Therefore, a reshape and a normalization were performed on the images of the dataset to obtain samples in the form of $\mathcal{D}^i \in [0, 1]^{784}$.
The binary classification problem to be solved, aims to classify a selected digit, therefore labels containing the target digit are assigned with 1, otherwise with -1. Thus, selected a target digit, say $\mathcal{D}_{tgt}$, we labeled the whole dataset in order to obtain the *ground-truth class labels*:

$$y^i = \begin{cases} 1 & \text{if } \mathcal{D}^i = \mathcal{D}_{tgt} \\ -1 & \text{otherwise} \end{cases}$$

```
if BINARY:
    # digit to be classified
    CLASS_IDENTIFIED = 4

    # build the label such that it will be 1 if the image contains the digit chosen by CLASS_IDENTIFIED
    Y_train_class = [1 if y == CLASS_IDENTIFIED else -1 for y in y_train]
    Y_test_class = [1 if y == CLASS_IDENTIFIED else -1 for y in y_test]
```

Figure 2: Labeling procedure

In order to evaluate the performance of the final model the entire dataset is then splitted in training and test set. MNIST dataset consists in $70,000$ handwritten digits. We splitted MNIST in a *training set* $\left\{\mathcal{D}^j, y^j\right\}_{j=1}^{60,000}$ and a *test set* $\left\{\mathcal{D}^k, y^k\right\}_{k=1}^{10,000}$. The Training set was splitted equally to each agent of the Communication Network.

Communication Network Design

Considering N agents, a communication graph $\mathcal{G}$ was built. Some constraints are necessary to guarantee the average consensus, as stated in **Consensus Theorem**.

- digraph $\mathcal{G}$ strongly connected: exists a direct path from any node to any other node;

- aperiodic, simply reached by adding self-loops;
- Adjacency matrix A double stochastic: rows and columns sum to 1.

A random binomial graph was generated for the network based on these considerations.

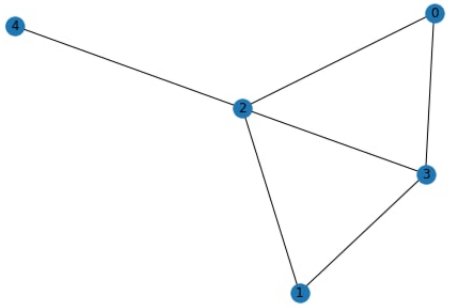

Figure 3: Example of a Binomial Graph

Distributed Neural Network Implementation

First of all, we need to set up the neural network, choosing its structure like the number of hidden layers, of neurons in each layer, the number of epochs and the learning rate. We implemented a structure with variable number of layers and neurons, that can be chosen by the user. As a default value we selected two hidden layers with 512 neurons each. We decided to opt for this solution because it reduces the computation needed but it does not simplify the pattern, providing enough weights for the classification. Since the focus is to deal with distributed systems, each agent run locally the neural network, having as input a different portion of the training set.

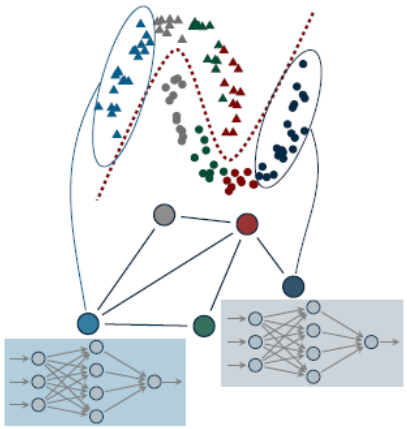

Figure 4: training of NN with local private data $\left\{\mathcal{D}^j, y^j\right\}$, for $j = 1, ..., m_i$

A generic neuron l is a computational unit in a layer vector x_l that has a set of weights u_l. We compute, for each neuron:

$$x_l = \sigma(x^T u_l + u_{l,0}) \tag{1}$$

where σ is the **activation function** and $u_{l,0}$ is the bias. We defined for our code the Sigmoid activation functions (function *sigmoid_fn*).

$$\sigma(\xi) = \frac{1}{1 + e^{-\xi}}$$

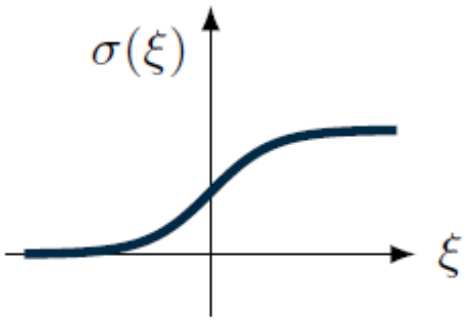

Figure 5: The Sigmoid functions

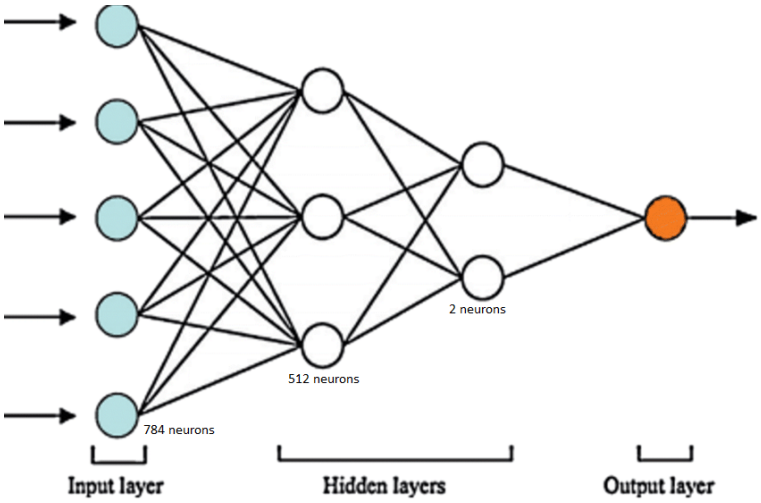

Figure 6: The Neural Network model for binary classification

Algorithm

With all the setup ready, we can now start the algorithm, initializing all the variables that will be useful during the procedure. We computed, for each agent, the initial weights u_0 and the initial Gradient direction y_0.
At this stage, we can run the algorithm in order to fill the network. The first layer ($t = 0$) is set equal to the input image:

$$x_0 = \mathcal{D} = ([\mathcal{D}]_1, ..., [\mathcal{D}]_j), \text{ with } j = 1, ..., m_i$$

For each layer t of the network, the update for each neuron $l = 1, ..., d$ of the next layer $t + 1$ is, like in (1) is known as **inference dynamics**, done in the function *inference_dynamics*:

$$x_{l,t+1} = \sigma(x_t^T u_{l,t} + u_{l,0}), \qquad x_{l,0} = [\mathcal{D}]_l$$

An aggregate formulation of the inference dynamics could be written in the matricial form as:

$$\begin{bmatrix} x_{1,t+1} \\ \vdots \\ x_{d,t+1} \end{bmatrix} = \begin{bmatrix} \sigma(x_t^T u_{1,t}) \\ \vdots \\ \sigma(x_t^T u_{d,t}) \end{bmatrix} \quad \Longrightarrow \quad x_{t+1} = f(x_t, u_t)$$

The **forward propagation**, also known as **forward pass**, refers to the calculation process and storage of intermediate variables, values of the output layers from the inputs data. It's traversing through all neurons from first to last layer. For this reason, in the function *forward_pass*, we iterate the inference dynamics for each layer present in the network.

The next step is to compute the so-called **backward propagation**, that is the essence of the neural network training. It is referred as the process of fine-tuning the weights (de facto learning), based

on the error rate (i.e. loss function) obtained in the previous epoch (i.e. iteration), using a gradient descent algorithm (or, in our case, the **Gradient Tracking)**. The cost function selected for this task is the **Binary Cross Entropy** adapted to handle class labels -1 and 1:

$$J_{i,T}^k(u) = \sum_{i=1}^{N} \frac{(y^i+1)}{2} \log(x_{i,T}^k) + (1 - \frac{(y^i+1)}{2}) \log(1 - x_{i,T}^k)$$

and:

$$\nabla J_{i,T} = \frac{x_{i,T}^k - \frac{y^i+1}{2}}{x_{i,T}^k(1 - x_{i,T}^k)}$$

The following backward propagation is done in the function *adjoint_dynamics*. Starting from the costate at time T:

$$\lambda_{i,T} = \nabla J_{i,T}(x_{i,T}^k; y^i)$$

we construct each value backward in time.

$$\lambda_{i,t} = {A_{i,t}^k}^T \lambda_{i,t+1}$$

$$\Delta u_{i,t}^k = {B_{i,t}^k}^T \lambda_{i,t+1}$$

with $A_{i,t}^k = \nabla_x f(x_{i,t}^k, u_t^k)^T$, $B_{i,t}^k = \nabla_u f(x_{i,t}^k, u_t^k)^T$, where:

$$\nabla_x f(x_{i,t}^k, u_t^k) = \begin{bmatrix} \nabla_x f_1(x_{i,t}^k, u_t^k) & \dots & \nabla_x f_d(x_t^k, u_t^k) \end{bmatrix} = \begin{bmatrix} u_{1,t}^k \sigma'({x_{i,t}^k}^T u_{1,t}^k) & \dots & u_{d,t}^k \sigma'({x_{i,t}^k}^T u_{d,t}^k) \end{bmatrix}$$

$$\nabla_u f(x_{i,t}^k, u_t^k) = \begin{bmatrix} \nabla_u f_1(x_{i,t}^k, u_t^k) & \dots & \nabla_u f_d(x_{i,t}^k, u_t^k) \end{bmatrix} = \begin{bmatrix} x_{i,t}^k \sigma'({x_{i,t}^k}^T u_{1,t}^k) & \dots & 0_d \\ 0_d & \ddots & 0_d \\ \vdots & & \vdots \\ 0_d & \dots & x_{i,t}^k \sigma'({x_{i,t}^k}^T u_{d,t}^k) \end{bmatrix}$$

$\sigma'({x_{i,t}^k}^T u_{l,t}^k)$ is the derivative of the activation function σ evaluated at ${x_{i,t}^k}^T u_{l,t}^k$. This is done in the function *sigmoid_fn_derivative*. and 0_d is a column vector of d zeros. Like the forward pass, the procedure to update the weights is done in a function named *backward_pass*.
The computation is made sequentially, running backward and then forward pass for each image, and agent, in the training set. Locally, each agent compute the descent as an average of the local gradients, after a complete run of its batch of images.

$$\Delta u_{i,avg}^k = \sum_{i=1}^{\mathcal{I}} \Delta u_{i,t}^k$$

In order to track the true gradient, we can introduce a local signal $r_{i,t}$, our before mentioned average of the local gradients Δu_{avg} (we are introducing a new "state" $y_{i,t}$, augmenting the order of the system, but getting a linear rate, in logarithmic scale, on the descent) that will help us steer the tracked gradient to true one. The local descent $y_{i,t}$ is updated as:

$$y_i^k = \sum_{j \in \mathcal{N}_i} a_{ij} y_j^{k-1} + (\Delta u_{i,avg}^k - \Delta u_{i,avg}^{k-1})$$

The $(\Delta u_{i,avg}^k - \Delta u_{i,avg}^{k-1})$, also known as innovation, converging to zero, will tell us how good we are tracking the descent. Then, we can finally update the local network weights with the tracked descent as follows:

$$u_i^{k+1} = \sum_{j \in \mathcal{N}_i} a_{ij} u_j^k - \alpha^k y_i^k$$

This procedure is iterated for k epochs, for the entire network.

Results

Running 100 epochs with 8 agents and 60 images per agent, we obtained the following results:

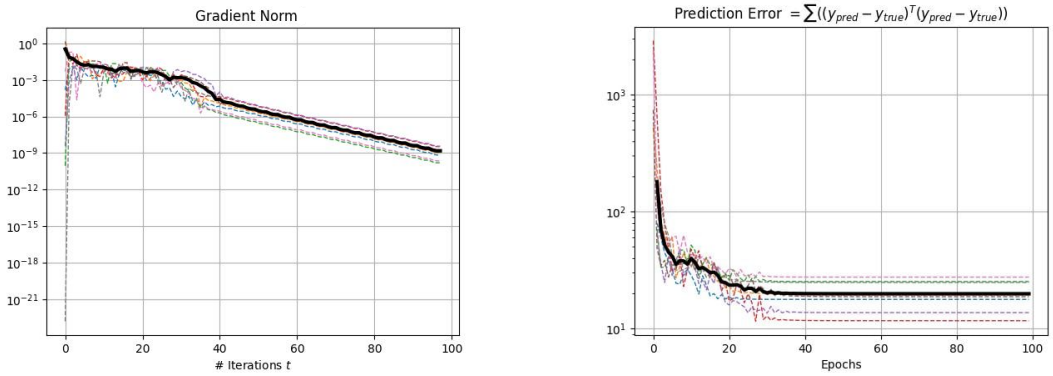

Figure 7: Results with a diminishing stepsize - the bold line is the average value

This plots are obtained training the network over an unbalanced training set. This means that, the probability to catch the wanted digit is far below the probability to catch the other ones. The prediction error converges to a value different from zero. Indeed, since the dataset is unbalanced, the network can't extract the information from the samples to correctly identify the digit. Anyway, the rate of convergence of the gradient is linear (in logarithmic scale).
To obtain better results we pre-processed the training dataset to balance it appropriately. With the same conditions, we obtained:

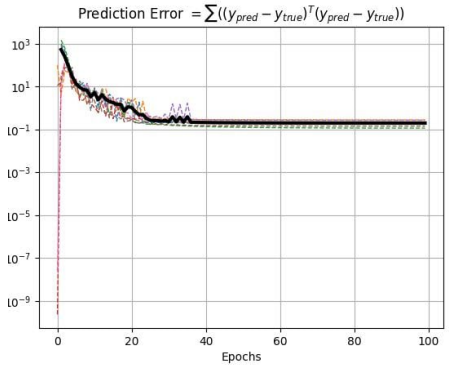

Figure 8: Prediction error for balanced dataset

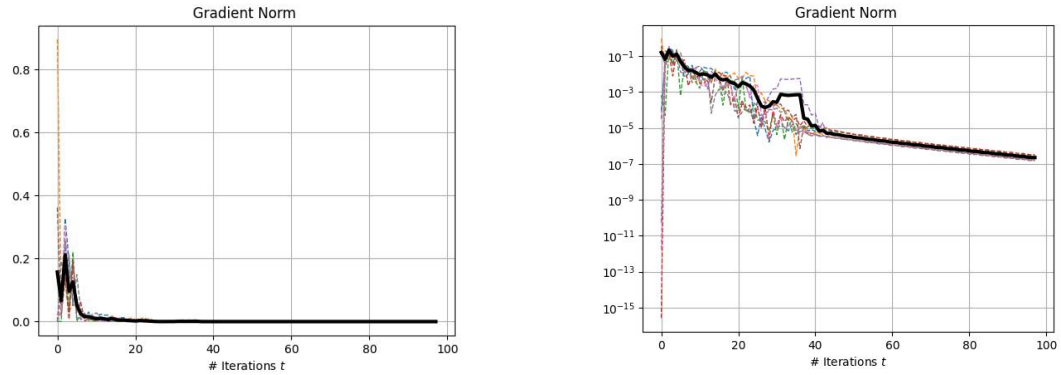

Figure 9: Gradient norm in linear scale (left) and logarithmic scale (right)

In this way, we clearly see that the cost function goes asymptotically to zero. The network is able to track the gradient which shows a linear rate in logarithmic scale.
Furthermore, we can also appreciate a good final accuracy computed on the test set:

```
############################    TEST SET ACCURACY    ############################

     Agent0  Agent1  Agent2  Agent3  Agent4  Agent5  Agent6  Agent7
     89.72%  89.72%  89.72%  89.72%  89.72%  89.72%  89.72%  89.72%
```

Figure 10: Accuracy of the network

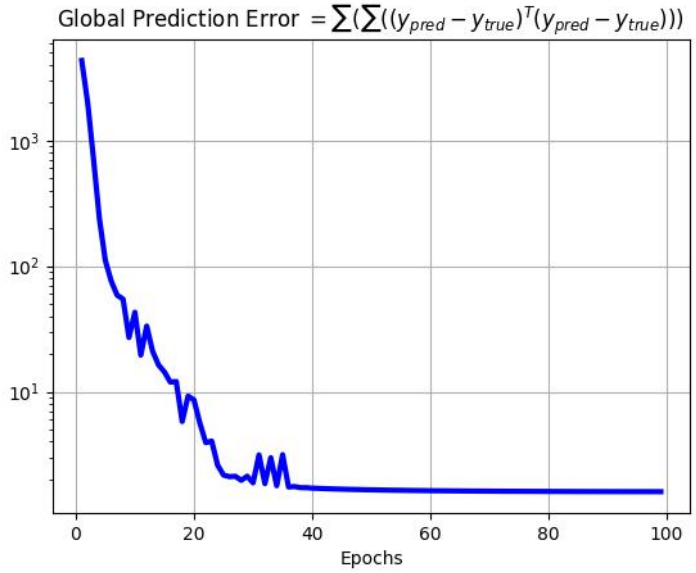

Figure 11: Global Prediction Error

Further Improvements

In order to improve the learning dynamics of the network, we implemented an **ADAM algorithm** [2] to compute an optimal learning rate for each epoch.
ADAM algortihm combicombines two optimization approaches of the descent:

- *Momentum*: algorithm used to accelerate the gradient descent algorithm by taking into consideration the 'exponentially weighted average' of the gradients. Using averages makes the algorithm converge towards the minima in a faster pace.

$$g(t+1) = \beta_1 g(t) + (1-\beta_1)\nabla J(u)$$

- *RMSprop*: is an adaptive learning algorithm that tries to improve AdaGrad. Instead of taking the cumulative sum of squared gradients like in AdaGrad, it takes the 'exponential moving average'.

$$s(t+1) = \beta_2 s(t) + (1-\beta_2)\nabla J(u) \cdot \nabla J(u)$$

Adam Optimizer inherits the strengths or the positive attributes of the above two methods and builds upon them to give a more optimized gradient descent. The ADAM gradient descent term will then be:

$$-\frac{lr}{\sqrt{s(t)} + \epsilon} \cdot g(t)$$

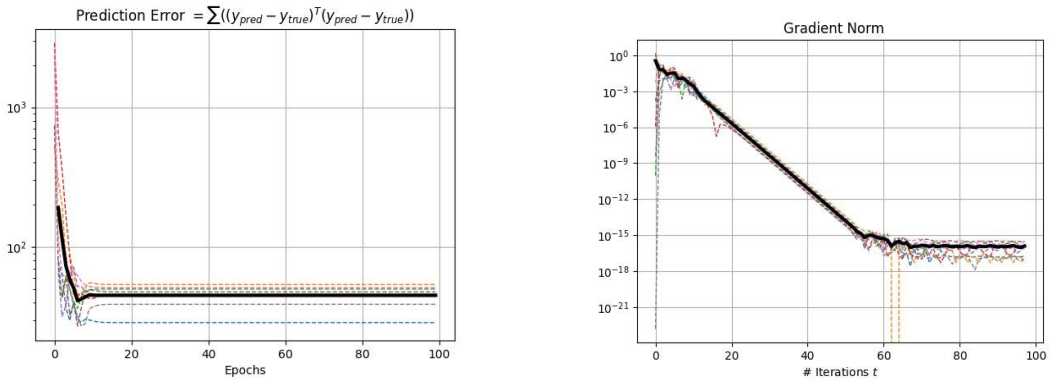

Figure 12: Prediction error (left) and gradient norm (right) with ADAM

A fast convergence to the minimum of the cost function can be seen from the plots above, and the gradient norm presents a steeper slope.

Conclusion

In this task we implemented a distributed classification with the Gradient Tracking algorithm to train a neural network. Extracting a dataset from MNIST, a network of agents learn how to identify correctly a digit through their local neural network. As a result, we noticed that a balanced dataset allow us to steer the cost function close to zero.

Task 2: Formation Control

Introduction

In this task we will use a bearing-based control to reach formation, since it allows to obtain invariance with respect to translation and scaling of a rigid formation. Translational maneuvers refer to when the agents move at a common velocity such that the formation translates as a rigid body. Scaling maneuvers refer to when the formation scale, which is defined as the average distance from the agents to the formation centroid, varies while the geometric pattern of the formation is preserved. Consider a network of N robotic agents modeled by a double-integrator dynamics, where the states con be identified in the position $p_i(t) \in \mathbb{R}^d$ and velocity $v_i(t) \in \mathbb{R}^d$, as follows:

$$x_i(t) = \begin{bmatrix} p_i(t) \\ v_i(t) \end{bmatrix}$$

The agents are divided in two groups, namely leaders n_l and followers n_f so that, the total number of agents is $n = n_l + n_f$. For this task, the motion of each leader is given a priori, and we assumed the velocity of each leader piecewise continuously differentiable [1]. Each follower has dynamics:

$$\dot{p_i}(t) = v_i(t) \qquad\qquad \dot{v_i}(t) = u_i(t)$$

The information flow that tells how the agents will communicate is described by a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} \triangleq \{1, ..., n\}$ is the number n of agents, and $\mathcal{E}$ is the edge set. If $(i, j) \in \mathcal{E}$, we are saying that agent i can exchange informations with agent j. In other words, agent i and agent j are neighbors. The set of neighbors of agent i is denoted as usual with $\mathcal{N}_i$. Now, we need to define a concept that will be fundamental in the following discussion. The **bearing** of agent j relative to agent i is described by the unit vector:

$$g_{ij} \triangleq \frac{p_j - p_i}{\|p_j - p_i\|}$$

We can notice that $g_{ji} = -\, g_{ij}$. For a certain g_{ij}, we can define the **orthogonal projection matrix**:

$$P_{g_{ij}} \triangleq I_d - g_{ij} {g_{ij}}^T$$

It geometrically projects any vector onto the orthogonal compliment of $P_{g_{ij}}$. Notice that $I_d \in \mathbb{R}^{d\times d}$ is the identity matrix. Let's now suppose that the actual bearings configuration at a certain time $t > 0$ is $\{g_{ij}(t)\}_{(i,j)\in\mathcal{E}}$, and the desired constant bearings are $\{g^*_{ij}(t)\}_{(i,j)\in\mathcal{E}}$. We can state that our objective is, according to [1]:
"Consider a formation $\mathcal{G}(p(t))$ where the position and velocity of the leaders are given. We want to design the acceleration control input $u_i(t)$ for each follower i based on the relative position $\{p_i(t) - p_j(t)\}_{j\in\mathcal{N}_i}$ and the relative velocity $\{v_i(t) - v_j(t)\}_{j\in\mathcal{N}_i}$ such that $g_{ij}(t) \to g^*_{ij}(t)$ for all $(i,j) \in \mathcal{E}$ as $t \to \infty$."

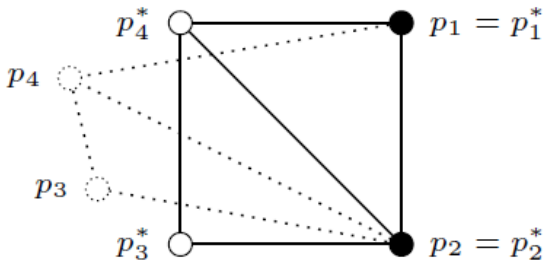

Figure 13: An illustration of a target formation

Another important problem regarding the target position is whether or not $p^*(t)$ exists and is unique. If $p^*(t)$ is not unique, there are multiple (infinite) possible formations satisfying the bearing constraints and leader positions, and as a consequence, the formation may not converge to the desired geometrical pattern.

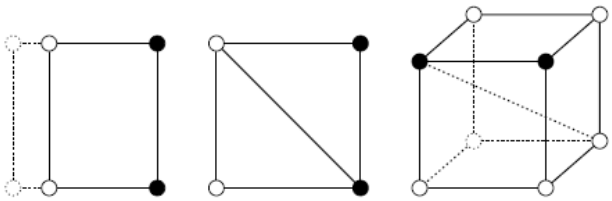

Figure 14: Examples of unique and non-unique target formations

To analyze and solve this problem, let's define the **Bearing Laplacian matrix** $\mathcal{B}(\mathcal{G}()p^*)$ which can be decompose in the following submatrices:

$$\mathcal{B}(\mathcal{G}(p^*))_{ij} = \begin{cases} 0_{d\times d} & i \neq j,\ (i,j) \notin \mathcal{E} \\ -P_{g^*_{ij}} & i \neq j,\ (i,j) \in \mathcal{E} \\ \sum_{k\in\mathcal{N}_i} P_{g^*_{ij}} & i = j,\ i \in \mathcal{V} \end{cases}$$

The matrix $\mathcal{B}(\mathcal{G}(p^*))$, that we will call in short $\mathcal{B}$, is our equivalent of the weighted graph Laplacian matrix. It gives us both interconnection topology and the bearings of the formation. To distinguish between followers and leaders, we can partition the matrix $\mathcal{B}$ so that:

$$\mathcal{B} = \begin{bmatrix} \mathcal{B}_{ff} & \mathcal{B}_{fl} \\ \mathcal{B}_{lf} & \mathcal{B}_{ll} \end{bmatrix} \tag{2}$$

Coming back to the concept of uniqueness of a target formation, we can notice that, given feasible bearing constraints and leader positions, the target formation is unique if and only if $\mathcal{B}_{ff}$ is non-singular. When $\mathcal{B}_{ff}$ is non-singular, the position and velocity of the followers in the target formation are uniquely determined [1]:

$$p^*_f(t) = -\mathcal{B}_{ff}^{-1}\mathcal{B}_{fl}p_l(t), \qquad v^*_f(t) = -\mathcal{B}_{ff}^{-1}\mathcal{B}_{fl}v_l(t)$$

As stated in [1], another useful necessary condition for uniqueness of the the target formation is that there exists at least two leaders. In our implementation we will rely on this assumption.

Formation maneuvering with constant leader velocity

In this section we implement a possible solution to steer the followers to track the maneuvering target formation. In particular, for this first task, the leaders will be considered with constant (zero) velocity. Being the leaders stationary, (2) has this particular configuration:

$$\mathcal{B} = \begin{bmatrix} \mathcal{B}_{ff} & \mathcal{B}_{fl} \\ 0 & 0 \end{bmatrix}$$

Notice that the submatrix $\mathcal{B}_{ff} \in \mathbb{R}^{dn_f \times dn_f}$ is, as proved in [1], symmetric and positive semi-definite. The goal is to guarantee that $g_{ij}(t) \to g^*_{ij}(t)$ for all $(i,j) \in \mathcal{E}$ as $t \to \infty$. To achieve this result, a bearing-based control law is proposed for follower $i \in \mathcal{V}_f$, where $\mathcal{V}_f$ is the set of followers. The control law reads:

$$u_i = -\sum_{j \in \mathcal{N}_i} P_{g^*_{ij}}[k_p(p_i - p_j) + k_v(v_i - v_j)] \tag{3}$$

where $P_{g^*_{ij}}$ is a constant orthogonal projection matrix. k_p and k_v are positive constant control gains, and the neighbours $j \in \mathcal{N}_i$ may be a follower or a leader.

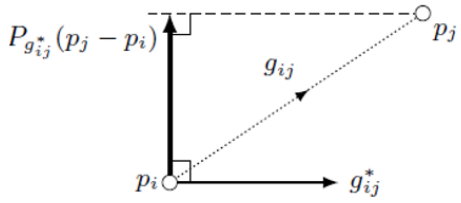

Figure 15: The geometric meaning of the $P_{g^*_{ij}}(p_j - p_i)$ in the control law

Figure 15 helps to explain the geometric meaning of the control law. As we can see, the term $P_{g^*_{ij}}(p_j - p_i)$ steers agent i toward a position where g_{ij} is aligned with g^*_{ij}. As stated in **Theorem 3** in [1]:
"under control law (4), when the leader velocity $v_l(t)$ is constant, the tracking errors:

$$\delta_p(t) = p_f(t) - p^*_f(t) \qquad \delta_v(t) = v_f(t) - v^*_f(t)$$

globally and exponentially converge to zero."

Implementation

The proposed solution has been implemented using **ROS2** with code written in **Python**. For the formation maneuvering with constant leader velocity task, we created two *launch files* denoted as formation.launch.py and multi_formation.launch.py. In launcher files, the following parameters of the network are provided:

- Maximum number of iterations for the simulation;
- communication time period for the exchange of messages between agents;
- dimension of vector state ($p(t)$ and $v(t)$);
- space dimension (2-D);
- control parameters (k_p and k_v);
- number of agents, leaders and followers;
- adjacency matrix;
- initial state;
- desired pattern position.

Several pattern formations are provided as an example: a square, an octagon, letter 'D' and letter 'A'.
The launcher initialize the network, generating N nodes (one for each agent) with random initial positions. In particular, for this task we generated leaders in the exact desired position so that their dynamics are zero. For the followers, the initial velocity is assumed zero. Each agent will run its own agent_i.py script that will effectively implement the bearing-based control algorithm.
Each agent updates, with data received from neighbours, its state with control law previously described (u_i). After that, we discretized the dynamics via *Forward Euler method*:

$$x_{t+1} = x_t + dt \cdot \dot{x}$$

Then, the agents communicate their updated position and velocity to each neighbour and reversely they wait to receive communication data from them.

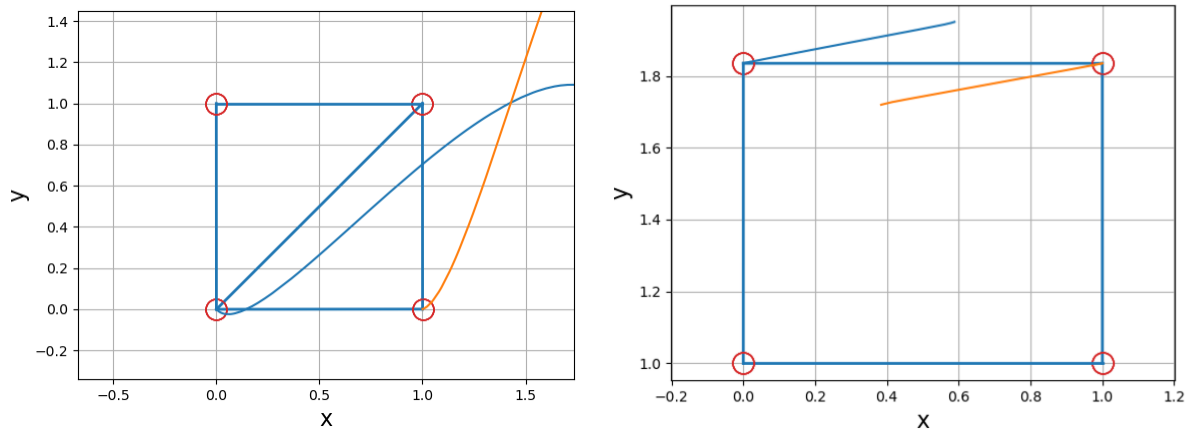

Figure 16: The image shows the difference between a unique (left) and non-unique (right) formation

From Figure 16 we can notice that the initial positions of the followers are random decided while the leader positions are fixed, as said. Indeed, in the picture, the trajectories of the followers, from the random position to the desired one, are highlighted with a colored continuous line. Another remark can be done about the uniqueness of the formation. In particular, as shown in Figure 16, with bold blue lines we are underlining the connections between neighbours. In the non-unique formation the lack of information between agents 1 and 3 make the agents converge to a wrong final formation. This is due to the fact that the bearing unit vector g^*_{ij} are correctly aligned but one constraint (the diagonal direction) is missing.

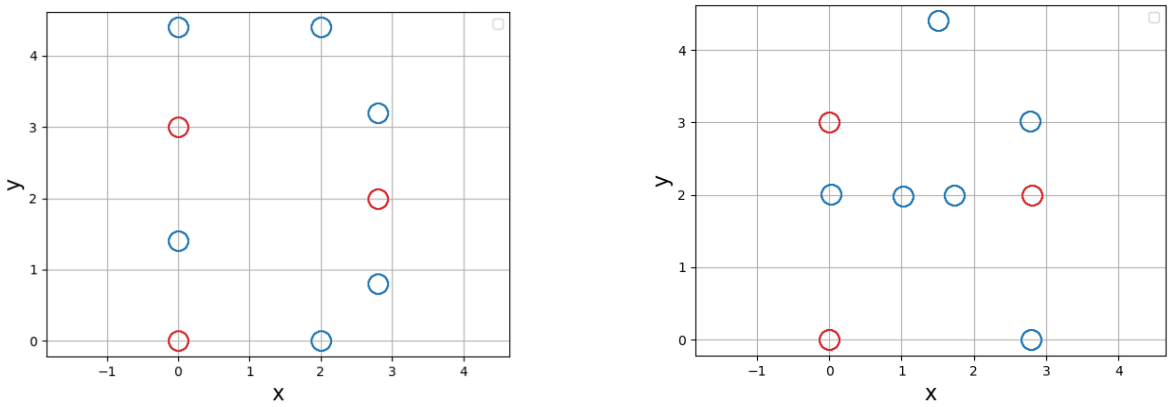

Figure 17: Letter 'D' and 'A' in sequential formations

We can notice in red the leaders, while in blue the followers. The leaders start from the desired positions and the followers, starting from randomly picked positions, reach the desired formation. Firstly to the network is asked to reach the pattern 'D', while, after half of the iterations, the desired pattern is changed into an 'A'.
As we can see, the position error is going to zero in both the situations. It is clear that, when the new configuration starts, all the follower errors drastically increase since the new target position is changed.

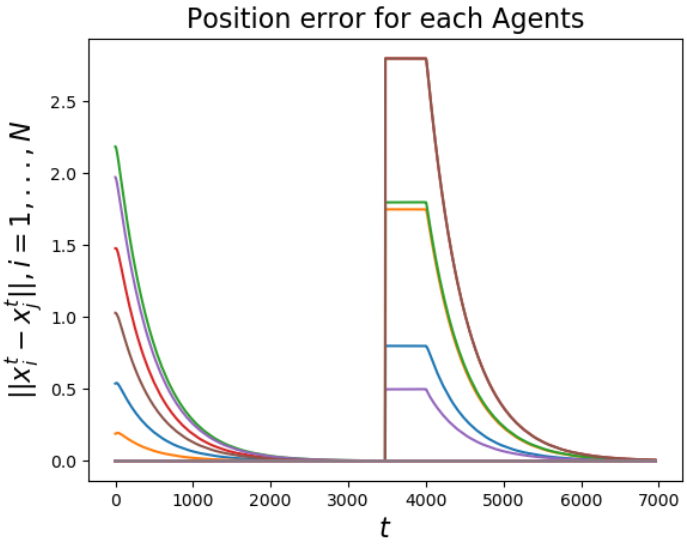

Figure 18: Position error of each agent

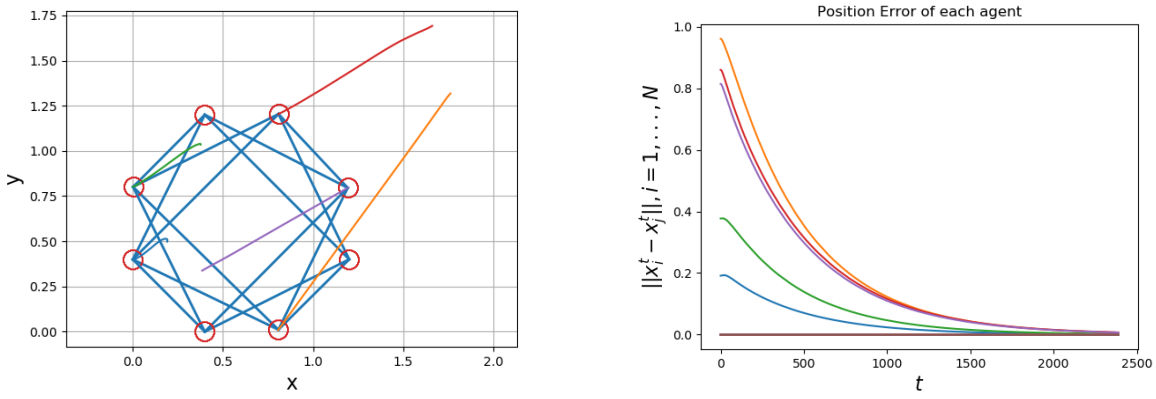

Figure 19: An octagon formation with the relative position error

Formation maneuvering with Time-Varying leader velocity

In order to perfectly track target formations with time-varying leader velocities, an additional feedback acceleration is needed. Assuming $\dot{v}_l(t)$ is piecewise continuous, we modified the control law in the following way:

$$u_i = -K_i^{-1} \sum_{j \in \mathcal{N}_i} P_{g_{ij}^*} [k_p(p_i - p_j) + k_v(v_i - v_j) - \dot{v}_j] \tag{4}$$

where $K_i = \sum_{j \in \mathcal{N}_i} P_{g_{ij}^*}$. For this task, the leaders have to follow a predefined trajectory, so that, their dynamics is non null. We implemented a position control to steer them to the desired positions. Our designed control reads:

$$u_{l,i} = K_p'(p_l^* - p_l) + K_v'(v_l^* - v_l)$$

Implementation

We implemented the launch file with the same structure of the previous case. The main difference is that we augmented the vector state dimension to 3 since we needed to pass information about accelerations too. We initialized the velocities and accelerations to zero, while the positions are randomly selected for both leaders and followers.The agent_i program implements the correct control input law, as in (4).

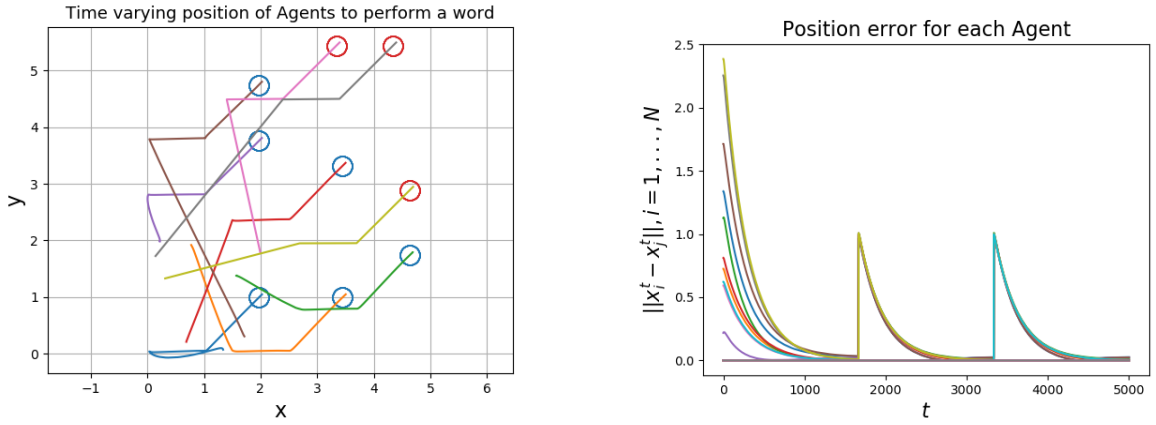

Figure 20: An 'S' formation with its relative position error

We can notice that, as said, this time the initialization is randomly selected for all the agents. In Figure 20 the trajectories of all the agents are plotted to show the translation performed in time.

For a better visualization and simulation, we used **Rviz visualizer**. Here we show some results.

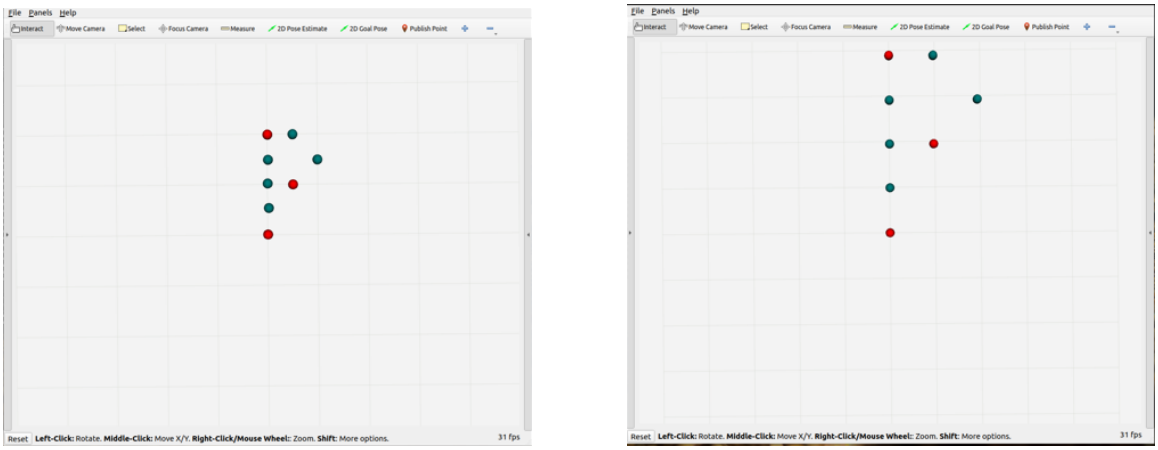

Figure 21: A 'P' that is scaled using the Rviz visualization

As usual, in red the leaders, while the others are the follower. We can notice that the formation is rigidly maintained scaling the letter.

Figure 22: A 'D-A-S' formation using the Rviz visualization

In this formation is really visible that the leaders are changing their position during the evolution. A really good result is achieved.

Conclusion

This task goal was to achieve a formation control, for a network of N robotic agents modeled by a double-integrator dynamics, applying a bearing-based approach. This method allow us to deal with translation of the formation, both with constant and time-varying velocity of the leaders.We proposed a variety of patterns with different number of agents and two different control laws that reaches global formation stability.

References

[1] S. Zhao and D. Zelazo, "Translational and scaling formation maneuver control via a bearing-based approach", IEEE Transactions on Control of Network Systems, vol. 4, no. 3, pp. 429438, 2015.

[2] Kingma and Ba, "Adam: A method for stochastic optimization", ICLR 2015