Modeling and Control of Cyber-Physical Systems

Politecnico di Torino

Graziosi Luigi – *s331564*
Luppino Marco – *s333997*
Paglialunga Federico – *s328876*

Politecnico di Torino
A.A. 2023/2024

Summary

Project Part 2

Introduction

In our project, we focus on the **distributed control of a multi-agent magnetic levitation system**, a classic example of a cyber-physical system (CPS). The system consists of seven magnetic levitation units: one of these acts as the leader node $\boldsymbol{S_0}$, while the other six operate as follower nodes $\boldsymbol{S_i}$, $i = 1, 2, \dots, 6$. Each individual agent in the system, including both the leader and the followers, is described by identical state-space equations, obtained by linearizing the physical equations around a suitable equilibrium point. These equations are expressed as

$$\dot{x}_i = Ax_i + Bu_i \text{ and } y_i = Cx_i$$

where the matrices A, B, C are defined respectively as:

$$A = \begin{bmatrix} 0 & 1 \\ 880.87 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 \\ -9.9453 \end{bmatrix}, C = [708.27], D = [0]$$

Our task is to design a distributed control protocol that enables the multi-agent system to perform cooperative tracking tasks, asymptotically reducing the global disagreement error to zero. We use a communication network for the system described as a graph G augmented where the leader node is not considered. A way to represent graphs is through adjacency matrices, where each entry represents the weight of the edge (v_i, v_j), indicating that i receives information from j.

We designed two types of dynamic regulators since the state variables of the agents are not directly measurable due to $C \neq I$: one based on a **local observer** and the other on a **neighborhood observer**.

In the case of the **local observer**, each agent estimates the state using its local information and the output from other agents. This approach is computationally less complex but may be limited in terms of global performance, especially in the presence of noise or disturbances.

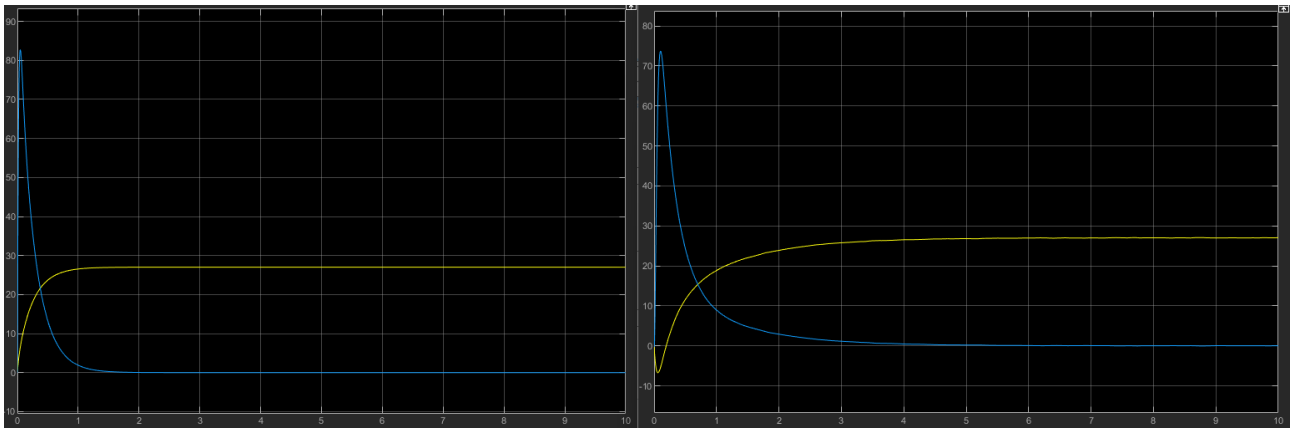
On the other hand, the **neighborhood observer** involves agents exchanging information with all their neighbours, creating a more densely connected network. This method can potentially enhance the system's overall performance but requires increased computational and communication complexity.

Regardless of the chosen approach, we need to conduct a numerical analysis to evaluate the effects of different leader node reference behaviors, such as constant signals, ramps, and sinusoidal signals, on the system. This involves modifying the dynamics of the leader by adjusting the eigenvalues of matrix A using the pole placement technique:

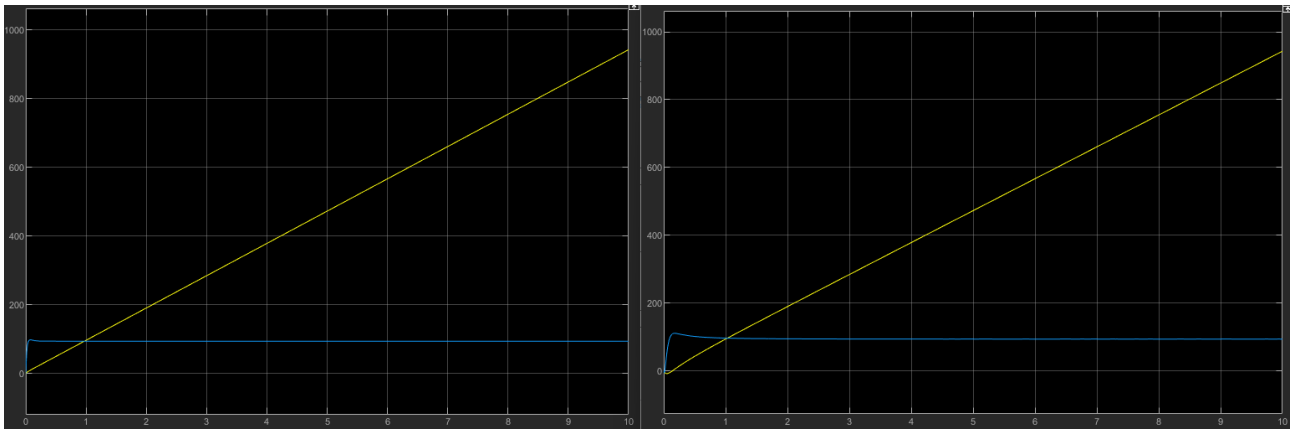
- For a **step** reference, we set one eigenvalue to zero and the remaining eigenvalues to have negative real parts.
- For a **ramp** reference, two coincident eigenvalues are needed with the remaining eigenvalues stable.
- For a **sinusoidal** reference, manipulation of the magnitude and phase of complex conjugate eigenvalues is necessary.

Plot of three kinds of references

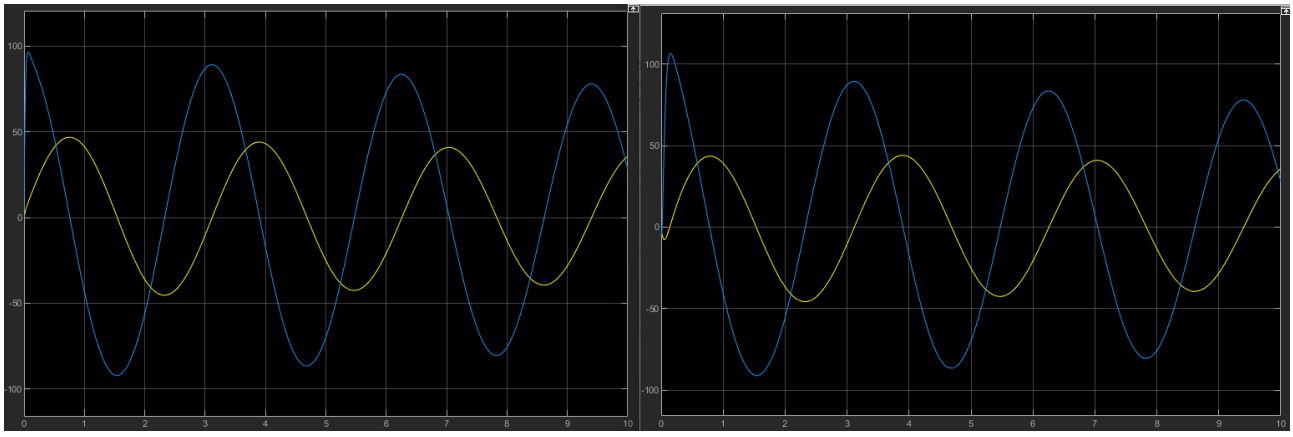
The plots of the two simulations are quite similar. As follows there will be the plots of the first case of study.



Pic. 12 – Step



Pic. 13 – Ramp



Pic. 14 – Sinusoidal signal

Preliminary considerations

We will consider in our discussion the trajectory of a ramp dictated by the local controller on the leader obtained by the eigenvalues $[0, -4]$. On the followers it may be applied (by correctly setting the manual switch) some White Gaussian noise with variance handled by the variable `dis`. All the follower nodes have the same initial conditions randomly generated by `randn(2,1)`For our discussion, we have saved a particular value of the initial conditions in order to observe the differences depending on the topology, noise, parameters and on the two cases of study in the same initial conditions.

Choice on topologies

As far as concerning the communication networks, we tried three possible configurations:

- **A)** “Snake” configuration
- **B)** Full Mesh configuration
- **C)** “Ring” configuration

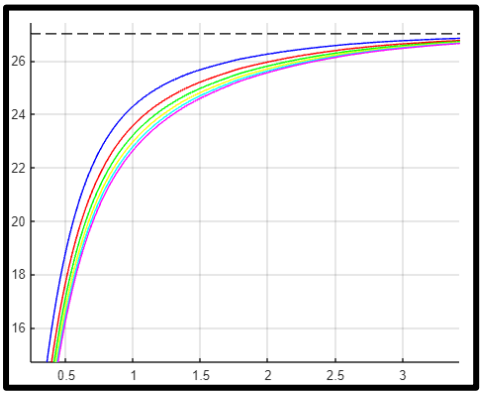
The following considerations are obtained at fixed parameters, noise and initial conditions. In particular, we will consider that the noise affects all the follower nodes with variance equal to 1.

Which one better fits our goals? Shall we analyse the three options.

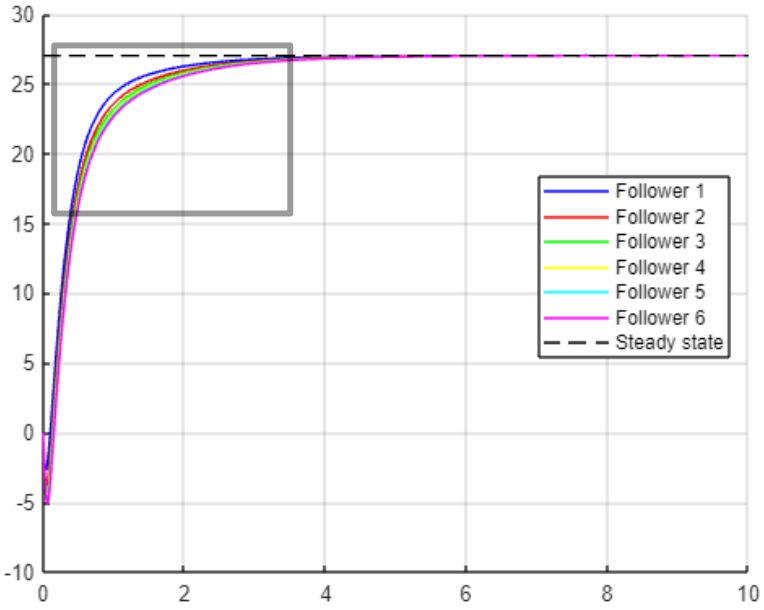
(A) In the first configuration we have that every node receives only the information from the previous node, so the convergence time is increasing with the number of the node we are going to consider. Actually, first node to converge to leader trajectory will be the node 1, while the last will be the node 6.

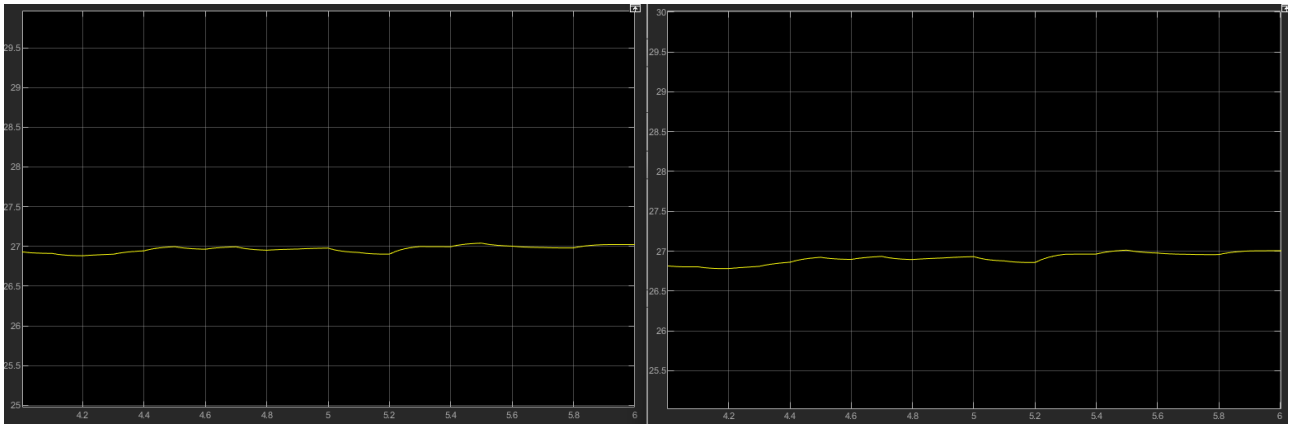
This topology is the most affected by the effect of the noise. By increasing the variance, we can notice an increment in convergence time. Moreover, we can observe an increasing tracking error from the first follower node to the last, even if it is not additive but the noise on one follower influences only to a small extent the state tracking.

Pic. 1 – Snake topology



Pic. 2 – Plot of follower states

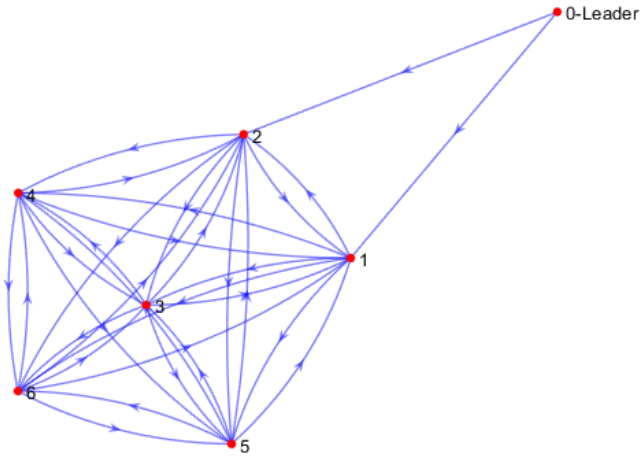




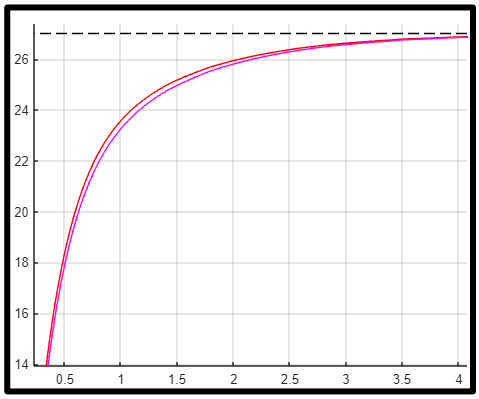
Pic. 3 – Follower 1 vs Follower 6 noise effect on position state and white noise with variance 100

(B) In the second configuration we have that all the follower nodes connected each other with the same weight, while the leader is connected only to nodes 1 and 2. The six followers converge together to the leader trajectory, because they all have a global information of the state estimation of the other nodes. This is not good, in general, in case of adversarial attacks on one node and in energy consumption. However, it allows much more resilience to noise and a lower tracking error. In the picture (6) we can notice that all the nodes have the same behaviour and that the error due to the noise is the same.

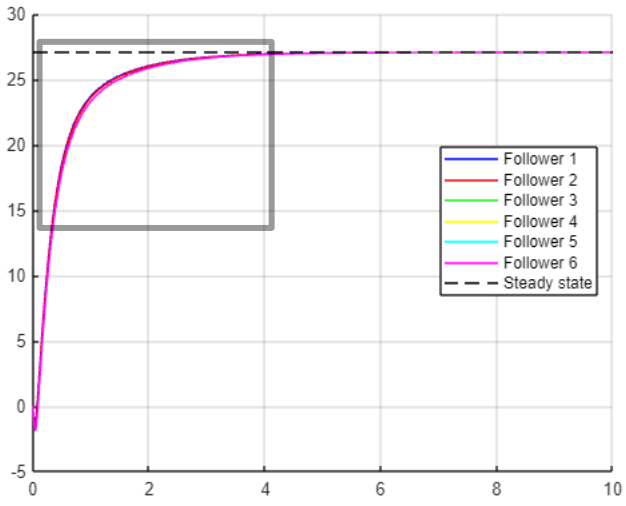
On the plot, we can distinguish two different lines, the first comprehending the states of nodes 1 and 2, directly communicating with the leader and so having a faster convergence, the second comprehending all the other nodes, that converge a bit later.

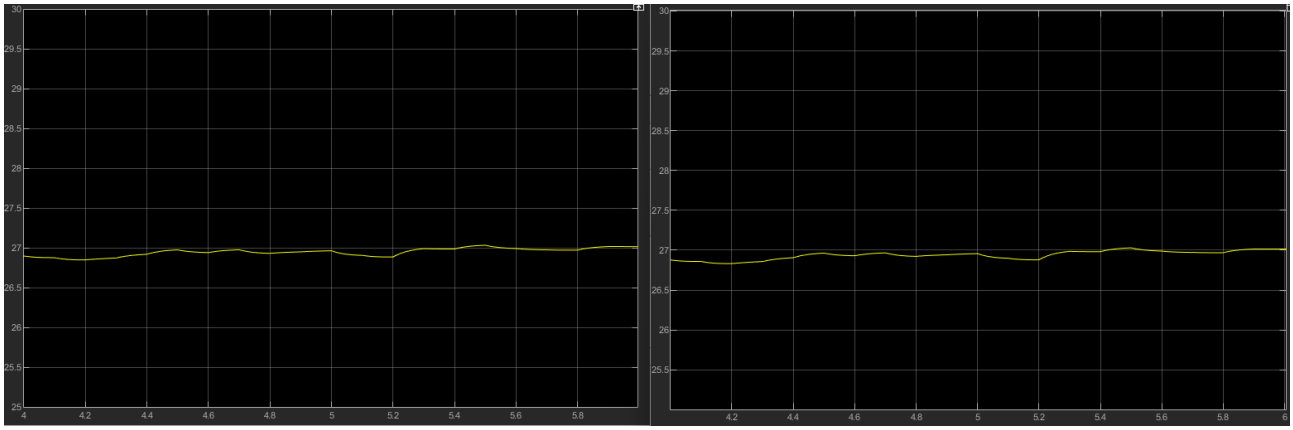


Pic. 4 – Full mesh topology



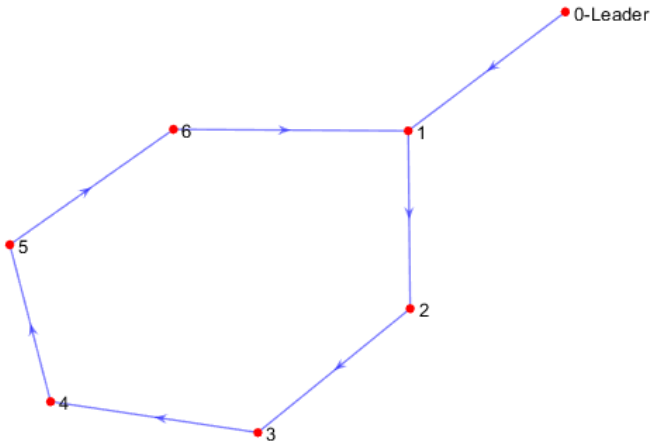
Pic. 5 – Plot of follower states



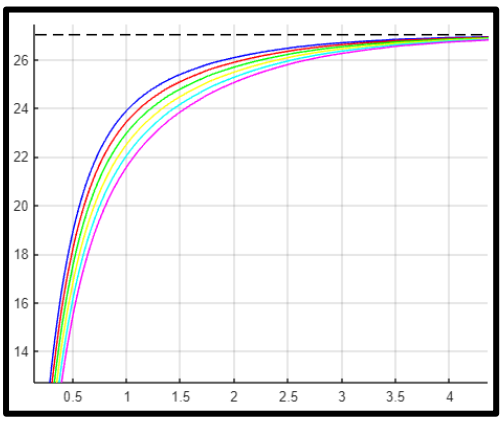


Pic. 6 - Follower 1 vs Follower 6 noise effect on position state and white noise with variance 100

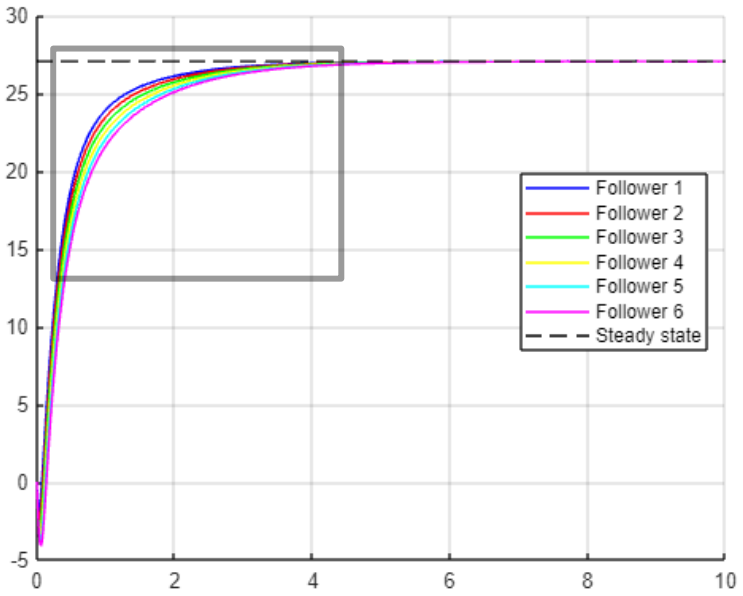
(C) In the third configuration we have “Ring” topology in which each node sends information to its next and the last sends to the first node. The 6 agents converge following in order from the first to the last node, just like first configuration, but they need a lower time of convergence, because of the communication from the agent 6 and the agent 1. We cannot notice relevant differences between the behaviour of the noise on configuration “A” and on configuration “C”.

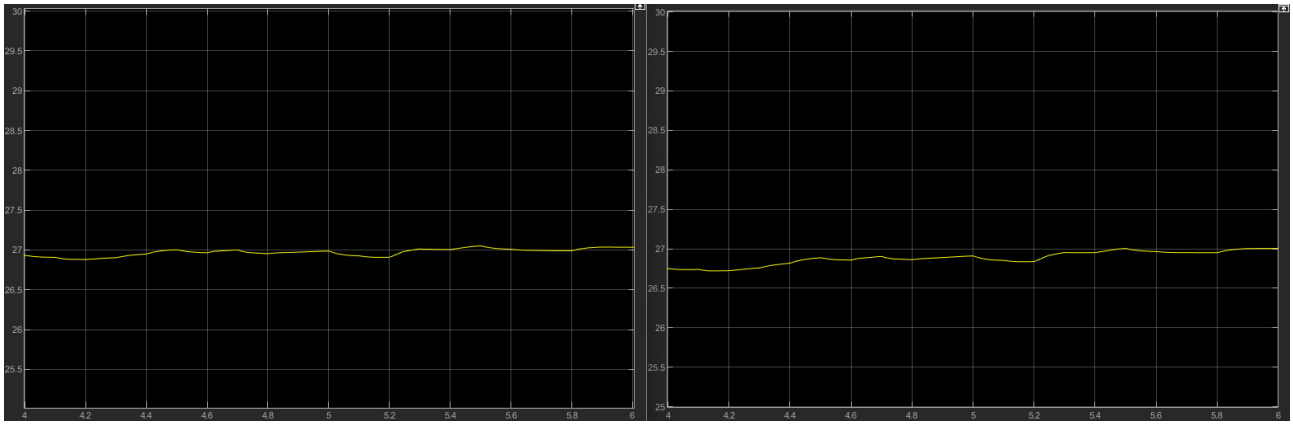


Pic. 7 – Ring topology



Pic. 8 – Plot of follower states





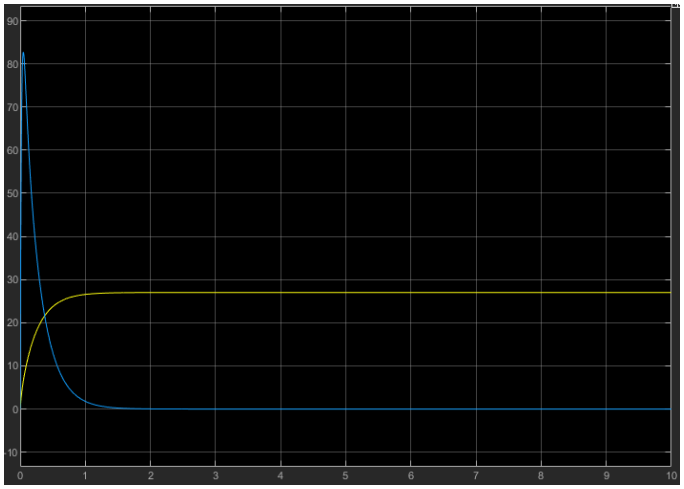
Pic. 9 - Follower 1 vs Follower 6 noise effect on position state and white noise with variance 100

The topology we chosen and on which we will base our discussion in the next paragraphs is the second because it allows a faster convergence and a lower tracking error of the follower nodes with respect to the leader. We wrote the function `graph_plot.m` to plot the augmented digraphs.

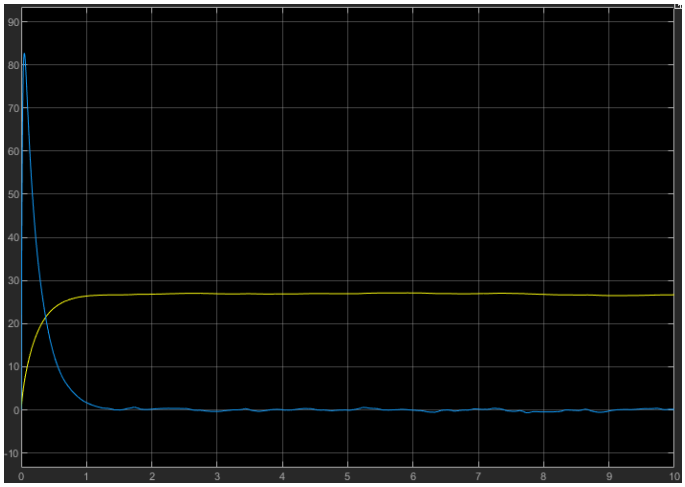
Preliminary Observations on Noise

By adding noise also to the leader node output, we can notice that the behaviour is not correct anymore, because the trajectory diverges after some time (it does not follow the defined state evolution).

As follows we can notice how also a white noise with a little variance can create unwanted oscillations of the position state.

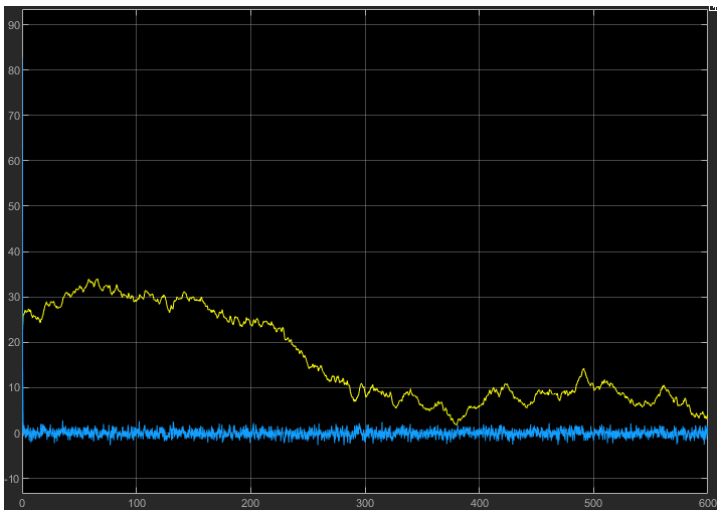


Pic. 10a - State variables without noise on leader



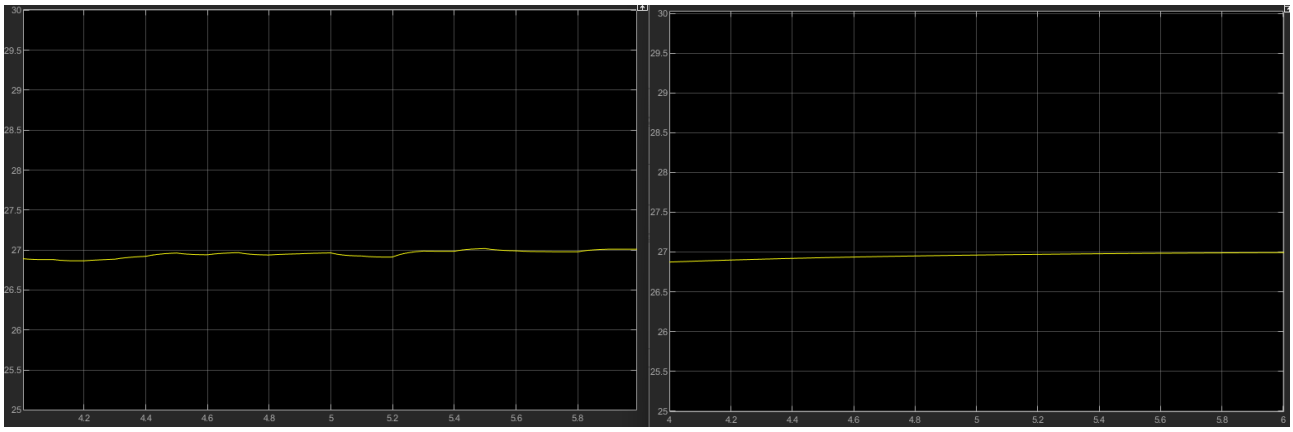
Pic. 10b - State variables with little noise on leader

By increasing the variance on leader noise, we will observe instability:



Pic. 10c - State variables with much noise on leader

In our discussion we will consider that the leader node is not affected by any noise (the noise block is switched on the constant 0).
The addition of noise on the followers' outputs causes more instability and a higher time of convergence. We can notice that by adding noise only to some follower agents the state of noise-free agents is not modified, they can perfectly follow the trajectory, while agents affected by noise continue following the trajectory in an inaccurate way. As follows we can notice the different of the position tracking of an agent affected by noise with respect to an agent noise-free:



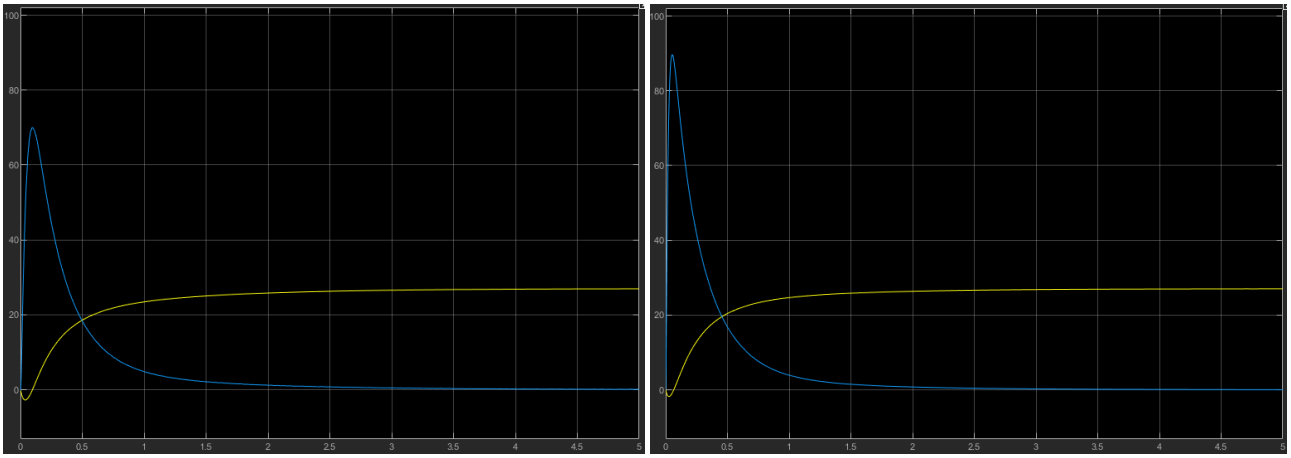
Pic. 11 – Follower 1 affected by noise vs Follower 2 noise-free

Preliminary considerations on the choice of c, Q, R

Regarding the parameters' choice, we considered the coupling gain c and the matrices Q and R used in the optimization problem to compute the observer and controller gain. Each parameter was modified individually while keeping the others constant.

Analysis on c

We define $c_{min} = \frac{1}{2\min(Re\{\lambda_i\})}$, where λ_i are the eigenvalues of $L + G$. We consider $c = c_{min} + b$ (we start from $b = 0.1$ because $c > c_{min}$), we are going to increase the b and to study the behaviour of the system.
By increasing the coupling gain the follower systems have a smaller transient, i.e., it converges more rapidly (settling time is lower) and the energy consumption on the output decreases.
On the other hand, we can notice that increasing the value of c requires more energy consumption on the input and the error on the output increases.

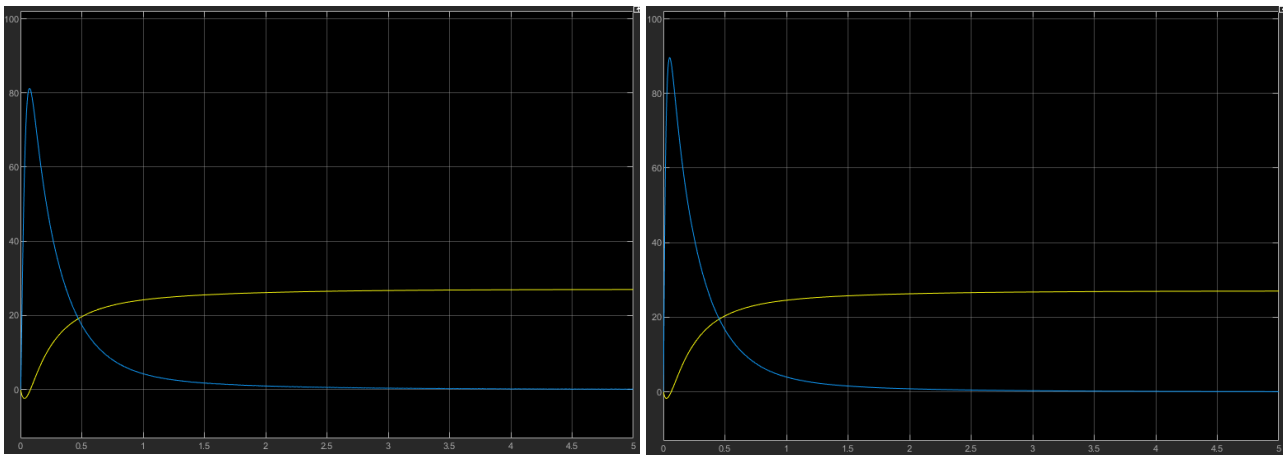


Pic. 12 – Followers' states with $c = 0.1$ vs $c = 10$

We have an upper bound for the value of c, imposed by the velocity of the state evolution of the leader, after which the growth of performance stops. Actually, we can notice that after a certain value of c the energy consumption on the outputs doesn't decrease anymore.

Analysis on Q

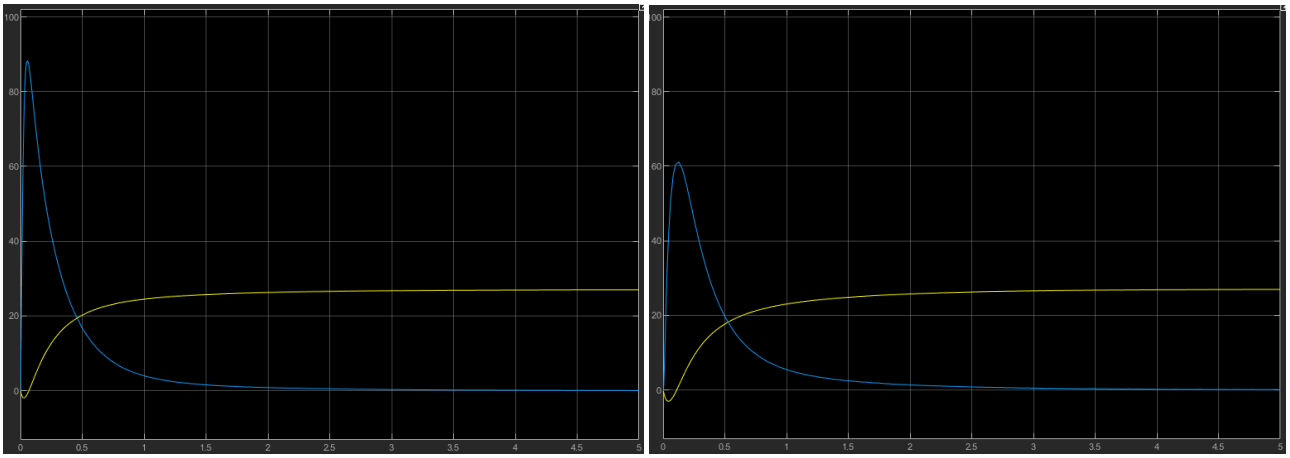
Q is a matrix that allows us to handle the energy consumption on the output. By increasing Q we decrease the output energy consumption and the convergence time, but it causes an increase in the output error estimation and on the input energy consumption, because we are giving more importance on the output.



Pic. 13 – Followers' states with $Q = 2I$ vs $Q = 200I$

Analysis on R

R is a matrix that allows us to handle the energy consumption on the input. By increasing R we decrease the input energy consumption, but it causes an increase in convergence time and in output consumption, because we are giving more importance on the input. We had not observed any pattern on the output error.



Pic. 14 – Followers' states with $R = 0.1$ vs R= 10

Conclusions

We can observe that the value of c does not significantly influence our system. By drastically increasing or decreasing the value of c, we do not notice substantial differences in the system's behavior. The upper bound would be imposed by the velocity of the leader node, as the follower nodes' velocity saturates when they try to converge faster than the leader's state evolution.

Considering the parameters Q and R, we can see how they similarly influence the system's behavior. For low values of Q and R, the tracking precision decreases and the convergence time increases. Consequently, our system converges more slowly and less precisely.

The values of c, Q and R used for the design are as follows: $c = c_{min} + 1$, $Q = 20I$, $R = 0.1$. These values are the ones on which are done the other discussions.

Case of Study I: Distributed Regulator based on Local Observers

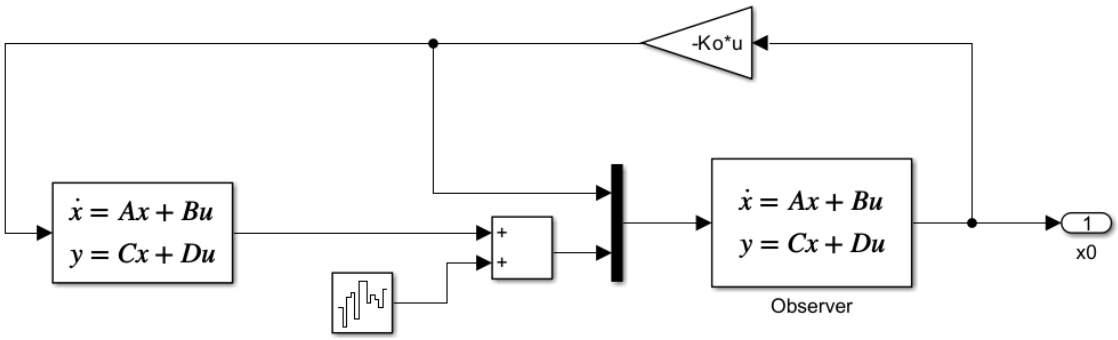
In this case of study, we have 7 nodes, each implementing a local controller and a local observer. The follower nodes' structure is identical.

The local controller in the leader node S_0 is implemented by **feedback**, so that we have a State-Space system with given matrices A, B, C, D which gives the output. The output and the result of the feedback controller are given in input to the observer, which is implemented as a State-Space system which matrices are $A - L_1C$, $[B\ L_1]$, I_2, $\emptyset_{2,2}$, where L_1 is computed on `MATLAB` with the command `L1=place(A',C',eigsL1)'`, where the eigenvalues have $Re(\lambda_i) < 0, \forall i$.

The output of the observer is then multiplied by the matrix $-K_o$, which gives us the feedback input for our system and for the observer itself. The matrix K_0 is computed on `MATLAB` with the command `Ko=acker(A,B,eigs)`.

The choice of the eigenvalues we are going to assign is very important, as it defines the trajectory of the leader node and, consequently, of the follower nodes. In particular, we are going to analyse three kinds of reference trajectories:

- Step
- Ramp
- Sinusoidal



Pic. 15 – Leader system

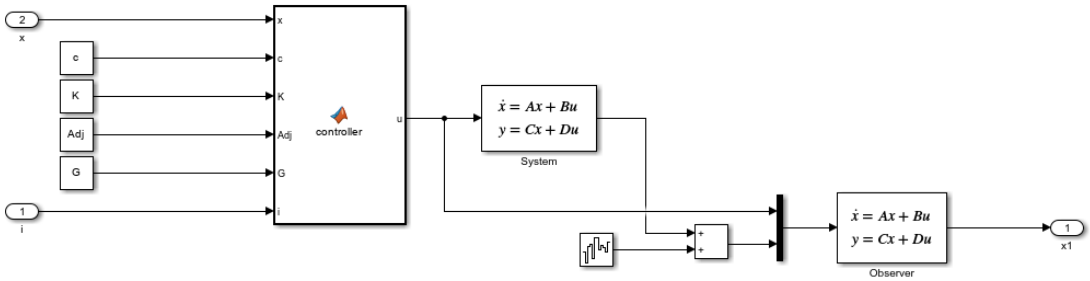
On the other hand, in addition to the controller given by K_0 (they have to be identical to the leader node otherwise they do not work), the follower nodes also implement a distributed controller given by the SVFB control protocol $u_i = cK\varepsilon_i$, where c is the coupling gain defined with respect to the constraint:

$$c > \frac{1}{2\min_{i \in \mathcal{N}} Re(\lambda_i)}$$

and $K = R^{-1}B^{\mathrm{T}}P$, according to the **Cooperative controller design Theorem**. The matrix P is the unique positive definite solution of the **algebraic Riccati equation** (**ARE**) computed on `MATLAB` by the command `P=are(A-B*K, B*R^-1*B', Q)`, where Q and R are diagonal positive definite matrices. The controller block receives as inputs the number of the node i, the **adjacency matrix** Adj, the **coupling gain** c, the **pinning matrix** G and the **feedback state** x. It computes:

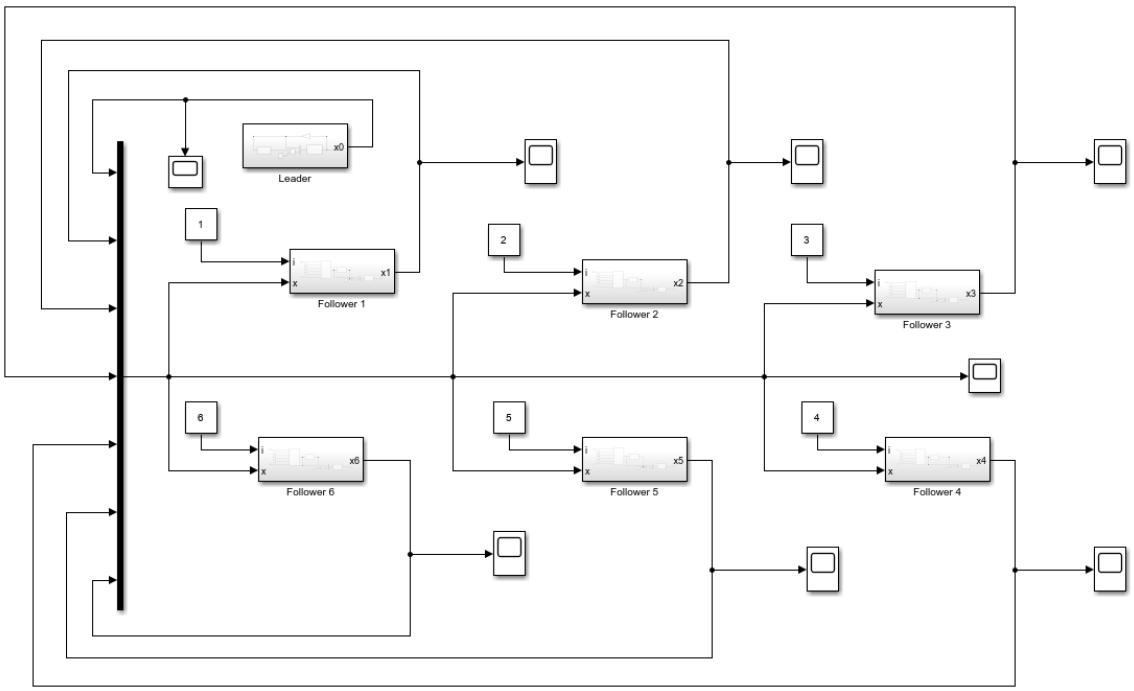
$$\varepsilon_i = \sum_{j=1}^{N} a_{ij}(x_j - x_i) + G_{ii}(x_0 - x_i)$$

where the first part of the equation is obtained on `MATLAB` by implementing a for loop on every node $j = 1, \dots, 6$. This controller is linked in input to the system controlled by the feedback $-K_0$.
The observer receives in input the output of the cooperative controller and output of the system. Note that, for graphical reasons, it has been preferred to implicitly implement feedback controller by inserting into the System State-Space block the matrices $A - BK_0$, B, C, D and into the Observer one the matrices $A - BK_0 - cFC$, $[B\ cF]$, I_2, $\emptyset_{2,2}$, where F is the local observer matrix obtained by computing the `MATLAB` command `F=place((A-B*Ko)',C',eigF)'`, $Re(eigF_i) < 0$.



Pic. 16 – Follower System

All the states $\hat{x}$ estimated by the observers of every node are combined by a multiplexer and feedback received in input by every follower node, because the controller will use it to compute ε_i (as explained before).



Pic. 17 - The entire system

It has been added a **White Gaussian Noise** to every follower output.

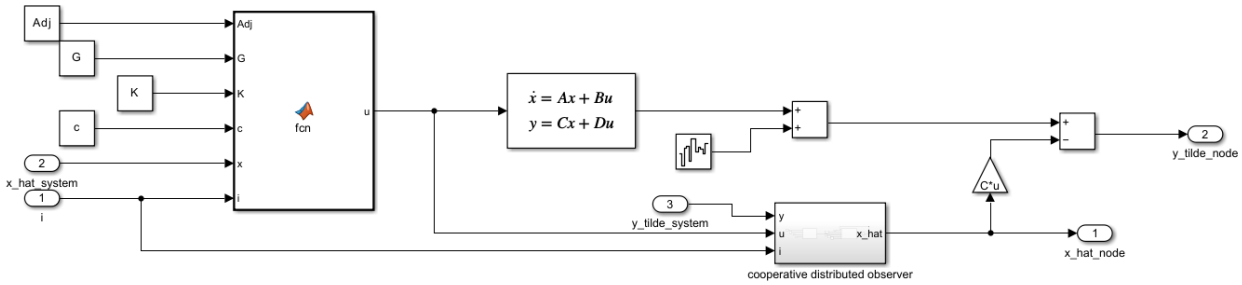
Case of Study II: Distributed Regulator based on a distributed Neighbourhood Observer structure

In the second case, a neighbourhood observer and a local controller is implemented on each follower with the primary goal of estimating the state of the system through the assessment of the neighbourhood.

As in the previous case, the local controller in the leader node is implemented by feedback, placing negative eigenvalues to K_0, the only difference is that in this case we also compute the $\tilde{y}$, that is the difference between the real output of the agent and the estimation provided by the local observer.

The inputs of the agent provided by the neighbourhood observer are $\hat{x}$ and $\tilde{y}$ that represent the state of all the agent present in the system. For each follower, the input goes through the control block and through the observer.

The follower nodes implement a local controller given by the SVFB control protocol $u_i = cK\hat{\varepsilon}_i$. The $\hat{\varepsilon}_i$ is the **local neighbourhood state estimation error**, computed using $\hat{x}$ instead x .



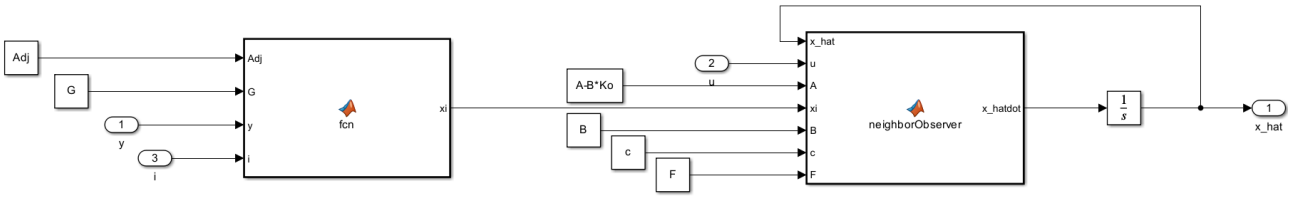
Pic. 18 - Structure of follower

$\hat{x}$ and $\tilde{y}$ are provided by the **cooperative distributed observe**r, having these dynamics:

$$\dot{\hat{x}}_i = A\hat{x}_i + Bu_i - cF\xi_i$$

where the term ξ_i is the ***local neighbourhood output estimation* error** given by:

$$\xi_i = \sum_{j=1}^{N} a_{ij}(\tilde{y}_j - \tilde{y}_i) + G_{ii}(\tilde{y_0} - \tilde{y}_i)$$



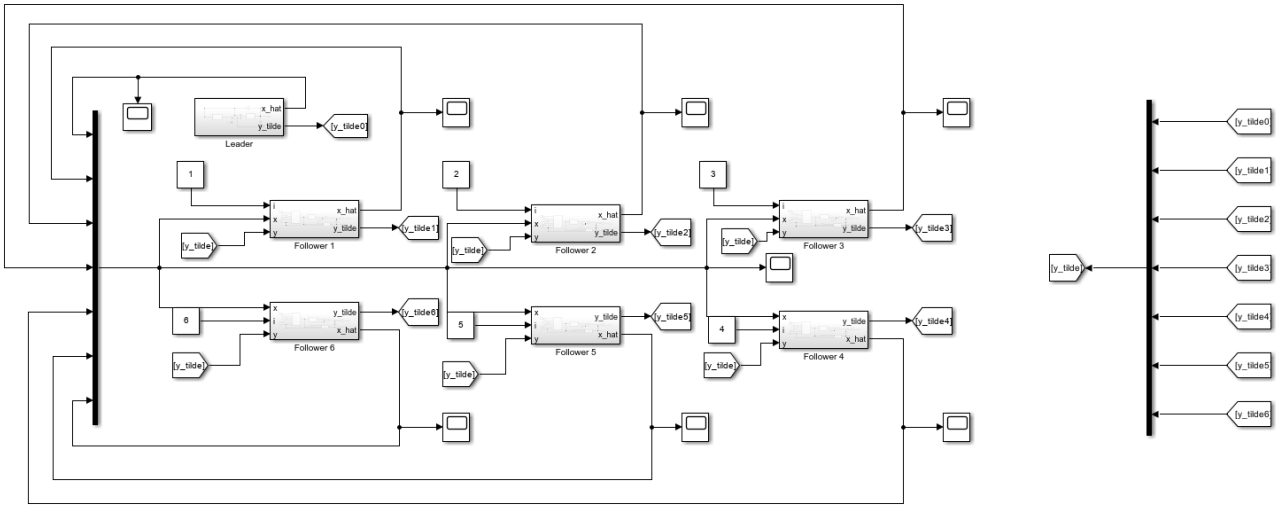
Pic. 19 - Structure of follower observer

The **local closed-dynamics** is given by:

$$\dot{x}_i = Ax_i + cBK\left(\sum_{j=1}^{N} a_{ij}(\hat{x}_j - \hat{x}_i) + g_i(\hat{x_0} - \hat{x}_i)\right)$$

$$\dot{\hat{x}}_i = A\hat{x}_i + Bu_i - cF\left(\sum_{i=1}^{N} a_{ij}(\tilde{y}_j - \tilde{y}_i) + g_i(\tilde{y}_j - \tilde{y}_i)\right)$$

The following photos is the entire system built on Simulink.



Pic. 20 - The entire system

It has been added a **White Gaussian Noise** to every follower output.

Comparison

We will compare the two algorithms in terms of settling time, energy consumption regarding to input and output, RMSE (Root Mean Square Error), noise tolerance.

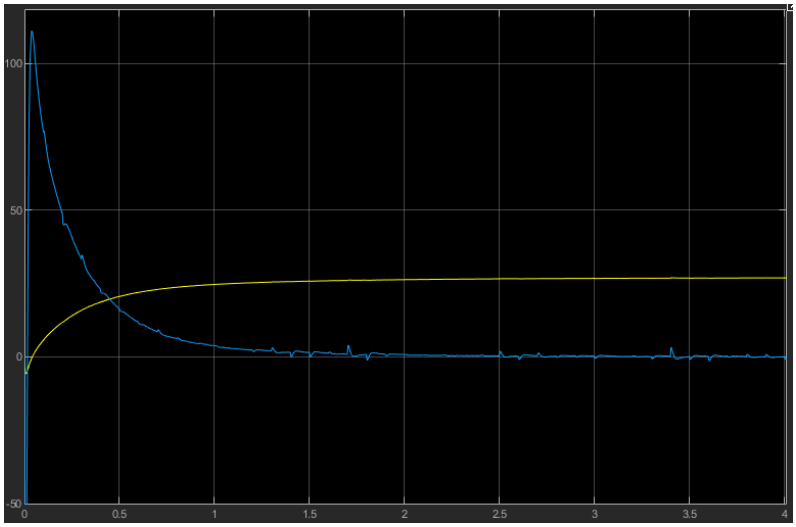
Complexity

First case of study (local observer) has a faster computation than the second case (distributed observer). This difference becomes relevant when we have many follower nodes or if the weights of the edges between the nodes increase significantly.

Time of Convergence and Noise

It is difficult to define a general rule on which kind of observer works better, because depending on the problem one algorithm can result better than the other or they can be the same.

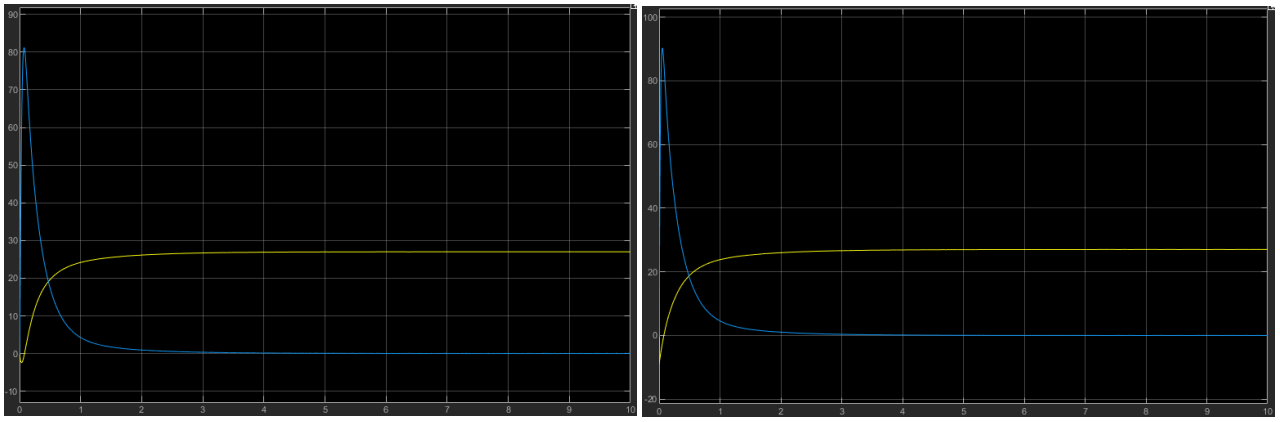
The local observer algorithm allows more flexibility in selecting the dynamics of the observer because we can choose the eigenvalues. Actually, by increasing (in magnitude) the eigenvalues associated with the dynamics of the observer, the stability tends to increase and the time of convergence decreases. By the way, after a threshold value the noise increases again, the velocity can suffer some unwanted oscillations.



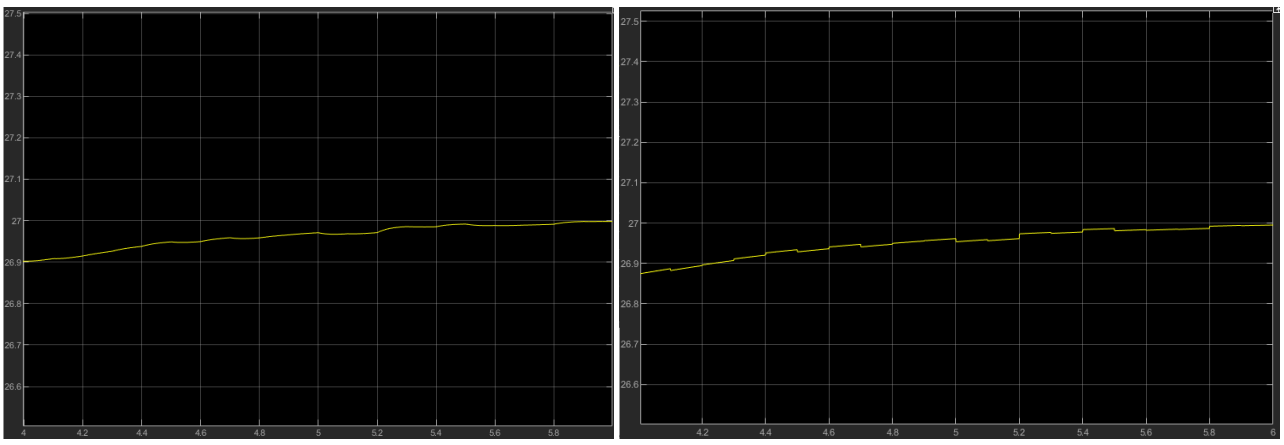
Pic. 21 - Oscillations on velocity

Moreover, by acting on the eigenvalues of the observer of the leader, we are acting also on the trajectory. For instance, if we increase (in magnitude) the eigenvalues of the leader observer, we can notice that the amplitude of the step decreases.

In this situation, starting from the same initial conditions and parameters, we can observe that the two algorithms perform almost in the same way, but the local observer is a bit more resilient to noise.



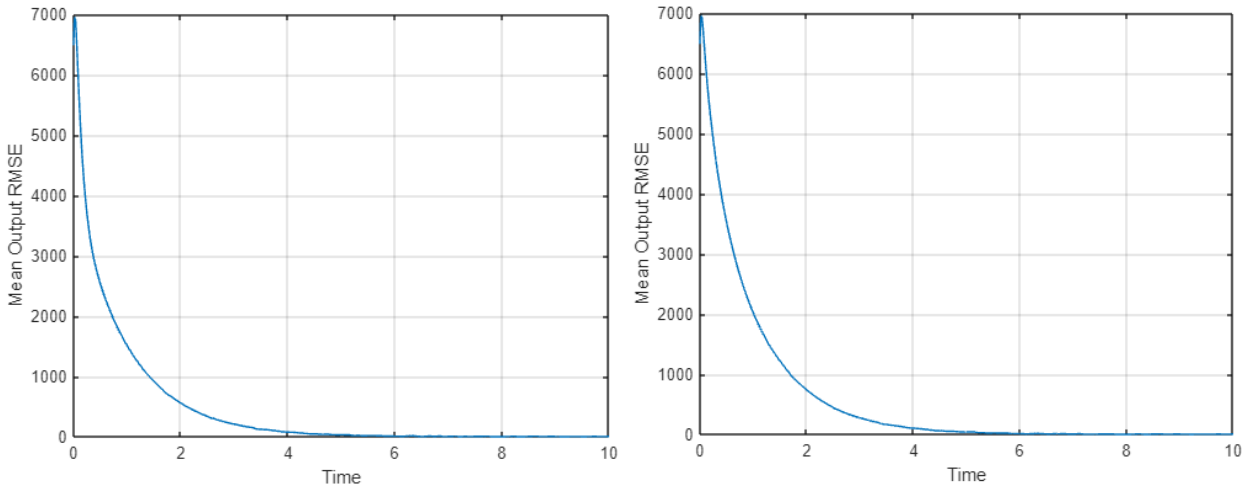
Pic 22 - Follower states in Local vs Distributed Observer



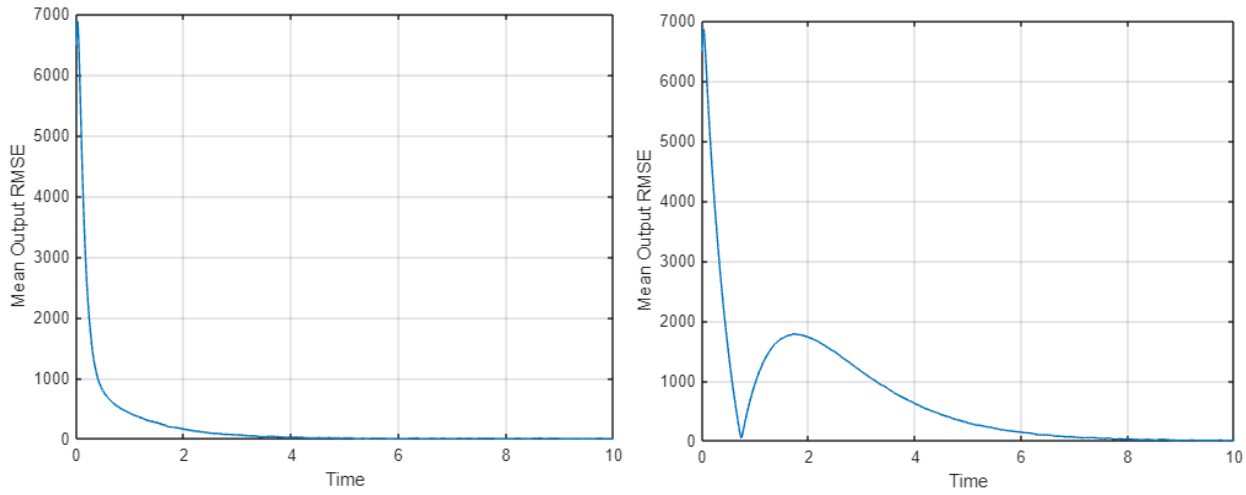
Pic 23 - Moise effect on Local vs Distributed Observer

	RMSE *Step*	*Settling time* $t_{s,1\%}$ *Step on position (s)*	***RMSE*** *Ramp*	*Settling time* $t_{s,1\%}$ *Ramp on velocity (s)*
Local Observer	1868.5	3.1	1603.8	0.7
Distributed Observer	1219.5	3.4	1374.2	3.9

Table A – Comparing Step and Ramp on two algorithms



Pic. 24 - Mean RMSE in local observer vs in distributed observer (Step)



Pic. 25 - Mean RMSE in local observer vs in distributed observer (Ramp)

Comparison on *c*

Note that in all the simulations we have been considering $Q = 20I$, $R = 0.1$, step reference and White Gaussian noise with variance 1 on all the followers.

Local Observer

	RMSE	Settling time $t_{s,1\%}$	$\|u\|_2^2$	$\|y\|_2^2$
$c = c_{min} + 1$	1868.5	3.15	7476.5	2.64e8
$c = c_{min} + 10$	2004.6	3.00	2.14e4	2.54e8
$c = c_{min} + 100$	2143.2	2.95	3.84e4	2.45e8

Table B – Comparison based on c in local observer

Neighbourhood Observer

	RMSE	Settling time $t_{s,1\%}$	$\|u\|_2^2$	$\|y\|_2^2$
$c = c_{min} + 1$	1219.5	3.40	4630.4	3.04e8
$c = c_{min} + 10$	1050.7	3.26	4.75e4	3.12e8
$c = c_{min} + 100$	967.9	3.26	2.69e6	3.16e8

Table C – Comparison based on c in cooperative observer

Comparison on Q

Note that in all the simulations we have been considering $c = c_{min} + 1$, $R = 0.1$, step reference and White Gaussian noise with variance 1 on all the followers.

Local Observer

	RMSE	Settling time $t_{s,1\%}$	$\|u\|_2^2$	$\|y\|_2^2$
$Q = I$	656.9	3.42	223.8	3.37e8
$Q = 20I$	1868.5	3.15	7476.5	2.64e8
$Q = 200I$	2017.1	3.05	1.68e4	2.53e8

Table D – Comparison based on Q in local observer

Neighbourhood Observer

	RMSE	Settling time $t_{s,1\%}$	$\|u\|_2^2$	$\|y\|_2^2$
$Q = I$	1488.4	3.66	1104.5	3.11e8
$Q = 20I$	1219.5	3.40	4630.4	3.04e8
$Q = 200I$	1081.3	3.35	1.64e4	2.96e8

Table E – Comparison based on Q in cooperative observer

Comparison on R

Note that in all the simulations we have been considering $Q = 20I$, $c = c_{min} + 1$, step reference and White Gaussian noise with variance 1 on all the followers.

Local Observer

	RMSE	Settling time $t_{s,1\%}$	$\|u\|_2^2$	$\|y\|_2^2$
$R = 0.1$	1868.5	3.15	7476.5	2.64e8
$R = 1$	584.3	3.29	246.9	3.39e8
$R = 10$	997.1	3.84	180.8	3.26e8

Table F – Comparison based on R in local observer

Neighbourhood Observer

	RMSE	Settling time $t_{s,1\%}$	$\|u\|_2^2$	$\|y\|_2^2$
$R = 0.1$	1219.5	3.40	4630.4	3.04e8
$R = 1$	1363.0	3.51	1504.2	3.00e8
$R = 10$	1918.0	4.00	502.44	2.85e8

Table G – Comparison based on R in cooperative observer

Final considerations

The project investigated the distributed control of a multi-agent magnetic levitation system, a key example of cyber-physical systems (CPS). By employing local and distributed neighbourhood observers, it was demonstrated that follower nodes can effectively track the state of the leader node, even under noisy conditions.

The two algorithms in this problem are comparable in performance and accuracy. We can observe that in this problem the local observer algorithm has lower settling time and a better tolerance to noise. On the other hand, the distributed observer algorithm can provide a more precise tracking. The energy consumption is comparable, and it highly depends on the parameters chosen.

We can notice that in the cooperative observer algorithm, the settling time saturates after a lower value of the coupling gain c than in the local observer.