POLITECNICO DI TORINO

Master's Degree in Computer Enegineering

Modeling and Control of Cyberphysical Systems

Project Activity

Authors:

Necerini Ivan

Rialti Jacopo

For code and implementations, see the Github repository:

Github

ACADEMIC YEAR 2024/2025

Contents

Part 1

Modeling

1.1 Static State Estimation under Sparse Sensor Attacks

1.1.1 Introduction

The first task addresses the problem of estimating the internal state of a static system when a subset of sensor measurements is corrupted by sparse, possibly malicious, attacks. The system is modeled using a linear measurement equation, and the goal is to reconstruct both the true system state and the set of compromised sensors. We evaluate and compare two sparse recovery algorithms ISTA and IJAM on this problem.
We simulate a static CPS with $n = 15$ state variables and $q = 30$ sensors, of which $h = 2$ are randomly attacked. The measurement matrix C is sampled from a standard normal distribution. The true state vector $\tilde{x}$ and attack vector $\tilde{a}$ are generated with random entries in predefined intervals, while Gaussian noise $\eta \sim \mathcal{N}(0, 10^{-2})$ is added to the measurements. Each experiment is repeated 20 times with different seeds, and the results are averaged. Performance is assessed using two metrics:

- **State estimation error:** $\frac{\|x^{(k)} - \tilde{x}\|_2}{\|\tilde{x}\|_2}$
- **Support recovery error:** $\sum_j |\mathbf{1}(\tilde{a}_j \neq 0) - \mathbf{1}(a_j^{(k)} \neq 0)|$

1.1.2 Analysis and Comparison

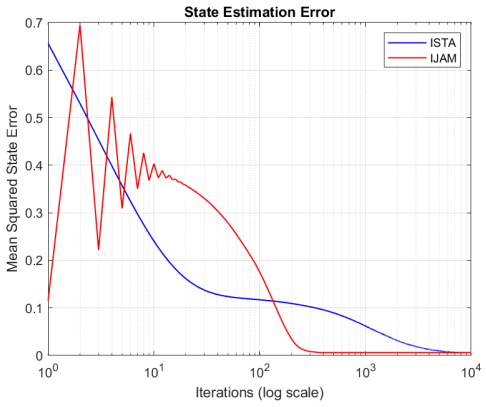


Figure 1.1: Mean squared state estimation error.

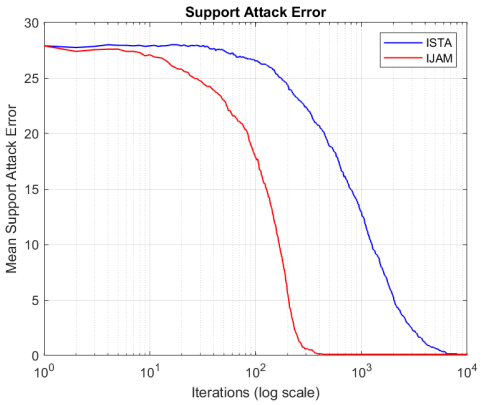


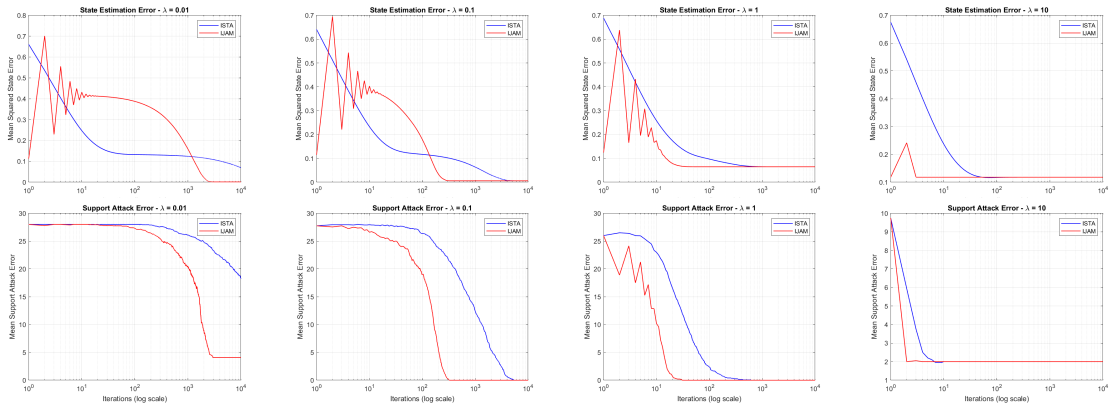
Figure 1.2: Mean support attack error.

The updated plots reveal significant differences in the convergence behavior of ISTA and IJAM, particularly in how they handle state estimation and support recovery over time. In the case of **state estimation**, both algorithms are able to converge to very low error values, eventually reaching zero. IJAM initially reduces the error more quickly, but its aggressive update strategy leads to noticeable oscillations in the early iterations. These fluctuations are a result of abrupt corrections driven by the joint update of the state and attack vectors. ISTA, in contrast, follows a smoother trajectory thanks to its more conservative gradient-based update with soft-thresholding. Although it requires more iterations

to reach the same level of accuracy, ISTA progressively reduces the error without instability, ultimately converging to the same final value as IJAM, but with a more stable evolution. For **support attack recovery**, IJAM clearly outperforms ISTA in both speed and effectiveness. The support error for IJAM drops rapidly to zero within a few hundred iterations, indicating that the attacked sensors are quickly and accurately identified. ISTA improves more slowly, requiring nearly an order of magnitude more iterations to reach comparable accuracy. This is due to its more cautious approach in thresholding small values, which delays the elimination of false positives.

1.1.3 Effect of λ on Estimation Performance

In this section, we investigate how the choice of the regularization parameter λ affects the convergence and accuracy of ISTA and IJAM. The parameter λ regulates the sparsity level in the estimated attack vector through soft-thresholding. Lower values of λ preserve more non-zero entries (potentially including noise) while higher values promote sparsity by forcing small entries to zero, possibly discarding useful components. The goal is to identify an optimal balance for convergence speed and estimation precision.



$\lambda = 0.01$ $\lambda = 0.10$ $\lambda = 1$ $\lambda = 10$

Figure 1.3: State and support attack errors for different values of λ (ISTA vs IJAM). Each column shows both metrics for a fixed λ.

From the plots, it is evident that $\lambda = 0.01$ is too small: both algorithms converge slowly and retain too many irrelevant components in the attack vector. As λ increases to 0.1, the convergence improves significantly in both metrics. The state error drops faster, and the support attack error reaches zero earlier—especially for IJAM, which benefits the most from mild sparsification.
At $\lambda = 1$, the support detection remains effective, but the state estimation begins to show signs of degradation, likely due to the premature removal of weak but informative signal components. With $\lambda = 10$, this trend becomes more pronounced: surprisingly, the state estimation error does not worsen drastically—it reaches levels similar to those observed with $\lambda = 1$. However, the support recovery fails to achieve full convergence. The support error settles around 2 rather than dropping to zero.
This behavior suggests that while a large λ may still allow the algorithm to fit the overall system behavior, it does so by effectively "absorbing" some attack effects into the state estimate. This compensation results in acceptable global performance, but a failure to isolate and identify all attacked sensors.

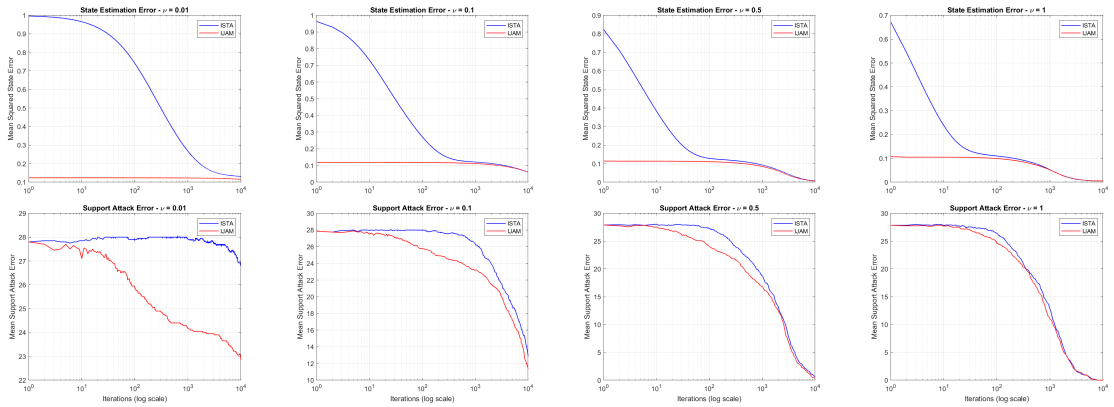
These observations confirm that $\lambda = 0.1$ remains the best compromise in this context, offering fast convergence and accurate, across both evaluation metrics.

1.1.4 Effect of ν on Convergence and Stability

The step-size parameter ν plays a fundamental role in the convergence behavior and stability of both ISTA and IJAM. Rather than selecting an absolute value for ν, we adopt a normalized formulation to ensure numerical stability and consistent scaling across experiments while adapting to the spectral properties of the measurement matrix G. Specifically, we define:

$$\nu = \frac{\text{factor}}{\|G\|_2^2}, \quad \text{where } G = [C\ I]$$

In preliminary experiments where ν was set directly as an absolute value, we observed instability and divergence (particularly in IJAM) where the state estimation error would exceed 10^4. The normalization effectively prevents such numerical issues and enables fair comparison between algorithms.
In the following analysis, we test four different multiplicative factors: 0.01, 0.1, 0.5, and 1.0, and assess their effect on estimation and support recovery.



ν factor = 0.01 | ν factor = 0.1 | ν factor = 0.5 | ν factor = 1.0

Figure 1.4: State and support attack errors for different normalized ν values (ISTA vs IJAM). Each column shows both metrics for a fixed ν factor.

From the plots, we observe the following behaviors:

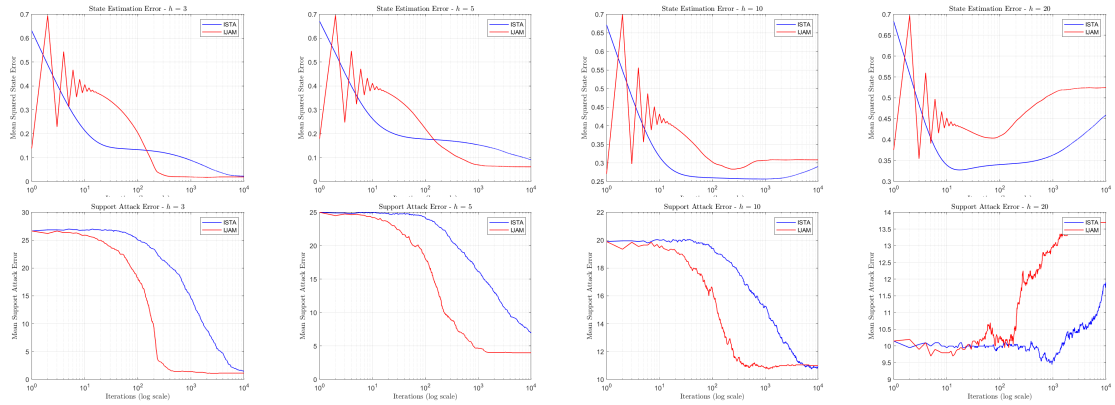
- **Small step-size (ν factor $= 0.01$):** ISTA converges slowly in both state and support errors. IJAM, on the other hand, quickly stabilizes at a low state estimation error but fails to reduce, in this number of iterations, the support attack error, which remains above 23. This indicates that while IJAM compensates for attacks internally, the update magnitude is insufficient to isolate corrupted sensors.
- **Moderate step-size (ν factor $= 0.1$):** Both algorithms benefit significantly. IJAM improves support recovery, and ISTA catches up, showing nearly the same support performance while converging faster in state estimation than before. This factor appears to unlock the full potential of soft-thresholding.

- **Larger step-size (ν factor $= 0.5$):** The performance of ISTA and IJAM becomes nearly indistinguishable, except for the early iterations. No instability is observed at this stage, suggesting this value is within a safe operational range.
- **High step-size (ν factor $= 1.0$):** Similar trends are observed as with the previous case. IJAM maintains stable behavior and similar accuracy, confirming that the algorithm can benefit from aggressive updates as long as normalization is applied.

The key insight is that increasing ν (via the normalized factor) enhances the effectiveness of the soft-thresholding operator, allowing small entries in the attack vector to cross the $\lambda\nu$ threshold more rapidly. This promotes sparsity and suppresses false positives in fewer iterations. However, such improvements are only achievable when ν is carefully scaled: too small and the algorithm stalls; too large (without normalization) and the algorithm diverges.

1.1.5 Resilience to Sparse Attacks: Increasing h

In this section, we investigate the resilience of ISTA and IJAM when increasing the number of sensors under attack. We test values $h = 3, 5, 10, 20$ while keeping all other parameters fixed. Both algorithms are evaluated in terms of their ability to reconstruct the state and identify the attacked sensors as the adversarial impact grows.



$h = 3$ $h = 5$ $h = 10$ $h = 20$

Figure 1.5: State and support attack error under increasing number of compromised sensors h. Each column shows both metrics for a fixed h.

The results indicate that both ISTA and IJAM lose effectiveness as the number of attacked sensors increases. For low values of h (e.g., $h = 3$), both algorithms accurately identify most of the attacked sensors and achieve low state estimation error. IJAM converges more rapidly in support detection, while ISTA shows smoother and more stable behavior over time.
As h increases, the support recovery error rises notably. At $h = 10$, IJAM slightly underperforms ISTA in state estimation, although the two algorithms achieve comparable performance in support recovery. Interestingly, ISTA exhibits a late-stage increase in state error, possibly due to instability or overfitting caused by corrupted measurements.
When $h = 20$, corresponding to heavy corruption (two-thirds of sensors attacked), ISTA surprisingly outperforms IJAM in both metrics. This behavior may be attributed to ISTA's

more conservative update rule, which prevents excessive reactions to misleading information. Still, for both methods, the error curves become non-monotonic: after initially decreasing, they tend to rise again in the later iterations.
This degradation can be explained by the accumulation of residual attack components and noise, which gradually distort the updates. Without a stopping criterion or adaptive step-size, both algorithms may drift away from the optimal solution during prolonged execution, especially under intense attack conditions.
In conclusion, both ISTA and IJAM rely on the sparsity of the attack vector. When this assumption is violated, ISTA tends to be more robust, whereas IJAM remains preferable under mild and sparse attack scenarios.

1.2 Target Localization under Sparse Sensor Attacks

We consider the problem of localizing a target in a 10×10 meter indoor area, using $q = 20$ RSS sensors. Some of the sensors may be compromised by sparse adversarial attacks. The system is modeled as:

$$y = Dx + a$$

where $x \in \mathbb{R}^{100}$ is a sparse vector with a single nonzero entry indicating the target's location, and $a \in \mathbb{R}^q$ is a sparse attack vector corrupting the sensor measurements.

To simultaneously estimate the target's position and identify the attacked sensors, we construct an **augmented state vector**:

$$z = \begin{bmatrix} x \\ a \end{bmatrix}, \quad G = \text{normalize}([D\ I])$$

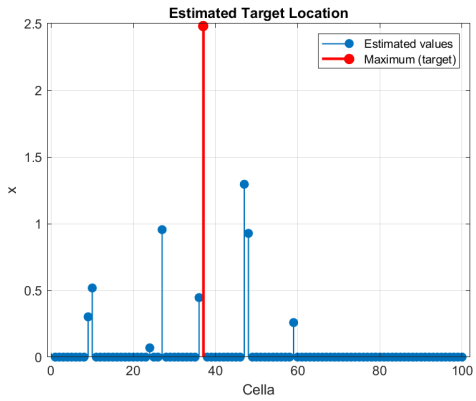
This reformulation allows us to use a single optimization framework to recover both the sparse position and the sparse attack components by solving the following weighted LASSO problem:

$$\min_{x,a} \left\| G \begin{bmatrix} x \\ a \end{bmatrix} - y \right\|_2^2 + \lambda(\|x\|_1 + \|a\|_1)$$

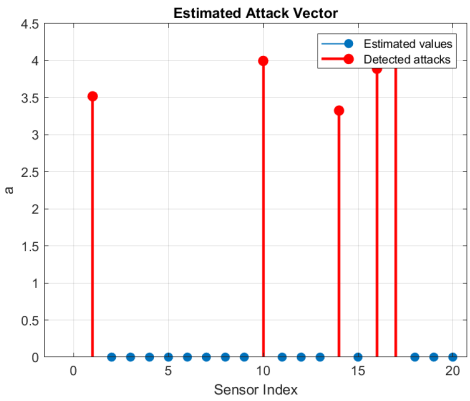
The use of the augmented state enables a unified sparse recovery strategy via ℓ_1 minimization. Normalization ensures balanced contributions from both parts of G during the ISTA updates.

Implementation. We implemented ISTA with soft-thresholding using $\lambda = 10$ and step-size $\nu = \|G\|_2^{-2}$. The optimization converges in a few hundred iterations.

Results. The algorithm successfully identifies both the target location and the attacked sensors, as shown in the figures below:



Estimated Target Location



Estimated Attack Vector

These results match the expected solution. Accurate support recovery is achieved thanks to the sparsity-promoting regularization and the properly scaled step-size. The augmented formulation plays a key role in simultaneously isolating the effects of the true target signal and the adversarial perturbations.

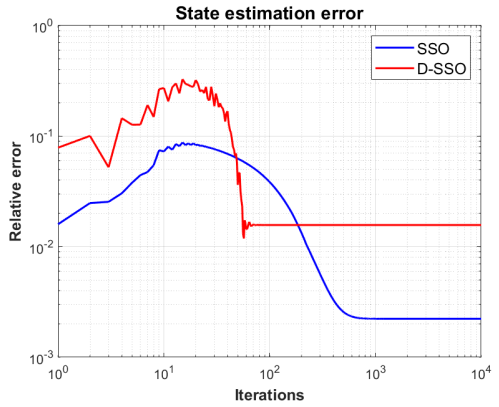
1.3 Secure State Estimation under Sparse Sensor Attacks

In this task, we consider a discrete-time dynamic CPS affected by sparse and constant sensor attacks:

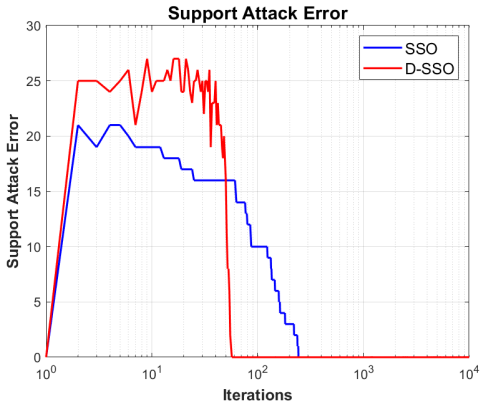
$$x(k+1) = Ax(k), \quad y(k) = Cx(k) + a$$

The objective is to estimate the true system state $x(k)$ while identifying the attack vector a, which is sparse and affects only $h = 3$ out of $q = 30$ sensors. Since the system is subject to adversarial corruption, a standard Luenberger observer is not suitable: even if a is constant, the presence of unobservable modes (e.g., eigenvalues of A equal to 1) can prevent correct reconstruction. In contrast, sparse observers exploit the sparsity of a using soft-thresholding strategies.

We implement and compare two approaches: the Sparse Sensor Observer (SSO), and its deadbeat variant D-SSO. Both estimate $x(k)$ and a online over $T = 10^4$ iterations using $\lambda = 0.1$, with step-size ν set according to the suggested formulas. From the plots in Figure 1.6, we observe that both methods initially struggle with the presence of attacks, but around iteration $k \approx 80$ they correctly detect the attack support. Moreover, their trends are very similar to those of the IJAM (D-SSO is the observer version of IJAM and they are equal if A=I) and ISTA (SSO is the observer versione of ISTA, they are equal if A=I) algorithms, which makes sense since they are the online versions of these offline methods. So, D-SSO shows slightly faster convergence while SSO is more stable and ultimately results in lower errors. Regarding the support attack error, both methods reach zero, confirming exact identification of the attacked sensors in finite time. However, D-SSO detects the correct support more rapidly, reaching zero support error several iterations before SSO.



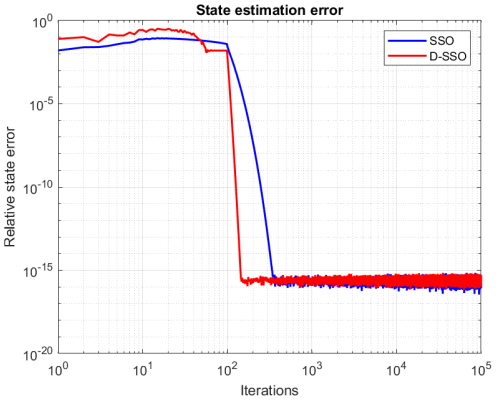
State estimation error (log-log)



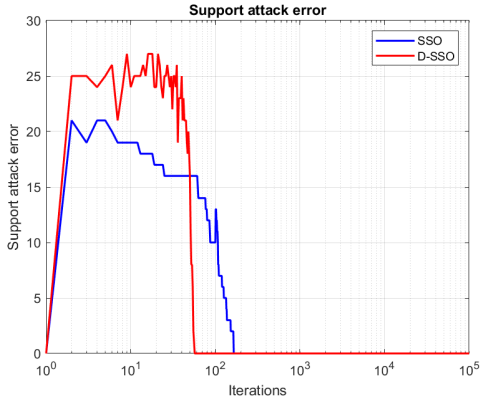
Support attack error (semi-log)

Figure 1.6: Task 3 – SSO vs D-SSO (Unrefined results).

After reliable detection of the attacked sensors (in our case, sensors 9, 16 and 26), we refine the estimation by zeroing the corresponding rows of C, effectively removing corrupted channels from the observer's input. This requires recomputing the observer gain L for D-SSO, since the output structure has changed. Figure 1.7 shows that this refinement leads to near-zero relative errors for both observers (below 10^{-14}), achieving machine-level precision. The support error remains zero, validating the effectiveness of excluding the corrupted rows after correct support recovery.



Refined state estimation error



Refined support attack error

Figure 1.7: Task 3 – Refined solution after support freezing.

In summary, both SSO and D-SSO successfully track the system dynamics and estimate the sparse attack vector. D-SSO converges faster regarding the attack, while SSO is slightly more robust. The refinement step dramatically improves accuracy, confirming the importance of removing compromised measurements once their support is reliably estimated. No measurement noise was present in the provided data, making the problem fully deterministic and ideal for testing sparse estimation techniques.

1.4 Target Tracking under Sparse Sensor Attacks

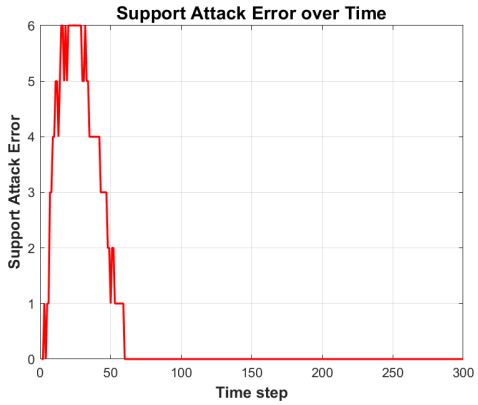
In this task, we address the problem of dynamically tracking the position of a moving target in a 6×6 square grid (with $n = 36$ states), using $q = 15$ RSS sensors, some of which are affected by constant sparse adversarial attacks. The target activates only one grid cell at each time step, resulting in a sparse system state $x(k)$, while the measurements $y(k)$ are corrupted by a fixed sparse attack vector $\tilde{a}$. The system evolution follows the model:

$$x(k+1) = Ax(k), \quad y(k) = Dx(k) + \tilde{a} + \eta$$

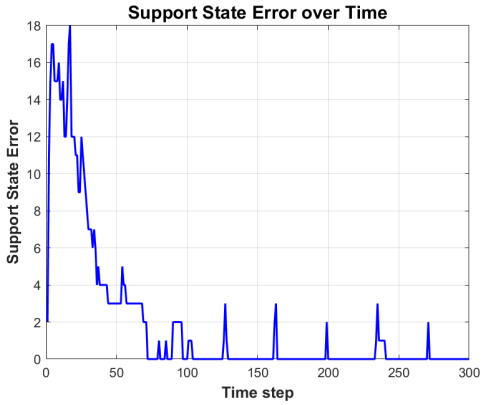
where η is measurement noise and D is the measurement matrix provided in the dataset. We implement a modified Sparse Secure Observer (SSO) based on ISTA, tailored for systems where both the state and the attack vectors are sparse. At each time step, the state is predicted via $x_{\text{pred}} = Ax(k)$, and the joint vector $z = [x_{\text{pred}}; \hat{a}]$ is updated via a gradient step using the normalized matrix $G = \text{normalize}([D\ I])$, followed by soft-thresholding. The normalization ensures that the columns of G have zero mean and unit variance, making the optimization more stable. The update rule is:

$$z \leftarrow \text{soft}(z - \nu G^\top (Gz - y(k)), \nu\lambda)$$

with parameters $\lambda = 10$ and $\nu = \|G\|_2^{-2}$.
Before validating the tracking results, we assess whether the attack-free system, described by the pair (A, D), is observable. This is done by computing the observability matrix and verifying that it has full rank. In our case, the matrix has rank 36, matching the system's state dimension, which confirms that the system is fully observable in absence of attacks. However, when augmenting the state to include constant attacks, forming the block-diagonal matrix $A_{\text{aug}} = \text{blkdiag}(A, I_q)$ and the corresponding measurement matrix $[D\ I]$, the observability matrix of the augmented system does not reach full rank (we obtain a rank of 50 instead of 51). This proves that the system is not observable when constant attacks are considered, which excludes the use of classical Luenberger observers in this setting. Moreover, implementing a D-SSO observer is also not feasible here: since $q < n$, the matrix G is fat and the pseudoinverse $G^\dagger$ is not uniquely defined. This makes the associated least-squares problem underdetermined and prevents the application of the D-SSO algorithm, as already discussed in Task 2.
The SSO algorithm, however, successfully leverages the sparse structure of both the state and attack signals. Since the true state is not provided, we approximate the true support at each time by placing the target in the cell with the maximum estimated activation. This allows us to compute support errors for both state and attack vectors. The results in Figure 1.8 confirm the effectiveness of this method. The support attack error drops to zero before iteration 60 and remains stable, indicating that the adversarial sensors have been correctly identified. The support state error also converges, albeit more slowly and with some oscillations, as the soft-thresholding operation introduces variability in the sparse state estimation, especially in the presence of noise or ambiguous measurements.



Support attack error over time



Support state error over time

Figure 1.8: Performance of SSO tracking in presence of sparse state and sensor attacks.

In conclusion, although the augmented system is not observable and D-SSO cannot be used due to the underdetermined nature of the problem, the modified SSO algorithm is effective in simultaneously estimating the target's sparse position and identifying the attacked sensors with high accuracy and within a finite number of iterations.

1.5 Distributed Localization under Sparse Sensor Attacks

In this final task, we address the problem of localizing a target within a 10×10 grid using $q = 20$ RSS sensors, some of which may be subject to sparse, constant attacks. Unlike previous tasks, here we adopt a distributed approach based on the DISTA algorithm, where each sensor updates its estimate using only local information and messages received from its neighbors, according to a predefined communication topology.
We solve a weighted LASSO problem via iterative gradient updates and soft-thresholding, promoting sparsity in both the estimated target position x and attack vector a. The observation matrix is normalized as $G = \text{normalize}([D\ I])$, and the solution is computed separately for two network topologies: a ring and a star (Figure 1.9).

Figure 1.9: Ring and star topologies used for distributed estimation.

From both topologies, the estimated target position correctly converges to cell 37, and the attacked sensors (1, 10, 14, 16, 17) are successfully identified. These results confirm that

DISTA can provide accurate localization and attack detection even without centralized coordination.
Regarding the implementation constraints, the IJAM algorithm is not applicable in this setting: it requires access to the full measurement vector and dictionary, which contradicts the distributed nature of the task. DISTA, on the other hand, fits the assumptions and is the only feasible approach.
As for the topologies, the star structure converges faster thanks to the central node that aggregates the local estimates, but is less robust: failure or corruption of the central node would compromise the entire system. The ring topology offers better fault tolerance and decentralization but tends to converge more slowly due to the absence of a global aggregator.
In summary, this task demonstrates how distributed estimation techniques can effectively localize a target and detect sensor attacks under realistic, decentralized conditions.

Part 2

Control

2.1 Introduction

This second part of the project focuses on the modeling and distributed control of a Cyber-Physical System (CPS) composed of 7 magnetic levitation (maglev) units. The system is structured as a multi-agent system where one maglev unit acts as the *leader node* S_0, and the remaining six units are *follower nodes* S_i, with $i = 1, \ldots, 6$.
Each maglev agent is modeled by a linear time-invariant (LTI) system, derived from the linearization of the nonlinear dynamics around a suitable equilibrium point. The state-space model for each agent is:

$$\dot{x}_i = Ax_i + Bu_i, \quad y_i = Cx_i,$$

with matrices defined as:

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

The state variables are not directly measurable ($C \neq I$), which requires the implementation of observer-based controllers. The main objective is to solve the *cooperative tracking problem* (drive the global disagreement error to zero, so that all follower agents asymptotically track the trajectory generated by the leader).
To achieve this goal, two different distributed regulators are implemented and compared:

- A regulator based on **local observers**, where each follower estimates its own state independently.
- A regulator based on a **distributed neighborhood observer structure**, where agents exploit local and neighbor information.

The system is tested under various *communication network topologies* (line, ring, mesh, full) and for different leader reference signals (step, ramp, sinusoidal). Moreover, the analysis includes the effects of measurement noise and the influence of design parameters such as the coupling gain c and the weighting matrices Q and R.

2.2 Design of the Distributed Control System

The control architecture developed in this project relies on a *cooperative State-Variable Feedback (SVFB)* strategy for multi-agent systems. The goal is to ensure that all follower agents asymptotically track the trajectory generated by the leader node, minimizing the global disagreement error through local control actions and inter-agent communication.

2.2.1 Communication Topology

The communication between agents is modeled by a directed graph, where each node represents a maglev unit, and each edge indicates a one-way communication link. This structure is encoded by an *adjacency matrix* $A = [a_{ij}]$, where $a_{ij} > 0$ means that agent i can access the state or estimate of agent j.

From this, the *Laplacian matrix* is computed as $L = D - A$, with D being the diagonal matrix of in-degrees. The influence of the leader node on the followers is modeled through the *pinning matrix* G, which contains ones in the diagonal entries corresponding to followers directly connected to the leader. In this project, only agent S_1 is pinned.
The network topology plays a fundamental role in the information exchange mechanism and, consequently, in the system's closed-loop behavior. Four different topologies are implemented and tested:

- **Line**: unidirectional chain;
- **Ring**: closed-loop structure with bidirectional links;
- **Mesh**: sparse graph with diagonal connections;
- **Full**: complete communication network.

2.2.2 Control Law

Each follower node S_i applies the following distributed control protocol:

$$u_i = cK\varepsilon_i,$$

where:

- K is the state-feedback gain matrix obtained via LQR;
- $c > 0$ is the coupling gain;
- ε_i is the *neighborhood tracking error* defined as:

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

  which measures the relative deviation of agent i from its neighbors and from the leader.

The control objective is to drive the global disagreement error

$$\delta_i(t) = x_i(t) - x_0(t)$$

to zero for all i, achieving consensus and synchronization.

2.2.3 Observer Strategies

Since the full state x_i is not directly measurable (i.e., $C \neq I$), each agent must estimate it from its output y_i. Two observer configurations are considered and compared:

Local Observer. Each agent estimates its own state using only its output. The observer is defined by:

$$\dot{\hat{x}}_i = A\hat{x}_i + Bu_i - cF\tilde{y}_i,$$

where $\tilde{y}_i = \hat{y}_i - y_i$ is the local output estimation error. This strategy is simple and does not require communication, but is more sensitive to noise and local disturbances.

Neighborhood Observer. This approach extends the standard Luenberger observer by incorporating information from neighboring agents. The estimation dynamics are:

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

with:

$$\xi_i = \sum_{j=1}^{N} a_{ij}(\hat{y}_j - \hat{y}_i) + g_i(\hat{y}_0 - \hat{y}_i),$$

where ξ_i is the *neighborhood output estimation error*. This structure improves robustness and estimation accuracy but increases communication complexity.
Both observers are implemented in `Local.slx` and `Neighborhood.slx`, and their effectiveness is evaluated under identical conditions.

2.2.4 Design Parameters: c, Q, R

The control and estimation performance is strongly influenced by three tunable parameters:

- **Coupling Gain** c: this scalar determines how strongly each agent reacts to its disagreement with neighbors and the leader. Its theoretical lower bound depends on the eigenvalues of the matrix $(L+G)$. A higher c typically leads to faster convergence, but may amplify noise and destabilize the observer.
- **State Weighting Matrix** Q and **Input Weighting Scalar** R: these two matrices define the trade-off between state tracking accuracy and control effort in the Linear-Quadratic (LQ) optimal control formulation. The goal is to minimize the infinite-horizon cost functional:

  $$u(t) = \arg\min_{u(t)} \left[\frac{1}{2} \int_0^\infty \Big(x(t)^\top Q x(t) + u(t)^\top R u(t) \Big)\, dt \right]$$

  In this context, Q assigns a penalty to the energy of the state signal $x(t)$, promoting precise tracking, while R penalizes the magnitude of the control input $u(t)$, encouraging energy-efficient actuation.

  For instance, choosing $R \ll Q$ yields a fast controller with high responsiveness, at the cost of increased control effort and possible actuator saturation.

  Choosing $Q = I_2$ penalizes deviations in both position and velocity equally.

These parameters do not affect only the controller but also the observer, since the gain F used in the estimation is derived from a dual Riccati equation:

$$F = PC^\top R^{-1}.$$

Thus, adjusting Q and R has a twofold effect: it modifies both the control law (K) and the observer gain (F), influencing tracking accuracy, convergence speed, and robustness.

Control Gain Design

The gain matrix K is computed by solving the continuous-time Algebraic Riccati Equation (ARE):

$$A^\top P + PA + Q - PBR^{-1}B^\top P = 0,$$

and then:

$$K = R^{-1}B^{\top}P.$$

The design ensures that the matrix $(A - BK)$ is Hurwitz, providing closed-loop stability for the isolated system. To extend this to the multi-agent setting, the coupling gain c is chosen to satisfy:

$$c \geq \frac{1}{2 \cdot \min_i \mathrm{Re}(\lambda_i)},$$

where λ_i are the nonzero eigenvalues of $(L+G)$. This guarantees convergence of the entire network to the leader trajectory.

2.3 Experimental Setup

2.3.1 Leader Configuration

To enable reference tracking, the leader is modeled as an autonomous system with state-feedback:

$$u_0 = -K_0 x_0, \qquad \dot{x}_0 = (A - BK_0)x_0.$$

Since the full state is not directly measurable, a Luenberger observer is used:

$$\dot{\hat{x}} = (A - LC)\hat{x} + Bu + Ly,$$

with L chosen so that $(A - LC)$ is Hurwitz, ensuring stable estimation.
The matrix K_0 is designed through pole placement to generate a desired reference trajectory. The eigenvalues of $(A - BK_0)$ are selected based on the type of reference:

- **Step:** $\lambda = \{0, -1\}$ — for asymptotic tracking of a constant signal;
- **Ramp:** $\lambda = \{0, 0\}$ — to simulate a system with integrative dynamics;
- **Sinusoid:** $\lambda = \{+j, -j\}$ — to generate oscillatory behavior.

The gain K_0 is computed using either the `place` function or Ackermann's formula (`acker`), depending on the pole configuration. Both methods are used to assign the closed-loop poles of the system:

- The `place` function uses numerical algorithms based on the controllability matrix and provides reliable performance in most cases, particularly when the desired poles are real and distinct.
- The `acker` function is based on Ackermann's formula and is more robust when assigning repeated poles. For this reason, it is preferred in the ramp case, where the desired eigenvalues are $\{0, 0\}$.

To create a meaningful transient, the leader is initialized in Simulink with the state:

$$x_0 = \begin{bmatrix} 0 \\ 100 \end{bmatrix},$$

corresponding to a zero initial position and a high velocity. *As a result, the shape and amplitude of the reference signal are fully governed by this initial condition and the closed-loop dynamics, with no need for analytical amplitude design. .*

2.3.2 Follower Configuration

All follower agents are configured to replicate the same closed-loop dynamics as the leader, ensuring uniform behavior across the network. Each follower applies the distributed control law based on its local tracking error and the coupling gain, and reconstructs its internal state using one of the two observer strategies.
This setup guarantees consistency in dynamics and allows for a fair comparison of performance under different network topologies and observer configurations.

2.3.3 Noise Configuration

As required by the project specifications, we investigate the impact of noise on output measurements exchanged between agents. We introduce noise only at the level of inter-agent communication, without corrupting the internal dynamics or the state estimation of each node. To achieve this, the noise is injected at the output of each agent before the cooperative control protocol, affecting only the shared information. The observers (both in the leader and in the followers) operate on clean, locally available outputs $y_i(t)$, ensuring that internal state estimation remains accurate and unaffected, preserving the integrity of the internal model while realistically simulating noisy communication links.
The noise is modeled as a **signal-dependent** process: a Gaussian signal with zero mean and unit variance is generated at each time step using the `Random Number` block in Simulink. This signal is then scaled proportionally to the magnitude of the output. For each agent S_i, the noise $\eta_i(t)$ added to its output is defined as:

$$\eta_i(t) = k_i \cdot |y_i(t)| \cdot \mathcal{N}(0, 1),$$

where k_i is a sensitivity coefficient specific to agent i. This approach ensures that the noise power grows with the amplitude of the signal, simulating realistic sensor behavior. In the Simulink implementation the resulting term $\eta_i(t)$ is added to the measured output and used only in the cooperative protocol (not in the local observer).

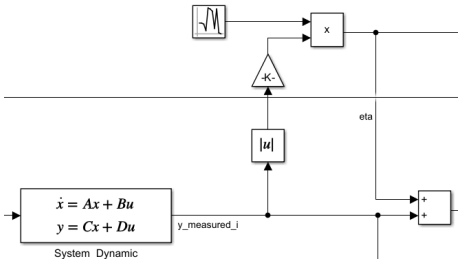


Figure 2.1

The parameters governing this behavior are defined in the `params.m` file:

- `p.noise_sensitivity` — a scalar specifying global noise sensitivity for the leader node;
- `p.agent_noise_sensitivity_vector` — a vector of 6 elements, where each entry defines the noise level for one follower agent.

This configuration supports flexible and realistic simulation scenarios where noise can be enabled selectively for chosen agents.

2.4 Analyses

We now begin the analysis of our simulation results. Unless otherwise specified, all tests are performed under a common configuration that serves as the default setup.

2.4.1 Default Settings

We adopt the following baseline parameters:

- **Observer:** Local observer structure is used;
- **Reference signal:** Step input generated by the leader node;
- **Weighting matrices:** $Q = I_2$ and $R = 1$, resulting in equal penalization of position and velocity, with moderate control effort;
- **Coupling gain:** The minimum value of c that guarantees convergence for the chosen topology.

2.4.2 Effect of Network Topology on System Behavior

We start by analyzing the influence of the communication topology on the system's collective dynamics. In this experiment, we simulate the agent outputs (position and velocity) under four different network configurations: Line, Ring, Mesh, and Fully Connected. In each case, the coupling gain c is set to the minimum value that guarantees convergence, as reported in the following table:

Topology	**Minimum c**
Line	0.5
Ring	4.5964
Mesh	3.5593
Fully connected	3.4271

Table 2.1: Minimum coupling gain c required for convergence in each topology.

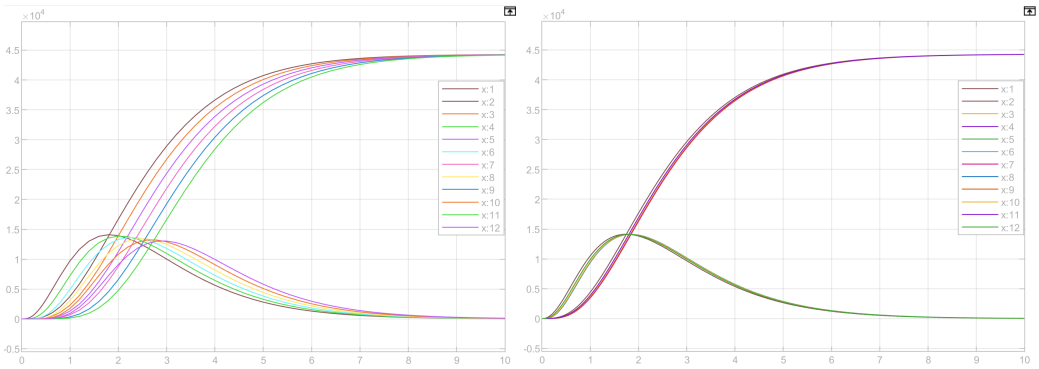


Figure 2.2: Agent outputs in Line (left) and Ring (right) topologies.

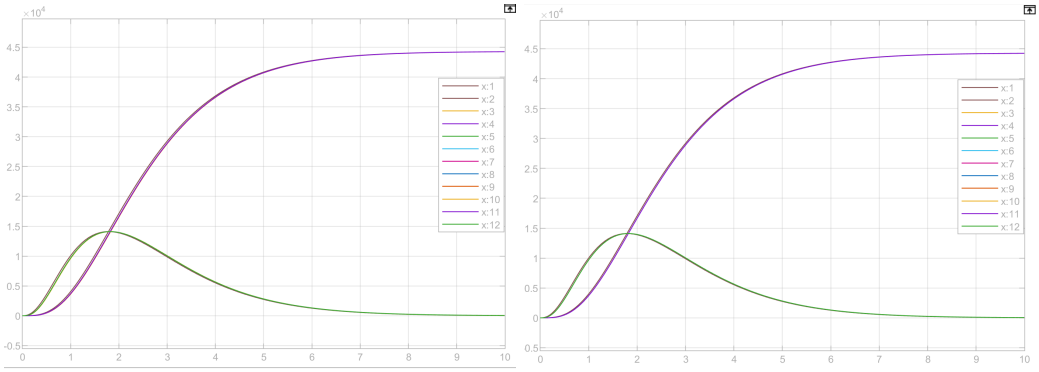


Figure 2.3: Agent outputs in Mesh (left) and Fully Connected (right) topologies.

The results clearly show how the network topology affects both the speed of convergence and the quality of synchronization across agents. In the *Line* configuration, where each agent communicates only with its immediate neighbor, we observe slow convergence and large transient differences. The agents closer to the leader align earlier, while those farther away follow with a visible delay, highlighting the limited information propagation through the chain.
Introducing a *Ring* topology reduces this effect by closing the loop and allowing bidirectional communication in a circular pattern. Although the improvement is visible, the convergence remains relatively slow compared to denser networks.
With the *Mesh* topology, the connectivity increases significantly, and the agents benefit from multiple communication paths. The trajectories converge faster and more uniformly, and the transient mismatch between agents becomes less noticeable.
Finally, the *Fully Connected* topology yields the best performance. Here, each agent directly receives information from all others, resulting in an almost immediate synchronization of both position and velocity. The trajectories overlap almost perfectly from the early stages of the simulation, confirming that high connectivity enables superior cooperative behavior.
So, better connectivity leads to faster and more accurate consensus, reducing both transient disagreement and steady-state error.

Tracking Error Across Topologies

In this analysis, we evaluate the overall tracking performance for different network topologies under the same coupling gain. Specifically, we set $c = 4.5964$, i.e., the convergence threshold of the ring topology, to ensure that all configurations converge and allow a fair comparison.
We compute three distinct performance metrics over time:

- The **global tracking error norm** $\|\delta(t)\|$, which measures the collective deviation of all agents from the leader;
- The **position tracking error** $\|\delta_{\text{pos}}(t)\|$, computed from the position components of each agent;
- The **velocity tracking error** $\|\delta_{\text{vel}}(t)\|$, computed from the velocity components.

Each agent's state is represented as $x_i(t) = [x_i^{(1)}(t),\ x_i^{(2)}(t)]^\top$, where $x^{(1)}$ and $x^{(2)}$ denote position and velocity respectively. The tracking error with respect to the leader is defined

as:

$$\delta_i(t) = x_i(t) - x_0(t), \quad \delta(t) = \begin{bmatrix} \delta_1(t) & \cdots & \delta_N(t) \end{bmatrix}.$$

The total error norm is computed as:

$$\|\delta(t)\| = \sqrt{\sum_{i=1}^{N} \|x_i(t) - x_0(t)\|^2}.$$

The position and velocity tracking errors are calculated analogously by isolating the corresponding components in each $\delta_i(t)$, and used to evaluate separately the agents' synchronization in space and speed.

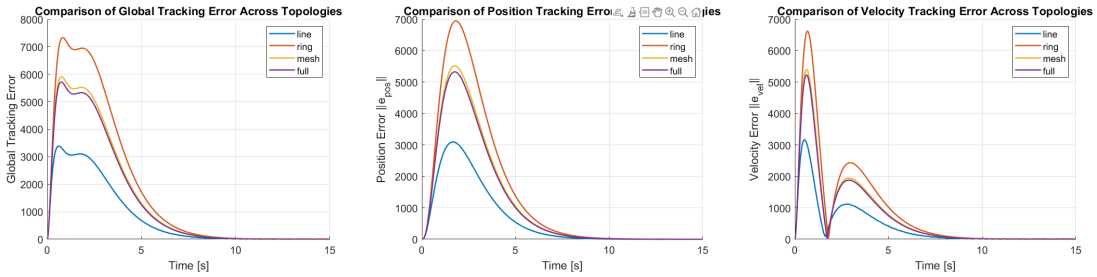


Figure 2.4: Tracking error norms over time for different topologies. Left: global error. Center: position error. Right: velocity error.

- The *line* topology shows the fastest convergence, despite being the least connected. This is due to the relatively large coupling gain c assigned in this simulation, which exceeds the threshold required for consensus.
- The *ring*, *mesh*, and *full* configurations converge more slowly in comparison. Although they feature better connectivity, they start from higher initial errors and require more time to synchronize under the same gain.
- The *velocity tracking error* exhibits two distinct oscillations before settling. This reflects the dynamic interactions within the system, where the observer and coupling gains jointly affect the transient response. Velocity, being the derivative of position, naturally responds more quickly to these transients.
- In contrast, the *position tracking error* shows a smoother decay, since it integrates the effect of the velocity and therefore tends to attenuate fast transients.
- Since the state cost matrix is set to $Q = I$, both position and velocity are weighted equally in the regulation objective. This ensures that the control law balances the convergence of the two states proportionally, which is consistent with the comparable steady-state behavior observed in the error plots.

Overall, these results confirm the correct operation of the cooperative control architecture. All follower agents successfully track the leader's state. The separate analysis of position and velocity further demonstrates the stability and convergence of the system under different topologies, validating the theoretical design with simulation results.

2.4.3 Effect of Reference Type on Tracking Performance

In this section, we analyze the impact of different reference signals on the system behavior, under fixed network conditions. We consider the *line* topology and maintain the default control parameters.
The experiment is repeated for three types of reference input applied to the leader: **step**, **ramp**, and **sine**. For each case, we plot the agent states and compute the three tracking error norms over time: global, position, and velocity.

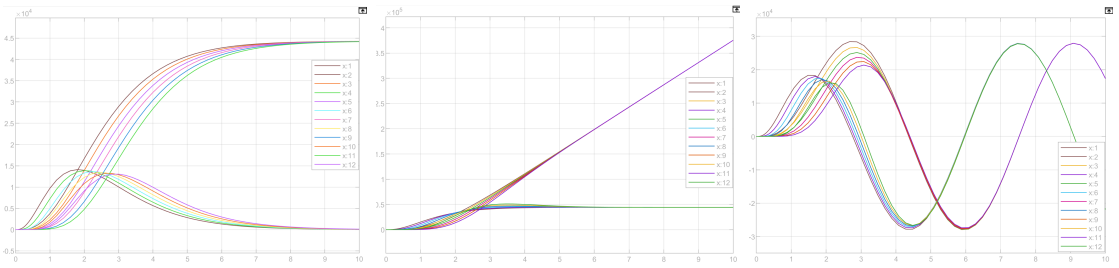


Figure 2.5: Agent outputs for different references. Left: Step. Center: Ramp. Right: Sine.

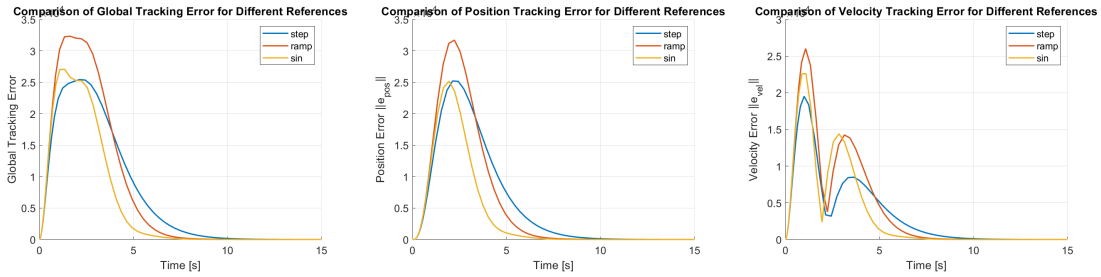


Figure 2.6: Tracking error norms for different references. Left: Global error. Center: Position error. Right: Velocity error.

The analysis of the tracking error plots reveals how the nature of the reference signal influences the transient and steady-state performance of the multi-agent system.
Among the three references, the *sine* input results in the fastest convergence across all error metrics. Despite introducing continuous oscillations in the leader trajectory, its bounded nature allows the agents to synchronize efficiently, leading to the lowest and smoothest tracking error profiles. The *ramp* reference exhibits the largest initial tracking error. This is due to the unbounded growth of the signal, which imposes a sustained control effort. However, the system manages to converge effectively after a longer transient. The *step* input, while producing a more pronounced but bounded response, leads to a slower decay of the tracking error compared to sine and ramp, suggesting that the system takes more time to settle after the initial jump.
Overall, these results highlight the robustness of the control strategy: in all cases, agents are able to asymptotically track the leader's state. The observed differences in error magnitude and convergence speed are consistent with the temporal structure of each reference signal, confirming the controller's ability to adapt to varying dynamic demands while maintaining consensus.

2.4.4 Effect of c and Q on Synchronization

We now investigate how the choice of control parameters influences the position tracking behavior of the agents. Specifically, we analyze the effect of varying the coupling gain c and the state cost matrix Q in the distributed control scheme. We focus here on the agents' position outputs.
The experiment is performed under the *line* topology with fixed initial conditions. Two values of coupling gain are tested: $c = 0.5$ and $c = 2$. For each case, we simulate the system with a diagonal state cost matrix $Q = \alpha \cdot I$, where $\alpha \in \{0.1,\ 1,\ 10\}$. This design allows us to evaluate the effect of both weak and strong state penalization.

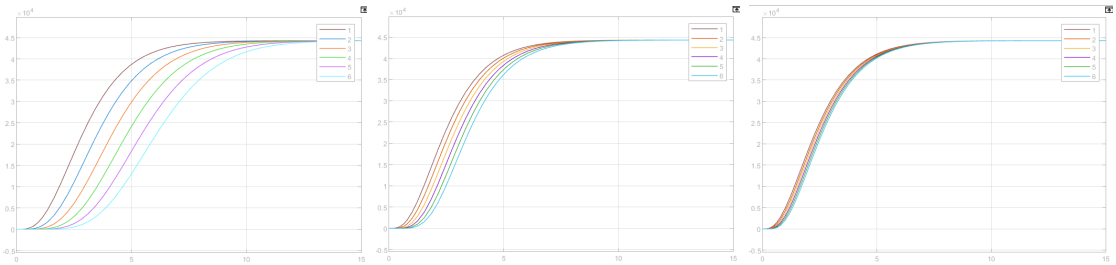


Figure 2.7: Position outputs with $c = 0.5$ and varying Q. Left: $Q = 0.1I$. Center: $Q = I$. Right: $Q = 10I$.

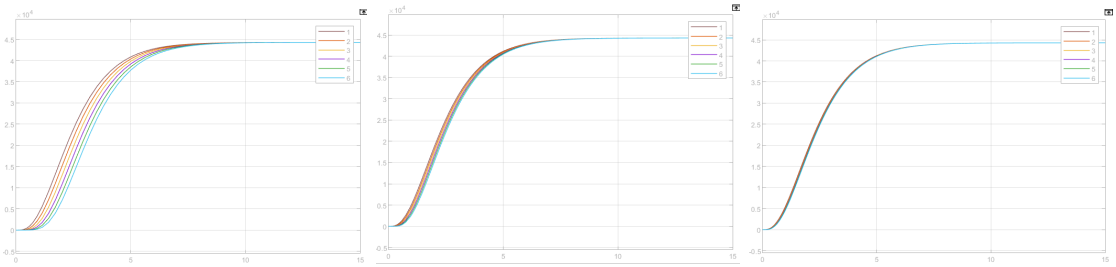


Figure 2.8: Position outputs with $c = 2$ and varying Q. Left: $Q = 0.1I$. Center: $Q = I$. Right: $Q = 10I$.

The plots reveal several key aspects of the interplay between c and Q:
With lower coupling $c = 0.5$, the information exchange between agents is weaker. In this configuration, the effect of Q becomes more pronounced. Increasing Q leads to stronger penalization of tracking errors, resulting in faster and more synchronized responses. When $Q = 10I$, the convergence is clearly accelerated compared to $Q = 0.1I$, where agents exhibit a delayed alignment and a wider spread in transient behavior.
With higher coupling $c = 2$, the network becomes more responsive and the inter-agent coordination improves significantly. Here, even the weakest cost weighting ($Q = 0.1I$) yields good synchronization performance. The differences across the three Q values are reduced, although a larger Q still induces slightly more pronounced convergence and tighter grouping in the transient phase.
These results highlight the complementary roles of c and Q in shaping the system dynamics. The gain c governs the flow of information across the network, while Q regulates the control effort and the weight assigned to state deviations. When the coupling is weak, increasing Q is essential to compensate for the limited coordination. Conversely, when coupling is strong, the system can tolerate smaller state penalties without compromising performance.

2.4.5 Effect of R and Energy Consumption Analysis

We investigate the effect of the input penalty matrix R on the control effort and energy usage of the system. We focus on the global input signal $u(t)$, obtained by concatenating all control inputs $u_i(t)$ across agents. This signal is directly produced by the local controllers and represents the physical effort applied to each agent.
To quantify the energy, we compute the integral of the squared input signal over time:

$$E_u = \int_0^T u(t)^\top u(t)\, dt$$

This quantity measures the cumulative energy injected by the control actions during the simulation window $[0, StopTime]$.
We vary the parameter R across three values: $R = 0.1$, $R = 1$, and $R = 10$, while keeping all other base parameters fixed. The input signal and the corresponding energy plots are reported below.

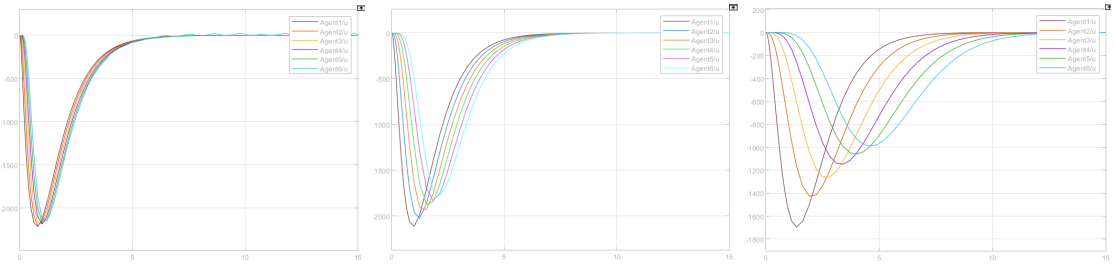


Figure 2.9: Control input signal $u(t)$ for different values of R. Left: $R = 0.1$, Center: $R = 1$, Right: $R = 10$.

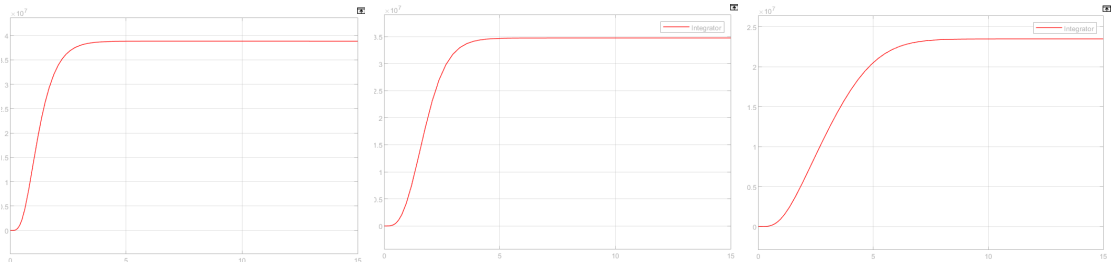


Figure 2.10: Energy of the control input for $R = 0.1$, $R = 1$, and $R = 10$, respectively.

The results show that increasing R leads to visibly lower control inputs and significantly reduced energy. This behavior is consistent with the function of R in the LQR formulation (2.2.4): higher values penalize input effort more heavily, resulting in more cautious and less demanding control actions.
The measured energies are shown in the Table 2.11a. We further explored the joint influence of Q and c by measuring the energy in selected extreme cases from the earlier position tracking analysis, all with fixed $R = 1$ (shown in Table 2.11b):

R	Q	c	Energy
0.1	I	0.5	3.883×10^7
1	I	0.5	3.475×10^7
10	I	0.5	2.348×10^7

(a) Energy for different R values.

R	Q	c	Energy
1	$0.1 \cdot I$	0.5	2.348×10^7
1	$10 \cdot I$	0.5	3.883×10^7
1	$0.1 \cdot I$	2	3.610×10^7
1	$10 \cdot I$	2	3.951×10^7

(b) Energy for different Q, c.

Figure 2.11: Comparison of control energy across different tuning parameters.

These results confirm the mutual influence among the tuning parameters. Increasing Q intensifies the control action to improve state regulation, while increasing c strengthens coordination between agents. Both lead to greater control activity and leads to higher energy consumption . In contrast, increasing R penalizes large inputs, effectively reducing the total energy used.
Notably, the color-coded values in Tables 2.11a and 2.11b show that the same energy level can result either from a high Q (with low R) or a low Q (with high R). This highlights the opposite effect of the two parameters: increasing Q pushes the controller to be more accurate, which increases the energy used, while increasing R discourages strong inputs, reducing the energy. When one increases, the other can compensate, keeping the total energy balanced.

In summary:

- Raising Q or c increases the energy due to tighter regulation and stronger coupling.
- Raising R decreases the energy by limiting control intensity.
- The balance between Q and R determines the trade-off between accuracy and efficiency.

An effective tuning must therefore consider all parameters jointly, depending on whether the application prioritizes precision, energy savings, or a compromise between the two.

2.4.6 Local vs Neighborhood Observers

In this experiment, we compare the system's performance using two observer configurations: the *local* observer, where each agent estimates its own state independently, and the *neighborhood* observer, where each agent fuses information from its neighbors. All other parameters are kept at their default values, and no noise is applied to the system.

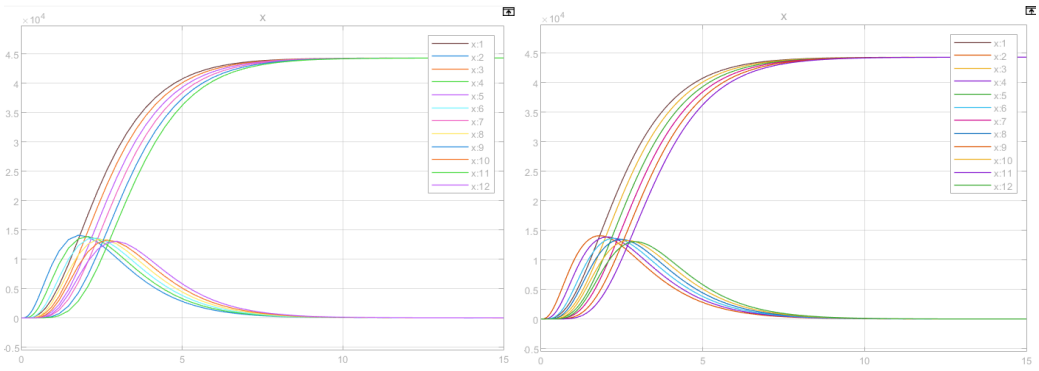


Figure 2.12: Agent trajectories using local (left) and neighborhood (right) observers.

As expected, the trajectories obtained from the two observer structures are identical. This result is a direct consequence of the **separation principle**, which guarantees that in linear time-invariant (LTI) systems, control and estimation can be designed independently without compromising closed-loop stability.
As described in Section 2.3.1, both observers use the same model and gain matrices to estimate the leader's state. In the *Local* structure, agents directly exchange state estimates, while in the *Neighborhood* structure, they receive output measurements and reconstruct the states via their observers. In the absence of noise, both strategies produce the same internal state reconstructions, leading to identical control inputs.
Therefore, the choice of observer has no effect on the global behavior of the system under ideal conditions. Differences may only appear when disturbances or modeling errors are introduced, which will be addressed in the next section.

2.4.7 Analysis Under Noise Conditions

This section investigates the robustness of the distributed control scheme in the presence of measurement noise. All simulations are based on the *neighborhood observer* and use the default control parameters, unless stated otherwise.
Noise is injected into the system as an additive disturbance on the output measurements. The configuration is controlled through the two parameters described in section 2.3.3.
We explore multiple scenarios, varying both the network topology and the distribution of noise across the agents.

Uniform Noise on All Nodes (Line Topology)

In this test, we simulate the effect of measurement noise applied uniformly to the entire system under a *line* topology. Both the leader and all follower agents are subject to identical disturbance levels. The noise configuration is:

- `p.noise_sensitivity` = 0.5
- `p.agent_noise_sensitivity_vector` = [0.5, 0.5, 0.5, 0.5, 0.5, 0.5]

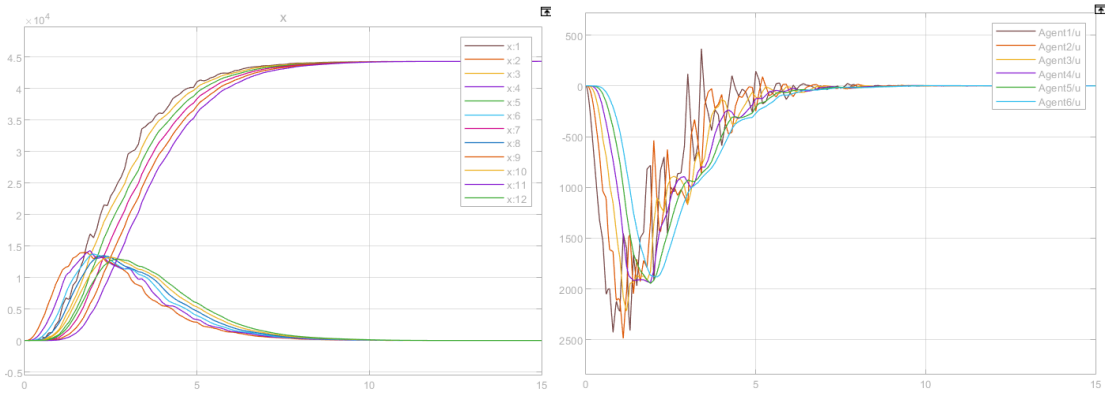


Figure 2.13: Effect of uniform noise (0.5) on all nodes, line topology. Left: agent positions and velocities. Right: control input $u_i(t)$.

Despite the presence of noise, the system remains stable and converges correctly to the desired state. However, both the trajectories and control signals display increased oscillations during the transient, especially in the initial 5 seconds.

The control effort also increases to compensate for the measurement uncertainty. The total control energy for this case is:

$$Energy = 3.506 \times 10^7$$

Comparison: Line vs Fully Connected Topology under Partial Noise

This scenario compares the behavior of two different topologies (*line* and *fully connected*) under the same noise configuration. In both cases, the leader node is noise-free, while all follower agents are disturbed. The settings are:

- `p.noise_sensitivity` = 0.0
- `p.agent_noise_sensitivity_vector` = [0.5, 0.5, 0.5, 0.5, 0.5, 0.5]
- $c = 3.4271$, i.e., the minimum gain required to stabilize the fully connected topology.

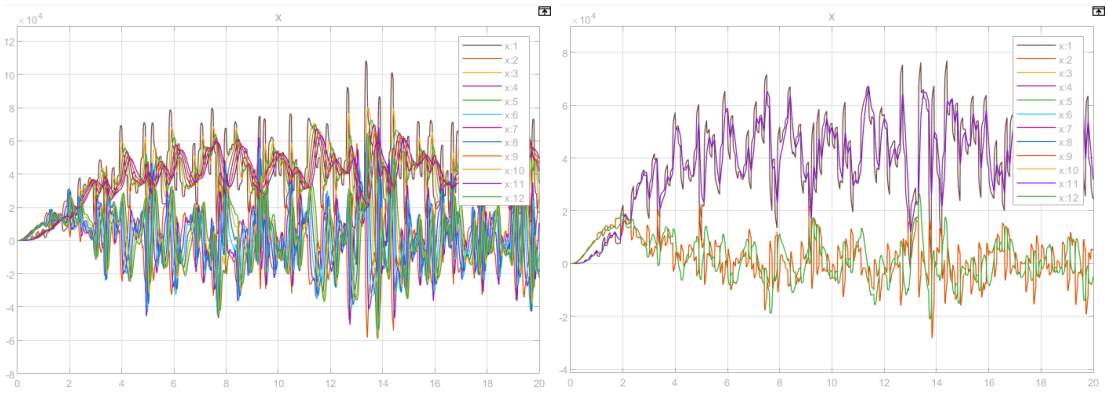


Figure 2.14: Comparison of agent trajectories under noise. Left: line topology. Right: fully connected topology.

In the line topology, *the agents closest to the leader exhibit larger oscillations due to noise*, while those further away show progressively smoother trajectories. This behavior is a consequence of the sequential communication structure: the first agent receives direct information from the leader but is also directly exposed to noise, which heavily perturbs its estimate. The subsequent agents rely on information coming from their predecessor, which already includes a degree of noise attenuation thanks to the dynamics of the system and the consensus mechanism. As a result, *the noise influence tends to dissipate along the chain*, leading to a gradient of decreasing oscillations from the first to the last agent.
An important aspect of this test is that the *leader node is kept noise-free*. While this improves the quality of the reference signal, it also increases the discrepancy between the leader and the noisy followers. ***Since the control action depends on the difference between estimated and measured states, this mismatch amplifies the transient oscillations as agents attempt to track a "perfect" but unperturbed leader.*** The contrast between noisy agents and a clean reference causes greater instability during convergence.
In contrast, the *fully connected* topology shows a more coordinated behavior under the same noise conditions. Thanks to its dense structure, each agent receives information from all others, enabling better averaging and smoothing of noisy estimates. Even if the

leader remains noise-free, the impact of follower disturbances is minimized through multiple communication paths.
This comparison highlights two critical factors that influence robustness to noise: the communication topology and the presence (or absence) of noise on the leader. A denser graph provides natural filtering, while balancing noise levels across the network avoids asymmetric dynamics.
In all the following simulations, the leader noise remains disabled to isolate and evaluate the effect of follower disturbances under different configurations.

Noise Applied to Individual Agents in Line Topology

In this analysis, we evaluate how the relative position of a noisy agent affects the overall system behavior under a *line* communication topology, with coupling set to $c = 0.5$. Two distinct configurations are considered:

- `p.noise_sensitivity` $= 0$ (no leader noise),
- `p.agent_noise_sensitivity_vector` $= [0.5,\ 0,\ 0,\ 0,\ 0,\ 0]$: noise only on Agent 1,
- `p.agent_noise_sensitivity_vector` $= [0,\ 0,\ 0,\ 0,\ 0,\ 0.5]$: noise only on Agent 6.

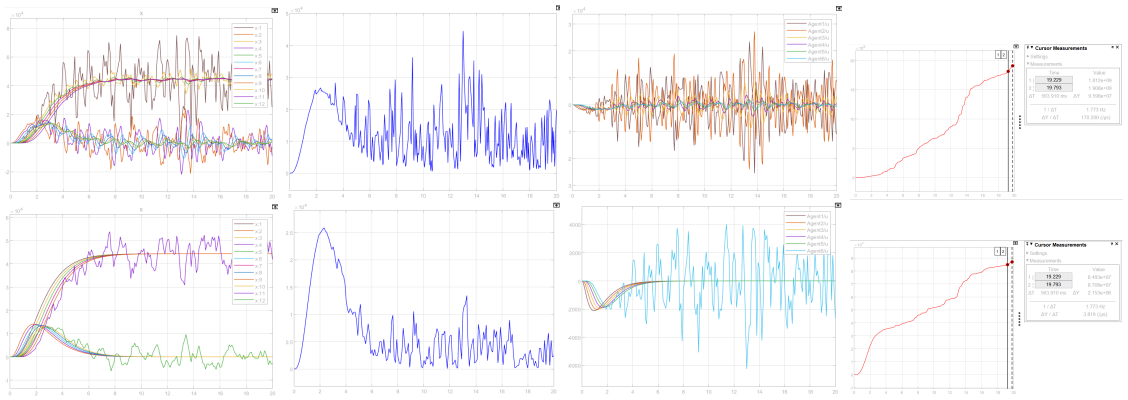


Figure 2.15: Top row: noise on Agent 1. Bottom row: noise on Agent 6. Left to right: Position and velocity, Position tracking error, Control input, Control energy.

When noise is introduced on **Agent 1** (the closest node to the leader) the plots show significant oscillations in its position and velocity trajectories, which also propagate to its neighboring agents. Despite this local origin, the noise spreads along the communication path, requiring all controllers to mediate. This leads to:

- *persistent and large fluctuations in the position tracking error;*
- high control input across all agents, as each contributes to stabilizing the disturbed node;
- an overall **increase in control energy**, since global compensation is needed.

On the other hand, when the same noise is applied to **Agent 6** (the farthest node from the leader) the disturbance remains highly localized. Only the trajectory of Agent 6 is affected, while the remaining agents maintain coordinated behavior. In this case:

- the position and velocity of other agents are unaffected;
- the position tracking error has lower peak amplitudes;
- only Agent 6 shows a high control input, while the others remain stable;
- the **total energy consumption is lower**, as fewer agents are involved in disturbance rejection.

This comparison highlights two important aspects:

1. In a sequential topology, nodes closer to the leader influence the rest of the network more strongly, and their noise propagates more widely.
2. The position of the disturbance affects how many agents must react and how much energy is required. Localizing noise on peripheral nodes results in more confined and energetically efficient compensation.

Noise Applied to Individual Agents in Ring Topology

In this analysis, we examine how the impact of noise depends on the node position within a circular communication structure. We consider a *ring* topology with fixed coupling gain $c = 4.5964$, and compare two noise scenarios:

- `p.noise_sensitivity` $= 0$ (leader unaffected),
- `p.agent_noise_sensitivity_vector` $= [0.5,\ 0,\ 0,\ 0,\ 0,\ 0]$: noise on node 1 only;
- `p.agent_noise_sensitivity_vector` $= [0,\ 0,\ 0,\ 0,\ 0,\ 0.5]$: noise on node 6 only.

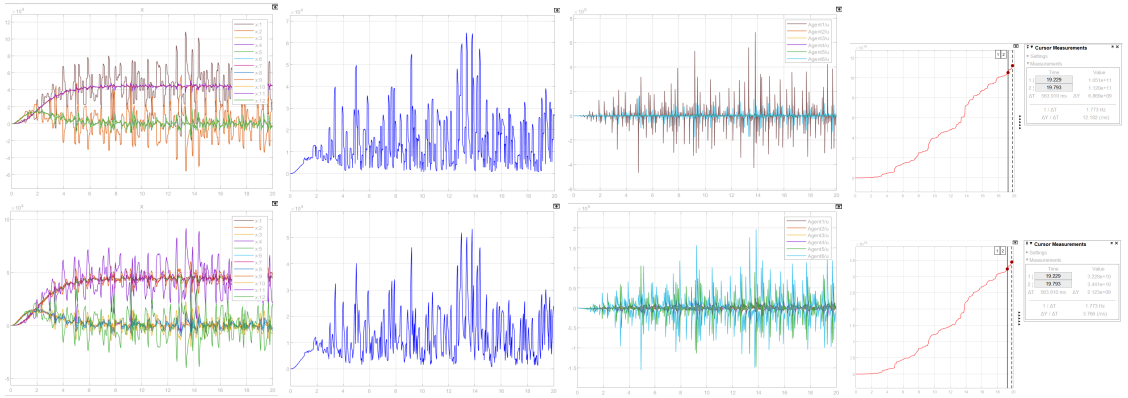


Figure 2.16: Top row: noise on Agent 1. Bottom row: noise on Agent 6. Left to right: Position and velocity, Position tracking error, Control input, Control energy.

The position and velocity trajectories appear similar in both cases, but the effect of noise is distributed differently across the agents. In the first case (noise on node 1), the disturbance is better attenuated by the rest of the network due to the circular topology. The resulting oscillations are mainly confined to Agent 1, while the others manage to maintain a more stable behavior. In contrast, when noise is injected at node 6, the oscillations propagate more clearly and with higher amplitude to neighboring agents. This suggests that the position of the noisy node within the ring still plays an important role.

The position tracking error is almost identical for the two cases, unlike in the line configuration, confirming that the ring structure equalizes the influence of each node on the global coordination.
As for the control input $u(t)$, the disturbance affects mostly the noisy agent in both scenarios, but when noise is applied to node 1, the system reacts more strongly overall. This results in more pronounced control actions across several agents.
Consequently, the total energy consumed is significantly higher—an order of magnitude greater (when the noise is on node 1 compared to node 6). This confirms that although the ring topology spreads the disturbance more uniformly, the control effort still depends on the relative importance of the node in the network flow and on how early the disturbance is injected into the system.

Noise Applied to Individual Agents in Fully Connected Topology

We now analyze the behavior of the system under a *fully connected* topology, using a coupling gain $c = 3.4271$. Two configurations are tested:

- `p.noise_sensitivity` $= 0$ (no leader noise);
- `p.agent_noise_sensitivity_vector` $= [0.5,\ 0,\ 0,\ 0,\ 0,\ 0]$: noise only on Agent 1;
- `p.agent_noise_sensitivity_vector` $= [0,\ 0,\ 0,\ 0,\ 0,\ 0.5]$: noise only on Agent 6.

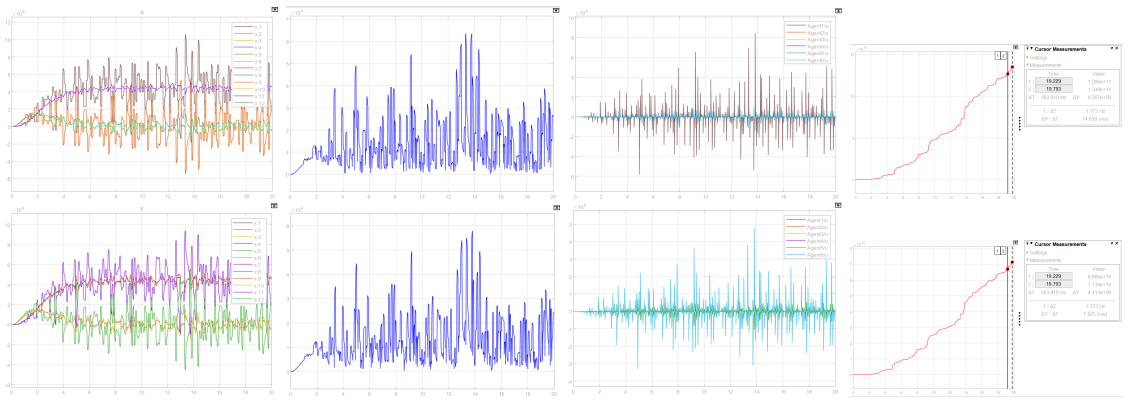


Figure 2.17: Top row: noise on Agent 1. Bottom row: noise on Agent 6. Left to right: Position and velocity, Position tracking error, Control input, Control energy.

Thanks to the dense interconnection structure, the system displays high resilience to localized disturbances. In both scenarios, the *position and velocity* plots are very similar, with visible oscillations only in the noisy agent. The others remain nearly unaffected.
When noise is applied to Agent 1, which is the only agent directly connected to the leader, the oscillations in its position and velocity are significantly amplified. However, all other agents show identical, low-level oscillations due to consensus behavior and effective information distribution. This results in almost perfectly overlapping curves for the remaining agents.
In contrast, when the noise is placed on Agent 6, its effect is still localized but the influence is less uniformly distributed: the remaining agents exhibit slightly different responses, though still limited.

The *position error* is nearly identical across both tests, confirming that the network compensates effectively regardless of noise location. The *control input* u is elevated only for the disturbed agent, while it remains nearly flat for the others.
Finally, the *total energy consumption is significantly higher* (about an order of magnitude) when noise affects Agent 1, compared to Agent 6. This reflects the higher control effort required to stabilize the most influential node in the network.