SC42101 Networked and Distributed Control Systems

Assignment 2 - Lectures 3 and 4

Instructions:

- The assignments are individual. You may consult and discuss with your colleagues, but you need to provide independent answers.
- You may use Matlab/Python/Julia to solve the assignment. Many questions require you clearly to do so (all practical problems in this set).
- Provide detailed answers, describing the steps you followed.
- Provide clear reports, typed, preferably using LaTex.
- Submit the reports digitally on Brightspace as indicated on the lectures.
- Include your code, for reproducibility of your results, in your Brightspace submission, and possibly also through a link on your report.
- The code will *only* be used to verify reproducibility in case of doubts. The grading will be performed based on the results described in the report.

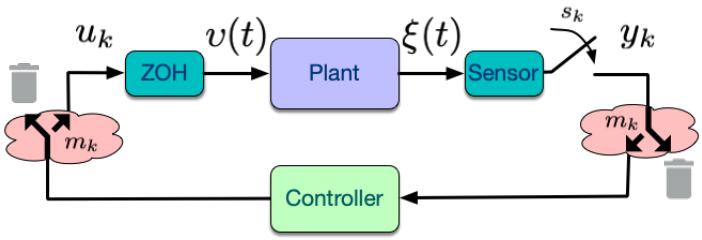

Figure 1: Schematic of a NCS including delays and packet-losses.

Consider a sample-data networked control system (NCS), as depicted in Figure 1. Assume that the plant dynamics are linear time-invariant given by:

$$\dot{\xi}(t) = A\xi(t) + Bv(t),$$

where the control signal v is a piece-wise constant signal resulting from the application of a controller in sampled-and-hold fashion, i.e. $v(t) = u_k,\ t \in [s_k,\ s_{k+1})$.

The system matrices are given by:

$$A = \begin{bmatrix} 0.3 + a - b & 0.5 - c \\ 0 & 1 \end{bmatrix}, \; B = \begin{bmatrix} 0 \\ 1 \end{bmatrix},$$

where a is given by the first digit, b by the 3rd digit, and c by the last digit of your student ID number. **This is the same plant from Assignment 1.** We shall call the control system controlling this plant *System 1*.

We revisit the problem in Questions 5 of Assignment 1. Considering the system with a constant sampling interval $h = s_{k+1} - s_k$ for all k, and assuming no delay is present $\tau = \tau^{sc} + \tau^{c} + \tau^{ca} = 0$. We consider a scenario where another control system, *System 2*, shares the network with System 1. System 2 has as system matrices $A_2 = 1/3A$ and $B_2 = B$, and it is implemented with a sampling interval three times the sampling interval of System 1. This means that whenever System 2 communicates it may induce packet drop-outs in System 1 or System 2. This situation is depicted in Figure 2.

Employ for System 1 the stabilizing feedback matrix $\bar{K}$ from Assignment 1, and the same stabilizing feedback matrix $\bar{K}_2$ for System 2 you used in Question 5 of Assignment 1. For Question 1 you have to analyze both the **to-hold** and **to-zero** approaches. Question 1 assumes a more general model than the one in Question 5 of Assignment 1, and Question 2 assumes a stochastic model of this situation, requiring a different analysis approach.

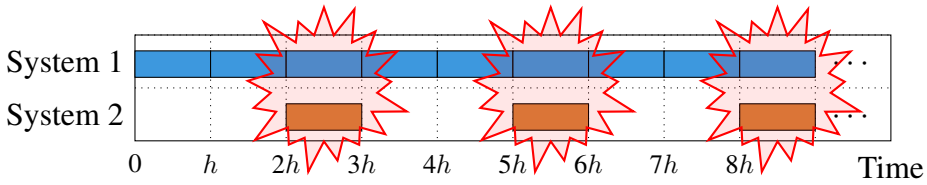

Figure 2: Gantt chart depicting the network time slots used by Systems 1 and 2 for Questions 1–3. The red starbursts indicate packet collisions, potentially causing a loss in the transmission of System 1 and/or System 2.

In this model we make the case studied in Assignment 1 more general. We assume now that whenever there is a collision, System 1's packet **may or may not be lost** and System 2 always looses every other packet. Note that we also assume that if System 2 looses a packet, System 1 may still loose its packet too.

Question 1: (10p)

1. (3p) Create an ω-automaton that models this behavior.
2. (2p) Write down the equations of the switched system describing the dynamics of the received samples using this network model.

3. (2p) Provide LMIs to verify stability of your closed-loop system.

4. (2p) For what range of sampling intervals h is System 1 stable in this scenario? What is best, to zero or to hold?

5. (1p) Is the range you have obtained larger or smaller than the one in Question 1? Comment on the differences: whether they are expected or not, and why.

In this model, compared to those in Assignment 1, we add probabilities. Assume that whenever there is no collision, the probability of each system losing a packet is independent and equal to 0.01 (1%); while when there is a collision, the probability of only System 1 losing a packet is 0.49 (49%) and the same holds for System 2, and there is a 0.02 probability of both of them loosing their packet. For this question consider **only** the to-hold case.

Question 2: (10p)

1. (3p) Create a Markov Chain that models this behavior.

2. (3p) Write down the equations of the Markovian Jump Linear System describing the dynamics of the received samples using this network model

3. (2p) Choose a stochastic stability notion and provide LMIs to verify stability of your closed-loop system.

4. (2p) Comment on the differences between the range of stabilizing sampling intervals h that you obtained now with respect to the ones in Questions 1 and 2.

Next we ignore System 2 and consider the case when delays are present in the networked system, but System 1 is no longer subject to packet drop-outs. Assume that the system is affected by a **constant** small delay $\tau \in [0, h)$, and it is controlled with the same static controller you designed.

Question 3: (10p)

1. (4p) Employ the so-called Jordan form approach (as in Lecture 4) to construct a polytopic discrete-time model over-approximating the uncertain exact discrete-time closed-loop NCS dynamics. By graphically inspecting the uncertain entries of the obtained matrices, what is the smallest number of vertices the polytope needs?
   Note: This approach would allow you to also deal with time-varying delays.

2. (2p) Perform stability analyses, solving the relevant LMIs resulting from the previous question when the polytopic over-approximation of item 1. Employing this analysis, produce a plot illustrating combinations of (h, τ) as in Question 2 of Assignment 1.

3. (2p) Refine the plot of the previous question, by employing a smaller polytope that over-approximattes the uncertain system, but with more vertices (thus requiring the solution of more LMIs).

4. (2p) Compare the resulting plot with the one from Assignment 1 and discuss any possible differences and the possible causes for such differences.

Finally we go back to the case with no delays to design an event-triggered controller.

Question 4: (10p)

1. (1p) Consider your system being controlled by the first static controller you designed for your continuous-time system. Design a quadratic event-triggered condition $\phi(\xi(t), \xi(s_k)) \leq 0$ for the system to guarantee global exponential stability of the closed-loop.

2. (4p) Simulate the resulting closed loop for various values of the σ parameter controlling the guaranteed performance of the closed-loop. Simulating for multiple initial conditions for each value of σ, compare the average amount of communications/sampling produced by each of the systems (different σ values) on a predefined fixed time-length of the simulations.

3. (1p) Let h^*_{avg} be the average inter-sample time for the value of σ in Item 2 that resulted in the lowest amount of communications/sampling. Does periodic sampling with $h = h^*_{\text{avg}}$ stabilize also your closed-loop system?

4. (2p) Consider now that the system is affected by a bounded disturbance d, i.e. $\dot{\xi}(t) = A\xi(t) + B\upsilon(t) + d(t)$, $\|d(t)\| < D$ for all t. Simulate again the closed-loop ETC system, now affected by the disturbance, for a sufficiently long time. Simulate for at least three different realizations of the disturbance. Discuss what you observe now happening with the system trajectory and the inter-sample times. You may choose the disturbance as you wish as long as it is persistent, e.g. $d(t) = \frac{D}{\sqrt{2}}[1\ 1]^T \sin(t)$.
Hint: You may want to do some simulations starting close to the origin (or at the origin itself).

5. (2p) Modify the triggering condition to take the form $\phi(\xi(t), \xi(s_k)) \leq \epsilon$, for some $\epsilon \in \mathbb{R}^+$ of your choice. Simulate again for the same initial conditions as in the previous question. How do the trajectories and inter-sample times differ now from the previous case when ϵ was zero?

Hint: To simulate event-triggered control in continuous-time, you need to detect the triggering even while solving the ODE. The naïve way is to simulate at a very fine discrete-time rate, but this ends up being imprecise and computationally inefficient. ODE solvers in Python, Matlab, and Simulink, for example, have event-checking mechanisms that you can set up to, e.g., stop simulation at an state-dependent event. You can use this to stop simulation, update the control action, and restart the simulation in a loop. Here are some pointers for event detection:

- **Matlab**: any ODE solver, e.g., `ode45`, has an optional `options` input; look up `odeset` for implementation details (e.g., type `doc odeset` in the Matlab prompt and look for the"events" Section.)
- **Python**: similar to Matlab, `scipy` offers event-handling through an optional `events` argument. See the documentation for implementation details.
- **Simulink**: The Triggered subsystem is the recommended block to implement an event-based update of a signal.

If you are familiar with hybrid systems, particularly the jump-flow formalism, you may also consider the Matlab/Simulink toolbox HyEQ from USCB.