MULTIVARIABLE CONTROL AND COORDINATION SYSTEMS (EE-477)

Case study:

Path tracking for an automated vehicle

`http://react.epfl.ch`

Denis Gillet, Ezequiel Debada, Matin Macktoobian
Lausanne, EPFL, 2020

Session 2

Control and observation

In this session we tackle the design of an LQR controller and apply them to our path tracking problem. The general idea is using the *discrete-time linear model* of the system to design an LQR controller that will be used to control the *nonlinear system*. Then, the LQR will be complemented with an observer that will use the model of the system to reconstruct one non-measured state.

2.1 Provided files

- `LQR_observer.slx`. A simulink files with some of the blocks needed to simulate the system + controller + observer combo.
- `ex2.m`. The class whose methods have to be completed. A description of every function to be completed is included within the file.
- `exercise2_LQRandObserver.m`. The Matlab script which sets and runs the required simulations. Modify its content only if you want to run other experiments in addition to the ones proposed.

  Note: Although you will be modifying methods in the `ex2.m` class, it is the `exercise2_LQRandObserver.m` script the one that you have to execute to run the proposed experiments and simulation.

- `utilities.m`. The class which gathers auxiliary functions that will be used in the scripts provided during the case study sessions. There is no need to review its content unless you want to make use of some of the functions included therein.
- `circle, path_1, path_2, path_3` mat files containing different paths to track that can be used in the experiments.

2.2 Exercises

Transfer your solutions from the first session:

1. Place within the Matlab's working directory the class `ex1.m` with your solutions to the exercises proposed in the first case-study session.

LQR design:

2. Complete the method `getLQRCostFunctArrays` for it to return the matrices

$$Q_1 = \begin{pmatrix} 0.00001 & 0 & 0 & 0 & 0 \\ 0 & 50 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0.5 \end{pmatrix} \qquad Q_2 = \begin{pmatrix} 1 & 0 \\ 0 & 0.00002 \end{pmatrix} \tag{2.1}$$

   defining the cost function of the optimal control problem.

3. Complete the method `getLQRGain` where the LQR gain will be calculated by solving the Ricatti equation iteratively.

   Note: The number of iterations required to obtain S_∞ will be obtained through trial and error. It must be sufficiently large so that S converges. This condition can be observed, for instance, by keeping track of the singular values of S (which would as well converge with S).

State observer design and implementation Once the LQR controller has been calculated, we will design an observer. To do so, we first need to change our assumption concerning the system's output equation since, if we continue considering the system's output to be

$$\mathbf{y} = \mathbf{x} \tag{2.2}$$

all states would be available and thus an observer would not be necessary[1].

For this reason, the first step consists in choosing an *alternative* system's output $\mathbf{y}'$, given by the equation

$$\tilde{\mathbf{y}}'(k) = C'\tilde{\mathbf{x}}(k) = C'\tilde{\mathbf{y}}(k) \tag{2.3}$$

so that the system's output become

$$\tilde{\mathbf{y}}'(k) = [x_1(k), x_2(k), x_4(k), x_5(k)]^\intercal. \tag{2.4}$$

Then, our observer will have to generate the estimated state vector $\hat{\tilde{\mathbf{x}}}(k)$ given the output $\tilde{\mathbf{y}}'$.

4. Complete the method `alternativeSystemsOutputEquation`, which returns the array C' defining the system's alternative output equation we will use to justify the use of an observer.

5. Complete the method `checkObservability` where you will assess whether the system is observable or not.

   Note: To calculate the observability matrix, you can use the matlab command `obsv`.

   Note: Here you need to take into account the *alterantive* output equation.

6. Complete the method `getObserverGain` where the observer gain is calculated. Start by setting the poles of the observation loop as the 99.9% of the poles of the closed control loop.

   Note: Remember that the eigenvalues of the control closed loop are given by `eig`$(\Phi - \Gamma * lqr_K)$

   Note: To calculate the observer gain, use the matlab command `place`. Read the help information and notice that the function is meant to be used to calculate control gains. That is it returns gains to impose certain poles assuming that the closed loop dynamics is given by (A-B*K). However, we want to use to to place the poles of the observer. Check and compare the observer close loop, build a parallelism with the control case, and introduce the needed modifications in the matrices given to the `places` method.

7. Complete the method `getObserverInitialState` that returns the state that the observer will initially assign to the states it has to observe. You can choose the first guess of the observer freely and it may differ from the system's real initial state.

[1] This is not totally true as an observer might as well be useful in scenarios where the full state vector is available and the system's initial state is unknown and/or the signals are affected by noise.

2.3 Simulink diagram and simulation

Once we have the LQR controller and the observer designed, you will complete one Simulink diagram allowing us to close the control loop in two different ways. A draft schematic that illustrates the task is provided in Fig. 2.1. The idea is including a switch in the scheme allowing us to close the control loop with the initially assumed system's output $\mathbf{y}$ or the alternative output $\mathbf{y}'$.

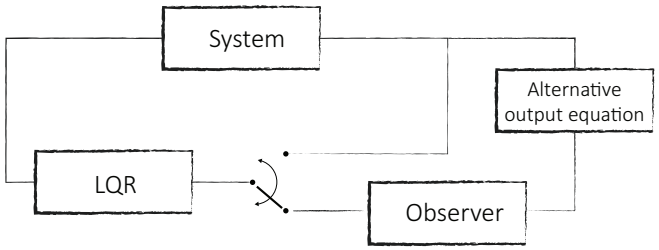

Figure 2.1: Draft of the targeted simulink diagram.

8. Complete the Simulink diagram `LQR_observer.slx` given the information provided above.

   Note: Several *subsystem* blocks are included in this Simulink file. Explore all of them to identify the ones you need to complete.

   Note: The observer is to be implemented using a state-space block. A example of how this should be done is provided in exercises 6.6.1-3.

9. Complete the method `getObserverImplementationArrays` where the arrays *AObs*, *BObs*, *CObs*, and *DObs* needed to implement the observer with a state-space block are calculated.

2.4 Questions

1. Report the value of the LQR gain for the proposed Q_1 and Q_2, as well as the evolution of the singular values of S obtained while solving the Riccati equation.
2. Report the explicit value of the observation close loop poles, and the observer gain resulting from it.
3. Provide a screen shot of your final Simulink scheme and the submodules you had to complete.
4. Report the simulation results for the proposed values.
5. The proposed value of Q_1 intends to assign very little importance to the error deriving from x_1. Could you explain why doing this might make sense?
6. Propose a different pair Q_1 and Q_2 such that heading deviation error is highly penalized compared to the other states and report simulation results where the impact of doing so can be clearly observed.
7. Is the proposed placement for the observation close loop poles appropriate? why? If not, propose a new set of poles to improve the observation performance.