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

Introduction.

In this case study corresponding to the course *Multivariable Control and Coordination Systems*, we aim at putting into practice the concepts and methods presented in the lectures, as well as at providing a quick glance at how Matlab-Simulink can be used to simulate dynamical systems, implement control strategies, and assess control results.

One exercise is proposed, which are inspired on the challenges that automated vehicles must address. Specifically, we will address a simplified version of the so-called path tracking problem. The path tracking problem refers to the challenge of generating control inputs such that an automated vehicle tracks a reference path accurately.

Due to the exceptional constraints we needs to handle during the this semester, we will carry out exercises related to simulation, control, and observation.

Files: In every case-study session you will be provided with the description of the exercise, a Matlab script `exercise#_*.m` (which does not need to be modified), a Matlab class `utilities.m` (common to all exercises) gathering some auxiliary functions the exercise scripts will use, a Matlab-class named `ex#.m` which includes the methods that you have to complete, and some Simulink files you also have to be completed.

Evaluation At the end of the course we expect you to submit a zip file (named `LastName-FirstName.zip`) with a total of 6 items:

- one pdf document report.
- two `ex#.m` files, one per session, containing the solution to the proposed exercises,
- and three Simulink files with the solution you implemented.

You are allowed to work in small groups (up to three people max) and we encourage you to discuss ideas to overcome the difficulties you might find. However keep in mind that the report **must** be written individually and we expect to find slightly different simulation results and discussions in each of them. Note as well that assessing simulation results is crucial and a bad/good discussion can invalidate/validate good/bad results.

Important: Even though the Matlab and Simulink files are requested, **the evaluation of the solutions will be solely based on the pdf report**. Meaning that references in the pdf document to the Simulink files or Matlab code such as *'as can be seen in the attached Simulink file'* or others of the sort will be ignored. Do not expect us exploring the Matlab files to find the solutions. Everything you want us to take into account when grading the exercises must be included in the report. The only purpose of requesting the Matlab files is guaranteeing the uniqueness of your solution.

Finally, note that this is the new adaptation of the case study we used in the past. Thus, typos, bugs, and inconsistencies may be found throughout the handout and code. We kindly ask you to point out any flaw you find so that we can improve and update the material of this case study.

Starting point Last but not least, you are expected to arrive to the first session of the case study with the solution of the modeling exercise proposed the 28th of September.

Session 0 Modeling, linearization, and discretization

Automated vehicles must follow paths that the vehicles' own decision-making layers generate to enable the vehicle to stay in its lane, change lanes, cross intersections, avoid collisions, etc. Some path-tracking methods found in the literature tackle the challenge by exploiting a kinematic model of the vehicle that describes the vehicle's position w.r.t. to the path that is to be tracked.

Consider a reference path defined by a function $\pi : \mathbb{R} \to \mathbb{R}^3$, that, given a distance $s_\pi \in \mathbb{R}$ along the path, outputs a tuple $(x_\pi(s_\pi), y_\pi(s_\pi), \theta_\pi(s_\pi))$ representing the x-y position $(x_\pi(s_\pi), y_\pi(s_\pi))$, and orientation $\theta_\pi(s_\pi)$ at s_π. Then, given a point $P_r \in \mathbb{R}^2$ positioned in the middle of the vehicle's rear bumper, and the point $P_{r,\pi} \in \mathbb{R}^2$ corresponding to the projection of P_r in the path π, the position of a vehicle could be described by:

- the distance $s \in \mathbb{R}$ along the path corresponding to the vehicle's projections,
- the so-called *lateral error* $d \in \mathbb{R}$, defined as $d = ||P_r - P_{r,\pi}||$, showing the distance to the path, and
- the so-called *heading error* $\theta_e \in \mathbb{R}$, defined as $\theta_e = (\theta - \theta_\pi)$, showing the difference between the vehicle's yaw angle θ and the path's orientation θ_π at $P_{r,\pi}$.

The coordinates listed above are illustrated in Fig. 1.

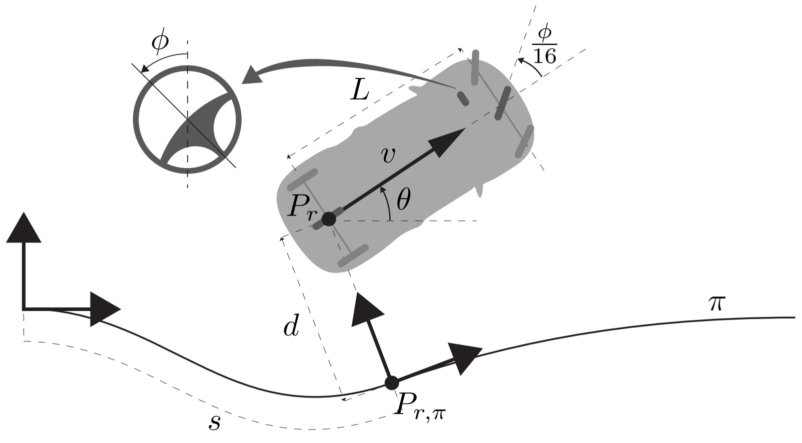

Figure 1: Coordinate system.

Representing by $\phi \in \mathbb{R}$ the position of the steering wheel (measured in radians), and assuming our control inputs to be the longitudinal speed reference $v_{ref} \in \mathbb{R}^+$ and the steering wheel's position reference $\phi_{ref} \in [-4\pi, 4\pi]$, the vehicle's model can be formulated as follows:

$$\dot{s} = \frac{v\cos(\theta_e)}{1 - d\kappa(s)} \tag{1a}$$

$$\dot{d} = v\sin(\theta_e) \tag{1b}$$

$$\dot{\theta}_e = \frac{v}{L}\tan(\phi/16) - \kappa(s)\dot{s} \tag{1c}$$

$$\dot{v} = \sigma_v(v_{ref} - v) \tag{1d}$$

$$\dot{\phi} = \sigma_\phi(\phi_{ref} - \phi) \tag{1e}$$

where $\kappa(s)$ denotes the curvature of the path at s, and σ_v and σ_ϕ represent the dynamic with which the speed and steering wheel references are followed. For the sake of further simplicity, from now on we will assume that the path has a constant curvature, i.e. $\kappa(s) = \kappa$.

The states and control inputs of our system are

$$\mathbf{x} = [x_1, x_2, x_3, x_4, x_5] = [s, d, \theta_e, v, \phi]\,, \tag{2}$$

$$\mathbf{u} = [u_1, u_2] = [v_{ref}, \phi_{ref}]\,. \tag{3}$$

Moreover, unless stated otherwise, we will consider the output of the system to be

$$\mathbf{y} = \mathbf{x}. \tag{4}$$

0.1 Questions

Answer the following questions:

- Rewrite the non-linear model in the standard form $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$.
- Given the chosen state space, why it wouldn't be appropriate linearizing the system around a nominal *point*.
- Calculate the *nominal trajectory* representing the constant-speed perfect tracking situation (that is, a situation where the vehicle drives perfectly on the path and at a constant speed v_{ref}) for a path of constant curvature κ_{ref}.
- Linearize the system around such a nominal trajectory.
- Discretize the system using Euler approximation.

Session 1 Model implementation and simulation

This exercise focuses on observing and discussing the similarities and differences between the continuous-time non-linear, continuous-time and linear, and discrete-time and linear models of the systems. We **do not** tackle the control problem yet, but will based the comparison on the differences we observe on the simulation results of each of the three models when a common input sequence (applied in open loop) is used for all of them.

Throughout the exercise, we must make use of a reference path simply because our vehicle's motion model is formulated w.r.t. one and thus it is a required step to simulate the vehicle's motion. Nonetheless, don't let the existence of a reference path confuse you. The reference path is considered for implementation purposes, but, strictly speaking, **we are not tackling the control problem (i.e. path tracking) yet**.

The proposed exercises aim to drive you through (i) the implementation of the models in Simulink, (ii) the application of control sequences in open loop, (iii) and the discussion of the observed differences.

1.1 Provided files

- `open_loop_experiments.slx`. A Simulink file with some of the blocks needed to implement and simulate the models. In this file you will implement the continuous time non-linear model, continuous time linear model, and discrete time linear model.
- `ex1.m`. A class whose methods have to be completed. A description of every function to be completed is included within the file.
- `exercise1_Linearization.m`. A Matlab script that sets up and runs the required simulations. Modify its content only if you want to run other experiments in addition to the ones proposed.
- `utilities.m`. A class that gathers auxiliary functions that will be used in the scripts provided during the case study sessions. There is no need to reviewing its content.
- `circle, path_1, path_2, path_3` mat files containing different paths to track that can be used in the experiments.

1.2 Exercises

1. Complete the method `getSystemParameters` in `ex1.m` so that it returns a **column vector** containing, in the shown order, the parameters of the system: $\kappa(s) = 1e^{-10}$, $L = 4$, $\sigma_v = 1$, $\sigma_\phi = 5$, $v_{ref} = 5$.
2. Complete the method `getWorkingTrajectory` which should return the nominal trajectories of the control inputs and states (i.e. the signals $\bar{\mathbf{u}}(t)$ and $\bar{\mathbf{x}}(t)$ you calculated in Session 0) in a format compatible with the Simulink block `from-workspace`.

   Note: You can always find additional information regarding Simulink blocks and their configuration options by double-clicking on the Simulink block of interest.

3. Complete the method `getLinealModelArrays` in `ex1.m` which should return the matrices A, B, C, D you calculated in Session 0.
4. Complete the method `getDiscreteLinearModel` which should return the matrices Φ, Γ describing the discrete-time linear model of the system and calculated following two different methods: Euler approximation (which you should have solved in Session 0), and using the Matlab command `c2d`.

   Note: Optionally, if time allows, you could as well implement the method based on the Taylor-series approximation of the exponential matrix (that is using the auxiliary array Ψ as described in section 2.2.) and compare it with the results of the two other methods mentioned above.
5. Implement the system's non-linear model using the block `Non linear model - continuous time` in the Simulink file.

   Note: Remember that the inputs of Simulink blocks are generally referred as `u()`. Thus, if you include the system's control input u_i as the n-th input of a Simulink block, you would have the signal u_i available in $u(n)$.

   Note: The curvature of the path κ in the equations must be taken from the `u(8)` (which contains the curvature of the reference path given the value of x_1) instead of using the corresponding fixed parameter. This is due to the fact that, even though we build the model considering a path of constant curvature we could use paths with more general geometry, and consider the curvature to remain constant during the short simulation time steps.
6. Complete the Simulink diagram so that it applies the same sequence of control inputs to the three models of the system.

   Note: The sequence of control inputs and states you receive and send through the *from/to workspace* blocks must be the real ones (i.e. $\mathbf{u}(t)$ and $\mathbf{x}(t)$) but the linear models require $\tilde{u}$ and return $\tilde{x}$ so you should transform them before they reach the linear models or sent to the workspace for further analysis.

   Note: To make a fair comparison between the continuous-time and discrete-time linear models, you must use the `zoh` blocks to guarantee that both linear models are receiveing the exact same (sampled) input signal.

   Note: Also keep in mind that Simulink can access all the variables in the workspace at the time the simulation is triggered.
7. Complete the method `getInitialState` which should return the initial state $x(0)$ of the system and the corresponding initial state $\tilde{x}(0)$ of the linear models.

   Note: You are free to choose the system's initial state, but you could start using the value $x(0) = [0, 0, 0, \bar{x}_4(t_0), \bar{x}_5(t_0)]$.
8. Complete the method `getOpenLoopInputSignal` which should return the sequence of inputs to be applied in open loop in a format that is compatible with the *from-workspace* Simulink blocks. The input sequences should be designed so that you are able to observe the differences between models. You should come up with, at least, two input sequences:
   (a) one that makes the simulated motions look the same for the three models,
   (b) one where differences among the three models can be observed.

1.3 Questions

Once the steps above have been completed, please answer the following questions.

1. Provide the arrays Φ and Γ obtained using the Euler approximation method and the Matlab command `c2d`.
2. Provide a screenshot of your implementation of the `Non linear model - continuous time` block.
3. Provide a screenshot of the block diagram implemented in `open_loop_experiments.slx`.
4. Report the simulation results, including the information concerning the initial state you used, the sequence of inputs you applied, and the simulated results.
5. Does the linear model resemble sufficiently well the behavior obtained from the non-liner model? Why?
6. Do the results obtained using the discrete-time linear model resemble those obtained by the continuous-time linear model? What is then interesting about using the discrete-time linear model instead of the continuous-time linear model to design control strategies?