Model Predictive Control - EL2700

State Feedback Control Design

Assignment 1

Division of Decision and Control Systems
School of Electrical Engineering and Computer Science
Kungliga Tekniska Högskolan

Gregorio Marchesini, Samuel Erickson Andersson, Pedro Roque, and
Mikael Johansson

Introduction

During the course, we will be exploring the dynamics of a vehicle as reference dynamics, with which we will experiment with different control approaches. We will be covering approaches ranging from classical control based on pole placements, to more advanced optimization-based techniques, including Linear Quadratic Regulators and the main topic of the course, Model Predictive Control. This assignment is the first of a series of assignments that will bring you from being a rookie pilot in the 125cc category to an expert pilot in the queen of the motorbike racing competitions: the MotoGP.

In this assignment, we will design a discrete-time linear state-feedback controller for a simple vehicle model. To this end, you will compute a discrete-time model that describes the state evolution between sampling instances and use this model to design a linear state feedback controller that achieves the desired control performance.

Although several of the tasks in this assignment can be carried out in MATLAB, you will be working with Python 3 and the CVXPY [1] framework. Later in the course, we will additionally use the CasADi solver to solve more complex nonlinear MPC frameworks. We encourage you to get familiar with the CVXPY framework early in the course by leveraging freely available resources like the short CVXPY course here `https://www.cvxpy.org/resources/short_course/index.html`, as well as a Cheat Sheet available on the Syllabus.

Figure 1: Valentino Rossi bringing his best driving at Mugello in 2017.

[1] Website: `https://www.cvxpy.org/`

Software Preparation

Before proceeding, let's make sure that you have all the software needed for this assignment. Some Python dependencies are needed to run the script from the assignments and projects. You can either install the dependencies on your machine or use the Virtual Machine we created for VMWare (version 16 and above) available on the Syllabus page on Canvas. This virtual machine contains an Ubuntu 20 installation, ready for all the assignments and projects. However, you can also install the dependencies on your system by following these steps:

1. Install Python 3[2] and Pip 3 on your system.
2. Install a Python editor, such as Visual Studio Code[3], Spyder, or PyCharm. Also, feel free to use any other tool of your liking.
3. Install the dependencies by running the command `pip install -r requirements.txt` script with Python 3. Furthermore, for installation of the `slycot` library, please refer to Appendix 1.1.

Design Task

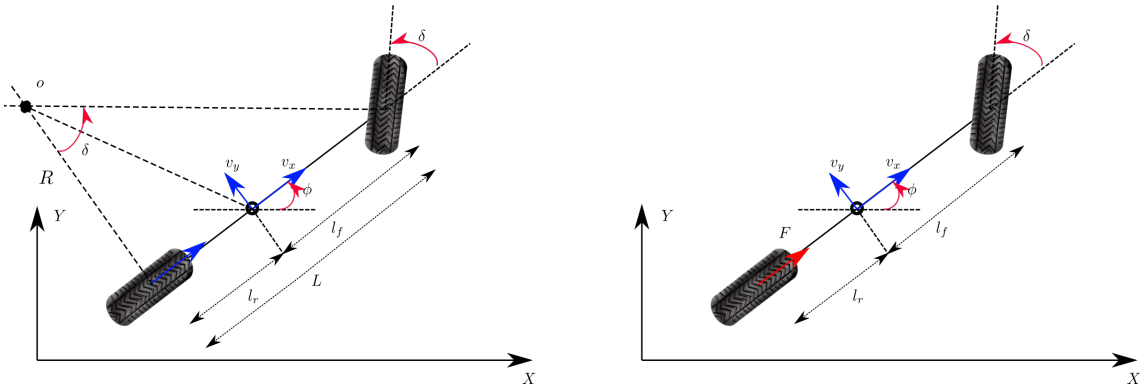

Figure 2: Graphical representation of the kinematic bicycle model

To complete the task, carefully look into the Python files that are part of the assignment and the classes that they implement. The code entry point is `task1.py`, but you will have to complete parts of the `LinearCarModel` and `Controller` classes.

Introduction to the bicycle model. One of the most widely used models for simulating a vehicle at a first control design stage is the so-called kinematic bicycle depicted in Figure 2. The model is physically composed of two wheels separated by a given longitudinal distance $L = l_r + l_f$ where l_f is the distance of the *front* wheel from the center of mass, while l_r is the distance of the *rear* wheel from the center of mass. From the two panels in Figure 2, we list the following variables, which will be clarified in the description of the assignment.

- X and Y represent the inertial position of the center of mass of the vehicle.
- The angle ϕ represents the orientation of the vehicle with respect to the horizontal axis X of the inertial frame.
- v_x and v_y represent the velocity of the vehicle *in the coordinate frame of the vehicle*, *i.e.*, seen from the driver's perspective v_x is the *forward* velocity while v_y is the *lateral* velocity.
- δ represents the steering angle of the front wheel.

[2]Python 3.10 or later: `https://www.python.org/downloads/`
[3]You can install VScode here `https://code.visualstudio.com/`

- L represents the distance between the wheels. The parameters l_f is the distance of the *front* wheel from the center of mass, while l_r is the distance of the rear wheel from the center of mass.
- R is the turning radius of the vehicle.
- o is the center of rotation of the vehicle.
- F is the force on the rear wheel imparted by the engine on the vehicle.

Now that we know the variables that characterize the model, we can start playing with some kinematic relations to get a model of the vehicle.

We will assume that the bicycle you see in Figure 2 is moving with a constant longitudinal speed v_x. At the same time, the front wheel of the vehicle can be rotated to a given steering angle, causing the vehicle to engage in a rotation by pivoting on the rear wheel. For a given steering angle δ, the center of rotation can be found by drawing the intersection of the two lines orthogonal to the wheels (left panel in Figure 2), and we denote this point as o. We call the distance of the rear wheel from the center of rotation o the turning radius R. From basic trigonometry, we can find that

$$L = R \cdot \tan(\delta) \Rightarrow R = \frac{L}{\tan(\delta)} \tag{1}$$

You should remember from rigid body mechanics that if you know the center of rotation of a body and you know the velocity of one of its points, then you can compute its angular velocity (in this case the change of heading $\dot{\psi}$) using the kinematic relation[4]

$$\vec{v_p} = \vec{\phi} \wedge \vec{r_{op}} \tag{2}$$

where $\vec{v}$ represents the velocity vector of the point considered, $\vec{\phi}$ is the angular velocity vector (in this case pointing outward in the figure), and $\vec{r_{op}}$ defines the position of the known point on the rigid body with respect to the center of rotation. In this case, we assume that we have the rear wheel velocity v_x as known (blue arrow on the left panel, the one positioned on the rear wheel), and we know its distance from the center of rotation, from which we get

$$\dot{\phi} = \frac{v_x}{R} = v_x \cdot \tan(\delta) \frac{1}{l_r + l_f}. \tag{3}$$

Using the same kinematic relation, we can similarly derive that the component v_x of the velocity is unchanged at the center of mass, while a small lateral velocity v_y appears at the center of mass

$$v_y = v_x \cdot \tan(\delta) \cdot \frac{l_r}{L} = v_x \cdot \tan(\delta) \cdot \frac{l_r}{l_r + l_f}. \tag{4}$$

Since a constant velocity model would be quite boring, the last ingredient of the model is given by the fact that we can apply a force on the vehicle by the rear wheel (assuming rear traction) equal to F that will cause a longitudinal acceleration as

$$\dot{v}_x = \frac{F}{m} \tag{5}$$

where m is the mass of the vehicle.

With the final assumption that the steering angle is small enough to allow for the common engineering approximation $\tan(\delta) \approx \delta$ (somewhat valid for $\delta < 30°$), we have then derived the simplest possible model of a vehicle

[4]Remember that $\wedge$ denotes the cross product. In the 2D case $\vec{a} \wedge \vec{b} = a \cdot b \cdot \sin(\alpha)$ where α is the angle between the two vectors

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{v}_x \\ \dot{v}_y \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} v_x \cos(\phi) + v_y \sin(\phi) \\ v_x \sin(\phi) + v_y \cos(\phi) \\ \frac{1}{m} F \\ v_x \cdot \delta \frac{l_r}{l_r + l_f} \\ v_x \cdot \delta \frac{1}{l_r + l_f} \end{bmatrix}. \tag{6}$$

with $v_y = v_x \cdot \delta \cdot \frac{l_r}{l_r + l_f}$. At this point, we might ask ourselves if this is enough to provide a meaningful model of a real vehicle. Unfortunately, the answer is negative. Indeed, the relation $v_y = v_x \cdot \delta \cdot \frac{l_r}{l_r + l_f}$ allows for an immediate change of lateral velocity corresponding to a sudden change of steering angle δ, which is not very realistic in most of the situations. But fear not, because we can obtain a much better model by simply considering the *steering rate* as an input to the system instead of the *steering angle*. Taking the first time derivative of the equations where δ appears, we get

$$\dot{v}_y = (\dot{v}_x \cdot \delta + \dot{\delta} v_x) \cdot \frac{l_r}{l_r + l_f}$$
$$\dot{\phi}_y = (\dot{v}_x \cdot \delta + \dot{\delta} v_x) \cdot \frac{1}{l_r + l_f}$$

such that the new model we obtain looks now like

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\phi} \\ \dot{v}_x \\ \dot{v}_y \\ \dot{r} \\ \dot{\delta} \end{bmatrix} = \begin{bmatrix} v_x \cos(\phi) - v_y \sin(\phi) \\ v_x \sin(\phi) + v_y \cos(\phi) \\ r \\ \frac{1}{m} F \\ (\dot{\delta} v_x + \dot{v}_x \delta) \cdot \frac{l_r}{l_r + l_f} \\ (\dot{\delta} v_x + \dot{v}_x \delta) \cdot \frac{1}{l_r + l_f} \\ u \end{bmatrix} \tag{7}$$

where we have called $\dot{\phi} = r$ for simplicity, and we have now changed δ from being an input to being a state of the system, while we have introduced the steering rate u as the new input to the system. To summarize, we have derived a model of the vehicle dynamics with seven states and two control inputs given by the rear traction force and the steering rate of the front wheel.

As you might have noticed, these system dynamics are nonlinear due to the presence of nonlinear functions (like sin and cos) and products between inputs and states. We suggest that you try to go through the modelling assumptions a few more times to make sure that everything makes sense, because we will later reuse this model multiple times for the next assignments.

For this assignment, we will adopt a simpler linearized model of the lateral dynamics of the vehicle to start our getting an intuition of the model. It is good that you remember the units of each state/input/parameter of the system to make sure your arguments are physically sound. Units are reported in the Table 1.

Variable	Units	Variable	Units
u	rad/s	X, Y	m
F	N	v_x, v_y	m/s
l_r, l_f	m	ϕ	rad
m	kg	r	rad/s
		δ	rad

Table 1: Table of units.

Introduction to the design task. Before you can compete at any level in motorbike racing, you usually need to pass a simple change of lane test to make sure you can safely avoid obstacles on the road. It is not uncommon for vehicles to lose parts, which can damage the following competitors if not carefully avoided. For this assignment, we will use the simplified lateral linear dynamics to undertake a change of lane maneuver and avoid an obstacle in the middle of the road (see Figure 3).

We will consider the vehicle as moving at a constant velocity v_x (*i.e.*, $\dot{v}_x = 0$), and we will consider the heading angle ϕ to be approximately constant and equal to $\phi = 0$ rad. With these assumptions, our nonlinear model can be greatly simplified, yielding the linear lateral dynamics

$$\begin{aligned} \dot{Y} &= v_y \\ \dot{v}_y &= v_x \delta \frac{l_f}{l_r + l_f} \end{aligned}$$

which is a simple double integrator dynamics defined as

$$\begin{bmatrix} \dot{Y} \\ \dot{v}_y \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} Y \\ v_y \end{bmatrix} + \begin{bmatrix} 0 \\ v_x \cdot \frac{l_r}{l_f + l_r} \end{bmatrix} u$$

Using the notation of the lecture slides, we can define the state $\mathbf{x} = [Y \ v_y]^T$ and define the matrices

$$A_c = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \quad B_c = \begin{bmatrix} 0 \\ v_x \cdot \frac{l_r}{l_f + l_r}. \end{bmatrix}$$

from which we can derive the continuous time model

$$\mathbf{x}(t) = A_c \mathbf{x}(t) + B_c u(t). \tag{8}$$

The longitudinal dynamics is a simple *uncontrolled* single integrator dynamics of the form

$$\dot{X} = v_x$$

from which we can simply derive that our X coordinate varies linearly over time. Namely, assuming the initial condition at time $t_0 = 0$ as $X(0) = 0$, then our position at each time can be determined as

$$X = v_x \cdot t.$$

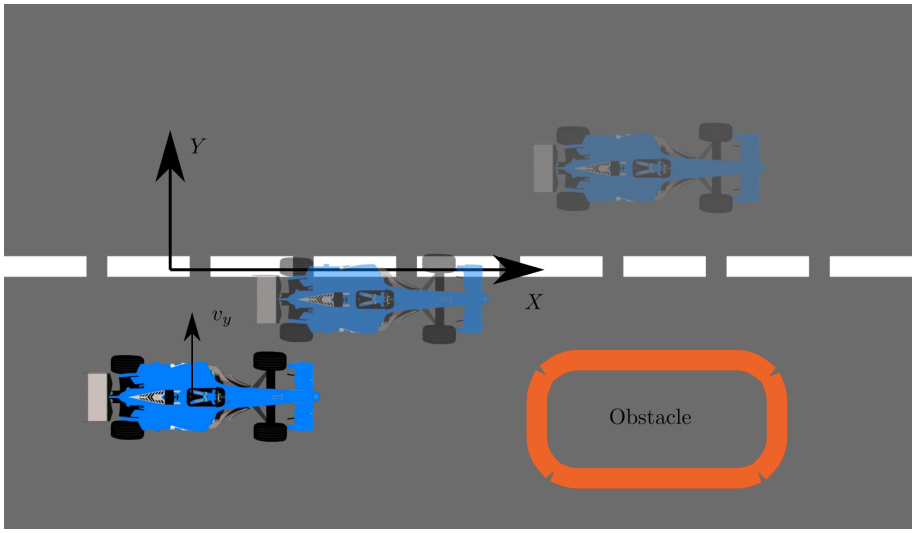

Figure 3: Overtake manoeuvre.

Design of the state feedback controller Now that we have defined our dynamical system, you are ready to advance in your first control design task :

Q1: Inside the class `LinearCarModel` implement the contiunous model dynamics above in the function `_compute_system_matrices`. Remember that you can access the parameters of the vehicle using the `self.` notation. For example, the mass of the vehicle is available as `self.mass`

Q2: As you may have noticed, the continuous-time model is a double integrator. Compute the discrete model matrices A, B from the continuous time matrices A_c, B_c and compare the result with the numerical values returned by `c2d` - are they the same?

Now, observe that we can write the discrete-time model as

$$\mathbf{x}_{k+1} = A\mathbf{x}_k + Bu_k \qquad y_k = C\mathbf{x}_k + Du_k \tag{9}$$

Q3: Compute the transfer function of the continuous-time model from the control input u to the output y, considering $C = \begin{bmatrix}1 & 0\end{bmatrix}$ and $D = 0$. How many poles and zeros does the system have? Where are they located? How many poles and zeros do you expect for the discrete-time model, and where should they be located? Was your intuition right?

Note: For this question, you can use Matlab or any other tool you are comfortable with. In Python, the `control` library provides the same functionality as Matlab (classes `StateSpace`, `TransferFunction` and method `control.pzmap`), but requires `slycot` to be installed—already available on the Virtual Machine. The method `poles_zeros` in the `LinearCarModel` class implements this functionality. For installation instructions, please check the Appendix 1.1 for more information.

Q4: We are finally ready to design our controller for the change of lane. We will assume the road is 4 m with the centerline located at the lateral coordinate $Y = 0$. The initial state of the car is $\mathbf{x}_0 = \begin{bmatrix}-1.0 & 0.0\end{bmatrix}^T$ corresponding to the center of the lower lane in Figure 3. In order to change lane and avoid the obstacle (orange oval in Figure 3), we want to reach the reference state $\mathbf{x}_{\text{ref}} = \begin{bmatrix}1.0 & 0.0\end{bmatrix}^T$ corresponding to the middle of the upper lane.

Based on the discretized linear model in (9), derive a state feedback controller of the form

$$u_k = -L(\mathbf{x}_k - \mathbf{x}_{\text{ref}}) \tag{10}$$

where L is the state feedback gain, designed by placing the poles of the closed-loop poles at the desired location using the function `place` in the function `get_closed_loop_gain`. Implement this controller in the function `control_law` and make sure the following constraints are satisfied.

- Since the obstacle is 200 m ahead and you are driving at $v_x = 20$ m/s we need to reach at least 75% of the reference within 10 s to be sure the obstacle is avoided.

- To not give the impression that we don't know what we are doing, you should avoid sharp driving for this simple scenario. Thus, the maximum steering rate u should satisfy $|u| < 0.25$ and the maximum lateral velocity should be $|v_y| < 1$ m/s.
- You should **not overshoot** more than 1 m to avoid going offroad!!

Q5: After several ground tests, we realized that our model presents some dynamical error that we approximate as a constant disturbance in the lateral velocity of 2 cm/s. This effect is simulated in the function `set_disturbance`. To fix this, we design an integral action according to

$$u_k = -L\mathbf{x}_k - K_i i_k \qquad i_{k+1} = i_k + h \cdot (p_k - p_{\text{ref}}) \tag{11}$$

where p_k is p_{X_k} and p_{ref} the desired reference position. Implement the integral action (11) in the functions `update_integral` and `control_law`, under the `Controller` class. Design the feedback such that you don't go offroad and you reach the reference within 30 s, respecting the initial given constraints on the input and velocity given in Q4. The function `activate_integral_action` can then be used to set an integral gain and system sampling time. **Note:** constraints in Q4 also apply here.

Submission To complete this design project, you should upload a `.zip` file containing the provided code, properly completed. The task grading is based on a successful run of `task1.py`, which should provide all the results. Post all your questions on the Slack workspace.

Good Luck!

1 Appendix

1.1 Slycot Installation

Slycot installation is only recommended on Ubuntu systems. To install `slycot`, run the following commands:

```
sudo apt install gfortran
sudo apt install liblapack3 liblapack-dev
sudo apt install libblas3 libblas-dev
pip install slycot
```

If any errors arise, **don't hesitate** to reach Gregorio Marchesini on **Slack** via direct message.