Model Predictive Control - EL2700

Linear Quadratic Regulator

Assignment 3

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

Gregorio Marchesini, Samuel Erickson Andersson, and Mikael Johansson

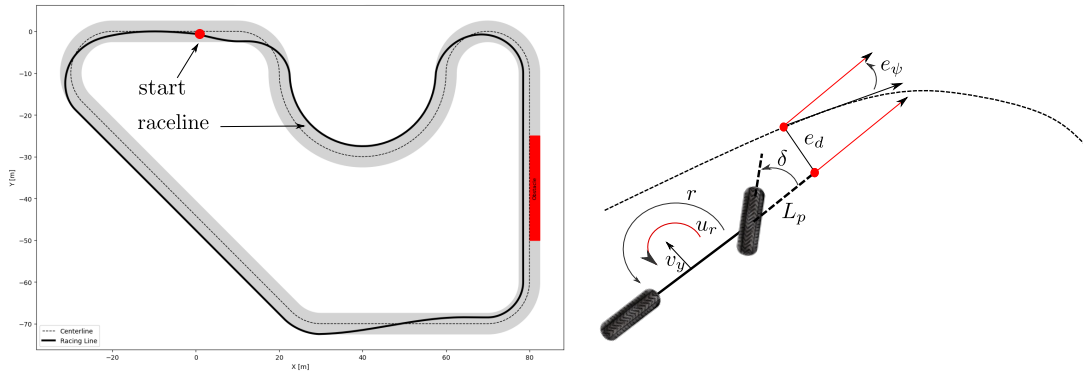


Figure 1: Racetrack circuit and optimal raceline. Red rectangle represents an obstacle to be avoided.

Introduction

In this assignment, we will explore how the Linear Quadratic Regulator model you have studied in class can actually be used to drive your vehicle close to the optimal raceline you have designed in the previous assignment. With this objective, we will first compute an optimal raceline and a *feedforward* controller that optimally tracks the given raceline. We will then define a feedback LQR controller to optimally track the given *feedforward* reference and optimally stabilize the system in the presence of unexpected disturbances.

In order to describe a vehicle as a linear model, we will start from the assumption that we have computed a raceline, for example, the black curve in Figure 1. You already saw in the previous assignment how this can be done using convex optimization. Moreover, the raceline will already be described for you with coordinates $X(s), Y(s)$ and curvature $\kappa(s)$ where s represents the intrinsic coordinate of the raceline such that

$$0 \leq s \leq S, \tag{1}$$

with S representing the raceline length in meters (see Figure 1). We already made this reparametrization from the centerline intrinsic coordinate to the recline intrinsic coordinate for you, so you don't need to worry about that.

To track the path, we select a point L_p meters in front of the vehicle and we define the displacement error e_d and the heading error e_ψ (see Figure 1 right panel). We usually call L_p the *look-ahead* distance. Assuming we are sufficiently close to the raceline and that we are tracking the raceline at a constant longitudinal speed $\bar{v}$, the error dynamics can then be modeled using the linear model

$$\begin{bmatrix} \dot{e}_d(t) \\ \dot{e}_\psi(t) \\ \dot{v}_y(t) \\ \dot{r}(t) \\ \dot{\delta}(t) \end{bmatrix} = \begin{bmatrix} 0 & \bar{v} & -1 & -L_p & 0 \\ 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & \frac{c_1}{m\bar{v}} & (\frac{c_2}{m\bar{v}^2} - 1) & \frac{C_f}{m} \\ 0 & 0 & \frac{c_2}{I_z} & \frac{c_3}{\bar{v}I_z} & \frac{C_f l_f}{I_z} \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} e_d(t) \\ e_\psi(t) \\ v_y(t) \\ r(t) \\ \delta(t) \end{bmatrix} + \begin{bmatrix} 0 \\ \bar{v} \\ 0 \\ 0 \\ 0 \end{bmatrix} \kappa(t) + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & \frac{1}{I_z} \\ 1 & 0 \end{bmatrix} \begin{bmatrix} u_\delta(t) \\ u_r(t) \end{bmatrix} \tag{2}$$

which we can write in compact form as

$$\dot{x}(t) = A^c x(t) + B^c u(t) + B_w^c \kappa(t), \tag{3}$$

with

$$\dot{x}(t) = [e_d(t)\ e_\psi(t)\ v_y(t)\ r(t)\ \delta(t)], \tag{4}$$

and

$$\dot{u}(t) = [u_\delta(t)\ u_r(t)]. \tag{5}$$

We use the superscript ^{c} to make clear that this is the continuous dynamics. The parameters of the system are found in Table 1, and we define the states of the system[1] as

1. e_d : displacement error [m]
2. e_ψ : heading error [rad]
3. v_y : lateral vehicle velocity [m/s]
4. r : yaw rate of the vehicle [rad/s]
5. δ : steering angle [rad]

and inputs

1. u_δ : steering rate [rad/s]
2. u_r : differential torque [Nm]

and finally $\kappa(t)$ is the curvature of the raceline expressed in [rad/m] at a given time t.

Since we are following a determined path, we want to rewrite our dynamics in terms of the independent curvilinear coordinate s rather than time t. This is actually quite important for our control approach to be robust since, when driving, we think of the control action to provide to a vehicle in terms of our position with respect to the line we want to follow rather than time. This is why, in rally competitions, the copilot might shout at you

Sharp right turn in 100 m!

and not

Sharp right turn in 10 s!

Because time is relative to the speed you are following the line. Even without knowing any notion of time, a good pilot should still know how to act based on his/her position and speed, which is something we want for our controller. By the fact that we are moving at constant velocity $\bar{v}$ along the raceline, we have that in a given small dt, we move along the path a distance $ds = \bar{v}dt$. We can use this observation to derive a relation between the time and *curvilinear derivative* as :

$$\frac{d}{ds}\frac{1}{\bar{v}} = \frac{d}{dt} \tag{6}$$

from which we can rewrite our dynamics in the curvilinear coordinate s as

$$\begin{aligned} \dot{x}(s) &= \frac{1}{\bar{v}}A^c x(s) + \frac{1}{\bar{v}}B^c u(s) + \frac{1}{\bar{v}}B_w^c \kappa(s) \\ y(s) &= Cx(s) = \left[e_d(s), e_\psi(s)\right]^T \end{aligned} \tag{7}$$

where y represents the output error from the system.

Design Task

In the assignment `.zip` file, you will find the following Python files

1. **`linear_car_model.py`**: Contains a class that stores the parameters of your vehicle.
2. **`qp_solver.py`**: This file contains an instance of the raceline optimizer we use for this assignment. You will **not** need to touch this file for this assignment.

[1] If you want to understand better where this dynamics comes from, you should check the appendix to the assignment. In practice, you will one day have to derive linear models from nonlinear ones, so it is a good exercise for you to check the assumptions considered to describe our vehicle as a linear system.

Parameter	Value	Units
m	1823	kg
l_f	1.27	m
l_r	1.90	m
I_z	6286	$kg \cdot m^2$
C_r	1.6	N/rad

Parameter	Value	Units
C_f	1.6	N/rad
c_1	-3.2	N/rad
c_2	1.008	$N \cdot m/\text{rad}$
c_3	-8.356	$N \cdot m/\text{rad}$

Table 1: The values and units of the parameters of the system.

3. **`road.py`**: This file contains an instance of the racetrack object that we use to store information about the racetrack. You will **not** need to touch this file for this assignment.
4. **`sim.py`**: This file contains the routines you will be using to simulate the performance of your LQR controller in tracking the optimal raceline computed using `qp_solver.py`.
5. **`task3.py`**: This is the main file that you will run to visualize the results of your simulations.

Your task will be to answer the following questions by appropriately **implementing** the code, where required, and by **motivating** your design choices. It is suggested that you use plots from your code to answer the questions!

Q1: Inside the class `LinearCarModel` define the continuous time system matrices

$$\frac{1}{v_x}B_w^c,\ \frac{1}{v_x}A_w^c,\ \frac{1}{v_x}B^c \text{ and } C$$

as per equation (7). Remember that you can access the parameters of the vehicle using the syntax `self.parameter`. After completing this piece of code, you can directly use the method `c2d` to obtain a discrete-time system as

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k + B_w\kappa_k \\ y_k &= Cx_k. \end{aligned}$$

Q2: Inside the file `sim.py`, we will design a *feedforward* action that tracks our reference raceline at best. In order to do so, we will use `cvxpy` to find an optimal input and state trajectory that minimizes the output error as

$$\text{minimize} \left(\sum_{k=0}^{N-1} y_k^T Q y_k + u_k^T R u_k \right) + y_N^T Q y_N \tag{8a}$$

$$\text{subject to } x_{k+1} = Ax_k + Bu_k + B_w\kappa_k \tag{8b}$$

$$|Sx_k| \leq 25\frac{\pi}{180} \tag{8c}$$

$$y_k = C_k x_k \tag{8d}$$

$$x_0 = 0_5 \tag{8e}$$

where

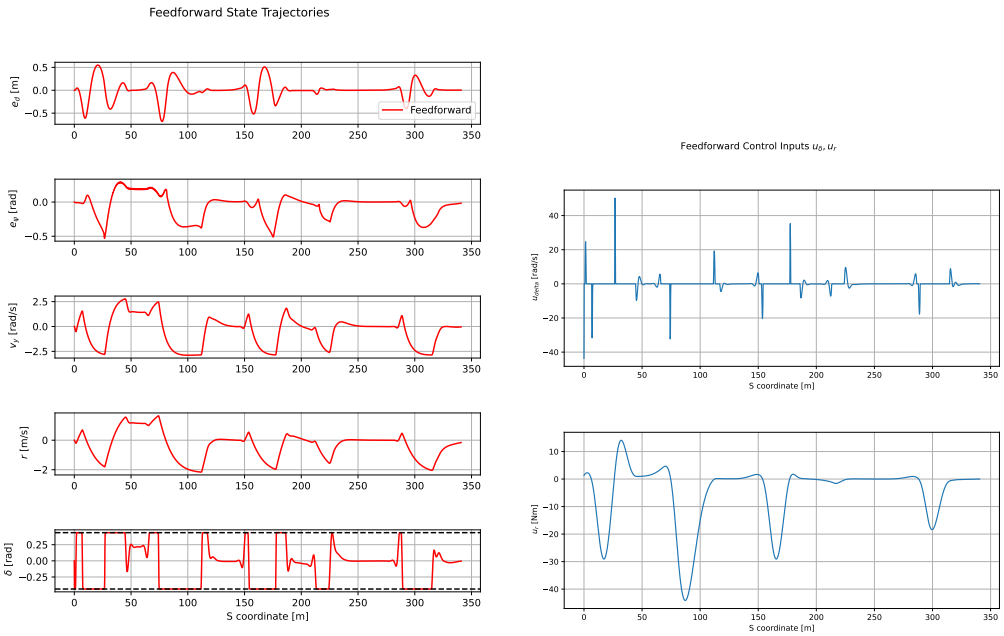
$$Q = \begin{bmatrix} 1000 & 0 \\ 0 & 1000 \end{bmatrix}, \quad R = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.001 \end{bmatrix} \text{ and } \quad S = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Note that constraint (8c) corresponds to the constraint

$$|\delta_k| \leq 25 \text{ deg}$$

limiting the maximum steering angle of the vehicle. This is indeed required for the linear model to be valid. Your task is to implement the given optimization program in (8) in the function `compute_feedforward_action` where the output should be given by two matrices representing the optimal feedforward input and state sequence as

$$x^{\text{ff}} = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_N \end{bmatrix} \in \mathbb{R}^{(N+1)\times 5}, \quad u^{\text{ff}} = \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ x_{N-1} \end{bmatrix} \in \mathbb{R}^{N\times 2}.$$



(a) Reference feedforward state **(b)** Feedforward reference example.

Figure 2: Feedforward example.

If you did everything correctly, your feedforward state and input trajectory will look something like Figure 2. You are welcome to play around with the parameters of the optimization (i.e., matrices Q and R), but resent the initial parameters after having fun.

Q3: Now that we have computed our feedforward action, we will design our LQR controller for the error dynamics. Namely, after defining the state and input error vectors

$$e = x_k - x_k^{\text{ff}}$$

Then the error dynamics become

$$e_{k+1} = Ae_k + B(u_k - u_k^{\text{ff}})$$

which we want to stabilize using an LQR controller in the form

$$u_k = -Le_k + u_k^{\text{ff}}$$

where L is computed using the state and input matrices, $Q \succcurlyeq 0$ and $R \succ 0$ according to the Riccati equation

$$P = Q + A^\top PA - A^\top PB(R + B^\top PB)^{-1}B^\top PA,$$
$$L = (R + B^\top PB)^{-1}B^\top PA,$$

and the closed-loop error dynamics become

$$e_{k+1} = (A - BL)e_k.$$

With this formulation, we want to test the performance of different controller designs on the racetrack. Namely, due to some *bumps* on the racetrack, we will be randomly offset from the raceline, and we will rely on our LQR controller to restabilize the system to its nominal state. Your task is to design an LQR controller for your system using the following cost matrices

$$Q = \begin{bmatrix}0.1 & 0.1 & 0.1 & 0.1 & 0.1\end{bmatrix} \qquad R = \begin{bmatrix}100 & 100\end{bmatrix},$$
$$Q = \begin{bmatrix}100 & 100 & 0.1 & 0.1 & 1\end{bmatrix} \qquad R = \begin{bmatrix}1 & 1\end{bmatrix},$$
$$Q = \begin{bmatrix}100 & 100 & 0.1 & 0.1 & 1\end{bmatrix} \qquad R = \begin{bmatrix}0.1 & 0.1\end{bmatrix},$$

by calling the method `system.get_lqr_controller(Q, R)`. Comment on the design differences that exist between these controllers. In particular, comment on the bandwidth of the controller for the different costs (i.e., maximum and minimum of the control input) and the tracking performance (i.e., overshooting, settling time).

Q4: As a last step, we will try to use the Bryson rule to tune our LQR controller. From the feedforward reference x^{ff} and u^{ff}, compute the maximum absolute value of each state and control variable along the trajectory. Namely, define the five coefficients[2]

$$q_i = \max_{k=0,\dots,N} |x^{\text{ff}}_{ki}| \quad \forall i = 1,\dots,5$$

and the two coefficients

$$r_i = \max_{k=0,\dots,N-1} |u^{\text{ff}}_{ki}| \quad \forall i = 1,2.$$

With these normalizing coefficients, define the cost matrices

$$Q = \text{diag}\left(\frac{p_1}{q_1^2}, \frac{p_2}{q_2^2}, \frac{p_3}{q_3^2}, \frac{p_4}{q_4^2}, \frac{p_5}{q_5^2}\right)$$

and

$$R = \text{diag}\left(\frac{l_1}{r_1^2}, \frac{l_2}{r_2^2}\right).$$

where we use the coefficients $p_i,\ \forall i \in 1,\dots 5$ and $l_i,\ \forall i \in 1,2$ to tune our controller.

Try to tune your LQR controller using the tuning gains q_i and l_i.

(a) Comment on what happens if you increase the gains l_i compared to p_i (i.e. $l_i >> p_i$).

(b) What happens if you do the opposite? (i.e. $p_i >> l_i$)?

(c) Can you find a tuning that does not excessively exceed the steering rate limit of 35 degrees, while still preserving a good system response (low overshooting and reasonable settling time)? (Hint: the steering angle is your main source of control authority over the vehicle, so better not to penalize it excessively.)

Submission Your submission should be a .ZIP file, named with your Group number (e.g., Group1.zip), containing:

1. a PDF document (can be a scanned document, generated from Word, Latex or others) containing the answers and motivation to the questions on to the assignment. Make sure to name your document Assignment3_STUDENT1_STUDENT2.pdf;
2. the *.py code files with a working task3.py that shows your solution on the plot output.

Good Luck!

[2]By the term x^{ff}_{ki} we mean the i^{th} element if the vector x^{ff}_k. Hence, spanning the index i is equivalent to spanning each single state in the state vector x^{ff}_k at step k. The same principle holds for u^{ff}_{ki}.

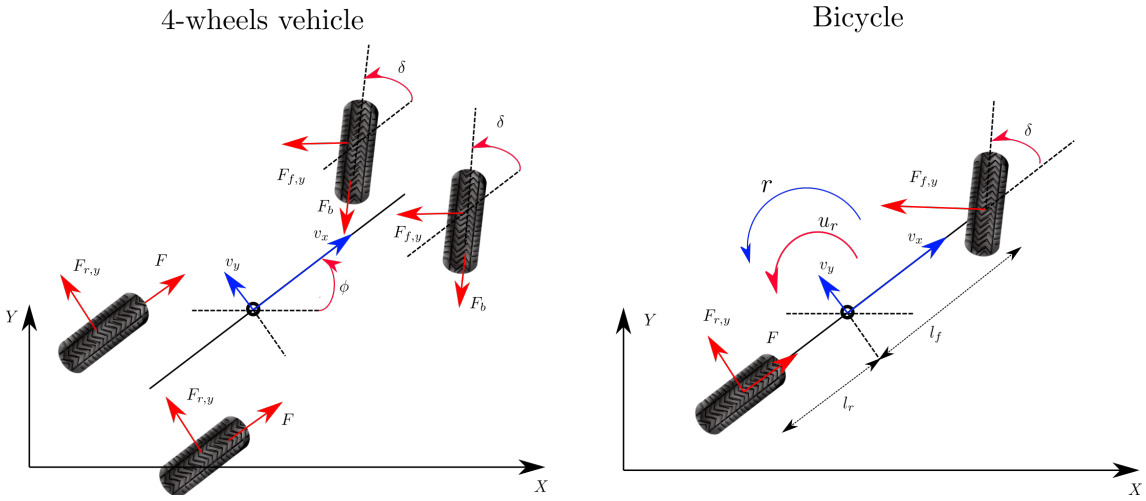


Figure 3: Dynamic Bicyle model

Appendix

Nonlinear Vehicle Model

In this section, you will find the main model applied to derive the linear dynamics that you will use in the assignment. In this assignment, we move from a kinematic model (the one used in the previous two assignments) to a dynamic model of the vehicle, which is more realistic, even if we will still make some simplifications. Namely, consider the four-wheeled vehicle schematically represented in the left panel in (3), designed in such a way that the front wheels provide braking force to decelerate and the rear wheels provide traction force to accelerate. On the left panel in Figure 3 we have depicted a simplified model of a 4-wheels vehicle where F is the driving force, $F_{r,y}$ and $F_{f,y}$ represent the force orthogonal to the rear and front tires, respectively, and lastly, F_b is the braking force. To obtain a simplified bicycle model from the 4-wheel vehicle model, we consider the following assumptions:

- We will idealize our vehicle as a bicycle model by summing forces on parallel wheels over a single wheel (left panel in Figure 3).
- Since most of the modern vehicles are able to impose differential braking on the front wheel (apply different braking force on the two front wheels), we model this effect as a control torque u_r that can be applied to favor the rotation of the bicycle model during turning.
- We introduce breaking force by allowing the force F (right panel in Figure 3) to take negative values for breaking.

By applying these simplifications, we arrive at the nonlinear dynamic bicycle model

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\phi} \\ \dot{v}_x \\ \dot{v}_y \\ \dot{r} \\ \dot{\delta} \end{bmatrix} = \begin{bmatrix} v_x \cos(\phi) - v_y \sin(\phi) \\ v_x \sin(\phi) + v_y \cos(\phi) \\ r \\ \frac{1}{m}(F - F_{f,y} \sin(\delta) + m v_y r) \\ \frac{1}{m}(F_{r,y} + F_{f,y} \cos(\delta) - m v_x r) \\ \frac{1}{I_z}(F_{f,y} l_f \cos(\delta) - F_{r,y} l_r + u_r) \\ u_\delta \end{bmatrix} \tag{9}$$

where

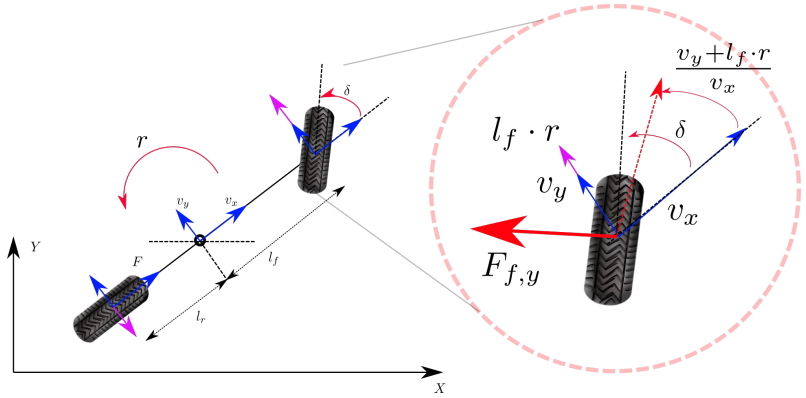


Figure 4: Force applied on the front wheel

1. v_x represents the longitudinal velocity of the vehicle in the body frame [m/s]
2. v_y represents the longitudinal velocity of the vehicle in the body frame [m/s]
3. ϕ represents the heading angle [rad]
4. r is the heading rate (yaw rate) [rad/s]
5. δ is the steering angle of the wheels [rad]
6. X and Y are the inertial coordinates of the vehicle [m]

The forces imparted by the tires can be modeled for simplicity as

$$\begin{aligned} F_{f,y} &= \left(\delta - \frac{l_f r + v_y}{v_x}\right) C_f, \\ F_{r,y} &= \left(\frac{l_r r - v_y}{v_x}\right) C_r, \end{aligned} \tag{10}$$

where C_r and C_f are denoted as *cornering stiffness* coefficient and are constant parameters of tires. The additional parameter I_z appearing in the dynamics represents the moment of inertia of the vehicle around the z axis that is pointing outward in the Figure 3.

While we do not step into describing the details of the dynamics, we want to give a hint the why the forces applied by the tires are modeled as in (10), by focusing on what happens to the front wheel when the vehicle is turning. Looking at Figure 4 we see that the velocity of the front wheel is given by the velocity of the center of mass, plus a small component in the y direction due to the rotation of the vehicle (purple arrow). For the front wheel, this additional component is equal to

$$\bar{v}_y = l_f \cdot r,$$

such that the angle between the velocity vector of the front wheel and the vehicle's longitudinal axis is defined by

$$\alpha = \arctan\left(\frac{l_f \cdot r + v_y}{v_x}\right) \approx \left(\frac{l_f \cdot r + v_y}{v_x}\right),$$

where we used the small angles approximation to approximate α. This assumption is generally reasonable for moderate driving regimes where $v_x >> v_y$. At this point, we can see that the force imparted from the front tires is proportional to the difference between the steering angle δ and the angle α as $F_{y,f} = (\delta - \alpha)C_f$, which we wrote explicitly in (10). This is in accordance with what we also experience with real vehicles. Namely, if we are going on a straight line (i.e., no lateral velocity or rotation $v_y \approx 0$ and $r \approx 0$), then a sudden change of steering produces a strong lateral force equal to the full steering angle δ (since in this case $\alpha \approx 0$). On the contrary, when we are already turning, we receive less force back from the tires for the same steering angle δ. All these arguments can be repeated on the rear tire, with the only difference that the rear tire does not have a steering angle in our model.

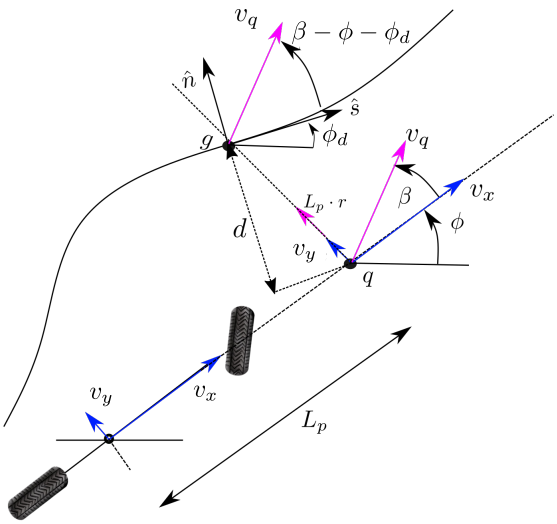


Figure 5: Path following task

Linearizing the model for path following. For the linearization, we will start with the assumption that we want to track a given path with constant velocity $\bar{v}$. We will also assume that we are sufficiently aligned with the given path such that the longitudinal velocity of the vehicle v_x is approximately equal to the desired reference velocity as

$$v_x \approx \bar{v}.$$

Given these settings, we need to define a meaningful state error to define a successful tracking performance. To do so, we project the state of the vehicle into the local frame of the path, which we will denote F frame (or Frenet frame in the literature).

At a given goal location g along the path, the $\mathcal{F}$ frame is defined by the tangent vector $\hat{s}$ and the cross-track vector $\hat{n}$ (see top frame of reference in Figure 5). Within this frame we can define the cross-track displacement of a point, which respect to the path by the displacement d.

The successful tracking performance will revolve around setting to zero the displacement d and the heading error of the vehicle orientation with respect to the desired heading of the track. Instead of defining the displacement with respect to the center of mass of the vehicle, we will define the displacement with respect to a *preview* point in front of the vehicle named q. This is usually to allow for a smoother tracking performance.

Namely, we define the state error as

$$\begin{aligned} e_\psi &= \phi_d - \phi \\ e_d &= -d, \end{aligned}$$

and we can obtain the error dynamics by noting that

$$\begin{aligned} \dot{\phi}_d &= \kappa \bar{v} \\ \dot{d} &= v_q \sin(\beta + \phi - \phi_d). \end{aligned}$$

The velocity v_q of the point q can be approximated noting that

$$v_q = \sqrt{(v_y + L_p \cdot r)^2 + v_x^2} \approx v_x \approx \bar{v}$$

where we have used the assumption that $v_x >> (v_y + L_p \cdot r)$ i.e. the longitudinal component of the velocity is much higher than the lateral component. Let now the angle

$$\beta = \tan\left(\frac{v_y + L_p \cdot r}{v_x}\right) \approx \frac{v_y + L_p \cdot r}{v_x} \approx \frac{v_y + L_p \cdot r}{\bar{v}},$$

Thus, we can project the velocity of the reference point q on the $\mathcal{F}$ frame to obtain the time variation of the displacement d in the $\hat{n}$ direction of the $\mathcal{F}$ frame as

$$\dot{d} = \bar{v} \cdot \sin(\beta + \phi - \phi_d)$$

which we can equivalently define based on the orientation error $e_\psi = \phi_d - \phi$ as

$$\dot{d} = \bar{v} \cdot \sin(\beta - e_\psi) = \bar{v}\sin(\beta)\cos(e_\psi) - \bar{v}\sin(e_\psi)\cos(\beta) \approx v_y + r \cdot L_p - \bar{v}e_\psi$$

where we have used the assumption that β and e_ψ are small angles for which $\cos(e_\psi) \approx 1$ and $\cos(\beta) \approx 1$. We can finally summarize the tracking error dynamics as

$$\begin{aligned}\dot{e}_\psi &= r - \kappa\bar{v} \\ \dot{e}_d &= -v_y - r \cdot L_p + \bar{v}e_\psi\end{aligned}$$

The last step we need to finalize the linear model is to write explicitly the equations of motion for v_y and r assuming a small steeing angle δ such that $\cos(\delta) \approx 1$, from which we get

$$\begin{aligned}\dot{v}_y &= -\left(\frac{C_f + C_r}{m\bar{v}}\right)v_y + \left(\frac{-l_f C_f + l_r C_r - \bar{v}m}{m\bar{v}}\right)r + \frac{C_f}{m}\delta \\ \dot{r} &= \left(-\frac{C_f l_f}{\bar{v}I_z} + \frac{C_r l_r}{\bar{v}I_z}\right)v_y - \left(\frac{l_f^2 C_f}{v_x I_z} + \frac{l_r^2 C_r}{v_x I_z}\right)r + \frac{C_f l_f}{I_z}\delta.\end{aligned}$$

By introducing the parameters

$$c_1 = -(C_f + C_r) \qquad c_2 = l_r C_r - l_f C_f \qquad c_3 = -(l_f^2 C_f + l_r^2 C_r),$$

we can compactly write

$$\begin{aligned}\dot{v}_y &= \frac{c_1}{m\bar{v}}v_y + \left(\frac{c_2}{m\bar{v}^2} - 1\right)r + \frac{C_f}{\bar{v}m}\delta \\ \dot{r} &= \frac{c_2}{I_z\bar{v}}v_y + \frac{c_3}{\bar{v}I_z}r + \frac{C_f l_f}{I_z}\delta.\end{aligned}$$

and the linear dynamics of the vehicle thus become

$$\begin{bmatrix}\dot{e}_d \\ \dot{e}_\psi \\ \dot{v}_y \\ \dot{r} \\ \dot{\delta}\end{bmatrix} = \begin{bmatrix}0 & \bar{v} & -1 & -L_p & 0 \\ 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & \frac{c_1}{m\bar{v}} & (\frac{c_2}{m\bar{v}^2} - 1) & \frac{C_f}{m} \\ 0 & 0 & \frac{c_2}{I_z} & \frac{c_3}{\bar{v}I_z} & \frac{C_f l_f}{I_z} \\ 0 & 0 & 0 & 0 & 0\end{bmatrix}\begin{bmatrix}e_d \\ e_\psi \\ v_y \\ r \\ \delta\end{bmatrix} + \begin{bmatrix}0 \\ \bar{v} \\ 0 \\ 0 \\ 0\end{bmatrix}\kappa + \begin{bmatrix}0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & \frac{1}{I_z} \\ 1 & 0\end{bmatrix}\begin{bmatrix}u_\delta \\ u_r\end{bmatrix}.$$