Gain Scheduling

Richard M. Murray, 19 Nov 2021

This notebook contains an example of using gain scheduling for feedback control of a nonlinear system. A gain scheduled controller has feedback gains that depend on a set of measured parameters in the system. For exampe:

$$u = u_\mathrm{d} - K(x_\mathrm{d}, u_\mathrm{d})(x - x_\mathrm{d}),$$

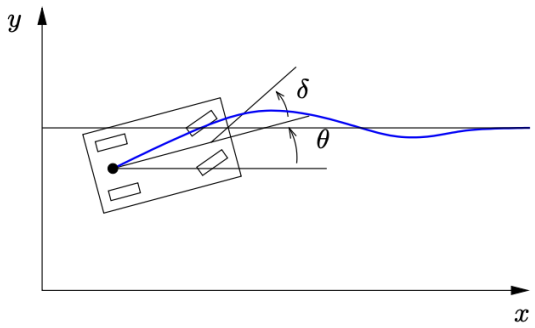
where $K(x_\mathrm{d}, u_\mathrm{d})$ depends on the desired system state and input.

In this notebook, we work through the gain scheduled controller in Example 2.1 of OBC.

In [1]:
```
# Import the packages needed for the examples included in this notebook
import numpy as np
import matplotlib.pyplot as plt
from cmath import sqrt
import control as ct
```

Vehicle Steering Dynamics

The vehicle dynamics are given by a simple bicycle model:



$$\begin{aligned}\dot{x} &= \cos\theta\, v \\ \dot{y} &= \sin\theta\, v \\ \dot{\theta} &= \frac{v}{l}\tan\delta\end{aligned}$$

We take the state of the system as (x, y, θ) where (x, y) is the position of the vehicle in the plane and θ is the angle of the vehicle with respect to horizontal. The vehicle input is given by (v, δ) where v is the forward velocity of the vehicle and δ is the angle of the steering wheel. The model includes saturation of the vehicle steering angle.

In [2]:
```
# Bicycle model dynamics
#
# System state: x, y, theta
# System input: v, delta
# System output: x, y
# System parameters: wheelbase, maxsteer
#
def bicycle_update(t, x, u, params):
```

```
    # Get the parameters for the model
    l = params.get('wheelbase', 3.)         # vehicle wheelbase
    deltamax = params.get('maxsteer', 0.5)  # max steering angle (rad)

    # Saturate the steering input
    delta = np.clip(u[1], -deltamax, deltamax)

    # Return the derivative of the state
    return np.array([
        np.cos(x[2]) * u[0],            # xdot = cos(theta) v
        np.sin(x[2]) * u[0],            # ydot = sin(theta) v
        (u[0] / l) * np.tan(delta)      # thdot = v/l tan(delta)
    ])

def bicycle_output(t, x, u, params):
    return x                            # return x, y, theta (full state)

# Define the vehicle steering dynamics as an input/output system
bicycle = ct.NonlinearIOSystem(
    bicycle_update, bicycle_output, states=3, name='bicycle',
    inputs=('v', 'delta'),
    outputs=('x', 'y', 'theta'))
```

Gain scheduled controller

For this system we use a simple schedule on the forward vehicle velocity and place the poles of the system at fixed values. The controller takes the current and desired vehicle position and orientation plus the velocity velocity as inputs, and returns the velocity and steering commands.

Linearizing the system about the desired trajectory, we obtain

$$A(x_\mathrm{d}) = \frac{\partial f}{\partial x}\bigg|_{(x_\mathrm{d},u_\mathrm{d})} = \begin{bmatrix} 0 & 0 & -\sin\theta_\mathrm{d}\, v_\mathrm{d} \\ 0 & 0 & \cos\theta_\mathrm{d}\, v_\mathrm{d} \\ 0 & 0 & 0 \end{bmatrix}\bigg|_{(x_\mathrm{d},u_\mathrm{d})} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & v_\mathrm{d} \\ 0 & 0 & 0 \end{bmatrix},$$

$$B(x_\mathrm{d}) = \frac{\partial f}{\partial u}\bigg|_{(x_\mathrm{d},u_\mathrm{d})} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & v_\mathrm{d}/l \end{bmatrix}.$$

We form the error dynamics by setting $e = x - x_\mathrm{d}$ and $w = u - u_\mathrm{d}$:

$$\dot{e}_x = w_1, \qquad \dot{e}_y = e_\theta, \qquad \dot{e}_\theta = \frac{v_\mathrm{d}}{l} w_2.$$

We see that the first state is decoupled from the second two states and hence we can design a controller by treating these two subsystems separately.

Suppose that we wish to place the closed loop eigenvalues of the longitudinal dynamics (e_x) at $-\lambda_1$ and place the closed loop eigenvalues of the lateral dynamics (e_y, e_θ) at the roots of the polynomial equation $s^2 + a_1 s + a_2 = 0$.

This can accomplished by setting

$$w_1 = -\lambda_1 e_x$$
$$w_2 = -\frac{l}{v_\mathrm{r}}\left(\frac{a_2}{v_\mathrm{r}} e_y + a_1 e_\theta\right).$$

Note that the gains depend on the velocity v_r (or equivalently on the nominal input u_d), giving us a gain scheduled controller.

In [3]:
```
# System state: none
# System input: x, y, theta, xd, yd, thetad, vd, delta
# System output: v, delta
# System parameters: longpole, latomega_c, latzeta_c
def gainsched_output(t, x, u, params):
    # Get the controller parameters
    longpole = params.get('longpole', -2.)
    latomega_c = params.get('latomega_c', 2)
    latzeta_c = params.get('latzeta_c', 0.5)
    l = params.get('wheelbase', 3)
    vref = params.get('vref', None)

    # Extract the system inputs and compute the errors
    x, y, theta, xd, yd, thetad, vd, deltad = u
    ex, ey, etheta = x - xd, y - yd, theta - thetad

    # Determine the controller gains
    lambda1 = -longpole
    a1 = 2 * latzeta_c * latomega_c
    a2 = latomega_c**2

    # Determine the speed to use for computing the gains
    if vref is None:
        vref = vd

    # Compute and return the control law
    v = -lambda1 * ex           # leave off feedforward to generate transien
    if vd != 0:
        delta = deltad - ((a2 * l) / vref**2) * ey - ((a1 * l) / vref) * eth
    else:
        # We aren't moving, so don't turn the steering wheel
        delta = deltad

    return  np.array([v, delta])

# Define the controller as an input/output system
gainsched = ct.NonlinearIOSystem(
    None, gainsched_output, name='controller',          # static system
    inputs=('x', 'y', 'theta', 'xd', 'yd', 'thetad',    # system inputs
            'vd', 'deltad'),
    outputs=('v', 'delta')                              # system outputs
)
```

Reference trajectory subsystem

The reference trajectory block generates a simple trajectory for the system given the desired speed (vref) and lateral position (yref). The trajectory consists of a straight line of the form (vref * t, yref, 0) with nominal input (vref, 0).

In [4]:
```
# System state: none
# System input: vref, yref
# System output: xd, yd, thetad, vd, deltad
# System parameters: none
#
def trajgen_output(t, x, u, params):
    vref, yref = u
    return np.array([vref * t, yref, 0, vref, 0])

# Define the trajectory generator as an input/output system
trajgen = ct.NonlinearIOSystem(
    None, trajgen_output, name='trajgen',
    inputs=('vref', 'yref'),
    outputs=('xd', 'yd', 'thetad', 'vd', 'deltad'))
```

System construction

The input to the full closed loop system is the desired lateral position and the desired forward velocity. The output for the system is taken as the full vehicle state plus the velocity of the vehicle.

We construct the system using the InterconnectedSystem constructor and using signal labels to keep track of everything.

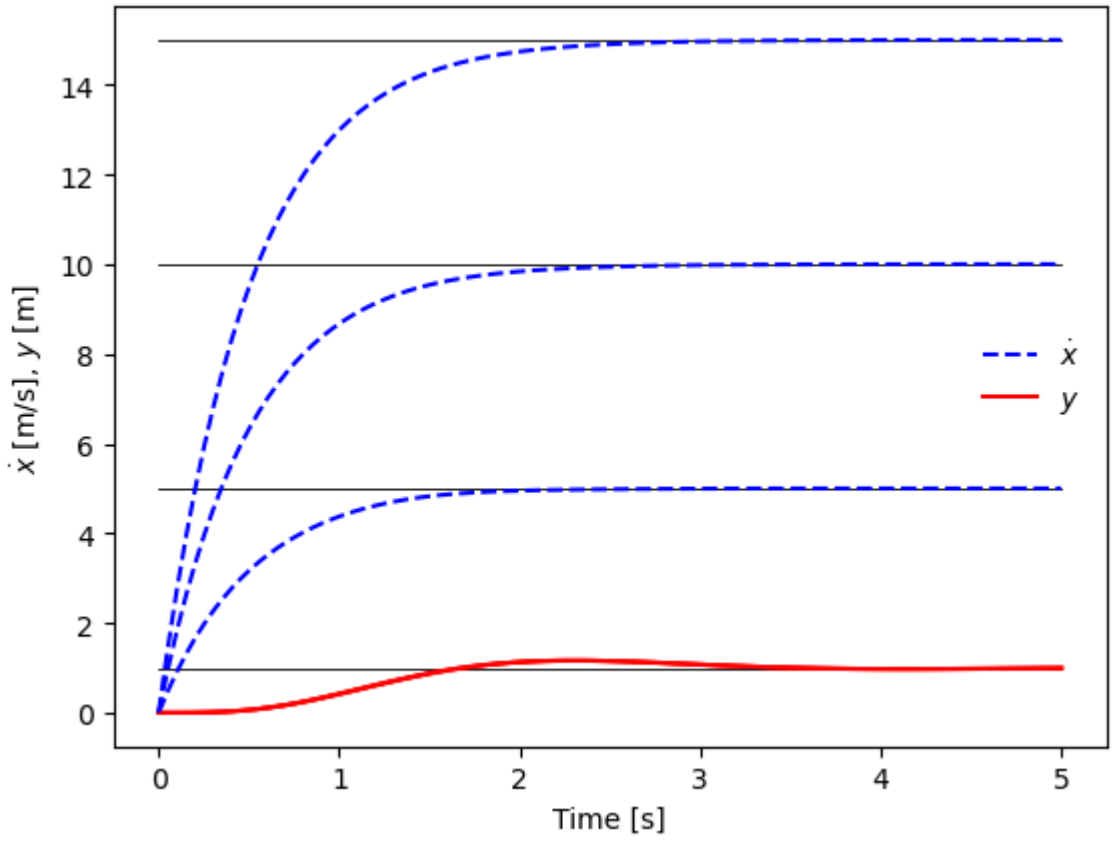
In [5]:
```
steering_gainsched = ct.interconnect(
    # List of subsystems
    (trajgen, gainsched, bicycle), name='steering',

    # System inputs
    inplist=['trajgen.vref', 'trajgen.yref'],
    inputs=['yref', 'vref'],

    #  System outputs
    outlist=['bicycle.x', 'bicycle.y', 'bicycle.theta', 'controller.v',
             'controller.delta'],
    outputs=['x', 'y', 'theta', 'v', 'delta']
)
```

Note the use of signals of the form `sys.sig` to get the signals from a specific subsystem.

System simulation

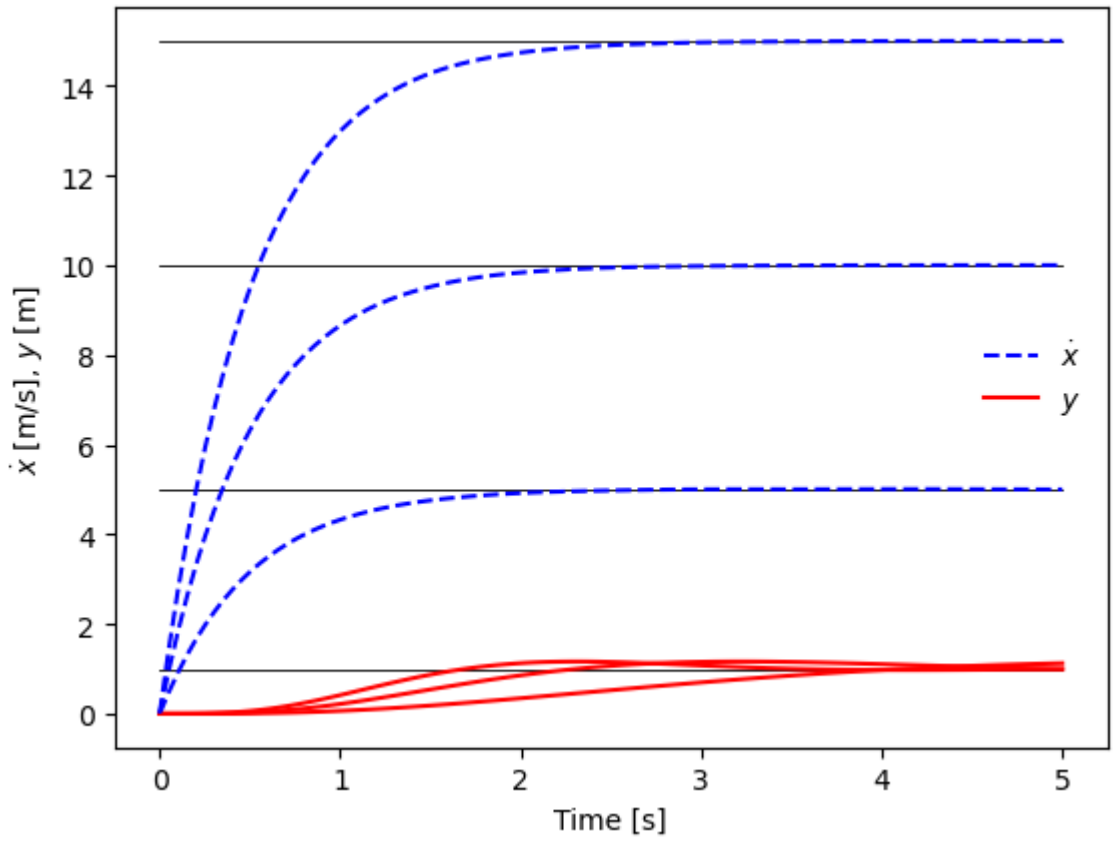
In [6]:
```
# Set up the simulation conditions
yref = 1
```

```
T = np.linspace(0, 5, 100)

# Plot the reference trajectory for the y position
plt.plot([0, 5], [yref, yref], 'k-', linewidth=0.6)

# Find the signals we want to plot
y_index = steering_gainsched.find_output('y')
v_index = steering_gainsched.find_output('v')

# Do an iteration through different speeds
for vref in [5, 10, 15]:
    # Simulate the closed loop controller response
    tout, yout = ct.input_output_response(
        steering_gainsched, T, [vref * np.ones(len(T)), yref * np.ones(len(T

    # Plot the reference speed
    plt.plot([0, 5], [vref, vref], 'k-', linewidth=0.6)

    # Plot the system output
    y_line, = plt.plot(tout, yout[y_index, :], 'r-')  # lateral position
    v_line, = plt.plot(tout, yout[v_index, :], 'b--')  # vehicle velocity

# Add axis labels
plt.xlabel('Time [s]')
plt.ylabel('$\dot{x}$ [m/s], $y$ [m]')
plt.legend((v_line, y_line), ('$\dot{x}$', '$y$'),
           loc='center right', frameon=False);
```



Comparison to fixed controller

In [7]:
```
# Rerun with no gain-scheduling

# Plot the reference trajectory for the y position
plt.plot([0, 5], [yref, yref], 'k-', linewidth=0.6)

# Do an iteration through different speeds
for vref in [5, 10, 15]:
    # Simulate the closed loop controller response
    tout, yout = ct.input_output_response(
        steering_gainsched, T, [vref * np.ones(len(T)), yref * np.ones(len(T
        params={'vref': 15})

    # Plot the reference speed
    plt.plot([0, 5], [vref, vref], 'k-', linewidth=0.6)

    # Plot the system output
    y_line, = plt.plot(tout, yout[y_index, :], 'r-')  # lateral position
    v_line, = plt.plot(tout, yout[v_index, :], 'b--')  # vehicle velocity

# Add axis labels
plt.xlabel('Time [s]')
plt.ylabel('$\dot{x}$ [m/s], $y$ [m]')
plt.legend((v_line, y_line), ('$\dot{x}$', '$y$'),
           loc='center right', frameon=False);
```



Things to try

- Use different reference trajectories (eg, flatness-based trajectory)
- Try scheduling on the current state rather than the desired state

In [ ]: