Introduction to the Python Control Systems Library (python-control)

Transfer Functions

Richard M. Murray, 30 Dec 2022

This notebook works through a transfer function-based control design and analysis, corresponding to the planar vertical takeoff and landing (PVTOL) aircraft in Astrom and Murray, Chapter 11. It is intended to demonstrate the basic functionality of the frequency domain tools in the python-control package.

We start by importing the packages that we need to carry out control calculations. The standard packages that we will use are NumPy, Matplotlib, and python-control.

In [1]:
```
# Import the packages needed for the examples included in this notebook
import numpy as np               # NumPy (array-based numerical calculations
import matplotlib.pyplot as plt  # MATLAB-like plotting functions
import control as ct             # python-control package
print(ct.__version__)
```

```
0.9.3.post2
```

Installation hints

If you get an error importing the `control` package, it may be that it is not in your current Python path. You can fix this by setting the PYTHONPATH environment variable to include the directory where the python-control package is located. If you are invoking Jupyter from the command line, try using a command of the form

```
PYTHONPATH=/path/to/control jupyter notebook
```

If you are using Google Colab, use the following command at the top of the notebook to install the `control` package:

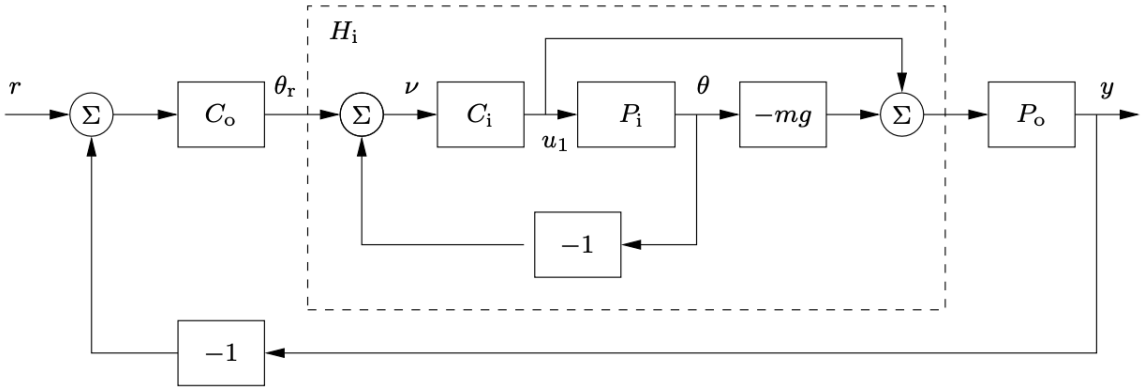
```
!pip install control
```

For the examples below, you will need version 0.9.3 or higher of the python-control toolbox. You can find the version number using the command

```
print(ct.__version__)
```

System definition

To control the lateral dynamics of the vectored thrust aircraft, we make use of an "inner/outer" loop design methodology, as illustrated below.



This diagram shows the process dynamics and controller divided into two components: an inner loop consisting of the roll dynamics and controller and an outer loop consisting of the lateral position dynamics and controller. The "inner loop" transfer function P_i (representing the roll dynamics) and an "outer loop" transfer function P_o (representing the position dynamics) are given in FBS2e, Exercise 9.11:

$$P_\mathrm{i} = \frac{r}{Js^2}, \qquad P_\mathrm{o} = \frac{1}{ms^2 + cs}$$

In python-control, transfer functions are created using the `tf` function, which takes as arguments the coefficients of the numerator and denominator polynomials of the transfer function.

In [2]:
```
# System parameters
m = 4                # mass of aircraft
J = 0.0475           # inertia around pitch axis
r = 0.25             # distance to center of force
g = 9.8              # gravitational constant
c = 0.05             # damping factor (estimated)

# Transfer functions for dynamics
Pi = ct.tf([r], [J, 0, 0])  # inner loop (roll)
Po = ct.tf([1], [m, c, 0])  # outer loop (position)
```

Control design

The approach that we take is to design a controller C_i for the inner loop so that the resulting closed loop system Hi assures that the roll angle θ follows its reference θr quickly and accurately. We then design a controller for the lateral position y that uses the approximation that we can directly control the roll angle as an input θ to the dynamics controlling the position. Under the assumption that the dynamics of the roll

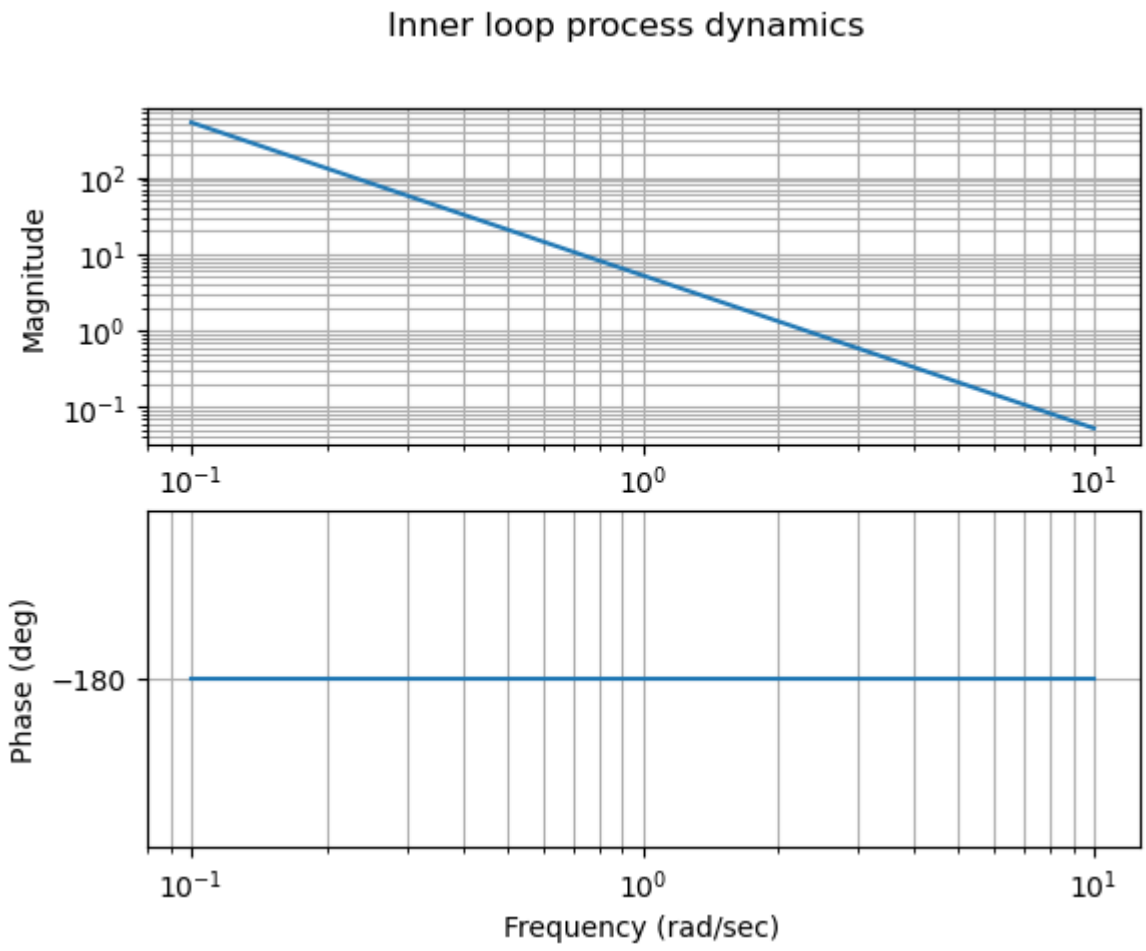
controller are fast relative to the desired bandwidth of the lateral position control, we can then combine the inner and outer loop controllers to get a single controller for the entire system. As a performance specification for the entire system, we would like to have zero steady-state error in the lateral position, a bandwidth of approximately 1 rad/s, and a phase margin of 45°.

Inner loop (pitch dynamics)

For the inner loop, we choose our design specification to provide the outer loop with accurate and fast control of the roll. The inner loop dynamics are given by

$$P_i(s) = H_{\theta,u_1}(s) = \frac{r}{Js^2}$$

In [3]:
```
# Bode plot for the open loop process
ct.bode_plot(Pi)
plt.suptitle("Inner loop process dynamics");
```

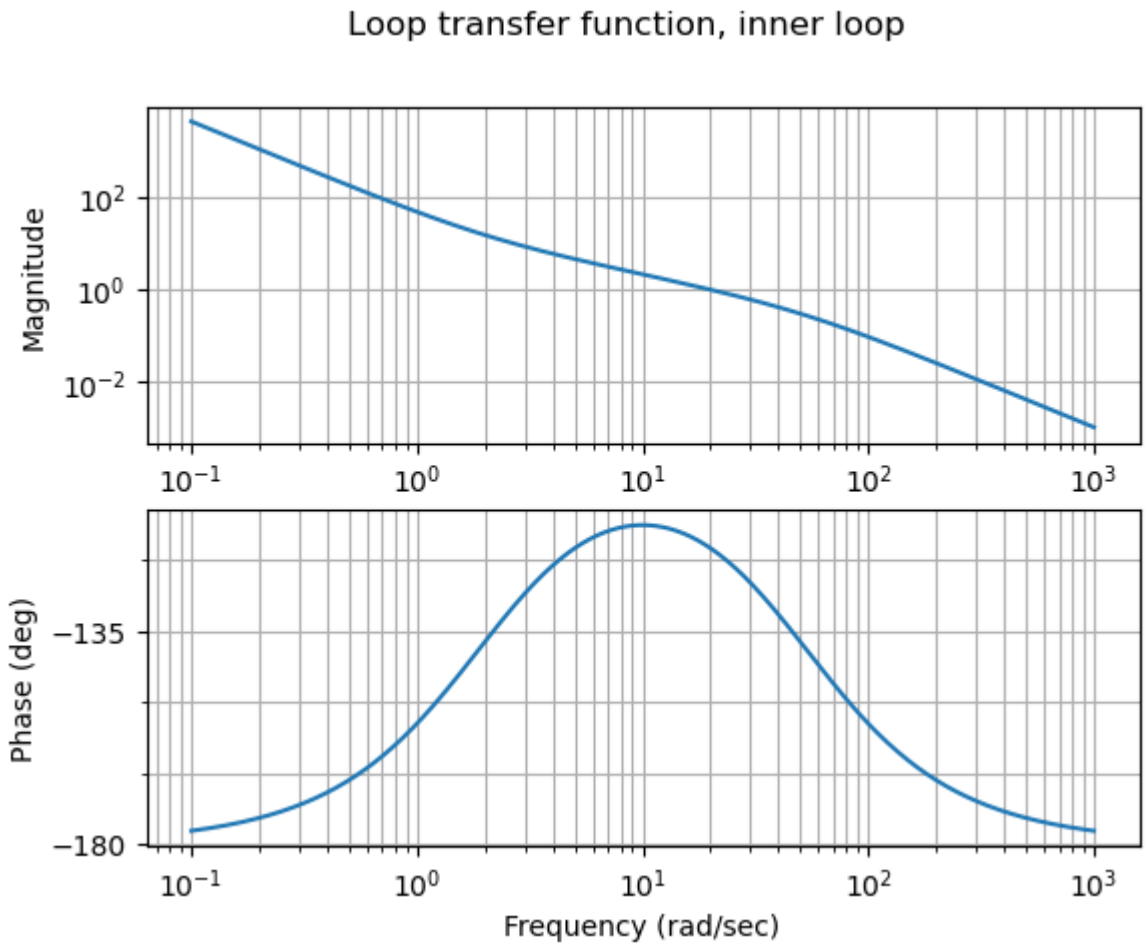


We choose the desired bandwidth to be 10 rad/s (10 times that of the outer loop) and the low-frequency error to be no more than 5%. This specification is satisfied using the lead compensator of FBS, Example 12.5, so we choose

$$C_i(s) = k\frac{s+a}{s+b}, \qquad a = 2, b = 50, k = 200.$$

In [4]:
```
# Design a simple lead controller for the system
k, a, b = 200, 2, 50
Ci = k * ct.tf([1, a], [1, b])  # lead compensator
Li = Pi * Ci

# Bode plot for the loop transfer function, with margins
ct.bode_plot(Li)
plt.suptitle("Loop transfer function, inner loop")

# Compute out the gain and phase margins
gm, pm, wcg, wcp = ct.margin(Li)
print("Gain margin:", gm)
print("Phase margin:", pm)
```

```
Gain margin: inf
Phase margin: 62.70603480451365
```

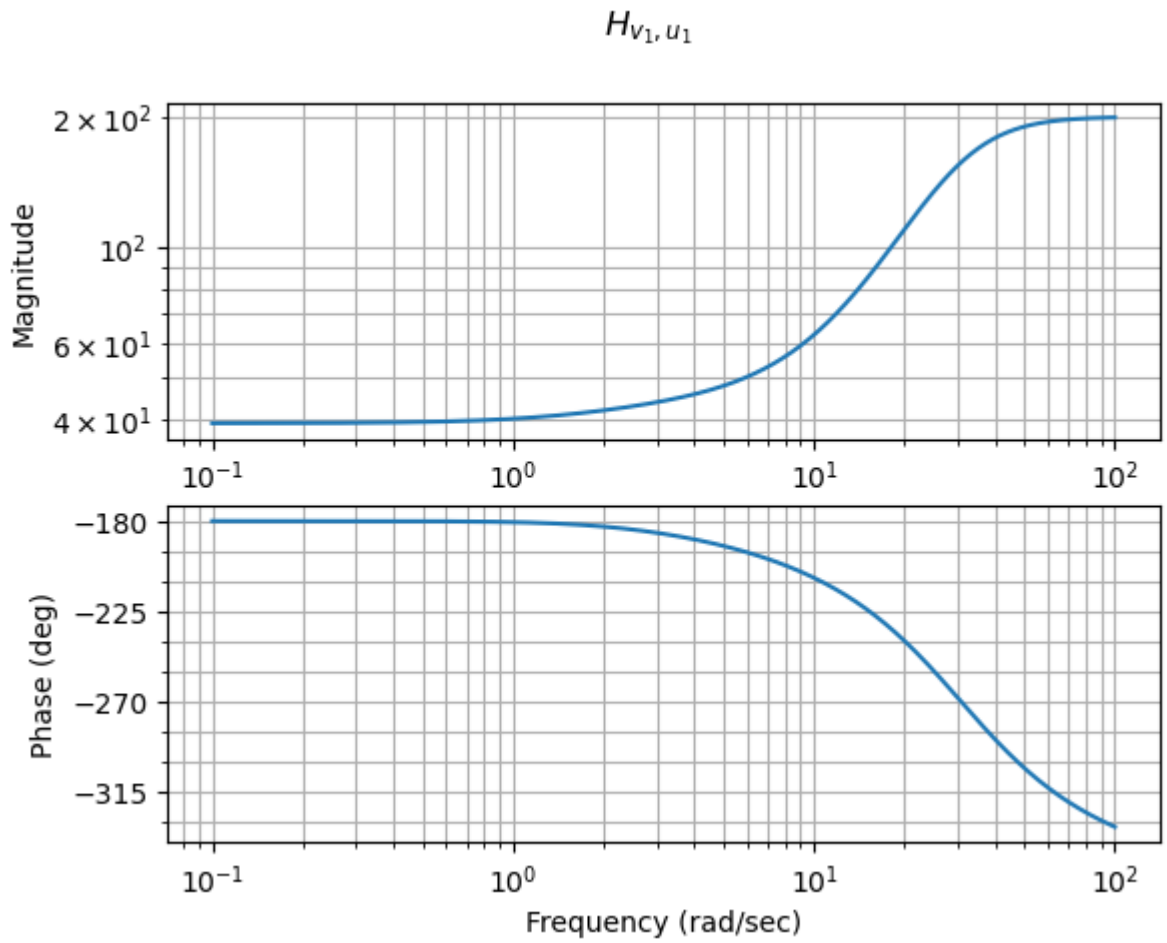


The closed loop dynamics for the system satisfy

$$H_{\mathrm{i}} = \frac{C_{\mathrm{i}}}{1 + C_{\mathrm{i}}P_{\mathrm{i}}} - mg\frac{C_{\mathrm{i}}P_{\mathrm{i}}}{1 + C_{\mathrm{i}}P_{\mathrm{i}}} = \frac{C_{\mathrm{i}}(1 - mgP_{\mathrm{i}})}{1 + C_{\mathrm{i}}P_{\mathrm{i}}}.$$

A plot of the magnitude of this transfer function is shown below, and we see that Hi ≈ −mg = −39.2 is a good approximation up to 10 rad/s.

In [5]:
```
# Compute out the actual transfer function from u1 to v1 (see L8.2 notes)
# Hi = Ci*(1-m*g*Pi)/(1+Ci*Pi)
Hi = ct.parallel(ct.feedback(Ci, Pi), -m * g *ct.feedback(Ci * Pi, 1))

ct.bode_plot(Hi)
plt.suptitle("$H_{v_1, u_1}$");
```



Outer loop (lateral dynamics)

To design the outer loop controller, we assume the inner loop roll control is perfect, so that we can take θr as the input to our lateral dynamics. Using the block diagram at the top of the notebook, the outer loop dynamics can be written as

$$P(s) = H_{\mathrm{i}}(0)P_{\mathrm{o}}(s) = \frac{H_{\mathrm{i}}(0)}{ms^2 + cs},$$

where we replace $H_{\mathrm{i}}(s)$ with $H_{\mathrm{i}}(0)$ to reflect our approximation that the inner loop will eventually track our commanded input. Of course, this approximation may not be valid, and so we must verify this when we complete our design.

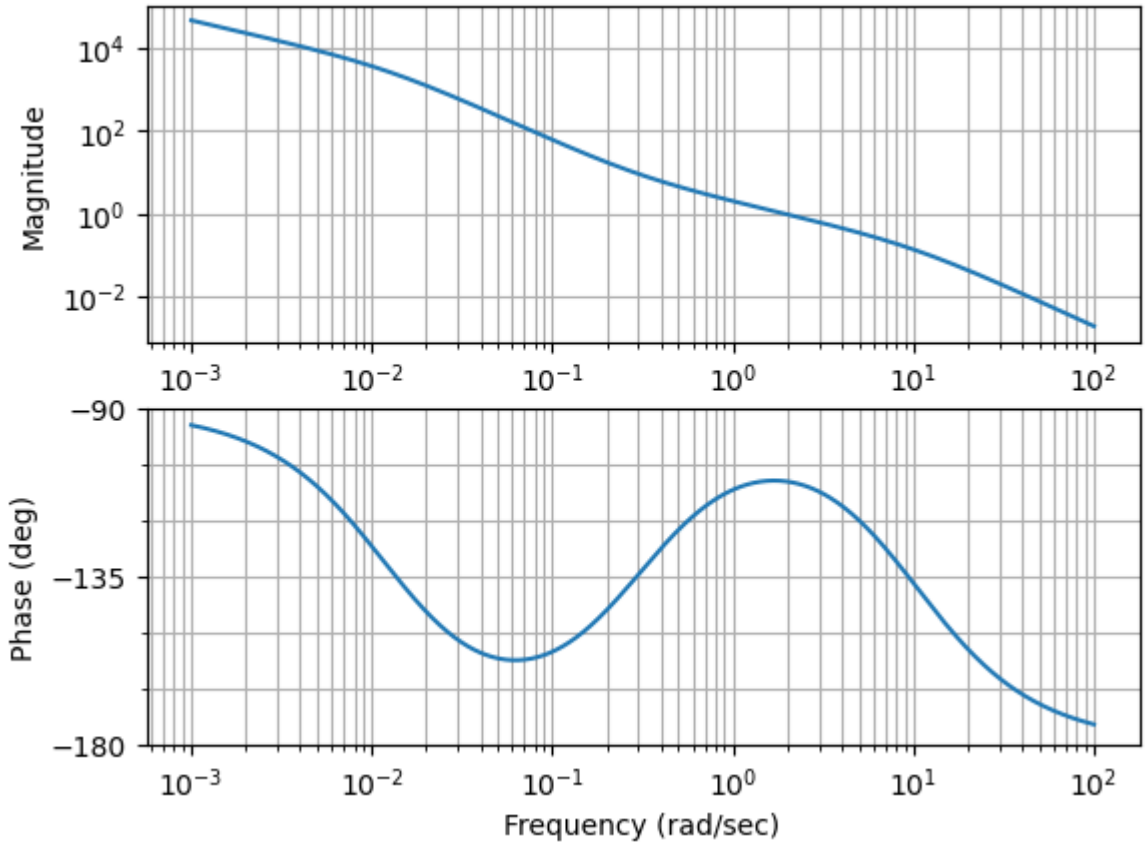
Our control goal is now to design a controller that gives zero steady-state error in y for a step input and has a bandwidth of 1 rad/s. The outer loop process dynamics are given by a double integrator, and we can again use a simple lead compensator to satisfy the specifications. We also choose the design such that the loop transfer function for the outer loop has $|L_\mathrm{o}| < 0.1$ for $\omega > 10$ rad/s, so that the Hi high-frequency dynamics can be neglected. We choose the controller to be of the form

$$C_\mathrm{o}(s) = -k_\mathrm{o}\frac{s + a_\mathrm{o}}{s + b_\mathrm{o}},$$

with the negative sign to cancel the negative sign in the process dynamics. To find the location of the poles, we note that the phase lead flattens out at approximately $b_\mathrm{o}/10$. We desire phase lead at crossover, and we desire the crossover at $\omega_\mathrm{gc} = 1$ rad/sec, so this gives $b_\mathrm{o} = 10$. To ensure that we have adequate phase lead, we must choose a_o such that $b_\mathrm{o}/10 < 10a_\mathrm{o} < b_\mathrm{o}$, which implies that a_o should be between 0.1 and 1. We choose $a_\mathrm{o} = 0.3$. Finally, we need to set the gain of the system such that at the desired crossover frequency the loop gain has magnitude 1 or more. A simple calculation shows that $k_\mathrm{o} = 2$ satisfies this objective. Thus, the final outer loop controller becomes

$$C_\mathrm{o}(s) = -2\frac{s + 0.3}{s + 10}.$$

In [6]:
```
# Now design the lateral control system
a, b, K = 0.02, 5, 2
Co = -K * ct.tf([1, 0.3], [1, 10])  # another lead compensator
Lo = -m*g*Po*Co

ct.bode_plot(Lo);  # margin(Lo)
```



Combined dynamics

Finally, we can combine the inner and outer loop controllers and verify that the system has the desired closed loop performance. The Bode and Nyquist plots with inner and outer loop controllers are shown below, and we see that the specifications are satisfied.

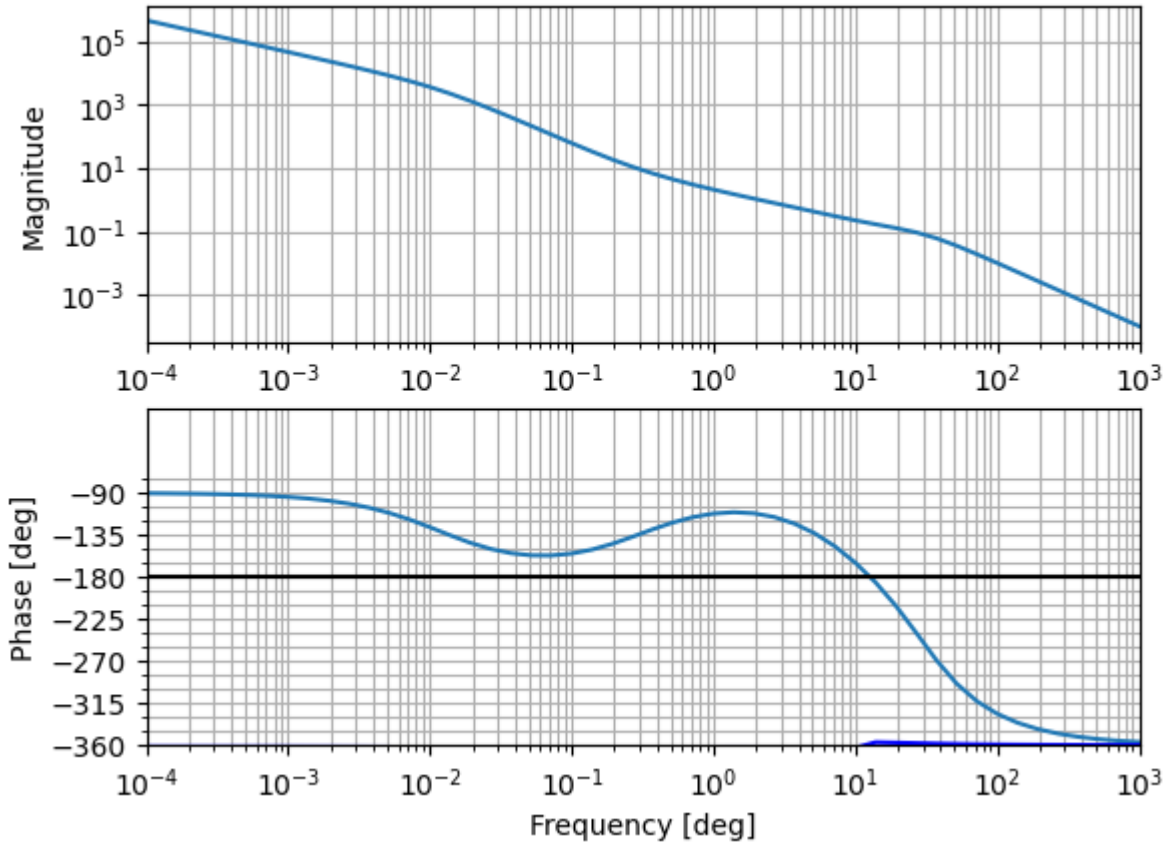
In [7]:
```
# Finally compute the real outer-loop loop gain + responses
L = Co * Hi * Po
S = ct.feedback(1, L)
T = ct.feedback(L, 1)

# Compute stability margins
gm, pm, wgc, wpc = ct.margin(L)
print("Gain margin: %g at %g" % (gm, wgc))
print("Phase margin: %g at %g" % (pm, wpc))
```

```
Gain margin: 5.56996 at 12.7395
Phase margin: 67.3933 at 2.08419
```

Bode plot

In [8]:
```
ct.bode_plot(L, np.logspace(-4, 3))

# Add crossover line to the magnitude plot
#
# Note: in matplotlib before v2.1, the following code worked:
#
```

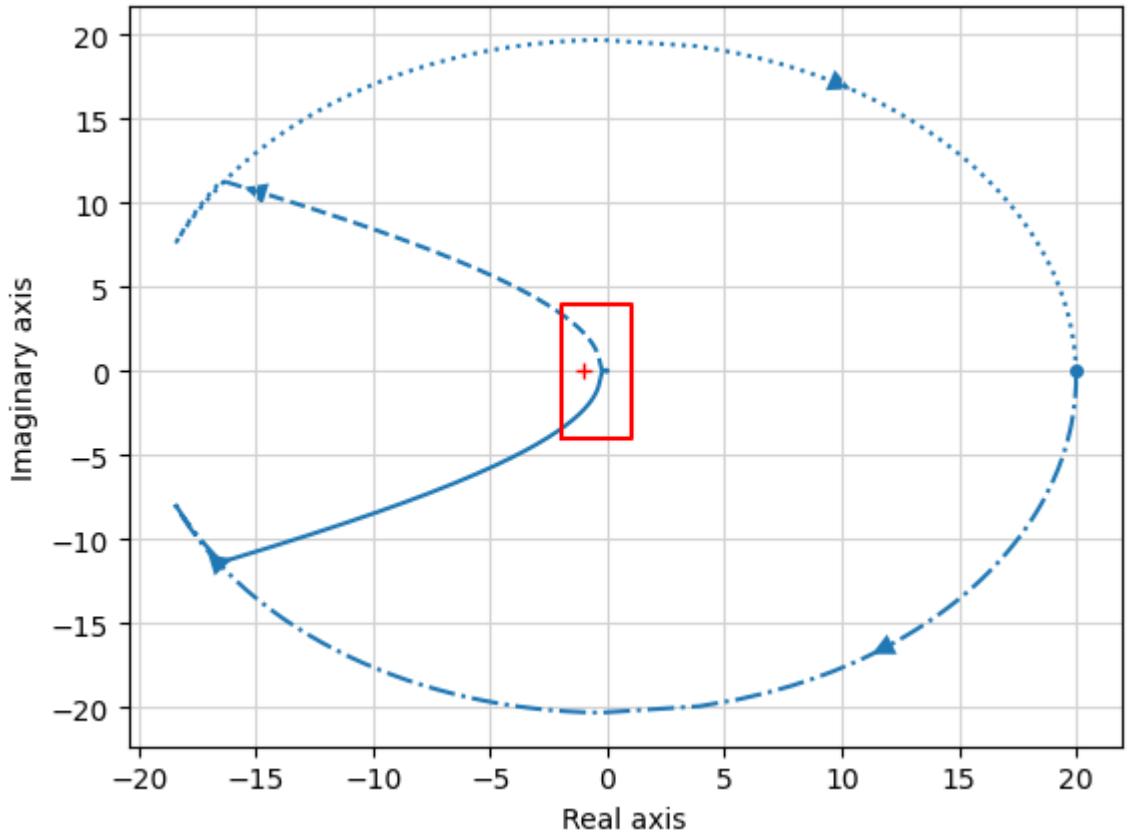
```
#   plt.subplot(211); hold(True);
#   loglog([1e-4, 1e3], [1, 1], 'k-')
#
# In later versions of matplotlib the call to plt.subplot will clear the
# axes and so we have to extract the axes that we want to use by hand.
# In addition, hold() is deprecated so we no longer require it.
#
for ax in plt.gcf().axes:
    if ax.get_label() == 'control-bode-magnitude':
        break
ax.semilogx([1e-4, 1e3], 20*np.log10([1, 1]), 'k-')

#
# Replot phase starting at -90 degrees
#
# Get the phase plot axes
for ax in plt.gcf().axes:
    if ax.get_label() == 'control-bode-phase':
        break

# Recreate the frequency response and shift the phase
mag, phase, w = ct.freqresp(L, np.logspace(-4, 3))
phase = phase - 360

# Replot the phase by hand
ax.semilogx([1e-4, 1e3], [-180, -180], 'k-')
ax.semilogx(w, np.squeeze(phase), 'b-')
ax.axis([1e-4, 1e3, -360, 0])
plt.xlabel('Frequency [deg]')
plt.ylabel('Phase [deg]');
```



In [9]:
```
#### Nyquist plot
```

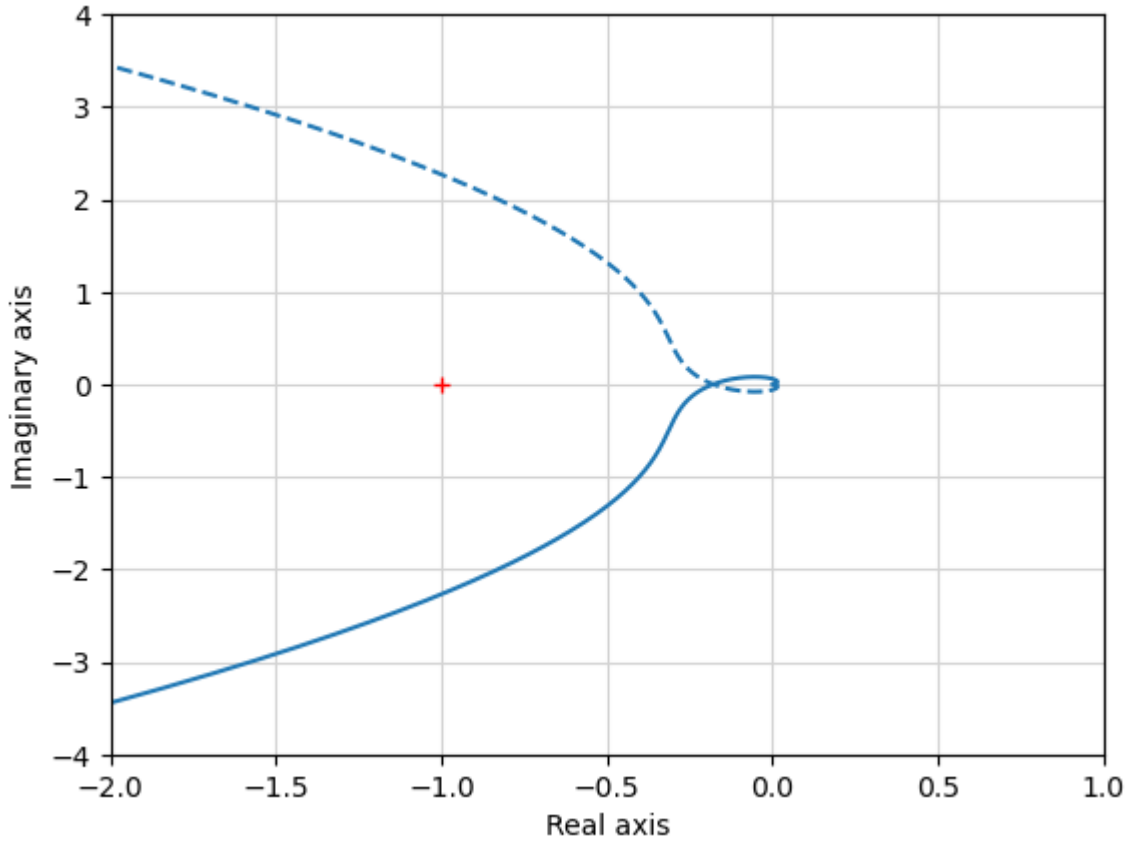
In [10]:
```
# Nyquist plot for complete design
ct.nyquist_plot(L)

# Add a box in the region we are going to expand
plt.plot([-2, -2, 1, 1, -2], [-4, 4, 4, -4, -4], 'r-')
```

Out[10]:
```
[<matplotlib.lines.Line2D at 0x7f9f804775b0>]
```



In [11]:
```
# Expanded region
ct.nyquist_plot(L)
plt.axis([-2, 1, -4, 4])
```

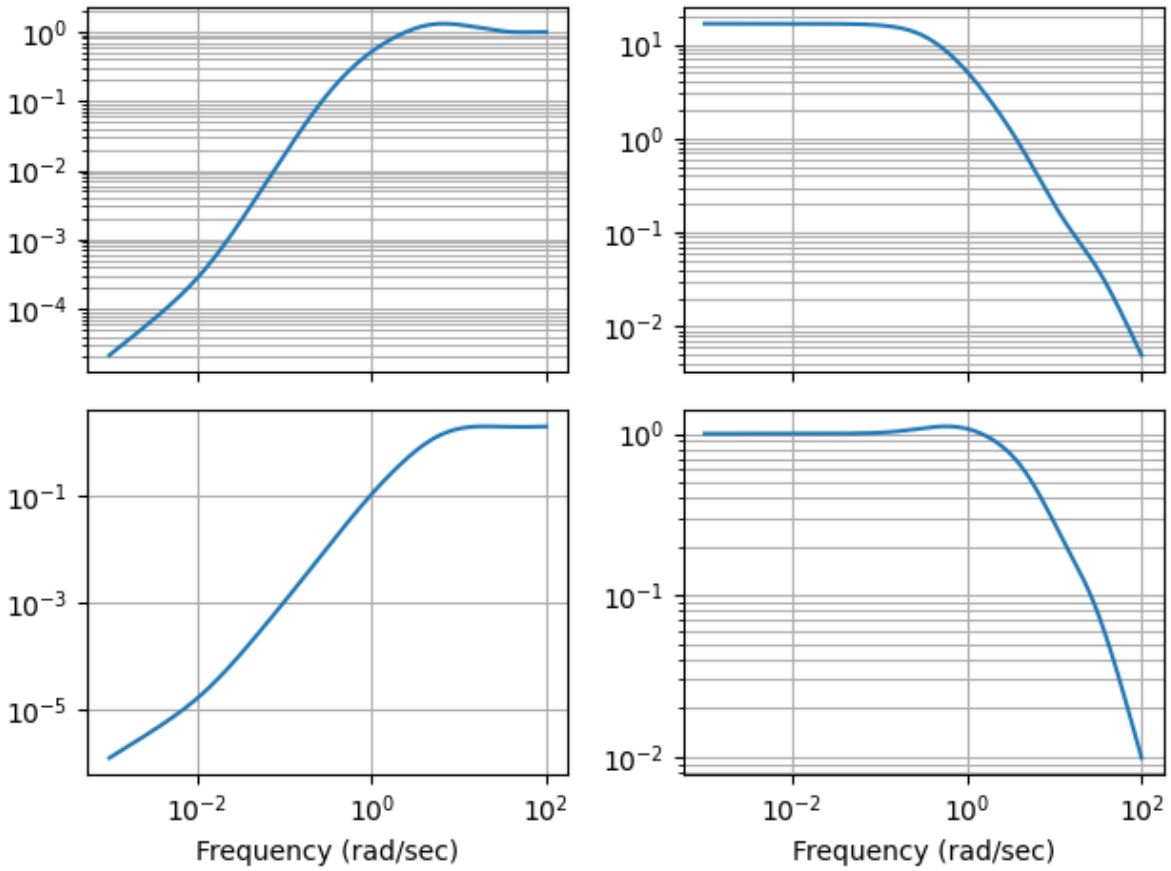
Out[11]:
```
(-2.0, 1.0, -4.0, 4.0)
```



Gang of Four

The gain curves of the Gang of Four show that the transfer functions between all inputs and outputs are reasonable. The sensitivity to load disturbances PS is large at low frequency because the controller does not have integral action.

In [12]:
```
ct.gangof4_plot(Hi * Po, Co)
```

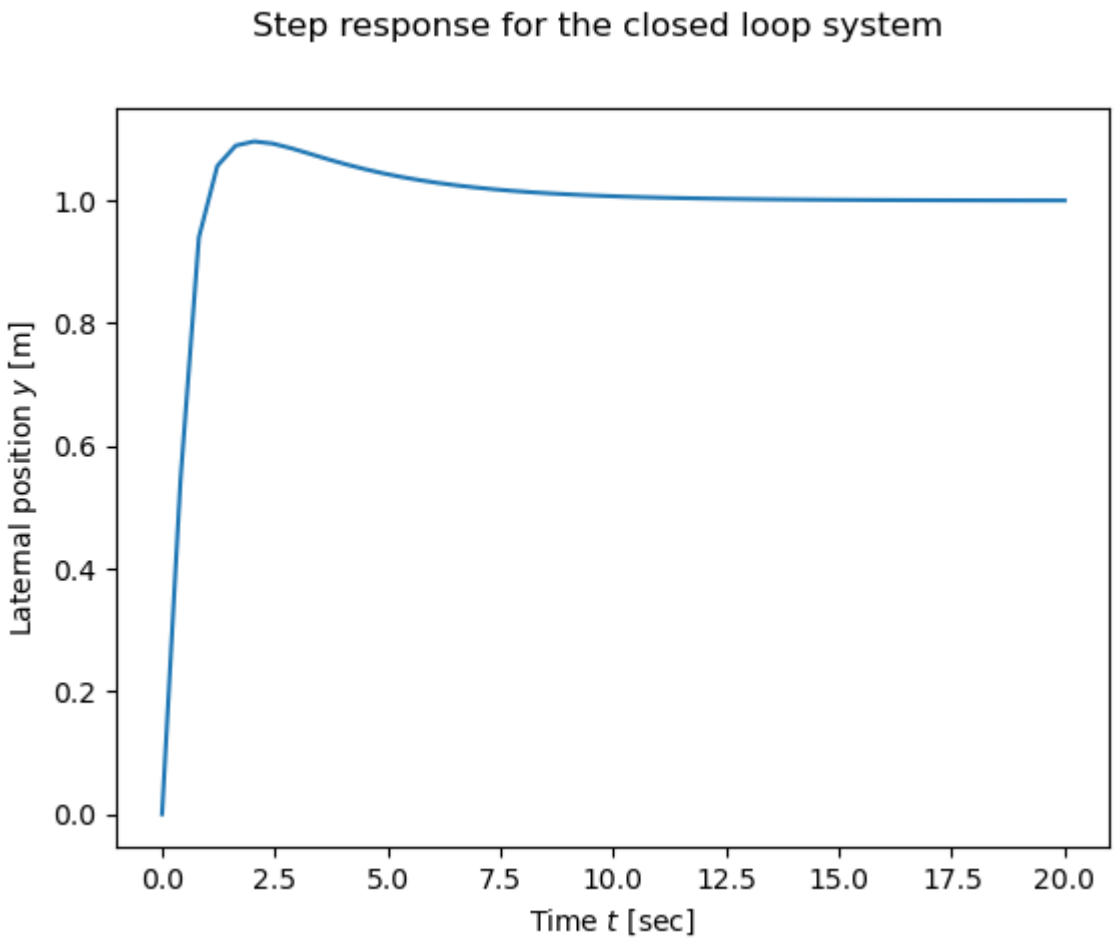


Step response

The step response for the system shows that we obtain a fast transition in the lateral position of the aircraft, with small overshoot.

In [13]:
```
Tvec, Yvec = ct.step_response(T, np.linspace(0, 20))
plt.plot(Tvec, Yvec)
plt.xlabel("Time $t$ [sec]")
plt.ylabel("Laternal position $y$ [m]")
plt.suptitle("Step response for the closed loop system")
```

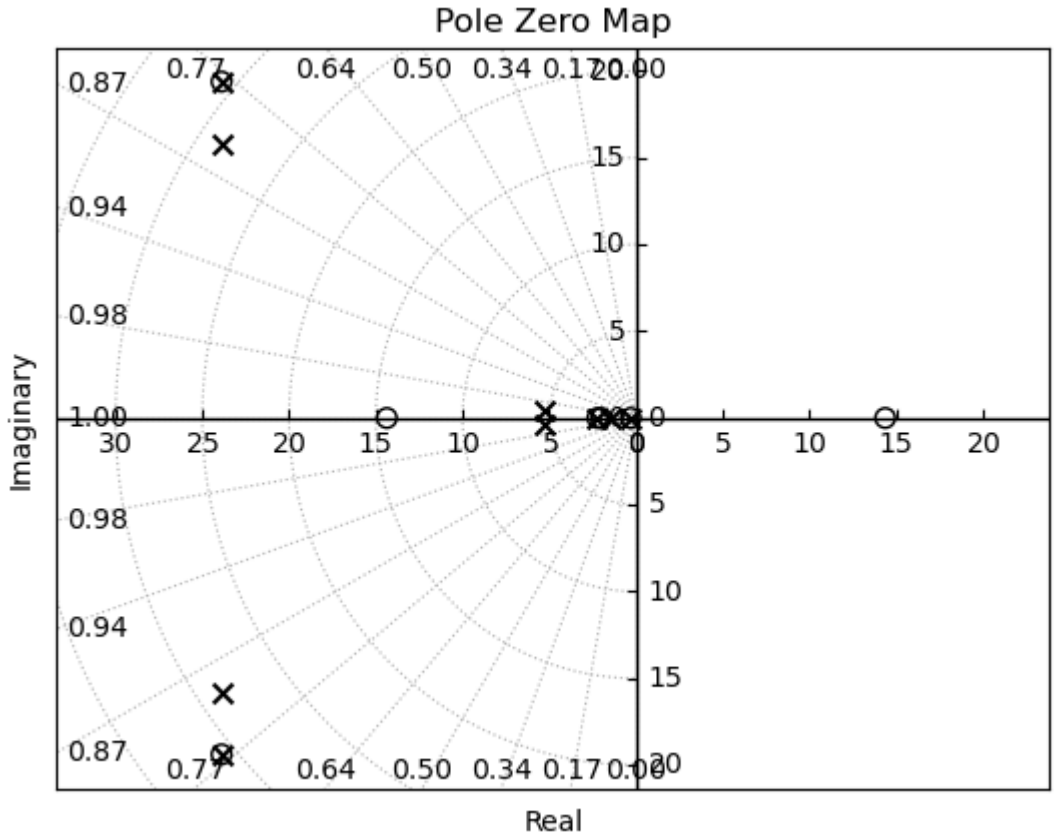
Out[13]:
```
Text(0.5, 0.98, 'Step response for the closed loop system')
```



Pole/zero map

In [14]:
```
P, Z = ct.pzmap(T, plot=True, grid=True)
print("Closed loop poles and zeros: ", P, Z)
```

```
Closed loop poles and zeros:  [-23.8876929 +19.38370205j -23.8876929 -19.38
370205j
 -23.83489459+15.78456798j -23.83489459-15.78456798j
  -5.23195887 +0.4117018j   -5.23195887 -0.4117018j
  -2.22461421 +0.j          -1.51604555 +0.j
  -0.36274752 +0.j        ] [-23.8876929 +19.38370205j -23.8876929 -19.3837
0205j
  14.36369693 +0.j         -14.36369693 +0.j
  -2.22461421 +0.j          -2.         +0.j
  -0.3        +0.j        ]
```


Pole Zero Map
Imaginary
Real
0.87
0.94
0.98
1.00
0.77
0.64
0.50
0.34
0.17
0.00
30
25
20
15
10
5
0
5
10
15
20

In [ ]: