Control with Python

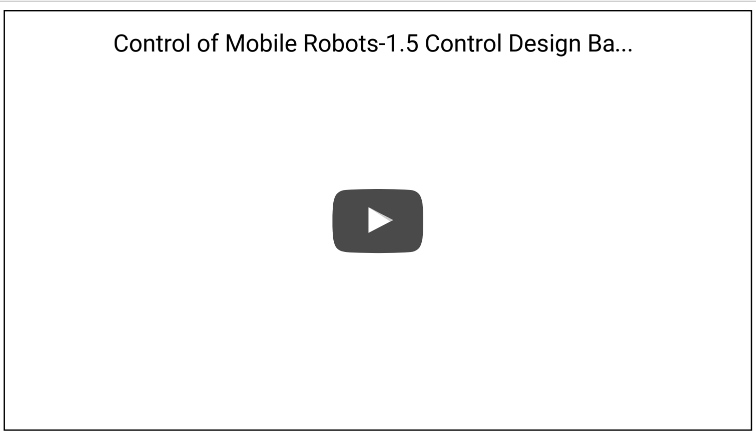
Prof. Seungchul Lee
iSystems (http://isystems.unist.ac.kr/)
UNIST

Table of Contents

1. PID Control

- 1.5 Control Design Basics | Control of Mobile Robots

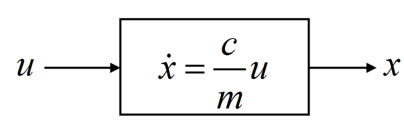
In [1]:
```
%%html
<iframe src="https://www.youtube.com/embed/DJuo9kLdr4M?list=PLp8ijpvp8iCvFDYdcXqqYU5Ibl_aOqwjr"
width="560" height="315" frameborder="0" allowfullscreen></iframe>
```



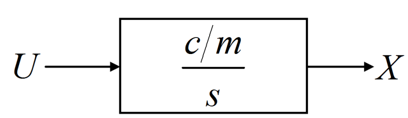
For the given car model

$$\dot{x} = \frac{c}{m}u \quad \text{for the velocity of a car, } x$$

In a bloack diagram



in a Laplace transform



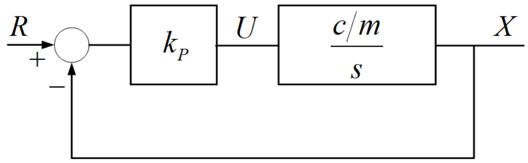
We want to achieve

$$x \to r \quad \text{as} \quad t \to \infty \;(e = r - x \to 0)$$

Attempt 2: P Regulator

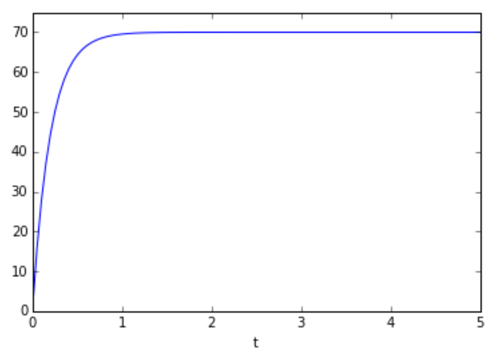
$$u = ke$$

- small error yeilds small control signals
- nice and smooth
- so-called proportional regulation (P regulator)



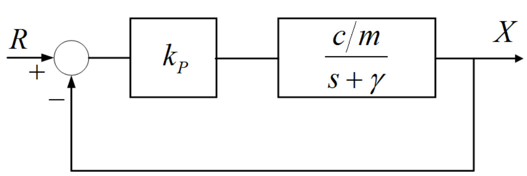
In [2]:
```
import control
import numpy as np
import matplotlib.pyplot as plt
from control import *

%matplotlib inline
```

In [3]:
```
c = 1
m = 1

G = tf(c/m,[1,0])

k = 5
C = k

Gcl = feedback(C*G,1,-1)

t = np.linspace(0,5,100)
r = 70*np.ones(t.shape)
x0 = 0

[y,tout,x] = lsim(Gcl,r,t,x0)

plt.plot(tout,y)
plt.xlabel('t')
plt.ylim(0,75)

plt.show()
```



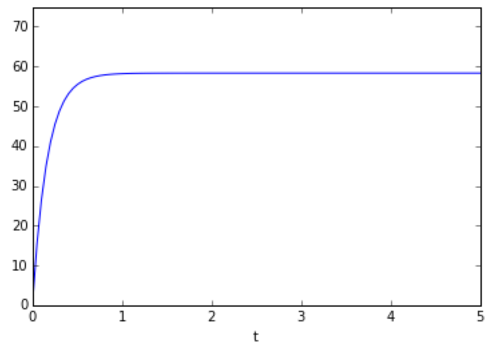
What if the true system is:

$$\dot{x} = \frac{c}{m}u - \gamma x$$



In [4]:

```
gamma = 1
Gtr = tf(c/m,[1,gamma])
C = k

Gcl = feedback(C*Gtr,1,-1)

x0 = 0
t = np.linspace(0,5,100)
r = 70*np.ones(t.shape)

[y,tout,x] = lsim(Gcl,r,t,x0)

plt.plot(tout,y)
plt.xlabel('t')
plt.ylim(0,75)

plt.show()
```

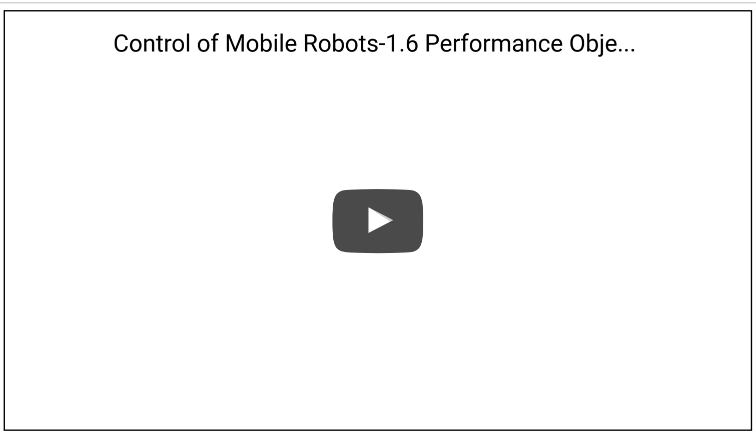


Attempt 3

- 1.6 Performance Objectives | Control of Mobile Robots
- 1.7 PID Control | Control of Mobile Robots

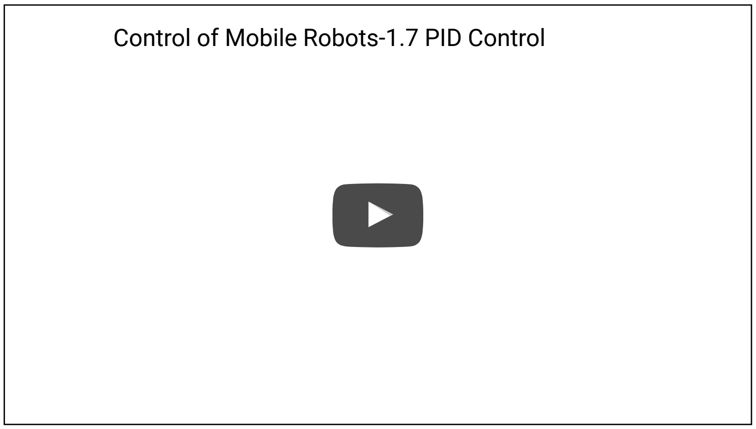
In [5]:

```
%%html
<iframe src="https://www.youtube.com/embed/cQhqx65kLfM?list=PLp8ijpvp8iCvFDYdcXqqYU5Ibl_aOqwjr"
width="560" height="315" frameborder="0" allowfullscreen></iframe>
```



In [6]:

```
%%html
<iframe src="https://www.youtube.com/embed/Mk1ygHj4zxw?list=PLp8ijpvp8iCvFDYdcXqqYU5Ibl_aOqwjr"
width="560" height="315" frameborder="0" allowfullscreen></iframe>
```



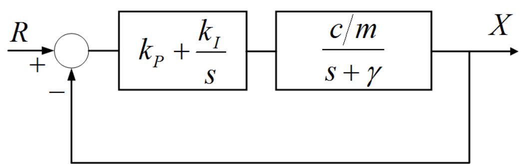
$$u = ke + \gamma \frac{m}{c} x$$

However, all of sudden we have to know all these physical parameters that we typically do not know - not robust !!!

Attempt 4: PI Regulators

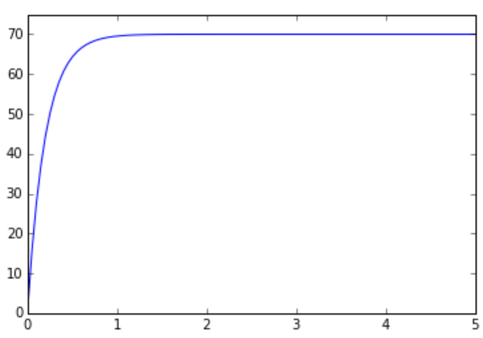
- Stability (BIBO)
- Tracking
- Robustness

$$u(t) = k_P\, e(t) + k_I \int_0^t e(\tau) d\tau$$



In [7]:

```
Gtr = tf(c/m,[1,gamma])
kP = 5
kI = 5
C = tf([kP,kI],[1,0])

Gcl = feedback(C*Gtr,1,-1)

x0 = 0
t = np.linspace(0,5,100)
r = 70*np.ones(t.shape)

[y,tout,x] = lsim(Gcl,r,t,x0)

plt.plot(tout, y)
plt.xlabel('t')
plt.ylim(0,75)

plt.show()
```

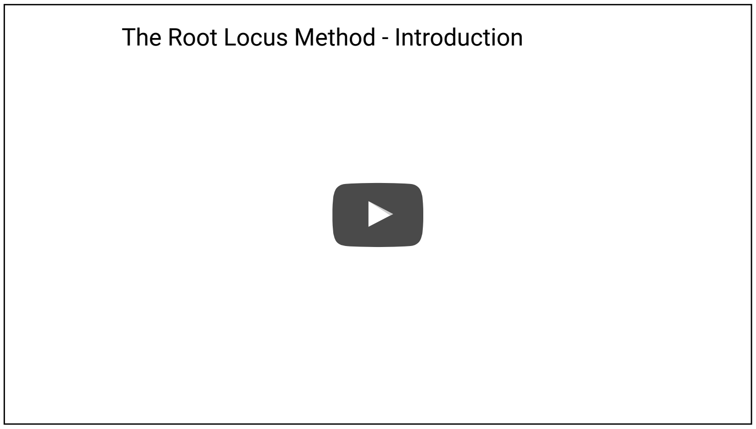


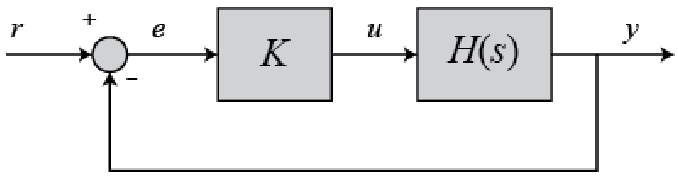
2. Root Locus

- The Root Locus Method by Brian Douglas
- from umich control (http://ctms.engin.umich.edu/CTMS/index.php?example=Introduction&section=ControlRootLocus)

In [8]:

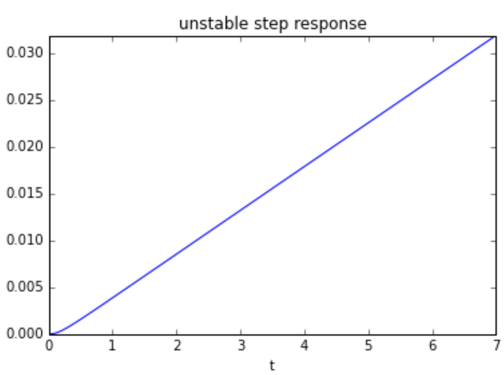
```
%%html
<iframe src="https://www.youtube.com/embed/CRvVDoQJjYI?list=PLUMWjy5jgHK1NC52DXXrriwihVrYZKqjk"
width="560" height="315" frameborder="0" allowfullscreen></iframe>
```





In [9]:

```
sys = tf([1,7],[1,0])*tf(1,[1,5])*tf(1,[1,15])*tf(1,[1,20])

[y, tout] = step(sys)

plt.plot(tout, y)
plt.xlabel('t')
plt.axis('tight')
plt.title('unstable step response')

plt.show()
```



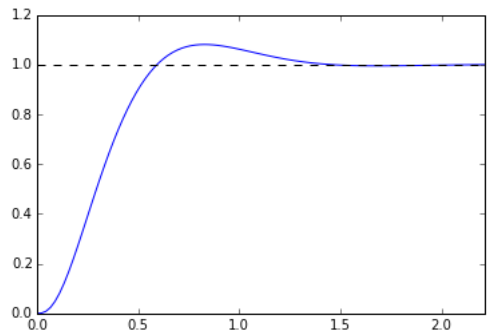
In [10]:

```
print(pole(sys))
```

```
[-20. -15.  -5.   0.]
```

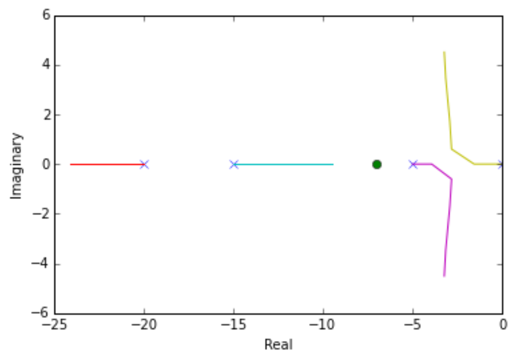
In [11]:

```
G = sys

# k = 10
k = 800
Gcl = feedback(k*G,1,-1)

[y,tout] = step(Gcl)

plt.plot(tout,y)
plt.plot(tout,np.ones(tout.shape),'k--')
plt.axis('tight')
plt.ylim(0.0,1.2)

plt.show()
```



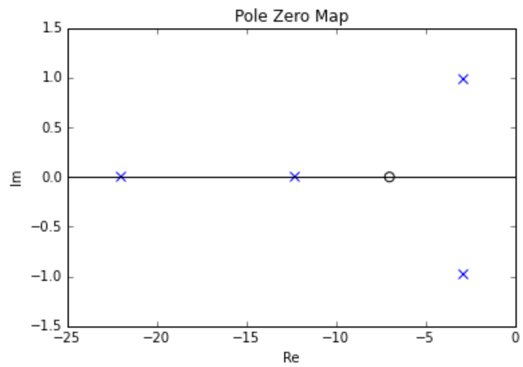
In [12]:

```
rlocus(G)
plt.show()
```



In [34]:

```
K = 350
sys_cl = feedback(K*sys,1,-1) #negative feedback

pzmap.pzmap(sys_cl)
plt.show()
```



In [14]:

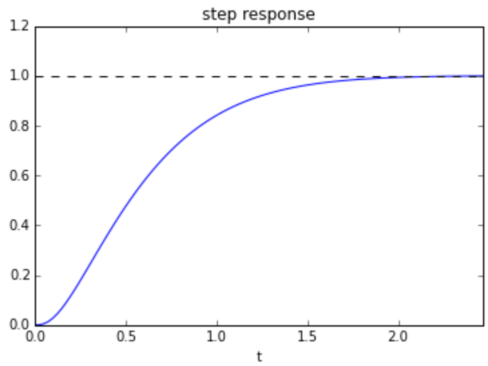
```
sys_cl
```

Out[14]:

```
             350 s + 2450
-------------------------------------
s^4 + 40 s^3 + 475 s^2 + 1850 s + 2450
```

In [15]:

```
[y,tout] = step(sys_cl)
plt.plot(tout,y)
plt.plot(tout,np.ones(tout.shape), 'k--')
plt.axis('tight')
plt.ylim(0,1.2)
plt.xlabel('t')
plt.title('step response')
plt.show()
```

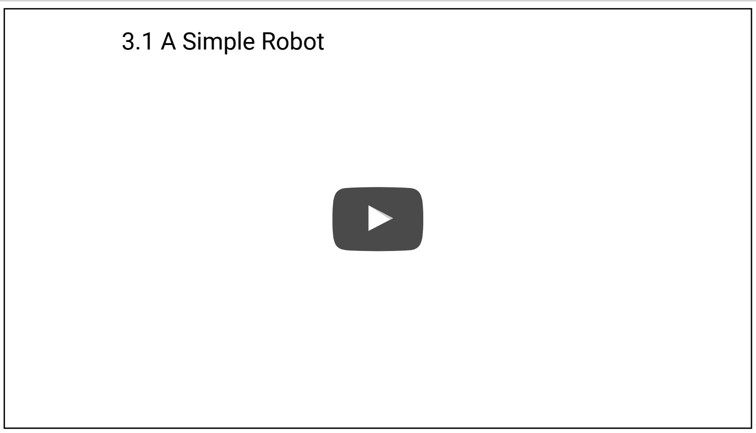


3. State Space Representation

- from 3.1 A Simple Robot | Control of Mobile Robots
- from 3.2 State Space Models | Control of Mobile Robots

In [16]:

```
%%html
<iframe src="https://www.youtube.com/embed/kQNUpNh6nBc?list=PLp8ijpvp8iCvFDYdcXqqYU5Ibl_aOqwjr"
width="560" height="315" frameborder="0" allowfullscreen></iframe>
```



In [17]:

```
%%html
<iframe src="https://www.youtube.com/embed/W6AUOyj5bFA?list=PLp8ijpvp8iCvFDYdcXqqYU5Ibl_aOqwjr"
width="560" height="315" frameborder="0" allowfullscreen></iframe>
```

Controlling a point mass

Given a point mass on a line whose acceleration is directly controlled:

$$\ddot{p} = u$$

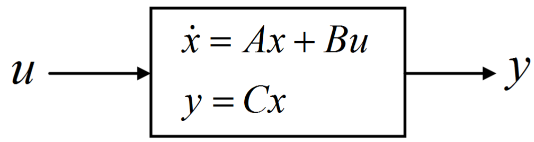
want to write this on a compact/general form

$$\dot{x}_1 = x_2$$
$$\dot{x}_2 = u$$

on a state space form

$$\dot{x} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u$$

$$y = p = x_1 = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$



3.1. The car model

If we care about/can measure the velocity:

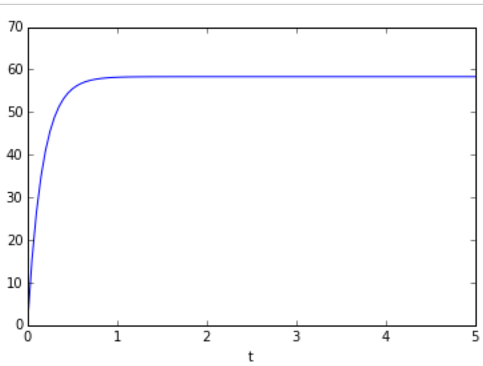
$$A = -\gamma, \qquad B = \frac{c}{m}, \qquad C = 1$$

If we care about/can measure the position we have the same general equation with different matrices:

$$A = \begin{bmatrix} 0 & 1 \\ 0 & -\gamma \end{bmatrix}, \qquad B = \begin{bmatrix} 0 \\ \frac{c}{m} \end{bmatrix}, \qquad C = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

In [18]:

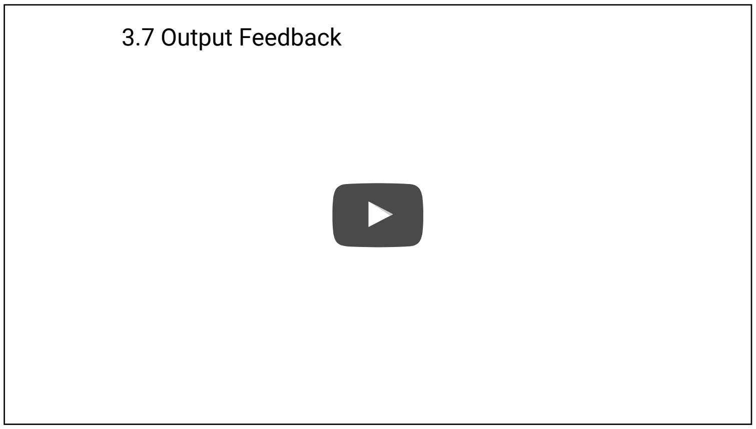
```
#system in ss
c = 1
m = 1
gamma = 1

A = -gamma
B = c/m
C = 1
D = 0

Gss = ss(A,B,C,D)

# P controller
k = 5
C = k

# close loop
Gcl = feedback(C*Gss,1,-1)

x0 = 0
t = np.linspace(0,5,100)
r = 70*np.ones(t.shape)

[y,tout,x] = lsim(Gcl,r,t,x0)

plt.plot(tout,y)
plt.xlabel('t')
plt.ylim(0,70)

plt.show()
```

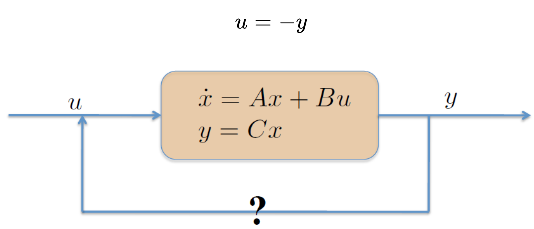


3.2. Back to the World's Simplest Robot (Output Feedback)

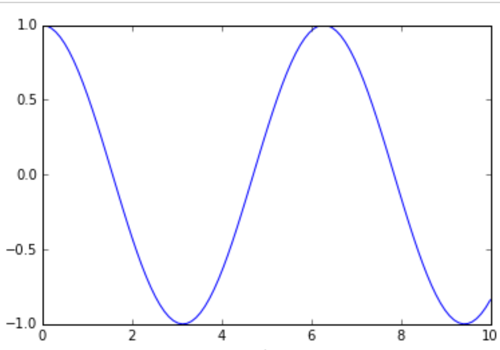
- from 3.7 Output Feedback | Control of Mobile Robots

In [19]:
```
%%html
<iframe src="https://www.youtube.com/embed/HmqOnsRH73w?list=PLp8ijpvp8iCvFDYdcXqqYU5Ibl_aOqwjr"
width="560" height="315" frameborder="0" allowfullscreen></iframe>
```





In [20]:
```
# to move towards the origin
# u = - y

A = np.array([[0, 1],[0,0]])
B = np.array([[0],[1]])
C = [1,0]
D = 0

G = ss(A,B,C,D)

K = 1
x0 = np.array([[1,0]]).T
t = np.linspace(0,10,100)
r = np.zeros(t.shape)

Gcl = feedback(G,K,-1)

[y,tout,x] = lsim(Gcl,r,t,x0)

plt.plot(tout,y)
plt.xlabel('t')

plt.show()
```

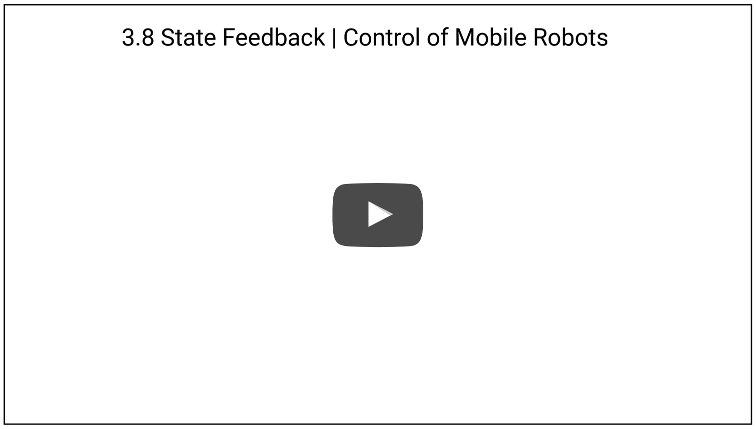


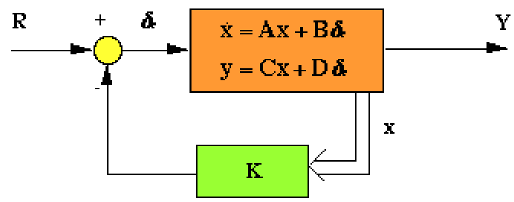
4. State Feedback

- from 3.8 State Feedback | Control of Mobile Robots

In [21]:

```
%%html
<iframe src="https://www.youtube.com/embed/yl5IiJOYQps?list=PLciAw3uhNCiD3dkLTPJgHoMnsu8XgCt1m"
width="560" height="315" frameborder="0" allowfullscreen></iframe>
```

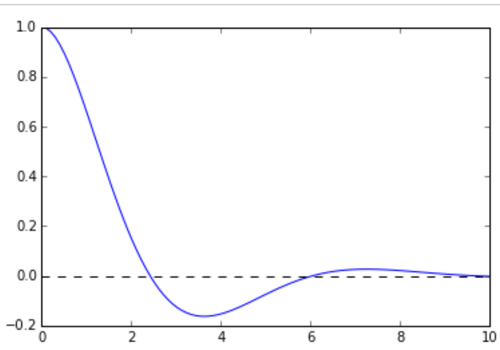




To move forwards origin, $R = 0$

In [22]:

```
A = np.array([[0,1],[0,0]])
B = np.array([[0],[1]])
C = [1,0]
D = 0

G = ss(A,B,C,D)

k1 = 1
k2 = 1
K = [k1,k2]

Gcl = ss(A-B*K,B,C,D)

x0 = np.array([[1,0]]).T
t = np.linspace(0,10,100);
r = np.zeros(t.shape)

[y,tout,x] = lsim(Gcl,r,t,x0)

plt.plot(tout,y)
plt.plot(tout,np.zeros(tout.shape),'k--')
plt.xlabel('t')

plt.show()
```



In [23]:

```
print(np.linalg.eigvals(Gcl.A))
```

```
[-0.5+0.8660254j -0.5-0.8660254j]
```

Eigenvalues Matter

- It is clear that some eigenvalues are better than others. Some cause oscillations, some make the system respond too slowly, and so forth ...
- In the next module we will see how to select eigenvalues and how to pick control laws based on the output rather than the state.

4.1. Pole Placement

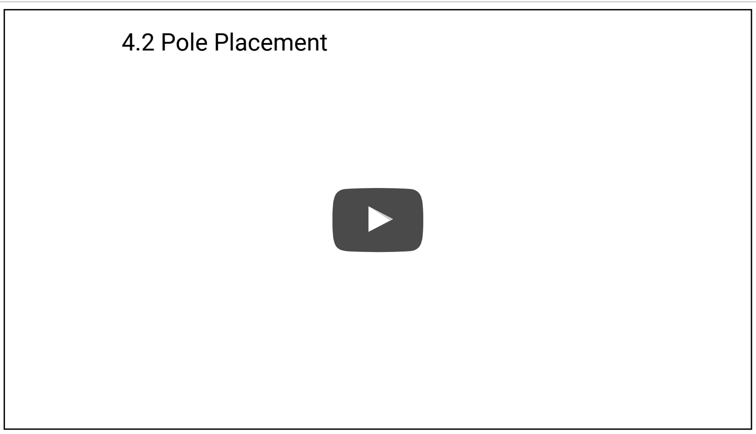
- from 4.1 Stabilizing the Point Mass | Control of Mobile Robots
- from 4.2 Pole Placement | Control of Mobile Robots

In [24]:

```
%%html
<iframe src="https://www.youtube.com/embed/S4WZTmEnbrY?list=PLp8ijpvp8iCvFDYdcXqqYU5Ibl_aOqwjr"
width="560" height="315" frameborder="0" allowfullscreen></iframe>
```

In [25]:

```
%%html
<iframe src="https://www.youtube.com/embed/5tWhOK8Klo0?list=PLp8ijpvp8iCvFDYdcXqqYU5Ibl_aOqwjr"
width="560" height="315" frameborder="0" allowfullscreen></iframe>
```



back to the point-mass, again

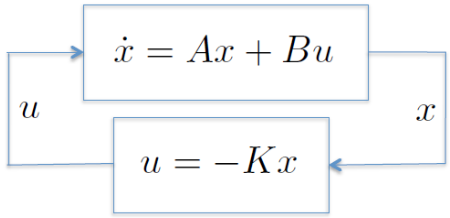
$$u = -Kx \rightarrow \dot{x} = (A - BK)x$$

$$A - BK = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} k_1 & k_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -k_1 & -k_2 \end{bmatrix}$$

$$\begin{vmatrix} 0 & 1 \\ -k_1 & -k_2 \end{vmatrix} = \lambda^2 + \lambda k_2 + k_1$$

Desired Eigenvalues: let's pick both eigenvalues at -1

$$(\lambda + 1)(\lambda + 1) = \lambda^2 + 2\lambda + 1$$
$$k_1 = 2, k_2 = 1$$



Pick the control gains such that the eigenvalues (poles) of the closed loop system match the desired eigenvalues

Questions

- Is this always possible? (No)
- How should we pick the eigenvalues? (Mix of art and science)

$$\dot{x} = \begin{bmatrix} 2 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} u$$
$$A - BK = \begin{bmatrix} 2-k_1 & -k_2 \\ 1-k_1 & 1-k_2 \end{bmatrix}$$
$$\varphi = \lambda^2 + \lambda(-3+k_1+k_2) + 2 - k_1 - k_2$$

Suppose

$$\varphi = (\lambda+1)^2 = \lambda^2 + \lambda(-3+k_1+k_2) + 2 - k_1 - k_2$$

Let's pick both eigenvalues at -1

$$-3 + k_1 + k_2 = 2 \quad \text{and} \quad 2 - k_1 - k_2 = 1$$

$\rightarrow$ no k_1 and k_2 exist

What's at play here is a lack of `controllability`, i.e., the effect of the input is not sufficiently rich to influence the system enough

In [26]:

```
A = np.array([[2,0],[1,-1]])
B = np.array([[1],[1]])
C = [1,0]

P = np.array([[-0.5+1j,0.5-1j]])
# P = [-0.1 + 1j, -0.1 - 1j];
# P = [-0.5, -1];
# P = [-5, -4];

#K = place(A,B,P)
# place command is not available yet
```

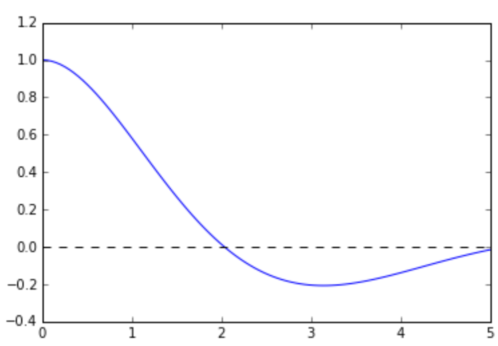
In [27]:

```
K = [2.6260,-0.6250]
```

$$\dot{x} = Ax + Bu = Ax - BKx = (A - BK)x$$

In [28]:

```
x0 = np.array([[1],[1]])
Gcl = ss(A-B*K,B,C,0)

t = np.linspace(0,5,100)
u = np.zeros(t.shape)

[y,tout,x] = lsim(Gcl,u,t,x0)

plt.plot(tout,y)
plt.xlabel('t')
plt.plot(tout,np.zeros(tout.shape),'k--')

plt.show()
```



4.2. Controllability

- When can we place the eigenvalues using state feedback?
- When is B matrix (the actuator configuration) rich enough so that we can make the system do whatever we want it to do?
- The answer revolves around the concept of `controllability`

Given a discrete-time system

$$x_{k+1} = Ax_k + Bu_k$$

We would like to drive this system in n steps to a particular target state x^*

$$\begin{aligned} x_1 &= Ax_0 + Bu_0 = Bu_0 \\ x_2 &= Ax_1 + Bu_1 = ABu_0 + Bu_1 \\ x_3 &= Ax_2 + Bu_2 = A^2Bu_0 + ABu_1 + Bu_2 \\ &\vdots \\ x_n &= A^{n-1}Bu_0 + \cdots + Bu_{n-1} \end{aligned}$$

We want to solve

$$x^* = \begin{bmatrix} B & AB & \cdots & A^{n-1}B \end{bmatrix} \begin{bmatrix} u_{n-1} \\ \vdots \\ u_1 \\ u_0 \end{bmatrix}$$

This is possible for any target state if and only if

$$\text{rank}\left(\begin{bmatrix} B & AB & \cdots & A^{n-1}B \end{bmatrix}\right) = n$$

Example 1

$$\dot{x} = \begin{bmatrix} 2 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} u$$

In [29]:

```
A = np.array([[2,0],[1,1]])
B = np.array([[1],[1]])

G = ctrb(A,B)
print(G)
```

```
[[1 2]
 [1 2]]
```

In [30]:

```
print(np.linalg.matrix_rank(G))
```

```
1
```

Example 2

$$\dot{x} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u$$

In [31]:

```
A = np.array([[0,1],[0,0]])
B = np.array([[0],[1]])

G = ctrb(A,B)
print(G)
```

```
[[0 1]
 [1 0]]
```

In [32]:

```
print(np.linalg.matrix_rank(G))
```

```
2
```

In [33]:

```
%%javascript
$.getScript('https://kmahelona.github.io/ipython_notebook_goodies/ipython_notebook_toc.js')
```