Mathematics for Robotics and Control SS2016

Assignment 7: State Space Representations

Modules:

- control
- numpy

Functions:

control:

- feedback, parallel, series, tf

State Space Representations

A state-space model (https://en.wikipedia.org/wiki/State-space_representation) is a structured form or representation of a set of differential equations. Statespace models are very useful in Control theory and design.

Questions :

1. What are state space models ? Write the standard form for state space equation ? Explain each terms and its semantics?
2. Why should we use them ?
3. What are the basic properties of State space models and how do we analyse them ?
4. Converting ODE to state space models (http://www.sharetechnote.com/html/DE_StateSpaceModel.html) Read the following link and write the steps involved in converting ODE to state space models
5. How many state space variables are required to describe a system fully ? (Difficult!)

Answers :

1*.It represents mathematical model of a physical system as a set of input, output and state variables related by first-order differential equations.

Standard form of state space equation :

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

Matrix A:

Matrix A is the system matrix, and relates how the current state affects the state change x' . If the state change is not dependent on the current state, A will be the zero matrix.

Matrix B:

Matrix B is the control matrix, and determines how the system input affects the state change. If the state change is not dependent on the system input, then B will be the zero matrix.

Matrix C:

Matrix C is the output matrix, and determines the relationship between the system state and the system output.

Matrix D:

Matrix D is the feed-forward matrix, and allows for the system input to affect the system output directly. A basic feedback system like those we have previously considered do not have a feed-forward element, and therefore for most of the systems we have already considered, the D matrix is the zero matrix.

2*. State space models are useful because it helps to represent more complex systems having single or multiple inputs, just with single first order matrix differential equation. Furthermore, transfer functions and response of complex systems can easily be computed with the help of state space models.

3*.The basic properties of state space models are stability, observability and controllability.

Controllability : It refers to the ability of a controller to arbitrarily alter the functionality of the system plant. A system with internal state vector x is called controllable if and only if the system states can be changed by changing the system input.

Observability : It describes whether the internal state variables of the system can be externally measured. A system with an initial state, $x(t_0)$ is observable if and only if the value of the initial state can be determined from the system output y(t) that has been observed through the time interval $t_0 < t < t_f$ $t_0 < t < t_f$. If the initial state cannot be so determined, the system is unobservable.

Stability :

The stability of a time-invariant state-space model can be determined by looking at the system's transfer function in factored form. It will then look something like this:

$$G(s) = k\frac{(s - z_1)(s - z_2)(s - z_3)}{(s - p_1)(s - p_2)(s - p_3)(s - p_4)}$$

Where roots of denominator are poles and roots of numerator are zeros. Therefore, poles can can be used to analyze whether the system is asymptotically stable or marginally stable. An alternative approach to determining stability, which does not involve calculating eigenvalues, is to analyze the system's Lyapunov stability.

4*. Let's say that we have a general 3rd order differential equation in terms of input u(t) and output y(t):

$$\frac{d^3y(t)}{dt^3} + a_2\frac{d^2y(t)}{dt^2} + a_1\frac{dy(t)}{dt} + a_0y(t) = u(t)$$

We can create the state variable vector x in the following manner:

$$x_1 = y(t)$$
$$x_2 = \frac{dy(t)}{dt}$$

$$x_3 = \frac{d^2y(t)}{dt^2}$$

Which now leaves us with the following 3 first-order equations:

$$x_1' = x_2$$
$$x_2' = x_3$$
$$x_3' = \frac{d^3y(t)}{dt^3}$$

Now, we can define the state vector x in terms of the individual x components, and we can create the future state vector as well:

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, x' = \begin{bmatrix} x_1' \\ x_2' \\ x_3' \end{bmatrix}$$

And with that, we can assemble the state-space equations for the system:

$$x' = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -a_0 & -a_1 & -a_2 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u(t)$$

$$y(t) = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} x(t)$$

5*. The minimum number of state variables required to represent a given system,n, is usually equal to the order of the system's defining differential equation. If the system is represented in transfer function form, the minimum number of state variables is equal to the order of the transfer function's denominator after it has been reduced to a proper fraction. In case of RLC circuit, number of state variables can be decided according to energy storing elements in the circuit. Therefore, number of state variables depends upon the differential equation defining the system or transfer function of the system.

References :

https://en.wikipedia.org/wiki/State-space_representation (https://en.wikipedia.org/wiki/State-space_representation)

https://en.wikibooks.org/wiki/Control_Systems/State-Space_Equations (https://en.wikibooks.org/wiki/Control_Systems/State-Space_Equations)

http://lpsa.swarthmore.edu/Representations/SysRepSS.html (http://lpsa.swarthmore.edu/Representations/SysRepSS.html)

Exercise 1.1

Given the following system :

$$\dot{x_1} = x_2$$
$$\dot{x_2} = -2x_1 - x_2 + 5u$$
$$y = x_2$$

Set the system on the following state space form :

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

write the A,B, C and D matrix

In [1]:

```
import numpy as np
import control
A = np.array([[0, 1],[-2, -1]])
B = np.array([[0],[5]])
C = np.array([0, 1])
D = np.array([0])
state_space_model =control.ss(A,B,C,D)
state_space_model
#print A.shape, B.shape, C.shape
```

```
/home/ramesh/anaconda2/lib/python2.7/site-packages/matplotlib/font_man
ager.py:273: UserWarning: Matplotlib is building the font cache using
 fc-list. This may take a moment.
  warnings.warn('Matplotlib is building the font cache using fc-list.
 This may take a moment.')
```

Out[1]:

```
A = [[ 0  1]
 [-2 -1]]

B = [[0]
 [5]]

C = [[0 1]]

D = [[0]]
```

Exercise 1.2

Given the following system :

$$\dot{x_1} = x_1 + u$$
$$\dot{x_2} = x_2$$
$$\dot{x_3} = x_3$$
$$y_1 = x_3$$
$$y_2 = x_1 + 2x_2 + 2x_2$$

Set the system on the following state space form :

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

write the A,B, C and D matrix

In [2]:

```
import numpy as np
import control
A = np.array([[1, 0, 0],[0, 1, 0],[0,0,1]])
B = np.array([[1],[0],[0]])
C = np.array([[0,0,1],[1,4,0]])
D = np.array([[0],[0]])
state_space_model =control.ss(A,B,C,D)
state_space_model
```

Out[2]:

```
A = [[1 0 0]
 [0 1 0]
 [0 0 1]]

B = [[1]
 [0]
 [0]]

C = [[0 0 1]
 [1 4 0]]

D = [[0]
 [0]]
```

Exercise 1.3

Given the following system :

$$\dot{x_1} = x_2$$
$$\dot{x_2} = -x_1 - 3x_2 + 2u_1 + 4u_2$$
$$y = x_1 + u_2$$

Set the system on the following state space form :

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

write the A,B, C and D matrix

In [3]:

```
import numpy as np
import control
A = np.array([[0, 1],[-1, -3]])
B = np.array([[0,0],[2,4]])

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

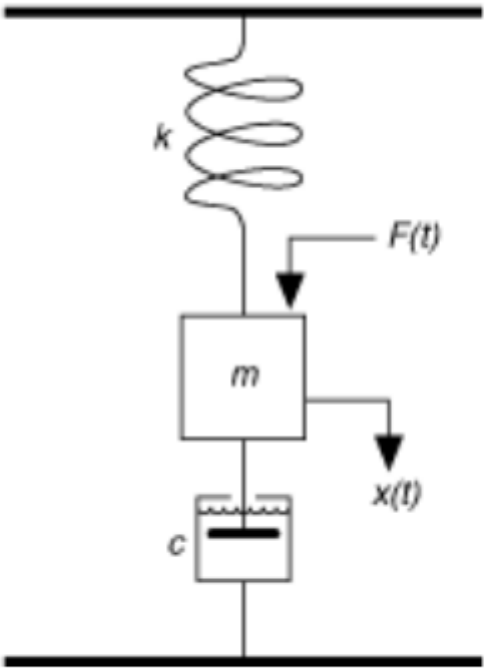
Out[3]:

```
A = [[ 0  1]
 [-1 -3]]

B = [[0 0]
 [2 4]]

C = [[1 0]]

D = [[0 1]]
```

Exercise 2.1

Mass Spring Damper system

Given a mass spring damper system :



Using Netwons Second Law :

$$\sum F = ma$$

The model of the system can be described as :

$$\ddot{x} = \frac{1}{m}(F - c\dot{x} - kx)$$

where : x - position , $\dot{x}$ - speed/velocity , $\ddot{x} - acceleration$ c - damping constant, m - mass, k - spring constant, F - force

Set the system on the following state space form :

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

write the A,B, C and D matrix Assume the control signal u is equal to the force F and that we only measure position

In [32]:

```
import control
import numpy as np
import sympy as sp
k,m,c = sp.symbols('k,m,c')
A = np.array([[0,1],[-k/m, -c/m]])
B = np.array([[0],[1/m]])
# if output is x
C = np.array([[1, 0]])
print C.shape
D = np.array([0])
state_space_equation = control.ss(A,B,C,D)
state_space_equation
```

```
(1, 2)
```

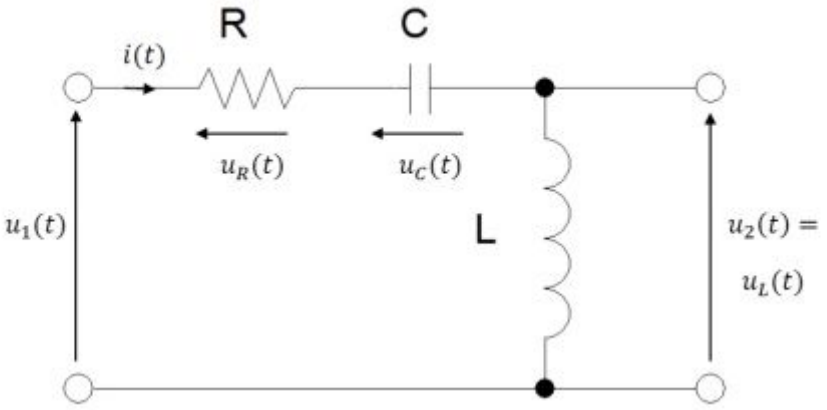
Out[32]:

```
A = [[0 1]
 [-k/m -c/m]]

B = [[0]
 [1/m]]

C = [[1 0]]

D = [[0]]
```

Exercise 2.2

RLC Circuit

Find the state space equation for the following RLC circuit



Applying KVL, we get :

$$u_1(t) = Ri(t) + u_c + L\frac{di}{dt} - - - - - (1)$$
$$u_L(t) = L\frac{di}{dt} - - - - (2)$$

State variables can be considered as :

$$x_1 = u_c$$
$$x_2 = i_L$$

Since current in series circuit same, $i_L = i(t)$ Take derivative of state variables, we have

$$\dot{x_1} = \dot{u_c} = \frac{1}{C}.x_2$$
$$\dot{x_2} = \frac{di}{dt}$$

substitue values of $\dot{x_1}$ and $\dot{x_2}$ in equation 1 and 2, and solve for $\dot{x_2}$, we have :

$$\dot{x_2} = \frac{u_1(t)}{L} - \frac{R}{L}x_2 - \frac{x_1}{L}$$

$$y = L\dot{x_2}$$

substitue value of $\dot{x_2}$ we have

$$y = -Rx_2 - x_1 + u_1(t)$$

In [4]:

```
import numpy as np
import control
L,C,R =  sp.symbols('L,C,R')
A = np.array([[0,1/C],[-1/L, -R/L]])
B = np.array([[0],[1/L]])
C = np.array([[-1, -R]])
D = np.array([1])
state_space_equation = control.ss(A,B,C,D)
state_space_equation
```

Out[4]:

```
A = [[0 1/C]
 [-1/L -R/L]]

B = [[0]
 [1/L]]

C = [[-1 -R]]

D = [[1]]
```

In [8]:

```
import IPython.core.display
import sys
if not "win" in sys.platform and not "linux" in sys.platform:
    %pylab
else:
    %pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

Exercise 3

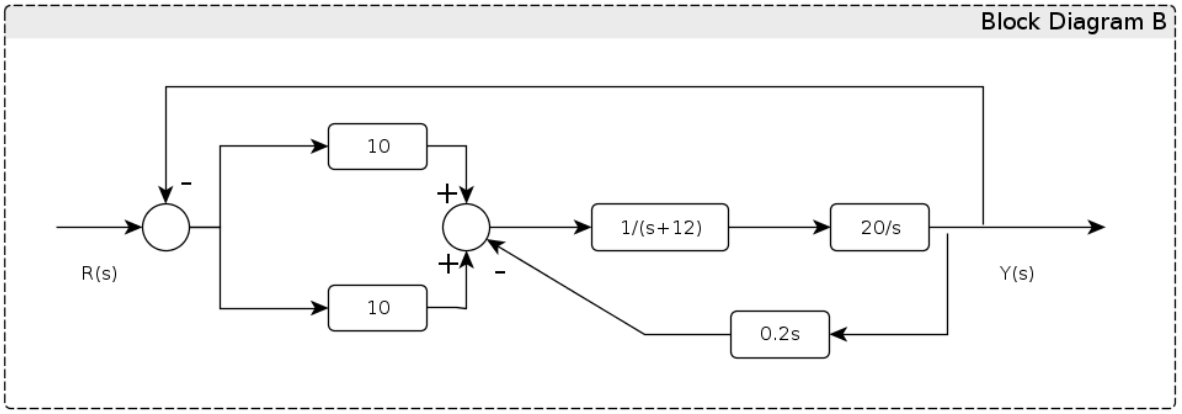
Go to http://python-control.readthedocs.io/en/latest/intro.html (http://python-control.readthedocs.io/en/latest/intro.html)and install the **python-control** library. Use Anaconda's **pip** to install the library into your local Anaconda environment. Familiarize yourself with pip here (http://www.pip-installer.org/en/latest/). Please note that a version of pip is *already installed* in your Anaconda environment.

Familiarize yourself with the control packages. Use control.parallel, control.series, control.feedback and control.tf to model the system depicted below. Obtain a SINGLE transfer function that is equivalent to the complete system, i.e. REDUCE the block diagram to a single transfer function.

In [35]:

```
IPython.core.display.Image(r"images/bdiag0002.png", embed=True)
```

Out[35]:



In [50]:

```
import control
#s = sp.symbols('s')
#since 10 and 10 are in parallel
sys1 = control.parallel(10,10)
#print sys1
#Compute transfer function.
num1  = [1]
den1  = [1, 12]
sys2 = control.tf(num1,den1)
#compute  transfer function
num2 = [20]
den2 = [1, 0]
sys3 = control.tf(num2,den2)
#since sys1 and sy2 are in series .
sys_series = control.series(sys2,sys3)
#compute transfer function of 0.2s
num3 = [0.2, 0]
den3 = [1]
sys4 = control.tf(num3,den3)
#Now sys_series and sys3 are in feedback .
feedback_section = control.feedback(sys_series,sys4)
#print feedback_section
#now sys1 and feedback_section are in series.
system = control.series(sys1,feedback_section)

#print system
#now solve final feedback.
#first compute transfer function of feedback section
num4 = [1]
den4 = [1]
sys5 = control.tf(num4,den4)
#Since system and sys5 are in feedback.
final_feedback = control.feedback(system,sys5)
print final_feedback
```

```
       400
----------------
s^2 + 16 s + 400
```

Exercise 4

From previous exercise determine block diagram and the transfer function for the **Mass Spring Damper System** and the **RLC** circuit

In [31]:

```
#Mass Spring Damper System

import control
import numpy as np
import sympy as sp

sp.init_printing()
k,m,c,s = sp.symbols('k,m,c,s')
#to convert state space into transfer function we have,
# T(s) = C([sI-A]^-1)B+D
A = sp.Matrix([[0,1],[-k/m, -c/m]])
B = sp.Matrix([[0],[1/m]])
C = sp.Matrix([[1,0]])
D = sp.Matrix([0])

identity_matrix = sp.Matrix([[1,0],[0,1]])

Identity_s = s*identity_matrix

subtract_SI_A = Identity_s-A
inverse = subtract_SI_A.inv()
product_c_with_inverse = C*inverse
transfer_function = product_c_with_inverse*B + D
transfer_function
```

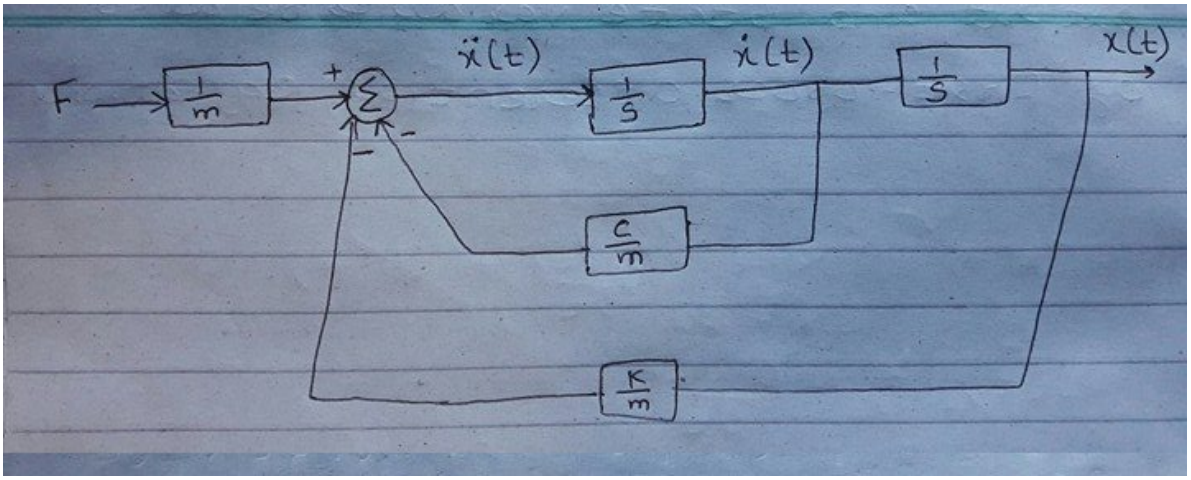
Out[31]:

$$\left[\frac{1}{ms\left(\frac{c}{m}+\frac{k}{ms}+s\right)} \right]$$

In [10]:

```
IPython.core.display.Image(r"images/dampersystem.jpg", embed=True)
```

Out[10]:



In [6]:

```
import numpy as np
import control
sp.init_printing()
import sympy as sp
L,C,R,c,s =  sp.symbols('L,C,R,c,s')


A = sp.Matrix([[0,1/c],[-1/L, -R/L]])

B = sp.Matrix([[0],[1/L]])

C = sp.Matrix([[-1, -R]])
D = sp.Matrix([1])

I = sp.Matrix([[1,0],[0,1]])
SI = s*I
subtract_SI_A = SI-A
inverse = subtract_SI_A.inv()
product_c_with_inverse = C*inverse
transfer_function = sp.simplify(product_c_with_inverse*B + D)
transfer_function
```

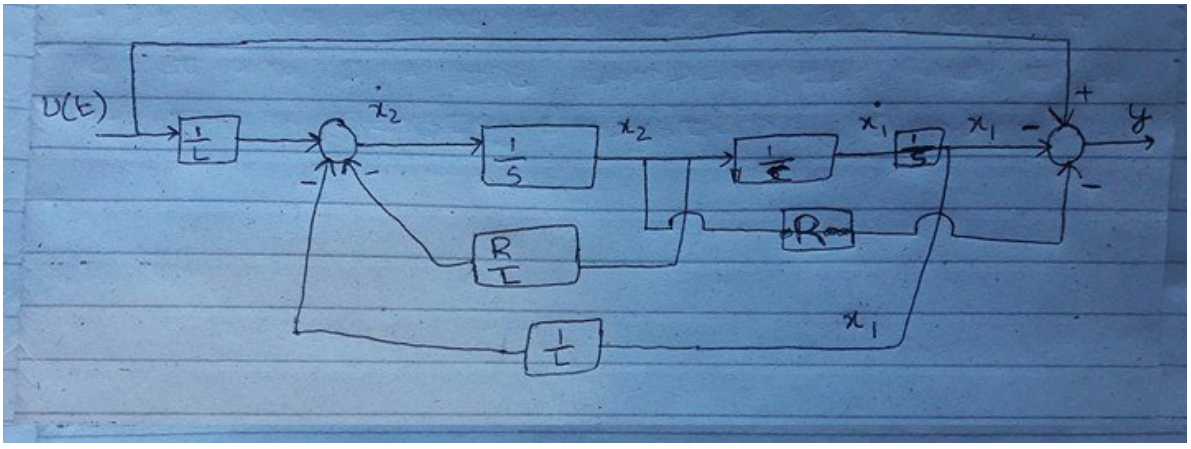
Out[6]:

$$\left[\begin{matrix}\frac{Lcs^{2}}{Lcs^{2}+Rcs+1}\end{matrix}\right]$$

In [12]:

```
IPython.core.display.Image(r"images/RLC.jpg", embed=True)
```

Out[12]:



Exercise 5

Differential drive robot

Find the state space model of a differntial drive robot where v_r and v_l are the velocities of the robot left and right wheel, the location of the robot is given by x, y and θ . Velocities is given by $\dot{x}, \dot{y}$. Angular velocity of the robot is $\dot{\theta}$

The velocities of robot is given by :

$$\dot{x} = \frac{(v_r + v_l)cos\phi}{2}$$
$$\dot{y} = \frac{(v_r + v_l)sin\phi}{2}$$
$$\dot{\phi} = \frac{(v_r - v_l)}{l}$$

where l is distance between two wheels

State variable equation can be written as :

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ 0 & 0 \\ \frac{1}{L} & -\frac{1}{L} \end{bmatrix} \begin{bmatrix} v_r \\ v_l \end{bmatrix}$$

where Rotation matrix is : $\begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$

So, B = $\begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ 0 & 0 \\ \frac{1}{L} & -\frac{1}{L} \end{bmatrix}$

A = $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

Since output are x, y and θ therefore, output can be written as :

$$\begin{bmatrix} x \\ y \\ \theta \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} v_r \\ v_l \end{bmatrix}$$

where C = $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

and D = $\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$

In [2]:

```
import sympy as sp
sp.init_printing()
theta, L = sp.symbols('theta,L')
A = sp.Matrix([[0,0,0],[0,0,0],
               [0,0,0]])
B = sp.Matrix([[1/2, 1/2],[0,0],[1/L,-1/L]])
Rotation_matrix = sp.Matrix([[sp.cos(theta), -sp.sin(theta),0],
                             [sp.sin(theta),sp.cos(theta),0],
                             [0,0,1]])
C = sp.Matrix([[1,0,0],[0,1,0],[0,0,1]])
D = sp.Matrix([[0,0],[0,0],[0,0]])
B = Rotation_matrix * B
B
```

Out[2]:

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \frac{1}{L} & -\frac{1}{L} \end{bmatrix}$$

In [ ]:

In [ ]: