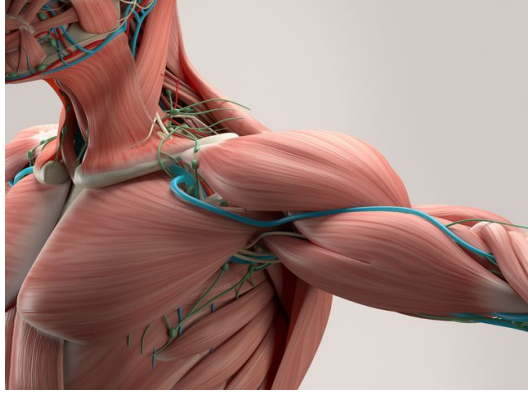


Day 8:
Jacobian

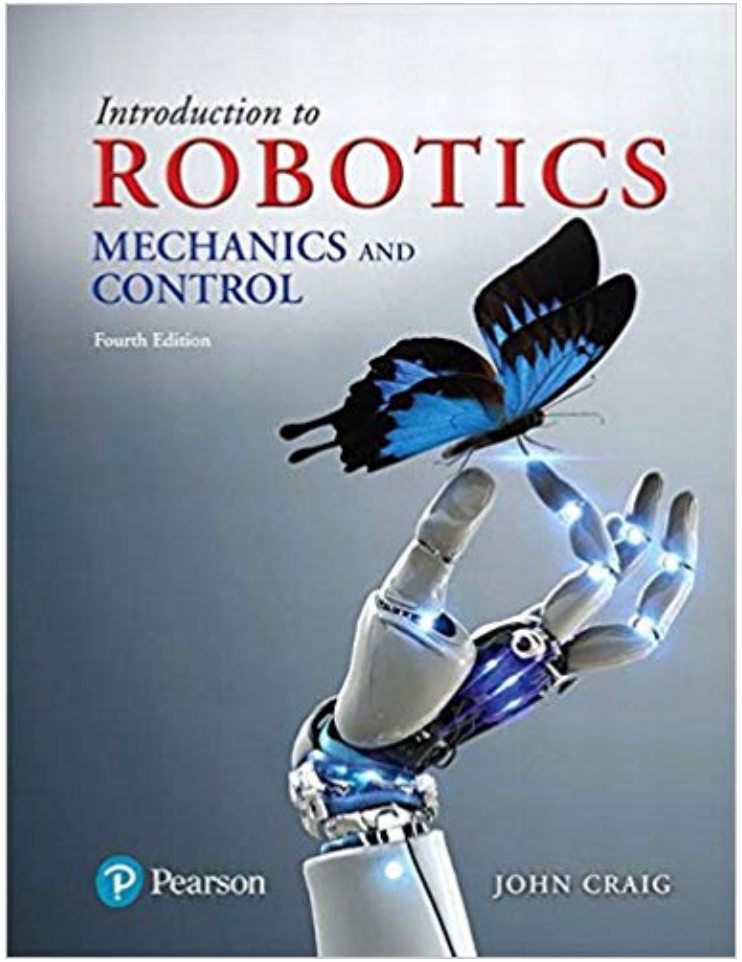
2018. 1. 17. (Thu.)

Dai Owaki, Ph. D.,

Assist. Prof. Neuro-robotics Lab. (Hayashibe Lab.),
Dept. Robotics, Graduate School of Engineering,
Tohoku University, Japan

Schedule for Robotics II in *Python*

7. Forward and Inverse Kinematics (1/10 Thu.)
8. Jacobians (1/17 Thu.)
9. Dynamics (1/21 Mon.)
10. Linear Control: Basis (1/28 Mon.)
11. Linear Control: Application (1/31 Thu.)
12. Nonlinear, Force and Impedance Control (2/4 Mon.)



python™

What is Jacobian (Jacobian matrix) ?

- Jacobian (Jacobian) is a multidimensional form of the derivative.
- In Robotics, we generally use Jacobians that related joint velocities to Cartesian velocities of the tip of the arm.

$$y_1 = f_1(x_1, x_2, x_3, x_4, x_5, x_6),$$
$$y_2 = f_2(x_1, x_2, x_3, x_4, x_5, x_6),$$
$$\vdots$$
$$y_6 = f_6(x_1, x_2, x_3, x_4, x_5, x_6).$$

$$\delta y_1 = \frac{\partial f_1}{\partial x_1}\delta x_1 + \frac{\partial f_1}{\partial x_2}\delta x_2 + \cdots + \frac{\partial f_1}{\partial x_6}\delta x_6,$$
$$\delta y_2 = \frac{\partial f_2}{\partial x_1}\delta x_1 + \frac{\partial f_2}{\partial x_2}\delta x_2 + \cdots + \frac{\partial f_2}{\partial x_6}\delta x_6,$$
$$\vdots$$
$$\delta y_6 = \frac{\partial f_6}{\partial x_1}\delta x_1 + \frac{\partial f_6}{\partial x_2}\delta x_2 + \cdots + \frac{\partial f_6}{\partial x_6}\delta x_6,$$

f_1 through f_6 are nonlinear, then the partial derivatives are a function of the x_i, so,

$$\frac{\delta Y}{\Delta t} = J(X)\frac{\delta X}{\Delta t}$$

Y velocity — *X* velocity

$$\dot{Y} = J(X)\dot{X}.$$

Time-varying linear transformations

Vector notion:

$$Y = F(X).$$

$$\delta Y = \frac{\partial F}{\partial X}\delta X.$$

Jacobian=matrix of partial derivatives

In Robotics,

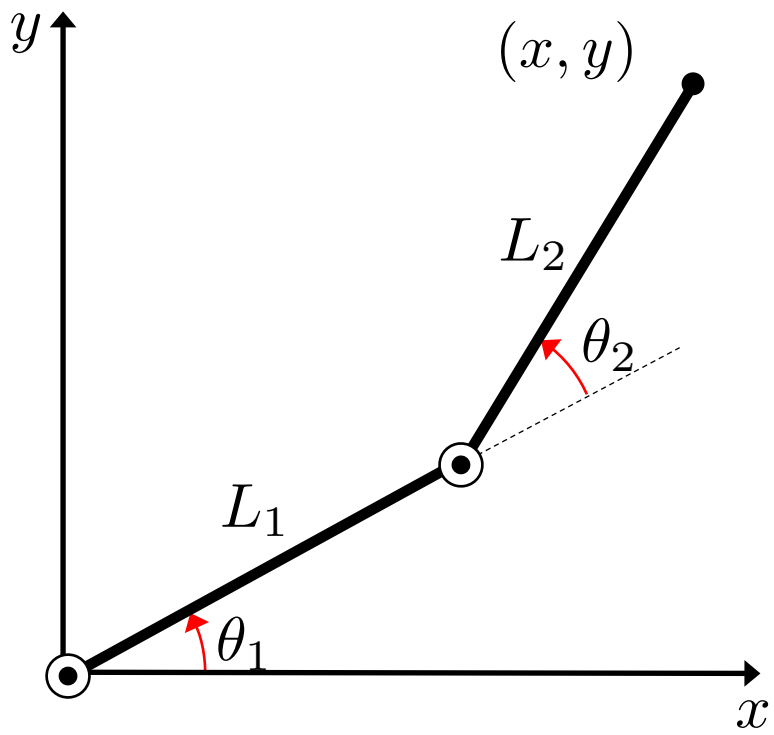
$$\dot{X} = J(\Theta)\dot{\Theta}$$

Cartesian velocity — Joint angular velocity

Jacobian in Robot Arm

Jacobian matrix $J(\Theta)$

$\dot{\theta}_1, \dot{\theta}_2$
Joint angle velocities

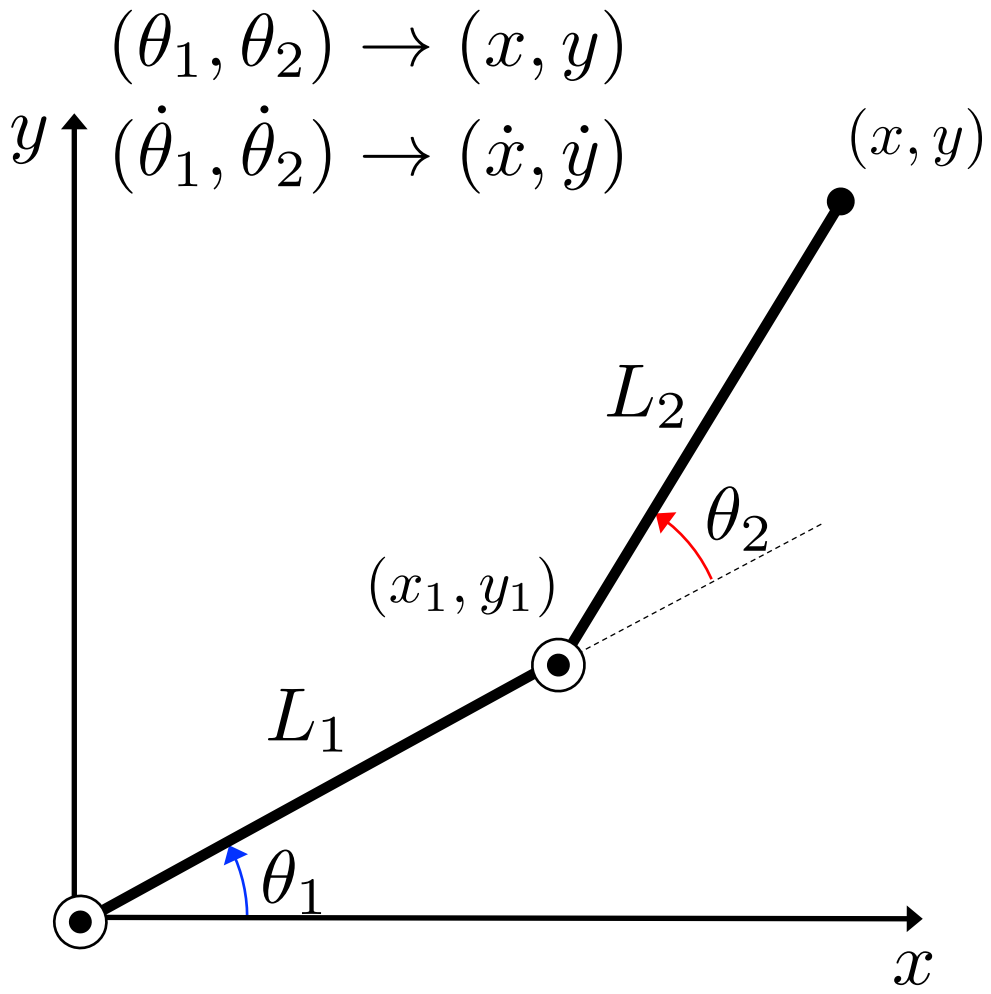


"Non-redundant" manipulator
End effector: 2 degrees of freedom (DoF)
Joint angles: 2 degrees of freedom (DoF)

Velocity of end effector
$\dot{x}, \dot{y}$

Inverse matrix of $J(\Theta)^{-1}$

Jacobian for 2-link 2-joint Manipulator



Non-redundant manipulator

End effector: 2 degrees of freedom (DoF)

Joint angles: 2 degrees of freedom (DoF)

Forward kinematics:

$$\begin{cases} x = L_1 \cos\theta_1 + L_2 \cos(\theta_1 + \theta_2) \\ y = L_1 \sin\theta_1 + L_2 \sin(\theta_1 + \theta_2) \end{cases}$$

$$\begin{cases} x = f_x(\theta_1, \theta_2) \\ y = f_y(\theta_1, \theta_2) \end{cases}$$

$X = (x, y)^T, \Theta = (\theta_1, \theta_2)^T$

$$\dot{X} = J(\Theta)\dot{\Theta}$$

(2*1) (2*2) (2*1)

$$J(\Theta) = \begin{bmatrix} \frac{\partial f_x}{\partial \theta_1} & \frac{\partial f_x}{\partial \theta_2} \\ \frac{\partial f_y}{\partial \theta_1} & \frac{\partial f_y}{\partial \theta_2} \end{bmatrix}$$: Square matrix (m=n)

$$J(\Theta) = \begin{bmatrix} -L_1 \sin\theta_1 - L_2 \sin(\theta_1 + \theta_2) & -L_2 \sin(\theta_1 + \theta_2) \\ L_1 \cos\theta_1 + L_2 \cos(\theta_1 + \theta_2) & L_2 \cos(\theta_1 + \theta_2) \end{bmatrix}$$

www.oscillex.org/robotics2day8

Dai Owaki

home | research | publication | robots | biography | contact

Robotics II Day 8: Jacobian

HOME » Robotics II Day 8: Jacobian

==========

Python script #5

Velocity_Jacobian_2Links.py

Python script #6

Velocity_Jacobian_3Links.py

Python script #7

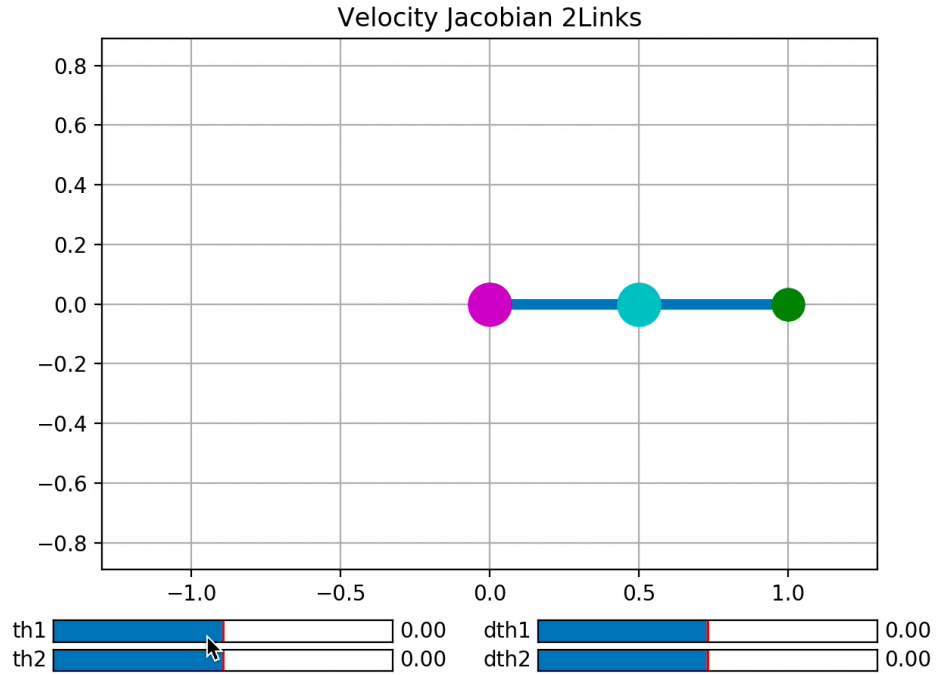
Velocity_Jacobian_3Links_3D.py

Python script #8

Velocity_InverseJacobian1_2Links.py

Python script #9

```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.widgets import Slider
import math

radius = 20

l_1 = 0.5 # length of link 1 [m]
l_2 = 0.5 # length of link 2 [m]

L   = [l_1, l_2] # link parameters
th  = [0.0*math.pi, 0.0*math.pi] # initial angles
dth = np.array([0.0, 0.0]) # initial angular velocity

def Jacobian(th, L):

    L1, L2 = L
    Th1, Th2 = th

    J11 =
    J12 =
    J21 =
    J22 =

    return np.array([[J11, J12],[J21, J22]])

def ForwardKinematics(th, L):

    L1, L2 = L
    Th1, Th2 = th

```



Definition of “Jacobian” function

L1 : link 1 length
L2 : link 2 length
Th1 : link 1 angle 1
Th2 : link 2 angle 2

math.cos() or np.cos()
math.sin() or np.cos()

```
53
54  def update_th1(slider_val):
55      # update the angle of link 1
56      th[0] = slider_val
57
58      # calculation of forward kinematics
59      p = ForwardKinematics(th, L)
60
61      # calculation of Jacobian
62      dx = np.dot(Jacobian(th, L), dth)
63
64      # update the potision of each point
65      graph.set_data(p.T[0], p.T[1]) # *.T means transpose the position vector
66      arrow.set_data([p[2,0],p[2,0]+dx[0]],[p[2,1],p[2,1]+dx[1]])
67      angularvel1.set_data(p[0,0], p[0,1])
68      angularvel2.set_data(p[1,0], p[1,1])
69
70      angularvel1.set_marker('o')
71      angularvel2.set_marker('o')
72      angularvel1.set_markersize(radius*(1.0+dth[0]))
73      angularvel2.set_markersize(radius*(1.0+dth[1]))
74      angularvel1.set_markerfacecolor('m')
75      angularvel2.set_markerfacecolor('c')
76      angularvel1.set_markeredgecolor('m')
77      angularvel2.set_markeredgecolor('c')
78
79
80      # re-draw of the links
81      fig.canvas.draw_idle()
82
```

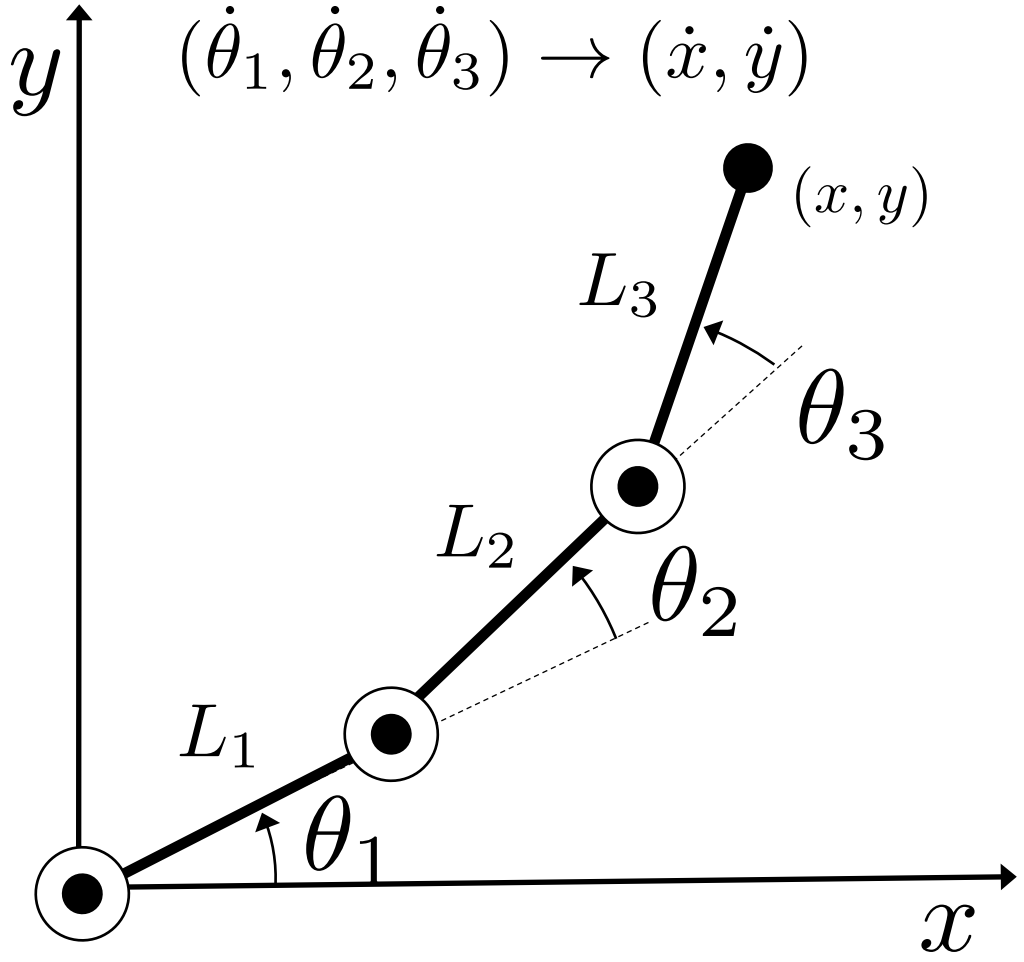
Update function for th[0] (link 1 angle) by using slider1

arrow = velocity of end effector (the direction and length)

angularvel1, angularvel2 = angular velocities of joints (sizes of marker)

Jacobian for 3-link 3-joint Manipulator

$$(\theta_1, \theta_2, \theta_3) \rightarrow (x, y)$$
$$(\dot{\theta}_1, \dot{\theta}_2, \dot{\theta}_3) \rightarrow (\dot{x}, \dot{y})$$



"Redundant" manipulator

End effector: 2 degrees of freedom (DoF)

Joint angles: 3 degrees of freedom (DoF)

Forward kinematics:

$$\begin{cases} x = L_1 \cos\theta_1 + L_2 \cos(\theta_1 + \theta_2) + L_3 \cos(\theta_1 + \theta_2 + \theta_3) \\ y = L_1 \sin\theta_1 + L_2 \sin(\theta_1 + \theta_2) + L_3 \sin(\theta_1 + \theta_2 + \theta_3) \end{cases}$$

$$\begin{cases} x = f_x(\theta_1, \theta_2, \theta_3) \\ y = f_y(\theta_1, \theta_2, \theta_3) \end{cases} \quad X = (x, y)^T, \Theta = (\theta_1, \theta_2, \theta_3)^T$$

$$\dot{X} = J(\Theta)\dot{\Theta}$$

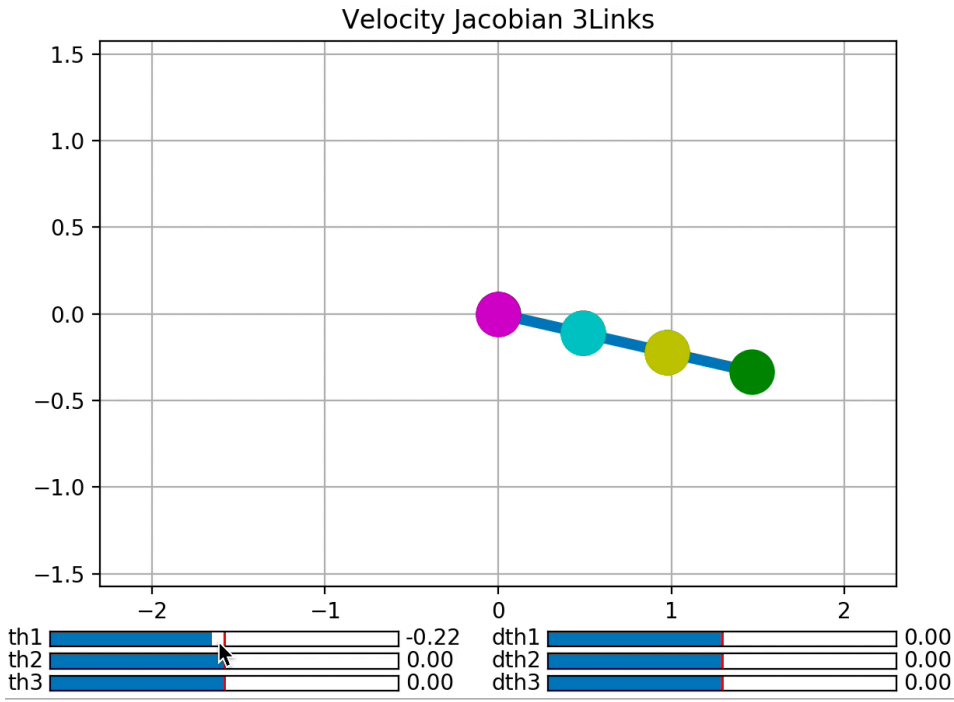
(2*1) (2*3) (3*1)

$$J(\Theta) = \begin{bmatrix} \frac{\partial f_x}{\partial \theta_1} & \frac{\partial f_x}{\partial \theta_2} & \frac{\partial f_x}{\partial \theta_3} \\ \frac{\partial f_y}{\partial \theta_1} & \frac{\partial f_y}{\partial \theta_2} & \frac{\partial f_y}{\partial \theta_3} \end{bmatrix}$$

: Non-square matrix ($m \neq n$)

$$\begin{cases} J_{11} = -L_1 \sin\theta_1 - L_2 \sin(\theta_1 + \theta_2) - L_3 \sin(\theta_1 + \theta_2 + \theta_3) \\ J_{12} = -L_2 \sin(\theta_1 + \theta_2) - L_3 \sin(\theta_1 + \theta_2 + \theta_3) \\ J_{13} = -L_3 \sin(\theta_1 + \theta_2 + \theta_3) \\ J_{21} = L_1 \cos\theta_1 + L_2 \cos(\theta_1 + \theta_2) + L_3 \cos(\theta_1 + \theta_2 + \theta_3) \\ J_{22} = L_2 \cos(\theta_1 + \theta_2) + L_3 \cos(\theta_1 + \theta_2 + \theta_3) \\ J_{23} = L_3 \cos(\theta_1 + \theta_2 + \theta_3) \end{cases}$$

```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.widgets import Slider
import math

radius = 20

l_1 = 0.5 # length of link 1 [m]
l_2 = 0.5 # length of link 2 [m]
l_3 = 0.5 # length of link 3 [m]

L = [l_1, l_2, l_3] # link parameters
th = [0.0*math.pi, 0.0*math.pi, 0.0*math.pi] # initial angles
dth = np.array([0.0, 0.0, 0.0]) # initial angular velocity

def Jacobian(th, L):

    L1, L2, L3 = L
    Th1, Th2, Th3 = th

    J11 =
    J12 =
    J13 =
    J21 =
    J22 =
    J23 =

    return np.array([[J11, J12, J13],[J21, J22, J23]])

def ForwardKinematics(th, L):

```

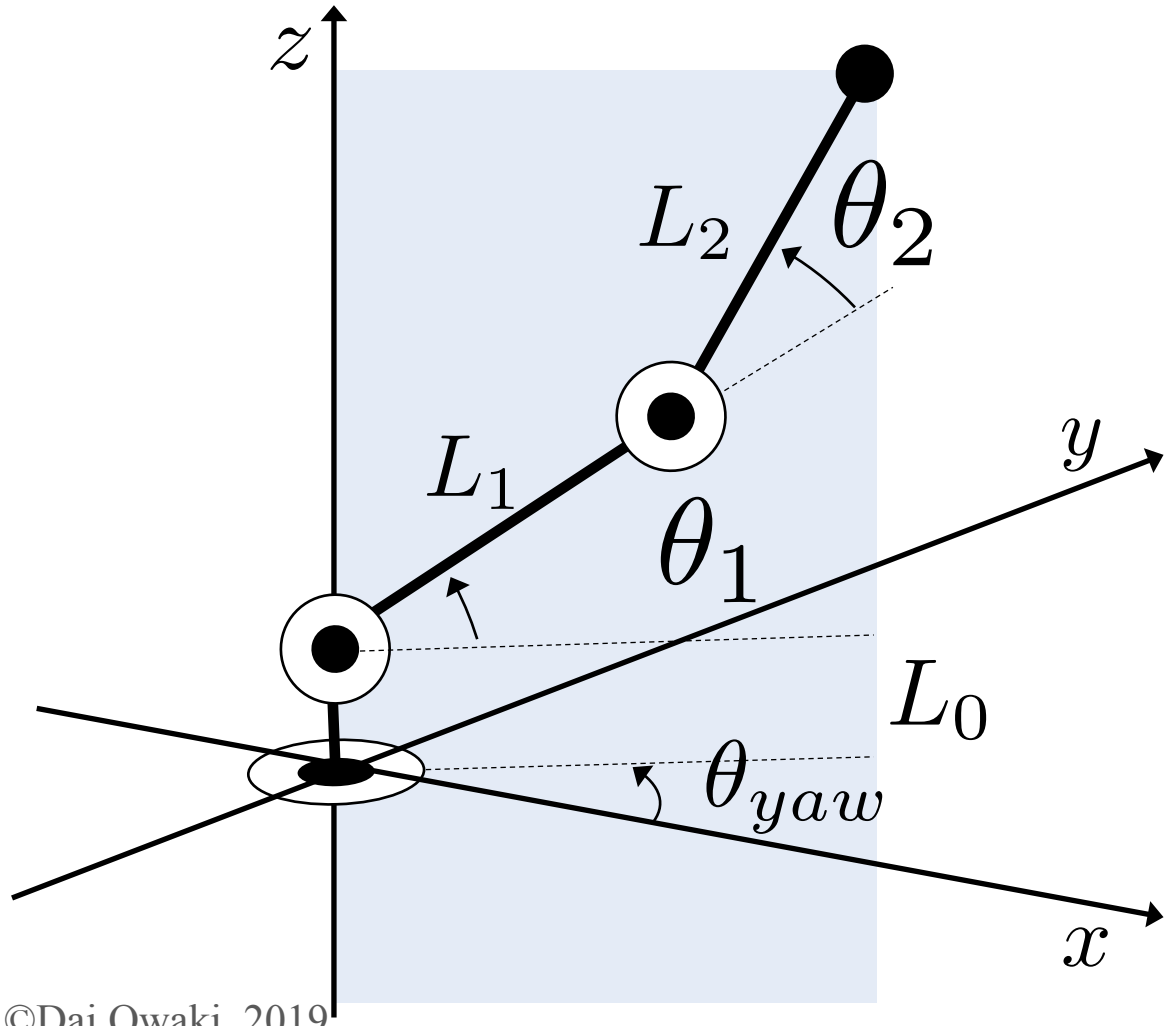


Definition of “Jacobian” function

L1 : link 1 length
L2 : link 2 length
L3 : link 3 length
Th1 : link 1 angle 1
Th2 : link 2 angle 2
Th3 : link 3 angle 3
math.cos() or np.cos()
math.sin() or np.cos()

Exercise 1: 3-link 3-joint Manipulator in 3D

$$(\theta_{yaw}, \theta_1, \theta_2) \rightarrow (x, y, z)$$



Forward kinematics:

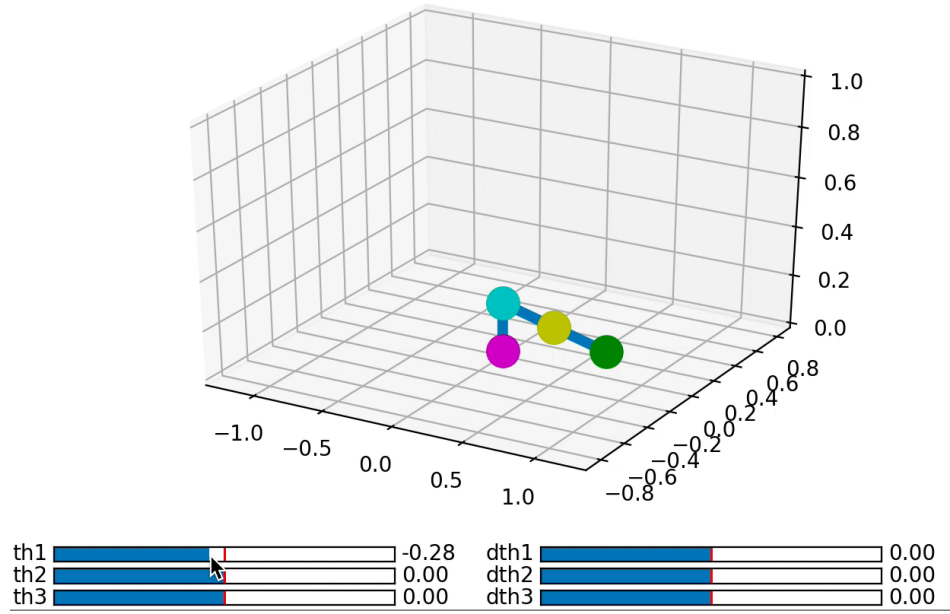
$$\begin{cases} x = \{L_1 \cos\theta_1 + L_2 \cos(\theta_1 + \theta_2)\} \cos\theta_{yaw} \\ y = \{L_1 \cos\theta_1 + L_2 \cos(\theta_1 + \theta_2)\} \sin\theta_{yaw} \\ z = L_1 \sin\theta_1 + L_2 \sin(\theta_1 + \theta_2) + L_0 \end{cases}$$

$$\begin{cases} x = f_x(\theta_{yaw}, \theta_1, \theta_2) \\ y = f_y(\theta_{yaw}, \theta_1, \theta_2) \\ z = f_z(\theta_{yaw}, \theta_1, \theta_2) \end{cases} \quad X = (x, y, z)^T, \Theta = (\theta_{yaw}, \theta_1, \theta_2)^T$$

$$\underset{(3*1)}{\dot{X}} = \underset{(3*3)}{J(\Theta)} \underset{(3*1)}{\dot{\Theta}}$$

This is for Report Topic No.1!!!

```python
# This import registers the 3D projection, but is otherwise unused.
from mpl_toolkits.mplot3d import Axes3D

radius = 15

l_0 = 0.2

l_1 = 0.5 # length of link 1 [m]
l_2 = 0.5 # length of link 2 [m]

L = [l_0, l_1, l_2] # link parameters
th = [0.0*math.pi, 0.0*math.pi, 0.0*math.pi] # initial angles
dth = np.array([0.0, 0.0, 0.0]) # initial angular velocity


def Jacobian(th, L):

    L0, L1, L2 = L
    Th1, Th2, Th3 = th

    J11 =
    J12 =
    J13 =
    J21 =
    J22 =
    J23 =
    J31 =
    J32 =
    J33 =

    return np.array([[J11, J12, J13],[J21, J22, J23],[J31, J32, J33]])

```



Definition of “Jacobian” function

L0 : link 1 length
L1 : link 2 length
L2 : link 3 length
Th1 : angle yaw
Th2 : angle 1
Th3 : angle 2
math.cos() or np.cos()
math.sin() or np.cos()