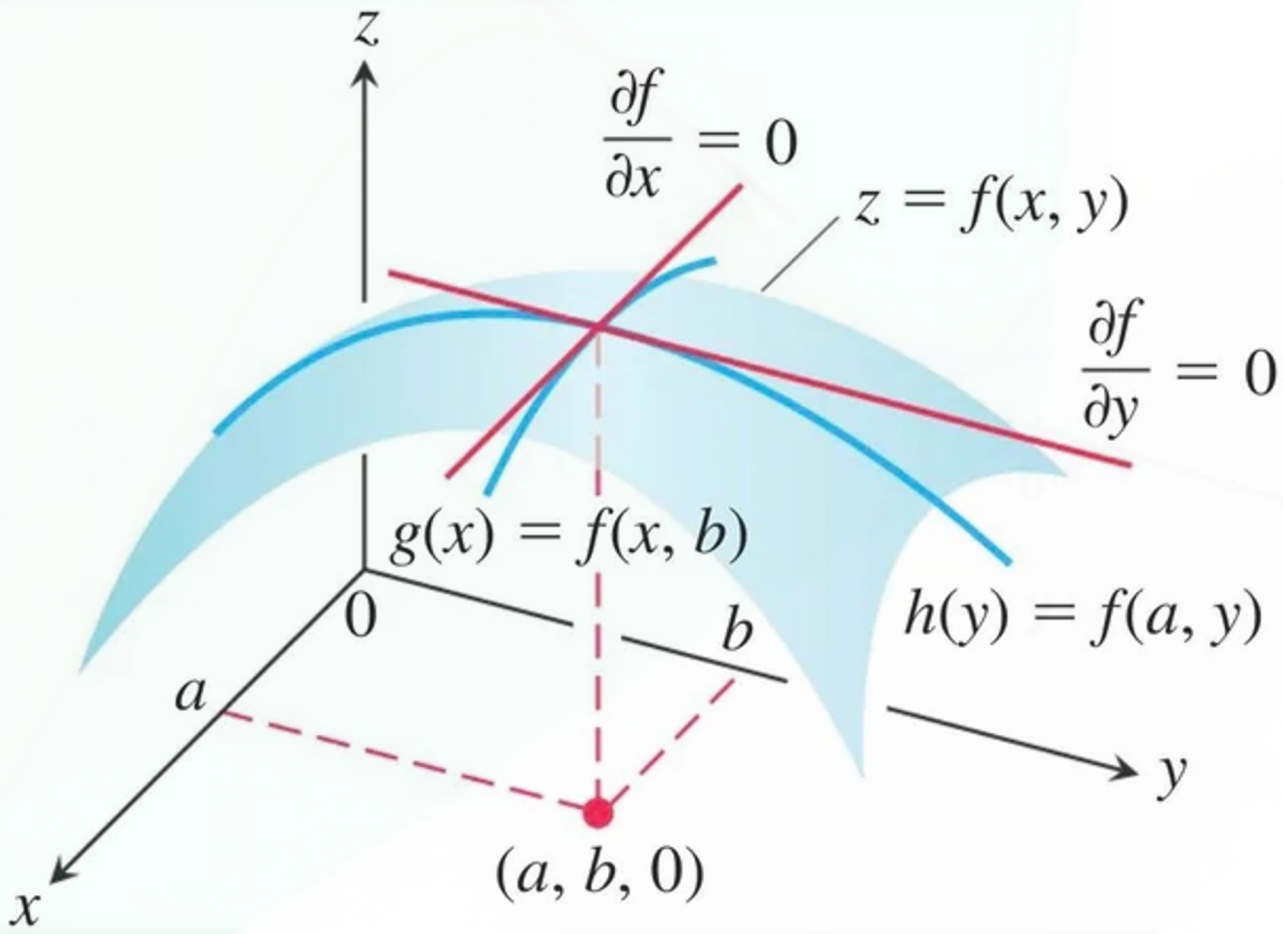
WPI

Lecture 9

Velocity Kinematics (VK),
Jacobian matrix, Kinematics Singularities
and Inverse Velocity Kinematics (IVK)

Alexandros Lioulemes, PhD

Derivatives and Partial Derivatives



https://en.wikipedia.org/wiki/Derivative

Math Reminder: Derivatives

- **Constant Rule:** $f(x) = c$ then $f'(x) = 0$
- **Constant Multiple Rule:** $g(x) = c \cdot f(x)$ then $g'(x) = c \cdot f'(x)$
- **Power Rule:** $f(x) = x^n$ then $f'(x) = nx^{n-1}$
- **Sum and Difference Rule:** $h(x) = f(x) \pm g(x)$ then $h'(x) = f'(x) \pm g'(x)$
- **Product Rule:** $h(x) = f(x)g(x)$ then $h'(x) = f'(x)g(x) + f(x)g'(x)$

https://en.wikipedia.org/wiki/Derivative

Math Reminder: Derivatives

- **Quotient Rule:** $h(x) = \dfrac{f(x)}{g(x)}$ then $h'(x) = \dfrac{f'(x)g(x) - f(x)g'(x)}{g(x)^2}$
- **Chain Rule:** $h(x) = f(g(x))$ then $h'(x) = f'(g(x))g'(x)$
- **Trig Derivatives:**
  - $f(x) = \sin(x)$ then $f'(x) = \cos(x)$
  - $f(x) = \cos(x)$ then $f'(x) = -\sin(x)$

https://en.wikipedia.org/wiki/Derivative

Math Reminder: Partial Derivatives

- For a multivariable function, like $f(x, y) = x^2 y$, computing partial derivatives looks something like this:

$$\frac{\partial f}{\partial x} = \underbrace{\frac{\partial}{\partial x} x^2 y}_{\substack{\text{Treat } y \text{ as constant;} \\ \text{take derivative.}}} = 2xy$$

$$\frac{\partial f}{\partial y} = \underbrace{\frac{\partial}{\partial y} x^2 y}_{\substack{\text{Treat } x \text{ as constant;} \\ \text{take derivative.}}} = x^2 \cdot 1$$

https://en.wikipedia.org/wiki/Partial_derivative

Math Reminder: Partial Derivatives - MATLAB

- For a multivariable function, like $f(x, y) = x^2 y$, computing partial derivatives looks something like this:

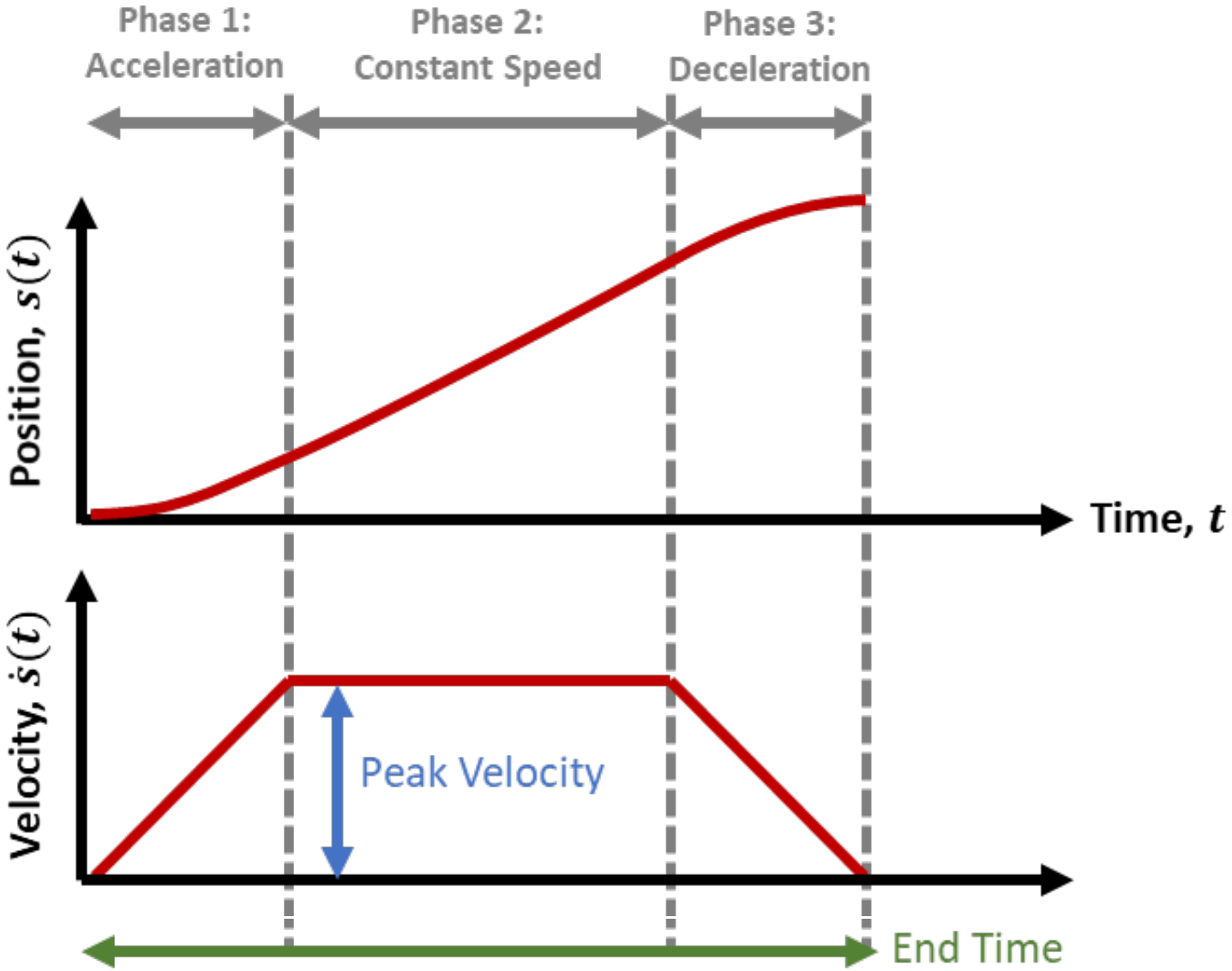
$$\frac{\partial f}{\partial x} = \underbrace{\frac{\partial}{\partial x} x^2 y}_{\text{Treat } y \text{ as constant; take derivative.}} = 2xy$$

$$\frac{\partial f}{\partial y} = \underbrace{\frac{\partial}{\partial y} x^2 y}_{\text{Treat } x \text{ as constant; take derivative.}} = x^2 \cdot 1$$

```
>> syms X Y
>> F=X^2*Y

F =

X^2*Y

>> diff(F,X)

ans =

2*X*Y

>> diff(F,Y)

ans =

X^2
```

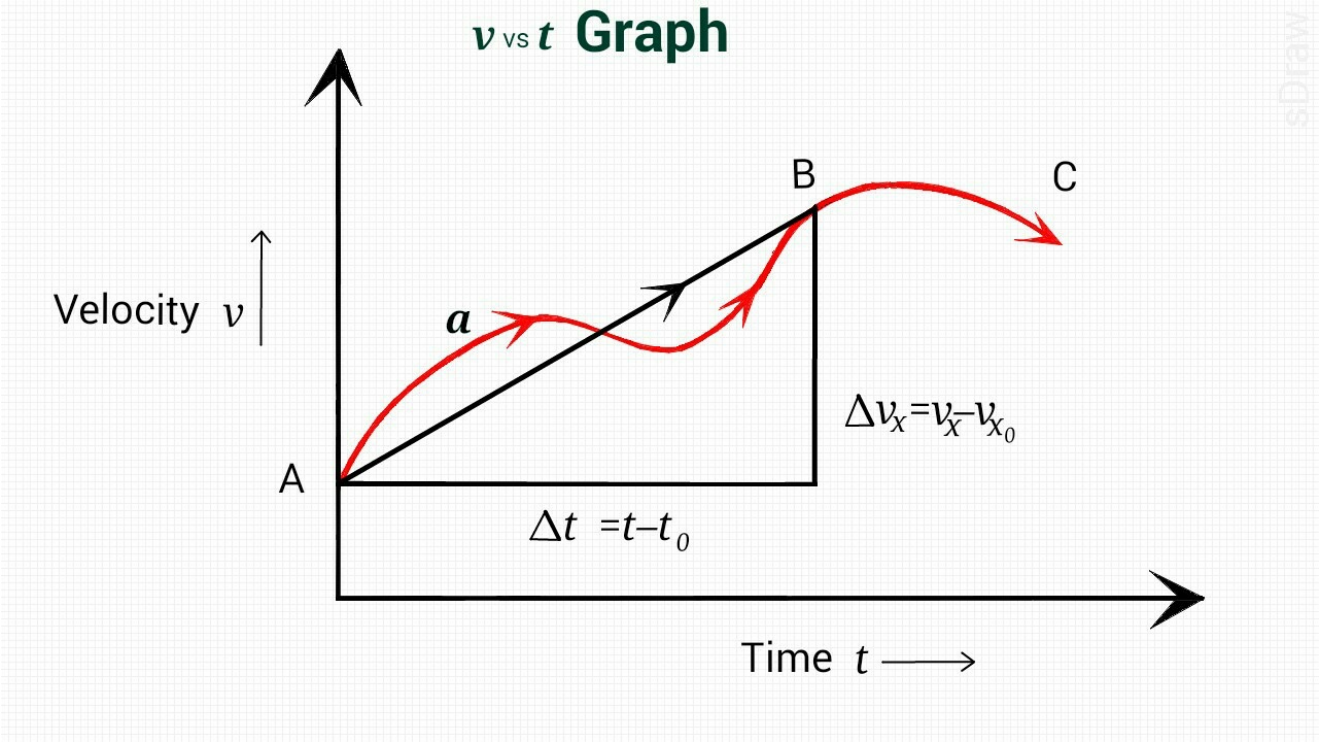
https://en.wikipedia.org/wiki/Partial_derivative

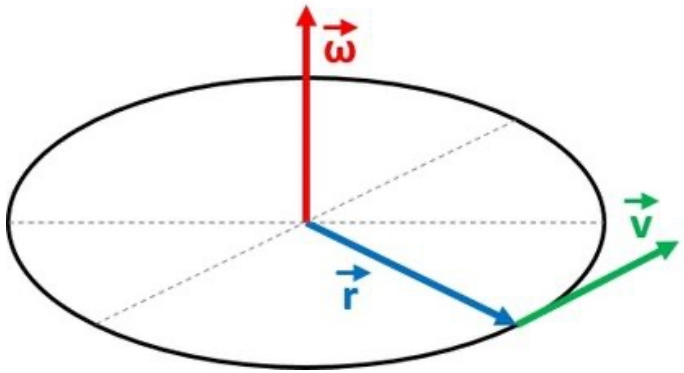
Math Reminder: Position and Velocity



Equations of Motion

- Linear velocity
- Angular velocity
- Rotational velocity
- Simultaneously Linear and Rotational velocity

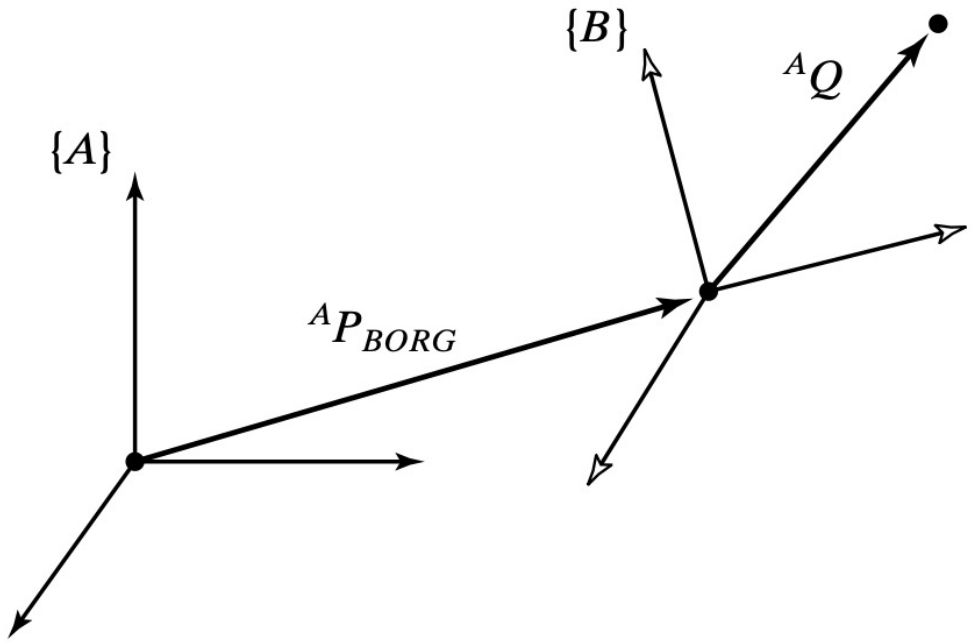




https://en.wikipedia.org/wiki/Equations_of_motion

Linear Velocity

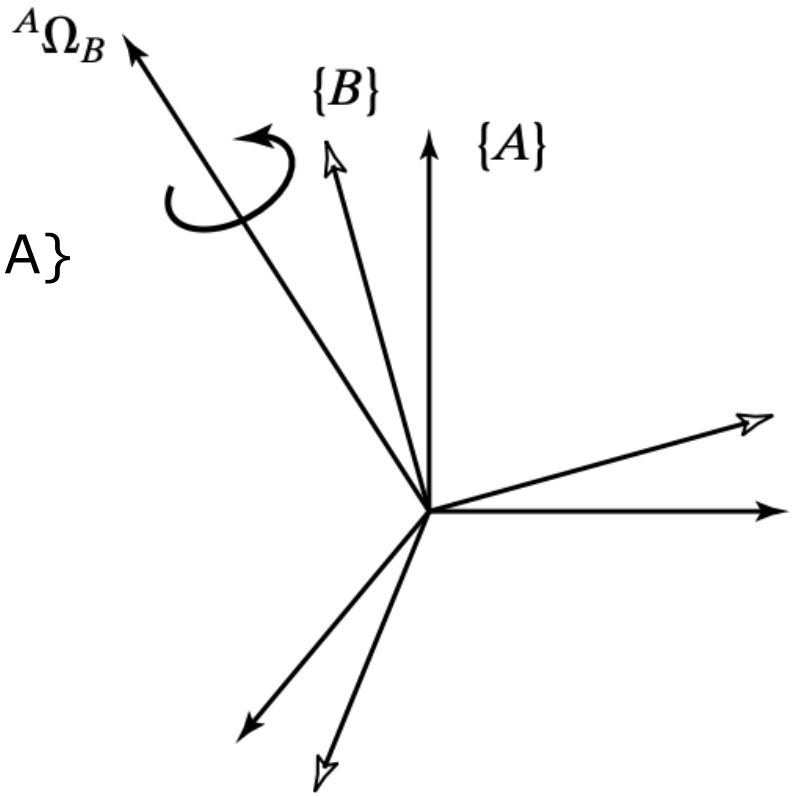
- $${}^{A}V_{Q} = {}^{A}V_{BORG} + {}^{A}_{B}R\,{}^{B}V_{Q}$$



Frame $\{B\}$ is translating with velocity ${}^{A}V_{BORG}$ relative to frame $\{A\}$.

Angular Velocity Vector

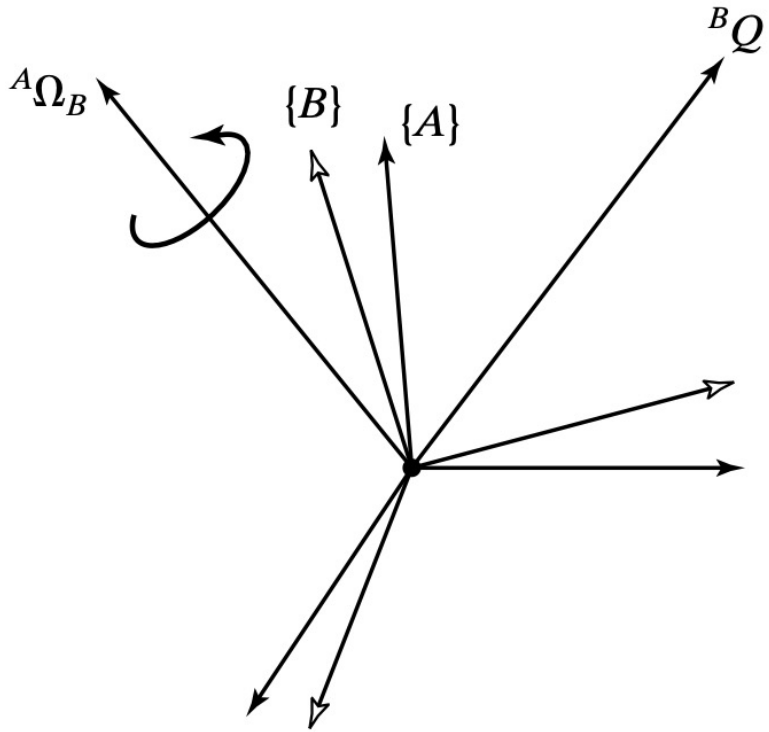
- Ω is the angular velocity which describes an attribute of a body
- ${}^{A}\Omega_{B}$ describes the rotation of a frame {B} relative to {A}



Frame $\{B\}$ is rotating with angular velocity ${}^{A}\Omega_{B}$ relative to frame $\{A\}$.

Rotational Velocity

- $${}^{A}V_{Q} = {}^{A}_{B}R\,{}^{B}V_{Q} + {}^{A}\Omega_{B} \times {}^{A}_{B}R\,{}^{B}Q$$



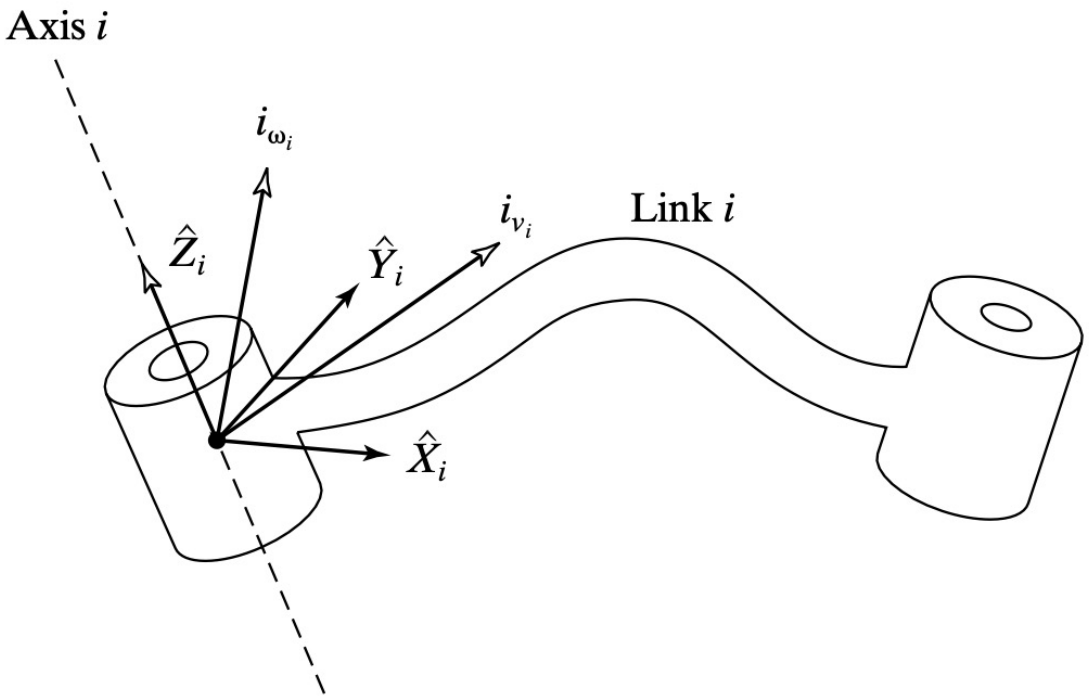
$${}^{B}V_{Q} = 0.$$

Vector ${}^{B}Q$, fixed in frame $\{B\}$, is rotating with respect to frame $\{A\}$ with angular velocity ${}^{A}\Omega_{B}$.

Simultaneously Linear and Rotational Velocity

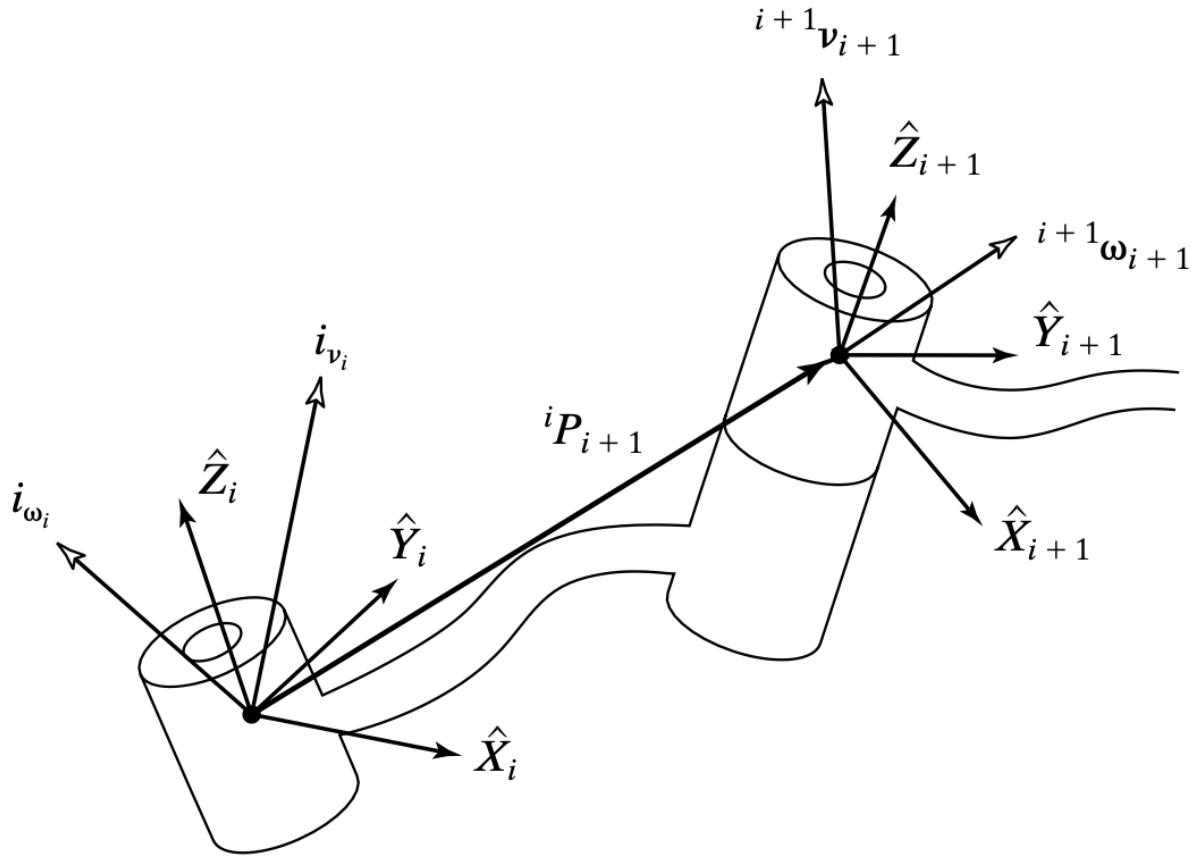
- $^{A}V_{Q} = {}^{A}_{B}R\,{}^{B}V_{Q} + {}^{A}\Omega_{B} \times {}^{A}_{B}R\,{}^{B}Q$
- $^{A}V_{Q} = {}^{A}V_{BORG} + {}^{A}_{B}R\,{}^{B}V_{Q}$
- $^{A}V_{Q} = {}^{A}V_{BORG} + {}^{A}_{B}R\,{}^{B}V_{Q} + {}^{A}\Omega_{B} \times {}^{A}_{B}R\,{}^{B}Q$

Velocities "Propagation" from Link to Link



The velocity of link i is given by vectors v_i and ω_i, which may be written in any frame, even frame $\{i\}$.

Velocities "Propagation" from Link to Link

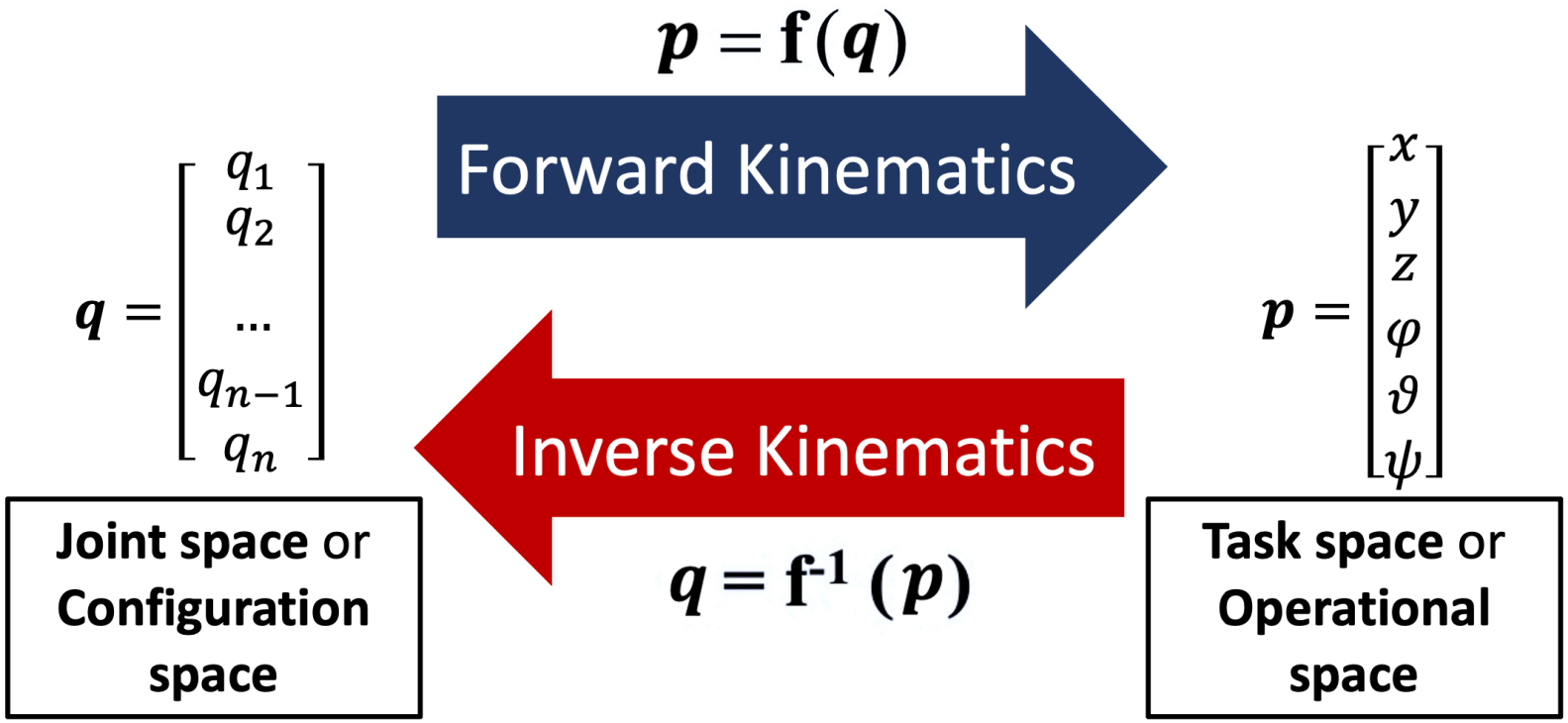


Velocity vectors of neighboring links.

$$^{i}\omega_{i+1} = {}^{i}\omega_{i} + {}^{i}_{i+1}R\,\dot{\theta}_{i+1}\,{}^{i+1}\hat{Z}_{i+1}.$$

Position Kinematics

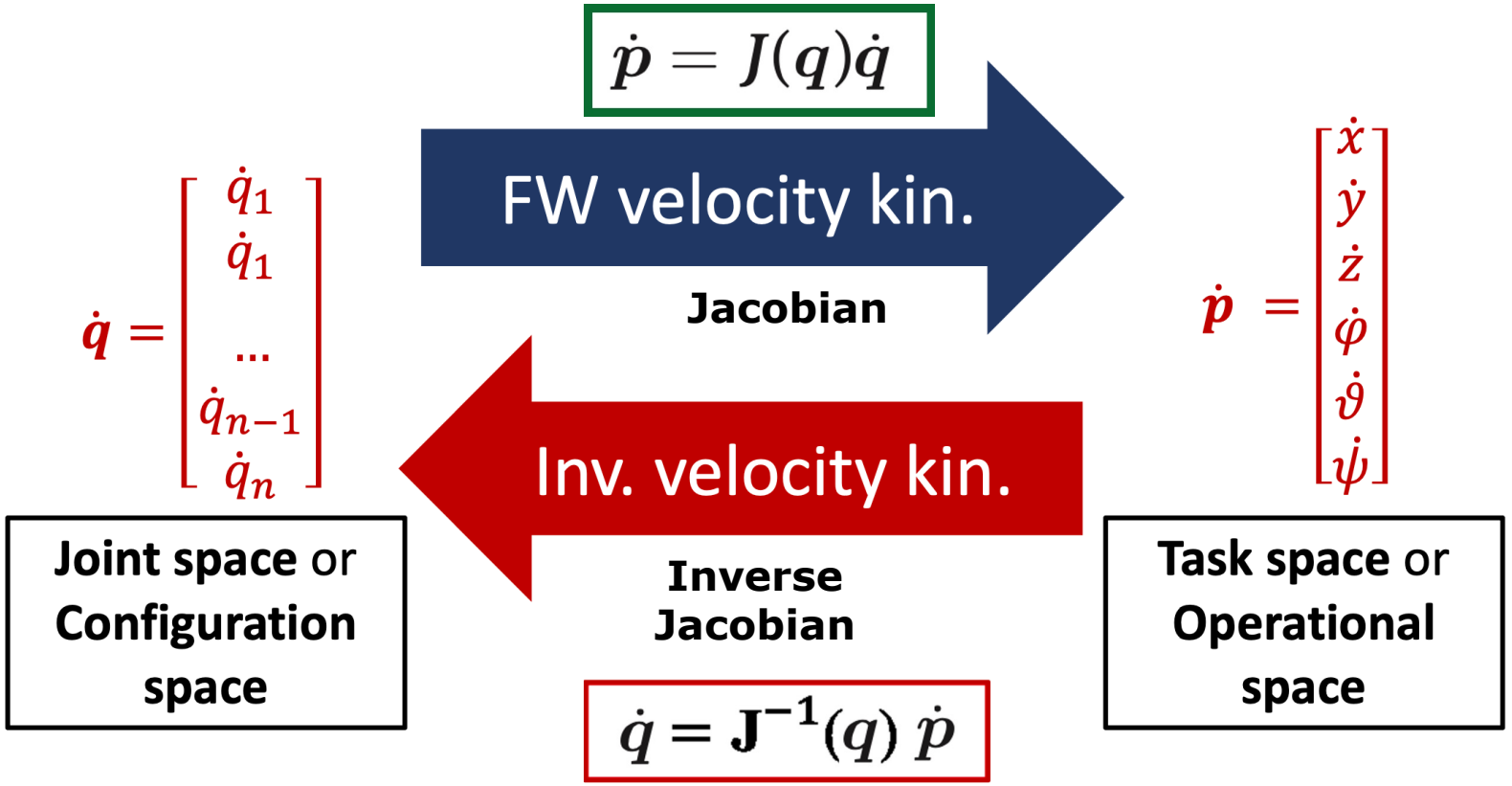
- Control the **pose** of the **end effector**



https://en.wikipedia.org/wiki/Kinematics

Velocity (Differential) Kinematics (VK)

- Control the **velocity** of the **end effector**



https://en.wikipedia.org/wiki/Robot_kinematics#Velocity_kinematics

Carl Jacobi → Jacobian matrix

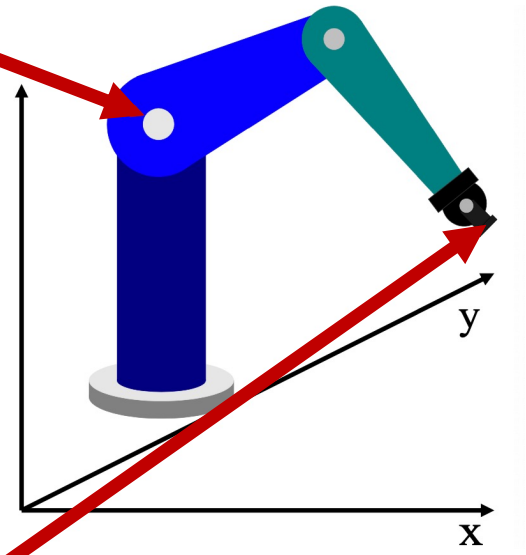
A Jacobian is the matrix equivalent of the derivative – the derivative of a vector-valued function of a vector with respect to a vector. If $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})$ and $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{y} \in \mathbb{R}^m$ then the Jacobian is the $m \times n$ matrix

$$J = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}} = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_n} \end{pmatrix}$$

https://en.wikipedia.org/wiki/Carl_Gustav_Jacob_Jacobi

Jacobian matrix

$$q = \begin{bmatrix} q_1 \\ q_2 \\ \dots \\ q_{n-1} \\ q_n \end{bmatrix}$$

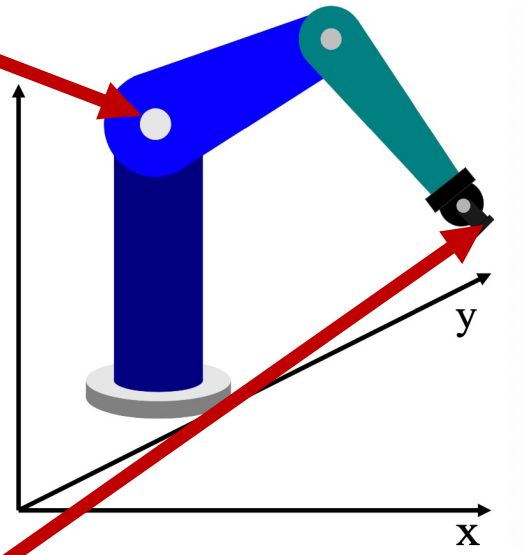


$$p = \begin{bmatrix} x \\ y \\ z \\ \varphi \\ \vartheta \\ \psi \end{bmatrix}$$

https://en.wikipedia.org/wiki/Jacobian_matrix_and_determinant

Jacobian matrix

$$\boldsymbol{q} = \begin{bmatrix} q_1 \\ q_2 \\ \dots \\ q_{n-1} \\ q_n \end{bmatrix}$$



$$\boldsymbol{p} = \begin{bmatrix} x \\ y \\ z \\ \varphi \\ \vartheta \\ \psi \end{bmatrix}$$

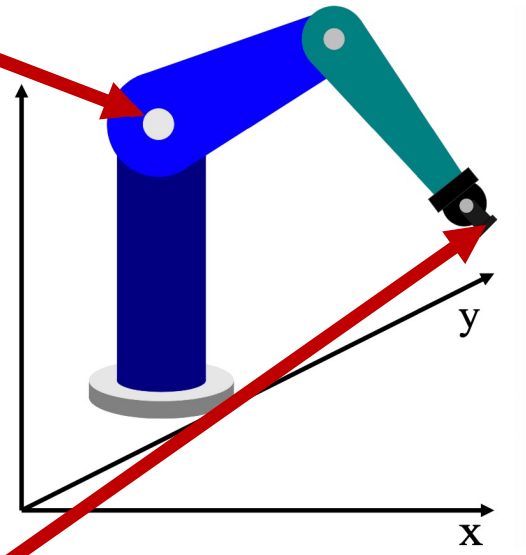
$$\frac{\mathrm{d}\boldsymbol{p}}{\mathrm{d}\boldsymbol{q}} = \boldsymbol{J}(\boldsymbol{q})$$

$$\mathrm{d}\boldsymbol{p} = \boldsymbol{J}(\boldsymbol{q})\mathrm{d}\boldsymbol{q}$$

https://en.wikipedia.org/wiki/Jacobian_matrix_and_determinant

Jacobian matrix

$$\boldsymbol{q} = \begin{bmatrix} q_1 \\ q_2 \\ \dots \\ q_{n-1} \\ q_n \end{bmatrix}$$



$$\boldsymbol{p} = \begin{bmatrix} x \\ y \\ z \\ \varphi \\ \vartheta \\ \psi \end{bmatrix}$$

$$\frac{\mathrm{d}\boldsymbol{p}}{\mathrm{d}\boldsymbol{q}} = J(\boldsymbol{q})$$

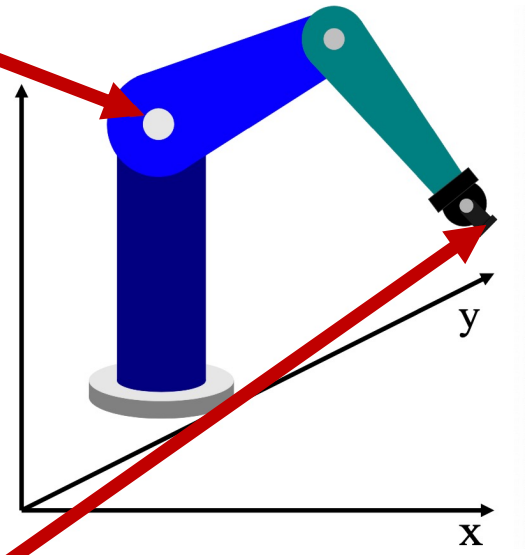
$$\mathrm{d}\boldsymbol{p} = J(\boldsymbol{q})\mathrm{d}\boldsymbol{q}$$

$$\frac{\mathrm{d}\boldsymbol{p}}{\mathrm{d}t} = J(\boldsymbol{q})\frac{\mathrm{d}\boldsymbol{q}}{\mathrm{d}t}$$

https://en.wikipedia.org/wiki/Jacobian_matrix_and_determinant

Jacobian matrix

$$\boldsymbol{q} = \begin{bmatrix} q_1 \\ q_2 \\ \dots \\ q_{n-1} \\ q_n \end{bmatrix}$$



$$\boldsymbol{p} = \begin{bmatrix} x \\ y \\ z \\ \varphi \\ \vartheta \\ \psi \end{bmatrix}$$

$$\frac{\mathrm{d}\boldsymbol{p}}{\mathrm{d}\boldsymbol{q}} = J(\boldsymbol{q})$$

$$\mathrm{d}\boldsymbol{p} = J(\boldsymbol{q})\mathrm{d}\boldsymbol{q}$$

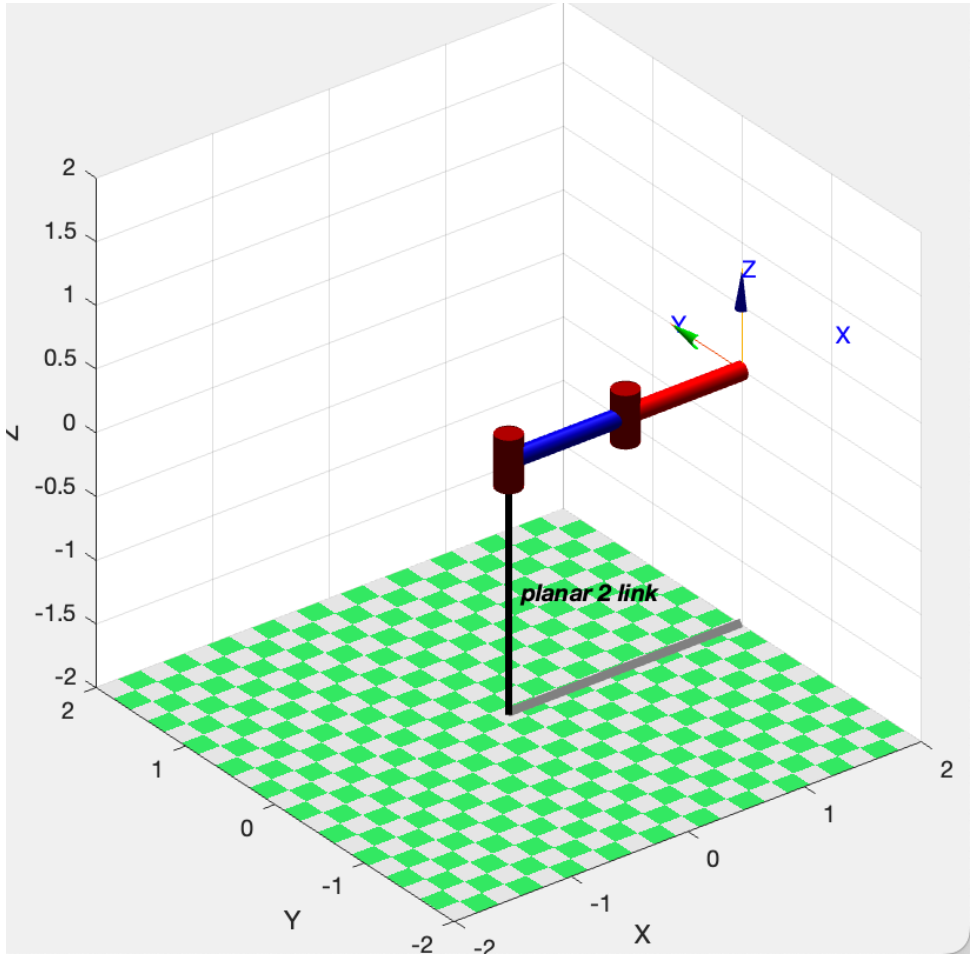
$$\frac{\mathrm{d}\boldsymbol{p}}{\mathrm{d}t} = J(\boldsymbol{q})\frac{\mathrm{d}\boldsymbol{q}}{\mathrm{d}t}$$

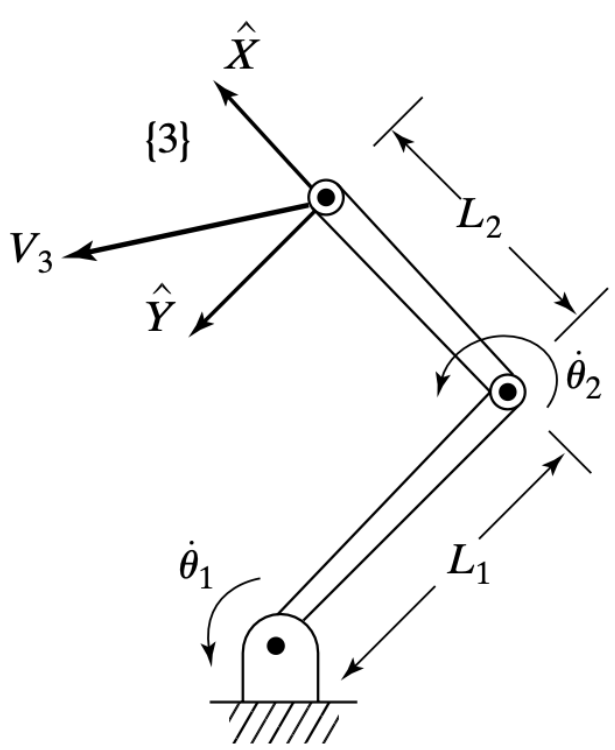
$$\boxed{\dot{\boldsymbol{p}} = J(\boldsymbol{q})\dot{\boldsymbol{q}}}$$

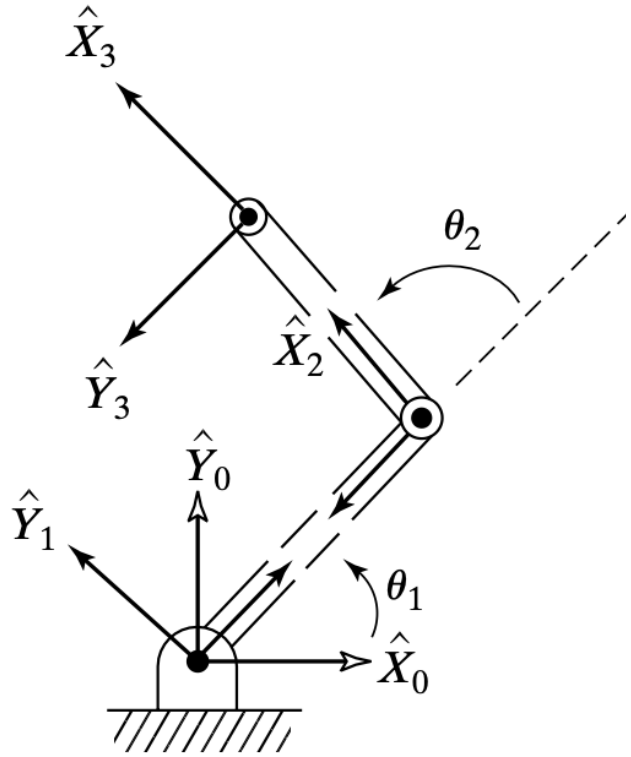
https://en.wikipedia.org/wiki/Jacobian_matrix_and_determinant

Jacobian matrix - Example 1

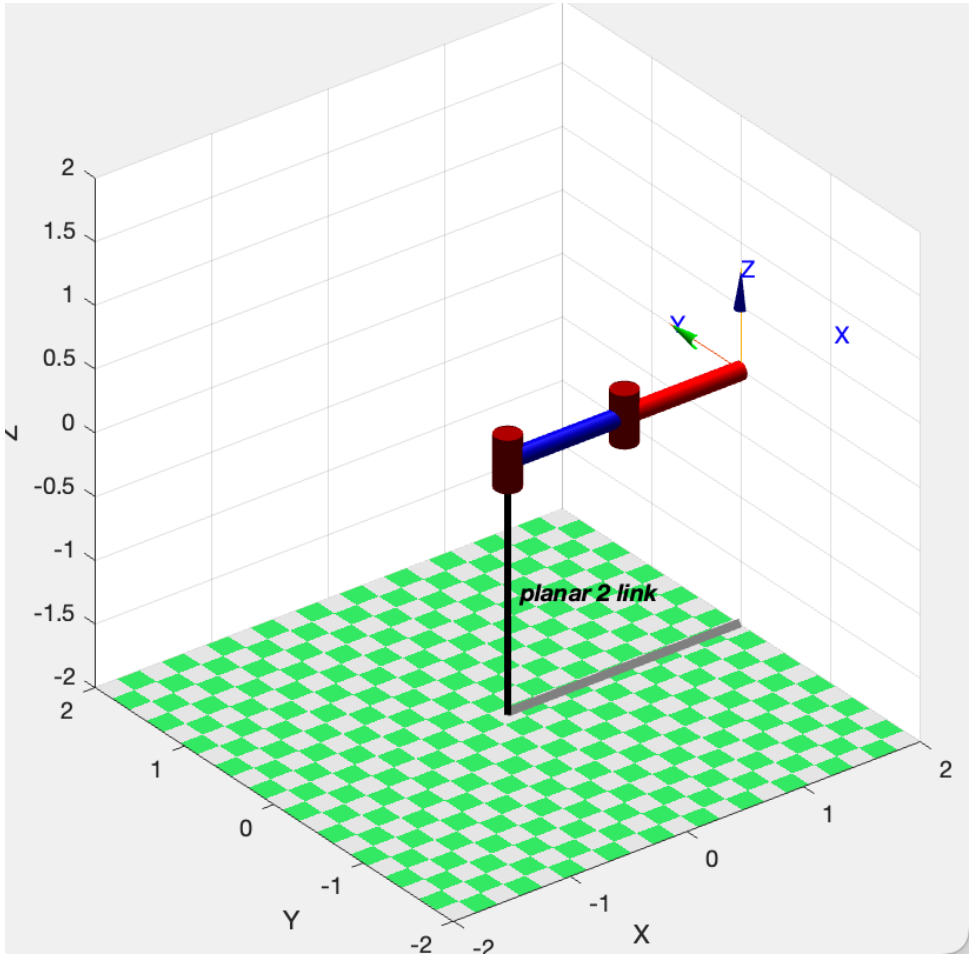
- Calculate the Jacobian matrix of the RR planar manipulator!







Jacobian matrix - Example 1 - Procedure



1. Derive the Forward Kinematics (FK)
2. Write the position equations
3. Take derivative for velocity
4. Put the matrix form: $\dot{p} = J(q)\dot{q}$

Jacobian matrix - Example 1 - Procedure - RR

- Derive the Forward Kinematics (FK)

```
mdl_planar2;
p2
```

```
p2 =

planar 2 link:: 2 axis, RR, stdDH, slowRNE
+---+----------+----------+----------+----------+----------+
| j |    theta |        d |        a |    alpha |   offset |
+---+----------+----------+----------+----------+----------+
|  1|        q1|         0|         1|         0|         0|
|  2|        q2|         0|         1|         0|         0|
+---+----------+----------+----------+----------+----------+
```

Jacobian matrix - Example 1 - Procedure - Step 1

- Derive the Forward Kinematics (FK)

```
mdl_planar2;
p2

syms q1 q2 real
FK = p2.fkine([q1 q2])
```

```
p2 =

planar 2 link:: 2 axis, RR, stdDH, slowRNE
+---+----------+----------+----------+----------+----------+
| j |    theta |        d |        a |    alpha |   offset |
+---+----------+----------+----------+----------+----------+
|  1|        q1|         0|         1|         0|         0|
|  2|        q2|         0|         1|         0|         0|
+---+----------+----------+----------+----------+----------+
```

```
[cos(q1 + q2), -sin(q1 + q2), 0, cos(q1 + q2) + cos(q1)]
[sin(q1 + q2),  cos(q1 + q2), 0, sin(q1 + q2) + sin(q1)]
[           0,             0, 1,                      0]
[           0,             0, 0,                      1]
```

Jacobian matrix - Example 1 - Procedure - Step 2

- Write the position equations

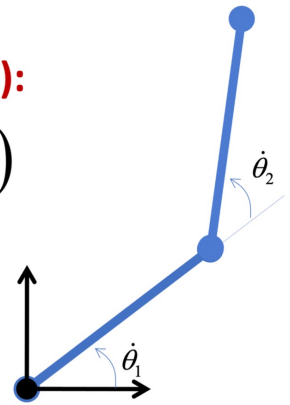
```
mdl_planar2;
p2

syms q1 q2 real
FK = p2.fkine([q1 q2])
T = FK.t(1:2)
```

```
p2 =

planar 2 link:: 2 axis, RR, stdDH, slowRNE
+---+----------+----------+----------+----------+----------+
| j |     theta |         d |         a |     alpha |    offset |
+---+----------+----------+----------+----------+----------+
|  1|         q1|          0|          1|          0|          0|
|  2|         q2|          0|          1|          0|          0|
+---+----------+----------+----------+----------+----------+


[cos(q1 + q2), -sin(q1 + q2), 0, cos(q1 + q2) + cos(q1)]
[sin(q1 + q2),  cos(q1 + q2), 0, sin(q1 + q2) + sin(q1)]
[           0,             0, 1,                      0]
[           0,             0, 0,                      1]
```

```
T =

cos(q1 + q2) + cos(q1)
sin(q1 + q2) + sin(q1)
```

Forward Kinematics (Last column of T):

$$x_t = l_1\cos(\theta_1) + l_2\cos(\theta_1 + \theta_2)$$
$$y_t = l_1\sin(\theta_1) + l_2\sin(\theta_1 + \theta_2)$$



Jacobian matrix - Example 1 - Procedure - Step 3

- Take derivative for velocity

```
mdl_planar2;
p2

syms q1 q2 real
FK = p2.fkine([q1 q2])
T = FK.t(1:2)

syms jc11 jc12 jc21 jc22
Jacob = [jc11 jc12; jc21 jc22];
Jacob(1,1) = diff(T(1,1), q1);
Jacob(1,2) = diff(T(1,1), q2);
Jacob(2,1) = diff(T(2,1), q1);
Jacob(2,2) = diff(T(2,1), q2);
```

Time derivative:

$$\dot{x}_t = -l_1\dot{\theta}_1\sin(\theta_1) - l_2\left(\dot{\theta}_1+\dot{\theta}_2\right)\sin(\theta_1+\theta_2)$$

$$\dot{y}_t = l_1\dot{\theta}_1\cos(\theta_1) + l_2\left(\dot{\theta}_1+\dot{\theta}_2\right)\cos(\theta_1+\theta_2)$$

```
p2 =

planar 2 link:: 2 axis, RR, stdDH, slowRNE
+---+----------+----------+----------+----------+----------+
| j |    theta |        d |        a |    alpha |   offset |
+---+----------+----------+----------+----------+----------+
|  1|        q1|         0|         1|         0|         0|
|  2|        q2|         0|         1|         0|         0|
+---+----------+----------+----------+----------+----------+


[cos(q1 + q2), -sin(q1 + q2), 0, cos(q1 + q2) + cos(q1)]
[sin(q1 + q2),  cos(q1 + q2), 0, sin(q1 + q2) + sin(q1)]
[           0,             0, 1,                      0]
[           0,             0, 0,                      1]


T =

cos(q1 + q2) + cos(q1)
sin(q1 + q2) + sin(q1)


Jacob =

[- sin(q1 + q2) - sin(q1), -sin(q1 + q2)]
[  cos(q1 + q2) + cos(q1),  cos(q1 + q2)]
```

Jacobian matrix - Example 1 - Procedure - Step 4

- Put the matrix form: $\dot{p} = J(q)\dot{q}$

```
mdl_planar2;
p2

syms q1 q2 real
FK = p2.fkine([q1 q2])
T = FK.t(1:2)

syms jc11 jc12 jc21 jc22
Jacob = [jc11 jc12; jc21 jc22];
Jacob(1,1) = diff(T(1,1), q1);
Jacob(1,2) = diff(T(1,1), q2);
Jacob(2,1) = diff(T(2,1), q1);
Jacob(2,2) = diff(T(2,1), q2);
```

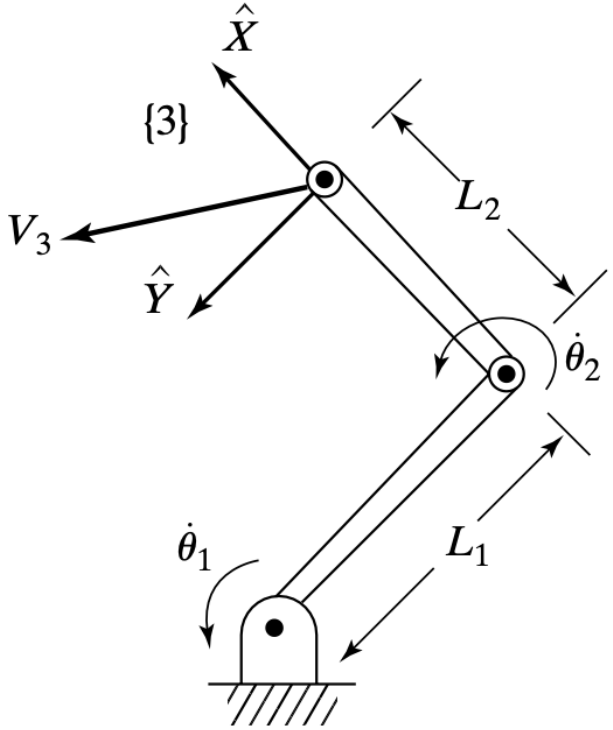
Put in Vector/Matrix Form:

$$\begin{bmatrix} \dot{x}_t \\ \dot{y}_t \\ \dot{z}_t \end{bmatrix} = \begin{bmatrix} -l_1\sin(\theta_1) - l_2\sin(\theta_1+\theta_2) & -l_2\sin(\theta_1+\theta_2) \\ l_1\cos(\theta_1) + l_2\cos(\theta_1+\theta_2) & l_2\cos(\theta_1+\theta_2) \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix}$$

3x1 — J(q) — 3x2 — 2x1

```
p2 =

planar 2 link:: 2 axis, RR, stdDH, slowRNE
+---+----------+----------+----------+----------+----------+
| j |    theta |        d |        a |    alpha |   offset |
+---+----------+----------+----------+----------+----------+
|  1|        q1|         0|         1|         0|         0|
|  2|        q2|         0|         1|         0|         0|
+---+----------+----------+----------+----------+----------+


[cos(q1 + q2), -sin(q1 + q2), 0, cos(q1 + q2) + cos(q1)]
[sin(q1 + q2),  cos(q1 + q2), 0, sin(q1 + q2) + sin(q1)]
[           0,             0, 1,                      0]
[           0,             0, 0,                      1]


T =

cos(q1 + q2) + cos(q1)
sin(q1 + q2) + sin(q1)


Jacob =

[- sin(q1 + q2) - sin(q1), -sin(q1 + q2)]
[  cos(q1 + q2) + cos(q1),  cos(q1 + q2)]
```

Jacobian matrix - Example 1 - 6 x 2 matrix



$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \begin{bmatrix} -l_1\sin(\theta_1) - l_2\sin(\theta_1+\theta_2) & -l_2\sin(\theta_1+\theta_2) \\ l_1\cos(\theta_1) + l_2\cos(\theta_1+\theta_2) & l_2\cos(\theta_1+\theta_2) \\ 0 & 0 \\ \hline & \\ & \end{bmatrix} \cdot \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix}$$

6 x 2

Jacobian matrix - Example 1 - World Frame

```
mdl_planar2;
p2

syms q1 q2 real
FK = p2.fkine([q1 q2])
T = FK.t(1:2)

syms jc11 jc12 jc21 jc22
Jacob = [jc11 jc12; jc21 jc22];
Jacob(1,1) = diff(T(1,1), q1);
Jacob(1,2) = diff(T(1,1), q2);
Jacob(2,1) = diff(T(2,1), q1);
Jacob(2,2) = diff(T(2,1), q2);
```

$$ {}^{0}J(\Theta) = \begin{bmatrix} -l_1 s_1 - l_2 s_{12} & -l_2 s_{12} \\ l_1 c_1 + l_2 c_{12} & l_2 c_{12} \end{bmatrix} $$

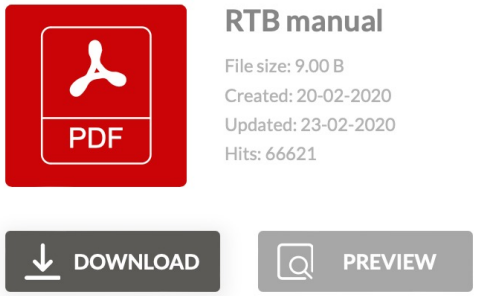
```
p2 =

planar 2 link:: 2 axis, RR, stdDH, slowRNE
+---+----------+----------+----------+----------+----------+
| j |     theta |         d |         a |     alpha |    offset |
+---+----------+----------+----------+----------+----------+
|  1|         q1|          0|          1|          0|          0|
|  2|         q2|          0|          1|          0|          0|
+---+----------+----------+----------+----------+----------+


[cos(q1 + q2), -sin(q1 + q2), 0, cos(q1 + q2) + cos(q1)]
[sin(q1 + q2),  cos(q1 + q2), 0, sin(q1 + q2) + sin(q1)]
[           0,             0, 1,                      0]
[           0,             0, 0,                      1]


T =

cos(q1 + q2) + cos(q1)
sin(q1 + q2) + sin(q1)


Jacob =

[- sin(q1 + q2) - sin(q1), -sin(q1 + q2)]
[  cos(q1 + q2) + cos(q1),  cos(q1 + q2)]
```

Jacobian matrix - Change frame of reference

$$ {}^{E}\nu = \begin{pmatrix} {}^{E}\boldsymbol{R}_0 & 0_{3\times3} \\ 0_{3\times3} & {}^{E}\boldsymbol{R}_0 \end{pmatrix} {}^{0}\boldsymbol{J}(q)\,\dot{q} = {}^{E}\boldsymbol{J}(q)\dot{q} $$

`jacob0` outputs a manipulator Jacobian matrix, with respect to the world frame, based on the input joint coordinate vector. outputs the Jacobian matrix. The robot object is a parameter.

`jacobe` outputs a manipulator Jacobian matrix, with respect to the end-effector frame, based on the input joint coordinate vector. outputs the Jacobian matrix. The robot object is a parameter.



https://petercorke.com/download/27/rtb/1050/rtb-manual.pdf

Jacobian matrix - Example 1 - Frame {0}

Change frame of reference

```
mdl_planar2;
p2

syms q1 q2 real

J0 = p2.jacob0([q1 q2]);
J_frame_0 = J0(1:2, :)
```

$$ {}^{0}J(\Theta) = \begin{bmatrix} -l_1 s_1 - l_2 s_{12} & -l_2 s_{12} \\ l_1 c_1 + l_2 c_{12} & l_2 c_{12} \end{bmatrix} $$

```
p2 =

planar 2 link:: 2 axis, RR, stdDH, slowRNE
+---+----------+----------+----------+----------+----------+
| j |    theta |        d |        a |    alpha |   offset |
+---+----------+----------+----------+----------+----------+
|  1|        q1|         0|         1|         0|         0|
|  2|        q2|         0|         1|         0|         0|
+---+----------+----------+----------+----------+----------+


J_frame_0 =

[cos(q1 + q2)*sin(q2) - sin(q1 + q2)*(cos(q2) + 1), -sin(q1 + q2)]
[cos(q1 + q2)*(cos(q2) + 1) + sin(q1 + q2)*sin(q2),  cos(q1 + q2)]
```

Jacobian matrix - Example 1 - Frame {3}

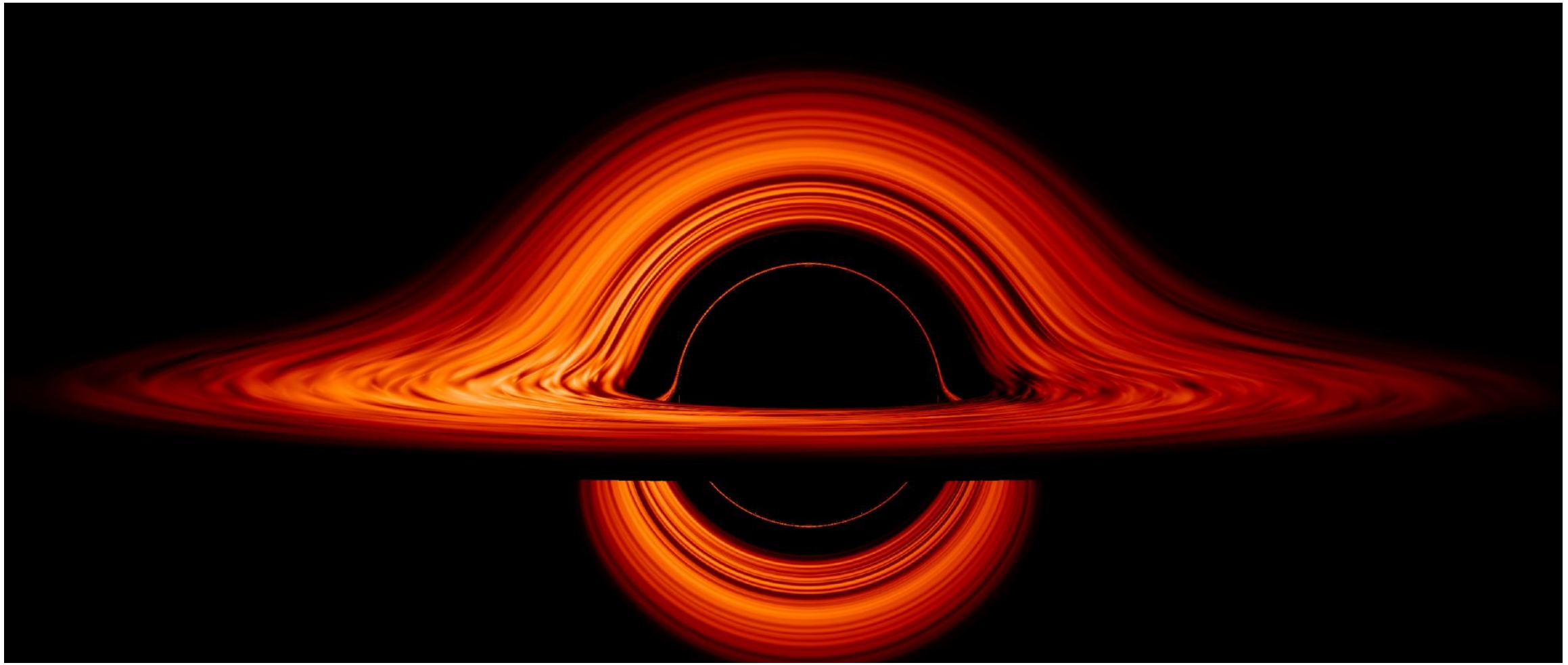
Change frame of reference

```
mdl_planar2;
p2

syms q1 q2 real

J0 = p2.jacob0([q1 q2]);
J_frame_0 = J0(1:2, :)

Je = p2.jacobe([q1 q2]);
J_frame_3 = Je(1:2, :)
```

$$^{0}J(\Theta) = \begin{bmatrix} -l_1 s_1 - l_2 s_{12} & -l_2 s_{12} \\ l_1 c_1 + l_2 c_{12} & l_2 c_{12} \end{bmatrix}$$

$$^{3}J(\Theta) = \begin{bmatrix} l_1 s_2 & 0 \\ l_1 c_2 + l_2 & l_2 \end{bmatrix}$$

```
p2 =

planar 2 link:: 2 axis, RR, stdDH, slowRNE
+---+----------+----------+----------+----------+----------+
| j |    theta |        d |        a |    alpha |   offset |
+---+----------+----------+----------+----------+----------+
|  1|        q1|         0|         1|         0|         0|
|  2|        q2|         0|         1|         0|         0|
+---+----------+----------+----------+----------+----------+


J_frame_0 =

[cos(q1 + q2)*sin(q2) - sin(q1 + q2)*(cos(q2) + 1), -sin(q1 + q2)]
[cos(q1 + q2)*(cos(q2) + 1) + sin(q1 + q2)*sin(q2),  cos(q1 + q2)]


J_frame_3 =

[    sin(q2), 0]
[cos(q2) + 1, 1]
```



What do robots and black holes have in common?

Event Horizon Telescope collaboration et al.

They both have singularities !!!

Singularity:

- A point of infinite density and infinitesimal volume, at which space and time become infinitely distorted according to the theory of General Relativity.
- According to the big bang theory, a gravitational singularity existed at the beginning of the universe.
- Singularities are also believed to exist at the center of black holes.

- ***Mathematics:*** A point at which the derivative does not exist for a given function but every neighborhood of which contains points for which the derivative exists.

Kinematics singularities

- **A singularity occurs when Jacobian is NOT FULL RANK!**

  rank(J) < n:

  where n = number of rows or columns in J, whichever is smaller

- **A singularity occurs when DETERMINANT of Jacobian is ZERO!**

  det(J) = 0

- **Beware of singularities:**
  1. Singularities represent configurations at which mobility of the structure is reduced: it is **not possible to impose an arbitrary motion** to the end-effector.
  2. When the structure is at a singularity, **infinite solutions to the inverse kinematics problem may exist.**
  3. Around a singularity, small velocities in the operational space may cause **large velocities in the joint space**.

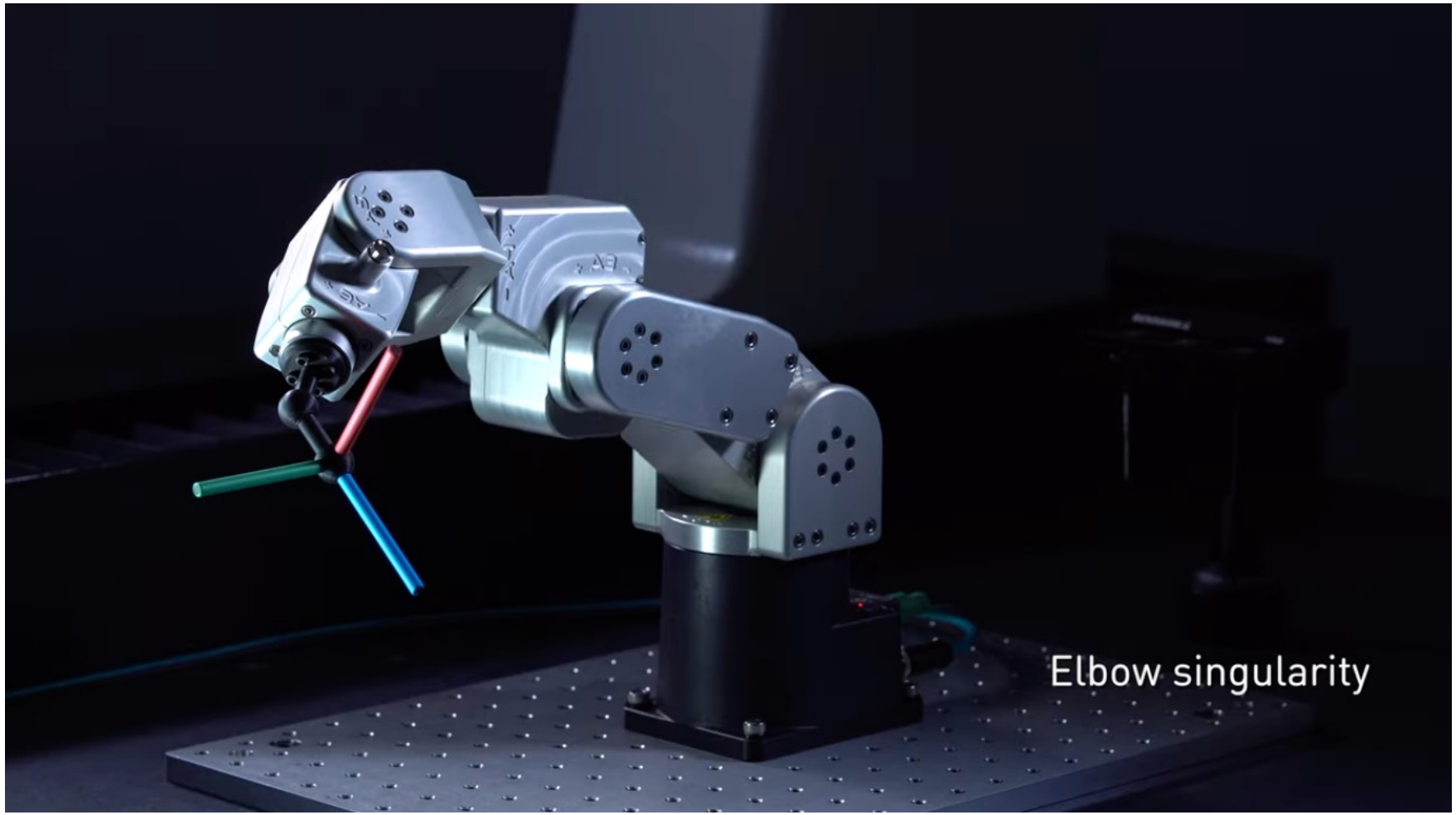
Singularities of a 6-DoF robot

https://www.youtube.com/watch?v=lD2HQcxeNoA

Singularities of a 6-DoF robot - Wrist

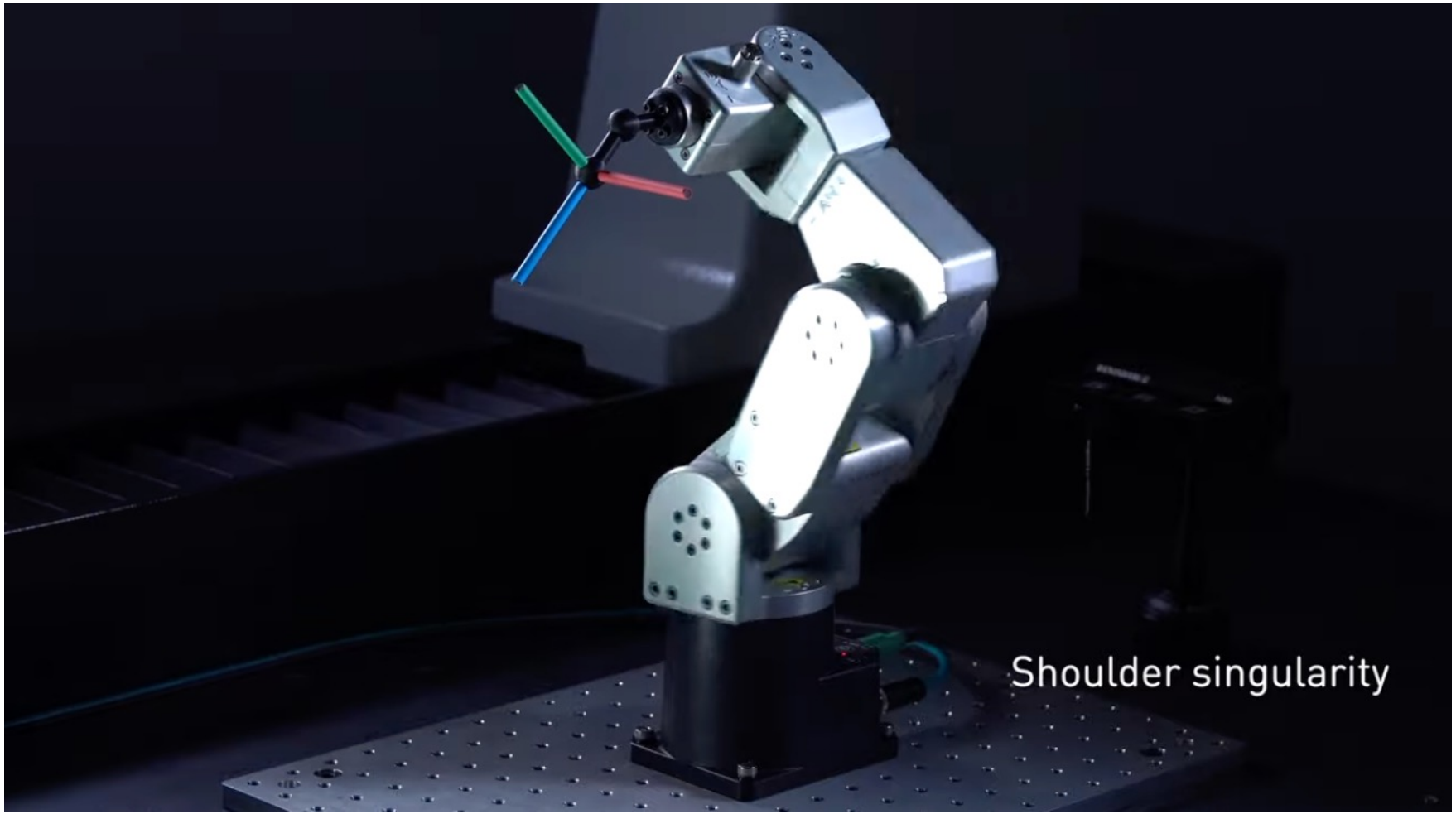
https://www.youtube.com/watch?v=lD2HQcxeNoA

Singularities of a 6-DoF robot - Elbow



https://www.youtube.com/watch?v=lD2HQcxeNoA

Singularities of a 6-DoF robot - Shoulder



https://www.youtube.com/watch?v=lD2HQcxeNoA

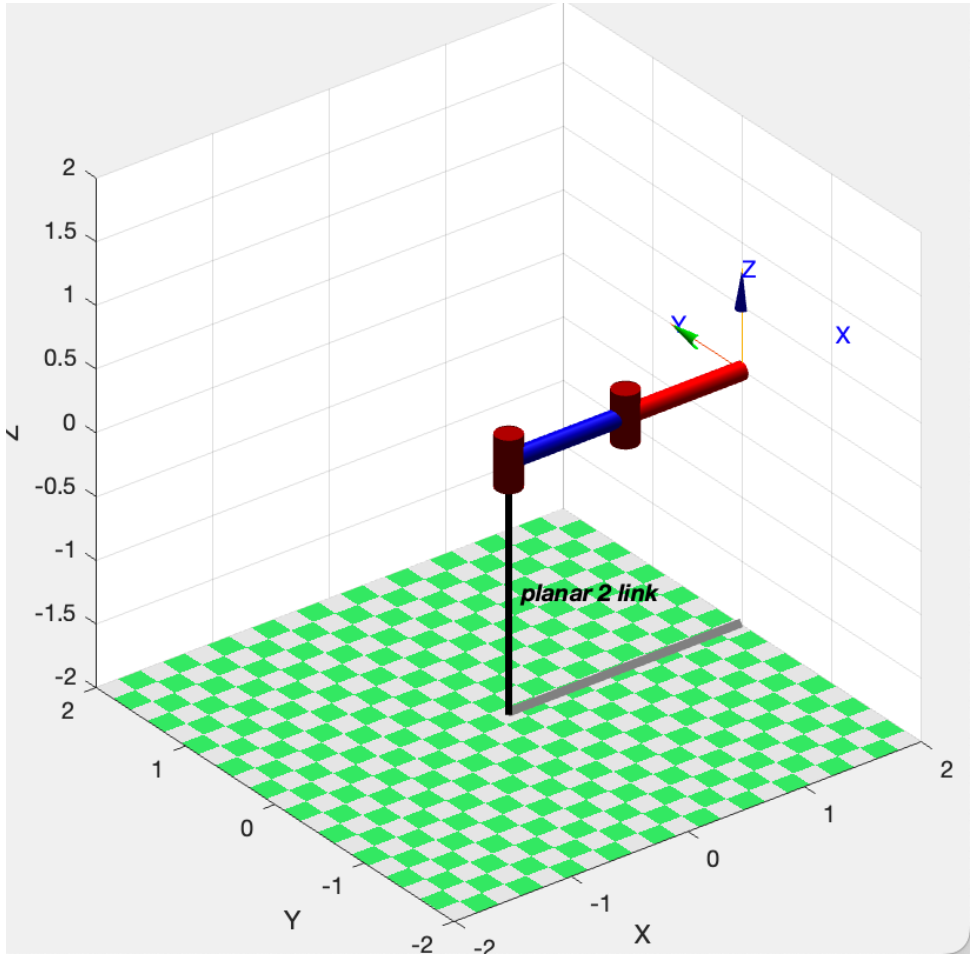
Singularities and categories

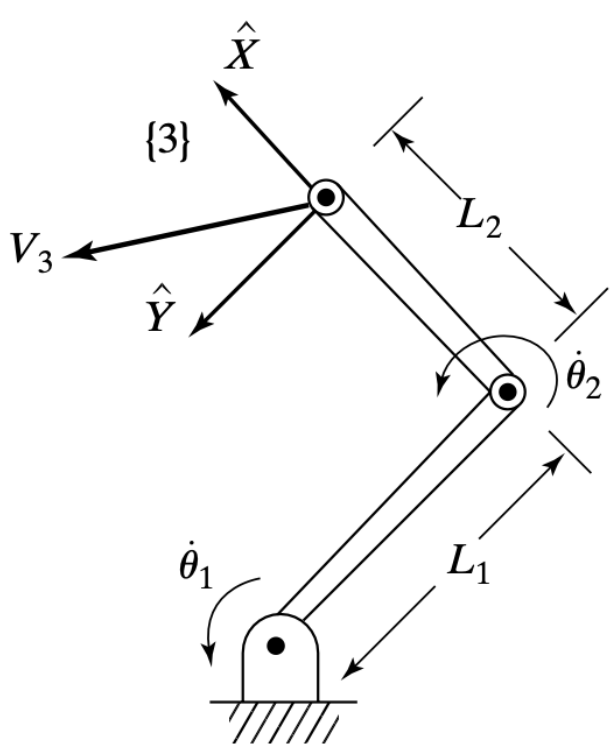
- **Workspace-boundary singularities:** occur when the manipulator is fully stretched out or folded back on itself in such a way that the end-effector is at or very near the boundary of the workspace

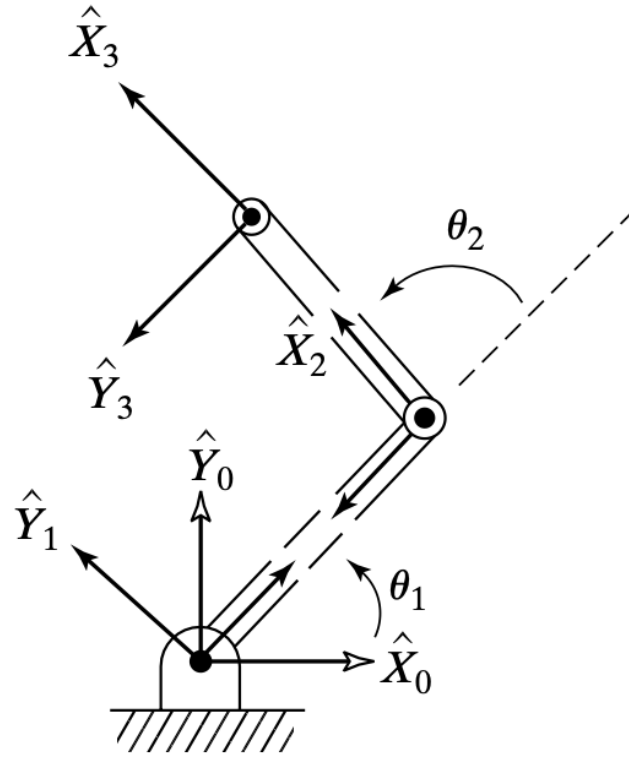
- **Workspace-interior singularities** occur away from the workspace boundary are caused by a lining up of two or more joint axes.

Singularities - Example 2

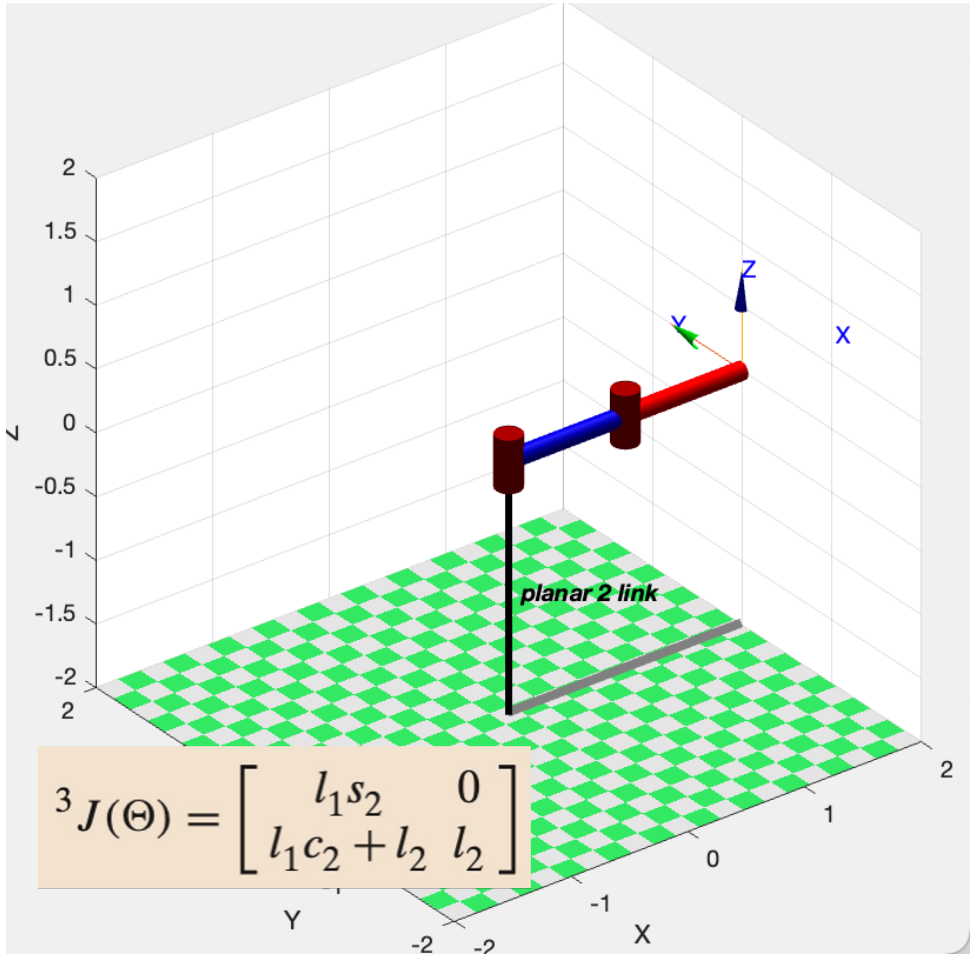
- Where are the singularities for the RR planar manipulator?





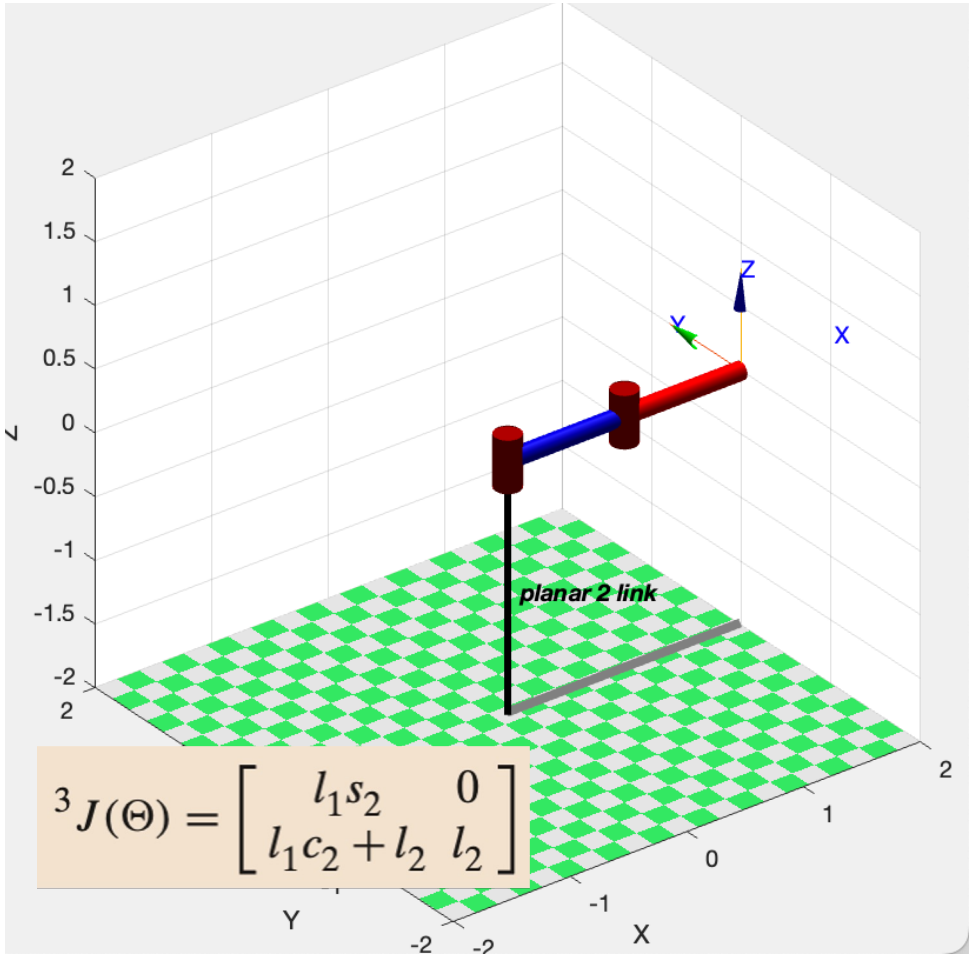


Singularities - Example 2 - Solution



$$^{3}J(\Theta) = \begin{bmatrix} l_1 s_2 & 0 \\ l_1 c_2 + l_2 & l_2 \end{bmatrix}$$

Singularities - Example 2 - Solution



$$^{3}J(\Theta) = \begin{bmatrix} l_1 s_2 & 0 \\ l_1 c_2 + l_2 & l_2 \end{bmatrix}$$

$$DET[J(\Theta)] = \begin{bmatrix} l_1 s_2 & 0 \\ l_1 c_2 + l_2 & l_2 \end{bmatrix} = l_1 l_2 s_2 = 0.$$

Singularity of the mechanism exists when:

- θ_2 is 0 degrees

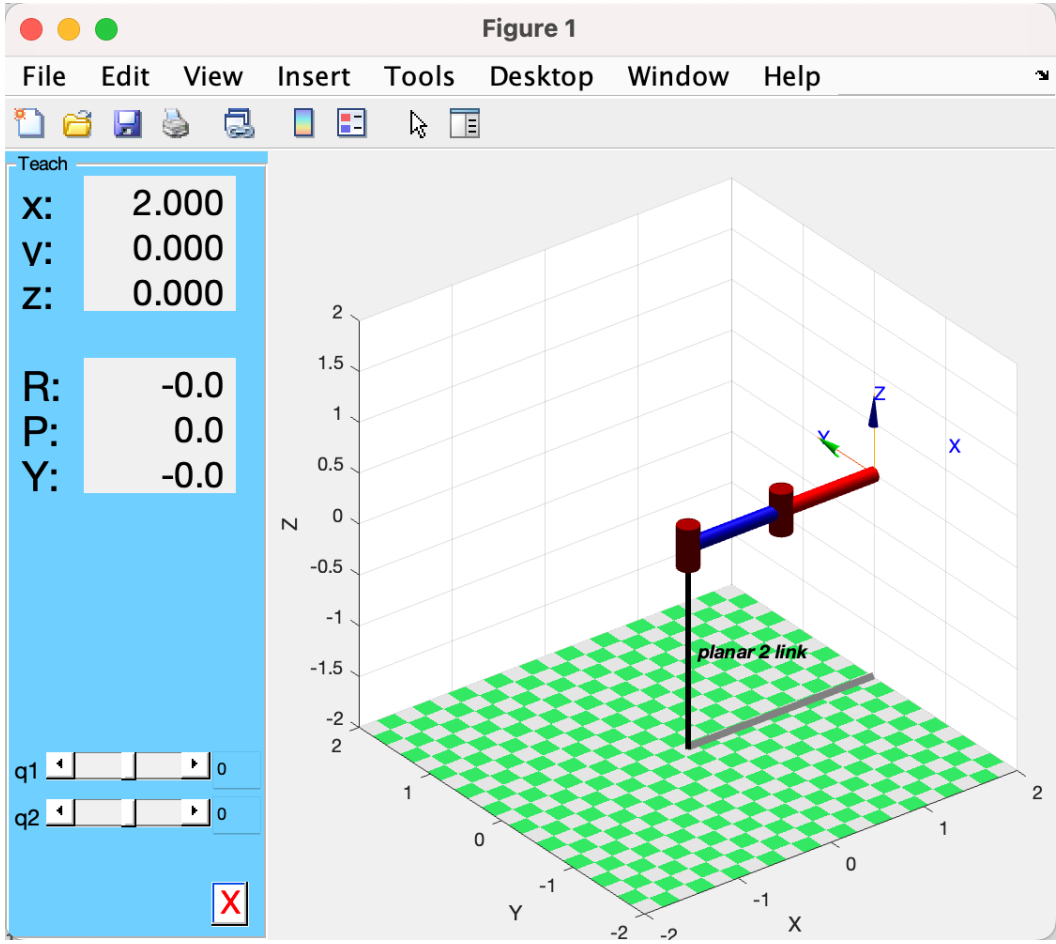
or

- θ_2 is 180 degrees

Workspace-boundary singularities

Singularities - Example 2 - Workspace boundary

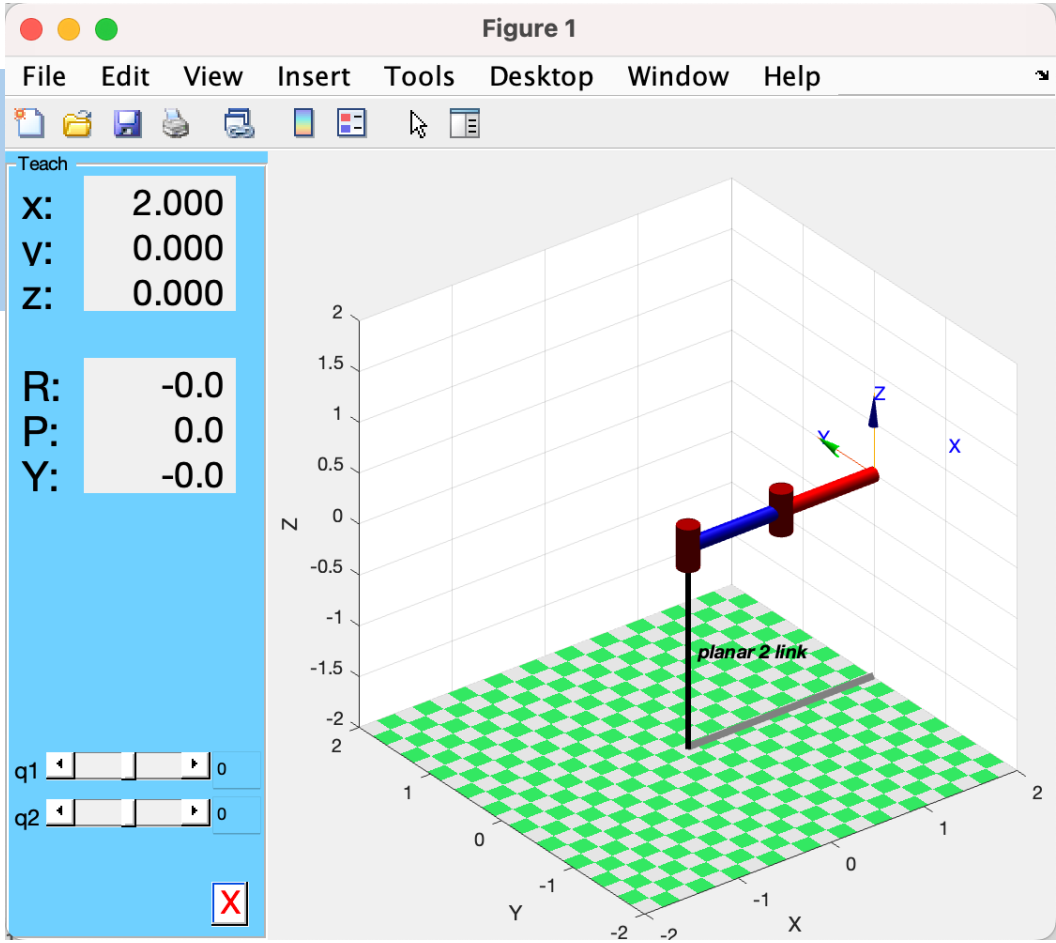
```
mdl_planar2;
p2

% At workspace boundaries
q0 = [0 0];
p2.teach(q0);
```



Singularities - Example 2 - Workspace boundary

```
mdl_planar2;
p2

% At workspace boundaries
q0 = [0 0];
p2.teach(q0);

J0 = p2.jacobe(q0);
J0sq = J0(1:2, :)
```

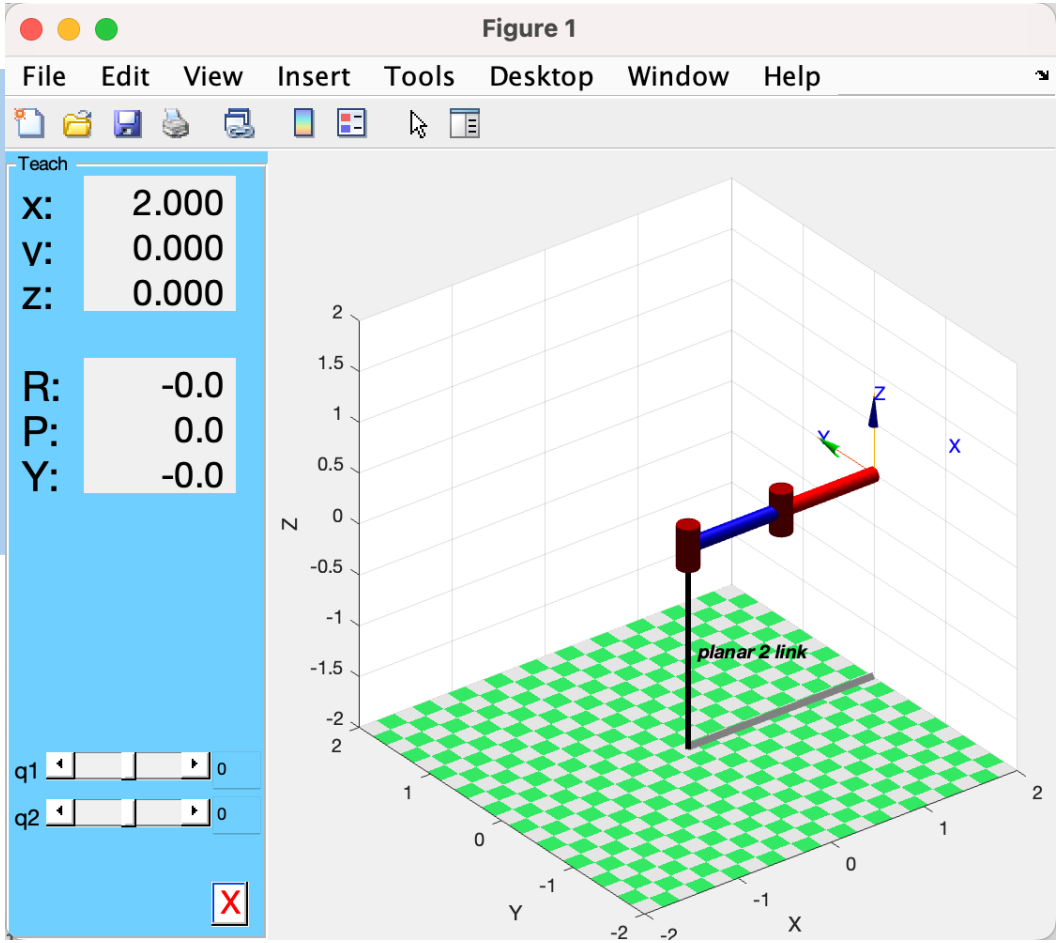
```
J0sq =

     0     0
     2     1
```



Singularities - Example 2 - Workspace boundary

```
mdl_planar2;
p2

% At workspace boundaries
q0 = [0 0];
p2.teach(q0);

J0 = p2.jacobe(q0);
J0sq = J0(1:2, :)
det(J0sq)
```

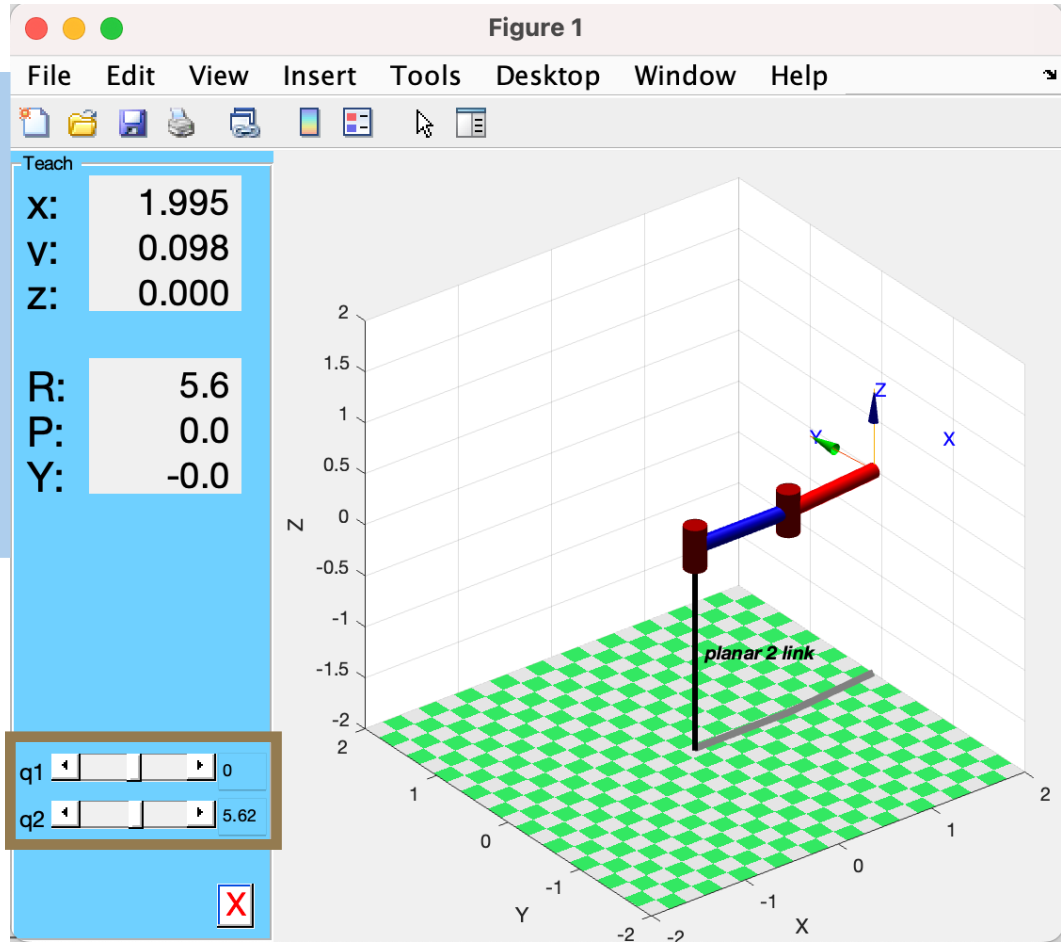
```
J0sq =

     0     0
     2     1


ans =

     0
```



Singularities - Example 2 - Close to Workspace b.

```
mdl_planar2;
p2

% At workspace boundaries
q0 = [0 0];
p2.teach(q0);

J0 = p2.jacobe(q0);
J0sq = J0(1:2, :)
det(J0sq)

% Close to workspace
% boundaries
q1 = [0 pi/32];
p2.teach(q1);
```

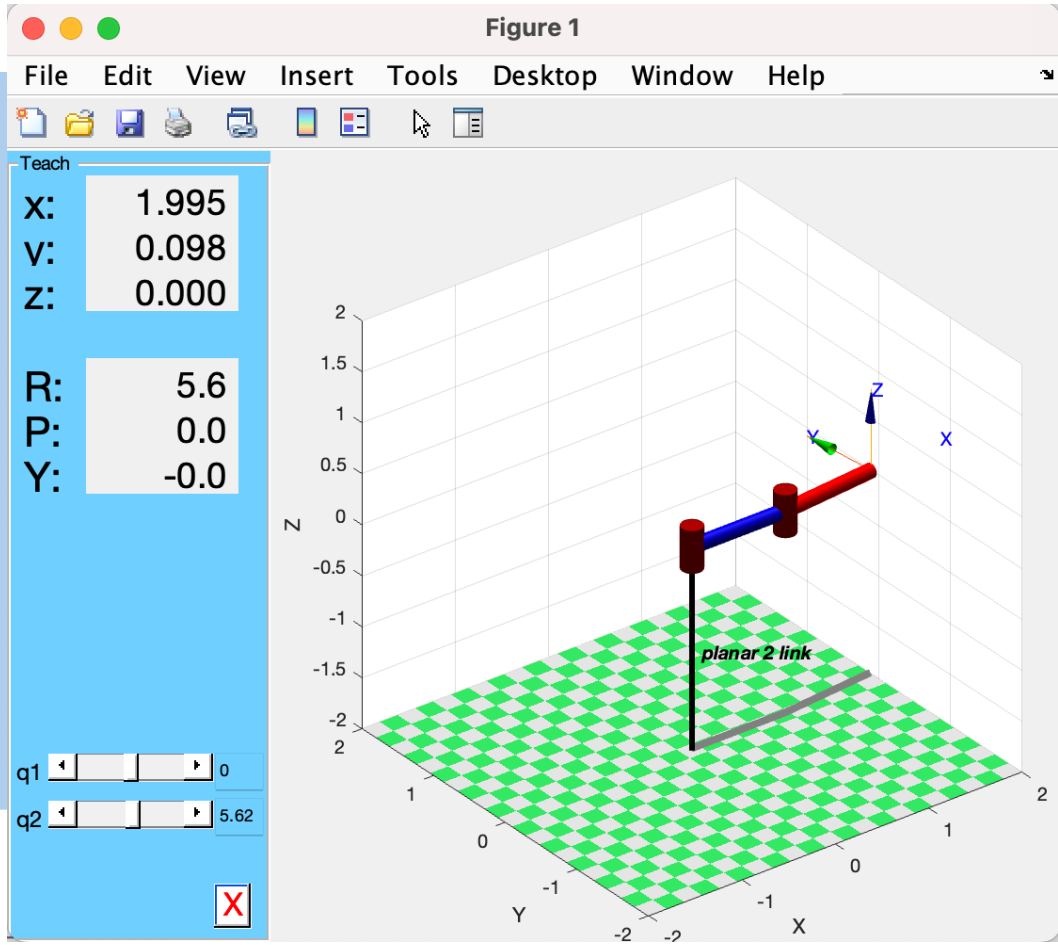
```
J0sq =

     0     0
     2     1


ans =

     0
```



Singularities - Example 2 - Close to Workspace b.

```
mdl_planar2;
p2

% At workspace boundaries
q0 = [0 0];
p2.teach(q0);

J0 = p2.jacobe(q0);
J0sq = J0(1:2, :)
det(J0sq)

% Close to workspace
% boundaries
q1 = [0 pi/32];
p2.teach(q1);

J1 = p2.jacobe(q1);
J1sq = J1(1:2, :)
```

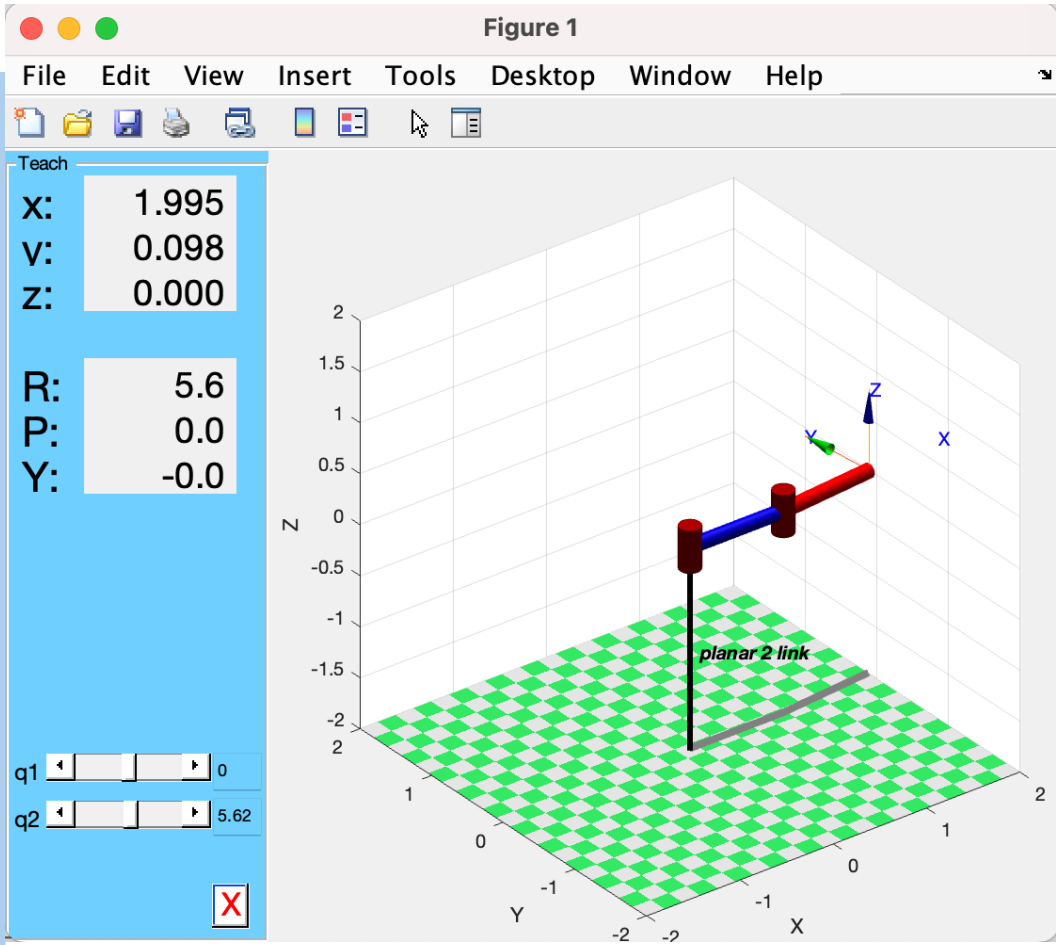
```
J0sq =

     0     0
     2     1


ans =

     0


J1sq =

   -0.0980   -0.0980
    1.9952    0.9952
```



Singularities - Example 2 - Close to Workspace b.

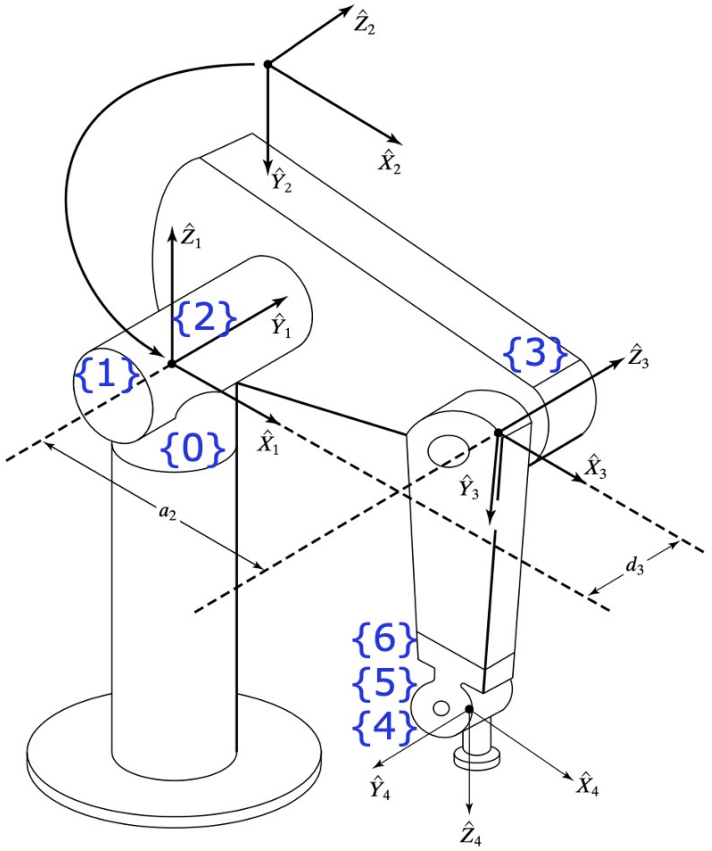
```
mdl_planar2;
p2

% At workspace boundaries
q0 = [0 0];
p2.teach(q0);

J0 = p2.jacobe(q0);
J0sq = J0(1:2, :)
det(J0sq)

% Close to workspace
% boundaries
q1 = [0 pi/32];
p2.teach(q1);

J1 = p2.jacobe(q1);
J1sq = J1(1:2, :)
det(J1sq)
```

```
J0sq =

     0     0
     2     1


ans =

     0


J1sq =

   -0.0980   -0.0980
    1.9952    0.9952


ans =

    0.0980
```



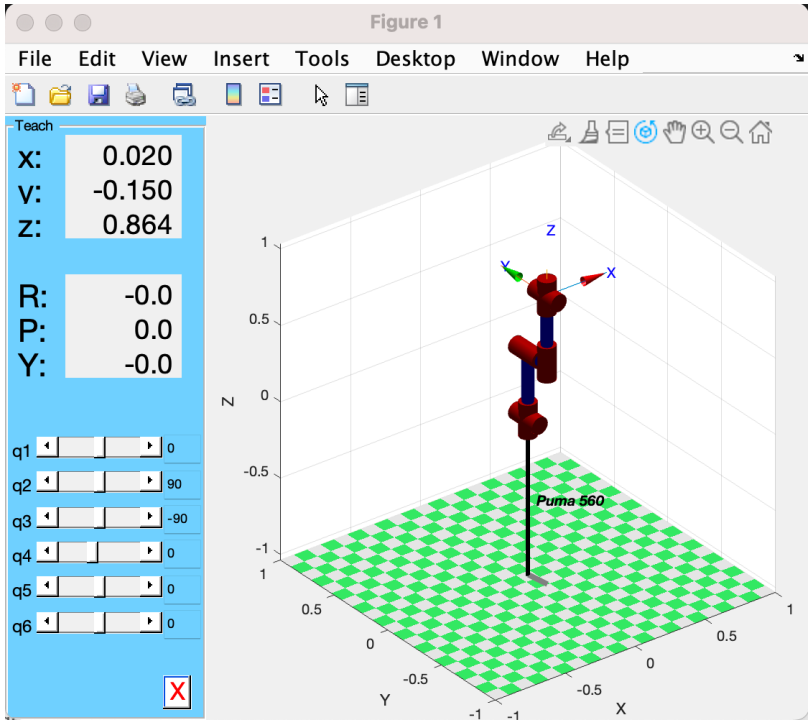
Singularities - Example 3

- Where are the singularities for the PUMA 560?



Singularities - Example 3 - Solution

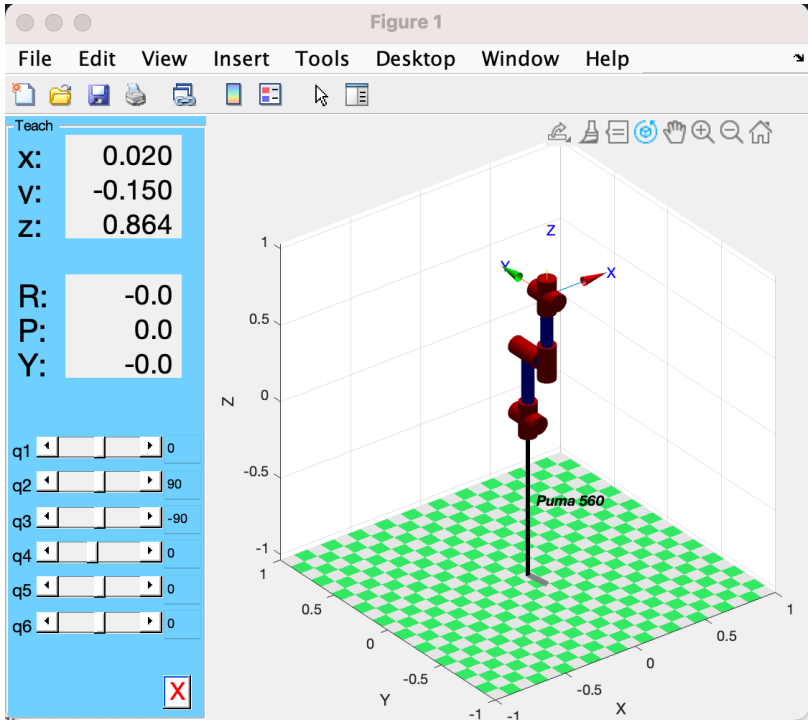
```
%% Workspace-boundary Singularities
mdl_puma560
p560
p560.teach(qr)
```



Singularities - Example 3 - Solution

```
%% Workspace-boundary Singularities
mdl_puma560
p560
p560.teach(qr)

Jqr = p560.jacob0(qr)
```

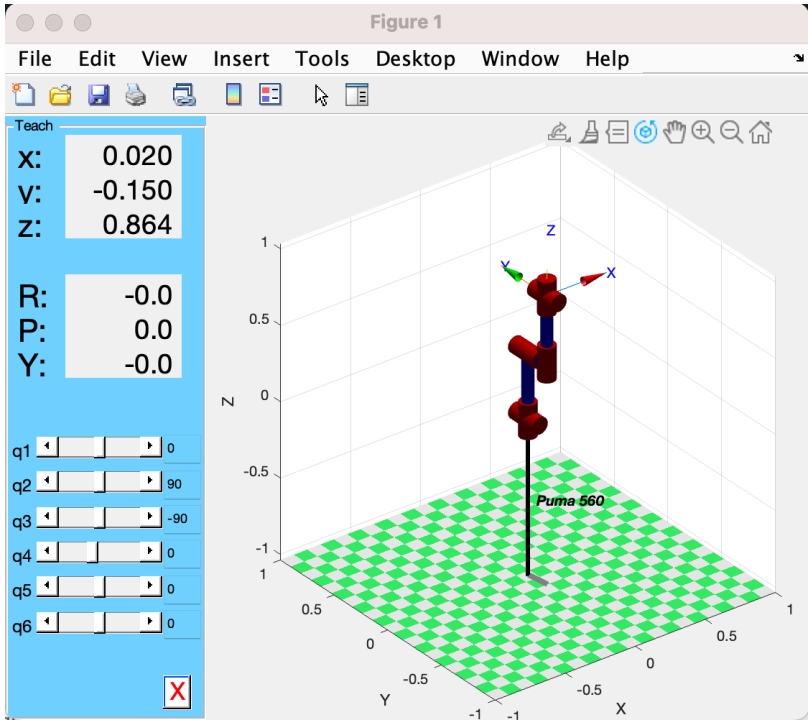
```
Jqr =

    0.1500   -0.8636   -0.4318         0         0         0
    0.0203    0.0000    0.0000         0         0         0
         0    0.0203    0.0203         0         0         0
         0         0         0         0         0         0
         0   -1.0000   -1.0000         0   -1.0000         0
    1.0000    0.0000    0.0000    1.0000    0.0000    1.0000
```



Singularities - Example 3 - Solution

```
%% Workspace-boundary Singularities
mdl_puma560
p560
p560.teach(qr)

Jqr = p560.jacob0(qr)

det(Jqr)
```

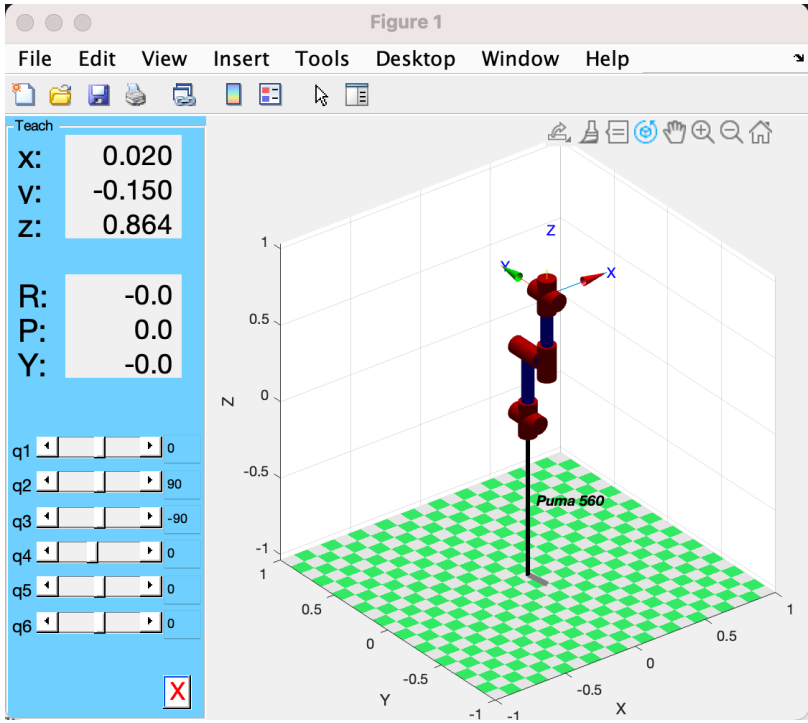
```
Jqr =

    0.1500   -0.8636   -0.4318         0         0         0
    0.0203    0.0000    0.0000         0         0         0
         0    0.0203    0.0203         0         0         0
         0         0         0         0         0         0
         0   -1.0000   -1.0000         0   -1.0000         0
    1.0000    0.0000    0.0000    1.0000    0.0000    1.0000


ans =

     0
```



Singularities - Example 3 - Solution

```
%% Workspace-boundary Singularities
mdl_puma560
p560
p560.teach(qr)

Jqr = p560.jacob0(qr)

det(Jqr)
rank(Jqr)
```

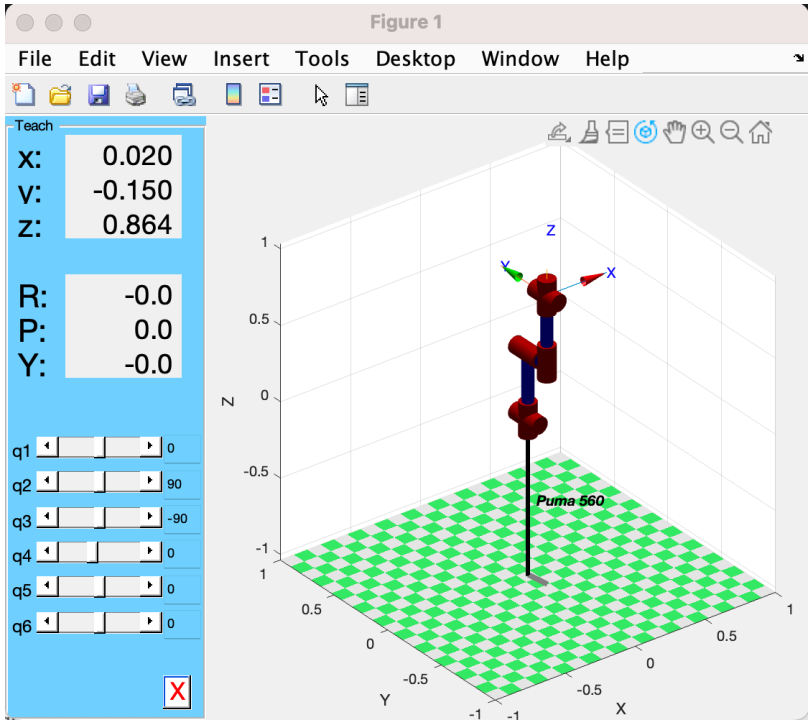
```
Jqr =

    0.1500   -0.8636   -0.4318         0         0         0
    0.0203    0.0000    0.0000         0         0         0
         0    0.0203    0.0203         0         0         0
         0         0         0         0         0         0
         0   -1.0000   -1.0000         0   -1.0000         0
    1.0000    0.0000    0.0000    1.0000    0.0000    1.0000


ans =

     0


ans =

     5
```



Singularities - Example 3 - Solution

```
%% Workspace-boundary Singularities
mdl_puma560
p560
p560.teach(qr)

Jqr = p560.jacob0(qr)

det(Jqr)
rank(Jqr)
jsingu(Jqr)
```

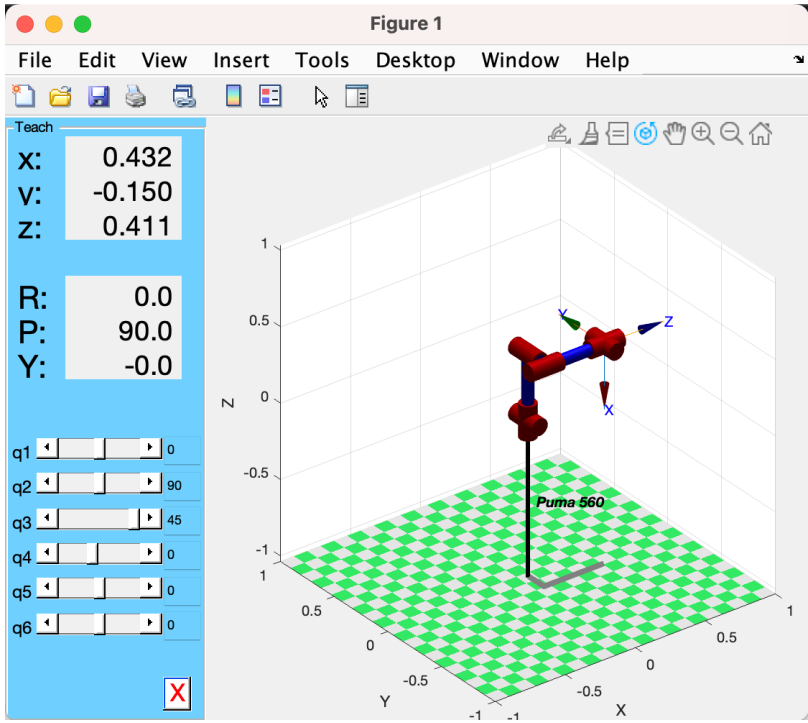
```
Jqr =

    0.1500   -0.8636   -0.4318         0         0         0
    0.0203    0.0000    0.0000         0         0         0
         0    0.0203    0.0203         0         0         0
         0         0         0         0         0         0
         0   -1.0000   -1.0000         0   -1.0000         0
    1.0000    0.0000    0.0000    1.0000    0.0000    1.0000


ans =

     0


ans =

     5

1 linearly dependent joints:
  q6 depends on: q4
```



Singularities - Example 3 - Solution

```
%% Workspace-boundary Singularities
mdl_puma560
p560
p560.teach(qr)

Jqr = p560.jacob0(qr)

det(Jqr)
rank(Jqr)
jsingu(Jqr)

%% Workspace-interior Singularities

qns = qr;
qns(3) = pi;

p560.teach(qns)
```

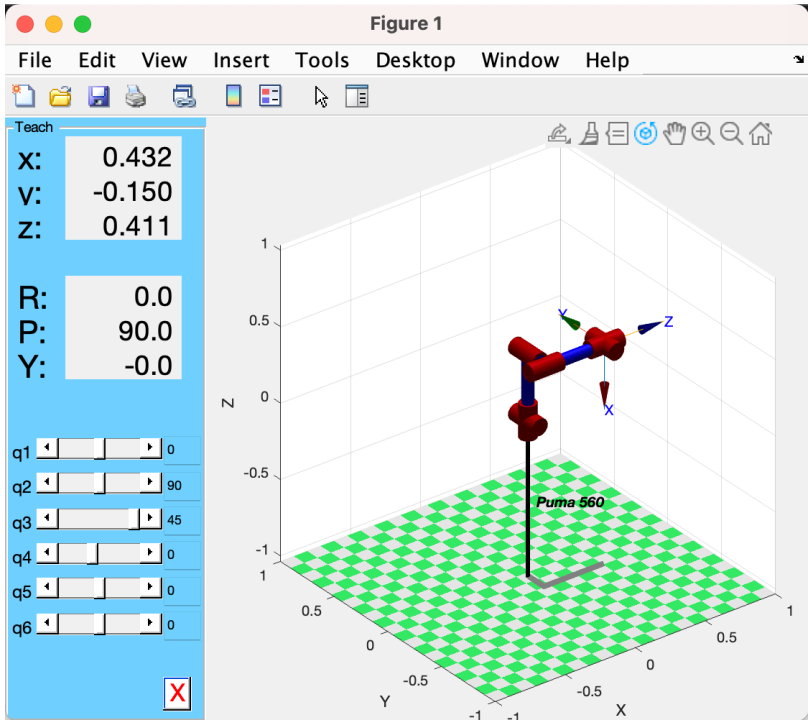
```
Jqr =

    0.1500   -0.8636   -0.4318         0         0         0
    0.0203    0.0000    0.0000         0         0         0
         0    0.0203    0.0203         0         0         0
         0         0         0         0         0         0
         0   -1.0000   -1.0000         0   -1.0000         0
    1.0000    0.0000    0.0000    1.0000    0.0000    1.0000


ans =

     0


ans =

     5

1 linearly dependent joints:
  q6 depends on: q4
```



Singularities - Example 3 - Solution

```
%% Workspace-boundary Singularities
mdl_puma560
p560
p560.teach(qr)

Jqr = p560.jacob0(qr)

det(Jqr)
rank(Jqr)
jsingu(Jqr)

%% Workspace-interior Singularities

qns = qr;
qns(3) = pi;

p560.teach(qns)
Jqnc = p560.jacob0(qns)

det(Jqnc)
```

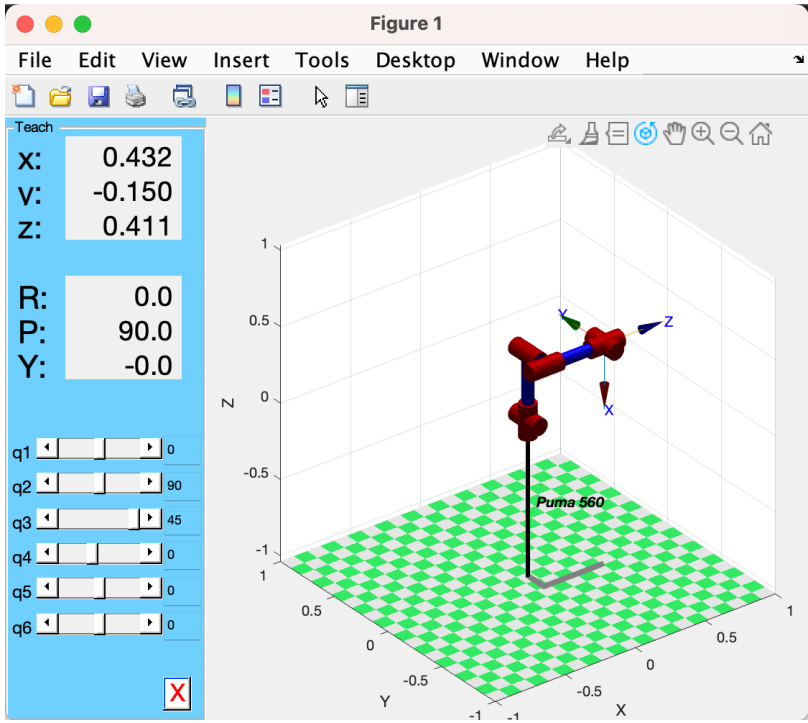
```
Jqr =

    0.1500   -0.8636   -0.4318         0         0         0
    0.0203    0.0000    0.0000         0         0         0
         0    0.0203    0.0203         0         0         0
         0         0         0         0         0         0
         0   -1.0000   -1.0000         0   -1.0000         0
    1.0000    0.0000    0.0000    1.0000    0.0000    1.0000


ans =

     0


ans =

     5

1 linearly dependent joints:
  q6 depends on: q4

Jqnc =

    0.1500   -0.4115    0.0203         0         0         0
    0.4318    0.0000    0.0000         0         0         0
   -0.0000    0.4318    0.4318         0         0         0
         0         0         0    1.0000         0    1.0000
   -0.0000   -1.0000   -1.0000   -0.0000   -1.0000   -0.0000
    1.0000    0.0000    0.0000   -0.0000    0.0000   -0.0000


ans =

     0
```



Singularities - Example 3 - Solution

```
%% Workspace-boundary Singularities
mdl_puma560
p560
p560.teach(qr)

Jqr = p560.jacob0(qr)

det(Jqr)
rank(Jqr)
jsingu(Jqr)

%% Workspace-interior Singularities

qns = qr;
qns(3) = pi;

p560.teach(qns)
Jqnc = p560.jacob0(qns)

det(Jqnc)
rank(Jqnc)
```

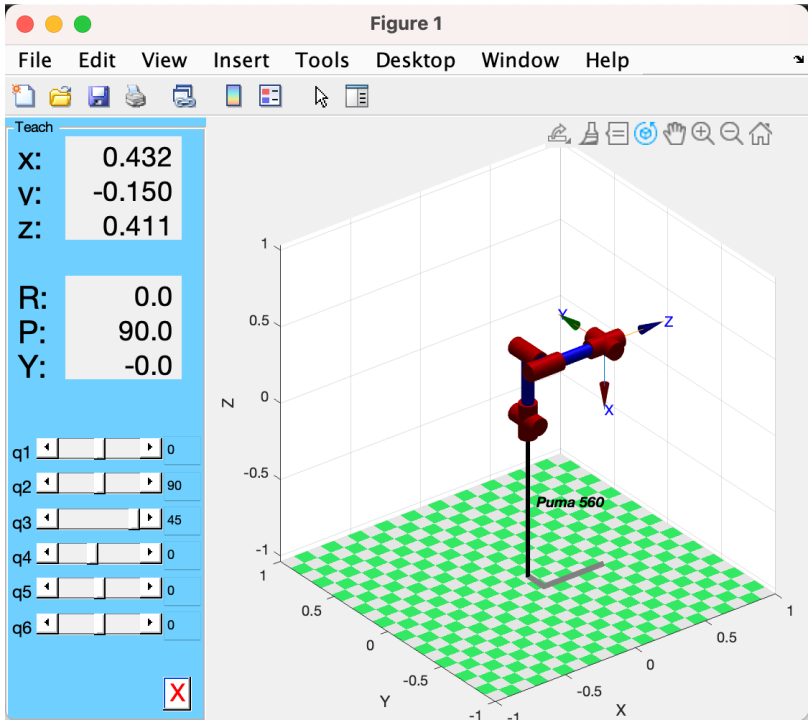
```
Jqr =

    0.1500   -0.8636   -0.4318         0         0         0
    0.0203    0.0000    0.0000         0         0         0
         0    0.0203    0.0203         0         0         0
         0         0         0         0         0         0
         0   -1.0000   -1.0000         0   -1.0000         0
    1.0000    0.0000    0.0000    1.0000    0.0000    1.0000


ans =

     0


ans =

     5

1 linearly dependent joints:
  q6 depends on: q4

Jqnc =

    0.1500   -0.4115    0.0203         0         0         0
    0.4318    0.0000    0.0000         0         0         0
   -0.0000    0.4318    0.4318         0         0         0
         0         0         0    1.0000         0    1.0000
   -0.0000   -1.0000   -1.0000   -0.0000   -1.0000   -0.0000
    1.0000    0.0000    0.0000   -0.0000    0.0000   -0.0000


ans =

     0


ans =

     5
```



Singularities - Example 3 - Solution

```
%% Workspace-boundary Singularities
mdl_puma560
p560
p560.teach(qr)

Jqr = p560.jacob0(qr)

det(Jqr)
rank(Jqr)
jsingu(Jqr)

%% Workspace-interior Singularities

qns = qr;
qns(3) = pi;

p560.teach(qns)
Jqnc = p560.jacob0(qns)

det(Jqnc)
rank(Jqnc)
jsingu(Jqnc)
```

```
Jqr =

    0.1500   -0.8636   -0.4318         0         0         0
    0.0203    0.0000    0.0000         0         0         0
         0    0.0203    0.0203         0         0         0
         0         0         0         0         0         0
         0   -1.0000   -1.0000         0   -1.0000         0
    1.0000    0.0000    0.0000    1.0000    0.0000    1.0000


ans =

     0


ans =

     5

1 linearly dependent joints:
  q6 depends on: q4

Jqnc =

    0.1500   -0.4115    0.0203         0         0         0
    0.4318    0.0000    0.0000         0         0         0
   -0.0000    0.4318    0.4318         0         0         0
         0         0         0    1.0000         0    1.0000
   -0.0000   -1.0000   -1.0000   -0.0000   -1.0000   -0.0000
    1.0000    0.0000    0.0000   -0.0000    0.0000   -0.0000


ans =

     0


ans =

     5

1 linearly dependent joints:
  q6 depends on: q4
```

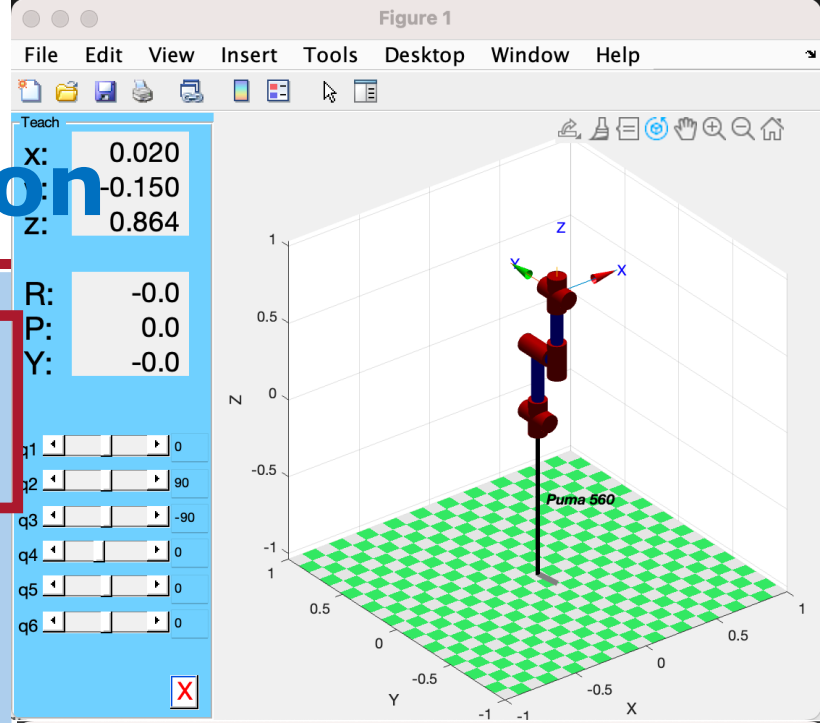


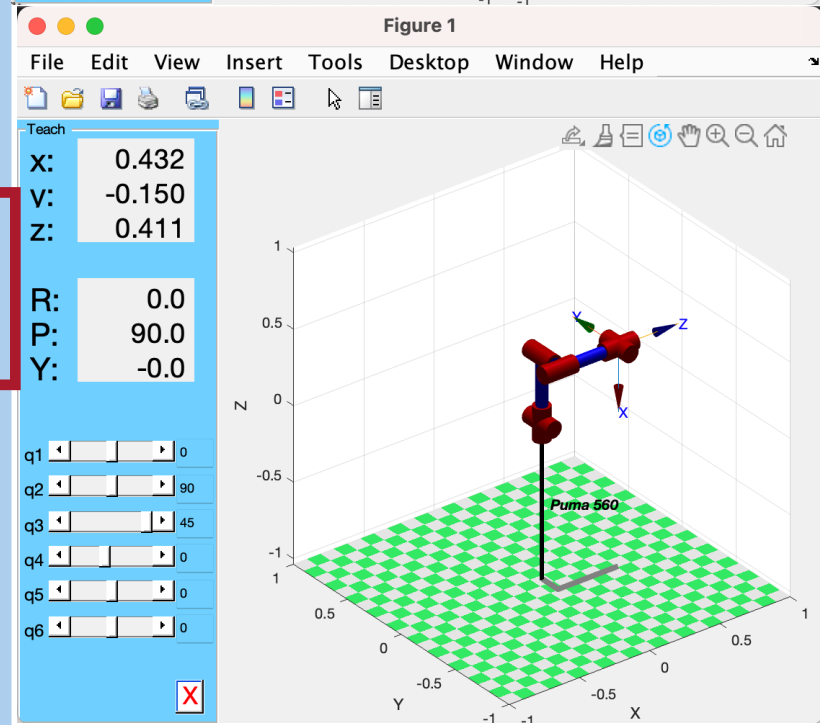
Singularities - Example 3 - Solution

```
%% Workspace-boundary Singularities
mdl_puma560
p560
p560.teach(qr)

Jqr = p560.jacob0(qr)

det(Jqr)
rank(Jqr)
jsingu(Jqr)
```

```
%% Workspace-interior Singularities

qns = qr;
qns(3) = pi;

p560.teach(qns)
Jqnc = p560.jacob0(qns)

det(Jqnc)
rank(Jqnc)
jsingu(Jqnc)
```

```
Jqr =

    0.1500   -0.8636   -0.4318         0         0         0
    0.0203    0.0000    0.0000         0         0         0
         0    0.0203    0.0203         0         0         0
         0         0         0         0         0         0
         0   -1.0000   -1.0000         0   -1.0000         0
    1.0000    0.0000    0.0000    1.0000    0.0000    1.0000

ans =

     0

ans =

     5

1 linearly dependent joints:
  q6 depends on: q4

Jqnc =

    0.1500   -0.4115    0.0203         0         0         0
    0.4318    0.0000    0.0000         0         0         0
   -0.0000    0.4318    0.4318         0         0         0
         0         0         0    1.0000         0    1.0000
   -0.0000   -1.0000   -1.0000   -0.0000   -1.0000   -0.0000
    1.0000    0.0000    0.0000   -0.0000    0.0000   -0.0000

ans =

     0

ans =

     5

1 linearly dependent joints:
  q6 depends on: q4
```





Singularities - Example 3 - Solution

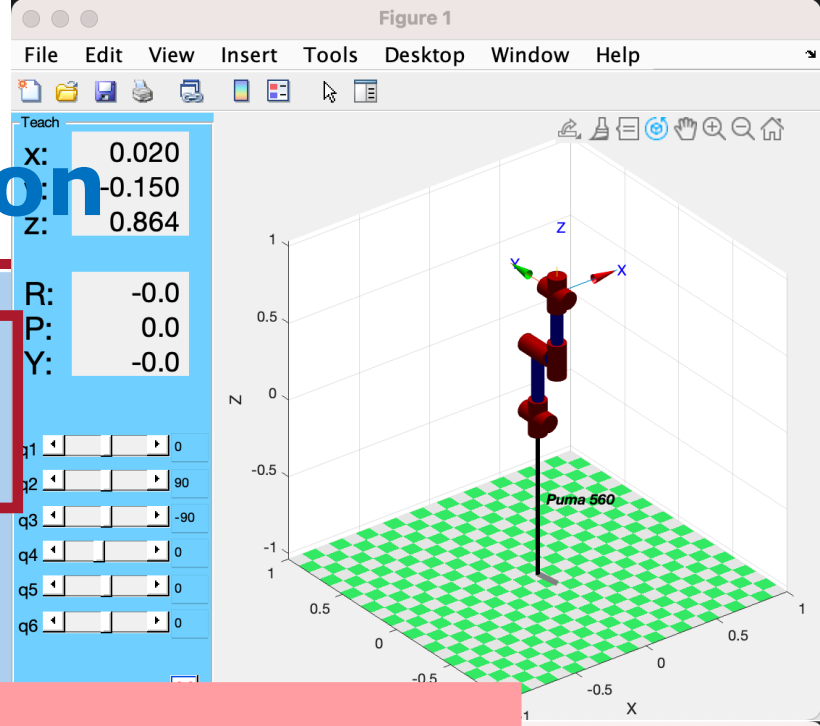
```
%% Workspace-boundary Singularities
mdl_puma560
p560
p560.teach(qr)

Jqr = p560.jacob0(qr)

det(Jqr)
rank(Jqr)
jsingu(Jqr)
```

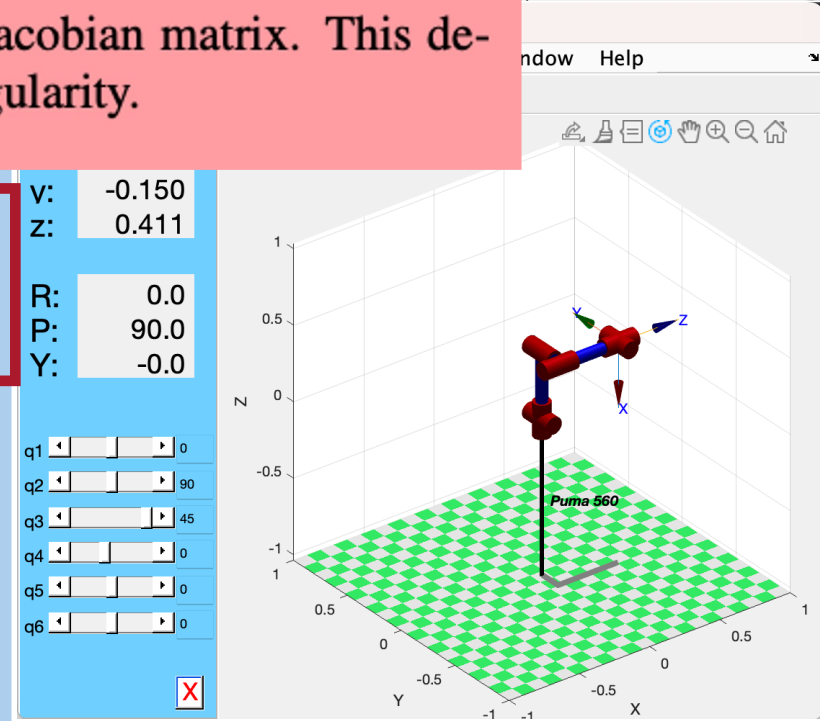
```
%% Workspace-int

qns = qr;
qns(3) = pi;

p560.teach(qns)
Jqnc = p560.jacob0(qns)

det(Jqnc)
rank(Jqnc)
jsingu(Jqnc)
```

```
Jqr =

    0.1500   -0.8636   -0.4318         0         0         0
    0.0203    0.0000    0.0000         0         0         0
         0    0.0203    0.0203         0         0         0
         0         0         0         0         0         0
         0   -1.0000   -1.0000         0   -1.0000         0
    1.0000    0.0000    0.0000    1.0000    0.0000    1.0000

ans =

     0
```

```
    0.1500   -0.4115    0.0203         0         0         0
    0.4318    0.0000    0.0000         0         0         0
   -0.0000    0.4318    0.4318         0         0         0
         0         0         0    1.0000         0    1.0000
   -0.0000   -1.0000   -1.0000   -0.0000   -1.0000   -0.0000
    1.0000    0.0000    0.0000   -0.0000    0.0000   -0.0000

ans =

     0


ans =

     5

1 linearly dependent joints:
  q6 depends on: q4
```

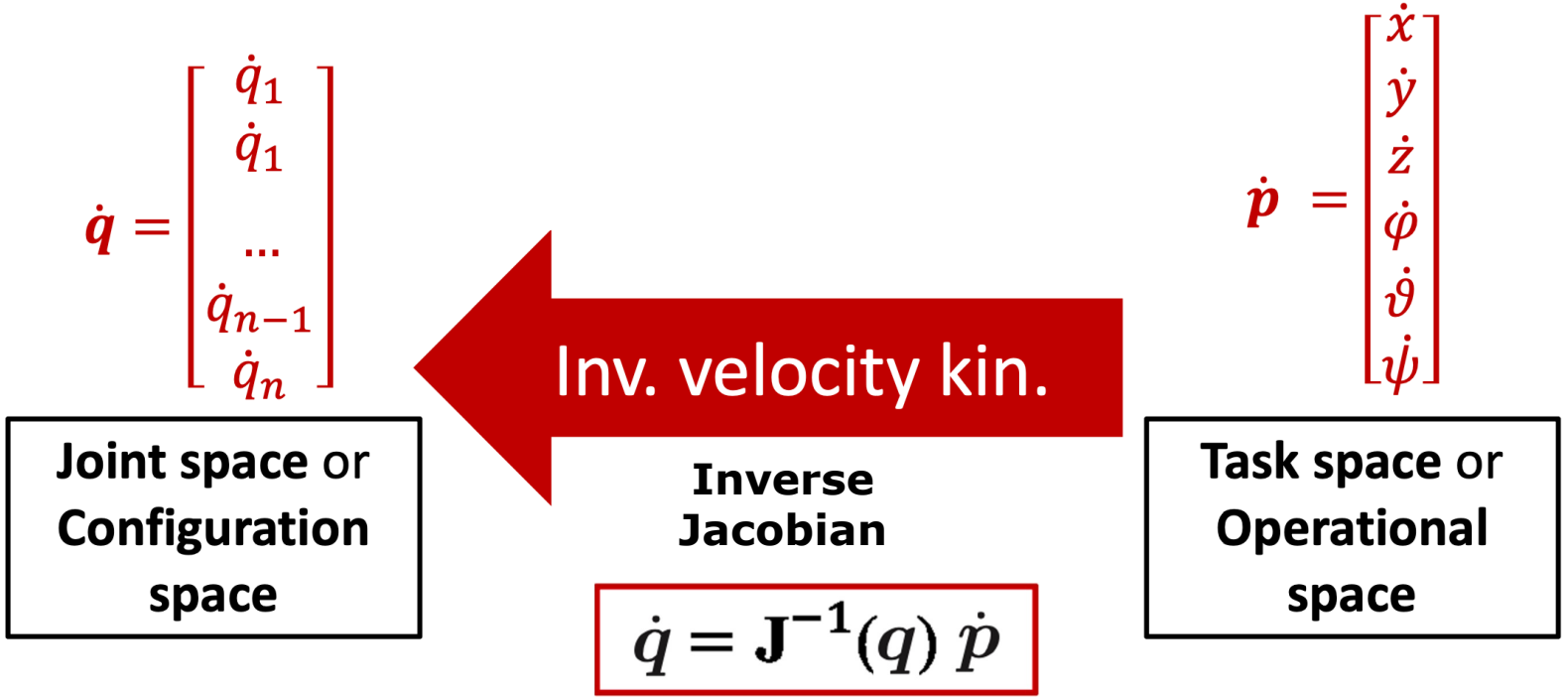
JSINGU(J) displays the linear dependency of joints in a Jacobian matrix. This dependency indicates joint axes that are aligned and causes singularity.





Inverse Velocity Kinematics (IVK)

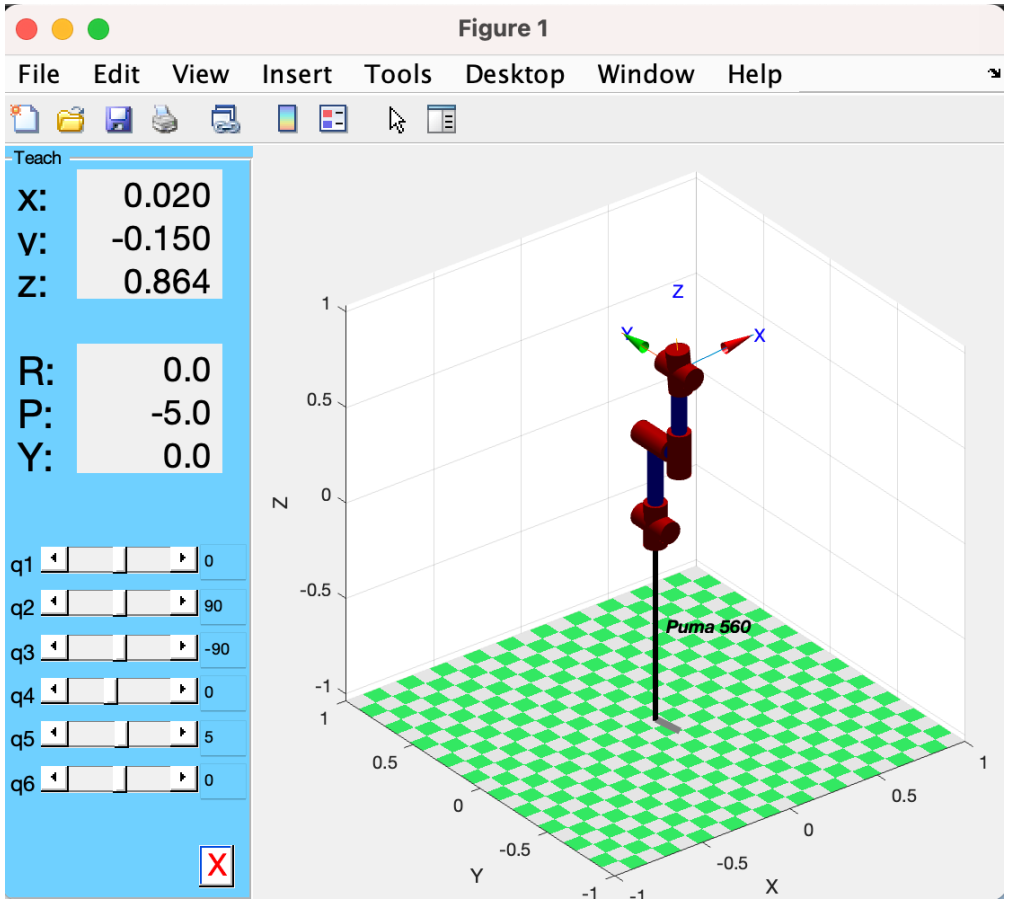
- Given a desired task-space velocities, what **joint velocities** do I need?



Inverse Velocity Kinematics (IVK) - Example 4

- Given relative slow end-effector motion of 0.1 ms^{-1} in the z-direction for the PUMA 560, what are the joint velocities at the *qv* position?

  qv = [0, 1.5708, -1.5708, 0, 0.0873, 0]



IVK - Example 4 - Solution

qv = [0, 1.5708, -1.5708, 0, 0.0873, 0]

```
mdl_puma560
p560
p560.teach(qn)

qv = [0 1.5708 -1.5708 0 0.0873 0];
```

```
p560 =

Puma 560 [Unimation]:: 6 axis, RRRRRR, stdDH, slowRNE
 - viscous friction; params of 8/95;
+---+-----------+-----------+-----------+-----------+-----------+
| j |     theta |         d |         a |     alpha |    offset |
+---+-----------+-----------+-----------+-----------+-----------+
|  1|         q1|          0|          0|     1.5708|          0|
|  2|         q2|          0|     0.4318|          0|          0|
|  3|         q3|    0.15005|     0.0203|    -1.5708|          0|
|  4|         q4|     0.4318|          0|     1.5708|          0|
|  5|         q5|          0|          0|    -1.5708|          0|
|  6|         q6|          0|          0|          0|          0|
+---+-----------+-----------+-----------+-----------+-----------+
```

IVK - Example 4 - Solution

qv = [0, 1.5708, -1.5708, 0, 0.0873, 0]

```
mdl_puma560
p560
p560.teach(qn)

qv = [0 1.5708 -1.5708 0 0.0873 0];

p560.teach(qv)
Jv = p560.jacob0(qv)
det(Jv)
```

```
p560 =

Puma 560 [Unimation]:: 6 axis, RRRRRR, stdDH, slowRNE
 - viscous friction; params of 8/95;
+---+-----------+-----------+-----------+-----------+-----------+
| j |     theta |         d |         a |     alpha |    offset |
+---+-----------+-----------+-----------+-----------+-----------+
|  1|         q1|          0|          0|     1.5708|          0|
|  2|         q2|          0|     0.4318|          0|          0|
|  3|         q3|    0.15005|     0.0203|    -1.5708|          0|
|  4|         q4|     0.4318|          0|     1.5708|          0|
|  5|         q5|          0|          0|    -1.5708|          0|
|  6|         q6|          0|          0|          0|          0|
+---+-----------+-----------+-----------+-----------+-----------+


Jv =

    0.1500   -0.8636   -0.4318         0         0         0
    0.0203    0.0000    0.0000         0         0         0
   -0.0000    0.0203    0.0203         0         0         0
         0    0.0000    0.0000         0    0.0000   -0.0872
   -0.0000   -1.0000   -1.0000   -0.0000   -1.0000   -0.0000
    1.0000    0.0000    0.0000    1.0000    0.0000    0.9962


ans =

  -1.5514e-05
```

IVK - Example 4 - Solution

qv = [0, 1.5708, -1.5708, 0, 0.0873, 0]

```
mdl_puma560
p560
p560.teach(qn)

qv = [0 1.5708 -1.5708 0 0.0873 0];

p560.teach(qv)
Jv = p560.jacob0(qv)
det(Jv)

pv = [0 0 0.1 0 0 0];
```

end-effector motion of 0.1 ms^{-1} in the z-direct

```
p560 =

Puma 560 [Unimation]:: 6 axis, RRRRRR, stdDH, slowRNE
 - viscous friction; params of 8/95;
+---+-----------+-----------+-----------+-----------+-----------+
| j |     theta |         d |         a |     alpha |    offset |
+---+-----------+-----------+-----------+-----------+-----------+
|  1|         q1|          0|          0|     1.5708|          0|
|  2|         q2|          0|     0.4318|          0|          0|
|  3|         q3|    0.15005|     0.0203|    -1.5708|          0|
|  4|         q4|     0.4318|          0|     1.5708|          0|
|  5|         q5|          0|          0|    -1.5708|          0|
|  6|         q6|          0|          0|          0|          0|
+---+-----------+-----------+-----------+-----------+-----------+


Jv =

    0.1500   -0.8636   -0.4318         0         0         0
    0.0203    0.0000    0.0000         0         0         0
   -0.0000    0.0203    0.0203         0         0         0
         0    0.0000    0.0000         0    0.0000   -0.0872
   -0.0000   -1.0000   -1.0000   -0.0000   -1.0000   -0.0000
    1.0000    0.0000    0.0000    1.0000    0.0000    0.9962


ans =

  -1.5514e-05
```

IVK - Example 4 - Solution

qv = [0, 1.5708, -1.5708, 0, 0.0873, 0]

```
mdl_puma560
p560
p560.teach(qn)

qv = [0 1.5708 -1.5708 0 0.0873 0];

p560.teach(qv)
Jv = p560.jacob0(qv)
det(Jv)

pv = [0 0 0.1 0 0 0];    end-effector motion of 0.1 ms-1 in the z-direction
qv_dot = pinv(Jv)*pv';
qv_dot'
```

$$\dot{q} = \mathbf{J}^{-1}(q)\,\dot{p}$$

```
p560 =

Puma 560 [Unimation]:: 6 axis, RRRRRR, stdDH, slowRNE
 - viscous friction; params of 8/95;
+---+----------+----------+----------+----------+----------+
| j |     theta |         d |         a |     alpha |    offset |
+---+----------+----------+----------+----------+----------+
|  1|         q1|          0|          0|     1.5708|          0|
|  2|         q2|          0|     0.4318|          0|          0|
|  3|         q3|    0.15005|     0.0203|    -1.5708|          0|
|  4|         q4|     0.4318|          0|     1.5708|          0|
|  5|         q5|          0|          0|    -1.5708|          0|
|  6|         q6|          0|          0|          0|          0|
+---+----------+----------+----------+----------+----------+


Jv =

    0.1500   -0.8636   -0.4318         0         0         0
    0.0203    0.0000    0.0000         0         0         0
   -0.0000    0.0203    0.0203         0         0         0
         0    0.0000    0.0000         0    0.0000   -0.0872
   -0.0000   -1.0000   -1.0000   -0.0000   -1.0000   -0.0000
    1.0000    0.0000    0.0000    1.0000    0.0000    0.9962


ans =

  -1.5514e-05


ans =

    0.0000   -4.9257    9.8514   -0.0000   -4.9257    0.0000
```

… end of Lecture 9