Jacobians: Velocity Relationships

Table of Contents

Introduction

In the last chapter we discussed the relationship between joint coordinates and end-effector pose – the manipulator kinematics . Now we investigate the relationship between the rate of change of these quantities – between joint velocity and velocity of the end-effector . This is called the velocity or differential kinematics of the manipulator.

Small angles and displacements

$$\cos(\delta)=1;\sin(\delta)=\delta$$

$$Rot(X,\delta_x)\approx\begin{bmatrix}1&0&0&0\\0&1&-\delta_x&0\\0&\delta_x&1&0\\0&0&0&1\end{bmatrix};Rot(Y,\delta_y)\approx\begin{bmatrix}1&0&\delta_y&0\\0&1&0&0\\-\delta_y&0&1&0\\0&0&0&1\end{bmatrix};Rot(Z,\delta_z)\approx\begin{bmatrix}1&-\delta_z&0&0\\\delta_z&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix};Trans(dx,dy,dz)=\begin{bmatrix}1&0&0&dx\\0&1&0&dy\\0&0&1&dz\\0&0&0&1\end{bmatrix}$$

Motion due to a robot’s joints moving during Δt

$$T+\Delta T = R_{\vec{K}}(\theta)transl\left(d_x,d_y,d_z\right)T;T\ original\ frame$$

$$T+\Delta T\approx\begin{bmatrix}1&-\delta_z&\delta_y&d_x\\\delta_z&1&-\delta_x&d_y\\-\delta_y&\delta_x&1&d_z\\0&0&0&1\end{bmatrix}T=T\begin{bmatrix}1&-\delta_z&\delta_y&d_x\\\delta_z&1&-\delta_x&d_y\\-\delta_y&\delta_x&1&d_z\\0&0&0&1\end{bmatrix}$$

```
close all
bdclose('all')
clear
```

```
clc
```

Differential motion

```
syms delta_x delta_y delta_z dx dy dz
Rx=[1 0 0 0;0 1 -delta_x 0;0 delta_x 1 0; 0 0 0 1]
```

```
Rx =
```

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & -\delta_x & 0 \\ 0 & \delta_x & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

```
Ry=[1 0 delta_y 0;0 1 0 0; -delta_y 0 1 0;0 0 0 1]
```

```
Ry =
```

$$\begin{pmatrix} 1 & 0 & \delta_y & 0 \\ 0 & 1 & 0 & 0 \\ -\delta_y & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

```
Rz=[1 -delta_z 0 0; delta_z 1 0 0;0 0 1 0; 0 0 0 1]
```

```
Rz =
```

$$\begin{pmatrix} 1 & -\delta_z & 0 & 0 \\ \delta_z & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

```
T=[1 0 0 dx;0 1 0 dy; 0 0 1 dz; 0 0 0 1]
```

```
T =
```

$$\begin{pmatrix} 1 & 0 & 0 & \mathrm{dx} \\ 0 & 1 & 0 & \mathrm{dy} \\ 0 & 0 & 1 & \mathrm{dz} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

```
T_dT=T*Rx*Ry*Rz
```

```
T_dT =
```

$$\begin{pmatrix} 1 & -\delta_z & \delta_y & \mathrm{dx} \\ \delta_z + \delta_x \delta_y & 1 - \delta_x \delta_y \delta_z & -\delta_x & \mathrm{dy} \\ \delta_x \delta_z - \delta_y & \delta_x + \delta_y \delta_z & 1 & \mathrm{dz} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

```
T_dT=Rx*Ry*Rz*T
```

```
T_dT =
```

$$\begin{pmatrix} 1 & -\delta_z & \delta_y & \mathrm{dx} - \delta_z\,\mathrm{dy} + \delta_y\,\mathrm{dz} \\ \delta_z + \delta_x\,\delta_y & 1 - \delta_x\,\delta_y\,\delta_z & -\delta_x & \mathrm{dx}\;(\delta_z + \delta_x\,\delta_y) - \delta_x\,\mathrm{dz} - \mathrm{dy}\;(\delta_x\,\delta_y\,\delta_z - 1) \\ \delta_x\,\delta_z - \delta_y & \delta_x + \delta_y\,\delta_z & 1 & \mathrm{dz} - \mathrm{dx}\;(\delta_y - \delta_x\,\delta_z) + \mathrm{dy}\;(\delta_x + \delta_y\,\delta_z) \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

In both cases if $\delta_{\mathrm{xyz}} \& d_{\mathrm{xyz}} \approx 0$ –

$$T + \Delta T \approx \begin{bmatrix} 1 & -\delta_z & \delta_y & d_x \\ \delta_z & 1 & -\delta_x & d_y \\ -\delta_y & \delta_x & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} T$$

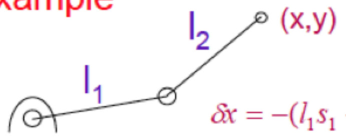
Twolink. Jacobian

$$\frac{\delta p}{\delta \theta} = J(\theta) \to \delta p = J(\theta)\delta\theta \to \frac{\delta p}{\delta t} = J(\theta)\frac{\delta\theta}{\delta t} \to \dot{p} = J(\theta)\dot{\theta} \to {}^{0}\upsilon = {}^{0}J(\theta)\dot{\theta}$$

The Jacobian computed by the method jacob0 maps joint velocity to the endeffector spatial velocity expressed in the world coordinate frame.

We illustrate the basics with 2-dimensional example using a TwoLink manipulator

Example



$$x = l_1 c_1 + l_2 c_{12}$$
$$y = l_1 s_1 + l_2 s_{12}$$
$$\delta x = -(l_1 s_1 + l_2 s_{12})\delta\theta_1 - l_2 s_{12}\delta\theta_2$$
$$\delta y = (l_1 c_1 + l_2 c_{12})\delta\theta_1 + l_2 c_{12}\delta\theta_2$$
$$\delta X = \begin{bmatrix} \delta x \\ \delta y \end{bmatrix} = \begin{bmatrix} -y & -l_2 s_{12} \\ x & l_2 c_{12} \end{bmatrix} \begin{pmatrix} \delta\theta_1 \\ \delta\theta_2 \end{pmatrix}$$

$$\boxed{\delta x = J(\theta)\delta\theta}$$
$$\dot{x} = J(\theta)\dot{\theta}$$

$$J \equiv \begin{pmatrix} \dfrac{\partial x}{\partial\theta_1} & \dfrac{\partial x}{\partial\theta_2} \\ \dfrac{\partial y}{\partial\theta_1} & \dfrac{\partial y}{\partial\theta_2} \end{pmatrix} = \begin{bmatrix} -y & -l_2 s_{12} \\ x & l_2 c_{12} \end{bmatrix}$$

The Jacobian J computed and maps joint velocity to the end effector spatial velocity expressed in the world coordinate frame.

```
mdl_planar2_sym
p2
```

```
p2 =
two link:: 2 axis, RR, stdDH, slowRNE, Symbolic
+---+-----------+-----------+-----------+-----------+-----------+
| j |     theta |         d |         a |     alpha |    offset |
+---+-----------+-----------+-----------+-----------+-----------+
|  1|         q1|          0|         a1|          0|          0|
|  2|         q2|          0|         a2|          0|          0|
+---+-----------+-----------+-----------+-----------+-----------+
```

```
syms theta_1 theta_2 real

TE = p2.fkine( [theta_1 theta_2] );

p = TE.t; p = p(1:2)
```

p =

$$\begin{pmatrix} a_2\cos(\theta_1+\theta_2)+a_1\cos(\theta_1) \\ a_2\sin(\theta_1+\theta_2)+a_1\sin(\theta_1) \end{pmatrix}$$

```
J = jacobian(p, [theta_1 theta_2]);
J=simplify(J)
```

J =

$$\begin{pmatrix} -a_2\sin(\theta_1+\theta_2)-a_1\sin(\theta_1) & -a_2\sin(\theta_1+\theta_2) \\ a_2\cos(\theta_1+\theta_2)+a_1\cos(\theta_1) & a_2\cos(\theta_1+\theta_2) \end{pmatrix}$$

Remember

```
p
```

p =

$$\begin{pmatrix} a_2\cos(\theta_1+\theta_2)+a_1\cos(\theta_1) \\ a_2\sin(\theta_1+\theta_2)+a_1\sin(\theta_1) \end{pmatrix}$$

$${}^0J_{EE}(\theta) = \begin{bmatrix} -y & -L_2 s_{12} \\ x & L_2 c_{12} \end{bmatrix}$$

Changing Jacobian’s Reference Frames

$${}^AJ(\theta) = \begin{bmatrix} {}^A_BR & 0_{3x3} \\ 0_{3x3} & {}^A_BR \end{bmatrix} {}^BJ(\theta) \rightarrow {}^BJ(\theta) = \begin{bmatrix} {}^A_BR & 0_{3x3} \\ 0_{3x3} & {}^A_BR \end{bmatrix}^{-1} {}^AJ(\theta)$$

$${}^1J(\theta) = {}^0_1R^{T\,0}J(\theta) = \begin{bmatrix} -L_2 s_2 & -L_2 s_2 \\ L_1 + L_2 c_2 & L_2 c_2 \end{bmatrix}$$

```
J_1=rot2(-theta_1)*J
```

```
J_1 =
```

$$\begin{pmatrix} \sin(\theta_1)\,\sigma_2 - \cos(\theta_1)\,\sigma_1 & a_2\cos(\theta_1+\theta_2)\sin(\theta_1) - a_2\sin(\theta_1+\theta_2)\cos(\theta_1) \\ \cos(\theta_1)\,\sigma_2 + \sin(\theta_1)\,\sigma_1 & a_2\cos(\theta_1+\theta_2)\cos(\theta_1) + a_2\sin(\theta_1+\theta_2)\sin(\theta_1) \end{pmatrix}$$

where

$$\sigma_1 = a_2\sin(\theta_1+\theta_2) + a_1\sin(\theta_1)$$

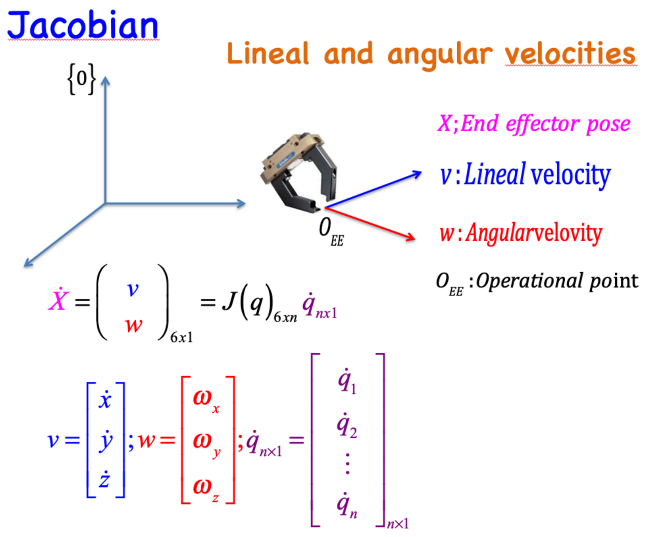
$$\sigma_2 = a_2\cos(\theta_1+\theta_2) + a_1\cos(\theta_1)$$

```
J_1=simplify(J_1)
```

```
J_1 =
```

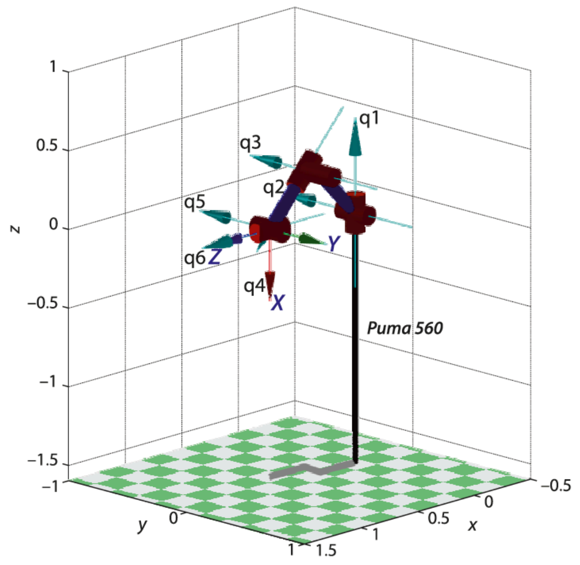
$$\begin{pmatrix} -a_2\sin(\theta_2) & -a_2\sin(\theta_2) \\ a_1 + a_2\cos(\theta_2) & a_2\cos(\theta_2) \end{pmatrix}$$

6R Robot. Jacobian


Jacobian
Lineal and angular velocities
{0}
X;End effector pose
v:Lineal velocity
w:Angularvelovity
0_{EE}
0_{EE}:Operational point
$\dot{X} = \begin{pmatrix} v \\ w \end{pmatrix}_{6x1} = J(q)_{6xn}\,\dot{q}_{nx1}$
$v = \begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{z}\end{bmatrix}; w = \begin{bmatrix}\omega_x\\ \omega_y\\ \omega_z\end{bmatrix}; \dot{q}_{n\times1} = \begin{bmatrix}\dot{q}_1\\ \dot{q}_2\\ \vdots\\ \dot{q}_n\end{bmatrix}_{n\times1}$

$^{0}v = \begin{bmatrix}\dot{x} & \dot{y} & \dot{z} & w_x & w_y & w_z\end{bmatrix}^T \in \Re^6$ and $^{0}J(\theta) \in \Re^{6\times N}$

For a realistic 3-dimensional robot this Jacobian matrix can be numerically computed by the 'jacob0' method of the SerialLink object, based on its Denavit-Hartenberg parameters. For the Puma robot in the pose shown in the nex figure,



The Jacobian is

```
mdl_puma560
J = p560.jacob0(qn)
```

```
J = 6×6
    0.1501    0.0144    0.3197         0         0         0
    0.5963    0.0000    0.0000         0         0         0
    0.0000    0.5963    0.2910         0         0         0
         0   -0.0000   -0.0000    0.7071   -0.0000    1.0000
    0.0000   -1.0000   -1.0000   -0.0000   -1.0000   -0.0000
    1.0000    0.0000    0.0000   -0.7071    0.0000   -0.0000
```

Making Jacobian by hand

```
T=p560.fkine(qn)
```

```
T = 
         0         0         1    0.5963
         0         1         0   -0.1501
        -1         0         0  -0.01435
         0         0         0         1
```

```
T_1=p560.fkine(qn+[0.000001 0 0 0 0 0])% small change in theta_1
```

```
T_1 = 
         0   -0.0000    1.0000    0.5963
         0    1.0000    0.0000     -0.15
        -1         0         0  -0.01435
         0         0         0         1
```

```
J13=(T_1.transl-T.transl)/0.000001 % must be the slope = (J(1:3,1))
```

```
J13 = 1×3
    0.1500    0.5963         0
```

Jacobian in end-effector

To obtain the spatial velocity in the end-effector coordinate frame we introduce the velocity transformation from the world frame to the end-effector frame which is a function of the end-effector pose

$$^{0}\upsilon = \begin{bmatrix} ^{0}v \\ ^{0}w \end{bmatrix} = \underbrace{^{0}J(\theta)}\dot{\theta}; the\ resulting^{0}\upsilon\ will\ be\ in\{0\}$$

$$^{B}\upsilon = \begin{bmatrix} ^{B}v \\ ^{B}w \end{bmatrix} = \underbrace{^{B}J(\theta)}\dot{\theta}; the\ resulting^{B}\upsilon\ will\ be\ in\{B\}$$

$$^{A}J(\theta) = \begin{bmatrix} ^{A}_{B}R & 0_{3x3} \\ 0_{3x3} & ^{A}_{B}R \end{bmatrix} {}^{B}J(\theta) \rightarrow {}^{B}J(\theta) = \begin{bmatrix} ^{A}_{B}R & 0_{3x3} \\ 0_{3x3} & ^{A}_{B}R \end{bmatrix}^{-1} {}^{A}J(\theta)$$

In the Toolbox this Jacobian is computed by the method jacobe and for the Puma robot at the pose used above is:

```
Je=p560.jacobe(qn)
T=p560.fkine(qn)
Je_hand=inv([T.R zeros(3);zeros(3) T.R])*J
```

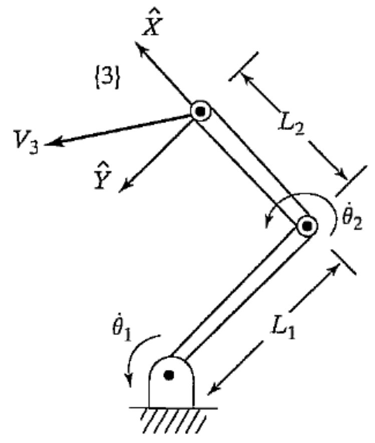
Singularities

We have discussed how the Jacobian matrix maps joint rates to end-effector Cartesian velocity but the inverse problem has strong practical use – what joint velocities are needed to achieve a required end-effector Cartesian velocity?

$$\dot{\theta} = J^{-1}(\theta)\dot{X};\ a\ problem\ occurs\ if\ \det(J(\theta)) = 0$$

Where are the singularities of the simple two-link arm? What is the

physical explanation of the singularities?

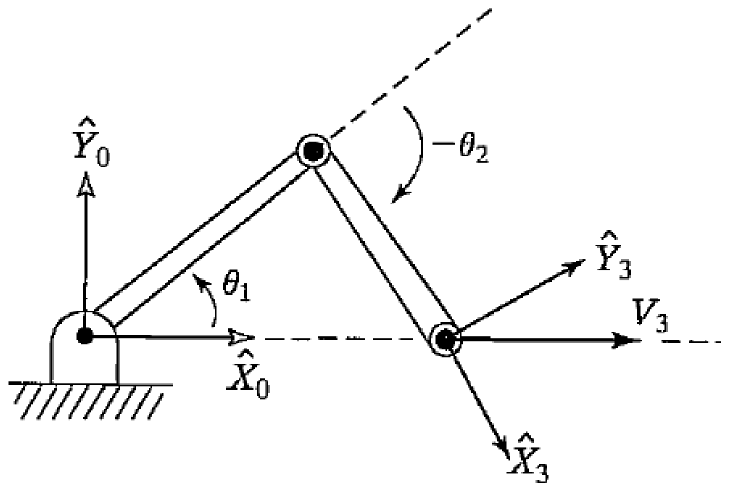


$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} L_1c_1 + L_2c_{12} \\ L_1s_1 + L_2s_{12} \end{pmatrix}; {}^0J(\theta) = \begin{pmatrix} -L_1s_1 - L_2s_{12} & -L_2s_{12} \\ L_1c_1 + L_2c_{12} & L_2c_{12} \end{pmatrix}_{c_{12}=c_1c_2-s_1s_2; s_{12}=c_1s_2+s_1c_2}$$

A solution exist if ${}^0J^{-1}(\theta)$ is posible, so a singularity exist when: $\det\left[{}^0J(\theta)\right] = L_1L_2s_2 = 0$;

Clearly, a singularity of the mechanism exists when θ_2 is 0 or 180 degrees. Physically, when $\theta_2 = 0$, the arm is stretched straight out. In this configuration, motion of the end-effector is possible along only one Cartesian direction (the one perpendicular to the arm). Therefore, the mechanism has lost one degree of freedom. Likewise, when $\theta_2 = 180$, the arm is folded completely back on itself, and motion of the hand again is possible only in one Cartesian direction instead of two.

Consider the two-link robot is moving its end-effector along the X axis at 1.0 m/s. Show that joint rates are reasonable when far from a singularity, but that, as a singularity is approached at $\theta_2 = 0$, joint rates tend to infinity.



We start by calculating the inverse of the Jacobian written in {0}:

$$ {}^{0}J^{-1}(\theta) = \frac{1}{L_1 L_2 s_2} \begin{bmatrix} L_2 c_{12} & L_2 s_{12} \\ -L_1 c_1 - L_2 c_{12} & -L_1 s_1 - L_2 s_{12} \end{bmatrix} $$

$$ \begin{pmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{pmatrix} = J^{-1}(\theta) \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} \dfrac{c_{12}}{L_1 s_2} \\ \dfrac{-L_1 c_1 - L_2 c_{12}}{L_1 L_2 s_2} \end{pmatrix} = \begin{pmatrix} \dfrac{c_{12}}{L_1 s_2} \\ -\dfrac{c_1}{L_2 s_2} - \dfrac{c_{12}}{L_1 s_2} \end{pmatrix} $$

Clearly, as the arm stretches out toward $\theta_2 = 0$, both joint rates go to infinity.

Trajectory with singularity

```
clear
mdl_twolink
twolink.plot(qz)
P1=[0.1  0  0]
```

```
P1 = 1×3
    0.1000         0         0
```

```
P2=[1.99  0  0]
```

```
P2 = 1×3
    1.9900         0         0
```

```
T1=transl(P1)
```

```
T1 = 4×4
    1.0000         0         0    0.1000
         0    1.0000         0         0
         0         0    1.0000         0
         0         0         0    1.0000
```

```
T2=transl(P2)
```

```
T2 = 4×4
    1.0000         0         0    1.9900
         0    1.0000         0         0
         0         0    1.0000         0
         0         0         0    1.0000
```

```
qu=twolink.ikine(T1,'mask',[1  1  0  0  0  0 ], 'q0',[pi/4  0])
```

```
qu = 1×2
    1.5208   -3.0416
```

```
twolink.plot(qu,'zoom',2)
hold on
line([P1(1)  P2(1)],[P1(2)  P2(2)], [P1(3)  P2(3)])
```

```
ts=0.05
```

```
ts = 0.0500
```

```
tf=2
```

```
tf = 2
```

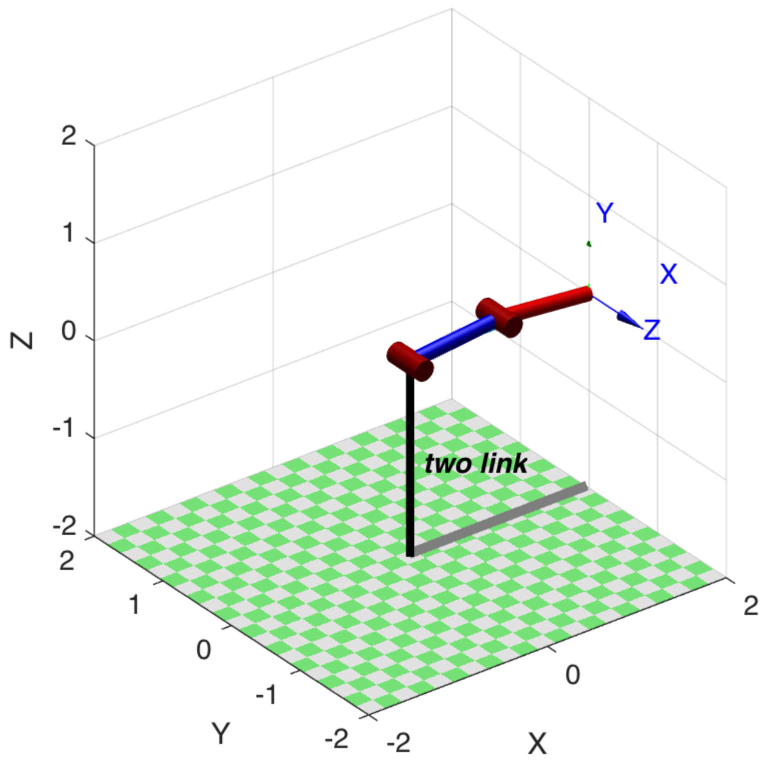
```
t=(0:ts:tf)'
```

```
t = 41×1
         0
    0.0500
    0.1000
    0.1500
    0.2000
    0.2500
    0.3000
    0.3500
    0.4000
    0.4500
      ⋮
```

```
Trajec=ctraj(T1,T2,40);
qs_u=twolink.ikine(Trajec,'mask',[1 1 0 0 0 0 ], 'q0',[pi/2 0])
```

```
qs_u = 40×2
    1.5208   -3.0416
    1.5194   -3.0388
    1.5152   -3.0304
    1.5082   -3.0163
    1.4984   -2.9967
    1.4857   -2.9715
    1.4703   -2.9406
    1.4520   -2.9040
    1.4309   -2.8617
    1.4068   -2.8137
      ⋮
```

```
twolink.plot(qs_u, 'zoom',2, 'workspace',[-0.5 3 -0.5 0.5 -0.5 2])
```



Joint velocities

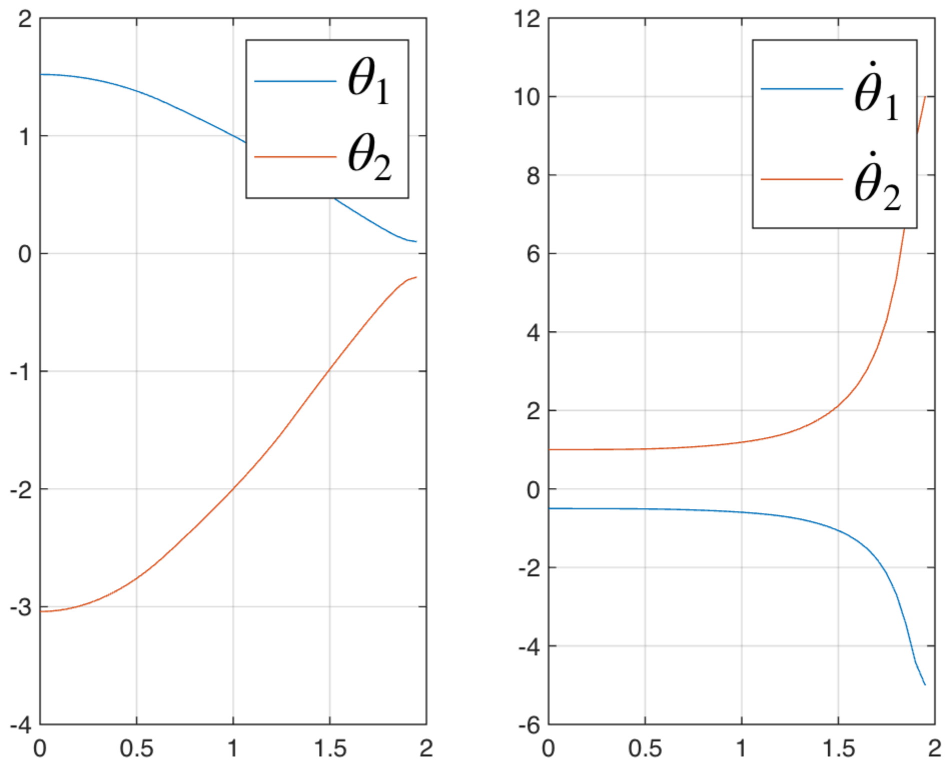
See function below

```
[q1_d,q2_d] = th_dot(qs_u(:,1),qs_u(:,2))
```

```
q1_d = 1×40
   -0.5006   -0.5007   -0.5008   -0.5010   -0.5013   -0.5018   -0.5025   -0.5035 ⋯
q2_d = 1×40
    1.0013    1.0013    1.0015    1.0020    1.0026    1.0036    1.0051    1.0071 ⋯
```

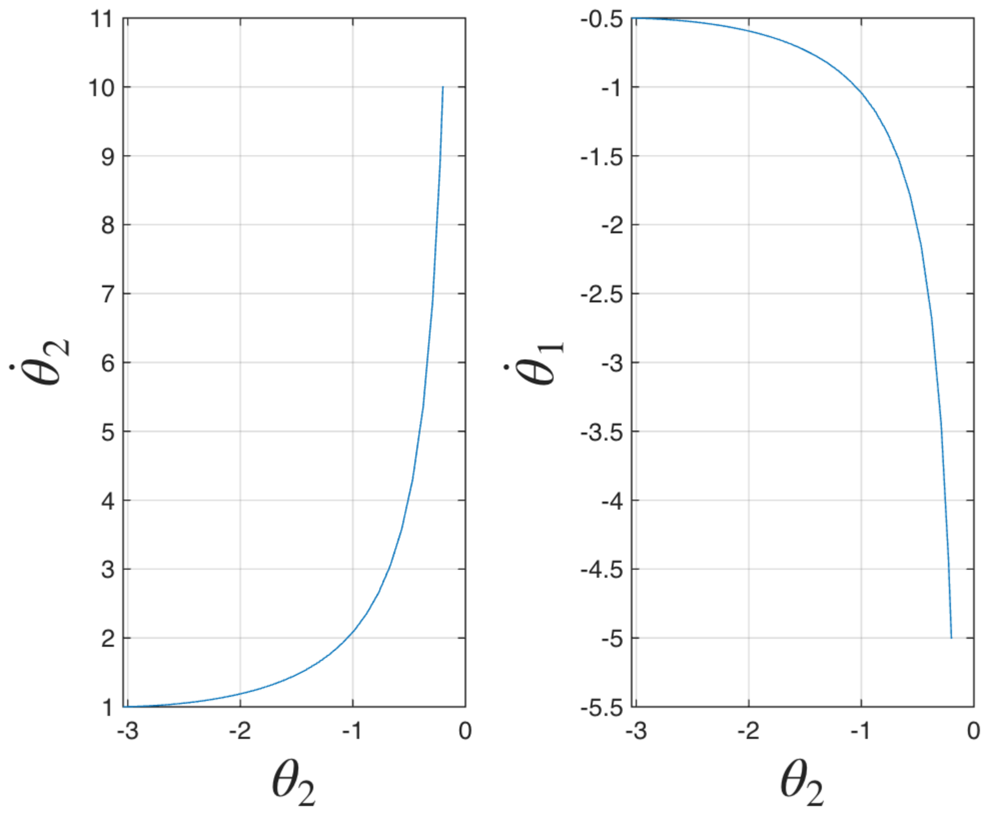
Plotting results: Time vs theta_dot's

```
figure
subplot(121)
plot(t(1:40),qs_u(:,1),t(1:40), qs_u(:,2))
legend ('$\theta_1$','$\theta_2$','Interpreter','latex','Fontsize',20)
grid on
subplot(122)
plot(t(1:40),q1_d,t(1:40), q2_d)
legend('${{\dot \theta }_1}$','${{\dot \theta }_2}$','Interpreter','latex','Fontsize',2
grid on
```



Plotting results: Theta_2 vs theta_dot's

```
figure
subplot(121)
plot(qs_u(:,2), q2_d)
xlabel ('$\theta_2$','Interpreter','latex','Fontsize',20)
ylabel('${{\dot \theta }_2}$','Interpreter','latex','Fontsize',20)
grid on
subplot(122)
plot(qs_u(:,2), q1_d)
xlabel ('$\theta_2$','Interpreter','latex','Fontsize',20)
ylabel('${{\dot \theta }_1}$','Interpreter','latex','Fontsize',20)
grid on
```



Manipulability-Twolink

Sphere equation

```
syms x y z real
[x y z]*[x y z]'
```

```
ans = 
```
$x^2 + y^2 + z^2$

Consider the set of generalized joint velocities with a unit norm which lie on the surface of a hypersphere in the N-dimensional joint velocity space.

$$\dot{\Theta} = J^{-1}(\Theta)\dot{X}$$

$$\dot{\Theta}^T\dot{\Theta} = 1$$

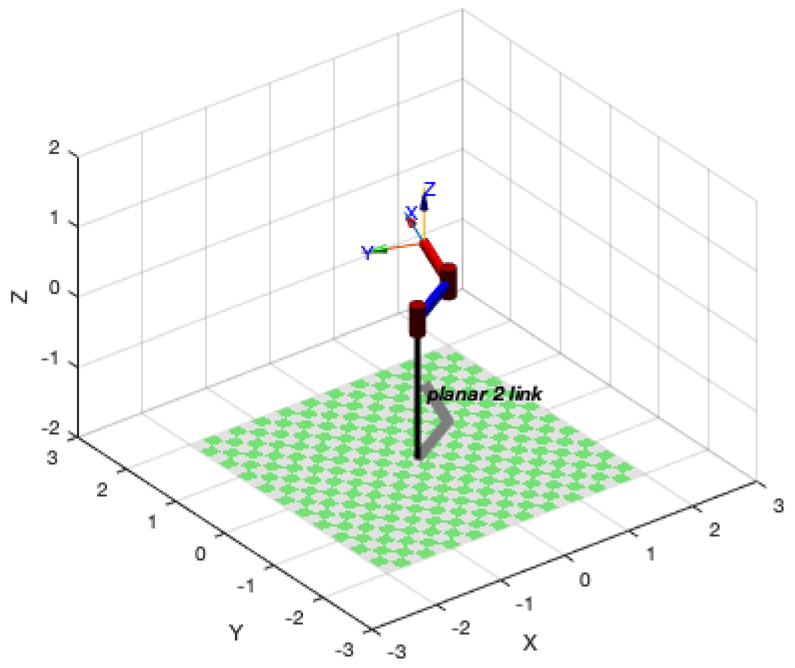
$$\dot{\Theta}^T\dot{\Theta} = 1 \rightarrow \left(J^{-1}(\Theta)\dot{X}\right)^T J^{-1}(\Theta)\dot{X} = \dot{X}^T\left(J(\Theta)J^T(\Theta)\right)^{-1} = 1$$

Which is the equation of points on the surface of an ellipsoid within the dim T-dimensional end-effector velocity space. If this ellipsoid is close to spherical, that is, its radii are of the same order of magnitude then all is well – the end-effector can achieve arbitrary Cartesian velocity. However if one or more radii are very small this indicates that the endeffector cannot achieve velocity in the directions corresponding to those small radii.

```
clear
clc
clf
mdl_planar2
q=[30*pi/180 40*pi/180]
```

```
q = 1×2
    0.5236    0.6981
```

```
p2.plot(q)
axis([-3 3 -3 3 -2 2])
hold on
```



Plotting the ellipses

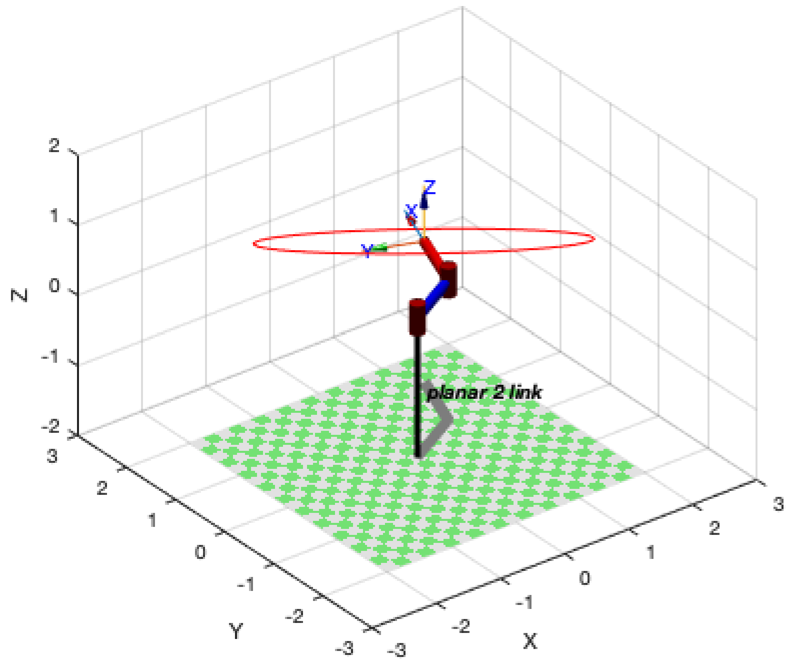
Case 1

```
q=[30*pi/180 40*pi/180]
```

```
q = 1×2
```

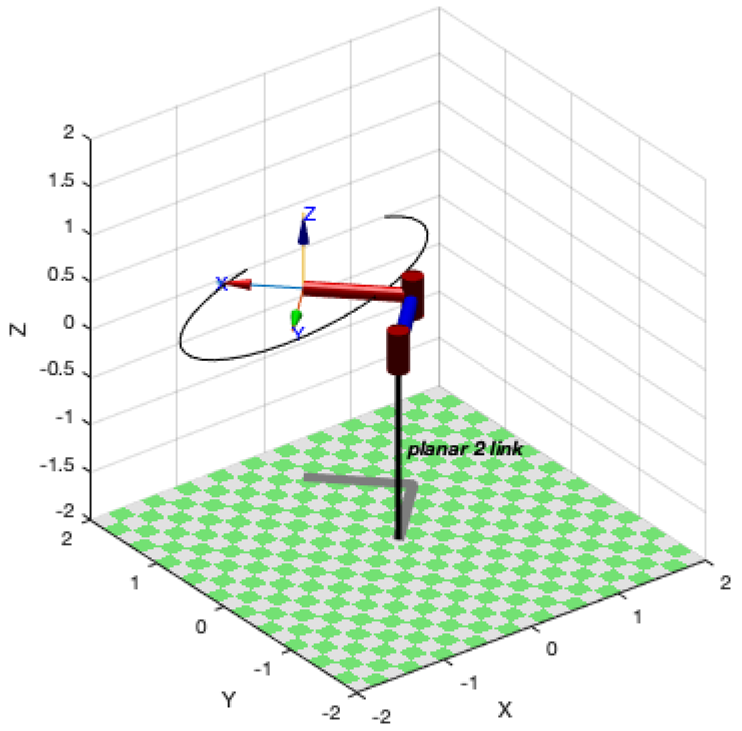
```
    0.5236    0.6981
```

```
J = p2.jacob0(q);
J = J(1:2,1:2);
N = (J*J');
t = transl(p2.fkine(q));
plot_ellipse(N, t(1:2))
p2.vellipse(q,'2d')
```



Case 2

```
q=[45*pi/180 90*pi/180];
J = p2.jacob0(q);
J = J(1:2,1:2);
N = (J*J');
clf
axis([-3 3 -3 3 -2 2])
p2.plot(q)
hold on
t= transl(p2.fkine(q));
plot_ellipse(N, t(1:2))
```

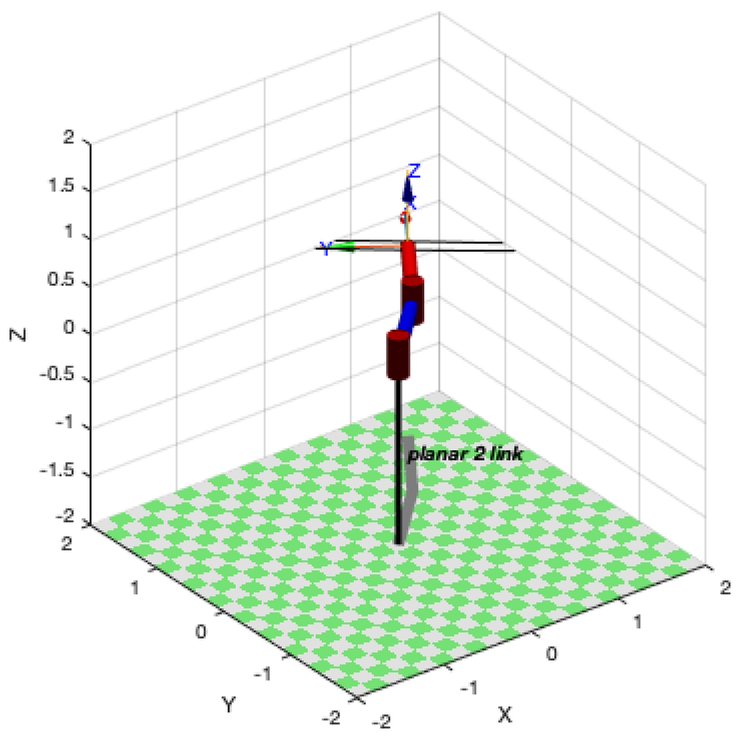


Case near singularity

```
q=[45*pi/180 10*pi/180]
```

```
q = 1×2
    0.7854    0.1745
```

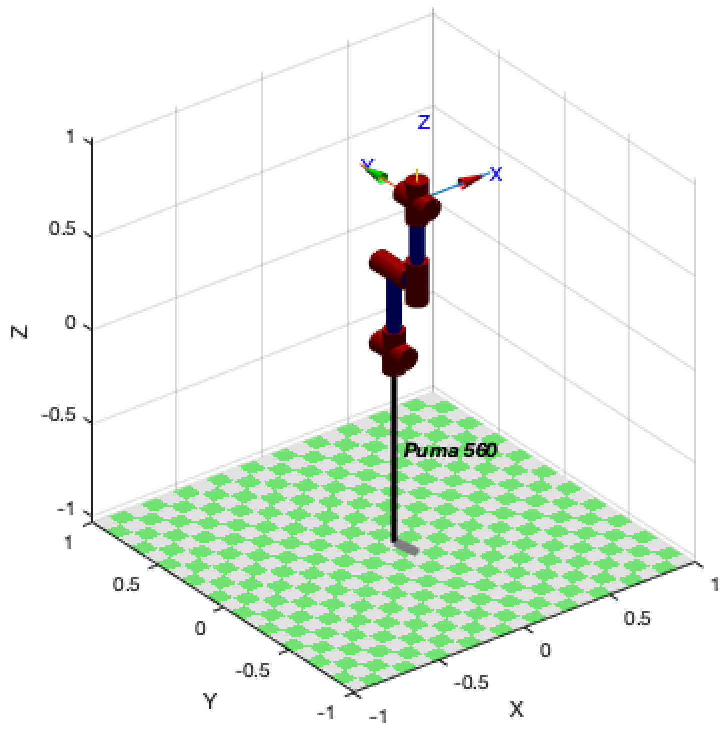
```
J = p2.jacob0(q);
J = J(1:2,1:2);
N = (J*J');
clf
axis([-3 3 -3 3 -2 2])
p2.plot(q)
hold on
t= transl(p2.fkine(q));
plot_ellipse(N, t(1:2))
```



Playing with the Puma

Jacobian for the ready pose

```
clear
mdl_puma560
p560.plot(qr)
```



```
J = p560.jacob0(qr)% see the dimension
```

```
J = 6×6
    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
```

```
Det_te=det(J)
```

```
Det_te = 0
```

```
Rango=rank(J)
```

```
Rango = 5
```

```
jsingu(J)
```

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

```
qd = (inv(J)*[0 0 0.1 0 0 0]')'
```

```
Warning: Matrix is singular to working precision.
qd = 1×6
   NaN   NaN   NaN   NaN   NaN   NaN
```

Jacobian near a singularity

```
qns = qr; qns(5) = 5*pi/180
```

```
qns = 1×6
         0    1.5708   -1.5708         0    0.0873         0
```

```
J=p560.jacob0(qns)
```

```
J = 6×6
    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
    0.0000         0         0    0.0000         0   -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
```

```
Det_te=det(J)
```

```
Det_te = -1.5509e-05
```

```
Rango=rank(J)
```

```
Rango = 6
```

```
qd_t = (inv(J)*[0 0 0.1 0 0 0]')'
```

```
qd_t = 1×6
   -0.0000   -4.9261    9.8522    0.0000   -4.9261         0
```

```
qd_r = (inv(J)*[0 0 0 0 0.2 0]')'
```

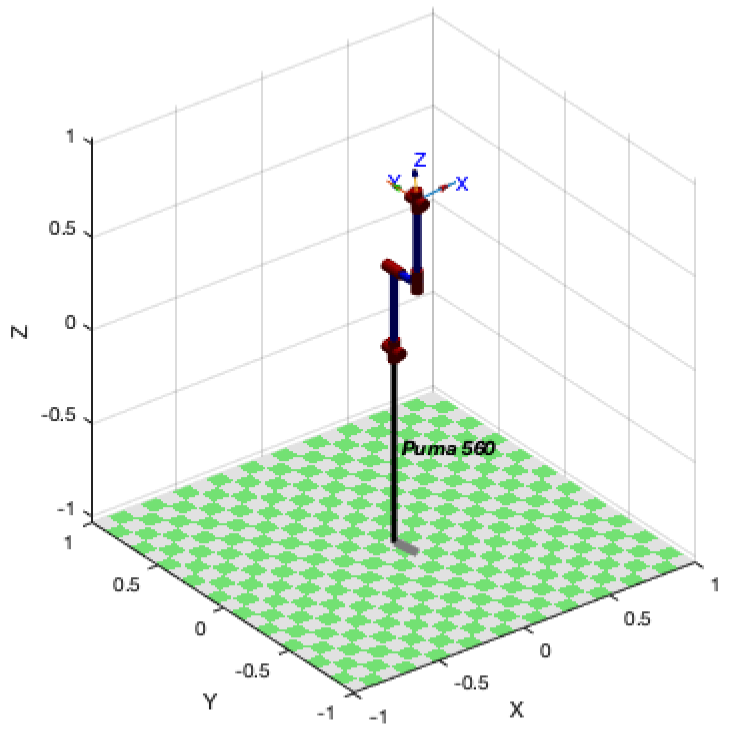
```
qd_r = 1×6
    0.0000    0.0000    0.0000    0.0000   -0.2000    0.0000
```

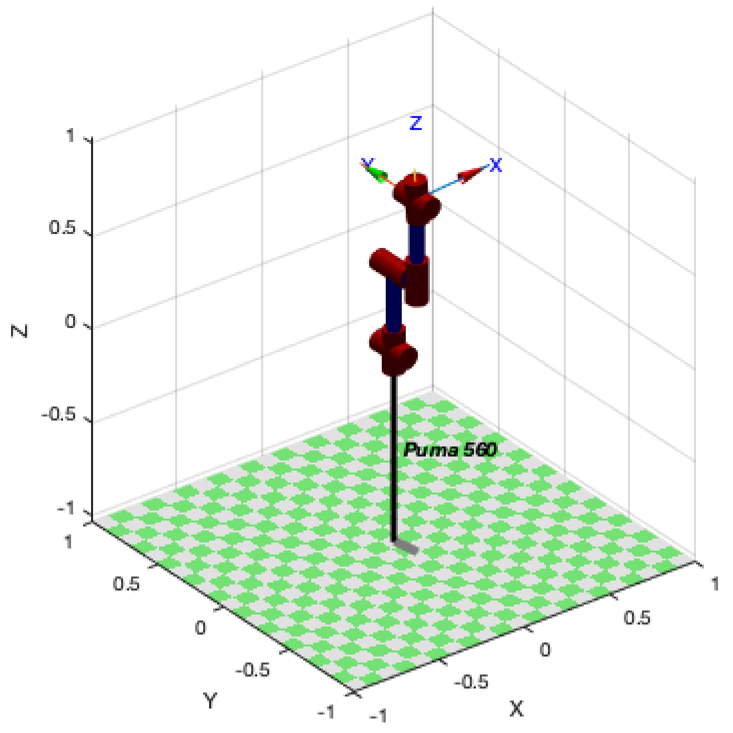
Plotting ellipses

```
mdl_puma560
qns = qr;
qns(5) = 5 * pi/180
```

```
qns = 1×6
         0    1.5708   -1.5708         0    0.0873         0
```

```
figure
p560.plot(qns,'zoom',2)
```





Translational ellipse

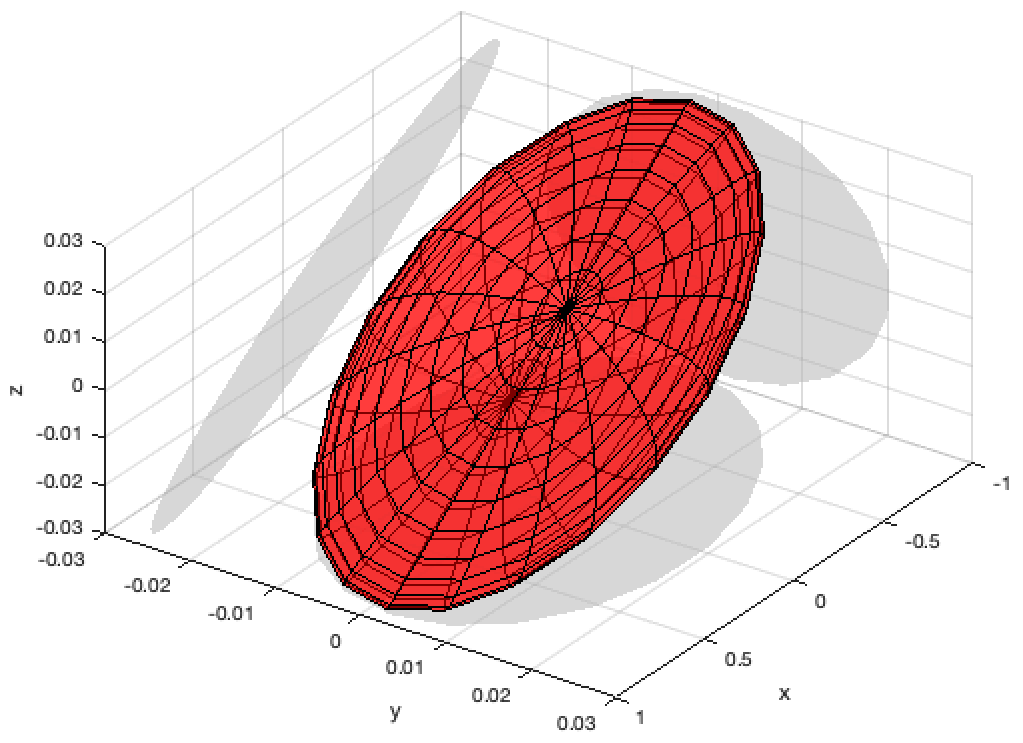
```
J0= p560.jacob0(qns)
```

```
J0 = 6×6
    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
    0.0000         0         0    0.0000         0   -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
```

```
J = J0(1:3,:)
```

```
J = 3×6
    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
```

```
figure
plot_ellipse(J*J','fillcolor','r','alpha',0.5,'shadow');
view (125,45)
%axis([-1 1 -1 1 -1 1])
xyzlabel
grid on
```



Rotational ellipse

```
J0= p560.jacob0(qns)
```

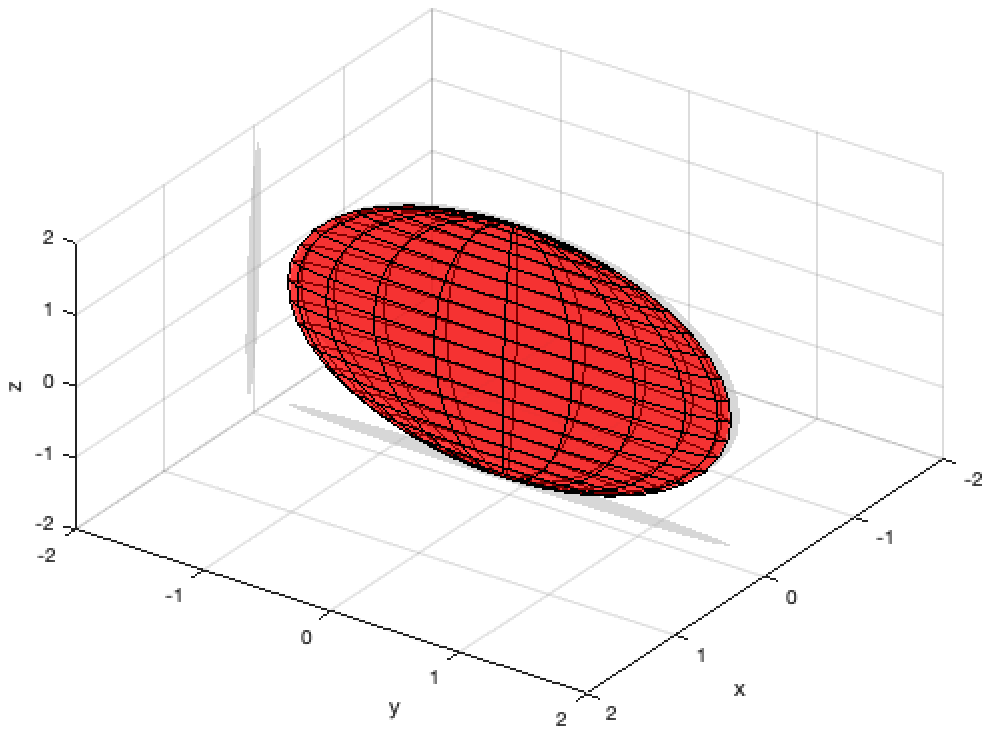
```
J0 = 6×6
```

```
     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
     0.0000         0         0    0.0000         0   -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
```

```
J = J0(4:6,:)
```

```
J = 3×6
     0.0000         0         0    0.0000         0   -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
```

```
figure
plot_ellipse(J*J','fillcolor','r','alpha',0.5,'shadow');
view (125,45)
axis([-2 2 -2 2 -2 2])
xyzlabel
grid on
```



Manipulability

$m = \sqrt{\det(JJ^T)}$;Yoshikawa's manipulability measure

```
m = p560.maniplty(qr)
```

```
m = 0
```

```
mn=p560.maniplty(qn)
```

```
mn = 0.0786
```

Functions

$$\begin{pmatrix}\dot{\theta}_1\\ \dot{\theta}_2\end{pmatrix} = J^{-1}(\theta)\begin{pmatrix}1\\0\end{pmatrix} = \begin{pmatrix}\dfrac{c_{12}}{L_1 s_2}\\ \dfrac{-L_1c_1 - L_2c_{12}}{L_1L_2s_2}\end{pmatrix} = \begin{pmatrix}\dfrac{c_{12}}{L_1 s_2}\\ -\dfrac{c_1}{L_2s_2} - \dfrac{c_{12}}{L_1s_2}\end{pmatrix}$$

$L_1 = L_2 = 1$

```
function [q1_d,q2_d] = th_dot(th_1,th_2)
for i=1:length(th_1)
c12=cos(th_1(i)+th_2(i));
s2=sin(th_2(i));
c1=cos(th_1(i));
q1_d(i)=c12/s2;
q2_d(i)=-(c1+c12)/s2;
end
end
```