Table of Contents

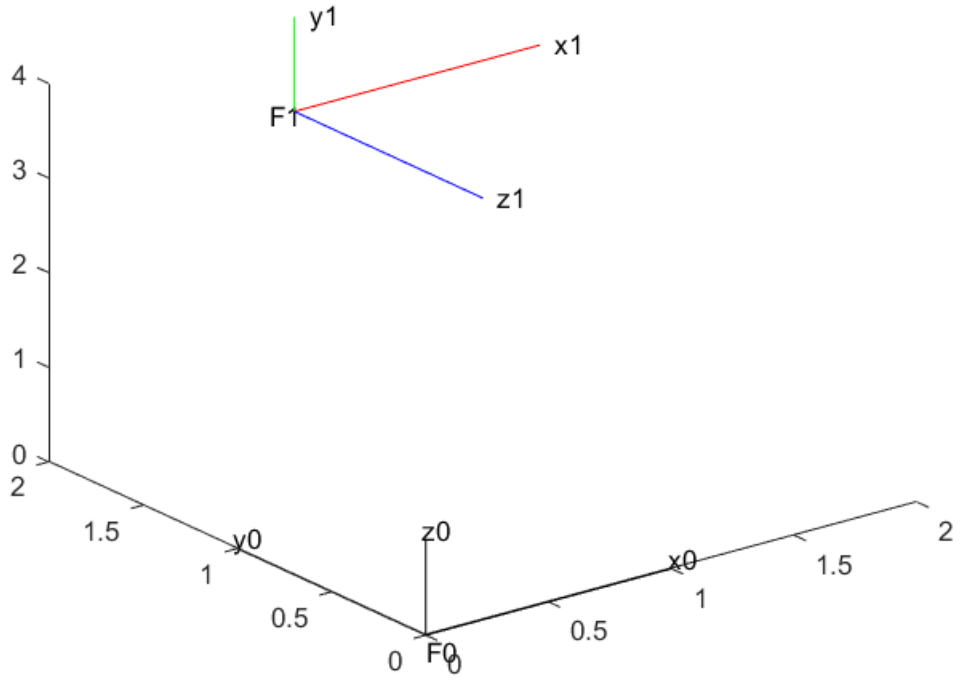

```
clear all; close all; clc
```

H_Trans

```
clc

%Create Homogeneous Transformations from fundamental rotations/
translations
H=H_Trans.rotZ(sym(pi/2))
H.Trans=[1;2;3];

%Access and modify sections of the transformation
Matrix=H.H
Rotation=H.Rot
Translation=H.Trans

disp('Get Euler Angles');
Euler=H.Euler
disp('Set Euler Angles');
H.Euler = [pi/2,0,0];
eval(H.Rot)
H.Euler

syms theta d a alpha

disp('Create From DH Parameters');
DH=H_Trans.fromDH([theta,d,a,alpha]);
DH.H

disp('Fast Inverse');
DH.inv.H

H_Trans().draw(1,'0')
hold on
H.draw(1,'1')
hold off
```

```
H =

  H_Trans with properties:

         H: [4x4 sym]
     Trans: [3x1 sym]
       Rot: [3x3 sym]
     Euler: [3x1 sym]
    Wrench: [6x1 sym]
    Column: [16x1 sym]


Matrix =

[ 0, -1, 0, 1]
[ 1,  0, 0, 2]
[ 0,  0, 1, 3]
[ 0,  0, 0, 1]


Rotation =

[ 0, -1, 0]
[ 1,  0, 0]
[ 0,  0, 1]


Translation =

 1
 2
 3

Get Euler Angles

Euler =

    0
    0
 pi/2

Set Euler Angles

ans =

    1.0000         0         0
         0    0.0000   -1.0000
         0    1.0000    0.0000


ans =

    1.5708         0         0
```

```
Create From DH Parameters

ans =

[ cos(theta), -1.0*cos(alpha)*sin(theta),      sin(alpha)*sin(theta),
 a*cos(theta)]
[ sin(theta),      cos(alpha)*cos(theta), -1.0*sin(alpha)*cos(theta),
 a*sin(theta)]
[          0,                 sin(alpha),                 cos(alpha),
           d]
[          0,                          0,                          0,
         1.0]

Fast Inverse

ans =

[                cos(theta),                 sin(theta),          0,
 - a*sin(theta)^2 - a*cos(theta)^2]
[ -1.0*cos(alpha)*sin(theta),      cos(alpha)*cos(theta), sin(alpha),
                    -d*sin(alpha)]
[      sin(alpha)*sin(theta), -1.0*sin(alpha)*cos(theta), cos(alpha),
                    -d*cos(alpha)]
[                         0,                          0,          0,
                             1]
```



Kin_Model

```
close all; clc;

q=sym('q',[3,1],'real');

disp('Create From DH Parameters');
kin=Kin_Model.fromDH([q(1),1,0,-pi/2;
                      q(2),0,1,0;
                      q(3),0,1,0],q);

disp('Joint Transformations')
kin.T(0,2).H
kin.T(3,1).H

disp('Forward/Inverse Kinematics')

test1=kin.forward_kin([0;0;0]);
test2=kin.forward_kin([pi/4;pi/3;3*pi/4]);
test3=kin.forward_kin([1;1;1]);

kin.inverse_kin(test1)
kin.inverse_kin(test2)
kin.inverse_kin(test3)

test4=[NaN,NaN,NaN,1;
       NaN,NaN,NaN,1;
       NaN,NaN,NaN,1;
       NaN,NaN,NaN,NaN];

result4 = kin.inverse_kin(test4)
kin.forward_kin(result4)

disp('Jacobian')
kin.J
kin.J([0;0;0])

disp('Inverse Jacobian')
kin.inv_J
```

```
Create From DH Parameters
Joint Transformations

ans =

[ cos(q1)*cos(q2), -1.0*cos(q1)*sin(q2), -1.0*sin(q1),
 cos(q1)*cos(q2)]
[ cos(q2)*sin(q1), -1.0*sin(q1)*sin(q2),  1.0*cos(q1),
 cos(q2)*sin(q1)]
[    -1.0*sin(q2),         -1.0*cos(q2),            0, 1.0 -
 1.0*sin(q2)]
[               0,                    0,            0,
 1.0]
```

```
ans =

[       cos(q2)*cos(q3) - 1.0*sin(q2)*sin(q3),     cos(q2)*sin(q3)
 + cos(q3)*sin(q2),   0,       - (cos(q2)*sin(q3) +
 cos(q3)*sin(q2))*(1.0*sin(q2) + cos(q2)*sin(q3) + cos(q3)*sin(q2))
 - (cos(q2)*cos(q3) - 1.0*sin(q2)*sin(q3))*(1.0*cos(q2) +
 cos(q2)*cos(q3) - 1.0*sin(q2)*sin(q3))]
[ - 1.0*cos(q2)*sin(q3) - 1.0*cos(q3)*sin(q2), cos(q2)*cos(q3)
 - 1.0*sin(q2)*sin(q3),   0, (1.0*cos(q2)*sin(q3) +
 1.0*cos(q3)*sin(q2))*(1.0*cos(q2) + cos(q2)*cos(q3)
 - 1.0*sin(q2)*sin(q3)) - (cos(q2)*cos(q3) -
 1.0*sin(q2)*sin(q3))*(1.0*sin(q2) + cos(q2)*sin(q3) +
 cos(q3)*sin(q2))]
[                                          0,
              0, 1.0,


   0]
[                                          0,
              0,   0,


   1]

Forward/Inverse Kinematics

ans =

     0
     0
     0


ans =

    0.7854
    1.0472
   -3.9270


ans =

    1.0000
    1.0000
    1.0000


result4 =

    0.7854
    0.7854
   -1.5708
```

```
ans =

    0.5000    0.5000   -0.7071    1.0000
    0.5000    0.5000    0.7071    1.0000
    0.7071   -0.7071         0    1.0000
         0         0         0    1.0000

Jacobian

ans =

[ -sin(q1)*(cos(q2 + q3) + cos(q2)), -cos(q1)*(sin(q2 + q3) +
 sin(q2)), -sin(q2 + q3)*cos(q1)]
[  cos(q1)*(cos(q2 + q3) + cos(q2)), -sin(q1)*(sin(q2 + q3) +
 sin(q2)), -sin(q2 + q3)*sin(q1)]
[                                 0,          - cos(q2 + q3) -
 cos(q2),         -cos(q2 + q3)]
[                                 0,                          -
sin(q1),              -sin(q1)]
[                                 0,
 cos(q1),               cos(q1)]
[                                 1,
 0,                     0]


ans =

     0     0     0
     2     0     0
     0    -2    -1
     0     0     0
     0     1     1
     1     0     0

Inverse Jacobian

ans =

[


                    -(cos(q2)*sin(q1) + cos(q2 + q3)*sin(q1))/(cos(q2
 + q3)^2*cos(q1)^2 + cos(q2 + q3)^2*sin(q1)^2 + cos(q1)^2*cos(q2)^2
 + cos(q2)^2*sin(q1)^2 + 2*cos(q2 + q3)*cos(q1)^2*cos(q2) + 2*cos(q2
 + q3)*cos(q2)*sin(q1)^2 + 1),
```

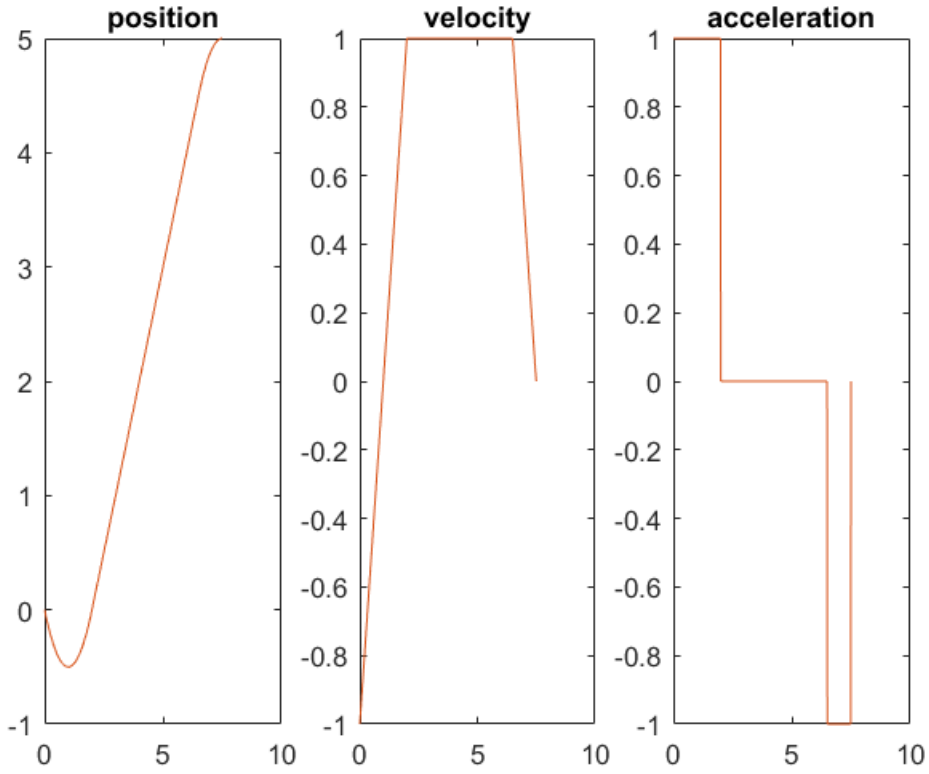
```
    (cos(q1)*(cos(q2 + q3) + cos(q2)))/(cos(q2 + q3)^2*cos(q1)^2 +
 cos(q2 + q3)^2*sin(q1)^2 + cos(q1)^2*cos(q2)^2 + cos(q2)^2*sin(q1)^2
 + 2*cos(q2 + q3)*cos(q1)^2*cos(q2) + 2*cos(q2 + q3)*cos(q2)*sin(q1)^2
 + 1),

                                                          0,

                                                                0,

                                                                  0, 1/(cos(q2
 + q3)^2*cos(q1)^2 + cos(q2 + q3)^2*sin(q1)^2 + cos(q1)^2*cos(q2)^2 +
 cos(q2)^2*sin(q1)^2 + 2*cos(q2 + q3)*cos(q1)^2*cos(q2) + 2*cos(q2 +
 q3)*cos(q2)*sin(q1)^2 + 1)]
[
                      -(cos(q1)^3*sin(q2) + cos(q2 +
 q3)^2*cos(q1)*sin(q2) + cos(q1)*sin(q1)^2*sin(q2) - cos(q2
 + q3)*sin(q2 + q3)*cos(q1)*cos(q2))/(cos(q1)^2*cos(q2)^2 +
 cos(q2)^2*sin(q1)^2 + cos(q1)^4*sin(q2)^2 + sin(q1)^4*sin(q2)^2 +
 2*cos(q1)^2*sin(q1)^2*sin(q2)^2 + cos(q2 + q3)^2*cos(q1)^2*sin(q2)^2
 + sin(q2 + q3)^2*cos(q1)^2*cos(q2)^2 + cos(q2 +
 q3)^2*sin(q1)^2*sin(q2)^2 + sin(q2 + q3)^2*cos(q2)^2*sin(q1)^2 -
 2*cos(q2 + q3)*sin(q2 + q3)*cos(q1)^2*cos(q2)*sin(q2) - 2*cos(q2 +
 q3)*sin(q2 + q3)*cos(q2)*sin(q1)^2*sin(q2)),
                                                    -(sin(q1)^3*sin(q2)
 + cos(q2 + q3)^2*sin(q1)*sin(q2) + cos(q1)^2*sin(q1)*sin(q2) -
 cos(q2 + q3)*sin(q2 + q3)*cos(q2)*sin(q1))/(cos(q1)^2*cos(q2)^2 +
 cos(q2)^2*sin(q1)^2 + cos(q1)^4*sin(q2)^2 + sin(q1)^4*sin(q2)^2 +
 2*cos(q1)^2*sin(q1)^2*sin(q2)^2 + cos(q2 + q3)^2*cos(q1)^2*sin(q2)^2
 + sin(q2 + q3)^2*cos(q1)^2*cos(q2)^2 + cos(q2 +
 q3)^2*sin(q1)^2*sin(q2)^2 + sin(q2 + q3)^2*cos(q2)^2*sin(q1)^2 -
 2*cos(q2 + q3)*sin(q2 + q3)*cos(q1)^2*cos(q2)*sin(q2) - 2*cos(q2
 + q3)*sin(q2 + q3)*cos(q2)*sin(q1)^2*sin(q2)),
```

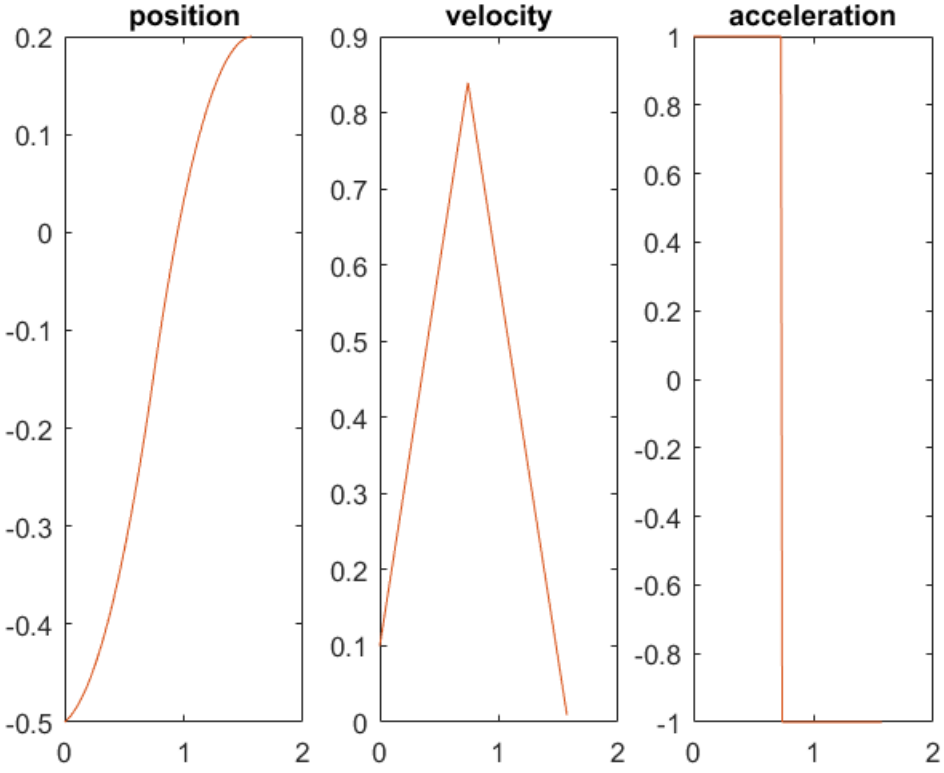
```
             -(cos(q1)^2*cos(q2) + cos(q2)*sin(q1)^2 + sin(q2
+ q3)^2*cos(q1)^2*cos(q2) + sin(q2 + q3)^2*cos(q2)*sin(q1)^2
- cos(q2 + q3)*sin(q2 + q3)*cos(q1)^2*sin(q2) - cos(q2 +
q3)*sin(q2 + q3)*sin(q1)^2*sin(q2))/(cos(q1)^2*cos(q2)^2 +
cos(q2)^2*sin(q1)^2 + cos(q1)^4*sin(q2)^2 + sin(q1)^4*sin(q2)^2 +
2*cos(q1)^2*sin(q1)^2*sin(q2)^2 + cos(q2 + q3)^2*cos(q1)^2*sin(q2)^2
+ sin(q2 + q3)^2*cos(q1)^2*cos(q2)^2 + cos(q2 +
q3)^2*sin(q1)^2*sin(q2)^2 + sin(q2 + q3)^2*cos(q2)^2*sin(q1)^2 -
2*cos(q2 + q3)*sin(q2 + q3)*cos(q1)^2*cos(q2)*sin(q2) - 2*cos(q2
+ q3)*sin(q2 + q3)*cos(q2)*sin(q1)^2*sin(q2)),
                                                    (sin(q2
+ q3)*sin(q2)*cos(q1)^2*sin(q1) + sin(q2 + q3)*sin(q2)*sin(q1)^3
+ cos(q2 + q3)*cos(q2)*sin(q1))/(cos(q1)^2*cos(q2)^2 +
cos(q2)^2*sin(q1)^2 + cos(q1)^4*sin(q2)^2 + sin(q1)^4*sin(q2)^2 +
2*cos(q1)^2*sin(q1)^2*sin(q2)^2 + cos(q2 + q3)^2*cos(q1)^2*sin(q2)^2
+ sin(q2 + q3)^2*cos(q1)^2*cos(q2)^2 + cos(q2 +
q3)^2*sin(q1)^2*sin(q2)^2 + sin(q2 + q3)^2*cos(q2)^2*sin(q1)^2 -
2*cos(q2 + q3)*sin(q2 + q3)*cos(q1)^2*cos(q2)*sin(q2) - 2*cos(q2
+ q3)*sin(q2 + q3)*cos(q2)*sin(q1)^2*sin(q2)),
                                                   -(sin(q2
+ q3)*sin(q2)*cos(q1)^3 + sin(q2 + q3)*sin(q2)*cos(q1)*sin(q1)^2
+ cos(q2 + q3)*cos(q2)*cos(q1))/(cos(q1)^2*cos(q2)^2 +
cos(q2)^2*sin(q1)^2 + cos(q1)^4*sin(q2)^2 + sin(q1)^4*sin(q2)^2 +
2*cos(q1)^2*sin(q1)^2*sin(q2)^2 + cos(q2 + q3)^2*cos(q1)^2*sin(q2)^2
+ sin(q2 + q3)^2*cos(q1)^2*cos(q2)^2 + cos(q2 +
q3)^2*sin(q1)^2*sin(q2)^2 + sin(q2 + q3)^2*cos(q2)^2*sin(q1)^2 -
2*cos(q2 + q3)*sin(q2 + q3)*cos(q1)^2*cos(q2)*sin(q2) - 2*cos(q2 +
q3)*sin(q2 + q3)*cos(q2)*sin(q1)^2*sin(q2)),


        0]
[ (cos(q1)^3*sin(q2) - sin(q2 + q3)*cos(q1)*cos(q2)^2 +
 cos(q2 + q3)^2*cos(q1)*sin(q2) + cos(q1)*sin(q1)^2*sin(q2)
 - cos(q2 + q3)*sin(q2 + q3)*cos(q1)*cos(q2) + cos(q2
 + q3)*cos(q1)*cos(q2)*sin(q2))/(cos(q1)^2*cos(q2)^2 +
 cos(q2)^2*sin(q1)^2 + cos(q1)^4*sin(q2)^2 + sin(q1)^4*sin(q2)^2 +
 2*cos(q1)^2*sin(q1)^2*sin(q2)^2 + cos(q2 + q3)^2*cos(q1)^2*sin(q2)^2
 + sin(q2 + q3)^2*cos(q1)^2*cos(q2)^2 + cos(q2 +
 q3)^2*sin(q1)^2*sin(q2)^2 + sin(q2 + q3)^2*cos(q2)^2*sin(q1)^2 -
 2*cos(q2 + q3)*sin(q2 + q3)*cos(q1)^2*cos(q2)*sin(q2) - 2*cos(q2 +
 q3)*sin(q2 + q3)*cos(q2)*sin(q1)^2*sin(q2)), (sin(q1)^3*sin(q2) -
 sin(q2 + q3)*cos(q2)^2*sin(q1) + cos(q2 + q3)^2*sin(q1)*sin(q2) +
 cos(q1)^2*sin(q1)*sin(q2) - cos(q2 + q3)*sin(q2 + q3)*cos(q2)*sin(q1)
 + cos(q2 + q3)*cos(q2)*sin(q1)*sin(q2))/(cos(q1)^2*cos(q2)^2 +
 cos(q2)^2*sin(q1)^2 + cos(q1)^4*sin(q2)^2 + sin(q1)^4*sin(q2)^2 +
 2*cos(q1)^2*sin(q1)^2*sin(q2)^2 + cos(q2 + q3)^2*cos(q1)^2*sin(q2)^2
 + sin(q2 + q3)^2*cos(q1)^2*cos(q2)^2 + cos(q2 +
 q3)^2*sin(q1)^2*sin(q2)^2 + sin(q2 + q3)^2*cos(q2)^2*sin(q1)^2 -
 2*cos(q2 + q3)*sin(q2 + q3)*cos(q1)^2*cos(q2)*sin(q2) - 2*cos(q2
 + q3)*sin(q2 + q3)*cos(q2)*sin(q1)^2*sin(q2)), (cos(q1)^2*cos(q2)
 + cos(q2)*sin(q1)^2 - cos(q2 + q3)*cos(q1)^2*sin(q2)^2 + sin(q2 +
 q3)^2*cos(q1)^2*cos(q2) - cos(q2 + q3)*sin(q1)^2*sin(q2)^2 + sin(q2
 + q3)^2*cos(q2)*sin(q1)^2 + sin(q2 + q3)*cos(q1)^2*cos(q2)*sin(q2)
 + sin(q2 + q3)*cos(q2)*sin(q1)^2*sin(q2) - cos(q2 + q3)*sin(q2 +
```

```
 q3)*cos(q1)^2*sin(q2) - cos(q2 + q3)*sin(q2 + q3)*sin(q1)^2*sin(q2))/
(cos(q1)^2*cos(q2)^2 + cos(q2)^2*sin(q1)^2 + cos(q1)^4*sin(q2)^2
 + sin(q1)^4*sin(q2)^2 + 2*cos(q1)^2*sin(q1)^2*sin(q2)^2
 + cos(q2 + q3)^2*cos(q1)^2*sin(q2)^2 + sin(q2 +
 q3)^2*cos(q1)^2*cos(q2)^2 + cos(q2 + q3)^2*sin(q1)^2*sin(q2)^2
 + sin(q2 + q3)^2*cos(q2)^2*sin(q1)^2 - 2*cos(q2 + q3)*sin(q2
 + q3)*cos(q1)^2*cos(q2)*sin(q2) - 2*cos(q2 + q3)*sin(q2 +
 q3)*cos(q2)*sin(q1)^2*sin(q2)), -(cos(q1)^2*sin(q1)*sin(q2)^2
 + sin(q2 + q3)*cos(q1)^2*sin(q1)*sin(q2) + cos(q2)^2*sin(q1)
 + cos(q2 + q3)*cos(q2)*sin(q1) + sin(q1)^3*sin(q2)^2 +
 sin(q2 + q3)*sin(q1)^3*sin(q2))/(cos(q1)^2*cos(q2)^2 +
 cos(q2)^2*sin(q1)^2 + cos(q1)^4*sin(q2)^2 + sin(q1)^4*sin(q2)^2 +
 2*cos(q1)^2*sin(q1)^2*sin(q2)^2 + cos(q2 + q3)^2*cos(q1)^2*sin(q2)^2
 + sin(q2 + q3)^2*cos(q1)^2*cos(q2)^2 + cos(q2 +
 q3)^2*sin(q1)^2*sin(q2)^2 + sin(q2 + q3)^2*cos(q2)^2*sin(q1)^2 -
 2*cos(q2 + q3)*sin(q2 + q3)*cos(q1)^2*cos(q2)*sin(q2) - 2*cos(q2 +
 q3)*sin(q2 + q3)*cos(q2)*sin(q1)^2*sin(q2)), (cos(q1)^3*sin(q2)^2
 + sin(q2 + q3)*cos(q1)^3*sin(q2) + cos(q1)*cos(q2)^2 + cos(q2
 + q3)*cos(q1)*cos(q2) + cos(q1)*sin(q1)^2*sin(q2)^2 + sin(q2
 + q3)*cos(q1)*sin(q1)^2*sin(q2))/(cos(q1)^2*cos(q2)^2 +
 cos(q2)^2*sin(q1)^2 + cos(q1)^4*sin(q2)^2 + sin(q1)^4*sin(q2)^2 +
 2*cos(q1)^2*sin(q1)^2*sin(q2)^2 + cos(q2 + q3)^2*cos(q1)^2*sin(q2)^2
 + sin(q2 + q3)^2*cos(q1)^2*cos(q2)^2 + cos(q2 +
 q3)^2*sin(q1)^2*sin(q2)^2 + sin(q2 + q3)^2*cos(q2)^2*sin(q1)^2 -
 2*cos(q2 + q3)*sin(q2 + q3)*cos(q1)^2*cos(q2)*sin(q2) - 2*cos(q2 +
 q3)*sin(q2 + q3)*cos(q2)*sin(q1)^2*sin(q2)),

        0]
```

Planner

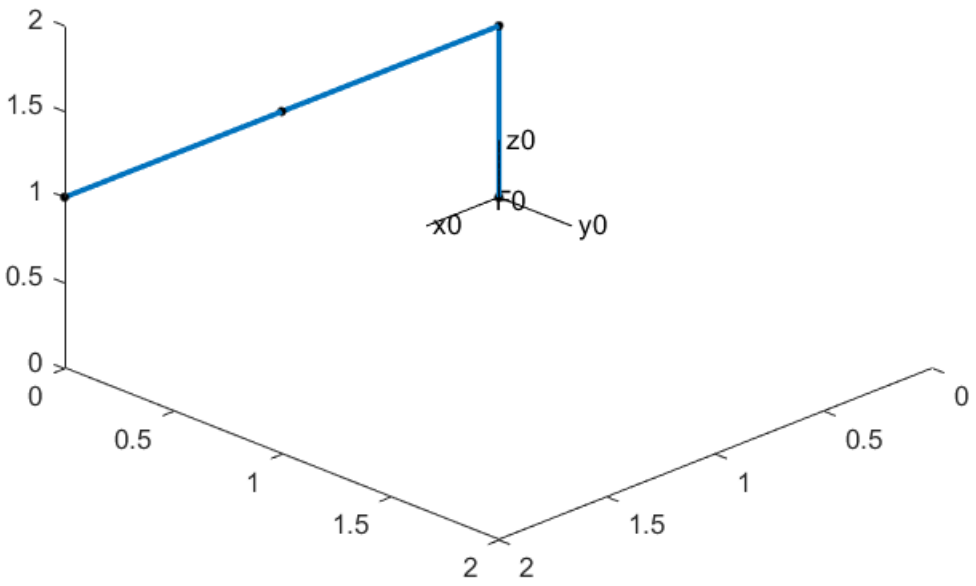
```
close all; clc;
t=sym('t','real');
t_range=0:0.02:10;

q_d = [t/8;0.2;sin(2*t)];

x0=[ 0      ,0;
    -0.5    ,0.1;
     0.2    ,0.1];

plan1=Planner.fromSym(q_d(1));
plan2=Planner.trapezoid(x0(2,:),[q_d(2),0],1,1);
plan3=Planner.fromSym(q_d(3));
plan=Planner.join({plan1,plan2,plan3});

testTrapezoid([0,-1],[5,0],1,1);
testTrapezoid(x0(2,:),[q_d(2),0],1,1);
```

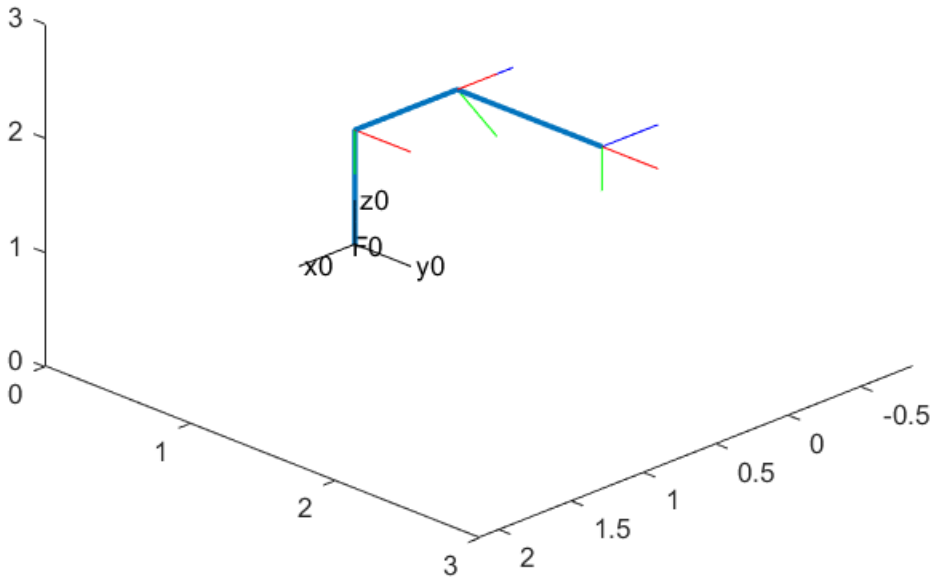

position
velocity
acceleration


position
velocity
acceleration

Draw/Plot

```
close all; clc;

kin.draw([0;0;0],0,[],{'linewidth',2},[1,1,1])
figure
kin.draw([pi/2;-pi/4;pi/4],1,[],{'linewidth',2},[1,1,1])
```





Dyn_Model

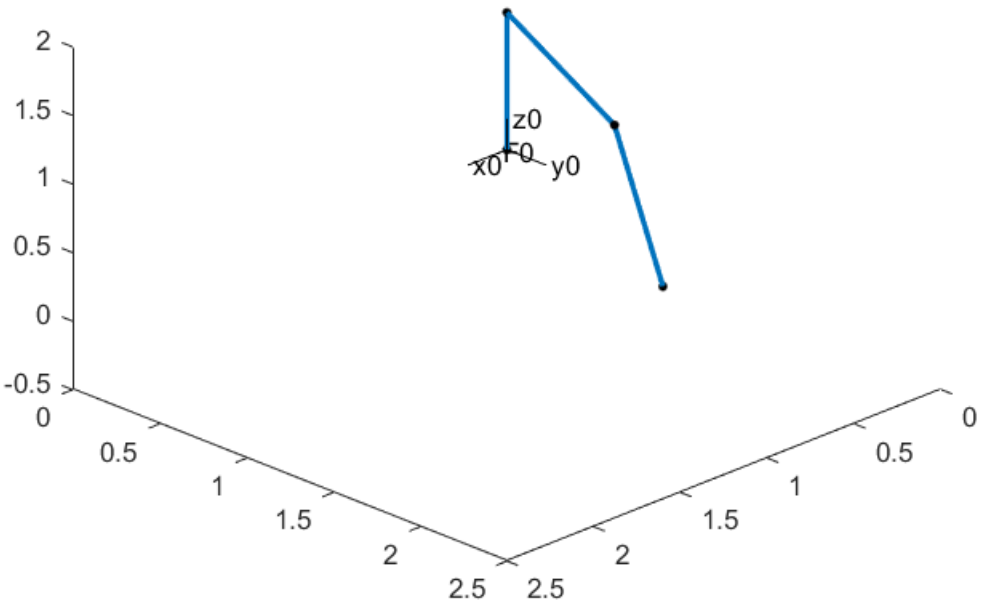
```
close all; clc;

model=Dyn_Model(kin);

model=model.add(H_Trans([-0.5,0,0]),5,10,1);
model=model.add(H_Trans([-0.5,0,0]),5,10,2);
model=model.add(H_Trans([-0.5,0,0]),5,10,3);

model.b=[1;1;1];

model.g_dir=[0;0;-1];

disp('Calculating Dynamics');
model = model.calculateDynamics();

M=model.M
G=model.G
```

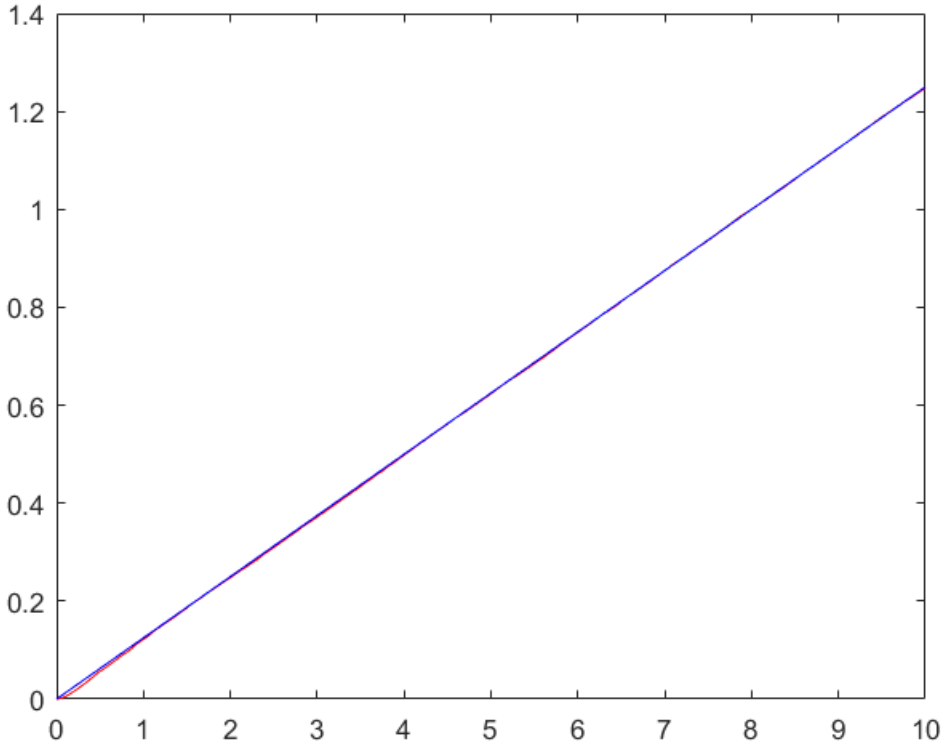
```
Calculating Dynamics

M =
```

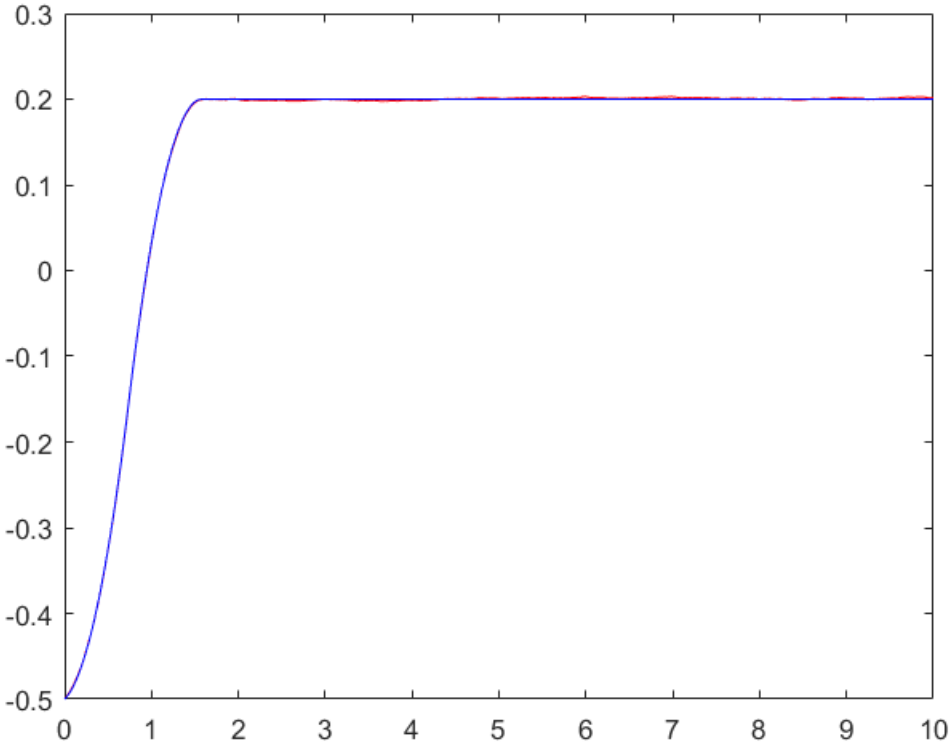
```
[ (5*cos(2*q2 + q3))/2 + (25*cos(2*q2))/8 + (5*cos(2*q2 + 2*q3))/8 +
 (5*cos(q3))/2 + 5,
              0,                                                    0]
[
                  0, 5*cos(2*q1 + 2*q2 + 2*q3) + 5*cos(2*q1 + 2*q2) +
 5*cos(q3) + 35/2, 5*cos(2*q1 + 2*q2 + 2*q3) + (5*cos(q3))/2 + 25/4]
[
                    0,                  5*cos(2*q1 + 2*q2 + 2*q3) +
 (5*cos(q3))/2 + 25/4,                  5*cos(2*q1 + 2*q2 + 2*q3) +
 25/4]


G =

                                                 0
 - (981*cos(q2 + q3))/40 - (2943*cos(q2))/40
                       -(981*cos(q2 + q3))/40
```

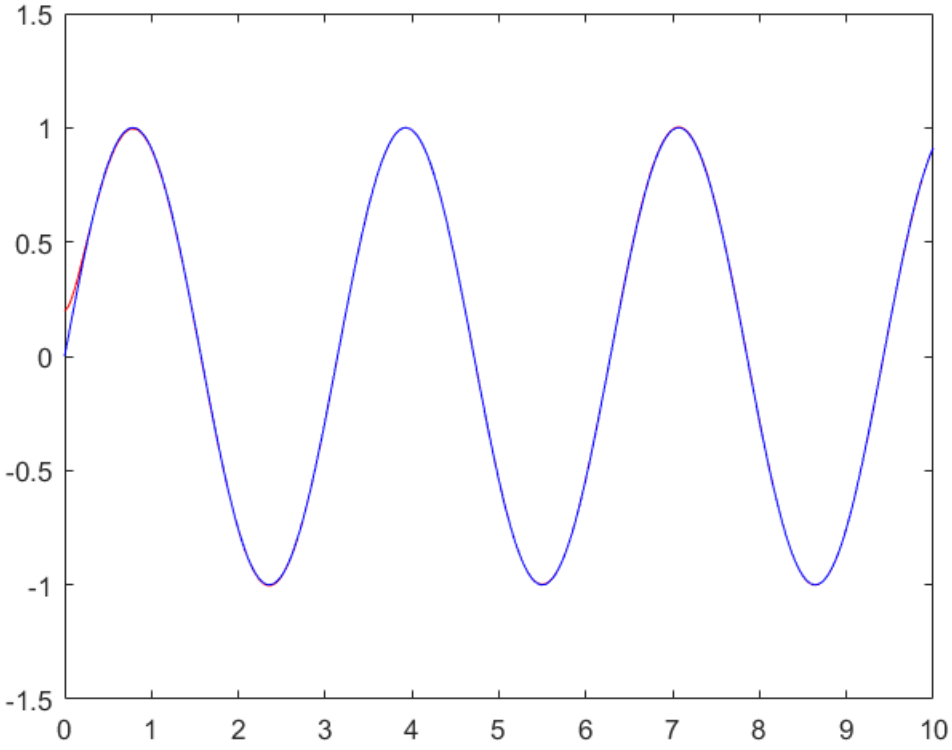
Simulate

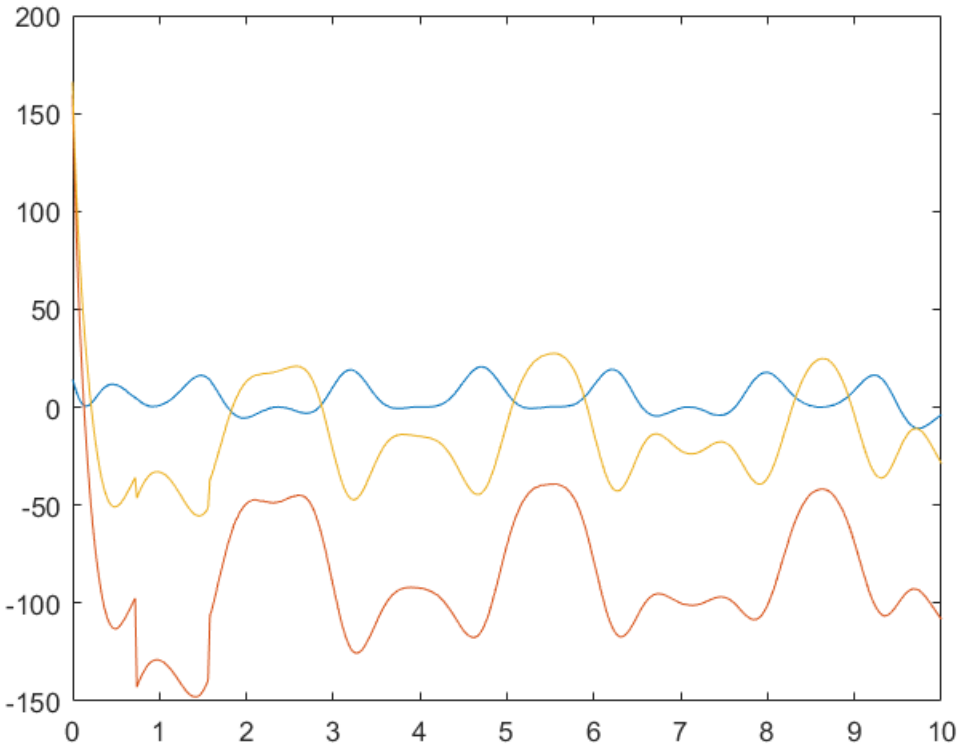
```
close all; clc;

Kp=[10;10;10];
Kv=[10;10;10];

control=Controller.ComputedTorque(@model.inverse_dyn,Kp,Kv);
noise=@(a,t)0.02*randn(size(a));

options =
 odeset('RelTol',1e-4,'AbsTol',1e-4.*ones(numel(model.q)*2,1));

model.simulate({plan,control,noise},t_range,x0,options,{0.01,false,{0,
[],{'linewidth',2},[1,1,1]}});

Beginning Simulation
Simulation Complete
```


z0
x0
y0
2
1.5
1
0.5
0
-0.5
0
0.5
1
1.5
2
2.5
2.5
2
1.5
1
0.5
0


1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
1
2
3
4
5
6
7
8
9
10


0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
0
1
2
3
4
5
6
7
8
9
10


1.5
1
0.5
0
-0.5
-1
-1.5
0
1
2
3
4
5
6
7
8
9
10

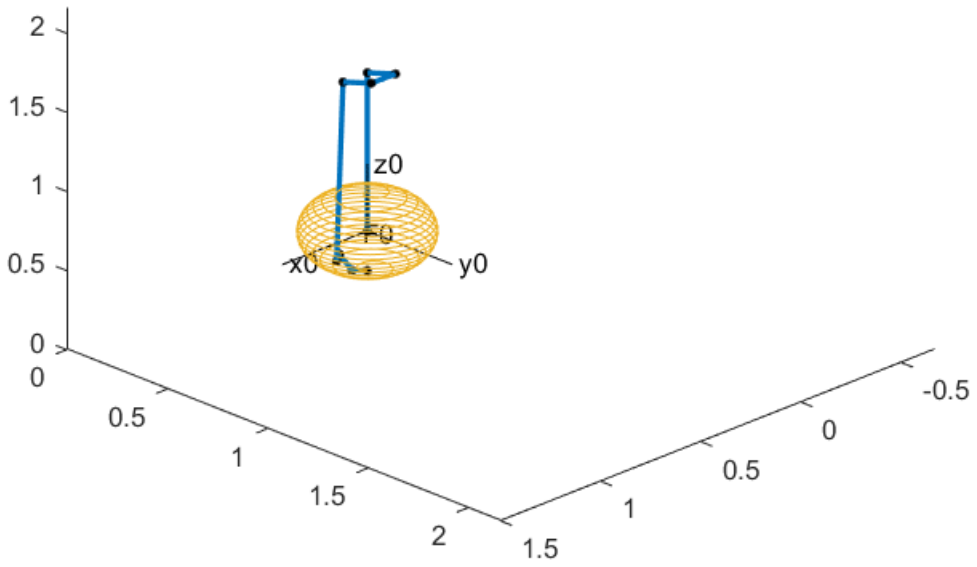


Workspace Planning

```
close all; clc;

t=sym('t','real');
t_range=0:0.02:10;

q=sym('q',[6,1],'real');

disp('Calculating Kinematics');
kin=Kin_Model.fromDH([q(1),1,0,pi/2;
                      q(2),-0.1,0,0;
                      0,0,1,0;
                      q(3),0.1,0,0;
                      0,0,1,pi/2;
                      q(4),0.1,0,-pi/2;
                      q(5),0.1,0,-pi/2;
                      q(6),0.1,0,pi/2],q);

disp('Creating Path');
r=.25;
p0=[.75;.75;.75];
z_t=r-2*r/t_range(numel(t_range))*t+p0(3);
r_t=sqrt(r^2-(z_t-p0(3))^2);
p_d=[p0(1)+r_t*cos(10*t);p0(2)+r_t*sin(10*t);z_t];
```

```
x_d=[NaN,NaN,NaN,p_d(1);
     NaN,NaN,NaN,p_d(2);
     NaN,NaN,NaN,p_d(3);
     NaN,NaN,NaN,NaN];

plan=Planner.fromSym(x_d);
disp('Converting to Joint Space');
plan=Planner.toJointSpace_func(plan,t_range,[],@kin.inverse_kin);

D=evalf(plan,t_range.');
kin.simulate(D(:,:,1).',0.01,true,{0,[],{'linewidth',2},[1,1,1]});
```

```
Calculating Kinematics
Creating Path
Converting to Joint Space
```



Published with MATLAB® R2015b