Contents


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

DH Parameter of Manipulator

DH-Parameter is the configuration that is needed to generate the robot simulation. Using those configurations a robot can be built into a software to generate for simulation. Here Robotic Toolbox of Peter Corke's is used for generating manipulators into MATLAB. As seen below, it can be seen that each joint of a manipulator is defined into an array, and then that array is called into the SerialLink which generates the manipulator based on the DH-Parameter.

```
L (1) = Revolute('d',0.345  ,'a',0.02   ,'alpha',pi/2       ,'qlim',[-2.9671 2.9671]);
L (2) = Revolute('d',0      ,'a',0.260  ,'alpha',0          ,'qlim',[-0.8727 2.9671]);
L (3) = Revolute('d',0      ,'a',0.02   ,'alpha',pi/2       ,'qlim',[(-2.7053+(pi/2)) (1.9198+(pi/2))]);
L (4) = Revolute('d',0.260  ,'a',0      ,'alpha',(-pi/2)    ,'qlim',[-3.0543 3.0543]);
L (5) = Revolute('d',0      ,'a',0      ,'alpha',(pi/2)     ,'qlim',[-2.0945 2.0945]);
L (6) = Revolute('d',0.075  ,'a',0      ,'alpha',0          ,'qlim',[(-6.1087) (6.1087)]);
qz = [0 0 0 0 0 0];
KR3 = SerialLink(L,'name','KUKA KR3 Agilus');
KR3.manufacturer = 'JD & M';
KR3.ikineType = 'KR3';
```

Inputs

In this section, The task space points which are generated using other software like SolidWorks are being converted using the ConversionC function into an array. Time limit is also given to complete the given task.

```
% Points = conversionP('Trianguler_Piramid.csv');
Points = conversionC('Circle.csv');
N = length(Points(1,:));
timeDuration = 4;
tic
```

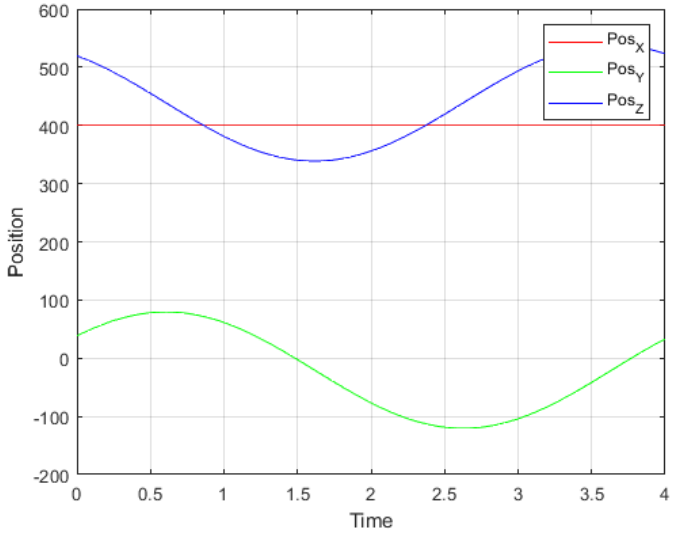
Trajectory

A quintic polynomial is generated using the trajectory function for a smoother path. Quintic polynomial can smoother the position until its fifth derivative. For smoother operation as in the manipulator's joint velocity and acceleration, quintic is used.

```
[traj,T] = trajectory(Points,timeDuration);


figure;trplot(KR3.fkine(qz),'length',0.001);
hold on;
for i = 1 : N-1
    for j = 1 : length(traj(1,1,:,1))
        trplot(traj(:,:,j,i),'length',0.01,'notext','color','r');
    end
end
```

```
Elapsed time is 1.214195 seconds.
```



Inverse Kinematics

In this section, Inverse kinematics is solved for the above-generated trajectory. Due to the absence of an analytical solution of KR3, the numerical iterative solution is used to calculate the inverse kinematics of the given trajectory using the fsolve. Although the analytical solution is derived partially and it's used as the initial value need into the fsolve.

```
count = 0;
for k = 1:N-1
    for h = 1:length(T)-1
```

```
        Des = traj(:,:,h,k);
        Ini_val = geo_sol(Des);
        if(h == 1 && k == 1)
            Ini_val = geo_sol(Des);
        elseif(h==1 && k~=1)
            Ini_val = OPx(:,length(T)-1,k-1);
        else
            Ini_val = OPx(:,h-1,k);
        end
        Func = 1e-3;
        Step = 1e-3;
        options = optimoptions('fsolve','Display','off','Algorithm','levenberg-marquardt','FunValCheck','on','FunctionTolerance',Func,'StepTolerance',Step);
        tempOpx(:,h,k) = fsolve(@ikine_Num,Ini_val,options,Des);
```

Checking Joint Limitation

Here to avoid collision of the joints into the simulation, a filter is used for joint angles. which converts the angle from one quadrant to another quadrant.

```
.....................Checking 2pi variation.................................
    ....................Th1.................................................
        if(tempOpx(1,h,k)>=(2*pi))
            tempOpx(1,h,k) = tempOpx(1,h,k) - 2*pi;
        elseif(tempOpx(1,h,k)<=(-2*pi))
            tempOpx(1,h,k) = tempOpx(1,h,k) - 2*pi;
        end
    ....................Th2.................................................
        if(tempOpx(2,h,k)>=(2*pi))
            tempOpx(2,h,k) = tempOpx(2,h,k) - 2*pi;
        elseif(tempOpx(2,h,k)<=(-2*pi))
            tempOpx(2,h,k) = tempOpx(2,h,k) - 2*pi;
        end
    ....................Th3.................................................
        if(tempOpx(3,h,k)>=(2*pi))
            tempOpx(3,h,k) = tempOpx(3,h,k) - 2*pi;
        elseif(tempOpx(3,h,k)<=(-2*pi))
            tempOpx(3,h,k) = tempOpx(3,h,k) - 2*pi;
        end
    ....................Th4.................................................
        if(tempOpx(4,h,k)>=(2*pi))
            tempOpx(4,h,k) = tempOpx(4,h,k) - 2*pi;
        elseif(tempOpx(4,h,k)<=(-2*pi))
            tempOpx(4,h,k) = tempOpx(4,h,k) - 2*pi;
        end
    ....................Th5.................................................
        if(tempOpx(5,h,k)>=(2*pi))
            tempOpx(5,h,k) = tempOpx(5,h,k) - 2*pi;
        elseif(tempOpx(5,h,k)<=(-2*pi))
            tempOpx(5,h,k) = tempOpx(5,h,k) - 2*pi;
        end
    ....................Th6.................................................
        if(tempOpx(6,h,k)>=(2*pi))
            tempOpx(6,h,k) = tempOpx(6,h,k) - 2*pi;
        elseif(tempOpx(6,h,k)<=(-2*pi))
            tempOpx(6,h,k) = tempOpx(6,h,k) - 2*pi;
        end
.....................Exception of Qlimit.....................................
```

```
.....Th1.....................................

        if(tempOpx(1,h,k)<KR3.qlim(1,1))
            OPx(1,h,k) = KR3.qlim(1,1);
            warning(' Q-limit Reached at Theta One')
        elseif(tempOpx(1,h,k)>KR3.qlim(1,2))
            OPx(1,h,k) = KR3.qlim(1,2);
            warning(' Q-limit Reached at Theta One')
        else
            OPx(1,h,k) = tempOpx(1,h,k);
        end

.....Th2.....................................

        if(tempOpx(2,h,k)<KR3.qlim(2,1))
            OPx(2,h,k) = KR3.qlim(2,1);
            warning(' Q-limit Reached at Theta Two')
        elseif(tempOpx(2,h,k)>KR3.qlim(2,2))
            OPx(2,h,k) = KR3.qlim(2,2);
            warning(' Q-limit Reached at Theta Two')
        else
            OPx(2,h,k) = tempOpx(2,h,k);
        end

.....Th3.....................................

        if(tempOpx(3,h,k)<KR3.qlim(3,1))
            OPx(3,h,k) = KR3.qlim(3,1);
            warning(' Q-limit Reached at Theta Three')
        elseif(tempOpx(3,h,k)>KR3.qlim(3,2))
            OPx(3,h,k) = KR3.qlim(3,2);
            warning(' Q-limit Reached at Theta Three')
        else
            OPx(3,h,k) = tempOpx(3,h,k);
        end

.....Th4.....................................

        if(tempOpx(4,h,k)<KR3.qlim(4,1))
            OPx(4,h,k) = KR3.qlim(4,1);
            warning(' Q-limit Reached at Theta Four')
        elseif(tempOpx(4,h,k)>KR3.qlim(4,2))
            OPx(4,h,k) = KR3.qlim(4,2);
            warning(' Q-limit Reached at Theta Four')
        else
            OPx(4,h,k) = tempOpx(4,h,k);
        end

.....Th5.....................................

        if(tempOpx(5,h,k)<KR3.qlim(5,1))
            OPx(5,h,k) = KR3.qlim(5,1);
            warning(' Q-limit Reached at Theta Five')
        elseif(tempOpx(5,h,k)>KR3.qlim(5,2))
            OPx(5,h,k) = KR3.qlim(5,2);
            warning(' Q-limit Reached at Theta Five')
        else
            OPx(5,h,k) = tempOpx(5,h,k);
        end

.....Th6.....................................

        if(tempOpx(6,h,k)<KR3.qlim(6,1))
            OPx(6,h,k) = KR3.qlim(6,1);
            warning(' Q-limit Reached at Theta Six')
        elseif(tempOpx(6,h,k)>KR3.qlim(6,2))
            OPx(6,h,k) = KR3.qlim(6,2);
            warning(' Q-limit Reached at Theta Six')
        else
            OPx(6,h,k) = tempOpx(6,h,k);
        end
```

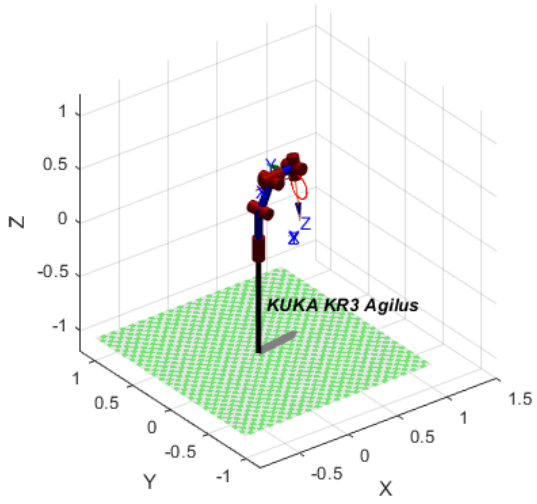
Checking Singularities

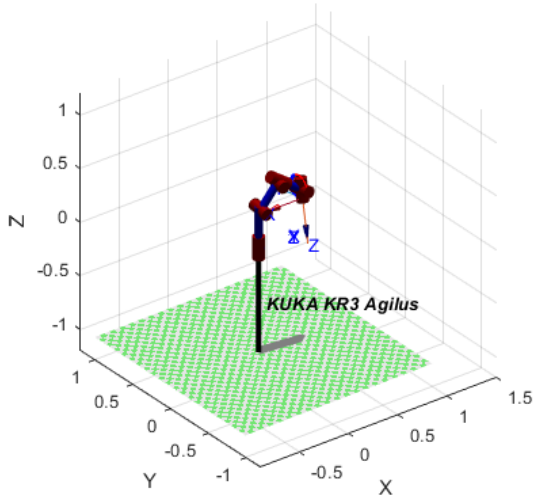
To detect any type of singularity, a conditional filter is used in which the Jacobian Matrix is calculated and its determinant is the which is used to separate the singular pose.

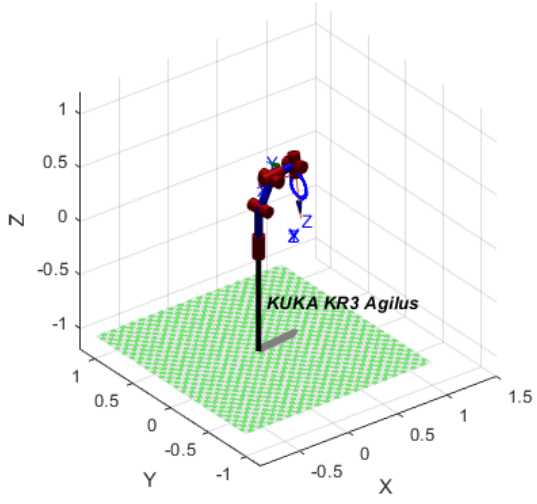
```
        Jacobian(:,:,h,k) = KR3.jacob0(OPx(:,h,k));
        Determinant(h,k) = det(Jacobian(:,:,h,k));

        if ((Determinant(h,k) <= 1e-4) && (Determinant(h,k) >= (-(1e-4))))
            fprintf('\n Jacobian of the pose(%d(%d)).\n',k,h);
            disp(Jacobian(:,:,h,k));
            fprintf('\n Determinant of the pose(%d(%d)): %f\n',k,h,Determinant(h,k));
            warning('Determinant of the pose zero it shows the singularity.')
        end

        KR3.plot(OPx(:,h,k)');
        Ans(h,k) = KR3.fkine(OPx(:,h,k));
        trplot(Ans(h,k),'length',0.03,'notext');
```








```
    end

TimeStamp(:,k) = T+((k-1)*(timeDuration/(N-1)));

end
toc
```

```
Elapsed time is 95.910649 seconds.
```