Simulation of 6 DOF KUKA KR210 Mounted on Linear Rail

Project Report

Jerrar Bukhari

ENPM 662 - Robot Modelling

University of Maryland, College Park

Abstract

This report thoroughly examines and simulates a 6 Degrees of Freedom KUKA robotic Arm mounted on a linear rail. The work presented encompasses constructing the DH parameters of the specific robotic arm and finding its forward and inverse kinematic equations with the help of scientific computational software. A computer model of the underlying physics is generated, and position and trajectory control of the end effector are visually demonstrated. This is verified by comparing results from the mathematical equations and the simulation. This work may serve as a proof of concept towards developing a rail mounted robotic arm suitable to operate continuously over relatively long distance e.g. ship hull welding, aircraft exterior painting, etc.

Motivation

Robotic arms have become the industry standard alternative to laborious and monotonous jobs disliked by human workers. Along with cost savings, deployment of robotic arms dramatically increases the quality of work done. Some industrial sectors still lack the desired level of automation due to unavailability of feasible robotic configurations. One such example is of the ship industry. Currently, the ships metallic hulls are welded manually by laborers since the welding spans long continuous contours. Deploying a fixed robotic arm in such cases would be fruitless. A robotic arm mounted on a linear rail, however, will solve this problem.

KR 210 R2700 Extra is a popular industrial robotic arm manufactured by KUKA and has a proven track record. Due to such prevalent use in the industry, this arm, is particularly well documented and its related resources are readily available over the internet. Utilizing the resources has greatly helped us in deliver results relatively in time. Furthermore, referencing material already published for this arm contributed towards validating of our methodology.

Objectives

The end goal of this project is to demonstrate the following:

1. Represent the forward and inverse kinematics of robotic system using mathematical equations
2. Solve the inverse kinematics using both equations and iterative computer algorithm
3. Model and visualize the robotic system in a simulation software
4. Validate the results obtained by cross comparison

Methodology

Classical DH-Jacobian Approach

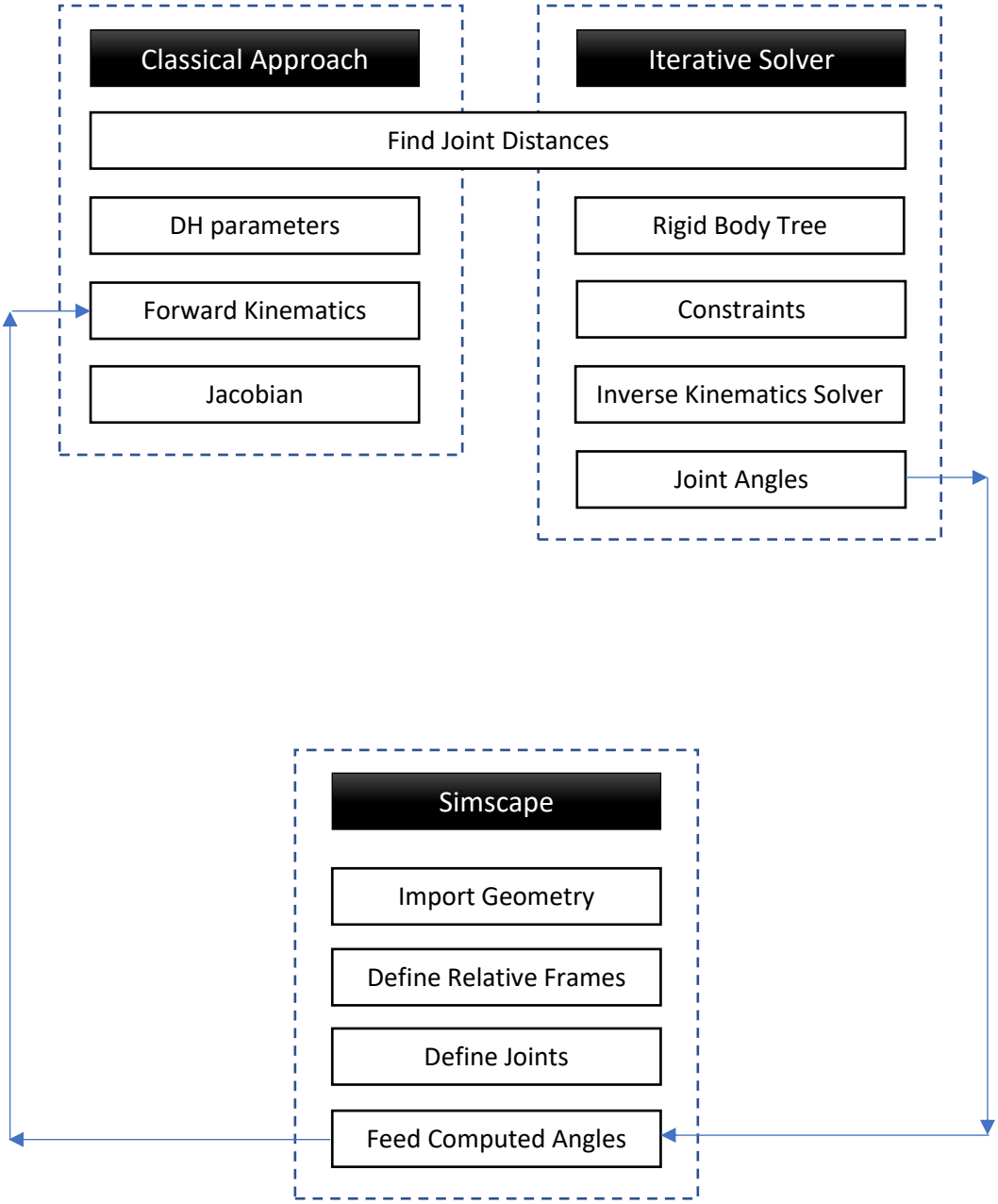
For calculating the closed form mathematical equations, we will utilize the classical DH parameterization for robotic arms. These parameters will be derived based on the CAD geometry of the robotic arm imported and its joint distances analyzed in the CAD software. Consequently, using transformations matrices we can find the forward kinematics. For inverse kinematics, we will find the Jacobian, inverse of which can be utilized to find the joint angles for reaching a desired position.

Iterative Solver Approach

The above results can be achieved by following a different approach which does not involve solving for closed form equations. This method involves a MATLAB based Robotics Toolbox. The robot model will be generated in the toolbox by specifying the frame transitions using homogenous transformation matrices resulting in a Rigid Body Tree model. The result will be an identical representation of the robotic arm as we get from the DH technique. This toolbox has inverse kinematics solver based on "Levenberg Marquardt" algorithm which iteratively solves for joint angles considering any user specified constraints to solutions.

Simscape Simulation

Simscape is a MATLAB tool for physical systems simulation. A 3D CAD model can be imported directly in to Simscape for analysis via the Multibody Link plugin. After the geometry is ready, one needs to set up frames of references and define joint types. Simscape has a plethora of tools to visualize and gather data of every physical form. In our case, the point of interest is just the translation and orientation of the end effector and intermediate joints. The simulated physical model can be fed the joint angles calculated using both the above techniques and the resulting behavior observed.

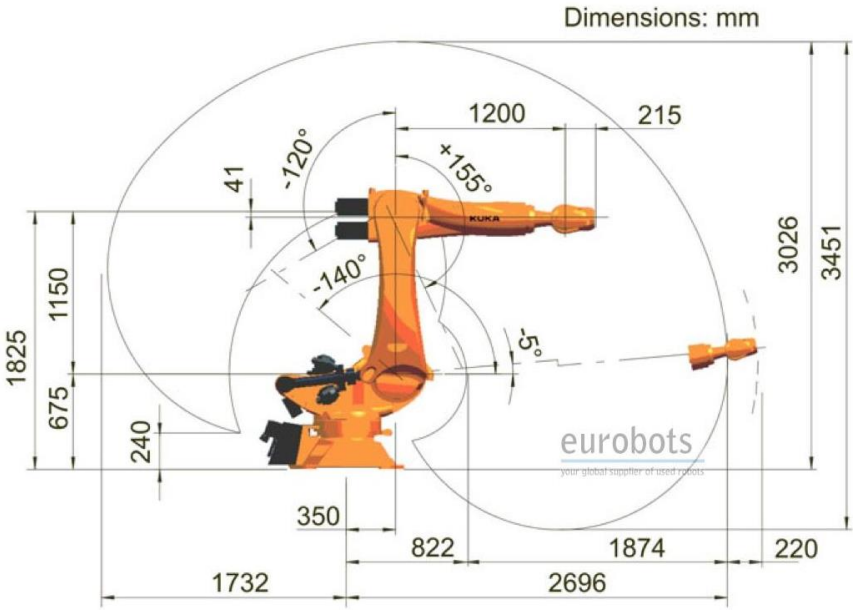

Classical Approach
Iterative Solver
Find Joint Distances
DH parameters
Rigid Body Tree
Forward Kinematics
Constraints
Jacobian
Inverse Kinematics Solver
Joint Angles
Simscape
Import Geometry
Define Relative Frames
Define Joints
Feed Computed Angles

KUKA KR210 R2700 Extra - Specifications

The KUKA KR 210 is a heavy industrial robot with an extensive work envelope and apt payload capacity for manipulating welding torch in a ship welding environment. The specifications listed below are populating from the manufacturer's website. [1]

Robot Specifications	
Axes:	6
Payload:	210kg
H-Reach:	2700mm
Repeatability:	± 0.06mm
Robot Mass:	1078kg
Structure:	Articulated
Mounting:	Floor, Ceiling

Robot Motion Range	
Axis 1	±185°
Axis 2	+140° - 5°
Axis 3	+120° - 155°
Axis 4	±350°
Axis 5	±125°
Axis 6	±350°



CAD Model

The open source CAD model of the KUKA KR 210 R2700 has been sourced form Grab CAD. Auto Desk Inventor Professional 2018 has been used to create a simplistic linear rail and the robotic arm appropriately mounted on the rail as seen below.

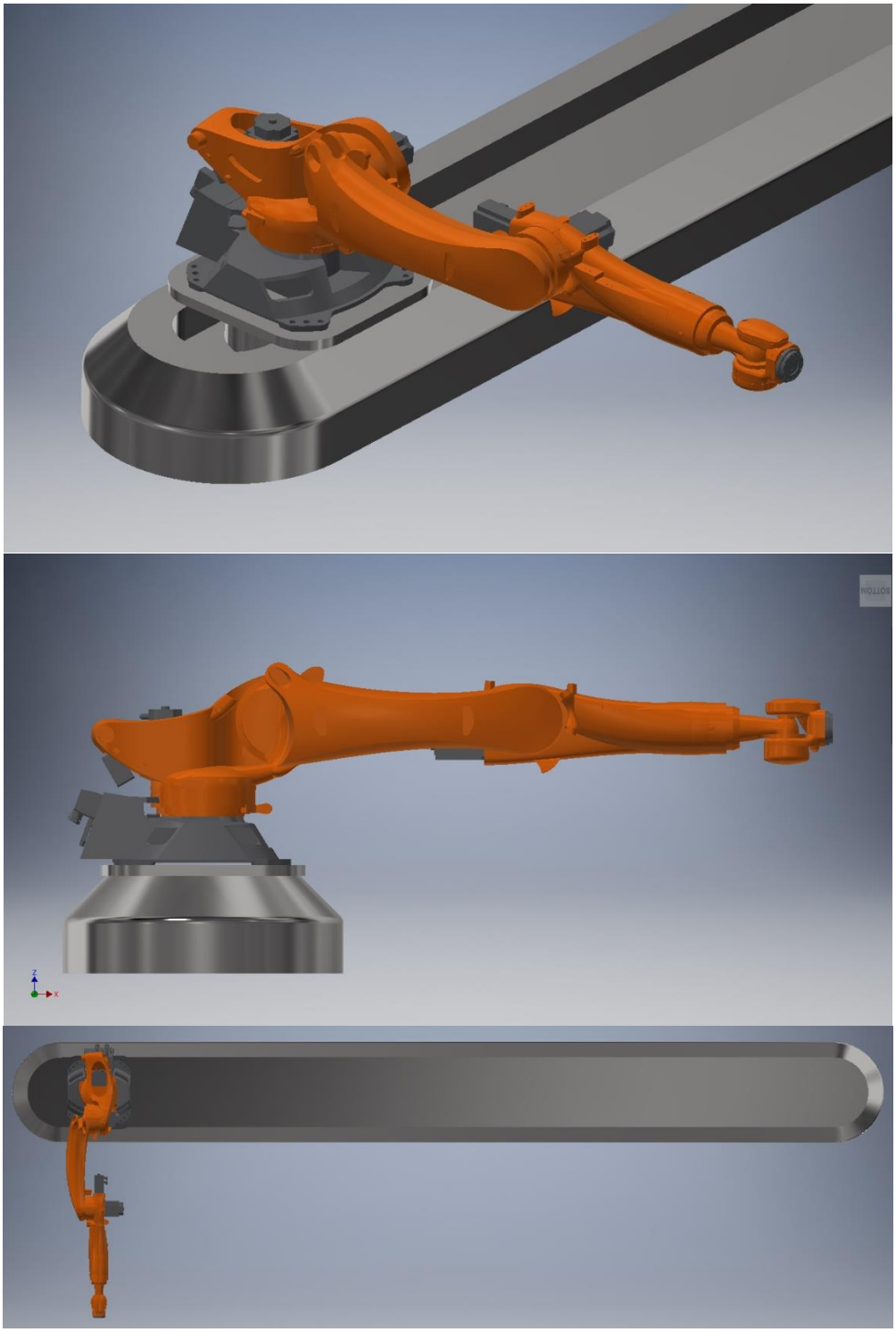


Figure 1 Isometric View(1st), Side View (2nd), Top View(3rd) of Robotic Arm

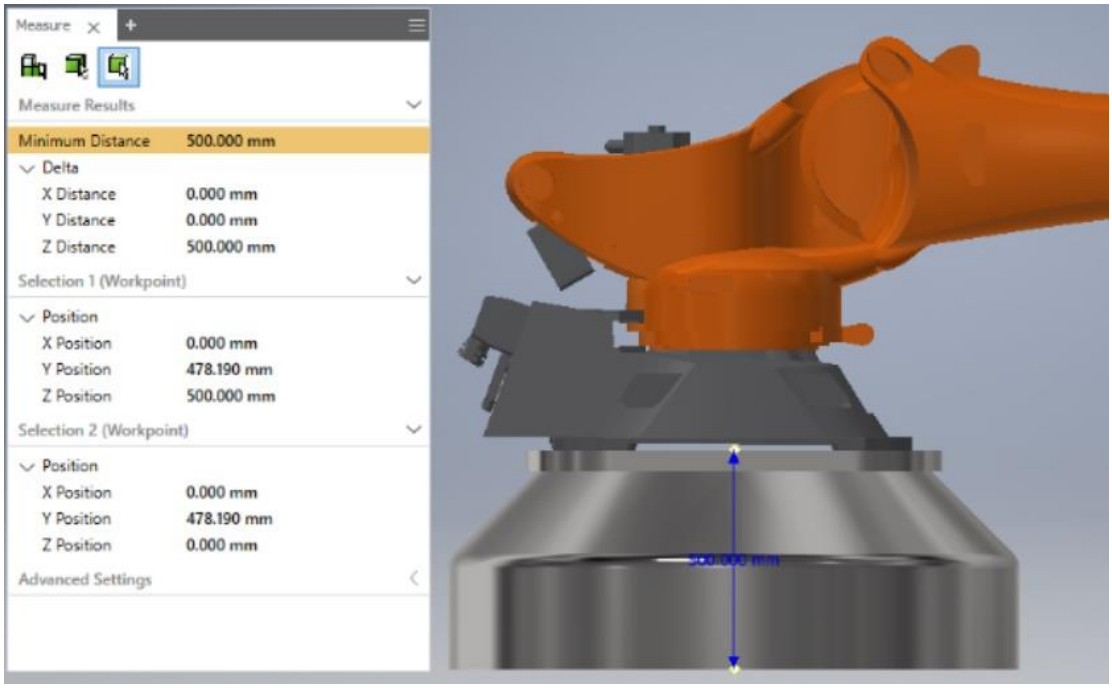
Joint Distances

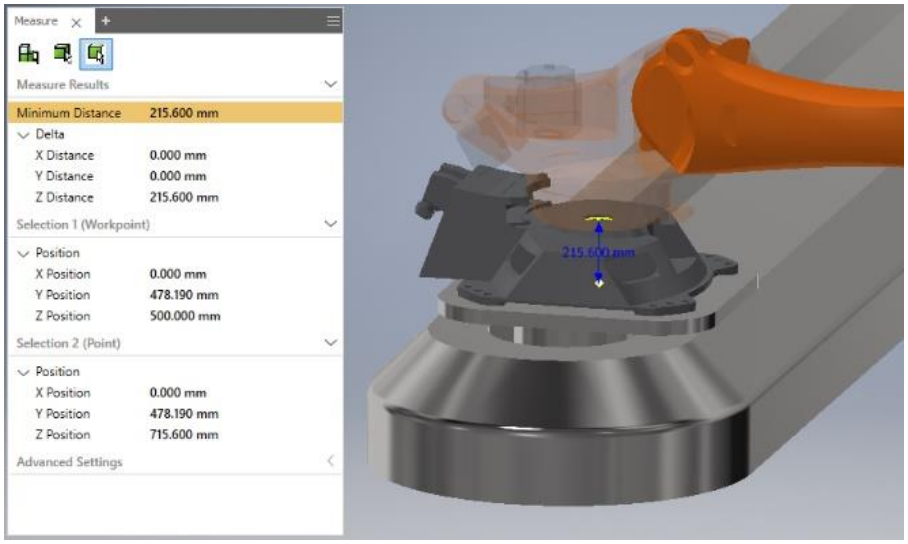
Drawing reference sketches and axis on the CAD geometry helps easily identify joint locations with reference to previous joints or the world frame. Measure tool in Auto Desk Inventor is a convenient way to examine distances between points. The distances between joints are tabulated below:

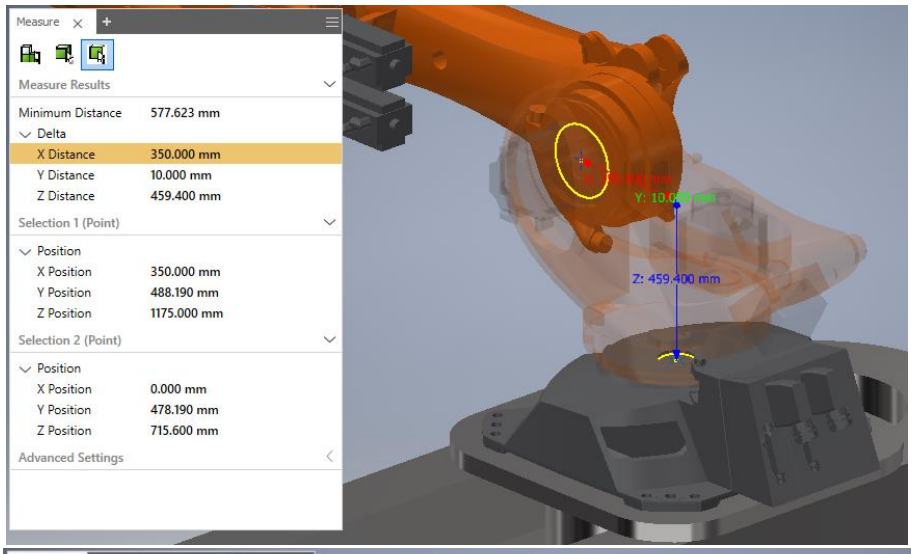
Joint Number	X (mm)	Y (mm)	Z (mm)
Joint 1	0	0	500
Joint 2	0	0	215.6
Joint 3	356	10	459.4
Joint 4	1150	-178	0
Joint 5	752	168	41
Joint 6	448	0	0
End Effector	207	0	0

Table 1 Cartesian Distances between Joints of Robotic Arm

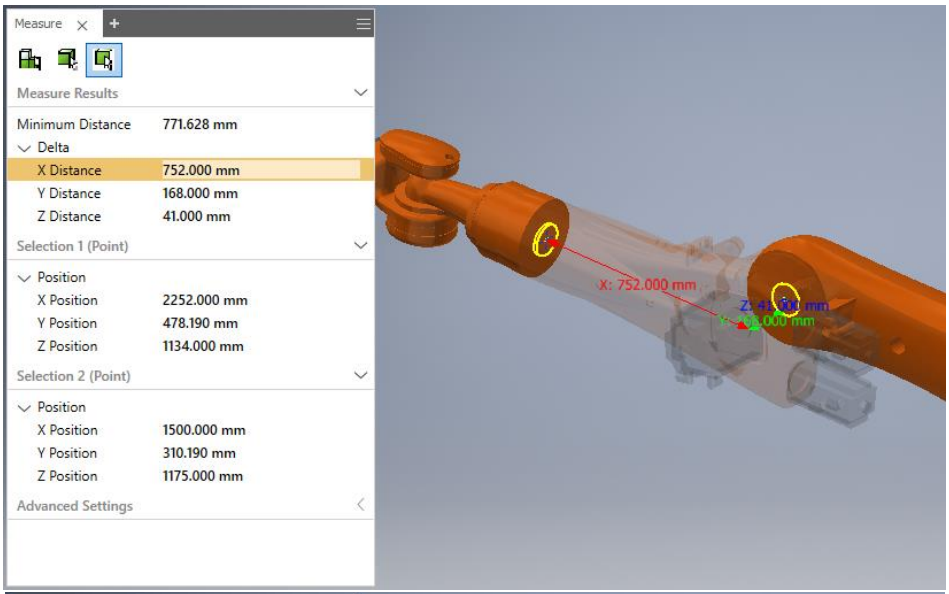
Images below illustrate the points of measurement and the corresponding values:

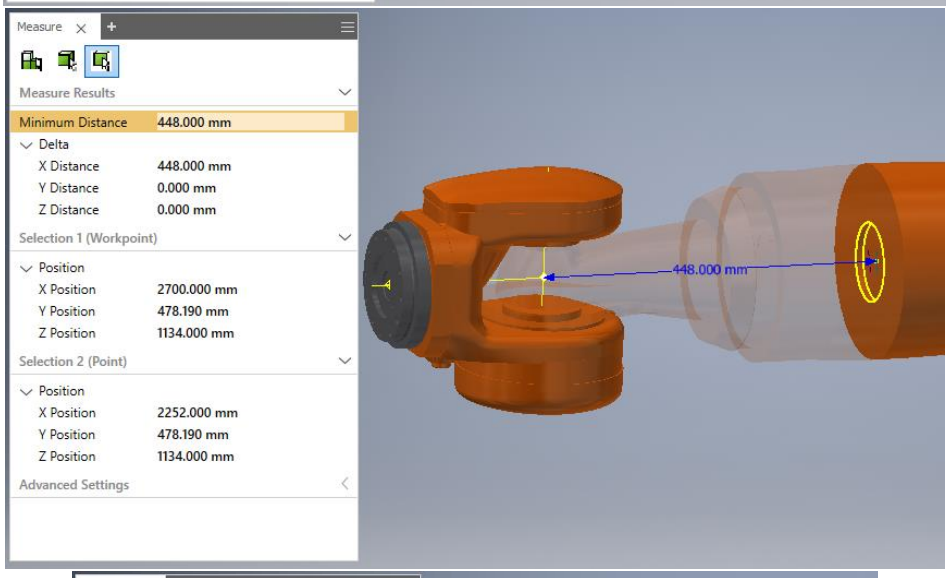


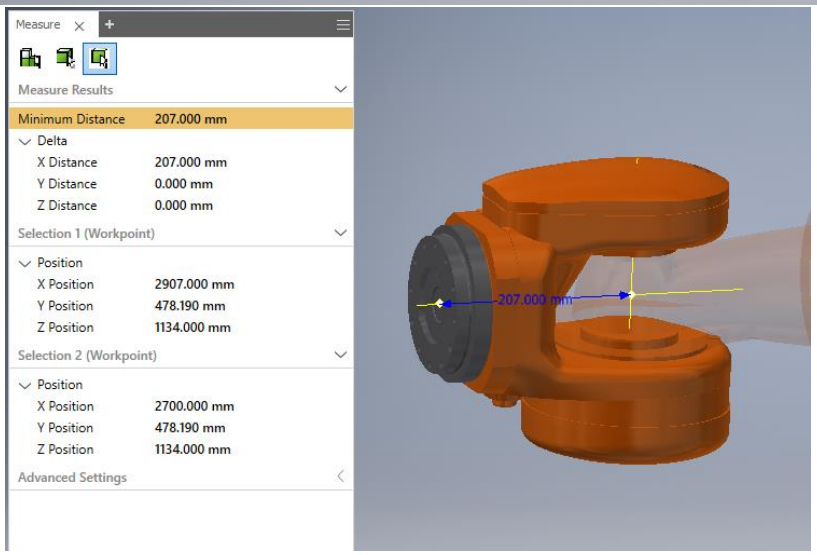

Measure
Measure Results
Minimum Distance 215.600 mm
Delta
X Distance 0.000 mm
Y Distance 0.000 mm
Z Distance 215.600 mm
Selection 1 (Workpoint)
Position
X Position 0.000 mm
Y Position 478.190 mm
Z Position 500.000 mm
Selection 2 (Point)
Position
X Position 0.000 mm
Y Position 478.190 mm
Z Position 715.600 mm
Advanced Settings
215.600 mm


Measure
Measure Results
Minimum Distance 577.623 mm
Delta
X Distance 350.000 mm
Y Distance 10.000 mm
Z Distance 459.400 mm
Selection 1 (Point)
Position
X Position 350.000 mm
Y Position 488.190 mm
Z Position 1175.000 mm
Selection 2 (Point)
Position
X Position 0.000 mm
Y Position 478.190 mm
Z Position 715.600 mm
Advanced Settings
Y: 10.000 mm
Z: 459.400 mm


Measure
Measure Results
Minimum Distance 1163.694 mm
Delta
X Distance 1150.000 mm
Y Distance 178.000 mm
Z Distance 0.000 mm
Selection 1 (Point)
Position
X Position 350.000 mm
Y Position 488.190 mm
Z Position 1175.000 mm
Selection 2 (Point)
Position
X Position 1500.000 mm
Y Position 310.190 mm
Z Position 1175.000 mm
Advanced Settings
X: 1150.000 mm


Measure
Measure Results
Minimum Distance 771.628 mm
Delta
X Distance 752.000 mm
Y Distance 168.000 mm
Z Distance 41.000 mm
Selection 1 (Point)
Position
X Position 2252.000 mm
Y Position 478.190 mm
Z Position 1134.000 mm
Selection 2 (Point)
Position
X Position 1500.000 mm
Y Position 310.190 mm
Z Position 1175.000 mm
Advanced Settings
X: 752.000 mm


Measure
Measure Results
Minimum Distance 448.000 mm
Delta
X Distance 448.000 mm
Y Distance 0.000 mm
Z Distance 0.000 mm
Selection 1 (Workpoint)
Position
X Position 2700.000 mm
Y Position 478.190 mm
Z Position 1134.000 mm
Selection 2 (Point)
Position
X Position 2252.000 mm
Y Position 478.190 mm
Z Position 1134.000 mm
Advanced Settings
448.000 mm


Measure
Measure Results
Minimum Distance 207.000 mm
Delta
X Distance 207.000 mm
Y Distance 0.000 mm
Z Distance 0.000 mm
Selection 1 (Workpoint)
Position
X Position 2907.000 mm
Y Position 478.190 mm
Z Position 1134.000 mm
Selection 2 (Workpoint)
Position
X Position 2700.000 mm
Y Position 478.190 mm
Z Position 1134.000 mm
Advanced Settings
207.000 mm

DH Parameters

A classic approach to decompose the complex robotic arm transformations is to limit them along two axis, x and z-axis. Although this approach is very archaic and contributes to a lot of confusion due to multiple ways it can be employed, it is still of preference in the academics due to its comparative simplicity in describing the kinematics.

The DH parameters define the robotic arm's configuration and are dependent mainly on the joints. Given the robotic arm's dimensions above, the following skeleton of joints can be produced.

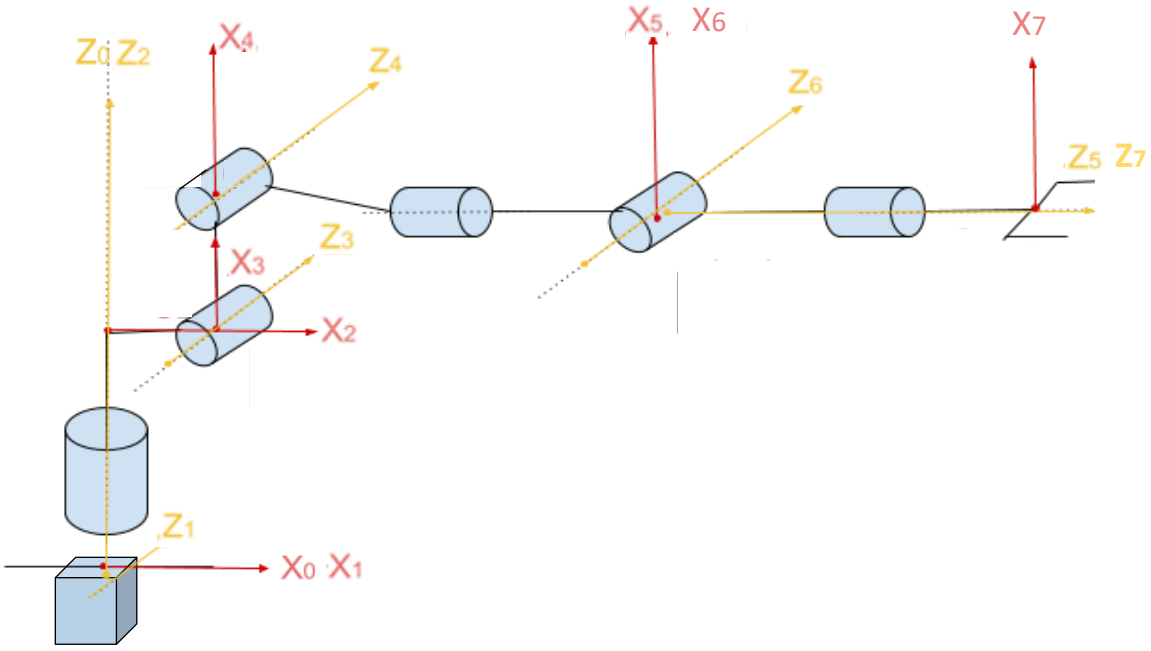


Figure 2 DH Parameters of the Robotic arm

Link	θ_i	$\alpha_{i-1}\ (rad)$	$a_{i-1}\ (m)$	$d_i\ (m)$
1	0	$-\pi/2$	0	d_1
2	θ_2	$\pi/2$	0	0.7156
3	$\theta_3 - \pi/2$	$-\pi/2$	0.35	0
4	θ_4	0	1.109	0
5	θ_5	$-\pi/2$	0	1.2
6	θ_6	$\pi/2$	0	0
7	θ_7	$-\pi/2$	0	0.207

Table 3 DH Parameters of the Robotic arm

Forward Kinematics

Computing forward kinematics from the DH parameters requires multiplication of Homogenous transformation matrix for each link together in the sequence of joints from the base to the end effector. Here modified DH parameters are used which have the following form:

$$ {}^{n-1}T_n = \mathrm{Rot}_{x_{n-1}}(\alpha_{n-1}) \cdot \mathrm{Trans}_{x_{n-1}}(a_{n-1}) \cdot \mathrm{Rot}_{z_n}(\theta_n) \cdot \mathrm{Trans}_{z_n}(d_n) $$

$$ {}^{n-1}T_n = \left[\begin{array}{cccc} \cos\theta_n & -\sin\theta_n & 0 & a_{n-1} \\ \sin\theta_n\cos\alpha_{n-1} & \cos\theta_n\cos\alpha_{n-1} & -\sin\alpha_{n-1} & -d_n\sin\alpha_{n-1} \\ \sin\theta_n\sin\alpha_{n-1} & \cos\theta_n\sin\alpha_{n-1} & \cos\alpha_{n-1} & d_n\cos\alpha_{n-1} \\ \hline 0 & 0 & 0 & 1 \end{array}\right] $$

Since this involves numerous matrix multiplications, it is wiser to write a computer program that does these multiplications and provides us with the results. The following script is written in MATLAB:

```
syms Q1 Q2 Q3 Q4 Q5 Q6 Q7 %Define Symbolic Variables
n=7; % Number of Joints
dhparam = [ Q1      0     0       -pi/2
            0.7156  Q2    0       pi/2
            0       Q3    0.35    -pi/2
            0       Q4    1.15    0
            1.2     Q5    -0.041  -pi/2
            0       Q6    0       pi/2
            -0.207  Q7    0       -pi/2];

  for i=1:n
      T(:,:,i)=[ cos(dhparam(i,2)) -sin(dhparam(i,2))*cos(dhparam(i,4))
sin(dhparam(i,2))*cos(dhparam(i,4))  dhparam(i,3)*cos(dhparam(i,2));
                 sin(dhparam(i,2)) cos(dhparam(i,2))*cos(dhparam(i,4))  -
cos(dhparam(i,2))*sin(dhparam(i,4)) dhparam(i,3)*sin(dhparam(i,2));
                 0                 sin(dhparam(i,4))
cos(dhparam(i,4))                    dhparam(i,1)                    ;
                 0                 0                                      0
1;                          ];
  End

A1=T(:,:,1);
A2=A1*T(:,:,2);
A3=A2*T(:,:,3);
A4=A3*T(:,:,4);
A5=A4*T(:,:,5);
A6=A5*T(:,:,6);
A7=A6*T(:,:,7);
```

$$T_6^0 = \begin{bmatrix} a11 & a12 & a13 & a14 \\ a21 & a22 & a23 & a24 \\ a31 & a32 & a33 & a34 \\ a41 & a42 & a43 & a44 \end{bmatrix}$$

$$a11 = cos(Q7) * (cos(Q6) * (cos(Q2) * cos(Q3) * cos(Q4) * cos(Q5) - cos(Q2) * cos(Q3) * sin(Q4) * sin(Q5)) - cos(Q2) * cos(Q3) * sin(Q4) * sin(Q6)) - cos(Q2) * cos(Q3) * cos(Q5) * sin(Q4) * sin(Q7)$$

$$a21 = -cos(Q7) * (sin(Q6) * (cos(Q3) + sin(Q3) * sin(Q4)) - cos(Q6) * (cos(Q4) * cos(Q5) * sin(Q3) - sin(Q3) * sin(Q4) * sin(Q5))) - cos(Q5) * sin(Q3) * sin(Q4) * sin(Q7)$$

$$a31 = -cos(Q7) * (cos(Q6) * (cos(Q5) * (cos(Q2) * sin(Q4) + cos(Q3) * cos(Q4) * sin(Q2)) + sin(Q5) * (cos(Q2) * cos(Q4) - cos(Q3) * sin(Q2) * sin(Q4))) - cos(Q3) * sin(Q2) * sin(Q4) * sin(Q6)) - cos(Q5) * sin(Q7) * (cos(Q2) * cos(Q4) - cos(Q3) * sin(Q2) * sin(Q4))$$

$$a41 = 0$$

$$a12 = -cos(Q2) * cos(Q3) * cos(Q6) * sin(Q4)$$

$$a22 = -cos(Q6) * (cos(Q3) + sin(Q3) * sin(Q4))$$

$$a32 = cos(Q3) * cos(Q6) * sin(Q2) * sin(Q4)$$

$$a42 = 0$$

$$a13 = -cos(Q2) * cos(Q3) * cos(Q5) * cos(Q7) * sin(Q4)$$

$$a23 = -cos(Q5) * cos(Q7) * sin(Q3) * sin(Q4)$$

$$a33 = -cos(Q5) * cos(Q7) * (cos(Q2) * cos(Q4) - cos(Q3) * sin(Q2) * sin(Q4))$$

$$a44 = 0$$

$$a14 = (cos(Q2) * cos(Q3) * (1150 * cos(Q4) + 1200 * sin(Q4) - 41 * cos(Q4) * cos(Q5) - 207 * cos(Q6) * sin(Q4) + 41 * sin(Q4) * sin(Q5) + 350))/1000$$

$$a24 = (6 * cos(Q3))/5 + (7 * sin(Q3))/20 - (207 * cos(Q6) * (cos(Q3) + sin(Q3) * sin(Q4)))/1000 + (23 * cos(Q4) * sin(Q3))/20 + (6 * sin(Q3) * sin(Q4))/5 - (41 * cos(Q4) * cos(Q5) * sin(Q3))/1000 + (41 * sin(Q3) * sin(Q4) * sin(Q5))/1000 + 1789/2500$$

$$a34 = Q1 - (7 * cos(Q3) * sin(Q2))/20 - (23 * cos(Q2) * sin(Q4))/20 + (41 * cos(Q5) * (cos(Q2) * sin(Q4) + cos(Q3) * cos(Q4) * sin(Q2)))/1000 + (41 * sin(Q5) * (cos(Q2) * cos(Q4) - cos(Q3) * sin(Q2) * sin(Q4)))/1000 - (23 * cos(Q3) * cos(Q4) * sin(Q2))/20 - (6 * cos(Q3) * sin(Q2) * sin(Q4))/5 + (207 * cos(Q3) * cos(Q6) * sin(Q2) * sin(Q4))/1000$$

$$a44 = 1$$

Jacobian

The Jacobian is a $6 \times n$ matrix that relates the joint velocities of the robot to the cartesian velocities of the end effector. This matrix is generally of full rank when the robot has 6 joints. In our case, due to presence of 7 Joints the robotic arm is redundant and hence the Jacobian doesn't have a unique inverse. The Jacobian is constructed using previously derived transformation matrices as follows:

$$J = [J_1 J_2 \cdots J_n]$$

where the i-th column J_i is given by

$$J_i = \begin{bmatrix} z_{i-1} \times (o_n - o_{i-1}) \\ z_{i-1} \end{bmatrix}$$

if joint i is revolute and

$$J_i = \begin{bmatrix} z_{i-1} \\ 0 \end{bmatrix}$$

The generalized MATLAB code to find the Jacobian, given our previous variables is a as follows:

```
A0=[1 0 0 0;0 1 0 0;0 0 1 0; 0 0 0 1];
Trf={A0,A1,A2,A3,A4,A5,A6,A7};
J=[0; 0; 1; 0; 0; 0]; % 1st Prismatic Joint

for i=2:7
    mat=Trf{1,i};
    On=A7(1:3,4);
    On1=mat(1:3,4);
    Zn1=mat(1:3,3);
    J=horzcat(J,([cross(Zn1,(On-On1));Zn1])); % Remaining Revolute Joints
End
```

Since the Jacobian computed using the above algorithm is extremely large and it serves no useful purpose of representing it in this representing, it is stated here in pictorial form directly from MATLAB, just for the sake of completeness. It can be generated by evaluating lines of code given previously.

$$\begin{pmatrix}
0 & \sigma_1 - \sigma_{10} - \frac{7\cos(Q_3)\sin(Q_2)}{20} + \sigma_{13} - \sigma_9 - \sigma_7 + \sigma_5 & -\cos(Q_2)\left(\sigma_{18} + \frac{7\sin(Q_3)}{20} - \sigma_{17} + \frac{23\cos(Q_4)\sin(Q_3)}{20} + \sigma_{16} - \sigma_{15} + \sigma_{14}\right) & -\cos(Q_3)\ (\sigma_{10} - \sigma_1 - \sigma_{13} + \sigma_9 + \sigma_7 - \sigma_5) \\
0 & 0 & \cos(Q_2)\ (\sigma_{11} + \sigma_8 + \sigma_6 - \sigma_4 - \sigma_3 + \sigma_2) & 0 \\
1 & \sigma_4 - \sigma_8 - \sigma_6 - \sigma_{11} + \sigma_3 - \sigma_2 & 0 & -\cos(Q_3)\ (\sigma_8 + \sigma_6 - \sigma_4 - \sigma_3 + \sigma_2) \\
0 & 0 & 0 & 0 \\
0 & 1 & 0 & \cos(Q_3) \\
0 & 0 & \cos(Q_2) & 0
\end{pmatrix}$$

$$\begin{pmatrix}
\sigma_{20}\ (\sigma_1 + \sigma_{13} - \sigma_7 + \sigma_5) + \cos(Q_3)\sin(Q_2)\sin(Q_4)\,\sigma_{12} & -\frac{207\cos(Q_5)\cos(Q_6)\,\sigma_{19}\,\sigma_{20}}{1000} - \frac{207\cos(Q_3)\cos(Q_5)\cos(Q_6)\sin(Q_2)\sin(Q_3)\sin(Q_4)^2}{1000} & 0 \\
-\cos(Q_3)\sin(Q_2)\sin(Q_4)\ (\sigma_6 - \sigma_4 - \sigma_3 + \sigma_2) - \cos(Q_2)\cos(Q_3)\sin(Q_4)\ (\sigma_1 + \sigma_{13} - \sigma_7 + \sigma_5) & \frac{207\cos(Q_2)\cos(Q_3)^2\cos(Q_5)\cos(Q_6)\sin(Q_2)\sin(Q_4)^2}{1000} + \frac{207\cos(Q_2)\cos(Q_3)\cos(Q_5)\cos(Q_6)\sin(Q_4)\,\sigma_{19}}{1000} & 0 \\
\cos(Q_2)\cos(Q_3)\sin(Q_4)\,\sigma_{12} - \sigma_{20}\ (\sigma_6 - \sigma_4 - \sigma_3 + \sigma_2) & \frac{207\cos(Q_2)\cos(Q_3)\cos(Q_5)\cos(Q_6)\sin(Q_4)\,\sigma_{20}}{1000} - \frac{207\cos(Q_2)\cos(Q_3)\cos(Q_5)\cos(Q_6)\sin(Q_3)\sin(Q_4)^2}{1000} & 0 \\
\cos(Q_2)\cos(Q_3)\sin(Q_4) & -\cos(Q_2)\cos(Q_3)\cos(Q_5)\sin(Q_4) & \cos(Q_2)\cos(Q_3)\cos(Q_6)\sin(Q_4) \\
\sigma_{20} & -\cos(Q_5)\sin(Q_3)\sin(Q_4) & \cos(Q_6)\,\sigma_{20} \\
-\sigma_{21} & -\cos(Q_5)\,\sigma_{19} & -\cos(Q_3)\cos(Q_6)\sin(Q_2)\sin(Q_4)
\end{pmatrix}$$

where

$$\sigma_1 = \frac{41\cos(Q_5)\ (\cos(Q_2)\sin(Q_4) + \cos(Q_3)\cos(Q_4)\sin(Q_2))}{1000}$$

$$\sigma_2 = \frac{41\cos(Q_2)\cos(Q_3)\sin(Q_4)\sin(Q_5)}{1000}$$

$$\sigma_3 = \frac{207\cos(Q_2)\cos(Q_3)\cos(Q_6)\sin(Q_4)}{1000}$$

$$\sigma_4 = \frac{41\cos(Q_2)\cos(Q_3)\cos(Q_4)\cos(Q_5)}{1000}$$

$$\sigma_5 = \frac{207\cos(Q_3)\cos(Q_6)\sin(Q_2)\sin(Q_4)}{1000}$$

$$\sigma_6 = \frac{6\cos(Q_2)\cos(Q_3)\sin(Q_4)}{5}$$

$$\sigma_7 = \frac{6\cos(Q_3)\sin(Q_2)\sin(Q_4)}{5}$$

$$\sigma_8 = \frac{23\cos(Q_2)\cos(Q_3)\cos(Q_4)}{20}$$

$$\sigma_9 = \frac{23\cos(Q_3)\cos(Q_4)\sin(Q_2)}{20}$$

$$\sigma_{10} = \frac{23\cos(Q_2)\sin(Q_4)}{20}$$

$$\sigma_{11} = \frac{7\cos(Q_2)\cos(Q_3)}{20}$$

$$\sigma_{12} = \sigma_{18} - \sigma_{17} + \sigma_{16} - \sigma_{15} + \sigma_{14}$$

$$\sigma_{13} = \frac{41\sin(Q_5)\,\sigma_{19}}{1000}$$

$$\sigma_{14} = \frac{41\sin(Q_3)\sin(Q_4)\sin(Q_5)}{1000}$$

$$\sigma_{15} = \frac{41\cos(Q_4)\cos(Q_5)\sin(Q_3)}{1000}$$

$$\sigma_{16} = \frac{6\sin(Q_3)\sin(Q_4)}{5}$$

$$\sigma_{17} = \frac{207\cos(Q_6)\,\sigma_{20}}{1000}$$

$$\sigma_{18} = \frac{6\cos(Q_3)}{5}$$

$$\sigma_{19} = \cos(Q_2)\cos(Q_4) - \sigma_{21}$$

$$\sigma_{20} = \cos(Q_3) + \sin(Q_3)\sin(Q_4)$$

$$\sigma_{21} = \cos(Q_3)\sin(Q_2)\sin(Q_4)$$

Iterative Solver Approach

An alternative approach to solving the inverse kinematics for complex end effector position and orientation is to employ a more iterative algorithmic approach.

In mathematics and computing, the Levenberg–Marquardt algorithm (LMA or just LM), also known as the damped least-squares (DLS) method, is used to solve non-linear least squares problems. This algorithm and supporting features are readily available in MATLAB environment. We will make use of this algorithm to find joint angles for inverse kinematics.

Constructing Rigid Body Tree in Robotics Toolbox (MATLAB)

The rigid body tree model is a representation of a robot structure. It can be used to represent robotic manipulators or other kinematic trees.

Base of the Rigid Body Tree defines the world coordinate frame and is the first attachment point for a rigid body. A Rigid Body is the basic building block of rigid body tree model also called a link, represents a solid body that cannot deform. The distance between any two points on a single rigid body remains constant. Each rigid body has one joint, which defines the motion of that rigid body relative to its parent. It is the attachment point that connects two rigid bodies in a robot model.

We can define the relevant objects in a MATLAB script as follows:

```
% Define Robot Object
KUKA = robotics.RigidBodyTree('MaxNumBodies',7);
% Define Body and Joint Objects
body1 = robotics.RigidBody('body1');
jnt1 = robotics.Joint('jnt1','prismatic');
body2 = robotics.RigidBody('body2');
jnt2 = robotics.Joint('jnt2','revolute');
body3 = robotics.RigidBody('body3');
jnt3 = robotics.Joint('jnt3','revolute');
body4 = robotics.RigidBody('body4');
jnt4 = robotics.Joint('jnt4','revolute');
body5 = robotics.RigidBody('body5');
jnt5 = robotics.Joint('jnt5','revolute');
body6 = robotics.RigidBody('body6');
jnt6 = robotics.Joint('jnt6','revolute');
body7 = robotics.RigidBody('body7');
jnt7 = robotics.Joint('jnt7','revolute');
```

Transformations between joints can be represented by simple Homogenous transformation matrices. When using the Robotics Toolbox, it is not necessary to follow the DH convention. We can use any number of homogenous transformations as required. This can easily be done using MATLAB's in-built function 'makehgtform' to compute the matrices accordingly:

```
% Joint Transformations
% Move Z 500 Rotate Y90
tform=makehgtform('translate',[0 0 0.5],'yrotate',pi/2,'xrotate',-pi/2);
setFixedTransform(jnt1,tform);
% Move X -215.6 Rotate Y-90
tform=makehgtform('translate',[-0.2156 0 0],'yrotate',-pi/2);
setFixedTransform(jnt2,tform);
%Move X -675+50 Y 350 Rotate Z90 Y180
tform=makehgtform('translate',[0 -0.35 0.4594],'yrotate',pi/2,'zrotate',-
pi/2);
setFixedTransform(jnt3,tform);
%Move X 1150
tform=makehgtform('translate',[1.15 0 0]);
setFixedTransform(jnt4,tform);
%Move X 762 Y 41 Rotate Y90
tform=makehgtform('translate',[0.762 0.041 0],'yrotate',pi/2);
setFixedTransform(jnt5,tform);
%Move Z 438 Rotate X90
tform=makehgtform('translate',[0 0 0.438],'xrotate',pi/2);
setFixedTransform(jnt6,tform);
%Move Y -207
tform=makehgtform('translate',[0 0.207 0],'xrotate',pi/2);
setFixedTransform(jnt7,tform);
```

To interconnect the these above created objects, we can invoke the following commands:

```
body1.Joint = jnt1;
body2.Joint = jnt2;
body3.Joint = jnt3;
body4.Joint = jnt4;
body5.Joint = jnt5;
body6.Joint = jnt6;
body7.Joint = jnt7;

addBody(KUKA,body1,'base')
addBody(KUKA,body2,'body1')
addBody(KUKA,body3,'body2')
addBody(KUKA,body4,'body3')
addBody(KUKA,body5,'body4')
addBody(KUKA,body6,'body5')
addBody(KUKA,body7,'body6')
```

We can visualize the joint arrange and frame position to confirm the configuration of the constructed model using the following command:

```
show(KUKA,KUKA.homeConfiguration)
```

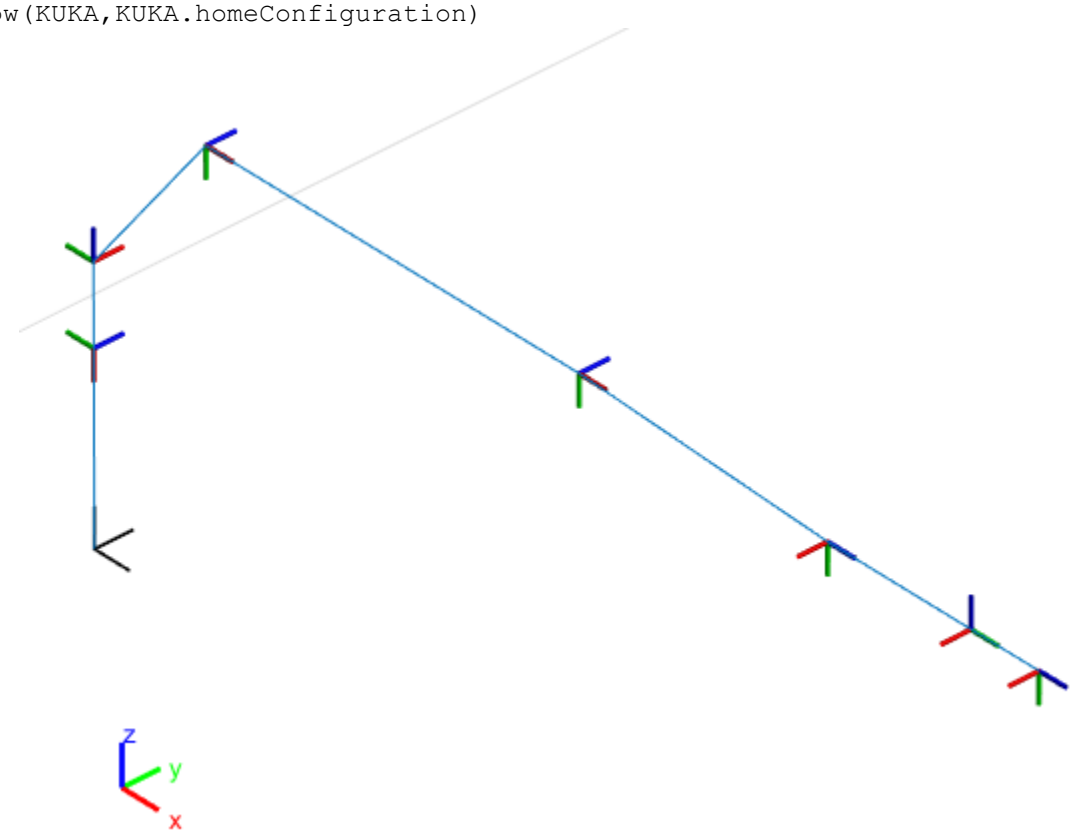


Figure 4 Rigid Body Tree Model of the Robotic Arm

For any given configuration, the Jacobian of arm for that configuration can be calculated using the following command:

```
Jacobian = geometricJacobian(KUKA,KUKA.homeConfiguration,'EndEffector')
```

For Home position illustrated above, the Jacobian is given as:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & -0.041 & -0.41 & 0 & 0 & 0 \\ 1 & 2.907 & 0 & 0 & 0 & 0.207 & 0 \\ 0 & 0 & -2.557 & -1.407 & 0 & 0 & 0 \end{bmatrix}$$

Levenberg-Marquardt Iterative Inverse Kinematics Solver

Levenberg-Marquardt is a popular alternative to the Gauss-Newton method of finding the minimum of a function $F(x)$ that is a sum of squares of nonlinear functions,

$$F(x) = \frac{1}{2}\sum_{i=1}^{m}[f_i(x)]^2.$$

Let the Jacobian of $f_i(x)$ be denoted $J_i(x)$, then the Levenberg-Marquardt method searches in the direction given by the solution $\mathbf{p}$ to the equations

$$\left(J_k^{\mathrm{T}} J_k + \lambda_k \mathrm{I}\right) p_k = -J_k^{\mathrm{T}} f_k,$$

where λ_k are nonnegative scalars and I is the identity matrix. The method has the nice property that, for some scalar Δ related to λ_k, the vector p_k is the solution of the constrained subproblem of minimizing $\|J_k\, p + f_k\|_2^2/2$ subject to $\|p\|_2 \le \Delta$ (Gill et al. 1981, p. 136).

MATLAB's Robotics Toolbox has two primary solvers for finding the inverse joint angles given a set of constraints. These are the 'Levenberg-Marquardt' and 'BFGSGradientProjection' methods. These implementations also allow for numerous constraints on the end effector position and intermediate trajectory coordinates while finding a solution.

Generally, the inverse kinematic solver for a modelled robot can be defined as in the following manner:

```
gik = robotics.GeneralizedInverseKinematics('RigidBodyTree', KUKA, ...
'ConstraintInputs',{'cartesian','position','aiming','orientation','joint'}...
    'SolverAlgorithm','LevenbergMarquardt')
```

Solver Parameters

The algorithm also has some specific tunable parameters to improve solutions. These parameters can be specified in the 'Solver Parameters' property of the object. A brief description of the parameters follows:

MaxIterations — Maximum number of iterations allowed.

MaxTime — Maximum number of seconds that the algorithm runs before timing out.

GradientTolerance — Threshold on the gradient of the cost function. The algorithm stops if the magnitude of the gradient falls below this threshold.

SolutionTolerance — Threshold on the magnitude of the error between the end-effector pose generated from the solution and the desired pose.

EnforceJointLimits — Indicator if joint limits are considered in calculating the solution. 'Joint Limits' is a property of the robot model in 'robotics.RigidBodyTree'.

AllowRandomRestarts — Indicator if random restarts are allowed. Random restarts are triggered when the algorithm approaches a solution that does not satisfy the constraints.

StepTolerance — Minimum step size allowed by the solver. Smaller step sizes usually mean that the solution is close to convergence.

ErrorChangeTolerance — Threshold on the change in end-effector pose error between iterations. The algorithm returns if the changes in all elements of the pose error are smaller than this threshold.

DampingBias — A constant term for damping. The LM algorithm has a damping feature controlled by this constant that works with the cost function to control the rate of convergence.

UseErrorDamping — Indicator of whether damping is used.

Set Up Target and Constraints for Solver

The purpose of the following lines of code is to establish a set of constraints and conditions for the final and intermediate configurations of the robot joints and end effector.

Firstly, we define target's cartesian coordinates with respect to origin. In this case we have taken it to be a cup which is convenient to display in a 3D plot:

```
cupHeight = 0.2;
cupRadius = 0.05;
cupPosition = [6, 0, 2]; %Change to place the cup at arbitrary coordinates

gripper = 'body7'; % name the EndEffector body of the Robot Arm

Attach the cup/target to the base of the Rigid Body Tree for consistent world
frame coordinate reference
cup_body = robotics.RigidBody('cupFrame');
setFixedTransform(cup_body.Joint, trvec2tform(cupPosition))
addBody(KUKA, cup_body, 'base');


numWaypoints = 5; %
q0 = homeConfiguration(KUKA);
qWaypoints = repmat(q0, numWaypoints, 1);

% Cartesian Bounds to Limit EndEffector in a given range
heightAboveTable = robotics.CartesianBounds('body6');
heightAboveTable.Bounds = [-inf, inf; ...
                           -inf, inf; ...
                           0,inf];

% Specify the Target Position, Object
distanceFromCup = robotics.PositionTarget('cupFrame');
distanceFromCup.ReferenceBody = gripper;
distanceFromCup.PositionTolerance = 0.005

% Specify alignment of End Effector at Target Position
alignWithCup = robotics.AimingConstraint('body7');
alignWithCup.TargetPoint = [0, 0, 100]

% Limit Movement of the Joints within bounds
limitJointChange = robotics.JointPositionBounds(KUKA)

% Specify Orientation constraints
fixOrientation = robotics.OrientationTarget(gripper);
fixOrientation.OrientationTolerance = deg2rad(1)

intermediateDistance = 0

% Specify weights to emphasize Joint movement priority in finding a solution
limitJointChange.Weights = ones(size(limitJointChange.Weights));
limitJointChange.Weights(2)=0.1
limitJointChange.Weights(3)=0.5
fixOrientation.Weights = 1;
alignWithCup.Weights = 1;
```

```
distanceFromCup.TargetPosition = [0,0,intermediateDistance]

fixOrientation.TargetOrientation = ...
    tform2quat(getTransform(KUKA,qWaypoints(2,:),gripper));

finalDistanceFromCup = 0;
distanceFromCupValues = linspace(intermediateDistance, finalDistanceFromCup,
numWaypoints-1);

maxJointChange = pi;
```

Solve for Initial Guess and Re-Solve for Smooth Trajectory

```
% Define Inverse Kinematic Solver
gik = robotics.GeneralizedInverseKinematics('RigidBodyTree', KUKA, ...

'ConstraintInputs',{'cartesian','position','aiming','orientation','joint'}...
    'SolverAlgorithm','LevenbergMarquardt')

% Solve for n number of waypoints to form a reaching trajectory
[qWaypoints(2, :),solutionInfo] = gik(q0, heightAboveTable, ...
                        distanceFromCup, alignWithCup, fixOrientation, ...
                        limitJointChange)

% Smooth out trajectory by re-solving with computed intermediate constraints
for k = 3:numWaypoints
    % Update the target position.
    distanceFromCup.TargetPosition(3) = distanceFromCupValues(k-1);
    % Restrict the joint positions to lie close to their previous values.
    limitJointChange.Bounds = [qWaypoints(k-1,:)' - maxJointChange,...
                               qWaypoints(k-1, :)' + maxJointChange];
    % Solve for a configuration and add it to the waypoints array.
    [qWaypoints(k, :),solutionInfo] = gik(qWaypoints(k-1, :)' ...
                                          heightAboveTable, ...
                                          distanceFromCup, alignWithCup, ...
                                          fixOrientation, limitJointChange)
end
```

Visualize Animation of Reaching Trajectory

The following code snippet helps visualize the robot skeleton movement while it reaches for the target object. It is an excellent way to observe and validate the working of the model and the solver algorithm.

```
Framerate = 15; % Animation  Speed
r = robotics.Rate(framerate);
tFinal = 30; % Animation Total Time
tWaypoints = [0,linspace(tFinal/2,tFinal,size(qWaypoints,1)-1)];
numFrames = tFinal*framerate;

% Interpolate joint angle waypoints to visualize smoother transition
qInterp = pchip(tWaypoints,qWaypoints,linspace(0,tFinal,numFrames))';

% Highlight EndEffector position in Plot
gripperPosition = zeros(numFrames,3);

for k = 1:numFrames
    gripperPosition(k, :)' = tform2trvec(getTransform(KUKA,qInterp(k, :)...
                                                      gripper));
end

% Show Animation
figure;
show(KUKA, qWaypoints(1, :)', 'PreservePlot', false);
hold on
exampleHelperPlotCupAndTable(cupHeight, cupRadius, cupPosition);
p = plot3(gripperPosition(1,1), gripperPosition(1,2),
gripperPosition(1,3),'r');

hold on
for k = 1:size(qInterp,1)
    axis([-10 10 -10 10 -10 10])
    show(KUKA, qInterp(k, :), 'PreservePlot', false);
    p.Xdata(k) = gripperPosition(k,1);
    p.Ydata(k) = gripperPosition(k,2);
    p.Zdata(k) = gripperPosition(k,3);
    waitfor®;
end
hold off
```

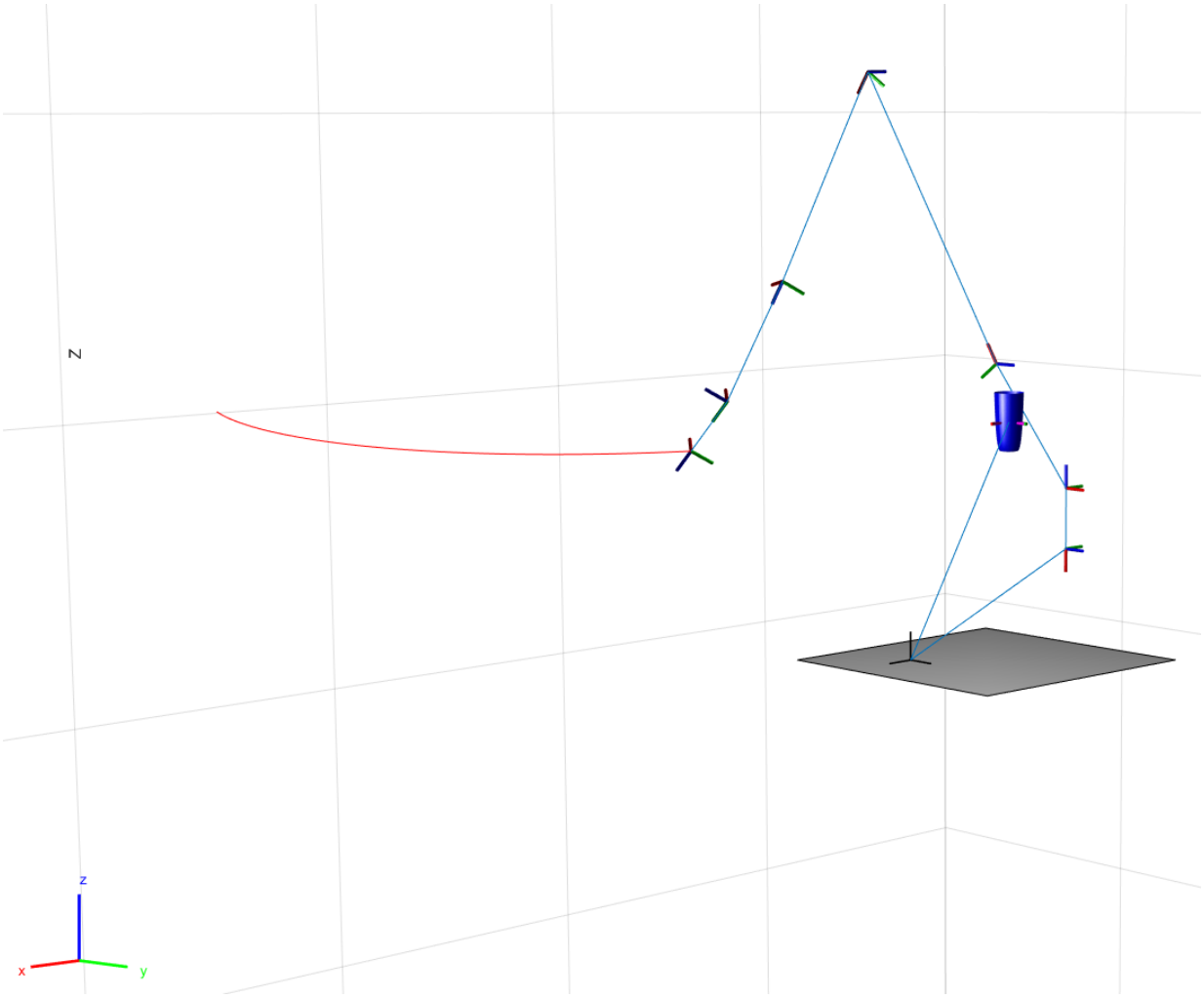


Figure 5 Robotic Arm Trajectory for Target at $[0.5\ 0.5\ 1]^T$

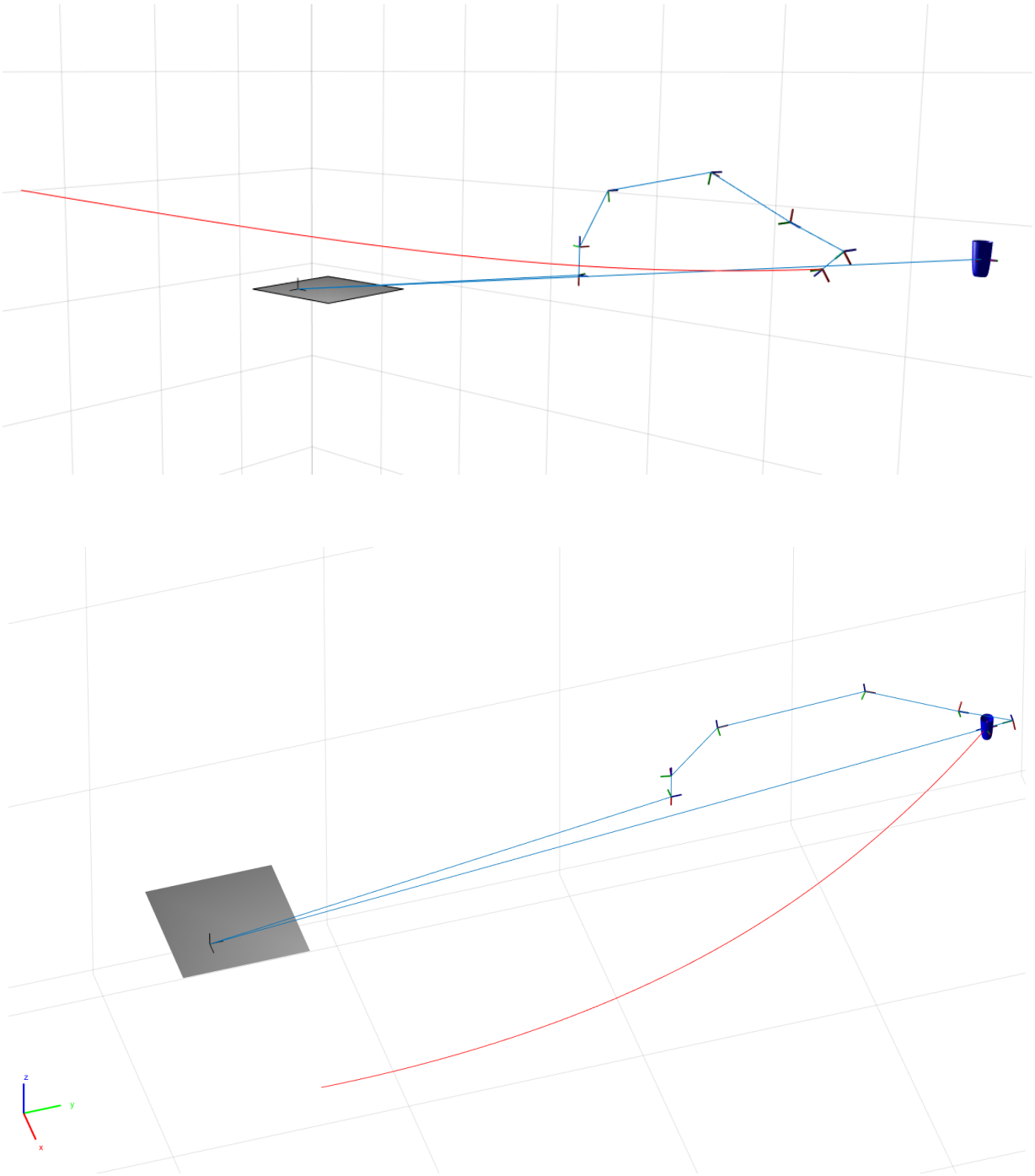


Figure 6 Robotic Arm Trajectory for Target at $[0.5\ 6\ 1]^T$

Simscape Simulation

To further verify the movements of the actual physical system, we model the Robotic Arm in Simscape. This environment preserves and considers the actual physical properties of the bodies such as, mass distribution and Inertia properties.

The most crucial part of modelling in Simscape is to create or import the actual geometry of the system being modelled. Multibody Link Plugin does exactly this job by importing the CAD data from Autodesk Inventor 2018, the CAD software we modelled our Robotic Arm in. This involves exporting your CAD files in .STEP file format.

Simscape provides an elaborate feature set to model relations between imported physical entities. The overview of the modelled 6DOF Arm mounted on Linear Rail system diagram is shown below:

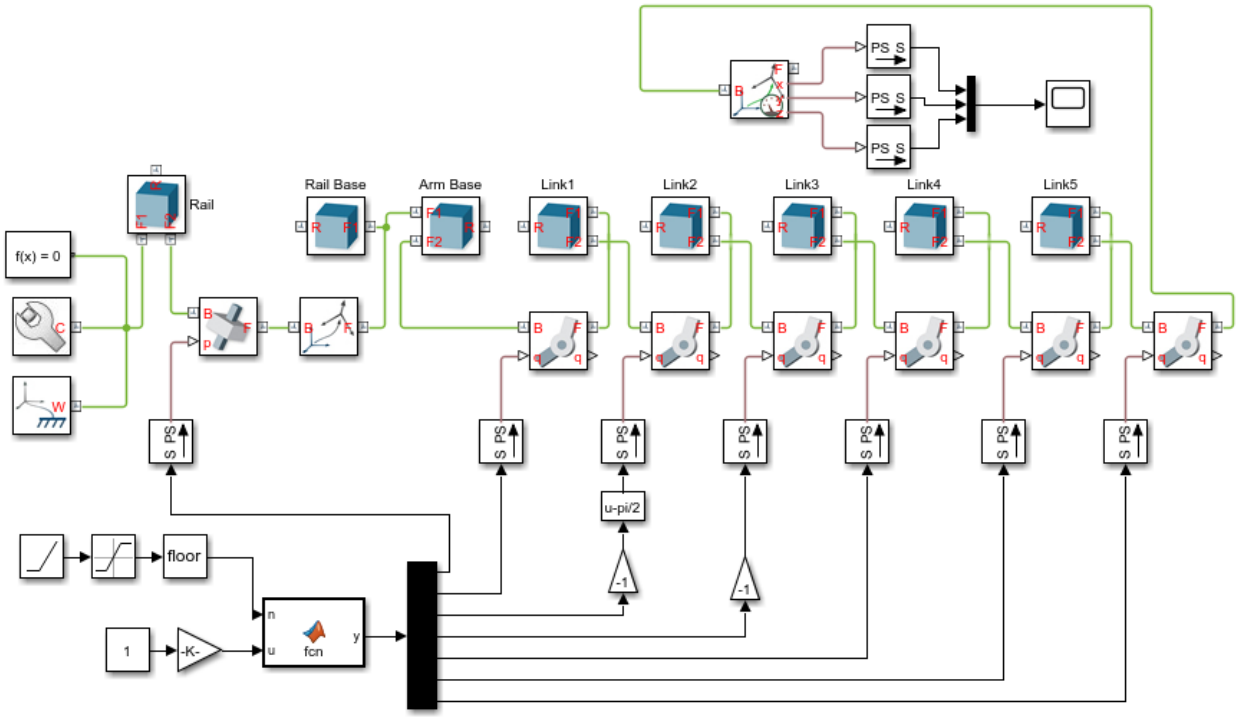
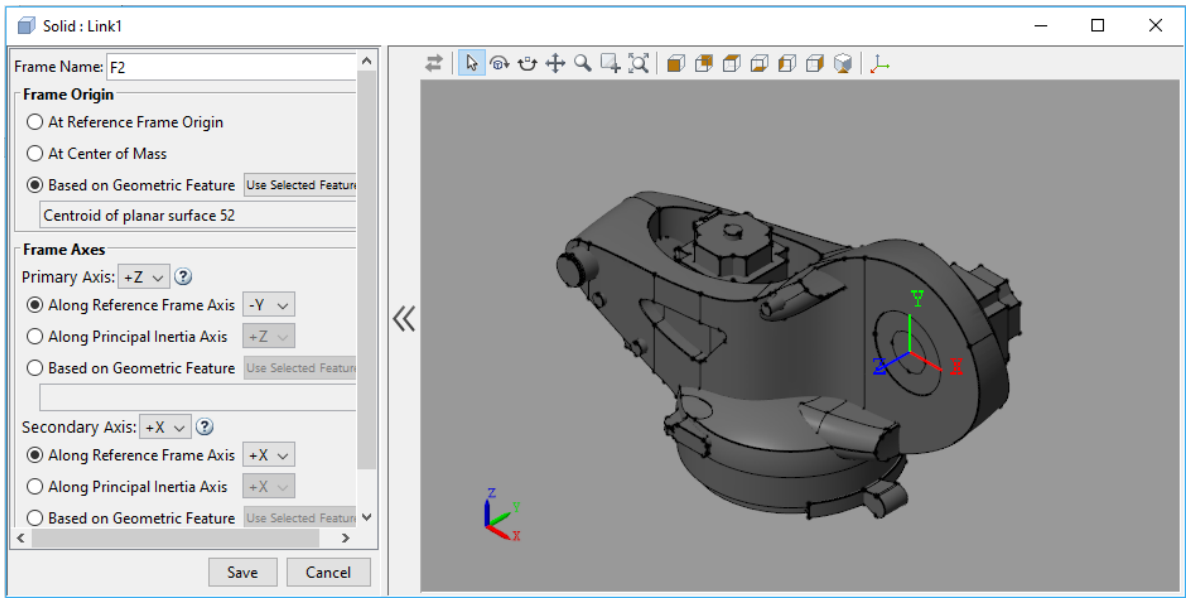


Figure 7 Simscape model of Robotic Arm mounted on Linear Rail

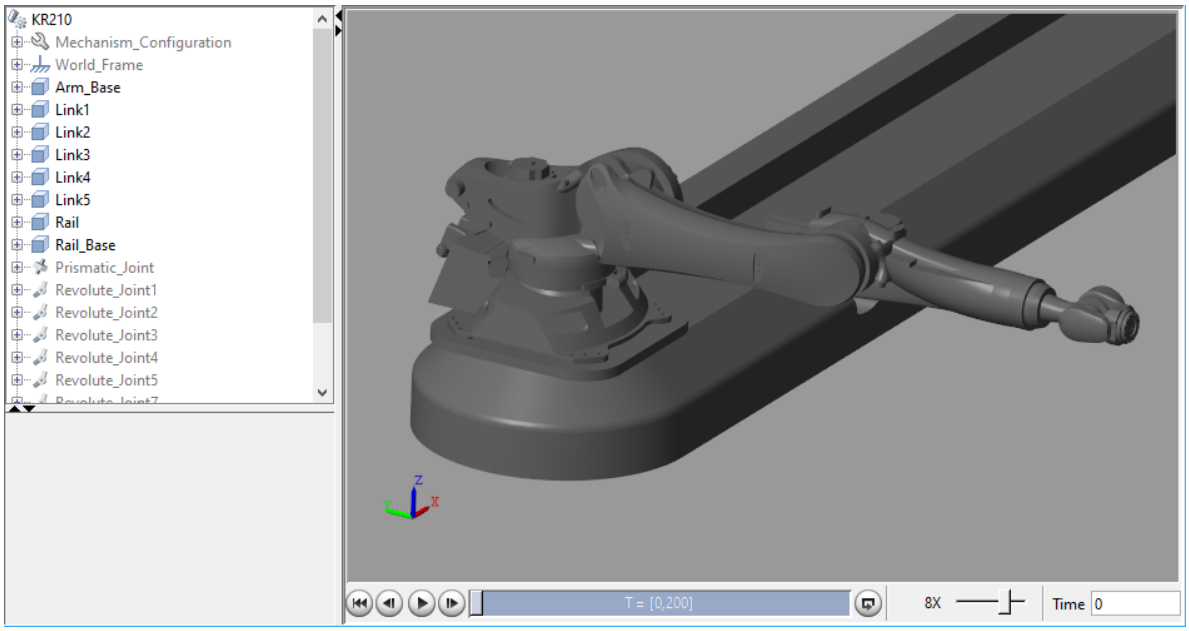
Setup Joint Frames

For systems to properly interact with each other, Simscape requires that each physical entity must be constrained either by a rigid body, transform or a joint. To identify frames on each body, instead of specifying absolute value of the coordinates, we can associate multiple frames to the solid body based on its geometric features. These features are usually the center points of circles representing Joints in the CAD model.



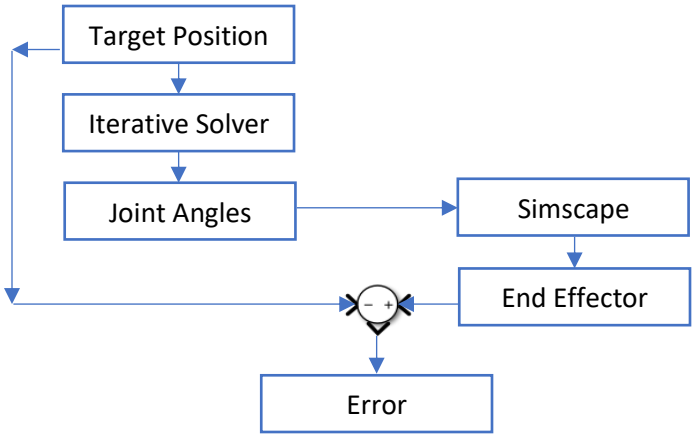
This process of specifying joint frames is repeated on all solid bodies of the robotic arm until there exists an appropriate frame for each Joint.

Then the physical assembly can be constructed by adding appropriate types of joints between each body. Running the simulation presents with a complete physical mode of the system below:



Target Position Validation

Now, we have two models to predict the outcome of input to the system. We can cross check these systems with each other and look for any underlying errors. This can be achieved by supplying random position coordinates, which fall within the work space of the robotic arm, to the generalized Inverse Kinematics solver to predict the joint angles. These Joint angles can be then fed to the Simulink model. In the Simulink model, a reference target body is defined as red blob with the same coordinates as those supplied to the inverse kinematic solver. The robotic arm moves according to the joint angles computed by the solver. Using the 'Transform Sensor' object in Simulink, the error between the original target setpoint and the destination point can be observed.



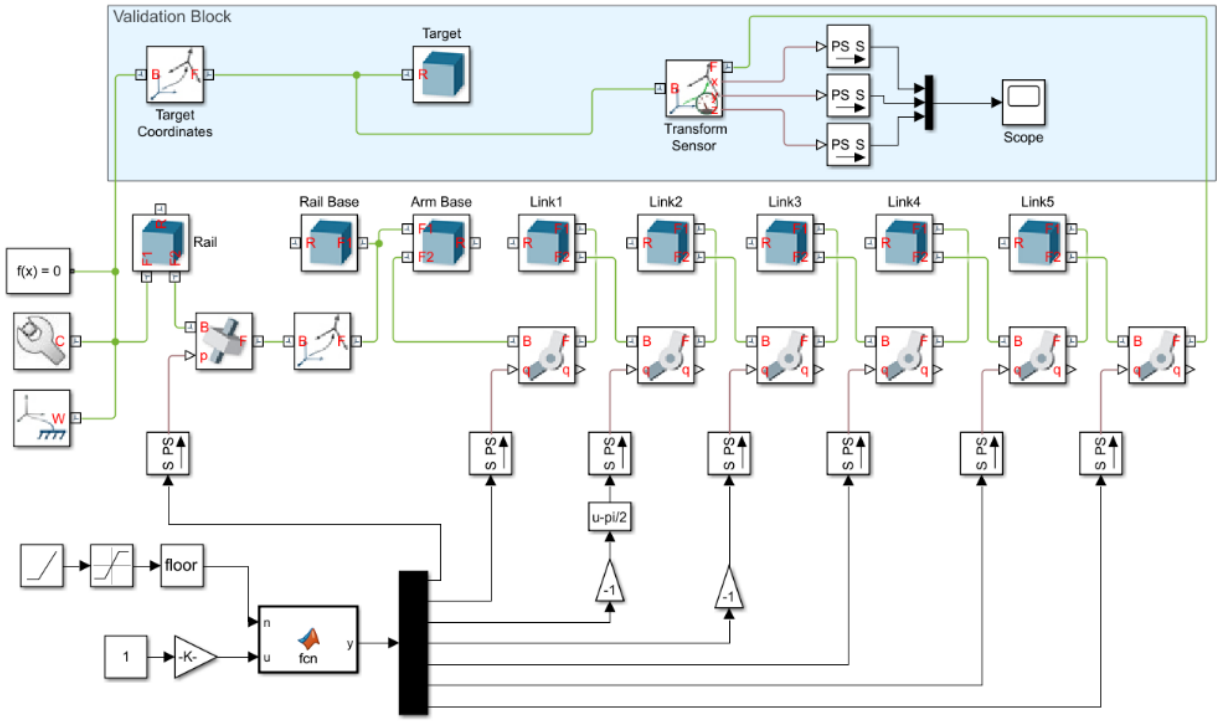


Figure 8 Modified Model for Validation Testing

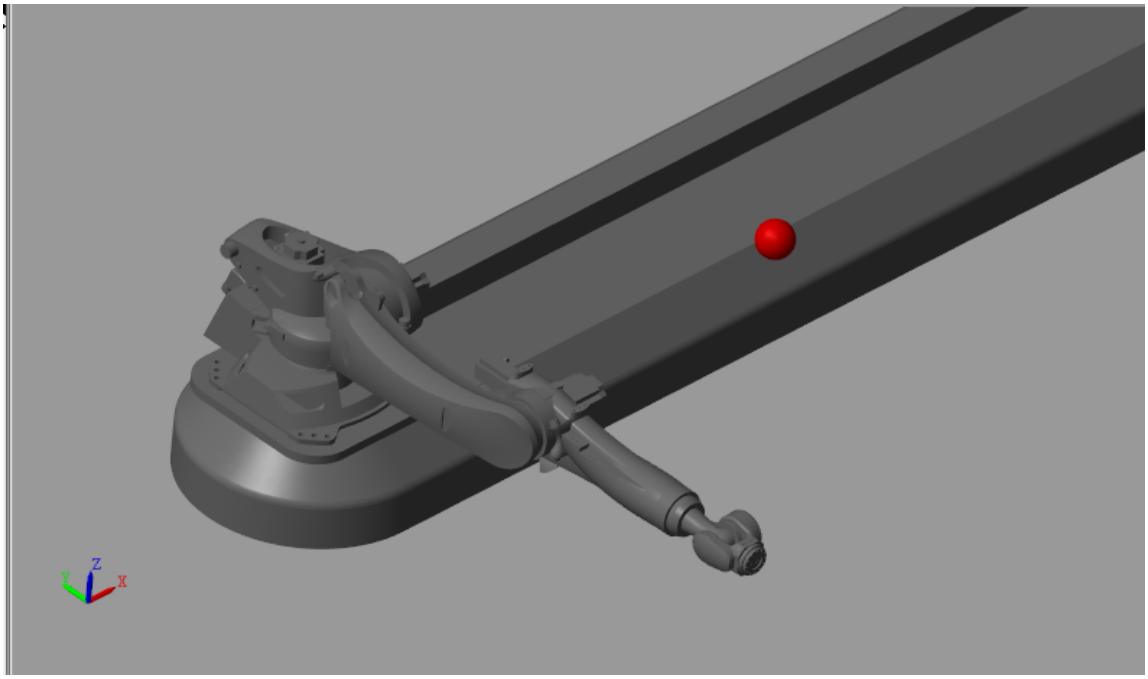


Figure 9 Robotic Arm Home Configuration and Red Target Position $[1\ 2\ 1]^T$

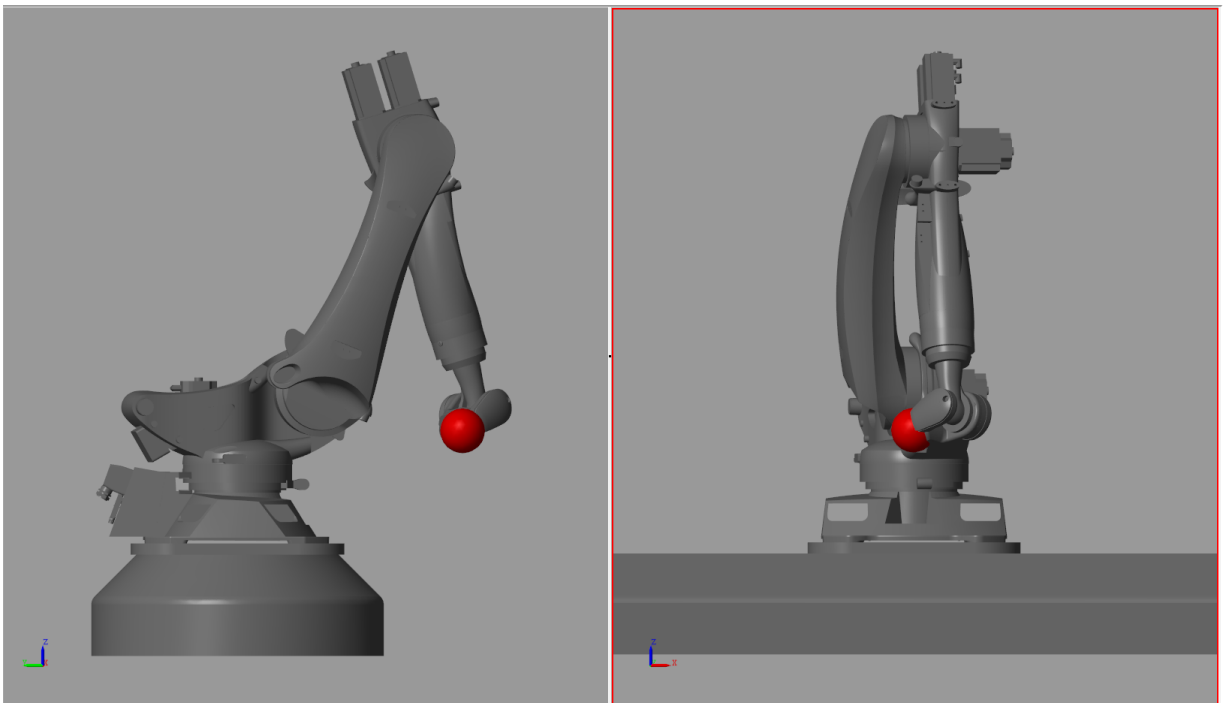


Figure 10 Robotic Arm with End Effector at Target Position $[1\ 2\ 1]^T$

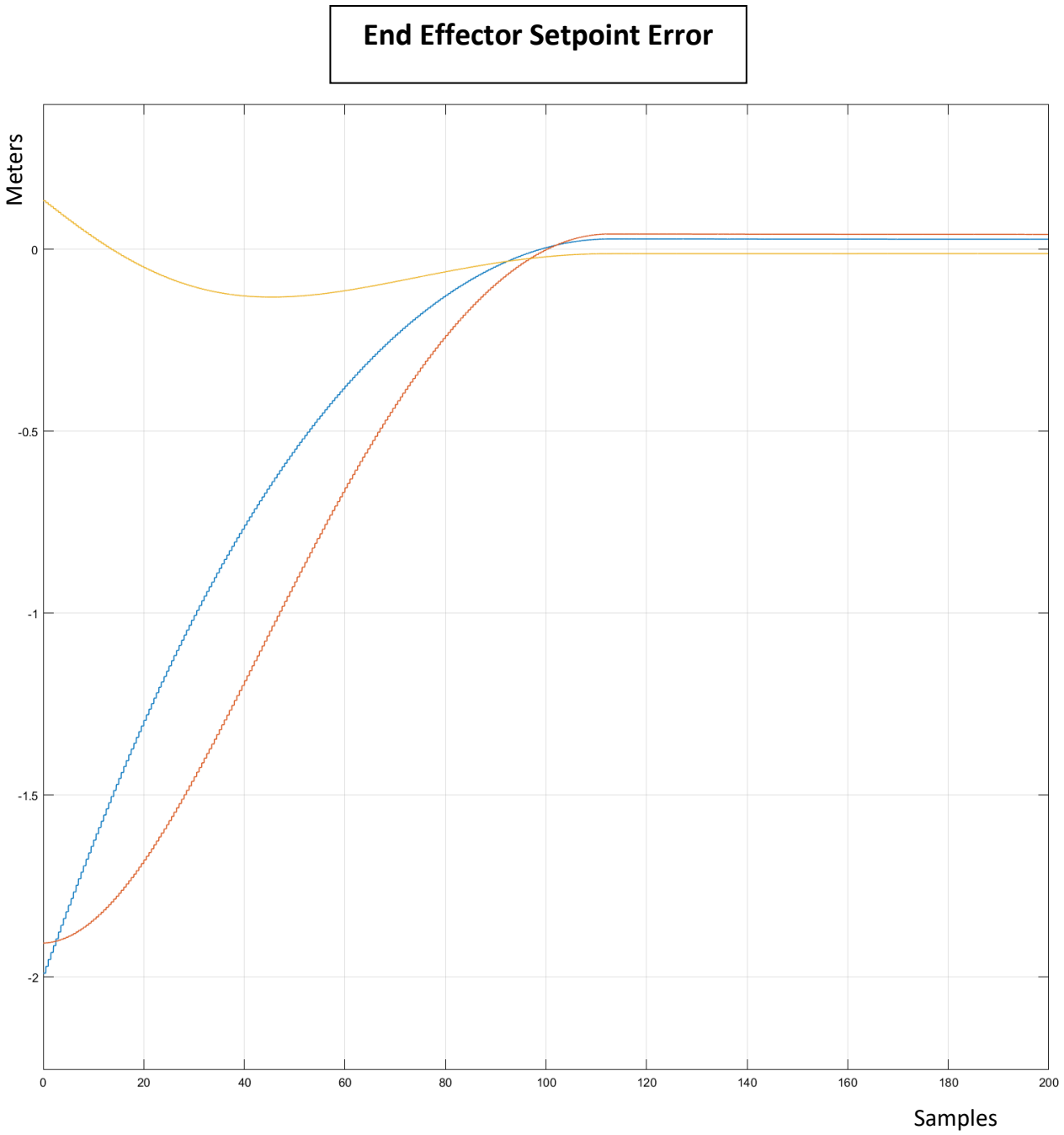


Figure 11 End Effector Distance to Target Position in 3 Axis

As it has been demonstrated that the steady state error falls below the 3% of the total movement, it can be safely assumed that the techniques outlined and applied in this project have been successful. Throughout, the work done has been kept as generalized as possible. Each code script written is complete and will accommodate any number of DH parameters and calculate its Jacobian.

The Rigid Body Tree modelling, inverse kinematic solver will also work for an n-link robotic manipulator.

Assumptions:

For the purpose of simulating the behavior of the arm, following assumptions were made:

- The robotic arm linkages were assumed to be entirely rigid with no flexibility
- The Motor joint friction was assumed to be zero
- Gear backlash and other non-linearity inducing factors were ignored.

Commonly used Industrial robots are actuated by advanced AC Servo motors, but for our case we make use of the permanent magnet geared DC motor. It is also assumed that the electrical time constant of this motor is much smaller as compared to the mechanical time constant.

Future Work

The work so far spanned the formulation and solution of the kinematics of the rail mounted arm. To make this idea a practically feasible one, it is necessary to proceed with the formulation of dynamic control of the arm. Owing to the high number of joints it can be safely said that the way forward in dealing with dynamics of this system is to heavily rely on computational techniques to obtain solutions.

Appendix A

Download Link for MATLAB Simulink and CAD files:

https://drive.google.com/open?id=1VEAQGrHQ9HjyvyML4TsigWOV1vppiP3g

Video Demonstration of Trajectory Tracking

https://youtu.be/TWIaoFdHMj4

Appendix B

Forward Kinematics

```
syms Q1 Q2 Q3 Q4 Q5 Q6 Q7 %Define Symbolic Variables
n=7; % Number of Joints
dhparam = [ Q1      0       0       -pi/2
            0.7156  Q2      0       pi/2
            0       (Q3-pi/2) 0.35    -pi/2
            0       Q4      1.109   0
            1.2     Q5      0       -pi/2
            0       Q6      0       pi/2
            0.207   Q7      0       -pi/2];
  for i=1:n
      T(:,:,i)=[ cos(dhparam(i,2))                        -sin(dhparam(i,2))
0                                    dhparam(i,3);
                 sin(dhparam(i,2))*cos(dhparam(i,4))
cos(dhparam(i,2))*cos(dhparam(i,4))   -sin(dhparam(i,4))                    -
dhparam(i,1)*sin(dhparam(i,4));
                 sin(dhparam(i,2))*cos(dhparam(i,4))
cos(dhparam(i,2))*sin(dhparam(i,4))   cos(dhparam(i,4))
dhparam(i,1)*cos(dhparam(i,4));
                 0                                      0
0                                    1;                            ];
  end

A1=T(:,:,1);
A2=A1*T(:,:,2);
A3=A2*T(:,:,3);
A4=A3*T(:,:,4);
A5=A4*T(:,:,5);
A6=A5*T(:,:,6);
A7=A6*T(:,:,7);

A0=[1 0 0 0;0 1 0 0;0 0 1 0; 0 0 0 1];
Trf={A0,A1,A2,A3,A4,A5,A6,A7};
J=[0; 0; 1; 0; 0; 0]; % 1st Prismatic Joint
for i=2:7
    mat=Trf{1,i-1};
    On=A7(1:3,4);
    On1=mat(1:3,4);
    Zn1=mat(1:3,3);
    J=horzcat(J,([cross(Zn1,(On-On1));Zn1])); % Remaing Revolute Joints
end
```

Rigid Body Tree, Inverse Kinematics, Trajectory generation and Animation

```
% Define Robot Object
KUKA = robotics.RigidBodyTree('MaxNumBodies',7);
% Define Body and Joint Objects
body1 = robotics.RigidBody('body1');
jnt1 = robotics.Joint('jnt1','prismatic');
body2 = robotics.RigidBody('body2');
jnt2 = robotics.Joint('jnt2','revolute');
body3 = robotics.RigidBody('body3');
jnt3 = robotics.Joint('jnt3','revolute');
body4 = robotics.RigidBody('body4');
jnt4 = robotics.Joint('jnt4','revolute');
body5 = robotics.RigidBody('body5');
jnt5 = robotics.Joint('jnt5','revolute');
body6 = robotics.RigidBody('body6');
jnt6 = robotics.Joint('jnt6','revolute');
body7 = robotics.RigidBody('body7');
jnt7 = robotics.Joint('jnt7','revolute');


% Move Z 500 Rotate Y90
tform=makehgtform('translate',[0 0 0.5],'yrotate',pi/2,'xrotate',-pi/2);
setFixedTransform(jnt1,tform);
% Move X -215.6 Rotate Y-90
tform=makehgtform('translate',[-0.2156 0 0],'yrotate',-pi/2);
setFixedTransform(jnt2,tform);
%Move X -675+50 Y 350 Rotate Z90 Y180
tform=makehgtform('translate',[0 -0.35 0.4594],'yrotate',pi/2,'zrotate',-
pi/2);
setFixedTransform(jnt3,tform);
%Move X 1150
tform=makehgtform('translate',[1.15 0 0]);
setFixedTransform(jnt4,tform);
%Move X 752 Y 41 Rotate Y90
tform=makehgtform('translate',[0.752 0.041 0],'yrotate',pi/2);
setFixedTransform(jnt5,tform);
%Move Z 438 Rotate X90
tform=makehgtform('translate',[0 0 0.448],'xrotate',pi/2);
setFixedTransform(jnt6,tform);
%Move Y -207
tform=makehgtform('translate',[0 0.207 0],'xrotate',-pi/2);
setFixedTransform(jnt7,tform);


body1.Joint = jnt1;
body2.Joint = jnt2;
body3.Joint = jnt3;
body4.Joint = jnt4;
body5.Joint = jnt5;
body6.Joint = jnt6;
body7.Joint = jnt7;

addBody(KUKA,body1,'base')
```

```
addBody(KUKA,body2,'body1')
addBody(KUKA,body3,'body2')
addBody(KUKA,body4,'body3')
addBody(KUKA,body5,'body4')
addBody(KUKA,body6,'body5')
addBody(KUKA,body7,'body6')

KUKA.Bodies{1,1}.Joint.PositionLimits=[0 10];

KUKA.DataFormat = 'row';
%%
Jacobian=geometricJacobian(KUKA,KUKA.homeConfiguration,'body7')

%%
gripper = 'body7';

cupHeight = 0.2;
cupRadius = 0.05;
cupPosition = [1, 2, 1];

cup_body = robotics.RigidBody('cupFrame');
setFixedTransform(cup_body.Joint, trvec2tform(cupPosition))
addBody(KUKA, cup_body, 'base');

numWaypoints = 5;
q0 = homeConfiguration(KUKA);
qWaypoints = repmat(q0, numWaypoints, 1);

heightAboveTable = robotics.CartesianBounds('body7');
heightAboveTable.Bounds = [-inf, inf; ...
                           -inf, inf; ...
                           0,inf];

distanceFromCup = robotics.PositionTarget('cupFrame');
distanceFromCup.ReferenceBody = gripper;
distanceFromCup.PositionTolerance = 0.005

alignWithCup = robotics.AimingConstraint('body7');
alignWithCup.TargetPoint = [0 0 0]

limitJointChange = robotics.JointPositionBounds(KUKA)

fixOrientation = robotics.OrientationTarget(gripper);
fixOrientation.OrientationTolerance = deg2rad(1)

intermediateDistance = 0

limitJointChange.Weights = ones(size(limitJointChange.Weights));
limitJointChange.Weights(2)=0.1
limitJointChange.Weights(3)=0.5
fixOrientation.Weights = 0;
alignWithCup.Weights = 1;

distanceFromCup.TargetPosition = [0,0,intermediateDistance]
```

```
gik = robotics.GeneralizedInverseKinematics('RigidBodyTree', KUKA, ...

'ConstraintInputs',{'cartesian','position','aiming','orientation','joint'},..
.
    'SolverAlgorithm','LevenbergMarquardt')

[qWaypoints(2,:),solutionInfo] = gik(q0, heightAboveTable, ...
                        distanceFromCup, alignWithCup, fixOrientation, ...
                        limitJointChange)


fixOrientation.TargetOrientation = ...
    tform2quat(getTransform(KUKA,qWaypoints(2,:),gripper));

finalDistanceFromCup = 0;
distanceFromCupValues = linspace(intermediateDistance, finalDistanceFromCup,
numWaypoints-1);

maxJointChange = pi;

for k = 3:numWaypoints
    % Update the target position.
    distanceFromCup.TargetPosition(3) = distanceFromCupValues(k-1);
    % Restrict the joint positions to lie close to their previous values.
    limitJointChange.Bounds = [qWaypoints(k-1,:)' - maxJointChange,...
                               qWaypoints(k-1,:)' + maxJointChange];
    % Solve for a configuration and add it to the waypoints array.
    [qWaypoints(k,:),solutionInfo] = gik(qWaypoints(k-1,:), ...
                                         heightAboveTable, ...
                                         distanceFromCup, alignWithCup, ...
                                         fixOrientation, limitJointChange)
end

framerate = 15;
r = robotics.Rate(framerate);
tFinal = 30;
tWaypoints = [0,linspace(tFinal/2,tFinal,size(qWaypoints,1)-1)];
numFrames = tFinal*framerate;
qInterp = pchip(tWaypoints,qWaypoints',linspace(0,tFinal,numFrames))';

gripperPosition = zeros(numFrames,3);

for k = 1:numFrames
    gripperPosition(k,:) = tform2trvec(getTransform(KUKA,qInterp(k,:), ...
                                                    gripper));
end

figure;
show(KUKA, qWaypoints(1,:), 'PreservePlot', false);
hold on
exampleHelperPlotCupAndTable(cupHeight, cupRadius, cupPosition);
```

```
p = plot3(gripperPosition(1,1), gripperPosition(1,2),
gripperPosition(1,3),'r');

hold on
for k = 1:size(qInterp,1)
    axis([-10 10 -10 10 -10 10])
    show(KUKA, qInterp(k,:), 'PreservePlot', false);
    p.XData(k) = gripperPosition(k,1);
    p.YData(k) = gripperPosition(k,2);
    p.ZData(k) = gripperPosition(k,3);
    waitfor(r);
end
hold off
```

Appendix C

References

[1] Kuka, “KR Quantec Extra”, Dec 23, 2017 retrieved https://www.kuka.com/en-us/products/robotics-systems/industrial-robots/kr-quantec-extra