ECTE471/871/971/MECH950

Group Project Report

Group Number: 11

	Student Number	Name of the Member
1	6793630	Alexander Halank
2	5447355	Daniel Rainey
3	7544248	Andres Collazos
4	8914230	Anaïs Dubrana
5	8945512	Mikael Aleksander Jansen Shahly
6		

Contents

Group Roles:

MATLAB (ARTE) script:

Submission by - Mikael Shahly

Report:

Submission by – Alex Halank

Presentation:

Submission by – Andes Halank

Project Brief:

PROJECT: Nut tightening task

The task shown in Figure 1 is required to be automated using a computer-controlled manipulator.

The task is to insert and tighten four nuts on four horizontal bolts. The nuts are available from a spring activated parts feeder. A washer should be loaded into the bolt before the nut is screwed in. The centroid of the nut and washer are located in the centroid of the part feeder. The bolts are

located on a vertical platform with their shafts 20 cm away from the edges of the platform. Your task is to design and model the robot arm following the procedures given below. Assume that

- The coordinate frame of the base of the robot is parallel with the world coordinates (OXYZ).
- The robot approaches the nuts and washers from the above in a vertical direction
- The robot approaches the bolts from the top in a horizontal direction.
- The fingers of the robot gripper (tool) have a length of 40 mm.

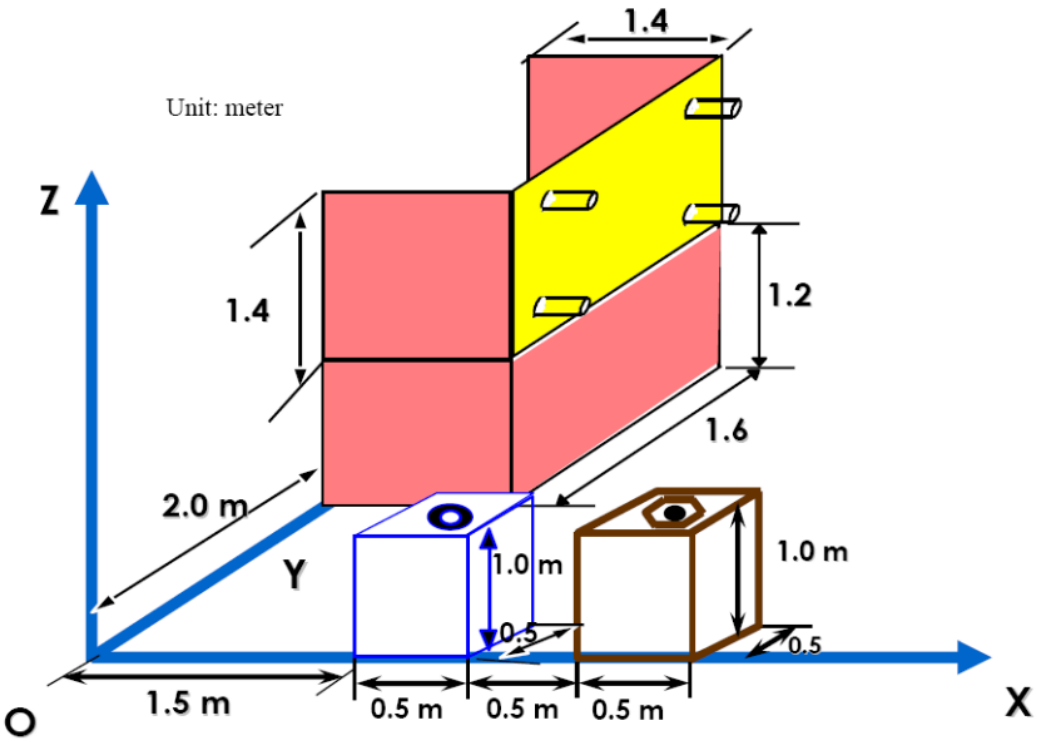

Figure 1: Nut Tightening Task (from project brief)

Section A

1 Robot Arm Characteristics

In this section, based on the assembly operation required, the characteristics a robotic arm is identified such that it can perform the assembly task automatically.

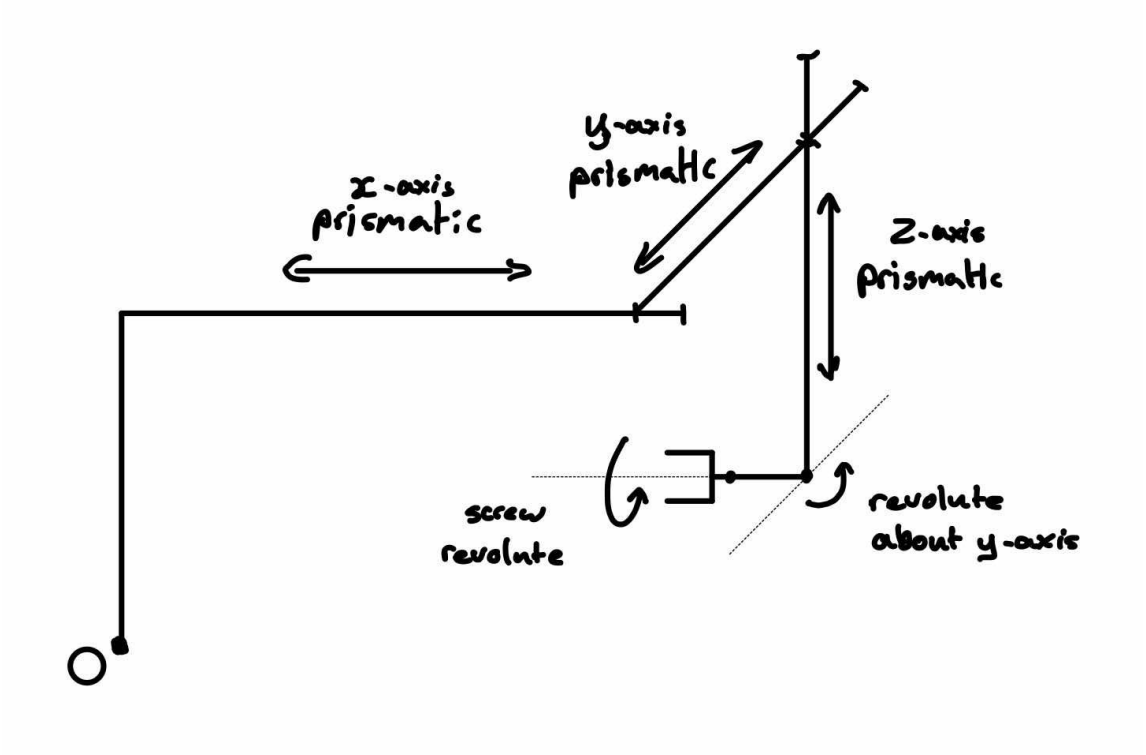

Figure 2: Preliminary Robot Design

a. The number of degrees of freedom of the robotic system.

   i. 5 Degrees of Freedom

b. The number and type of the joints of the robotic system.

   i. 3 prismatic joints and 2 revolute joints

c. The length of the links of the robot.

Link	Preliminary Length	Final Length
L0	2.6	2.60
L1	1.6 -> 2.75	1.44 – 2.75
L2	0.25 -> 3.4	0.25 – 3.40
L3	0.2 -> 1.4	0.20 – 1.56
L4	0.2 (incl. tool)	0 + tool = 0.04

The horizontal and vertical reaches of the system.

	Min	Max

X	1.40m	2.75m
Y	0.25m	3.40m
Z	1.00m	2.40m

d. The joint space work envelope required by the robotic system to perform the task.

The robot joint space work envelope is shown below as the transparent blue shape. Note that due being made up of prismatic joints, it has a relatively simple rectangular workspace with two notches on diagonally opposite edges caused by the tool being mounted directly after a final rotational joint.

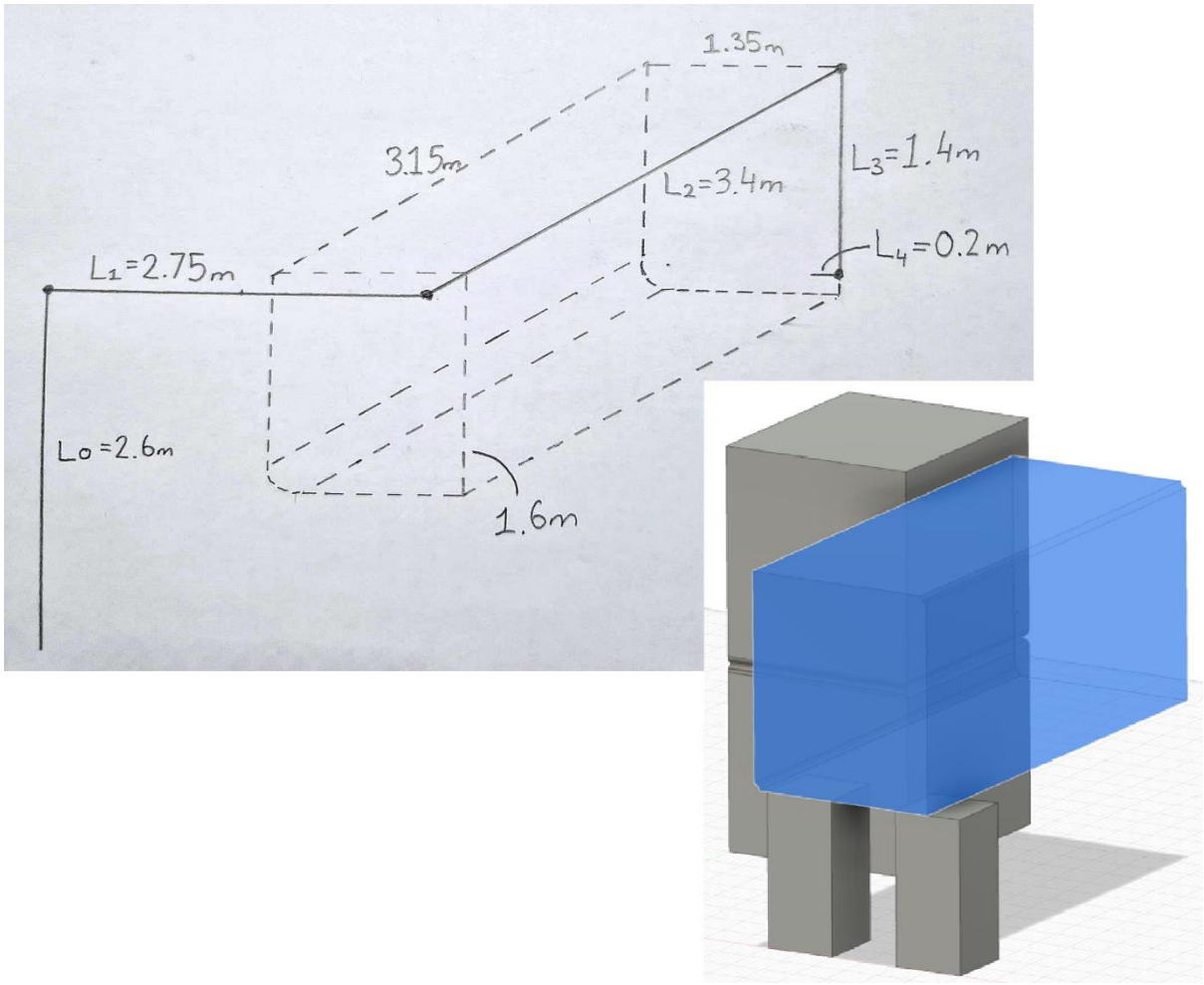

Figure 3: Robot work envelope

Link	Range of Movement
L0	0m
L1	1.15m
L2	3.15m
L3	1.40m
L4	0.20m (revolute)

2 Tool Orientation, Location and Approach Vectors

Derivation the orientation and location of the tool when at the nut pickup and placement on each bolt in terms of normal, sliding and approach vectors.

$$T_{washer} = \begin{bmatrix} -1 & 0 & 0 & 1.70 \\ 0 & 1 & 0 & 0.25 \\ 0 & 0 & -1 & 1.02 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad T_{nut} = \begin{bmatrix} -1 & 0 & 0 & 2.70 \\ 0 & 1 & 0 & 0.25 \\ 0 & 0 & -1 & 1.02 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{top\ left} = \begin{bmatrix} 0 & 0 & -1 & 1.48 \\ -1 & 0 & 0 & 0.85 \\ 0 & 1 & 0 & 2.40 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad T_{bottom\ left} = \begin{bmatrix} 0 & 0 & -1 & 1.48 \\ -1 & 0 & 0 & 0.85 \\ 0 & 1 & 0 & 1.41 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{top\ right} = \begin{bmatrix} 0 & 0 & -1 & 1.48 \\ -1 & 0 & 0 & 2.04 \\ 0 & 1 & 0 & 2.40 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad T_{bottom\ right} = \begin{bmatrix} 0 & 0 & -1 & 1.48 \\ -1 & 0 & 0 & 2.04 \\ 0 & 1 & 0 & 1.41 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3 Robot Kinematic Model:

The kinematic model of the robot is calculated in this section, including coordinate frames of the links of the robotic system. Afterwards the kinematic parameters and arm equation are found. The Arm equation relates the coordinate frame of the robot tool to the base coordinate frame

The reference frames of the robot where assigned such that they would be orthogonal. As a result, the kinematic parameters, and the resulting arm matrix of the tool to the base of the robot, can be easily defined according to the Denavit-Hartenberg (D-H) Representation.

Below is a figure of the robot with coordinate frames attached to it:

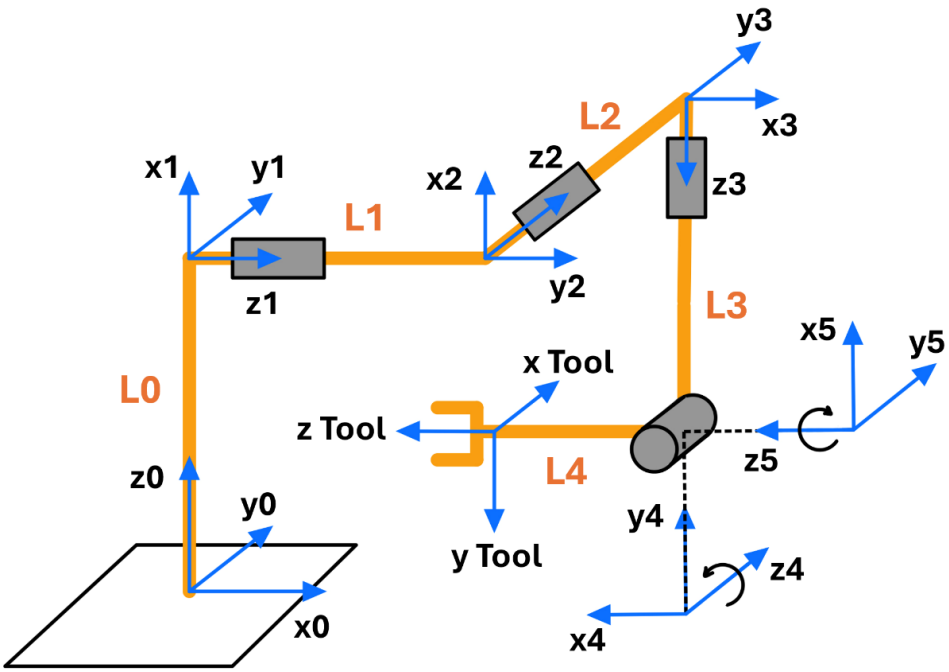

Figure 4: Robot coordinate frames assignment

Note that Link 4 in the robot design has a length of zero, which is reflected in the corresponding kinematic parameters

Based on this the kinematic parameters were found. These are represented in the table below:

Link	θ	d	a	α	Home
1	pi/2	2.6	0	pi/2	0
2	pi/2	q_1 + 1.6	0	pi/2	10
3	pi/2	q_2 + 0.25	0	-pi/2	10
4	pi	q_3 + 0.2	0	-pi/2	10
5	q_4	0	0	pi/2	0
6	q_5	0	0	0	0

Where q_4 and q_5 represent the variable rotation of the revolute joint 4 and screw joint 5. q_3, q_2, q_1 represents the variable distance of the 3 prismatic joints respectively.

The arm matrix is then calculated as the composite multiplication of each transformation matrix for each frame T_{k-1}^{k}. This gives the following calculations and results:

$T_{base}^{tool} = T_0^1 * T_1^2 * T_2^3 * T_3^4 * T_4^5$

$$\begin{pmatrix} \sin(q_5) - \cos(q_5)\cos(q_4) & \sin(q_5)\cos(q_4) & -\sin(q_4) & q_1 + 1.6 \\ -\sin(q_5) & -\cos(q_5) & 0 & q_2 + 0.25 \\ -\sin(q_4)\cos(q_5) & -\sin(q_4)\sin(q_5) & -\cos(q_4) & -q_3 + 2.4 \\ 0 & 0 & 0 & 1.0 \end{pmatrix}$$

The code to calculate this is:

```
%%Symbolic Computations for paramaters
syms  q_1 q_2 q_3 q_4 q_5
link_6_length = 0;
link_1_length = 2.6;



theta = [pi/2 pi/2 pi/2 pi q_4 q_5];
d=     [2.6 q_1+1.6 q_2+0.25 q_3+0.2 0 0];
a=     [0 0 0 0 0 0];
alpha= [pi/2 pi/2 -pi/2 -pi/2 pi/2 0];


T_01 = transformation_matrix(theta(1), alpha(1), d(1), a(1));
T_12 = transformation_matrix(theta(2), alpha(2), d(2), a(2));
T_23 = transformation_matrix(theta(3), alpha(3), d(3), a(3));
T_34 = transformation_matrix(theta(4), alpha(4), d(4), a(4));
T_45 = transformation_matrix(theta(5), alpha(5), d(5), a(5));
T_56 = transformation_matrix(theta(6), alpha(6), d(6), a(6));

```

```
 T_06=T_01*T_12*T_23*T_34*T_45*T_56; %Arm Matrix
 T_06_rounded = vpa(T_06, 4);
 disp("======================ARM MATRIX==================");
 disp(T_06_rounded);
```

```
function T = transformation_matrix(theta, alpha, d, a)
    % Calculate the cosines and sines
    Ctheta = cos(theta);
    Stheta = sin(theta);
    Calpha = cos(alpha);
    Salpha = sin(alpha);

    % Construct the transformation matrix
    T = [Ctheta, -Calpha*Stheta, Salpha*Stheta, a*Ctheta;
         Stheta, Calpha*Ctheta, -Salpha*Ctheta, a*Stheta;
         0, Salpha, Calpha, d;
         0, 0, 0, 1];
end
```

4 Robot Tool Characteristics

In this section the tool characteristics of the robot is found. Primarily the tool configuration vector of the robot and the tool configuration Jacobean matrix

The tool configuration vector is given by:

$$w = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ r_1^3 e^{q_n/\pi} \\ r_2^3 e^{q_n/\pi} \\ r_3^3 e^{q_n/\pi} \end{bmatrix}$$

Based on the arm matrix the tool configuration vector can be found as:

$$\begin{pmatrix} q_1 + 1.6 \\ q_2 + 0.25 \\ -q_3 + 2.4 \\ -\sin(q_4) \cdot e^{q_5/\pi} \\ 0 \\ -\cos(q_4) \cdot e^{q_5/\pi} \end{pmatrix}$$

Based on this the Jacobian of the tool vector with regards to **q** can be calculated using matlab**.** The expression is then:

$$\begin{pmatrix} 1.0 & 0 & 0 & 0 & 0 \\ 0 & 1.0 & 0 & 0 & 0 \\ 0 & 0 & -1.0 & 0 & 0 \\ 0 & 0 & 0 & -e^{q_5/\pi} \cdot \cos(q_4) & \frac{-e^{q_5/\pi} \cdot \sin(q_4)}{\pi} \\ 0 & 0 & 0 & 0 & \frac{e^{q_5/\pi}}{\pi} \\ 0 & 0 & 0 & e^{q_5/\pi} \cdot \sin(q_4) & -\frac{e^{q_5/\pi} \cdot \cos(q_4)}{\pi} \end{pmatrix}$$

The code for calculating this jacobian is given as:

```
%% FINDING THE JACOBIAN MATRIX
tool_config_vector = [T_06_rounded(1:3, 4); T_06_rounded(1, 3)*exp(q_5 / pi);
    T_06_rounded(2, 3)*exp(q_5 / pi); T_06_rounded(3, 3)*exp(q_5 / pi)];
disp("======================TOOL CONFIG VECTOR==================");
disp(tool_config_vector);

j1 = diff(tool_config_vector,q_1);
j2 = diff(tool_config_vector,q_2);
j3 = diff(tool_config_vector,q_3);
j4 = diff(tool_config_vector,q_4);
j5 = diff(tool_config_vector,q_5);

jacobian=[j1 j2 j3 j4 j5];
```

Section B

5 ARTE Model Development

The ARTE model was developed and added to the ARTE library. A picture of our robot in its workspace can be seen below:

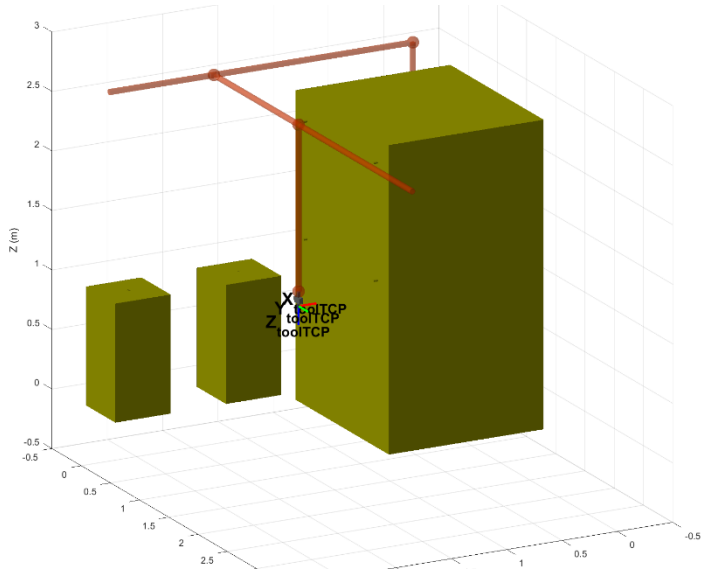

Figure 5: Arte robot implementation with work environment and tool

The Arte code to load our robot is shown below. This is out "parameters" file. To load the robot, simply run robot = load_robot and navigate to the folder in Arte where you have stored the parameters file. Note that the Arte library must be initialized first. The tool as well as the environment and equipment (boxes, wall, nuts and washers) were loaded using the teach GUI.

```
%% ROBOT PARAMATERS FILE
function robot = parameters()

robot.name='my_robot';

Link1_length = 5;
link_6_length = 5;
q = [0 0 0 0 0];

robot.DH.theta = '[pi/2 pi/2 pi/2 pi q(4) q(5)]';
robot.DH.d=     '[2.6 q(1)+1.6 q(2)+0.25 q(3)+0.2 0 0]';
robot.DH.a=     '[0 0 0 0 0 0]';
robot.DH.alpha= '[pi/2 pi/2 -pi/2 -pi/2 pi/2 0]';
robot.J=[];

%inverse kinematics for robot

robot.inversekinematic_fn = 'inversekinematic(robot, T)';

%R: rotational, T: translational
robot.kind=['T' 'T' 'T' 'R' 'R'];

% Initialize the base transformation matrix T0
robot.T0 = eye(4); % Identity matrix as the base frame

%number of degrees of freedom
robot.DOF = 5;

```

```
%maximum absolute speed of each joint rad/s or m/s
robot.velmax = [1
                1
                1
                1
                1];%not available

%minimum and maximum rotation angle in rad
robot.maxangle =[deg2rad(-0.75) deg2rad((2.75 - 1.6)); %Axis 1, minimum, maximum
                deg2rad(0) deg2rad((3.4 - 0.25)); %Axis 2, minimum, maximum
                deg2rad(0) deg2rad((1.4 - 0.2)); %Axis 3, min max
                deg2rad(0) deg2rad(90);%Axis 4, min max
                deg2rad(-360) deg2rad(360)]; %Axis 4: Unlimited (400? default)

robot.accelmax=robot.velmax/0.1; % 0.1 is here an acceleration time
% end effectors maximum velocity
robot.linear_velmax = 0.5; %chosen somewhat randomly

%% INITIALIZATION OF VARIABLES REQUIRED FOR THE SIMULATION
%position, velocity and acceleration
robot=init_sim_variables(robot);
robot.path = pwd;

%% GRAPHICS
robot.graphical.has_graphics=1;
robot.graphical.color = [25 20 40];
%for transparency
robot.graphical.draw_transparent=1;
%draw DH systems
robot.graphical.draw_axes=1;
%DH system length and Font size, standard is 1/10. Select 2/20, 3/30 for
%bigger robots
robot.graphical.axes_scale=1;
%adjust for a default view of the robot
robot.axis=[-10 10 -10 10 0 10];
robot = read_graphics(robot);

%% Robot dynamics
robot.has_dynamics=0;

%consider friction in the computations
robot.dynamics.friction=0;

robot.motors=load_motors([5 5 5 4 4 4]);
%Speed reductor at each joint
robot.motors.G=[300 300 300 300 300 300];
```

The parameters file to load in the work environment (so the washer / bolts / nuts etc) is shown below:

```
function robot = parameters()

robot.name= 'eqproject';

robot.DH.theta= '[]';
robot.DH.d='[]';
robot.DH.a='[]';
robot.DH.alpha= '[]';
robot.J=[];


robot.inversekinematic_fn = '';

%number of degrees of freedom
robot.DOF = 0;

%rotational: 0, translational: 1
robot.kind=[];

%minimum and maximum rotation angle in rad
robot.maxangle =[];

%maximum absolute speed of each joint rad/s or m/s
robot.velmax = [];
% end effectors maximum velocity
robot.linear_velmax = 0; %m/s
```

```
 % GRAPHICS
 robot.graphical.has_graphics=1;
 robot.graphical.color = [150 150 0]./255;
 %for transparency
 robot.graphical.draw_transparent=0;
 %draw DH systems
 robot.graphical.draw_axes=1;
 %DH system length and Font size, standard is 1/10. Select 2/20, 3/30 for
 %bigger robots
 robot.graphical.axes_scale=1;
 %adjust for a default view of the robot
 robot.axis=[-0.75 0.75 -0.75 0.75 0 1.1];
 %read graphics files
 robot = read_graphics(robot);

 %DYNAMICS
 robot.has_dynamics=0;
```

```
%base reference system
%robot.T0 = eye(4);
robot.T0 = [1 0 0 0;
            0 1 0 0;
            0 0 1 0;
            0 0 0 1];


%definition of the tool center point with respect to the last referenc
%system.
%for tools, this TCP usually means the transformation from system
%(X_tool0,Y_tool0,Z_tool0) to (X_tool1,Y_tool1,Z_tool1)
robot.TCP = [1 0 0 0;
             0 1 0 0;
             0 0 1 0;
             0 0 0 1];

%INITIALIZATION OF VARIABLES REQUIRED FOR THE SIMULATION
%position, velocity and acceleration
robot=init_sim_variables(robot);
robot.path = pwd;
```

6 ARTE Derivation of the Kinematic Model

In this section Arte is used to derive the kinematic model of the robot. These results are then compared to the previously found kinematic model in task 3A.

To find the kinematic model in ARTE the function directkinematic(Robot, **q**) is used, where **q** relates to the joint vector. In the home positions (**q** = [0 0 0 0 0]) we get the following results:

```
>> directkinematic(robot, [0 0 0 0 0])

ans =

   -1.0000         0    0.0000    1.6000
         0    1.0000   -0.0000    0.2500
   -0.0000   -0.0000   -1.0000    2.4000
         0         0         0    1.0000
```

Which maps perfectly to the calculated Arm matrix with **q** = [0 0 0 0 0] in task A3.

7 Jacobian Matrix of Robot Tool

In this section Arte is used to determine the tool Jacobian matrix of the robot for 2 different positions. These are for when the robot picks up a nut and for when it places it on the top left bolt of the platform.

For this task we need to find the joint vectors (**q)** for the robot in the 2 positions; picking up the nut and placing the nut on the top left bolt of the platform. From task 2A we know the arm matrix for the tooltip in these positions, meaning we know the tool configuration. Based on this the custom “inversekinematics(Robot, T)” function.

Code and results are shown below:

$$J_{\text{nut}} = \begin{bmatrix} 0 & 1.0000 & 0.0000 & 0.0000 & 0 \\ 0 & -0.0000 & 1.0000 & 0.0000 & 0 \\ 1.0000 & 0.0000 & -0.0000 & 0 & 0 \\ 0 & 0 & 0 & 0.0000 & 0.0000 \\ 0 & 0 & 0 & -0.0000 & 1.0000 \\ 0 & 0 & 0 & -1.0000 & -0.0000 \end{bmatrix}$$

$$J_{\text{top\_left}} = \begin{bmatrix} 0 & 1.0000 & 0.0000 & 0.0000 & 0 \\ 0 & -0.0000 & 1.0000 & 0.0000 & 0 \\ 1.0000 & 0.0000 & -0.0000 & 0 & 0 \\ 0 & 0 & 0 & 0.0000 & 0.0000 \\ 0 & 0 & 0 & -0.0000 & 1.0000 \\ 0 & 0 & 0 & -1.0000 & -0.0000 \end{bmatrix}$$

```
%% Task 7
robot = load_robot;

% T_nut matrix
T_nut = [
    -1  0  0  2.70;
     0  1  0  0.25;
     0  0 -1  1.02;
     0  0  0  1
];

% T_top_left matrix
T_top_left = [
     0  0 -1  1.48;
    -1  0  0  0.85;
     0  1  0  2.40;
     0  0  0  1
];

q_nut = inversekinematic(robot, T_nut);
q_topleft = inversekinematic(robot, T_top_left);

J_nut = manipulator_jacobian(robot, q_nut);
J_topleft = manipulator_jacobian(robot, q_topleft);

disp(J_nut);
disp(J_topleft);
```

8 Tool Position and Orientation at four different joint vectors

In this section Arte is used to display the tooltip position and orientation of four different sets of joint vectors.

Calculations

This section has been solved using the directkinematic(Robot, **q**) function in Arte to explicitly calculate the tool pose. It could also have been solved using teach to set the joint vectors and the following robot drawing.

The following code is used to calculate and draw the robot. Note that the figures generated are adjusted manually for a better view after being created in the code.

The variable **endpoint_pos_1, endpoint_pos_2, endpoint_pos_3, endpoint_pos_4** contain tooltip orientation and position

$$\text{tooltip\_position\_1} = \begin{bmatrix} 0 & 0 & 0 & -1 & 1.6000 \\ 0 & 1 & 0 & 0 & 0.2500 \\ 1 & 0 & 0 & 0 & 2.4000 \\ 0 & 0 & 0 & 1 & \end{bmatrix} \qquad \text{tooltip\_position\_2} = \begin{bmatrix} -1 & 0 & 0 & 1.8000 \\ 0 & 1 & 0 & 1.4500 \\ 0 & 0 & -1 & 1.2000 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{tooltip\_position\_3} = \begin{bmatrix} 0 & 0 & 0 & -1 & 1.8000 \\ 0 & 1 & 0 & 0 & 2.2500 \\ 1 & 0 & 0 & 2.4000 & \\ 0 & 0 & 0 & 1 & \end{bmatrix} \qquad \text{tooltip\_position\_4} = \begin{bmatrix} 0 & 0 & -1 & 1.8000 \\ 0 & 1 & 0 & 2.2500 \\ 1 & 0 & 0 & 1.4000 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
%% Calculting the tool-tip in 4 different joint vectors
robot = load_robot;
robot.equipment{1} = load_robot;
robot.tool = load_robot;
robot.graphical.draw_axes=0;

joints_1 = [0, 0, 0, 0, pi/2];
joints_2 = [1, 1, 1.2, 0, 0];
joints_3 = [0.2, 1.4, 0.7, pi/2 0];
joints_4 = [0.5, 0.7, 0.4, pi/2, 0];

endpoint_pos_1 = directkinematic(robot, joints_1);
endpoint_pos_2 = directkinematic(robot, joints_2);
endpoint_pos_3 = directkinematic(robot, joints_3);
endpoint_pos_4 = directkinematic(robot, joints_4);
```

```
16
17      drawrobot3d(robot, joints_1);
18      saveas(gcf, 'robot-pos-1.fig');
19      drawrobot3d(robot, joints_2 );
20      saveas(gcf, 'robot-pos-2.fig');
21      drawrobot3d(robot, joints_3);
22      saveas(gcf, 'robot-pos-3.fig');
23      drawrobot3d(robot, joints_4);
24      saveas(gcf, 'robot-pos-4.fig');
```

Display:

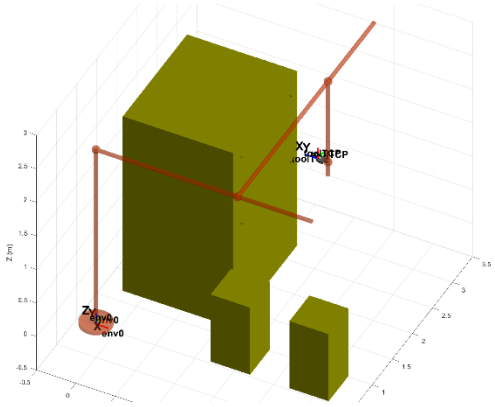

Figure 6: **Position for joints 1**

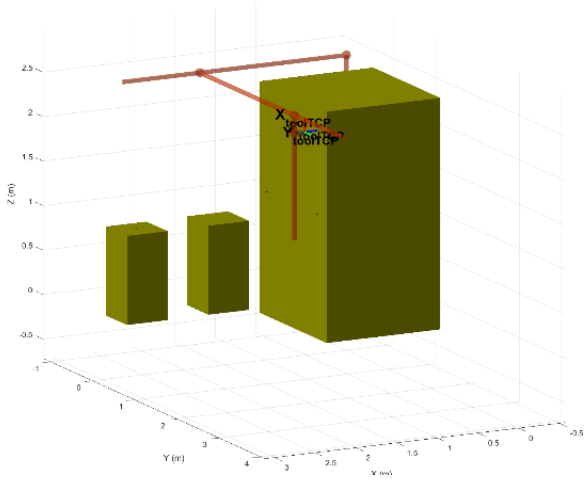

Figure 7: Position for joints 2

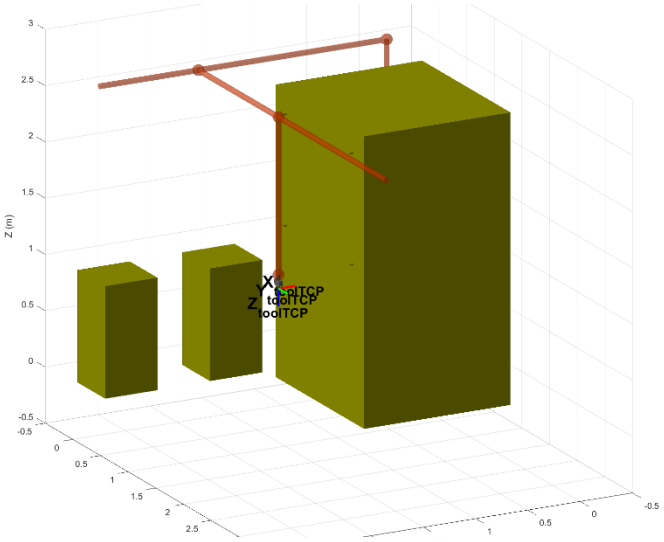

Figure 8: Position for joints 3

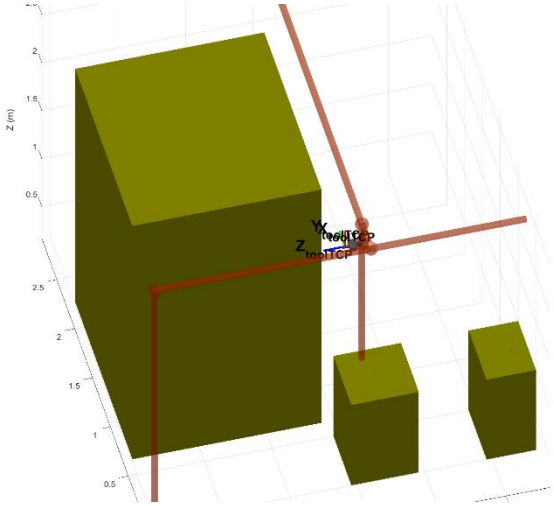

Figure 9: Position for joints 4

9 Developing Robot Paths

Using ARTE, we have determined a smooth transition for the tooltip of the robot when picking a nut and placing on the up-left bolt of the platform.

Several target points were saved as can be seen in the figures below, including; home, nut, washer and deposit approach and pickup locations.

The tool tip was also tested using animations such as the open gripper function; simulation_open_tool.

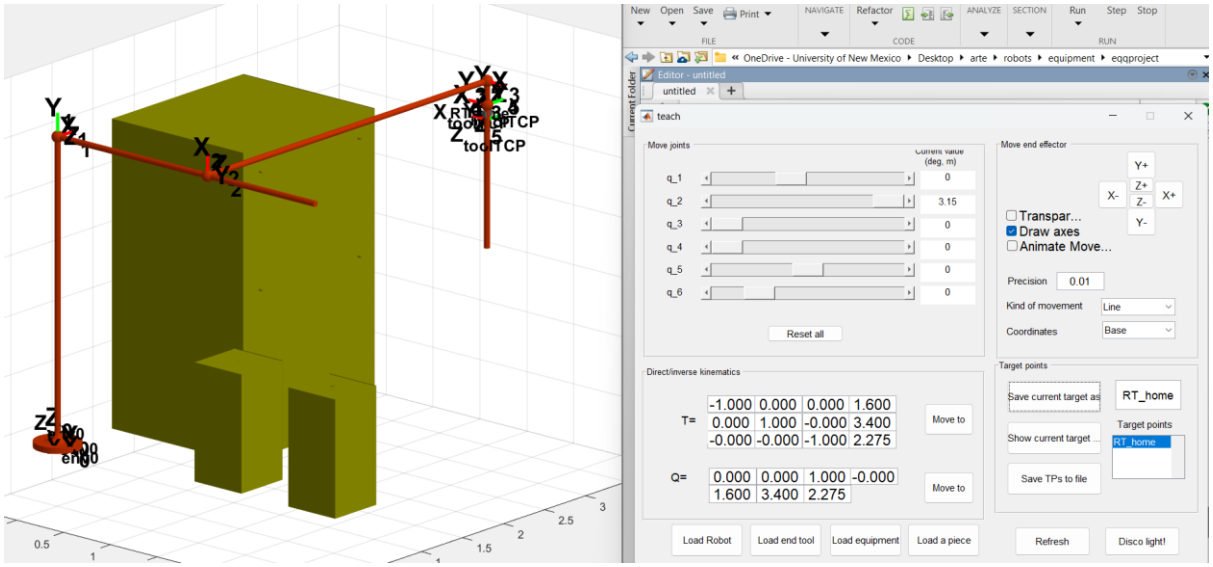

Figure 10: simulation screenshot

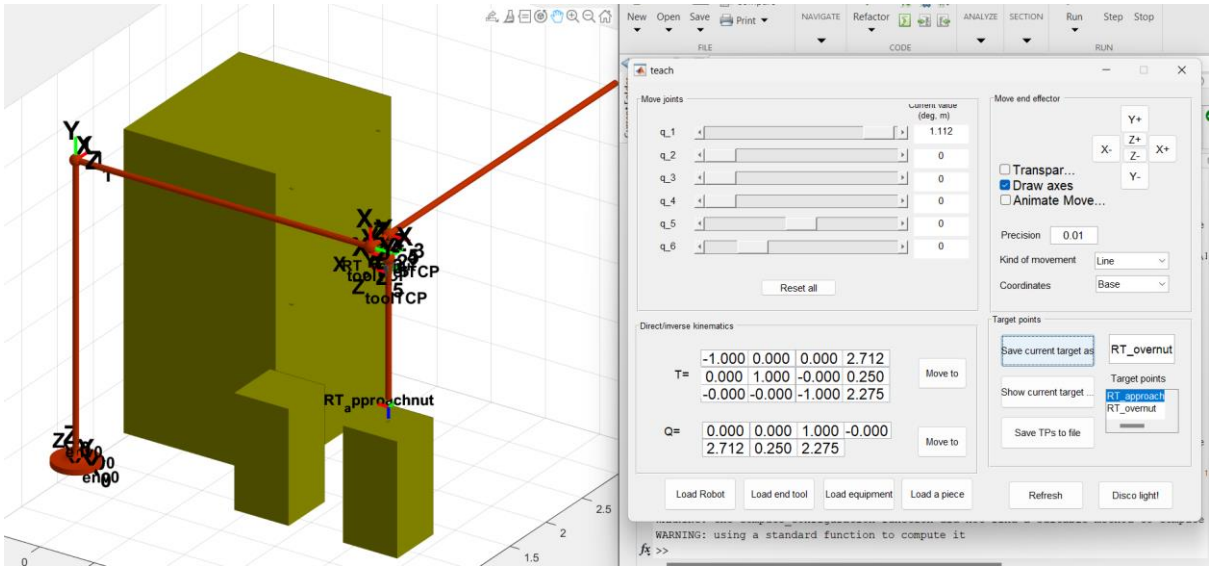

Figure 11: simulation screenshot

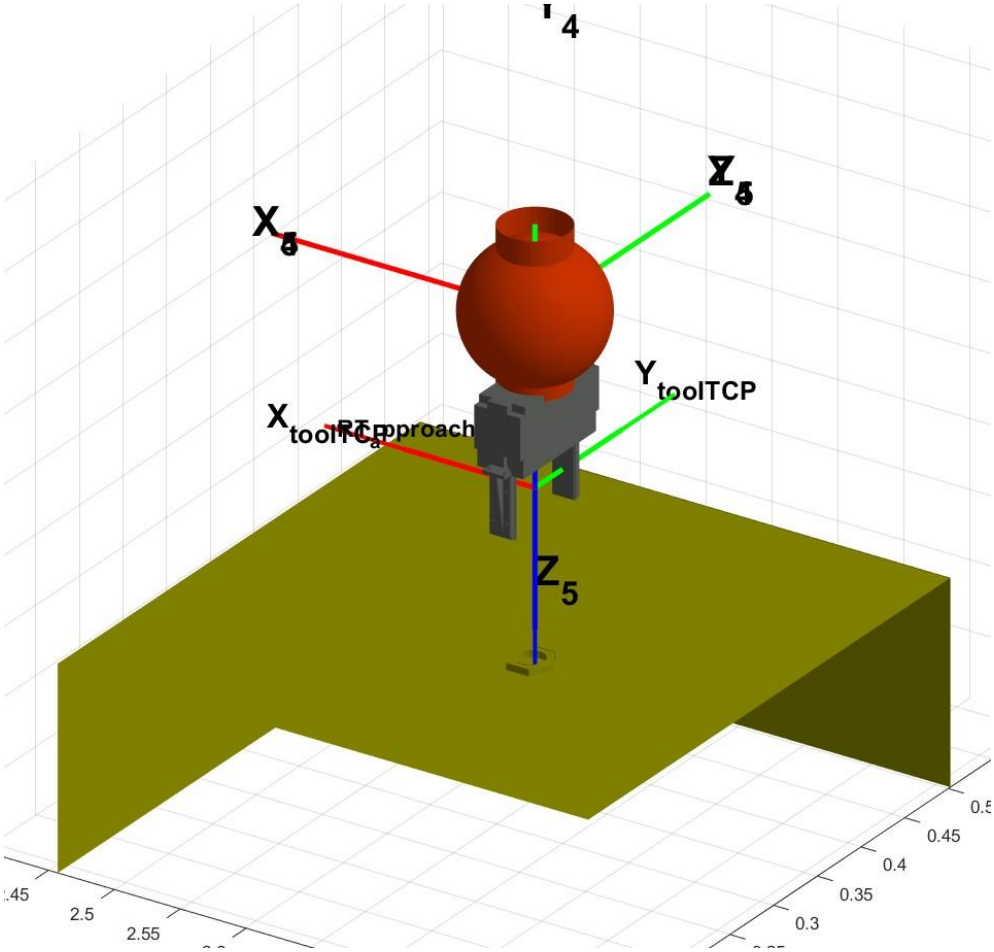

Figure 12: simulation screenshot

10 Velocity and Trajectory Analysis

The velocity of the robot joint in order to produce the tool trajectory you have been generated for picking a nut and placing on the up-left bolt of the platform.

Trajectories are calculated using ARTE built in functions and a custom inverse kinematics function for the robot. The code and trajectories are shown below:

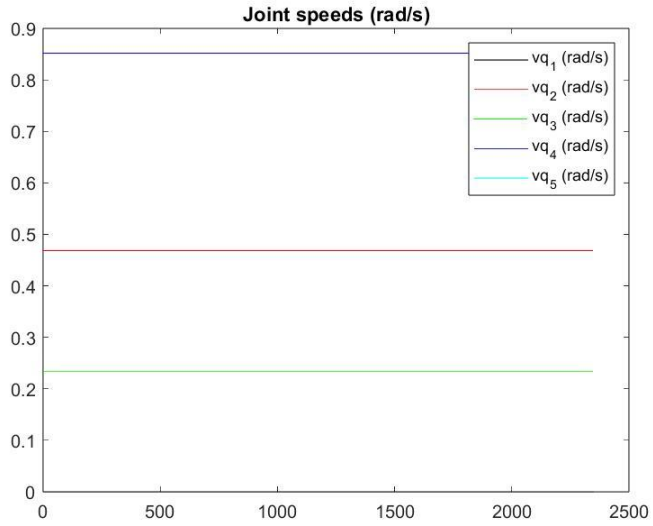

Figure 13: joint speeds from Arte calculations

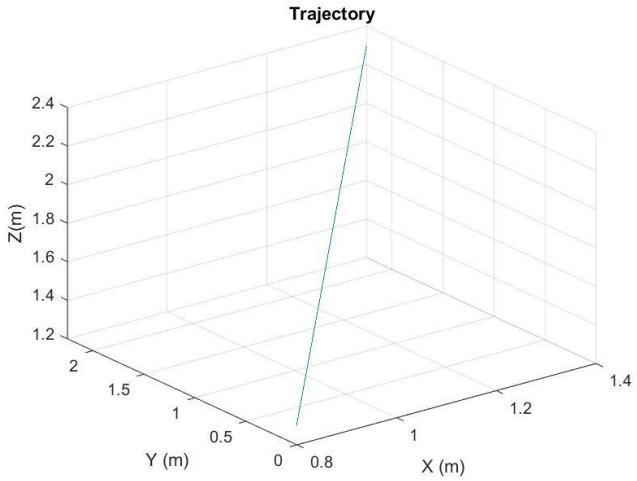

Figure 14: Trajectory in base coordinate frame

NOTE: to run the code a simple change must be made to the built in ARTE function MANIPULATOR_JACOBIAN(ROBOT, Q). Line 119 must be changed to:

for i=1:n-1,

As the MANIPULATOR_JACOBIAN(ROBOT, Q) function assumes that your robot has 6 joints.

```
113
114  %now compute conventional Jacobian
115  %The conventional Jacobian is a 6xDOF matrix that allows to compute the
116  %linear and angular speed of the end effector as a function of the joint
117  %speeds qp
118  J=zeros(6,robot.DOF);
119  for i=1:n-1,
120      %rotational
121      if robot.kind(i)=='R'
122          J(:,i) =[cross(z(:,i),pn(:,i)); z(:,i)];
123      else %translational joint
124          J(:,i) =[z(:,i); zeros(3,1)];
125      end
126  end
```

11 Graphical Simulation of Assembly Process

ARTE has been used to graphically simulate the robotics arm when performing the assembly process by using TEACH tool For full animation please refer to the PowerPoint slides.

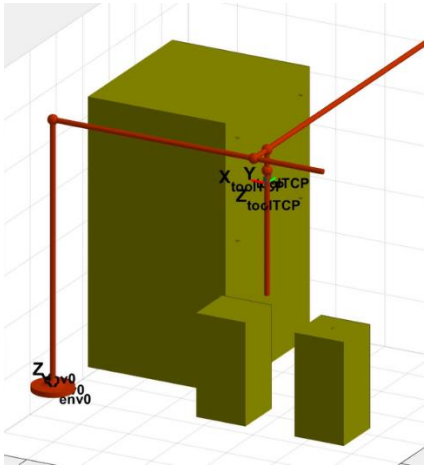

Figure 13: simulation screenshot

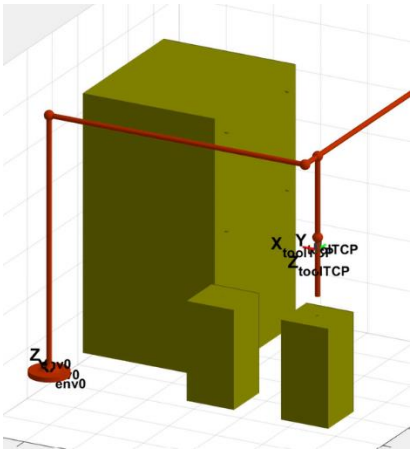

Figure 14: simulation screenshot

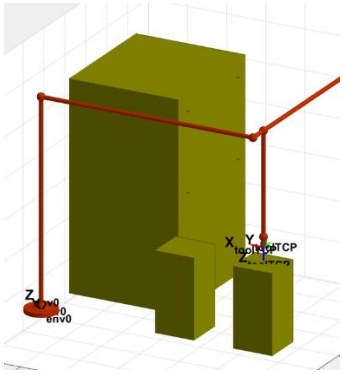

Figure 15: simulation screenshot