Introduction to Robot Modeling (ENPM662)

7-DOF Cobot Arm for Pick and Place Operation

Final-Project

Group-Members:

Ankur Chavan (119284513)

Krishna Hundekari (119239049)

Table of Content

Introduction:

This is the era of Fourth Industrial Revolution, where the factories are becoming smarter, and the robots are becoming an important part of the Factory floors. But we still have not reached the pinnacle where the idea is to make factories fully automated and human free. In the present setting, Robots must work alongside humans. Hence, rather we need a more collaborative association between Robot and Human. This is where "Cobot" as in Collaborative Robot comes in the picture.

In this collaborative environment, safety of the fellow human is of utmost importance. Cobots work alongside Human where no precautionary measures needed from Human side, they are programmed reliably to interact with the environment and collaborate with Human without causing them any harm. We both are interested in the Industrial robotics, that is why we were looking for the applications of the Cobots in the industrial setting that we can work on for this project.

Pick and place robots enable companies to use automated solutions for lifting objects from one location and placing them at other locations. Pick and place robots are one of the most utilized automation machines in the food packaging workspace. Simple tasks such as lifting objects or moving them do not require a lot of thought processes. Therefore, using human workers on these tasks can be wasteful, as the workforce can be used for other tasks that require higher mental abilities.

While working with last employer, Tetra Pak, which is food processing and packaging industry, one of the teammates came across smart food processing and packaging factories. The food processing and packaging industry is growing at a rapid pace. To meet the growing demands and required efficiency, manufacturers are using robotics to achieve tasks that would otherwise require dedicated manual labor.

There are automated packaging machines that pack the food at tremendous rates, but they need reloading of the packaging material roll from time to time. This is a demanding and a repetitive task for the Human. We cannot remove the human part completely because we need humans to inspect the packaging machines parameter monitoring and to conduct the scheduled maintenance. Hence, We decided to simulate the Pick and Place operation for reloading the roll of packaging material on the Packaging machine using Kuka Cobot as our project.

Application:

For our project, we are considering a factory floor where a packaging machine is active and packing the food inside the packaging material automatically. After the roll exhausts, we need to reload the roll of packaging material on the packaging machine. This is repetitive task, and we are going to automate this task by using KUKA iiwa Cobot.

The important parameters in consideration:

The weight of the roll : 8 kg

The distance between the stack of the roll and the packaging machine: 600mm

The task will be to pick up the roll from the stack and load it on the machine. To simulate the operation, we are going to design the KUKA cobot in solidworks and operate this cobot in the Gazebo environment using Teleop Script. We have installed a camera instead of a griping mechanism on the end effector and using the feed of the camera in the Rviz we are verifying that the cobot is controlled in a desired way using Teleop and the pick and place operation is successfully validated.

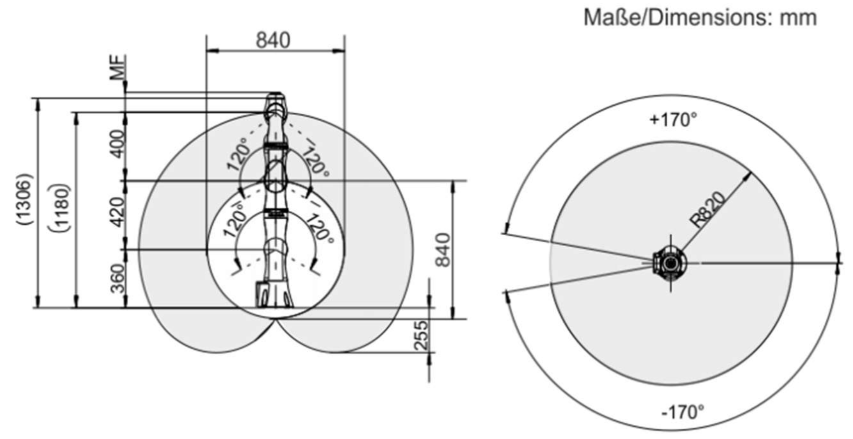
Robot Type:

Our robot is a serial manipulator with eight links and a camera mounted on the end effector. In robotics, a manipulator is a device used to manipulate materials without direct physical contact by the operator. They are created from a sequence of link and joint combinations. The links are the rigid members connecting the joints, or axes. The axes are the movable components of the robotic manipulator that cause relative motion between adjoining links.

Degree of Freedom (DOF):

The term degrees of freedom is widely used to define the motion capabilities of robots. The number of a robot's moveable joints is often referred to as its degrees of freedom. For example, a robot with three movable joints will have three axes and three degrees of freedom, a robot with four movable joints will have four axes, and so on. In this project the robot has 7 joints hence 7 DOF. To reach a position in a workspace from every angle, at least 6 degrees of freedom are required. With more than six joints, a robot can reach the same location at the same angle in multiple ways, making our robot more dexterous.

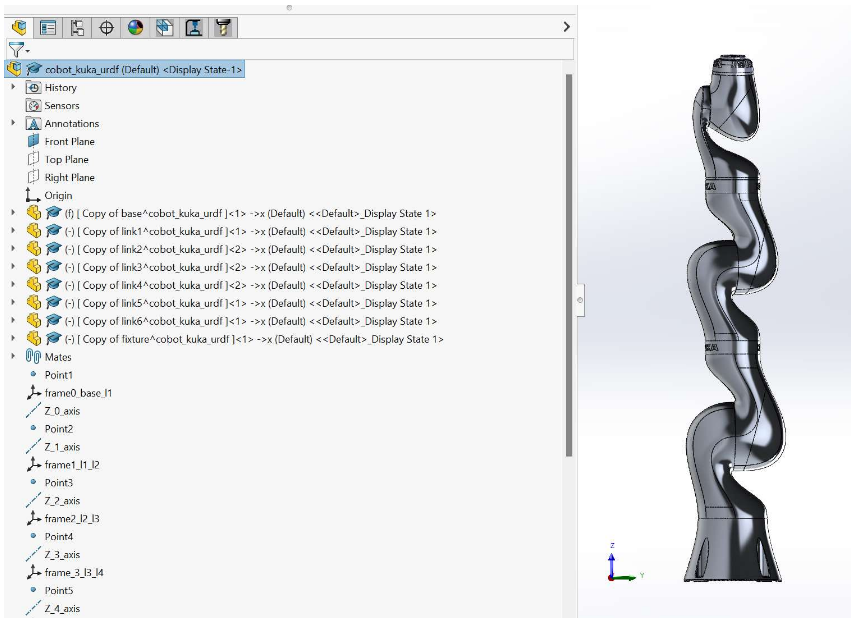
Dimensions:



As you can see in the figure above, the maximum reach of our robot is 820mm. The robotic arm's "reach," as the name implies, is a measurement of how far it can extend when fully extended. It thereby establishes the boundaries of the robot's work area. It's crucial to remember that the reach only provides a general idea of the robot's workspace.

CAD Model:

Using the official dimensions available of Kuka lbr iiwa we made the cad model utilizing the solidworks software. We then created a assembly file of the robot which we later exported to urdf. Please find in the submission folder all the parts and assembly of the cobot arm.



```
cobot_kuka_urdf (Default) <Display State-1>
  History
  Sensors
  Annotations
  Front Plane
  Top Plane
  Right Plane
  Origin
  (f) [ Copy of base^cobot_kuka_urdf ]<1> ->x (Default) <<Default>_Display State 1>
  (-) [ Copy of link1^cobot_kuka_urdf ]<1> ->x (Default) <<Default>_Display State 1>
  (-) [ Copy of link2^cobot_kuka_urdf ]<2> ->x (Default) <<Default>_Display State 1>
  (-) [ Copy of link3^cobot_kuka_urdf ]<2> ->x (Default) <<Default>_Display State 1>
  (-) [ Copy of link4^cobot_kuka_urdf ]<2> ->x (Default) <<Default>_Display State 1>
  (-) [ Copy of link5^cobot_kuka_urdf ]<1> ->x (Default) <<Default>_Display State 1>
  (-) [ Copy of link6^cobot_kuka_urdf ]<1> ->x (Default) <<Default>_Display State 1>
  (-) [ Copy of fixture^cobot_kuka_urdf ]<1> ->x (Default) <<Default>_Display State 1>
  Mates
  Point1
  frame0_base_l1
  Z_0_axis
  Point2
  Z_1_axis
  frame1_l1_l2
  Point3
  Z_2_axis
  frame2_l2_l3
  Point4
  Z_3_axis
  frame_3_l3_l4
  Point5
  Z_4_axis
```

DH Parameters:

A commonly used convention for selecting frames of reference in robotics applications is the **Denavit and Hartenberg (D–H) convention** which was introduced by Jacques Denavit and Richard S. Hartenberg. In this convention, each homogenous transformation is A_i is represented as a product of four basic transformations. The four quantities are parameters associated with link i and joint I and are generally given the names link length, link twist, link offset, and joint angle, respectively.

The following four transformation parameters are known as D–H parameters:

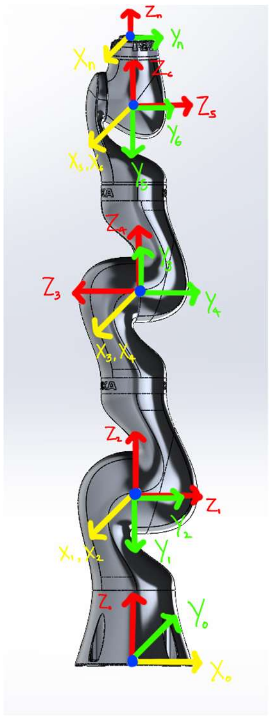
d: offset along previous z to the common normal

ϑ: angle about previous z, from old x to new x

r: length of the common normal (aka a, but if using this notation, do not confuse with α). Assuming a revolute joint, this is the radius about previous z.

α: angle about common normal, from old z axis to new z axis

Frame assignment:



DH Table:

Frames	θ	**d**	$\propto$	**a**
0-1	$\theta_1^* - \pi/2$	d_1 = 360 mm	$-\pi/2$	0
1-2	θ_2^*	0	$\pi/2$	0
2-3	θ_3^*	d_3 = 420 mm	$\pi/2$	0
3-4	θ_4^*	0	$-\pi/2$	0
4-5	θ_5^*	d_5 = 400 mm	$-\pi/2$	0
5-6	θ_6^*	0	$\pi/2$	0
6-7	θ_7^*	d_7 = 126 mm	0	0

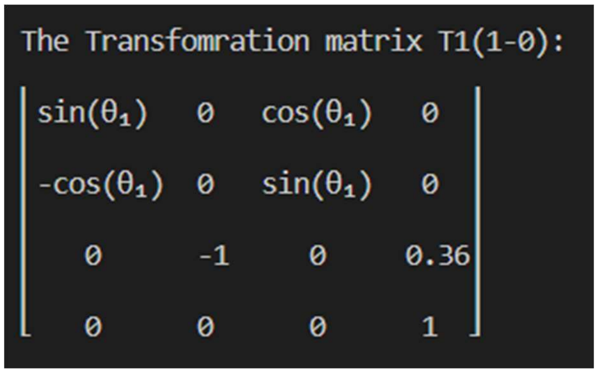
Forward Kinematics:

Computed the transformation matrices to map between the frames and they will be pasted on the output screen. The final T (7-0) transformation is also printed on the output terminal. In the further sections we will validate the transformation matrices by comparing them with the Gazebo simulation for four different poses.

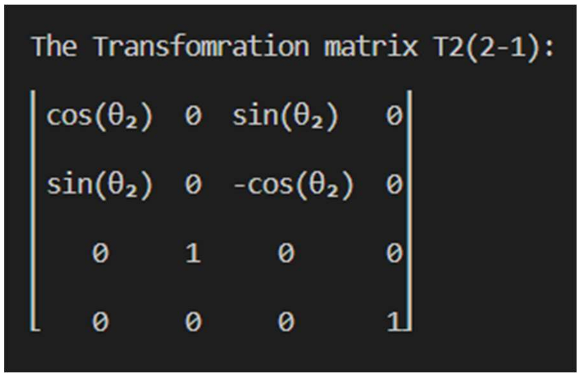
We computed these transformation matrices using the DH table mentioned above and the general matrix as below:

Transformation matrix

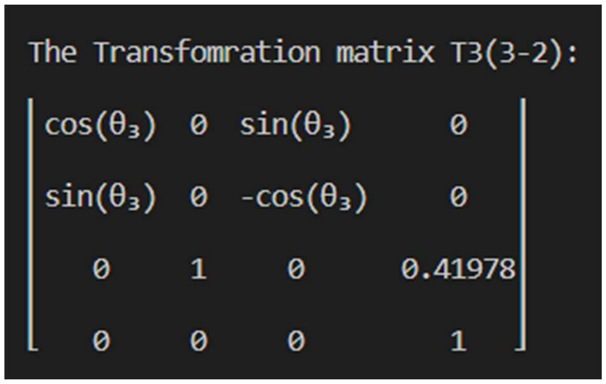
$$^{i-1}_{\ i}T = A_i = \begin{bmatrix} c\theta_i & -c\alpha_i s\theta_i & s\alpha_i s\theta_i & a_i c\theta_i \\ s\theta_i & c\theta_i c\alpha_i & -s\alpha_i c\theta_i & a_i s\theta_i \\ 0 & s\alpha_i & c\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

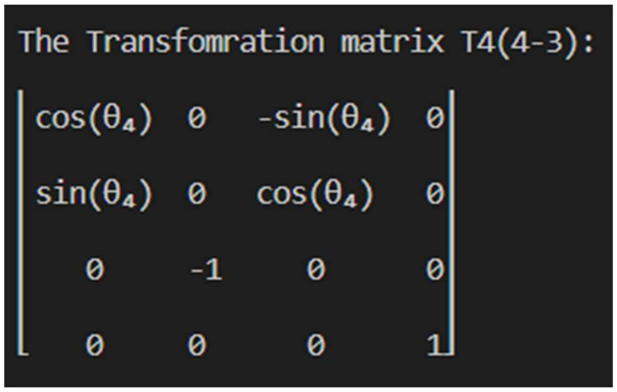


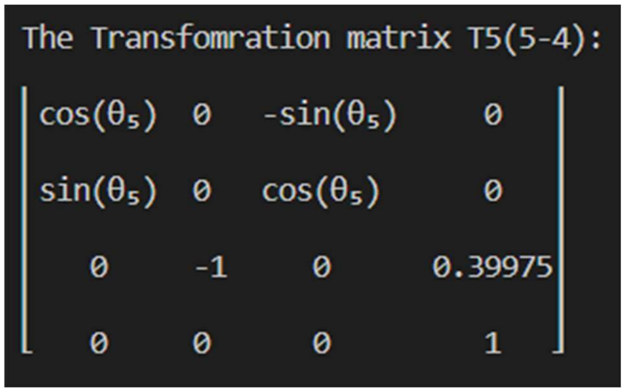
```
The Transfomration matrix T1(1-0):

│sin(θ₁)   0   cos(θ₁)   0  │
│-cos(θ₁)  0   sin(θ₁)   0  │
│   0     -1      0     0.36│
│   0      0      0      1  │
```

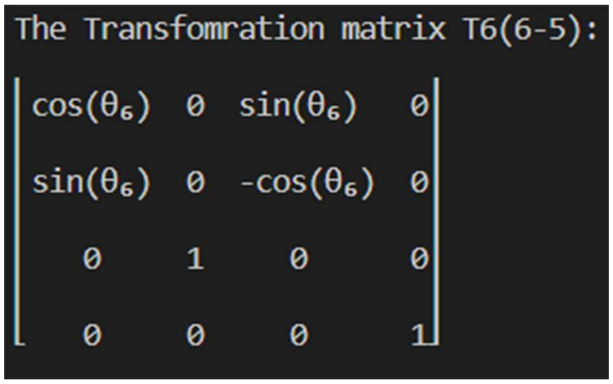


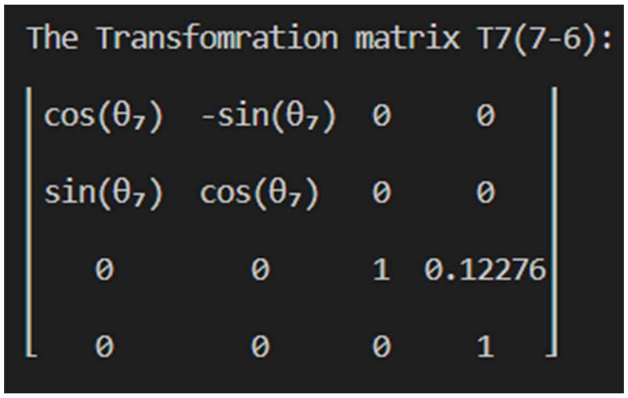
```
The Transfomration matrix T2(2-1):

│cos(θ₂)  0  sin(θ₂)   0│
│sin(θ₂)  0  -cos(θ₂)  0│
│   0     1     0      0│
│   0     0     0      1│
```


```
The Transfomration matrix T3(3-2):

⎡cos(θ₃)  0  sin(θ₃)      0   ⎤
⎢sin(θ₃)  0  -cos(θ₃)     0   ⎥
⎢   0     1     0     0.41978 ⎥
⎣   0     0     0         1   ⎦
```


```
The Transfomration matrix T4(4-3):

⎡cos(θ₄)  0   -sin(θ₄)  0⎤
⎢sin(θ₄)  0   cos(θ₄)   0⎥
⎢   0    -1      0      0⎥
⎣   0     0      0      1⎦
```


```
The Transfomration matrix T5(5-4):

⎡cos(θ₅)  0   -sin(θ₅)      0   ⎤
⎢sin(θ₅)  0   cos(θ₅)       0   ⎥
⎢   0    -1      0      0.39975 ⎥
⎣   0     0      0          1   ⎦
```


```
The Transfomration matrix T6(6-5):

⎡cos(θ₆)  0  sin(θ₆)   0⎤
⎢sin(θ₆)  0  -cos(θ₆)  0⎥
⎢   0     1     0      0⎥
⎣   0     0     0      1⎦
```


```
The Transfomration matrix T7(7-6):

⎡cos(θ₇)  -sin(θ₇)  0     0   ⎤
⎢sin(θ₇)  cos(θ₇)   0     0   ⎥
⎢   0        0      1  0.12276⎥
⎣   0        0      0     1   ⎦
```

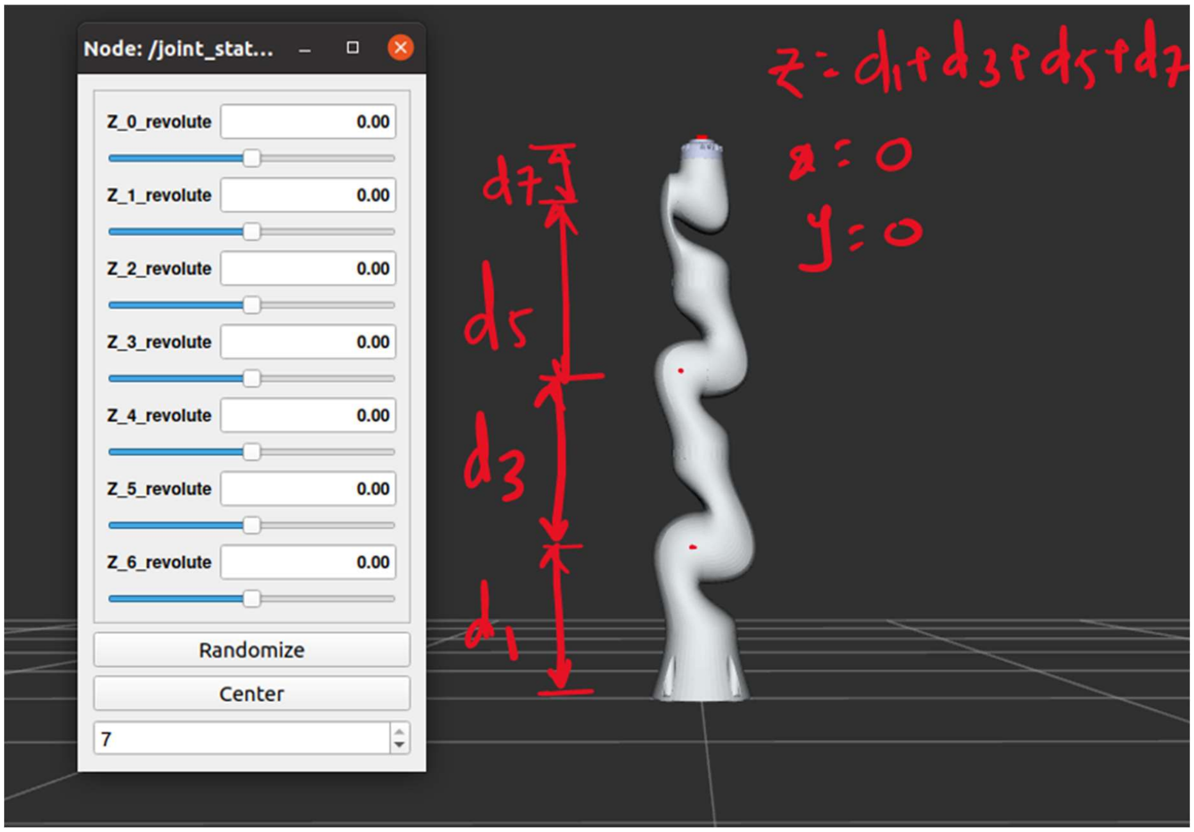
The final transformation matrix between frame 7 to base frame can be obtained by:

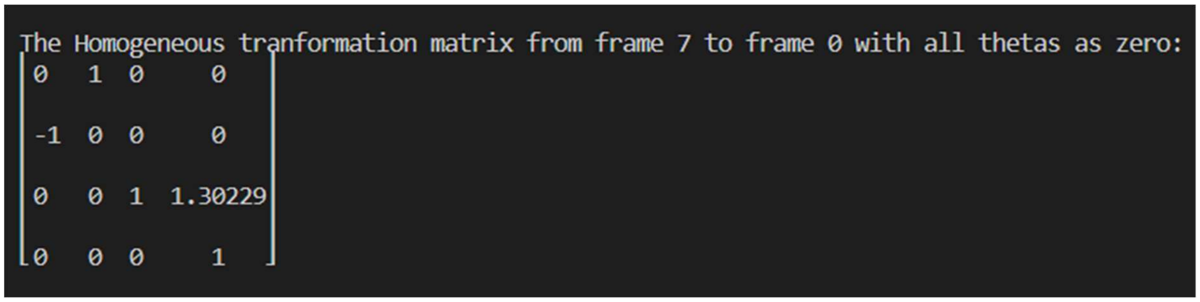
$$\boxed{T_7^0 = T_1^0 \cdot T_2^1 \cdot T_3^2 \cdot T_4^3 \cdot T_5^4 \cdot T_6^5 \cdot T_7^6}$$

Forward Kinematics Validation:

For validation of forward kinematics, we took four different orientations, each orientation with different combination of joint angles. We then compared the coordinates of the end effector we got in rviz for each orientation with that of coordinated obtained from transformation matrix. The coordinates from both the sources match for all five different orientations, which validates the forward kinematics. The final transformation matrix is also printed on the screen.

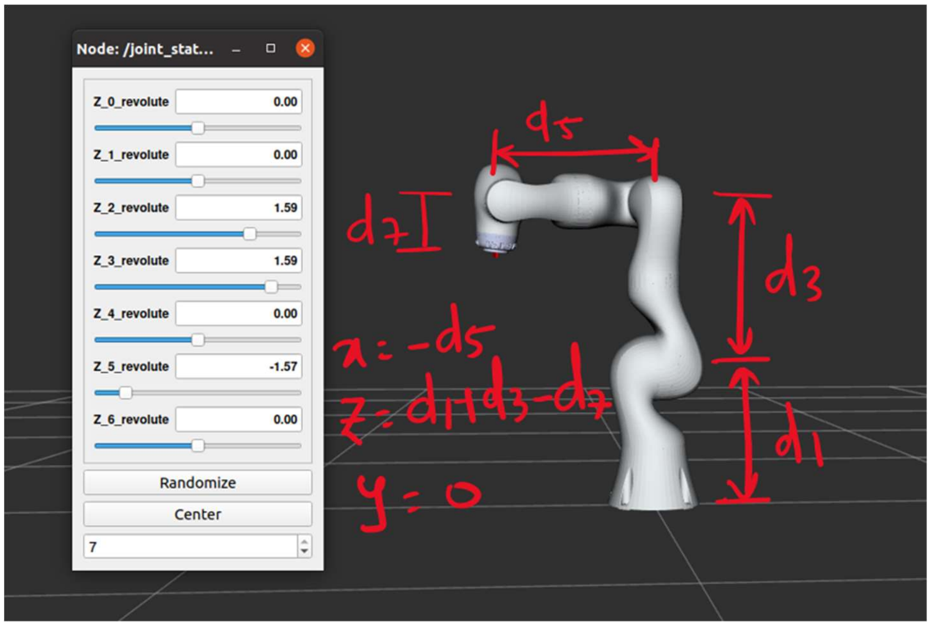
Default Pose: (0, 0, 0, 0, 0, 0, 0):



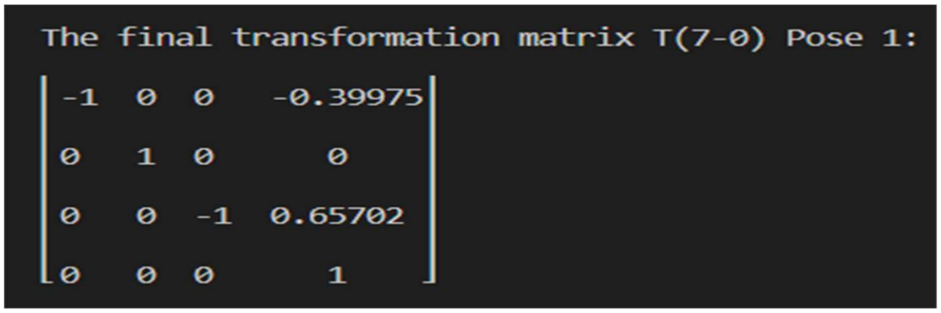


```
The Homogeneous tranformation matrix from frame 7 to frame 0 with all thetas as zero:
⎡0   1  0     0    ⎤
⎢                  ⎥
⎢-1  0  0     0    ⎥
⎢                  ⎥
⎢0   0  1  1.30229 ⎥
⎢                  ⎥
⎣0   0  0     1    ⎦
```

Pose 1: (0, 0, sp.pi/2, sp.pi/2, 0, -sp.pi/2, 0):

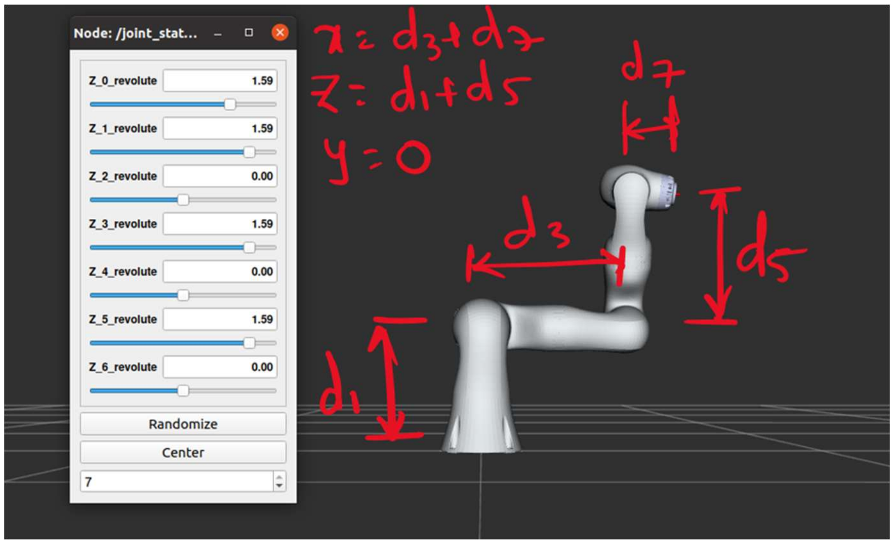


```
Z_0_revolute   0.00
Z_1_revolute   0.00
Z_2_revolute   1.59
Z_3_revolute   1.59
Z_4_revolute   0.00
Z_5_revolute  -1.57
Z_6_revolute   0.00
```

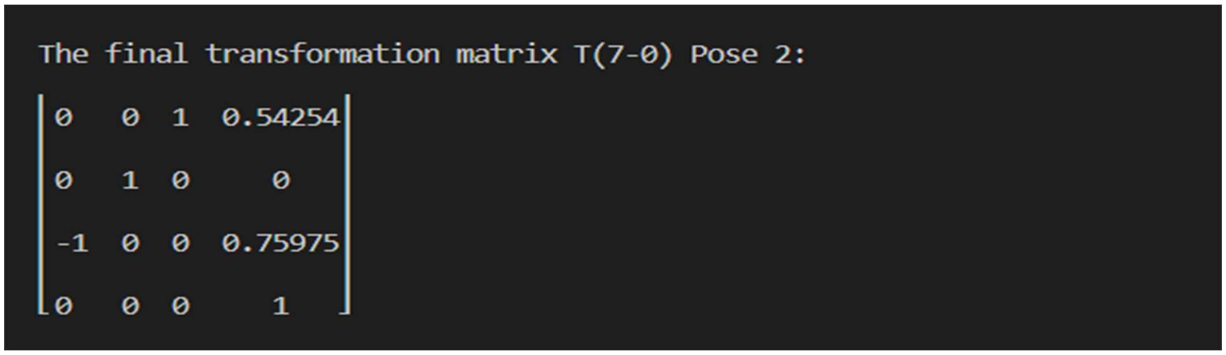


```
The final transformation matrix T(7-0) Pose 1:
|-1   0   0    -0.39975|
| 0   1   0        0   |
| 0   0  -1    0.65702 |
| 0   0   0        1   |
```

Pose 2: (sp.pi/2, sp.pi/2, 0, sp.pi/2, 0, sp.pi/2, 0):

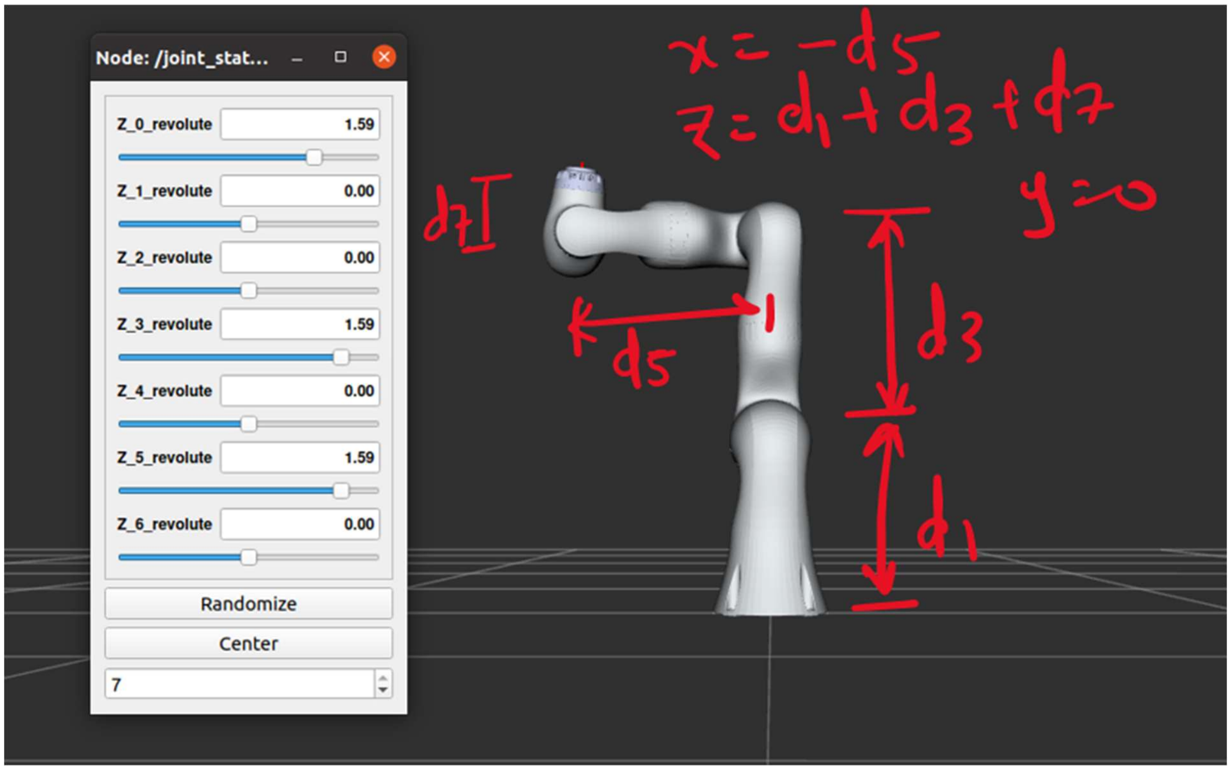


```
Z_0_revolute   1.59
Z_1_revolute   1.59
Z_2_revolute   0.00
Z_3_revolute   1.59
Z_4_revolute   0.00
Z_5_revolute   1.59
Z_6_revolute   0.00
```

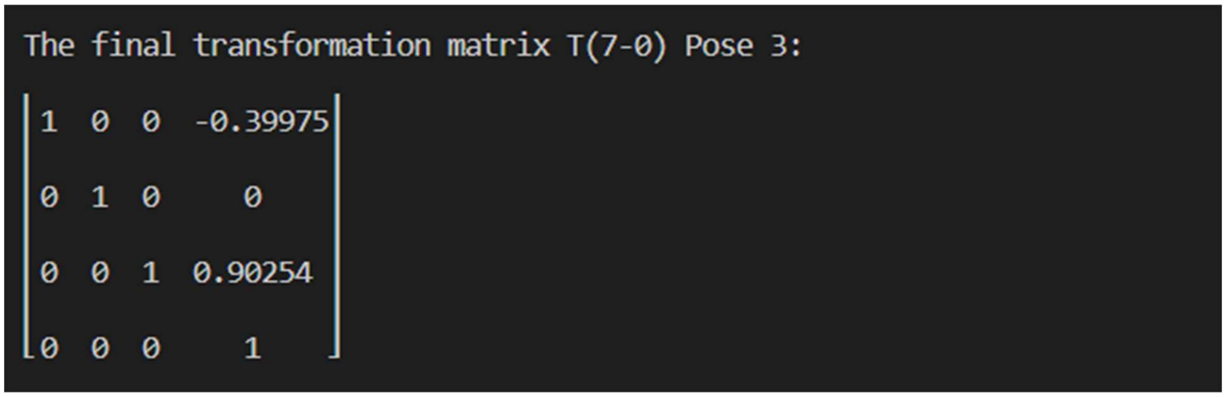


```
The final transformation matrix T(7-0) Pose 2:

⎡0   0  1  0.54254⎤
⎢0   1  0     0   ⎥
⎢-1  0  0  0.75975⎥
⎣0   0  0     1   ⎦
```

Pose 3: (sp.pi/2, 0, 0, sp.pi/2, 0, sp.pi/2, 0):



Z_0_revolute 1.59
Z_1_revolute 0.00
Z_2_revolute 0.00
Z_3_revolute 1.59
Z_4_revolute 0.00
Z_5_revolute 1.59
Z_6_revolute 0.00



```
The final transformation matrix T(7-0) Pose 3:

⎡1  0  0  -0.39975⎤
⎢0  1  0     0    ⎥
⎢0  0  1  0.90254 ⎥
⎣0  0  0     1    ⎦
```

Inverse Kinematics:

We used the second method to compute the Jacobian Matrix where we use the approach of Partial differentiation of the translation component w.r.t. the joint angles.

Once we got all the transformation matrices mentioned in the forward kinematics, we defined another function to compute the Homogeneous transformation matrix to map every frame with respect to the base frame zero. The return of this functions gives all the required Homogeneous Transformation matrices as below:

$$T_1^0 \; ; \; T_2^0 = T_1^0 T_2^1 \; ; \; T_3^0 = T_1^0 \cdot T_2^1 \cdot T_3^2$$

$$T_4^0 = T_1^0 T_2^1 T_3^2 T_4^3 \; ; \; T_5^0 = T_1^0 \cdot T_2^1 \cdot T_3^2 \cdot T_4^3 \cdot T_5^4$$

$$T_6^0 = T_5^0 T_6^5 \; ; \; T_7^0 = T_6^0 \cdot T_7^6$$

After that, we compute the Z axis from the first three elements of third column of these Homogeneous transformation Matrices from corresponding frames to frame 0 and the Xp from the first three elements of the fourth column of the Homogeneous transformation matrix from frame 7 to frame 0.

$$ {}_{n}^{0}J = \begin{bmatrix} \frac{\partial^0 X_p}{\partial q_1} & \cdots & \frac{\partial^0 X_p}{\partial q_i} & \cdots & \frac{\partial^0 X_p}{\partial q_n} \\ {}^0Z_1 & \cdots & {}^0Z_i & \cdots & {}^0Z_n \end{bmatrix} $$

$$J_1 = \begin{bmatrix} dx_p/d\theta_1 \\ Z_1^0 \end{bmatrix}_{6\times1} \quad J_2 = \begin{bmatrix} dx_p/d\theta_2 \\ Z_2^0 \end{bmatrix}_{6\times1}$$

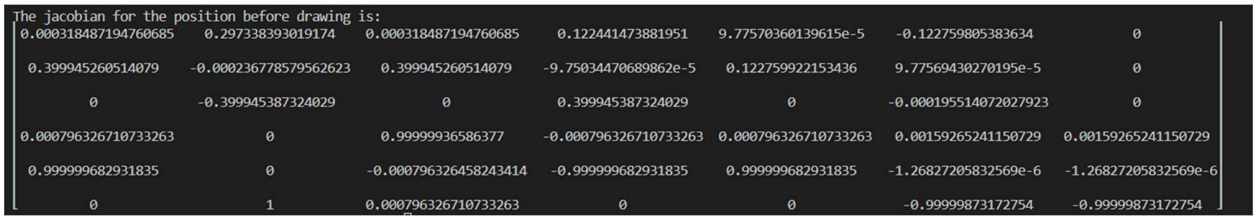
$$J_3 = \begin{bmatrix} dx_p/d\theta_3 \\ Z_3^0 \end{bmatrix}_{6\times1} \quad J_4 = \begin{bmatrix} dx_p/d\theta_4 \\ Z_4^0 \end{bmatrix}_{6\times1}$$

$$J_5 = \begin{bmatrix} dx_p/d\theta_5 \\ Z_5^0 \end{bmatrix}_{6\times1} \quad J_6 = \begin{bmatrix} dx_p/d\theta_6 \\ Z_6^0 \end{bmatrix}_{6\times1}$$

$$J_7 = \begin{bmatrix} dx_p/d\theta_7 \\ Z_7^0 \end{bmatrix}_{6\times1}$$

$$J = \begin{bmatrix} J_1 & J_2 & J_3 & J_4 & J_5 & J_6 & J_7 \end{bmatrix}_{7\times6}$$

The Jacobian matrix for the configuration [90, 0, 0, -90, 0 , 90, 0] before drawing the circle for validation is as below:



```
The jacobian for the position before drawing is:
⎡0.000318487194760685    0.297338393019174    0.000318487194760685     0.122441473881951    9.77570360139615e-5    -0.122759805383634            0        ⎤
⎢ 0.399945260514079    -0.000236778579562623    0.399945260514079    -9.75034470689862e-5    0.122759922153436    9.77569430270195e-5            0        ⎥
⎢        0              -0.399945387324029           0               0.399945387324029           0              -0.000195514072027923           0        ⎥
⎢0.000796326710733263           0              0.99999936586377     -0.000796326710733263  0.000796326710733263  0.00159265241150729   0.00159265241150729 ⎥
⎢ 0.999999682931835             0           -0.000796326458243414   -0.999999682931835    0.999999682931835    -1.26827205832569e-6  -1.26827205832569e-6⎥
⎣        0                      1            0.000796326710733263           0                     0              -0.99999873172754     -0.99999873172754  ⎦
```

The general Jacobian matrix with all theta variables will be printed on the output of the python file:

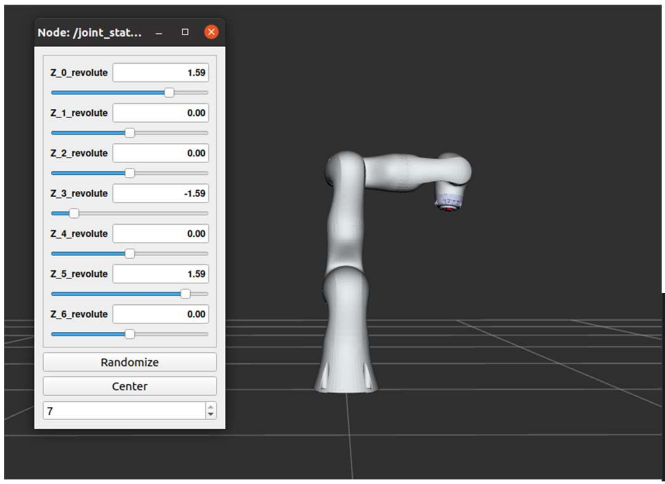
KUKA_7X7_jacobian.py

Inverse Kinematics Validation:

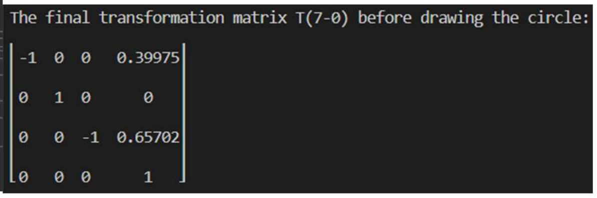
We tried to get the validation of this Jacobian matrix using the technique we used in the fourth homework, but our python script stuck at the for loop while computing the inverse of the Jacobian Matrix at the computed pose.

Hence, we fixed the third joint as we did earlier in the Homework four, hence theta3 was fixed to zero for further inverse Kinematics validation. Now the Jacobian is 6X6 and we also added small radian values to the thetas (0.0001 rad) to avoid divide by zero error and singularity.

Initial pose considered for the IK validation is : [90, 0, -90, 0 , 90, 0]



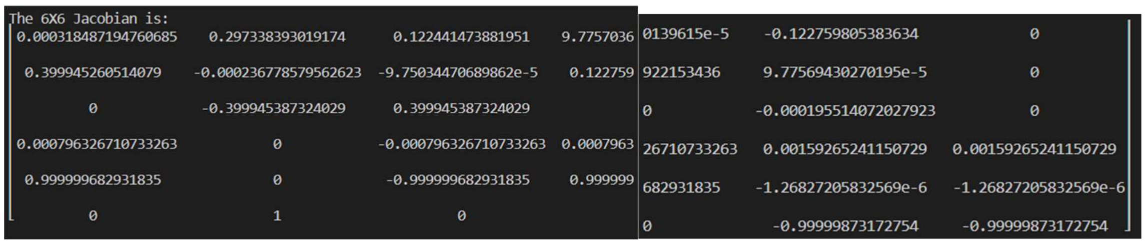
```
Z_0_revolute  1.59
Z_1_revolute  0.00
Z_2_revolute  0.00
Z_3_revolute  -1.59
Z_4_revolute  0.00
Z_5_revolute  1.59
Z_6_revolute  0.00
```



```
The final transformation matrix T(7-0) before drawing the circle:
⎡-1  0  0   0.39975⎤
⎢0   1  0      0   ⎥
⎢0   0  -1  0.65702⎥
⎣0   0  0      1   ⎦
```

We will plot the circle in X-Y plane, and from the transformation matrix at the initial pose the circle will be drawn at the height of 0.657mm, with X = 0.39975 mm as center.

The Jacobian Matrix (6X6) for the initial pose: (This will be in the output screen as well)



```
The 6X6 Jacobian is:
 0.000318487194760685    0.297338393019174      0.12244147381951     9.7757036 0139615e-5    -0.122759805383634         0
 0.399945260514079    -0.000236778579562623 -9.75034470689862e-5    0.122759 922153436     9.77569430270195e-5          0
        0               -0.399945387324029      0.399945387324029           0           -0.000195514072027923         0
 0.000796326710733263          0            -0.000796326710733263 0.0007963 26710733263   0.00159265241150729   0.00159265241150729
 0.999999682931835             0              -0.999999682931835    0.999999 682931835  -1.26827205832569e-6  -1.26827205832569e-6
        0                      1                      0                     0             -0.99999873172754     -0.99999873172754
```

Now, we wanted to draw a circle of r= 10 cm at the XY plane at Z= 657mm in 5 secs. The equation of the circle is :

$$x = \mathrm{r} \times \cos\theta + 0.366 \text{ mm}$$
$$y = \mathrm{r} \times \sin\theta \text{ mm}$$
$$z = 0.65702 \text{ mm}$$

To complete the circle in 5 sec, the angular velocity is given by:

$$\omega = \frac{2\pi}{5}$$

Now, the velocity matrix of the end effector is given by:

$$\mathrm{x\_d} = -\mathrm{r} \times \sin(\omega t) \times \omega$$

$$y\mathrm{\_d} = \mathrm{r} \times \cos(\omega t) \times \omega$$

z_d = 0

Hence, $[x\mathrm{\_dot}] = [\mathrm{x\_d\ y\_d\ z\_d}\ \omega_x\ \omega_y\ \omega_z]$

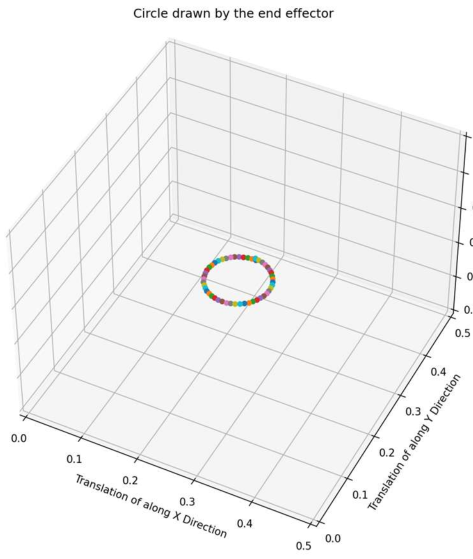
As $\omega_x = \omega_y = \omega_z = \mathrm{z\_d} = 0; [x\mathrm{\_dot}] = [\mathrm{x\_d\ \ y\_d\ \ 0\ 0\ 0\ 0}]$

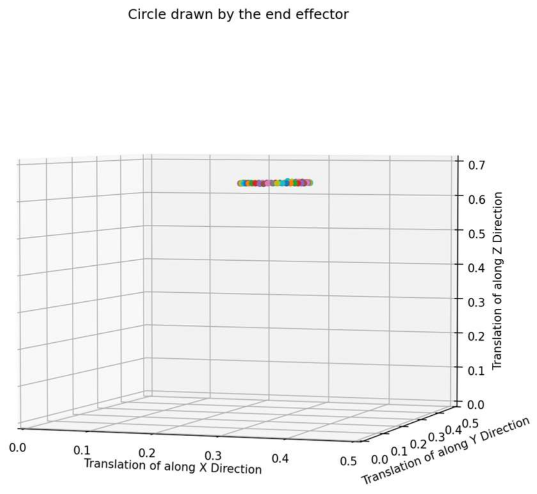
Using the Inverse Velocity Kinematics and Jacobian Matrix, the corresponding Joint Angular Velocity Matrix are calculated as:

$$[\theta\mathrm{\_dot}] = J^{-1}[x\mathrm{\_dot}]$$

Once we get these theta_dot values, using the integration we can get the individual joint angles. Using these joint angles, we verify the trajectory tracked by the end effector.

The final plot of the circle is as below, which validated the Inverse Kinematics:

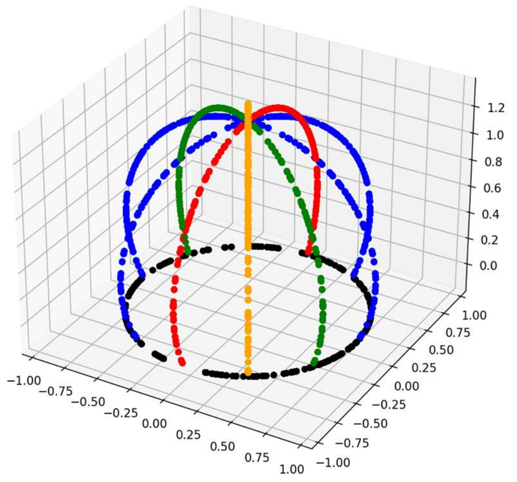




Workspace Study:

For workspace study we plotted the end effector position for different configuration. The maximum reach of our robot is 820 mm. So the end effector can reach any point in the space within its maximum reach and the angle constraints. So to study the workspace we plotted the maximum reach position In all direction giving us a rough idea of the workspace. We just plotted the maximum reach points because being a 7 Dof robot, it can reach any point within its limit.

As we are computing more that 1000 positions of the end effector, the program takes some time to give the below plots:



Assumptions:

- All links are rigid.
- Cobot is stationary.
- The roll is of 8kg (or less than 14 kgs) and the packaging machine should be in the 820mm distance from the Cobot's base (within the workspace of the cobot.)
- To prove inverse kinematics using the python script, we assumed the third joint to be fixed hence theta3 = 0 to get the Square Jacobian Matrix.
- We are using the camera as the end effector, so that we can see the picking object and the loading position in the Rviz which will validate our application.

Control Method:

In an ROS package the controllers are defined using yaml files stored in config folder. In our package we have defined the controller in "config_kuka_controllers.yaml" file. For our robot we used JointPositionControllers of position_controllers type. These controllers are used for joined which are to be controlled by position commands. They transfer the position input as a position command to the hardware interface. The joint state controller is used to publish the joint states of the arm with a publishing rate set to 50 Hz. To simulate the robot in Gazebo with those controllers, we added the corresponding plugin to the URDF file. To use ros_control within Gazebo, the URDF model of the robot must include one additional element, transmissions. Mechanical transmission is implemented by transmission interfaces. We are using Teleop to manually operate the robotic arm which gives it the functionality of cobot i.e collaborative robot.

Gazebo and RViz Visualization:

We spawned our robot in rviz and gazebo, controlled in gazebo using Teleop and visualized it inside the rviz. Please find the video of us operating the KUKA cobot arm using Teleop, and we added the camera so the feed from camera can be visualized using the Rviz. The camera shows that the arm goes till the object of interest and then again using teleop we took the arm to the desired placing position. This can also be seen in the Rviz camera feed. Hence, we verified our pick and place operation scope as mentioned in the proposal.

1. We launched the submitted package in the Gazebo, then launched Rviz and streamed the camera feed. The link to the video:

   https://drive.google.com/file/d/1t82u_cCCnZdAm0vJG0dwsVCfcxsx7XGa/view?usp=share_link

2. We controlled the cobot arm using teleop and showed that the arm goes to the pick place and then took the arm to the drop place. The pick and drop places confirmed by referring the camera feed from Rviz.

   https://drive.google.com/file/d/1kvMZsTBaxkdASuH5KDzIhSfBdr_wc2f9/view?usp=share_link

Problems Faced:

We faced many problems during the entire process of the project. And it is due to these problems we learnt a lot of things. Below is the list of the problems we faced and how we resolved the issue.

1. Forward Kinematics Validation: When we first exported the urdf and launched the robot in rviz, we tried validating the forward kinematics by comparing the coordinates of the end effector obtained from rviz whit that of transformation matrix. The coordinates didn't match each other. So initially we thought we were facing this issue due to wrong DH parameters, so we calculated DH parameters again for different initial orientation, and compared the coordinates again, but still the same problem, they didn't match. We even changed the frame assignment but still no solution. Then at last we tried investigating the exported urdf file, in which we found that the axis orientation of the frames was different for different joints. It was "0 0 1" for some and "0 0 -1" for some, so we made it uniform as "0 0 1" for all the joints, which finally resolved the issue.

2. Inverse Kinematics:
   We computed the Jacobian initially assuming all the joints free to rotate, as we have seven DoF we got a 7X6 Jacobian matrix. We were having issues when we went for plotting the end effector trajectory in the for loop, there were singularities and the program stuck in the loop. Hence, we solved this issue by making theta3 fixed, and Jacobian 6X6, a square matrix. Then we got the circular trajectory that we wanted to prove the Inverse Kinematics.

3. Spawning in gazebo with after adding the plugins: We faced few issues while spawning our robot in gazebo world after we added ros_control gazebo plugin in our urdf file. We faced parser error due to the wrong gazebo plugin format, which we resolved by changing the format. But when we finally be able to spawn the robot in gazebo, it would stop responding after few seconds. But when we removed the gazebo plugin, the software would work smoothly. Initially I thought there might be some installation issues with my gazebo as I didn't get any error, but the gazebo would just stop responding. So, I tried spawning the model in my partner's laptop, but the same problem persisted. I tried it on 3rd laptop but still faced the same issue. Researched a lot about this issue on different forums but couldn't find any solution. So as a last try, I tried spawning the model in 4th laptop, and the gazebo worked smoothly!

Lessons Learned:

1. Cad modelling of the Robot
2. Frame assignment to joints while exporting urdf file
3. Exporting urdf file
4. DH parameters
5. Forward Kinematics
6. Inverse Kinematics
7. ROS 1 concepts like package creation, subscriber, publisher, etc.
8. Rviz visualization
9. Gazebo simulation
10. Python programming

Conclusion:

1. Successfully modelled our robot in solidworks.
2. Calculated DH parameters
3. Validated forward and inverse kinematics.
4. Created ROS package of our robot
5. Visualized and simulated in rviz and gazebo

Future Work:

Future scope includes integrating of lidar sensor, image recognition to automatically locate the objects. Obstacle and human detection will be also our future scope, as we wanted Cobot to collaborate with human without compromising the safety of the personnel involved.

We tried to integrate the gripper mechanism, but in the available time span we were not able to integrate the gripping action. Our future scope will be to integrate a gripper and perform the pick and place operation in Gazebo.

We are also planning to integrate more packaging machines and recreate the scheduling algorithm so that same cobot can load the rolls on these packaging machines.

References:

1. http://wiki.ros.org
2. https://www.kuka.com/en-us/products/robotics-systems/industrial-robots/lbr-iiwa
3. https://ros-planning.github.io/moveit_tutorials/
4. Robot Modeling and Control, - Mark W. Spong, Seth Hutchinson, and M. Vidyasagar