Robotic Arm: Pick & Place

Project Report

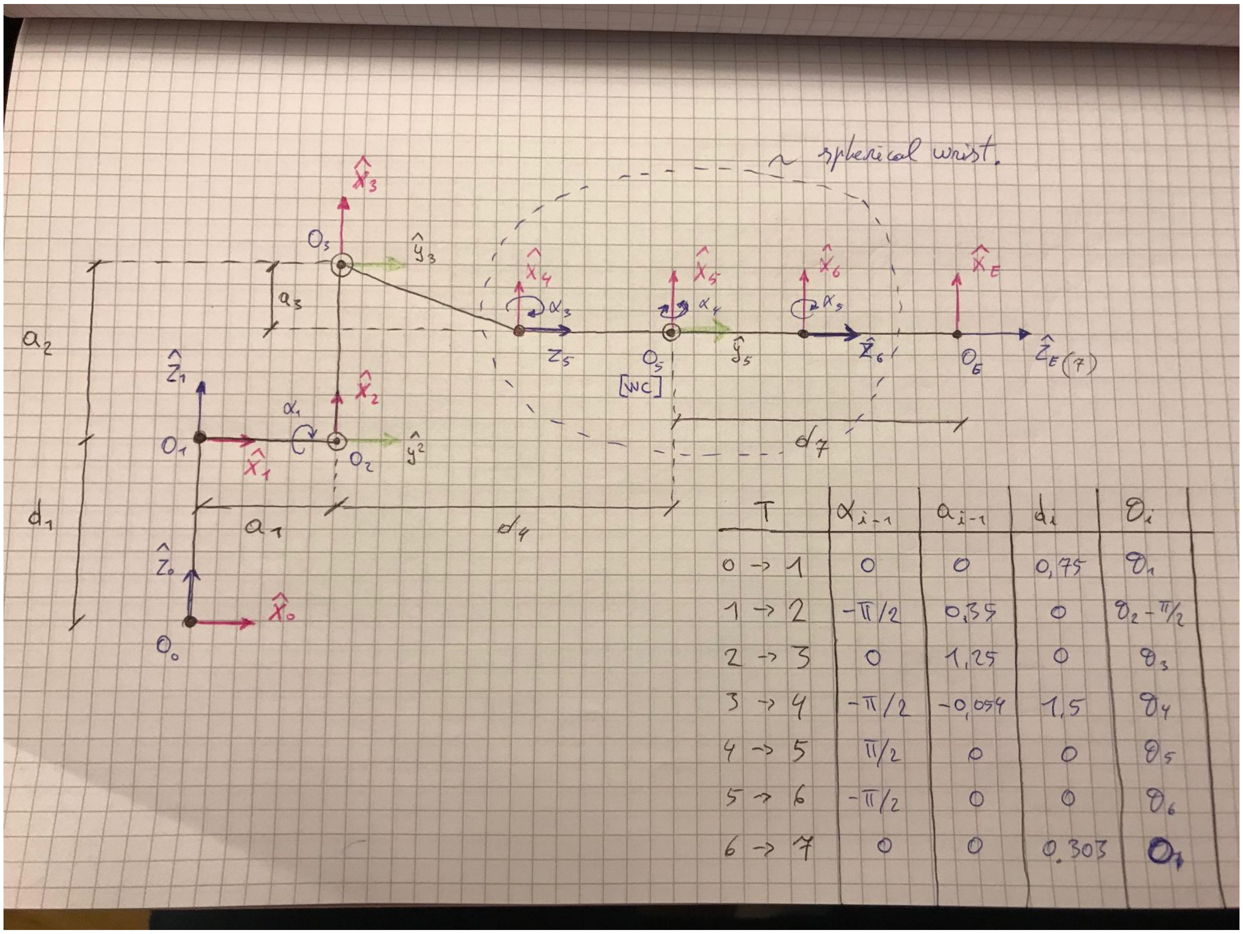
Virtual Machine: The virtual machine has been setup and the Linux distro is in use as per guidelines. The setting for the virtual machine is 16G of RAM, 6 cores/12 threads and max GPU memory. At the moment not experimenting latency. Can continue to work on the virtualization.

ROS: In this section I have dive into the ROS environment and concepts. For this I have read as well "*A gentle introduction to ROS*". Currently trying to get into more details by reading "*A Systematic Approach to Learning Robot Programming with ROS*". The section on the URDF could be a bit improved. Overall ROS is not only an "OS" is a set of tools and packages for different purposes and getting a good overview is not immediate. Again indication of reference material papers/books for Rviz/Gazebo, ROS etc would be appreciated as there might more choices but I would trust your recommendation as hopefully you have had the chance to see more of them.

Project repository: The project has been downloaded from git in a catkin workspace and all the steps to setup and run the project and experiment with Gazebo and Rviz have been taken.

Kinematics: This was interesting chapter. I have built a notebook where i have experimented with the concepts, built functions which generate rotation ,homogeneous transforms and other utilities. Schematics tend to be quickly complex with lot of symbols maybe a multi-layer schematic is helpful here. Both forward and inverse kinematics are understood and also the challenge in computation for finding a good solution in case of IK. The spherical wrist was also interesting. My understanding is that it is an extension of a 3DOF manipulator such that the end-effector can take just any orientation due to the presence of the spherical wrist.

The following diagram has been used to analyse the kinematics and derive the DH parameter table.



T	α_{i-1}	a_{i-1}	d_i	θ_i
0 -> 1	0	0	0,75	θ_1
1 -> 2	$-\pi/2$	0,35	0	$\theta_2 - \pi/2$
2 -> 3	0	1,25	0	θ_3
3 -> 4	$-\pi/2$	-0,054	1,5	θ_4
4 -> 5	$\pi/2$	0	0	θ_5
5 -> 6	$-\pi/2$	0	0	θ_6
6 -> 7	0	0	0.303	θ_7

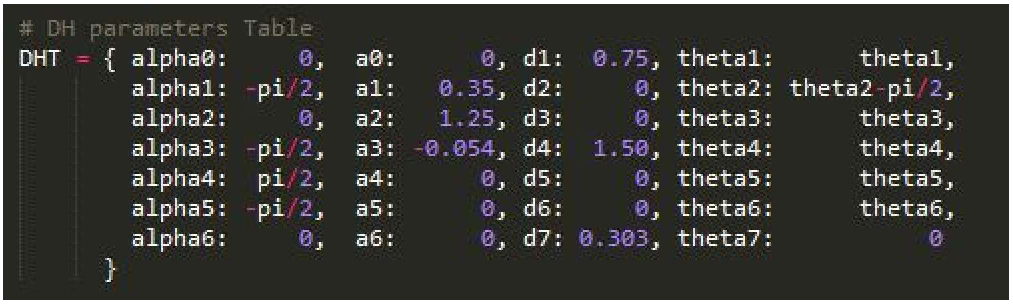
In order to not complicate the diagram, only to unit axis have been shown for each joint position. The third is derived from the hand right coordinate system rule. The spherical wrist with its center has been shown together with the end effector reference frame and the base link. The diagram has been annotated with their distance and angles.

Notice:

- z0, z1 are collinear.
- Z1 ortogonal to z2
- Z2 || z3
- Z4 ortogonal to z3
- Z5 ortogonal to z4
- Z6 ortogonal to z5

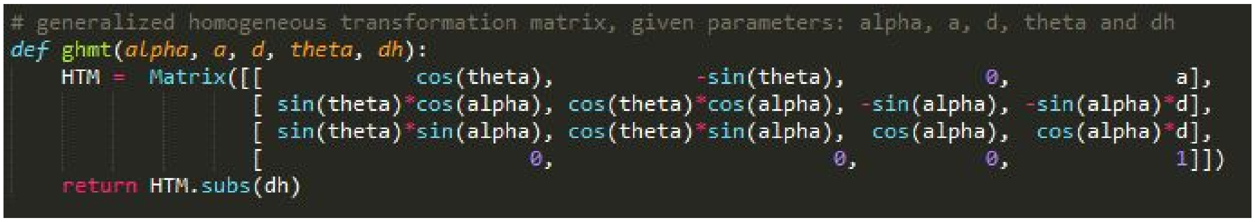
- Ze, z6 are collinear

The above, considered also with a good choice for the origin of each join lead to the description of simplified DH parameters. Base on this analysis the DH Table was derived and for clarity indicated below as in the code:



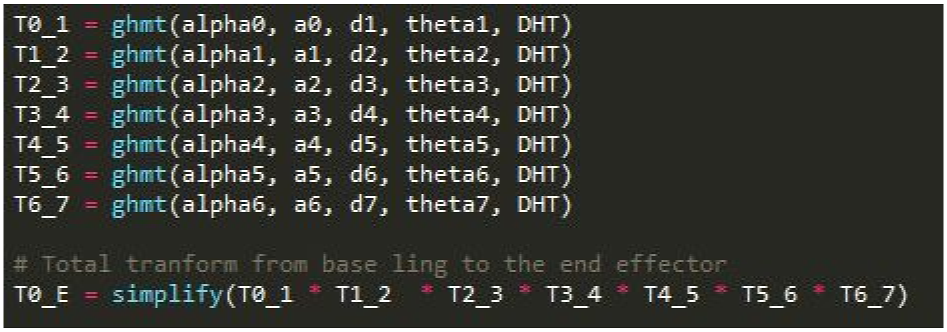
```
# DH parameters Table
DHT = { alpha0:     0,  a0:      0, d1:  0.75, theta1:      theta1,
        alpha1: -pi/2,  a1:   0.35, d2:     0, theta2: theta2-pi/2,
        alpha2:     0,  a2:   1.25, d3:     0, theta3:      theta3,
        alpha3: -pi/2,  a3: -0.054, d4:  1.50, theta4:      theta4,
        alpha4:  pi/2,  a4:      0, d5:     0, theta5:      theta5,
        alpha5: -pi/2,  a5:      0, d6:     0, theta6:      theta6,
        alpha6:     0,  a6:      0, d7: 0.303, theta7:           0
      }
```

The homogeneous transform can be generated in a simplified version by using only rotation around X, Z axis and related displacement if the convention used in choosing axis are done as in the above schematic.



```
# generalized homogeneous transformation matrix, given parameters: alpha, a, d, theta and dh
def ghmt(alpha, a, d, theta, dh):
    HTM =  Matrix([[            cos(theta),            -sin(theta),           0,             a],
                   [ sin(theta)*cos(alpha), cos(theta)*cos(alpha), -sin(alpha), -sin(alpha)*d],
                   [ sin(theta)*sin(alpha), cos(theta)*sin(alpha),  cos(alpha),  cos(alpha)*d],
                   [                     0,                     0,           0,             1]])
    return HTM.subs(dh)
```

The individual Transforms can then be easily calculated as:



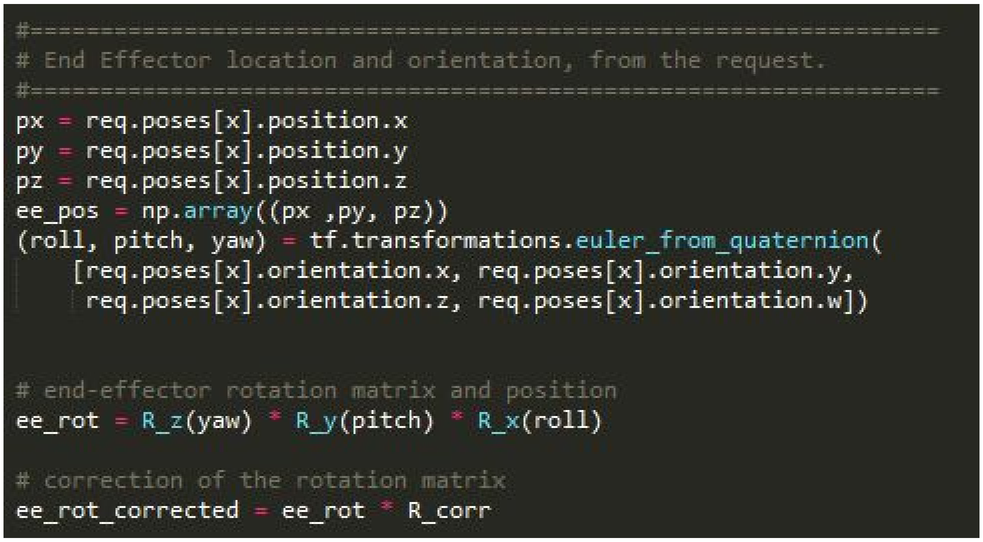
```
T0_1 = ghmt(alpha0, a0, d1, theta1, DHT)
T1_2 = ghmt(alpha1, a1, d2, theta2, DHT)
T2_3 = ghmt(alpha2, a2, d3, theta3, DHT)
T3_4 = ghmt(alpha3, a3, d4, theta4, DHT)
T4_5 = ghmt(alpha4, a4, d5, theta5, DHT)
T5_6 = ghmt(alpha5, a5, d6, theta6, DHT)
T6_7 = ghmt(alpha6, a6, d7, theta7, DHT)

# Total tranform from base ling to the end effector
T0_E = simplify(T0_1 * T1_2  * T2_3 * T3_4 * T4_5 * T5_6 * T6_7)
```

For the purpose of the code, only the first 3 will be used for the purpose of calculating the position of the wrist center with respect to the reference base frame.

Demo Mode: The robotic arm have been executed in demo mode to experiment with different features. Essentially the position of the object ot be picked up is been given in terms of cartesian coordinates. This seems to be the position of the reference frame origin of the grip. There is a phase where the path is being planned as a sequence of points in space which I understand is the planning of the path. Who decides how many points ?

IK_Server: when the simulation is run in test mode the angles of all the joints are been calculated. The flow of the program is as follows:

1) Get position and roll, pitch, yaw angles for the end-effector with the calculation request from the client (for correction rotational matrix refer to the code). having position vector and rotation matrix, we can build the Homogenous transform which represent the transform between end-effector and base frame.

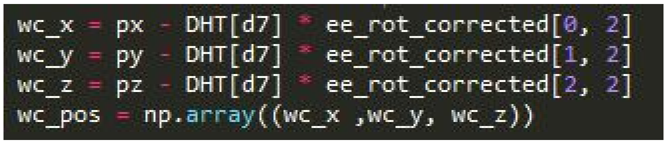


```
#=================================================================
# End Effector location and orientation, from the request.
#=================================================================
px = req.poses[x].position.x
py = req.poses[x].position.y
pz = req.poses[x].position.z
ee_pos = np.array((px ,py, pz))
(roll, pitch, yaw) = tf.transformations.euler_from_quaternion(
    [req.poses[x].orientation.x, req.poses[x].orientation.y,
     req.poses[x].orientation.z, req.poses[x].orientation.w])


# end-effector rotation matrix and position
ee_rot = R_z(yaw) * R_y(pitch) * R_x(roll)

# correction of the rotation matrix
ee_rot_corrected = ee_rot * R_corr
```

2) Generate a homogeneous transform which represent the T0_Grip. The components l, m, n are the component of the rotation matrix while p is the end effector position.

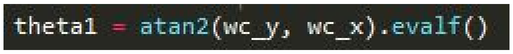
$$\begin{bmatrix} l_x & m_x & n_x & p_x \\ l_y & m_y & n_y & p_y \\ l_z & m_z & n_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3) Since there is a displacement along z-axis, use the column of the above rotation matrix, part of the homogeneous transform (the corrected matrix) to get the unit vector indicating the unit z axis of the displacement.
4) Calculate the wrist center in terms of cartesian coordinates referring to the base frame. In order to decouple the IK problem we need as next to calculate the wrist center position.



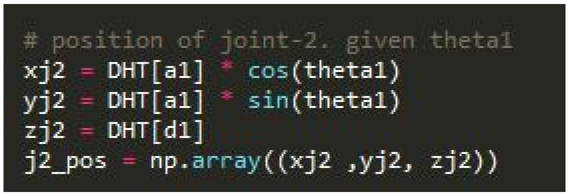
```
wc_x = px - DHT[d7] * ee_rot_corrected[0, 2]
wc_y = py - DHT[d7] * ee_rot_corrected[1, 2]
wc_z = pz - DHT[d7] * ee_rot_corrected[2, 2]
wc_pos = np.array((wc_x ,wc_y, wc_z))
```

5) Calculate the Theta_1 angle with the atan2. This is immediate since the wrist center coordinates are already referring to the base frame.



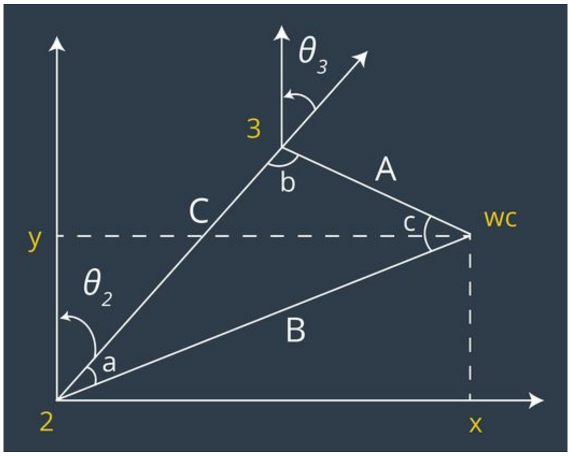
```
theta1 = atan2(wc_y, wc_x).evalf()
```

6) Calculate the position of the Joint 2 given Theta_1.

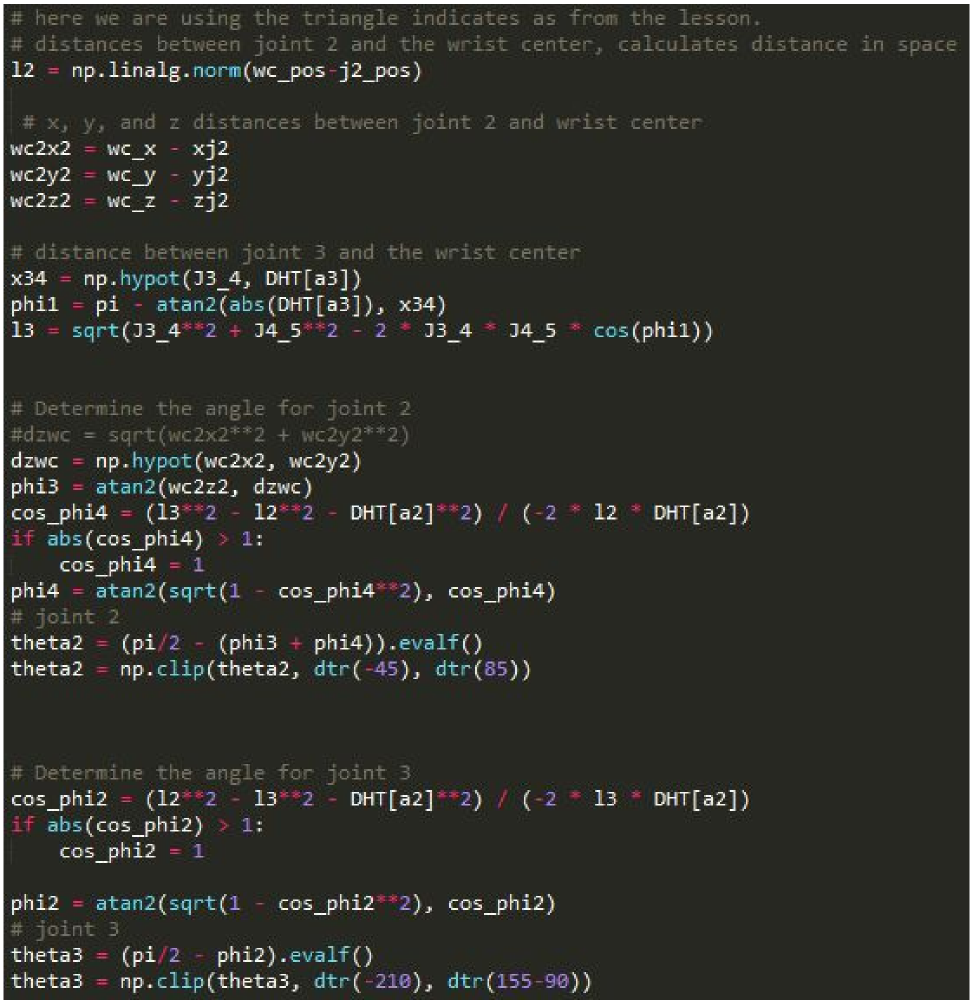


```
# position of joint-2. given theta1
xj2 = DHT[a1] * cos(theta1)
yj2 = DHT[a1] * sin(theta1)
zj2 = DHT[d1]
j2_pos = np.array((xj2 ,yj2, zj2))
```

7) Once we know the position of Joint 2, can use the triangle and related trigonometry trigonometry to get the values for Theta_2, Theta_3.

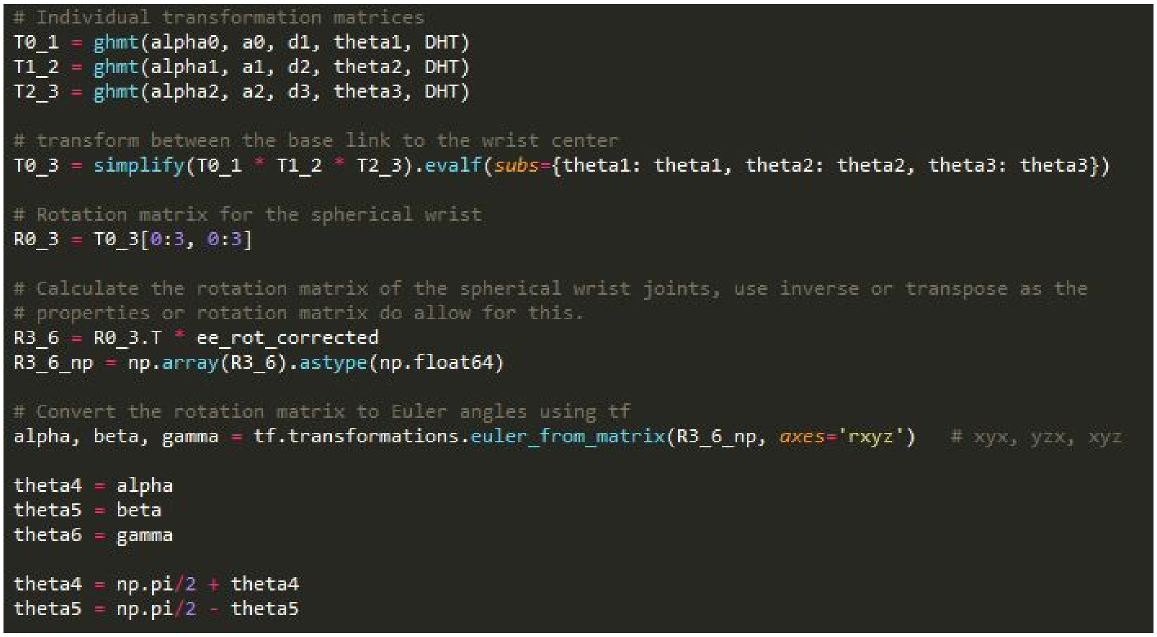


The following code has been used with the above triangle:



```
# here we are using the triangle indicates as from the lesson.
# distances between joint 2 and the wrist center, calculates distance in space
l2 = np.linalg.norm(wc_pos-j2_pos)

 # x, y, and z distances between joint 2 and wrist center
wc2x2 = wc_x - xj2
wc2y2 = wc_y - yj2
wc2z2 = wc_z - zj2

# distance between joint 3 and the wrist center
x34 = np.hypot(J3_4, DHT[a3])
phi1 = pi - atan2(abs(DHT[a3]), x34)
l3 = sqrt(J3_4**2 + J4_5**2 - 2 * J3_4 * J4_5 * cos(phi1))


# Determine the angle for joint 2
#dzwc = sqrt(wc2x2**2 + wc2y2**2)
dzwc = np.hypot(wc2x2, wc2y2)
phi3 = atan2(wc2z2, dzwc)
cos_phi4 = (l3**2 - l2**2 - DHT[a2]**2) / (-2 * l2 * DHT[a2])
if abs(cos_phi4) > 1:
    cos_phi4 = 1
phi4 = atan2(sqrt(1 - cos_phi4**2), cos_phi4)
# joint 2
theta2 = (pi/2 - (phi3 + phi4)).evalf()
theta2 = np.clip(theta2, dtr(-45), dtr(85))



# Determine the angle for joint 3
cos_phi2 = (l2**2 - l3**2 - DHT[a2]**2) / (-2 * l3 * DHT[a2])
if abs(cos_phi2) > 1:
    cos_phi2 = 1

phi2 = atan2(sqrt(1 - cos_phi2**2), cos_phi2)
# joint 3
theta3 = (pi/2 - phi2).evalf()
theta3 = np.clip(theta3, dtr(-210), dtr(155-90))
```

8) Inverse Orientation problem. Here we first calculate the rotation matrix of the spherical wrist via forward kinematics. The rotation is a partition of the homogeneous transform from the base link to the wrist center. The transform from wrist center to end effector is calculated knowing the step 0-E and 0-3 which gives us step 3-6. In this case we used the Transposed matrix rather then the inverse calculation due to the fact they are the same as per properties of the Rotation matrix.

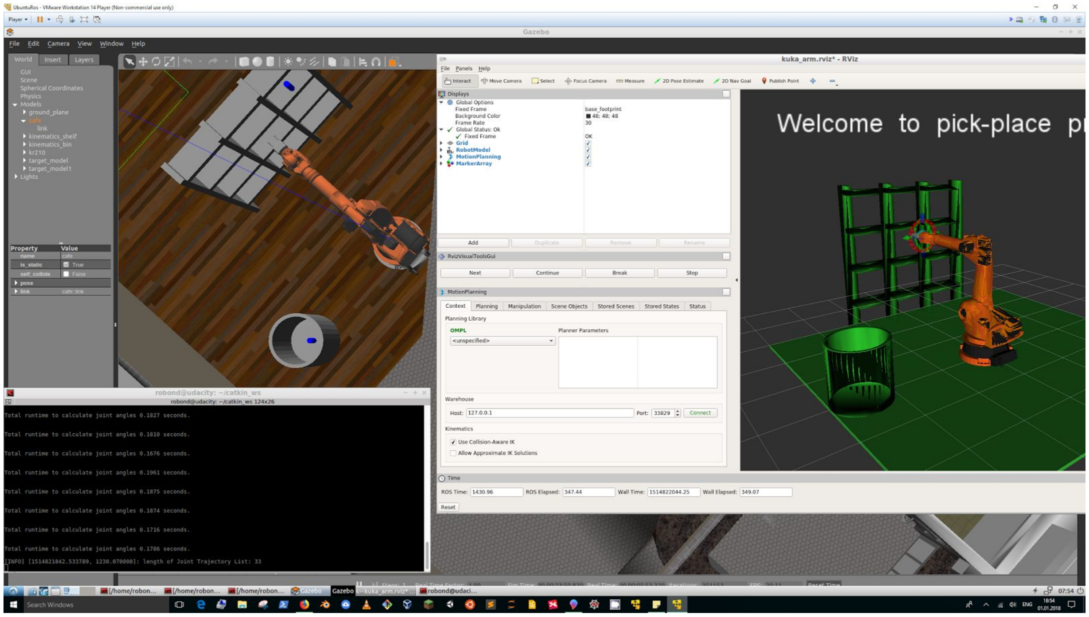


```
# Individual transformation matrices
T0_1 = ghmt(alpha0, a0, d1, theta1, DHT)
T1_2 = ghmt(alpha1, a1, d2, theta2, DHT)
T2_3 = ghmt(alpha2, a2, d3, theta3, DHT)

# transform between the base link to the wrist center
T0_3 = simplify(T0_1 * T1_2 * T2_3).evalf(subs={theta1: theta1, theta2: theta2, theta3: theta3})

# Rotation matrix for the spherical wrist
R0_3 = T0_3[0:3, 0:3]

# Calculate the rotation matrix of the spherical wrist joints, use inverse or transpose as the
# properties or rotation matrix do allow for this.
R3_6 = R0_3.T * ee_rot_corrected
R3_6_np = np.array(R3_6).astype(np.float64)

# Convert the rotation matrix to Euler angles using tf
alpha, beta, gamma = tf.transformations.euler_from_matrix(R3_6_np, axes='rxyz')   # xyx, yzx, xyz

theta4 = alpha
theta5 = beta
theta6 = gamma

theta4 = np.pi/2 + theta4
theta5 = np.pi/2 - theta5
```

9) np.linalgebra and np.hypot used to calculate distances

10) Clamped the calculated angles as for physical limitation of the joints.

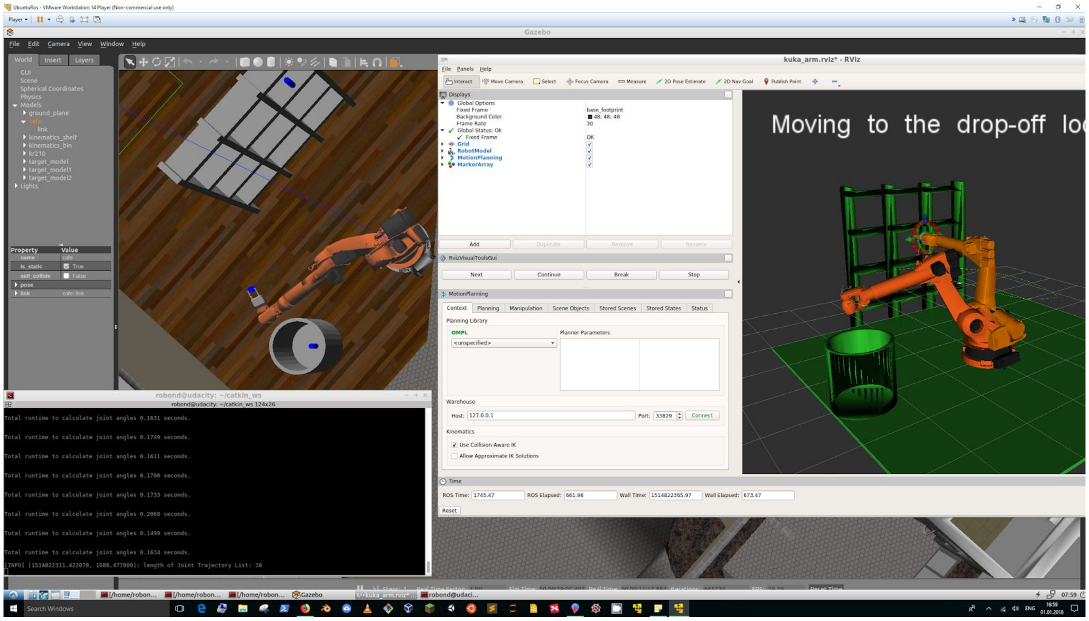
Calculation will happen for each point in the path. Currently I see 0.15 second to calculate the joint angles per each cartesian coordinate. It looks ok but the robot looks slow to move from one point to the other. Randomly happens that the grip although in the right position and with the right closure does not pick up the object. I have modified the .cpp file as per the Q&A so to give to the grip more time but this is not solving always the problem. According to my mentor this is a known bug with Gazebo. A timing function has been inserted so to see how much it takes to calculate all the joint angles for a certain point given a request. The following pictures are from the simulation environment showing Gazebo, Rviz and IK_Server in action.



Fixed Frame base_footprint
Background Color 48; 48; 48
Frame Rate 30

ROS Time: 1430.96 ROS Elapsed: 347.44 Wall Time: 1514822044.25 Wall Elapsed: 349.07

```
Total runtime to calculate joint angles 0.1827 seconds.

Total runtime to calculate joint angles 0.1810 seconds.

Total runtime to calculate joint angles 0.1676 seconds.

Total runtime to calculate joint angles 0.1961 seconds.

Total runtime to calculate joint angles 0.1875 seconds.

Total runtime to calculate joint angles 0.1874 seconds.

Total runtime to calculate joint angles 0.1716 seconds.

Total runtime to calculate joint angles 0.1786 seconds.
[INFO] [1514821842.533789, 1230.070000]: length of Joint Trajectory List: 33
```

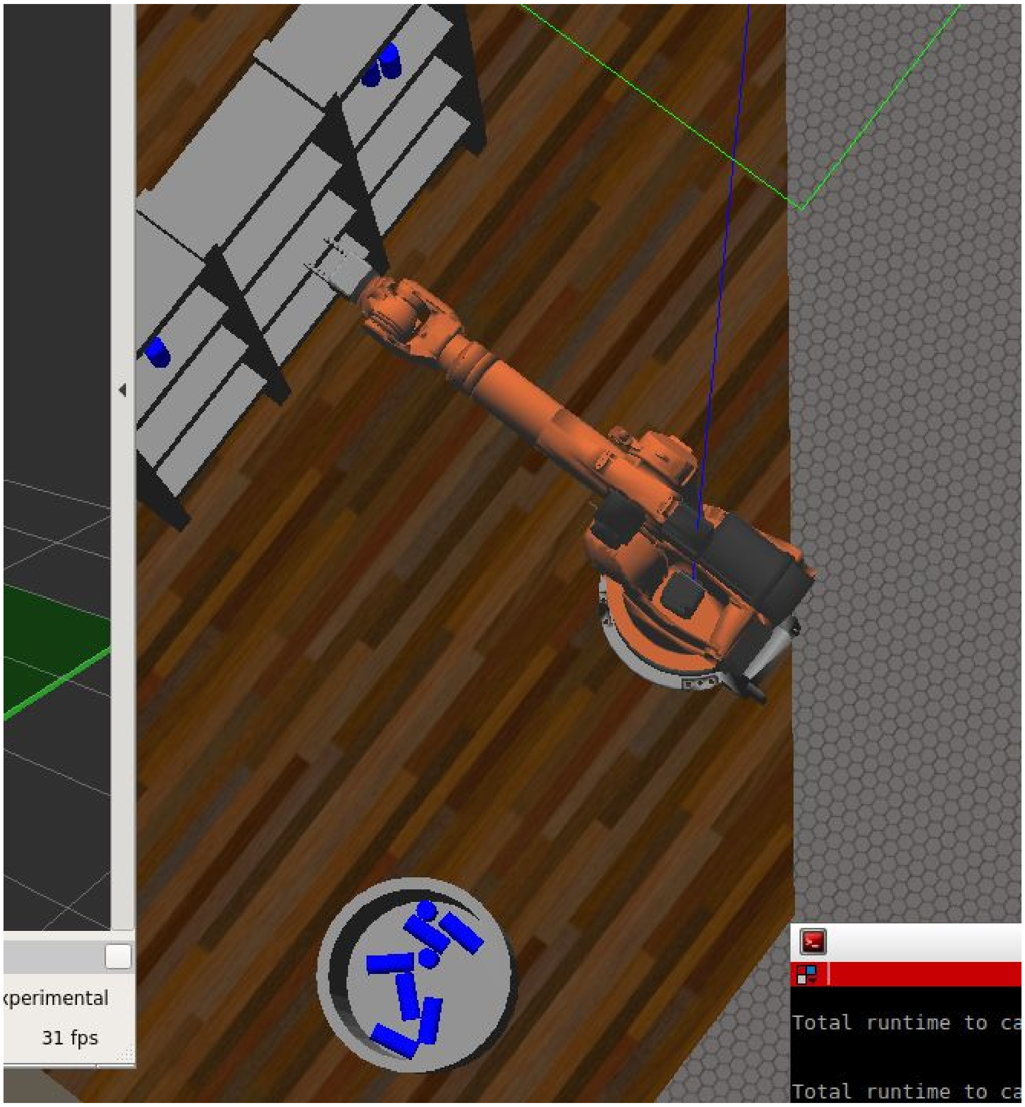


Fixed Frame base_footprint
Background Color 48; 48; 48
Frame Rate 30

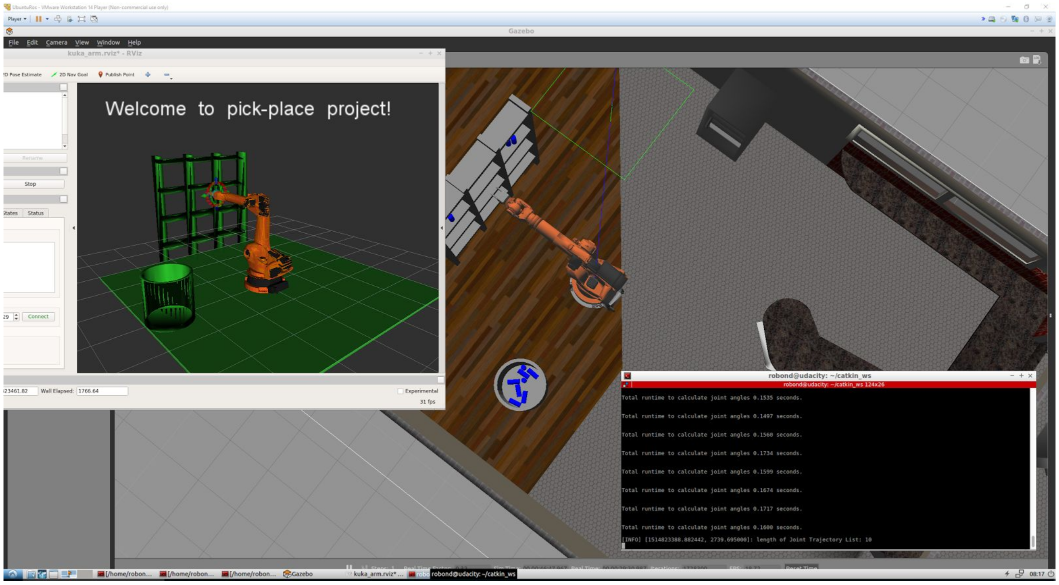
ROS Time: 1745.47 ROS Elapsed: 661.96 Wall Time: 1514822365.97 Wall Elapsed: 673.47

```
Total runtime to calculate joint angles 0.1631 seconds.

Total runtime to calculate joint angles 0.1749 seconds.

Total runtime to calculate joint angles 0.1611 seconds.

Total runtime to calculate joint angles 0.1790 seconds.

Total runtime to calculate joint angles 0.1733 seconds.

Total runtime to calculate joint angles 0.2060 seconds.

Total runtime to calculate joint angles 0.1499 seconds.

Total runtime to calculate joint angles 0.1634 seconds.
[INFO] [1514822311.422078, 1688.477000]: length of Joint Trajectory List: 38
```

Results: In this project knowledge about the ROS, Gazebo, Rviz and python have been put into practice. Python code which calculate the inverse kinematics as a ROS service has been implemented and tested in the simulation. The results do fulfill the project specification for a submission. Randomly, the grip seems not working as expected but the trajectory calculation (Rviz) and the movement observed in Gazebo seems to be correct. The picture below show several runs. 8 cylinders are in the bin, 2 have remain in the shelf due to grip not picking them up.



```
Total runtime to ca

Total runtime to ca
```


UbuntuRos - VMware Workstation 14 Player (Non-commercial use only)
Player
Gazebo
File Edit Camera View Window Help
kuka_arm.rviz* - RViz
2D Pose Estimate
2D Nav Goal
Publish Point
Rename
Stop
States
Status
Connect
Welcome to pick-place project!
Experimental
31 fps
08:17

Wall Elapsed: 1766.64

```
robond@udacity: ~/catkin_ws
robond@udacity: ~/catkin_ws 124x26
Total runtime to calculate joint angles 0.1535 seconds.

Total runtime to calculate joint angles 0.1497 seconds.

Total runtime to calculate joint angles 0.1560 seconds.

Total runtime to calculate joint angles 0.1734 seconds.

Total runtime to calculate joint angles 0.1599 seconds.

Total runtime to calculate joint angles 0.1674 seconds.

Total runtime to calculate joint angles 0.1717 seconds.

Total runtime to calculate joint angles 0.1600 seconds.
[INFO] [1514823388.882442, 2739.695000]: length of Joint Trajectory List: 10
```