ROS Workspace

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/
$ catkin_make
```

Clone repository

```
$ cd ~/catkin_ws/src
$ git clone https://github.com/udacity/RoboND-Kinematics-Project.git
```

Update all required ros dependencies and permissions

```
$ cd ~/catkin_ws
$ rosdep install --from-paths src --ignore-src --rosdistro=kinetic -y
$ cd ~/catkin_ws/src/RoboND-Kinematics-Project/kuka_arm/scripts
$ sudo chmod +x target_spawn.py
$ sudo chmod +x IK_server.py
$ sudo chmod +x safe_spawner.sh
```

Build project

```
cd ~/catkin_ws
catkin_make
```

Add to .bashrc file

```
export GAZEBO_MODEL_PATH=~/catkin_ws/src/RoboND-Kinematics-Project/kuka_arm/models
```

source /home/robond/catkin_ws/devel/setup.bash

inverse_kinematics.launch file

```
demo flag is set to "false" to run new IK_server.py
```

Run project

```
$ cd ~/catkin_ws/src/RoboND-Kinematics-Project/kuka_arm/scripts
$ ./safe_spawner.sh
```

```
$ cd ~/catkin_ws/src/RoboND-Kinematics-Project/kuka_arm/scripts
$ rosrun kuka_arm IK_server.py
```

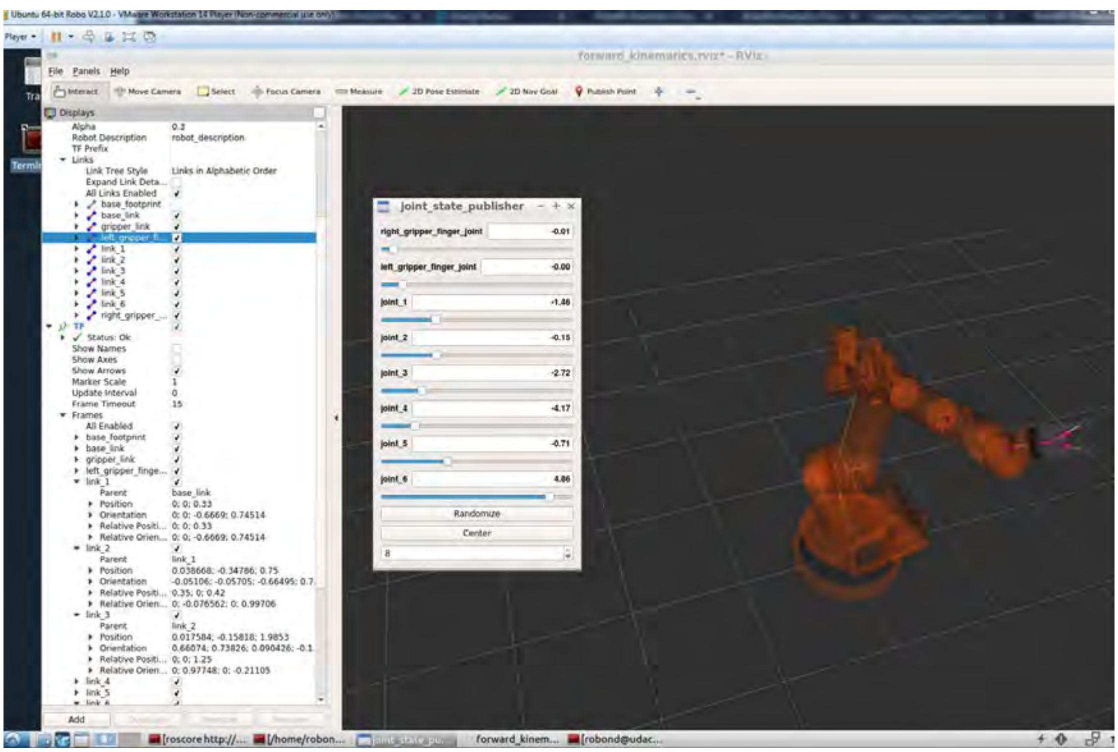
FORWARD KINEMATIC

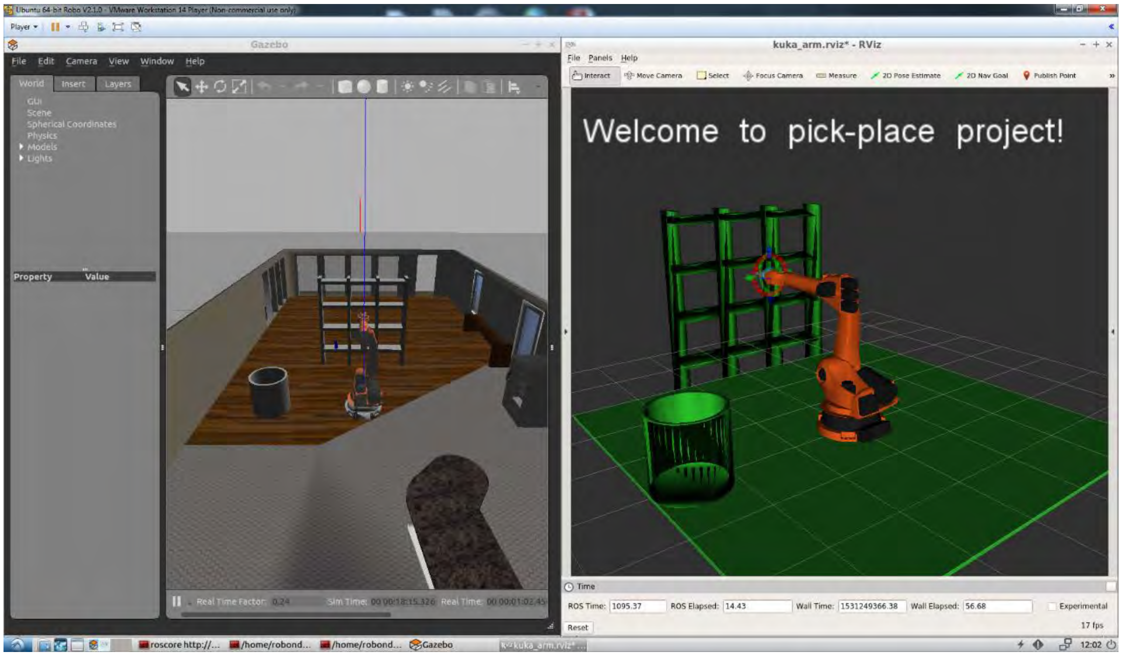
Experiment with the forward kinematics environment and get familiar with the robot.

```
$ roslaunch kuka_arm forward_kinematics.launch
```

ROS makes it very easy to get the transform between any two given frames with the tf_echo command.

```
$ rosrun tf tf_echo base_link link_6
```





KR210.urdf.xacro

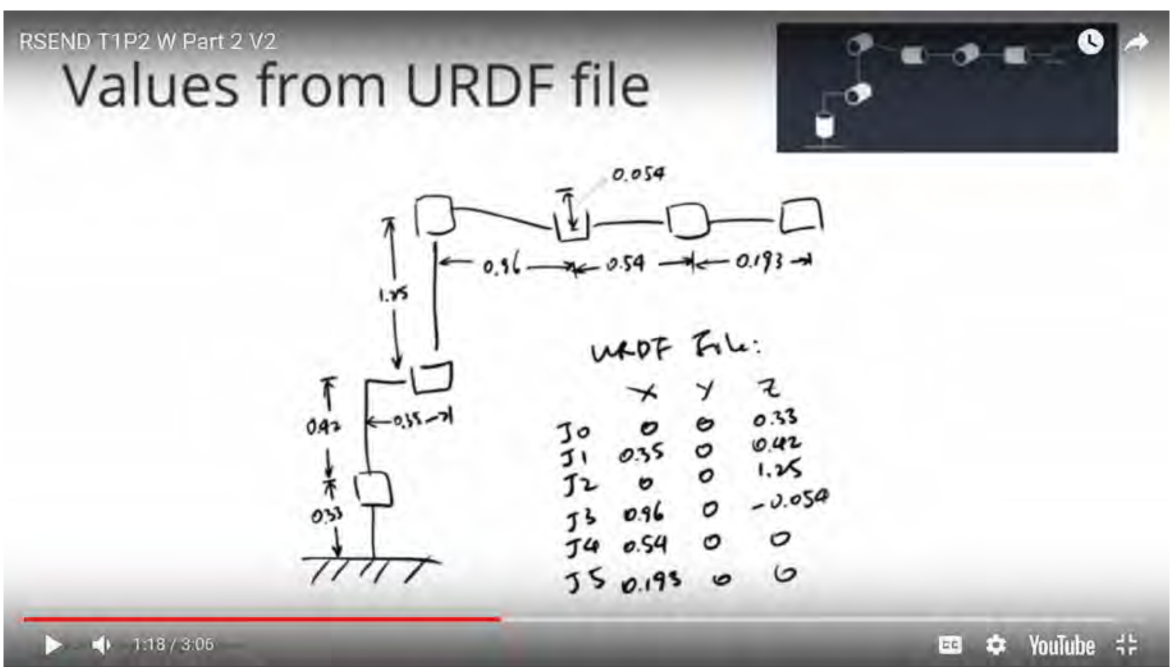
All the values for the robot geometry are contained inside urdf file.

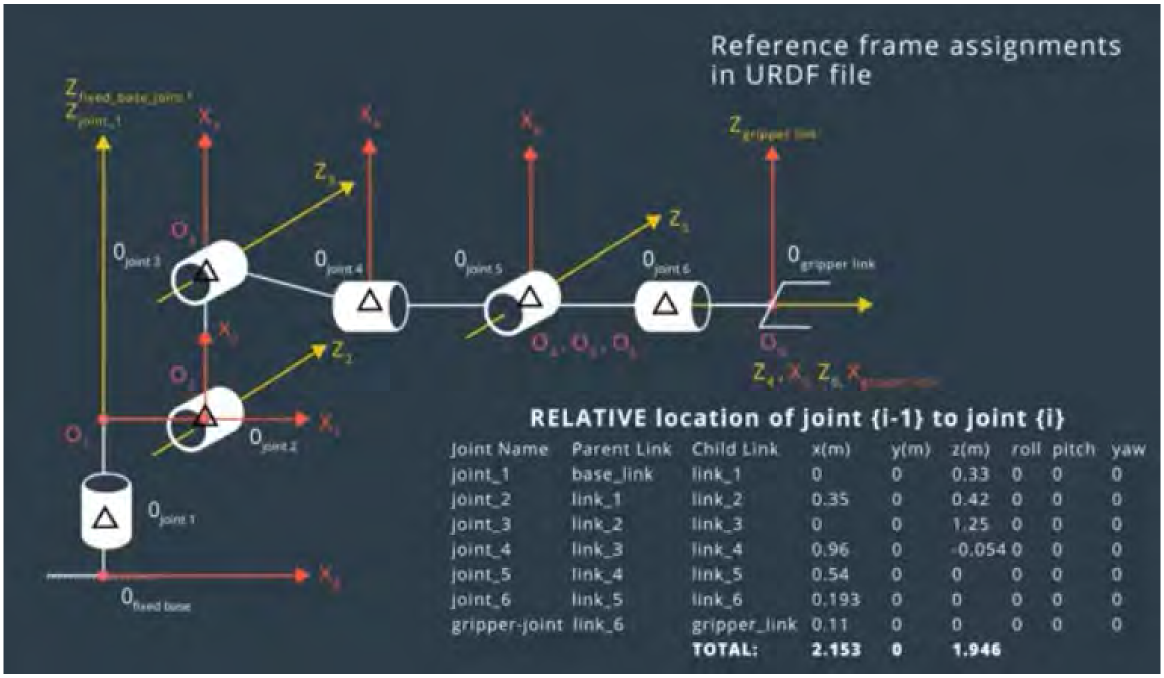
It contains static and dynamic friction coefficient for the links, each link origin w.r.t to its local C.S., mass, inertia and also each JOINT type, origin, parent/child LINK, axis, and physical limits.

```
<!-- joints -->
<joint name="fixed_base_joint" type="fixed">
  <parent link="base_footprint"/>
  <child link="base_link"/>
  <origin xyz="0 0 0" rpy="0 0 0"/>
</joint>
<joint name="joint_1" type="revolute">
  <origin xyz="0 0 0.33" rpy="0 0 0"/>
  <parent link="base_link"/>
  <child link="link_1"/>
  <axis xyz="0 0 1"/>
  <limit lower="${-185*deg}" upper="${185*deg}" effort="300" velocity="${123*deg}"/>
</joint>
<joint name="joint_2" type="revolute">
  <origin xyz="0.35 0 0.42" rpy="0 0 0"/>
  <parent link="link_1"/>
  <child link="link_2"/>
  <axis xyz="0 1 0"/>
  <limit lower="${-45*deg}" upper="${85*deg}" effort="300" velocity="${115*deg}"/>
</joint>
<joint name="joint_3" type="revolute">
  <origin xyz="0 0 1.25" rpy="0 0 0"/>
  <parent link="link_2"/>
  <child link="link_3"/>
  <axis xyz="0 1 0"/>
  <limit lower="${-210*deg}" upper="${(155-90)*deg}" effort="300" velocity="${112*deg}"/>
</joint>
```

```xml
<joint name="joint_4" type="revolute">
  <origin xyz="0.96 0 -0.054" rpy="0 0 0"/>
  <parent link="link_3"/>
  <child link="link_4"/>
  <axis xyz="1 0 0"/>
  <limit lower="${-350*deg}" upper="${350*deg}" effort="300" velocity="${179*deg}"/>
</joint>
<joint name="joint_5" type="revolute">
  <origin xyz="0.54 0 0" rpy="0 0 0"/>
  <parent link="link_4"/>
  <child link="link_5"/>
  <axis xyz="0 1 0"/>
  <limit lower="${-125*deg}" upper="${125*deg}" effort="300" velocity="${172*deg}"/>
</joint>
<joint name="joint_6" type="revolute">
  <origin xyz="0.193 0 0" rpy="0 0 0"/>
  <parent link="link_5"/>
  <child link="link_6"/>
  <axis xyz="1 0 0"/>
  <limit lower="${-350*deg}" upper="${350*deg}" effort="300" velocity="${219*deg}"/>
</joint>
```

```xml
<joint name="right_gripper_finger_joint" type="prismatic">
  <origin rpy="0 0 0" xyz="0.15 -0.0725 0" />
  <parent link="gripper_link" />
  <child link="right_gripper_finger_link" />
  <axis xyz="0 1 0" />
  <limit effort="100" lower="-0.01" upper="0.06" velocity="0.05" />
  <dynamics damping="0.7" />
</joint>
<joint name="left_gripper_finger_joint" type="prismatic">
  <origin rpy="0 0 0" xyz="0.15 0.0725 0" />
  <parent link="gripper_link" />
  <child link="left_gripper_finger_link" />
  <axis xyz="0 -1 0" />
  <limit effort="100" lower="-0.01" upper="0.06" velocity="0.05" />
  <dynamics damping="0.7" />
</joint>
<joint name="gripper_joint" type="fixed">
  <parent link="link_6"/>
  <child link="gripper_link"/>
  <origin xyz="0.11 0 0" rpy="0 0 0"/><!--0.087-->
  <axis xyz="0 1 0" />
</joint>
```

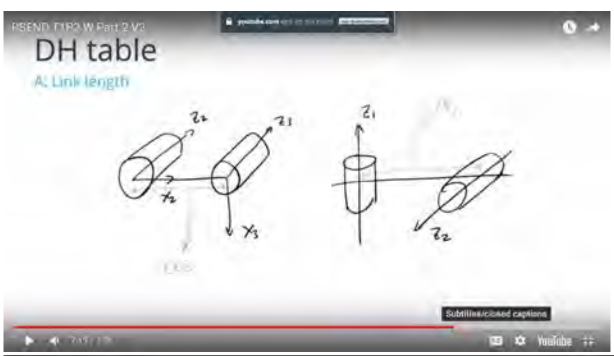


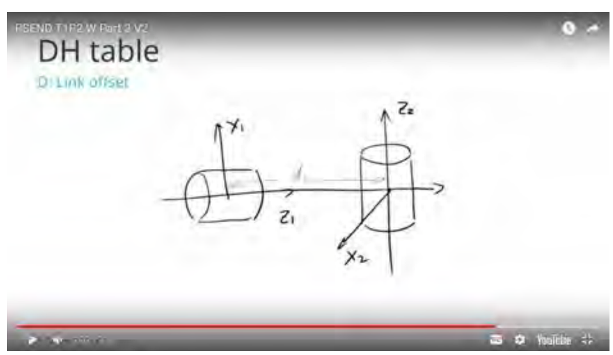


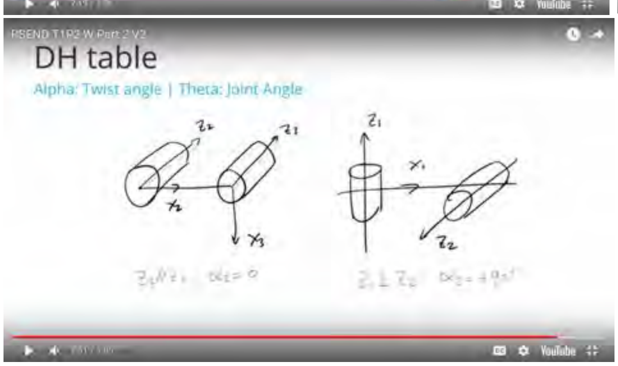
Joint Name	Parent Link	Child Link	x(m)	y(m)	z(m)	roll	pitch	yaw
joint_1	base_link	link_1	0	0	0.33	0	0	0
joint_2	link_1	link_2	0.35	0	0.42	0	0	0
joint_3	link_2	link_3	0	0	1.25	0	0	0
joint_4	link_3	link_4	0.96	0	-0.054	0	0	0
joint_5	link_4	link_5	0.54	0	0	0	0	0
joint_6	link_5	link_6	0.193	0	0	0	0	0
gripper-joint	link_6	gripper_link	0.11	0	0	0	0	0
		TOTAL:	2.153	0	1.946			

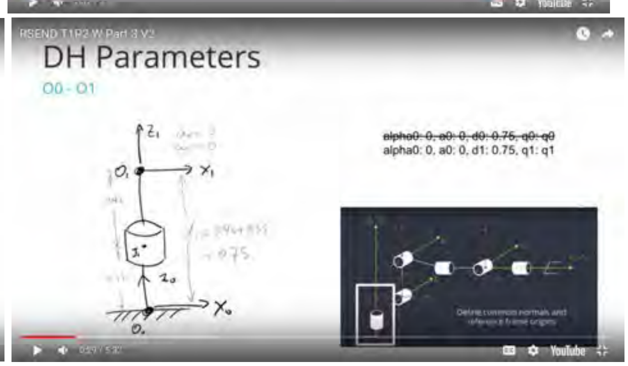
DH Table

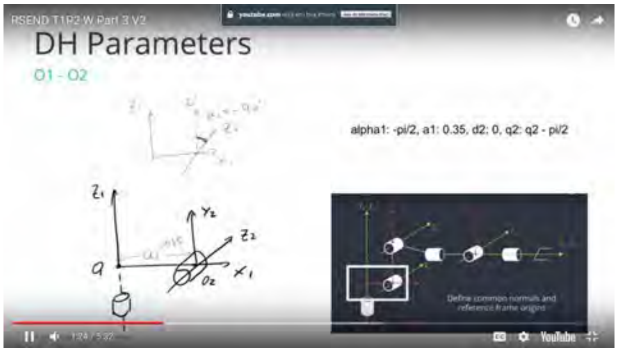
- alpha = twist angle
- a= link length
- d= link offset
- q= joint angle

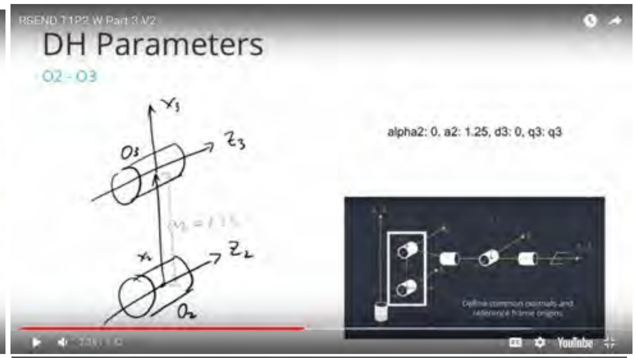


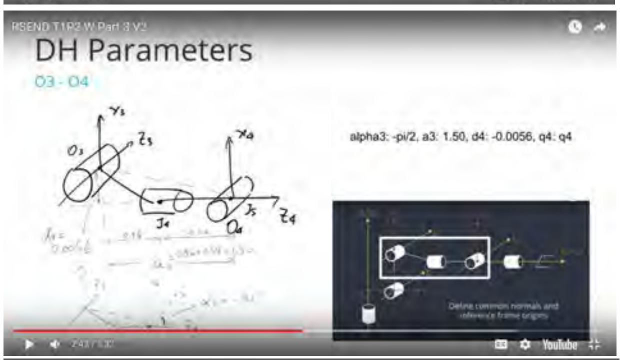


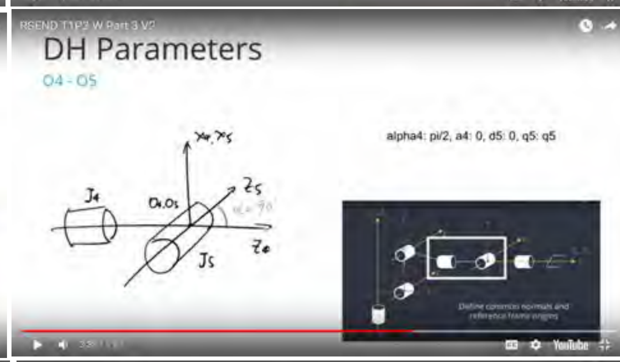


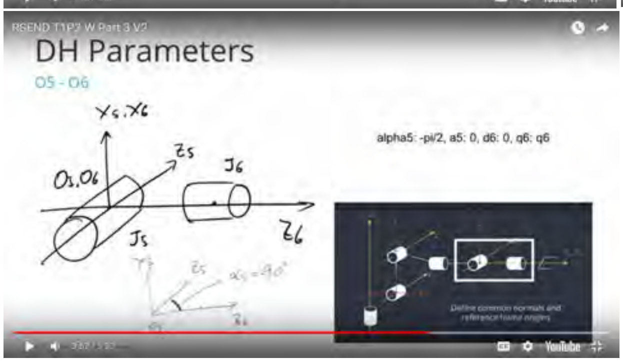


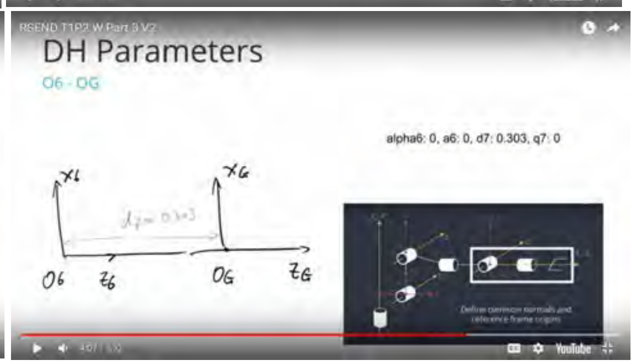


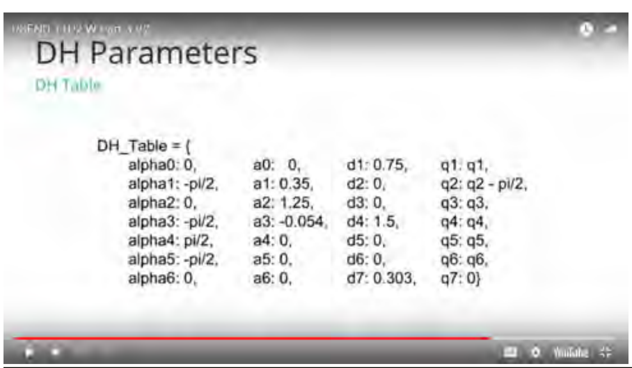













```
# DH Parameters
DH_Table = {alpha0:      0,  a0:      0, d1:  0.75, q1:          q1,
            alpha1:  -pi/2,  a1:   0.35, d2:     0, q2: -pi/2 + q2,
            alpha2:      0,  a2:   1.25, d3:     0, q3:          q3,
            alpha3:  -pi/2,  a3: -0.054, d4:   1.5, q4:          q4,
            alpha4:   pi/2,  a4:      0, d5:     0, q5:          q5,
            alpha5:  -pi/2,  a5:      0, d6:     0, q6:          q6,
            alpha6:      0,  a6:      0, d7: 0.303, q7:           0}
```

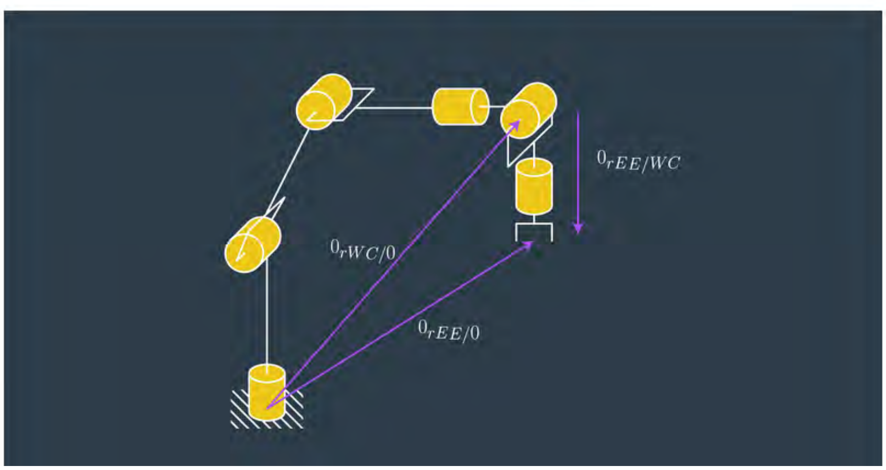
Matrix for translation and orientation

```
# Create individual transformation matrices
T0_1 =  TF_Matrix(alpha0, a0, d1, q1).subs(DH_Table)
T1_2 =  TF_Matrix(alpha1, a1, d2, q2).subs(DH_Table)
T2_3 =  TF_Matrix(alpha2, a2, d3, q3).subs(DH_Table)
T3_4 =  TF_Matrix(alpha3, a3, d4, q4).subs(DH_Table)
T4_5 =  TF_Matrix(alpha4, a4, d5, q5).subs(DH_Table)
T5_6 =  TF_Matrix(alpha5, a5, d6, q6).subs(DH_Table)
T6_EE = TF_Matrix(alpha6, a6, d7, q7).subs(DH_Table)
```

```
# Transformation to find end-effector position
T0_EE = T0_1 * T1_2 * T2_3 * T3_4 * T4_5 * T5_6 * T6_EE
```

Inverse position and orientation kinematics

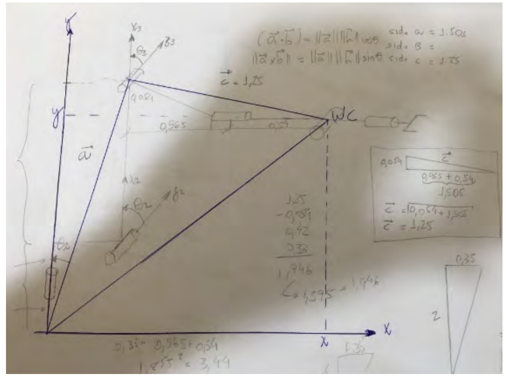
We will now formalize the solution procedure for serial manipulators with a spherical wrist. Consider the six degree of freedom manipulator shown here with joints 4, 5, and 6 comprising the spherical wrist. The location of the wrist center (WC) and end effector (EE) relative to the base frame "0" is given by, ${}^0\mathbf{r}_{WC/0}$ and ${}^0\mathbf{r}_{EE/0}$, respectively. The location of the EE relative to the WC is given by, ${}^0\mathbf{r}_{EE/WC}$. Note that all three vectors are expressed in terms of the base frame as is indicated by the leading superscript, "0".



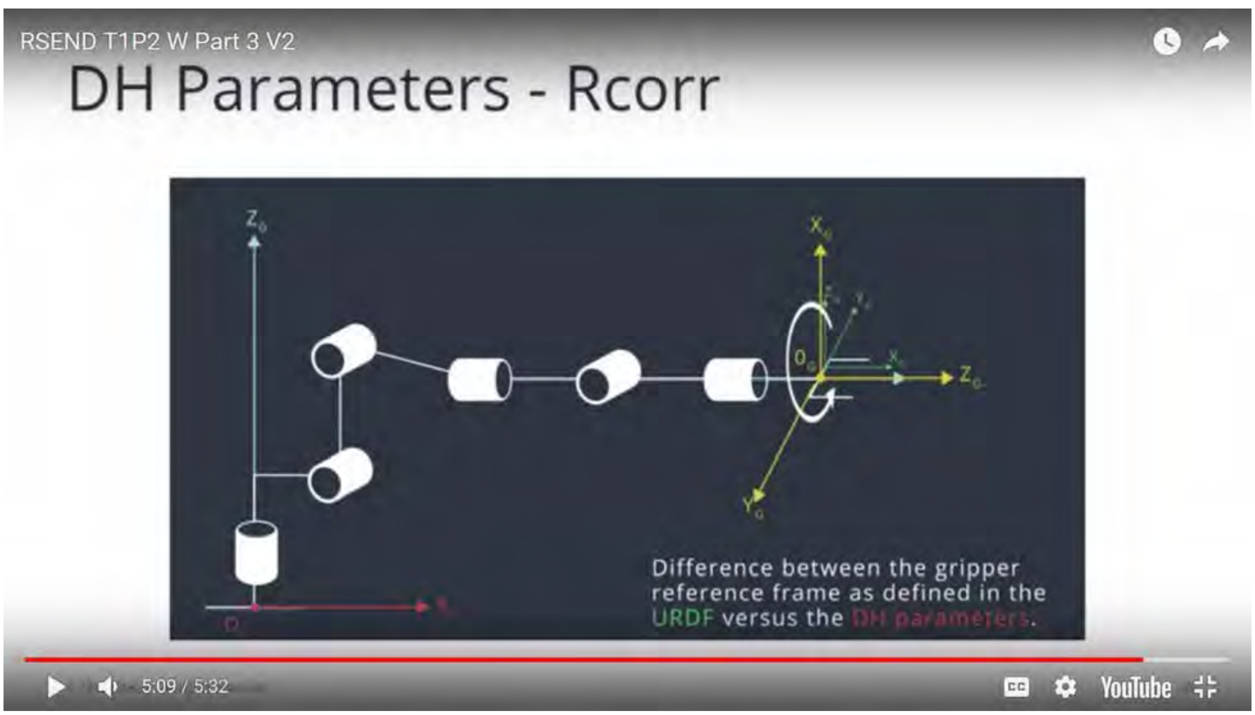
$$
{}^{0}_{EE}T = \left[\begin{array}{c:c} {}^{0}_{6}R & {}^{0}\mathbf{r}_{EE/0} \\ \hdashline 0 \quad 0 \quad 0 & 1 \end{array}\right] = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
{}^{0}\mathbf{r}_{WC/0} = {}^{0}\mathbf{r}_{EE/0} - d \cdot {}^{0}_{6}R \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} - d \cdot {}^{0}_{6}R \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}
$$

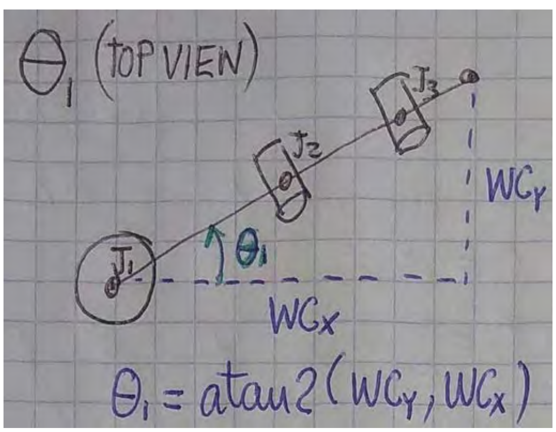
$$
{}^{3}_{6}R = \left({}^{0}_{3}R\right)^{-1} {}^{0}_{6}R = \left({}^{0}_{3}R\right)^{T} {}^{0}_{6}R
$$



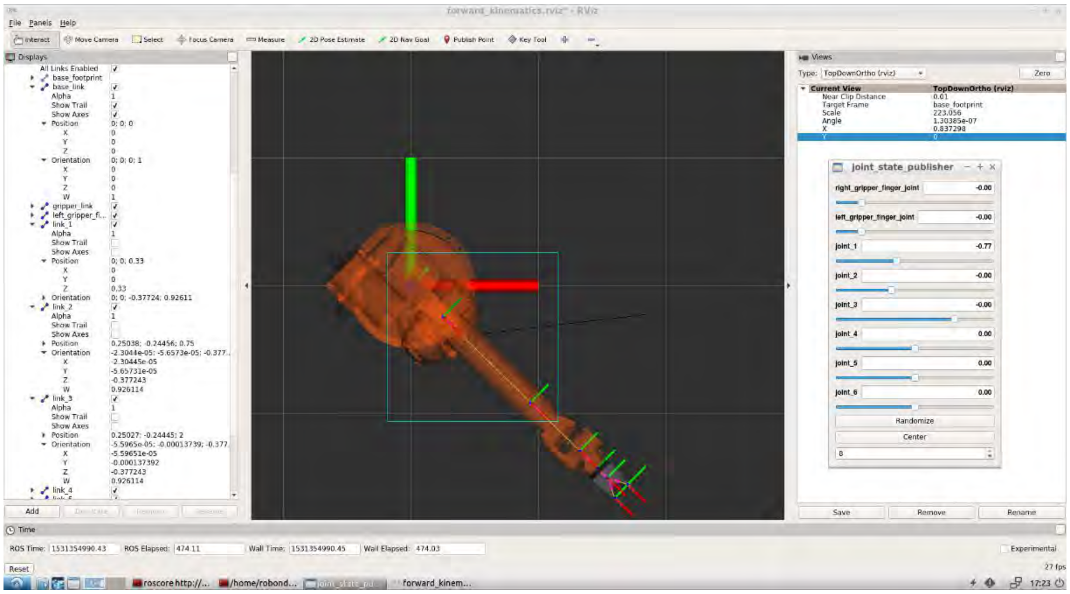
```
# Define additional symbols for roll, pitch and yaw for the end-effector orientatio
r, p, y = symbols('r p y')
# Rotation Matrices for x,y and z consisting of end effector orientation parameters
R_x = Matrix([[1, 0, 0], [0, cos(r), -sin(r)], [0, sin(r), cos(r)]]) #ROLL
R_y = Matrix([[cos(p), 0, sin(p)], [0, 1, 0], [-sin(p), 0, cos(p)]]) #PITCH
R_z = Matrix([[cos(y), -sin(y), 0], [sin(y), cos(y), 0], [0, 0, 1]]) #YAW
```



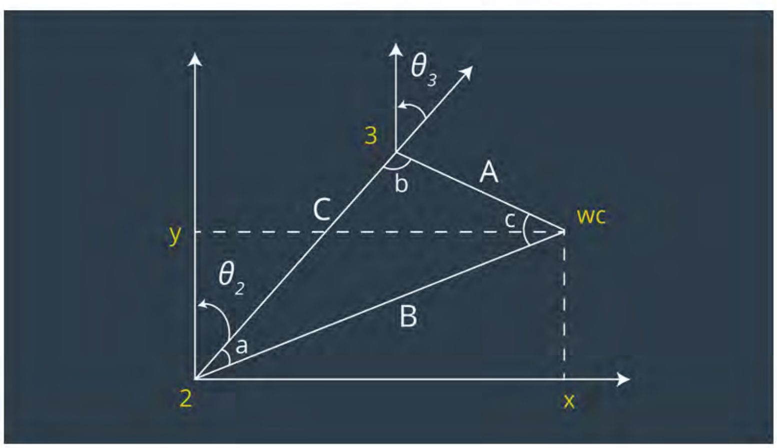
```
#####IK code start HERE ##################################################
    # Calculate joint angles using Geometric IK method
    # Calculating positions of the wrist center


    ROT_EE = R_z*R_y*R_x

    # Compensate for rotation discrepancy between DH parameters and Gazebo
    Rot_correction = R_z.subs(y,pi)*R_y.subs(p,-pi/2)

    ROT_EE = ROT_EE * Rot_correction
    ROT_EE = ROT_EE.subs({'r': roll, 'p': pitch, 'y': yaw})

    end_effector_pos = Matrix([px, py, pz])
    wrist_center = end_effector_pos - (0.303)*ROT_EE[:,2]

    Wx, Wy, Wz = wrist_center[0], wrist_center[1], wrist_center[2]

    # SS triangle for theta2 and theta3
    side_a = 1.50
    side_b = sqrt(pow(sqrt(Wx**2 + Wy**2) - 0.35,2) + pow((Wz - 0.75),2))
    side_c = 1.25

    angle_a = acos((side_b**2 + side_c**2 - side_a**2)/(2*side_b*side_c))
    angle_b = acos((side_a**2 + side_c**2 - side_b**2)/(2*side_a*side_c))
    angle_c = acos((side_a**2 + side_b**2 - side_c**2)/(2*side_a*side_b))
```



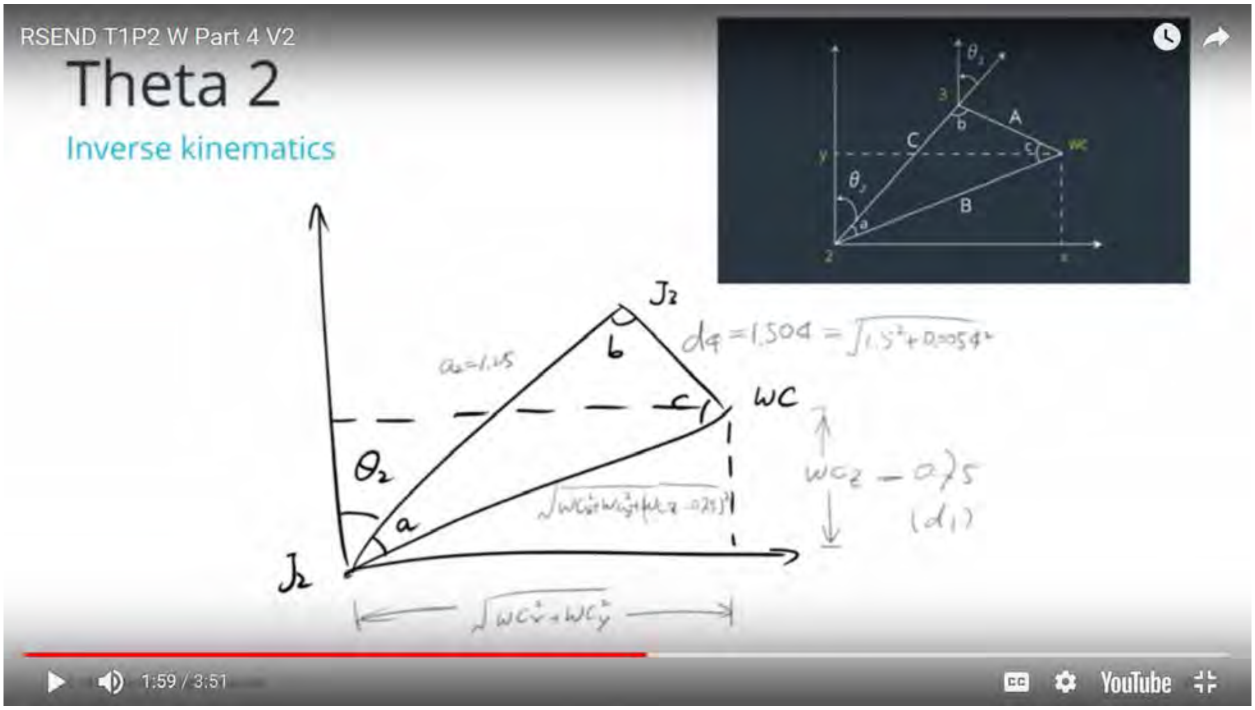
theta1 = atan2(Wy, Wx)



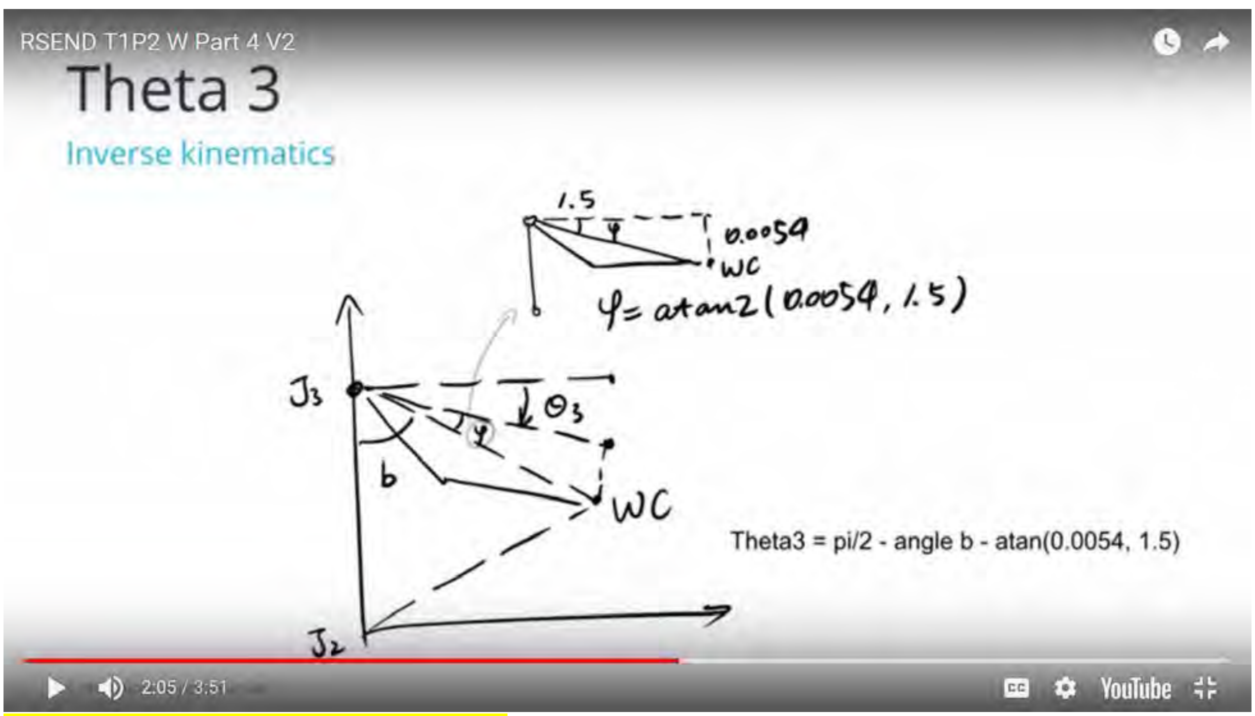
Calculating `theta 1` will be relatively straightforward once you have the wrist center position from above. `Theta 2` and `theta 3` can be relatively tricky to visualize. The following diagram depicts the angles for you.



The labels **2**, **3** and **WC** are Joint 2, Joint 3, and the Wrist Center, respectively. You can obtain, or rather visualize, the triangle between the three if you project the joints onto the z-y plane corresponding to the world reference frame. From your DH parameters, you can calculate the distance between each joint above. Using trigonometry, specifically the Cosine Laws, you can calculate `theta 2` and `theta 3`.



theta2 = pi/2 - angle_a - atan2((Wz - 0.75), sqrt(Wx**2 + Wy**2) - 0.35)



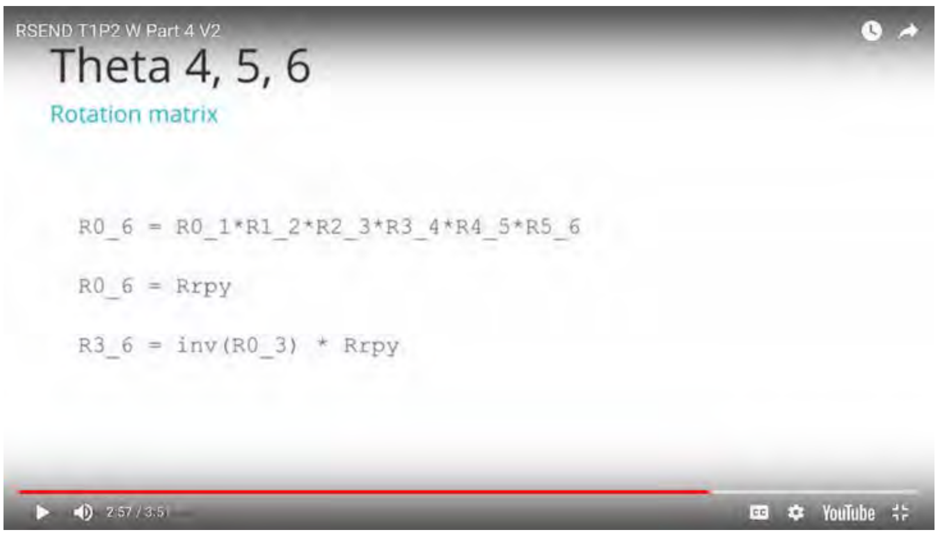
theta3 = pi/2 - angle_b – atan(0.0054, 1.5)

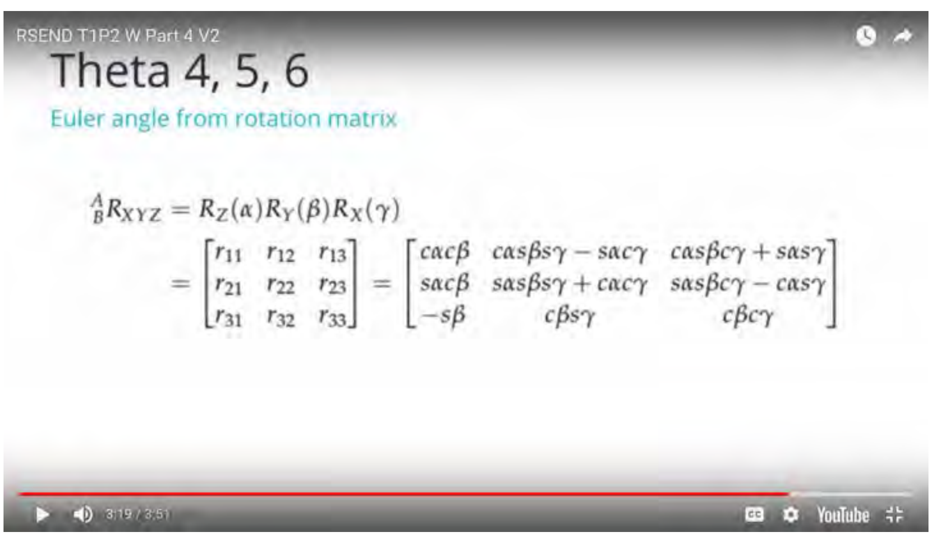
```
R3_6 = inv(R0_3) * Rrpy
```

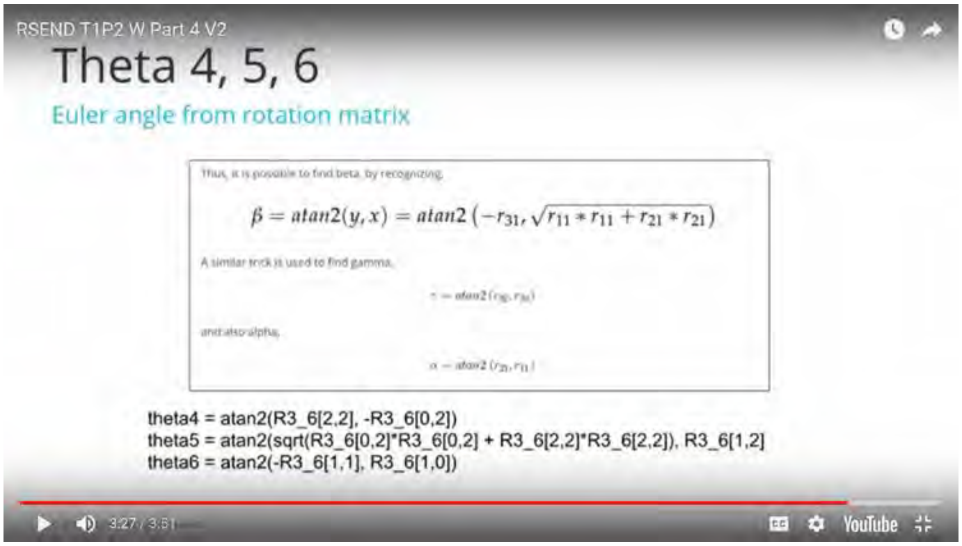
The resultant matrix on the RHS (**R**ight **H**and **S**ide of the equation) does not have any variables after substituting the joint angle values, and hence comparing LHS (**L**eft **H**and **S**ide of the equation) with RHS will result in equations for joint 4, 5, and 6.

Note: While calculating the **inverse** above, using Sympy's `inv()` method, please make sure to pass `"LU"` as an argument to ensure that it's calculated using the LU decomposition. You can refer to the documentation on how to use it here.

Finally, we have equations describing all six joint variables, next we will turn our kinematic analysis into a ROS python node which will perform the Inverse Kinematics for the pick and place operation.








```python
# Finding the last three joint angles 4, 5, 6
R0_3 = T0_1[0:3,0:3]*T1_2[0:3,0:3]*T2_3[0:3,0:3]
R0_3 = R0_3.evalf(subs={q1:theta1, q2:theta2, q3:theta3})

R3_6 = R0_3.inv('LU') * ROT_EE

# Euler angles from rotation matrix
theta4 = atan2(R3_6[2,2], -R3_6[0,2])
theta5 = atan2(sqrt(R3_6[0,2]**2 + R3_6[2,2]**2), R3_6[1,2])
theta6 = atan2(-R3_6[1,1], R3_6[1,0])
```