Project Tools:

1- **Gazebo**: a physics based 3D simulator extensively used in the robotics world.
2- **RViz**: a 3D visualizer for sensor data analysis, and robot state visualization.
3- **Moveit!**: a ROS based software framework for motion planning, kinematics and robot control.

Environment Setup:

1- Create active ROS workspace:
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/
$ catkin_make

2- Clone the project repository into the **src** directory of the workspace:
$ cd ~/catkin_ws/src
$ git clone https://github.com/udacity/RoboND-Kinematics-Project.git

3- Install missing dependencies:
$ cd ~/catkin_ws
$ rosdep install --from-paths src --ignore-src --rosdistro=kinetic -y

4- Change the permissions of script files to turn them executable:
$ cd ~/catkin_ws/src/RoboND-Kinematics-Project/kuka_arm/scripts
$ sudo chmod u+x target_spawn.py
$ sudo chmod u+x IK_server.py
$ sudo chmod u+x safe_spawner.sh

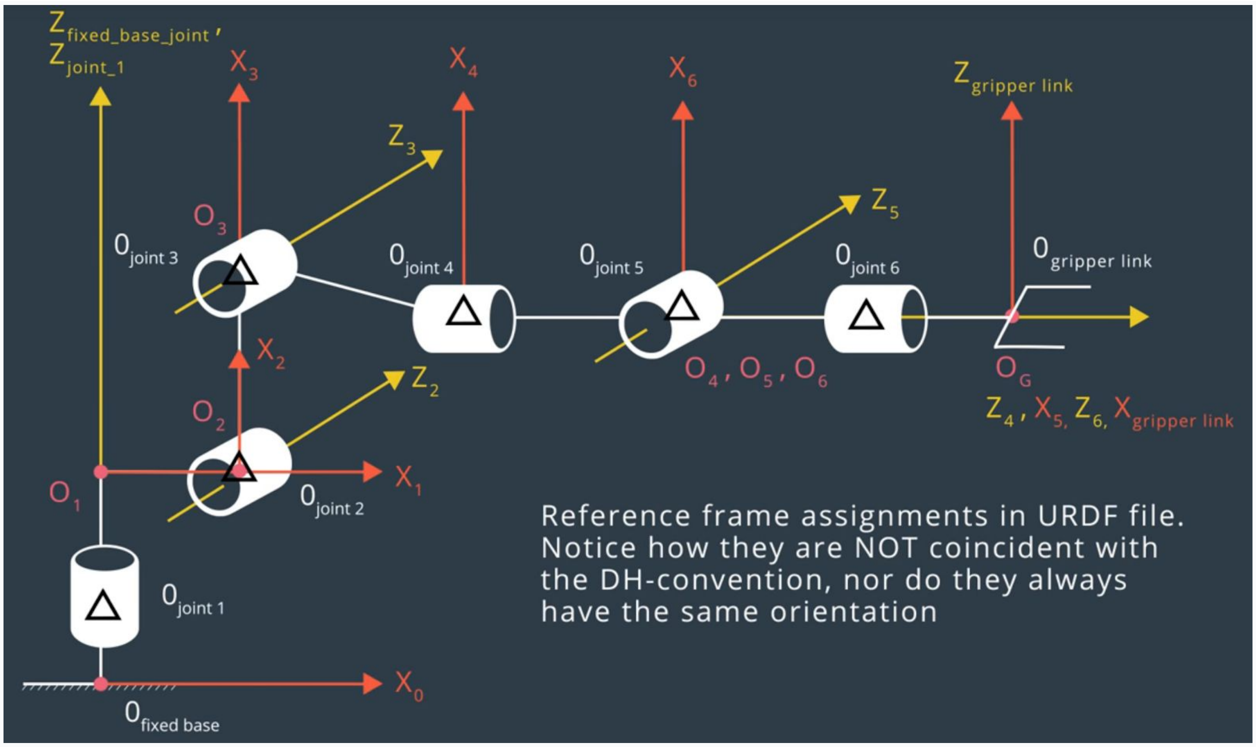
5- Build the project:
$ cd ~/catkin_ws
$ catkin_make

6- Add the following to .bashrc file at the end:
source ~/catkin_ws/devel/setup.bash
export GAZEBO_MODEL_PATH=~/catkin_ws/src/RoboND-Kinematics-Project/kuka_arm/models

7- Save the .bashrc file.

Kinematic Analysis:

1-Denavit-Hartenberg Diagram:



The arm consists of six revolute joints connected in linear fashion.

The following steps are performed:

1. Label all joints from 1 to n = 6.
2. Label all links from 0 to n = 6.
3. Define the joint axes.
4. Define z-axes as the joint axes.
5. Define the x-axes as the common normals between zi-1 and zi.
6. Define the origin of frame {i} as the intersection of xi with zi.

2- Denavit-Hartenberg Table:

i	theta(i)	d(i)	a(i-1)	alpha(i-1)
1	q1	0.75	0	0
2	-pi/2 + q2	0	0.35	-pi/2
3	q3	0	1.25	0
4	q4	1.5	-0.054	-pi/2
5	q5	0	0	pi/2
6	q6	0	0	-pi/2
7	0	0.303	0	0

Transformation Matrices:

All of the joints have their own transformation matrix that describes their position and orientation relative to prior joints.

1- Define DH transformation matrix:

```
def TF_Matrix(alpha, a, d, q):
    TF = Matrix([
                    [cos(q)              ,          -sin(q),          0,              a],
                    [sin(q)*cos(alpha), cos(q)*cos(alpha), -sin(alpha), -sin(alpha)*d],
                    [sin(q)* sin(alpha),  cos(q)*sin(alpha), cos(alpha), cos(alpha)*d],
                    [                  0,                   0,          0,              1]
                ])
        return TF
```

2- Define 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)
```

3- Define transformation matrix from the base link to the end effector:

```
T0_EE = simplify(T0_1 * T1_2 * T2_3 * T3_4 * T4_5 * T5_6 * T6_EE)
```

The transformation matrices for each joint:

For example, T0_1 will be:

```
[
 [cos(q1), -sin(q1), 0,     0],
 [ sin(q1), cos(q1), 0,     0],
 [       0,       0, 1, 0.75],
 [       0,       0, 0,     1]
]
```

Inverse Kinematics:

1- Find rotation matrix for the end effector:

```
# Roll
ROT_x = Matrix([[ 1,      0,       0],
                [ 0, cos(r), -sin(r)],
                [ 0,  sin(r), cos(r)]])
# Pitch
ROT_y = Matrix([[cos(p), 0, sin(p)],
                [      0,1,      0],
                [-sin(p),0,cos(p)]])
# Yaw
ROT_z = Matrix([[cos(y), -sin(y),0],
                [sin(y),  cos(y),0],
                [     0,       0, 1]])

ROT_EE = simplify(ROT_z * ROT_y * ROT_x)
Rot_Error = ROT_z.subs(y, radians(180)) * ROT_y.subs(p, radians(-90))
ROT_EE = simplify(ROT_EE * Rot_Error)
```

2- Get the end-effector position (px,py,pz) and orientation (roll, pitch, yaw) from request:

```
px = req.poses[x].position.x
py = req.poses[x].position.y
pz = req.poses[x].position.z

(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])

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

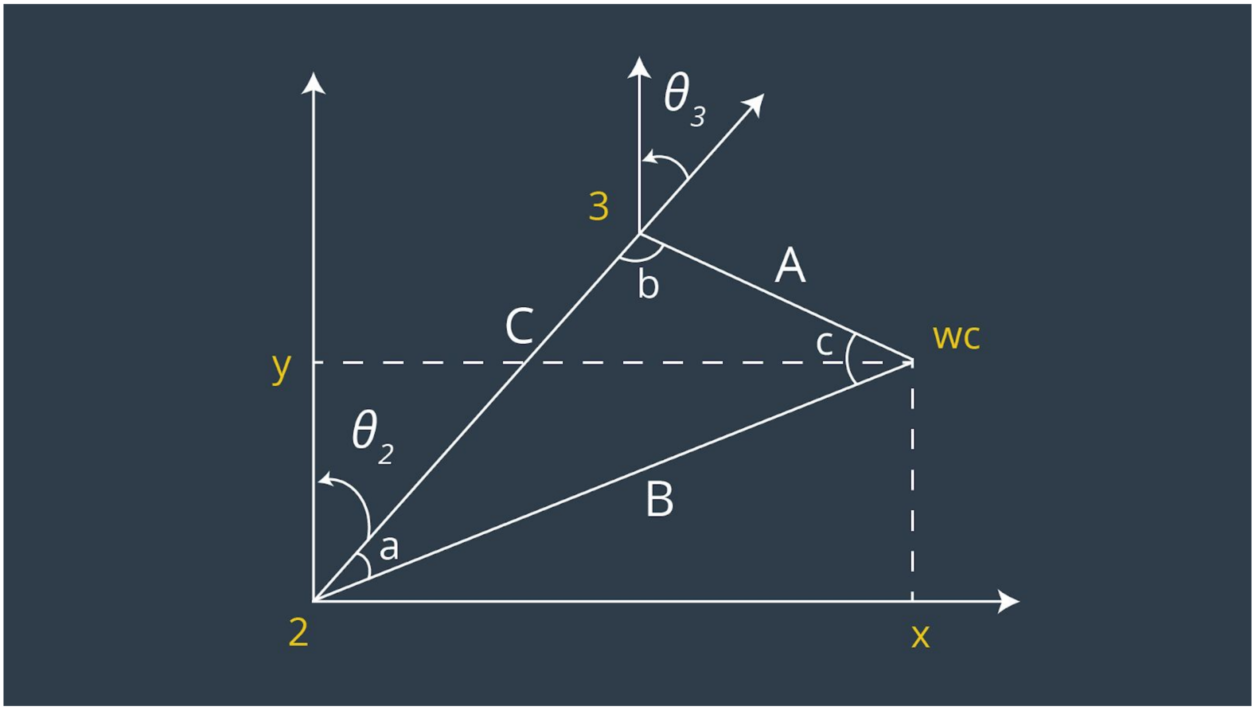
3- Calculate Wrest Center:

```
EE = Matrix([[px], [py], [pz]])
WC = EE - (0.303) * ROT_EE[:,2]
```

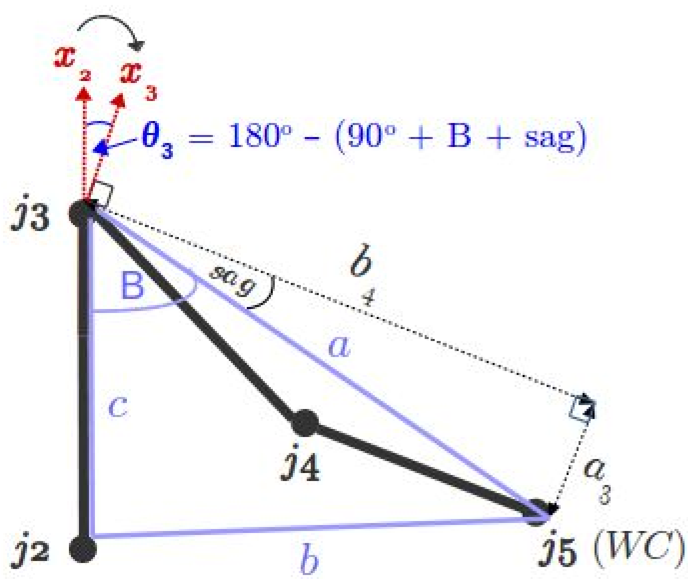
4- Calculate joint angles using Geometric IK method:

```
theta1 = atan2(WC[1], WC[0])

side_a = 1.501
side_c = 1.25
side_b = sqrt(pow(sqrt(WC[0] * WC[0] + WC[1] * WC[1]) - 0.35, 2)+ pow((WC[2] - 0.75), 2))

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

For calculating the joints 2 and 3, applying the cosine rule to obtain the angles, first for angle 3, then calculating angle 2:



theta2 = pi/2 - angle_a - atan2(WC[2] - 0.75, sqrt(WC[0] + WC[1] * WC[1]) - 0.35)



# 0.036 accounts for sag in link4 of -0.054m
theta3 = pi/2 - (angle_b + 0.036)

5- Calculate rotation matrix from base to third link:

```
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})

R0_3 = Matrix([
                    [sin(q2 + q3)*cos(q1), cos(q1)*cos(q2 + q3), -sin(q1)],
                    [ sin(q1)*sin(q2 + q3),  sin(q1)*cos(q2 + q3), cos(q1)],
                    [          cos(q2 + q3),            -sin(q2 + q3),          0]
                ])
```

6- Calculate rotation matrix from three to six:

```
R3_6 = R0_3.inv("LU") * ROT_EE

R3_6 = Matrix([
[-sin(q4)*sin(q6) + cos(q4)*cos(q5)*cos(q6), -sin(q4)*cos(q6) - sin(q6)*cos(q4)*cos(q5), -sin(q5)*cos(q4)],
[                                sin(q5)*cos(q6),                                -sin(q5)*sin(q6),             cos(q5)],
[ -sin(q4)*cos(q5)*cos(q6) - sin(q6)*cos(q4),   sin(q4)*sin(q6)*cos(q5) - cos(q4)*cos(q6),   sin(q4)*sin(q5)]
])
```

7- Calculate Euler angles from rotation matrix:

Using the DH transforms to obtain the resultant transform and hence resultant rotation, then substitute the values calculated for joints 1 to 3 in their respective individual rotation matrices and pre-multiply both sides of the above equation by inv(R0_3)

The roll, pitch, yaw for the end effector relative to the base link:

Theta 4, 5, 6

Euler angle from rotation matrix

$$
{}^{A}_{B}R_{XYZ} = R_Z(\alpha)R_Y(\beta)R_X(\gamma)
= \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}
= \begin{bmatrix} c\alpha c\beta & c\alpha s\beta s\gamma - s\alpha c\gamma & c\alpha s\beta c\gamma + s\alpha s\gamma \\ s\alpha c\beta & s\alpha s\beta s\gamma + c\alpha c\gamma & s\alpha s\beta c\gamma - c\alpha s\gamma \\ -s\beta & c\beta s\gamma & c\beta c\gamma \end{bmatrix}
$$

There are two possible solutions to solve angle β “theta5”:

1- Although beta appears in isolation in element r31, it is not a good idea to solve for angles using the inverse of the sine or cosine functions. The reason is the ambiguity in sign: if -sin(beta) = 0.5, in which quadrant is the angle?.

2- This type of ambiguity in the first solution is avoided by using the atan2 function:

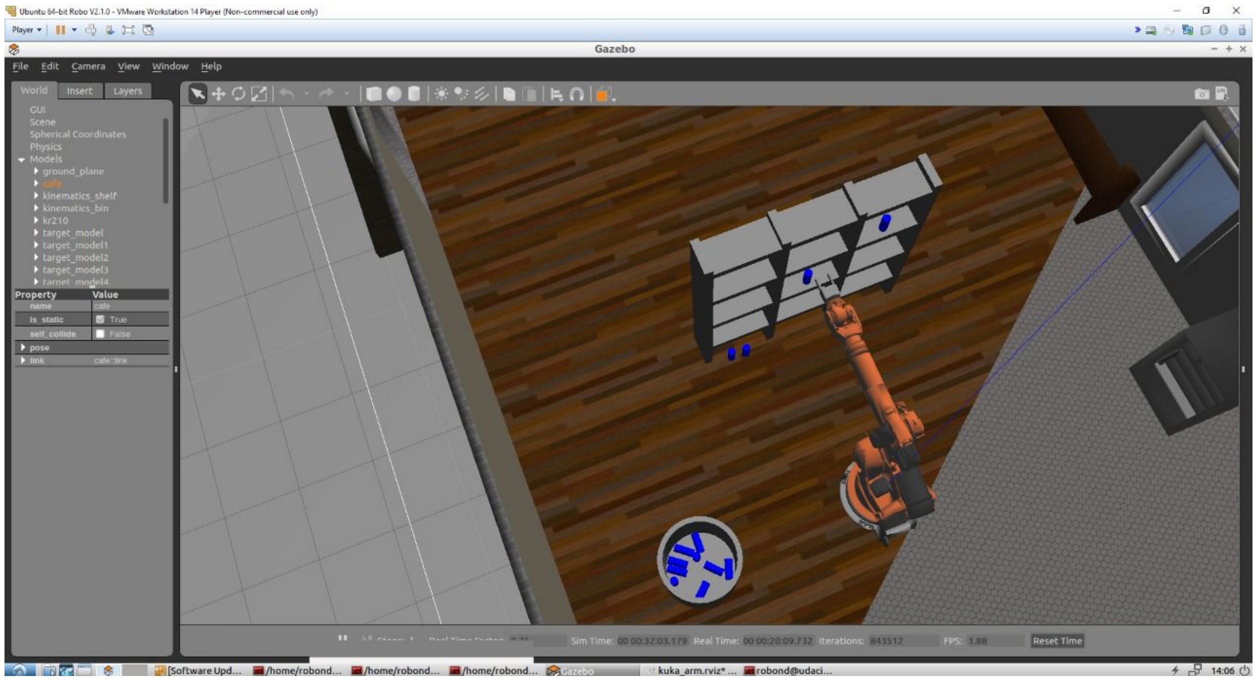
$\beta = atan2(y,x) = atan2(-r31, r11*r11+r21*r21)$

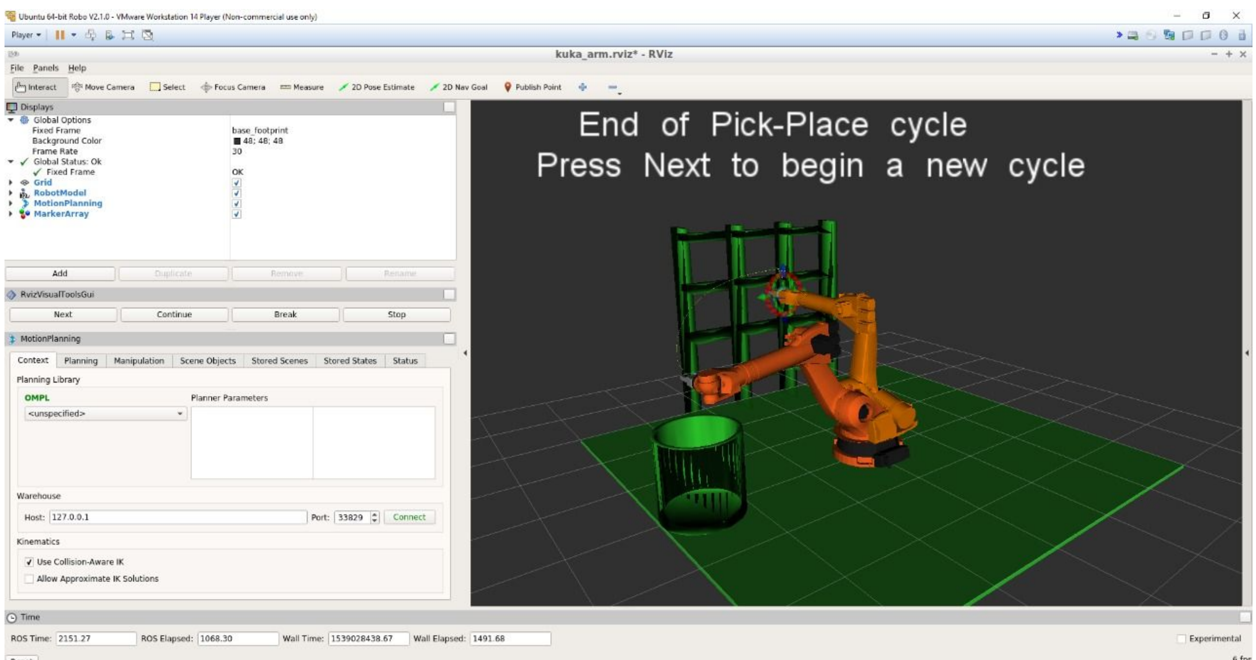
So I prefer the second solution but this will affect theta4 & theta6 when cos(beta) = 0, that is, when beta = +/-90 degrees, At this point atan2 is undefined and, as we saw with Euler Angles, the system exhibits a *singularity of representation*.

```
theta5 = atan2(sqrt(R3_6[0,2]*R3_6[0,2] + R3_6[2,2] * R3_6[2,2]), R3_6[1,2])
# Select best solution based on theta5
        if (theta5 > pi) :
          theta4 = atan2(-R3_6[2,2], R3_6[0,2])
          theta6 = atan2(R3_6[1,1],-R3_6[1,0])
        else:
          theta4 = atan2(R3_6[2,2], -R3_6[0,2])
          theta6 = atan2(-R3_6[1,1],R3_6[1,0])
```

Screenshot of the completed pick and place process:

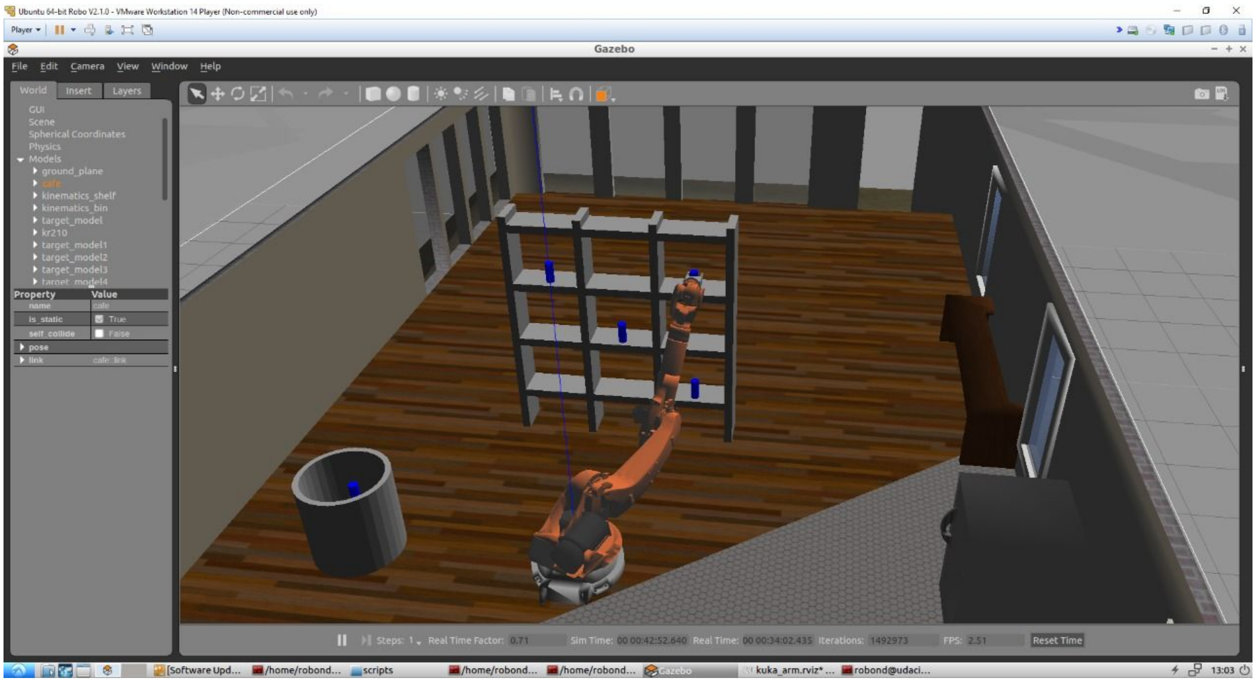
- Arm success 9/10 times performing a complete pick and place operation.



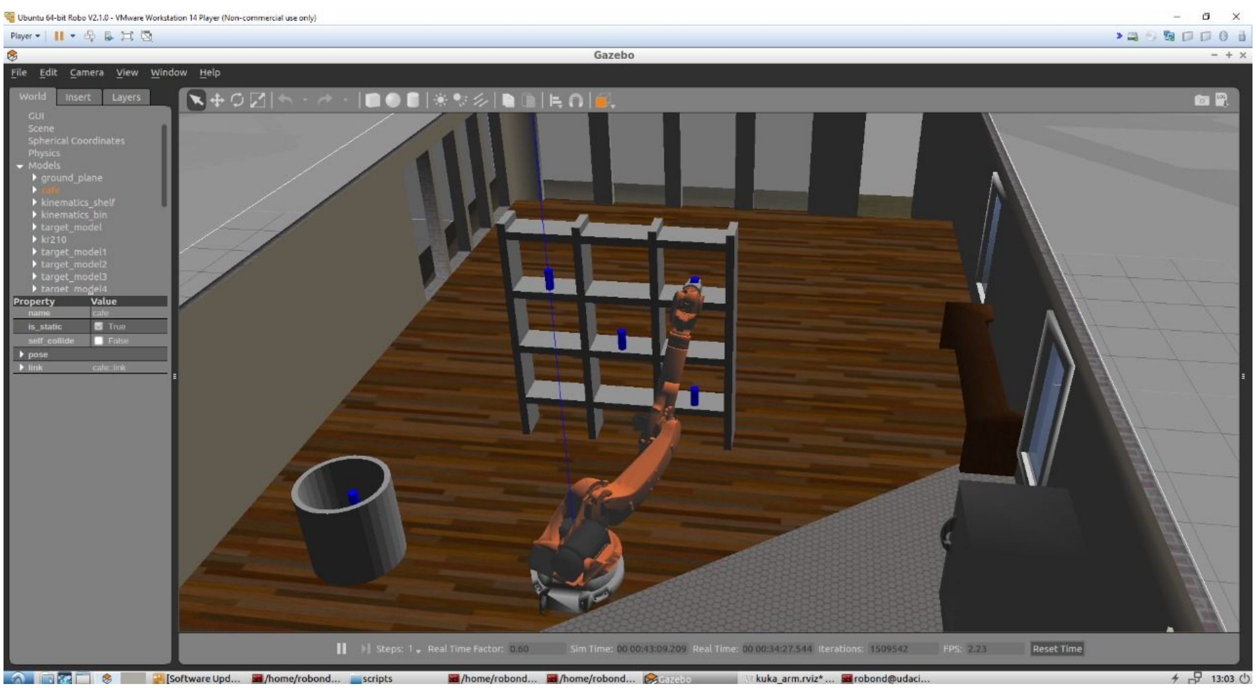


- One pick and place process in details:

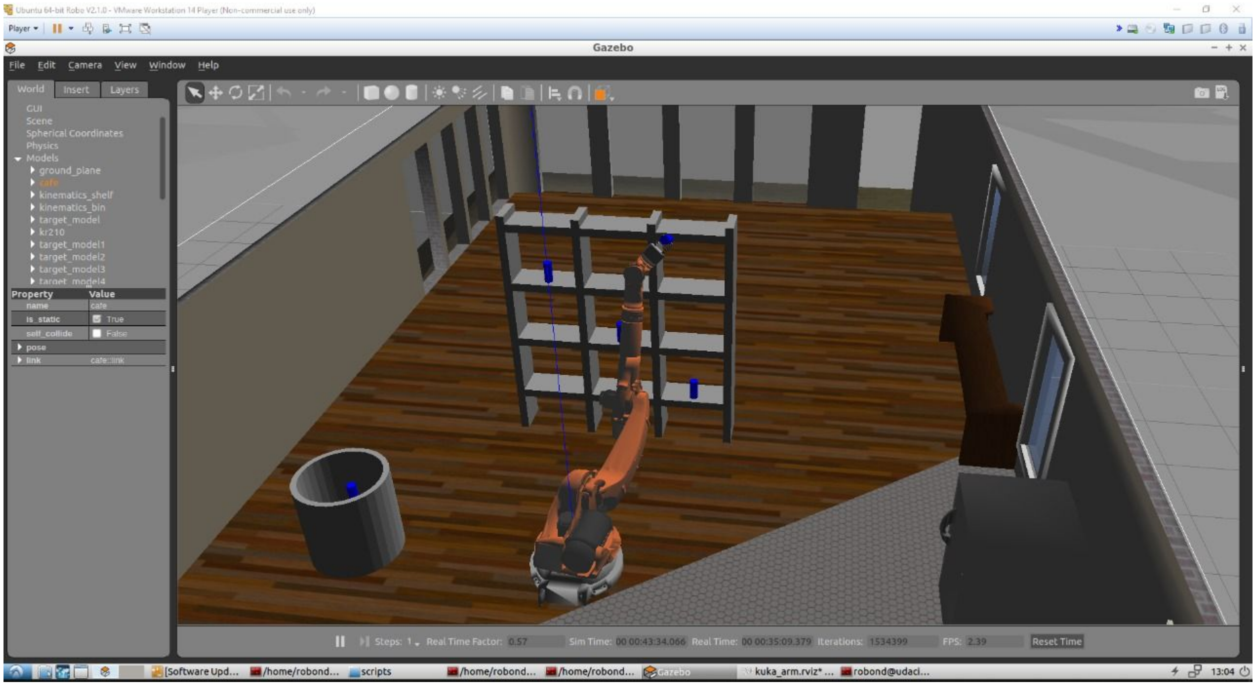
1-


Ubuntu 64-bit Robo V2.1.0 - VMware Workstation 14 Player (Non-commercial use only)
Player
Gazebo
File
Edit
Camera
View
Window
Help
World
Insert
Layers
GUI
Scene
Spherical Coordinates
Physics
Models
ground_plane
kinematics_shelf
kinematics_bin
target_model
kr210
target_model1
target_model2
target_model3
Property
Value
Reset Time
[Software Upd...
/home/robond...
scripts
/home/robond...
/home/robond...
Gazebo
kuka_arm.rviz* ...
robond@udaci...
13:03

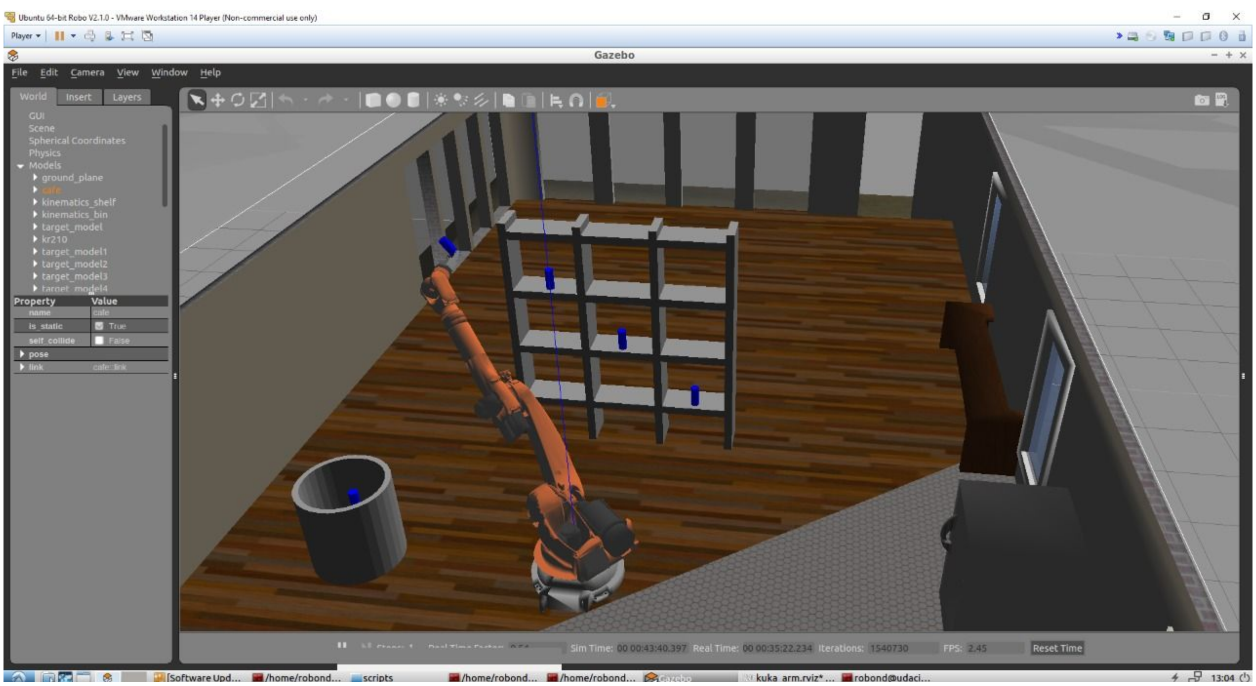
2-


Ubuntu 64-bit Robo V2.1.0 - VMware Workstation 14 Player (Non-commercial use only)
Player
Gazebo
File
Edit
Camera
View
Window
Help
World
Insert
Layers
GUI
Scene
Spherical Coordinates
Physics
Models
ground_plane
kinematics_shelf
kinematics_bin
target_model
kr210
target_model1
target_model2
target_model3
Property
Value
Reset Time
[Software Upd...
/home/robond...
scripts
/home/robond...
/home/robond...
Gazebo
kuka_arm.rviz* ...
robond@udaci...
13:03

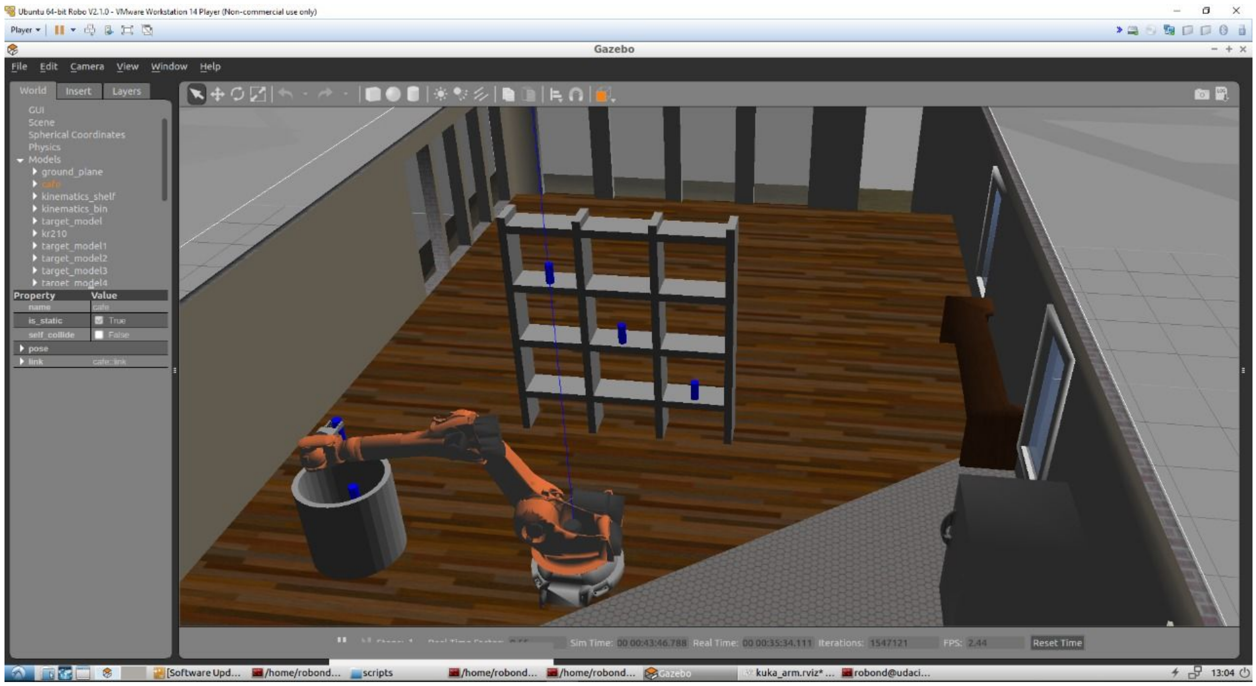
3-


Gazebo
File
Edit
Camera
View
Window
Help
World
Insert
Layers
GUI
Scene
Spherical Coordinates
Physics
Models
ground_plane
kinematics_shelf
kinematics_bin
target_model
kr210
target_model1
target_model2
target_model3
Property
Value
Reset Time

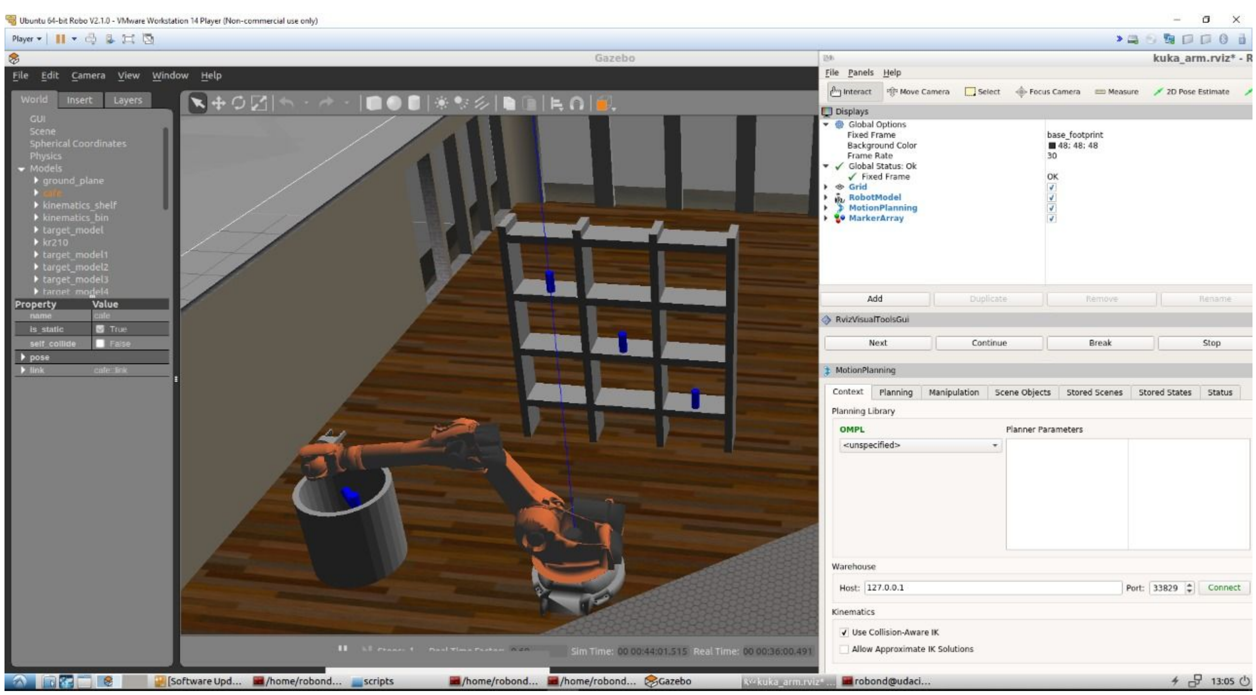
4-


Gazebo
File
Edit
Camera
View
Window
Help
World
Insert
Layers
GUI
Scene
Spherical Coordinates
Physics
Models
ground_plane
kinematics_shelf
kinematics_bin
target_model
kr210
target_model1
target_model2
target_model3
Property
Value
Reset Time

5-


Ubuntu 64-bit Robo V2.1.0 - VMware Workstation 14 Player (Non-commercial use only)
Gazebo
File Edit Camera View Window Help
World Insert Layers
GUI
Scene
Spherical Coordinates
Physics
Models
ground_plane
kinematics_shelf
kinematics_bin
target_model
kr210
target_model1
target_model2
target_model3
Property Value
is_static True
self_collide False
pose
link
Sim Time:
Real Time:
Iterations:
FPS:
Reset Time
13:04

6-


Ubuntu 64-bit Robo V2.1.0 - VMware Workstation 14 Player (Non-commercial use only)
Gazebo
File Edit Camera View Window Help
World Insert Layers
kuka_arm.rviz*
File Panels Help
Interact Move Camera Select Focus Camera Measure 2D Pose Estimate
Displays
Global Options
Fixed Frame base_footprint
Background Color 48; 48; 48
Frame Rate 30
Global Status: Ok
Fixed Frame OK
Grid
RobotModel
MotionPlanning
MarkerArray
Add Duplicate Remove Rename
RvizVisualToolsGui
Next Continue Break Stop
MotionPlanning
Context Planning Manipulation Scene Objects Stored Scenes Stored States Status
Planning Library
OMPL
Planner Parameters
<unspecified>
Warehouse
Host: 127.0.0.1
Port: 33829
Connect
Kinematics
Use Collision-Aware IK
Allow Approximate IK Solutions
13:05

Difficulties:

- The pick and place operation was somehow slow; the response of the robot arm definitely need modifications to increase it's response, besides that the arm dropped some cans on the ground.
- Gazebo crashes frequently.
- RViz Simulator is very very slow and require high GPU drivers.
- VMPlayer is a low performance virtual machine.

Improvements:

- Recommend better simulation programs.
- Add more lessons to the class.