CS 6301 Introduction to Robot Manipulation and Navigation Homework 3

Professor Yu Xiang

October 26, 2022

In this homework, write down your solutions for problems 1, 2, 3 and finish the coding problem 4. Upload your solutions and code to eLearning. Our TA will check your solutions and run your scripts to verify them.

Problem 1

(2 points)

Forward Kinematics. Exercise 4.12 in Lynch and Park, Modern Robotics.

The RRPRRR spatial open chain of Figure 1 is shown in its zero position (all joints lie on the same plane). Determine the end-effector zero position configuration M, the screw axes $\mathcal{S}_i$ in {0}, and the screw axes $\mathcal{B}_i$ in {b}. Setting $\theta_5 = \pi$ and all other joint variables to zero, find T_{06} and T_{60}.

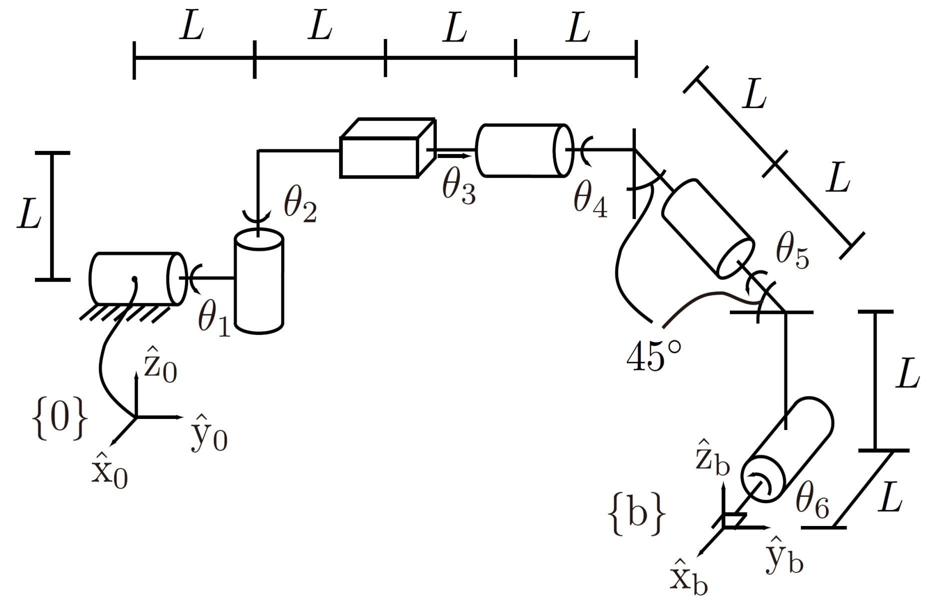


Figure 1: An RRPRRR spatial open chain.

Problem 2

(2 points)

Velocity Kinematics. Exercise 5.11(a) in Lynch and Park, Modern Robotics.

The spatial 3R open chain of Figure 2 is shown in its zero position. Let p be the coordinates of the origin of {b} expressed in {s}. In its zero position, suppose we wish to make the end-effector move with linear velocity $\dot{p} = (10, 0, 0)$. What are the required input joint velocities $\dot{\theta}_1$, $\dot{\theta}_2$ and $\dot{\theta}_3$?

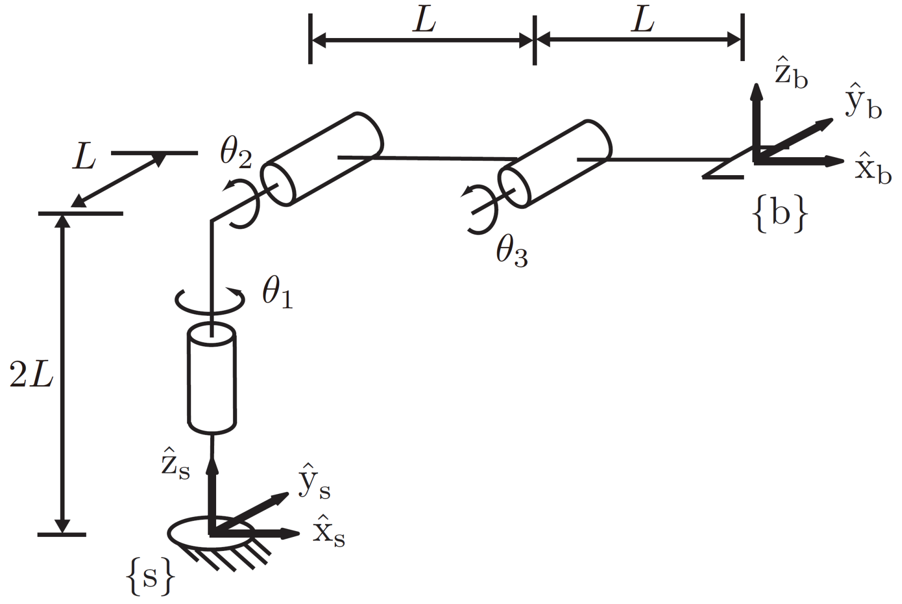


Figure 2: A spatial 3R open chain.

Problem 3

(2 points)

Inverse Kinematics. Exercise 6.4(a) in Lynch and Park, Modern Robotics.

The RRP open chain of Figure 3 is shown in its zero position. Joint axes 1 and 2 intersect at the fixed frame origin, and the end-effector frame origin p is located at $(0, 1, 0)$ when the robot is in its zero position. Suppose that $\theta_1 = 0$. Solve for θ_2 and θ_3 when the end-effector frame origin p is at $(-6, 5, \sqrt{3})$.

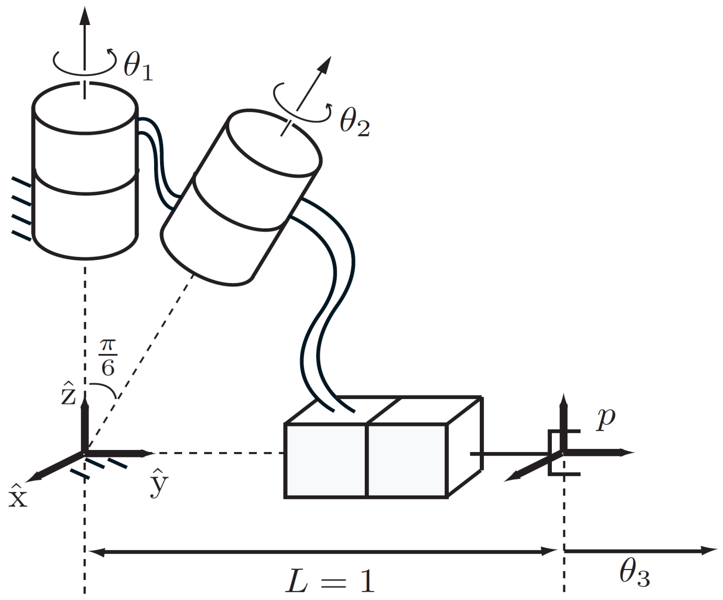


Figure 3: An RRP open chain.

Problem 4

(4 points)

ROS programming, forward kinematics and inverse kinematics.

In this problem, you will learn the forward kinematics and inverse kinematics in ROS. **You can directly use Ubuntu, or Docker or virtual machine to install ROS according to your own set up**. Refer to the ROS wiki page if needed `http://wiki.ros.org/`.

(4.1) Mounting a host folder into Docker if you use Docker. For example, the following command will mount a folder in Windows "C:\data" as a folder "/data" in Docker:

- docker run -it -v C:\data:/data ubuntu:ros

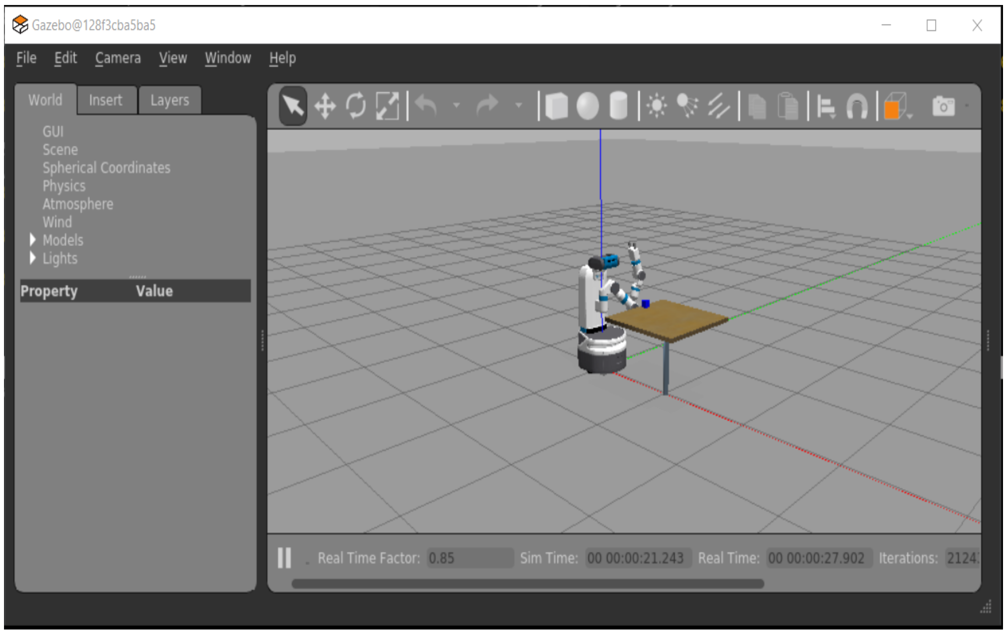
In this way, you can save all your code in the host machine and use them in the Docker environment.

(4.2) Creating a ROS workspace. You can also reuse your workspace from the previous homework. A ROS workspace is a place to store your own ROS packages. Following the link here to create a ROS workspace `http://wiki.ros.org/catkin/Tutorials/create_a_workspace`. You should create the ROS workspace in the mounted folder from the host machine.

(4.3) Install and launch Fetch Gazebo Simulator by following the steps in Homework 2. Use terminator to start multiple windows.

- roslaunch fetch_gazebo simple_grasp.launch.

You shall see the Gazebo environment as in Figure 4.

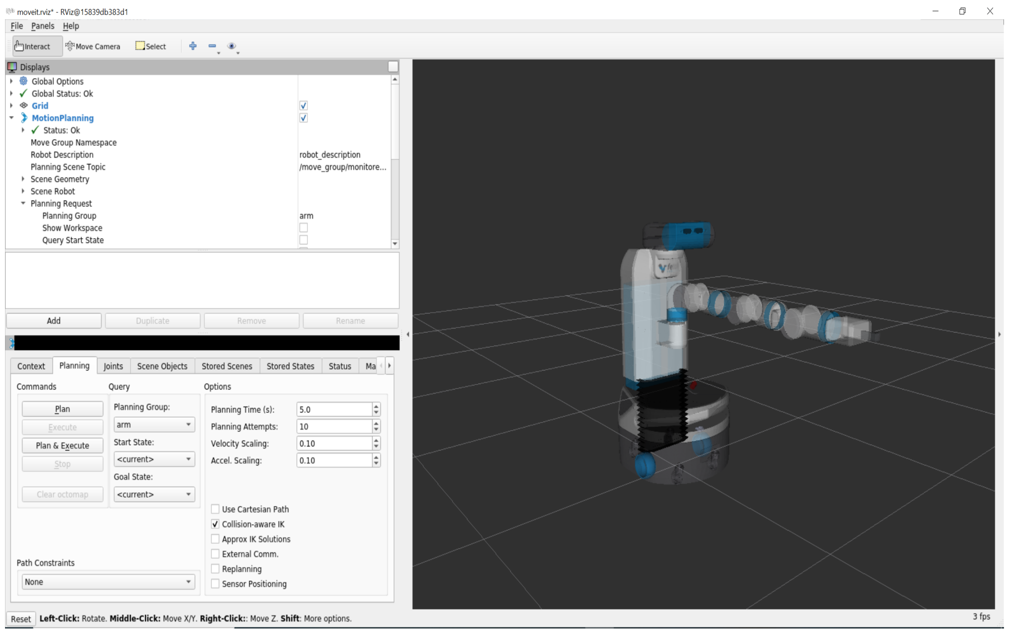


Real Time Factor: 0.85 Sim Time: 00 00:00:21.243 Real Time: 00 00:00:27.902 Iterations: 2124

Figure 4: A Fetch Gazebo Interface.

(4.4) Install MoveIt and the fetch_ros package and then launch MoveIt planning. Follow the steps:

- Moveit is a motion planning package `https://moveit.ros.org/`. Install it by *apt install ros-noetic-moveit*.
- Git clone the source code of the fetch_ros package to the src folder of your ROS workspace: git clone −−branch ros1 `https://github.com/fetchrobotics/fetch_ros.git`
  We need to use the ros1 branch for ROS noetic.
- Build your ROS workspace by catkin_make. You may need to install the following package if you see missing package errors: *ros-noetic-grid-map-costmap-2d*. Use apt install. Make sure that you successfully build the workspace without errors.
- Test your installation by running *roslaunch fetch_moveit_config demo.launch* according to `https://docs.fetchrobotics.com/manipulation.html`. You shall see the Rviz interface as in Figure 5. After the testing, you can kill this launch.
- Start the Moveit planning package by *roslaunch fetch_moveit_config move_group.launch*. You need to see the output as in Figure 6 that indicates the planning is ready.



Global Options
Global Status: Ok
Grid ✓
MotionPlanning ✓
Status: Ok
Move Group Namespace
Robot Description robot_description
Planning Scene Topic /move_group/monitore...
Scene Geometry
Scene Robot
Planning Request
Planning Group arm
Show Workspace ☐
Query Start State ☐

Planning Group: arm
Start State: <current>
Goal State: <current>
Planning Time (s): 5.0
Planning Attempts: 10
Velocity Scaling: 0.10
Accel. Scaling: 0.10
☐ Use Cartesian Path
✓ Collision-aware IK
☐ Approx IK Solutions
☐ External Comm.
☐ Replanning
☐ Sensor Positioning
Path Constraints: None

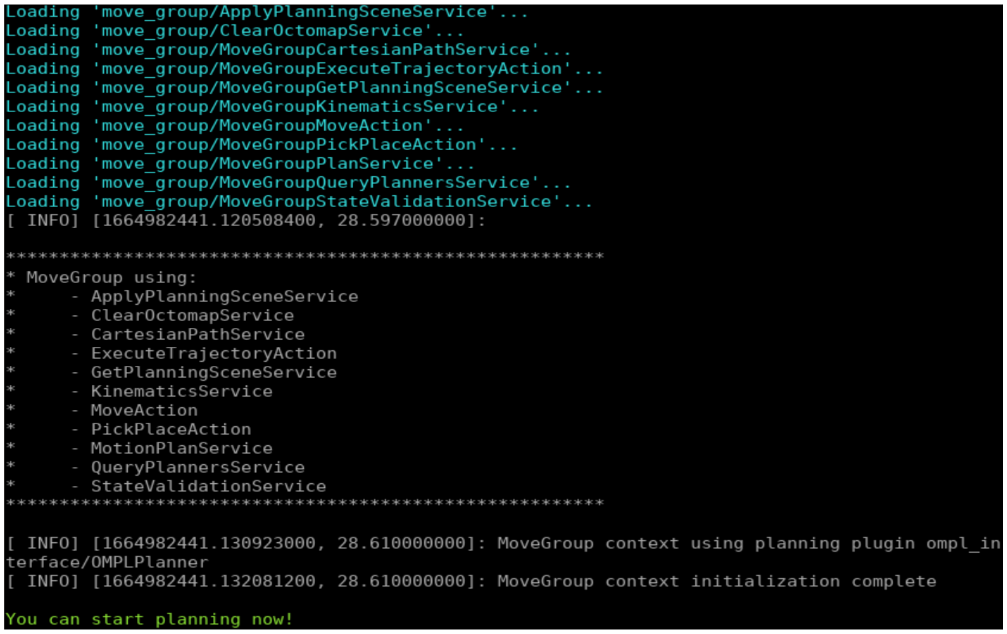
Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click:**: Move Z. **Shift**: More options.

Figure 5: The Moveit Rviz Interface by launching *roslaunch fetch_moveit_config demo.launch*.

(4.5) Up to now, you shall have the Gazebo and Moveit running. In this is coding assignment, you need to use the pose of the demo cube computed from Homework 2 to set up a planning scene in Moveit, and use track_ik `https://bitbucket.org/traclabs/trac_ik/src/master/` for inverse kinematics.

Download the planning_scene_block.py file from eLearning. This python script first queries the pose of the cube in the Gazebo environment as in Homework 2. Then it creates a planning scene using Moveit `https://ros-planning.github.io/moveit_tutorials/` and adds the cube block into the planning scene. Finally, it computes inverse kinematics of the robot using track_ik.

To install track_ik, run



```
Loading 'move_group/ApplyPlanningSceneService'...
Loading 'move_group/ClearOctomapService'...
Loading 'move_group/MoveGroupCartesianPathService'...
Loading 'move_group/MoveGroupExecuteTrajectoryAction'...
Loading 'move_group/MoveGroupGetPlanningSceneService'...
Loading 'move_group/MoveGroupKinematicsService'...
Loading 'move_group/MoveGroupMoveAction'...
Loading 'move_group/MoveGroupPickPlaceAction'...
Loading 'move_group/MoveGroupPlanService'...
Loading 'move_group/MoveGroupQueryPlannersService'...
Loading 'move_group/MoveGroupStateValidationService'...
[ INFO] [1664982441.120508400, 28.597000000]:

********************************************************
* MoveGroup using:
*     - ApplyPlanningSceneService
*     - ClearOctomapService
*     - CartesianPathService
*     - ExecuteTrajectoryAction
*     - GetPlanningSceneService
*     - KinematicsService
*     - MoveAction
*     - PickPlaceAction
*     - MotionPlanService
*     - QueryPlannersService
*     - StateValidationService
********************************************************

[ INFO] [1664982441.130923000, 28.610000000]: MoveGroup context using planning plugin ompl_in
terface/OMPLPlanner
[ INFO] [1664982441.132081200, 28.610000000]: MoveGroup context initialization complete

You can start planning now!
```

Figure 6: Output by launching *roslaunch fetch_moveit_config move_group.launch*.

- *apt-get install ros-noetic-trac-ik*

Check the readme in `https://bitbucket.org/traclabs/trac_ik/src/HEAD/trac_ik_python/` to understand how to use it.

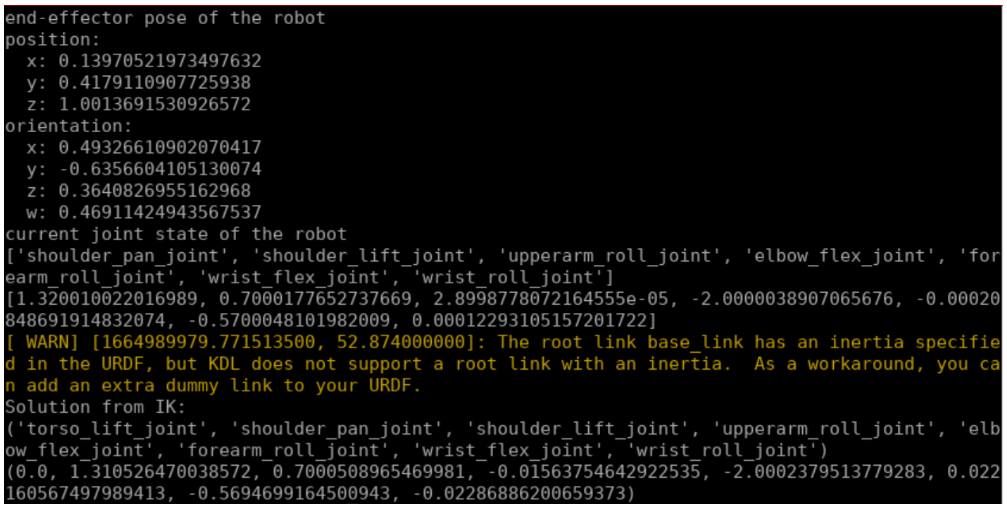
Finish the implementation of the TODOs in the python script. Then you can run the python script. Figure 7 shows the output from the script. You need to see the compute IK solution is close to the current joint coordinates of the robot.

(4.6) Visualize the planning scene with Rviz. Use another terminator window to start Rviz with command: *rosrun rviz rviz*. We need to keep the Gazebo and Moveit running. Follow the steps:

- In Global Options, change Fixed Frame from map to base_link.
- Click the Add button to add a Robot Model.
- Click the Add button to add an Image. Change the image topic to "/head_camera/rgb/image_-raw".
- Click the Add button to add a PlanningScene under moveit_ros_visualization.

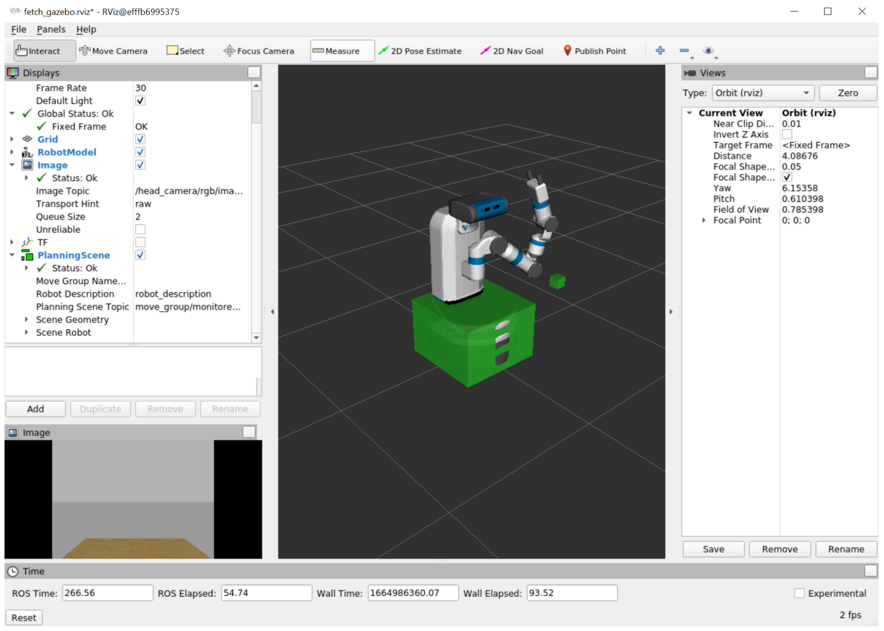
After these steps, you should see a Rviz window as in Figure 8.

Submission guideline: Upload your implemented planning_scene_block.py file and the screen capture of the planning scene in (4.6) to eLearning.



```
end-effector pose of the robot
position:
  x: 0.13970521973497632
  y: 0.4179110907725938
  z: 1.0013691530926572
orientation:
  x: 0.49326610902070417
  y: -0.6356604105130074
  z: 0.3640826955162968
  w: 0.46911424943567537
current joint state of the robot
['shoulder_pan_joint', 'shoulder_lift_joint', 'upperarm_roll_joint', 'elbow_flex_joint', 'for
earm_roll_joint', 'wrist_flex_joint', 'wrist_roll_joint']
[1.320010022016989, 0.7000177652737669, 2.8998778072164555e-05, -2.0000038907065676, -0.00020
848691914832074, -0.5700048101982009, 0.00012293105157201722]
[ WARN] [1664989979.771513500, 52.874000000]: The root link base_link has an inertia specifie
d in the URDF, but KDL does not support a root link with an inertia.  As a workaround, you ca
n add an extra dummy link to your URDF.
Solution from IK:
('torso_lift_joint', 'shoulder_pan_joint', 'shoulder_lift_joint', 'upperarm_roll_joint', 'elb
ow_flex_joint', 'forearm_roll_joint', 'wrist_flex_joint', 'wrist_roll_joint')
(0.0, 1.310526470038572, 0.7000508965469981, -0.01563754642922535, -2.0002379513779283, 0.022
160567497989413, -0.5694699164500943, -0.02286886200659373)
```

Figure 7: Output from the IK solver.



```
Frame Rate                30
Default Light             ✓
Global Status: Ok
  Fixed Frame             OK
Grid                      ✓
RobotModel                ✓
Image                     ✓
  Status: Ok
  Image Topic             /head_camera/rgb/ima...
  Transport Hint          raw
  Queue Size              2
  Unreliable              ☐
TF                        ☐
PlanningScene             ✓
  Status: Ok
  Move Group Name...
  Robot Description       robot_description
  Planning Scene Topic    move_group/monitore...
  Scene Geometry
  Scene Robot
```

```
Type: Orbit (rviz)
Current View              Orbit (rviz)
  Near Clip Di...         0.01
  Invert Z Axis           ☐
  Target Frame            <Fixed Frame>
  Distance                4.08676
  Focal Shape...          0.05
  Focal Shape...          ✓
  Yaw                     6.15358
  Pitch                   0.610398
  Field of View           0.785398
  Focal Point             0; 0; 0
```

```
ROS Time: 266.56   ROS Elapsed: 54.74   Wall Time: 1664986360.07   Wall Elapsed: 93.52
2 fps
```

Figure 8: Rviz visualization of the planning scene.