ROS机械臂开发

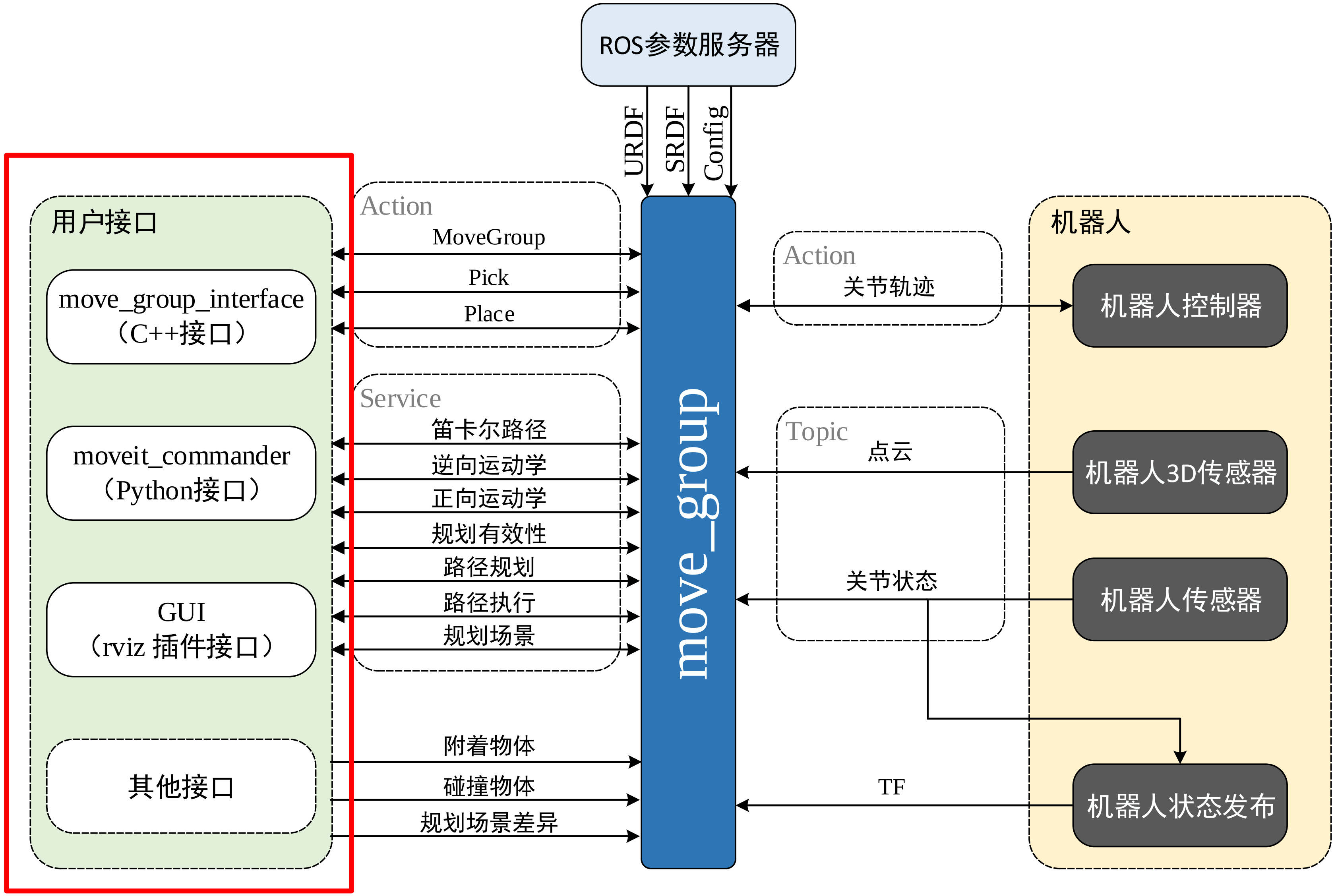
——7.MoveIt!编程接口

目录

目录

- 1. MoveIt!的编程接口

1. MoveIt!的编程接口



MoveIt!的核心节点——move_group

1. MoveIt!的编程接口 —— C++ & Python

"move_group" Python Interface

```python
group = moveit_commander.MoveGroupCommander("left_arm")

pose_target = geometry_msgs.msg.Pose()
pose_target.orientation.w = 1.0
pose_target.position.x = 0.7
pose_target.position.y = -0.05
pose_target.position.z = 1.1
group.set_pose_target(pose_target)

plan1 = group.plan()
```

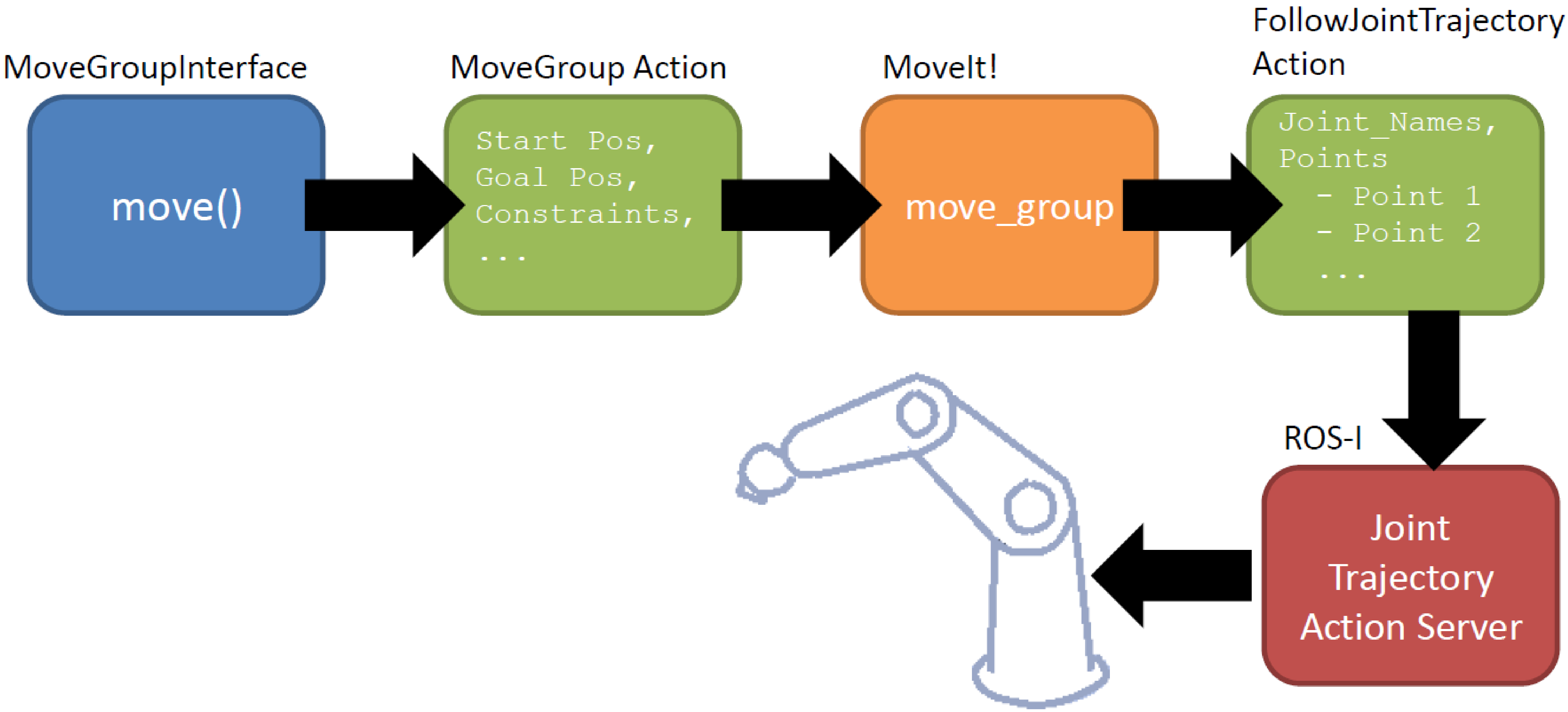
Python

"move_group" C++ Interface

```cpp
moveit::planning_interface::MoveGroup group("right_arm");

geometry_msgs::Pose target_pose;
target_pose.orientation.w = 1.0;
target_pose.position.x = 0.28;
target_pose.position.y = -0.7;
target_pose.position.z = 1.0;
group.setPoseTarget(target_pose);

moveit::planning_interface::MoveGroup::Plan my_plan;
bool success = group.plan(my_plan);
```

C++

1. MoveIt!的编程接口 —— 接口实现流程



1. MoveIt!的编程接口 —— 编程模式

- 连接控制需要的规划组
- 设置目标位姿（关节空间或笛卡尔空间）
- 设置运动约束（可选）
- 使用MoveIt!规划一条到达目标的轨迹
- 修改轨迹（如速度等参数）
- 执行规划出的轨迹

1. MoveIt!的编程接口 —— 官方API文档

moveit.ros.org/code-api/

MoveIt! Home Install Documentation Tutorials Robots About Support Contribute Blog

ROS Kinetic Code API

Move Group Interface

- MoveGroupInterface class - the main C++ interface to the *move_group_node*. `MoveGroupInterface` is the successor to the `MoveGroup` class from previous releases, which is now deprecated.
- PlanningSceneInterface class - a C++ interface to the planning scene
- MoveIt! commander - documentation for the MoveIt! commander.

MoveIt! ROS

This API is meant for advanced developers. Most users should use the Move Group interface (above).

- Planning - The planning components in MoveIt! ROS, especially the planning scene, kinematics and monitors
- Move Group - The *move_group_node*
- Perception - The perception components in MoveIt! ROS
- Robot Interaction - The Interactivity components in MoveIt! ROS

MoveIt! Core

This API is meant for advanced developers. Most users should use the Move Group interface (above).

- Core - The core components in MoveIt! for kinematics, planning scene, constraints, motion planning, collision checking and plugin interfaces

MoveIt! OMPL Interface

This API is meant for advanced developers. Most users should use the Move Group interface (above).

- OMPL Interface - The set of classes that allow MoveIt! to talk with OMPL.

BLOG

The MoveIt! Project is Selected for 3 Google Summer of Code Grants

We are excited to announce that MoveIt! was selected for 3 Google Summer of Code (GSoC) grants for the summer of 2018.

MoveIt! Migrating from Google Groups to ROS Discourse

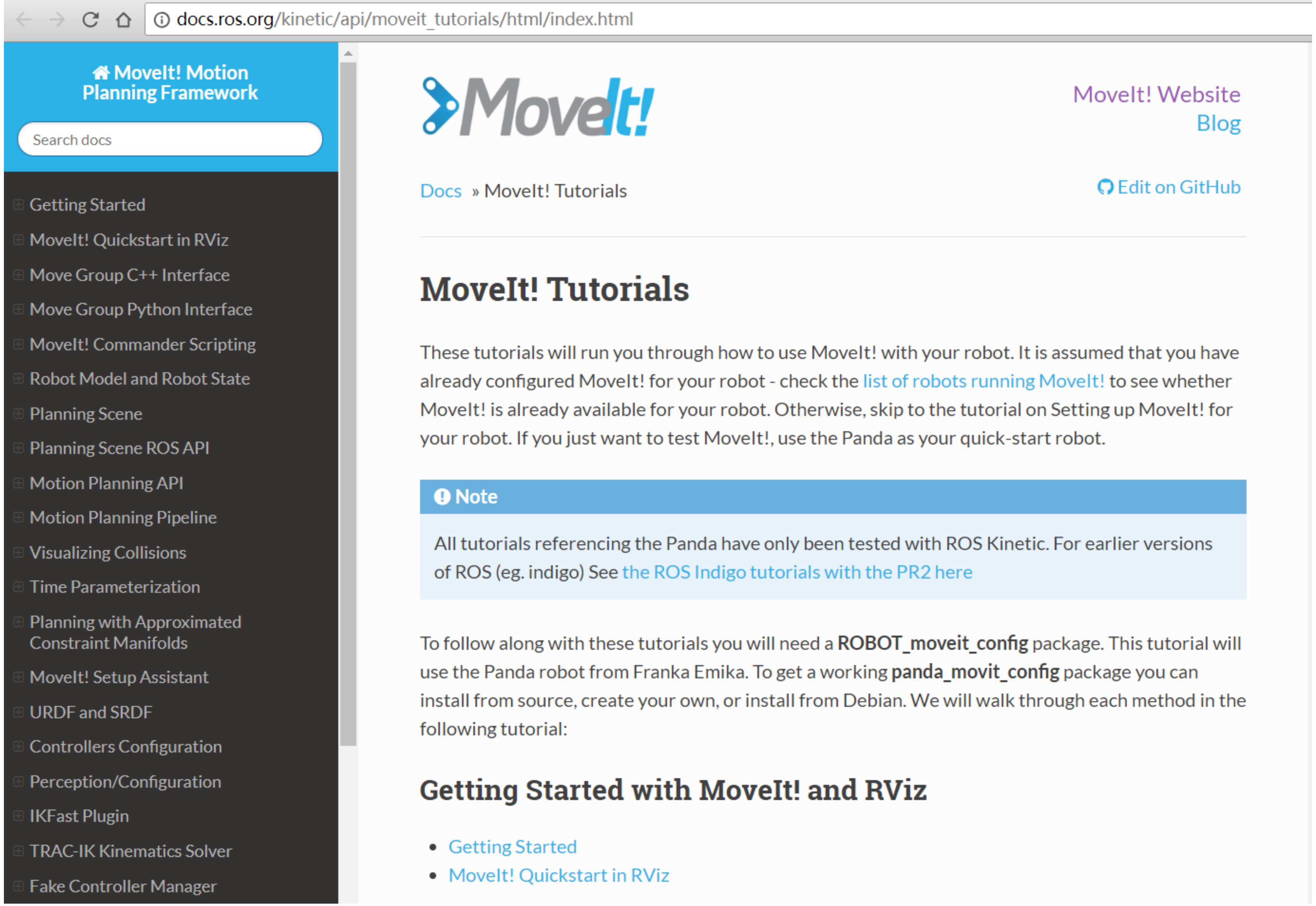
In recent years there has been a migration related to ROS discussions, Q&A, and collaboration to ROS Discourse. Inspired by ROS-Industrial's migration this month away from Google Groups, and we believe its time for MoveIt! to migrate as well.

Announcing MoveIt! Tutorials Codesprint with Franka Emika & PickNik

We are excited to announce that the robotic arm manufacturer Franka Emika is partnering with PickNik to sponsor a MoveIt! codesprint to further improve MoveIt!'s tutorials, documentation, and website.

http://moveit.ros.org/code-api/

1. MoveIt!的编程接口 —— 官方基础教程

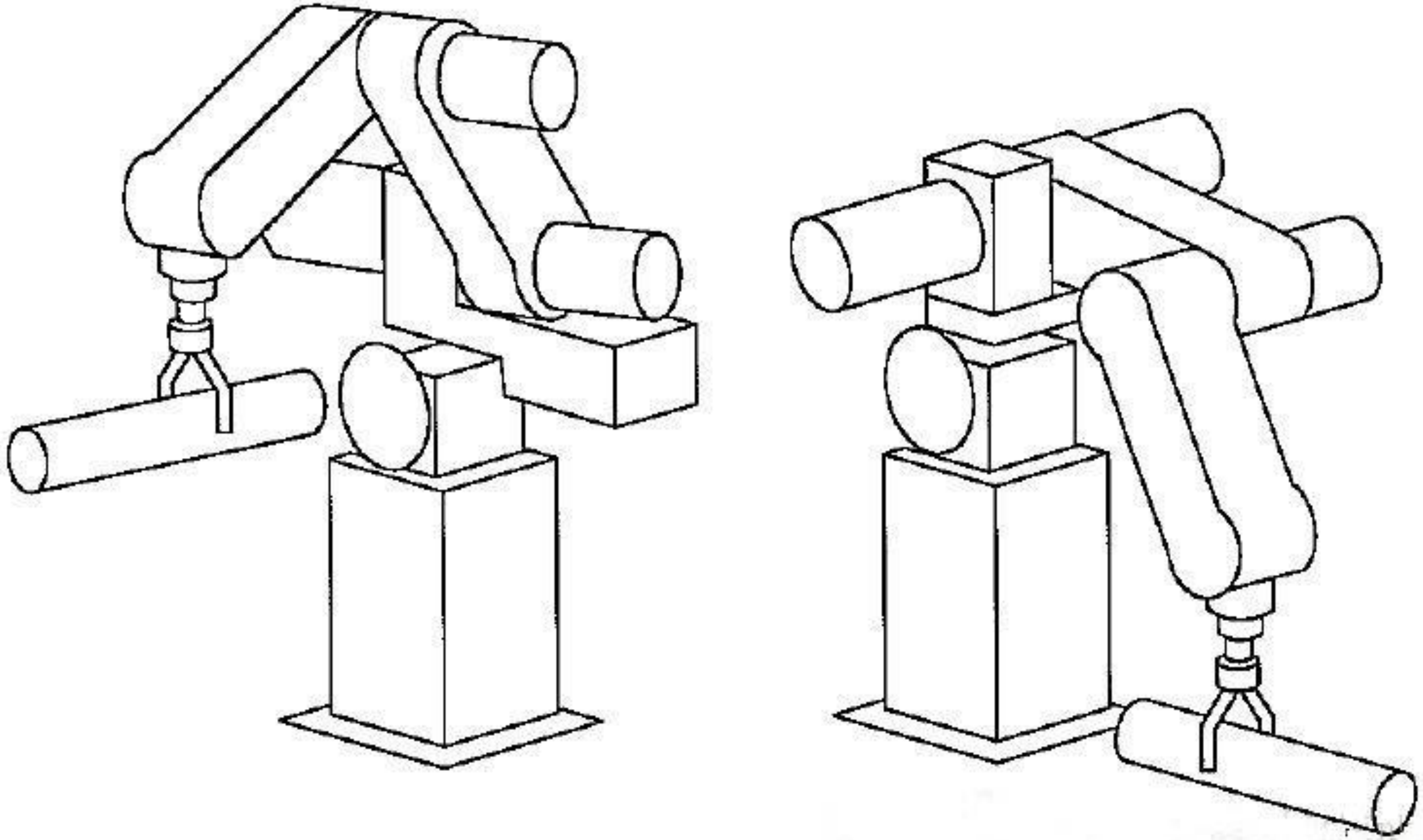


http://docs.ros.org/kinetic/api/moveit_tutorials/html/index.html

目录

➢ 2. 关节空间运动规划

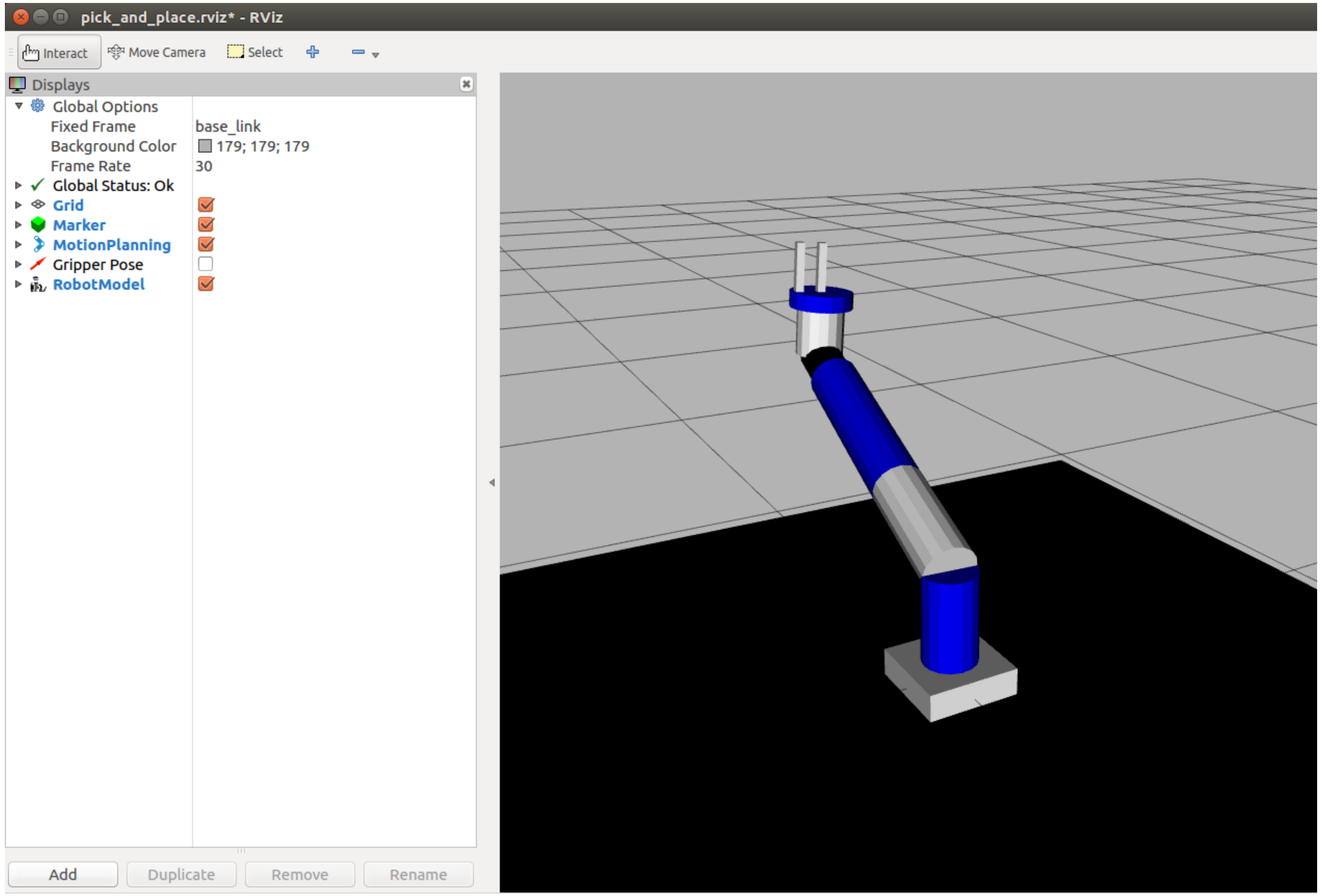
2. 关节空间运动规划



点到点运动：不需要在笛卡尔空间规划末端运动轨迹，机器人各个关节运动不需要联动。

2. 关节空间运动规划 —— 正向运动学

```python
# 初始化需要使用move group控制的机械臂中的arm group
arm = moveit_commander.MoveGroupCommander('arm')

# 初始化需要使用move group控制的机械臂中的gripper group
gripper = moveit_commander.MoveGroupCommander('gripper')

# 设置机械臂和夹爪的允许误差值
arm.set_goal_joint_tolerance(0.001)
gripper.set_goal_joint_tolerance(0.001)

# 控制机械臂先回到初始化位置
arm.set_named_target('home')
arm.go()
rospy.sleep(2)

# 设置夹爪的目标位置，并控制夹爪运动
gripper.set_joint_value_target([0.01])
gripper.go()
rospy.sleep(1)

# 设置机械臂的目标位置，使用六轴的位置数据进行描述（单位：弧度）
joint_positions = [-0.0867, -1.274, 0.02832, 0.0820, -1.273, -0.003]
arm.set_joint_value_target(joint_positions)

# 控制机械臂完成运动
arm.go()
rospy.sleep(1)

# 关闭并退出moveit
moveit_commander.roscpp_shutdown()
moveit_commander.os._exit(0)
```



关节空间规划例程

$ roslaunch marm_moveit_config demo.launch

$ rosrun marm_planning moveit_fk_demo.py

2. 关节空间运动规划 —— 正向运动学

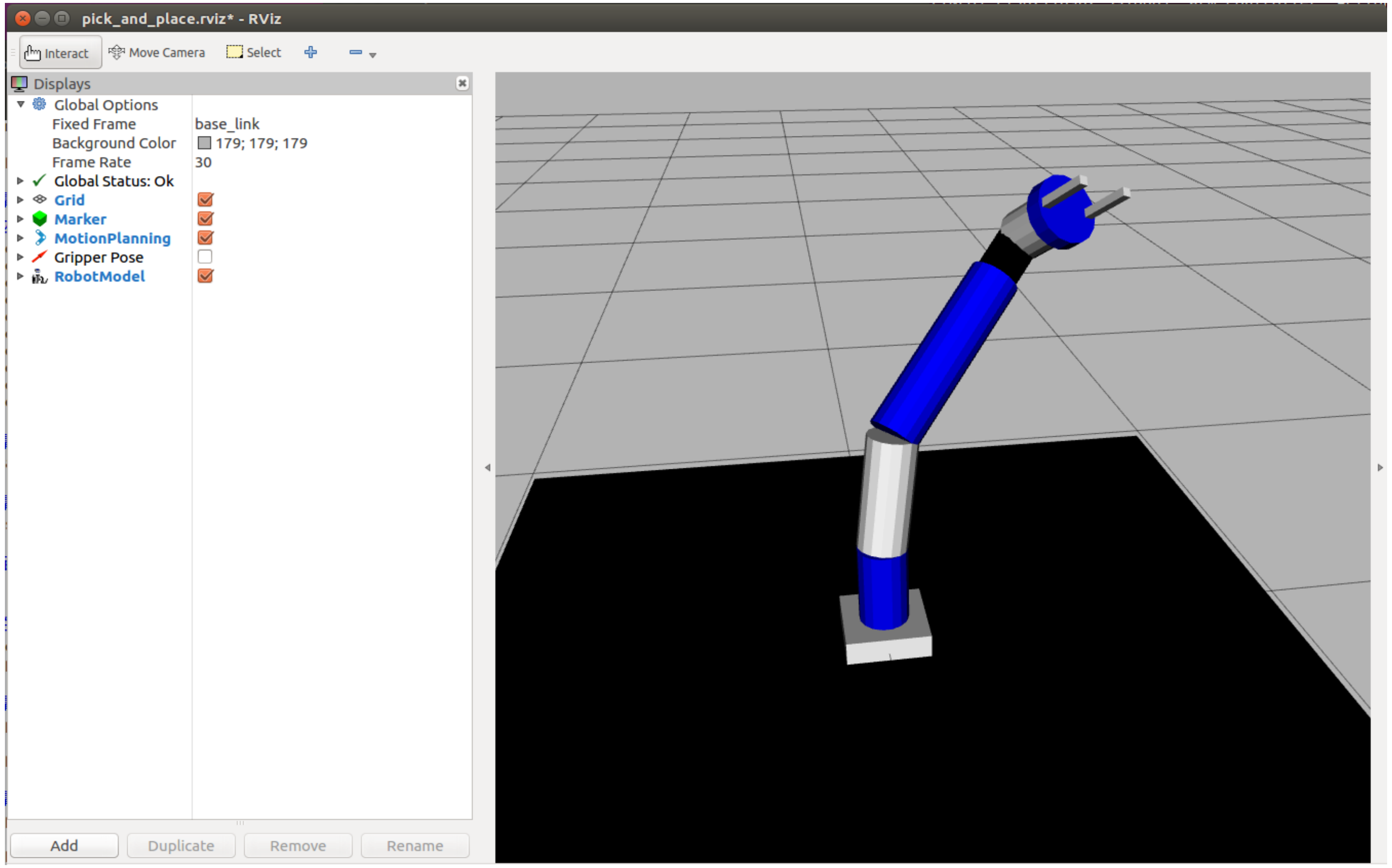
关键API的实现步骤

```
arm = moveit_commander.MoveGroupCommander('arm')

joint_positions = [-0.0867, -1.274, 0.02832, 0.0820, -1.273, -0.003]

arm.set_joint_value_target(joint_positions)

arm.go()
```

- 创建规划组的控制对象；
- 设置关节空间运动的目标位姿；
- 完成规划并控制机械臂完成运动。

2. 关节空间运动规划 —— 逆向运动学

```
# 设置机器臂当前的状态作为运动初始状态
arm.set_start_state_to_current_state()

# 设置机械臂终端运动的目标位姿
arm.set_pose_target(target_pose, end_effector_link)

# 规划运动路径
traj = arm.plan()

# 按照规划的运动路径控制机械臂运动
arm.execute(traj)
rospy.sleep(1)

# 控制机械臂终端向右移动5cm
arm.shift_pose_target(1, -0.05, end_effector_link)
arm.go()
rospy.sleep(1)

# 控制机械臂终端反向旋转90度
arm.shift_pose_target(3, -1.57, end_effector_link)
arm.go()
rospy.sleep(1)

# 控制机械臂回到初始化位置
arm.set_named_target('home')
arm.go()
```



工作空间规划例程

$ roslaunch marm_moveit_config demo.launch

$ rosrun marm_planning moveit_ik_demo.py

2. 关节空间运动规划 —— 逆向运动学

关键API的实现步骤

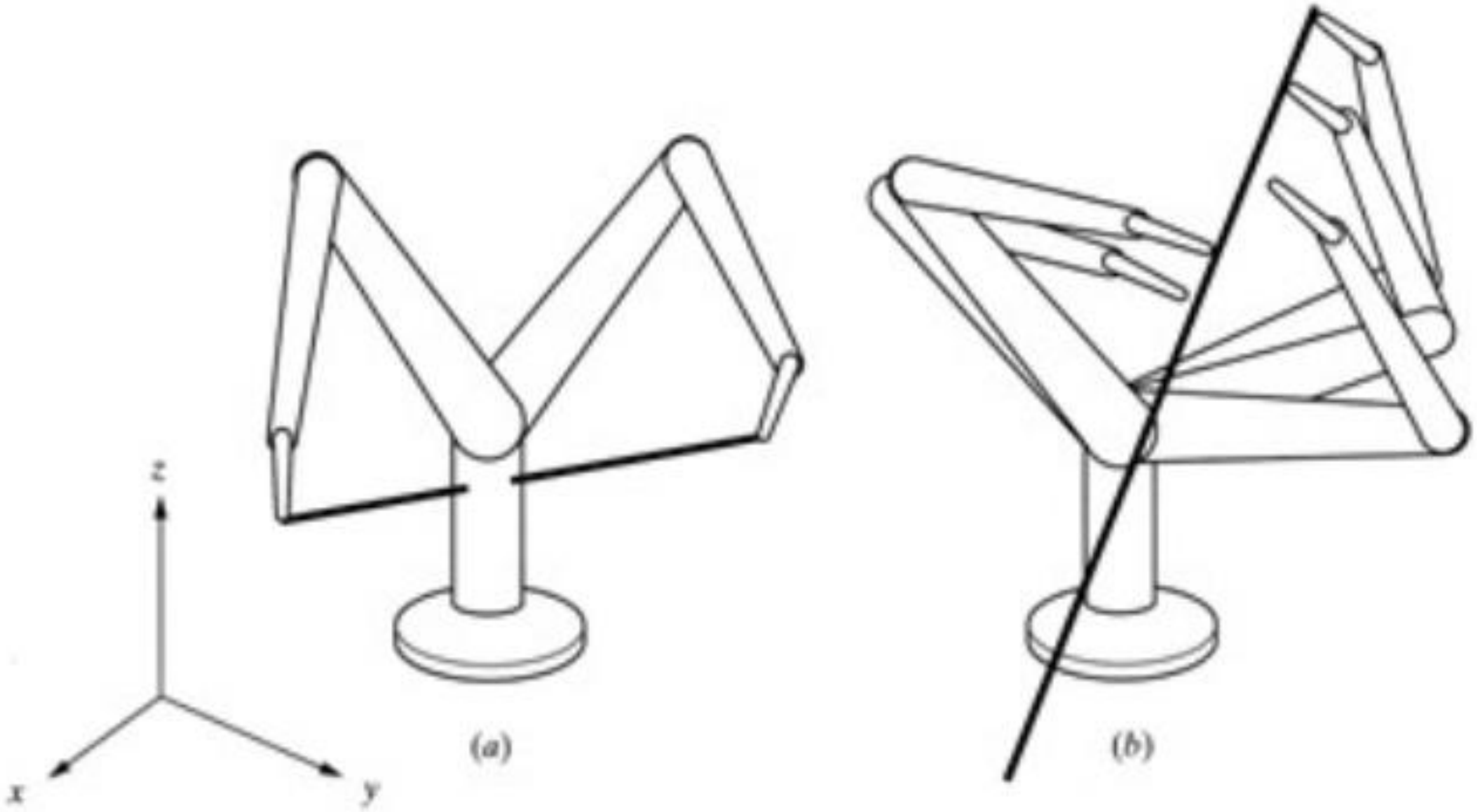
```
arm = moveit_commander.MoveGroupCommander('arm')
end_effector_link = arm.get_end_effector_link()

reference_frame = 'base_link'
arm.set_pose_reference_frame(reference_frame)

arm.set_start_state_to_current_state()
arm.set_pose_target(target_pose, end_effector_link)

traj = arm.plan()
arm.execute(traj)

arm.shift_pose_target(1, -0.05, end_effector_link)
arm.go()
```

- 创建规划组的控制对象；
- 获取机器人的终端link名称；
- 设置目标位姿对应的参考坐标系和起始、终止位姿；
- 完成规划并控制机械臂完成运动。

目录

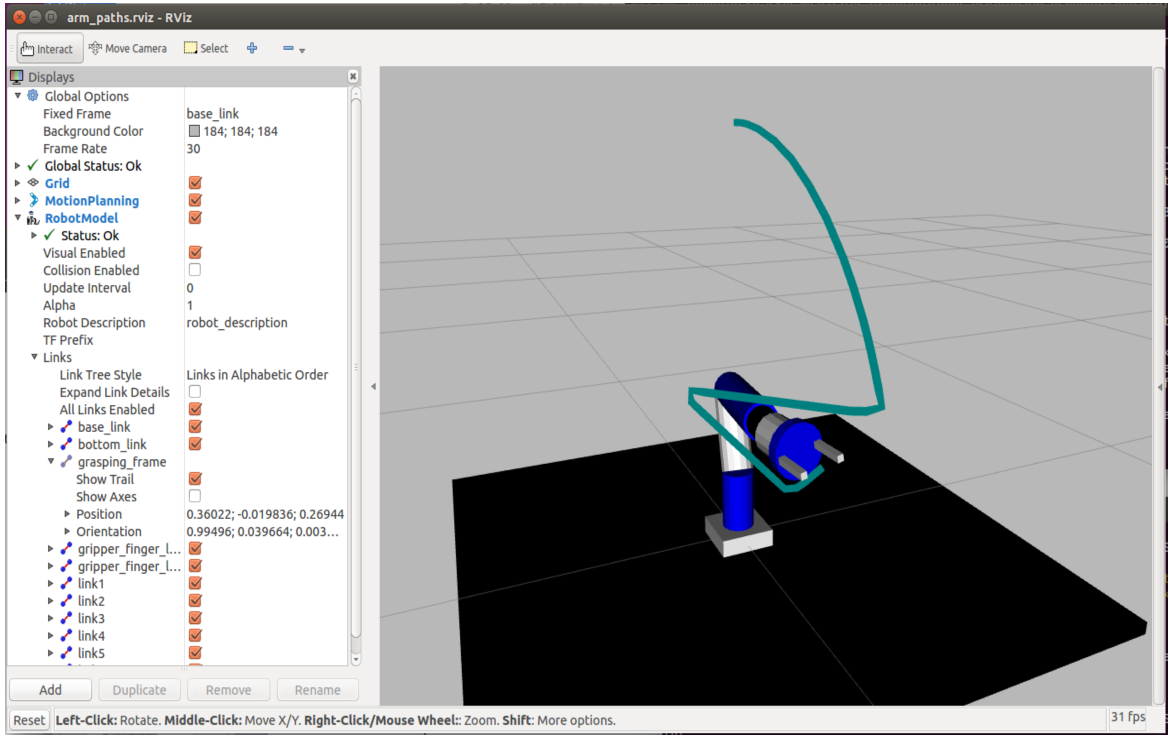
➢3.笛卡尔运动规划

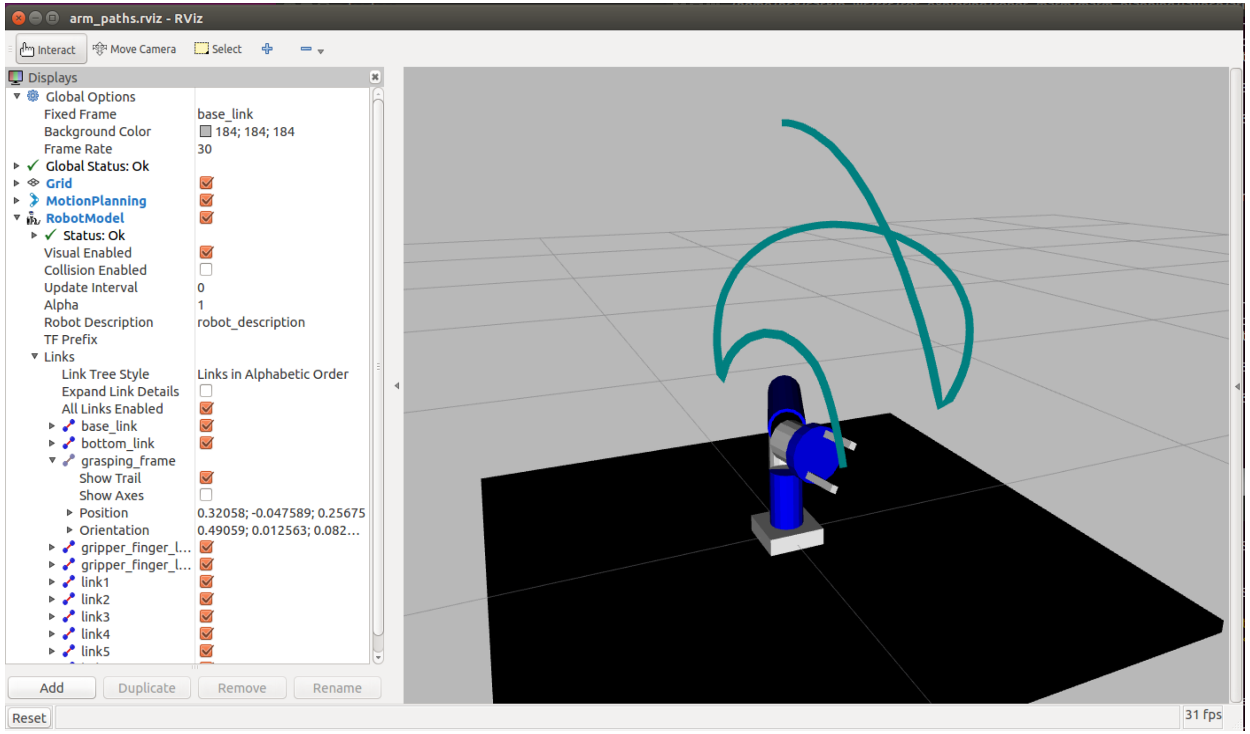
3.笛卡尔运动规划



笛卡尔路径约束，路径点之间的路径形状是一条直线。

3.笛卡尔运动规划





工作空间
规划例程

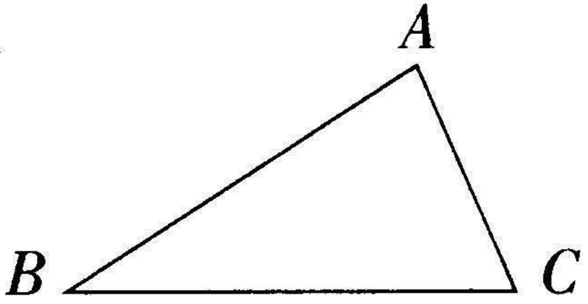
$ roslaunch marm_moveit_config demo.launch

$ rosrun marm_planning moveit_cartesian_demo.py _cartesian:=True（走直线）

$ rosrun marm_planning moveit_cartesian_demo.py _cartesian:=False（走曲线）

3.笛卡尔运动规划

关键API的实现步骤



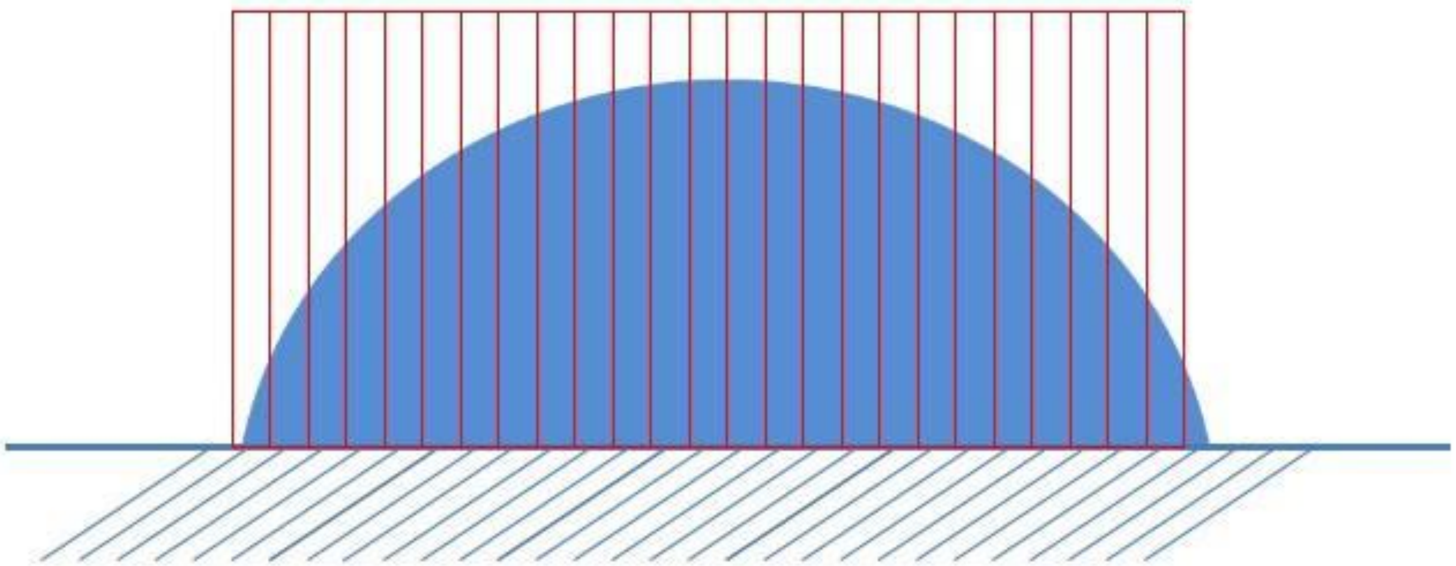

```
(plan, fraction) = arm.compute_cartesian_path (
                         waypoints,    # waypoint poses, 路点列表
                         0.01,         # eef_step, 终端步进值
                         0.0,          # jump_threshold, 最小移动值
                         True)         # avoid_collisions, 避障规划
```

返回值

- plan：规划出来的运动轨迹
- fraction：描述规划成功的轨迹在给定路点列表中的覆盖率[0~1]。如果fraction小于1，说明给定的路点列表没办法完整规划。

3.笛卡尔运动规划

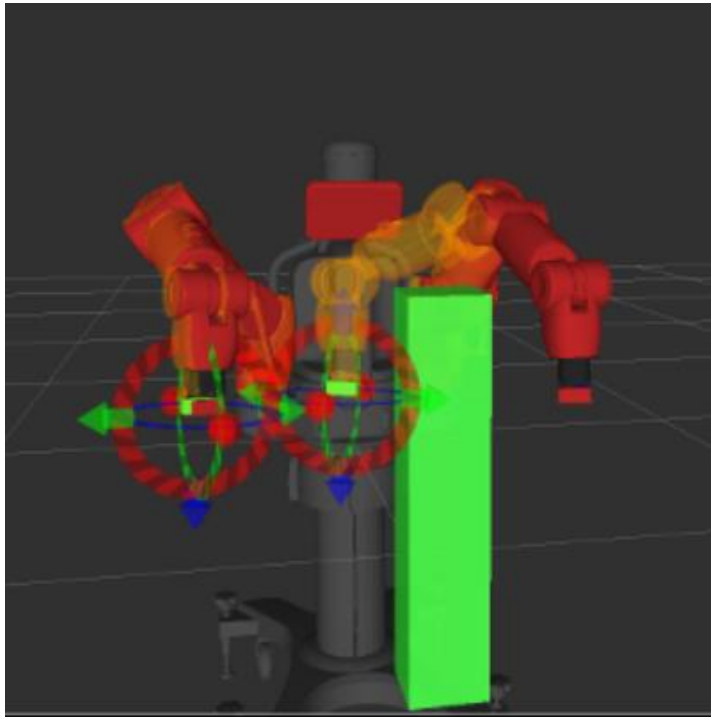
如何走出笛卡尔空间下的圆弧轨迹？

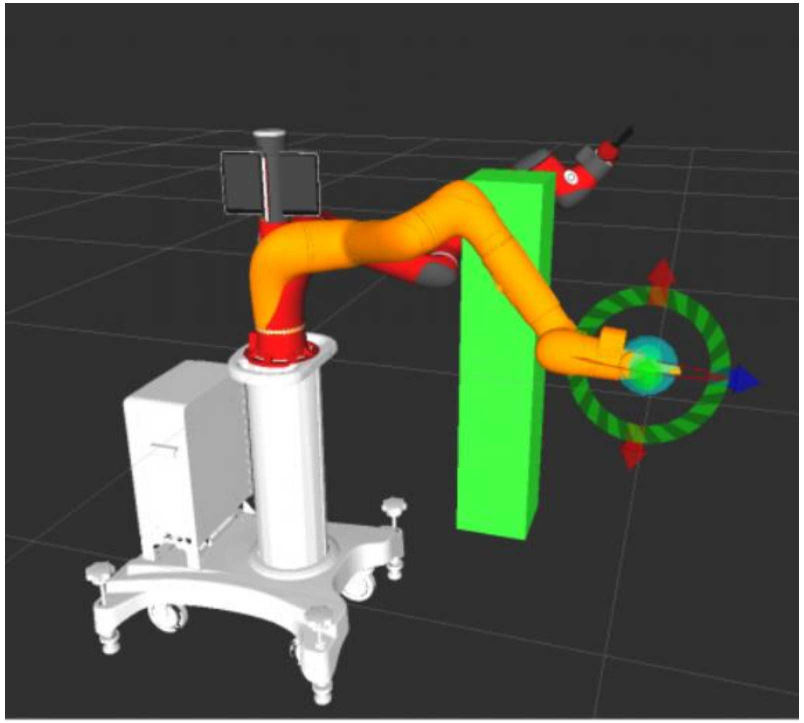


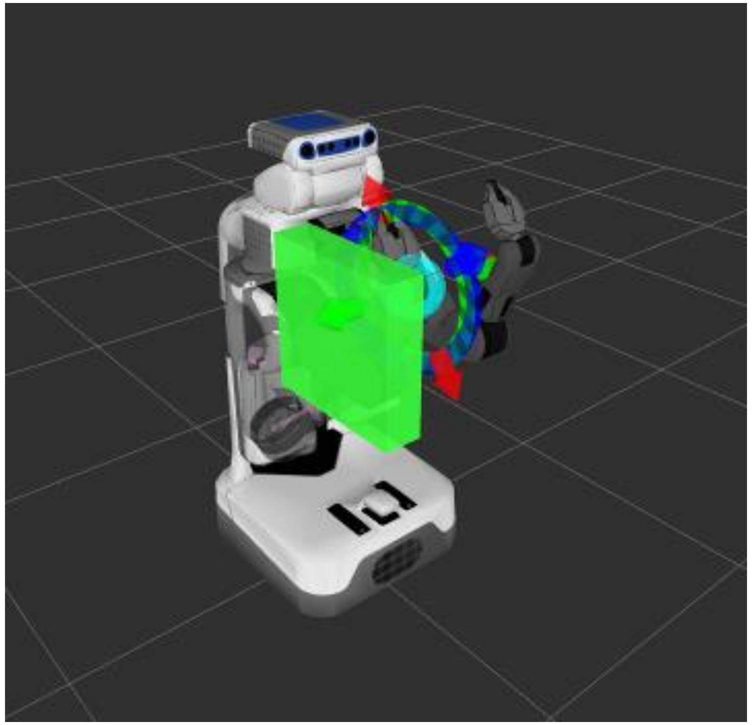
目录

➢4. 碰撞检测

4. 碰撞检测







MoveIt!可以在运动规划时检测碰撞，并规划轨迹绕过障碍

4. 碰撞检测 —— 规划场景

➢ 监听信息

- 状态信息

（ State Information ）

机器人的关节话题joint_states；

- 传感器信息

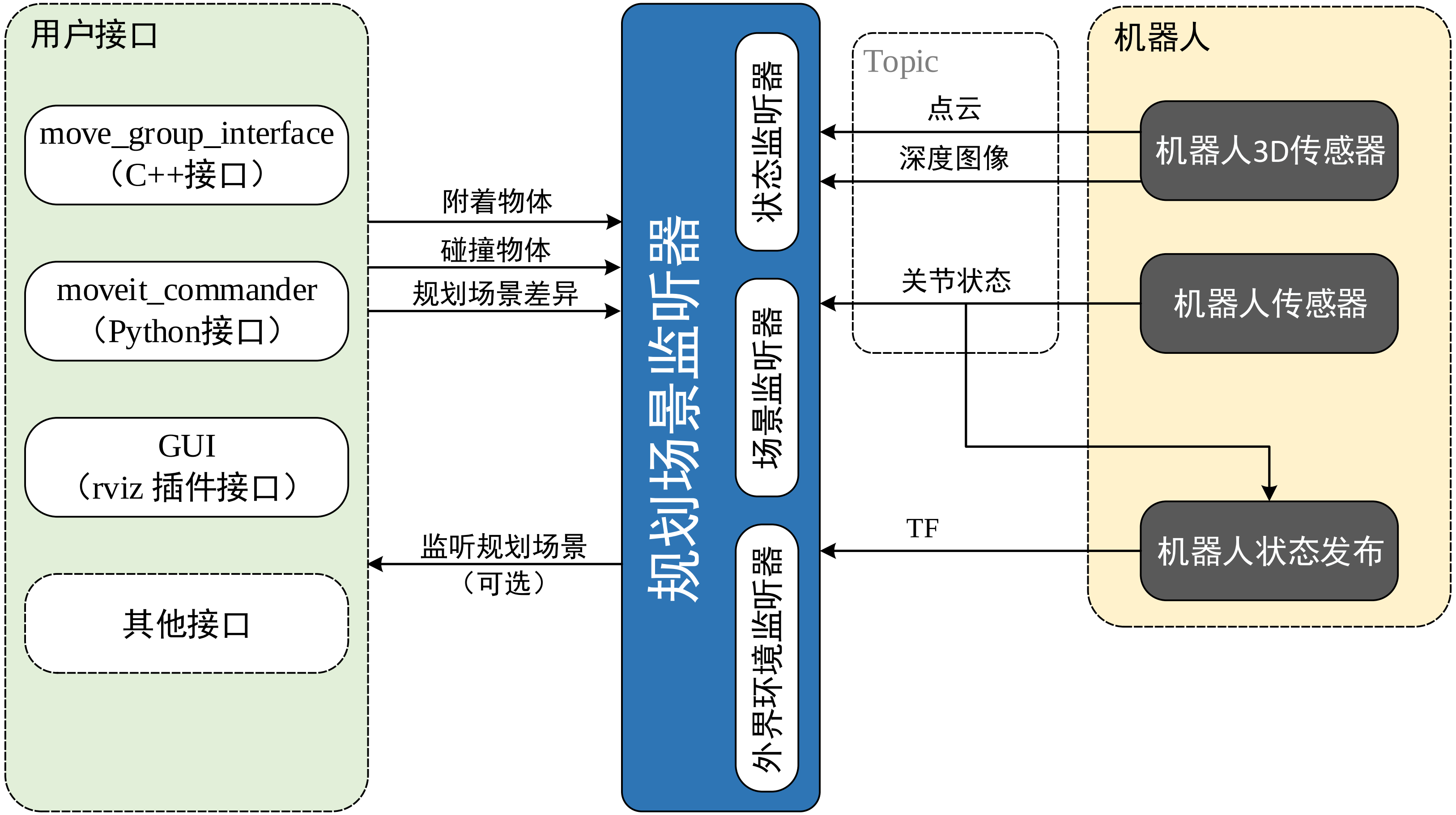
（ Sensor Information ）

机器人的传感器数据；

- 外界环境信息

（ World geometry information ）

通过传感器建立的周围环境信息。



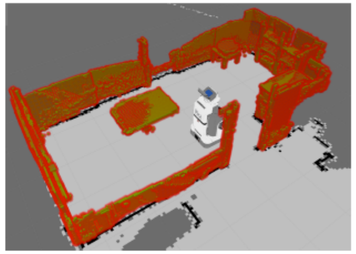
规划场景模块的结构

4. 碰撞检测 —— 碰撞检测算法

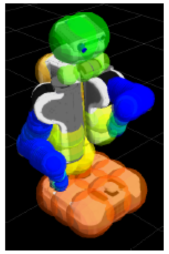
- MoveIt!使用Collision World 对象进行碰撞检测；
- 采用FCL（Flexible Collision Library）功能包实现；
- 碰撞检测是运动规划中最耗时的运算之一，往往会占用90%左右的时间，为了减少计算量，可以在MoveIt! Setup Assistant工具中设置免检冲突矩阵（ACM，Allowed Collision Matrix ）进行优化。

Collision Checking

- FCL - Flexible Collision Library*
  - parallelizable collision checking
  - Maximum about 2-3,000 full body collision checks for the PR2 per second
    - ✓ with realtime sensor data
  - + high fidelity mesh model



- Proximity Collision Detection

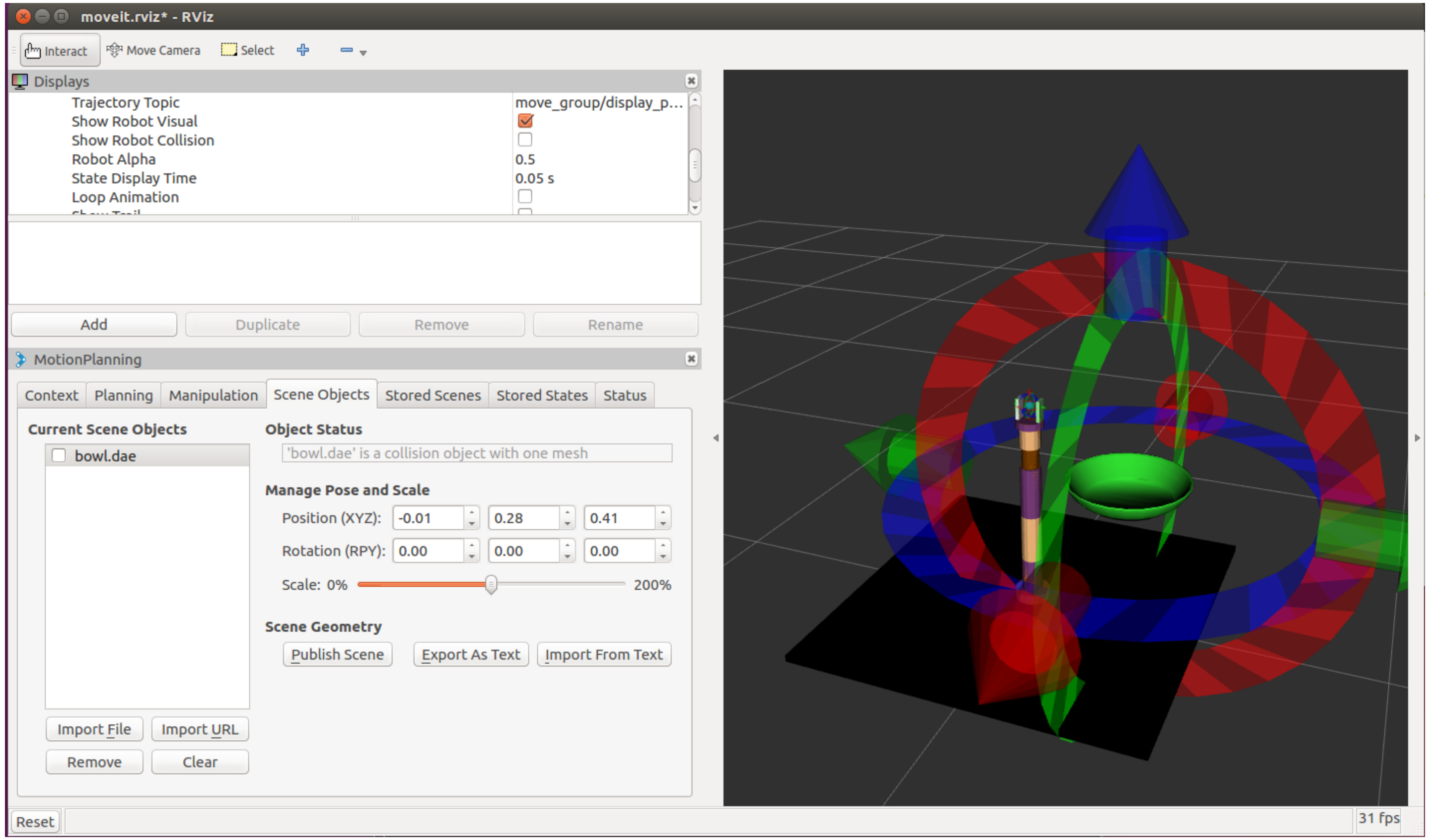


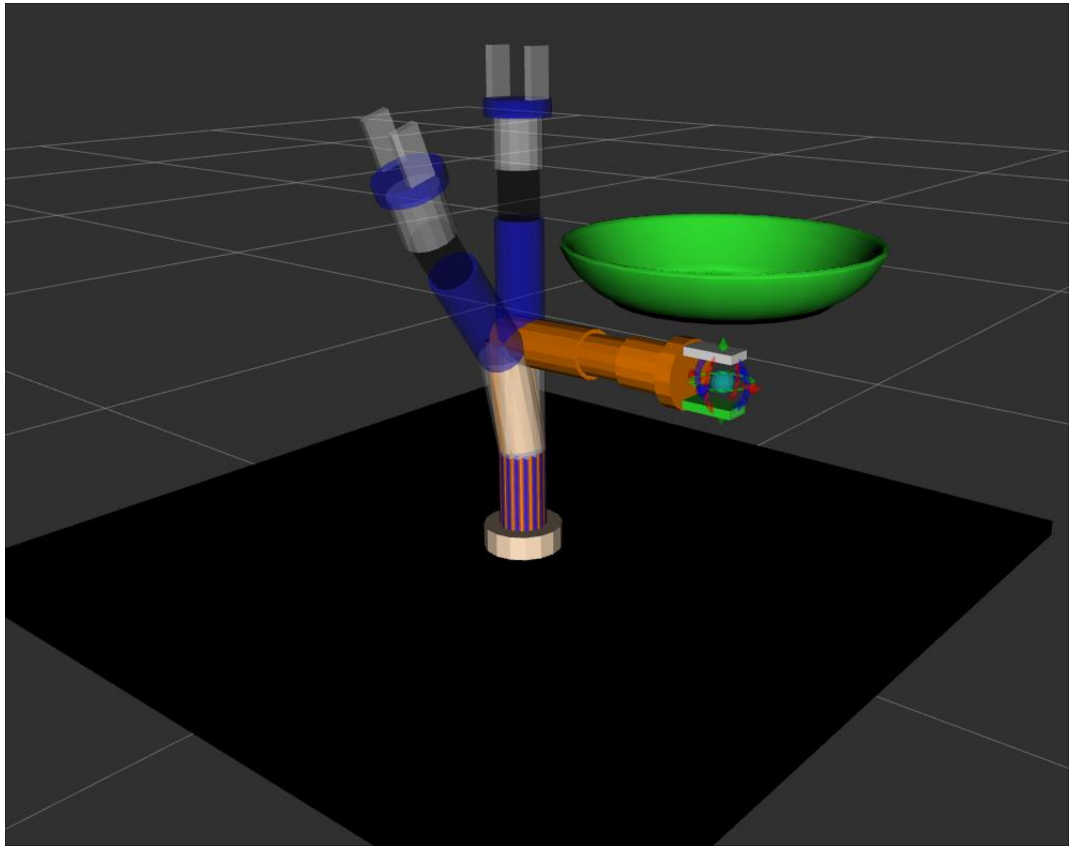
  - Uses 3D distance transform to determine distance to nearest obstacle and gradient
  - + very fast - 40 to 80,000 collision checks per second for the full body of the PR2
  - - not as accurate

*Jia Pan, Ioan Sucan, Sachin Chitta, Dinesh Manocha

*参考链接：http://gamma.cs.unc.edu/FCL/fcl_docs/webpage/generated/index.html

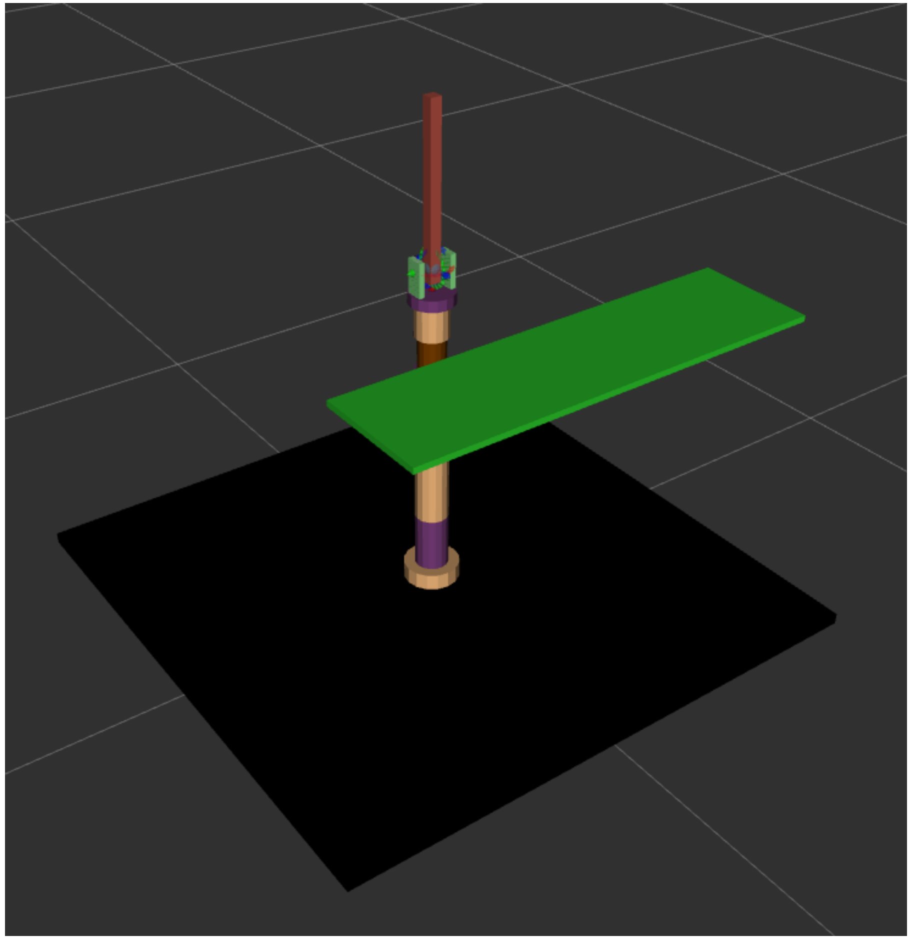
4. 碰撞检测 —— 可视化插件添加障碍

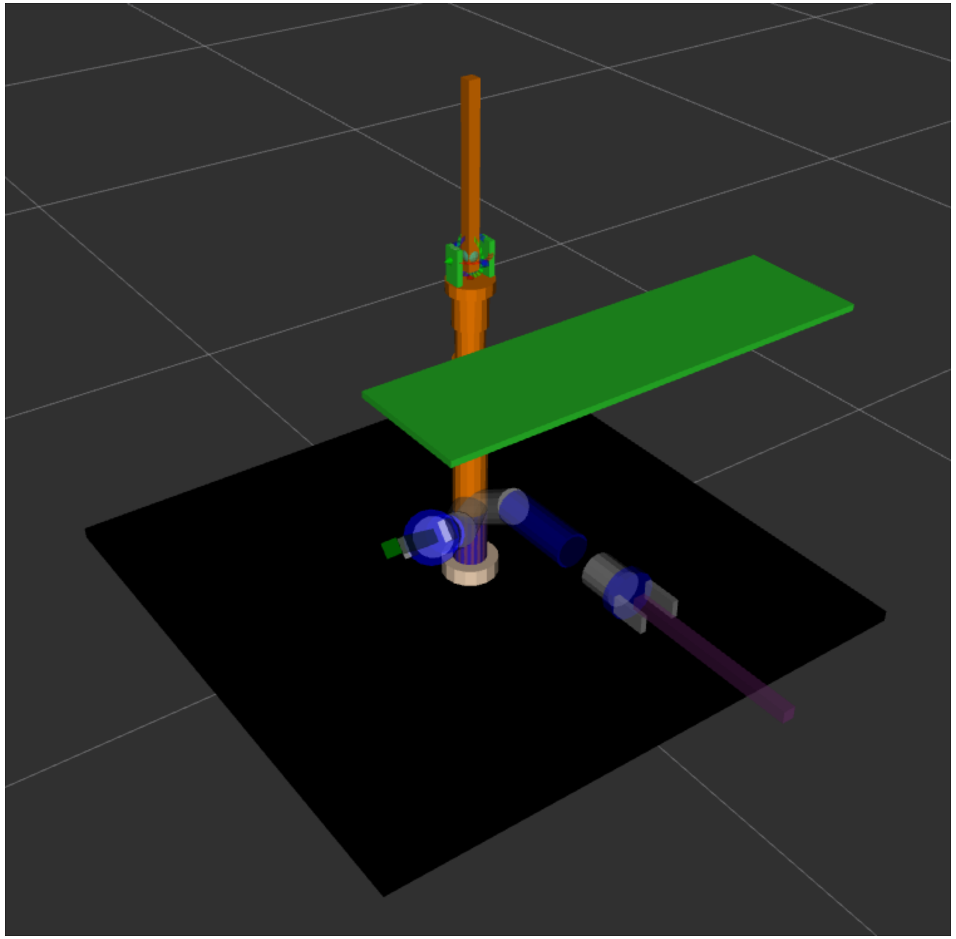




通过MoveIt!可视化插件添加模型，在运动规划时会考虑碰撞检测

4. 碰撞检测 —— 附着物体





附着物体避障例程

$ roslaunch marm_moveit_config demo.launch

$ rosrun marm_planning moveit_attached_object_demo.py

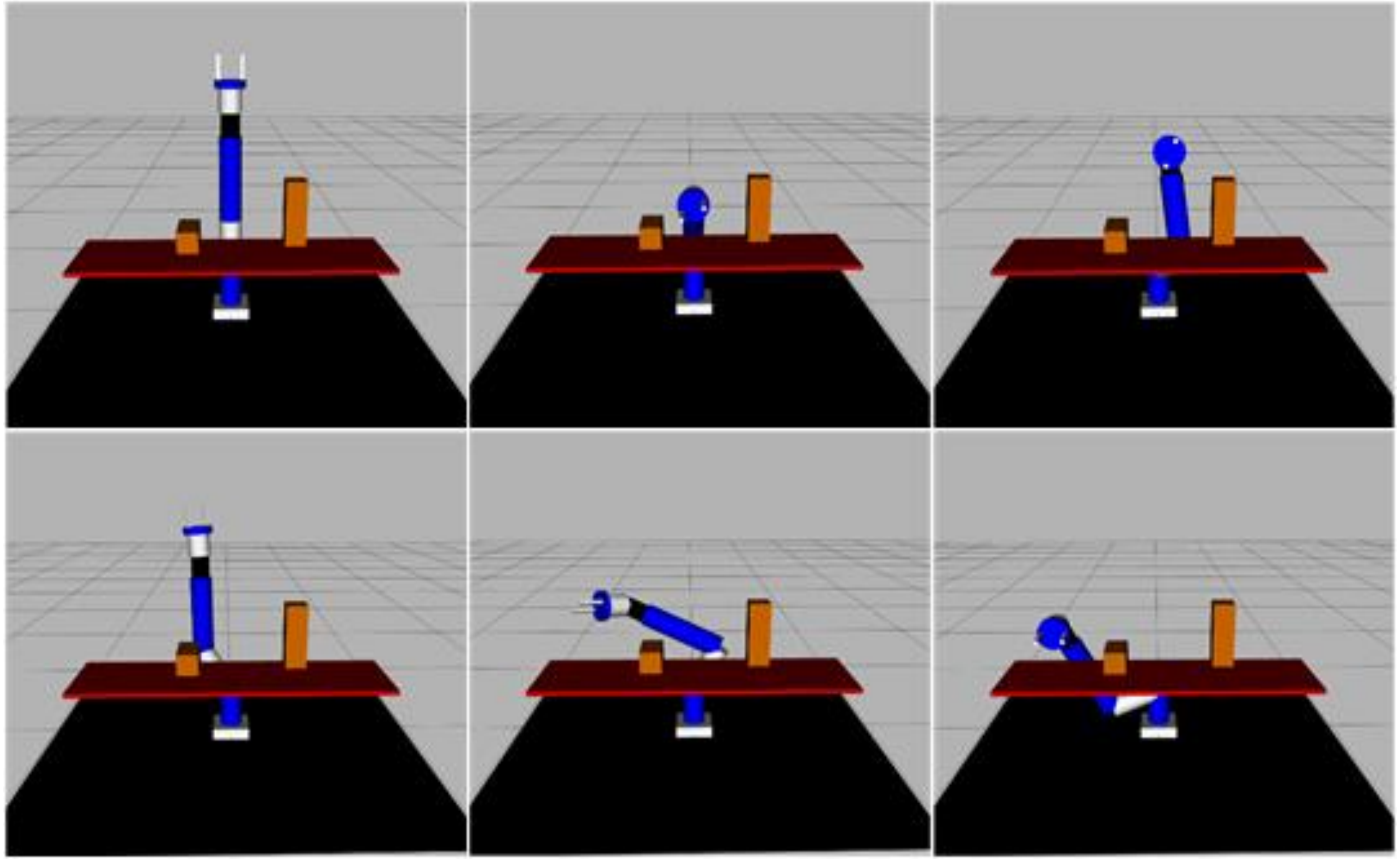
4. 碰撞检测 —— 附着物体

```python
# 移除场景中之前运行残留的物体
scene.remove_attached_object(end_effector_link, 'tool')
scene.remove_world_object('table')
scene.remove_world_object('target')

# 设置桌面的高度
table_ground = 0.6

# 设置table和tool的三维尺寸
table_size = [0.2, 0.7, 0.01]
tool_size = [0.3, 0.02, 0.02]

# 设置tool的位姿
p = PoseStamped()
p.header.frame_id = end_effector_link

p.pose.position.x = tool_size[0] / 2.0 - 0.025
p.pose.position.y = 0.0
p.pose.position.z = 0.0
p.pose.orientation.x = 0
p.pose.orientation.y = 0
p.pose.orientation.z = 0
p.pose.orientation.w = 1

# 将tool附着到机器人的终端
scene.attach_box(end_effector_link, 'tool', p, tool_size)
```

```python
# 将table加入场景当中
table_pose = PoseStamped()
table_pose.header.frame_id = 'base_link'
table_pose.pose.position.x = 0.35
table_pose.pose.position.y = 0.0
table_pose.pose.position.z = table_ground + table_size[2] / 2.0
table_pose.pose.orientation.w = 1.0
scene.add_box('table', table_pose, table_size)

rospy.sleep(2)

# 更新当前的位姿
arm.set_start_state_to_current_state()

# 控制机械臂运动到forward位姿
arm.set_named_target('forward')
arm.go()
rospy.sleep(2)

# 控制机械臂回到初始化位姿
arm.set_named_target('home')
arm.go()
rospy.sleep(2)

scene.remove_attached_object(end_effector_link, 'tool')
```

marm_planning/scripts/moveit_attached_object_demo.py

4. 碰撞检测 —— 避障规划



自主避障规划例程

$ roslaunch marm_moveit_config demo.launch

$ rosrun marm_planning moveit_obstacles_demo.py

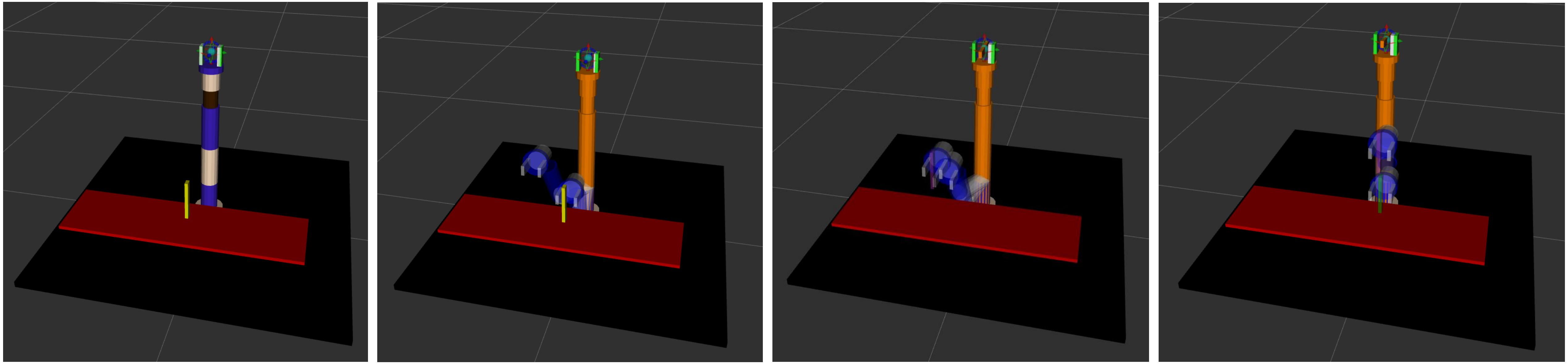
4. 碰撞检测 —— 避障规划

```python
# 初始化场景对象
scene = PlanningSceneInterface()

# 创建一个发布场景变化信息的发布者
self.scene_pub = rospy.Publisher('planning_scene', PlanningScene, queue_size=5)

# 创建一个存储物体颜色的字典对象
self.colors = dict()

# 设置桌面的高度
table_ground = 0.25

# 设置table、box1和box2的三维尺寸
table_size = [0.2, 0.7, 0.01]
box1_size = [0.1, 0.05, 0.05]
box2_size = [0.05, 0.05, 0.15]

# 将三个物体加入场景当中
table_pose = PoseStamped()
table_pose.header.frame_id = reference_frame
table_pose.pose.position.x = 0.26
table_pose.pose.position.y = 0.0
table_pose.pose.position.z = table_ground + table_size[2] / 2.0
table_pose.pose.orientation.w = 1.0
scene.add_box(table_id, table_pose, table_size)

box1_pose = PoseStamped()
box1_pose.header.frame_id = reference_frame
box1_pose.pose.position.x = 0.21
box1_pose.pose.position.y = -0.1
box1_pose.pose.position.z = table_ground + table_size[2] + box1_size[2] / 2.0
box1_pose.pose.orientation.w = 1.0
scene.add_box(box1_id, box1_pose, box1_size)

box2_pose = PoseStamped()
box2_pose.header.frame_id = reference_frame
box2_pose.pose.position.x = 0.19
box2_pose.pose.position.y = 0.15
box2_pose.pose.position.z = table_ground + table_size[2] + box2_size[2] / 2.0
box2_pose.pose.orientation.w = 1.0
scene.add_box(box2_id, box2_pose, box2_size)
```

```python
# 设置场景物体的颜色
def setColor(self, name, r, g, b, a = 0.9):
    # 初始化moveit颜色对象
    color = ObjectColor()

    # 设置颜色值
    color.id = name
    color.color.r = r
    color.color.g = g
    color.color.b = b
    color.color.a = a

    # 更新颜色字典
    self.colors[name] = color

# 将颜色设置发送并应用到moveit场景当中
def sendColors(self):
    # 初始化规划场景对象
    p = PlanningScene()

    # 需要设置规划场景是否有差异
    p.is_diff = True

    # 从颜色字典中取出颜色设置
    for color in self.colors.values():
        p.object_colors.append(color)

    # 发布场景物体颜色设置
    self.scene_pub.publish(p)
```

marm_planning/scripts/moveit_obstacles_demo.py

4. 碰撞检测 —— Pick&Place实例演示与分析



Pick&Place
例程

$ roslaunch marm_moveit_config demo.launch

$ rosrun marm_planning moveit_pick_and_place_demo.py

4. 碰撞检测 —— Pick&Place实例演示与分析

创建抓取的目标物体

```
# 将桌子设置成红色，两个box设置成橙色
self.setColor(table_id, 0.8, 0, 0, 1.0)

# 设置目标物体的尺寸
target_size = [0.02, 0.01, 0.12]

# 设置目标物体的位置，位于桌面之上两个盒子之间
target_pose = PoseStamped()
target_pose.header.frame_id = REFERENCE_FRAME
target_pose.pose.position.x = 0.32
target_pose.pose.position.y = 0.0
target_pose.pose.position.z = table_ground + table_size[2] + target_size[2] / 2.0
target_pose.pose.orientation.w = 1.0

# 将抓取的目标物体加入场景中
scene.add_box(target_id, target_pose, target_size)
```

4. 碰撞检测 —— Pick&Place实例演示与分析

设置目标物体的放置位置

```
# 设置一个place阶段需要放置物体的目标位置
place_pose = PoseStamped()
place_pose.header.frame_id = REFERENCE_FRAME
place_pose.pose.position.x = 0.32
place_pose.pose.position.y = 0.05
place_pose.pose.position.z = table_ground + table_size[2] + target_size[2] / 2.0
place_pose.pose.orientation.w = 1.0
```

4. 碰撞检测 —— Pick&Place实例演示与分析

生成抓取姿态

```python
# 将目标位置设置为机器人的抓取目标位置
grasp_pose = target_pose

# 生成抓取姿态
grasps = self.make_grasps(grasp_pose, [target_id])

# 将抓取姿态发布，可以在rviz中显示
for grasp in grasps:
    self.gripper_pose_pub.publish(grasp.grasp_pose)
    rospy.sleep(0.2)
```

```python
# 改变姿态，生成抓取动作
for y in yaw_vals:
    for p in pitch_vals:
        # 欧拉角到四元数的转换
        q = quaternion_from_euler(0, p, y)

        # 设置抓取的姿态
        g.grasp_pose.pose.orientation.x = q[0]
        g.grasp_pose.pose.orientation.y = q[1]
        g.grasp_pose.pose.orientation.z = q[2]
        g.grasp_pose.pose.orientation.w = q[3]

        # 设置抓取的唯一id号
        g.id = str(len(grasps))

        # 设置允许接触的物体
        g.allowed_touch_objects = allowed_touch_objects

        # 将本次规划的抓取放入抓取列表中
        grasps.append(deepcopy(g))
```

4. 碰撞检测 —— Pick&Place实例演示与分析

Pick

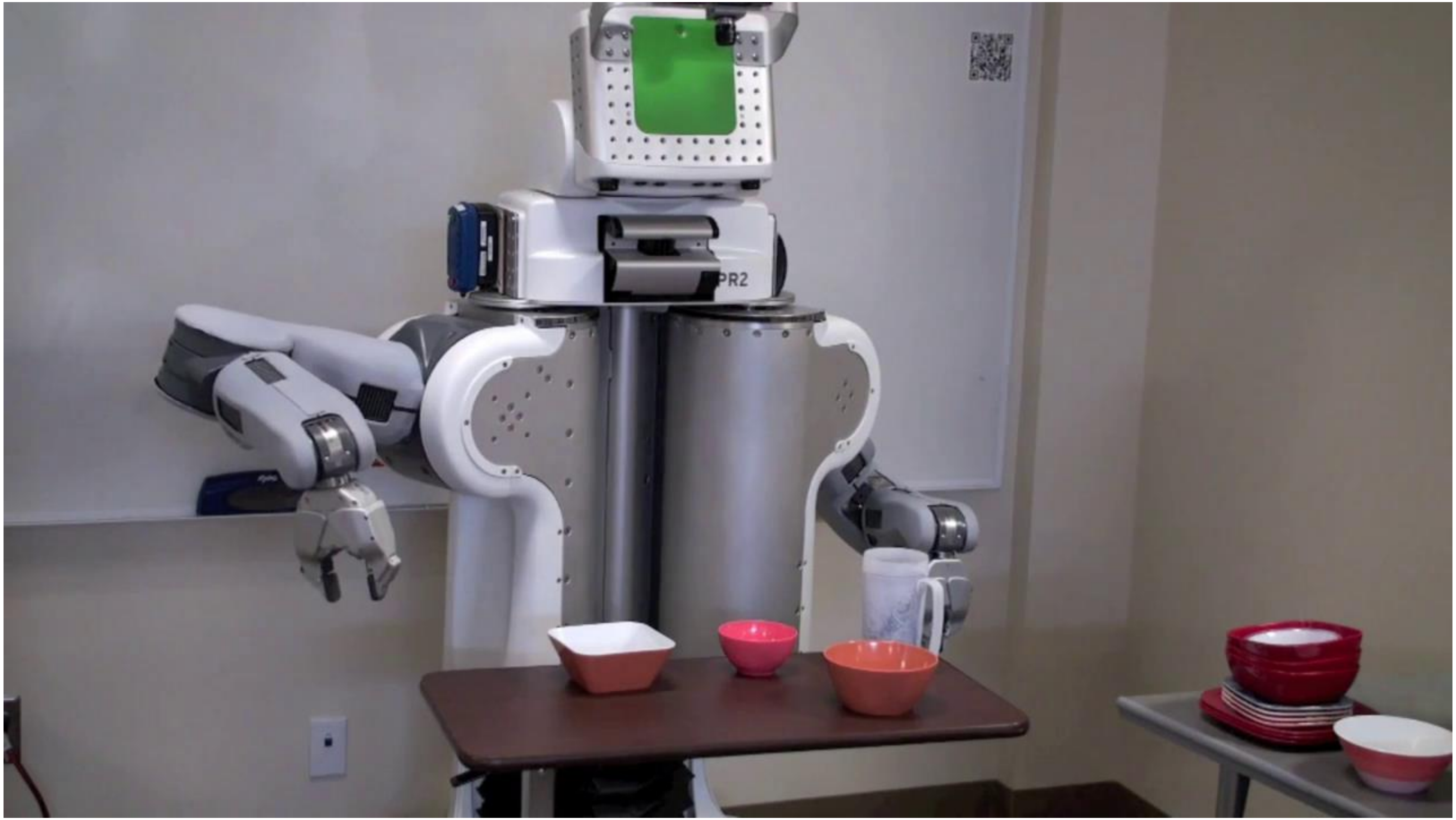
```python
# 追踪抓取成功与否，以及抓取的尝试次数
result = None
n_attempts = 0

# 重复尝试抓取，直道成功或者超多最大尝试次数
while result != MoveItErrorCodes.SUCCESS and n_attempts < max_pick_attempts:
    n_attempts += 1
    rospy.loginfo("Pick attempt: " +  str(n_attempts))
    result = arm.pick(target_id, grasps)
    rospy.sleep(0.2)
```

4. 碰撞检测 —— Pick&Place实例演示与分析

Place

```
# 如果pick成功，则进入place阶段
if result == MoveItErrorCodes.SUCCESS:
    result = None
    n_attempts = 0

    # 生成放置姿态
    places = self.make_places(place_pose)

    # 重复尝试放置，直道成功或者超多最大尝试次数
    while result != MoveItErrorCodes.SUCCESS and n_attempts < max_place_attempts:
        n_attempts += 1
        rospy.loginfo("Place attempt: " +  str(n_attempts))
        for place in places:
            result = arm.place(target_id, place)
            if result == MoveItErrorCodes.SUCCESS:
                break
        rospy.sleep(0.2)

    if result != MoveItErrorCodes.SUCCESS:
        rospy.loginfo("Place operation failed after " + str(n_attempts) + " attempts.")
else:
    rospy.loginfo("Pick operation failed after " + str(n_attempts) + " attempts.")
```

4. 碰撞检测 —— Pick&Place实例演示与分析



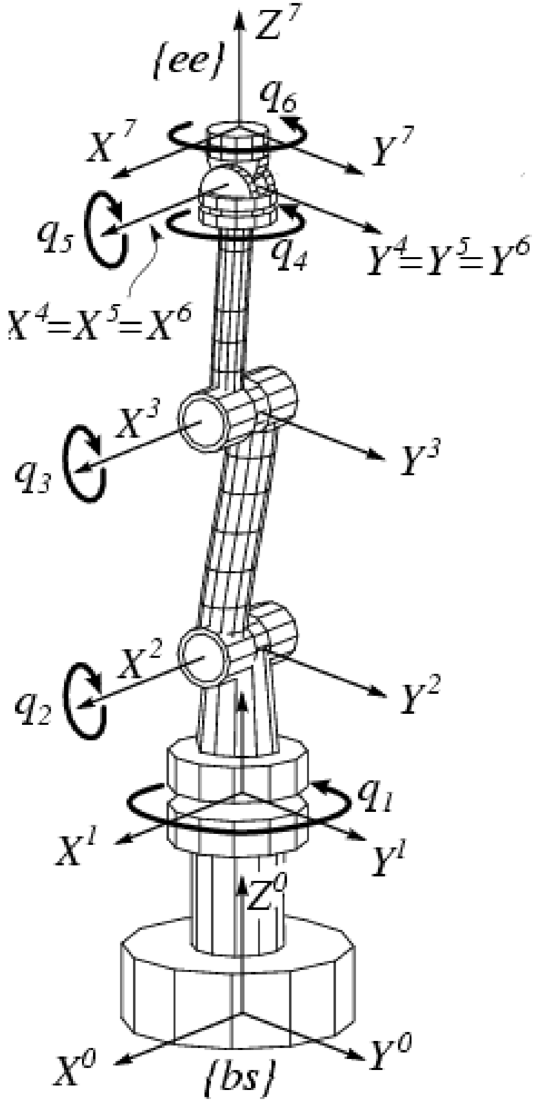
目录

- 5.运动学插件的配置

5. 运动学插件的配置 —— KDL

Orocos Kinematics and Dynamics

Smarter in motion!



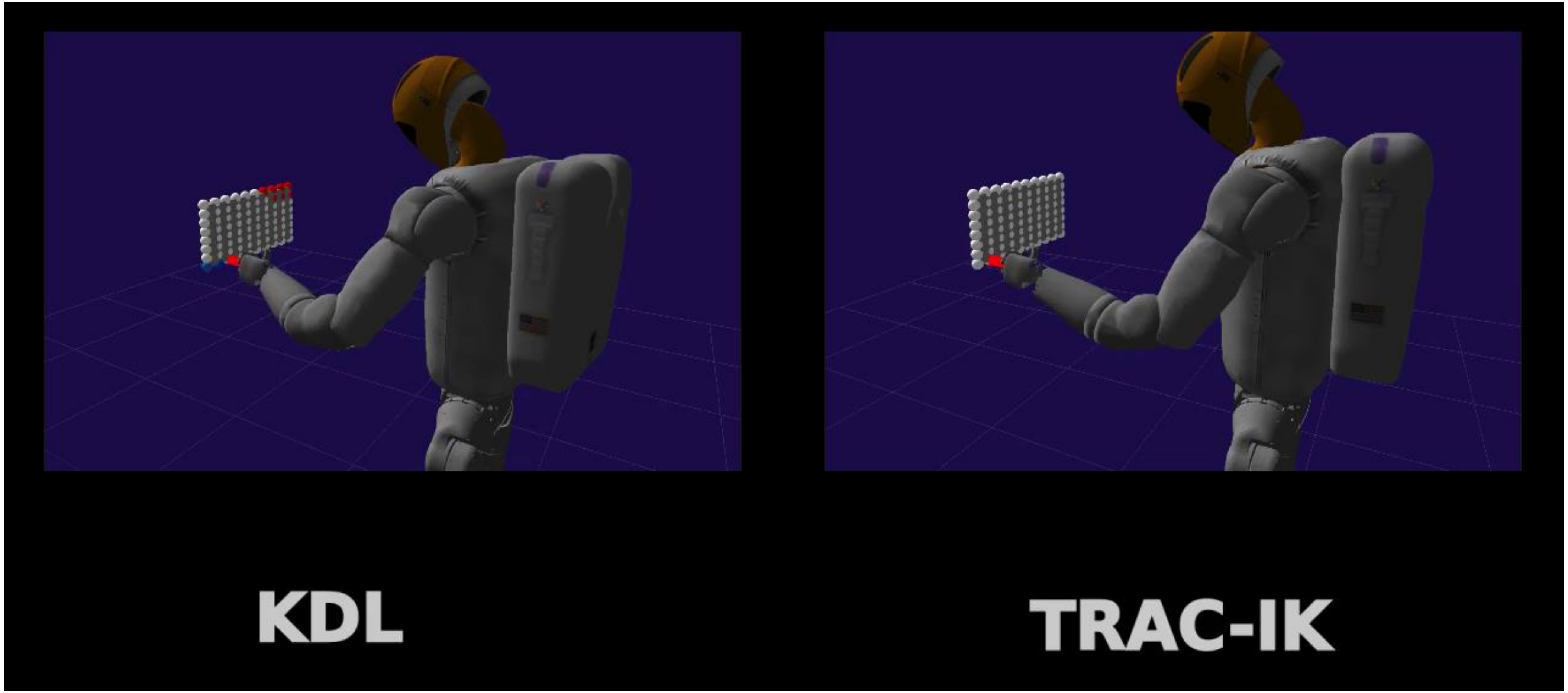
MoveIt!默认使用的运动学求解器

- 数值解
- 优点：可求解封闭情况下逆运动学
- 缺点：速度慢、可能找不到解

* 参考链接：http://wiki.ros.org/kdl

5. 运动学插件的配置 —— TRAC-IK

traclabs



* 参考链接：http://docs.ros.org/kinetic/api/moveit_tutorials/html/doc/trac_ik/trac_ik_tutorial.html

5. 运动学插件的配置 —— MoveIt! TRAC-IK配置方法

安装　$ sudo apt-get install ros-kinetic-trac-ik-kinematics-plugin

配置　$ rosed "$MYROBOT_NAME"_moveit_config/config/kinematics.yaml

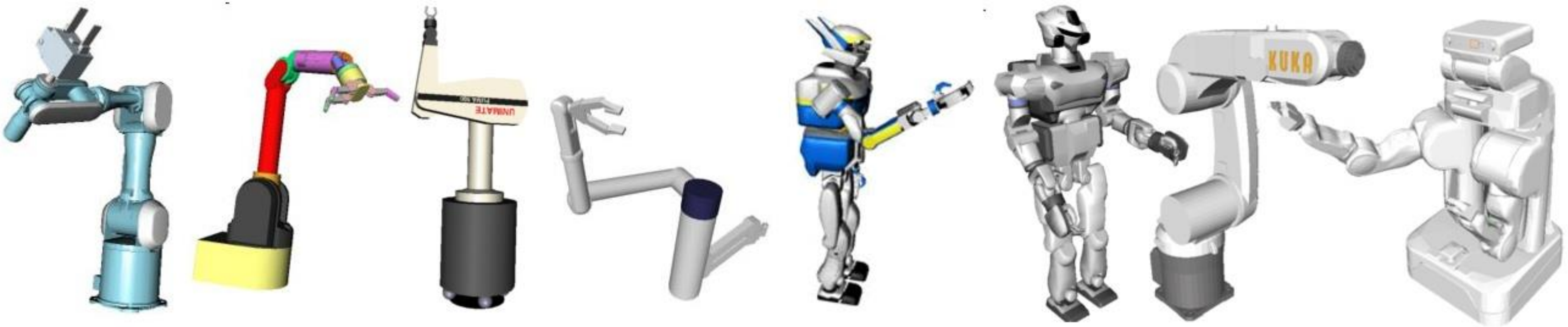
```yaml
arm:
  kinematics_solver: trac_ik_kinematics_plugin/TRAC_IKKinematicsPlugin
  kinematics_solver_attempts: 3
  kinematics_solver_search_resolution: 0.005
  kinematics_solver_timeout: 0.05
```

测试　$ sudo marm_moveit_config demo.launch

* 参考链接：http://docs.ros.org/kinetic/api/moveit_tutorials/html/doc/trac_ik/trac_ik_tutorial.html

5. 运动学插件的配置 —— IKFast

- IKFast，由Rosen Diankov编写的OpenRAVE运动规划软件提供；
- 可以求解任意复杂运动链的运动学方程（解析解），并产生特定语言的文件（如C++）后供使用；
- 比较稳定、速度快，在最新的处理器上能以5微秒的速度完成运算。



* 参考链接：http://openrave.org/docs/0.8.2/openravepy/ikfast/

5. 运动学插件的配置 —— MoveIt! IKFast配置方法

安装依赖程序和库

$ sudo apt-get install cmake g++ git ipython minizip python-dev python-h5py python-numpy python-scipy qt4-dev-tools

$ sudo apt-get install libassimp-dev libavcodec-dev libavformat-dev libavformat-dev libboost-all-dev libboost-date-time-dev libbullet-dev libfaac-dev libglew-dev libgsm1-dev liblapack-dev liblog4cxx-dev libmpfr-dev libode-dev libogg-dev libpcrecppov5 libpcre3-dev libqhull-dev libqt4-dev libsoqt-dev-common libsoqt4-dev libswscale-dev libswscale-dev libvorbis-dev libx264-dev libxml2-dev libxvidcore-dev

安装 OpenSceneGraph

$ sudo apt-get install libcairo2-dev libjasper-dev libpoppler-glib-dev libsdl2-dev libtiff5-dev libxrandr-dev

$ git clone https://github.com/openscenegraph/OpenSceneGraph.git --branch OpenSceneGraph-3.4

$ cd OpenSceneGraph

$ mkdir build; cd build

$ cmake .. -DDESIRED_QT_VERSION=4

$ make -j$(nproc)

$ sudo make install

* 参考链接：http://docs.ros.org/kinetic/api/moveit_tutorials/html/doc/ikfast/ikfast_tutorial.html

5. 运动学插件的配置 —— MoveIt! IKFast配置方法

安装Python工具

$ pip install --upgrade --user sympy==0.7.1

$ sudo apt remove python-mpmath

安装IKFast和OpenRave功能包

$ sudo apt-get install ros-kinetic-moveit-kinematics

$ sudo apt-get install ros-kinetic-openrave

创建collada文件

$ export MYROBOT_NAME="marm"

$ rosrun xacro xacro --inorder -o "$MYROBOT_NAME".urdf "$MYROBOT_NAME".xacro

$ rosrun collada_urdf urdf_to_collada "$MYROBOT_NAME".urdf "$MYROBOT_NAME".dae

创建dae文件

$ export IKFAST_PRECISION="5"

$ cp "$MYROBOT_NAME".dae "$MYROBOT_NAME".backup.dae

$ rosrun moveit_kinematics round_collada_numbers.py "$MYROBOT_NAME".dae "$MYROBOT_NAME".dae "$IKFAST_PRECISION"

* 参考链接：http://docs.ros.org/kinetic/api/moveit_tutorials/html/doc/ikfast/ikfast_tutorial.html

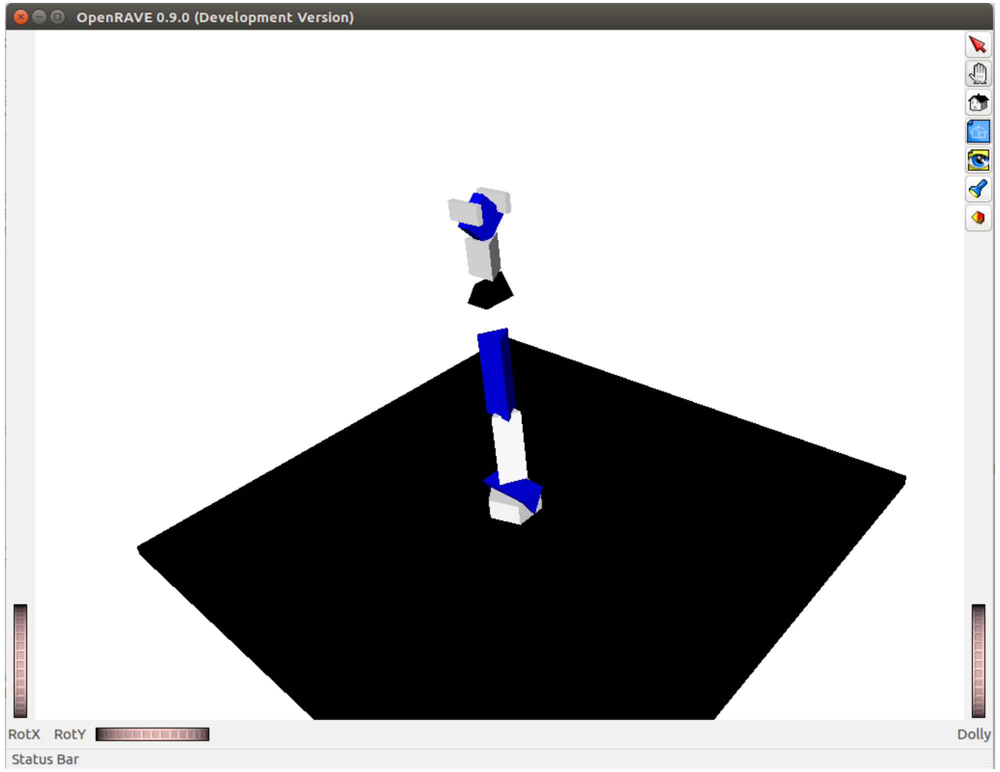
5. 运动学插件的配置 —— MoveIt! IKFast配置方法

查看生成的模型

$ openrave-robot.py "$MYROBOT_NAME".dae --info links

$ openrave "$MYROBOT_NAME".dae

```
→ urdf openrave-robot.py "$MYROBOT_NAME".dae --info links
name                    index parents
-------------------------------------
bottom_link             0
base_link               1     bottom_link
link0                   2     base_link
link1                   3     link0
link2                   4     link1
link3                   5     link2
link4                   6     link3
link5                   7     link4
link6                   8     link5
gripper_finger_link1    9     link6
gripper_finger_link2    10    link6
grasping_frame          11    link6
-------------------------------------
name                    index parents
```



* 参考链接：http://docs.ros.org/kinetic/api/moveit_tutorials/html/doc/ikfast/ikfast_tutorial.html

5. 运动学插件的配置 —— MoveIt! IKFast配置方法

生成程序文件

```
$ export PLANNING_GROUP="arm"
$ export BASE_LINK="1"
$ export EEF_LINK="11"
$ export IKFAST_OUTPUT_PATH=`pwd`/ikfast61_"$PLANNING_GROUP".cpp
$ python `openrave-config --python-dir`/openravepy/_openravepy_/ikfast.py --robot="$MYROBOT_NAME".dae --
iktype=transform6d --baselink="$BASE_LINK" --eelink="$EEF_LINK" --savefile="$IKFAST_OUTPUT_PATH"
```

创建插件

```
$ export MOVEIT_IK_PLUGIN_PKG="$MYROBOT_NAME"_ikfast_"$PLANNING_GROUP"_plugin
$ cd ~/catkin_ws/src
$ catkin_create_pkg "$MOVEIT_IK_PLUGIN_PKG"
$ rosrun moveit_kinematics create_ikfast_moveit_plugin.py "$MYROBOT_NAME" "$PLANNING_GROUP"
"$MOVEIT_IK_PLUGIN_PKG" "$IKFAST_OUTPUT_PATH"
```

* 参考链接：http://docs.ros.org/kinetic/api/moveit_tutorials/html/doc/ikfast/ikfast_tutorial.html

5. 运动学插件的配置 —— MoveIt! IKFast配置方法

修MoveIt!配置文件

$ catkin_make（工作空间根路径下）

$ rosed "$MYROBOT_NAME"_moveit_config/config/kinematics.yaml

```
arm:
  kinematics_solver: marm_arm_kinematics/IKFastKinematicsPlugin
  kinematics_solver_attempts: 3
  kinematics_solver_search_resolution: 0.005
  kinematics_solver_timeout: 0.05
```

测试IKFast插件 $ sudo marm_moveit_config demo.launch

* 模型发生变化后，IKFast插件也要重新生成

* 参考链接：http://docs.ros.org/kinetic/api/moveit_tutorials/html/doc/ikfast/ikfast_tutorial.html

Thank you!