ROS机械臂开发：从入门到实战

—— 第7讲：MoveIt!中不得不说的“潜规则”

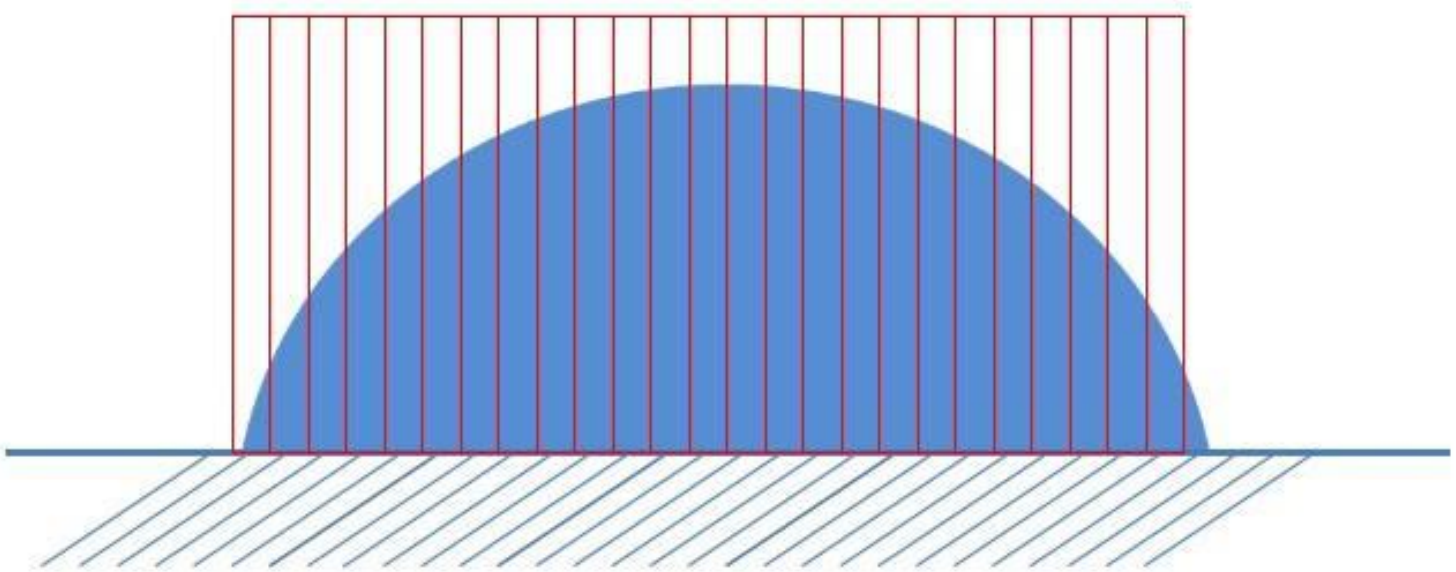
主讲人 胡春旭

机器人博客“古月居”博主
《ROS机器人开发实践》作者
武汉精锋微控科技有限公司 联合创始人
华中科技大学 自动化学院 硕士

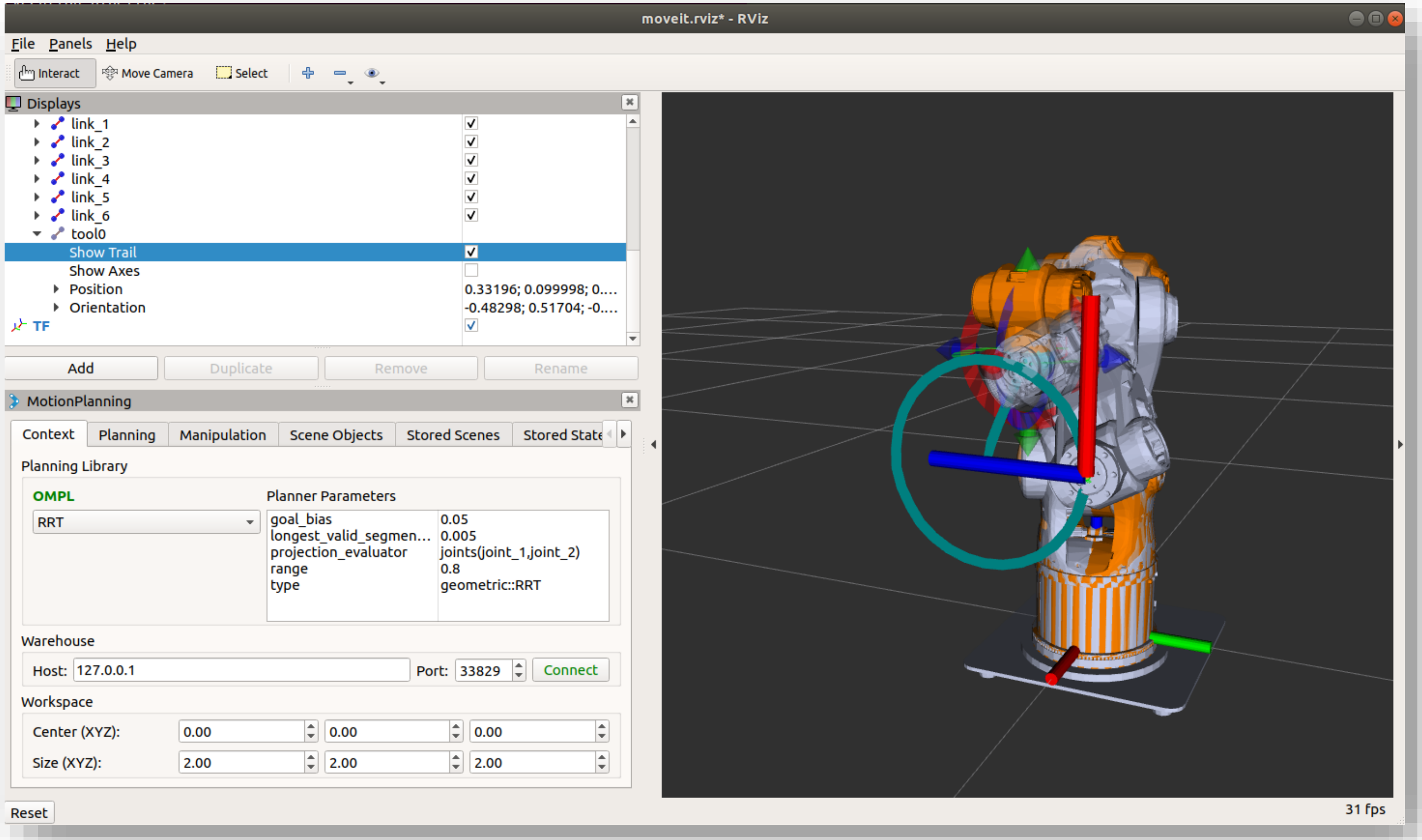
目录

1. 圆弧轨迹规划

如何走出笛卡尔空间下的圆弧轨迹?



1. 圆弧轨迹规划



圆弧轨迹规划例程

$ roslaunch probot_anno_moveit_config demo.launch

$ rosrun probot_demo moveit_circle_demo

1. 圆弧轨迹规划

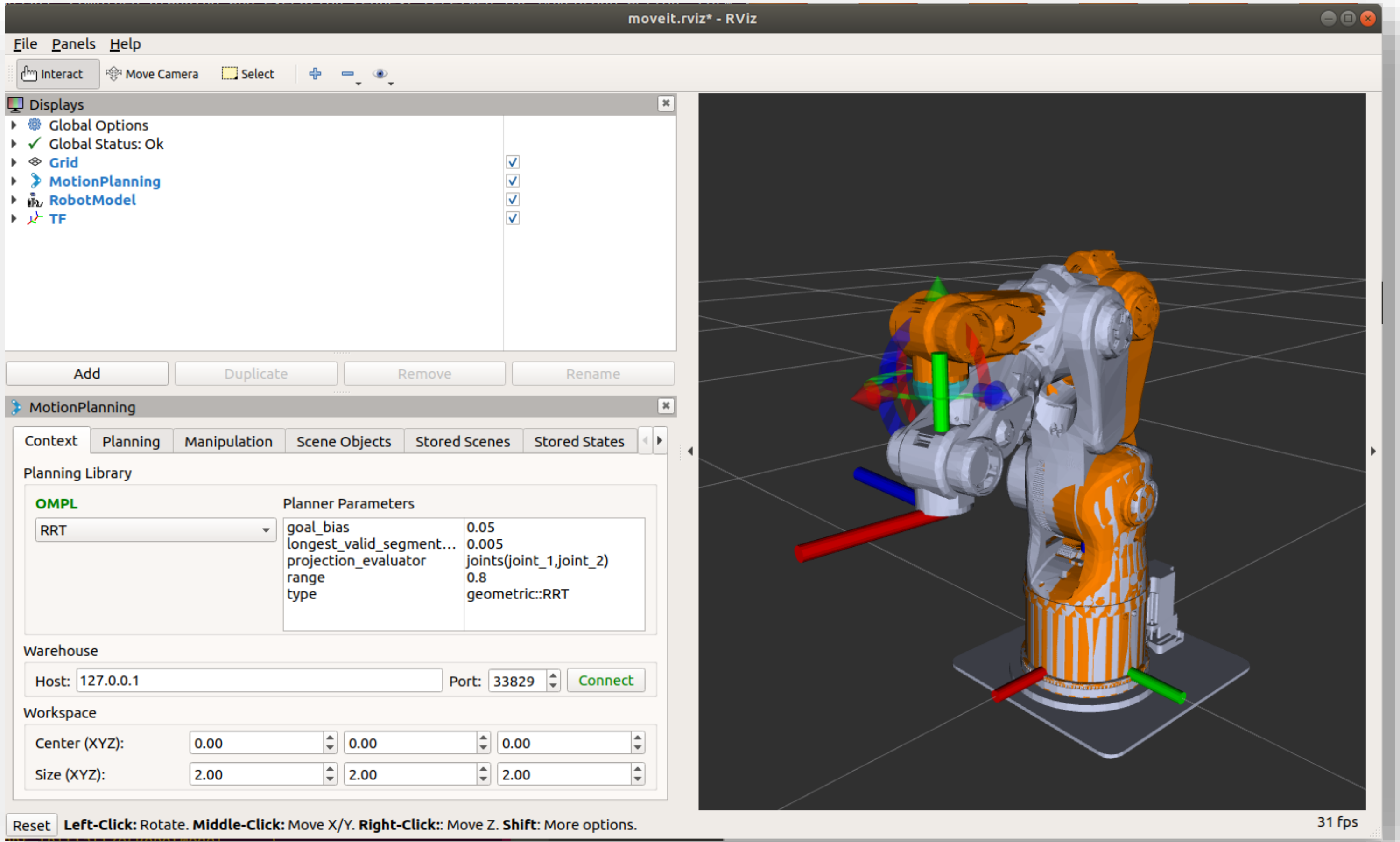
```
std::vector<geometry_msgs::Pose> waypoints;

//将初始位姿加入路点列表
waypoints.push_back(target_pose);

double centerA = target_pose.position.y;
double centerB = target_pose.position.z;
double radius = 0.1;

for(double th=0.0; th<6.28; th=th+0.01)
{
    target_pose.position.y = centerA + radius * cos(th);
    target_pose.position.z = centerB + radius * sin(th);
    waypoints.push_back(target_pose);
}

// 笛卡尔空间下的路径规划
moveit_msgs::RobotTrajectory trajectory;
const double jump_threshold = 0.0;
const double eef_step = 0.01;
double fraction = 0.0;
int maxtries = 100;   //最大尝试规划次数
int attempts = 0;     //已经尝试规划次数

while(fraction < 1.0 && attempts < maxtries)
{
    fraction = arm.computeCartesianPath(waypoints, eef_step, jump_threshold, trajectory);
    attempts++;

    if(attempts % 10 == 0)
        ROS_INFO("Still trying after %d attempts...", attempts);
}
```

计算圆弧轨迹

规划笛卡尔路径

2. 轨迹重定义

2. 轨迹重定义



轨迹重定义例程

$ roslaunch probot_anno_moveit_config demo.launch

$ rosrun probot_demo moveit_revise_trajectory_demo

2. 轨迹重定义

```
arm.setJointValueTarget(joint_group_positions);
moveit::planning_interface::MoveGroupInterface::Plan plan;
moveit::planning_interface::MoveItErrorCode success = arm.plan(plan);

ROS_INFO("Plan (pose goal) %s",success?"":"FAILED");
```

轨迹规划

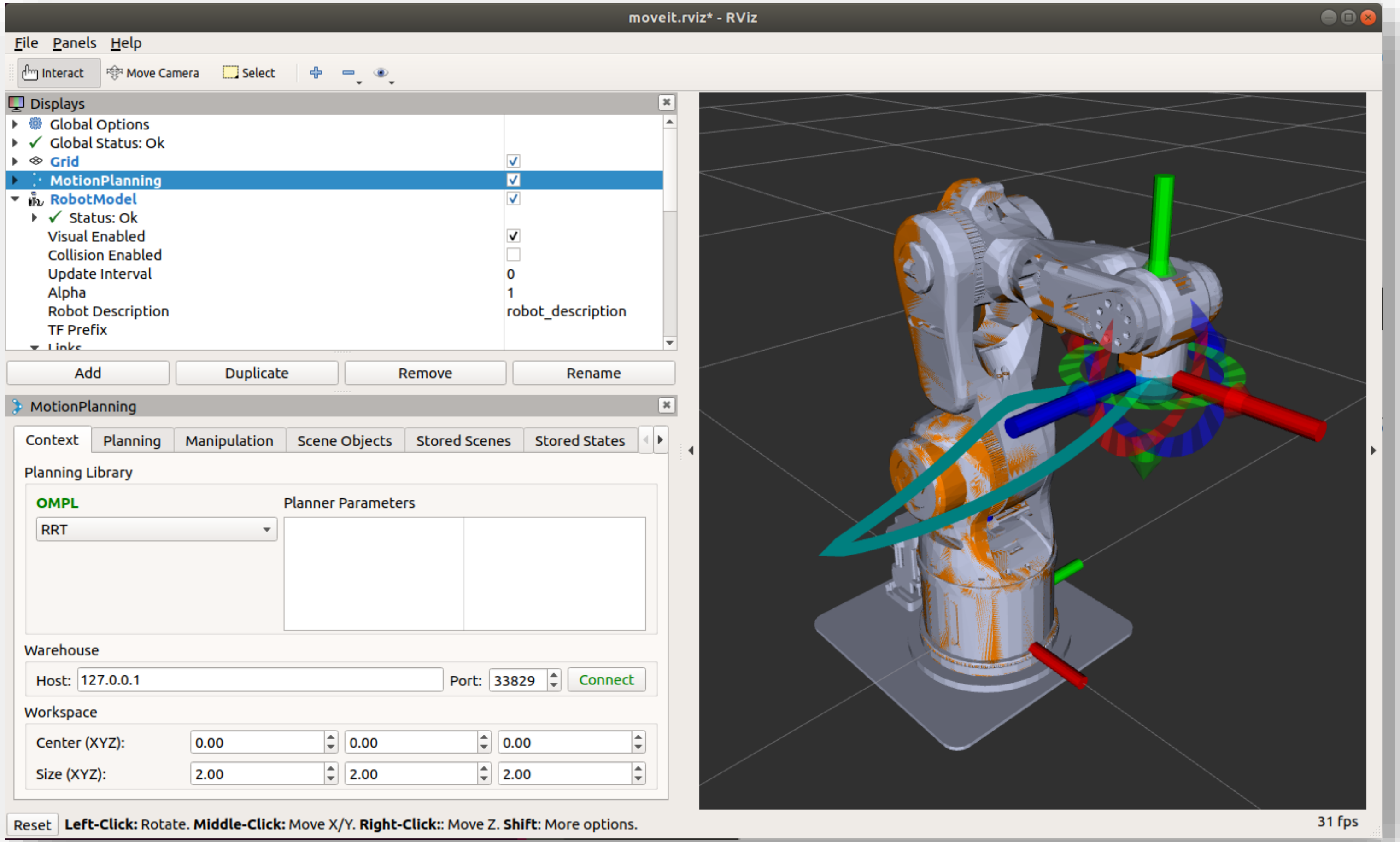
```
scale_trajectory_speed(plan, 0.25);

//让机械臂按照规划的轨迹开始运动。
if(success)
    arm.execute(plan);
sleep(1);
```

```
void scale_trajectory_speed(moveit::planning_interface::MoveGroupInterface::Plan &plan, double scale)
{
    int n_joints = plan.trajectory_.joint_trajectory.joint_names.size();

    for(int i=0; i<plan.trajectory_.joint_trajectory.points.size(); i++)
    {
        plan.trajectory_.joint_trajectory.points[i].time_from_start *= 1/scale;

        for(int j=0; j<n_joints; j++)
        {
            plan.trajectory_.joint_trajectory.points[i].velocities[j] *= scale;
            plan.trajectory_.joint_trajectory.points[i].accelerations[j] *= scale*scale;
        }
    }
}
```

轨迹重定义

3. 多轨迹连续运动

3. 多轨迹连续运动



多轨迹连续运动例程

$ roslaunch probot_anno_moveit_config demo.launch

$ rosrun probot_demo moveit_continue_trajectory_demo

3. 多轨迹连续运动

```
// 获取机器人的起始位置
moveit::core::RobotStatePtr start_state(arm.getCurrentState());
const robot_state::JointModelGroup *joint_model_group = start_state->getJointModelGroup(arm.getName());

std::vector<double> joint_group_positions;
start_state->copyJointGroupPositions(joint_model_group, joint_group_positions);
```

```
//设置第一个目标点
joint_group_positions[0] = -0.6;  // radians
arm.setJointValueTarget(joint_group_positions);

// 计算第一条轨迹
moveit::planning_interface::MoveGroupInterface::Plan plan1;
moveit::planning_interface::MoveItErrorCode success = arm.plan(plan1);

joint_model_group = start_state->getJointModelGroup(arm.getName());
start_state->setJointGroupPositions(joint_model_group, joint_group_positions);
arm.setStartState(*start_state);
```

规划轨迹1

```
//设置第二个目标点
joint_group_positions[0] = -1.2;  // radians
joint_group_positions[1] = -0.5;  // radians
arm.setJointValueTarget(joint_group_positions);

// 计算第二条轨迹
moveit::planning_interface::MoveGroupInterface::Plan plan2;
success = arm.plan(plan2);

joint_model_group = start_state->getJointModelGroup(arm.getName());
start_state->setJointGroupPositions(joint_model_group, joint_group_positions);
arm.setStartState(*start_state);
```

规划轨迹2

3. 多轨迹连续运动

```
//连接两条轨迹
moveit_msgs::RobotTrajectory trajectory;

trajectory.joint_trajectory.joint_names = plan1.trajectory_.joint_trajectory.joint_names;
trajectory.joint_trajectory.points = plan1.trajectory_.joint_trajectory.points;
for (size_t j = 1; j < plan2.trajectory_.joint_trajectory.points.size(); j++)
{
    trajectory.joint_trajectory.points.push_back(plan2.trajectory_.joint_trajectory.points[j]);
}

moveit::planning_interface::MoveGroupInterface::Plan joinedPlan;
robot_trajectory::RobotTrajectory rt(arm.getCurrentState()->getRobotModel(), "manipulator");
rt.setRobotTrajectoryMsg(*arm.getCurrentState(), trajectory);
trajectory_processing::IterativeParabolicTimeParameterization iptp;
iptp.computeTimeStamps(rt, velScale, accScale);

rt.getRobotTrajectoryMsg(trajectory);
joinedPlan.trajectory_ = trajectory;

if (!arm.execute(joinedPlan))
{
    ROS_ERROR("Failed to execute plan");
    return false;
}
```

轨迹1 + 轨迹2

重规划

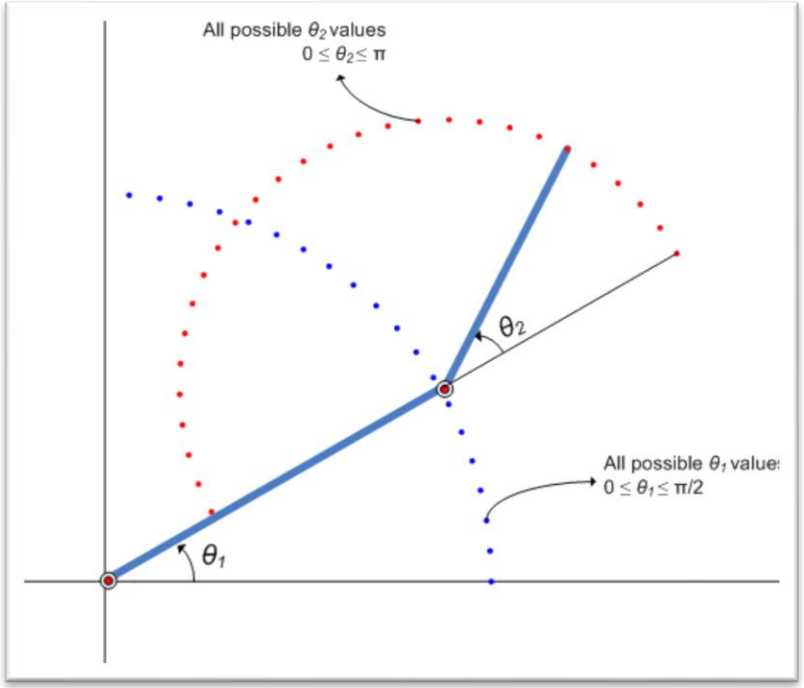
4. 更换运动学插件

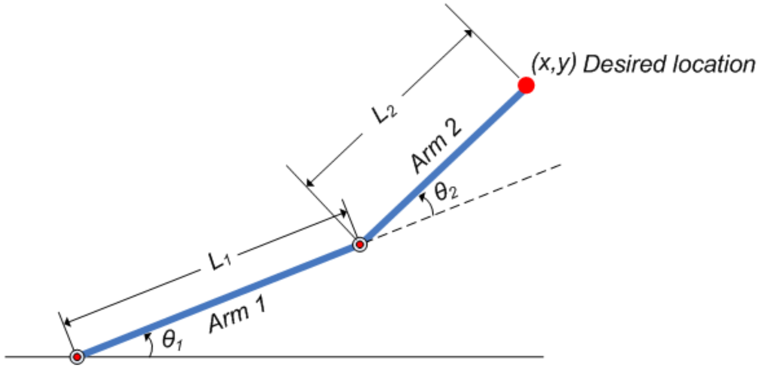
4. 更换运动学插件

什么是运动学?

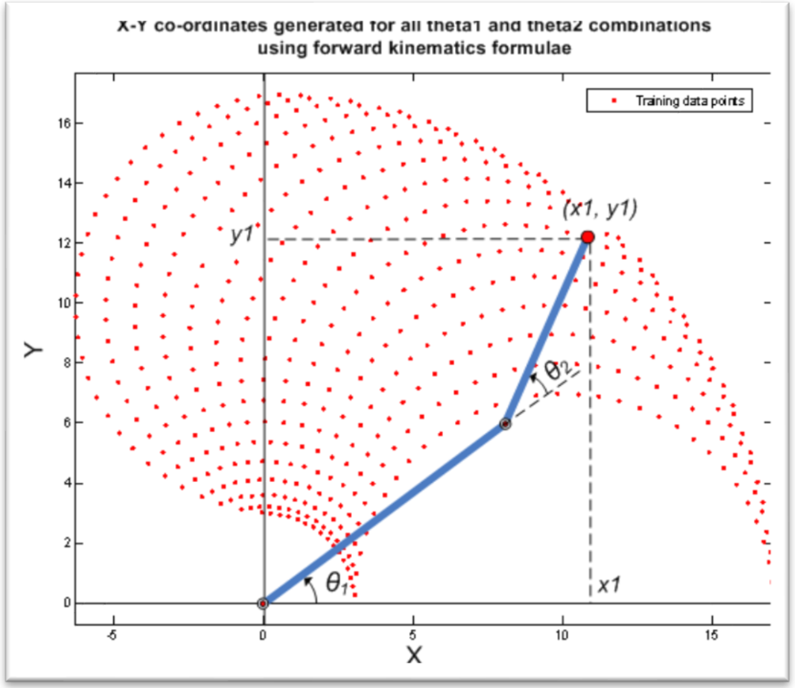
```
l1 = 10; % length of first arm
l2 = 7; % length of second arm

theta1 = 0:0.1:pi/2; % all possible theta1 values
theta2 = 0:0.1:pi; % all possible theta2 values

[THETA1,THETA2] = meshgrid(theta1,theta2); % generate a grid of theta1 and theta2 values

X = l1 * cos(THETA1) + l2 * cos(THETA1 + THETA2); % compute x coordinates
Y = l1 * sin(THETA1) + l2 * sin(THETA1 + THETA2); % compute y coordinates
```

```
[X,Y] = meshgrid(x,y);

c2 = (X.^2 + Y.^2 - l1^2 - l2^2)/(2*l1*l2);
s2 = sqrt(1 - c2.^2);
THETA2D = atan2(s2,c2); % theta2 is deduced

k1 = l1 + l2.*c2;
k2 = l2*s2;
THETA1D = atan2(Y,X) - atan2(k2,k1); % theta1 is deduced
```





* 理论知识请参考：《机器人学导论》

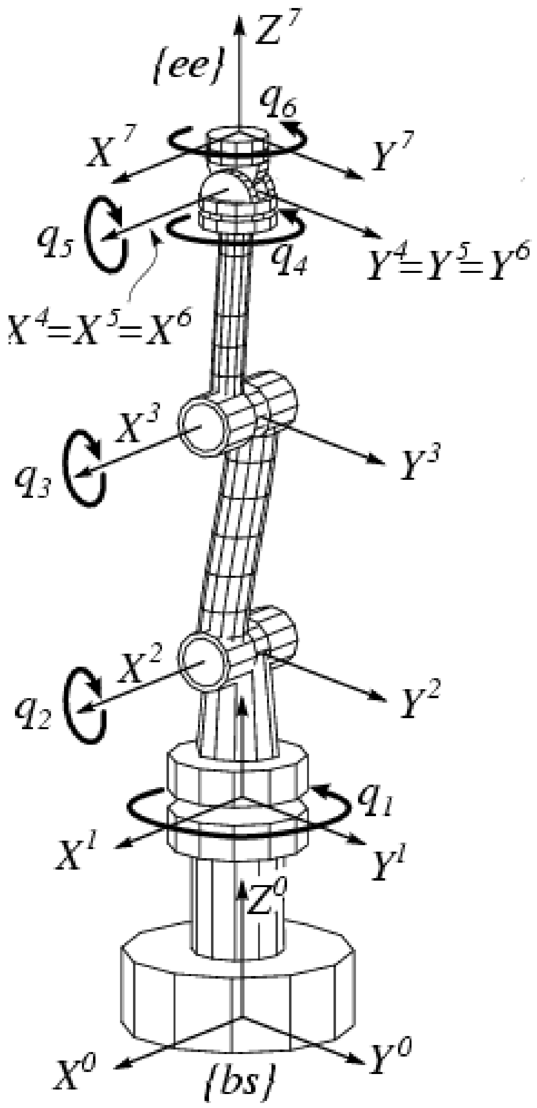


4. 更换运动学插件

古月居 GYH.AI

Orocos Kinematics and Dynamics

Smarter in motion!



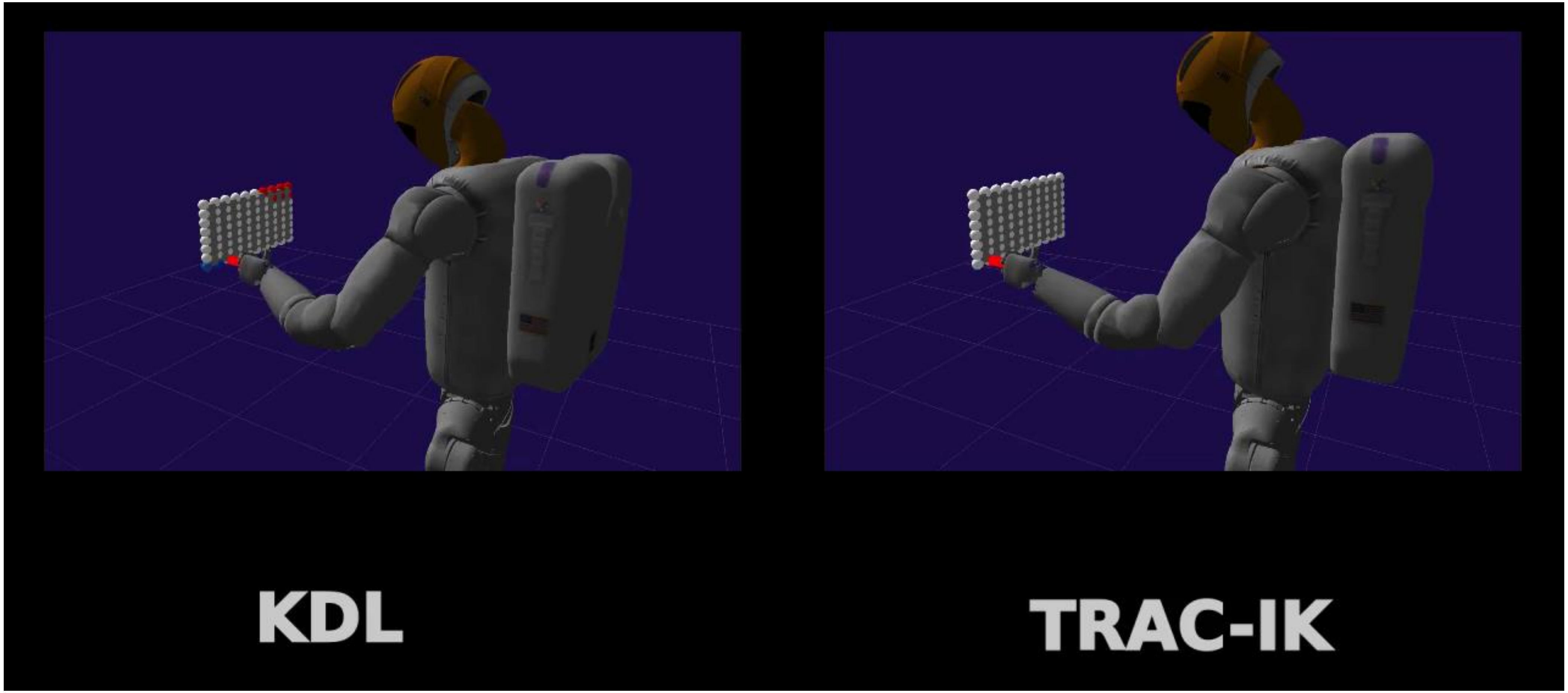
MoveIt!默认使用的运动学求解器

- 数值解
- 优点：可求解封闭情况下逆运动学
- 缺点：速度慢、失败率高

*** 参考链接：http://wiki.ros.org/kdl**

4. 更换运动学插件

traclabs



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

4. 更换运动学插件

古月居 GYH.AI

安装	$ sudo apt-get install ros-kinetic-trac-ik-kinematics-plugin
配置	$ rosed "$MYROBOT_NAME"_moveit_config/config/kinematics.yaml
测试	$ sudo "$MYROBOT_NAME"_moveit_config demo.launch

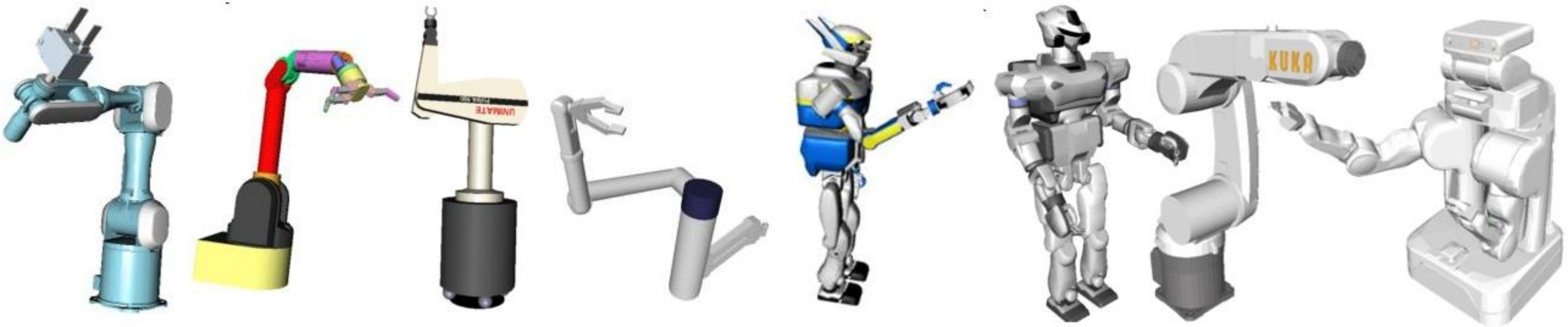
```
arm:
  kinematics_solver: trac_ik_kinematics_plugin/TRAC_IKKinematicsPlugin
  kinematics_solver_attempts: 3
  kinematics solver search resolution: 0.005
```

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

4. 更换运动学插件

OpenRAVE

- 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/

4. 更换运动学插件

古月居 GYH.AI

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 libpcrecpp0v5 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**

4. 更换运动学插件

古月居 GYH.AI

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="probot_anno" $ 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**

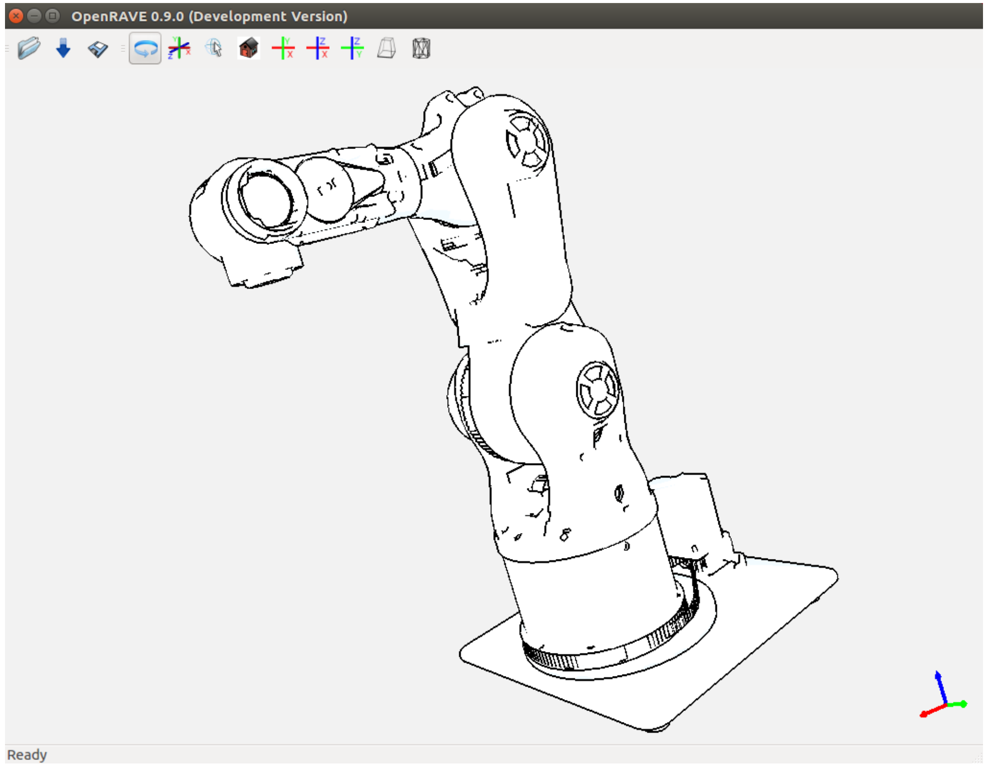
4. 更换运动学插件

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
-----------------------------------
base_footprint 0
base_link       1     base_footprint
link_1          2     base_link
link_2          3     link_1
link_3          4     link_2
link_4          5     link_3
link_5          6     link_4
link_6          7     link_5
tool0           8     link_6
-----------------------------------
name            index parents
```



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

4. 更换运动学插件

生成程序文件

```
$ export PLANNING_GROUP="manipulator"
$ export BASE_LINK="1"
$ export EEF_LINK= “8"
$ 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"
```

* 注意：机械臂的初始位姿不能是奇异姿态，否则会报错

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 “
$ cd ~/catkin_ws  & catkin_make
```

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

4. 更换运动学插件

修MoveIt!配置文件　$ rosed "$MYROBOT_NAME"_moveit_config/config/kinematics.yaml

```yaml
manipulator:
  kinematics_solver: probot_anno_manipulator_kinematics/IKFastKinematicsPlugin
  kinematics_solver_search_resolution: 0.005
  kinematics_solver_timeout: 0.05
```

测试IKFast插件　$ sudo probot_anno_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

本讲小结

圆弧轨迹规划	• 计算圆弧轨迹 → 规划笛卡尔路径 → 执行轨迹运动
轨迹重定义	• 规划轨迹 → 轨迹冲定义 → 执行轨迹运动
多轨迹连续运动	• 计算多条轨迹 → 轨迹拼接 → 重新规划速度、加速度 → 执行轨迹运动
更换运动学插件	• KDL：易使用，但失败率高、效率低 • TRAC-IK：成功率高，但求解不稳定 • IKFAST：成功率高、求解稳定、速度快，但存在多解选择问题

1. 使用自己的机械臂模型，分别编写程序，实现以下功能：

（1）圆弧运动：机械臂终端完成圆弧轨迹的规划运动，半径和圆心根据模型确定接口；

（2）轨迹重定义：针对规划得到的轨迹，缩减1/4的轨迹点，并完成运动，例如：原本有20个轨迹点，每隔4个删掉一个，最后得到16个点（首尾两点不能删除），再完成运动；

（3）多轨迹连续运动：完成至少两条轨迹的拼接、重规划和连续运动，具体运动类型不限制。

2. 修改运动学插件：

（1）配置TRAC-IK运动学插件，测试运行以上例程；

（2）配置IKFAST运动学插件，测试运行以上例程。（选做）

扩展阅读

- MoveIt! API Document

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

- ROS技术点滴 —— MoveIt!中的运动学插件

https://mp.weixin.qq.com/s/Ce64XdT8GxjbejeQOayiFw

- IKFAST Tutorials

http://docs.ros.org/kinetic/api/moveit_tutorials/html/doc/ikfast/ikfast_tutorial.html

- 《Introduction to ROBOTICS》，John J. Craig，Chapter 3~4

Thank You

怕什么真理无穷，进一寸有一寸的欢喜

更多精彩，欢迎关注

古月居

古月春旭