3 AUBO 协作机器人 ROS 环境建模

3.1 ROS 建模

在上一章的实践课程中我们学习了 ROS 下的基本程序开发，这一章我们要开始进行一个实际工业机械臂的 ROS 建模，首先我们还是来看看我们要建模的机器人长什么样，图 3.1 为 AUBO-i5 的结构简图。

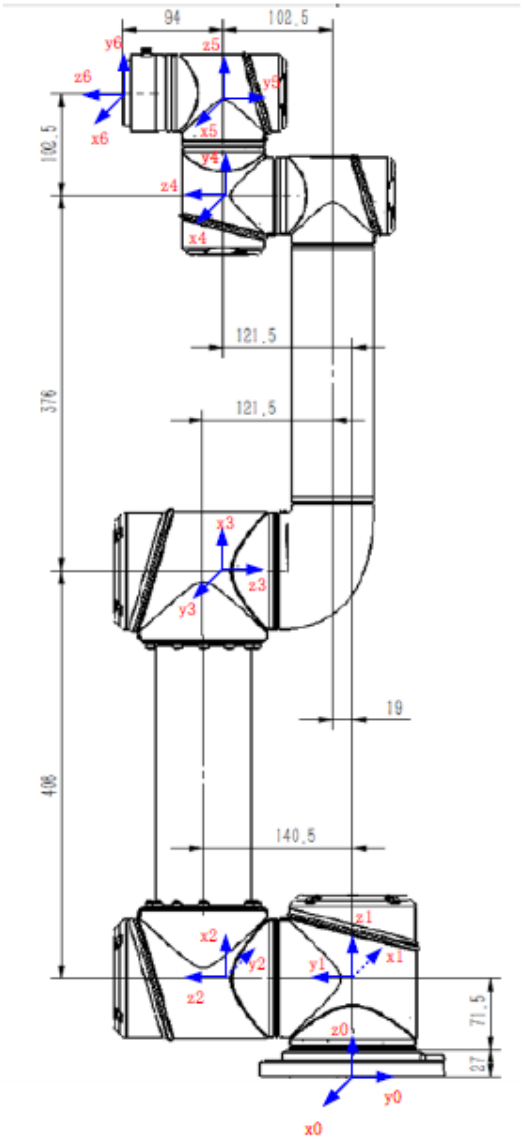

图 3.1 AUBO i5 结构简图

ROS 下的仿真，具有图形化界面的工具有 rviz 以及 gazebo，rviz 适合用来学习运动学仿真，gazebo 既可以完成运动学仿真，也可以完成动力学仿真。确切的说，rviz 只是一个三维可视化工具，强调把已有的数据可视化显示，有很多资料介绍搭配 ArbotiX 进行仿真；gazebo 是三维物理仿真平台，强调的是创建一个虚拟的仿真环境，也可以连接实际的机器人进行控制。一谈到 ROS 机器人仿真，大家可能会说 MoveIt!，MoveIt!是 ROS 中一个重要的集成化开发平台，由一系列

移动操作的功能包组成，提供运动规划、操作控制、3D 感知、运动学等功能模块，是 ROS 社区中使用度排名前几的功能包，目前已经支持众多机器人硬件平台。MoveIt!中有很多开源的运动学求解器，典型代表性的有 KDL（Kinematics and Dynamics Library）、TRAC-IK 以及 IKFAST 等，目前实验室在做双臂协调控制，也正考虑将它们进行对比，看哪一个更适合做，但这是后话。目前对于初学者来说，这些软件在 ROS 下都需要搭配 MoveIt!来完成，MoveIt!非常庞大，对于初学者来说不是一个好的选择。我们的理论课程以前选择 matlab，matlab 下有 peter corke(http://petercorke.com/wordpress/toolboxes/robotics-toolbox) 的 robotics toolbox，对于教学还是非常好的，但很难对实物仿真，即使用 matlab+vrep 也离实物操作很远，研发就更不用谈了。几次的教学下来，发觉这个环境脱离实际平台的控制，算法理解验证较好上手，但大部分学生学完了，对于实际机器人的开发、控制还是没有什么概念。所以这一次的教程将完全使用 rviz 来建模，不使用 MoveIt!，编程使用 c++，后面上层的东西也可以使用 python 和 lisp 等语言，再复杂一些的研究我们再上手 gazebo 和 MoveIt!。我们的目的是，仿真加实际控制，所以后面一上来就会去控制实物机器人，没有实际控制实验，一些都是虚的！

好了，闲话少说，我们开始工作！

1） 将机械图纸转换为 URDF 文件

一般来说，在 URDF 描述中，我们需要描述一个机器人的所有关节和连杆属性。

URDF 实际是一个 XML 格式的文本文件，其项目的解释见网页（http://wiki.ros.org/urdf/XML/link）。

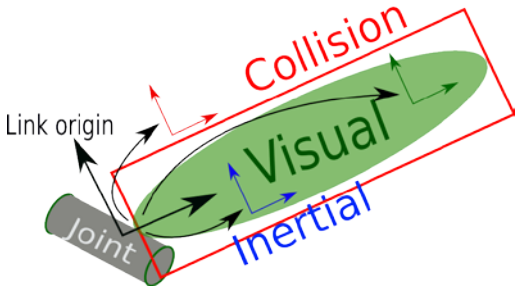

图 3.2 URDF 中连杆 inertial- visual 与关节之间的表示关系

一个机器人的 URDF 应该表示的规范格式如下：

```
<?xml version=”1.0”?>
<robot name=”aubo_i5”>
    <link name=”base_link”>
    ……
    /link>
    <joint name=”first_joint”>
    ……
```

/joint>

……

/robot>

连杆属性一般包含<inertial>惯量、<visual>可视化外观（用来显示模型）、<collision>用来做碰撞检测三个基本的标签，每类标签下面还有几个子属性。

<inertial>惯量标签一般需要指明重心、姿态、质量、转动惯量矩阵。

<visual>可视化标签一般要表明位姿、形状和材料。其中的几何形状可以采用机械制图 DAE 文件格式来加载。

<collision>碰撞检测标签一般要表明位姿、形状等。其中的几何形状可以采用机械制图 STL 文件格式来加载。

图 3.3 为 AUBO-i5 的 base_link 的属性。

```
<link name="base_link">
  <inertial>
    <origin xyz="-1.4795E-13 0.0015384 0.020951" rpy="0 0 0" />  重心和RPY姿态
    <mass value="0.83419" />  质量
    <inertia ixx="0.0014414" ixy="7.8809E-15" ixz="8.5328E-16" iyy="0.0013542" iyz="-1.4364E-05" izz="0.0024659" />
  </inertial>  惯性矩阵
  <visual>
    <origin xyz="0 0 0" rpy="0 0 0" />  世界坐标系下的位姿
    <geometry>
      <mesh filename="package://aubo_simulation/meshes/aubo_i5/visual/base_link.DAE" />  由外部可视化DAE格式文件加载外观
    </geometry>
    <material name="">
      <color rgba="1 1 1 1" />  材料颜色
    </material>
  </visual>
  <collision>
    <origin xyz="0 0 0" rpy="0 0 0" />  世界坐标系下的位姿
    <geometry>
      <mesh filename="package://aubo_simulation/meshes/aubo_i5/collision/base_link.STL" />  由外部STL格式文件加载碰撞检测的外形
    </geometry>
  </collision>
</link>
```

图 3.3 AUBO-i5 的 base_link 的 URDF 描述

图 3.4 为 AUBO-i5 的 shoulder_joint 的属性。

```
<joint name="shoulder_joint" type="revolute">  shoulder joint为旋转关节
  <origin xyz="0 0 0.1215" rpy="0 0 3.1416" /> <!--0.122?-->
  <parent link="base_link" />  上一级连杆为base_link
  <child link="shoulder_Link" />  下一级连杆为shoulder Link
  <axis xyz="0 0 1" />  轴心
  <limit lower="-3.05" upper="3.05" effort="0" velocity="0" />  关节限制，转动角弧度值
</joint>
```

图 3.4 AUBO-i5 的 shoulder_joint 的 URDF 描述

AUBO-i5 是一个六关节的串联型协作机械臂，所以目前状态下，我们还没有末端工具及传感器的情况下，最少也要构建六个关节和七个连杆的模型。需要注意的是 base_link 始终是第一个连杆，我们依次需要构建 shoulder_link、shoulder_joint、upperArm_link、upperArm_joint、foreArm_link、foreArm_joint、wrister1_link 、 wrister1_joint 、 wrister2_link 、 wrister2_joint 、 wrister3_link 、wrister3_joint。另外我们还需要指定一个 world 连杆，并定义一个 world_joint，world_joint 的为 fixed 类型关节，父连杆是 world，子连杆是 base_link。

至此，我们明白机械臂在 rviz 世界坐标系下的空间几何关系，那么我们可以根据图 3.1 给出完整的 URDF 文件。

```
<?xml version="1.0"?>
<robot  name="aubo_i5">

  <link name="base_link">
    <inertial>
      <origin xyz="-1.4795E-13 0.0015384 0.020951" rpy="0 0 0" />
      <mass value="0.83419" />
      <inertia ixx="0.0014414" ixy="7.8809E-15" ixz="8.5328E-16"
iyy="0.0013542" iyz="-1.4364E-05" izz="0.0024659" />
    </inertial>
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh
filename="package://aubo_description/meshes/aubo_i5/visual/base_link.DAE" />
      </geometry>
      <material name="">
        <color rgba="1 1 1 1" />
      </material>
    </visual>
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh
filename="package://aubo_description/meshes/aubo_i5/collision/base_link.STL" /
      </geometry>
    </collision>
  </link>
```

```
  <link name="shoulder_Link">
    <inertial>
      <origin xyz="3.2508868974735E-07 0.00534955349296065 -0.00883689325611056"
rpy="0 0 0" />
      <mass value="1.57658348693929" />
      <inertia ixx="0.0040640448663128" ixy="0" ixz="0" iyy="0.00392863238466817"
iyz="-0.000160151642851425" izz="0.0030869857349184" />
    </inertial>
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh
filename="package://aubo_description/meshes/aubo_i5/visual/shoulder_Link.DAE" />
      </geometry>
      <material name="">
        <color rgba="1 1 1 1" />
      </material>
    </visual>
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh
filename="package://aubo_description/meshes/aubo_i5/collision/shoulder_Link.STL" />
      </geometry>
    </collision>
  </link>

  <joint name="shoulder_joint" type="revolute">
    <origin xyz="0 0 0.1215" rpy="0 0 3.1416" />
    <parent link="base_link" />
    <child link="shoulder_Link" />
    <axis xyz="0 0 1" />
    <limit lower="-3.05" upper="3.05" effort="0" velocity="0" />
  </joint>
```

```
  <link name="upperArm_Link">
    <inertial>
      <origin xyz="0.203996646979614 2.01304585036544E-10 0.0127641545395984"
rpy="0 0 0" />
      <mass value="4.04175782265494" />
      <inertia ixx="0.00965399211106204" ixy="0" ixz="0" iyy="0.144993869035655"
iyz="0" izz="0.142607184038966" />
    </inertial>
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh
filename="package://aubo_description/meshes/aubo_i5/visual/upperArm_Link.DAE"
/>
      </geometry>
      <material name="">
        <color rgba="1 1 1 1" />
      </material>
    </visual>
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh
filename="package://aubo_description/meshes/aubo_i5/collision/upperArm_Link.STL" />
      </geometry>
    </collision>
  </link>

  <joint name="upperArm_joint" type="revolute">
    <origin xyz="0 0.1215 0" rpy="-1.5708 -1.5708 0" />
    <parent link="shoulder_Link" />
    <child link="upperArm_Link" />
    <axis xyz="0 0 1" />
    <limit lower="-3.05" upper="3.05" effort="0" velocity="0" />
  </joint>
```

```
  <link name="foreArm_Link">
    <inertial>
      <origin xyz="0.188922115560337 6.78882434739072E-07 0.0981026740461557"
rpy="0 0 0" />
      <mass  value="2.27145669098343" />
      <inertia ixx="0.00214322284946289" ixy="0" ixz="-0.00073120631553383"
iyy="0.0443926090878205" iyz="0" izz="0.0441273797128365" />
    </inertial>
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh
filename="package://aubo_description/meshes/aubo_i5/visual/foreArm_Link.DAE" />
      </geometry>
      <material name="">
        <color rgba="1 1 1 1" />
      </material>
    </visual>
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh
filename="package://aubo_description/meshes/aubo_i5/collision/foreArm_Link.STL
" />
      </geometry>
    </collision>
  </link>

  <joint name="foreArm_joint" type="revolute">
    <origin xyz="0.408 0 0" rpy="-3.1416 -5.1632E-18 -5.459E-16" />
    <parent link="upperArm_Link" />
    <child link="foreArm_Link" />
    <axis xyz="0 0 1" />
    <limit lower="-3.05" upper="3.05" effort="0" velocity="0" />
  </joint>
```

```
  <link name="wrist1_Link">
    <inertial>
      <origin xyz="7.54205137428592E-07 0.0062481254331257 -0.00392367464072373"
rpy="0 0 0" />
      <mass value="0.500477539188764" />
      <inertia ixx="0.00071194605962081" ixy="0" ixz="0"
iyy="0.00040588242872958" iyz="-2.30808694377512E-05"
izz="0.000685574004861334" />
    </inertial>
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh
filename="package://aubo_description/meshes/aubo_i5/visual/wrist1_Link.DAE" />
      </geometry>
      <material name="">
        <color rgba="1 1 1 1" />
      </material>
    </visual>
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh
filename="package://aubo_description/meshes/aubo_i5/collision/wrist1_Link.STL"
/>
      </geometry>
    </collision>
  </link>

  <joint name="wrist1_joint" type="revolute">
    <origin xyz="0.376 0 0" rpy="3.1416 -1.8323E-15 1.5708" />
    <parent link="foreArm_Link" />
    <child link="wrist1_Link" />
    <axis xyz="0 0 1" />
    <limit lower="-3.05" upper="3.05" effort="0" velocity="0" />
  </joint>
```

```
  <link name="wrist2_Link">
    <inertial>
      <origin xyz="-7.54207620578635E-07 -0.00624812542617262
-0.00392367464115684" rpy="0 0 0" />
      <mass value="0.500477539245988" />
      <inertia ixx="0.00071194605981829" ixy="0" ixz="0"
iyy="0.000405882428755442" iyz="2.30808694515886E-05"
izz="0.000685574005112107" />
    </inertial>
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh
filename="package://aubo_description/meshes/aubo_i5/visual/wrist2_Link.DAE" />
      </geometry>
      <material name="">
        <color rgba="1 1 1 1" />
      </material>
    </visual>
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh
filename="package://aubo_description/meshes/aubo_i5/collision/wrist2_Link.STL"
/>
      </geometry>
    </collision>
  </link>

  <joint name="wrist2_joint" type="revolute">
    <origin xyz="0 0.1025 0" rpy="-1.5708 -1.8709E-15 -1.6653E-16" />
    <parent link="wrist1_Link" />
    <child link="wrist2_Link" />
    <axis xyz="0 0 1" />
    <limit lower="-3.05" upper="3.05" effort="0" velocity="0" />
  </joint>
```

```
  <link name="wrist3_Link">
    <inertial>
      <origin xyz="3.92048778449938E-10 0.000175788057281467
-0.0213294490706684" rpy="0 0 0" />
      <mass value="0.158309554874285" />
      <inertia ixx="7.31376196034769E-05" ixy="0" ixz="0"
iyy="7.19528188876563E-05" iyz="0" izz="0.000108772439051422" />
    </inertial>
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh
filename="package://aubo_description/meshes/aubo_i5/visual/wrist3_Link.DAE" />
      </geometry>
      <material name="">
        <color rgba="1 1 1 1" />
      </material>
    </visual>
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh
filename="package://aubo_description/meshes/aubo_i5/collision/wrist3_Link.STL"
/>
      </geometry>
    </collision>
  </link>

  <joint name="wrist3_joint" type="revolute">
    <origin xyz="0 -0.094 0" rpy="1.5708 0 1.7907E-15" />
    <parent link="wrist2_Link" />
    <child link="wrist3_Link" />
    <axis xyz="0 0 1" />
    <limit lower="-3.05" upper="3.05" effort="0" velocity="0" />
  </joint>



  <link name="world" />

  <joint name="world_joint" type="fixed">
    <parent link="world" />
    <child link = "base_link" />
    <origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 0.0" />
  </joint>
</robot>
```

有关连杆的重心、质量、转动惯量矩阵，这些参量对于动力学仿真来说是很

重要的，这些参数可以在设计机械结构的时候由 CAD 软件算出，比如我们可以在 Solidworks 软件里打开 base_link.stl 文件，单击“质量测量”，如图 3.5 所示。

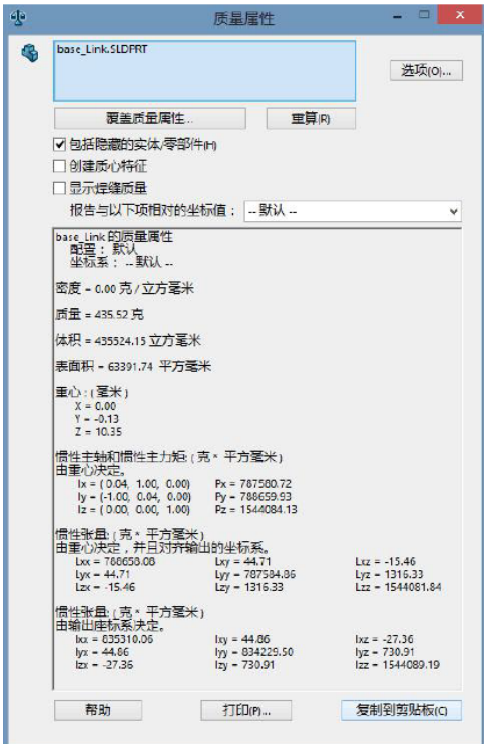

图 3.5 计算连杆重心、质量、转动惯量矩阵

3) 第三步，我们开始创建一个 ROS 包，完成 AUBO i5 的 rviz 仿真。

cd catkin_ws/src

catkin_create_pkg aubo_simulation

上面的命令会为我们创建一个 aubo_simulation 目录，并生成 CMakeLists.txt 以及 package.xml 文件

我们在 aubo_simulation 目录下创建几个目录：

urdf 目录：存放我们编写的 urdf 文件；

meshes 目录：存放前面介绍的几个 STL 和 DAE 文件；

src 目录：准备后面放我们编的程序文件；

rviz 目录：准备放入 rviz 的 GUI 配置文件，urdf.rviz 这个文件可以直接从 /opt/ros/kinetic/share/urdf_tutorial/rviz 目录下拷贝过来，这样便于以后修改配置文件；

launch 目录：准备放入后面我们编写的 launch 文件。

将刚才我们编写的 urdf 命名为 aubo_i5.urdf，存放在 urdf 目录，从遨博官方网站下载各连杆部件的 STL 以及 DAE 文件，网站地址：

https://github.com/lg609/aubo_robot/tree/master/aubo_description。

把它们分别放在 meshes 目录下的 visual 和 collision 目录里面。

4) 第四步，编写一个 launch 文件，内容截图见图 3.7。

```
<launch>

  <arg name="model" />
  <arg name="gui" default="true" />
  <arg name="rvizconfig" default="$(find aubo_simulation)/rviz/urdf.rviz" />

  <param name="robot_description" command="$(find xacro)/xacro.py $(find aubo_simulation)/urdf/aubo_i5.urdf" />
  <param name="use_gui" value="$(arg gui)"/>

  <!--node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher" /-->
  <node name="robot_state_publisher" pkg="robot_state_publisher" type="state_publisher" />
  <node name="rviz" pkg="rviz" type="rviz" args="-d $(arg rvizconfig)" required="true" />

</launch>
```

图 3.6 display.launch 文件

把它命名为 display.launch。

回到 catkin_ws 目录，执行 catkin_make，编译通过，然后执行 source 命令，为了加深理解，先把 display.launch 文件里面 joint_state_publisher 节点的注释取消，看看效果。

在 catkin_ws 目录下，运行命令：

roslaunch aubo_simulation display.launch gui:= true

可以看到 rviz 为我们成功加载了机器人模型，见图 3.7。并且还有一个关节控制的 GUI 界面，可以通过拖动关节来让机械臂动起来。

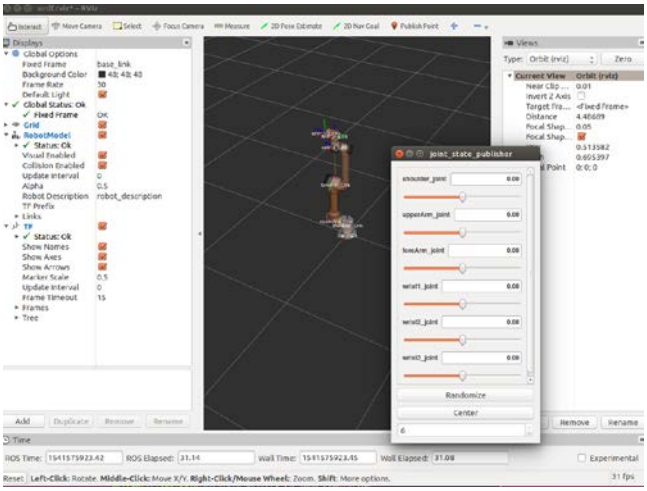

图 3.7 rviz 仿真界面

新开一个终端窗口，键入 rostopic list 命令查看一下，从图 3.8 可以看看刚才的 roslaunch 加载了那些东西。

```
^Cesir@esir-XPS-8920:~/catkin_ws$ rostopic list
/clicked_point
/initialpose
/joint_states
/move_base_simple/goal
/rosout
/rosout_agg
/tf
/tf_static
esir@esir-XPS-8920:~/catkin_ws$ rosnode list
/joint_state_publisher
/robot_state_publisher
/rosout
/rviz
esir@esir-XPS-8920:~/catkin_ws$
```

图 3.8 加载的 topic 和 node 列表

可以看到刚才的 display.launch 加载了四个节点，由于我们没有预先运行 roscore,所以 rosout 节点是默认先行加载的，然后加载了 joint_state_publisher、robot_state_publisher 以及 rviz 节点。主题 topic 里面我们这个阶段比较关心的是 joint_states 主题，它负责向机器人模型发送关节消息。

运行 rostopic echo /joint_states 命令，见图 3.9，可以看到系统不停的在发布关节位置信息，拖动 joint_state_publisher 的滑竿控制，可以看到关节位置的改变。

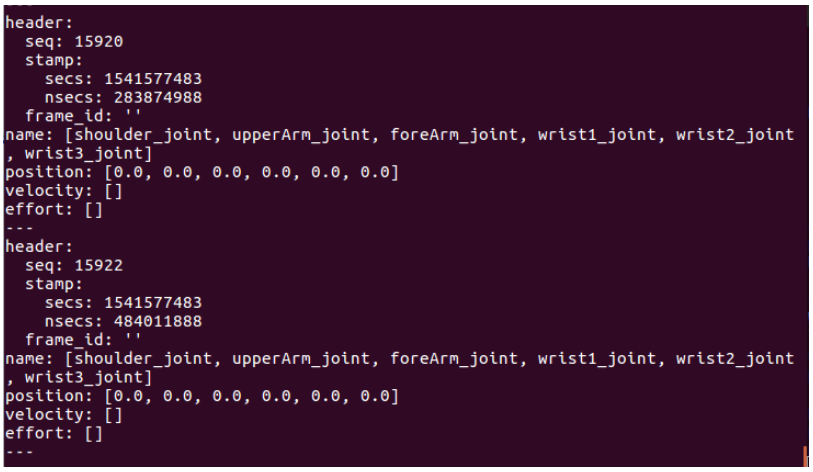

图 3.9　joint_states 主题消息显示

从这个我们可以看到 rviz 里面的关节控制实际也是通过 topic 来发布消息的，既然这样，那么我们是否可以直接编程控制呢？

目前为止本节实验实际还一句 C++代码都没编，我们先手动发布一下 joint_states 主题消息看看是否可以控制机械臂转动。按照图 3.7 把 joint_state_publisher 节点注释掉，重新运行 roslaunch aubo_simulation display.launch GUI:= true，可以 rviz 左边的 GUI 控制面板中看到 RobotModel 显示错误了，点开 Status 可以看到 URDF 解析是 OK 的，但除了 base_link 之外的其他 6 个连杆会显示 No transform，这个是由于没有向/joint_states 主题发布关节控制消息，所以 rviz 认为无法找到其它 6 个关节的 TF 转换关系。所以我们手动发布一个 rostopic 消息，命令如下：

Rostopic pub /joint_states sensor_msgs/JointSate ‘{header: {stamp: now, frame_id: ‘’},name: [‘shoulder_joint’, ‘upperArm_joint’, ‘foreArm_joint’, ‘wrist1_joint’, ‘wrist2_joint’, wrist3_joint’], position: [0.5, 0.5, 0.0, 0.0, 0.0, 0.0], velocity: [], effort: []}　--once

在 rviz 里面我们可以看到 RobotModel 没有任何错误状态了，TF 里面也将这 6 个连杆的 TF 关系添加进来，’Transform OK’了。

3.2 编程仿真

我们有了一个好看的仿真模型，接下来就是编程控制机械臂模型运动。

我们在 aubo_simulation 包的 src 目录底下编写一个简单的测试程序，命名 test.cpp，内容如图 3.10 所示，请自行手打代码练习。

```
#include<iostream>
#include<string>
#include<ros/ros.h>
#include<sensor_msgs/JointState.h>
#include<robot_state_publisher/robot_state_publisher.h>

using namespace std;

sensor_msgs::JointState joint_state;
ros::Publisher joint_pub;

int moveJ(double j1,double j2,double j3,double j4,double j5,double j6);

int main(int argc, char** argv) {
        ros::init(argc, argv, "aubo_joint_publisher");
        ros::NodeHandle n;
        joint_pub = n.advertise<sensor_msgs::JointState>("joint_states", 1);
        ros::Rate loop_rate(10);
        int command;
        char Menu[256]="\n0. Exit.\n1. Control joints to move.\nPlease input your command:";
        double j1,j2,j3,j4,j5,j6;
        double arcAng1,arcAng2,arcAng3,arcAng4,arcAng5,arcAng6;
        while (ros::ok()) {
                printf("%s",Menu);

                scanf("%d",&command);
                switch(command){
                case 0:
                        exit(1);
                        break;
                case 1:
                        printf("Please input 6 joint angle:\n");
                        scanf("%lf%lf%lf%lf%lf%lf",&j1,&j2,&j3,&j4,&j5,&j6);
                        arcAng1=j1*M_PI/180.0;
                        arcAng2=j2*M_PI/180.0;
                        arcAng3=j3*M_PI/180.0;
                        arcAng4=j4*M_PI/180.0;
                        arcAng5=j5*M_PI/180.0;
                        arcAng6=j6*M_PI/180.0;
                        printf("%lf %lf %lf %lf %lf %lf\n", arcAng1,arcAng2,arcAng3,arcAng4,arcAng5,arcAng6);
                        moveJ(arcAng1,arcAng2,arcAng3,arcAng4,arcAng5,arcAng6);
                        break;
                default:
                        printf("No this command!\n");
                        break;
                }
                ros::spinOnce();
                // This will adjust as needed per iteration
                loop_rate.sleep();
        }

        return 0;
}
```

图 3.10 test.cpp 文件

我们自定义了一个 moveJ 函数，用来发布关节运动信息，驱动模型转动，其内容见图 3.11。

编好这个简单的程序后，编译、运行、测试。

代码写完，别忘了修改 CMakeLists.txt 以及 package.xml 文件，在本例程中我们将 test.cpp 编译成目标 myTest，注意，不要把编译目标命名为 test，catkin_make 有可能认为这个目标名冲突。如果 catkin_make 通过，那么开始测试。

打开一个终端，运行:

cd catkin_ws

source devel/setup.bash

roslaunch aubo_simulation display.launch

注意上面这个 display.launch 文件不加载 joint_state_publisher 节点。同样由于没有加载 joint_state_publisher 节点，rviz 的 TF 报错，这个没有关系，等会我们的测试软件发送一条 JointState 主题消息即可。

```
int moveJ(double j1,double j2,double j3,double j4,double j5,double j6){
        //update joint_state
        joint_state.header.stamp = ros::Time::now();
        joint_state.header.frame_id= "";
        joint_state.name.resize(6);
        joint_state.position.resize(6);

        joint_state.name[0]="shoulder_joint";
        joint_state.position[0] = j1;
        //ROS_INFO("shoulder: %f  ",joint_state.position[0]);

        joint_state.name[1] ="upperArm_joint";
        joint_state.position[1] = j2;
        //ROS_INFO("upperArm: %f  ",joint_state.position[1]);

        joint_state.name[2] ="foreArm_joint";
        joint_state.position[2] = j3;
        //ROS_INFO("foreArm: %f  ",joint_state.position[2]);

        joint_state.name[3] ="wrist1_joint";
        joint_state.position[3] = j4;
        //ROS_INFO("wrist1: %f  ",joint_state.position[3]);

        joint_state.name[4] ="wrist2_joint";
        joint_state.position[4] = j5;
        //ROS_INFO("wrist2: %f  ",joint_state.position[4]);

        joint_state.name[5] ="wrist3_joint";
        joint_state.position[5] = j6;
        //ROS_INFO("wrist3: %f\n",joint_state.position[5]);

        joint_state.velocity.resize(6);
        joint_state.effort.resize(6);

        //send the joint state and transform
        joint_pub.publish(joint_state);

        return 0;
}
```

图 3.11 moveJ 函数代码

新开一个终端，运行第一个 c++测试程序，命令如下：

cd catkin_ws

source devel/setup.bash

rosrun aubo_simulation myTest

测试结果见图 3.12 所示，可以看到我们输入 6 个关节角，程序自动转换成弧度，然后发布 JointState 主题消息，控制机械臂转动了。

这次的教程告一段落，下去后请大家认真体会 URDF 模型的编写，以及 JointState 消息的控制，下次课讲解运动学仿真，请大家再复习一下课堂上所学的 DH 表示法、运动学正解、逆解等知识。

希望通过本章实验的学习，可以掌握机器人的仿真模型建立，为后面的运动学仿真、动力学仿真打下基础，理清思路。从简单到复杂，再逐步深入。

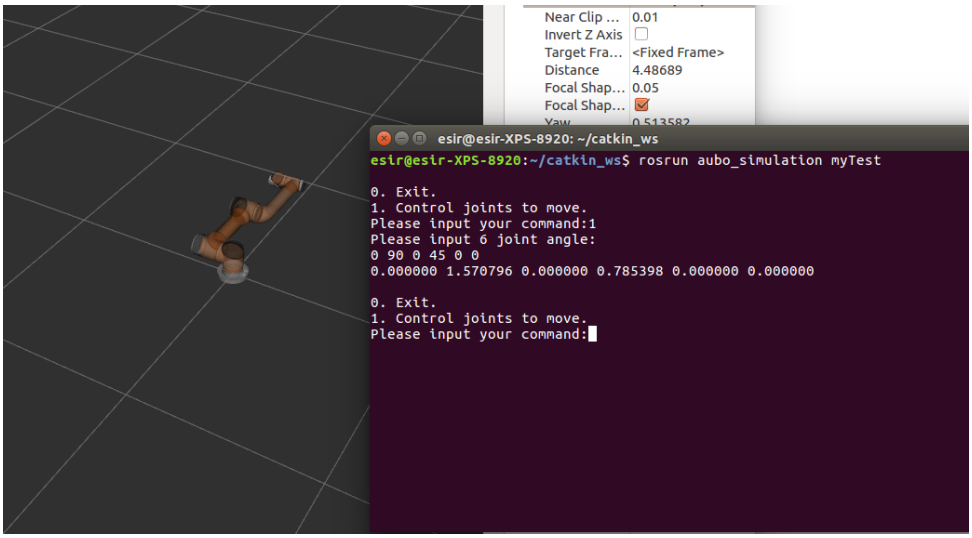

图 3.12 测试结果图

2.8 本章版权及声明

本章内容由闵华松(mhuasong@wust.edu.cn)编写。