韭菜盒子用户手册 V1.2

LABOX USER MANUAL V1.2

主编：陈睿 刘田

武汉京天电器有限公司

WUHAN JINGTIAN ELECTRICAL CO.,LIMITED

2020 年 1 月 武汉

用户手册会定期进行检查和修正，更新后的内容将出现在新版本中。本手册中的内容或信息如有变更，恕不另行通知。
对本手册中可能出现的任何错误或遗漏，或因使用本手册及其中所述产品而引起的意外或间接伤害，武汉京天电器有限公司概不负责。
安装、使用产品前，请阅读本手册。
请保管好本手册，以便可以随时阅读和参考。
本手册中所有图片仅供示意参考，请以收到的实物为准。

注意:

1、这些指令是在Ubuntu 16.04 和 ROS Kinetic测试运行的。
2、如果您想了解更多关于开源韭菜盒子的信息，请参考武汉京天电器手持机械臂用户手册以及华夫派用户手册。
我们很高兴宣布一本新书《ROS机器人编程》，这本书是由Tuttlebot3开发人员编写的。这本书以韩文、英文、中文和日本出版。它包含以下内容：

- ROS动力学KAM:基本概念、指令和工具
- 如何在ROS上使用传感器和执行器包
- 嵌入式ROS板：OpenCR
- 用Turtlebot3进行SLAM和导航
- 如何使用ROS Java编程实现递送机器人
- *韭菜盒子（LABOX-V1）移动机械臂使用MOVEIT的仿真Gazebo

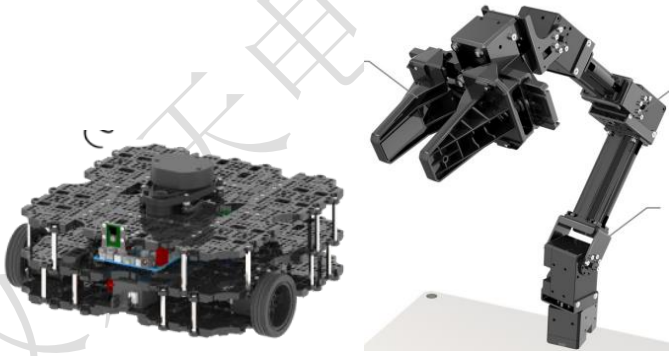
如果需要详细了解，请参阅了解更多关于ROS、SLAM和导航的信息。

目录

1 韭菜盒子介绍与安装

1.1 韭菜盒子核心技术

韭菜盒子是一套为人工智能和机器人学科教学实验而设计的解决方案，是武汉京天电器公司在为全国各大高校的实验教学需求而设计。本着智能机器人在“感知”、“决策”、“执行”三个层次的功能架构，集成了目前最受欢迎的一系列应用技术模块。



感知层 集成有 Turtlebot3 移动机器人的激光雷达、单目相机、IMU、里程计等感知模块，可分别获得环境的距离信息、图像信息、全球绝对位置和自身相对位置信息。

决策层 采用树莓派 3B+作为处理器，拥有更强的 CPU 性能以及更强的无线（WiFi/BT，支持 5G 频段的 WiF 以及支持 BT 4.2&BLE），采用博通集成四核 64 位@ 1.4GHz CPU，集成博通 GPU，内存：1GB LPDDR2 SDRAM。满足智能机器人的机器视觉、地图构建、定位、导航规划等任务的实时运算。

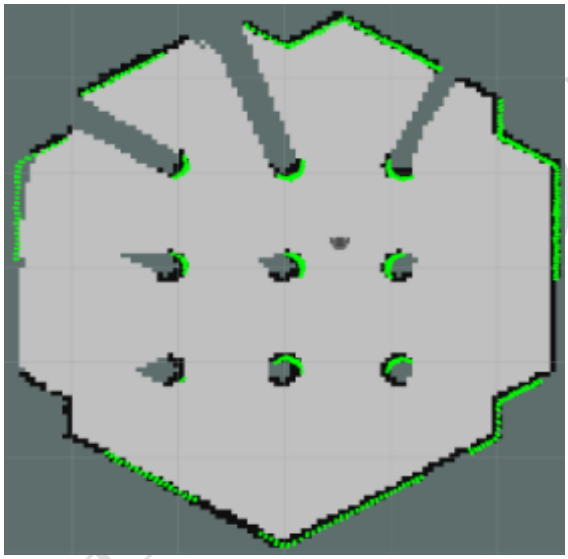
控制层 包含有轮式移动和机械臂操作两部分，轮式移动扩大了机械臂的操作范围，机械臂操作增强了轮式移动处理复杂任务的能力。移动部分是采用双轮差速驱动的 Turtlebot3 机器人，它是目前 ROS 官方原生支持的机器人平台。机械臂操作部分采用 OpenManipulater 机械臂，它具有 4 个关节自由度和 1 个夹抓自由度。Turtlebot3 和 OpenManipulater 使韭菜盒子具有了自主定位与地图构建、定位导航、机械臂操作等丰富的功能，完成决策层发布的多样化任务。

1.2 韭菜盒子应用场景

1.2.1 SLAM 地图构建

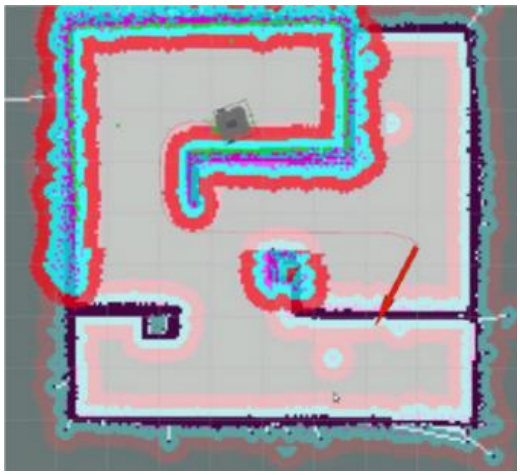
SLAM 是同步定位与地图构建(Simultaneous Localization And Mapping)的缩写，它主要用于解决移动机器人在未知环境中运行时定位导航与地图构建的问题。

在韭菜盒子中，首先获得 HLS 激光雷达采集的点云，IMU 惯性测量单元累计积分的轨迹和位移信息，差速轮的里程计记录的差速移动信息，然后在 ROS 系统中对三者信息进行融合（常用的算法有粒子滤波、扩展卡尔曼滤波、图优化），得到机器人的最佳状态估计，最后融合序列扫描数据，得到环境地图。



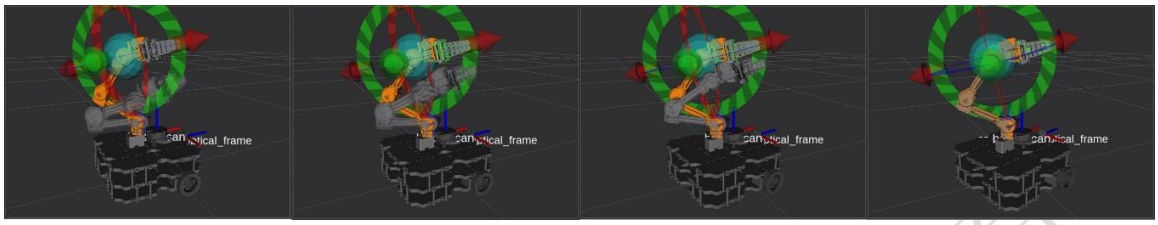
1.2.2 自主导航

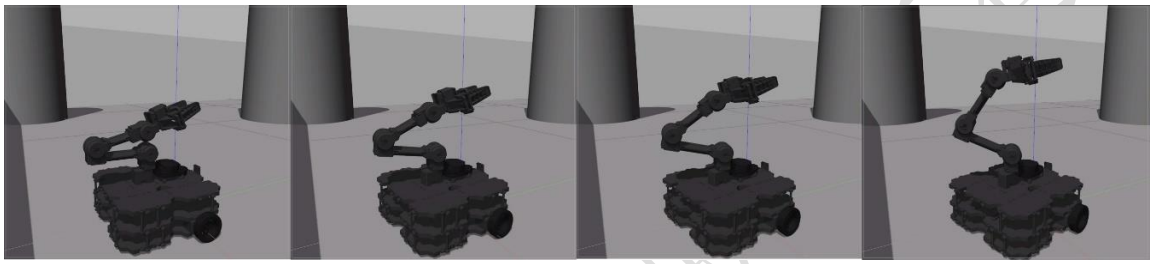
自主导航功能是移动机器人智能化的基础功能，自主导航就像是一位“出租车司机”，“乘客”告诉目的地，韭菜盒子通过对地图的处理、规划出全局的轨迹路线，并实时读取雷达信息进行动态避障，最终到达目的地。高性能的处理器和高精度的 dynamixel 舵机，让韭菜盒子成为了一名更加优秀的智能“司机”。



1.2.3 机械臂运动规划

在ROS下的gazebo仿真软件功能十分强大，可以在ompl开源运动规划库中选择合适的路径规划算法应用在机械臂上，并在gazebo中进行机械臂运动规划仿真。它可以很好的模拟实际环境避开各种障碍物进行抓取。



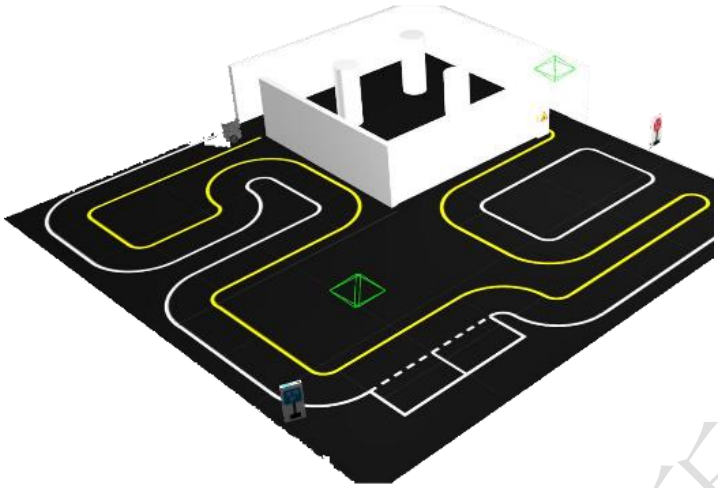


1.2.4 移动抓取操作

机械臂操是具有模仿人类手臂功能并可完成各种作业的自动控制设备，但由于基座固定，只能在局限的空间作业，这极大的限制了机械臂的能力。
为打破这一限制，在韭菜盒子中，将机械臂与移动底盘结合，使机械臂在操作控制中不受基座的限制，能完成更大范围的任务。

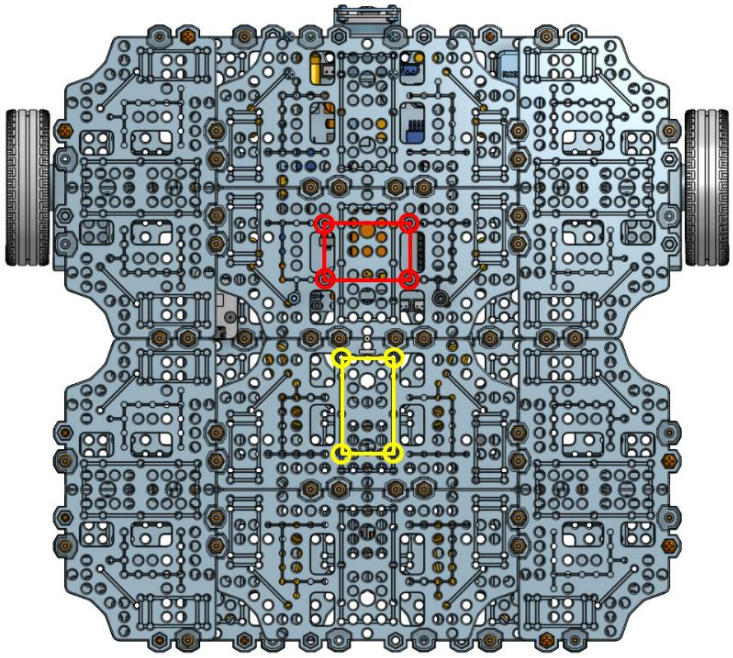
1.2.5 无人驾驶和机器视觉

韭菜盒子中的单目相机和 GPU 运算使韭菜盒子能够完成视觉感知和图像处理识别等功能，可以完成无人驾驶中的车道线识别、交通信号读取、路障、停车位置等任务。同时激光雷达也可以在无光或者暗光的环境中执行任务，与视觉感知相互补充。

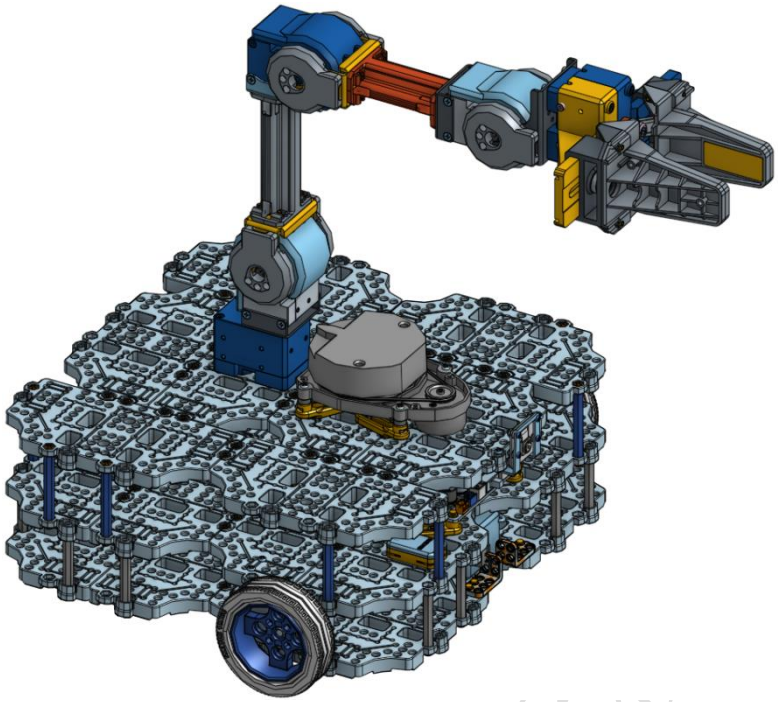


1.3 韭菜盒子硬件安装

韭菜盒子的底盘和机械臂分别按照前文中关于 turtlebot3 和机械臂的结构安装方法进行安装。移动底盘的安装流程与 turtlebot3 相同，先按照 turtlebot3 的机械结构安装方法进行安装。然后将雷达和机械臂的位置进行更改。



其中红色位置为激光雷达的安装孔，黄色位置为机械臂的安装孔。安装好后的整体结构如下图所示。



1.4 韭菜盒子软件安装

1. 首先按照 Turtlebot 3 的装机教程分别安装（笔记本电脑）的 Ubuntu 系统、ROS 以及 Turtlebot3 的依赖软件包。整体购买韭菜盒子的用户树莓派系统已经烧写完成，只需配置工作空间的软件部分。

注意：安装 ubuntu debain 系统可能无法使用 sudo apt-get 安装，建议使用 git clone 源码安装。

2. 安装 Openmanipulator 的 ROS 依赖包

$ cd ~/catkin_ws/src/

$ git clone https://github.com/ROBOTIS-GIT/open_manipulator_with_tb3.git

$ git clone https://github.com/ROBOTIS-GIT/open_manipulator_with_tb3_msgs.git

$ git clone https://github.com/ROBOTIS-GIT/open_manipulator_with_tb3_simulations.git

$ git clone https://github.com/ROBOTIS-GIT/open_manipulator_perceptions.git

$ sudo apt-get install ros-kinetic-smach* ros-kinetic-ar-track-alvar ros-kinetic-ar-track-alvar-msgs

$ cd ~/catkin_ws && catkin_make

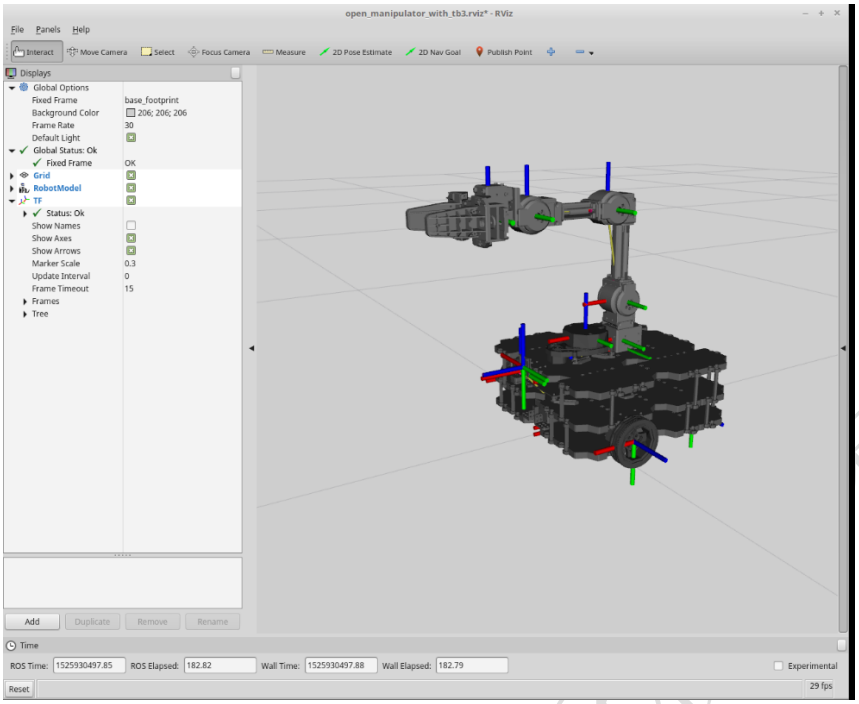
启动韭菜盒子命令：

$ export TURTLEBOT3_MODEL=${TB3_MODEL}

$ roslaunch open_manipulator_with_tb3_description open_manipulator_with_tb3_rviz.launch

如果 catkin_make 编译完成且无任何错误的指令，韭菜盒子（LABOX）的准备工作就已经完成。然后在 RViZ 上用 OpenManipulator 加载 TurtleBot3 Waffle Pi。

TIP：在执行此命令之前，必须指定 TurtleBot3 的型号名称。${TB3_MODEL} 是您在华夫派中使用的模型的名称。



1.5 韭菜盒子 OpenCR 固件设置

OpenCr 的固件可采用两种方法进行烧录，建议使用 shell 脚本，如果需要自定义修改 TurtleBot3 的固件，可以使用第二种方法。

方法＃1：Shell 脚本，使用 shell 脚本上传预先构建的二进制文件。

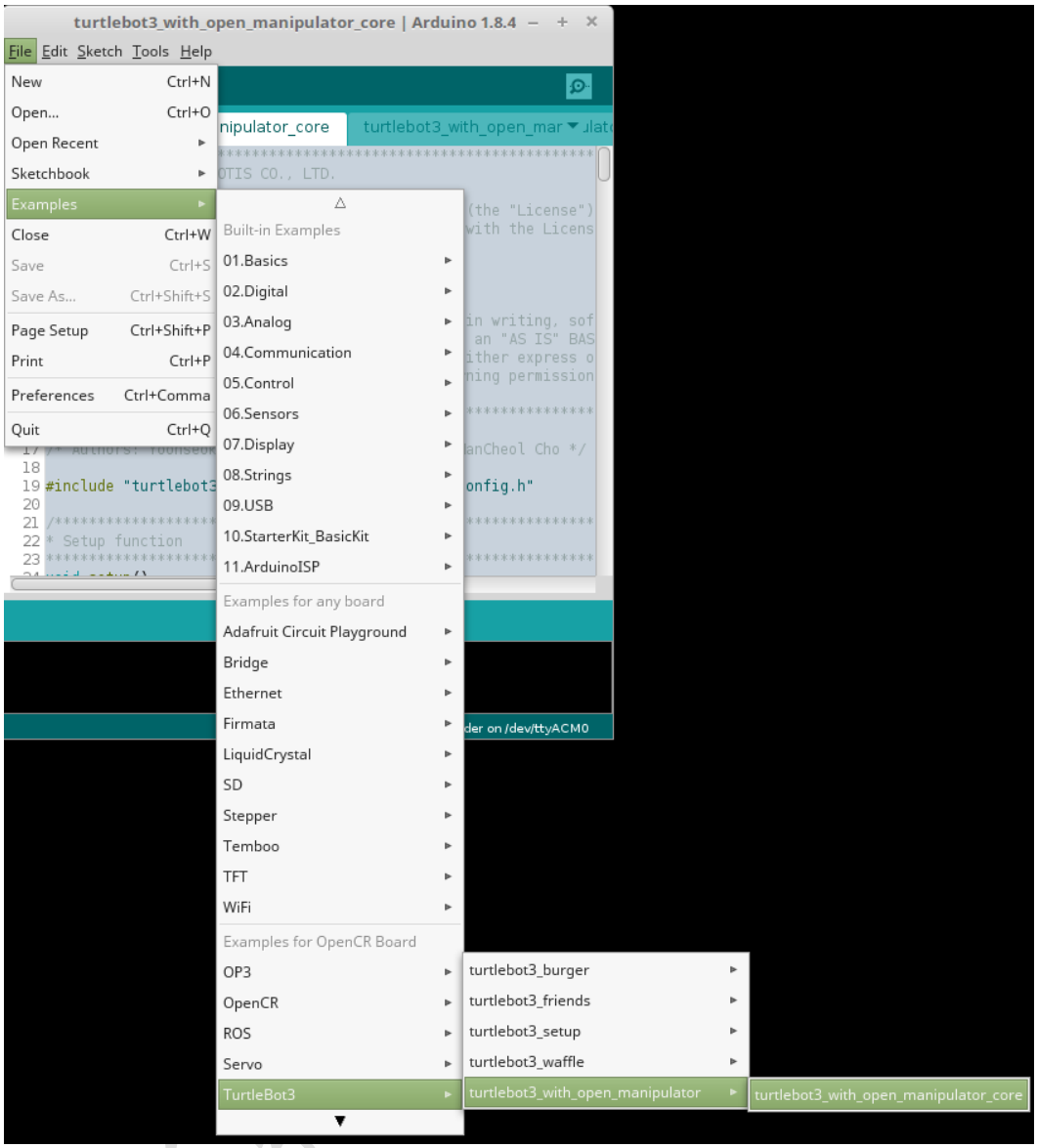
```
$ export OPENCR_PORT=/dev/ttyACM0
$ export OPENCR_MODEL=om_with_tb3
$ rm -rf ./opencr_update.tar.bz2
$ wget https://github.com/ROBOTIS-GIT/OpenCR-Binaries/raw/master/turtlebot3/ROS1/latest/opencr_update.tar.bz2 && tar -xvf opencr_update.tar.bz2 && cd ./opencr_update && ./update.sh $OPENCR_PORT $OPENCR_MODEL.opencr && cd ..
```

方法＃2：Arduino IDE，构建提供的源代码并使用 Arduino IDE 上传生成的二进制文件。

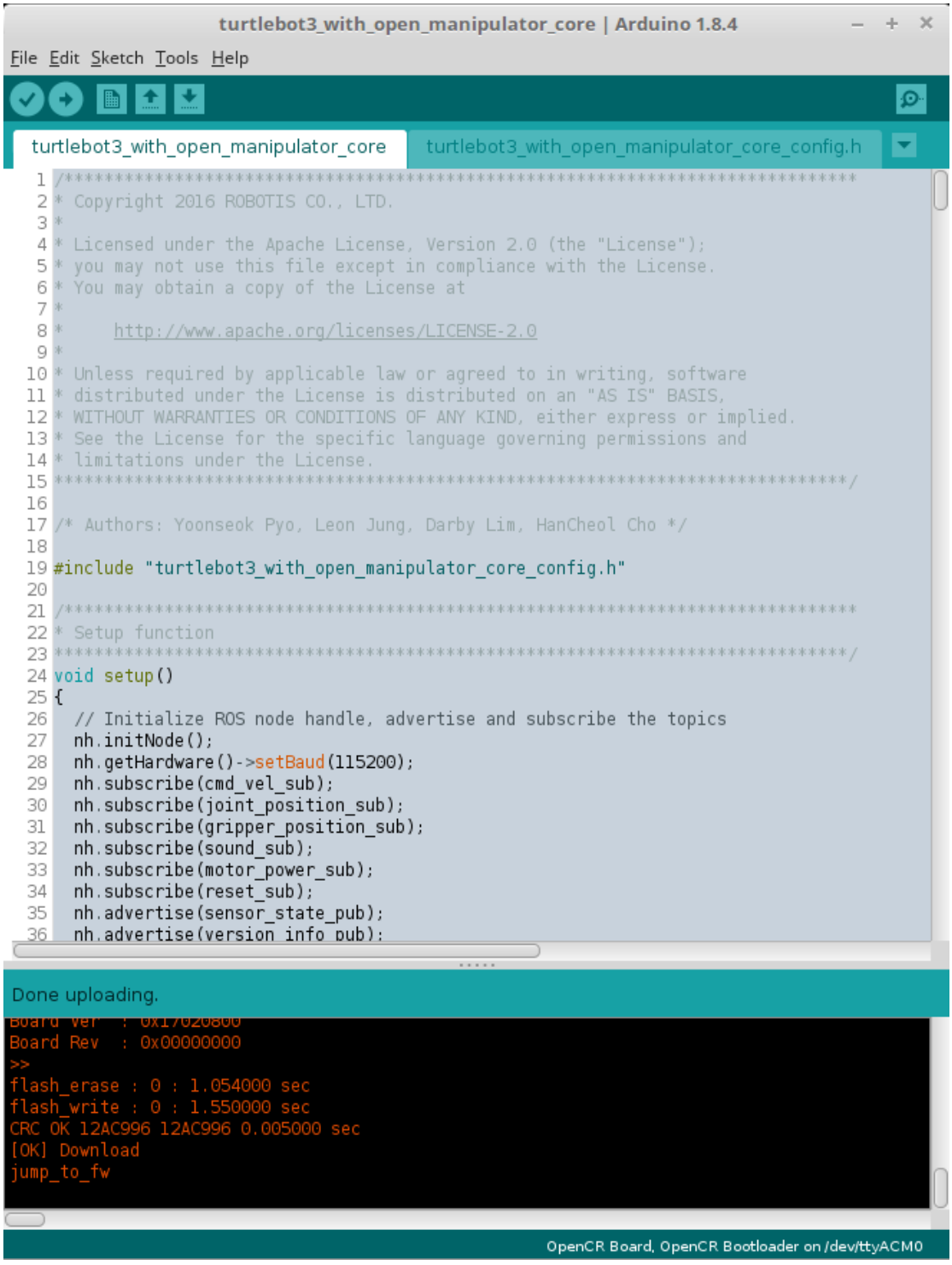
使用 OpenMANIPULATOR 的 TurtleBot3 的 OpenCR 固件（或源代码）是控制 ROS 中的 DYNAMIXEL 和传感器。固件位于 OpenCR 示例中，由板管理器下载。

固件烧录步骤如下所示：

File→Examples→TurtleBot3→turtlebot3_with_open_manipulator→turtlebot3_with_open_manipulator_core。



点击上传按钮，上传成功界面如下图所示：


turtlebot3_with_open_manipulator_core | Arduino 1.8.4
File Edit Sketch Tools Help
turtlebot3_with_open_manipulator_core
turtlebot3_with_open_manipulator_core_config.h
/***
* Copyright 2016 ROBOTIS CO., LTD.
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*   http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
***/
/* Authors: Yoonseok Pyo, Leon Jung, Darby Lim, HanCheol Cho */
#include "turtlebot3_with_open_manipulator_core_config.h"
/***
* Setup function
***/
void setup()
{
// Initialize ROS node handle, advertise and subscribe the topics
nh.initNode();
nh.getHardware()->setBaud(115200);
nh.subscribe(cmd_vel_sub);
nh.subscribe(joint_position_sub);
nh.subscribe(gripper_position_sub);
nh.subscribe(sound_sub);
nh.subscribe(motor_power_sub);
nh.subscribe(reset_sub);
nh.advertise(sensor_state_pub);
nh.advertise(version_info_pub);
Done uploading.
Board Rev : 0x00000000
>>
flash_erase : 0 : 1.054000 sec
flash_write : 0 : 1.550000 sec
CRC OK 12AC996 12AC996 0.005000 sec
[OK] Download
jump_to_fw
OpenCR Board, OpenCR Bootloader on /dev/ttyACM0

2.韭菜盒子使用

2.1 韭菜盒子 SLAM 与自主导航

2.1.1 启动 turtlebot3 节点

注意：请仔细检查在 OpenCR USB 端口名称 turtlebot3_core.launch
提示：在执行此命令之前，您必须指定 TurtleBot3 的型号名称。该 ${TB3_MODEL}是你正在使用的模型的名称 waffle，waffle_pi。如果要永久设置导出设置，请将该命令写到 .bashrc 文件中。

- [TurtleBot3]启动 rosserial 和激光雷达节点

$ export TURTLEBOT3_MODEL=${TB3_MODEL}

$ ROS_NAMESPACE=om_with_tb3 roslaunch turtlebot3_bringup

turtlebot3_robot.launch multi_robot_name:=om_with_tb3

set_lidar_frame_id:=om_with_tb3/base_scan

- [TurtleBot3]启动 rpicamera 节点

$ ROS_NAMESPACE=om_with_tb3 roslaunch turtlebot3_bringup

turtlebot3_rpicamera.launch

- [Remote PC]启动启动 robot_state_publisher 节点

$ ROS_NAMESPACE=om_with_tb3 roslaunch open_manipulator_with_tb3_tools

om_with_tb3_robot.launch

2.1.2 SLAM

[Remote PC]启动 slam 节点

$ export TURTLEBOT3_MODEL=${TB3_MODEL}

$ roslaunch open_manipulator_with_tb3_tools slam.launch use_platform:=true

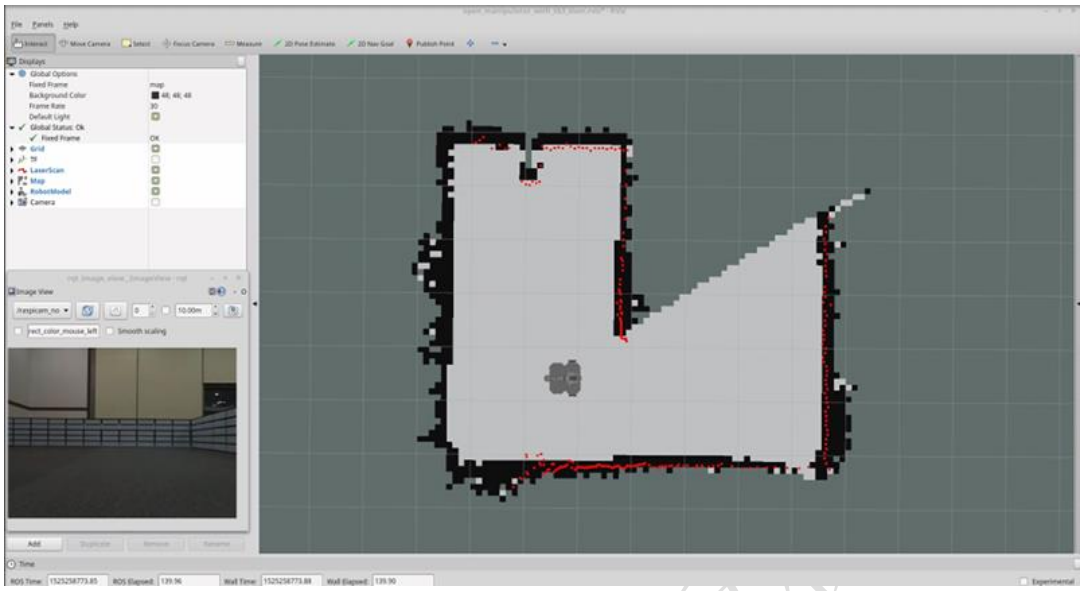
[远程 PC]启动远程节点

$ ROS_NAMESPACE=om_with_tb3 roslaunch turtlebot3_teleop

turtlebot3_teleop_key.launch

[Remote PC]启动 map_saver 节点

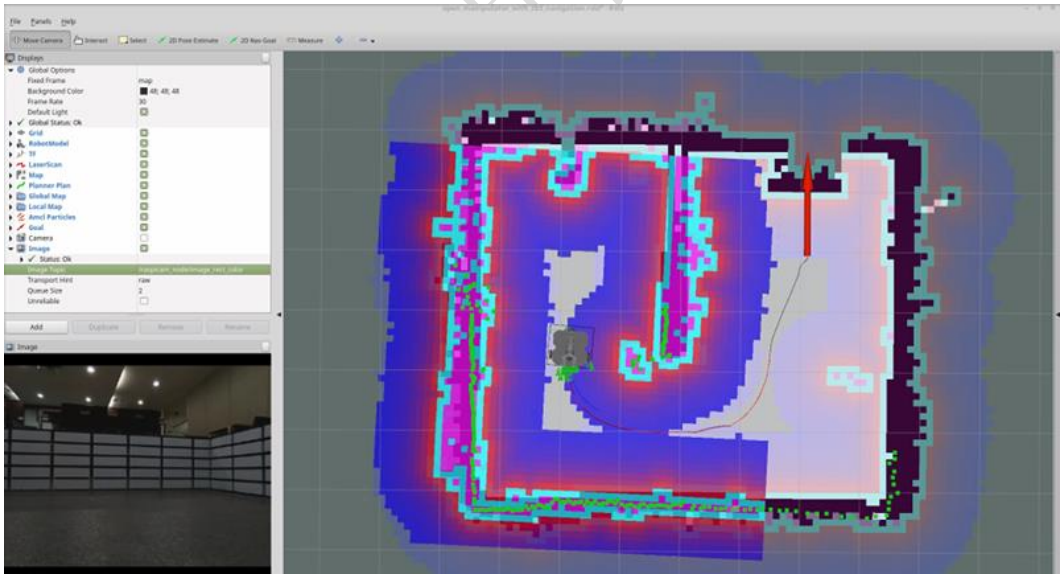
$ ROS_NAMESPACE=om_with_tb3 rosrun map_server map_saver -f ~/map



2.1.3 自主导航

$ export TURTLEBOT3_MODEL=${TB3_MODEL}

$ roslaunch open_manipulator_with_tb3_tools navigation.launch use_platform:=true

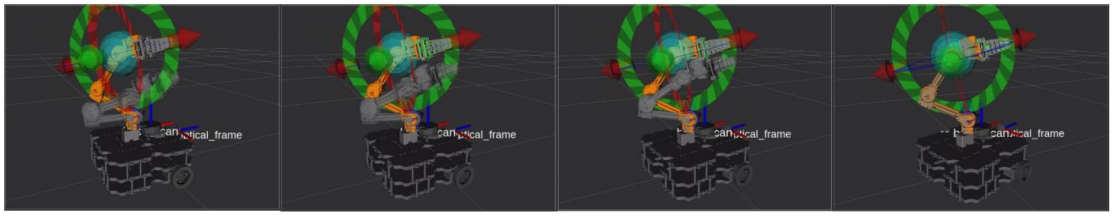


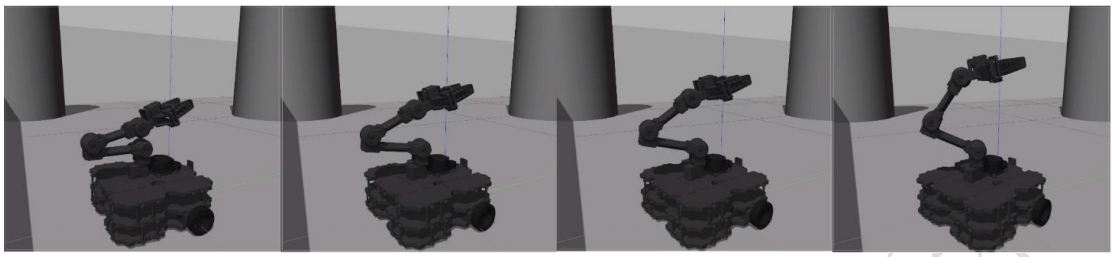
2.1.4 MoveIt！机械臂控制

为了运行 MoveIt！，打开一个新的终端窗口，然后在下面输入命令。

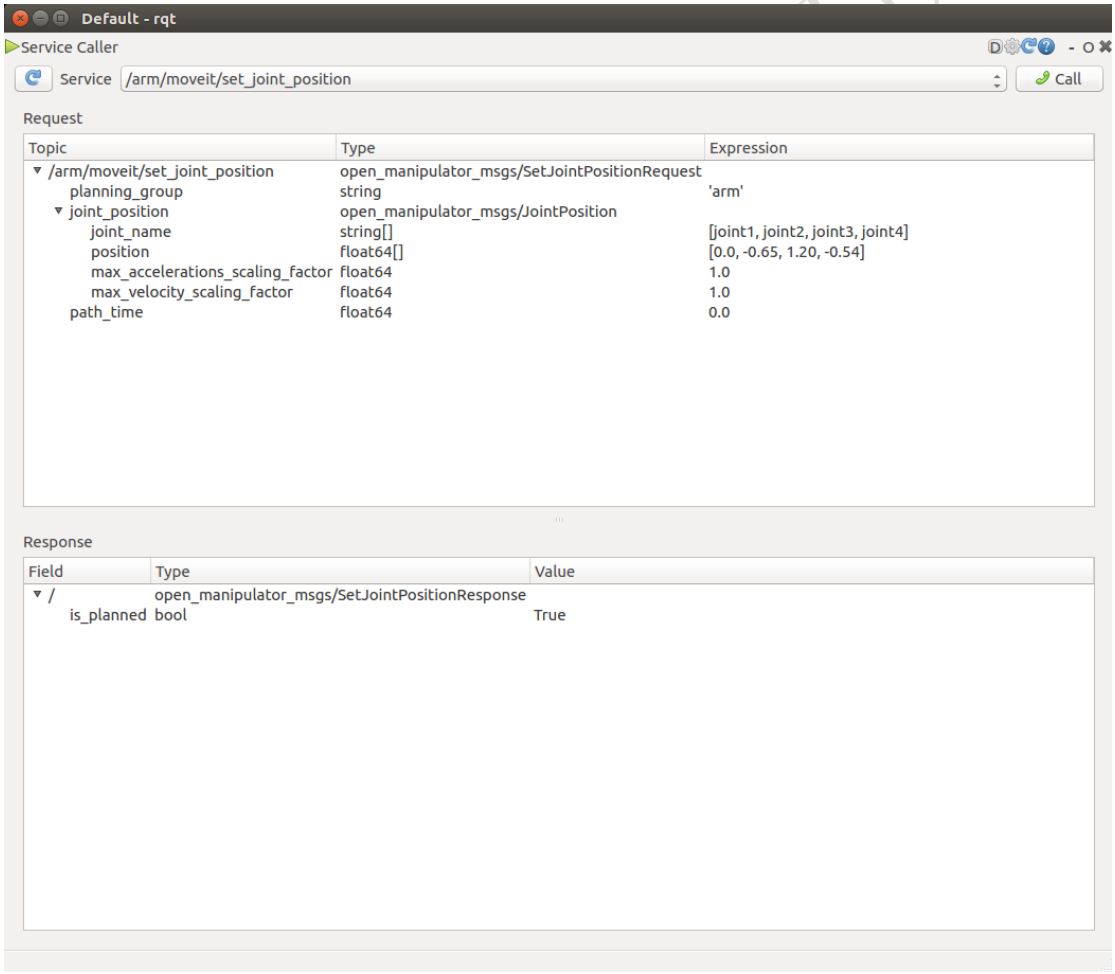
$ export TURTLEBOT3_MODEL=${TB3_MODEL}

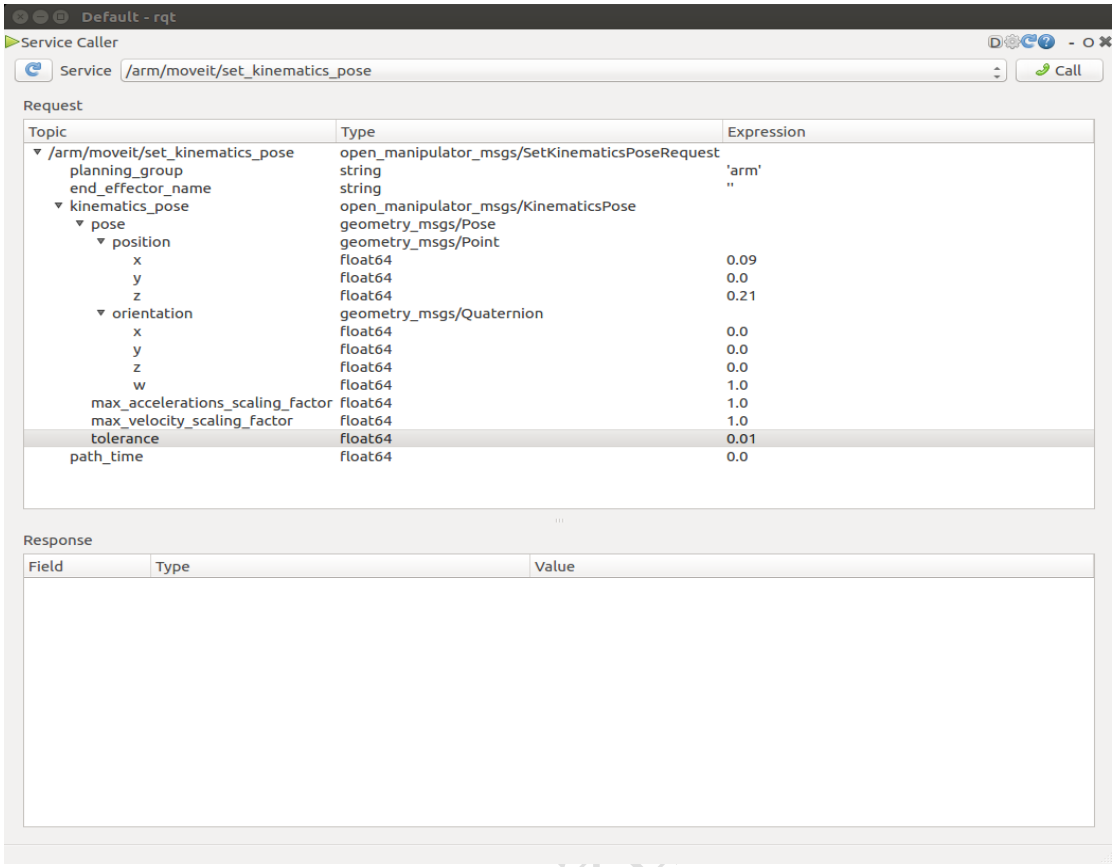
$ roslaunch open_manipulator_with_tb3_tools manipulation.launch use_platform:=true





rqt 插件下面显示了 OpenMANIPULATOR 控件的示例。





通过 rosservice 消息获取机械手状态。

```
$ rosservice call /arm/moveit/get_joint_position "planning_group: 'arm'"
joint_position:
  joint_name: [joint1, joint2, joint3, joint4]
  position: [-0.003067961661145091, -0.42644667625427246,
1.3084856271743774, -0.8452234268188477]
  max_accelerations_scaling_factor: 0.0
  max_velocity_scaling_factor: 0.0
$ rosservice call /arm/moveit/get_kinematics_pose "planning_group: 'arm'
end_effector_name: ""
header:
  seq: 0
  stamp:
    secs: 1550714737
    nsecs: 317547871
  frame_id: "/base_footprint"
kinematics_pose:
  pose:
    position:
      x: 0.0918695085861
      y: -0.000263644738325
```

```
        z: 0.218597669468
      orientation:
        x: 1.82347658316e-05
        y: 0.023774433021
        z: -0.000766773548775
        w: 0.999717054001
  max_accelerations_scaling_factor: 0.0
  max_velocity_scaling_factor: 0.0
  tolerance: 0.0
```

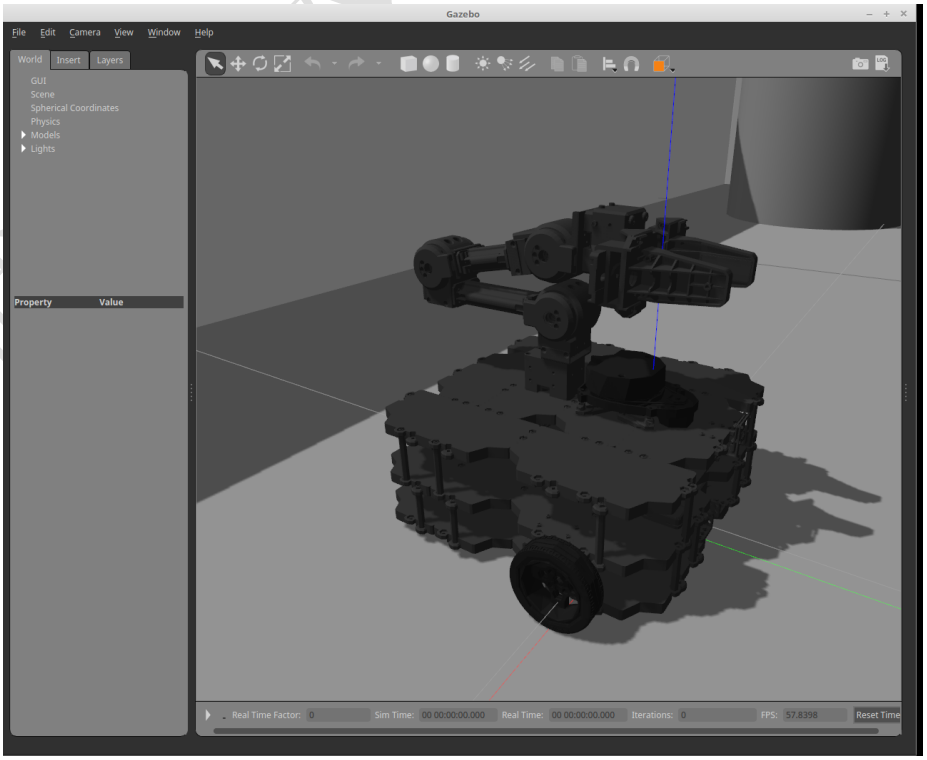
为了控制机械手（范围 -0.01∽0.01），下面的服务命令可能会有所帮助。

```
$ rosservice call /om_with_tb3/gripper "planning_group: "
joint_position:
  joint_name:
  - "
  position:
  - 0.15
  max_accelerations_scaling_factor: 0.0
  max_velocity_scaling_factor: 0.0
path_time: 0.0"
```

2.1.5 Gazebo 仿真

在 gazebo 模拟器上使用机械臂加载 TurtleBot3，然后单击 Play 按钮

```
$ export TURTLEBOT3_MODEL=${TB3_MODEL}
$ roslaunch open_manipulator_with_tb3_gazebo empty_world.launch
```



- 键入 `rostopic list` 以检查激活了哪个主题。

```
$ rostopic list
/clock
/gazebo/link_states
/gazebo/model_states
/gazebo/parameter_descriptions
/gazebo/parameter_updates
/gazebo/set_link_state
/gazebo/set_model_state
/gazebo_gui/parameter_descriptions
/gazebo_gui/parameter_updates
/gazebo_ros_control/pid_gains/gripper/parameter_descriptions
/gazebo_ros_control/pid_gains/gripper/parameter_updates
/gazebo_ros_control/pid_gains/gripper/state
/gazebo_ros_control/pid_gains/gripper_sub/parameter_descriptions
/gazebo_ros_control/pid_gains/gripper_sub/parameter_updates
/gazebo_ros_control/pid_gains/gripper_sub/state
/gazebo_ros_control/pid_gains/joint1/parameter_descriptions
/gazebo_ros_control/pid_gains/joint1/parameter_updates
/gazebo_ros_control/pid_gains/joint1/state
/gazebo_ros_control/pid_gains/joint2/parameter_descriptions
/gazebo_ros_control/pid_gains/joint2/parameter_updates
/gazebo_ros_control/pid_gains/joint2/state
/gazebo_ros_control/pid_gains/joint3/parameter_descriptions
/gazebo_ros_control/pid_gains/joint3/parameter_updates
/gazebo_ros_control/pid_gains/joint3/state
/gazebo_ros_control/pid_gains/joint4/parameter_descriptions
/gazebo_ros_control/pid_gains/joint4/parameter_updates
/gazebo_ros_control/pid_gains/joint4/state
/om_with_tb3/camera/parameter_descriptions
/om_with_tb3/camera/parameter_updates
/om_with_tb3/camera/rgb/camera_info
/om_with_tb3/camera/rgb/image_raw
/om_with_tb3/camera/rgb/image_raw/compressed
/om_with_tb3/camera/rgb/image_raw/compressed/parameter_descriptions
/om_with_tb3/camera/rgb/image_raw/compressed/parameter_updates
/om_with_tb3/camera/rgb/image_raw/compressedDepth
/om_with_tb3/camera/rgb/image_raw/compressedDepth/parameter_descripti
ons
/om_with_tb3/camera/rgb/image_raw/compressedDepth/parameter_updates
/om_with_tb3/camera/rgb/image_raw/theora
/om_with_tb3/camera/rgb/image_raw/theora/parameter_descriptions
```

```
/om_with_tb3/camera/rgb/image_raw/theora/parameter_updates
/om_with_tb3/cmd_vel
/om_with_tb3/gripper_position/command
/om_with_tb3/gripper_sub_position/command
/om_with_tb3/imu
/om_with_tb3/joint1_position/command
/om_with_tb3/joint2_position/command
/om_with_tb3/joint3_position/command
/om_with_tb3/joint4_position/command
/om_with_tb3/joint_states
/om_with_tb3/odom
/om_with_tb3/scan
/rosout
/rosout_agg
/tf
/tf_static
```

gazebo 中的机械臂由 ROS 消息控制。例如，要使用以下命令，请发布关节位置（弧度）。

$ rostopic pub /om_with_tb3/joint4_position/command std_msgs/Float64 "data: -0.21" --once

2.1.6 取放示例

我们提供了移动操作的选择示例。此示例用于 smach（任务级体系结构）将动作发送给机器人。

带上 gazebo 模拟器

$ roslaunch open_manipulator_with_tb3_gazebo rooms.launch use_platform:=false

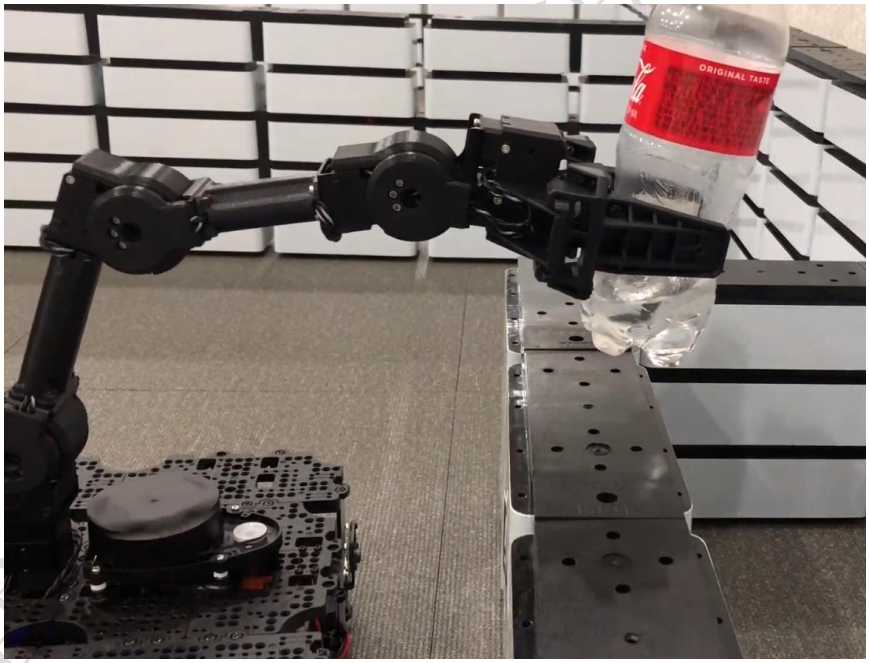
启动导航，moveIt！

$ roslaunch open_manipulator_with_tb3_tools rooms.launch use_platform:=false

启动任务控制器

$ roslaunch open_manipulator_with_tb3_tools task_controller.launch

提示：Smach 提供状态图。尝试运行 smach 查看器，以及如何机械手拾取和放置。`rosrun smach_viewer smach_viewer.py`



2.2 韭菜盒子二维码定位

韭菜盒子要定位二维码，首先需要求取二维码的坐标位置，本实验中采用 RGB 单目相机计算二维码位置，采用 ar_track_alvar 算法包进行运算。但由于每一个相机的制造误差使他们的内参外参都有一定程度的差异，因而本实验分两步，第一步先对相机的内外参作标定，第二部通过标定好的相机做视觉定位。

2.2.1 标定相机

首先插上 USB 相机，编辑相机发布配置文件：

$ roscd usb_cam/launch/
$ gedit calib.launch
输入以下内容

```
<launch>
  <node name="usb_cam" pkg="usb_cam" type="usb_cam_node" output="screen" >
    <param name="video_device" value="/dev/video0" />
    <param name="image_width" value="640" />
    <param name="image_height" value="480" />
    <param name="pixel_format" value="yuyv" />
    <param name="io_method" value="mmap"/>
  </node>
</launch>
```

然后运行标定程序
$ roslaunch usb_cam calib.launch
$ rosrun camera_calibration cameracalibrator.py --size 8x6 --square 0.03 image:=/usb_cam/image_raw camera:=/usb_cam
其中“--size 8x6”表示标定板每行 8 个内点，每列 6 个内点。“--square 0.03”表示标定板每个方格子边长为 0.03 米。
用标定板：

等待右边的 x y size skaw 都变绿，就可以点击 Calibration 运算，运算过程会非常的缓慢，电脑会卡住不动，稍安勿躁。
然后运算完成后，点“SAVE”，点“COMMIT”
然后我们在“~/.ros/camera_info/”得到了这个文件，它对我们后面的定位过程非常重要。
~/.ros/camera_info/head_camera.yaml
我们下一次运行 usb_cam 就会自动调用文件，并有如下的信息：
[INFO] [1555832606.313276288]: camera calibration URL: file:///home/sc/.ros/camera_info/head_camera.yaml

2.2.2 相机追踪

新建相机运行文件
$ roslaunch usb_cam usb_cam-test.launch
文件内容如下：

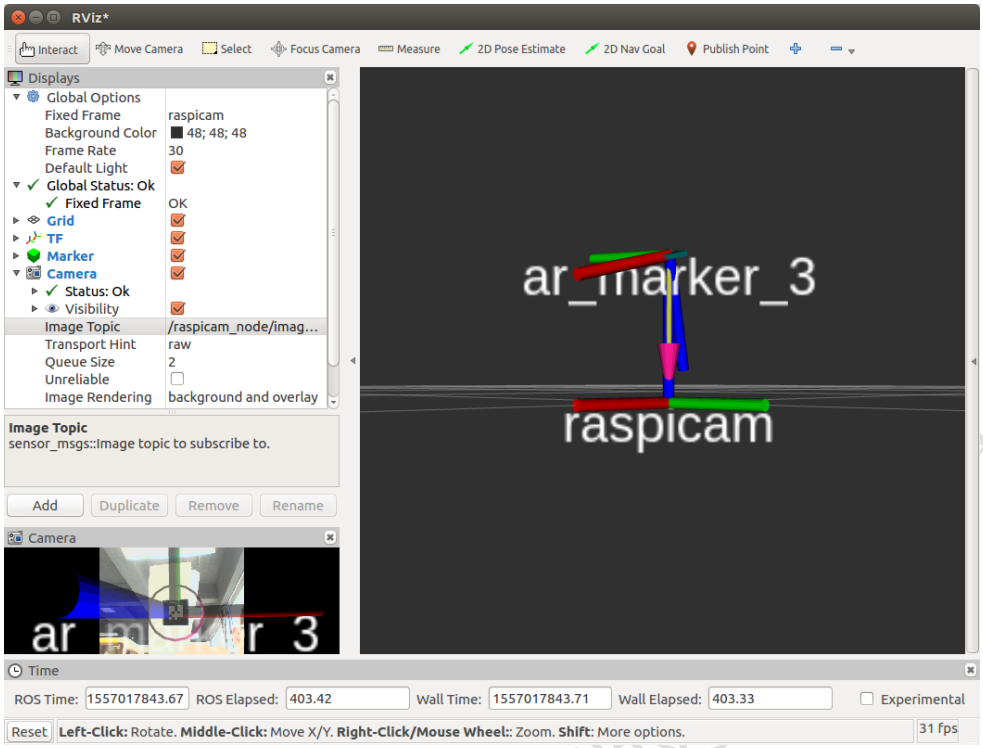
```
<launch>
  <node name="usb_cam" pkg="usb_cam" type="usb_cam_node" output="screen" >
    <param name="video_device" value="/dev/video0" />
    <param name="image_width" value="640" />
    <param name="image_height" value="480" />
    <param name="pixel_format" value="yuyv" />
    <param name="camera_frame_id" value="raspicam" />
      <remap from="/usb_cam/image_raw" to="/raspicam_node/image_raw"/>
    <param name="io_method" value="mmap"/>
```

```
  </node>
  <node name="image_view" pkg="image_view" type="image_view"
respawn="false" output="screen">
    <remap from="image" to="/raspicam_node/image_raw"/>
    <param name="autosize" value="true" />
  </node>
</launch>
```

编辑文件：
roslaunch usb_cam usb_cam_ar_tracker.launch
该内容的如下：

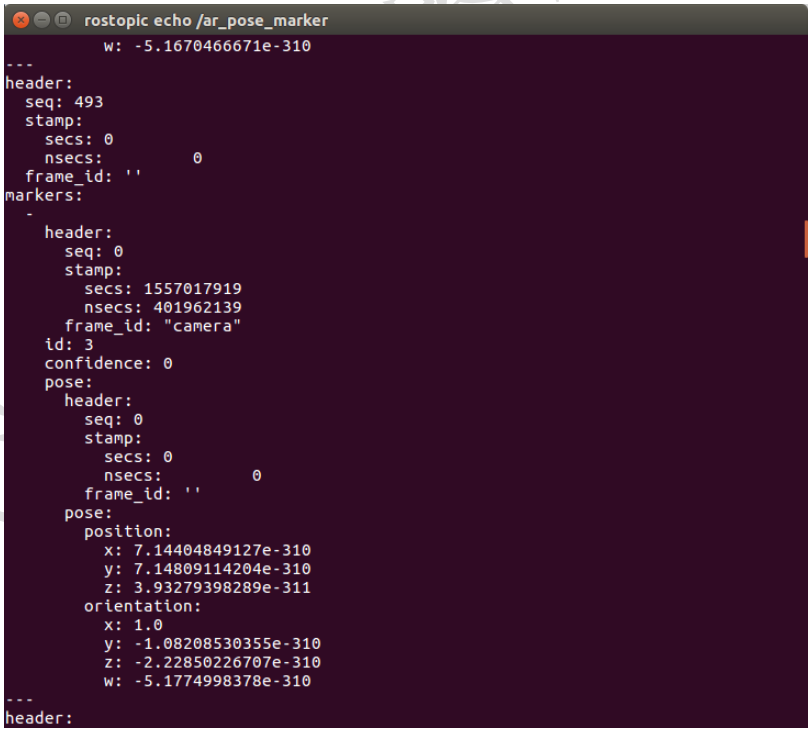
```
<launch>
    <arg name="marker_size"          default="5" />
    <arg name="max_new_marker_error" default="0.05" />
    <arg name="max_track_error"      default="0.05" />
    <arg name="cam_image_topic"      default="/raspicam_node/image_raw" />
    <arg name="cam_info_topic"       default="/raspicam_node/camera_info" />
    <arg name="output_frame"         default="/map" />
<node pkg ="tf2_ros" type ="static_transform_publisher" name ="map_camera" args
=" 0 0 0 0 0 0 1 /map /raspicam"/>
 <node name="ar_track_alvar" pkg="ar_track_alvar"
type="individualMarkersNoKinect" respawn="false" output="screen">
        <param name="marker_size"          type="double" value="$(arg
marker_size)" />
        <param name="max_new_marker_error"  type="double" value="$(arg
max_new_marker_error)" />
        <param name="max_track_error"       type="double" value="$(arg
max_track_error)" />
        <param name="output_frame"          type="string" value="$(arg
output_frame)" />
        <remap from="camera_image"  to="$(arg cam_image_topic)" />
        <remap from="camera_info"   to="$(arg cam_info_topic)" />
  </node>
</launch>
```

运行:
rosrun rviz rviz
并选择 FixedFrame 为 raspicam，然后点“add”添加 Mrker 主题。



如果想要获得二维码的位置数据值，可以在命令行运行

$ rostopic echo /ar_pose_marker



如果您使用的是树莓派相机，可以按照如下命令启动相机。

$ roscore

$ roslaunch turtlebot3_bringup turtlebot3_rpicamera.launch

$ roslaunch usb_cam usb_cam_ar_tracker.launch #内容如下

```
<launch>

    <arg name="marker_size"                 default="4.3" />
    <arg name="max_new_marker_error" default="0.05" />
    <arg name="max_track_error"           default="0.05" />

    <arg name="cam_image_topic"          default="/raspicam_node/image_raw" />
    <arg name="cam_info_topic"            default="/raspicam_node/camera_info" />
    <arg name="output_frame"              default="/camera" />
<!-- node pkg ="tf2_ros" type ="static_transform_publisher" name ="map_camera"
args =" 0 0 0 0 0 0 1 /map /camera"/ -->

    <node name="ar_track_alvar" pkg="ar_track_alvar"
type="individualMarkersNoKinect" respawn="false" output="screen">
        <param name="marker_size"                 type="double" value="$(arg
marker_size)" />
        <param name="max_new_marker_error"   type="double" value="$(arg
max_new_marker_error)" />
        <param name="max_track_error"          type="double" value="$(arg
max_track_error)" />
        <param name="output_frame"               type="string" value="$(arg
output_frame)" />

        <remap from="camera_image"    to="$(arg cam_image_topic)" />
        <remap from="camera_info"      to="$(arg cam_info_topic)" />
    </node>
<node name="republish" type="republish" pkg="image_transport" output="screen"
args="compressed in:=raspicam_node/image raw out:=raspicam_node/image_raw" />
</launch>
```

2.3 韭菜盒子机械臂操作

2.3.1 先启动 turtlebot3 节点

- [TurtleBot3]启动 rosserial 和激光雷达节点

```
$ export TURTLEBOT3_MODEL=wafflepi
$ ROS_NAMESPACE=om_with_tb3 roslaunch turtlebot3_bringup
turtlebot3_robot.launch multi_robot_name:=om_with_tb3
set_lidar_frame_id:=om_with_tb3/base_scan
```

- [TurtleBot3]启动 rpicamera 节点

```
$ ROS_NAMESPACE=om_with_tb3 roslaunch turtlebot3_bringup
turtlebot3_rpicamera.launch
```

➢ [Remote PC]启动启动 robot_state_publisher 节点

$ ROS_NAMESPACE=om_with_tb3 roslaunch open_manipulator_with_tb3_tools om_with_tb3_robot.launch

2.3.2 启动机械臂 MoveIt！

为了运行 MoveIt！，打开一个新的终端窗口并输入以下命令。

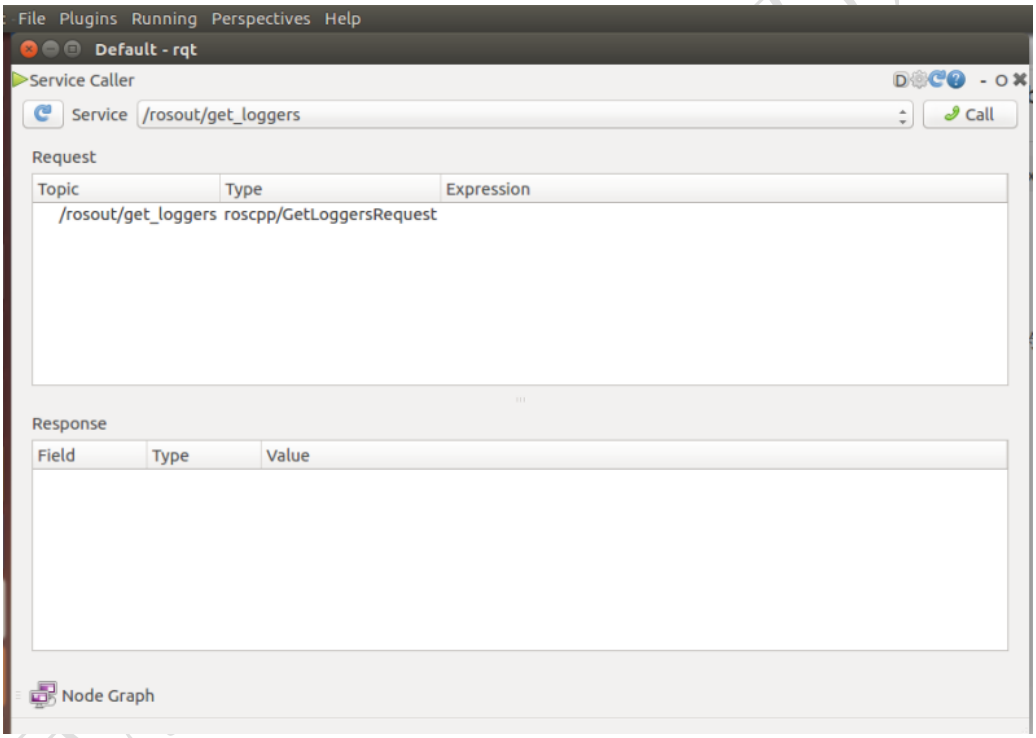
$ export TURTLEBOT3_MODEL=${TB3_MODEL}

$ roslaunch open_manipulator_with_tb3_tools manipulation.launch use_platform:=true

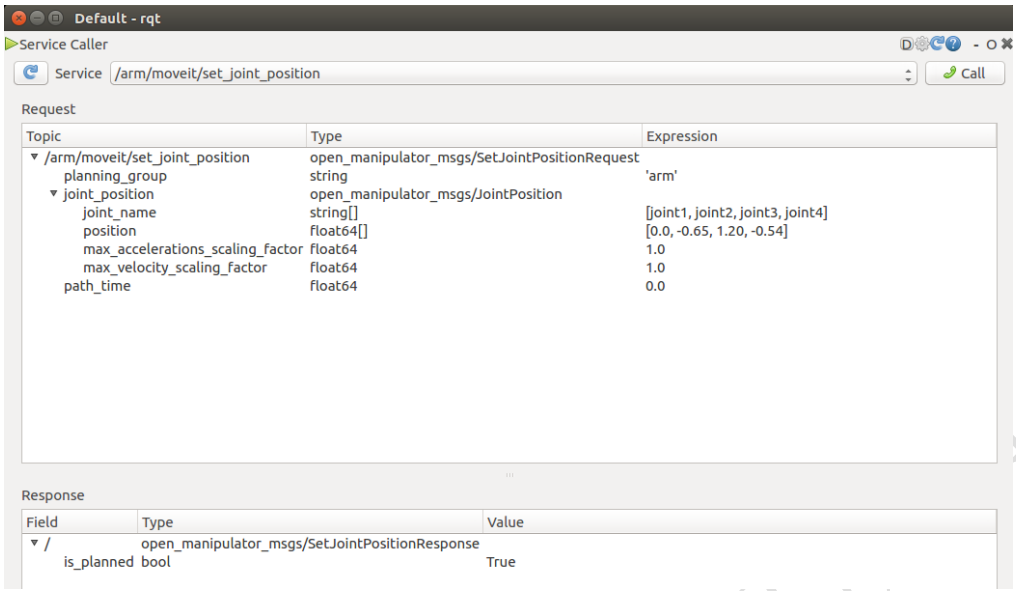
2.3.3 使用 rqt 控制机器人

开启一个新终端，运行：rqt

选择左上角菜单栏的 “Plugins”，下面的“Services”，选下面的“Service Caller”，出现如下面板：

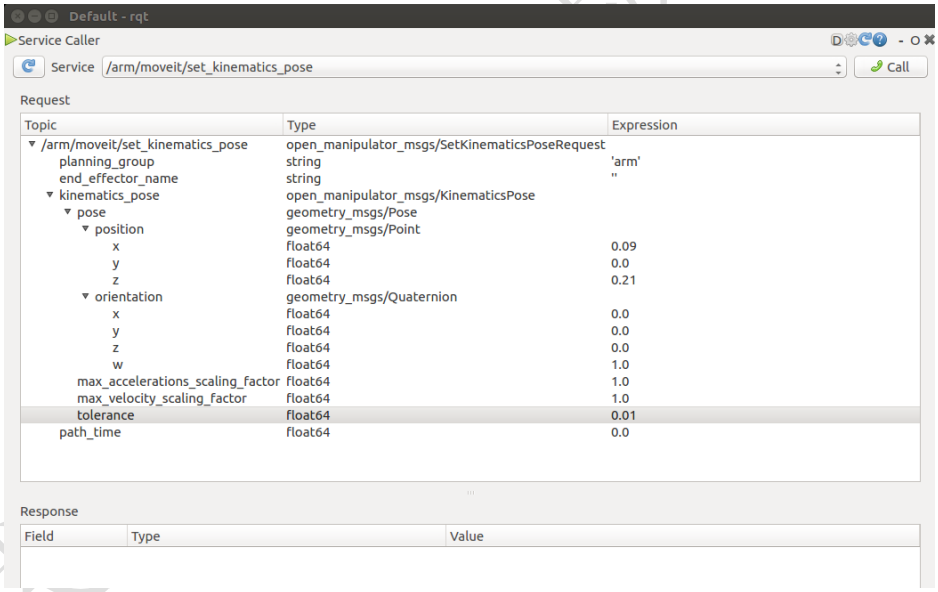


关节空间控制：将配置参数按照下面示例中填写：



完成后点击右上角的“Call”，如果前面的步骤配置正确，下面的 Response 中会显示“True”。

工作空间控制：选择对应的 Service，并将配置参数按照下面示例中填写：



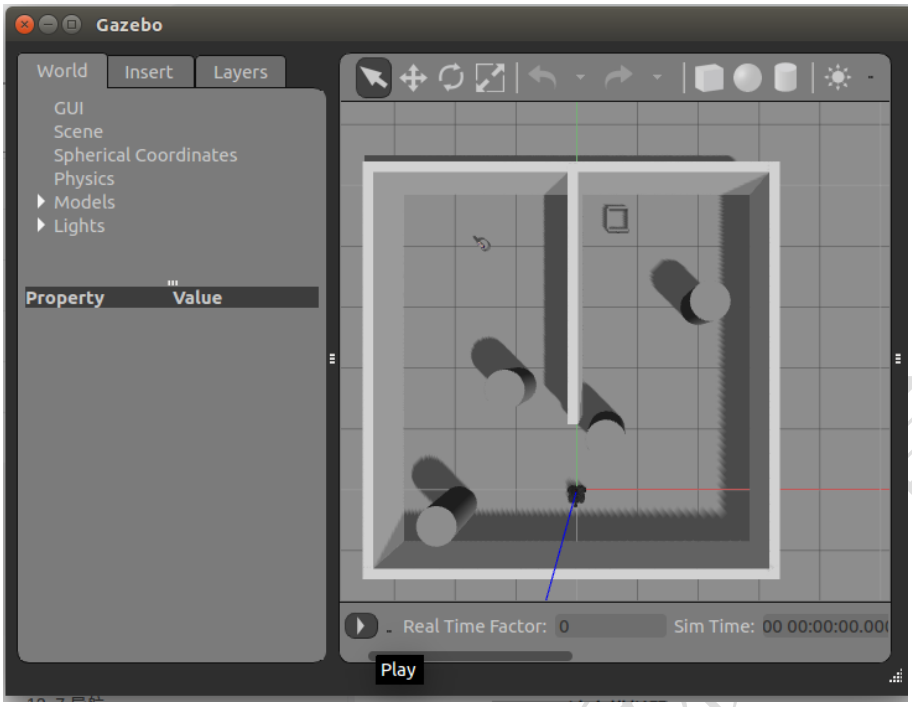
2.4 韭菜盒子移动抓取仿真示例

由于抓取示例需要根据地图来做位置修改，所以本示例采用 Gazebo 仿真环境中的地图。

发布 Gazebo 模拟器

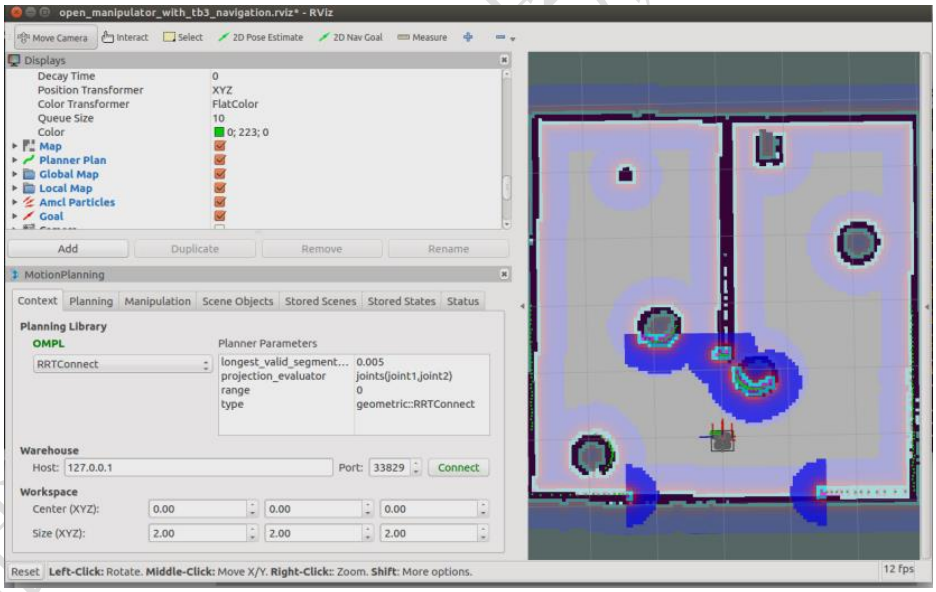
$ roslaunch open_manipulator_with_tb3_gazebo rooms.launch use_platform:=false

在 gazebo 中点击“play”按钮，开始仿真。



【PC】启动导航节点和机械臂节点

$ roslaunch open_manipulator_with_tb3_tools rooms.launch use_platform:=false

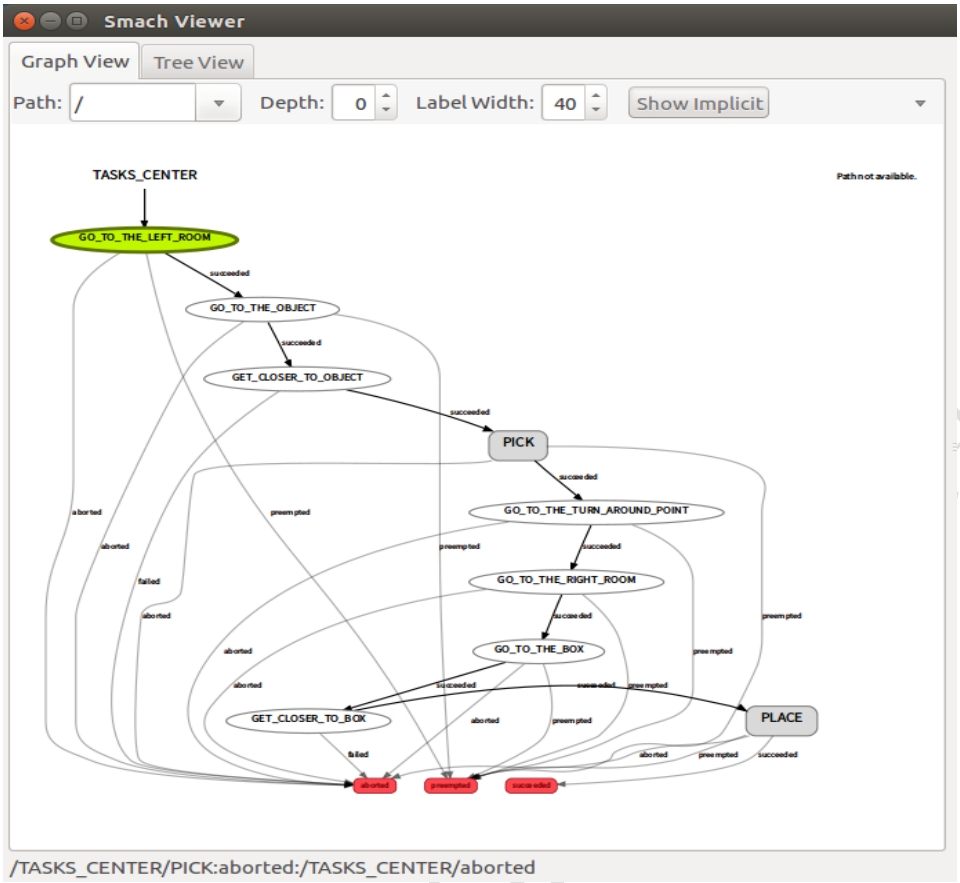


【PC】启动 Smatch 任务控制器

$ roslaunch open_manipulator_with_tb3_tools task_controller.launch

【可不运行】运行 smach viewer 查看机器人每隔任务状态之间如何转换

$ rosrun smach_viewer smach_viewer.py



2.5 京天例程：自动跟随二维码并抓取

2.5.1 配置二维码移动抓取包 auto_pick_sc

A、下载，编译

```
git clone https://github.com/JTDQ/Auto_Pick_QR_Object.git
cd catkin_ws/src
catkin_make
```

B、配置相机

这个文件仅作为试用，你应该根据你的相机，进行标定后的文件做替换。
复制 文件夹内相机标定文件 `head_camera.yaml` 文件到

`/home/sc/.ros/camera_info/head_camera.yaml`.

C、运行 (根据相机是'树莓派相机、USB 相机、JetsonTx2 的 Csi 相机，选择)
方案一，如果你用的是树莓派相机，
先在树莓派上运行：

```
roslaunch turtlebot3_bringup turtlebot3_rpicamera.launch
```

```
ROS_NAMESPACE=om_with_tb3 roslaunch turtlebot3_bringup
turtlebot3_robot.launch multi_robot_name:=om_with_tb3
set_lidar_frame_id:=om_with_tb3/base_scan
```

然后，在 PC 电脑上运行：

```
# turtlebot3，机械臂 moveit，ar_tracker
roslaunch auto_pick_sc raspicamera_ar_tracker.launch
# 任务控制节点
roslaunch auto_pick_sc sc_task_controller.launch
```

方案二：如果你是用 usb 相机

然后在远程 PC 上运行

```
# 启动相机、turtlebot3、机械臂 moveit、ar_tracker
$ roslaunch auto_pick_sc follow_ar.launch camera_type:=usb
# 启动’任务树管理‘节点
$ roslaunch auto_pick_sc sc_task_controller.launch
```

方案三：如果你是用 JetsonTX2 自带的 CSI

需要安装 TX2 板载 csi 相机的驱动 `gscam`

If you'd like to learn more about `gscam`, check out their ROS wiki page or

their Github repository.

Note: This package was tested on a Nvidia Jetson TX2 with L4T R27.1, ROS

Kinetic, and the Leopard Imaging IMX377CS CSI camera.

```
# 下载
$ cd ~/catkin_ws/src
$ git clone https://github.com/peter-moran/jetson_csi_cam.git

# 编译
$ cd ~/catkin_ws/
$ catkin_make
```

运行

```
# 启动相机、turtlebot3、机械臂 moveit、ar_tracker
$ roslaunch auto_pick_sc follow_ar.launch camera_type:=jetson
# 启动’任务树管理‘节点
$ roslaunch auto_pick_sc sc_task_controller.launch
```

2.5.2 移动抓取包内文件解释

1.follow_ar.launch

包括 usb_cam_ar_tracker.launch，turtlebot3_robot.launch: TB3 启动文件，manipulation.launch：机械臂 Moveit 启动。

a.usb_cam_ar_tracker.launch

这个里面配置了 USB 摄像头节点 usb_cam，二维码的节点 ar_track_alvar

2.sc_task_controller.launch

主要是包括一个 Python 脚本 pick_and_place_state_machine_0310.py，python 脚本加入到工作空间中，一定要给可执行权限。

3.pick_and_place_state_machine_0310.py

这个代码是 smach 任务节点管理的代码。

https://github.com/JTDQ

获取更多相关资料请参考京天电器 github 链接，会不定期更新最新的韭菜盒子 demo 供大家参考学习。