

● 概述

zju_bin_detect 里包含了两个节点，一个是做料盒检测的主节点，另一个是料盒检测功能的调用例程：

源码文件	作用
bin_dim_node.cpp	料盒检测功能主节点
bin_detect_demo.cpp	调用检测功能的例子程序

这两个节点之间的通讯，主要通过两个主题：

主题名称	作用
/box/param	发送检测参数给 bin_dim_node 节点，启动料盒检测
/box/pose	bin_dim_node 节点返回的料盒检测结果

● 示例程序

通过例子程序 bin_detect_demo.cpp，可以了解启动料盒检测只需要两步：
(一) 定义一个 ROS 主题的发布对象，将需要检测的料盒参数发送到主题“/box/param”；
(二) 订阅主题 “/box/pose” 获取料盒检测结果；
bin_detect_demo.cpp 代码如下：

```
#include <ros/ros.h>
#include <geometry_msgs/Pose.h>
#include <sensor_msgs/JointState.h>
#include <zju_bin_detect/BoxParam.h>

static ros::Publisher box_param_pub;
static zju_bin_detect::BoxParam box_param_msg;
static geometry_msgs::Pose box_pose;

void BoxPoseCallback(const geometry_msgs::Pose::ConstPtr &msg)
{
    // 盒子坐标
    box_pose = *msg;
    ROS_WARN("[BoxPose] = (%.2f , %.2f , %.2f)", box_pose.position.x,
box_pose.position.y, box_pose.position.z);
}

int main(int argc, char** argv)
{
    ros::init(argc, argv, "bin_detect_demo");

    // 发布主题
    ros::NodeHandle n;
```

```

box_param_pub = n.advertise<zju_bin_detect::BoxParam>("/box/param", 2);
ros::Subscriber box_pose_sub = n.subscribe("/box/pose", 1, BoxPoseCallback);

sleep(1);

box_param_msg.z = 0.6;           //盒子可能的高度（单位：米）
box_param_msg.color = 0;         //0-绿色盒子  1-黄色盒子
box_param_pub.publish(box_param_msg);

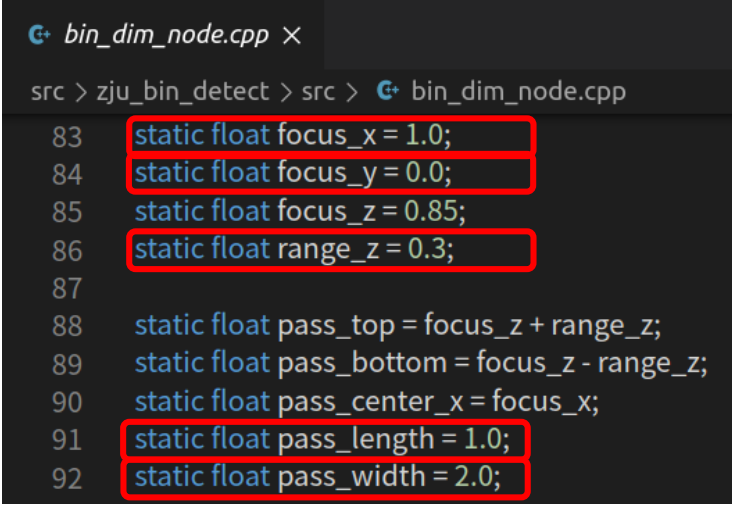
ros::spin();

return 0;
}

```

在这个程序里，进行了如下操作：

- (一) 定义了发布对象 `box_param_pub`，并在 `main` 函数里将发布主题设为“`/box/param`”，用于发送检测参数，启动料盒检测功能；
- (二) 定义了一个回调函数 `BoxPoseCallback()`，用于获取料盒检测结果，参数 `msg` 里就包含了料盒的位置信息。这个函数会通过 `main` 函数里的订阅操作关联到主题“`/box/pose`”上，这样每次返回料盒检测结果，都会自动调用这个函数。我们只需要在这个回调函数里对 `msg` 里的数据进行读取即可；
- (三) 在 `main` 函数里，对检测参数消息包 `box_param_msg` 进行赋值，主要是料的大致高度和要检测的料盒颜色。这两个参数属于容易变化比较大的，另外一些相对固定的参数在 `bin_dim_node.cpp` 中进行调节：



```

bin_dim_node.cpp x
src > zju_bin_detect > src > bin_dim_node.cpp
83 static float focus_x = 1.0;
84 static float focus_y = 0.0;
85 static float focus_z = 0.85;
86 static float range_z = 0.3;
87
88 static float pass_top = focus_z + range_z;
89 static float pass_bottom = focus_z - range_z;
90 static float pass_center_x = focus_x;
91 static float pass_length = 1.0;
92 static float pass_width = 2.0;

```

上述参数通常情况下不需要调整，除非遇到特别特殊的情况，主要参数为：

参数变量	意义
focus_x	点云处理范围的前后距离的中心，比如这里赋值 1.0，意思是所处理的点云会以 Kinect 前方 1.0 米距离左右作为中心。
focus_y	点云处理范围的左右偏移的中心，比如这里赋值 0.0，意思是所处理的点云会以 Kinect 前方正中心为中心。与 focus_x 结合起来，就可以限定 Kinect 处理点云的范围中心点。另外扩展的范围由下面的 pass_length 和 pass_width 来确定。

range_z	用来设置处理点云的高度范围：以 box_param_msg.z 传递过来的数值为中值，额外设置的上下限范围。比如 box_param_msg.z 为 0.6，则所处理点云的上限高度为 0.6+range_z，下限高度为 0.6-range_z。超出这个上下限高度范围的点云会被抛弃不做处理。
pass_length	点云处理的前后范围，比如这里赋值 1.0，意思就是点云处理范围的前后距离为 1.0 米。结合上面的 focus_x，可以知道点云的处理范围为（focus_x - pass_length / 2， focus_x + pass_length / 2）即（1.0-1.0/2， 1.0+1.0/2）=（0.5,1.5）。也就是只处理 Kinect2 正前方距离 0.5 米到 1.5 米范围内的点云，超出这个范围的点云会被抛弃不做处理。
pass_width	点云处理的左右范围，和 pass_length 作用类似。这里赋值 2.0，结合上面 focus_y 数值，可以知道点云处理的左右范围为（-1.0,1.0）。也就是只处理 Kinect2 正前方左侧 1.0 米到右侧 1.0 米范围内的点云，超出这个范围的点云会被抛弃不做处理。

这几个参数修改后，需要重新 catkin_make 编译才能生效。

● 源码下载

这个例子程序在 Github 上有相应的包，网址是：

https://github.com/zju-g/zju_bin_detect

需要将这个 zju_bin_detect 包 clone 到 catkin 工作空间里，运行 catkin_make 编译：

```
cd ~/catkin_ws/src
git clone https://github.com/zju-g/zju_bin_detect.git
cd ~/catkin_ws
catkin_make
```

● 参数标定

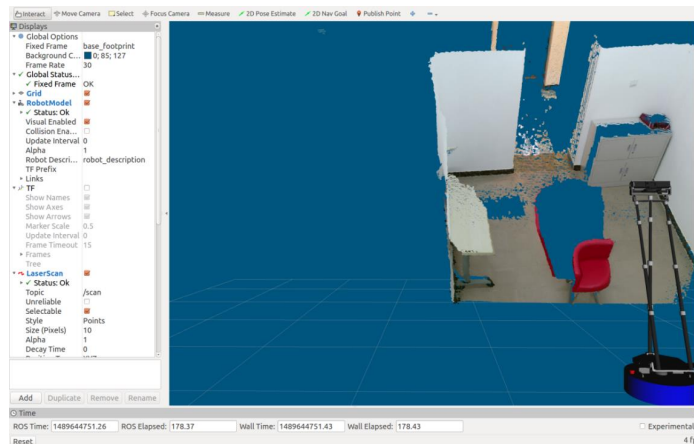
在运行料盒检测功能之前，需要对 Kinect 的高度和视角参数进行一个标定，步骤如下：

(1) 给 Kinect2 通上电，连接到电脑的 USB 口，面朝一片地面平整的开阔空间。

(2) 运行如下指令启动 Kinect2 的数据采集和点云显示功能：

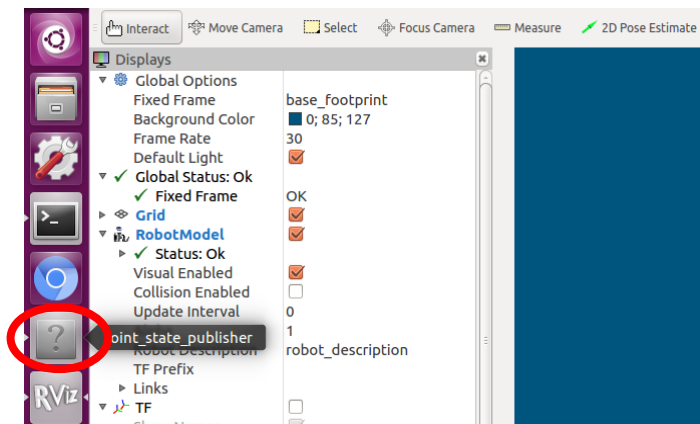
```
roslaunch zju_bin_detect bin_detect.launch
```

顺利的情况下会看到如下界面：



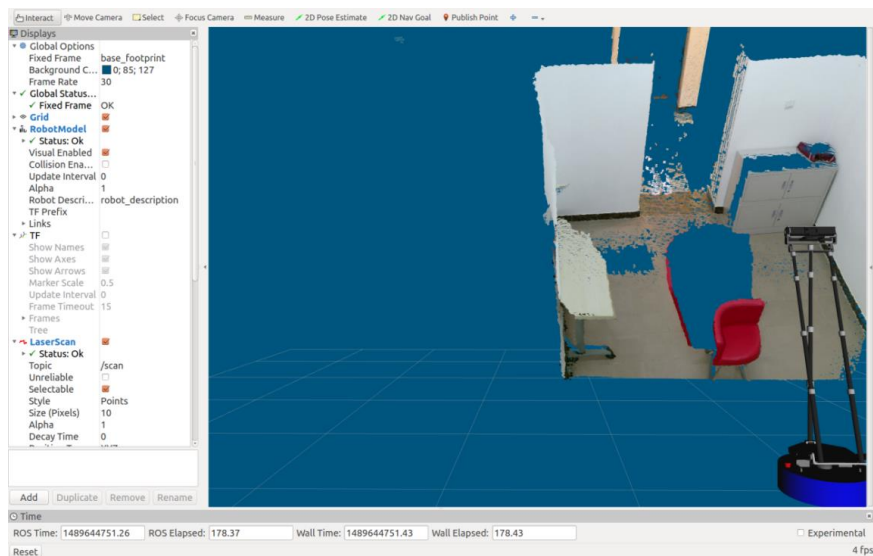
因为 Kinect2 的视角参数还未调整,所以看到的点云有可能和上面的图一样,是倾斜的。下面开始进行标定。

- (3) 保持 Rviz 里 Kinect2 点云里持续出现大量地面的三维点, 且处于不断刷新状态。
- (4) 从 Ubuntu 桌面左侧的启动栏里点击“Joint State Publisher”的图标, 弹出滑条窗口。

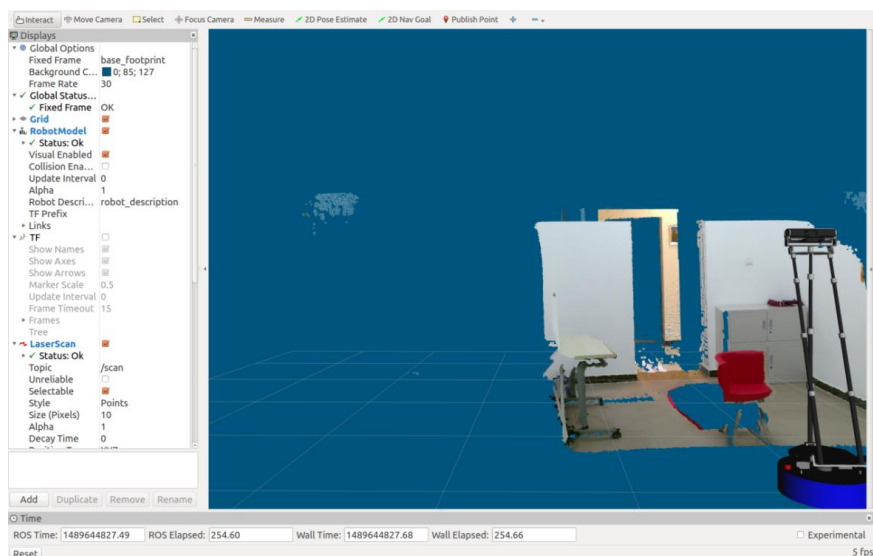


拖动滑条窗口上的“kinect_height”和“kinect_pitch”滑块, 观察 Rviz 里的 Kinect2 的实时点云。调整到合适的参数, 使得点云成像的地面部分和 Rviz 里的基准栅格贴合。

- 调节前地面点云和基准栅格不重合:



- 调节完成后，地面点云与基准栅格完全重合：



- (5) 记下地面贴合时“kinect_height”和“kinect_pitch”的数值。打开 Visual Studio Code，将数值填写到 catkin_ws 目录下的/src/zju_bin_detect/config/wpb_home.yaml 文件里。

```
! wpb_home.yaml x
src > zju_bin_detect > config > ! wpb_home.yaml
1  zeros:
2  kinect_height: 1.37
3  kinect_pitch: -0.50
4
```

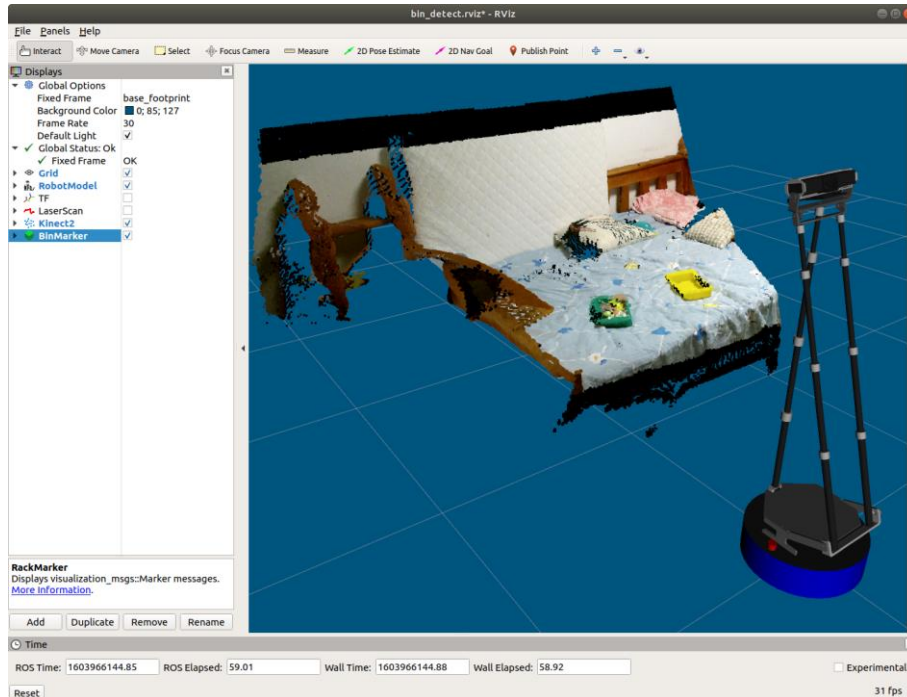
- (6) 关闭 Rviz，在终端程序里再次输入 `roslaunch zju_bin_detect bin_detect.launch` 指令再次启动 Rviz 查看 Kinect2 输出的三维点云。若设置成功，则 Rviz 启动后无需再通过滑块调节“kinect_height”和“Kinect_pitch”就可以显示已经贴合地面基准栅格的点云。

● 运行测试

(一) 将料盒放置在 Kinect2 的前面 2 米距离内,料盒高度和 bin_detect_demo.cpp 的 main 函数里 box_param_msg.z 大致接近。

(二) 运行 bin_dim_node 主节点、Kinect2 节点以及 Rviz 界面:

```
roslaunch zju_bin_detect bin_detect.launch
```



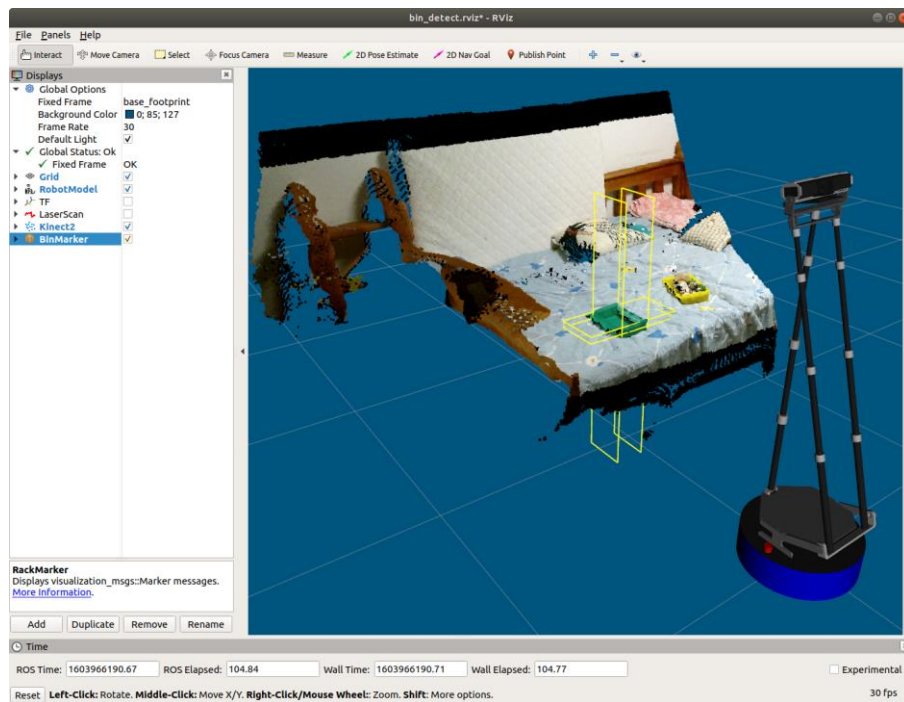
(三) 运行功能调用的例子程序。保持上面的 Rviz 界面别关闭,另外开一个终端,输入:

```
roslaunch zju_bin_detect bin_detect_demo
```

运行后,料盒检测功能被激活,终端里可以看到料盒检测的结果信息:



切换回 Rviz, 可以看到几个黄色矩形把料盒位置标注出来:



需要注意的是，每次调用 `box_param_pub.publish(box_param_msg)` 激活料盒检测之后，`bin_dim_node` 都会重新搜索新的料盒并锁定。所以要控制好 `publish()` 的时机，一旦目标料盒被锁定后，就不要再 `publish()` 新的料盒参数了，不然会激活新的料盒搜索操作，导致之前锁定的料盒被解除锁定。