动态窗口法局部规划仿真

github源码地址为：

https://github.com/bit-ivrc/ros\_navigation\_stack\_vrep\_simulation.git

# 1 环境配置

* 打开终端，输入

git clone https://github.com/bit-ivrc/ros\_navigation\_stack\_vrep\_simulation.git && cd ros\_navigation\_stack\_vrep\_simulation

* 如果没有安装ROS，输入以下命令安装ROS

./scripts/install\_ros.sh

* 如果没有安装v-rep，输入以下命令安装v-rep

./scripts/install\_vrep.sh

* 配置v-rep和ROS的通信

./scripts/install\_vrep\_ros\_interface.sh

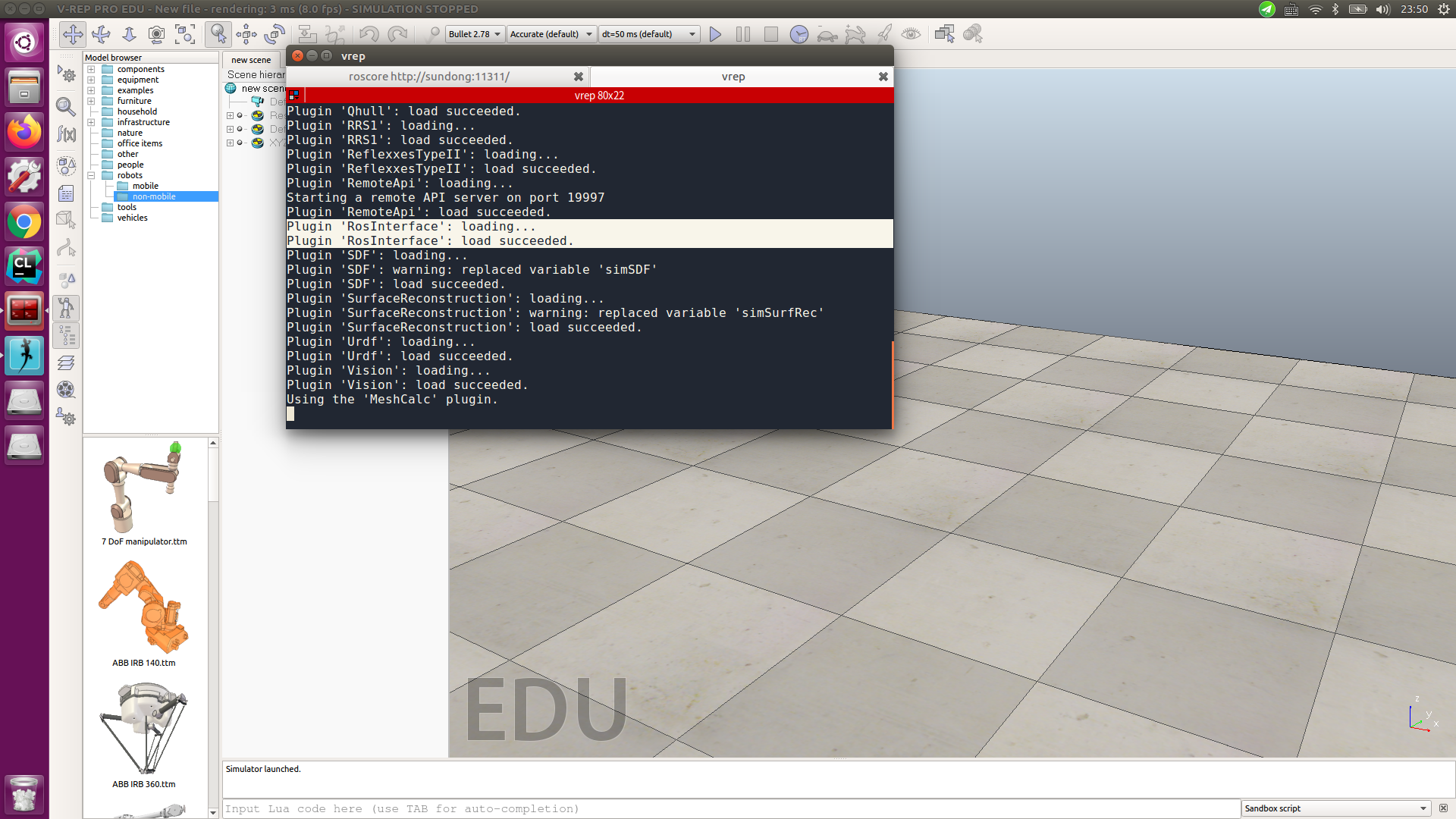
运行ROS

roscore

新开另一终端，运行v-rep

vrep

如果通信配置成功，将会出现“‘RosInterface’:loading succeeded.”如图1所示。

图1 通信配置成功

关闭程序。

* 配置其他依赖库

sudo apt-get install ros-kinetic-bfl

sudo apt-get install ros-kinetic-move-base

sudo apt-get install libbullet-dev libsdl1.2-dev libsdl-image1.2-dev

# 2 编译程序

在 ros\_navigation\_stack\_vrep\_simulation目录下打开终端,输入：

catkin build

如果出现报错，一般是依赖库没有安装完整，请根据提示安装对应的依赖库。

# 3 运行仿真

打开终端，输入：

roscore

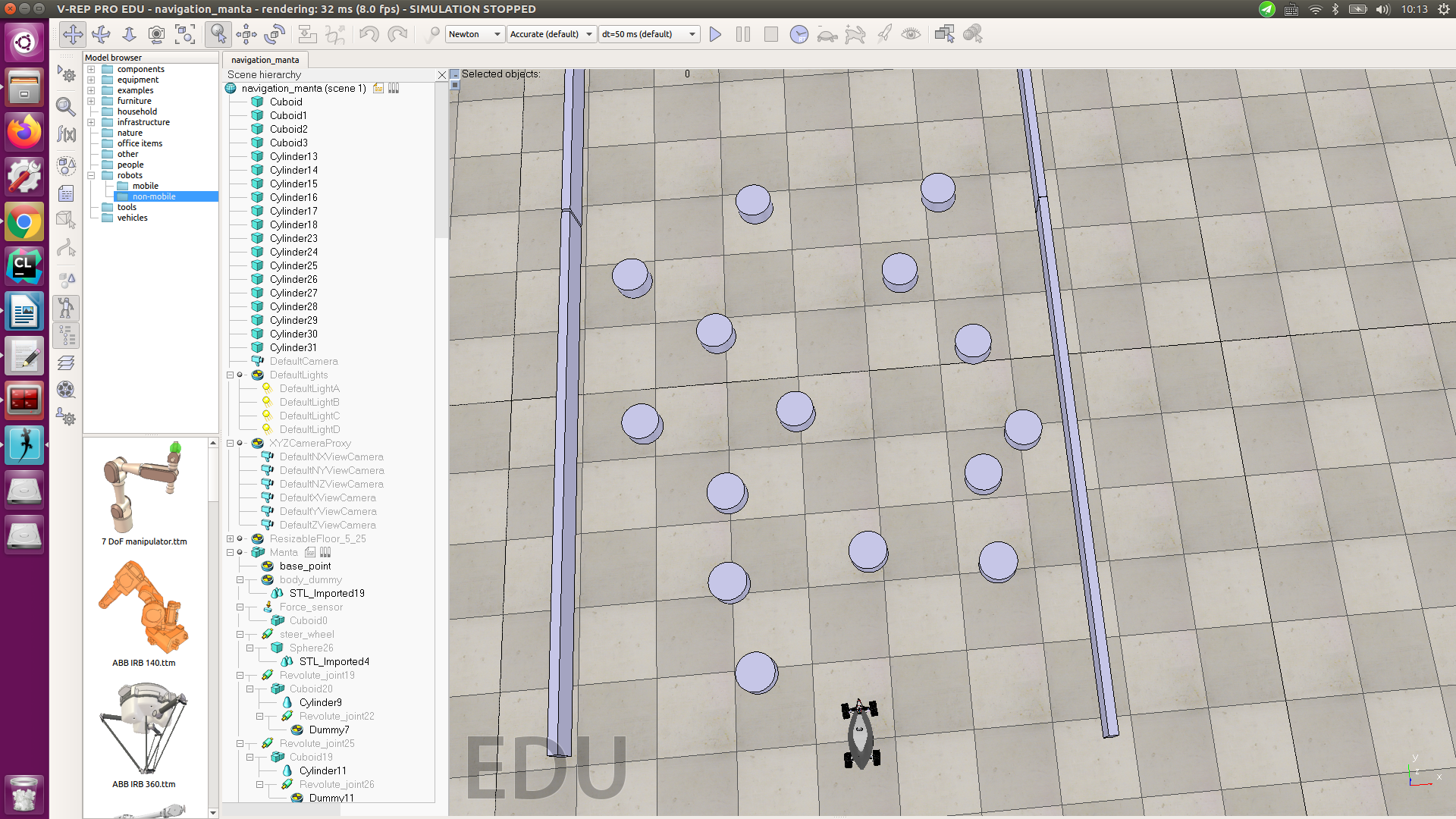
新开另一终端，输入：

vrep

**注意：**一定要先打开roscore再打开vrep。

此时可以看到vrep软件被打开，选择File→Open sence，打开ros\_navigation\_stack\_vrep\_simulation/scene文件夹下的navigation\_manta.ttt模型文件

可以看到出现图2所示的仿真场景：

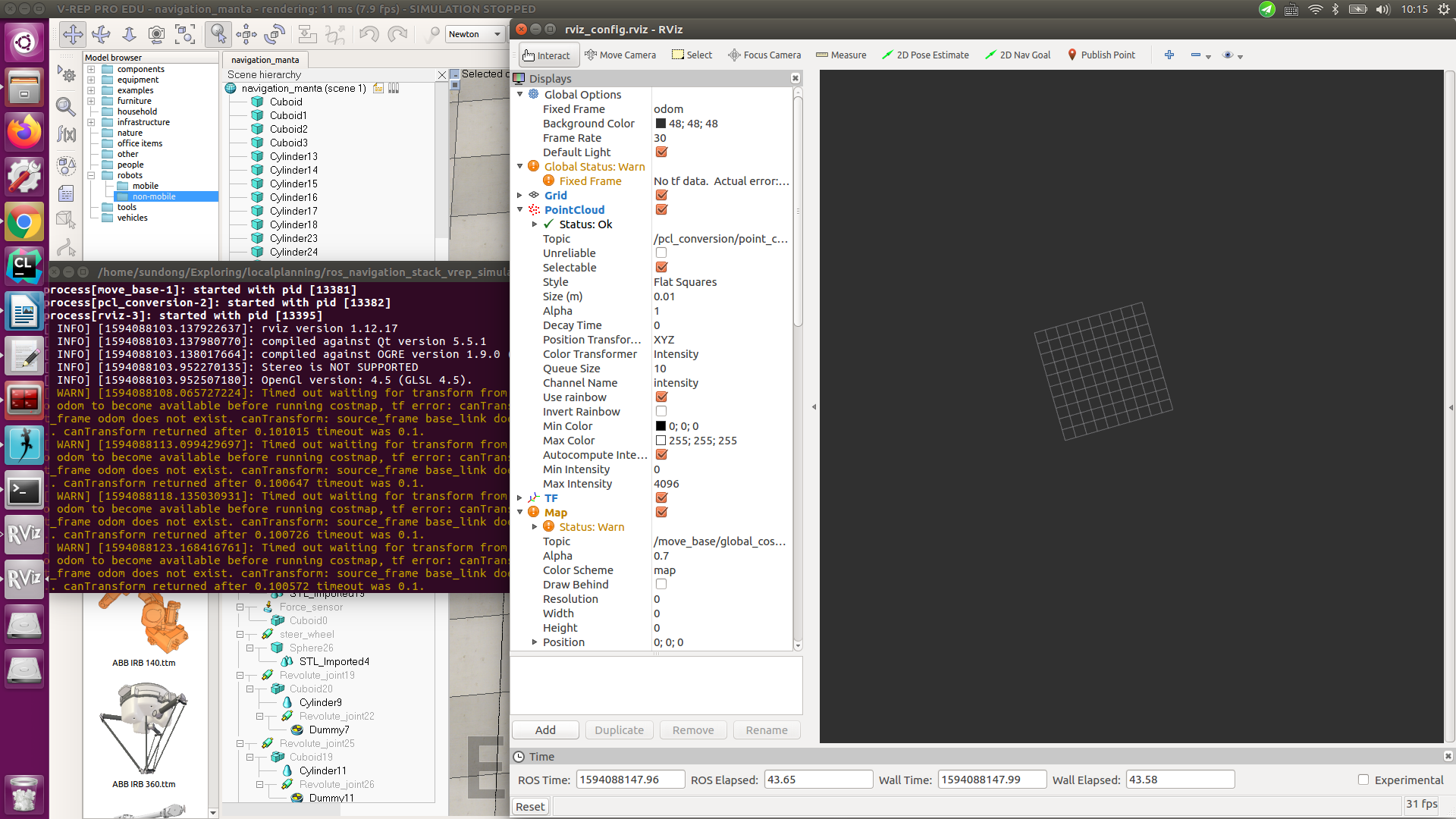
图2 vrep仿真场景

在 ros\_navigation\_stack\_vrep\_simulation目录下打开终端,输入：

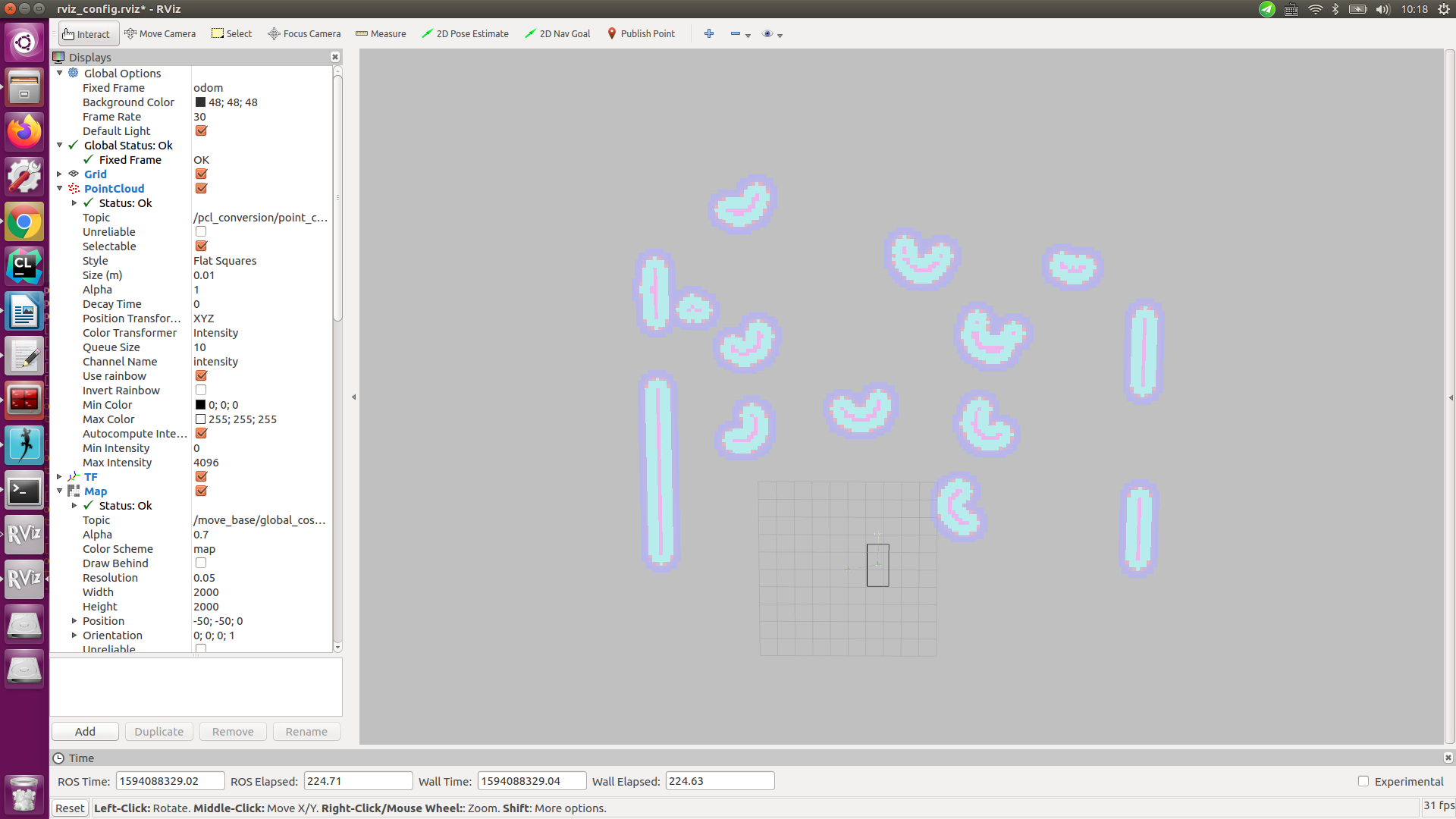
source devel/setup.bash

roslaunch move\_base move\_base.launch

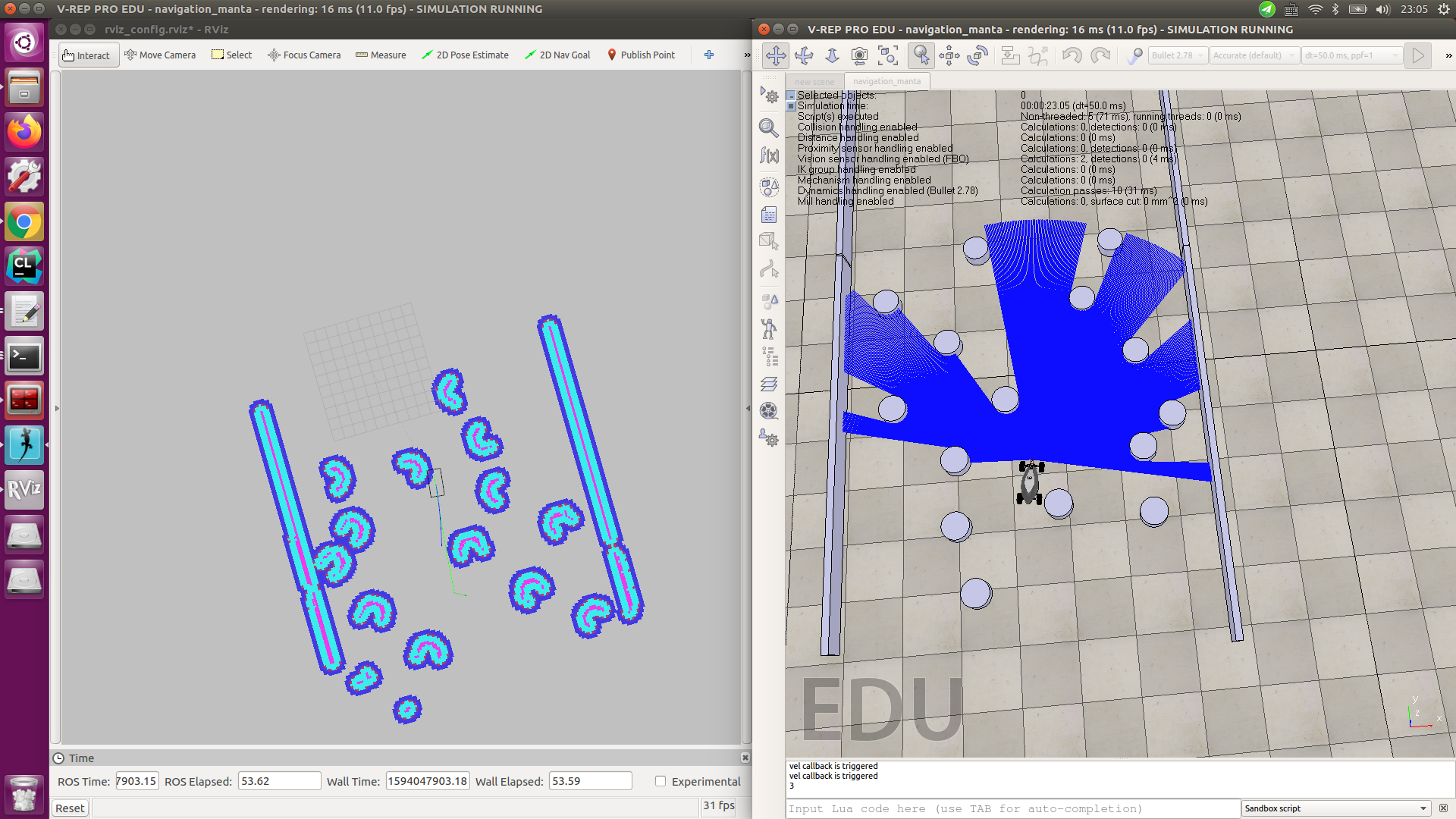
可以看到rviz被打开了，如图3所示

图3 运行程序，打开rviz

在v-rep界面中点击上方三角形按钮，启动仿真，就可以看到rviz中出现了障碍物和车辆信息，如图4所示

图4 rivz显示障碍物信息

点击rviz中2D Nav Goal，在地图中点击并移动鼠标来设置终点，可以看到仿真车辆在向目标点前进，如图5所示

图5 车辆向目标点前进

至此，仿真完成。

# 4 参数调节

仿真在运行前需要四个配置文件，在ros\_navigation\_stack\_vrep\_simulation/src/Navigation/move\_base/cfg/文件夹中，这些文件定义了一系列相关参数，包括越过障碍物的代价、车俩的尺寸、路径规划时需要考虑未来多长的路、车辆的移动速度等。这四个配置文件分别是：

base\_local\_planner\_params.yaml

costmap\_common\_params.yaml

global\_costmap\_params.yaml

local\_costmap\_params.yaml

下面分别介绍每个配置文件的主要内容。

* base\_local\_planner\_params.yaml

max\_vel\_x、min\_vel\_x：车辆的速度范围；

max\_vel\_theta：最大角速度；

acc\_lim\_theta：角加速度上限；

acc\_lim\_x：纵向加速度上限；

acc\_lim\_y：横向加速度上限；

sim\_time：采样时间；

* costmap\_common\_params.yaml

obstacle\_range：传感器探测范围，即障碍物更新范围；

raytrace\_range：障碍物清除范围，代价地图将清除指定范围外的障碍物；

footprint：车辆向地面投影的四个角点；

inflation\_radius：车辆面积膨胀尺寸；

observe\_sources：把信息传递给代价地图的传感器列表；

* global\_costmap\_params.yaml

global\_frame：代价地图应该运行的坐标系；

robot\_base\_frame：车辆的坐标系；

update\_frequency：代价地图更新的频率（以Hz为单位）；

static\_map：确定是否由map\_server提供的地图服务来进行代价地图的初始化；

width、height：全局代价地图的宽和高；

origin\_x、origin\_y：坐标系原点的位置；

* local\_costmap\_params.yaml

global\_frame、robot\_base\_frame、update\_frequency和static\_map参数与上述“global\_costmap\_params.yaml”部分中描述的相同；

publish\_frequency：确定代价地图发布可视化信息的速率（以Hz为单位）；

rolling\_window：当机器人移动时，是否保持机器人在本地代价地图中心；

width、height：局部代价地图的宽和高；

resolution：代价地图的分辨率。

# 备注

本仿真引用了ROS的move\_base包，源代码在

<https://github.com/ros-planning/navigation.git>。

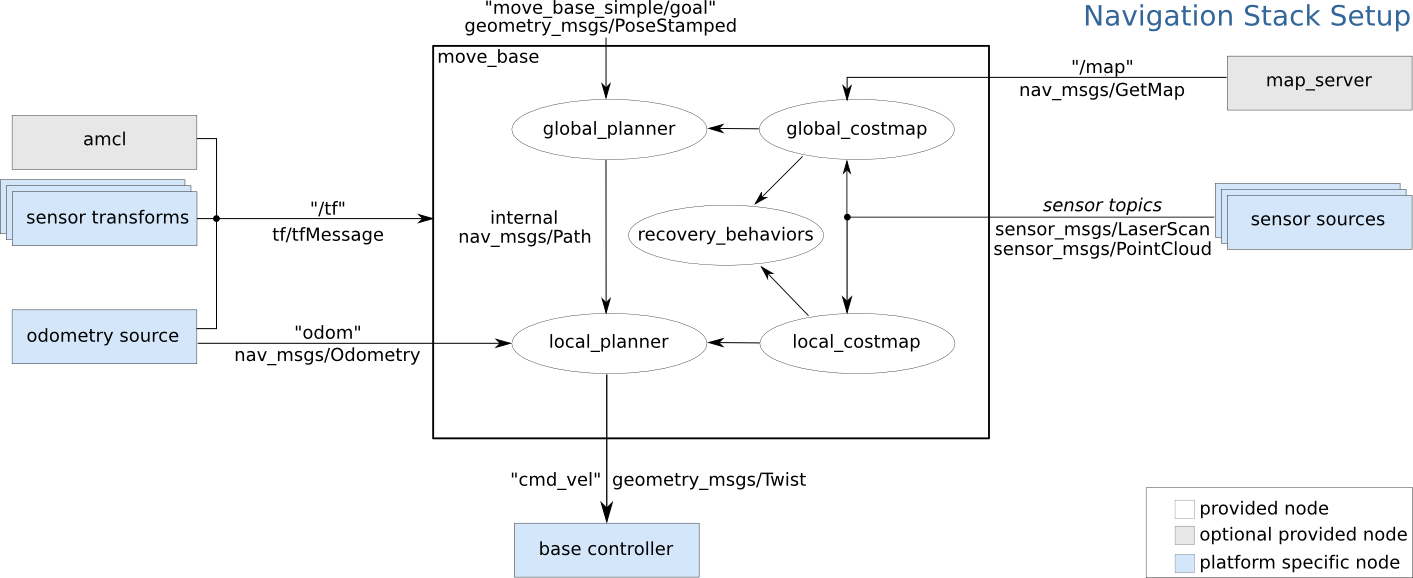
move\_base是ROS下关于机器人路径规划的中心枢纽。它通过订阅激光雷达、map地图、amcl的定位等数据，然后规划出全局和局部路径，再将路径转化为机器人的速度信息，最终实现机器人导航。具体框架如图6所示：

图6 move\_base框架

amcl：ROS的导航定位模块，amcl也叫自适应蒙特卡罗定位，amcl通过订阅scan、map和tf信息，发布出机器人的pose，以供move\_base使用；

sensor transforms：传感器位置；

odometry source：机器人里程计信息；

map\_server：通过解析建好的地图并发布出去；

costmap：代价地图，目前主要的有inflation\_layer、obstacle\_layer、static\_layer、

voxel\_layer四个plugins。分别为膨胀层、障碍物层、静态层和体素层。一般我们的全局路径需要静态层和膨胀层，因为全局规划应该只考虑到地图信息，所以一般都是静态的，而局部路径规划则需要考虑到实时的障碍物信息，所以需要障碍物层和膨胀层；

global planner：根据给定的目标位置进行总体路径的规划；

local planner：根据附近的障碍物信息进行路线规划。