

目 录

实验一 智能小车初始配置及远程遥控实验	2
一、实验目的	2
二、实验内容	2
三、实验器材	2
四、实验原理	2
五、实验步骤	4
六、实验报告要求	7
实验二 智能小车惯性导航自主移动实验	8
一、实验目的	8
二、实验内容	8
三、实验器材	8
四、实验原理	8
五、实验步骤	9
六、实验报告要求	11
实验三 智能小车自主移动避障实验	12
一、实验目的	12
二、实验内容	12
三、实验器材	12
四、实验原理	12
五、实验步骤	13
六、实验报告要求	18
实验四 基于深度学习的图片识别实验	19
一、实验目的	19
二、实验内容	19
三、实验器材	19
四、实验原理	19
五、实验步骤	20
六、实验报告要求	28

实验一 智能小车初始配置及远程遥控实验

一、实验目的

- 1、熟悉小车配置流程。
- 2、熟悉小车控制原理。

二、实验内容

- 1、组装小车，了解小车硬件结构。
- 2、配置网络并连接至本地遥控端，了解小车控制原理。
- 3、学习远程遥控小车移动方法。

三、实验器材

- 1、小车 ROS 机器人一套
- 2、装有 ubuntu 的主机一台
- 3、显示器、键盘、鼠标

四、实验原理

ROS 是 Robot Operation System 的简称。是专门为机器人控制开发的系统。和一般的操作系统相比增加了软件包管理，信息通信框架等。同时这个系统对硬件进行了抽象使得各个组件之间的耦合更低，程序开发也更加方便快捷。

ROS 与其说是操作系统其实更像一个通信框架。ROS 的基本单元是节点 (Node)。节点之间可以通过订阅和发布 Topic 进行通信。节点的基本信息会存储在 Master 里面。下面以一个例子具体说明。假如我们现在有一个激光雷达，然后我们想要通过图形界面显示雷达的数据。那么整个结构就如图 1-1 所示。激光雷达有一个节点叫做 hokuyo，图形界面有一个节点叫做 viewer。激光雷达节点从传感器读出数据之后，首先告诉 Master 自己在发布一个叫做 scan 的 topic。然后在本地监听 2345 端口，等待其他节点连接。图形界面节点 viewer 启动之后，

首先告诉 Master 自己在订阅一个叫做 scan 的 topic。这时候 Master 就会告诉 viewer 节点谁在发布 scan。Viewer 获得到 hokuyo 的地址和端口信息后和 hokuyo 节点建立 TCP 连接。这样激光雷达的数据就从 hokuyo 节点发送至 viewer 节点了。

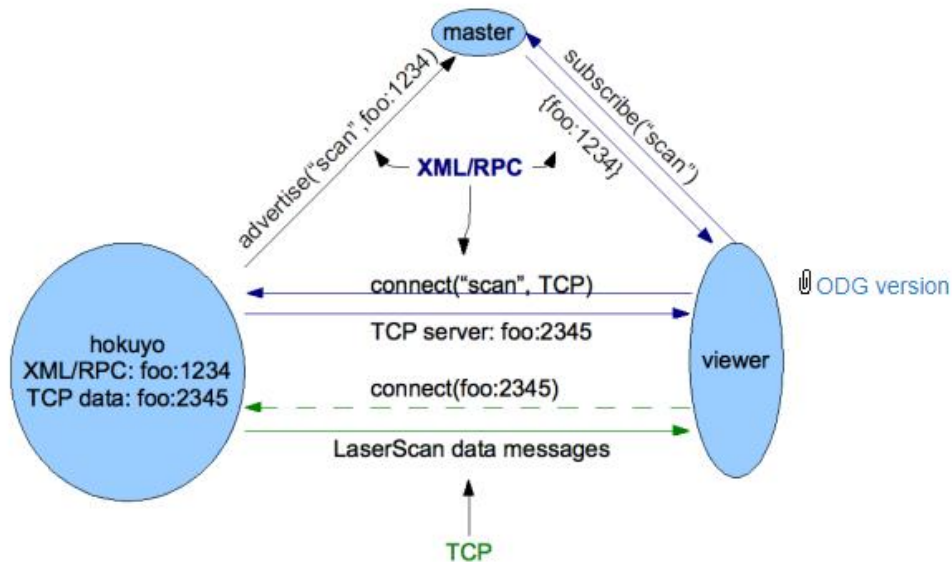


图 1-1 ROS 的架构

通过这种架构可以实现节点的分布式处理功能。即不同的节点不要求一定在同一个机器人上。比如上例中的激光雷达节点和 viewer 节点就可以在不同的机器人上。也可以进一步实现多台机器人的调度。从上例我们可以看出 ROS 中有以下几个比较重要的基本概念

1.Master

Master 就是存放各种节点和 topic 信息的地方。节点之间通过 Master 了解到对方的信息。同时在 Master 的帮助下节点之间建立网络连接。

2.Node

基本的 ROS 计算单元。

3.Topic

节点之间的通信通道

4.Parameter Server

参数服务器。在使用过程中可能不同的节点间要共享一些参数。这些参数就可以放在参数服务器里面。其作用就类似于不同节点公用的全局变量。在上面的

例子中没有用到参数服务器。

五、实验步骤

1、依据线标提示将小车接线连接好，完整结构如图 1-2 所示。



图 1-2 小车装配图

2、配置小车网络

使用赠送的 hdmi 转 vga 插头，将显示器和键盘鼠标接入小车主机后开机，进入主机 ubuntu 系统（小车默认密码是 xiaoqiang）。

选择需要接入的无线网络，接入网络后，得到小车的实际 ip 地址，后续会频繁使用到这个 ip 信息。关机后拔掉键盘鼠标和显示器，下次小车直接开机就能使用。

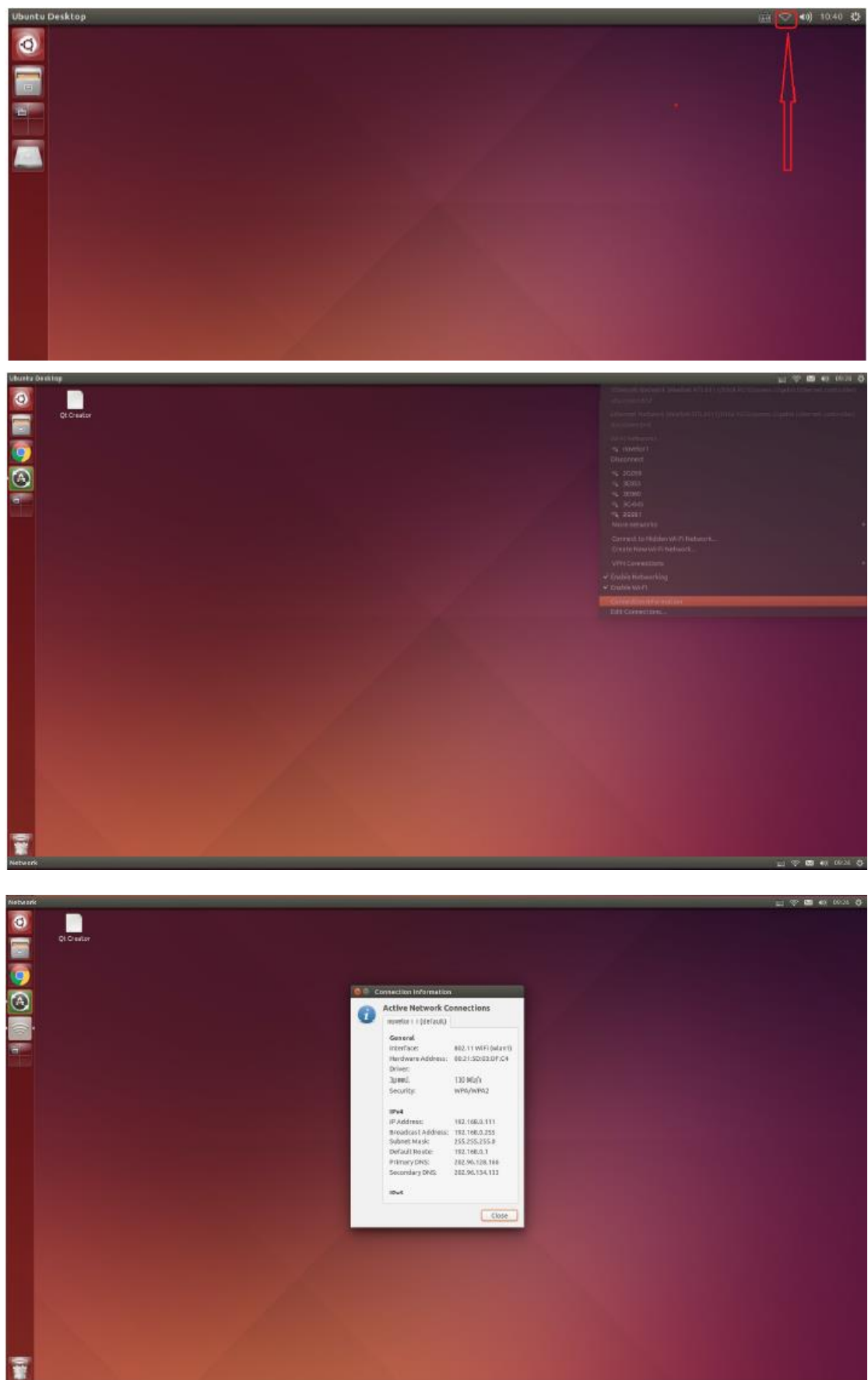


图 1-3 小车网络配置操作示意图

3、本地遥控端配置

因为后续需要经常操作更改小车主机上的文件，现在我们将小车主机远程目录添加到本地遥控端，这样在本地就可以直接图形化操作小车主机上的文件（小车主机相当于本地 ubuntu 系统的外挂硬盘）。

点击图 1-4 位置，添加小车远程目录。

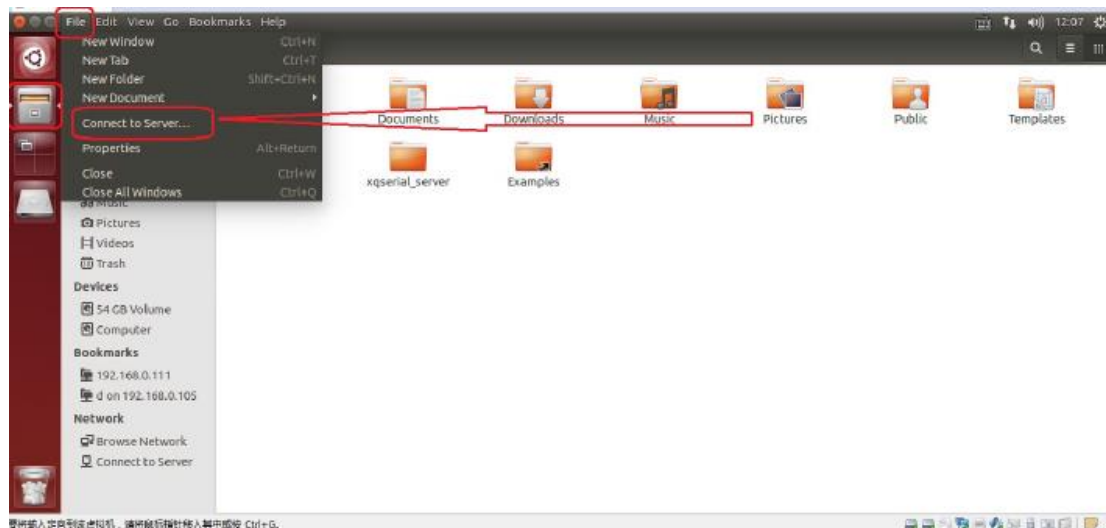


图 1-4 本地遥控端配置操作示意图 1

输入小车远程目录，请将 ip 换成上文提到的实际 ip 地址，根据提示输入小车主机用户名和密码，如图 1-5 所示。

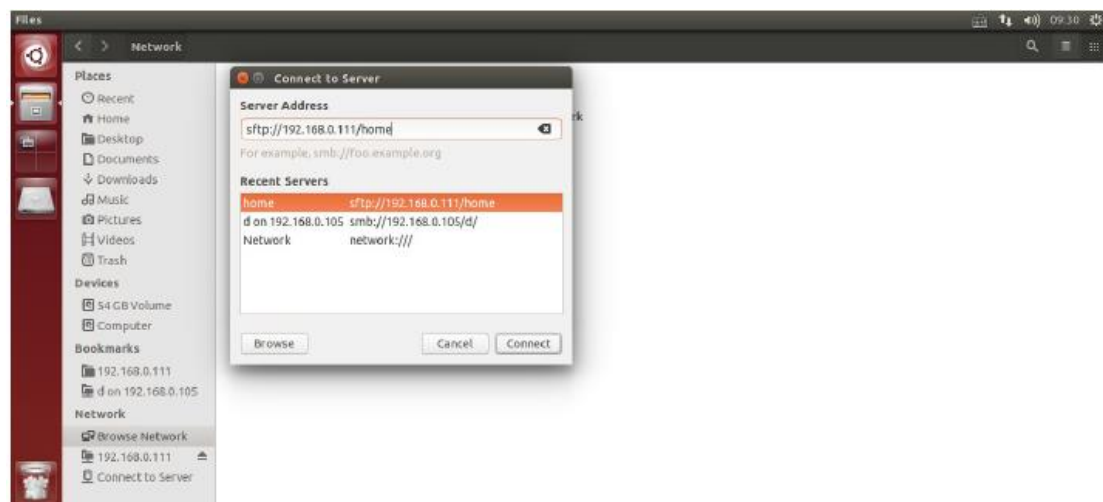


图 1-5 本地遥控端配置操作示意图 2

一切正常的话，已经打开了小车主机的 home 目录，为了未来使用方便，可以将这个地址添加到 bookmark，如图 1-6 所示，下次直接点击这个 bookmark，就能访问小车主机的 home 目录。

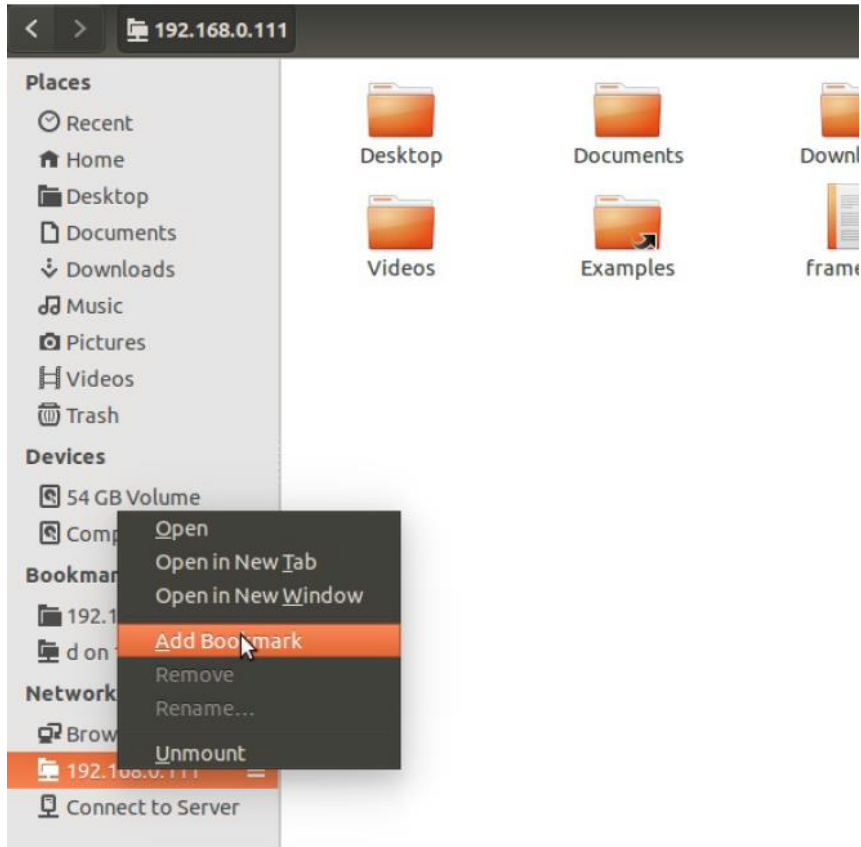


图 1-6 本地遥控端配置操作示意图 3

4、ssh 登录小车主机，用键盘遥控小车移动

(1)在本地遥控端打开一个终端

(2)通过 ssh 连接小车，请将 xxx 换成实际的 ip

```
ssh xiaoqiang@xxx.xxx.xxx.xxx
```

(3)启动遥控程序

```
roslaunch nav_test control.py
```

(4)可以开始通过方向键来控制小车的移动了。空格键是停止。Ctrl+C 退出程序。

六、实验报告要求

- 1、叙述 ROS 的基本原理和架构。
- 2、阅读小车控制代码，说明遥控小车控制的流程。
- 3、记录实验过程中遇到的问题并进行分析，提出解决方法。

实验二 智能小车惯性导航自主移动实验

一、实验目的

- 1、了解 ROS 导航框架。
- 2、了解小车惯性导航自主移动原理。

二、实验内容

- 1、了解小车上携带的惯性传感器等硬件原理。
- 2、阅读相应的 ROS 软件包，了解小车惯性导航自主移动原理。
- 3、学习 rviz 软件，并掌握控制小车惯性导航自主移动的方法。

三、实验器材

- 1、小车 ROS 机器人一套
- 2、装有 ubuntu 的主机一台
- 3、显示器、键盘、鼠标

四、实验原理

这里的惯性导航，是指利用小车自身佩戴的惯性传感器（加速度和陀螺仪）和底盘编码器信息定位和移动。需要的 ROS 软件包有：1.底层驱动 `xqserial_server`,2.机器人模型包 `xiaoqiang_udrf`,3.惯性导航测试软件包 `nav_test`. 整个导航测试的内部框架如图 2-1 所示：

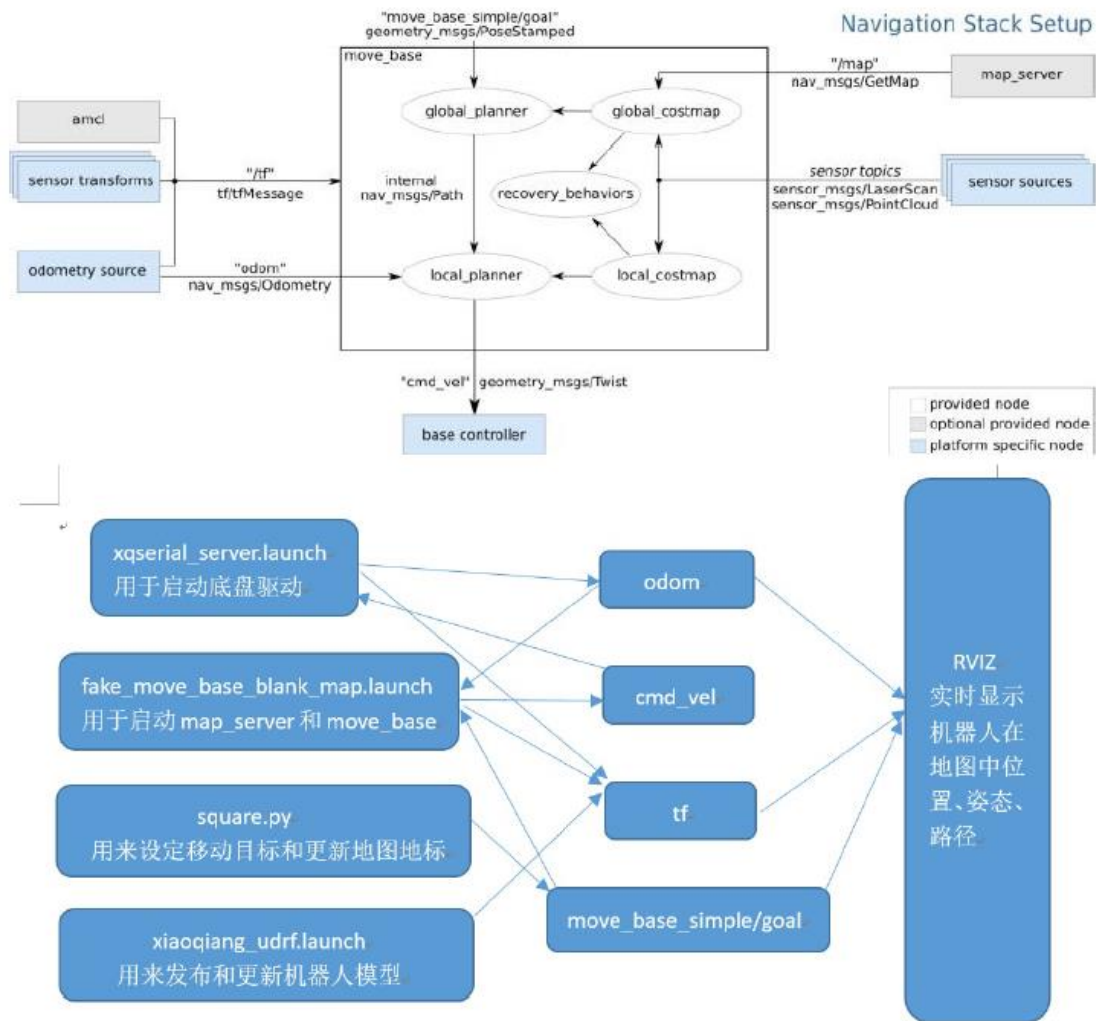


图 2-1 导航测试内部框架

五、实验步骤

1、ssh 方式在小车主机上完成的操作

(1) 新开一个终端，启动导航基础程序。(如果出现了 You must specify at least three point for the robot footprint 这样的错误可以忽略，这个并没有什么影响。这是由于 foot_print 要求设置一个形状，所以至少要三个点。foot_print 就是机器人的轮廓形状，这个参数会用于避障。小车的 foot_print 设置的是圆形，所以并不需要指定三个点。如果出现的 tf 树和下图并不相同，请检查一下底盘驱动是不是正常运行)

```
ssh -X xiaoqiang@192.168.xxx.xxx
```

```
# 重启服务程序，有些程序基础可能没有运行
```

```
sudo service startup restart
```

```
roslaunch nav_test fake_move_base_blank_map.launch
```

(2) 新开一个终端，检查是否所有 tf 都已经就位，正常会显示图 2-2。

```
ssh -X xiaoqiang@192.168.xxx.xxx
```

```
roslaunch tf view_frames
```

```
evince frames.pdf
```

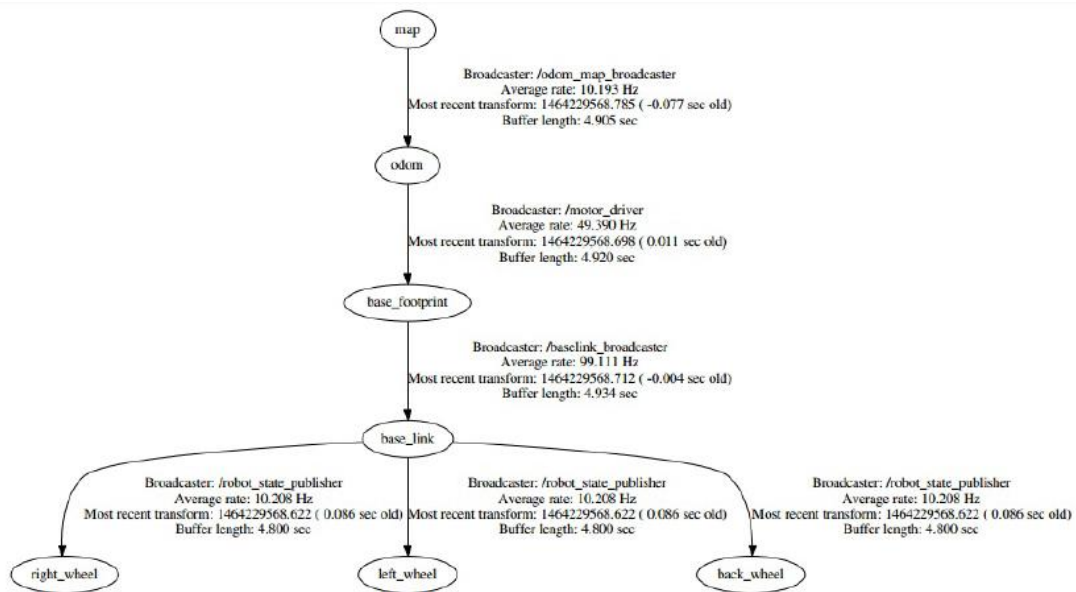


图 2-2 导航测试框架示意图

2、在本地遥控端上完成的操作

(1) 在本地开一个命令行终端，在本地的 hosts 文件内添加小车的 ip。

```
sudo gedit /etc/hosts
```

```
#添加
```

```
xxx.xxx.xxx.xxx xiaoqiang-desktop #请将 xx 改成实际 ip
```

(2) 新开一个命令终端输入，如果可以看到小车的 topic 了,就说明配置成功。

```
export ROS_MASTER_URI=http://xiaoqiang-desktop:11311
```

```
#继续执行
```

```
rostopic list
```

(3) 打开 rviz 图形界面。

```
export ROS_MASTER_URI=http://xiaoqiang-desktop:11311
```

```
source ~/Documents/ros/devel/setup.sh
```

rviz

当窗口打开后，点击左上角的 `file->open`，选择小车里的 `/home/xiaoqiang/Documents/ros/src/nav_test/config/nav.rviz` 文件，这时界面应该如图 2-3 所示。

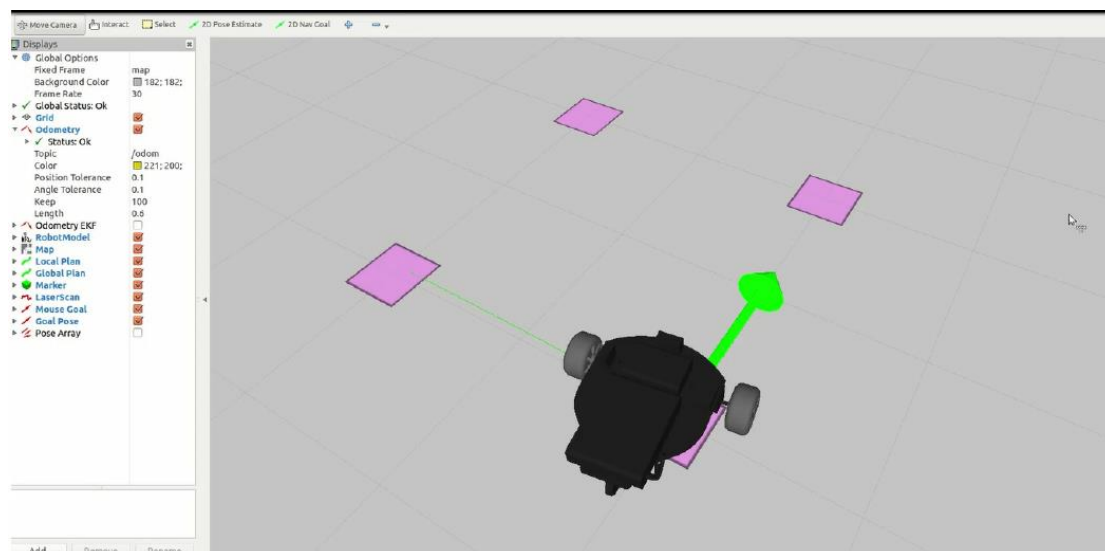


图 2-3 rviz 界面图

3、在小车主机远程 ssh 窗口内完成最后操作

```
roslaunch nav_test square.py
```

如果小车无法移动，请检查底盘驱动程序是否正常运行。可以输入 `rostopic echo /system_monitor/report` 如果有电压显示则底盘驱动正常。如果还是无法移动，可以再检查小车红外传感器是否被触发。触发时红外传感器会发红光。

六、实验报告要求

- 1、了解 ROS 导航框架，叙述小车导航框架原理。
- 2、阅读惯性导航测试软件包，说明惯性导航的工作流程。
- 3、记录实验过程中遇到的问题并进行分析，提出解决方法。

实验三 智能小车自主移动避障实验

一、实验目的

- 1、熟悉 kinect 工作原理。
- 2、熟悉小车自主移动避障原理。

二、实验内容

- 1、学习掌握 kinect 用于避障的原理。
- 2、熟悉小车和本地虚拟机的配置，并掌握控制小车自主移动避障的方法。

三、实验器材

- 1、小车 ROS 机器人一套
- 2、装有 ubuntu 的主机一台
- 3、显示器、键盘、鼠标

四、实验原理

Kinect 水平方向安置三个镜头，中间的镜头是 RGB 彩色摄影机，可以获取彩色影像，左右两边镜头分别是红外线发射器和 CMOS 红外线摄影机，识别的是一个“深度场”(Depth Field)，其中每个像素的颜色的深浅表示那一点物体离摄像头远近，离摄像头近的颜色比较亮和深，而离摄像头远的颜色则偏暗色，甚至全黑，它们构成了 3D 深度感应器。

CMOS 红外线传感器令 Kinect 即使在光线不足的情况下也可以侦测环境，它利用某一颜色的深浅来表示周围环境的深度场。远处的场景和物体表现为暗色，近处的为亮色。表示无穷远及无穷近的黑白色间过度地带的颜色深浅对应表示物体到传感器摄像头物理距离的远近。它收集视野范围内的每一点，并形成一幅代表周围环境的景深图像。

Kinect 的工作构架如图 3-1 所示，它使用的数据采集技术是非接触式的，

采用光编码(Light Coding)技术进行 3D 侦测，获取深度影像资料。激光照到物体表面会产生高度随机的散斑，场景中不同点到摄像头的距离不同，散斑图案也不同。在 Kinect 中，这一差别被用来对场景空间的任意点进行标记。通过空间中物体上的散斑图案即可计算出物体所在的确切位置，即深度信息。Light Coding 技术原理是利用连续近红外线激光对测量场景进行编码，即产生激光散斑，然后经感应器读取编码的光线，交由晶片运算进行解码后，产生一张具有深度信息的图像。Kinect 的红外线发射器通过镜头前的光栅将激光均匀投射在测量空间中，再经由红外线摄影机记录下空间中的经过编码的散斑，然后经过晶片执行复杂的并行逻辑运算生成具有 3D 深度信息的图像。

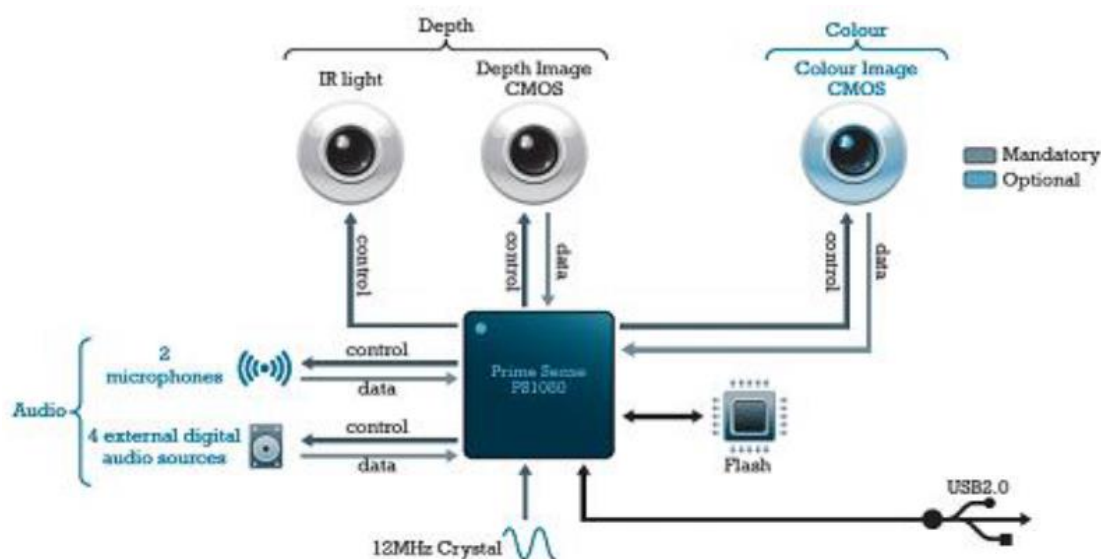


图 3-1 Kinect 工作构架

freemove_stack 包提供 kinect 驱动，其发布的点云通过 image_pipeline 转换成障碍物栅格分布图。nav_test 软件包启动底盘导航程序后会自动处理分析障碍物分布图，之后根据 rivz 发布的目标导航点自主移动。

五、实验步骤

1、配置小车主机和本地虚拟机的 hosts 文件，使两者能互相访问对应的 ros 数据

(1) 配置本地虚拟机，将图 3-2 中的 ip 地址换成小车实际数值。

```
sudo gedit /etc/hosts
```

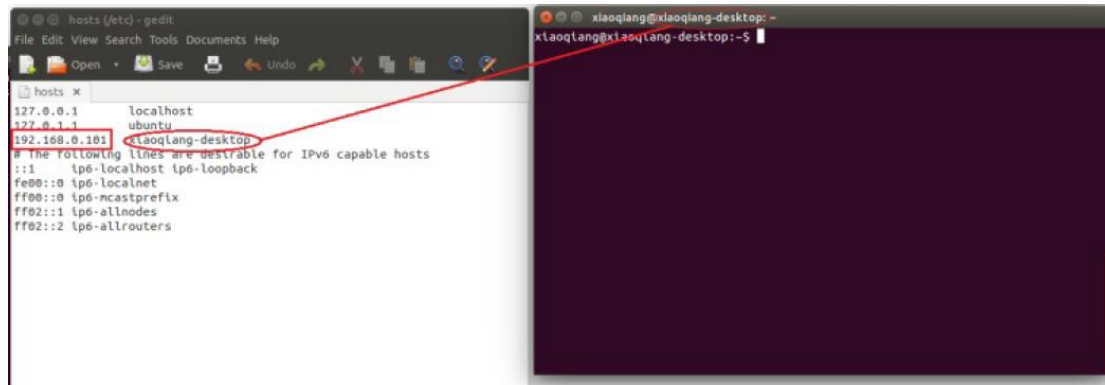


图 3-2 本地虚拟机配置图

(2) 配置小车主机，将图 3-3 中的 ip 地址换成本地虚拟机实际数值，主机名称改为虚拟机名称。

局域网 ssh 登入小车主机

ssh xiaoqiang@192.168.0.101 -X

sudo gedit /etc/hosts

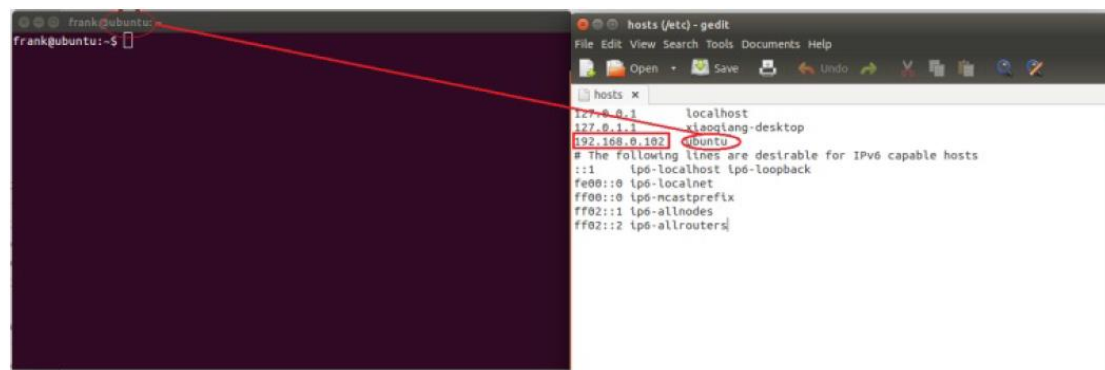


图 3-3 小车虚拟机配置图 1

2、在本地虚拟机中新开 3 个窗口，分别 ssh 登入小车，启动原理部分提及的相关软件包。

(1) ssh 登入，如图 3-4 所示。

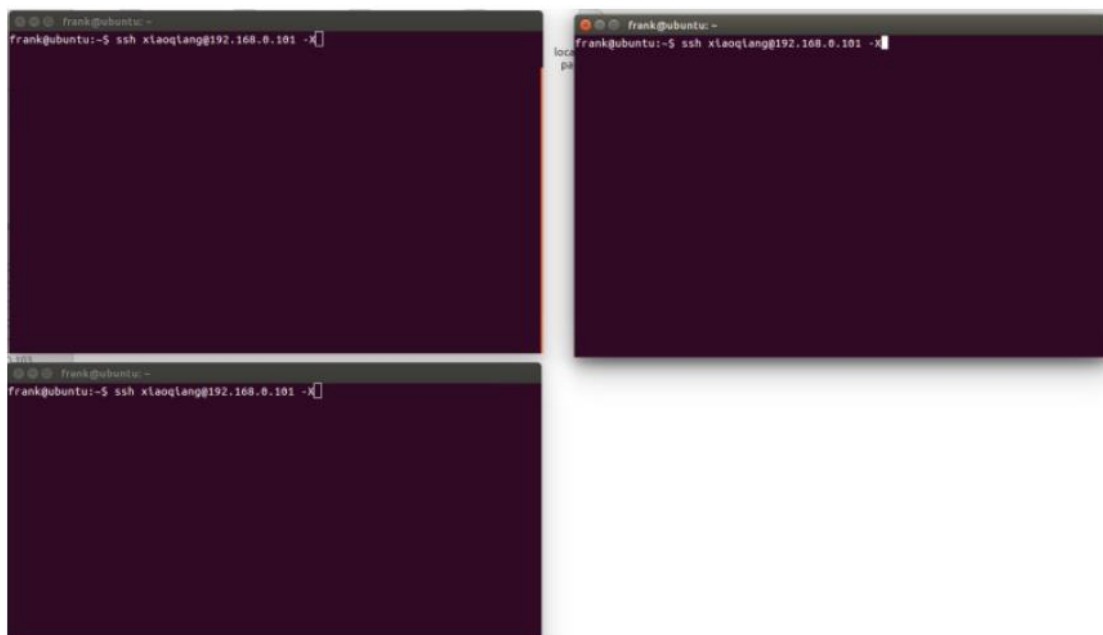


图 3-4 本地虚拟机配置图 2

- (2) 在第一个窗口启动 kinect 驱动
`roslaunch freenect_launch kinect-xyz.launch`
- (3) 在第二个窗口设置 kinect 俯仰角，这个角度不是任意的
`rostopic pub /set_tilt_degree std_msgs/Int16 '{data: -19}' -1`
- (4) 编辑底盘导航程序配置文件
`/home/xiaoqiang/Documents/ros/src/nav_test/config/fake/base_local_planner_params2.yaml`，使能 kinect。如图 3-5 所示

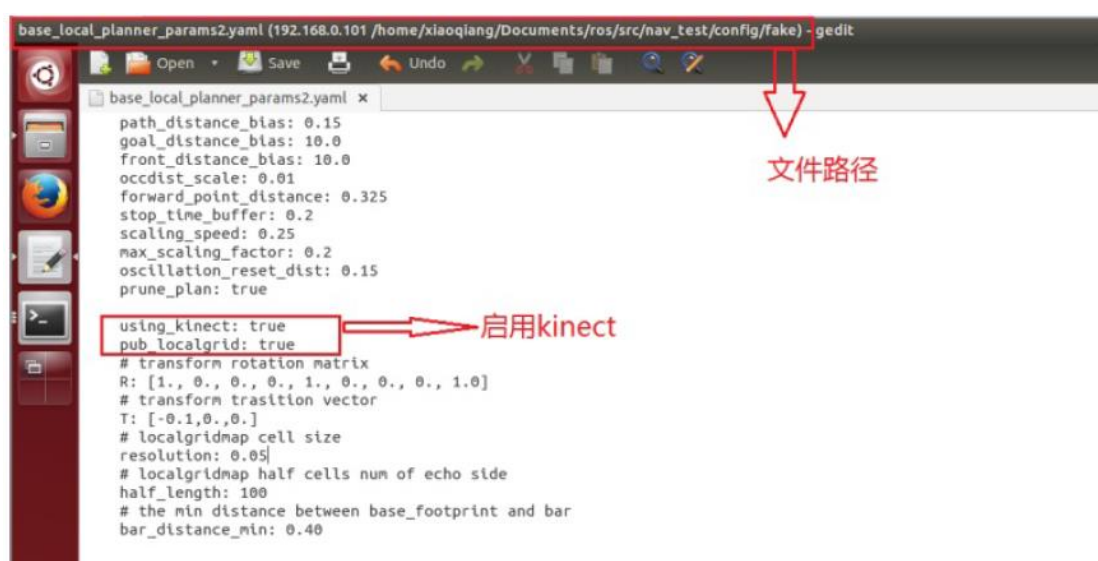
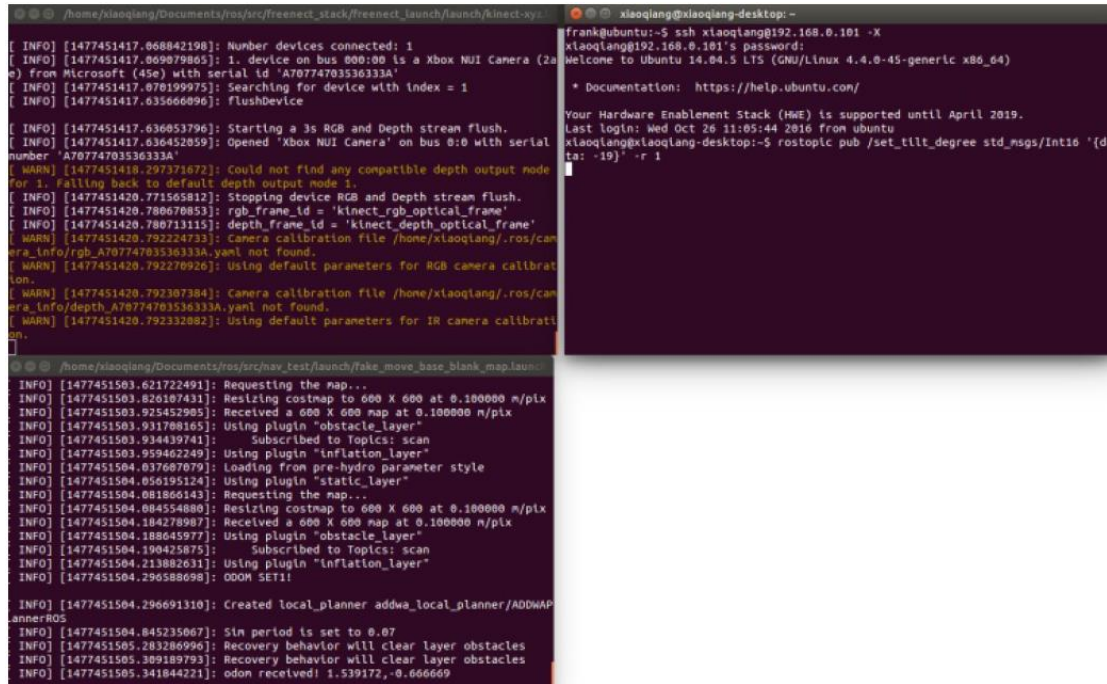


图 3-5 小车虚拟机配置图 2

(5) 在第三个窗口启动底盘导航程序

```
roslaunch nav_test fake_move_base_blank_map.launch
```

(6) 全部正常，会出现类似下图 3-6 的界面，到此小车端配置启动完成。



```
[ INFO] [1477451417.068042198]: Number devices connected: 1
[ INFO] [1477451417.069079065]: 1. device on bus 000:00 is a Xbox NUI Camera (2a
e) from Microsoft (45e) with serial id 'A70774703536333A'
[ INFO] [1477451417.070199975]: Searching for device with index = 1
[ INFO] [1477451417.035660696]: FlushDevice
[ INFO] [1477451417.636053796]: Starting a 3s RGB and Depth stream flush.
[ INFO] [1477451417.636452859]: Opened 'Xbox NUI Camera' on bus 0:0 with serial
number 'A70774703536333A'
[ WARN] [1477451418.297371672]: Could not find any compatible depth output mode
for 1. Falling back to default depth output mode 1.
[ INFO] [1477451420.771565812]: Stopping device RGB and Depth stream flush.
[ INFO] [1477451420.780670853]: rgb_frame_id = 'kinect_rgb_optical_frame'
[ INFO] [1477451420.780713115]: depth_frame_id = 'kinect_depth_optical_frame'
[ WARN] [1477451420.792224733]: Camera calibration file /home/xiaoqiang/.ros/cam
era_info/rgb_A70774703536333A.yaml not found.
[ WARN] [1477451420.792270926]: Using default parameters for RGB camera calibrat
ion.
[ WARN] [1477451420.792307384]: Camera calibration file /home/xiaoqiang/.ros/cam
era_info/depth_A70774703536333A.yaml not found.
[ WARN] [1477451420.792332082]: Using default parameters for IR camera calibrati
on.
[ INFO] [1477451503.621722491]: Requesting the map...
[ INFO] [1477451503.826187431]: Resizing costmap to 600 X 600 at 0.100000 m/pix
[ INFO] [1477451503.925452905]: Received a 600 X 600 map at 0.100000 m/pix
[ INFO] [1477451503.931708165]: Using plugin "obstacle_layer"
[ INFO] [1477451503.934439741]: Subscribed to Topics: scan
[ INFO] [1477451503.959462249]: Using plugin "inflation_layer"
[ INFO] [1477451504.037607079]: Loading from pre-hydro parameter style
[ INFO] [1477451504.056195124]: Using plugin "static_layer"
[ INFO] [1477451504.081866143]: Requesting the map...
[ INFO] [1477451504.084554880]: Resizing costmap to 600 X 600 at 0.100000 m/pix
[ INFO] [1477451504.104278987]: Received a 600 X 600 map at 0.100000 m/pix
[ INFO] [1477451504.188645977]: Using plugin "obstacle_layer"
[ INFO] [1477451504.190425875]: Subscribed to Topics: scan
[ INFO] [1477451504.213882631]: Using plugin "inflation_layer"
[ INFO] [1477451504.296588698]: ODOM SET!
[ INFO] [1477451504.296691310]: Created local_planner addwa_local_planner/ADDWAPl
annerROS
[ INFO] [1477451504.845235067]: Sin period is set to 0.07
[ INFO] [1477451505.283286996]: Recovery behavior will clear layer obstacles
[ INFO] [1477451505.309189793]: Recovery behavior will clear layer obstacles
[ INFO] [1477451505.341844221]: odom received! 1.539172, 0.666669
```

图 3-5 配置成功示意图

3、在本地虚拟机中新开 1 个窗口，用来启动 rviz。

(1) 加入 ros 局域网后，打开 rviz。

```
export ROS_MASTER_URI=http://xiaoqiang-desktop:11311
```

```
rviz
```

(2) 点击 rviz 界面左上角的 open config，选择小车主机上的
/home/xiaoqiang/Documents/ros/src/nav_test/config/nav_addwa_kinect.rvi
z 配置文件，如图 3-6 所示。

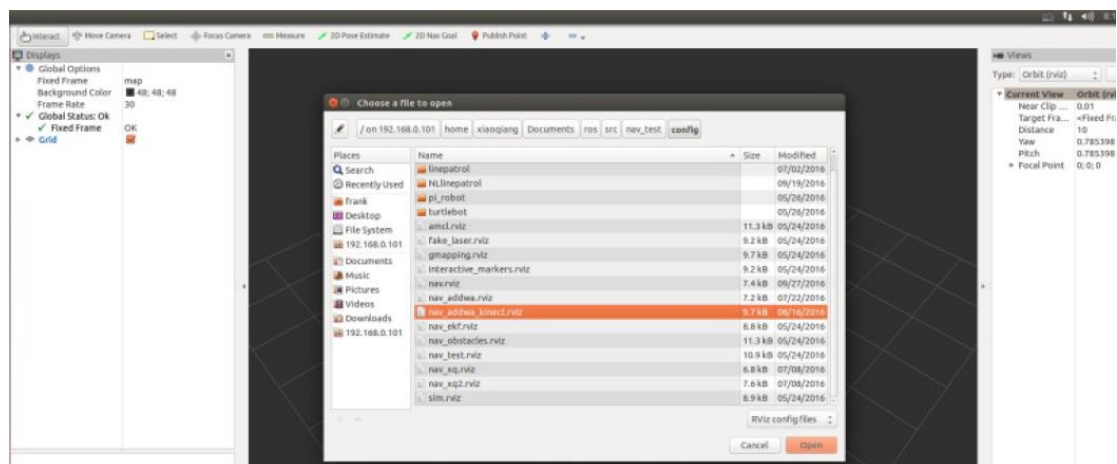


图 3-6 rviz 配置图

(3) 正常的话，现在 rviz 中将出现类似图 3-7 的画面，现在所有配置都已经完成，开始发布导航目标点。

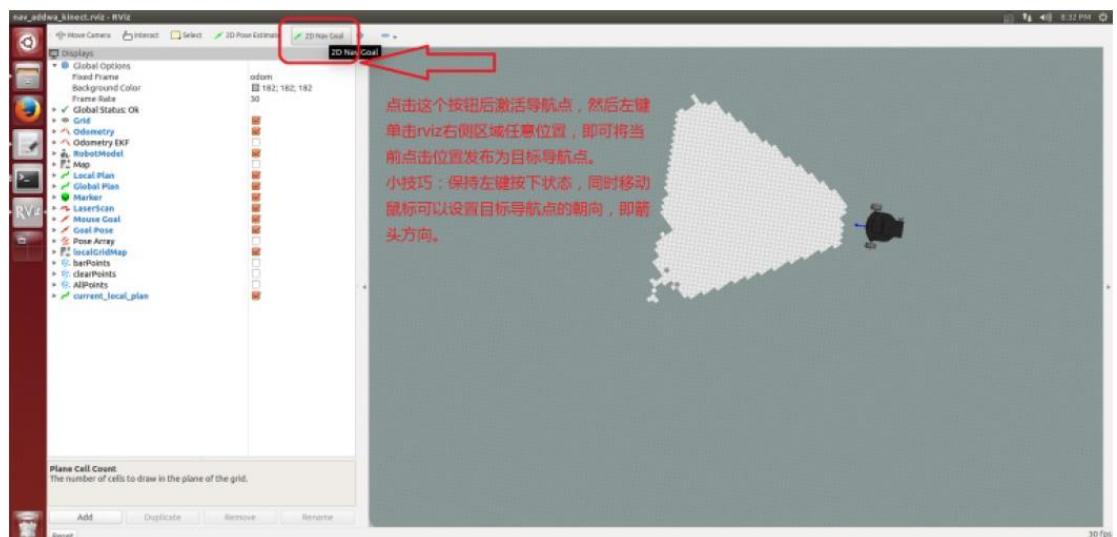


图 3-7 rviz 配置成功示意图

(4) 发布一个目标点，小车会开始自主移动，如图 3-8 所示。

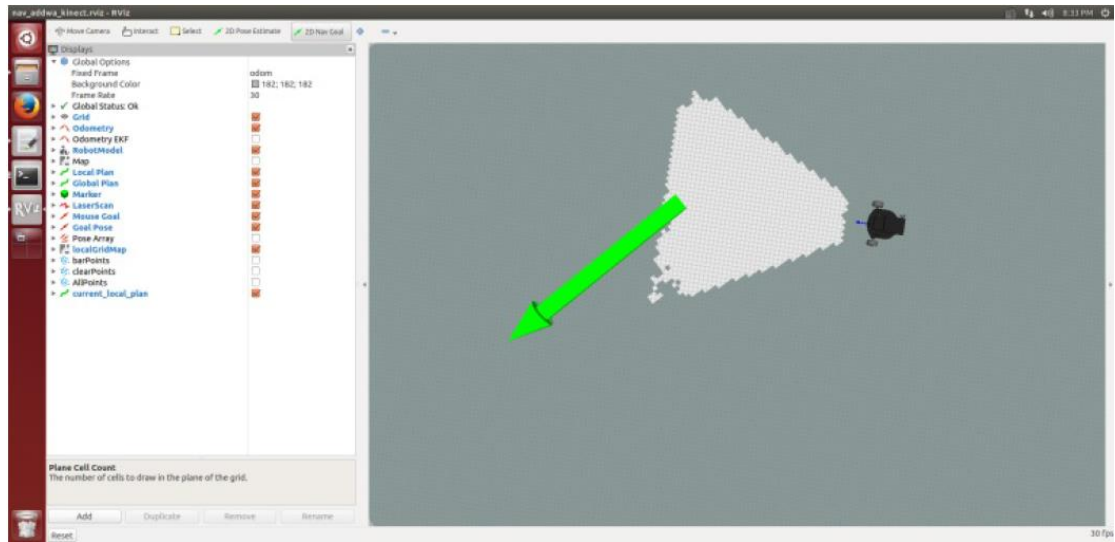


图 3-8 小车自主导航避障示意图

六、实验报告要求

- 1、了解并叙述 Kinect 生成点云数据的原理。
- 2、阅读自主避障软件包，并介绍小车自主导航避障的原理。
- 3、记录实验过程中遇到的问题并进行分析，提出解决方法。

实验四 基于深度学习的图片识别实验

一、实验目的

- 1、熟悉并使用 tensorflow 框架。
- 2、熟悉基本深度学习算法原理。

二、实验内容

- 1、学习掌握 tensorflow 用于图片识别的原理。
- 2、掌握 tensorflow 的使用方法。

三、实验器材

- 1、英伟达 jetson tx2 开发板一套
- 2、显示器、键盘、鼠标

四、实验原理

TensorFlow 是一个采用数据流图 (data flow graphs), 用于数值计算的开源软件库。节点 (Nodes) 在图中表示数学操作, 图中的线 (edges) 则表示在节点间相互联系的多维数据数组, 即张量 (tensor)。它灵活的架构让你可以在多种平台上展开计算, 例如台式计算机中的一个或多个 CPU (或 GPU), 服务器, 移动设备等等。TensorFlow 最初由 Google 大脑小组 (隶属于 Google 机器智能研究机构) 的研究员和工程师们开发出来, 用于机器学习和深度神经网络方面的研究, 但这个系统的通用性使其也可广泛用于其他计算领域。

数据流图用“结点”(nodes)和“线”(edges)的有向图来描述数学计算, 如图 4-1 所示。“节点”一般用来表示施加的数学操作, 但也可以表示数据输入(feed in)的起点/输出(push out)的终点, 或者是读取/写入持久变量(persistent variable)的终点。“线”表示“节点”之间的输入/输出关系。这些数据“线”可以输运“size 可动态调整”的多维数据数组, 即“张量”(tensor)。张量从图中流过的直观图像是这个

工具取名为“Tensorflow”的原因。一旦输入端的所有张量准备好，节点将被分配到各种计算设备完成异步并行地执行运算。

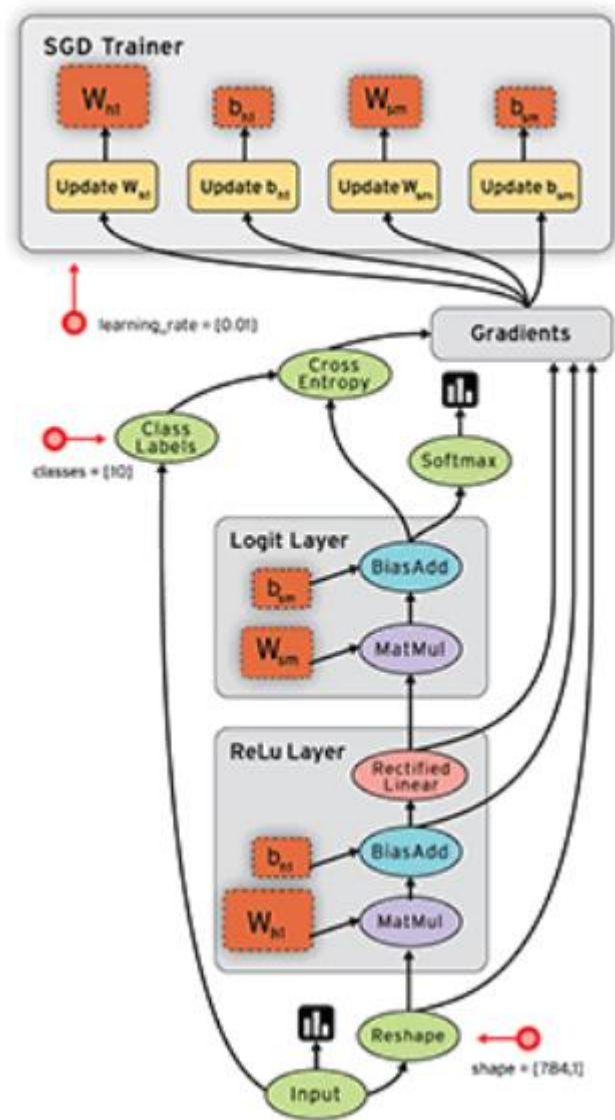


图 4-1 tensorflow 框架原理图

TensorFlow Object Detection API 是一个建立在 TensorFlow 之上的开源框架，在上面可以很容易地建立，训练和部署对象检测模型，因此本实验将在该框架下运行。

五、实验步骤

1、打开终端运行 jupyter lab,如图 4-2 所示。

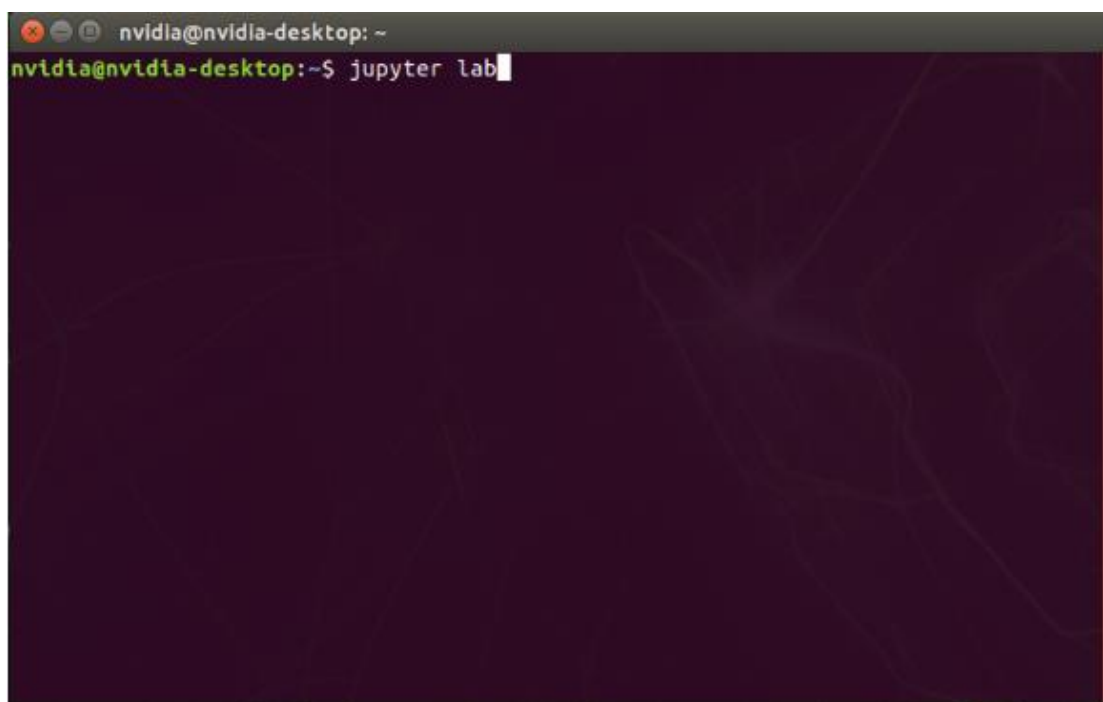


图 4-2 jupyter lab 开启图示

2、在左侧资源栏依次点击文件夹 tensorflow-models, research, object_detection, 进入到 object_detection 文件夹。点击左上角的 File, 然后点击 New, 新建一个 Notebook 文件, 如图 4-3 所示。

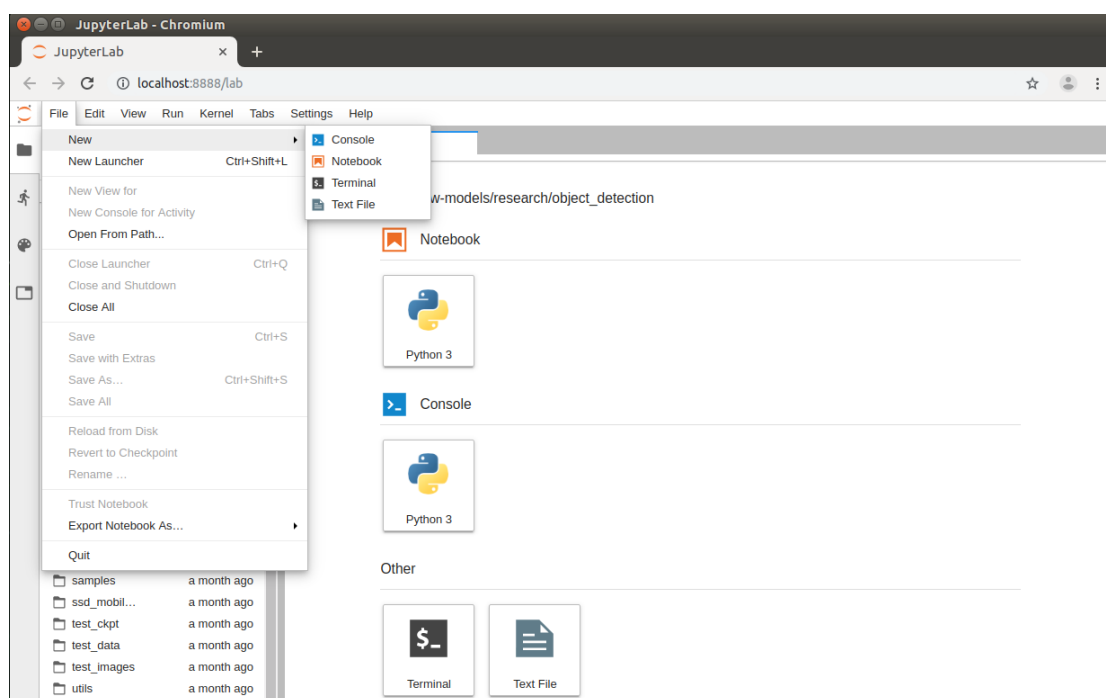


图 4-3 新建 Notebook 文件图示

3、在单元格中输入代码

(1) 导入需要的包

```
import numpy as np
import os
import six.moves.urllib as urllib
import sys
import tarfile
import tensorflow as tf
import zipfile
from collections import defaultdict
from io import StringIO
import matplotlib as mpl
mpl.use('TkAgg')
from matplotlib import pyplot as plt
from PIL import Image
# This is needed since the notebook is stored in the object_detection folder.
sys.path.append("..")
from object_detection.utils import ops as utils_ops
```

(2) 设置系统环境

```
# This is needed to display the images.
```

```
%matplotlib inline
```

(3) 导入 Object detection 需要的模块，如果报错，说明工作目录设置不对，或者.../research 以及.../research/slim 的环境变量没有设置好。

```
from utils import label_map_util
from utils import visualization_utils as vis_util
```

(4) 设置模型的对应参数，github 上有对应官方的各种模型

(https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md), 这些都是基于不同的数据集事先训练好的模型, 下载好以后就可以直接调用。下载的文件以 '.tar.gz' 结尾。'PATH_TO_CKPT' 为 '.pb' 文件的目录, '.pb' 文件是训练好的模型 (frozen detection graph), 即用来预测时使用的模型。'PATH_TO_LABELS' 为标签文件, 记录了哪些标签需要识别, 如图 4-4 所示, 第一列是模型名字, 第二列是速度, 第三列是精度。

COCO-trained models

Model name	Speed (ms)	COCO mAP[^1]	Outputs
ssd_mobilenet_v1_coco	30	21	Boxes
ssd_mobilenet_v1_0.75_depth_coco ☆	26	18	Boxes
ssd_mobilenet_v1_quantized_coco ☆	29	18	Boxes
ssd_mobilenet_v1_0.75_depth_quantized_coco ☆	29	16	Boxes
ssd_mobilenet_v1_ppn_coco ☆	26	20	Boxes
ssd_mobilenet_v1_fpn_coco ☆	56	32	Boxes
ssd_resnet_50_fpn_coco ☆	76	35	Boxes
ssd_mobilenet_v2_coco	31	22	Boxes
ssd_mobilenet_v2_quantized_coco	29	22	Boxes
ssdlite_mobilenet_v2_coco	27	22	Boxes
ssd_inception_v2_coco	42	24	Boxes
faster_rcnn_inception_v2_coco	58	28	Boxes
faster_rcnn_resnet50_coco	89	30	Boxes

图 4-4 tensorflow model zoo 图示

```
# What model to download.

MODEL_NAME = 'ssd_mobilenet_v1_coco_2017_11_17'
MODEL_FILE = MODEL_NAME + '.tar.gz'
DOWNLOAD_BASE='http://download.tensorflow.org/models/object_detection/'

# Path to frozen detection graph. This is the actual model that is used for the object
detection.

PATH_TO_FROZEN_GRAPH = MODEL_NAME + '/frozen_inference_graph.pb'

# List of the strings that is used to add correct label for each box.

PATH_TO_LABELS = os.path.join('data', 'mscoco_label_map.pbtxt')
```

(5) 下载模型，通过向对应网站发送请求进行下载解压操作。

```
opener = urllib.request.URLopener()
opener.retrieve(DOWNLOAD_BASE + MODEL_FILE, MODEL_FILE)
tar_file = tarfile.open(MODEL_FILE)
for file in tar_file.getmembers():
    file_name = os.path.basename(file.name)
    if 'frozen_inference_graph.pb' in file_name:
        tar_file.extract(file, os.getcwd())
```

(6) 将训练完的模型载入内存，将标签 map 载入并定义图片转换函数。

```
detection_graph = tf.Graph()
with detection_graph.as_default():
    od_graph_def = tf.GraphDef()
    with tf.gfile.GFile(PATH_TO_FROZEN_GRAPH, 'rb') as fid:
        serialized_graph = fid.read()
        od_graph_def.ParseFromString(serialized_graph)
        tf.import_graph_def(od_graph_def, name='')
category_index=label_map_util.create_category_index_from_labelmap(PATH_TO_LABELS, use_display_name=True)
def load_image_into_numpy_array(image):
    (im_width, im_height) = image.size
    return np.array(image.getdata()).reshape(
        (im_height, im_width, 3)).astype(np.uint8)
```

(7) 设置检测目标文件夹。

```
# For the sake of simplicity we will use only 2 images:
# image1.jpg
# image2.jpg
```



```

PATH_TO_TEST_IMAGES_DIR = 'test_images'
TEST_IMAGE_PATHS=[os.path.join(PATH_TO_TEST_IMAGES_DIR,'image{}.jpg
'.format(i)) for i in range(1, 3) ]
# Size, in inches, of the output images.
IMAGE_SIZE = (12, 8)

```

(8) 开始检测

```

def run_inference_for_single_image(image, graph):
    with graph.as_default():
        with tf.Session() as sess:
            # Get handles to input and output tensors
            ops = tf.get_default_graph().get_operations()
            all_tensor_names = {output.name for op in ops for output in op.outputs}
            tensor_dict = {}
            for key in [
                'num_detections', 'detection_boxes', 'detection_scores',
                'detection_classes', 'detection_masks'
            ]:
                tensor_name = key + ':0'
                if tensor_name in all_tensor_names:
                    tensor_dict[key] = tf.get_default_graph().get_tensor_by_name(
                        tensor_name)
            if 'detection_masks' in tensor_dict:
                # The following processing is only for single image
                detection_boxes = tf.squeeze(tensor_dict['detection_boxes'], [0])
                detection_masks = tf.squeeze(tensor_dict['detection_masks'], [0])
                # Reframe is required to translate mask from box coordinates to image
                coordinates and fit the image size.

                real_num_detection = tf.cast(tensor_dict['num_detections'][0], tf.int32)

```

```

detection_boxes = tf.slice(detection_boxes, [0, 0], [real_num_detection, -1])
detection_masks = tf.slice(detection_masks, [0, 0, 0], [real_num_detection, -
1, -1])

detection_masks_reframed=utils_ops.reframe_box_masks_to_image_masks(
    detection_masks, detection_boxes, image.shape[1], image.shape[2])
detection_masks_reframed = tf.cast(
    tf.greater(detection_masks_reframed, 0.5), tf.uint8)
# Follow the convention by adding back the batch dimension
tensor_dict['detection_masks'] = tf.expand_dims(
    detection_masks_reframed, 0)
image_tensor = tf.get_default_graph().get_tensor_by_name('image_tensor:0')

# Run inference
output_dict = sess.run(tensor_dict,
                        feed_dict={image_tensor: image})

# all outputs are float32 numpy arrays, so convert types as appropriate
output_dict['num_detections'] = int(output_dict['num_detections'][0])
output_dict['detection_classes'] = output_dict[
    'detection_classes'][0].astype(np.int64)
output_dict['detection_boxes'] = output_dict['detection_boxes'][0]
output_dict['detection_scores'] = output_dict['detection_scores'][0]
if 'detection_masks' in output_dict:
    output_dict['detection_masks'] = output_dict['detection_masks'][0]
return output_dict

for image_path in TEST_IMAGE_PATHS:
    image = Image.open(image_path)
    # the array based representation of the image will be used later in order to prepare

```

the

```
# result image with boxes and labels on it.

image_np = load_image_into_numpy_array(image)

# Expand dimensions since the model expects images to have shape: [1, None, None,
3]

image_np_expanded = np.expand_dims(image_np, axis=0)

# Actual detection.

output_dict = run_inference_for_single_image(image_np_expanded,
detection_graph)

# Visualization of the results of a detection.

vis_util.visualize_boxes_and_labels_on_image_array(
    image_np,
    output_dict['detection_boxes'],
    output_dict['detection_classes'],
    output_dict['detection_scores'],
    category_index,
    instance_masks=output_dict.get('detection_masks'),
    use_normalized_coordinates=True,
    line_thickness=8)

plt.figure(figsize=IMAGE_SIZE)

plt.imshow(image_np)
```

4、点击运行，最终结果如图 4-5 所示。

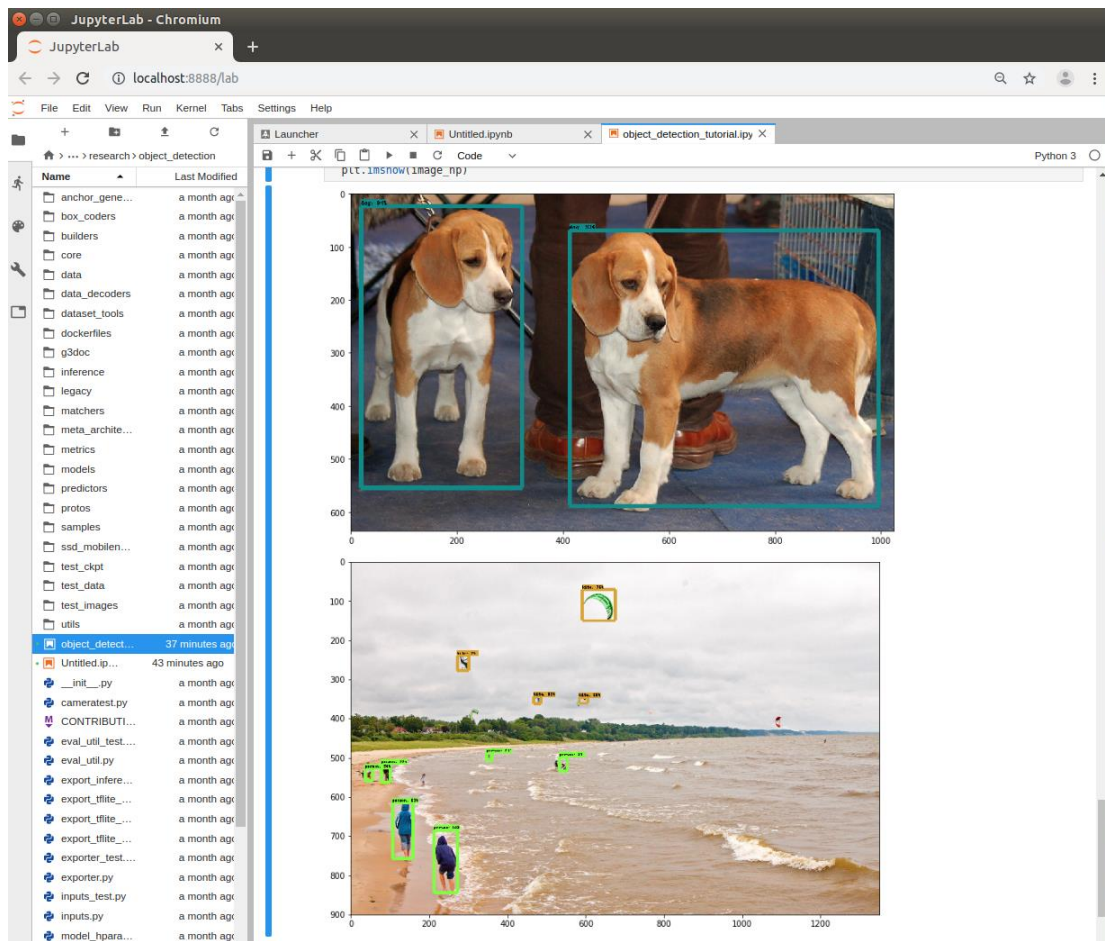


图 4-5 检测结果图

六、实验报告要求

- 1、了解并叙述本次实验所用的图片识别算法原理。
- 2、修改提供的代码，使用其他模型和测试图片进行图片识别实验，记录实验结果。
- 3、记录实验过程中遇到的问题并进行分析，提出解决方法。