

# 无人小车实训教程与技术手册

时间： 2021-1-21

# 目录

第一部分 开机运行 .....	8
1 无人小车平台介绍 .....	8
1.1 硬件平台.....	8
1.1.1 硬件清单.....	8
1.1.2 技术参数.....	8
1.1.3 整车.....	9
1.2 轮胎安装.....	9
1.3 充电与开机.....	10
1.4 相机安装.....	11
1.5 软件平台.....	12
1.5.1 软件架构.....	12
1.5.2 程序运行.....	13
第二部分 实训教程 .....	27
2 开发工具入门与实践 .....	27
2.1 ROS 操作系统基础入门.....	27
2.1.1 ROS 概述.....	27
2.1.2 ROS 安装.....	28
2.1.3 ROS 入门实例.....	28
2.1.4 ROS 架构.....	29
2.1.5 ROS 文件系统级.....	30
2.1.6 ROS 计算图级.....	31
2.1.7 ROS 社区级.....	32
2.2 ROS 订阅和发布消息内容.....	33
2.2.1 编写发布者节点.....	33
2.2.2 编写订阅器节点.....	34
2.2.3 编译与运行节点.....	35

2.3	利用 ROS 读取相机数据 .....	36
2.3.1	相机介绍 .....	36
2.3.2	摄像头启动 .....	37
2.3.3	图像显示 .....	37
3	车辆控制 .....	38
3.1	速度及转向控制 .....	39
3.1.1	键盘控制小车加减速 .....	39
3.1.2	转向控制 .....	42
3.1.3	停车控制 .....	42
3.2	巡线控制 .....	42
3.2.1	PID 算法 .....	42
3.2.2	巡线控制程序 .....	44
4	环境感知 .....	45
4.1	车道线检测 .....	45
4.1.1	启动摄像头 .....	46
4.1.2	车道线检测 .....	46
4.1.3	转向量计算 .....	48
4.2	深度学习基础 .....	48
4.2.1	深度学习的起源 .....	49
4.2.2	深度学习的发展 .....	50
4.2.3	深度学习的爆发 .....	50
4.2.4	思考题 .....	51
4.2.5	目标检测 .....	51
4.3	红绿灯识别 .....	52
4.3.1	程序运行 .....	52
4.3.2	红绿灯识别输出 .....	54
4.3.3	红绿灯识别程序逻辑 .....	55
4.4	交通标志检测 .....	55
5	路径规划 .....	56

5.1	原理介绍.....	56
5.1.1	定义.....	56
5.1.2	分类.....	57
5.1.3	步骤.....	58
5.1.4	方法.....	58
5.2	全局路径规划.....	62
5.2.1	定位.....	62
5.2.2	全局路径规划.....	66
5.2.3	路径控制.....	67
5.2.4	实时地图显示.....	69
5.2.5	指定路径程序.....	70
5.2.6	示例程序.....	71
5.3	局部路径规划.....	73
5.4	习题及参考答案.....	74
5.4.1	习题.....	74
5.4.2	参考答案.....	75
	第三部分 技术手册 .....	76
6	车辆控制 .....	76
6.1	原理介绍.....	76
6.2	示例：键盘控制小车行走.....	76
6.2.1	主程序"car_control.py" .....	76
6.2.2	程序运行.....	78
6.3	示例：车道线巡线车辆控制.....	79
6.3.1	主程序"decision_control.py" .....	79
6.3.2	程序执行.....	81
6.4	示例：键盘控制小车行驶里程和速度.....	82
6.4.1	主程序"sam_control.py" .....	82
6.4.2	主程序"decision_control.py" .....	83
6.4.3	程序执行.....	86

6.5	示例：激光雷达障碍物检测与车辆控制.....	88
6.5.1	主程序"decision_control.py" .....	88
6.5.2	程序执行.....	90
6.6	示例：全局路径规划与车辆控制.....	90
6.6.1	主程序"decision_control.py" .....	90
6.6.2	程序执行.....	93
6.7	示例：目标检测与车辆控制.....	95
6.7.1	主程序"decision_control.py" .....	95
6.7.2	程序执行.....	98
6.8	车辆决策控制综合算法流程.....	99
7	电机与编码器 .....	101
7.1	电机控制.....	101
7.1.1	主机处理程序.....	101
7.2	串口通信.....	105
7.2.1	速度通信.....	105
7.2.2	编码器通信.....	106
7.3	编码器数值操作.....	108
7.3.1	初始化函数.....	108
7.3.2	数值操作函数.....	109
8	摄像头.....	110
8.1	车载相机介绍.....	110
8.2	相机读取图像.....	111
8.2.1	打开相机.....	111
8.2.2	ros 订阅相机发出的消息并显示图像.....	111
9	激光雷达 .....	113
9.1	雷达数据读取.....	114
9.1.1	程序运行.....	114
9.1.2	节点主程序 "note.cpp" .....	116
9.2	雷达障碍物检测.....	121

9.2.1	程序运行.....	122
9.2.2	主程序 "my_pcl_node.cpp" .....	124
10	车道线提取 .....	127
10.1	原理介绍.....	127
10.2	程序运行过程.....	127
10.3	车道线提取程序及讲解.....	128
10.3.1	主程序.....	128
10.3.2	各函数讲解.....	131
10.3.3	巡线控制教程——转向量计算.....	141
11	目标检测 .....	143
11.1	目标检测的基础——卷积神经网络.....	143
11.1.1	二维互相关运算.....	143
11.1.2	二维卷积层.....	144
11.1.3	多输入通道和多输出通道.....	144
11.1.4	池化层.....	145
11.2	深度学习基础.....	146
11.2.1	线性回归.....	146
11.2.2	softmax 回归.....	148
11.2.3	思考.....	151
11.3	目标检测程序及讲解.....	151
11.3.1	参数设定.....	151
11.3.2	模型声明.....	152
11.3.3	加载数据.....	152
11.3.4	导入图像进行检测.....	152
11.3.5	设置颜色.....	153
11.3.6	图像标注.....	153
11.3.7	图像存储.....	154
11.3.8	消息订阅和发布.....	154
11.4	目标检测常用框架.....	155

11.4.1 RCNN .....	155
11.4.2 SPPNET .....	156
11.4.3 其他常用框架.....	156

# 第一部分 开机运行

## 1 无人小车平台介绍

本项目的无人小车平台是一款搭载相机传感器，利用图像处理，深度学习等技术，实现在室内模拟城市道路场景下自主行驶的轮式机器人小车。下面将分别介绍无人小车平台的硬件组件及软件系统。

### 1.1 硬件平台

#### 1.1.1 硬件清单

器件	数量
车体	1
轮胎及配件	4
激光雷达	1 (未安装在车上)
相机	1
传感器支架	1
充电器	1
电池	1 (已安装在车内)
7 寸显示屏	1

表格 1-1 无人小车平台硬件清单

#### 1.1.2 技术参数

电池	12800mah 容量聚合物锂电池
主控板	NVIDIA Jeston Nano
下位机	STM32
供电	电池输入 12v, 输出 5V
预留接口	USB
最大运行速度与载重	0.2 m/s, 载重 2 kg
续航时间	2-3h

充电时间	3-4h
高度	36cm
长度	37.5cm
宽度	25cm

表格 1-2 无人小车平台技术参数

### 1.1.3 整车

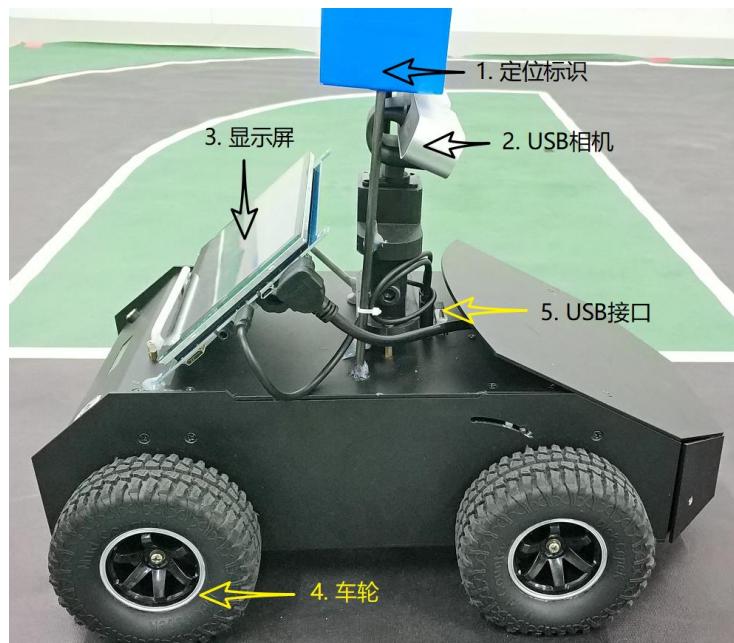


图 1-1 无人小车

如错误!未找到引用源。所示，小车搭载 USB 深度相机，车顶安装有定位标识，供外部摄像头检测定位，同时无人小车底盘上固定有显示屏，并提供 USB 接口，显示屏通过 USB 接口供电，无线键鼠接收器通过 USB 接口与车载电脑相连。

### 1.2 轮胎安装

首先将轮胎的联轴器带螺丝孔的一面和电机轴平的一面对齐，如图 1-2 所示。然后将车轮推到头，再使用六角扳手拧上螺丝固定，如图 1-3 所示。



图 1-2 轮胎安装



图 1-3 联轴器上紧螺丝

### 1.3 充电与开机

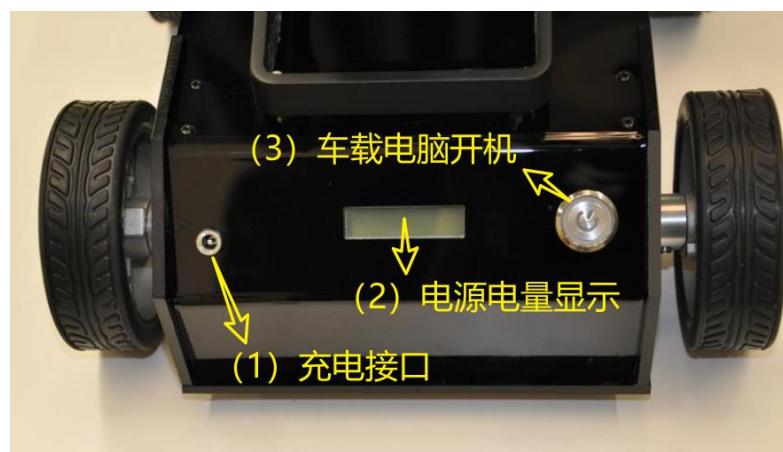


图 1-4 无人小车充电接口



图 1-5 充电器

如图 1-4 和图 1-5 所示，小车使用圆头充电器，充电电压为 12v，充电时间大约在 3-4 小时，续航时间大约在 3-4 小时。充电时充电器指示灯为红色，充满或者充电器未连接小车充电接口时，指示灯为绿色。开机键按下后可一键开启车载电脑 Jeston Nano，开机进入 ubuntu 18.04 系统，开机密码为 1234。同时小车上搭载有电量显示屏，可实时显示电源电量。

## 1.4 相机安装

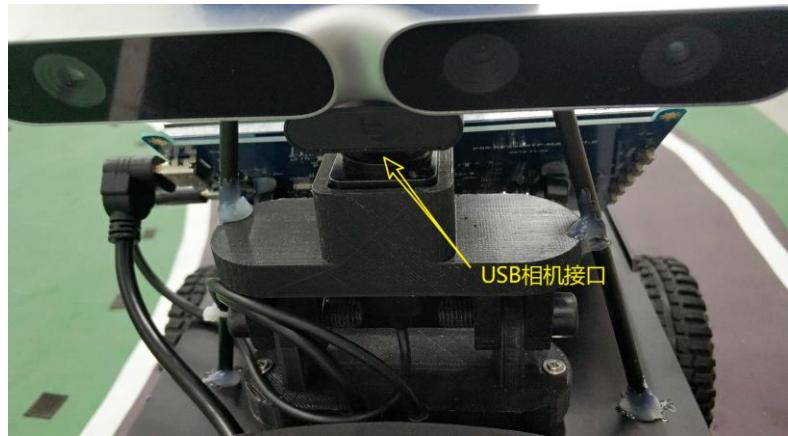


图 1-6 USB 相机

如图 1-6 所示，相机直接插入 USB 底座与小车进行连接。

## 1.5 软件平台

### 1.5.1 软件架构

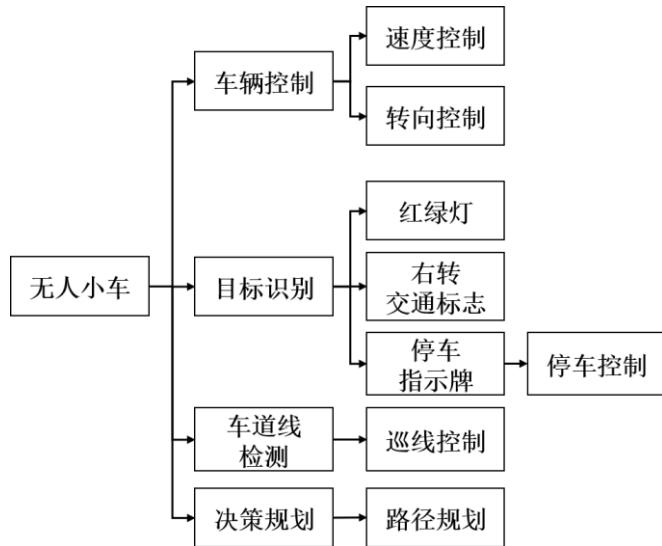


图 1-7 无人小车平台软件架构

无人小车平台的软件架构如图 1-7 所示，各部分软件功能如下：

- (1) **车辆控制：**如图 1-8 所示，无人小车底盘与四路电机相连，底盘已封装好电机控制电路，车载电脑 NVIDIA Jetson Nano 只需通过 USB 串口与底盘相连接，即可向下传输控制信号，驱动小车完成速度和转向控制。

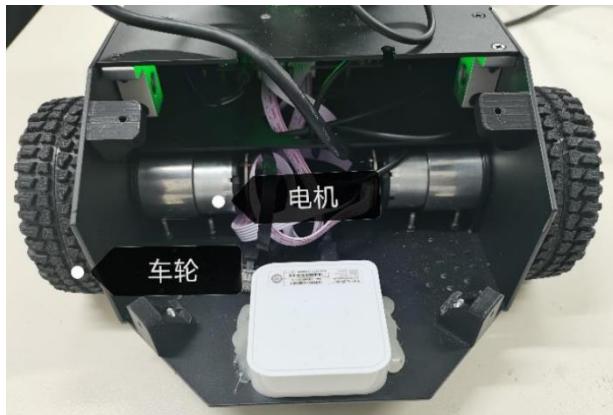


图 1-8 无人小车车轮与电机

- (2) **目标识别：**无人小车搭载相机传感器，通过深度学习算法处理采集的相机图像，以识别红绿灯、停车指示牌和右转标志这三类交通元素。其中，在识别到停车标志后，会传输信号给决策规划模块，以指示小车进入停车控制。



图 1-9 目标识别过程

(3) 车道线检测：无人小车通过相机传感器实时采集图像。在模拟的城市道路场景中，车道线与车道颜色与真实世界相近，车道线为显著的白色。在此基础上，应用图像处理算法可以将无人小车视野内的车道线检测出来，图 1-10 所示为车道线检测过程。

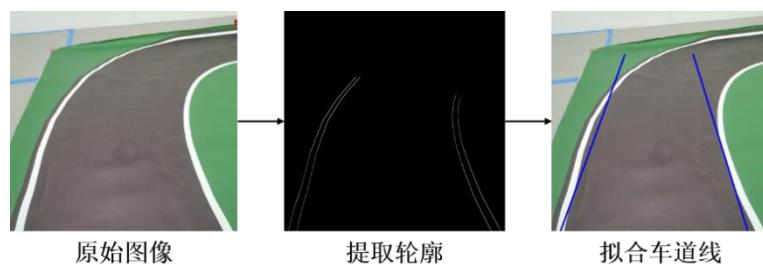


图 1-10 车道线检测

(4) 决策规划：无人小车在如图 1-11 所示的模拟城市道路环境内运行，其决策规划由程序指定，通过规划地图中不同路点，使得无人小车按不同轨迹自主行驶。



图 1-11 模拟地图内供决策规划的 5 个路点

### 1.5.2 程序运行

按下车载电脑的开机键，即可启动小车搭载的车载电脑 Jeston Nano，该主机为 Ubuntu 18.04 系统，属于 Linux 系统，按  $\text{Ctrl}+\text{Alt}+\text{T}$  可以启动 Ubuntu 系统的终端（Terminal）。在 Linux 系统中，一切任务都可以在终端（Terminal）

通过命令行完成，因此学习简单的 Linux 命令行指令，对于熟悉小车的软件系统有极大助力。同样，本无人小车的所有程序都需要首先启动 ubuntu 的终端，在终端环境下运行相应的程序，后文给出的程序运行指令都按照此标准执行，特就此说明。

### 1.5.2.1 源码组成与路径介绍

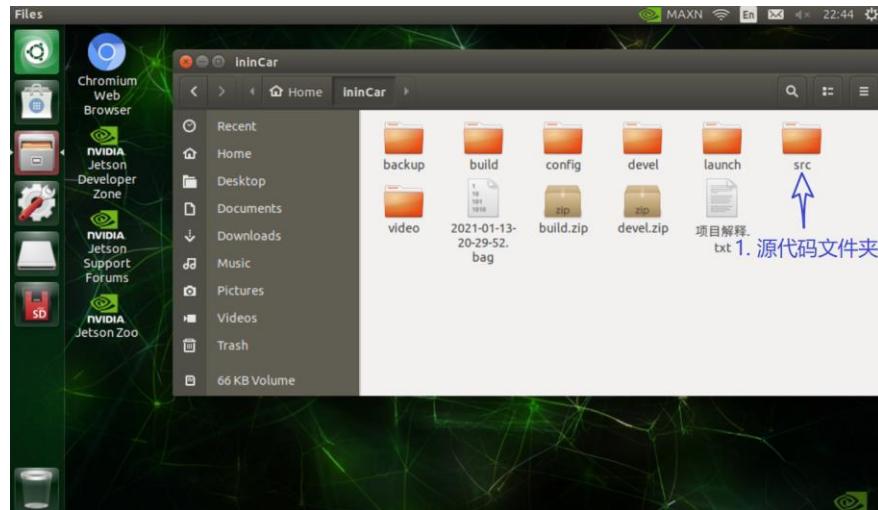


图 1-12 工程代码路径

小车的程序主要依赖于 ROS 操作系统进行开发，程序语言环境为 python 2.7，源程序主要放在 /home/deepcar/ininCar 文件夹下，该文件夹为 ROS 工作空间。其中，src 存放源代码，devel 文件夹包含环境变量及初始化配置等文件。

### 1.5.2.2 键盘控制小车运动状态程序启动

打开一个终端，在终端输入如下指令，启动键盘控制小车运动状态程序：

```
cd ininCar/  
roslaunch launch/1_keyboard_control_car.launch
```

启动后的终端运行效果及该程序运行的效果如下图所示：

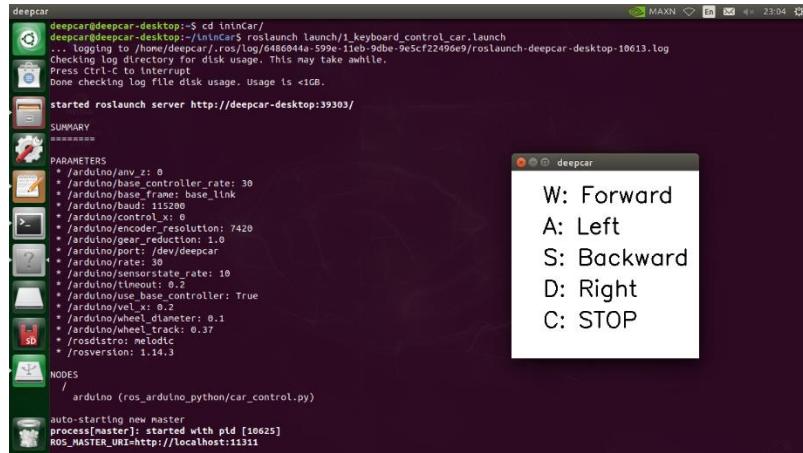


图 1-13 键盘控制小车运动状态程序显示结果

鼠标需要点击上图右侧的 OpenCV 库所构建的窗口，才可有效检测键盘按键。其中，W 键会使小车向前加速，S 键会使小车向后加速，A 键会使小车左转加速，D 键会使小车右转加速，C 键会使小车停止运动。长按按键才会执行上述功能，松开按键会在短暂停时间内停止运动。键盘控制终端显示结果如下图所示。

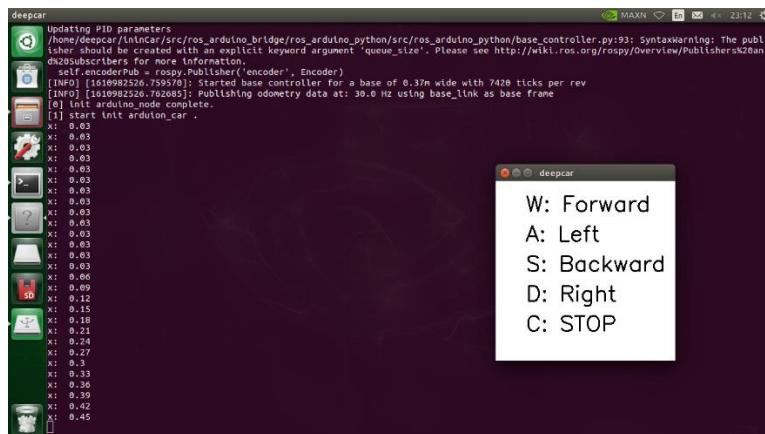


图 1-14 键盘控制终端显示结果

### 1.5.2.3 小车固定里程控制程序启动

打开一个终端，在终端输入如下指令，启动小车固定里程控制程序：

```
cd ininCar/
roslaunch launch/ 2_length_control.launch
```

启动后的终端运行效果及该程序运行的效果如下图所示：

```

launch/2_length_control.launch http://localhost:11311
deepcar@deepcar-desktop:~/inincar$ cd inincar/
deepcar@deepcar-desktop:~/inincar$ roslaunch launch/2_length_control.launch
[INFO] [ros@192.168.1.11-11311]: Logging to /home/deepcar/.ros/log/d837b6cc-599e-11eb-9efc-9e5cf2249de9/roslaunch-deepcar-desktop-10968.log
[INFO] [ros@192.168.1.11-11311]: Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://deepcar-desktop:36669/
SUMMARY
======
PARAMETERS
  * /arduno/base_controller_rate: 30
  * /arduno/base_frame: base_link
  * /arduno/baud: 115200
  * /arduno/car_theta: 0.0
  * /arduno/gear_reduction: 1.0
  * /arduno/encoder_resolution: 7420
  * /arduno/gear_reduction: 1.0
  * /arduno/port: /dev/deepcar
  * /arduno/rate: 30
  * /arduno/sensorstate_rate: 10
  * /arduno/use_base_controller: 1.0
  * /arduno/vel_x: 0.2
  * /arduno/wheel_diameter: 0.1
  * /arduno/wheel_track: 0.37
  * /rosdistro: melodic
  * /rosversion: 1.14.3
NODES
  /
    arduino (ros_arduino_python/length_control.py)

```

图 1-15 小车固定里程控制程序显示结果

运行该程序后，小车会以 0.2 的前进速度前进 0.8m 后停车。程序运行终端显示结果如下图所示。

```

launch/2_length_control.launch http://localhost:11311
[odom] x: 0.241 m ; y: 0.007 m ; theta: 0.016 degree
[odom] x: 0.277 m ; y: 0.002 m ; theta: 0.019 degree
[odom] x: 0.296 m ; y: 0.002 m ; theta: 0.020 degree
[odom] x: 0.314 m ; y: 0.003 m ; theta: 0.021 degree
[odom] x: 0.332 m ; y: 0.003 m ; theta: 0.022 degree
[odom] x: 0.350 m ; y: 0.004 m ; theta: 0.023 degree
[odom] x: 0.368 m ; y: 0.004 m ; theta: 0.027 degree
[odom] x: 0.386 m ; y: 0.005 m ; theta: 0.028 degree
[odom] x: 0.404 m ; y: 0.005 m ; theta: 0.030 degree
[odom] x: 0.422 m ; y: 0.006 m ; theta: 0.032 degree
[odom] x: 0.440 m ; y: 0.006 m ; theta: 0.034 degree
[odom] x: 0.459 m ; y: 0.007 m ; theta: 0.035 degree
[odom] x: 0.478 m ; y: 0.007 m ; theta: 0.036 degree
[odom] x: 0.495 m ; y: 0.008 m ; theta: 0.038 degree
[odom] x: 0.513 m ; y: 0.009 m ; theta: 0.040 degree
[odom] x: 0.531 m ; y: 0.009 m ; theta: 0.041 degree
[odom] x: 0.549 m ; y: 0.010 m ; theta: 0.043 degree
[odom] x: 0.566 m ; y: 0.012 m ; theta: 0.044 degree
[odom] x: 0.684 m ; y: 0.012 m ; theta: 0.045 degree
[odom] x: 0.623 m ; y: 0.013 m ; theta: 0.046 degree
[odom] x: 0.641 m ; y: 0.014 m ; theta: 0.048 degree
[odom] x: 0.659 m ; y: 0.015 m ; theta: 0.050 degree
[odom] x: 0.677 m ; y: 0.016 m ; theta: 0.052 degree
[odom] x: 0.695 m ; y: 0.017 m ; theta: 0.054 degree
[odom] x: 0.713 m ; y: 0.019 m ; theta: 0.056 degree
[odom] x: 0.731 m ; y: 0.020 m ; theta: 0.058 degree
[odom] x: 0.749 m ; y: 0.020 m ; theta: 0.060 degree
[odom] x: 0.768 m ; y: 0.021 m ; theta: 0.062 degree
[odom] x: 0.786 m ; y: 0.022 m ; theta: 0.063 degree
[odom] x: 0.804 m ; y: 0.023 m ; theta: 0.065 degree
[INFO] [1516982442.581598]: Shutting down Arduino Node...
stopping the robot.
[Arduino-2] process has finished cleanly
log file: /home/deepcar/.ros/log/d837b6cc-599e-11eb-9efc-9e5cf2249de9/arduino-2*.log

```

图 1-16 程序运行终端显示结果

#### 1.5.2.4 相机程序启动

下面先介绍相机程序的启动流程。

打开一个终端，在终端输入如下指令，打开相机：

```
cd inincar/
roslaunch astra_camera astrapro.launch
```

终端程序运行效果如下图所示：

```

/home/deepcar/inInCar/src/roslaunch/astrapro.launch http://localhost:11311 MAXN En 22:31
deepcar@deepcar-desktop:~/inInCar$ /home/deepcar/inInCar/src/roslaunch/astrapro.launch http://localhost:11311
deepcar@deepcar-desktop:~/inInCar$ rosrun astra_camera astrapro.launch
WARNING: Package name "detectionInfo_msgs" does not follow the naming conventions. It should start with a lower case letter and only contain lower case letters, digits, underscores, and dashes.
... logging to /home/deepcar/.ros/log/af0ebd5a-55ab-11eb-93d2-9e5cf22496e9/roslaunch-deepcar-desktop-1449.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

WARNING: Package name "detectionInfo_msgs" does not follow the naming conventions. It should start with a lower case letter and only contain lower case letters, digits, underscores, and dashes.
started roslaunch server http://deepcar-desktop:42485/
SUMMARY
=====
PARAMETERS
  * /camera/camera_nodelet_manager/num_worker_threads: 4
  * /camera/camera_rgb/camera_info_url:
  * /camera/camera_rgb/frame_rate: 30
  * /camera/camera_rgb/height: 480
  * /camera/camera_rgb/index: 0
  * /camera/camera_rgb/product: 0x0502
  * /camera/camera_rgb/serial: 0
  * /camera/camera_rgb/timestamp_method: start
  * /camera/camera_rgb/vendor: 0x2bc5
  * /camera/camera_rgb/video_mode: yuyv
  * /camera/camera_rgb/width: 640
  * /camera/depth_rectify_depth/interpolation: 0

```

图 1-17 在终端启动相机

我们可以通过显示相机发出的图像消息判断相机是否已经启动。

可以使用 ros 的 rviz 显示图像。rviz 是 ros 自带的一个图形化工具,可以方便的对 ros 的程序进行图形化操作。具体操作如下:

- 打开新的终端, 在终端输入如下指令打开 rviz:

rviz

- 在 rviz 的左边界面找到“Add”按钮, 点击该按钮。

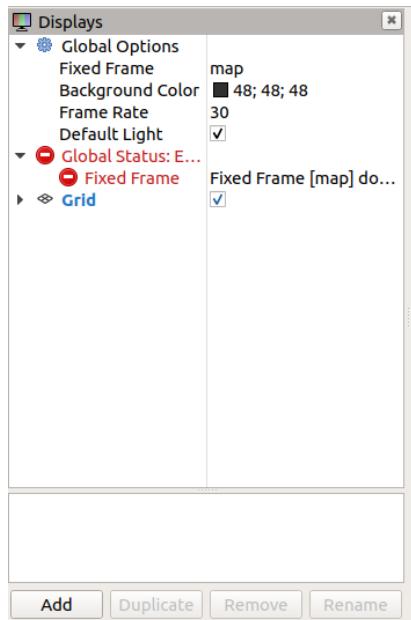


图 1-18 rviz 界面选择要播放的 topic

- 在弹出的界面中找到/rgb 下/image\_raw 中的 Image, 双击它。

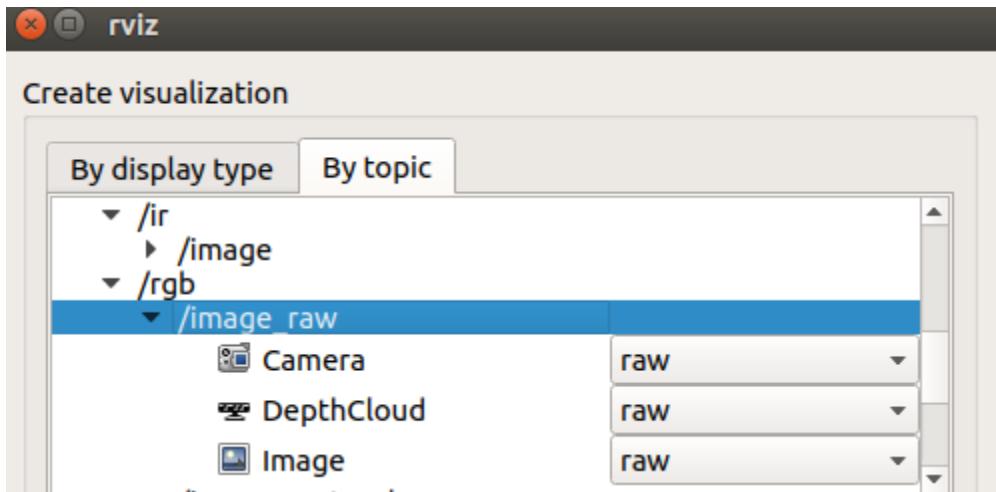


图 1-19 选择相机的 topic: /rgb/image\_raw/image

可以看到相机发出的图像消息已经显示出来，这说明相机成功启动，并能够向外发送图像消息。



图 1-20 相机显示的图像

#### 1.5.2.5 目标检测程序启动

目标（红绿灯、交通标志）识别启动程序内容如下：

```
cd ininCar/  
source devel/setup.bash  
roslaunch launch/14_object_detection_astra.launch  
... (启动此命令后需要等待几分钟才可看到目标检测结果)
```

这样就可以开启目标检测代码，但是目标检测还需要使用摄像头，所以如果没有开启摄像头需要先开一个终端单独开启摄像头。注意：由于车载电脑算力较低，程序需要花费几分钟来启动 GPU 相关运算模块。

目标识别程序运行界面如下图所示：

图 1-21 目标识别程序运行界面示意图

检测结果如下图所示：



图 1-22 目标检测结果界面

#### 1.5.2.6 巡线检测程序启动

### 1) 打开相机

要进行巡线检测，需要先启动相机（详见相机启动程序小节）。

## 2) 启动巡线程序

打开一个终端，在终端输入如下指令，启动巡线程序：

```
cd ininCar/  
roslaunch launch/13_xunxian.launch
```

启动后的终端运行效果及巡线程序运行的效果如下图所示：

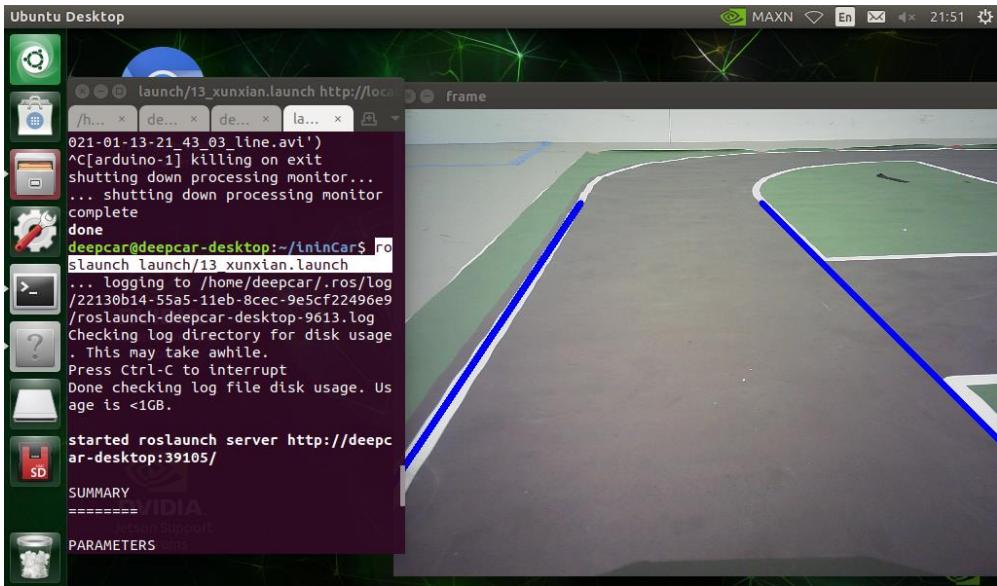


图 1-23 巡线程序显示结果

### 1.5.2.7 键盘控制小车速度和里程程序启动

首先启动相机；随后，打开一个终端，在终端输入如下指令，启动键盘控制小车速度和里程程序：

```
cd ininCar/  
roslaunch launch/ 9_speedANDmile_control.launch
```

启动后的终端运行效果及该程序运行的效果如下图所示：

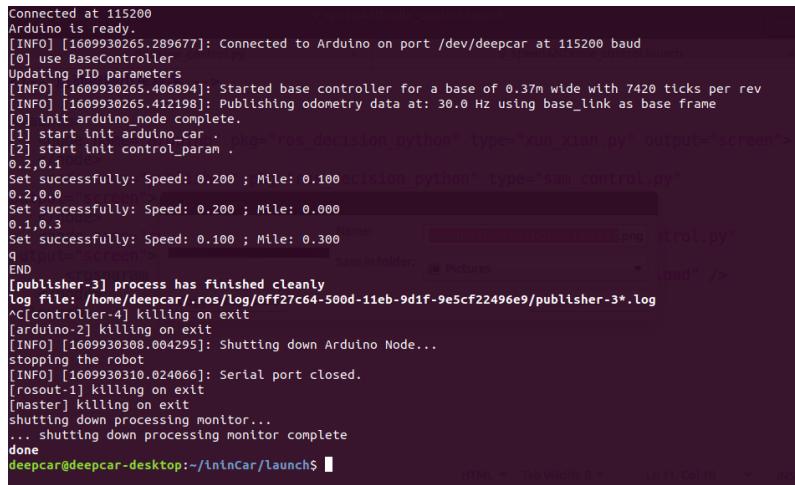
```
deepcar@deepcar-desktop:~$ cd ininCar/  
deepcar@deepcar-desktop:~/ininCar$ rosrun launch/9_speedANDmile_control.launch  
... logging to /home/deepcar/.ros/log/0991d4bc-5980-11eb-b081-9e5cf22496e9/roslaunch-deepcar-desktop-17388.log  
Checking log directory for disk usage  
. This may take awhile.  
Press Ctrl-C to interrupt  
Done checking log file disk usage. Usage is <1GB.  
started roslaunch server http://deepcar-desktop:38057/  
SUMMARY  
=====
```

图 1-24 键盘控制小车速度和里程程序显示结果

运行该程序后，可以通过人为设定所需的小车前进速度和行驶里程，控制小车运动。输入的格式为：

```
# 速度,里程  
# 例如：0.2,0.8 表示以 0.2 的速度行驶，行驶里程为 0.8m
```

程序运行终端显示结果如下图所示。



```
Connected at 115200
Arduino is ready.
[INFO] [1609930265.289677]: Connected to Arduino on port /dev/deepcar at 115200 baud
[0] use BaseController
Updating PID parameters
[INFO] [1609930265.406894]: Started base controller for a base of 0.37m wide with 7420 ticks per rev
[INFO] [1609930265.412198]: Publishing odometry data at: 30.0 Hz using base_link as base frame
[0] init arduino_node complete.
[1] start init arduino_car .
[2] start init control_param .
0.2,0.1
Set successfully: Speed: 0.200 ; Mile: 0.100
0.2,0.0
Set successfully: Speed: 0.200 ; Mile: 0.000
0.1,0.3
Set successfully: Speed: 0.100 ; Mile: 0.300
q
END
[publisher-3] process has finished cleanly
log file: /home/deepcar/.ros/log/0ff27c64-500d-11eb-9d1f-9e5cf22496e9/publisher-3*.log
^C[controller-4] killing on exit
[arduino-2] killing on exit
[INFO] [1609930308.004295]: Shutting down Arduino Node...
stopping the robot
[INFO] [1609930310.024066]: Serial port closed.
[rosout-1] killing on exit
[master] killing on exit
shutting down processing monitor...
... shutting down processing monitor complete
done
deepcar@deepcar-desktop:~/inincar/launch$
```

图 1-25 程序运行终端显示结果

### 1.5.2.8 全局规划程序启动

#### 1) 打开相机

全局规划程序需要和巡线程序相互配合，故也需要打开相机，步骤相同。

#### 2) 启动巡线程序

全局路径规划有两种程序。第一种是最短路径规划，输入期望位置之后小车将沿着最短路径到达期望位置（详见全局路径规划小节）；第二种是指定路径规划，路径在程序中可更改，小车将按照指定路径行驶（详见指定路径规划小节）。

对于第一种最短路径规划，打开一个终端，在终端输入如下指令，启动巡线程序：

```
cd inincar/
roslaunch launch/11_global_path_planning.launch
```

启动后的终端运行效果如下图所示：

```

Terminal
deepcar@deepcar-desktop:~/inInCar
deepcar@deepcar-desktop:~$ cd inInCar/
deepcar@deepcar-desktop:~/inInCar$ rosrun navigation 11_global_path_planning.launch
... logging to /home/deepcar/.ros/log/7d6bef46-599a-11eb-a125-9e5cf22496e9/roslaunch-deepcar-desktop-11454.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://deepcar-desktop:39315/
SUMMARY
=====
PARAMETERS
  * /controller/anv_z: 0
  * /controller/base_controller_rate: 30
  * /controller/base_frame: base_link
  * /controller/baud: 115200
  * /controller/control_x: 0
  * /controller/encoder_resolution: 7420
  * /controller/gear_reduction: 1.0
  * /controller/port: /dev深深车
  * /controller/rate: 30
car_xy = img_car_binary.nonzero()
#如果检测到了

```

图 1-26 最短路径规划程序启动结果

对于第二种指定路径规划，打开一个终端，在终端输入如下指令：

```

cd inInCar/
rosrun navigation 12_global_path_planning.launch

```

启动后的终端运行效果如下图所示：

```

Terminal
deepcar@deepcar-desktop:~/inInCar
deepcar@deepcar-desktop:~$ cd inInCar/
deepcar@deepcar-desktop:~/inInCar$ rosrun navigation 12_global_path_planning.launch
... logging to /home/deepcar/.ros/log/c53664d2-599a-11eb-b793-9e5cf22496e9/roslaunch-deepcar-desktop-11910.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://deepcar-desktop:43187/
SUMMARY
=====
PARAMETERS
  * /controller/anv_z: 0
  * /controller/base_controller_rate: 30
  * /controller/base_frame: base_link
  * /controller/baud: 115200
  * /controller/control_x: 0
  * /controller/encoder_resolution: 7420
  * /controller/gear_reduction: 1.0
  * /controller/port: /dev深深车
  * /controller/rate: 30
car_xy = img_car_binary.nonzero()
#如果检测到了

```

图 1-27 指定路径规划程序启动结果

### 1.5.2.9 外部相机标定介绍

对于小车的定位，考虑了多种方式，权衡利弊之后最终选用颜色识别定位。使用正面的摄像头观察整个地图，识别到小车顶部的蓝色标志物，根据标志物在摄像头图像中的位置对小车进行定位。

摄像头标定的目的就是能够根据小车在摄像头中的位置，完成对小车位置的表示输出，为后续的全局路径规划奠定基础。基础的程序已经基本完成，如果移

动了摄像头或地图的位置，则需要对摄像头进行重新标定，具体标定步骤如下。

### A 摄像头测试与安装

摄像头将 USB 摄像头插入图传模块，并使用 5V 电源对图传模块进行供电。

连接完成之后可以测试摄像头是否正常运行。在浏览器界面输入网址 <http://192.168.1.1:8080/?action=stream>，如果能观察到正常的摄像头图像界面即表示成功。

接着将全局摄像车悬挂在地图纵向中线的正面斜上方位置，合理摆放摄像头的位置和姿态，确保摄像头能观察到整个地图，并且确保小车在地图中时，能观察到整个小车。摄像头安装之后如下图所示：



图 1-28 安装好的全局摄像头——背面



图 1-29 安装好的全局摄像头——正面

摄像头安装完成之后，需要修改定位程序 "deepcar\_location.py" 中的一些

参数，程序位于路径 `inincar/src/global_path_planning/path_planning/srcs` 底下。

主要修改的参数如下：

```
self.h_camera = 208 #摄像头高度 (20 行)
self.y_camera_low = 349 #摄像头到地图底线的距离 (22 行)
```

## B 程序准备

为了之后的调试顺利，需要先取消注释程序中一些被注释的地方（完成所有标定任务之后，可重新注释）：

```
cv2.circle(frame, tuple(self pts1[0]), 3, (0, 0, 255), 3, 8) #174 行
cv2.circle(frame, tuple(self pts1[1]), 3, (0, 0, 255), 3, 8)
cv2.circle(frame, tuple(self pts1[2]), 3, (0, 0, 255), 3, 8)
cv2.circle(frame, tuple(self pts1[3]), 3, (0, 0, 255), 3, 8)

cv2.line(frame, tuple(self pts1[0]), tuple(self pts1[1]), (0, 0, 255),
3)
cv2.line(frame, tuple(self pts1[1]), tuple(self pts1[3]), (0, 0, 255),
3)
cv2.line(frame, tuple(self pts1[0]), tuple(self pts1[2]), (0, 0, 255),
3)
cv2.line(frame, tuple(self pts1[2]), tuple(self pts1[3]), (0, 0, 255),
3) #182 行

cv2.imshow('video', frame) #196 行

print self.x_mean_true, self.y_mean_true, "true" #219 行
```

接下来的部分可能需要单独运行程序来进行调试，运行方法为，打开一个终端，在终端输入如下指令，启动巡线程序：

```
cd inincar/
rosrun path_planning deepcar_location.py
```

## C 感兴趣区域调整

摄像头观察到的图像除去必要的地图之外，还存在很多周围环境的噪声，所以对感兴趣区域的提取很必要的。选取的原则为：保证选取的区域在实际中是一个矩形，小车标志物在小车运动的过程中不会超过所选区域的范围。如可以选取地图底边为矩形的底边，选取距离地图左右白线 20cm 的直线作为矩形的左右边，选取距离地图顶边 166cm 的直线作为矩形的顶边，如图所示：

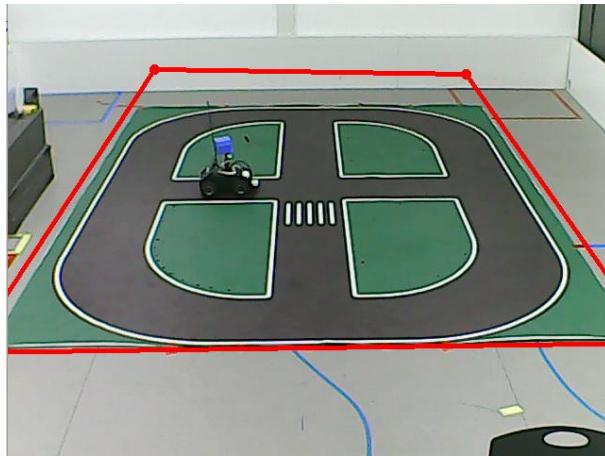


图 1-30 红框内感兴趣区域

改变感兴趣区域需要修改程序 "deepcar\_location.py" 中的参数:

```
# 原图的四个角点(左上、右上、左下、右下),与变换后矩阵位置 (27、28、29 行)
self.pts1 = np.float32([[155, 55], [473, 50], [-45, 360], [686, 350]])
self.pts2 = np.float32([[0, 0],[self.w_img_map,0],[0,
self.h_img_map],[self.w_img_map,self.h_img_map]])
```

提取完成之后，需要根据提取到的感兴趣区域实际的长度和宽度在程序 "deepcar\_location.py" 中进行一些参数的修改:

```
self.h_img_map = 535 # 提取的地图长度 (24 行)
self.w_img_map = 317 # 提取的地图宽度 (25 行)
```

因为地图在摄像头视角下并不是一个完好的俯视图，所以需要使用透视变换得到一个俯视图。完成上述操作和参数修改之后，已经可以得到了一个较好的俯视图。

#### D 具体端点标定

在完成透视变换之后经过一系列操作得到了小车的大致坐标位置。但相对于小车的实际位置还是会有些偏差，所以需要在 5 个端点出对小车进行精准标定。在每个端点出读取中断发送出来的位置，修改程序中参数:

```
self.x_2 = 21 #48 行
self.x_3 = 138
self.x_4 = 254
self.y_1 = 42
self.y_2 = 211
self.y_3 = 204
self.y_4 = 204
```

```
self.y_5 = 373 #55 行
```

标定时将小车放置在每个端点的中心位置，运行定位程序，根据终端发送出来的坐标位置更改上面的参数，如图为在端点 2、3、4 时的调试过程：

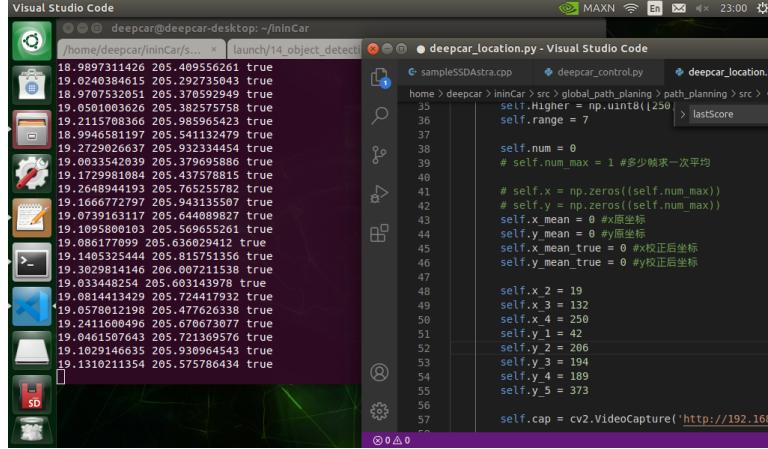


图 1-31 端点 2 的调试过程

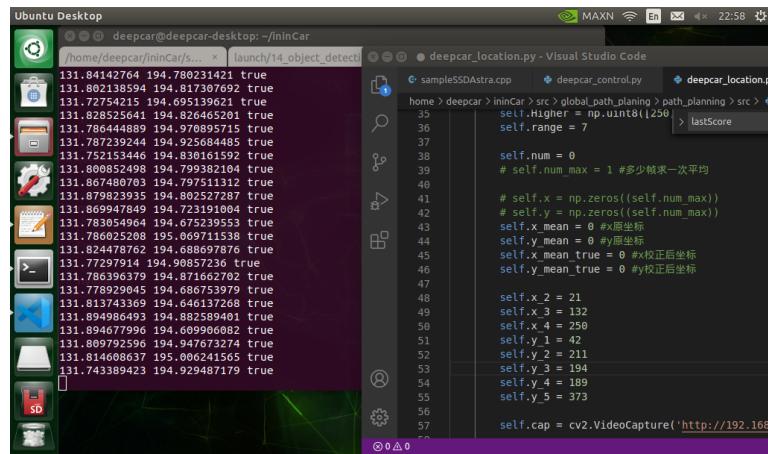


图 1-32 端点 3 的调试过程

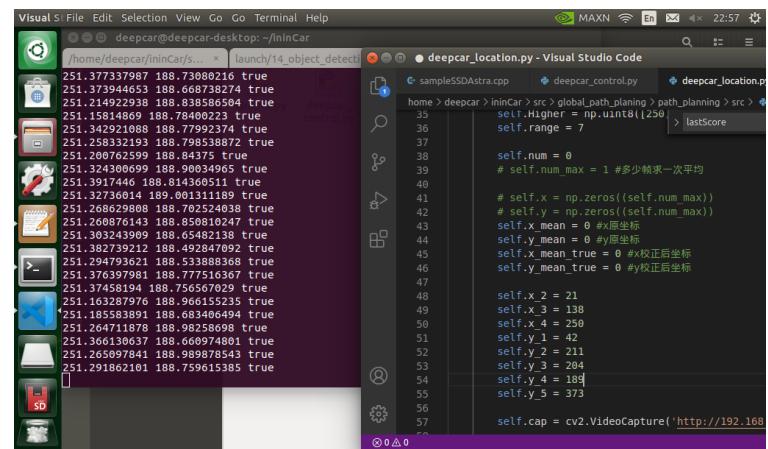


图 1-33 端点 4 的调试过程

完成以上参数修改之后，可以实现对小车的定位。如果出现细节上的问题，

则需自行参阅实训手册、技术方案和程序的全局路径规划部分，了解具体的标定思路进行调整。

#### 1.5.2.10 启动程序注意事项

巡线时候小车的速度是固定的，目标检测程序启动需要好几分钟，目标检测程序消耗的算力较大，会造成其他程序延时提高，因此目前小车仅提供运行纯巡线和目标检测的程序或纯巡线和路径规划的程序。

## 第二部分 实训教程

# 2 开发工具入门与实践

本项目主要使用 ROS 机器人操作系统作为软件开发工具，下面先对 ROS 操作系统做简单介绍。

## 2.1 ROS 操作系统基础入门

### 2.1.1 ROS 概述

Robot Operating System (ROS) 是一个得到广泛使用的机器人系统的软件框架。ROS 的基本思想是无须改动就能够在不同的机器人上复用代码。基于此，我们就可以在不同的机器人上分享和复用已经实现的功能，而不需要做太多的工作，这避免了重复劳动。2007 年，斯坦福大学人工智能实验室 (SAIL) 在斯坦福 AI 机器人的支持下开发了 ROS。2008 年之后，Willow Garage 继续开发 ROS，如今开源机器人基金会 (OSRF) 开始接管 ROS 及其相关工程的维护工作，也包括新功能的开发。

ROS 提供了一个标准的操作系统环境，包括硬件抽象、底层设备控制、通用功能的实现、进程间消息转发和使用 catkin 和 cmake 管理功能包等。它基于一个集中式拓扑的图结构，其中处理节点与其他节点之间在通信图网络上接收和发布的信息。节点是任意进程，它从传感器读取数据、控制执行器，或运行用于在环境中自主映射或导航的高级复杂的机器人或视觉算法。

ROS 是一个基于 BSD (Berkeley Software Distribution) 开源协议的开源软件。无论是商业应用还是科学研究它都是免费的。应用 ROS 可以使用库中的代码，改进后再次共享，这种观念就是开源软件的本质。

## 2.1.2 ROS 安装

若要在系统中安装 ROS，可根据链接 <http://wiki.ros.org/Installation/Ubuntu> 中的指导来完成。主要包括以下七个步骤：

1. 设置系统软件源，修改为国内源。
2. 添加 ROS 的软件源地址到 sources.list 文件中。
3. 安装软件包授权密钥。
4. 更新软件源信息。
5. 安装 ROS 软件包。
6. 初始化 rosdep。
7. 设置环境变量。

## 2.1.3 ROS 入门实例

在 ROS 安装完成后，我们需要测试 ROS 是否安装成功。

1. 打开一个终端，输入以下命令，初始化 ROS 环境：

```
roscore
```

2. 再打开一个新的终端，输入以下命令，弹出一个小乌龟窗口：

```
rosrun turtlesim turtlesim_node
```

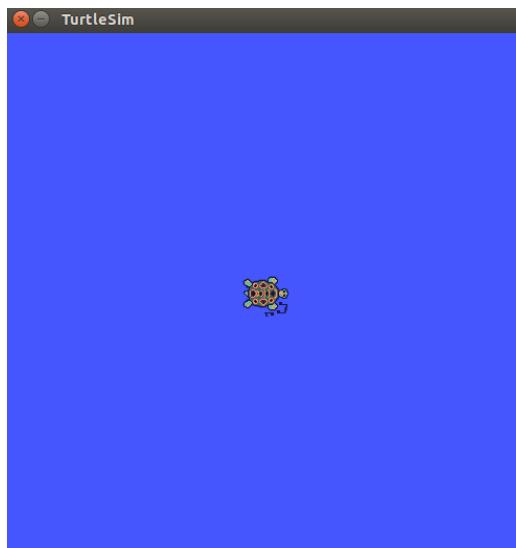


图 2-1 小乌龟窗口

3. 再打开一个新的终端，输入以下命令，可以在终端中通过方向键控小鸟

龟的移动：

```
rosrun turtlesim turtle_teleop_key
```

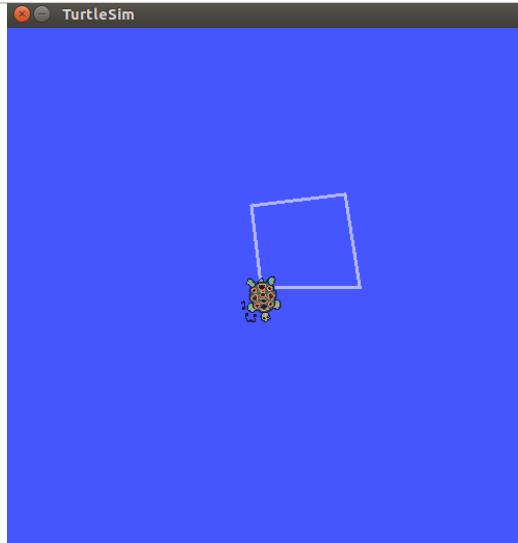


图 2-2 小乌龟移动示意图

4. 再打开新的终端，输入以下命令，弹出新的窗口查看 ROS 节点信息：

```
rosrun rqt_graph rqt_graph
```

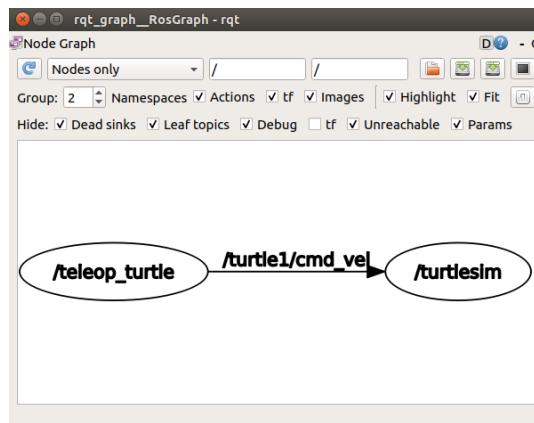


图 2-3 ROS 节点信息

分多个终端的目的是让多个指令同时运行。如果一切正常，证明 ROS 已成功安装，可以开始接下来的学习。

#### 2.1.4 ROS 架构

ROS 的架构经过设计并划分成了三部分，每一部分都代表一个层级的概念：

- 文件系统级 (Filesystem level)
- 计算图级 (Computation Graph level)
- 社区级 (Community level)

下面，对这三个层级分别进行介绍。

### 2.1.5 ROS 文件系统级

与其他操作系统相类似，一个 ROS 程序的不同组件要被放在不同的文件夹下。这些文件夹是根据功能的不同来对文件进行组织的：

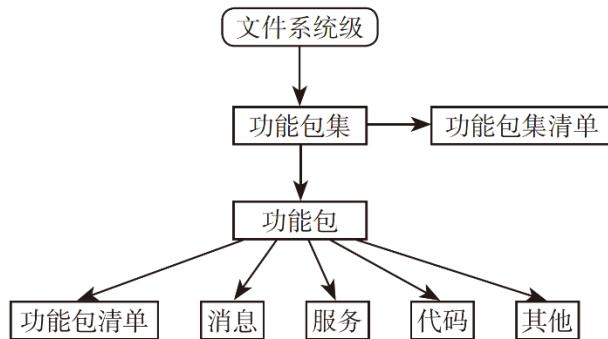


图 2-4 ROS 文件系统

- **功能包 (Package)**：功能包是 ROS 中软件组织的基本形式。一个功能包具有最小的结构和最少的内容，用于创建 ROS 程序，它可以包含 ROS 运行的进程（节点）、配置文件等。
- **功能包清单 (Manifest)**：功能包清单提供关于功能包、许可信息、依赖关系、编译标志等的信息。功能包清单是一个 manifests.xml 文件，通过这个文件能够实现对功能包的管理。
- **功能包集 (Stack)**：如果你将几个具有某些功能的功能包组织在一起，那么你将会获得一个功能包集。在 ROS 系统中，存在大量的不同用途的功能包集，例如导航功能包集。
- **功能包集清单 (Stack manifest)**：功能包集清单 (stack.xml) 提供一个关于功能包集的清单，包括开源代码的许可证信息、与其他功能包集的依赖关系等。
- **消息类型 (Message /msg type)**：消息是一个进程发送到其他进程的信息。ROS 系统有很多的标准类型消息。消息类型的说明存储在对应功能包的 msg 文件夹下。

- **服务类型 (Service /srv type)**：对服务的类型进行描述说明的文件在 ROS 系统中定义了服务的请求和响应的数据结构。这些描述说明存储在对应功能包的 srv 文件夹下。

### 2.1.6 ROS 计算图级

ROS 会创建一个连接到所有进程的网络。在系统中的任何节点都可以访问此网络，并通过该网络与其他节点交互，获取其他节点发布的信息，并将自身数据发布到网络上。

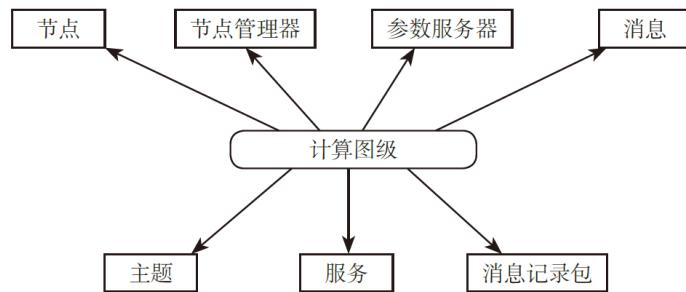


图 2-5 ROS 计算图级

在这一层级中最基本的概念包括节点、节点管理器、参数服务器、消息、服务、主题和消息记录包，这些概念都以不同的方式向计算图级提供数据：

- **节点 (Node)**：节点是主要的计算执行进程。如果你想要有一个可以与其他节点进行交互的进程，那么你需要创建一个节点，并将此节点连接到 ROS 网络。通常情况下，系统包含能够实现不同功能的多个节点。你最好让每一个节点都具有特定的单一的功能，而不是在系统中创建一个包罗万象的大节点。节点需要使用如 roscpp 或 rospy 的 ROS 客户端库进行编写。
- **节点管理器 (Master)**：节点管理器用于节点的名称注册和查找等。如果在你的整个 ROS 系统中没有节点管理器，就不会有节点、服务、消息之间的通信。需要注意的是，由于 ROS 本身就是一个分布式网络系统，你可以在某一台计算机上运行节点管理器，在其他计算机上运行由该管理器管理的节点。
- **参数服务器 (Parameter Server)**：参数服务器能够使数据通过关键词存储在一个系统的中心位置。通过使用参数，就能在运行时配置节

点或改变节点的工作任务。

- **消息 (Message)**：节点通过消息完成彼此的沟通。消息包含一个节点发送到其他节点的数据信息。ROS 中包含很多种标准类型的消息，同时你也可以基于标准消息开发自定义类型的消息。
- **主题 (Topic)**：主题是由 ROS 网络对消息进行路由和消息管理的数据总线。每一条消息都要发布到相应的主题。当一个节点发送数据时，我们就说该节点正在向主题发布消息。节点可以通过订阅某个主题，接收来自其他节点的消息。一个节点可以订阅一个主题，而并不需要该节点同时发布该主题。这就保证了消息的发布者和订阅者之间相互解耦，完全无需知晓对方的存在。主题的名称必须是独一无二的，否则在同名主题之间的消息路由就会发生错误。
- **服务 (Service)**：在发布主题时，正在发送的数据能够以多对多的方式交互。但当你需要从某个节点获得一个请求或应答时，就不能通过主题来实现了。在这种情况下，服务能够允许我们直接与某个节点进行交互。此外，服务必须有一个唯一的名称。当一个节点提供某个服务时，所有的节点都可以通过使用 ROS 客户端库所编写的代码与它通信。
- **消息记录包 (Bag)**：消息记录包是一种用于保存和回放 ROS 消息数据的文件格式。消息记录包是一种用于存储数据的重要机制。它能够获取并记录各种难以收集的传感器数据。我们可以通过消息记录包反复获取实验数据，进行必要的开发和算法测试。在使用复杂机器人进行实验工作时，需要经常使用消息记录包。

### 2.1.7 ROS 社区级

ROS 开源社区级的概念主要关于 ROS 资源，能够通过独立的网络社区分享软件和知识。这些资源包括：

- **发行版 (Distribution)**：ROS 发行版是可以独立安装的、带有版本号的一系列功能包集。ROS 发行版像 Linux 发行版一样发挥类似的作用。这使得 ROS 软件安装更加容易，而且能够通过一个软件集合来维持一致的版本。

- **软件源 (Repository)**：ROS 依赖于共享开源代码与软件源的网站或主机服务，在这里不同的机构能够发布和分享各自的机器人软件与程序。
- **ROS Wiki**: ROS Wiki 是用于记录有关 ROS 系统信息的主要论坛。任何人都可以注册账户和贡献自己的文件、提供更正或更新、编写教程以及其他信息。
- **邮件列表 (Mailing list)**：ROS 用户邮件列表是关于 ROS 的主要交流渠道，能够交流从 ROS 软件更新到 ROS 软件使用中的各种疑问或信息。

## 2.2 ROS 订阅和发布消息内容

整个订阅和发布过程主要由以下两个功能节点实现，分别是发布者节点和订阅者节点。下面就如何编写两个节点以及如何编译做一个介绍。

### 2.2.1 编写发布者节点

节点(Node) 是指 ROS 网络中可执行文件。首先在自己的工作空间 catkin\_ws 里创建一个包 package，创建该包的时候会生成一系列文件夹和配置文件，其中就包含 src。然后我们开始创建一个发布器节点“**talker**”，它将不断的在 ROS 网络中广播消息，这里所创建的话题是 **chatter**。

首先回到我们创建的 package 路径下：

```
cd ~/catkin_ws/src/package
```

在 package 路径下的 src 文件夹用来放置 package 的所有源代码，这里首先创建一个发布文件 **talker.cpp**。主要实现功能和具体做法如下：

```
//ros/ros.h 是一个实用的头文件，它引用了 ROS 系统中大部分常用的头文件。
#include "ros/ros.h"
//引用了 std_msgs/String 消息，它存放在 std_msgs package 里，是由
String.msg 文件自动生成的头文件。
#include "std_msgs/String.h"
#include <iostream>

int main(int argc, char **argv){
    //初始化 ROS
```

```

ros::init(argc, argv, "talker");
//为这个进程的节点创建一个句柄。
ros::NodeHandle n;
//告诉 master 将要在 chatter (话题名) 上发布 std_msgs/String 消息类型
//的消息。这样 master 就会告诉所有订阅了 chatter 话题的节点，将要有数据发布。
//第二个参数是发布序列的大小。
ros::Publisher chatter_pub =
n.advertise<std_msgs::String>("chatter", 1000);
//指定自循环的频率。它会追踪记录自上一次调用 Rate::sleep() 后时间的流逝,
//并休眠直到一个频率周期的时间。在这个例子中, 让它以 10Hz 的频率运行。
ros::Rate loop_rate(10);
int count = 0;
while (ros::ok())
{
    //使用一个由 msg file 文件产生的消息自适应类在 ROS 网络中广播消息。
    //现在使用标准的 String 消息, 它只有一个数据成员 “data”。当然, 也可以发布更复杂
    //的消息类型。
    std_msgs::String msg;
    std::stringstream ss;
    ss << "hello world " << count;
    msg.data = ss.str();
    //ROS_INFO 和其他类似的函数可以用来代替 printf/cout 等函数。
    ROS_INFO("%s", msg.data.c_str());
    //向所有订阅 chatter 话题的节点发送消息。
    chatter_pub.publish(msg);
    ros::spinOnce();
    //这条语句是调用 ros::Rate 对象来休眠一段时间以使得发布频率为
    //10Hz。
    loop_rate.sleep();
    ++count;
}
return 0;
}

```

## 2.2.2 编写订阅器节点

在 src 中写入订阅文件 `listener.cpp`:

```

#include "ros/ros.h"
#include "std_msgs/String"
//定义一个回调函数, 当接收到 chatter 话题的时候就会被调用。消息是以 boost
//shared_ptr 指针的形式传输, 这就意味着你可以存储它而又不需要复制数据。

```

```

void chatterCallback(const std_msgs::String::ConstPtr& msg)
{
    ROS_INFO("I heard: [%s]", msg->data.c_str());
}

int main(int argc, char **argv){
    ros::init(argc, argv, "listener");
    ros::NodeHandle n;
    //告诉 master 要订阅 chatter 话题上的消息。当有消息发布到这个话题时,
    ROS 就会调用 chatterCallback() 函数。第二个参数是队列大小, 以防处理消息的速度
    不够快, 当缓存达到 1000 条消息后, 再有新的消息到来就将开始丢弃先前接收的消息。
    有各种不同的 NodeHandle::subscribe() 函数, 可以指定类的成员函数, 甚至是
    Boost.Function 对象可以调用的任何数据类型。
    ros::Subscriber sub = n.subscribe("chatter", 1000,
chatterCallback);
    //进入自循环, 可以尽可能快的调用消息回调函数。
    ros::spin();
    return 0;
}

```

### 2.2.3 编译与运行节点

在创建 package 包的时候, 会同时自动生成 `package.xml` 和 `CMakeLists.txt` 文件。下面就可以编译整个包的发布和订阅节点了。

`catkin_make`

完成编译之后, 开始运行。

首先需要确保 `rescore` 可用, 在一个窗口中运行:

`roscore`

新开一个窗口运行:

`rosrun package talker`

出现如下所示:

```
[ INFO] [1495039634.704443313]: hello world 0
[ INFO] [1495039634.805231417]: hello world 1
[ INFO] [1495039634.905569506]: hello world 2
[ INFO] [1495039635.004821578]: hello world 3
[ INFO] [1495039635.105197141]: hello world 4
[ INFO] [1495039635.205296358]: hello world 5
[ INFO] [1495039635.304792273]: hello world 6
[ INFO] [1495039635.404978344]: hello world 7
[ INFO] [1495039635.504606982]: hello world 8
[ INFO] [1495039635.605256801]: hello world 9
[ INFO] [1495039635.704843209]: hello world 10
[ INFO] [1495039635.805375149]: hello world 11
[ INFO] [1495039635.904738599]: hello world 12
[ INFO] [1495039636.005312713]: hello world 13
[ INFO] [1495039636.104750013]: hello world 14
```

图 2-6 发布程序运行结果

再开一个窗口运行：

```
rosrun package listener
```

如下所示：

```
[ INFO] [1495039700.814505519]: I heard: [hello world 28]
[ INFO] [1495039700.914675743]: I heard: [hello world 29]
[ INFO] [1495039701.014886681]: I heard: [hello world 30]
[ INFO] [1495039701.115726218]: I heard: [hello world 31]
[ INFO] [1495039701.214178230]: I heard: [hello world 32]
[ INFO] [1495039701.315041511]: I heard: [hello world 33]
[ INFO] [1495039701.414082857]: I heard: [hello world 34]
[ INFO] [1495039701.514349885]: I heard: [hello world 35]
[ INFO] [1495039701.615054193]: I heard: [hello world 36]
[ INFO] [1495039701.714480492]: I heard: [hello world 37]
[ INFO] [1495039701.814043800]: I heard: [hello world 38]
[ INFO] [1495039701.914712204]: I heard: [hello world 39]
[ INFO] [1495039702.015251383]: I heard: [hello world 40]
[ INFO] [1495039702.114955129]: I heard: [hello world 41]
```

图 2-7 订阅程序运行结果

## 2.3 利用 ROS 读取相机数据

### 2.3.1 相机介绍

无人小车 deepcar 搭载的摄像头为双目相机。双目相机和普通 usb 摄像头的区别是，普通 usb 摄像头只能输出 RGB 图，即我们平常见到的彩色图像；而双目相机除了输出彩色图像外，还会同步输出深度图，如下图所示。



图 2-8 双目相机输出的彩色图与深度图

相机的实物如下图所示：



图 2-9 相机实物图

### 2.3.2 摄像头启动

启动终端，运行如下命令，程序运行界面如下图所示：

```
cd ininCar/  
roslaunch astra_camera astrapro.launch
```

运行界面如下图所示：

```
deepcar@deepcar-desktop:~/InInCar$ deepcar@deepcar-desktop:~/InInCar$ roslaunch astra_camera astrapro.launch
WARNING: Package name "detectionInfo_msgs" does not follow the naming conventions. It should start with a lower case letter and only contain lower case letters, digits, underscores, and dashes.
... logging to /home/deepcar/.ros/log/af8ebd5a-55a8-11eb-93d2-9e5cf22496e9/roslaunch-deepcar-desktop-14496.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

WARNING: Package name "detectionInfo_msgs" does not follow the naming conventions. It should start with a lower case letter and only contain lower case letters, digits, underscores, and dashes.
started roslaunch server http://deepcar-desktop:42485/
```

图 2-10 摄像头启动示意图

命令注解：使用 rosrun 调用 astrapro.launch 文件，启动摄像头。

### 2.3.3 图像显示

显示图像的方法是通过程序接收相机发出的图像消息，并进行显示。程序名称为 `xiangji.py`，该程序的运行方法如下：在该程序所在目录打开终端，同时保证相机已经启动，运行如下命令即可启动程序：

python xiangji.py

`xiangji.py` 的程序逻辑如下：

```
#接收相机消息
result=rospy.Subscriber("/camera/rgb/image_raw", Image,callback)
#进入回调函数
callback(data):
#显示图像
cv2.imshow('frame',frame)
```

程序运行后，接收到相机发出的图像，显示的图像如下图所示：

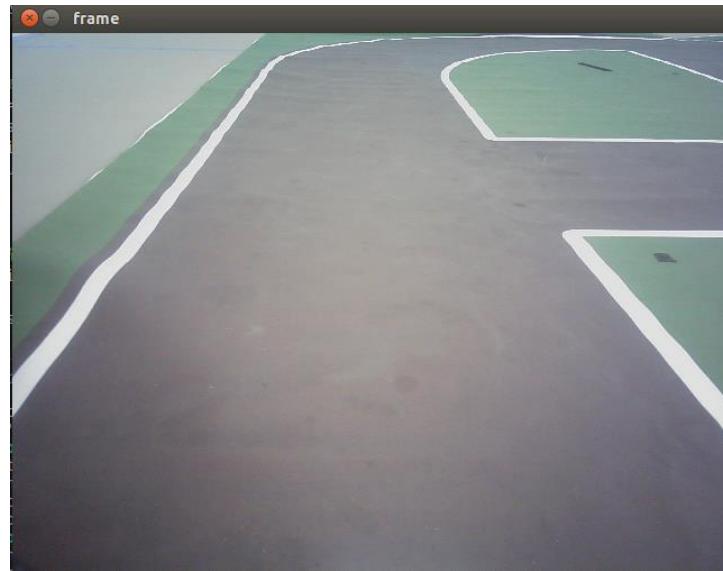


图 2-11 运行程序后显示的图像

### 3 车辆控制

本项目的无人小车平台是模拟真实驾驶场景的无人驾驶系统进行设计的，其中小车通过车载中心电脑（即 NVIDIA Jetson Nano）连接驱动板自主地控制小车的转向和速度，使得小车能够在模拟道路上安全、可靠的行驶。无人小车平台的关键技术是环境感知技术和车辆控制技术，其中环境感知技术是无人小车自主行驶的基础，车辆控制技术是无人小车行驶的核心。本项目无人小车的车辆控制技术主要包括速度控制与转向控制两部分。

### 3.1 速度及转向控制



图 3-1 无人小车电机与车轮

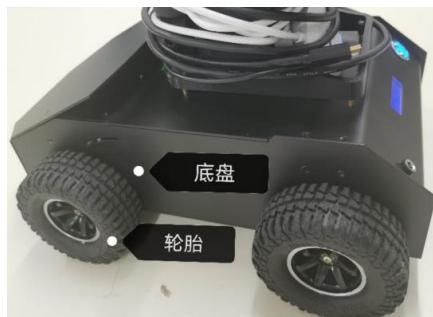


图 3-2 无人小车底盘和轮胎

如上图所示，无人小车的底盘有四路电机，四路电机都连接到驱动板。左侧的前后电机使用同一路控制信号进行驱动，右侧同理。驱动板与车载中心电脑通过串口线进行连接，车载中心电脑会根据控制信号下发左侧电机的控制值与右侧电机的控制值。在此基础上，小车在进行速度控制时，根据给定的平均速度给左右电机下发相同的速度信号，驱动板上的单片机封装好了速度闭环控制算法，以驱动小车平稳的完成速度变化。

下面将以键盘控制小车行走程序为例，展示如何通过程序设置控制小车的速度。

#### 3.1.1 键盘控制小车加减速

此程序为利用键盘操纵小车行走，程序拟定的键盘按键控制功能如下  
(注：小车速度为负数时表示小车将向后运动)：

按键	功能
W	增加小车速度
S	减少小车速度
A	增加小车逆时针旋转方向的角速度，即小车向左转弯运动
D	减少小车顺时针旋转方向的角速度，即小车向右转弯运动

C	将小车速度和转向置 0，即停车控制
Q	停车控制，并终止 python 程序

### 3.1.1.1 程序运行

打开小车车载中心电脑，进行终端界面，运行如下命令：

```
cd ininCar/
source devel/setup.bash
cd launch/
roslaunch 1_keyboard_control_car.launch
```

```
deepcar@deepcar-desktop:~$ cd ininCar/
deepcar@deepcar-desktop:~/ininCar$ source devel/setup.bash
deepcar@deepcar-desktop:~/ininCar$ cd launch/
deepcar@deepcar-desktop:~/ininCar/launch$ roslaunch 1_keyboard_control_car.launch
... logging to /home/deepcar/.ros/log/d4147ef4-202d-11eb-a631-9e5cf22496e9/roslaunch-deepcar-desktop-26239.log
'Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://192.168.43.171:40245/
```

图 3-3 键盘控制小车运行界面

命令注解：使用 `roslaunch` 调用 `1_keyboard_control_car.launch` 文件，启动小车控制程序。

注：`1_keyboard_control_car.launch` 是使用 xml 语法编写的 ros 启动文件，通过 `launch` 文件可以大批量地调用不同包中的结点，同时开启 `roscore`，提高运行结点的效率。

以下为 `1_keyboard_control_car.launch` 中的内容：

```
<!-- 程序 1. 启动键盘控制小车程序。
@init: 初始化设置小车速度为 0，转弯的角速度为 0
@param: W 加速. S 减速. A 左转弯. D 右转弯. C 停车(设置速度=0, 转向=0)
        Q 终止程序
@note: 主程序在 car_control.py 中，参数文件为
1_keyboard_control_car.yaml
-->
<launch>
    <node name="arduino" pkg="ros_arduino_python" type="car_control.py"
output="screen">
        <rosparam file="../config/1_keyboard_control_car.yaml"
command="load" />
    </node>
</launch>
```

- 使用 `<Launch>` 声明 `Launch` 内容的开始，使用 `</Launch>` 声明内容的结束。
- 使用 `<node>` 声明结点，`</node>` 表示结点声明结束，如果声明的内容只有

一行则 `<node "content" />` 即可

- **name** 表示结点运行时的名称
- **pkg** 表示结点所在的包
- **type** 表示可执行文件名称
- **output** 表示内容输出, `output="screen"` 表示输出到屏幕上。
- 使用 `rosparam` 表示结点执行时的参数
  - 在 `1_keyboard_control_car.launch` 中采用的是 `yaml` 文件导入。

### 3.1.1.2 主程序 "car\_control.py"

"`car_control.py`" 位于 `ininCar\src\ros_arduino_bridge\ros_arduino_python\nodes` 路径底下。该程序将会调用 OpenCV 库构建一个窗口:

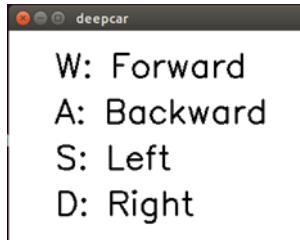


图 3-4 程序创建的控制窗口

注: 鼠标需要点击上图所示窗口, 才可有效检测键盘按键。

程序的主要结构如下:

```
class Arduino_Car(ArduinoROS):  
    """  
    @note: 继承小车底层控制类 ArduinoROS  
    """  
    def __init__(self):  
        ... # 初始化程序  
  
    def send_vel_cmd(self, vel_x, anv_z):  
        ... # 向小车底层发布速度 vel_x 和转向角速度 anv_z  
  
    def get_encoder(self):  
        ... # 读取小车编码器计数  
  
    def stop_car(self):  
        ... # 停止小车控制  
  
    def get_odom(self):  
        ... # 返回小车底层编码器计数累计的里程:  
        # odom_x(前进方向), odom_y(左右方向), odom_theta(转弯弧度)  
  
    def keyboard_control(car_x, car_th):  
        ... # 键盘按键控制函数
```

```

# 'a' : 转弯角速度加 0.01
# 'd' : 转弯角速度减 0.01
# 'w' : 小车速度加 0.01
# 's' : 小车速度减 0.01
# 'c' : 设置速度=0, 转向角速度=0
# 'q' : 终止 ros 程序
# 返回键盘按键设置的速度 car_x 与转弯角速度 car_th

if __name__ == '__main__':
    try:
        myArduino = Arduino_Car()
        r = rospy.Rate(50)
        car_x = 0 # 设置初始速度与转弯角速度为0
        car_th = 0

        while not rospy.is_shutdown():
            # 检测键盘按键，获取键盘设置的速度及转弯角速度
            car_x,car_th = keyboard_control(car_x,car_th)
            # 向小车底层发布速度及转弯角速度
            myArduino.send_vel_cmd(car_x,car_th)

```

### 3.1.2 转向控制

前面已经提及小车的速度控制是车载中心电脑通过下发左侧电机速度值和右侧电机速度值实现的。在此基础上，小车的转向控制利用左右侧电机转动差速实现。即对左右电机分别给予不同速度值，当右侧电机设定速度值大于左侧电机设定速度值时，小车向左转弯。为方便程序调试，小车的控制程序已经封装为直接设定转向角速度实现，如键盘控制小车加减速小节示例程序所示，键盘按键 **A** 和按键 **D** 分别实现加减转向角速度，实现小车向左和向右转向运动。

### 3.1.3 停车控制

在本项目中，小车的速度变化为通过设定速度值，驱动电机转动实现。由于无人小车仅是对真实的车辆的模拟，其行驶速度在 1m/s 以下，与真实的车辆运动存在极大差距。在此速度下，小车的停车距离可忽略不计，当设定速度值为零时，无人小车可立即制动。

## 3.2 巡线控制

### 3.2.1 PID 算法

PID 控制器在连续控制系统中是技术成熟、应用最为广泛的一种控制器，它的结构简单，参数易于调整。

PID 控制器是按偏差的比例、积分、微分进行控制的控制器，PID 的控制量输出如下式所示，式中的第一项为比例项，输出与误差呈比例关系，式中第二项为积分项，输出与误差的积分呈正比，式中的第三项为微分项，输出与误差的微分呈正比。

$$u(t) = K_p(e(t) + \frac{1}{T_i} \int_0^t e(\tau) d\tau + T_d \frac{de(t)}{dt})$$

连续 PID 控制的系统结构图如下图所示。图中的比例、积分、微分分别对应了式中的比例项、积分项和微分项。

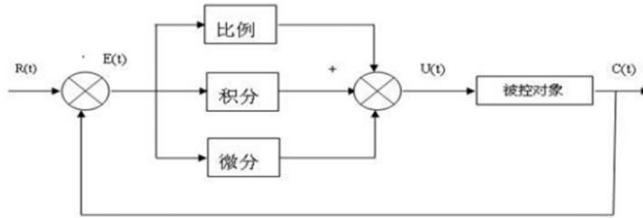


图 3-5 PID 控制流程

但由于小车属于离散系统，所以需要对 PID 进行离散化处理，数字 PID 控制器输出如下式所示。

$$u(k) = K_p \{e(k) + \frac{T_s}{T_i} \sum_{j=0}^k e(j) + \frac{Td}{T_s} [e(k) - e(k-1)]\}$$

在进行参数整定的时候需要了解式中的每一个参数的作用。

比例系数  $K_p$  用于调节比例项的作用效果，比例控制能够迅速反应误差，提高系统的响应速度，但是会引起系统稳定性的降低，超调增加，此外，比例控制还能减小稳态误差，但是并不能完全消除稳态误差。

积分时间  $T_i$  用于调节积分项的作用效果，积分项的输出与误差的积分呈正比，只要系统有误差存在，积分项就会一直累计误差，所以积分项可以起到消除静差的作用，但是积分会使系统相位滞后，响应变慢，增加系统的超调，甚至会使系统振荡。

微分时间  $T_d$  用于调节微分项的作用效果，微分项的输出与误差的微分呈正比，误差的微分反映了误差变化的快慢，所以微分控制是超前控制，可以使系统的相位超前，响应变快，使系统的稳定性提高，动态响应速度变快，减小超调量，改善系统的动态性能，但由于微分项对干扰较为敏感，所以  $T_d$  也不易过大。

### 3.2.2 巡线控制程序

巡线控制程序位于 `ininCar\src\decision\ros_decision_python\nodes` 路径下的 "`decision_control.py`" 中，算法原理为：

小车的线速度为用户设定值 `self.forwardSpeed`（默认为 0.2）：

```
class Arduino_Car(ArduinoROS):
    # 继承类
    def __init__(self):
        ...
        self.forwardSpeed = 0.2    # 小车前进速度， 默认为 0.2
        ...
```

小车的角速度为首先接收车道线检测程序发布的小车位置偏差

```
self.lineErr:

class Arduino_Car(ArduinoROS):
    # 继承类
    def __init__(self):
        ...
        # 巡线订阅器
        self.lane_error_result = rospy.Subscriber('/lane_error',
Lanesite, self.laneErrorCallback)
        ...
        # 车道线巡线回调函数
    def laneErrorCallback(self, data):
        self.lineErr = data.err
        ...
```

然后将小车的位置偏差作为 PID 控制器的输入，PID 控制器的输出即为小车的角速度，PID 控制算法程序如下：

```
def motion_control(self, car_x, car_th):
    ...
    # 小车的线速度设置为预定的前进速度
    car_x = self.forwardSpeed
    # 小车的角速度根据 PID 算法进行设置
    car_th = self.pid_p * self.lineErr + self.pid_d * (self.lineErr -
self.lastErr)
    # 记录上一次小车的偏差，给 PID 控制用
    self.lastErr = self.lineErr
```

...

算法流程图如下图所示。

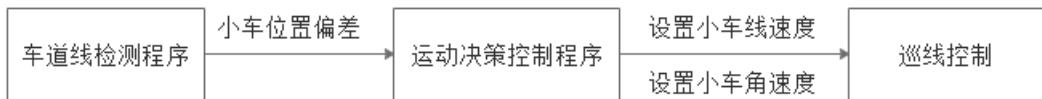


图 3-6 巡线控制流程

启动终端，运行如下命令，即可实现巡线控制。程序运行界面如下所示：

```
cd ininCar/  
roslaunch launch/8_decision_control.launch
```

注：上述程序需在相机开启后才能正常实现巡线控制。

以下为 8\_decision\_control.launch 中的内容：

```
<!--启动控制小车程序-->  
<launch>  
  <node name="arduino" pkg="ros_arduino_python" type="xun_xian.py"  
output="screen">  
  </node>  
  <node name="arduino" pkg="ros_decision_python"  
type="decision_control.py" output="screen">  
    <rosparam file="../config/2_decision_control_car.yaml"  
command="load" />  
  </node>  
</launch>
```

其中，该 launch 文件同时运行了两个结点程序，“`xun_xian.py`”为车道线检测程序，“`decision_control.py`”为小车运动决策控制程序。

## 4 环境感知

### 4.1 车道线检测

模拟城市的交通场景如下图所示。



图 4-1 模拟城市的交通场景图

我们需要从摄像头采集到的城市交通图中提取出白色的车道线。

#### 4.1.1 启动摄像头

车道线的图像是通过摄像头采集的，因此需要打开摄像头，这在上文中已经说明，按照操作即可。注意保持摄像头开启状态。

#### 4.1.2 车道线检测

车道线检测程序运行方法如下：

新打开一个终端，输入如下指令：（注意，该程序和运动控制程序关联，运行该程序后，小车可能会低速运动）

```
cd ininCar/  
source devel/setup.bash  
cd launch/  
roslaunch 8_decision_control.launch
```

终端运行效果如下图所示：

```

NODES
/
    arduino (ros_decision_python/xun_xian.py)
    controller (ros_decision_python/decision_control.py)

ROS_MASTER_URI=http://localhost:11311

process[arduino-1]: started with pid [13145]
process[controller-2]: started with pid [13146]
Connecting to Arduino on port /dev/deepcar ...
Connected at 115200
Arduino is ready.
[INFO] [1609926757.444878]: Connected to Arduino on port /dev/deepcar at 115200
baud
[0] use BaseController
Updating PID parameters
[INFO] [1609926757.536610]: Started base controller for a base of 0.37m wide with 7420 ticks per rev
[INFO] [1609926757.540544]: Publishing odometry data at: 30.0 Hz using base_link as base frame
[0] init arduino_node complete.
[1] start init arduino_car .
[2] start init control_param .

```

图 4-2 车道线程序启动后示意图

显示出的车道线检测效果如下图所示。



图 4-3 车道线检测效果图

程序逻辑如下：

```

#读取车道线图像
self.result=rospy.Subscriber("/camera/rgb/image_raw",
Image,self.callback, queue_size=1, buff_size=2**24)
#进入回调函数，图像预处理
callback(self,data):
    ...
#车道线检测与拟合，显示车道线拟合后的图像
cv2.imshow('src',frame)

```

### 4.1.3 转向量计算

为了让小车可以正常巡线，需要根据检测到的车道线计算转向量。小车之所以能够巡线行驶，是因为小车可以调整自身位置，尽量行驶在道路中间。要做到这一点，需要实时计算出小车自身与道路中间位置的偏差量，从而可以将该偏差量作为 PID 控制算法的控制量，控制小车的转向。这里将小车自身与道路中间位置的偏差量称作转向量。

在上一小节所运行的程序中计算了转向量，转向量在终端显示的结果如下图所示。

```
8_decision_control.launch http://localhost:11311
236.261958727
284.281724546 info: [actErr]
234.917568316
252.240744371
247.994367505 INFO      Property Depth.FirmwareMirror was changed
272.007078564 INFO      Property Depth.Mirror was changed to 1.
239.353255412 INFO      Stream 'Depth' was initialized.
237.744018578 INFO      'Depth' stream was created.
227.983125885 VERBOSE   Configuring module 'Depth' from section ''
230.851557667 or/catkin_ws/src/ros_astra_camera-master/include/open
250.888365692 vers/orbbec.ini'...
258.021458419 INFO      Setting Depth.InputFormat to 3...
251.407172693 INFO      Property Depth.InputFormat was changed to
252.456726611 INFO      Depth.InputFormat was successfully set.
250.321005504 INFO      Setting Depth.Resolution to VGA...
240.846075945 INFO      Property Depth.Resolution was changed to
284.470754531 INFO      Property Depth.XRes was changed to 640.
265.680736601 INFO      Property Depth.RequiredDataSize was changed
241.007201445 INFO      Property Depth.YRes was changed to 480.
235.182141709 INFO      Property Depth.RequiredDataSize was changed
250.683767405 INFO      Depth.Resolution was successfully set.
231.37052916 INFO      Module 'Depth' configuration was loaded
250.712134987 INFO      Setting Depth.Registration to 1...
```

图 4-4 转向量显示图

## 4.2 深度学习基础

最近很火的三个名词，人工智能、机器学习、深度学习，那么这三者的关系是什么样的呢，我们可以用下面一张图片来表达。

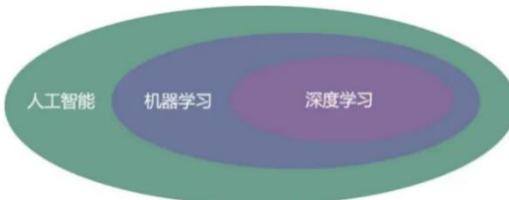


图 4-5 深度学习与人工智能

通俗来说，机器学习是一门讨论各式各样的适用于不同问题的函数形式，

以及如何使用数据来有效地获取函数参数具体值的学科。深度学习是指机器学习中的一类函数，它们的形式通常为多层次神经网络。近年来，仰仗着大数据集和强大的硬件，深度学习已逐渐成为处理图像、文本语料和声音信号等复杂高维度数据的主要方法。

深度学习涵盖了许多知识，如卷积神经网络，循环神经网络，本讲义不可能一一讲述，在此主要为读者讲述一些深度学习中的基础知识，为读者后继对深度学习的研究做铺垫。

### 4.2.1 深度学习的起源

1943 年，心理学家麦卡洛克和数学逻辑学家皮兹发表论文《神经活动中内在思想的逻辑演算》，提出了 MP 模型。MP 模型是模仿神经元的结构和工作原理，构成出的一个基于神经网络的数学模型，本质上是一种“模拟人类大脑”的神经元模型。MP 模型作为人工神经网络的起源，开创了人工神经网络的新时代，也奠定了神经网络模型的基础。

1949 年，加拿大著名心理学家唐纳德·赫布在《行为的组织》中提出了一种基于无监督学习的规则——海布学习规则(Hebb Rule)。海布规则模仿人类认知世界的过程建立一种“网络模型”，该网络模型针对训练集进行大量的训练并提取训练集的统计特征，然后按照样本的相似程度进行分类，把相互之间联系密切的样本分为一类，这样就把样本分成了若干类。海布学习规则与“条件反射”机理一致，为以后的神经网络学习算法奠定了基础，具有重大的历史意义。

20 世纪 50 年代末，在 MP 模型和海布学习规则的研究基础上，美国科学家罗森布拉特发现了一种类似于人类学习过程的学习算法——感知机学习。并于 1958 年，正式提出了由两层神经元组成的神经网络，称之为“感知器”。感知器本质上是一种线性模型，可以对输入的训练集数据进行二分类，且能够在训练集中自动更新权值。感知器的提出吸引了大量科学家对人工神经网络研究的兴趣，对神经网络的发展具有里程碑式的意义。

但随着研究的深入，在 1969 年，“AI 之父”马文·明斯基和 LOGO 语言的创始人西蒙·派珀特共同编写了一本书籍《感知器》，在书中他们证明了单层感知器无法解决线性不可分问题（例如：异或问题）。由于这个致命的缺陷

以及没有及时推广感知器到多层神经网络中，在 20 世纪 70 年代，人工神经网络进入了第一个寒冬期，人们对神经网络的研究也停滞了将近 20 年。

### 4.2.2 深度学习的发展

1982 年，著名物理学家约翰 · 霍普菲尔德发明了 Hopfield 神经网络。Hopfield 神经网络是一种结合存储系统和二元系统的循环神经网络。Hopfield 网络也可以模拟人类的记忆，根据激活函数的选取不同，有连续型和离散型两种类型，分别用于优化计算和联想记忆。但由于容易陷入局部最小值的缺陷，该算法并未在当时引起很大的轰动。

直到 1986 年，深度学习之父杰弗里 · 辛顿提出了一种适用于多层感知器的反向传播算法——BP 算法。BP 算法在传统神经网络正向传播的基础上，增加了误差的反向传播过程。反向传播过程不断地调整神经元之间的权值和阈值，直到输出的误差达到减小到允许的范围之内，或达到预先设定的训练次数为止。BP 算法完美的解决了非线性分类问题，让人工神经网络再次的引起了人们的广泛的关注。

### 4.2.3 深度学习的爆发

2006 年，杰弗里 · 辛顿以及他的学生鲁斯兰 · 萨拉赫丁诺夫正式提出了深度学习的概念。他们在世界顶级学术期刊《科学》发表的一篇文章中详细的给出了“梯度消失”问题的解决方案——通过无监督的学习方法逐层训练算法，再使用有监督的反向传播算法进行调优。该深度学习方法的提出，立即在学术圈引起了巨大的反响，以斯坦福大学、多伦多大学为代表的众多世界知名高校纷纷投入巨大的人力、财力进行深度学习领域的相关研究。而后又在迅速蔓延到工业界中。

2012 年，在著名的 ImageNet 图像识别大赛中，杰弗里 · 辛顿领导的小组采用深度学习模型 AlexNet 一举夺冠。AlexNet 采用 ReLU 激活函数，从根本上解决了梯度消失问题，并采用 GPU 极大的提高了模型的运算速度。同年，由斯坦福大学著名的吴恩达教授和世界顶尖计算机专家 Jeff Dean 共同主导的深度神经网络——DNN 技术在图像识别领域取得了惊人的成绩，在 ImageNet 评测中成功的把错误率从 26% 降低到了 15%。深度学习算法在世界大赛的脱颖而出，也

再一次吸引了学术界和工业界对于深度学习领域的关注。

随着深度学习技术的不断进步以及数据处理能力的不断提升，2014年，Facebook 基于深度学习技术的 DeepFace 项目，在人脸识别方面的准确率已经能达到 97% 以上，跟人类识别的准确率几乎没有差别。这样的结果也再一次证明了深度学习算法在图像识别方面的一骑绝尘。

2016 年，随着谷歌公司基于深度学习开发的 AlphaGo 以 4:1 的比分战胜了国际顶尖围棋高手李世石，深度学习的热度一时无两。后来，AlphaGo 又接连和众多世界级围棋高手过招，均取得了完胜。这也证明了在围棋界，基于深度学习技术的机器人已经超越了人类。

2017 年，基于强化学习算法的 AlphaGo 升级版 AlphaGo Zero 横空出世。其采用“从零开始”、“无师自通”的学习模式，以 100:0 的比分轻而易举打败了之前的 AlphaGo。除了围棋，它还精通国际象棋等其它棋类游戏，可以说是真正的棋类“天才”。此外在这一年，深度学习的相关算法在医疗、金融、艺术、无人驾驶等多个领域均取得了显著的成果。所以，也有专家把 2017 年看作是深度学习甚至是人工智能发展最为突飞猛进的一年。

#### 4.2.4 思考题

(1) 你现在正在编写的代码有没有可以被“学习”的部分，也就是说，是否有可以被机器学习改进的部分？

(2) 你在生活中有没有这样的场景：虽有许多展示如何解决问题的样例，但缺少自动解决问题的算法？它们也许是深度学习的最好猎物。

(3) 如果把人工智能的发展看作是新一次工业革命，那么深度学习和数据的关系是否像是蒸汽机与煤炭的关系呢？为什么？

(4) 端到端的训练方法还可以用在哪里？物理学，工程学还是经济学？

(5) 为什么应该让深度网络模仿人脑结构？为什么不该让深度网络模仿人脑结构？

#### 4.2.5 目标检测

计算机视觉领域有四大主要任务，分别是图像分类、目标检测、目标跟踪、图像分割。图像分类的目标是将给定的图像进行分类，给图片或视频分配

一个类别标签（比如图像中大部分都是气球，还有其他物体，要给这个图片或者视频提供气球的标签）。目标检测的有两个主要任务，分别是物体分类和定位，具体来讲就是在图像中用边界框(bounding box)标定多个目标的位置和类别（具体是用边界框标出图像中气球的位置，并打上气球的标签）。目标跟踪任务就是在给定某视频序列初始帧的目标大小与位置的情况下，预测后续帧中目标的大小与位置。图像分割主要有语义分割和实例分割。语义分割是目标检测更进阶的任务，目标检测只需要框出每个目标的边界框，语义分割需要进一步判断图像中哪些像素属于哪个目标。实例分割不但要进行像素级别的分类，还需在具体的类别基础上区别开不同的实例。本讲义主要介绍的就是计算机视觉中的目标检测任务。

除了图像分类之外，目标检测要解决的核心问题是：

1. 目标可能出现在图像的任何位置。
2. 目标有各种不同的大小。
3. 目标可能有各种不同的形状。

## 4.3 红绿灯识别

红绿灯识别模块主要在于判断摄像头采集到的图像中是否具有红绿灯标志，并且对其种类进行判断，并且在图像上对红绿灯的位置、大小进行标注，并且标明种类和置信度，并且以消息的形式进行输出。

### 4.3.1 程序运行

红绿灯识别启动程序内容如下：

注意：由于车载电脑算力较低，程序需要花费几分钟来启动 GPU 相关运算模块。

```
cd ininCar/  
source devel/setup.bash  
roslaunch object_detection object_detection_astra.launch  
... (启动此命令后需要等待几分钟才可看到目标检测结果)
```

这样就可以开启目标检测代码，但是目标检测还需要使用摄像头，所以如果没有开启摄像头需要按照如下代码开启摄像头。

```
cd ininCar/
source devel/setup.bash
roslaunch ros_astra_camera-master astrapro.py
```

红绿灯识别程序运行界面如下图所示：

```
deepcar@deepcar-desktop:~$ cd catkin_ws
deepcar@deepcar-desktop:~/catkin_ws$ source devel/setup.bash
deepcar@deepcar-desktop:~/catkin_ws$ roslaunch object_detection object_detection_astra.launch
WARNING: Package name "detectionInfo_msgs" does not follow the naming conventions. It should start with a
lower case letter and only contain lower case letters, digits, underscores, and dashes.
... logging to /home/deepcar/.ros/log/9342ce12-500c-11eb-b12c-9e5cf22496e9/roslaunch-deepcar-desktop-8301
.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://deepcar-desktop:39831/
SUMMARY
=====
PARAMETERS
  * /rosdistro: melodic
  * /rosversion: 1.14.3
NODES
  /object_detection_astra_node (object_detection/object_detection_astra_node)
auto-starting new master
process[master]: started with pid [8353]
ROS_MASTER_URI=http://localhost:11311

setting /run_id to 9342ce12-500c-11eb-b12c-9e5cf22496e9
WARNING: Package name "detectionInfo_msgs" does not follow the naming conventions. It should start with a

SUMMARY
=====
PARAMETERS
  * /rosdistro: melodic
  * /rosversion: 1.14.3
NODES
  /object_detection_astra_node (object_detection/object_detection_astra_node)
auto-starting new master
process[master]: started with pid [8353]
ROS_MASTER_URI=http://localhost:11311

setting /run_id to 9342ce12-500c-11eb-b12c-9e5cf22496e9
WARNING: Package name "detectionInfo_msgs" does not follow the naming conventions. It should start with a
lower case letter and only contain lower case letters, digits, underscores, and dashes.
process[rosout-1]: started with pid [8365]
started core service [/rosout]
process[object_detection_astra_node-2]: started with pid [8368]
[I] Begin parsing model...
[I] FP32 Mode running...
[I] End parsing model...
[I] Begin building engine...
[I] End building engine...
[I] *** deserializing
[I] *** end deserializing
```

图 4-6 红绿灯识别程序运行界面示意图

相机程序运行界面如下图所示：

```

deepcar@deepcar-desktop:~/catkin_ws$ source devel/setup.bash
deepcar@deepcar-desktop:~/catkin_ws$ roslaunch astra_camera astrapro.launch
WARNING: Package name "detectionInfo_msgs" does not follow the naming conventions. It should start with a
lower case letter and only contain lower case letters, digits, underscores, and dashes.
... logging to /home/deepcar/.ros/log/9342ce12-500c-11eb-b12c-9e5cf22496e9/roslaunch-deepcar-desktop-9960
.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

WARNING: Package name "detectionInfo_msgs" does not follow the naming conventions. It should start with a
lower case letter and only contain lower case letters, digits, underscores, and dashes.
started roslaunch server http://deepcar-desktop:35253/
SUMMARY ros System Volume Information 2 items 8 7月 2019
===== WindowsServices 0 items 19 12月 2019
PARAMETERS
* /camera/camera_nodelet_manager/num_worker_threads: 4
* /camera/camera_rgb/camera_info_url:
* /camera/camera_rgb/frame_rate: 30
* /camera/camera_rgb/height: 480
* /camera/camera_rgb/index: 0
* /camera/camera_rgb/product: 0x0502
* /camera/camera_rgb/serial: 0
* /camera/camera_rgb/timestamp_method: start
* /camera/camera_rgb/vendor: 0x2bc5
* /camera/camera_rgb/video_mode: yuyv
* /camera/camera_rgb/width: 640
* /camera/depth_rectify_depth/interpolation: 0
"模糊自适应" selected 3/24 containing 20 radios
5268703 VERBOSE cmostype: 1 m_VerticalFOV: 0.796616, m_HorizontalFOV: 1.022600
5268755 INFO Property Depth.Gain was changed to 42.
5268791 VERBOSE Getting algorithm params 0x2 for resolution 4 and fps 30....
5269346 VERBOSE Getting algorithm params 0x2 for resolution 0 and fps 30....
5270341 VERBOSE Getting algorithm params 0x2 for resolution 1 and fps 30....
5270804 VERBOSE Getting algorithm params 0x3 for resolution 4 and fps 30....
5271470 VERBOSE Getting algorithm params 0x3 for resolution 0 and fps 30....
5271946 VERBOSE Getting algorithm params 0x3 for resolution 1 and fps 30....
5343304 INFO Property Depth.FirmwareMirror was changed to 1.
5343347 INFO Property Depth.Mirror was changed to 1.
5343372 INFO Stream 'Depth' was initialized.
5343387 INFO 'Depth' stream was created.
5343404 VERBOSE Configuring module 'Depth' from section 'Depth' in file '/home/deepcar/catkin_ws/src
/ros_astra_camera-master/include/openni2_redist/arm64/OpenNI2/Drivers/orbbec.ini'...
5343640 INFO Setting Depth.InputFormat to 3...
5343669 INFO Property Depth.InputFormat was changed to 3.
5343743 INFO Depth.InputFormat was successfully set.
5345779 INFO Setting Depth.Resolution to VGA...
5345813 INFO Property Depth.Resolution was changed to VGA.
5345834 INFO Property Depth.XRes was changed to 640.
5345849 INFO Property Depth.RequiredDataSize was changed to 153600.
5345864 INFO Property Depth.YRes was changed to 480.
5345878 INFO Property Depth.RequiredDataSize was changed to 614400.
5345894 INFO Depth.Resolution was successfully set.
5346934 INFO Module 'Depth' configuration was loaded from file.
5347601 INFO Setting Depth.Registration to 1...
5347636 INFO Property Depth.FirmwareRegistration was changed to 1.
5347656 INFO Property Depth.Registration was changed to 1.
5347700 INFO Depth.Registration was successfully set.
"模糊自适应" selected 3/24 containing 20 radios

```

图 4-7 相机程序运行界面示意图

### 4.3.2 红绿灯识别输出

如下面各图所示，系统能够训练并识别出红绿灯，识别结果如下图所示：



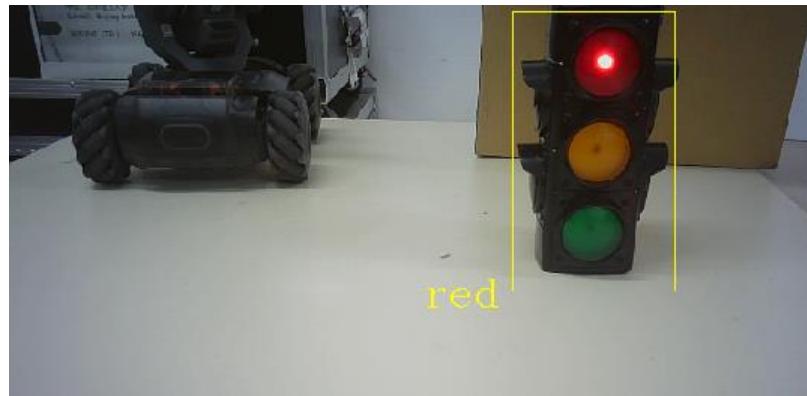


图 4-8 红绿灯识别结果图

### 4.3.3 红绿灯识别程序逻辑

```
【1】 初始化一个 Darknet 模型  
model = Darknet(opt.model_def, img_size=opt.img_size).to(device)
```

```
【2】 加载图像数据  
dataloader = DataLoader()
```

```
【3】 输入图像进行目标检测  
detections = model(input_imgs)
```

```
【4】 图像标注  
detections = rescale_boxes(detections, opt.img_size,  
img.shape[:2])
```

```
【5】 结果输出  
result_pub =  
nh_.advertise<etectionInfo_msg::detectionInfo>("/object_detection",  
1)  
image_pub = it_.advertise("/usb_cam/image_raw", 1);
```

## 4.4 交通标志检测

交通标志检测模块主要在于判断摄像头采集到的图像中是否具有交通标志，并且对其种类进行判断，并且在图像上对交通标示的位置、大小进行标注，并且标明种类和置信度，并且以消息的形式进行输出。

交通标志检测的启动方式和红绿灯标志检测相似，在此不过多赘述。

交通标志检测的结果如下图所示。



图 4-9 交通标志显示结果图

## 5 路径规划

### 5.1 原理介绍

#### 5.1.1 定义

路径规划是运动规划的主要研究内容之一。运动规划由路径规划和轨迹规划组成，连接起点位置和终点位置的序列点或曲线称之为路径，构成路径的策略称之为路径规划。

在一些科学文献中，路径规划曾被这样的定义：路径规划是自治式移动机器人的重要组成部分，它的任务就是在具有障碍物的环境内按照一定的评价标准，寻找一条从起始状态（包括位置和姿态）到达目标状态（包括位置和姿态）的无碰路径。障碍物在环境中的不同分布情况当然直接影响到规划的路径，而目标位置的确定则是由更高一级的任务分解模块提供的。

## 5.1.2 分类

根据对环境信息的把握程度可把路径规划划分为基于先验完全信息的全局路径规划和基于传感器信息的局部路径规划。其中，从获取障碍物信息是静态或是动态的角度看，全局路径规划属于静态规划(又称离线规划)，局部路径规划属于动态规划(又称在线规划)。

全局路径规划需要掌握所有的环境信息，根据环境地图的所有信息进行路径规划；局部路径规划只需要由传感器实时采集环境信息，了解环境地图信息，然后确定出所在地图的位置及其局部的障碍物分布情况，从而可以选出从当前结点到某一子目标结点的最优路径。



图 5-1 全局路径规划

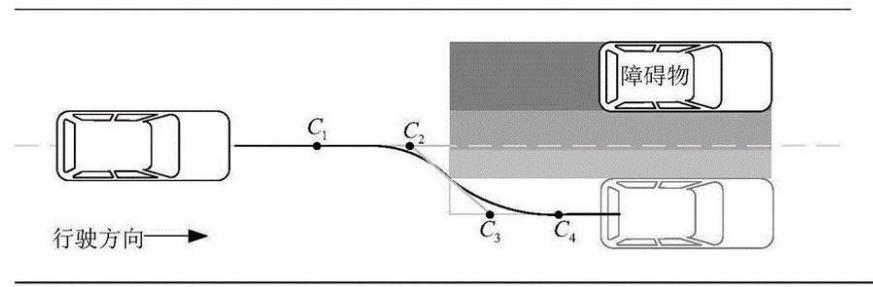


图 5-2 局部路径规划

通俗地来讲，全局路径规划是指规划一条从起点至终点的运动路径，局部路径规划是寻找小范围内的行走路径。全局路径规划和局部路径规划并没有本质上的区别，很多适用于全局路径规划的方法经过改进也可以用于局部路径规划，而适用于局部路径规划的方法同样经过改进后也可适用于全局路径规划。两者协同工作，机器人可更好的规划从起始点到终点的行走路径。

根据所研究环境的信息特点,路径规划还可分为离散域范围内的路径规划问题和连续域范围内的路径规划问题。

### 5.1.3 步骤

一般的连续域范围内路径规划问题,如机器人、飞行器等的动态路径规划问题,其一般步骤主要包括环境建模、路径搜索、路径平滑三个环节。

- ❖ 环境建模。环境建模是路径规划的重要环节,目的是建立一个便于计算机进行路径规划所使用的环境模型,即将实际的物理空间抽象成算法能够处理的抽象空间,实现相互间的映射。
- ❖ 路径搜索。路径搜索阶段是在环境模型的基础上应用相应算法寻找一条行走路径,使预定的性能函数获得最优值。
- ❖ 路径平滑。通过相应算法搜索出的路径并不一定是一条运动体可行走的可行路径,需要作进一步处理与平滑才能使其成为一条实际可行的路径。

对于离散域范围内的路径规划问题,或者在环境建模或路径搜索前已经做好路径可行性分析的问题,路径平滑环节可以省去。

为了更好地理解路径规划的步骤,还可以参考下图。

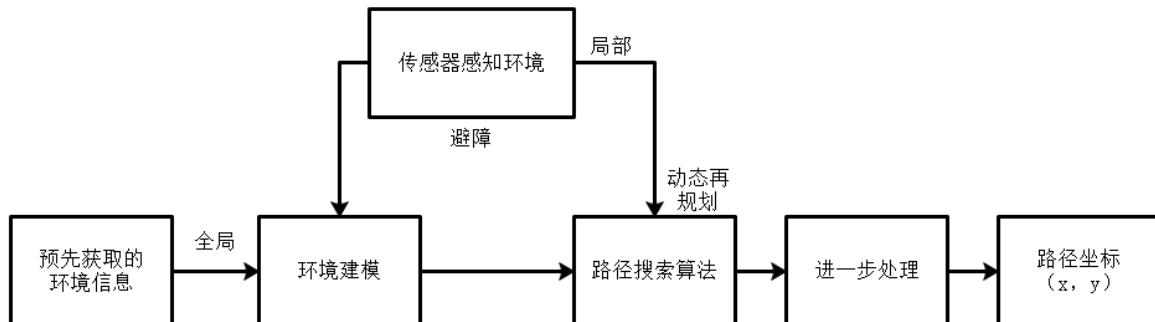


图 5-3 路径规划流程图

### 5.1.4 方法

#### 5.1.4.1 路径规划算法

机器人的路径规划方法大致可以分为两类:传统方法和智能方法。除此之外还可以进行算法的改进或对几种算法进行组合。将主要用到的一些算法罗列在下图中。

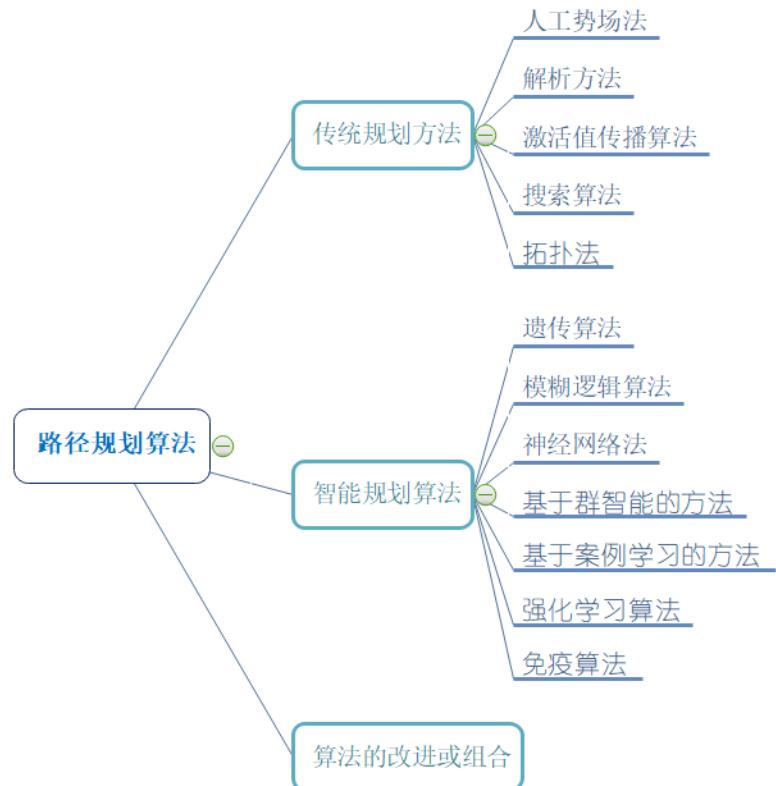


图 5-4 主要的路径规划算法

#### ➤ 人工势场法

最初是 1986 年出 Khatib 提出，其基本思想是将移动机器人在环境中的运动视为一种虚拟人工受力场中的运动。该法结构简单，便于低层的实时控制，在实时避障和平滑的轨迹控制方面，得到了广泛应用，其不足在于存在局部最优解，容易产生死锁现象，因而可能使移动机器人在到达目标点之前就停留在局部最优点。

#### ➤ 解析法

用一个依赖于位形空间参数的不等式组来表示机器人躲避障碍物的要求。路径规划由起始位形到终止位形寻找路径时最小化一个标量函数而转化为一个数学优化问题。出于这种优化是非线性的，并有很多限制条件，故往往需要使用离散化的方法来找这个最优解。

#### ➤ 激活值传播算法

激活值传播算法（Activity Propagation）是一种基于栅格类环境表示的并行路径规划方法。一个二维平面，划分成  $N \times M$  个大小相等的正方形小方格。这些小方格称为栅格节点，每个栅格节点都对应于一个变量  $a$ ，称为该栅格节点的激

活值（Activity）。激活值传播算法是利用此激活值进行搜索的一种方法。

#### ➤ 搜索算法

又可分为基于微积分的搜索技术、有指导的随机搜索技术和美剧技术。基于微积分的搜索技术是使用由优化问题的解来满足的一组充分必要条件。包括像 Newton、Fibonacci 和“爬山”法等，这些技术一般只能用于解较简单的问题。有指导的随机搜索技术以枚举技术为基础，附加些指导搜索过程的信息。其两个主要的子集是模拟退火算法（Simulated Annealing）和进化算法（Evlutinary Algorithm）。遗传算法（Genetic Algorithm）是其特例之一，90 年代以后，使用有指导的随机搜索技术，尤其是遗传算法进行路径规划的文献呈逐步增多的趋势。枚举技术是搜索目标函数的域空间中的每一个节点。实现简单，但需要大量的计算。常用的深度优先搜索、广度优先搜索、A\*搜索和 Dijkstra 搜索都是其特例。

#### ➤ 遗传算法

遗传算法（GeneticAlgorithmn，简称 GA）最先是由 JohnHofland 于 1975 年提出的。从那以后，它逐渐展成为一种通过模拟自然进化过程解决最优化问题的计算模型。目前，基于遗传算法的全局路径规划主要是针对二维平面规划问题。

#### ➤ 模糊逻辑算法

模糊逻辑是研究模糊命题的逻辑，它是二值逻辑的推广，是对经典的二值逻辑的模糊化。模糊逻辑方法最大的特点是参考经验，计算量不大，易做到边运动边规划，能够满足实时性要求。同时存在复杂环境中规则库构造难度大，调整和修改已构成的规则库不容易和适应能力差等缺点。

#### ➤ 神经网络算法

人工神经网络开始于 1952 年 Hodgkin 和 A.F.Huxley 的研究，主要是模拟人脑结构的模型，将障碍物均假设为多面体，对一系列的路径点进行规划，使得整个路径的长度尽量短，同时又尽可能远离障碍物。主要优点是其隐含的并行性，具有并行处理、容错性和知识分布存储，但是较难找到三维物体的不等式表达式。

#### ➤ 基于群智能的方法

近几年模仿自然界中生物的一些群体行为，诞生了群智能算法，包括粒子群算法，蚁群算法等。粒子群算法（Particle Swann OPtimization，简称 PSO）是由 Eberhart 博士和 Kennedy 博士于 1995 年提出的一种模拟鸟群飞行的仿生算法，

有着个体数目少、计算简单、鲁棒性好等优点。蚁群算法是上世纪 90 年代由意大利学者 M.Dorigo, V.Mnaiezoz 等人提出的，基于生物界群体启发行为的一种随机搜索寻优方法，隐含的并行性更使其具有极强的发展潜力，在解决优化组合的问题上有良好的适应性。

#### ➤ 基于案例学习的方法

基于案例的学习方法（case-based learning）具有类比学习的功能，它依靠过去的经验进行学习及问题求解。优点是适合较难发现规律性的知识，因果关系难于用确切的模型或者规则来表达，而且实际的案例通常比规则提供的信息更多的领域；难点是案例库的建立以及组织、管理案例库，检索到匹配的案例，根据匹配结果的好坏去扩充案例库等，而且可能产生曲折的路径。

#### ➤ 强化学习算法

Q-学习算法是由瓦特金斯在 1989 年提出的类似于动态规划算法的一种方法，它提供 Agent 在马尔可夫环境中，利用经历的动作序列执行最优动作的一种学习能力。Q-学习算法实际是马尔可夫决策过程的一种变化形式。进一步又研究了几种改进的 Q-学习算法，如  $Q(\lambda)$  算法、SARSA (0) 算法及其收敛性、SARSA ( $\lambda$ ) 算法等。

#### ➤ 免疫算法

免疫算法是一种确定性和随机性选择相结合并具有勘测与开采能力的启发式随机搜索算法，它被认为是对自适应免疫应答中体液免疫的简单模拟。

### 5.1.4.2 环境表示算法

环境表示问题：指环境中障碍物的表示和自由空间的表示。环境建模就是对机器人活动空间的有效描述。机器人在规划前首先要做的就是将环境的描述由外部的原始形式通过一系列处理转化为适合规划的内部的世界模型，这个过程称为环境建模，其中主要是障碍物的表示方法。合理的环境表示才能有利于规划中搜索量的减少，才能有利于时空开销的减少。不同的规划方法是基于各种不同的环境建模来进行的。

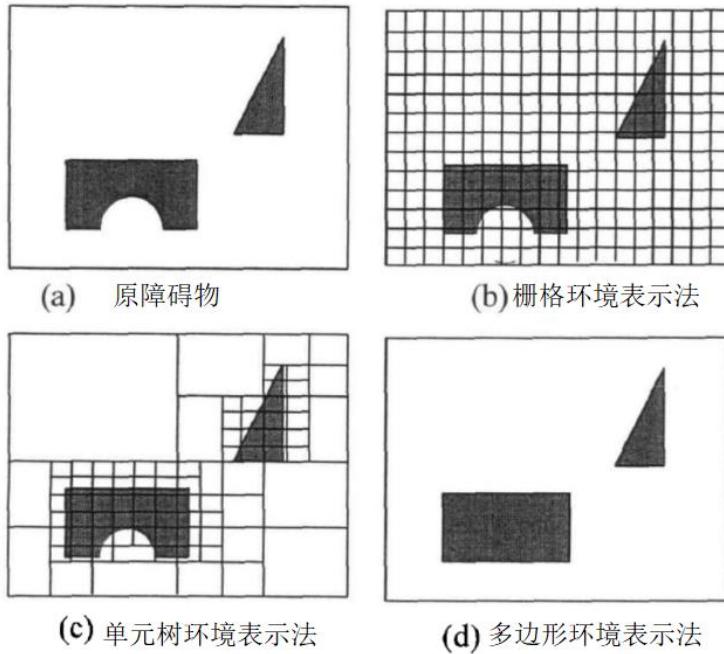


图 5-5 几种常用的环境表示方法

常见的环境建模方法有栅格环境表示法、单元树环境表示法、多边形环境表示法、Roadmap 环境表示法、C 空间环境标识法、切线图环境表示法、拓扑环境表示法、八叉树环境表示法等等。这里不再一一展开。

## 5.2 全局路径规划

### 5.2.1 定位

#### 5.2.1.1 介绍

定位是路径规划的基础，定位可以帮智能车明确自己此时的空间位置，为后续的运动控制提供依据。定位又可以分为相对定位和绝对定位。相对定位是指使用小车自身的传感器来感应周围环境，完成相对于自身的定位，如使用自身的里程计定位或使用自带的摄像头进行定位。绝对定位是指使用一些外部的传感器检测小车的位置和姿态，然后将信息发送给小车，如 GPS 定位或使用外部摄像头进行定位。由于小车地图较小，所以考虑使用自身里程计定位或外部摄像头定位，又考虑到差速驱动模型的误差过大，最终采用外部摄像头来定位。

这里的外部摄像头选用了小 R 科技的 RobotEyes USB 标清摄像头以及双天线 WiFi 模块用来图传。使用起来也很简单，将 USB 摄像头接在图传模块上，并将图传模块用 5V 供电，就可以完成图传。如果想查看摄像头的视频，首先将主

机连接到以 `wifi-robots.com` 开头的信号，便可以直接使用浏览器输入网址 <http://192.168.1.1:8080/?action=stream> 来查看视频图像。

忽略地图的道路宽度，使小车总是定位在道路的中线位置。使用地图的五个岔路口作为五个端点，五个端点的序号标识在图 3-1 中。小车位置使用和其连接的最近的两个端点和到端点的距离表示，两个端点不分先后顺序，但是距离一定要和端点对应。如果小车位于某个端点上，则该位置的两个端点均使用同一个数字，如小车位于端点 2，则小车位置为“2”“2”“0”“0”，小车位于 1 和 2 的中点，则小车位置为“1”“2”“127”“127”（端点 1、2 之间距离为 254cm）。



图 5-6 五个端点

从摄像头得到的视频中取出图片进行处理。首先是提取出感兴趣区域（小车标志物所能到达的范围），这里使用了透视变换（利用透视中心、像点、目标点三点共线的条件，按透视旋转定律使承影面绕迹线旋转某一角度，破坏原有的投影光线束，仍能保持承影面上投影几何图形不变的变换）。因为摄像头悬挂在地图的斜上方，所以得到的地图相对于俯视图来说是存在畸变的。采用透视变换的方法可以把地图矫正为近乎俯视图的样子。如下图所示，红色框内的部分是地图的兴趣区域，为了方便以后对于距离的估计，将其按照真实的几何大小透视变换。

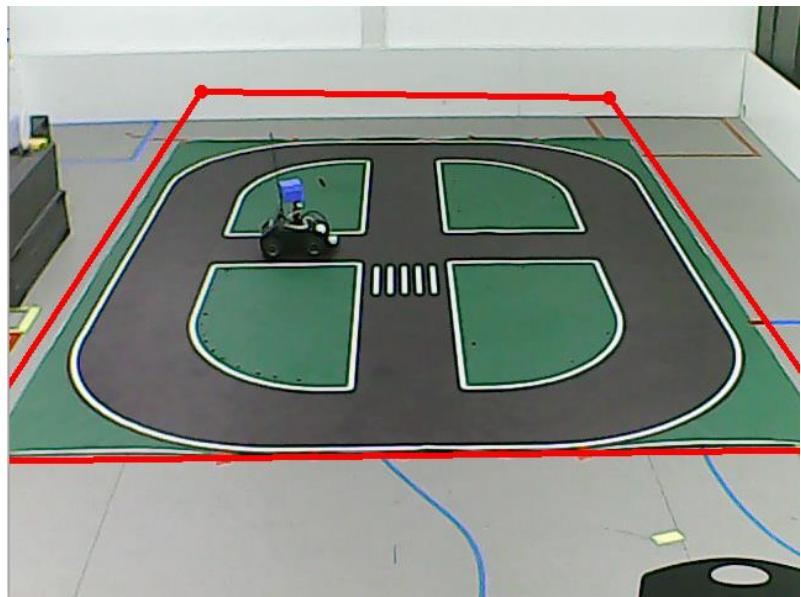


图 5-7 感兴趣区域



图 5-8 透视变换得到的地图

接着提取透视变换之后的地图的特殊颜色(为区别于其他颜色, 使用蓝色), 调整颜色阈值, 使得地图上只剩下小车的标志物。然后使用高斯模糊的方法, 去除掉可能存在的噪声, 最后提取到的小车标志物下图所示。可以看到, 此时的小地图(以后称透视变换之前的地图为大地图, 透视变换之后的地图为小地图)中只剩下小车标志物部分。

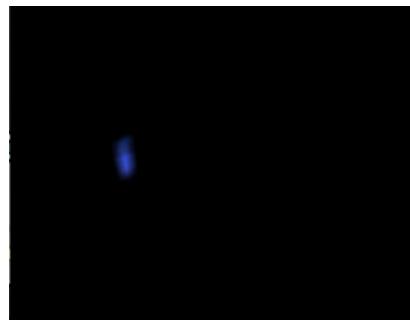


图 5-9 提取出小车的标志物

为了使得标志物和其他部分区分更为明显，对提取出来的图像进行灰度化和二值化处理，便得到了一个仅含数据 0 和 1 的二维矩阵，如图 3-5 所示。找到矩阵中非 0 数据的索引并求其平均值，即可得到小车标志物在小地图上的坐标值。但这个坐标值并不是真实的坐标值，这是因为摄像头是一个点，而小车的标志物是有高度的，在摄像头的图像中，小车标志物的坐标不等同于小车底部位置的坐标，所以还需要进行变换。变换的原理为相似三角形（空间上的），这里不在展开叙述。得到小车真实坐标之后便可以进行标定。标定可以使用不同的方法，只要能够根据坐标得到位置表示即可。将图中小车坐标转化为上文中介绍的小车位表示方式，即为“3”“2”“55”“60”。完成定位之后，需要将位置信息的发布出来以供其他程序使用。

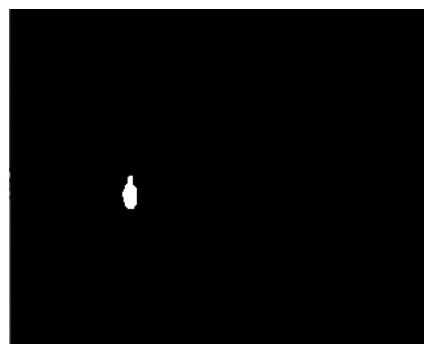


图 5-10 二值化后的小车的标志物

发布的消息类型为 location.msg，其具体格式如下：

```
int32 first_point #位置的第一个端点  
int32 second_point #位置的第二个端点  
int32 distance #位置距离第一个端点的距离
```

**注：**没有发送距离第二的端点的距离是因为其他程序中存有地图信息，可以自行计算。

### 5.2.1.2 主程序

"deepcar\_location.py" 位于  
inInCar\src\global\_path\_planning\path\_planning\srcs 路径底下。

```
class Arduino_location():
    def __init__(self):
        ... # 初始化程序

    def pub_location(self):
        ... # 发布获取到的位置

    def xy_true(self, x, y):
        ... # 将透视变换后的地图的 xy 值转化为真实值

    def xy_to_location(self, x, y):
        ... # 将 xy 真实值转化为需要的位置格式

    # 将原视频中的地图模块提取出来并处理
    def video_to_map(self):
        ... # 将原视频地图提取出来并处理

if __name__ == '__main__':
    try:
        mylocation = Arduino_location()
        while not rospy.is_shutdown():
            ret, frame = mylocation.cap.read() # 从视频中读取出一帧图像
            if ret == True:
                mylocation.video_to_map() # 对图像处理得到小车位置
                if cv2.waitKey(1) == 27:
                    exit(0)
            if not ret:
                break
    except:
        os._exit(0)
```

## 5.2.2 全局路径规划

### 5.2.2.1 介绍

完成定位之后便可以开始静态的全局路径规划。全局规划的目标也可以分为很多种，如总路程最短、总耗时最短、转弯次数最少等等。这里选用的最常用的指标——总路程最短。由于地图比较简单，只存在 5 个岔路口（即端点），且每个端点之间的最短距离和最短路径很容易看出，所以直接使用两个数组对他们进行存储。

从起始位置到期望位置的方式无非四种情况，即路径在位置的不同端点上（如果有位置在端点的情况，则四种情况中存在重复的现象）。遍历这些情况，

找出其中最小值，便可以得到最短路径。对于路径的表示使用了这样一个消息类型——Plan.msg，其格式如下：

```
string path_str #路径经过的端点序列
int32 start_add_point #起始位置两个端点中不在路径上的那个
int32 end_add_point #期望位置两个端点中不在路径上的那个
int32 start_distance #起始位置和起始位置两个端点中不路径上的那个之间的距离
int32 end_distance #期望位置距离end_add_point 的距离
```

### 5.2.2.2 主程序

"global\_path\_planning.py" 位于  
in inCar\src\global\_path\_planning\path\_planning\srcs 路径底下。

```
class car_location():
    #一个表示位置的类,用两个端点和距离端点的距离表示
    def __init__(self):
        ... #初始化程序

class Arduino_path():
    def __init__(self):
        ... #初始化程序

    def callback_plan(self, data):
        ... #路径控制消息回调函数, 判断是否到达期望位置

    def callback(self, data):
        ... #位置消息回调函数, 得到当前位置

    def path_planning(self):
        ... #主要循环程序, 输入期望位置, 得到最优路径, 发布出去

if __name__ == '__main__':
    try:
        mypath = Arduino_path()
        now_location = car_location() #当前位置
        except_location = car_location() #期望位置
        while not rospy.is_shutdown():
            mypath.path_planning() #主要循环程序
    except:
        os._exit(0)
```

### 5.2.3 路径控制

#### 5.2.3.1 介绍

在上一小节中，得到了最优路径，但这个路径并不能用来对小车进行控制，需要给底层规划传输直接的控制指令。对于小车的控制，需要给出转弯角度，是否停车等信息。这就需要制定出协议，也就是 Plan.msg，其格式如下：

```
bool flag_branch #岔路口标志, True 为是岔路口, False 为非岔路口
```

```

bool flag_stop #停车标志, True 为停车, False 为不停车
bool flag_turn #转弯标志, True 为需要转弯, False 为不需要转弯
bool flag_direction #转弯方向, True 为逆时针, False 为顺时针
float64 angle #转弯角度

```

对于控制消息的发布有这样几个触发点：

一是最开始控制的时候，默认小车开始朝向为向左（以透视变换得到的平面地图为视角），控制小车转向到可以正常巡线的方向；

二是小车经过端点的时候，将 `flag_branch` 置为真，判断此时是否需要转弯，如果需要转弯，将 `flag_trun` 置为真，并按照实际情况赋予 `flag_direction`, `angle` 相应的值；

三是小车达到指定位置的时候，将 `flag_stop` 置为真，控制小车停车，此时全局路径规划程序可以再次输入期望位置。

程序中具体的控制算法，由于太过复杂这里不再详细讲述，感兴趣的同學可以按照自己的需求阅读。

### 5.2.3.2 主程序

"control.py" 位于 `ininCar\src\global_path_planning\path_planning\srcs` 路径底下。

```

class Arduino_conrrol():
    def __init__(self):
        ... # 程序初始化

    def adjust_point_angle(self, data):
        ... # 初始点位于端点时的朝向调整

    def out_angle(self, x, num):
        ... # 弯道上的朝向

    def adjust_load_angle(self, data):
        ... # 初始点位于端点中间（道路上）的朝向调整

    def stop(self, data):
        ... # 根据实时位置，等待达到停车条件，发送有关消息

    def callback_location(self, data):
        ... # 位置消息回调函数，获取当前位置，并判断是否为岔路口

    def callback_path(self, data):
        ... # 路径消息回调函数，主要控制程序，根据路径持续控制直到小车达到期望为止

if __name__ == '__main__':
    try:

```

```

mycontrol = Arduino_control()
now_location = car_location()
while not rospy.is_shutdown():
    pass
except:
    os._exit(0)

```

## 5.2.4 实时地图显示

### 5.2.4.1 介绍

此部分主要为绘制地图，并在地图上根据小车的实时位置显示出来。原理也很简单，根据地图的实际尺寸，以每  $1\text{cm}^2$  为一个像素点，得到地图。设计一个接收器接受小车的位置消息，并将位置消息转化为需要绘制的 xy 坐标值，表示在地图上。为了保证地图能够一直显示，使用了 `waitKey(0)`，并位置消息的回调函数中不断修改显示的图像。得到的最后效果如图 3-6 所示。

**注：**真实地图的尺寸为：从端点 2 到端点 4 的距离为 230cm，从端点 1 到端点 5 的距离为 340cm，四个角分别为四个半径为 72cm 的四分之一圆。

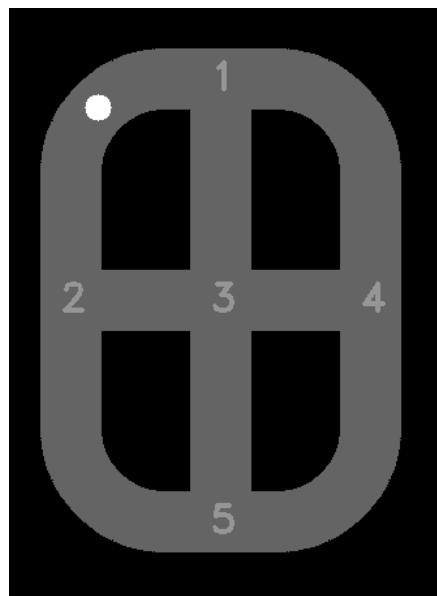


图 5-11 小车在地图上的位置

### 5.2.4.2 主程序

"map.py" 位于 `inInCar\src\global_path_planning\path_planning\srcs` 路径底下。

```

class Arduino_map():
    def __init__(self):
        ... # 程序初始化
    def base_map(self):

```

```

... # 绘制出没有小车的地图

def dis_to_xy(self):
    ... # 将小车位置转化为地图上的 xy 值

def callback(self, data):
    ... # 位置消息回调函数, 将小车展示在地图上
    if __name__ == '__main__':
        if __name__ == '__main__':
            try:
                mymap = Arduino_map()
                now_location = car_location()
                #显示地图
                cv2.namedWindow("deepcar")
                cv2.resizeWindow("deepcar", 540, 530)
                cv2.namedWindow("deepcar", cv2.WINDOW_AUTOSIZE)
                mymap.base_map()
                map_finall = mymap.map_base.copy()
                cv2.imshow("deepcar", map_finall)
                cv2.waitKey(0)
            except:
                os._exit(0)

```

## 5.2.5 指定路径程序

### 5.2.5.1 介绍

很多时候，我们不仅期望小车可以按照点到点的最短路径形式，也希望小车可以接受指定的路径控制。指定路径程序的功能就是使小车根据你期望的行驶路径进行行驶，这里的路径限制为在几个端点之间的行驶路径。

指定路径规划程序需要手动更改的参数为：

```

path_dir = 0 #小车摆放或行驶方向, 顺时针为0, 逆时针为1 (64行)
path_str = "1,4,3,5,2" #指定的路径, 中间使用逗号隔开 (65行)

```

### 5.2.5.2 主程序

"deepcar\_control.py" 位于  
in inCar\src\global\_path\_planning\path\_planning\srcs 路径底下。

```

class Arduino_conrrol():
    def __init__(self):
        ...# 程序初始化

    def callback_location(self, data):
        ...# 获取位置回调函数

```

```
def control(self):
    ...# 主要程序，根据路径控制小车

if __name__ == '__main__':
    try:
        mycontrol = Arduino_conrrol()
        now_location = car_location()
        mycontrol.control()

    except:
        os._exit(0)
```

### 5.2.6 示例程序

路径规划的有关程序需要配合寻线程序一起启动才能完成。启动巡线程序之前需要先启动相机，有关说明详见巡线部分，这里不再说明。

启动终端，运行如下命令，程序运行界面如下图所示（此时是最短路径规划程序，如果想要更改为指定路径规划程序，将 `11_global_path_planning.launch` 更改为 `12_global_path_planning.launch` 即可）：

```
cd ininCar/  
source devel/setup.bash  
cd launch/  
11 global path planning.launch
```

```
started rosrun server http://deepcar-desktop:37677/
SUMMARY           decision_control.py          11_global_path_planning.launch
=====
<launch>
PARAMETERS
  * name="arduino" pkg="ros_decision_python" type="xun_xian.py" output="screen"
  * /rosdistro: melodic
  * /rosversion: 1.14.3
</launch>
NODES
<node name="controller" pkg="ros_decision_python" type="decision_control.py">
</node>
  arduino (ros_decision_python/xun_xian.py)
  arduino02 (path_planning/global_path_planning.py)_control_car.yaml" command="load" />
  arduino03 (path_planning/control.py)
  arduino04 (path_planning/sub.py)

auto-starting new master [no1] pkg="path_planning" type="deepcar_location.py"
process[master]: started with pid [9909]
ROS_MASTER_URI=http://localhost:11311

setting /run_id to d6a58512-4e7e-11eb-a492-9e5cf22496e9
process[rosout-1]: started with pid [9920]
started core service [/rosout]
process[arduino-2]: started with pid [9927]
process[arduino02-3]: started with pid [9928]
process[arduino03-4]: started with pid [9929]
process[arduino04-5]: started with pid [9930]
Please input the first point of the except location: 1 <type="sub.py" output="screen">
Please input the second point of the except location: 2
Please input the distance to the first point of the except location: 100.0 <type="sub.py" output="screen">
```

图 5-12 程序运行界面

以下为 `11_global_path_planning.launch` 中的内容：

```
<!-- 程序. 控制小车路径规划程序.  
@init: 初始化设置小车速度为 0.2, 转弯的角速度为 0.3
```

```

@input: 输入为小车的期望位置
@note: 结合了路径规划和巡线程序
-->
<launch>
    <node name="arduino" pkg="ros_arduino_python" type="xun_xian.py"
output="screen">
        </node>
    <node name="controller" pkg="ros_decision_python"
type="decision_control_stop.py" output="screen">
        <rosparam
file="/home/deepcar/ininCar/config/2_decision_control_car.yaml "
command="load" />
        </node>

    <node name="arduino1" pkg="path_planning" type="deepcar_location.py"
output="screen">
        </node>
    <node name="arduino2" pkg="path_planning"
type="global_path_planning.py" output="screen">
        </node>
    <node name="arduino3" pkg="path_planning" type="control.py"
output="screen">
        </node>
    <node name="arduino4" pkg="ros_arduino_python" type="sub.py"
output="screen">
        </node>
    <node name="arduino5" pkg="ros_arduino_python" type="map.py"
output="screen">
        </node>
</launch>

```

以下为 12\_global\_path\_planning.launch 中的内容:

```

<!-- 程序. 控制小车指定路径程序.
@init: 初始化设置小车速度为 0.2, 转弯的角速度为 0.3
@input: 无输入, 但需修改方向和路径
@note: 结合了路径规划和巡线程序
-->
<launch>
    <node name="arduino" pkg="ros_arduino_python" type="xun_xian.py"
output="screen">
        </node>
    <node name="controller" pkg="ros_decision_python"
type="decision_control.py" output="screen">
        <rosparam
file="/home/deepcar/ininCar/config/2_decision_control_car.yaml "
command="load" />
        </node>

    <node name="arduino1" pkg="path_planning" type="deepcar_location.py"
output="screen">
        </node>
    <node name="arduino2" pkg="path_planning" type="deepcar_control.py"
output="screen">
        </node>
</launch>

```

```
<node name=" arduino4" pkg="ros_arduino_python" type="sub.py">
  output="screen">
</node>
<node name=" arduino5" pkg="ros_arduino_python" type="map.py">
  output="screen">
</node>
</launch>
```

## 5.3 局部路径规划

在完成环境感知和全局路径规划之后，需要有一个综合的决策部分，这就是本节将要介绍的局部路径规划。

局部路径规划要处理的信息有很多，比如从激光雷达程序传输过来的障碍物信息，从目标检测程序传输过来的物体类别信息从巡线控制程序输出的转角，从全局路径规划传输过来的停车标志、岔路口标志、转弯标志、转弯方向、转角等。除此之外，还有对小车速度和行驶里程的控制，即允许小车本次行驶多长的距离。怎么将这些信息统筹起来，控制小车运动，成为局部路径规划要解决的根本问题。

这里面最重要的是优先级的问题，理论分析可知，避障程序应该是优先级最高的程序。当激光雷达检测到前方存在障碍物的时候，小车将会自动执行避障程序，而忽视其他程序传递过来的消息。

当前方不存在障碍物时，行驶里程的控制便成为优先级最高的。当用户输入的行程里程数等于 0 时，代表着此时没有里程数的限制，小车可以一直运行下去；当用户输入的里程数不等于 0 时，小车无论运动情况如何，运动达到相应的里程数就会停止前进。

接着，下一级的优先级是目标检测。目标检测又分为对红绿灯的判断和对停车标志的判断。当在一定距离内检测到红绿灯为红灯状态或检测到停车标志时，小车会停止运动，直到检测检测到其他的标志才会继续运动；如果检测到了右转的标志牌，会根据此时所处的位置进行控制，如果是在弯道上右转，则直接巡线即可，如果是在岔路口右转，则需要原地顺时针旋转 90° 之后再继续行驶。

然后下一个全局路径规划优先级。这将会和全局路径规划进行配合。根据全局路径规划中发出的控制消息完成对小车的控制，如是否停车、转弯方向和角度等。

优先级最低的便是巡线了，上面所有过程都需要巡线的基础上进行。比如说，全局路径规划过程中只会在岔路口、起始点或期望点产生相关控制指令，而在这

些点之间运动都是依靠着小车巡线进行的。

将以上的控制思想用流程图绘制出来如下图所示。

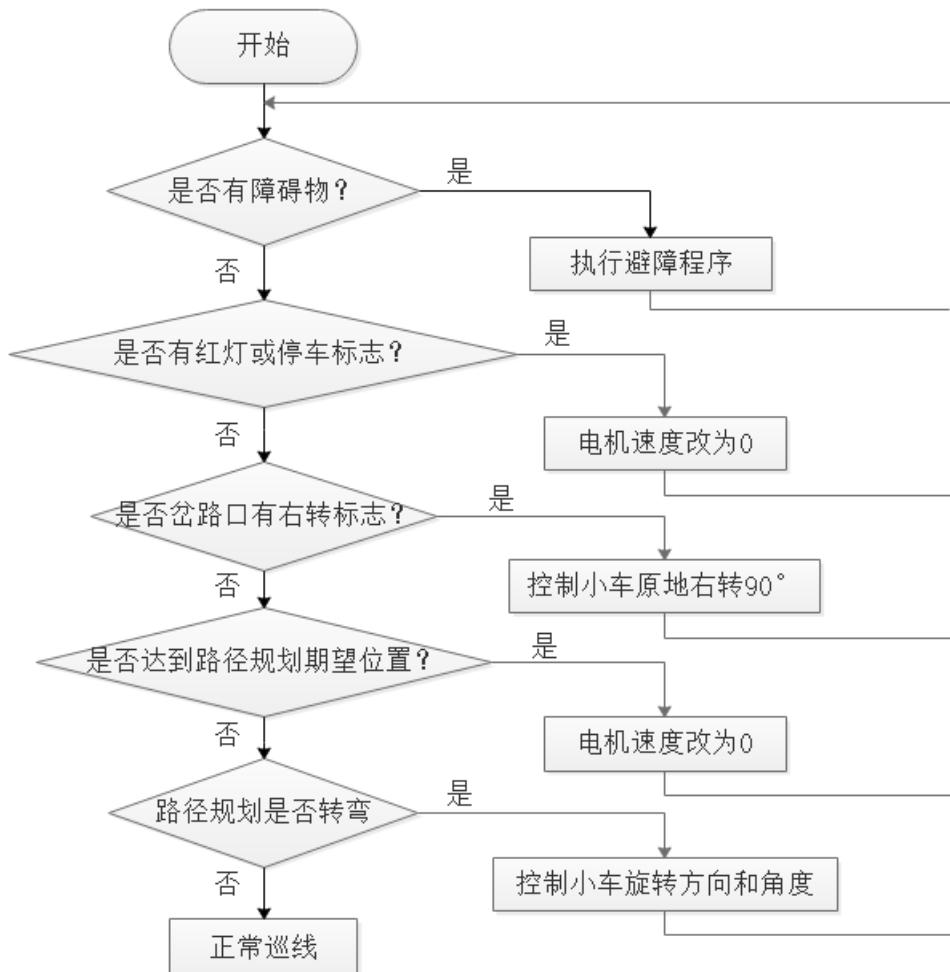


图 5-13 局部路径规划算法图

局部路径规划属于小车的底层控制部分，它需要接收所有上层的运行结果。因为小车除去主机之外还有一个 STM32 的电机控制板，所以这里的局部路径规划是根据上层的计算结果，计算出小车在连续时间内的两个电机的速度，并通过串口将其发送下去。

## 5.4 习题及参考答案

### 5.4.1 习题

- 1、路径规划根据对环境信息的把握程度可分为\_\_\_\_\_和\_\_\_\_\_两大类。
- 2、列写出至少三个常用的路径规划算法：\_\_\_\_\_
- 3、图像显示时，`waitKey(delay)`函数的作用。

4、试编写一段 Python 代码，打开笔记本的摄像头，并转化为灰度图之后显示出来，按 q 退出。

### 5.4.2 参考答案

1、全局路径规划 局部路径规划

2、人工势场法、解析法、激活值传播算法、搜索算法、遗传算法、模糊逻辑算法、神经网络算法、基于群智能的方法、基于案例学习的方法、强化学习算法、免疫算法、环境表示方法等（答出三点即可）

3、`waitKey(delay)`函数的功能是不断刷新图像，频率时间为 `delay`，单位为 ms。常用在 `imshow()`函数的后面，返回值为当前键盘按键值。

4、

```
import cv2
cap = cv2.VideoCapture(0)
while cap.isOpened():
    catch, frame = cap.read()
    img_gray = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)
    cv2.imshow("gray", img_gray)
    key = cv2.waitKey(10)
    if key & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

## 第三部分 技术手册

# 6 车辆控制

## 6.1 原理介绍

车辆控制的算法原理为：通过设定车辆运动的线速度（即前进速度）和角速度（即转弯速度），来控制车辆的运动状态和行驶轨迹。

下面，通过几个不同功能的示例程序，具体介绍车辆控制的相关算法。

## 6.2 示例：键盘控制小车行走

### 6.2.1 主程序"car\_control.py"

该程序位于 `inInCar\src\ros_arduino_bridge\ros_arduino_python\nodes` 路径下。运行该程序将会调用 OpenCV 库构建一个窗口：

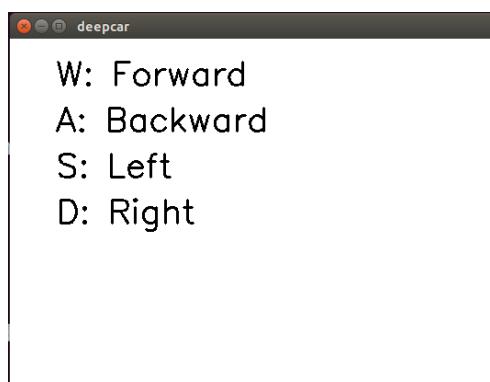


图 6-1 创建窗口图

注：鼠标需要点击上图所示窗口，才可有效检测键盘按键。

程序的主要内容如下：

```
class Arduino_Car(ArduinoROS):
    """
    @note: 继承小车底层控制类 ArduinoROS
    """
    def __init__(self):
        ... # 初始化程序

    def send_vel_cmd(self, vel_x, anv_z):
        ... # 向小车底层发布速度 vel_x 和转向角速度 anv_z

    def get_encoder(self):
        ... # 读取小车编码器计数
```

```

def stop_car(self):
    ... # 停止小车控制

def get_odom(self):
    ... # 返回小车底层编码器计数累计的里程:
        # odom_x(前进方向), odom_y(左右方向), odom_theta(转弯弧度)

def keyboard_control(car_x,car_th):
    ... # 键盘按键控制函数
    # 'a' : 转弯角速度加 0.01
    # 'd' : 转弯角速度减 0.01
    # 'w' : 小车速度加 0.01
    # 's' : 小车速度减 0.01
    # 'c' : 设置速度=0, 转向角速度=0
    # 'q' : 终止 ros 程序
    # 返回键盘按键设置的速度 car_x 与转弯角速度 car_th

if __name__ == '__main__':
    try:
        myArduino = Arduino_Car()
        r = rospy.Rate(50)
        car_x = 0 # 设置初始速度与转弯角速度为0
        car_th = 0

        while not rospy.is_shutdown():
            # 检测键盘按键, 获取键盘设置的速度及转弯角速度
            car_x,car_th = keyboard_control(car_x,car_th)
            # 向小车底层发布速度及转弯角速度
            myArduino.send_vel_cmd(car_x,car_th)
            r.sleep()
    except SerialException:
        rospy.logerr("Serial exception trying to open port.")
        os._exit(0)

```

此程序为利用键盘操纵小车行走，具体的小车按键控制功能如下：

按键	功能
W	增加小车速度
S	减少小车速度
A	增加小车逆时针旋转方向的角速度，即小车向左转弯运动
D	减少小车顺时针旋转方向的角速度，即小车向右转弯运动
C	将小车速度和转向置 0，即停车控制
Q	停车控制，并终止 python 程序

注：小车速度为负数时表示小车将向后运动。

## 6.2.2 程序运行

启动终端，运行如下命令，程序运行界面如下图所示：

```
cd ininCar/  
source devel/setup.bash  
cd launch/  
roslaunch 1_keyboard_control_car.launch
```

```
deepcar@deepcar-desktop:~$ cd ininCar/  
deepcar@deepcar-desktop:~/ininCar$ source devel/setup.bash  
deepcar@deepcar-desktop:~/ininCar$ cd launch/  
deepcar@deepcar-desktop:~/ininCar/launch$ roslaunch 1_keyboard_control_car.launch  
... logging to /home/deepcar/.ros/log/d4147ef4-202d-11eb-a631-9e5cf22496e9/roslaunch-deepcar-desktop-26239.log  
Checking log directory for disk usage. This may take awhile.  
Press Ctrl-C to interrupt.  
Done checking log file disk usage. Usage is <1GB.  
started roslaunch server http://192.168.43.171:40245/  
  
SUMMARY  
=====
```

PARAMETERS

- \* /arduino/avz: 0
- \* /arduino/base\_controller\_rate: 30
- \* /arduino/base\_frame: base\_link
- \* /arduino/baud: 115200
- \* /arduino/control\_x: 0
- \* /arduino/encoder\_resolution: 7420
- \* /arduino/gear\_reduction: 1.0
- \* /arduino/port: /dev深深车
- \* /arduino/rate: 30
- \* /arduino/sensorstate\_rate: 10
- \* /arduino/timeout: 0.2
- \* /arduino/use base controller: True
- \* /arduino/vel\_x: 0.2
- \* /arduino/wheel\_diameter: 0.1
- \* /arduino/wheel\_track: 0.444
- \* /rosdistro: melodic
- \* /rosversion: 1.14.3

NODES

- / arduino (ros\_arduino\_python/car\_control.py)

图 6-2 程序运行界面

命令注解：使用 `roslaunch` 调用 `1_keyboard_control_car.launch` 文件，启动小车控制程序。

其中，`launch` 文件的编写格式为：

使用 `<launch>` 声明 `launch` 内容开始，使用 `</launch>` 声明内容结束。

使用 `<node>` 声明结点，`</node>` 表示结点声明结束，如果声明的内容只有一行则 `<node “content” />` 即可：

- **name** 表示结点运行时的名称；
- **pkg** 表示结点所在的包；
- **type** 表示可执行文件名称；
- **output** 表示内容输出，`output=“screen”` 表示输出到屏幕上。

使用 `rosparam` 表示结点执行时的参数。

`1_keyboard_control_car.launch` 中的内容如下：

```
<!-- 程序 1. 启动键盘控制小车程序。  
@init: 初始化设置小车速度为 0, 转弯的角速度为 0
```

```

@param: W 加速. S 减速. A 左转弯. D 右转弯. C 停车(设置速度=0, 转向=0)
        Q 终止程序
@note: 主程序在 car_control.py 中, 参数文件为
1_keyboard_control_car.yaml
-->
<launch>
    <node name="arduino" pkg="ros_arduino_python" type="car_control.py"
output="screen">
        <rosparam file="../config/1_keyboard_control_car.yaml"
command="load" />
    </node>
</launch>

```

该 launch 文件运行了"car\_control.py"结点程序，并将 yaml 格式文件 "1\_keyboard\_control\_car.yaml" 导入作为结点执行时的参数。

## 6.3 示例：车道线巡线车辆控制

### 6.3.1 主程序"decision\_control.py"

该程序位于 ininCar\src\decision\ros\_decision\_python\nodes 路径下。

程序的相关内容如下：

```

...
    @note: 只给出 decision_control.py 中有关车道线巡线控制的程序
...
class Arduino_Car(ArduinoROS):
    # 继承类
    def __init__(self):
        ...
        # 初始化一个 Twist 消息
        vel_x_param = rospy.get_param("vel_x", 0.0)
        anv_z_param = rospy.get_param("anv_z", 0.0)
        self.cmd_vel.linear.x = vel_x_param
        self.cmd_vel.angular.z = anv_z_param
        ...
        self.forwardSpeed = 0.2           # 小车前进速度, 默认为 0.2
        self.angularSpeed = 0.3          # 小车转弯速度, 默认为 0.3
        ...
        # 巡线参数
        self.pid_p = 0.005             # PID 参数 Kp
        self.pid_d = 0.0004            # PID 参数 Kd
        self.lastErr = 0.0              # 上一次的偏差
        self.lineErr = 0.0              # 当前偏差
        ...

```

```

# 巡线订阅器
self.lane_error_result = rospy.Subscriber('/lane_error',
Lanesite, self.laneErrorCallback)

...
# 发布速度控制值
def send_vel_cmd(self,vel_x,anv_z):
    self.cmd_vel.linear.x = vel_x
    self.cmd_vel.angular.z = anv_z
    self.cmd_vel_pub.publish(self.cmd_vel)
    self.RunFunctionForArduino()

...
# 车道线巡线回调函数
def laneErrorCallback(self,data):
    self.lineErr = data.err
    #rospy.loginfo("receive_lerr %f", self.lineErr)
...

# 运动控制算法
def motion_control(self,car_x,car_th):
    ...
        # 车道线巡线 PID 控制
        car_x = self.forwardSpeed
        car_th = self.pid_p * self.lineErr + self.pid_d *
(self.lineErr - self.lastErr)
            #记录上一次小车的偏差, 给 pid 控制用
            self.lastErr = self.lineErr

    return car_x,car_th

# 主函数
if __name__ == '__main__':
    try:
        myArduino = Arduino_Car()

        r = rospy.Rate(25)
        # 设置初始速度为0
        car_x = 0
        car_th = 0

        while not rospy.is_shutdown():

            car_x,car_th = myArduino.motion_control(car_x,car_th)
            # 发布速度到电机

```

```

myArduino.send_vel_cmd(car_x,car_th)
r.sleep()

except SerialException:
    rospy.logerr("Serial exception trying to open port.")
    os._exit(0)

```

其算法原理为：当接收到车道线巡线程序发布过来的小车位置偏差后，通过 PID 控制得到小车角速度控制值，直接控制小车做巡线运动，PID 控制器选用离散 PD 控制算法。

### 6.3.2 程序执行

启动终端，运行如下命令，程序运行界面如下图所示：

```

cd ininCar/
source devel/setup.bash
cd launch/
roslaunch 8_decision_control.launch

```

注：上述命令执行前需开启相机。

```

deepcar@deepcar-desktop:~$ cd ininCar/
deepcar@deepcar-desktop:~/ininCar$ source devel/setup.bash
deepcar@deepcar-desktop:~/ininCar$ cd launch/
deepcar@deepcar-desktop:~/ininCar/launch$ roslaunch 8_decision_control.launch
... logging to /home/deepcar/.ros/log/d3a309f2-500e-11eb-aa03-9e5cf22496e9/roslaunch-deepcar-desktop-1145
4.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://deepcar-desktop:37953/

SUMMARY
=====

PARAMETERS
  * /controller/any_z: 0
  * /controller/base_controller_rate: 30
  * /controller/base_frame: base_link
  * /controller/baud: 115200
  * /controller/control_x: 0
  * /controller/encoder_resolution: 7420
  * /controller/gear_reduction: 1.0
  * /controller/port: /dev深深
  * /controller/rate: 30
  * /controller/sensorstate_rate: 10
  * /controller/timeout: 0.2
  * /controller/use_base_controller: True
  * /controller/vel_x: 0
  * /controller/wheel_diameter: 0.1
  * /controller/wheel_track: 0.37
  * /rostdistro: melodic

```

图 6-3 程序运行界面

命令注解：使用 roslaunch 调用 8\_decision\_control.launch 文件，启动车道线巡线车辆控制程序。

其中，8\_decision\_control.launch 中的内容如下：

```
<!-- 启动控制小车程序 -->
```

```

<launch>
  <node name="arduino" pkg="ros_decision_python" type="xun_xian.py"
output="screen">
  </node>
  <node name="controller" pkg="ros_decision_python"
type="decision_control.py" output="screen">
    <rosparam file="../config/2_decision_control_car.yaml"
command="load" />
  </node>
</launch>

```

该 launch 文件同时运行了两个结点程序——"xun\_xian.py"为车道线检测程序； "decision\_control.py"为小车运动决策控制程序，并将 yaml 格式文件 "2\_decision\_control\_car.yaml" 导入作为结点执行时的参数。

## 6.4 示例：键盘控制小车行驶里程和速度

### 6.4.1 主程序"sam\_control.py"

该程序位于 `iniinCar\src\decision\ros_decision_python\nodes` 路径下。

程序的主要内容如下：

```

def sam_talker():
    # 定义一个发布器
    pub = rospy.Publisher('/speedMile', speedANDmile, queue_size=10)
    # 初始化一个结点
    rospy.init_node('sam_talker', anonymous=True)
    # 设置程序执行频率为 10Hz
    rate = rospy.Rate(10) # 10hz

    while not rospy.is_shutdown():
        # 获取键盘输入值
        setValue = raw_input()
        # 当输入为 q 时，停止该程序
        if setValue == 'q':
            print "END"
            break
        # 将键盘输入的字符串按，拆分并赋值
        first_value, second_value = setValue.split(',')
        temp = speedANDmile()
        # 将字符串转化为浮点数

```

```

        temp.set_speed = float(first_value)
        temp.set_mile = float(second_value)
        # 发布速度控制值和里程控制值
        pub.publish(temp)
        rate.sleep()

    if __name__ == '__main__':
        try:
            sam_talker()

        except rospy.ROSInterruptException:
            pass

```

运行该程序后，在键盘输入[浮点数 1,浮点数 2]，然后 Enter，即可将速度设定值和里程设定值发布出来，供决策控制程序处理（其中，浮点数 1 为速度设定值，浮点数 2 为里程设定值）；当键盘输入为“q”并回车时，该发布程序退出运行。

#### 6.4.2 主程序"decision\_control.py"

该程序位于 `ininCar\src\decision\ros_decision_python\nodes` 路径下。

程序的相关内容如下：

```

...
    @note: 只给出 decision_control.py 中有关键盘控制的程序
...

class Arduino_Car(ArduinoROS):
    # 继承类
    def __init__(self):
        ...
        # 键盘控制参数
        self.mainStop = False          # 优先级最高的停止标志，默认为否
        self.forwardSpeed = 0.2         # 小车前进速度，默认为 0.2
        self.angularSpeed = 0.3         # 小车转弯速度，默认为 0.3
        self.mile = 0.0                 # 小车行驶里程设置值，默认为 0
        self.wholeMile = 0.0            # 小车总行驶里程，默认为 0
        self.milePoint = True           # 是否开始里程控制，默认为是
        self.temp_now1 = 0.0             # 本次中间变量 1
        self.temp_now2 = 0.0             # 本次中间变量 2
        self.temp_now3 = 0.0             # 本次中间变量 3
        self.temp_last1 = 0.0            # 上一次中间变量 1
        self.temp_last2 = 0.0            # 上一次中间变量 2

```

```

    ...
# 键盘控制订阅器
self.sam_control_result = rospy.Subscriber('/speedMile',
speedANDmile, self.samCallback)

# 发布速度控制值
def send_vel_cmd(self,vel_x,anv_z):
    self.cmd_vel.linear.x = vel_x
    self.cmd_vel.angular.z = anv_z
    self.cmd_vel_pub.publish(self.cmd_vel)
    self.RunFunctionForArduino()

...
# 获取里程计值
def get_odom(self):
    return self.odom_x,self.odom_y,self.odom_theta
...

# 键盘控制回调函数，获取发布器值
def samCallback(self,data):
    self.forwardSpeed = data.set_speed
    self.mile = data.set_mile
    self.mainStop = False
    self.milePoint = True
    print 'Set successfully: Speed: %.3f ; Mile: %.3f
' %(self.forwardSpeed, self.mile)

# 运动控制算法
def motion_control(self,car_x,car_th):
    # 如果接收到的里程数设定值不为0
    if self.mile != 0.0:
        # 读取里程计对应值
        self.temp_now1, self.temp_now2, self.temp_now3 =
self.get_odom()
        # 获取计算里程计初始值
        if self.milePoint:
            self.temp_last1 = self.temp_now1
            self.temp_last2 = self.temp_now2
            self.milePoint = False
        # 计算当前里程计x 和y 方向的差值
        temp1 = self.temp_now1 - self.temp_last1
        temp2 = self.temp_now2 - self.temp_last2
        # 计算当前里程值并累加
        self.wholeMile += math.sqrt(temp1*temp1 + temp2*temp2)

```

```

# 若里程累计值大于等于设定值，停车
if self.wholeMile >= self.mile:
    self.mainStop = True
    self.milePoint = True
    self.wholeMile = 0.0
    self.temp_last1 = self.temp_now1
    self.temp_last2 = self.temp_now2
    #print self.wholeMile

if self.mainStop:
    car_x = 0
    car_th = 0
    #print "main stop"

else:
    ...
    car_x = self.forwardSpeed # 设定小车线速度
    ...

return car_x,car_th

# 主函数
if __name__ == '__main__':
    try:
        myArduino = Arduino_Car()

        r = rospy.Rate(25)
        # 设置初始速度为0
        car_x = 0
        car_th = 0

        while not rospy.is_shutdown():

            car_x,car_th = myArduino.motion_control(car_x,car_th)
            myArduino.send_vel_cmd(car_x,car_th) # 发布速度控制值
            r.sleep()

    except SerialException:
        rospy.logerr("Serial exception trying to open port.")
        os._exit(0)

```

其算法原理为：当接收到"sam\_control.py"发布出来的小车行驶速度设定值

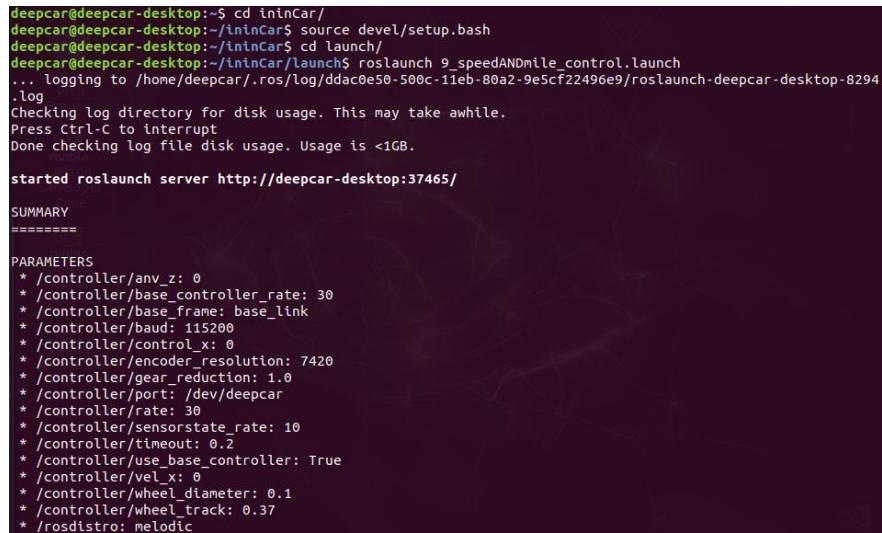
和行驶里程设定值后，首先修改决策控制程序中的速度和里程值，修改完成在终端界面显示修改成功。则小车的行驶速度直接修改为用户设定值；行驶里程若用户设定值为 0，则小车无行驶里程限制；若设定值不为 0，则根据里程计计算小车前一时刻的位置以及当前位置，计算两者距离，并逐渐累积作为小车的实际里程值；当实际里程值大于等于用户设定里程值，停车。然后，继续等待下次用户设定。

### 6.4.3 程序执行

启动终端，运行如下命令，程序运行界面如下图所示：

```
cd ininCar/  
source devel/setup.bash  
cd launch/  
roslaunch 9_speedANDmile_control.launch
```

注：上述命令执行前需开启相机。



```
deepcar@deepcar-desktop:~$ cd ininCar/  
deepcar@deepcar-desktop:~/ininCar$ source devel/setup.bash  
deepcar@deepcar-desktop:~/ininCar$ cd launch/  
deepcar@deepcar-desktop:~/ininCar/launch$ rosrun 9_speedANDmile_control.launch  
... logging to /home/deepcar/.ros/log/ddac0e50-500c-11eb-80a2-9e5cf22496e9/roslaunch-deepcar-desktop-8294  
.log  
Checking log directory for disk usage. This may take awhile.  
Press Ctrl-C to interrupt  
Done checking log file disk usage. Usage is <1GB.  
started roslaunch server http://deepcar-desktop:37465/  
  
SUMMARY  
=====
```

PARAMETERS

- \* /controller/any\_z: 0
- \* /controller/base\_controller\_rate: 30
- \* /controller/base\_frame: base\_link
- \* /controller/baud: 115200
- \* /controller/control\_x: 0
- \* /controller/encoder\_resolution: 7420
- \* /controller/gear\_reduction: 1.0
- \* /controller/port: /dev深深车
- \* /controller/rate: 30
- \* /controller/sensorstate\_rate: 10
- \* /controller/timeout: 0.2
- \* /controller/use\_base\_controller: True
- \* /controller/vel\_x: 0
- \* /controller/wheel\_diameter: 0.1
- \* /controller/wheel\_track: 0.37
- \* /rosdistro: melodic

图 6-4 程序运行界面

键盘控制效果如下图所示。

```

Connected at 115200
Arduino is ready.
[INFO] [1609930265.289677]: Connected to Arduino on port /dev/deepcar at 115200 baud
[0] use BaseController
Updating PID parameters
[INFO] [1609930265.406894]: Started base controller for a base of 0.37m wide with 7420 ticks per rev
[INFO] [1609930265.412198]: Publishing odometry data at: 30.0 Hz using base_link as base frame
[0] init arduino_node complete.
[1] start init arduino_car .
[2] start init control_param .
0.2,0.1
Set successfully: Speed: 0.200 ; Mile: 0.100
0.2,0.0
Set successfully: Speed: 0.200 ; Mile: 0.000
0.1,0.3
Set successfully: Speed: 0.100 ; Mile: 0.300
q
END
[publisher-3] process has finished cleanly
log file: /home/deepcar/.ros/log/0ff27c64-500d-11eb-9d1f-9e5cf22496e9/publisher-3*.log
^C[controller-4] killing on exit
[arduino-2] killing on exit
[INFO] [1609930308.004295]: Shutting down Arduino Node...
stopping the robot
[INFO] [1609930310.024066]: Serial port closed.
[rosout-1] killing on exit
[master] killing on exit
shutting down processing monitor...
... shutting down processing monitor complete
done
deepcar@deepcar-desktop:~/inInCar/launch$ 

```

图 6-5 键盘控制效果运行界面

注：当希望停止程序时，应首先输入"q"并回车，停止键盘输入程序，再 Ctrl-C 结束程序进程。

命令注解：使用 roslaunch 调用 9\_speedANDmile\_control.launch 文件，启动小车速度和里程键盘控制程序。

其中，9\_speedANDmile\_control.launch 中的内容如下：

```

<!-- 启动小车运动速度和里程键盘控制程序 -->

<launch>
  <node name="arduino" pkg="ros_decision_python" type="xun_xian.py"
        output="screen">
    </node>
    <node name="publisher" pkg="ros_decision_python" type="sam_control.py"
          output="screen">
    </node>
    <node name="controller" pkg="ros_decision_python"
          type="decision_control.py" output="screen">
      <rosparam file="../config/2_decision_control_car.yaml"
                command="load" />
    </node>
  </launch>

```

该 launch 文件同时运行了三个结点程序——"xun\_xian.py"为车道线检测程序； "sam\_control.py"为键盘指令输入程序； "decision\_control.py"为小车运动决策控制程序，并将 yaml 格式文件"2\_decision\_control\_car.yaml"导入作为结点执行时的参数。

## 6.5 示例：激光雷达障碍物检测与车辆控制

### 6.5.1 主程序"decision\_control.py"

该程序位于 `ininCar\src\decision\ros_decision_python\nodes` 路径下。

程序的相关内容如下：

```
...
@note: 只给出 decision_control.py 中有关激光雷达障碍物检测车辆控制的程序
...

class Arduino_Car(ArduinoROS):
    # 继承类
    def __init__(self):
        ...
        # 初始化一个 Twist 消息
        vel_x_param = rospy.get_param("vel_x", 0.0)
        anv_z_param = rospy.get_param("anv_z", 0.0)
        self.cmd_vel.linear.x = vel_x_param
        self.cmd_vel.angular.z = anv_z_param
        ...
        self.forwardSpeed = 0.2           # 小车前进速度，默认为 0.2
        self.angularSpeed = 0.3          # 小车转弯速度，默认为 0.3
        ...
        # 激光雷达参数
        self.obstacleStatus = False      # 障碍物判断，默认为无
        ...
        # 激光雷达订阅器
        self.lidar_result = rospy.Subscriber('/ros_tutorial_msg',
                                             lidarInfo, self.lidarCallback)
        ...
        # 发布速度控制值
        def send_vel_cmd(self, vel_x, anv_z):
            self.cmd_vel.linear.x = vel_x
            self.cmd_vel.angular.z = anv_z
            self.cmd_vel_pub.publish(self.cmd_vel)
            self.RunFunctionForArduino()
        ...
        # 激光雷达回调函数
        def lidarCallback(self, data):
            if data.flag == 1:
                self.obstacleStatus = True
            else:
```

```

        self.obstacleStatus = False
        #rospy.loginfo("receive_Lidar %f", self.distanceValue)
        ...
# 运动控制算法
def motion_control(self,car_x,car_th):
    ...
    # 激光雷达判断
    if self.obstacleStatus:
        car_x = 0
        car_th = 0
        print "find obstacle!" ,self.obstacleStatus

    else:
        ...

    return car_x,car_th

# 主函数
if __name__ == '__main__':
    try:
        myArduino = Arduino_Car()

        r = rospy.Rate(25)
        # 设置初始速度为0
        car_x = 0
        car_th = 0

        while not rospy.is_shutdown():

            car_x,car_th = myArduino.motion_control(car_x,car_th)
            myArduino.send_vel_cmd(car_x,car_th)
            r.sleep()

    except SerialException:
        rospy.logerr("Serial exception trying to open port.")
        os._exit(0)

```

其算法原理为：当接收到激光雷达障碍物检测程序发布的障碍物信息时，若前方有障碍物，修改停车标志位 self.obstacleStatus 为真，停车；若前方无障碍物，修改停车标志位 self.obstacleStatus 为假，执行其他运动控制程序。

## 6.5.2 程序执行

启动终端，运行如下命令，程序运行界面如下图所示：

```
cd ininCar/  
source devel/setup.bash  
cd launch/  
roslaunch 8_decision_control.launch
```

注：上述命令执行前需开启激光雷达，并执行激光雷达障碍物检测模块程序。

```
deepcar@deepcar-desktop:~$ cd ininCar/  
deepcar@deepcar-desktop:~/ininCar$ source devel/setup.bash  
deepcar@deepcar-desktop:~/ininCar$ cd launch/  
deepcar@deepcar-desktop:~/ininCar/launch$ roslaunch 8_decision_control.launch  
... logging to /home/deepcar/.ros/log/d3a309f2-500e-11eb-aa03-9e5cf22496e9/roslaunch-deepcar-desktop-1145  
4.log  
Checking log directory for disk usage. This may take awhile.  
Press Ctrl-C to interrupt  
Done checking log file disk usage. Usage is <1GB.  
  
started roslaunch server http://deepcar-desktop:37953/  
  
SUMMARY  
=====
```

PARAMETERS

- \* /controller/anv\_z: 0
- \* /controller/base\_controller\_rate: 30
- \* /controller/base\_frame: base\_link
- \* /controller/baud: 115200
- \* /controller/control\_x: 0
- \* /controller/encoder\_resolution: 7420
- \* /controller/gear\_reduction: 1.0
- \* /controller/port: /dev深深车
- \* /controller/rate: 30
- \* /controller/sensorstate\_rate: 10
- \* /controller/timeout: 0.2
- \* /controller/use\_base\_controller: True
- \* /controller/vel\_x: 0
- \* /controller/wheel\_diameter: 0.1
- \* /controller/wheel\_track: 0.37
- \* /rosdistro: melodic

图 6-6 程序运行界面

命令注解：使用 roslaunch 调用 8\_decision\_control.launch 文件，启动车道线巡线车辆控制程序。

其中，8\_decision\_control.launch 中的内容同上。

## 6.6 示例：全局路径规划与车辆控制

### 6.6.1 主程序"decision\_control.py"

该程序位于 ininCar\src\decision\ros\_decision\_python\nodes 路径下。

程序的相关内容如下：

```
'''  
    @note: 只给出 decision_control.py 中有关全局路径规划车辆控制的程序  
'''  
  
class Arduino_Car(ArduinoROS):  
    # 继承类
```

```

def __init__(self):
    ...
    # 初始化一个 Twist 消息
    vel_x_param = rospy.get_param("vel_x", 0.0)
    anv_z_param = rospy.get_param("anv_z", 0.0)
    self.cmd_vel.linear.x = vel_x_param
    self.cmd_vel.angular.z = anv_z_param
    ...
    self.forwardSpeed = 0.2           # 小车前进速度, 默认为 0.2
    self.angularSpeed = 0.3          # 小车转弯速度, 默认为 0.3
    ...
    # 路程计读数
    self.now_x_value = 0             # 当前里程计 x 值
    self.now_y_value = 0             # 当前里程计 y 值
    self.now_theta_value = 0         # 当前里程计角度值
    self.start_theta_value = 0       # 转弯初始点里程计角度值
    self.d_theta_value = 0           # 里程计角度差值
    self.turnPoint = True           # 判断是否为转弯初始点
    self.right_angle = 1.54          # 90 度
    self.turn_flag = False          # 判断是否为转弯标志
    ...
    # 全局路径规划参数
    self.branchStatus = False        # 叉路口判断, 默认为否
    self.stopStatus = False          # 停车判断, 默认为否
    self.turnStatus = False          # 原地转弯判断, 默认为否
    self.directionStatus = False     # 原地转弯方向判断, 默认为顺时针
    self.angle = 0.0                 # 原地转弯角度设定, 默认为 0
    ...
    # 全局路径规划订阅器
    self.global_plan_result = rospy.Subscriber('/global_plan',
                                                globalPlan, self.globalPlanCallback)
    ...
    # 发布速度控制值
    def send_vel_cmd(self, vel_x, anv_z):
        self.cmd_vel.linear.x = vel_x
        self.cmd_vel.angular.z = anv_z
        self.cmd_vel_pub.publish(self.cmd_vel)
        self.RunFunctionForArduino()
    ...
    # 获取里程计值
    def get_odom(self):
        return self.odom_x, self.odom_y, self.odom_theta

```

```

...
# 全局规划回调函数
def globalPlanCallback(self,data):
    self.branchStatus = data.flag_branch
    self.stopStatus = data.flag_stop
    self.turnStatus = data.flag_turn
    self.directionStatus = data.flag_direction
    self.angle = data.angle

# 运动控制算法
def motion_control(self,car_x,car_th):
    ...
        #路径规划
    if self.stopStatus:
        car_x = 0
        car_th = 0

    elif self.turnStatus:
        self.now_x_value, self.now_y_value,
self.now_theta_value = self.get_odom()

        if self.turnPoint:
            self.start_theta_value = self.now_theta_value
            self.turnPoint = False

        self.d_theta_value = abs(self.now_theta_value -
self.start_theta_value)

        if self.directionStatus == False:
            #右转self.angle 度
            car_x = 0
            car_th = -self.angularSpeed
        else:
            #左转self.angle 度
            car_x = 0
            car_th = self.angularSpeed

        if self.d_theta_value > self.angle:
            self.turnStatus = False
            self.turnPoint = True

    else:

```

```

# 车道线巡线 PID 控制
...
return car_x,car_th

# 主函数
if __name__ == '__main__':
    try:
        myArduino = Arduino_Car()

        r = rospy.Rate(25)
        # 设置初始速度为0
        car_x = 0
        car_th = 0

        while not rospy.is_shutdown():

            car_x,car_th = myArduino.motion_control(car_x,car_th)
            myArduino.send_vel_cmd(car_x,car_th)
            r.sleep()

    except SerialException:
        rospy.logerr("Serial exception trying to open port.")
        os._exit(0)

```

其算法原理为：通过接收全局路径规划程序发布消息，改变全局路径规划算法的相关标志位，从而实现全局路径规划车辆控制。实时判断当停车标志位 self.stopStatus 为真时，车辆停止运动；当停车标志位 self.stopStatus 为假时，判断转弯标志位 self.turnStatus 的状态：当转弯标志位为真时，判断转弯方向标志位 self.directionStatus 的状态：为假，车辆的角速度方向为顺时针；为真，车辆的角速度方向为逆时针；并读取转角值 self.angle，通过里程计转角的读取，控制车辆转 self.angle 大小的角度值。当转弯标志位为假时，执行车道线巡线程序，控制小车在全局路径规划的过程中在车道中间行驶。

## 6.6.2 程序执行

启动终端，运行如下命令，程序运行界面如下图所示：

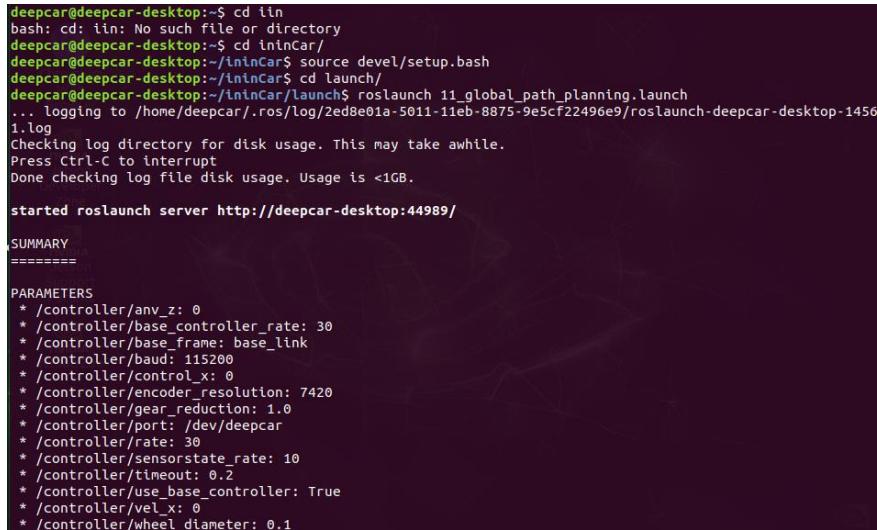
```

cd ininCar/
source devel/setup.bash

```

```
cd launch/
roslaunch 11_global_path_planning.launch
```

注：上述命令执行前需开启相机。



```
deepcar@deepcar-desktop:~$ cd iin
bash: cd: iin: No such file or directory
deepcar@deepcar-desktop:~$ cd ininCar/
deepcar@deepcar-desktop:~/inInCar$ source devel/setup.bash
deepcar@deepcar-desktop:~/inInCar$ cd launch/
deepcar@deepcar-desktop:~/inInCar/launch$ roslaunch 11_global_path_planning.launch
... logging to /home/deepcar/.ros/log/2ed8e01a-5011-11eb-8875-9e5cf22496e9/roslaunch-deepcar-desktop-1456
1.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://deepcar-desktop:44989/

SUMMARY
=====
PARAMETERS
  * /controller/avng_z: 0
  * /controller/base_controller_rate: 30
  * /controller/base_frame: base_link
  * /controller/baud: 115200
  * /controller/control_x: 0
  * /controller/encoder_resolution: 7420
  * /controller/gear_reduction: 1.0
  * /controller/port: /dev深深
  * /controller/rate: 30
  * /controller/sensorstate_rate: 10
  * /controller/timeout: 0.2
  * /controller/use_base_controller: True
  * /controller/vel_x: 0
  * /controller/wheel_diameter: 0.1
```

图 6-7 程序运行界面

命令注解：使用 roslaunch 调用 11\_global\_path\_planning.launch 文件，启动全局路径规划车辆控制程序。

其中，11\_global\_path\_planning.launch 中的内容如下：

```
<!-- 程序。控制小车路径规划程序。
@init: 初始化设置小车速度为 0.2，转弯的角速度为 0.3
@input: 输入为小车的期望为止
@note: 结合了路径规划和巡线程序
-->
<launch>
  <node name="arduino" pkg="ros_decision_python" type="xun_xian.py"
output="screen">
  </node>
  <node name="location" pkg="path_planning" type="deepcar_location.py"
output="screen">
  </node>
  <node name="path" pkg="path_planning" type="global_path_planning.py"
output="screen">
  </node>
  <node name="control" pkg="path_planning" type="control.py"
output="screen">
  </node>
  <node name="planning" pkg=" path_planning" type="sub.py"
output="screen">
  </node>
  <node name="controller" pkg="ros_decision_python"
type="decision_control.py" output="screen">
    <rosparam file="../config/2_decision_control_car.yaml"
command="load" />
```

```
</node>
</launch>
```

该 launch 文件同时运行了六个结点程序——"xun\_xian.py"为车道线检测程序; "deepcar\_location.py"、"global\_path\_planning.py"、"control.py"、"sub.py"为全局路径规划相关程序; "decision\_control.py"为小车运动决策控制程序, 并将 yaml 格式文件"2\_decision\_control\_car.yaml"导入作为结点执行时的参数。

## 6.7 示例：目标检测与车辆控制

### 6.7.1 主程序"decision\_control.py"

该程序位于 `ininCar\src\decision\ros_decision_python\nodes` 路径下。

程序的相关内容如下:

```
...
@note: 只给出 decision_control.py 中有关目标检测车辆控制的程序
...

# 枚举类, 代表目标识别红绿灯状态
class ObjInfoType(IntEnum):
    BACKGROUND = 0
    RED = 1
    GREEN = 2
    YELLOW = 3
    RIGHT = 4
    STOP = 5

class Arduino_Car(ArduinoROS):
    # 继承类
    def __init__(self):
        ...
        # 初始化一个 Twist 消息
        vel_x_param = rospy.get_param("vel_x", 0.0)
        anv_z_param = rospy.get_param("anv_z", 0.0)
        self.cmd_vel.linear.x = vel_x_param
        self.cmd_vel.angular.z = anv_z_param
        ...
        self.forwardSpeed = 0.2                      # 小车前进速度, 默认为 0.2
        self.angularSpeed = 0.3                      # 小车转弯速度, 默认为 0.3
        ...
        # 里程计读数
        self.now_x_value = 0                         # 当前里程计 x 值
```

```

        self.now_y_value = 0           # 当前里程计y值
        self.now_theta_value = 0       # 当前里程计角度值
        self.start_theta_value = 0    # 转弯初始点里程计角度值
        self.d_theta_value = 0        # 里程计角度差值
        self.turnPoint = True         # 判断是否为转弯初始点
        self.right_angle = 1.54       # 90度
        self.turn_flag = False        # 判断是否为转弯标志

        # 目标检测参数
        self.detectStatus = ObjInfoType.BACKGROUND      # 目标状态， 默认值为赛道
        ...
        # 全局路径规划参数
        self.branchStatus = False      # 直路口判断，默认为否
        ...
        # 目标检测订阅器
        self.object_detection_result =
rospy.Subscriber("/object_detection", detectionInfo,
self.objDetectionCallback)
        ...
        # 发布速度控制值
def send_vel_cmd(self,vel_x,anv_z):
        self.cmd_vel.linear.x = vel_x
        self.cmd_vel.angular.z = anv_z
        self.cmd_vel_pub.publish(self.cmd_vel)
        self.RunFunctionForArduino()
        ...
        # 获取里程计值
def get_odom(self):
        return self.odom_x,self.odom_y,self.odom_theta

        # 目标检测回调函数
def objDetectionCallback(self,data):
#rospy.Loginfo("receive_obj %s", data.label)
        if data.label == "red":
            self.detectStatus = ObjInfoType.RED
        elif data.label == "green":
            self.detectStatus = ObjInfoType.GREEN
        elif data.label == "yellow":
            self.detectStatus = ObjInfoType.YELLOW
        elif data.label == "right":
            self.detectStatus = ObjInfoType.RIGHT

```

```

    elif data.label == "stop":
        self.detectStatus = ObjInfoType.STOP
    else:
        self.detectStatus = ObjInfoType.BACKGROUND
    ...
# 运动控制算法
def motion_control(self,car_x,car_th):
    ...
    # 目标检测判断
    if self.detectStatus == ObjInfoType.RED or
self.detectStatus == ObjInfoType.YELLOW or self.detectStatus ==
ObjInfoType.STOP:
        car_x = 0
        car_th = 0
        print "find object!",self.detectStatus

    elif (self.detectStatus == ObjInfoType.RIGHT or
self.turn_flag) and self.branchStatus:
        # 右转为岔路口时，原地向右旋转90度
        print "find branch and turn right!"
        self.turn_flag = True
        self.now_x_value, self.now_y_value,
self.now_theta_value = self.get_odom()

        if self.turnPoint:
            self.start_theta_value = self.now_theta_value
            self.turnPoint = False

        self.d_theta_value = abs(self.now_theta_value -
self.start_theta_value)
        car_x = 0
        car_th = -self.angularSpeed

        if self.d_theta_value > self.right_angle:
            self.branchStatus = False
            self.turn_flag = False
            self.turnPoint = True

    else:
        ...
return car_x,car_th

```

```

# 主函数
if __name__ == '__main__':
    try:
        myArduino = Arduino_Car()

        r = rospy.Rate(25)
        # 设置初始速度为0
        car_x = 0
        car_th = 0

        while not rospy.is_shutdown():

            car_x,car_th = myArduino.motion_control(car_x,car_th)
            myArduino.send_vel_cmd(car_x,car_th)
            r.sleep()

    except SerialException:
        rospy.logerr("Serial exception trying to open port.")
        os._exit(0)

```

其算法原理为：当接收到目标检测程序发布出来的目标信息，修改决策控制程序中的目标状态 self.detectStatus，通过目标状态的不同，控制车辆的运动状态。当目标状态为红灯、黄灯和停车标志时，小车停止运动；当目标状态为右转标志时，判断右转位置是否为岔路口（此处通过全局路径规划程序发布的 self.branchStatus 状态进行判断），如果是岔路口，令车辆原地右转 90 度；如果不是岔路口，车辆保持全局路径规划和车道线巡线控制的运动状态；当目标状态为绿灯时，车辆也保持全局路径规划和车道线巡线控制的运动状态不变。

### 6.7.2 程序执行

启动终端，运行如下命令，程序运行界面如下图所示：

```

cd ininCar/
source devel/setup.bash
cd launch/
roslaunch 8_decision_control.launch

```

注：上述命令执行前需开启相机，并执行目标检测模块程序。

```

deepcar@deepcar-desktop:~$ cd inInCar/
deepcar@deepcar-desktop:~/inInCar$ source devel/setup.bash
deepcar@deepcar-desktop:~/inInCar$ cd launch/
deepcar@deepcar-desktop:~/inInCar/launch$ roslaunch 8_decision_control.launch
... logging to /home/deepcar/.ros/log/d3a309f2-500e-11eb-aa03-9e5cf22496e9/roslaunch-deepcar-desktop-1145
4.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://deepcar-desktop:37953/

SUMMARY
=====
PARAMETERS
  * /controller/anv_z: 0
  * /controller/base_controller_rate: 30
  * /controller/base_frame: base_link
  * /controller/baud: 115200
  * /controller/control_x: 0
  * /controller/encoder_resolution: 7420
  * /controller/gear_reduction: 1.0
  * /controller/port: /dev深深车
  * /controller/rate: 30
  * /controller/sensorstate_rate: 10
  * /controller/timeout: 0.2
  * /controller/use_base_controller: True
  * /controller/vel_x: 0
  * /controller/wheel_diameter: 0.1
  * /controller/wheel_track: 0.37
  * /rosdistro: melodic

```

图 6-8 程序运行界面

命令注解：使用 `roslaunch` 调用 `8_decision_control.launch` 文件，启动车道线巡线车辆控制程序。

其中，`8_decision_control.launch` 中的内容同上。

## 6.8 车辆决策控制综合算法流程

将上述 7.3、7.4、7.5、7.6、7.7 车辆控制算法相结合，得到车辆决策控制综合算法流程图如下所示。

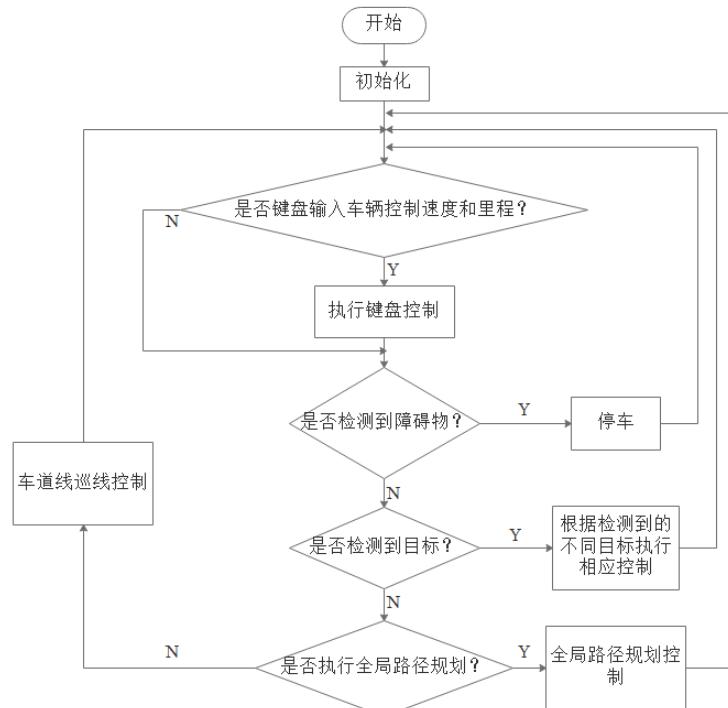


图 6-9 车辆决策控制综合算法流程图

车辆综合控制算法如下所示（在"decision\_control.py"中）：

```
# 运动控制算法
def motion_control(self,car_x,car_th):
    # 键盘控制
    if self.mile != 0.0:
        ...
    # 判断是否键盘控制引起的停车
    if self.mainStop:
        car_x = 0
        car_th = 0
        #print "main stop"

    else:

        # 激光雷达障碍物检测控制
        if self.obstacleStatus:
            car_x = 0
            car_th = 0
            print "find obstacle!",self.obstacleStatus

        else:
            # 目标检测控制
            if self.detectStatus == ObjInfoType.RED or
self.detectStatus == ObjInfoType.YELLOW or self.detectStatus ==
ObjInfoType.STOP:
                # 发现红灯、黄灯和停车牌，停车
                ...
            elif (self.detectStatus == ObjInfoType.RIGHT or
self.turn_flag) and self.branchStatus:
                # 右转为岔路口时，原地向右旋转 90 度
                ...
            else:
                #路径规划
                if self.stopStatus:
                    ...
                elif self.turnStatus:
                    ...
                ...
            ...
        ...
    ...
}
```

```

    else:
        # 车道线巡线 PID 控制

        car_x = self.forwardSpeed
        car_th = self.pid_p * self.lineErr + self.pid_d *
(self.lineErr - self.lastErr)
        self.lastErr = self.lineErr

    return car_x,car_th

```

其算法流程为：首先判断是否有键盘输入小车行驶里程和速度，若有，执行键盘控制处理程序；若无，则不执行键盘控制处理程序，直接判断是否检测到障碍物，若有，小车停止运动；若无，则判断是否检测到目标，若是，则根据检测到的不同目标执行相应控制；若否，则判断是否用户发布全局路径规划指令，若是，执行全局路径规划控制；若否，执行车道线巡线控制。基于以上思路，可将上述功能较好的融合在一起，实现车辆的多功能控制。

## 7 电机与编码器

### 7.1 电机控制

首先，用户根据需求给出小车运动的线速度和角速度大小，在程序中将其封装为一条消息并发布，与其对应的订阅器收到消息后，将线速度和角速度处理转化为左电机和右电机的速度期望值，并根据编码器返回的速度当前值与期望值的关系，得到当前时刻的速度控制值，通过串口发送到 STM32 单片机上，实时控制电机的运转，从而实现小车运动时线速度和角速度的改变。

下面，以指定小车直行距离程序为例，具体介绍小车电机控制的代码实现过程。

#### 7.1.1 主机处理程序

##### 7.1.1.1 线速度与角速度的设定

在 `2_length_control.launch` 文件中，“`car_x`”和“`car_theta`”分别为小车线速度与角速度的设置属性。

```
<param name="car_x" type="double" value="0.05"/>
```

```

<!--设定线速度为 0.05m/s-->
<param name="car_theta"    type="double"    value="0"/>
<!--设定角速度为 0rad/s-->

```

### 7.1.1.2 线速度与角速度的发布

在 "length\_control.py" 文件中，读取了 launch 文件中设置的参数值并将其封装为一条消息发布。

```

class Arduino_Car(ArduinoROS):
    ...
    @note: 只给出该类中有关电机控制的程序
    ...

    def __init__(self): #初始化程序
        super(Arduino_Car, self).__init__()

        print '[1] start init arduion_car .'

        # 初始化一个 Twist 消息
        vel_x_param = rospy.get_param("vel_x", 0.2)
        # 设定一个线速度值, 0.2 为默认值
        anv_z_param = rospy.get_param("anv_z", 0)
        # 设定一个角速度值, 0 为默认值
        self.cmd_vel.linear.x = vel_x_param
        # 将线速度默认值赋给 cmd_vel.linear.x
        self.cmd_vel.angular.z = anv_z_param
        # 将角速度默认值赋给 cmd_vel.angular.z

    def send_vel_cmd(self, vel_x, anv_z):
        # 向小车底层发布线速度 vel_x 和转向角速度 anv_z
        self.cmd_vel.linear.x = vel_x
        # 将线速度 vel_x 值赋给 cmd_vel.linear.x
        self.cmd_vel.angular.z = anv_z
        # 将角速度值 anv_z 赋给 cmd_vel.angular.z
        self.cmd_vel_pub.publish(self.cmd_vel)
        # 发布线速度和角速度

        self.RunFunctionForArduino()
    ...

if __name__ == '__main__':
    try:
        ...
        car_x = rospy.get_param("~car_x", 0.0)
        # 读取 Launch 文件中设置的线速度值
        car_th = rospy.get_param("~car_theta", 0.0)
        # 读取 Launch 文件中设置的角速度值
        ...
        while not rospy.is_shutdown():

            myArduino.send_vel_cmd(car_x, car_th) #发布线速度和角速度
        ...
    except SerialException:

```

```
rospy.logerr("Serial exception trying to open port.")
os._exit(0)
```

其中，涉及并应用了在如下路径所包含的 `arduino_node.py` 文件中的部分函数。

路径: `ininCar\src\ros_arduino_bridge\ros_arduino_python\nodes`

```
class ArduinoROS(object):
    ...
    @note: 只给出该类中有关电机控制的程序
    ...
    # 初始化一个 Twist 消息
    self.cmd_vel = Twist()

    # 建立消息发布器并发布 Twist 类型的消息
    self.cmd_vel_pub = rospy.Publisher('cmd_vel', Twist, queue_size=5)
```

### 7.1.1.3 线速度与角速度的接收与处理

该功能的实现应用了在如下路径所包含的 `base_controller.py` 文件中的部分函数。路径:

`ininCar\src\ros_arduino_bridge\ros_arduino_python\src\ ros_arduino_python`

```
class BaseController:
    ...
    @note: 只给出该类中有关电机控制的程序
    ...
    def __init__(self, arduino, base_frame, name="base_controllers"): # 初始化
        ...
        now = rospy.Time.now() # 初始化时间
        self.then = now # 确定当前采样时间
        self.t_delta = rospy.Duration(1.0 / self.rate) # 设置采样时间间隔
        self.t_next = now + self.t_delta # 确定下次采样时间

        # 内部数据
        self.v_left = 0 # 左电机控制值
        self.v_right = 0 # 右电机控制值
        self.v_des_left = 0 # 左电机期望值
        self.v_des_right = 0 # 右电机期望值
        self.last_cmd_vel = now # 控制时刻

        # 订阅接收 Twist 消息
        rospy.Subscriber("cmd_vel", Twist, self.cmdVelCallback)
        ...
    def poll(self):
        now = rospy.Time.now() # 确定当前采样时间
        ...
        if now > (self.last_cmd_vel + rospy.Duration(self.timeout)):
            # 超时则将电机期望值设为零
            self.v_des_left = 0
            self.v_des_right = 0
```

```

        if self.v_left < self.v_des_left:
            #左电机控制值小于期望值
            self.v_left += self.max_accel
            #令左电机控制值以最大加速度加速
            if self.v_left > self.v_des_left:
                #加速后左电机控制值大于期望值
                self.v_left = self.v_des_left
                #令左电机控制值等于期望值
            else: #左电机控制值大于期望值
                self.v_left -= self.max_accel
                #令左电机控制值以最大加速度减速
                if self.v_left < self.v_des_left:
                    #减速后左电机控制值小于期望值
                    self.v_left = self.v_des_left
                    #令左电机控制值等于期望值
            #右电机控制值处理方法同左电机
            if self.v_right < self.v_des_right:
                self.v_right += self.max_accel
                if self.v_right > self.v_des_right:
                    self.v_right = self.v_des_right
            else:
                self.v_right -= self.max_accel
                if self.v_right < self.v_des_right:
                    self.v_right = self.v_des_right

        #未有停车指令，将电机控制值传送给 STM32
        if not self.stopped:
            selfarduino.drive(self.v_left, self.v_right)

        self.t_next = now + self.t_delta #确定下次采样时间

    def stop(self): #停车控制指令
        self.stopped = True
        selfarduino.stop()
        print("stopping the robot")
        #selfarduino.drive(0, 0)

    def cmdVelCallback(self, req): #将用户设定速度转化为左右轮的期望速度

        self.last_cmd_vel = rospy.Time.now() #初始化当前时间

        x = req.linear.x          # 线速度值, 单位 m/s
        th = req.angular.z        # 角速度值, 单位 rad/s

        if x == 0: #如果线速度为零
            # 只考虑转向, 计算得到对应左右轮的速度期望值
            right = th * self.wheel_track * self.gear_reduction / 2.0
            left = -right
        elif th == 0: #如果角速度为零
            # 只考虑前进/后退, 计算得到对应左右轮的速度期望值
            left = right = x
        else: #线速度和角速度都不为零
            # 考虑二维平面运动, 计算得到对应左右轮的速度期望值
            left = x - th * self.wheel_track * self.gear_reduction / 2.0

```

```

        right = x + th * self.wheel_track * self.gear_reduction / 2.0
#将左右轮速度期望值对应转换为左右电机编码器的期望值
    self.v_des_left =
int(left*self.ticks_per_meter/selfarduino.PID_RATE)
    self.v_des_right =
int(right*self.ticks_per_meter/selfarduino.PID_RATE)

```

## 7.2 串口通信

### 7.2.1 速度通信

将主机处理得到的左右电机的控制值通过串口发送到 STM32 单片机，实现对小车的运动控制。其中，应用了在如下路径所包含的 `arduino_driver.py` 文件中的部分函数。路径：

`in inCar\src\ros_arduino_bridge\ros_arduino_python\src\ros_arduino_python`

```

class Arduino:
    ...
    @note: 只给出该类中有关电机控制的程序
    ...
    def __init__(self, port="/dev/ttyUSB0", baudrate=57600, timeout=0.5,
motors_reversed=False):
        ... # 初始化串口配置

    def connect(self):
        ... # 建立串口连接

    def open(self):
        ... # 打开串口

    def close(self):
        ... # 关闭串口

    def send(self, cmd):
        ... # 串口写入 cmd
        ...
    def execute_ack(self, cmd):
        ... # 线程安全发送数据，发送成功返回 OK
        ...
    def drive(self, right, left):
        # 发送左右电机控制值
        if self.motors_reversed:
            left, right = -left, -right
        # 发送格式为"m right left"
        return self.execute_ack('m %d %d' %(right, left))
    ...
    def stop(self):
        ... # 停车时左右电机控制值的发送，格式为"m 0 0"

```

## 7.2.2 编码器通信

### 7.2.2.1 通信协议

小车中 NVIDIA 主机与 STM32 下位机之间存在有关编码器信息的通信，主要包括获取编码器计数累积值和复位编码器计数值两部分。

#### a. 获取编码器计数累积值

为获得编码器累积计数值，规定首先 NVIDIA 主机向端口/dev/deepcar 写入字符'e'，然后开始等待接收 STM32 下位机的返回的字符，从第一个非'\r'字符开始接收，直到再次接收到'\r'停止接收。得到的字符串分为两段，中间用' '字符隔开，两段字符分别表示左编码器和右编码器的计数累积值，字符的长度由实际的编码器计数累计值确定，没有固定限制。

例如：NVIDIA 主机发送完字符'e'之后，接收到字符串'1000 1500'，表示左编码器和右编码器的计数累计值分别为正转 1000 和正转 1500；

又如：NVIDIA 主机发送完字符'e'之后，接收到字符串'-500 700'，表示左编码器和右编码器的计数累计值分别为反转 500 和正转 700。

注：得到的编码器计数累计值可用于后续数值操作。

#### b. 复位编码器计数值

用于初始化时对编码器计数值的清空，规定 NVIDIA 主机向端口/dev/deepcar 写入字符'r'代表要清空编码器的计数值，STM32 下位机无需返回数值。

### 7.2.2.2 通信程序

此部分程序主要为存放在位于如下路径所包含的 "[Aduino\\_driver.py](#)" 文件中。

路径：[ininCar\src\ros\\_arduino\\_bridge\ros\\_arduino\\_python\src](#)

主要使用到了类 [Arduino](#) 的一些函数，相关函数结构如下。

```
class Arduino:  
    ...  
  
    @note: 只给出了该类中有关编码器通信的程序  
    ...  
  
    def __init__(self, port, baudrate=115200, timeout=0.5,  
motors_reversed=False):  
        ...# 初始化程序
```

```

def connect(self):
    ...# 建立Arduino 与 STM32 的连接

def open(self):
    ...# 打开端口

def close(self):
    ...# 关闭端口

def send(self, cmd):
    ...# 向端口写入命令或数据
    # ‘cmd’：待写入的命令或数据

def recv(self, timeout=0.5):
    ...# 从端口读取一串字符
    # ‘timeout’：最多等待时间，不可超时
    # 返回读取到的字符串

def recv_array(self):
    ...# 从端口读取一串字符，并转换为需要的整形列表，会调用recv(self,
    timeout=0.5)
    # 返回转换之后的整形列表

def execute_array(self, cmd):
    ...# 在Arduino 上对返回数据的命令或数据进行线程安全执行，会调用
    recv_array(self)
    # ‘cmd’：待写入的命令或数据
    # 返回线程安全执行的转换之后的整形列表

def execute_ack(self, cmd):
    ...# 在Arduino 上对无需返回数据的命令或数据进行线程安全执行，会调用
    recv(self, timeout=0.5)
    # ‘cmd’：待写入的命令或数据
    # 返回确认字符‘OK’

def get_encoder_counts(self):
    ...# 获取编码器的计数累计值，调用execute_array(self, cmd)，‘cmd’为‘e’
    # 返回一个长度为 2 整形列表，数据分别为左右编码器的计数累计值

def reset_encoders(self):
    ...# 复位编码器，会调用execute_ack(self, cmd)，‘cmd’为‘r’

```

```
# 返回确认字符'OK'
```

## 7.3 编码器数值操作

获取的编码器计数累计值可应用于绝对坐标系下小车位置、姿态、速度和角速度的计算。具体程序位于如下路径所包含的 "**base\_controller.py**" 文件中。

路径: **iniinCar\src\ros\_arduino\_bridge\ros\_arduino\_python\src**

在文件的主要类 **BaseController** 中, 编码器部分主要使用到了两个函数, 分别是初始化函数和数值操作函数。下面对这两个函数进行具体介绍。

### 7.3.1 初始化函数

初始化函数为

```
__init__(self, arduino, base_frame, name="base_controllers")
```

首先需要从配置文件 **/iniinCar/config/1\_keyboard\_control\_car.yaml** 中获得以下参数值

```
pid_params['wheel_diameter'] = rospy.get_param("~wheel_diameter", "")  
#车轮直径  
pid_params['wheel_track'] = rospy.get_param("~wheel_track", "")  
#车轮轮距  
pid_params['encoder_resolution'] =  
rospy.get_param("~encoder_resolution", "") #编码器分辨率  
pid_params['gear_reduction'] = rospy.get_param("~gear_reduction", 1.0)  
#齿轮减速比
```

接着需要利用公式计算小车行驶每米有多少个编码器脉冲值, 如下所示。

```
self.ticks_per_meter = self.encoder_resolution * self.gear_reduction /  
(self.wheel_diameter * pi) #每米有多少个编码器脉冲 = 编码器分辨率 * 齿轮减速比 / (车轮直径 * pi)
```

有关采样时间的确定如下:

```
now = rospy.Time.now() #初始化时间  
self.then = now #确定 dx/dy 的时间  
self.t_delta = rospy.Duration(1.0 / self.rate) #采样时间间隔  
self.t_next = now + self.t_delta #下一次采样的时间
```

除此之外, 还需要清空旧里程表信息, 并建立里程计和编码器的发布器。

```
self.arduino.reset_encoders() #清楚所有旧的里程表信息
```

```

self.odomPub = rospy.Publisher('odom', Odometry, queue_size=5)
#设置一个里程计的发布器#odom 是主题, Odometry 是消息的类, queue_size=5 代表发布队列大小为5
self.encoderPub = rospy.Publisher('encoder', Encoder)
#设置一个编码器的发布器
#encoder 是主题, Encoder 是消息的类

```

### 7.3.2 数值操作函数

数值操作函数 `poll(self)` 主要对在 8.2 小节中从 `STM32` 获取的左右编码器计数累积值进行操作。

首先判断是否达到采样时间，到达采样时间则获取编码器的计数累积值：

```

left_enc, right_enc = selfarduino.get_encoder_counts()
#获得左右编码器的值存放在 left_enc, right_enc

```

接下来可以根据这两个值来计算小车的位置、姿态、速度和角速度。

```

dright=(right_enc-self.enc_right)/self.ticks_per_meter #查看右轮采样间隔前进的距离(后退为负)
yleft=(left_enc-self.enc_left)/self.ticks_per_meter #查看左轮采样间隔内前进的距离(后退为负)

dxy_ave = (dright + yleft) / 2.0 #前进距离为左右轮平均前进距离
dth = (dright - yleft) / self.wheel_track
#角度变化为(右轮前进距离-左轮前进距离)/轮距(理论上pi为360°, 实际情况可实际测试)
vxy = dxy_ave / dt #前进速度
vth = dth / dt #角速度

if (dxy_ave != 0): #位置更新
    dx = cos(dth) * dxy_ave #在x方向(当前面对的方向)行驶的距离
    dy = -sin(dth) * dxy_ave #在y方向(当前面对的方向的右手边)行驶的距离
    self.x += (cos(self.th) * dx - sin(self.th) * dy) #xy 绝对位置更新,由几何关系得到
    self.y += (sin(self.th) * dx + cos(self.th) * dy)

if (dth != 0): #角度更新
    self.th += dth

```

## 8 摄像头

### 8.1 车载相机介绍

无人小车 deepcar 搭载的摄像头为双目相机。双目相机和普通 usb 摄像头的区别是,普通 usb 摄像头只能输出 RGB 图,即我们平常见到的彩色图像;而双目相机除了输出彩色图像外,还会同步输出深度图,如下图所示。



图 8-1 双目相机彩色图和深度图

上图中右边是我们常见的彩色图像,图像中的每个像素代表摄像头采集的彩色信息,左边是深度图,图像中的每个像素值代表的是图像拍摄到的物体与摄像头的距离。通过使用双目相机,我们既可以使用 rgb 图做图像处理,用来判断目标识别、行人检测等人工智能任务,同时我们还能通过深度图得到图像中各物体距离 deepcar 的距离,从而在做自动驾驶时可以更加智能和方便。

deepcar 使用的双目深度相机是乐视三合一体感摄像头 LeTMC-520,它可以在 30fps 的帧率下同步输出 1280\*720 大小的彩色图和深度图,深度距离的测量范围是 0.6m-4m。相机的实物如下图所示:



图 8-2 相机实物图

无人小车 deepcar 所装配摄像头的具体信息如下:

名称	乐视三合一体感摄像头
型号	LeTMC-520
尺寸 (长×宽×高)	19.5*8.5*5.5cm

重量	0.6 千克
功能	可以输出彩色图和深度图
深度距离的测量范围	0.6-4m
延迟	30-45ms
RGB	1080P
数据传输接口	USB2.0

## 8.2 相机读取图像

读取相机图像主要使用 ROS 的发布消息和订阅消息功能，即 `rospy.Publisher` 和 `rospy.Subscriber`。使用 ROS 读取相机数据主要分为几个步骤：（1）打开相机；（2）ros 订阅相机发出的消息并显示。下面对这两个步骤进行详细讲解。

### 8.2.1 打开相机

打开一个终端，在终端输入如下指令，打开相机：

```
cd catkin_ws/
source devel/setup.bash
roslaunch astra_camera astrapro.launch
```

解释：打开相机后，相机会不断向外发布消息，其中就包括采集到的图像消息，下面需要接收相机发出的图像消息，并显示。

### 8.2.2 ros 订阅相机发出的消息并显示图像

**注意保证相机已经启动**，即不要关闭 9.2.1 中打开的终端或者停止终端中的指令。显示图像的方法是通过程序接收相机发出的图像消息，并进行显示。

程序名称为 `xiangji.py`，该程序的运行方法如下：

在该程序所在目录打开一个新的终端，同时保证相机已经启动，运行如下命令即可启动程序：

```
python xiangji.py
```

具体程序如下：

```

# -*- coding: utf-8 -*-
import cv2
import rospy
from sensor_msgs.msg import Image
from cv_bridge import CvBridge, CvBridgeError


def callback(data):
    # 主循环
    bridge = CvBridge()
    # 获取原始图像
    frame = bridge.imgmsg_to_cv2(data, "bgr8")
    # 显示图像
    cv2.imshow('frame', frame)
    cv2.waitKey(1)

result=rospy.Subscriber("/camera/rgb/image_raw", Image,callback)

if __name__ == '__main__':
    try:
        # 初始化节点，便于接收相机的 topic
        rospy.init_node('astra', anonymous=True)
        r = rospy.Rate(25)
        while not rospy.is_shutdown():
            r.sleep()

    except SerialException:
        rospy.logerr("Serial exception trying to open port.")
        os._exit(0)

cv2.destroyAllWindows()

```

### 对程序的解释：

ros 订阅相机发出的消息主要通过 ros 的消息订阅器来完成。具体语句如下：

```
result=rospy.Subscriber("/camera/rgb/image_raw", Image,callback)
```

该语句中使用了 ros 的 Subscriber 函数，该函数有如下参数：

"/camera/rgb/image\_raw"：指所接收到的 rostopic 的名称，这里是指相机发出的消息的名称。

**Image:** 是 Subscriber 实际接收到的消息变量。

**callback :** 指接收到消息后，执行的回调函数的名称。Subscriber 每接受到一次消息，会触发回调函数 callback() 的执行，图像的显示就是在 callback() 中实现的。

显示结果：

程序运行后，接收到相机发出的图像，显示的图像如下图所示：

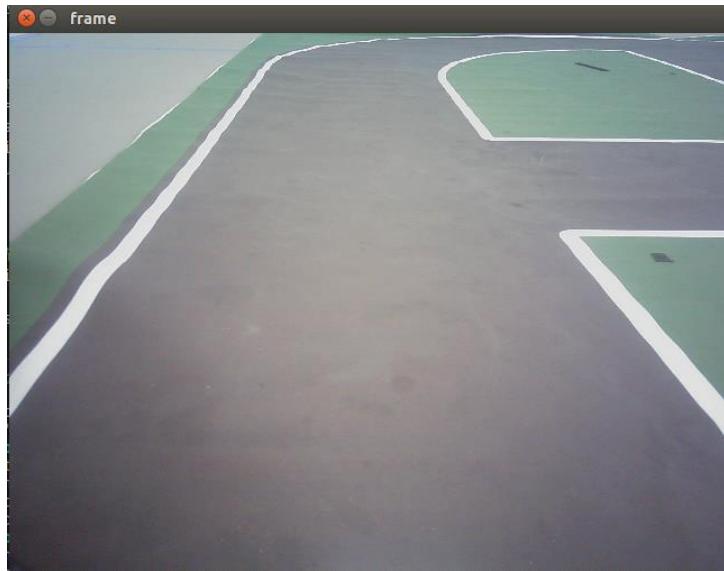


图 8-3 相机显示图

## 9 激光雷达

本项目所选用的雷达是 RPLIDAR A1 360 度激光扫描测距雷达，它由 SLAMTEC 公司开发，采用了激光三角测距技术，配合 SLAMTEC 研发的高速的视觉采集处理机构，可进行每秒高达 8000 次以上的测距动作。可以实现在二维平面的 12 米半径范围内进行 360 度全方位的激光测距扫描，并产生所在空间的平面点云地图信息。

如下图所示，在每次测距过程中，RPLIDAR A1 发射调制过的红外激光信号，激光信号在照射到目标物体后的反光再被 RPLIDAR A1 的视觉采集系统接受。由于 RPLIDAR A1 内部具有 DSP 处理器，每次测距目标物体与 RPLIDAR A1 之间的距离值和夹角值可以实时测算出来并输出。

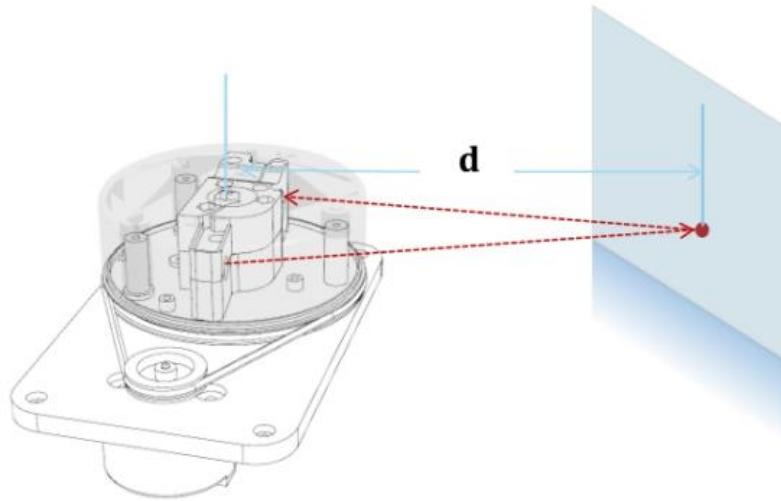


图 9-1 单线激光雷达结构图

电机机构驱动 RPLIDAR A1 测距核心进行顺时针旋转，从而可以实现 360 度全方位的激光测距扫描。

## 9.1 雷达数据读取

### 9.1.1 程序运行

启动终端，运行如下命令，程序运行界面如下图所示：

```
cd ininCar/
source devel/setup.bash
cd launch/
roslaunch 4_start_Lidar.launch
```

```
NODES
/
rplidarNode (rplidar_ros/rplidarNode)
rviz (rviz/rviz)

auto-starting new master
process[master]: started with pid [14331]
ROS_MASTER_URI=http://192.168.1.102:11311

setting /run id to d2b27afa-242b-11eb-a2de-00044be7550f
WARNING: Package name "detectionInfo_msgs" does not follow the naming convention
s. It should start with a lower case letter and only contain lower case letters,
| digits, underscores, and dashes.
process[rosout-1]: started with pid [14343]
started core service [/rosout]
process[rplidarNode-2]: started with pid [14350]
process[rviz-3]: started with pid [14355]
RPLIDAR running on ROS package rplidar_ros
SDK Version: 1.5.7
RPLIDAR S/N: 80989A84C7EA9CD4A2EB93F9154B3E68
Firmware Ver: 1.28
Hardware Rev: 5
RPLidar health status : 0
```

图 9-2 雷达启动终端界面

注：当 `roslaunch 4_start_Lidar.launch` 启动程序后，激光雷达开始旋转并采集数据，与此同时，rviz 打开，能够看到采集好的雷达点云图像。如下图

所示。

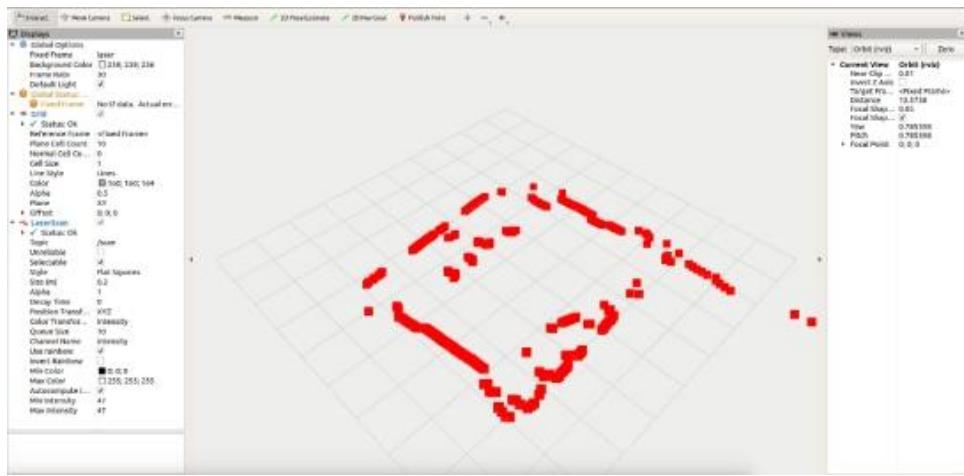


图 9-3 雷达数据显示

通过在某一方向增加障碍物，使雷达点云图像发生突变，可以辨别空间方向。以下是 `4_start_Lidar.launch` 文件中的内容：

```
<!-- 程序4. 激光雷达采集点云图像程序。
@init: 初始化设置接入端口为 /dev/rplidar，波特率为 115200，id 为 laser，转变
情况为 false，角度补偿情况为 true
@note: 参数文件为 rplidar.rviz
-->

<launch>
    <node name="rplidarNode" pkg="rplidar_ros" type="rplidarNode"
output="screen">
        <param name="serial_port" type="string"
value="/dev/rplidar"/>
        <param name="serial_baudrate" type="int" value="115200"/>
        <param name="frame_id" type="string" value="laser"/>
        <param name="inverted" type="bool" value="false"/>
        <param name="angle_compensate" type="bool" value="true"/>
    </node>
    <node name="rviz" pkg="rviz" type="rviz" args="-d rplidar.rviz"/>
</launch>
```

注：该部分主要完成的任务是用 launch 文件打开 rviz 文件。从初始设置可知，功能包为 `rplidar_ros`，将 `4_start_Lidar.launch` 文件保存在 `rplidar_ros` 下的文件夹下，将当前 rviz 的配置文件 `rplidar.rviz` 保存在自定义功能包 `rplidar_ros` 下。保存好后，在终端运行 launch 文件即可。

方便的是，这里打开 rviz 不需要修改 `name`, `pkg`, `type`，只要确定下配置文件及路径，也就是确定下文件中 `args` 即可。

## 9.1.2 节点主程序 "note.cpp"

"note.cpp" 位于 ininCar\src\rplidar\_ros\src 路径底下。

```
#include "ros/ros.h"
#include "sensor_msgs/LaserScan.h"
#include "std_srvs/Empty.h"
#include "rplidar.h"

//定义数组长度
#ifndef _countof
#define _countof(_Array) (int)(sizeof(_Array) / sizeof(_Array[0]))
#endif

//将角度制转化为弧度制
#define DEG2RAD(x) ((x)*M_PI/180.)

using namespace rp::standalone::rplidar;

RPlidarDriver * drv = NULL;

void publish_scan(ros::Publisher *pub,
                  rplidar_response_measurement_node_t *nodes,
                  size_t node_count, ros::Time start,
                  double scan_time, bool inverted,
                  float angle_min, float angle_max,
                  std::string frame_id)
{
    ... //数据处理并发布。
    //将获得的angle_max 和angle_min 进行角度统一，所得到的
    scan_msg.angle_min 必小于scan_msg.angle_max;
    //计算每通过一个点所跨过的角度值scan_msg.angle_increment,
    //扫描一周时间scan_msg.scan_time,
    //每扫描一个点所需时间scan_msg.time_increment,
    //扫描点距离原点的距离大小scan_msg.ranges
    //描述信号质量强度的参数scan_msg.intensities。
}

bool getRPLIDARDeviceInfo(RPlidarDriver * drv)
{
    ... //如果返回值正常，输出函数的模型、固件版本、硬件版本、序列号等信息；否则，输出错误信息。
}
bool checkRPLIDARHealth(RPlidarDriver * drv)
{
```

```

    ... //返回运行状况是否健康信息
}

bool stop_motor(std_srvs::Empty::Request &req,
                std_srvs::Empty::Response &res)
{
    ... //停止电机函数
}

bool start_motor(std_srvs::Empty::Request &req,
                 std_srvs::Empty::Response &res)
{
    ... //开启电机函数
}

int main(int argc, char * argv[])
{
    ros::init(argc, argv, "rplidar_node"); //初始化节点名称

    std::string serial_port; //定义串口名字
    int serial_baudrate = 115200; //定义波特率
    std::string frame_id; //id
    bool inverted = false; //转动状态
    bool angle_compensate = true; //角度补偿状态

    ros::NodeHandle nh; //声明节点句柄
    ros::Publisher scan_pub =
    nh.advertise<sensor_msgs::LaserScan>("scan", 1000); //定义消息发送者的节点
    //名字和类型
    ros::NodeHandle nh_private("~");
    //初始化串口/波特率/id/转变状态/角度补偿状态
    nh_private.param<std::string>("serial_port", serial_port,
    "/dev/ttyUSB0");
    nh_private.param<int>("serial_baudrate", serial_baudrate, 115200);
    nh_private.param<std::string>("frame_id", frame_id, "laser_frame");
    nh_private.param<bool>("inverted", inverted, false);
    nh_private.param<bool>("angle_compensate", angle_compensate, true);

    printf("RPLIDAR running on ROS package rplidar_ros\n"
           "SDK Version: " RPLIDAR_SDK_VERSION "\n");

    u_result      op_result;//返回结果
}

```

```

// create the driver instance, 创建串行端口
drv =
RPlidarDriver::CreateDriver(RPlidarDriver::DRIVER_TYPE_SERIALPORT);

//如果未能连上，输出错误信息，返回-2
if (!drv) {
    fprintf(stderr, "Create Driver fail, exit\n");//标准错误
    return -2;
}

//如果串行端口连接不上，输出错误信息，并删除该端口，返回-1
if (IS_FAIL(drv->connect(serial_port.c_str(),
(_u32)serial_baudrate))) {
    fprintf(stderr, "Error, cannot bind to the specified serial
port %s.\n"
        , serial_port.c_str());
    RPlidarDriver::DisposeDriver(drv);
    return -1;
}

// get rplidar device info, 如果未能成功获取信息，返回-1
if (!getRPLIDARDeviceInfo(drv)) {
    return -1;
}

//如果端口运行状况不正常，则删除该端口，返回-1
if (!checkRPLIDARHealth(drv)) {
    RPlidarDriver::DisposeDriver(drv);
    return -1;
}

//回调函数
//声明底层服务器分别为 stop_motor_service，服务对象名称为 stop_motor，且
当有服务请求时，执行 stop_motor 函数
ros::ServiceServer stop_motor_service =
nh.advertiseService("stop_motor", stop_motor);
//同上
ros::ServiceServer start_motor_service =
nh.advertiseService("start_motor", start_motor);

drv->startMotor();//打开电机
drv->startScan();//开始扫描

```

```

ros::Time start_scan_time;//开始扫描时间
ros::Time end_scan_time;//结束扫描时间
double scan_duration;//持续时间
while (ros::ok()) {

    rplidar_response_measurement_node_t nodes[360*2];//储存信号质量,
朝向角度, 距离信息。
    size_t count = _countof(nodes);//统计已有的节点数

    start_scan_time = ros::Time::now();//扫描开始时间
    op_result = drv->grabScanData(nodes, count); //根据节点数给出运行结
果, count≠0, 返回结果RESULT_OK; 否则, 返回RESULT_OPERATION_FAIL
    end_scan_time = ros::Time::now();//扫描结束时间
    scan_duration = (end_scan_time - start_scan_time).toSec() * 1e-
3;//时间间隔转化成ms 级

    if (op_result == RESULT_OK) {//扫描成功
        op_result = drv->ascendScanData(nodes, count); //根据角度值进
行排序, 得到排序后的雷达数据

        float angle_min = DEG2RAD(0.0f);
        float angle_max = DEG2RAD(359.0f);
        if (op_result == RESULT_OK) {
            if (angle_compensate) {//角度补偿, 调整角度表示以符合scan 话
题消息的格式 (角度值按照固定角间隔递增排列)。经过补偿, rplidar 的扫描点角度相
当于全部转换为正值, 且按照角度值大小递增的顺序填充入扫描点集合, 该集合的范围是
[0,359]度, 步长是1 度。
                const int angle_compensate_nodes_count = 360;
                const int angle_compensate_multiple = 1;
                int angle_compensate_offset = 0;
                rplidar_response_measurement_node_t
angle_compensate_nodes[angle_compensate_nodes_count];
                memset(angle_compensate_nodes, 0,
angle_compensate_nodes_count*sizeof(rplidar_response_measurement_node_t)
);//初始化
                int i = 0, j = 0;
                for( ; i < count; i++ ) {
                    if (nodes[i].distance_q2 != 0) {
                        float angle =
(float)((nodes[i].angle_q6_checkbit >>

```

```

RPLIDAR_RESP_MEASUREMENT_ANGLE_SHIFT)/64.0f); //与 lidar 的朝向夹角处理成实际角度
    int angle_value = (int)(angle *
angle_compensate_multiple); //将获得的角度整数化
    if ((angle_value - angle_compensate_offset) <
0) angle_compensate_offset = angle_value;
    //计算当前角度, 如果小于前一值, 就把它更新记录下来
    for (j = 0; j < angle_compensate_multiple;
j++) {
        angle_compensate_nodes[angle_value-
angle_compensate_offset+j] = nodes[i];
        } //对角度值进行修正
    }
}

publish_scan(&scan_pub, angle_compensate_nodes,
angle_compensate_nodes_count,
            start_scan_time, scan_duration, inverted,
            angle_min, angle_max,
            frame_id);
} else { //不进行角度补偿, 则默认第一个有效节点对应角度值最小, 最后一个有效节点对应角度值最大
    int start_node = 0, end_node = 0;
    int i = 0;
    // 找到第一个有效节点和最后一个有效节点
    while (nodes[i++].distance_q2 == 0);
    start_node = i-1;
    i = count -1;
    while (nodes[i--].distance_q2 == 0);
    end_node = i+1;
    //首、尾节点分别对应角度最小、大值
    angle_min =
DEG2RAD((float)(nodes[start_node].angle_q6_checkbit >>
RPLIDAR_RESP_MEASUREMENT_ANGLE_SHIFT)/64.0f);
    angle_max =
DEG2RAD((float)(nodes[end_node].angle_q6_checkbit >>
RPLIDAR_RESP_MEASUREMENT_ANGLE_SHIFT)/64.0f);

publish_scan(&scan_pub, &nodes[start_node], end_node-
start_node +1,
            start_scan_time, scan_duration, inverted,
            angle_min, angle_max,

```

```
        frame_id);

    }

} else if (op_result == RESULT_OPERATION_FAIL) {
    // All the data is invalid, just publish them
    float angle_min = DEG2RAD(0.0f);
    float angle_max = DEG2RAD(359.0f);

    publish_scan(&scan_pub, nodes, count,
                 start_scan_time, scan_duration, inverted,
                 angle_min, angle_max,
                 frame_id);
}

}

ros::spinOnce(); //等待服务请求
}

//扫描完成
drv->stop();
drv->stopMotor();
RPlidarDriver::DisposeDriver(drv);
return 0;
}
```

注：雷达点云模块主要是通过 `note.cpp` 发布 `/scan` 这个话题，主要工作就是发布雷达扫描获得的点云数据，然后通过 `client.cpp` 去订阅。

## 9.2 雷达障碍物检测

激光雷达障碍物检测模块实现的功能为用雷达探测小车周围障碍物。当检测到障碍物距离小车小于一定距离时，发布消息，使小车停止。本身可以订阅到的雷达数据类型为 `LaserScan`，当需要对雷达数据进行处理时，需要将它转化为 pcl 点云库可以使用的数据，如 `pcl::PointCloud<T>`。具体的转换方法和原理如下图所示。



图 9-4 点云数据消息类型转换

首先我们需要订阅激光雷达 topic (如`/scan`) 获取到 `sensor_msgs::LaserScan` 的 message。然后使用 ROS 提供的 `laser_geometry` 包将其转化为 `sensor_msgs::PointCloud2` 格式的 message. 接着将 `sensor_msgs::PointCloud2` 的 message 转换为 PCL 点云库所需的数据格式. 有两种方法，一种方法是使用 ROS 提供的 `pcl_conversions` 包，另一种方法是直接订阅之前转化的 `PointCloud2` 的数据，这样可以自动完成两种类型的转换，需要注意的是这种方法需要引入 `pcl_ros/point_cloud.h`.

对该数据进行一个每一帧的扫描，每段扫描时间均可获得一周数据，每个数据点的值可以用一个  $3 \times 1$  的向量表示。根据每个点的 xyz 数据，我们对小车附近一段区域进行限制，当探测到的点距离小车的值小于我们所规定的阈值时，我们再发布一个消息，即下发障碍物标志位令 `flag=1`。由小车控制器接收，并对小车做出命令，比如停下。以上操作均建立在激光雷达已经打开的情况下，具体打开雷达，观察 `rviz` 的过程如上图 10-3 所示。

本模块主要操作为新建立了一个包 `my_pcl`，并新建立了一个节点 `mypcl_node`。通过该节点完成如上订阅 `/scan` 数据，扫描确定小车所在位置周边有无障碍物，以及发布 `ros_tutorials_msg` 消息的任务。

### 9.2.1 程序运行

启动终端，运行如下命令：

```
cd ininCar/  
source devel/setup.bash  
cd launch  
roslaunch 4_start_Lidar.launch
```

命令注解：使用 `roslaunch` 调用 `4_start_Lidar.launch` 文件，启动小车控制程序。以下为 `4_start_Lidar.launch` 中的内容：

```
<launch>  
  <node name="rplidarNode" pkg="rplidar_ros" type="rplidarNode"  
        output="screen">  
    <param name="serial_port" type="string"  
          value="/dev/rplidar"/>  
    <param name="serial_baudrate" type="int" value="115200"/>
```

```

<param name="frame_id" type="string" value="laser"/>
<param name="inverted" type="bool" value="false"/>
<param name="angle_compensate" type="bool" value="true"/>
</node>
<node name="mypcl_node" pkg="my_pcl" type="mypcl_node"
output="screen">
</node>
<node name="rviz" pkg="rviz" type="rviz" args="-d rplidar.rviz"/>
</launch>

```

注：当运行 `mypcl_node` 节点后，一方面点云数据从 `laserscan` 转化为 `pointcloud2`，并且数据以新的类型实时输出“XYZ=”，同时 `rviz` 也可以通过订阅 `/cloud2` 实时读取发布的 `pointcloud2` 类型的数据。当扫描范围内出现小于给定距离的点时，能够成功发布消息“`flag`”。理想情况是，当小车控制模块成功读取发布的标志位消息的第一时刻，小车能够做出反应，立刻停止。

如下图所示，运行程序可以实时输出每一刻的 XYZ 坐标值，并返回标志位 `flag` 的值；当 `flag=0` 时无障碍物；当 `flag=1` 时，有障碍物。

```

4_start_Lidar.launch http://localhost:11311
-3.12495 1.45719 0
-3.13072 1.39389 0
-2.19172 0.93033 0
-2.15292 0.869836 0
-2.17057 0.833205 0
-3.20435 1.16629 0
-3.21476 1.10693 0
-3.20506 1.04139 0
-3.23709 0.989677 0
-1.65722 0.475198 0
-1.64497 0.440768 0
-0.274859 0.0584231 0
-0.276819 0.0538081 0
-0.276553 0.0438016 0
-0.248557 0.0349324 0
-0.250122 0.030711 0
-0.249625 0.0262366 0
-0.249049 0.0217889 0
-0.253381 0.0177181 0
-0.254651 0.0133456 0
-0.256843 0.00896911 0
-0.295 -7.13966e-08 0
[ INFO] [1609044437.621105778]: flag: 0

```

```
[ INFO] [1609044459.227557385]: flag: 1  
XYZ: 0.165441 -0.0505803 0有障碍了!  
[ INFO] [1609044459.227682801]: flag: 1  
XYZ: 0.163415 -0.0468583 0有障碍了!  
[ INFO] [1609044459.227850926]: flag: 1  
XYZ: 0.160344 -0.0429639 0有障碍了!  
[ INFO] [1609044459.228062072]: flag: 1  
XYZ: 0.160099 -0.0399171 0有障碍了!  
[ INFO] [1609044459.228248322]: flag: 1  
XYZ: 0.157848 -0.036442 0有障碍了!  
[ INFO] [1609044459.228340978]: flag: 1  
XYZ: 0.15846 -0.0336816 0有障碍了!  
[ INFO] [1609044459.228473114]: flag: 1  
XYZ: 0.158042 -0.0307202 0有障碍了!  
[ INFO] [1609044459.228610353]: flag: 1  
XYZ: 0.152104 -0.0240909 0有障碍了!  
[ INFO] [1609044459.228740406]: flag: 1  
XYZ: 0.150521 -0.0211543 0有障碍了!  
[ INFO] [1609044459.228857593]: flag: 1  
XYZ: 0.149874 -0.0184022 0有障碍了!  
[ INFO] [1609044459.228980458]: flag: 1  
XYZ: 0.15441 -0.0135091 0有障碍了!  
[ INFO] [1609044459.229100197]: flag: 1
```

图 9-5 激光雷达检测障碍物

### 9.2.2 主程序 "my\_pcl\_node.cpp"

"my\_pcl\_node.cpp"位于 `inInCar\src\my_pcl\src\` 路径底下。

```
#include <iostream>  
#include <ros/ros.h>  
#include <tf/transform_listener.h>  
#include <laser_geometry/Laser_geometry.h>  
#include <sensor_msgs/PointCloud2.h>  
#include <sensor_msgs/LaserScan.h>  
#include <pcl_conversions/pcl_conversions.h>  
#include <pcl_ros/point_cloud.h>  
#include <pcl/point_cloud.h>  
#include <pcl/point_types.h>  
#include <pcl/io/pcd_io.h>  
#include <my_pcl/MsgTutorial.h>  
  
class My_Filter {  
public:  
    My_Filter();  
    //订阅LaserScan 数据，并发布PointCloud2 点云  
    void scanCallback(const sensor_msgs::LaserScan::ConstPtr&  
scan);  
        //订阅 LaserScan 数据，先转换为 PointCloud2，再转换为  
pcl::PointCloud  
        void scanCallback_2(const sensor_msgs::LaserScan::ConstPtr&  
scan);  
        //直接订阅 PointCloud2 然后自动转换为 pcl::PointCloud  
        void pclCloudCallback(const  
pcl::PointCloud<pcl::PointXYZ>::ConstPtr& cloud);  
private:
```

```

        ros::NodeHandle node_;
        laser_geometry::LaserProjection projector_;
        tf::TransformListener tfListener_;

        //发布 "PointCloud2"
        ros::Publisher point_cloud_publisher_;
        //订阅 "/scan"
        ros::Subscriber scan_sub_;
        //发布 "flag"
        ros::Publisher flag_publisher_;
        //订阅 "/cloud2" -> "PointCloud2"
        ros::Subscriber pclCloud_sub_;
    };

My_Filter::My_Filter(){
    //scan_sub_ = node_.subscribe<sensor_msgs::LaserScan> ("/scan",
    100, &My_Filter::scanCallback, this);
    //订阅 "/scan"
    scan_sub_ = node_.subscribe<sensor_msgs::LaserScan> ("/scan",
    100, &My_Filter::scanCallback_2, this);
    //发布 LaserScan 转换为 PointCloud2 的后的数据
    point_cloud_publisher_ =
    node_.advertise<sensor_msgs::PointCloud2> ("/cloud2", 100, false);
    //发布 flag 消息
    flag_publisher_ = node_.advertise<my_pcl::MsgTutorial>("/ros_tutorials_
msg",100);
    //此处的 tf 是 Laser_geometry 要用到的
    tfListener_.setExtrapolationLimit(ros::Duration(0.1));
    //前面提到的通过订阅 PointCloud2, 直接转化为 pcl::PointCloud 的方式
    pclCloud_sub_ =
    node_.subscribe<pcl::PointCloud<pcl::PointXYZ> >("/cloud2", 10,
    &My_Filter::pclCloudCallback, this);
}

void My_Filter::scanCallback(const sensor_msgs::LaserScan::ConstPtr&
scan){
    sensor_msgs::PointCloud2 cloud;
    projector_.transformLaserScanToPointCloud("laser", *scan, cloud,
    tfListener_);
    point_cloud_publisher_.publish(cloud);
}

void My_Filter::scanCallback_2(const sensor_msgs::LaserScan::ConstPtr&
scan){
    sensor_msgs::PointCloud2 cloud;
    //Laser_geometry 包中函数, 将 sensor_msgs::LaserScan 转换为
    sensor_msgs::PointCloud2
    //普通转换
    //projector_.projectLaser(*scan, cloud);
    //使用 tf 的转换
    projector_.transformLaserScanToPointCloud("laser", *scan, cloud,
    tfListener_);

    int row_step = cloud.row_step;
    int height = cloud.height;
}

```

```

//将 sensor_msgs::PointCloud2 转换为 pcl::PointCloud<T>
//注意要用fromROSMsg 函数需要引入 pcl_versions (见头文件定义)
pcl::PointCloud<pcl::PointXYZ> rawCloud;
pcl::fromROSMsg(cloud, rawCloud);

for(size_t i = 0; i < rawCloud.points.size(); i++){

std::cout<<rawCloud.points[i].x<<"\t"<<rawCloud.points[i].y<<"\t"<<rawCloud.points[i].z<<std::endl;
}
point_cloud_publisher_.publish(cloud);
}

void My_Filter::pclCloudCallback(const
pcl::PointCloud<pcl::PointXYZ>::ConstPtr& cloud){
my_pcl::MsgTutorial msg;
msg.flag=0;
ROS_INFO("flag: %d",msg.flag);
flag_publisher_.publish(msg);

//实时输出 XYZ 的值
for(size_t i = 0; i < cloud->points.size(); i++){
std::cout<<"XYZ:
"<<cloud->points[i].x<<"\t"<<cloud->points[i].y<<"\t"<<cloud->points[i].z<<std::endl;
//判断障碍物,具体阈值根据具体情况进行修改
if (cloud->points[i].x > 0 && cloud->points[i].x < 0.5 &&
cloud->points[i].y > -0.5 && cloud->points[i].y < 0.5) {
double range = sqrt(pow(cloud->points[i].x, 2.0) +
pow(cloud->points[i].y, 2.0));
if (range > 0 && range < 0.3) {
msg.flag=1;
ROS_INFO("flag: %d",msg.flag);
flag_publisher_.publish(msg);
}
}
}
int main(int argc, char** argv)
{
ros::init(argc, argv, "my_pcl_node");
My_Filter filter;
ros::spin();
return 0;
}

```

注: `my_pcl_node.cpp` 中设置扫描环境为  $0 < x < 0.5, -0.5 < y < 0.5$ , 并且当障碍物距离小于 0.3 时, 输出 flag=1.

# 10 车道线提取

## 10.1 原理介绍

模拟城市的交通场景如下图所示。



图 10-1 模拟城市交通场景图

我们需要从摄像头采集到的城市交通图中提取出白色的车道线，下面介绍车道线的提取方法。

车道线的提取使用了一系列 opencv 库函数和一些算法。基本思路为将 RGB 的彩色图像做灰度化处理，然后进行 Canny 算子处理的边缘检测，再进行高斯滤波，对滤波后的图像进行直线检测，再对检测到的所有直线进行加权拟合，得到最终拟合出的车道线。

## 10.2 程序运行过程

车道线检测程序运行方法如下：

新打开一个终端，输入如下指令：（注意，该程序和运动控制程序关联，运行该程序后，小车可能会低速运动）

```
cd iinCar/  
source devel/setup.bash  
cd launch/
```

```
roslaunch 8_decision_control.launch
```

终端运行效果如下图所示：

```
NODES
/
  arduino (ros_decision_python/xun_xian.py)
  controller (ros_decision_python/decision_control.py)

ROS_MASTER_URI=http://localhost:11311

process[arduino-1]: started with pid [13145]
process[controller-2]: started with pid [13146]
Connecting to Arduino on port /dev/deepcar ...
Connected at 115200
Arduino is ready.
[INFO] [1609926757.444878]: Connected to Arduino on port /dev/deepcar at 115200
baud
[0] use BaseController
Updating PID parameters
[INFO] [1609926757.536610]: Started base controller for a base of 0.37m wide wit
h 7420 ticks per rev
[INFO] [1609926757.540544]: Publishing odometry data at: 30.0 Hz using base_link
as base frame
[0] init arduino_node complete.
[1] start init arduino_car .
[2] start init control_param .
```

图 10-2 车道线检测程序运行效果图

显示出的车道线检测效果如下图所示。



图 10-3 车道线检测效果图

## 10.3 车道线提取程序及讲解

### 10.3.1 主程序

具体程序如下：

```

#!/usr/bin/env python
# -*- coding: utf-8 -*-
# 代码用途：使用usb 摄像头通过传统图像识别提取车道线
import rosbag
import cv2
import numpy as np
import csv
import datetime
import os
import uuid
import sys
import time
import serial
import threading
import time
import rospy
from geometry_msgs.msg import Twist
from std_msgs.msg import String
from sensor_msgs.msg import Image
from sensor_msgs.msg import CompressedImage
from cv_bridge import CvBridge, CvBridgeError
from ros_decision_python.msg import Lanesite

frame = np.zeros((480,640))
class lane_detection():
    def __init__(self):
        ...
        # change to gray 转为灰度图
    def convert_gray_scale(self,image):
        ...
        # blur process 进行高斯模糊化
    def apply_smoothing(self,image, kernel_size=3):
        ...
        # 对图像边缘检测
    def detect_edges(self,image, low_threshold=300, high_threshold=350):
        ...
        # 设定感兴趣区域，该区域内的图像保留，该区域外的图像变黑
    def filter_region(self,image, vertices,vertices2):
        ...
        # 制作出一个下面为矩形，上面为同底的梯形 的感兴趣区域
    def select_region(self,image):
        ...
        # 检测图像中的直线
    def hough_lines(self,image):
        ...
        # 对一幅图像中检测到的所有直线进行斜率和截距的加权计算，得到左右两条线的斜率和截距
    def average_slope_intercept( self,lines ):
        ...
        # 将一组直线画在图像 image 上，color 为线的颜色，thickness 为线的宽度
    def draw_lane_lines(self,image, lines, color=[255, 0, 0],
thickness=5):
        ...
        # 将以 (slope,intercept) 形式表示的直线转换成用 两个点的坐标(x1,y1)、
        (x2,y2) 表示，其中y1,y2 点的坐标可以自己定。（要求点的坐标为整数）
    def make_line_points(self,y1, y2, line):

```

```

...#
# 将左右两条边线用 两个点的坐标来表示
def lane_lines(self,image, lines):
...#

def callback(self,data):
    # 主循环
    bridge = CvBridge()
    # 每次循环初始化是否找到左右边线的标志位
    findLeftLine = False
    findRightLine = False
    #获取原始图像
    global frame
    frame = bridge.imgmsg_to_cv2(data, "bgr8")
    #将原始图像转换为灰度图
    gray_images = self.convert_gray_scale(frame)
    #对灰度图进行高斯滤波
    blurred_images = self.apply_smoothing(gray_images)
    # 进行边缘检测 和绘制感兴趣区域
    edge_images = self.detect_edges(blurred_images)
    roi_images =self.select_region(edge_images)
    #对感兴趣区域的图像进行直线检测
    hough_list_lines=self.hough_lines(roi_images)
    #对检测到的直线进行加权拟合，提取出左右两条边线。
    if hough_list_lines is not None:
        self.left_lane,
\self.right_lane=self.average_slope_intercept(hough_list_lines)

        self.draw_lane_lines(frame,
\self.lane_lines(frame,hough_list_lines))

# 主函数
if __name__ == '__main__':
    try:
        #初始化节点，便于接收相机的topic
        rospy.init_node('startcar', anonymous=True)
        # 为Lane_detection()定义一个对象，同时完成类中变量的初始化工作。
        lane=lane_detection()
        r = rospy.Rate(25)
        while not rospy.is_shutdown():
            global frame
            cv2.imshow('frame',frame)
            cv2.waitKey(1)
            r.sleep()
    except SerialException:
        rospy.logerr("Serial exception trying to open port.")
        os._exit(0)
    cv2.destroyAllWindows()

```

## 10.3.2 各函数讲解

### 10.3.2.1 彩色图像转换为灰度图

函数如下：

```
# change to gray 转为灰度图
def convert_gray_scale(self,image):
    return cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
```

该函数：输入的图像为 RGB 彩色图 image。

返回值为经过 cv2.cvtColor 函数转换的灰度图。

cv2.cvtColor() 函数中的 **cv2.COLOR\_RGB2GRAY** 参数表示将图像从 RGB 颜色空间转换为 GRAY 颜色空间。

图像在计算机中的存储形式是二维矩阵。对于 RGB 颜色空间类型的图像，其对应二维矩阵的每个元素为 [B,G,R] 形式的一维数组，B，G，R 分别代表 B、G、R 三个通道的值。

从 RGB 颜色空间向 GRAY 颜色空间转换的计算公式有很多，其中的一种转换方式如下：

$$\text{Gray} = R \cdot 0.299 + G \cdot 0.587 + B \cdot 0.114$$

完成转换后，得到了新的灰度图像，它同样由二维数组构成，并且二维数组的行数和列数与 RGB 图像相同，不同之处在于：二维数组的每个元素是一个 [0,255] 之间的整数，该整数由上面的公式计算得到。

灰度图一个元素越接近 0，灰度越大，颜色越接近黑色；反之，灰度图的一个元素越接近 255，颜色越接近白色。

选取初始待处理 RGB 图像如下。



图 10-4 初始待处理 RGB 图像

对上图进行灰度化的处理，得到的灰度图如下：



图 10-5 灰度化处理后的图像

### 10.3.2.2 对灰度图高斯滤波

高斯滤波是一种线性平滑滤波，适用于消除高斯噪声，广泛应用于图像处理的减噪过程。

高斯滤波是通过对输入的二维数组的每个点与输入的高斯滤波模板执行卷积计算，然后将这些结果一块组成了滤波后的二维输出数组。通俗的讲就是高斯滤波是对整幅图像进行加权平均的过程，每一个像素点的值都由其本身和邻域内的其他像素值经过加权平均后得到。高斯滤波的具体操作是：用一个模板

(或称卷积、掩模) 扫描图像中的每一个像素，用模板确定的邻域内像素的加权平均灰度值去替代模板中心像素点的值。

函数如下：

```
def apply_smoothing(self,image, kernel_size=3):
    #kernel_size 需要是：正值和奇数
    return cv2.GaussianBlur(image, (kernel_size, kernel_size), 0)
```

该函数的核心是 cv2.GaussianBlur(src,ksize,sigmaX)，也就是高斯滤波函数，它的参数介绍如下：

**src**: 输入图像，图像可以具有任意数量的通道，这些通道可以独立处理，但深度应为 CV\_8U, CV\_16U, CV\_16S, CV\_32F 或 CV\_64F。

**ksize**: 高斯内核大小，也就是上面介绍的高斯模板的大小。 ksize.width 和 ksize.height 可以不同，但它们都必须为正数和奇数。这里 ksize 的高和宽取为 3。

**sigmaX**: X 方向上的高斯核标准偏差，这里取 0。

下面对 2.2.1 中的灰度图进行高斯滤波，分别取不同的 ksize，得到滤波后的结果如下所示：

当 ksize = 3 时，得到的图像如下：

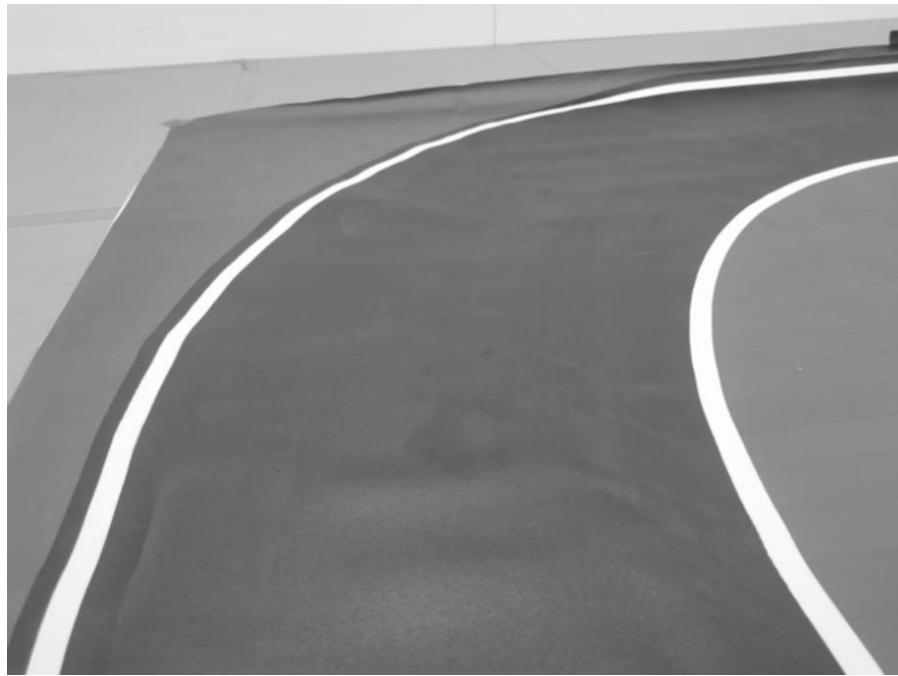


图 10-6  $\text{ksize}=3$  滤波后的图像

由于  $\text{ksize}=3$  很小，高斯滤波处理后的输出图像和输入的灰度图差别不大。

当  $\text{ksize} = 9$  时，得到的图像如下：



图 10-7  $\text{ksize}=9$  滤波后图像

此时  $\text{ksize} = 9$ ,输出的图像比较模糊，这是由于高斯滤波模板的边长较大，图像的不同灰度的像素点之间进行了较大程度的加权平均，导致不同像素值的

边缘部分不再明显。

在本例中灰度图没有太多噪声，因此高斯滤波的效果不明显。但是往往灰度图中会有大量噪声，这些噪声必须经过滤除，才能使后面的边缘检测步骤效果良好，因此高斯滤波或者其他滤波方式是提取边线算法中必不可少的一环。

### 10.3.2.3 对高斯滤波后图像的边缘检测

为了提取出车道线，边缘检测是必不可少的一步。边缘检测是一个多阶段的算法，即由多个步骤构成。

边缘检测的步骤有：

- a. 图像降噪
- b. 计算图像梯度
- c. 非极大值抑制
- d. 阈值筛选

#### a. 图像降噪

图像降噪部分在 2.2.2 的高斯滤波中已经处理过。

#### b. 计算图像梯度

计算图像梯度，得到可能的边缘。边缘是灰度变化明显的地方，梯度同样是灰度变化明显的地方，因此对边缘的检测可以通过对图像中梯度的检测得到。当然，灰度变化明显的地方不一定是边缘，因此这一步实际得到了所有可能的边缘的集合。

#### c. 非极大值抑制

非极大值抑制。通常灰度变化的地方都比较集中，将局部范围内的梯度方向上，灰度变化最大的保留下来，其它的不保留，这样可以剔除掉大部分的点。将有多个像素宽的边缘变成一个单像素宽的边缘。即“胖边缘”变成“瘦边缘”。

#### d. 阈值筛选

双阈值筛选。通过非极大值抑制后，仍然有很多的可能边缘点。进一步的设置一个双阈值，即低阈值（low），高阈值（high）。灰度变化大于 high 的，

设置为强边缘像素，灰度变化低于 low 的，剔除。在 low 和 high 之间的灰度变化值设置为弱边缘。再对弱边缘进一步判断，如果弱边缘像素连接到一个强边缘像素，就保留弱边缘，如果没有，就剔除弱边缘。

函数如下：

```
def detect_edges(image, low_threshold=300, high_threshold=350):
    return cv2.Canny(image, low_threshold, high_threshold)
```

该函数中输入图像为高斯滤波处理之后的灰度图，设定的低阈值和高阈值分别为 300, 350。输出的边缘检测图像如下所示：

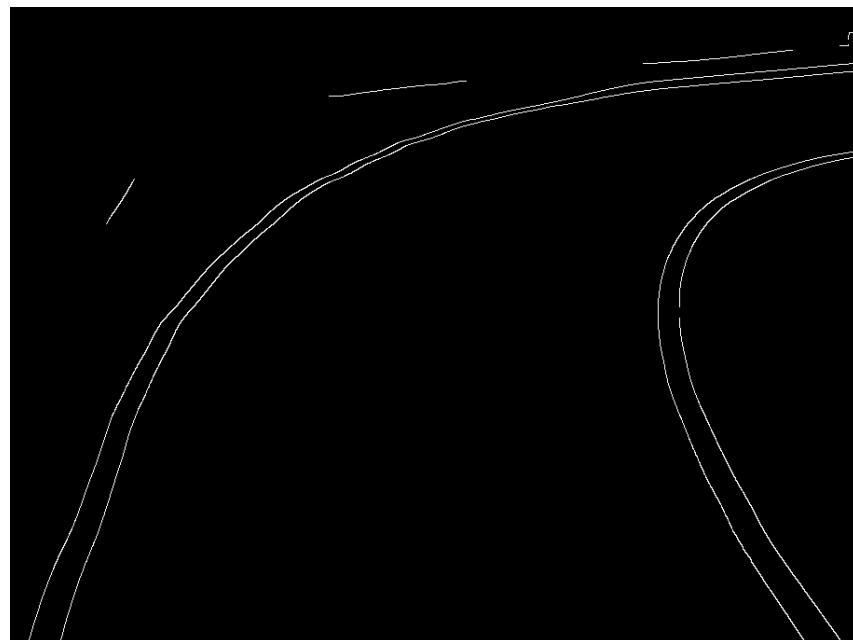


图 10-8 边缘检测图像

从上图中可以看到车道线被完全提取出来。但是同样有一些杂线也被检测出来。下面需要对杂线做进一步的处理。

#### 10.3.2.4 感兴趣区域的选取(ROI)

为了将边缘检测所得图像中的杂线去除，考虑到车道线只会出现在图像中较为固定的位置，这里采取的方法是设置感兴趣区域，也就是只保留图像中特定区域内的像素值，特定区域外的像素值设置为 0，也就是黑色。实践证明，通过设置感兴趣区域，可以很好地滤除杂线，保留车道线，为后面的车道线拟合工作打下了好的基础。

函数如下：

```
def select_region(image):
    """ 这个函数使得图像中被‘verticals’包围的部分的像素得以保留，剩余的部分像素值变为0 """
    rows, cols = image.shape[:2]
    bottom_left = [cols*0.0, rows*1]
    top_left = [cols*0, rows*0.6]
    top_left_middle = [cols*0.3, rows*0.3]
    top_right_middle = [cols*0.7, rows*0.3]
    bottom_right = [cols*1, rows*1]
    top_right = [cols*1, rows*0.6]

    vertices = np.array([bottom_left, top_left, top_right, bottom_right],
                        dtype=np.int32)
    tixing = np.array([top_left_middle,top_right_middle, top_right,
                      top_left], dtype=np.int32)
    return filter_region(image, vertices,tixing)

def filter_region(image, vertices,vertices2):
    mask = np.zeros_like(image)
    if len(mask.shape)==2:
        cv2.fillPoly(mask, [vertices,vertices2], 255)
    else:
        cv2.fillPoly(mask, [vertices,vertices2], (255,) * mask.shape[2])
    return cv2.bitwise_and(image, mask)
```

函数 `select_region()` 用于定义感兴趣区域（实际上是多边形）的形状和各个顶点的位置。这里选择的感兴趣区域是一个矩形和一个等腰梯形的组合，如下图所示。

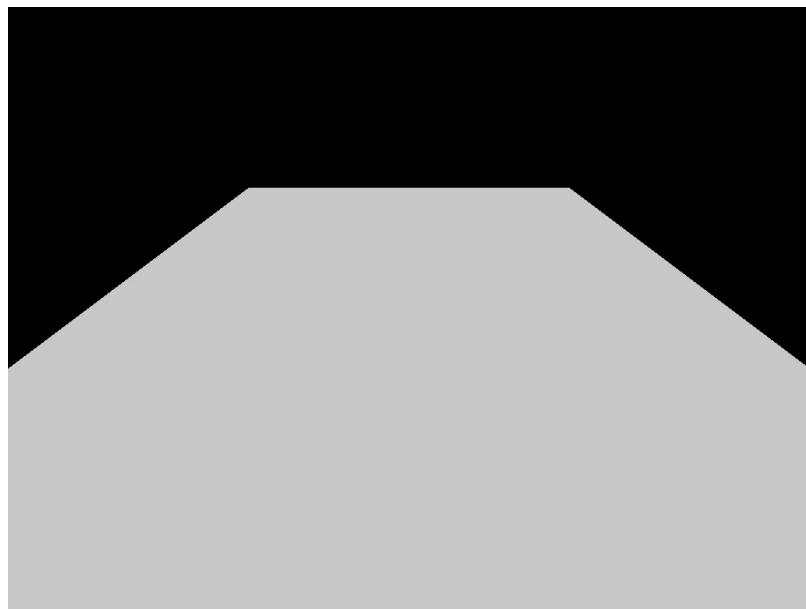


图 10-9 感兴趣区域图

这里感兴趣区域为了区分文档背景，不以白色显示，实际编程中感兴趣区

域为白色，像素值为 255。

函数 `filter_region (image, vertices, vertices2)` 完成感兴趣区域的生成以及对输入图像 `image` 的感兴趣区域截取。其中 `image` 为输入的图像，`vertices` 和 `vertices2` 为感兴趣区域的顶点的集合。需要注意的是，感兴趣区域的生成与 `vertices` 数组中各个顶点的排列顺序有关，按照顶点顺序依次连线生成感兴趣区域。最终，`image` 中处在感兴趣区域内的像素值被保留，其他像素值取 0。

下面对边缘检测后图像进行感兴趣区域截取，生成图像如下所示。

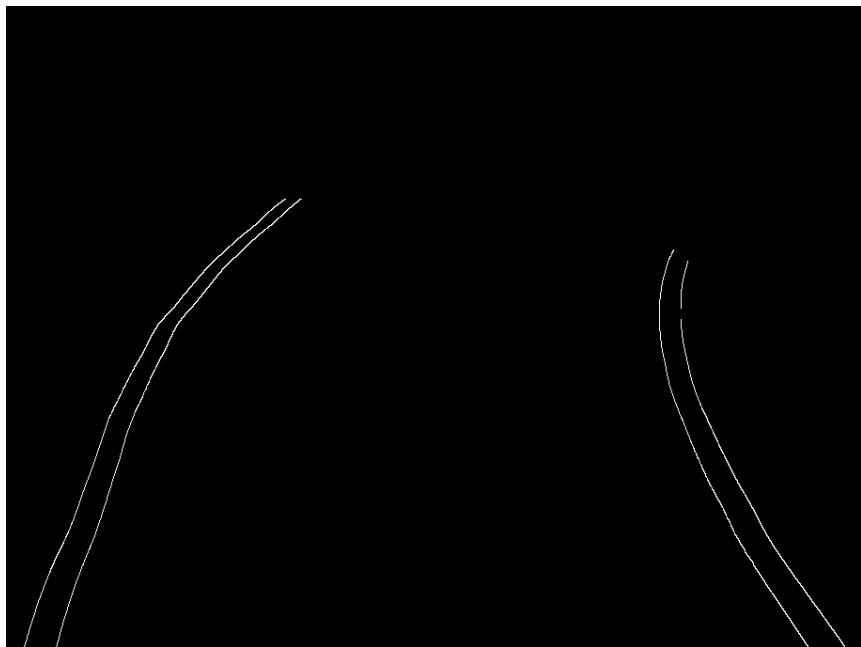


图 10-10 边缘检测进行感兴趣区域截取后的图

上图中杂线已经被去除，虽然远处部分车道线同样被去除，但是不影响对近处车道线的拟合。

#### 10.3.2.5 直线检测与车道线直线拟合

下面对经过感兴趣区域截取得到的图像进行直线检测和拟合，得到拟合出的左右两条车道线。

函数如下：

```
#检测图像中的直线 threshold: 认定为一条直线所需点的数量; minLineLength: 线段的最小长度 maxLineGap: 一条直线上两个点的最大距离
def hough_lines(self,image):
    """
    `image` should be the output of a Canny transform.
```

```

Returns hough Lines (not the image with lines)
"""

return cv2.HoughLinesP(image, rho=1, theta=np.pi/180,
threshold=10, minLineLength=
50, maxLineGap=5)

# 对一幅图像中检测到的所有直线进行斜率和截距的加权计算，得到左右两条线的斜率和截距
def average_slope_intercept( self,lines ):
    left_lines = [] (slope, intercept)
    left_weights = [] (length,)
    right_lines = [] (slope, intercept)
    right_weights = [] (length,)

#lines=hough_Lines(roi_images)

for line in lines:
    #print line
    x1, y1, x2, y2 = line[0]
    if x2==x1:
        continue # ignore a vertical line
    slope = float((y2-y1))/(x2-x1)
    intercept = y1 - slope*x1
    length = np.sqrt((y2-y1)**2+(x2-x1)**2)
    if slope < -0.2 and slope >-10: # y is reversed in image
        left_lines.append((slope, intercept))
        left_weights.append((length))
    elif slope > 0.4 and slope <10:
        right_lines.append((slope, intercept))
        right_weights.append((length))

    # add more weight to longer lines
    left_lane_1 = np.dot(left_weights, left_lines)
    /np.sum(left_weights) if len(left_weights) >0 else None
    right_lane_1 = np.dot(right_weights,
right_lines)/np.sum(right_weights) if len(right_weights)>0 else None

return left_lane_1, right_lane_1 # 返回的左右两条线的数据形式
(slope, intercept), (slope, intercept)

```

函数 cv2.HoughLinesP(image, rho=1, theta=np.pi/180, threshold=10, minLineLength=0, maxLineGap=5) 是累计概率霍夫变换函数，为霍夫变换的一种。该函数可以检测到图像中的直线。

该函数的返回值为检测到的所有的直线段。

该函数的具体输入参数介绍如下：

image：输入图像，必须是二值图像。

rho、theta：取默认值 1、np.pi/180 即可。

**threshold**：在极坐标空间相交的曲线的个数阈值。该值越大，能够检测出的直线数量越少。

**minLineLength**：检测出的线段的长度阈值大小。只有长度大于该值的线段才会被检测出来。

**maxLineGap**：同一方向上两条线段判定为一条线段的最大允许间隔（断裂）。值越大，允许线段上的断裂越大，越可能检测出潜在的直线段。

函数 `average_slope_intercept( lines )` 用于对检测到的直线段进行加权平均，拟合出左右两条车道线。

该函数的输入参数是使用 `HoughLinesP()` 检测到的所有直线段 `lines`。

根据左右车道线的斜率特征可知，左车道线的斜率为负值，右车道线的斜率为正值。因此将输入的直线段按照斜率的正负分成两组分别计算。分别计算出每一条直线段的斜率、截距和长度。对于两组直线段，以长度作为权值，对各直线段的斜率和截距进行加权取平均计算，各得到一条直线段，作为拟合出的左右车道线。

对进行感兴趣区域提取后的图像采取直线检测和拟合的处理，将拟合出的左右车道线画在原始图像上，如下所示：

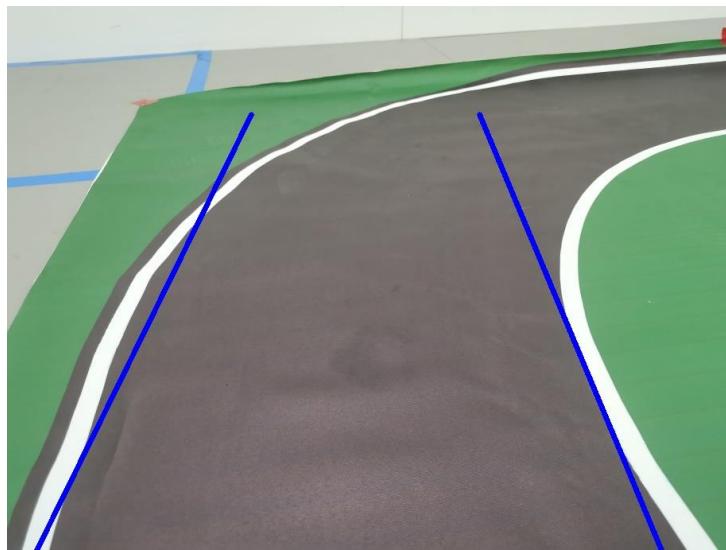


图 10-11 图像处理后车道线显示图

上图中显示了拟合出的左右两条车道线，可以看到拟合的车道线和实际的白色车道线的割线基本一致，说明成功从原始图像中提取出了车道线。

### 10.3.3 巡线控制教程——转向量计算

小车之所以能够巡线行驶，是因为小车可以调整自身位置，尽量行驶在道路中间。要做到这一点，需要实时计算出小车自身与道路中间位置的偏差量，从而可以将该偏差量作为 PID 控制算法的控制量，控制小车的转向。这里将小车自身与道路中间位置的偏差量称作转向量。

小车通过搭载的摄像头接收前方交通道路图像，由于图像的大小固定为高 480，宽 640，因此可以将图像中的半屏宽，也就是宽 320 的位置作为小车自身的位置。

首先介绍图像中坐标系的定义，如下图所示：

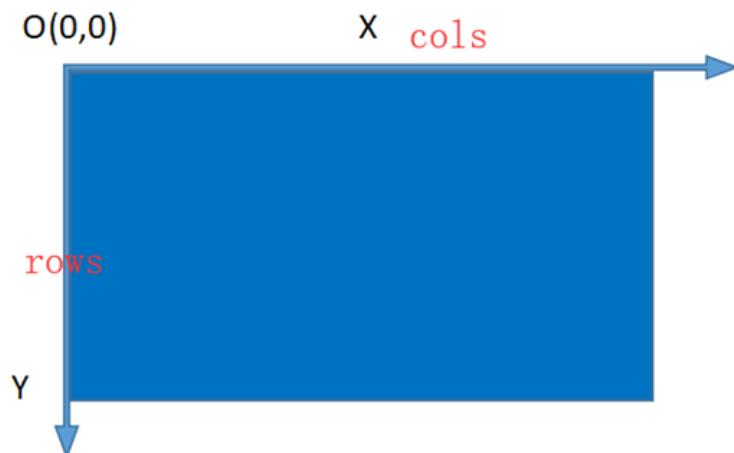


图 10-12 图像中坐标系定义图

车道线中间位置的确定如下：设定图像中  $n$  个固定的  $y$  值，分别计算对应的两条拟合的车道线的  $x$  值的平均值，再对所得的  $n$  个  $x$  值加权平均，作为车道中间位置的  $x$  值。

转向量 =  $320$ （也就是半屏宽） - 车道线中间位置的  $x$  值。当转向量为正值，说明小车处在车道线中间位置的右方，此时需要向左转向；当转向量为负值，说明小车处在车道线中间位置的左方，此时需要向右转向。

小车的实际行驶过程中，并不能一直检测到左右两条车道线，有时会出现只检测到一条左车道线或一条右车道线或没有检测到车道线的情况。在不同的情况下，需要计算不同变量的值：

当左右两条线都检测到时，需要计算转向量、道路宽度。转向量的计算前

面提到过，特定  $y$  值处的道路宽度 = 右车道线的  $x$  值 - 左车道线的  $x$  值。

当只检测到左车道线，需要计算右车道线位置、转向量。右车道线位置  $x$  值 = 左车道线位置  $x$  值 + 道路宽度。右车道线位置计算出后，转向量就可以计算出来。

当只检测到右车道线，需要计算左车道线位置、转向量。左车道线位置  $x$  值 = 右车道线位置  $x$  值 - 道路宽度。左车道线位置计算出后，转向量同样可以计算出来。

当没有检测出任何一条车道线，当前时刻转向量 = 前一时刻转向量。

具体算法程序此处不列出。

计算出转向量后，将转向量发送给运动决策层，通过 PID 算法控制小车巡线。

### 习题：

1. 请指出 RGB 图像和灰度图像在计算机表示中的区别。
2. 请指出高斯滤波的原理和滤波的具体过程。
3. 在边缘检测前，如果不对图像进行滤波的预处理可能会发生什么。
4. 为什么车道线检测中的左车道线斜率一般为负值，右车道线斜率为正值？

### 参考答案：

1. RGB 图像和灰度图在计算机中都以二维数组的形式存储，不同在于 RGB 图像二维数组的每个元素由 3 个在 0-255 之间的整数构成，灰度图的二维数组的每个元素由一个在 0-255 之间的整数构成。
2. 高斯滤波通过对输入的二维数组的每个点与输入的高斯滤波模板执行卷积计算，然后将这些结果一块组成了滤波后的二维输出数组。  
高斯滤波的具体操作是：用一个模板（或称卷积、掩模）从左到右、从上到

下依次扫描图像中的每一个像素。模板每次扫描都确定一个邻域，图像该领域内的各像素进行加权取平均，用得到的灰度值去替代模板中心像素点的值。

3. 边缘检测前，如果不对图像进行滤波的预处理操作，可能因为图像中存在大量噪声，导致边缘检测时检测出除目标外大量无关线段。
4. 一般左车道线方向为左下到右上，右车道线方向为右下到左上，根据 OpenCV 中图像的 x、y 坐标系建立方式（见巡线控制教程——转向量计算教程中图片），可以推算出左车道线斜率一般为负值，右车道线斜率为正值。

## 11 目标检测

### 11.1 目标检测的基础——卷积神经网络

卷积神经网络（convolutional neural network）是含有卷积层（convolutional layer）的神经网络。本讲义介绍的卷积神经网络均使用最常见的二维卷积层。它有高和宽两个空间维度，常用来处理图像数据。

#### 11.1.1 二维互相关运算

虽然卷积层得名于卷积（convolution）运算，但我们通常在卷积层中使用更加直观的互相关（cross-correlation）运算。在二维卷积层中，一个二维输入数组和一个二维核（kernel）数组通过互相关运算输出一个二维数组。如下图所示，输入是一个高和宽均为 3 的二维数组。我们将该数组的形状记为  $3 \times 3$  或  $(3, 3)$ 。核数组的高和宽分别为 2。该数组在卷积计算中又称卷积核或过滤器（filter）。卷积核窗口（又称卷积窗口）的形状取决于卷积核的高和宽，即  $2 \times 2$ 。图中的阴影部分为第一个输出元素及其计算所使用的输入和核数组元素： $0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3 = 19$

输入	核	输出													
<table border="1" style="border-collapse: collapse; width: 100%;"><tr><td style="padding: 2px;">0</td><td style="padding: 2px;">1</td><td style="padding: 2px;">2</td></tr><tr><td style="padding: 2px;">3</td><td style="padding: 2px;">4</td><td style="padding: 2px;">5</td></tr><tr><td style="padding: 2px;">6</td><td style="padding: 2px;">7</td><td style="padding: 2px;">8</td></tr></table>	0	1	2	3	4	5	6	7	8	$\ast$	<table border="1" style="border-collapse: collapse; width: 100%;"><tr><td style="padding: 2px;">0</td><td style="padding: 2px;">1</td></tr><tr><td style="padding: 2px;">2</td><td style="padding: 2px;">3</td></tr></table>	0	1	2	3
0	1	2													
3	4	5													
6	7	8													
0	1														
2	3														
		=													
		<table border="1" style="border-collapse: collapse; width: 100%;"><tr><td style="padding: 2px;">19</td><td style="padding: 2px;">25</td></tr><tr><td style="padding: 2px;">37</td><td style="padding: 2px;">43</td></tr></table>	19	25	37	43									
19	25														
37	43														

图 11-1 卷积原理图

在二维互相关运算中，卷积窗口从输入数组的最左上方开始，按从左往右、从上往下的顺序，依次在输入数组上滑动。当卷积窗口滑动到某一位置时，窗口中的输入子数组与核数组按元素相乘并求和，得到输出数组中相应位置的元素。

我们可以依据这个原理使用 pytorch 定义我们自己的互相关函数。

```
def corr2d(X,k):
    h,w = k.shape           #通过 shape 获得卷积核的宽和高
    Y = torch.zeros((X.shape[0] - h + 1, X.shape[1] - w + 1))  #定义输出
    for i in range(Y.shape[0]):      #依据互相关运算对输出进行计算
        for j in range(Y.shape[1]):
            Y[i][j] = (X[i:i+h, j:j+w]* k).sum()
    return Y
```

## 11.1.2 二维卷积层

二维卷积层将输入和卷积核做互相关运算，并加上一个标量偏差来得到输出。卷积层的模型参数包括了卷积核和标量偏差。在训练模型的时候，通常我们先对卷积核随机初始化，然后不断迭代卷积核和偏差。

我们可以利用上述我们的互相关函数定义我们的卷积层。

```
class Conv2d(nn.Module):
    def __init__(self,kernel_size,**kwargs):
        super(Conv2d,self).__init__(**kwargs)
        self.weight = nn.Parameter(torch.randn(kernel_size))
        self.bias = nn.Parameter(torch.randn(1))
    def forward(self,x):
        return corr2d(x,self.weight)+self.bias
```

上述代码中我们使用 `nn.Parameter` 定义卷积层的参数，这些参数是可以通过优化器更新的。

## 11.1.3 多输入通道和多输出通道

前面我们用到的输入和输出都是二维数组，但真实数据的维度经常更高。例如，彩色图像在高和宽 2 个维度外还有 RGB（红、绿、蓝）3 个颜色通道。假设彩色图像的高和宽分别是  $h$  和  $w$ （像素），那么它可以表示为一个  $3 \times h \times w$  的多维数组。我们将大小为 3 的这一维称为通道（channel）维。这里我们将介绍含多个输入通道或多个输出通道的卷积核。

### 11.1.3.1 多输入通道

我们用一副图片来表示多输入通道的运算

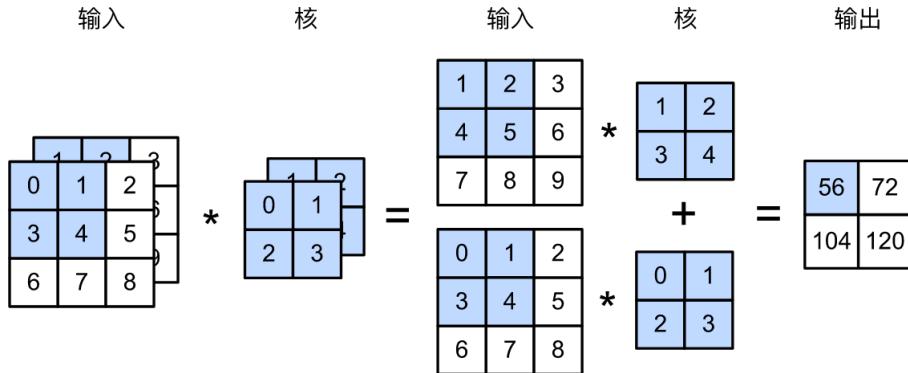


图 11-2 多通道输入运算图

如上图所示，多输入通道其实就是在对应的通道做之前我们定义的互相关运算，再求和。

```
def corr2d_multi_in(X, K):
    # 沿着 X 和 K 的第 0 维（通道维）分别计算再相加
    res = d2l.corr2d(X[0, :, :], K[0, :, :])
    for i in range(1, X.shape[0]):
        res += corr2d(X[i, :, :], K[i, :, :])
    return res
```

### 11.1.3.2 多输出通道

当输出也是多通道，我们可以再在卷积核上增加一个维度，来对应输出的通道。

```
def corr2d_multi_in_out(X, K):
    # 对 K 的第 0 维遍历，每次同输入 X 做互相关计算。所有结果使用 stack
    # 函数合并在一起

    return torch.stack([corr2d_multi_in(X, k) for k in K])
```

这里的 `stack` 函数就是将 `corr2d_multi_in(X, k)` 输出的结果在一个新的维度上进行合并。

### 11.1.4 池化层

在这里我们介绍池化（pooling）层，它的提出是为了缓解卷积层对位置的过度敏感性。

同卷积层一样，池化层每次对输入数据的一个固定形状窗口（又称池化窗

口) 中的元素计算输出。不同于卷积层里计算输入和核的互相关性，池化层直接计算池化窗口内元素的最大值或者平均值。该运算也分别叫做最大池化或平均池化。在二维最大池化中，池化窗口从输入数组的最左上方开始，按从左往右、从上往下的顺序，依次在输入数组上滑动。当池化窗口滑动到某一位置时，窗口中的输入子数组的最大值即输出数组中相应位置的元素。

它跟二维卷积里 corr2d 函数非常类似，唯一的区别在计算输出 Y 上

```
def pool2d (x,pool_size,mode='max'):
    x = x.float()
    h,w = x.shape
    p_h,p_w = pool_size
    y = torch.zeros((h-p_h+1,w-p_w+1))
    for i in range(y.shape[0]):
        for j in range(y.shape[1]):
            if mode == 'max':
                y[i][j] = x[i:i+p_h,j:j+p_w].max()
            if mode == 'avg':
                y[i][j] = x[i:i+p_h,j:j+p_w].mean()
    return y
```

## 11.2 深度学习基础

### 11.2.1 线性回归

现实中有很多因素之间的关系可以用线性来表示，线性回归要做的就是使用线性回归模型来表达这些因素之间的关系。我们以一个简单的房屋价格预测作为例子来解释线性回归的基本要素。这个应用的目标是预测一栋房子的售出价格（元）。我们知道这个价格取决于很多因素，如房屋状况、地段、市场行情等。为了简单起见，这里我们假设价格只取决于房屋状况的两个因素，即面积（平方米）和房龄（年）。接下来我们希望探索价格与这两个因素的具体关系。

#### 11.2.1.1 线性回归模型

线性回归顾名思义，线性回归假设输出与各个输入之间是线性关系。线性回归的数学表达式可以用下式表示。

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_k x_k + \varepsilon$$

有了线性回归的数学表达式就可以在已知权重、偏差的情况下将输入代入

上述式子得到输出。 线性回归的神经网络模型如下图所示。

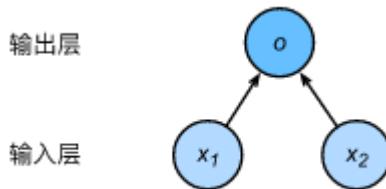


图 11-3 线性回归的神经网络模型

### 11.2.1.2 线性回归的简洁实现

我们主要通过代码来讲述线性回归的实现，本讲义的代码都是在 pytorch 上实现的。

```
import torch          #pytorch 库
from torch import nn #pytorch 中与神经网络相关的库
class LinearNet(nn.Module): #通过继承 nn.Module 来声明
    LinearNet
    def __init__(self,n_features): #初始化函数
        super(LinearNet,self).__init__() #调用父类的初始化函数
        self.linear = nn.Linear(n_features,1) #这里的 nn.Linear 是
    pytorch 提供的层
    def forward(self,x): #前向传播
        y = self.linear(x) #调用 linear 进行计算
        return y

net = nn.Sequential(nn.Linear(num_inputs,1))
```

由于之后读者可能会定义自己的层或者网络，所以这里采用了自己定义网络的方式，如果想要直接通过调用 pytorch 提供的层来实现的话可以直接通过下列代码实现。

```
net = nn.Sequential(nn.Linear(n_features,1))
y = net(x)
```

通过上述代码我们就定义了 LinearNet，在开始训练之前，要对网络的参数进行初始化，这里我们将 weight 进行正态分布式的随机初始化，并且将 bias 初始化为一个常量。

```
for param in net.parameters(): #net.parameters()返回 net 的参数迭代器
    print(param)               #查看 net 的参数
params = net.parameters()
init.normal_(next(params),mean=0,std=0.01) #对 weight 进行初始化
```

```
init.constant_(next(params), val=0) #对 bias 进行初始化
```

对参数进行初始化后，我们还需要准备训练时需要的工具，也就是损失函数和优化器，这里我们采用平方函数作为我们的损失函数，采用 SGD 作为我们的优化器。

```
loss = nn.MSELoss() #采用 pytorch 库中的平方函数
optimizer = optim.SGD(net.parameters(), lr=0.03) #第一个参数为我们要进行训练的参数，第二个参数是学习率
```

做好上述准备之后，就可以进行训练了。

```
num_epochs = 3 #训练的周期数

for epoch in range(num_epochs):
    for x,y in data_iter: #遍历数据
        x = x.to(torch.float32) #转换类型，确保类型为 float32
        y = y.to(torch.float32)
        y_hat = net(x) #获得网络预测的输出
        l = loss(y_hat,y.view(y_hat.size())) #求取损失函数值
        optimizer.zero_grad() #清空网络参数的梯度值
        l.backward() #进行反向传播
        optimizer.step() #更新网络参数值
        print('epoch:%d, loss:%f'%(epoch,l.item())) #打印训练成果

有了网络，又可以进行训练之后，我们就可以将我们准备好的训练数据用来训练网络了，这里我们也可以直接采取自己生成数据的方式。
num_examples = 1000 #样本数
num_inputs = 2 #自变量个数
true_w = torch.ones(1,2,dtype=torch.double) #用来生成数据的 weight
true_w = torch.transpose(true_w,0,1)
true_b = 4.2 #用来生成数据的 bias
features =
torch.from_numpy(np.random.normal(0,1,(num_examples,num_inputs))) #获得自变量
labels = torch.mm(features,true_w)+true_b #获得标签

batch_size = 10
dataset = Data.TensorDataset(features,labels)
data_iter = Data.DataLoader(dataset,batch_size,shuffle=True)
```

这样我们就可以采用我们自己生成的数据来验证我们的网络以及训练函数是否正确。

## 11.2.2softmax 回归

之前介绍的线性回归模型适用于输出为连续值的情景。在另一类情景中，模型输出可以是一个像图像类别这样的离散值。对于这样的离散值预测问题，我们可以使用诸如 softmax 回归在内的分类模型。和线性回归不同，softmax 回

归的输出单元从一个变成了多个，且引入了 softmax 运算使输出更适合离散值的预测和训练。本节以 softmax 回归模型为例，介绍神经网络中的分类模型。

### 11.2.2.1 softmax 模型

softmax 回归跟线性回归一样将输入特征与权重做线性叠加。与线性回归的一个主要不同在于，softmax 回归的输出值个数等于标签里的类别数。因为一共有 4 种特征和 3 种输出动物类别，所以权重包含 12 个标量、偏差包含 3 个标量。softmax 的神经网络模型如下图所示。

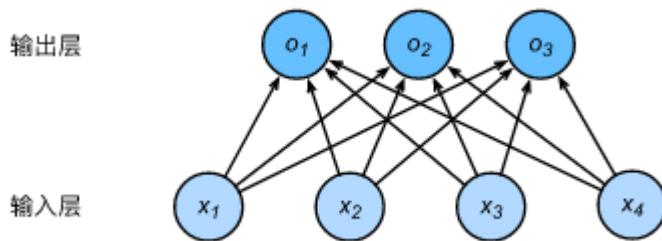


图 11-4 softmax 的神经网络模型

### 11.2.2.2 softmax 的简洁实现

这里我们首先实现 softmax 的模型，我们沿用线性回归的模型来进行实现。

```
class Linear_Net(nn.Module):           #线性回归模型
    def __init__(self,num_inputs,num_outputs):
        super(Linear_Net,self).__init__()
        self.Linear = nn.Linear(num_inputs,num_outputs)
    def forward(self,features):
        net = self.Linear
        return net(features.view(features.shape[0],-1))

class Flatten(nn.Module):               #扁平层
    def __init__(self):
        super(Flatten,self).__init__()
    def forward(self,X):
        return X.view(X.shape[0],-1)      #将 x 展平

net = nn.Sequential()
net.add_module('Flatten',Flatten())
net.add_module('Linear',nn.Linear(num_inputs,num_outputs))
```

由于我们采用的数据集是图片，要将图片输入线性回归模型中，我们需要将图片转换成一维的，也就是上述代码中的 Flatten 之后进行的参数初始化和损失函数、优化器的代码与线性回归的想类似，这里直接给出代码

```

init.normal(net.Linear.weight,0,0.01)
init.constant(net.Linear.bias,0)

loss = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(),lr=0.3)

```

这里需要注意的是，在这里采用的损失函数是交叉熵损失函数。

进行了上述准备工作之后可以进行模型的训练

```

for epoch in range(num_epochs):
    train_l_sum, train_acc_sum, n = 0.0, 0.0, 0
    for features, labels in train_iter:
        y_hat = net(features)
        l = loss(y_hat,labels)
        optimizer.zero_grad()
        l.backward()
        optimizer.step()
        train_l_sum += l*labels.shape[0]
        train_acc_sum += Evaluate_Accuracy(y_hat,labels)*labels.shape[0]
        n += labels.shape[0]
    print('epoch:%d, train_l:%.3f,
train_acc:%0.3f'%(epoch,train_l_sum/n,train_acc_sum/n))

```

这里的 Evaluate\_Accuracy 函数是用来评估模型预测的准确度

```

def Evaluate_Accuracy(y_hat,y):
    return ((y_hat.argmax(dim=1,keepdim=True)==y.view(-1,1)).float().mean()).item()

```

argmax 是找出对应维度(dim)下的最大值的下标， keepdim 是保持 tensor 的维度不变。 y.view(-1,1)是将 y 由行向量转换成列向量。

最后，我们就可以获得数据来训练我们的网络了，这里采用的是 torchvision 提供的 FashionMNIST 数据集

```

import torchvision #导入 torchvision 库
from torchvision import transforms #导入 torchvision 中的 tranforms 库

mnist_train =
torchvision.datasets.FashionMNIST(root='./Datasets/FashionMNIST',train=True,transform=transforms.ToTensor(),download=True) #获得训练数据集
mnist_test =
torchvision.datasets.FashionMNIST(root='./Datasets/FashionMNIST',train=False,transform=transforms.ToTensor(),download=True) #获得测试数据集

batch_size = 256
num_inputs = 784
num_outputs = 10

train_iter = Data.DataLoader(mnist_train,batch_size,shuffle=True) #获得训练数据迭代器

```

```
test_iter = Data.DataLoader(mnist_test,batch_size,shuffle=True)      #获得  
测试数据迭代器
```

### 11.2.3思考

现实生活中有什么数据可以用线性关系表示？

尝试不同的学习率，看看不同学习率对应的学习效果，学习率过大和过小会造成什么不好的影响？

利用 softmax 对图像进行分类有什么缺点？应该用什么方式来弥补？

尝试使用 GPU 对网络进行训练。

试着改变网络的参数，结果有什么不同？

## 11.3 目标检测程序及讲解

### 11.3.1参数设定

#测试数据所在的文件夹

```
parser.add_argument("--image_folder", type=str,  
default=root_path+"/data/mydatasets/testdata", help="path to dataset")  
    #yolov3 模型  
parser.add_argument("--model_def", type=str,  
default=root_path+"/config/yolov3-custom.cfg", help="path to model  
definition file")  
    #模型权重  
parser.add_argument("--weights_path", type=str,  
default=root_path+"/weights/five_cls_20.pth", help="path to weights  
file")  
    #分类类别  
parser.add_argument("--class_path", type=str,  
default=root_path+"/data/mydatasets/classes.txt", help="path to class  
label file")  
    #目标置信度阈值，用来过滤掉置信度较小的  
parser.add_argument("--conf_thres", type=float, default=0.8,  
help="object confidence threshold")  
    #非极大值抑制  
parser.add_argument("--nms_thres", type=float, default=0.6, help="iou  
threshold for non-maximum suppression")  
    #batches  
parser.add_argument("--batch_size", type=int, default=1, help="size  
of the batches")  
    #cpu number  
parser.add_argument("--n_cpu", type=int, default=0, help="number of  
cpu threads to use during batch generation")  
    parser.add_argument("--img_size", type=int, default=416, help="size  
of each image dimension")  
    parser.add_argument("--checkpoint_model", type=str, help="path to  
checkpoint model")
```

```
opt = parser.parse_args()
```

parser 可以实现命令行交互的功能，在启动 python 文件时注明想要输入的参数可以在执行 python 文件时将该参数设定为自己想要设定的值。

### 11.3.2 模型声明

```
model = Darknet(opt.model_def, img_size=opt.img_size).to(device)

if opt.weights_path.endswith(".weights"):
    # Load darknet weights
    model.load_darknet_weights(opt.weights_path)
else:
    # Load checkpoint weights
    model.load_state_dict(torch.load(opt.weights_path,
map_location='cpu'))
    # Set in evaluation mode
    model.eval()
```

Darknet 是 Yolov3 用来检测模型的一个 model；model.load\_darknet\_weights 用来装载已经训练好了的模型的权值；model.eval() 用来设定 model 于 evaluation 模式。

### 11.3.3 加载数据

```
dataloader = DataLoader()
```

DataLoader 是一个数据装载的函数，返回一个数据加载器，可以用作迭代器，在迭代中每轮填充数据并输出。

### 11.3.4 导入图像进行检测

```
for batch_i, (img_paths, input_imgs) in enumerate(dataloader):
    # Configure input
    # 将图像转换成 Variable 类型，使其可以实现反向传播
    input_imgs = Variable(input_imgs.type(Tensor))

    # Get detections
    # 通过 torch.no_grad() 设定 requires_grad=False
    with torch.no_grad():
        detections = model(input_imgs) # 当 model 设定为 evaluate 模式，输入图片可以输出 detection ((x1, y1, x2, y2, object_conf, class_score, class_pred))
        detections = non_max_suppression(detections, opt.conf_thres,
opt.nms_thres) # conf_thres 置信度阈值 nms_thres 非极大值抑制阈值

    # Log progress
    current_time = time.time()
```

```

        inference_time = datetime.timedelta(seconds=current_time -
prev_time)
        prev_time = current_time
        print("\t+ Batch %d, Inference Time: %s" % (batch_i,
inference_time))

        # Save image and detections
        imgs.extend(img_paths)
        img_detections.extend(detections)

```

使用 enumerate 函数利用 dataloader 的可迭代功能，dataloader 每一轮循环输出图片的路径和图片；

input\_imgs.type(Tensor)将图像转换成张量模式，神经网络的输入数据的类型要求是张量；

Variable 相当于是张量的容器，Variable 可以使张量完成反向传播的功能；non\_max\_suppression()为非极大值抑制函数，其可以防止图片中的同一个目标被重复识别并标记。

### 11.3.5 设置颜色

```

#获取'tab20b'配色盘
cmap = plt.get_cmap("tab20b")
#获得'tab20b'的前 20 个配色
colors = [cmap(i) for i in np.linspace(0, 1, 20)]

```

cmap 参数接受一个值（每个值代表一种配色方案），并将该值对应的颜色图分配给当前图窗。

如果将当前图窗比作一幅简笔画，则 cmap 就代表颜料盘的配色，用所提供的颜料盘自动给当前简笔画上色，就是 cmap 所做的事。

### 11.3.6 图像标注

```

if detections is not None:
    # Rescale boxes to original image
    detections = rescale_boxes(detections, opt.img_size,
img.shape[:2])
    #猜测 unique_labels 的第一个元素是序号 (0,1,2,3)
    unique_labels = detections[:, -1].cpu().unique()
    n_cls_preds = len(unique_labels) #图片中类别中物体的数量
    bbox_colors = random.sample(colors, n_cls_preds) #随机获取
colors 中 n_cls_preds 数量的颜色
    for x1, y1, x2, y2, conf, cls_conf, cls_pred in detections:
        #打印 label 和分数

        if(cls_conf.item() < 0.8):

```

```

        continue

    print("\t+ Label: %s, Conf: %.5f" %
(classes[int(cls_pred)], cls_conf.item()))

    box_w = x2 - x1
    box_h = y2 - y1

    color = bbox_colors[int(np.where(unique_labels ==
int(cls_pred))[0])]
    # Create a Rectangle patch
    bbox = patches.Rectangle((x1, y1), box_w, box_h,
linewidth=2, edgecolor=color, facecolor="none")
    # Add the bbox to the plot
    ax.add_patch(bbox)
    # Add label
    plt.text(
        x1,
        y1,
        s=classes[int(cls_pred)]+str("%.6f"%cls_conf.item()),
        color="white",
        verticalalignment="top",
        bbox={"color": color, "pad": 0},
    )
)

```

判断 `xiaion` 为空则不执行下列操作。

判断 `detection` 中有几类被识别的物体，并从 `cmap` 中获取相对应数量的颜色。

将置信度小于 0.8 的被识别物体丢弃。

如果置信度大于 0.8，则使用 `patches.Rectangle` 函数在图片上用矩形边框标注，

并使用 `plt.text` 函数在图片上 `xia`。

### 11.3.7 图像存储

```
plt.savefig(save_path, bbox_inches="tight", pad_inches=0.0)
```

使用 `savefig` 函数将处理后的图片存储在 `save_path` 路径下。

### 11.3.8 消息订阅和发布

```
result_pub = nh_.advertise<detectionInfo_msg::detectionInfo>("/object_detection",1)
```

```
image_pub = it_.advertise("/usb_cam/image_raw", 1);
```

目标检测模块发布了`/object_detection` 消息和`/usb_cam/image_raw` 消息。

`/object_detection` 是目标检测结果。

```

string label
float32 score
float32 left
float32 right

```

```
float32 top  
float32 bottom
```

/usb\_cam/image\_raw 是图片信息。

```
image_sub_ = it_.subscribe("/camera/rgb/image_raw", 1,  
                           &ImageConverter::imageCb, this);
```

目标检测模块订阅了/camera/rgb/image\_raw 消息，这是摄像头发布的 rgb 图像信息。

## 11.4 目标检测常用框架

当我们只是单纯地想实现目标检测这个目的，我们并不需要自己写神经网络的代码，我们只需要使用现成的用于目标检测的框架即可。在这里，我们介绍一些常用的目标检测的框架。

### 11.4.1 RCNN

首先训练分为 4 个阶段：

通过 selective search 算法对于每个图片选取 2000 个建议框（region proposal—rp），框出来有可能有目标的小区域(这也是 rcnn 计算量消耗大的一个原因) 利用得到的 2000 个 rps，对特征提取网络（rcnn 使用的是 Alexnet）进行微调。值得注意的是，在微调 Alexnet 之前，会使用 ImageNet 进行预训练，然后使用 128 的 batchsize 进行微调，其中有 32 个 positive rps ( $\text{IoU} > 0.5$ ) 和 96 个背景 rps。训练 SVM。利用得到的 conv 特征与训练 label，训练 SVM，其中 positive 选取 ground-truth 的 box 框图，negative 选取  $\text{IoU} < 0.3$  以及背景（使用了一种难样本的选取方法） bbox 回归训练。然后对于测试阶段：

对于每张图片，提取 2000 个 rps，要 resize 成一样的大小 将每个 rp 输入 conv，得到特征（也就是要进行 2000 次 conv 提取特征） 然后将 2000 个特征输入训练好的 SVM 以及 bbox 回归器得到分类结果以及回归结果。具体过程是，首先根据 SVM 得到的  $2000 \times 20$  矩阵（假设 20 个分类），进行非最大值抑制，对于每一列（也就是每一类）得分最高的框，对此类剩下的所有的框计算 IoU，如果小于 threshold，则剔除这个框，然后对此类中次高的框进行如上操作直到最后一个框。SVM 以及非最大值抑制处理之后，将剩余的框进行 bbox 回归处理。最后输出评分最高的框。

## 11.4.2 SPPNET

SPP-Net 是一种可以不用考虑图像大小，输出图像固定长度网络结构，并且可以做到在图像变形情况下表现稳定。SPP-net 的效果已经在不同的数据集上面得到验证，速度上比 R-CNN 快 24-102 倍。在 ImageNet 2014 的比赛中，此方法检测中第二，分类中第三。

一个正常的深度网络由两部分组成，卷积部分和全连接部分，要求输入图像需要固定 size 的原因并不是卷积部分而是全连接部分。所以 SPP 层就作用在最后一层卷积之后，SPP 层的输出就是固定大小。SPP-net 不仅允许测试的时候输入不同大小的图片，训练的时候也允许输入不同大小的图片，通过不同尺度的图片同时可以防止 overfit。

## 11.4.3 其他常用框架

Faster R-CNN、You Only Look Once（YOLO）、Single Shot MultiBox Detector（SSD）等等都是当今较为常用的目标检测的框架，大家可以在 Github 上下载代码，根据指示进行环境配置，就可以在自己的电脑上进行目标检测啦。