计算机网络大作业实验报告

目录

1 需求分析

通过ROS平台实现与duckietown小车的通信控制小车进行前进、后退、转弯等基本操作；实时检测摄像头传回的图像，从而可以根据边线与箭头的指示进行运动。

2 整体方案设计

本次大作业运行环境为Ubuntu 16.04，主程序编程语言为python 3.6，Android APP开发环境为 Android Studio，编程语言为Java.

整个大作业的第一步也是和计算机网络联系最紧密的就是网络环境的配置，虚拟机与双系统条件下的配置我都进行了尝试。在虚拟机下无论是NAT还是桥接网卡都无法使主机与树莓派的ROS之间建立网络通信，最终选择了双系统环境下完成了网络配置。

使用键盘按键实现小车运动的控制这一步中最核心的就是学会并正确使用ROS的消息收发机制，在此基础上，利用pyhton脚本捕获键盘按键信息并发送即可。

在大作业的要求外，我还**额外完成了控制小车运动的Android APP开发。**以电脑主机为服务器，通过Socket建立手机与电脑之间的TCP连接并进行数据传输，主机再进一步根据接收到的数据向小车发出指令。

要实现小车对周围环境的感知，最重要的是从摄像头回传图像中准确、鲁棒地识别出关键的图像信息——箭头与边线。为此我使用色块提取、边缘检测等方法，并通过大量测试不断调整参数。

最终完成小车的导航，需要对控制算法进行设计，在之前完成的ROS通信与图像识别之间搭建起桥梁。

3 网络配置

3.1 虚拟机

3.1.1 NAT

虚拟机连接外网的一般方式是通过NAT的端口多路复用（Port address Translation,PAT)方法，即通过主机的特定端口与外网进行通信，不具有独立的IP地址。在这种情况下，虚拟机可以作为客户机但不可以作为服务器，而在ROS通信中，需要使用电脑主机作为主节点，其IP必须为透明的才可以。当然，我也尝试了寻找虚拟机所使用的端口号，试图通过这个端口结合主机原系统的IP地址进行通信，但没有成功，所以只好放弃这个选择。

3.1.2 桥接网卡

虚拟机也可以通过在原系统上创建一个虚拟网卡并将物理网卡与虚拟网卡桥接，实现两者网络的共享，这样，虚拟机和原系统可以获得同一网段中的两个不同地址，使得虚拟机在网络中也成为一个独立主机。但是经过诸多设置和尝试之后，尽管我的原系统和虚拟机之间可以相互ping通，但是用树莓派依旧无法访问虚拟机，最终不得已放弃了虚拟机环境，转而使用双系统。

3.2 双系统

3.2.1 ssh连接

安装双系统后，上述问题自动解决。为了尽可能避免使用显示屏，可以通过ssh建立树莓派与电脑的通信。但如何在不知道树莓派IP地址的情况下建立ssh呢？我回顾了这学期计算机网络课程实验中学到的一些技巧，探索出一个“曲线救国”的方法：先用最后一个计网实验中所制作的网线连接主机和树莓派，通过arp -a命令更新arp表，使得主机知晓树莓派在子网中的存在并获取其主机名。这样，在断开网线后，便可以使用ssh duckiebot@duckiebot1.local命令建立

起 ssh 连接。当然，直接通过网线连接获得的以太网 IP 地址建立 ssh 连接，再用 ifconfig 查询树莓派的无线网 IP 地址也是可以的。

实践中发现，以上两种方法中，arp -a 都是必不可少的一步，否则主机无法获取树莓派的主机名或 IP 地址等信息，而且 arp -a 命令一般不返回无线网子网中没有建立过连接的主机信息，所以我使用了网线连接。

3.2.2 修改 .bashrc 文件

为了正确建立起树莓派上 ROS 节点与主节点之间的通信，需要分别在两者的系统中配置 ROS_HOSTNAME 与 ROS_MASTER_URI，在主机上两者都为主机的地址，在树莓派上分别为树莓派地址与主机地址。其中，ROS_MASTER_URI 写为如下格式：

```
http://[ip of the master]:11311
```

为了避免每次打开终端都需要执行 export 命令来进行配置，可以直接将这两行命令写入.bashrc 文件中，此后打开的新终端便可以自动配置好。（**注意**：一定要在修改配置后打开新终端进行进一步操作）

3.2.3 消息节点的建立与监视

在主机上键入 roscore 命令便可以运行主节点（**注意**：一定保证在建立其他节点之前运行主节点，且在通信过程中一直保持运行），在 ssh 连接的终端中键入 roslaunch 命令运行所需节点，如：

```
$roslaunch duckietown joystick.launch veh:=duckiebot1
```

```
$roslaunch duckietown camera.launch veh:=duckiebot10 raw:=true rect:=true freq:=10
```

（以上命令分别运行了运动控制节点与摄像头节点）

此时可以使用 rostopic 命令查看节点内容、进行消息发布与接收，如：

```
$rostopic list
```

```
$rostopic pub /duckiebot1/joy_mapper_node/car_cmd duckietown_msgs/Twist2DStamped "header:
  seq:0
  stamp:0
    secs:0
```

```
   nsecs:0
   frame_id: ''
v:1.0
omega:0.0"
```

```
$rostopic echo /duckiebot1/joy_mapper_node/car_cmd
```

（以上命令分别实现了查看当前所有的 topic、向运动控制节点发送消息实现小车向前运动、显示运动控制节点的通信信息）

4 键盘控制小车运动与获取图像

4.1 获取键盘指令

使用 termios 包中的相关函数获取键盘输入，可以通过两种方式进行判断按键：按键的编号或者按键代表的字符。由于方向键返回的不是字符而是一个字符串，则可以通过其中是否包含特征字符来判断（上下右左分别包含 ABCD）。

需要注意读取键盘输入会改变系统标准输入的设置，所以在结束循环时应恢复原设置，并在使用按键强制中断程序时抛出异常，恢复设置，以免发生之后无法使用键盘输入的情况。

4.2 Python 中发送 ROS 消息

用 rospy 包中的函数初始化一个节点以及 publisher，通过一个循环不断读取键盘输入并判断，写入消息并 publish，小车在接收到消息后便会进行相应运动。循环可以通过设置节点的运行频率来进行控制。

我们可以用 rospy.loginfo 来记录与输出程序中的关键信息，便于观察与调试。

4.3 获取摄像头图像

与之前类似，只不过这一步不是发送信息而是接收信息。新建一个 subscriber 来不断获取传回的图像信息，在回调函数中用 numpy 读取返回的字

符串流再用 opencv 解码为图像并显示。

为了两者可以不冲突地运行，我在主函数中开了两个线程分别运行。

4.3 操作步骤

1. 网络配置与建立 ssh 连接
2. 主机运行 roscore
3. 树莓派运行：

```
$roslaunch duckietown joystick.launch veh:=duckiebot1
$roslaunch duckietown camera.launch veh:=duckiebot10 raw:=true rect:=true freq:=10
```

4. 主机运行 move+camera.py 脚本，可以看到实时回传的图像
5. 通过上下左右方向键或者 WSAD 键控制小车运动，按下空格停止，按下 q 退出程序

5 Android APP 开发

5.1 主机服务器搭建

在上述运动控制部分的代码中，将从键盘读入字符换为通过 socket 通信获取控制命令即可。具体实现为：每次进入发送消息的循环时，开始建立 socket 并对某一特定端口进行监听，获取到信息后发送确认信息并关闭 TCP 连接，再把这个信息通过 ROS 通信发送给小车。这样，主机相当于实现了移动端 APP 与小车通信的中继。

5.2 APP 开发

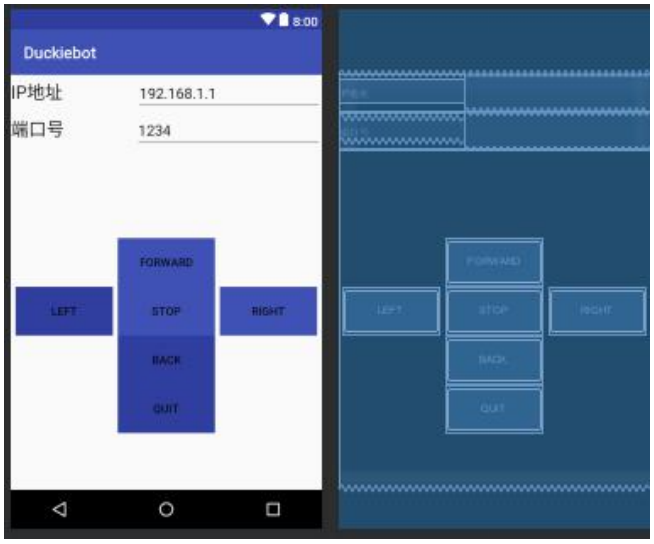
5.2.1 网络通信

类似地，在 Java 中也有 socket 通信的包，可以比较方便的调用。所以这一部分的开发主要在于各种网络情况下客户端给使用者的反馈：如连接失败、端口错误等，并抛出相应异常，保证程序不会崩溃。另外，Android 中的网络通信开

发还有诸多限制，如 APP 要向手机提出网络权限请求、网络通信必须运行在主线程之外的线程中、对 Android 系统的版本有要求等等，我都一一解决。

5.2.2 界面设计

界面设计以及界面相应与网络通信之间的接口设计是这个 APP 开发的核心任务。由于时间有限，界面较为简陋，没有进一步美化。



（截图自 Android Studio 开发界面）

每按下按键，APP 都会根据 IP 地址和端口号尝试建立 TCP 连接，如果成功，便发送相应的字符（通信协议与键盘控制小车相同），成功后关闭连接。

5.3 操作步骤

1. 网络配置与建立 ssh 连接
2. 主机运行 roscore
3. 树莓派运行：

```
$roslaunch duckietown joystick.launch veh:=duckiebot1
```

4. 主机根据 ifconfig 中的 IP 地址修改 server.py 脚本中的地址，设置端口号，运行

在安卓手机上安装应用，保证手机连入网络，输入 IP 地址以及端口号，按下按键发送指令，按下 QUIT 服务器退出。

6 图像识别

6.1 提取关键信息

在获取了摄像头回传图像的基础上，把图像信息映射到 hsv 色彩空间，通过设定参数范围将红色块与黑色块提取出来，再从中取出边缘。随着环境亮度、摄像头角度等外界条件的改变，图像信息也会有偏差，需要对参数不断修正才能满足要求。而且，提取出来的信息还有很多干扰信息或是无用信息，需要进一步的判断进行过滤。

6.2 箭头的判断

遍历所有红色边缘，取出中心点在图片中心点附近一定范围内的、边缘点数最大的那个红色边缘，便是小车路径前方的箭头。这样就过滤掉了其他路径也可能被看到的箭头以及环境中红色的干扰物。

进一步做箭头方向的判断：将所有边缘点坐标取平均，得到中心点。如果箭头中心点一侧的边缘点数比另一侧多足够多，那么箭头朝向该方向；如果两侧的边缘点数相差足够小，那么箭头是停止标志；两者都不是，那么再进行一次判断。

由于存在光源频闪以及色块提取对环境变化无法做到足够鲁棒，所以对箭头方向的判断要进行多次后进行投票再做出最后决定，向小车发出相应的指令。

以上描述的判断参数均通过实际测试中不断调整确定的，而且可能会随着环境变化做进一步修正。下文中提到的参数也是如此。

6.3 边线的判断

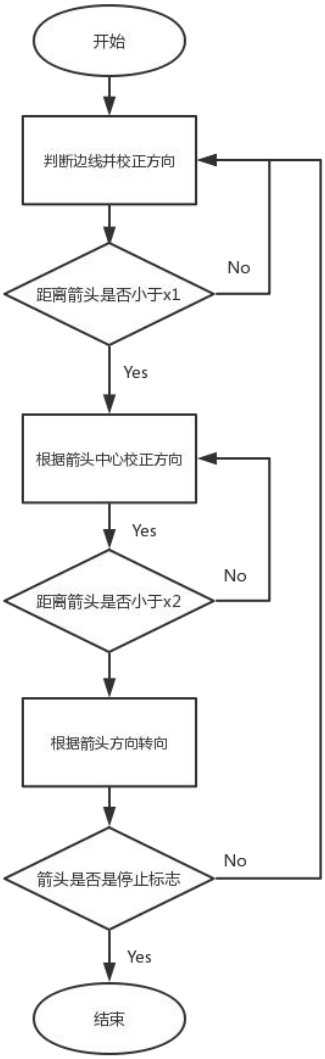
遍历所有黑色边缘，取出中心点在图片中心点以下的、边缘点数最大的那个黑色边缘，便是小车校正方向的参考边线。这样就过滤掉了其他路径也可能被看到的边线以及环境中深色的干扰物。

进一步做调整方向的判断：如果边线中心点在图片中心点的左侧，而边线边缘中横坐标最大的点在右侧或者距离中心点不远，那么需要右转；左转与之类似；两者都不是，那么就继续前进。

7 控制算法

7.1 程序框图

（下图中 x1、x2 为距离参数）



相当于整个过程有三个状态：边线校正、箭头校正、根据箭头转向或停止。其中，箭头校正的设置是因为当小车走到岔路口时，不仅无法看到之前所参照的边线，还容易被岔路中其他路径的边线干扰，产生错误的转向。所以此时放弃边线校正的方式，而是转为根据偏离箭头中心点的距离来对小车的方向进行校正，直到小车距离箭头足够近可以转向(程序中通过箭头边缘点数的多少来判断与箭头的距离）。

7.2 小车运动的控制

7.2.1 旋转 90 度的控制

由于程序会不断根据新接收到的图像信息向小车发送相应的指令，而且之后的指令会将上一个指令覆盖，从而每个指令执行的时间都是有限的。为了保证小车可以按照箭头信息顺利完成左右转，单独定义左右转 90 度的控制指令，且在发送之后，通过 time.sleep()函数对程序进行阻塞，等待小车完成转向，阻塞时间由实验获得。

7.2.2 应对处理延迟的鲁棒性控制

由于图像回传、图像处理计算使得控制信号发送与接收图像之间有一定的延迟，容易产生误判从而使小车失控，所以要进行调节使得图像回传、图像处理与发送指令的帧率一致。在这里我使用的方法是：取一定时间长度为一个周期，在一个周期中，只可以有效地发送一次控制指令，其他时刻自动发送停止指令，使小车在原地等待。这样，有了更多的缓冲时间，即使网络状态不稳定或者运算时间较长小车也可以做出正确的反应。当然，这样做也要付出小车行走卡顿、速度较慢的代价，但鲁棒性得到大大提高。经测试，效果良好。

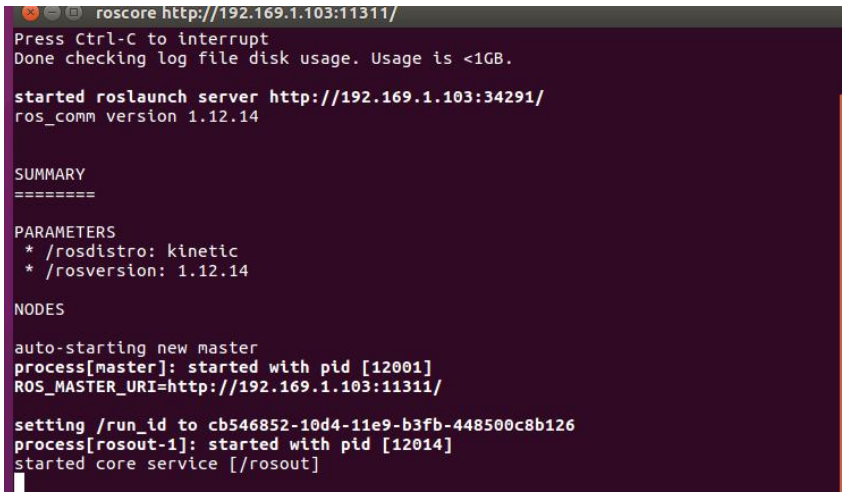
7.3 操作步骤

1. 网络配置与建立 ssh 连接
2. 主机运行 roscore
3. 树莓派运行：

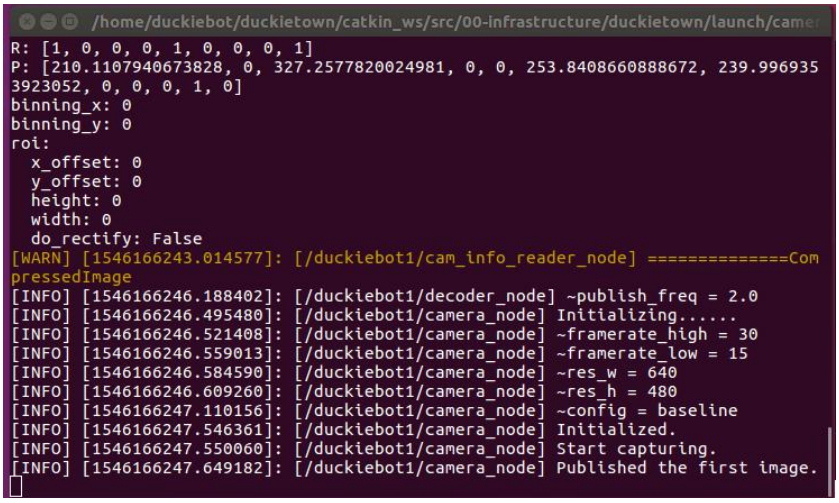
```
$roslaunch duckietown joystick.launch veh:=duckiebot1
$roslaunch duckietown camera.launch veh:=duckiebot10 raw:=true rect:=true freq:=10
```

4. 主机运行 AutoNavigation.py 脚本，可以看到实时回传的图像以及相应的处理图像，将小车放在轨迹上就会自动开始前进。
5. 小车到达停止标志后，程序退出。

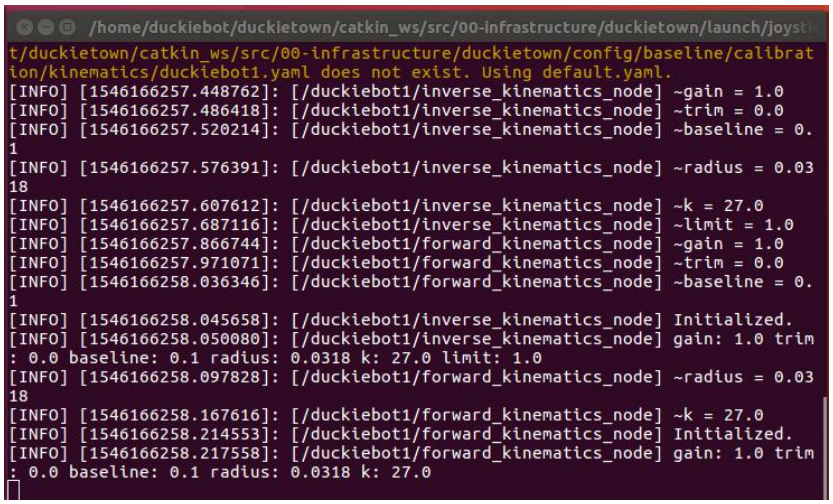
8 运行效果



主机运行 roscore 界面

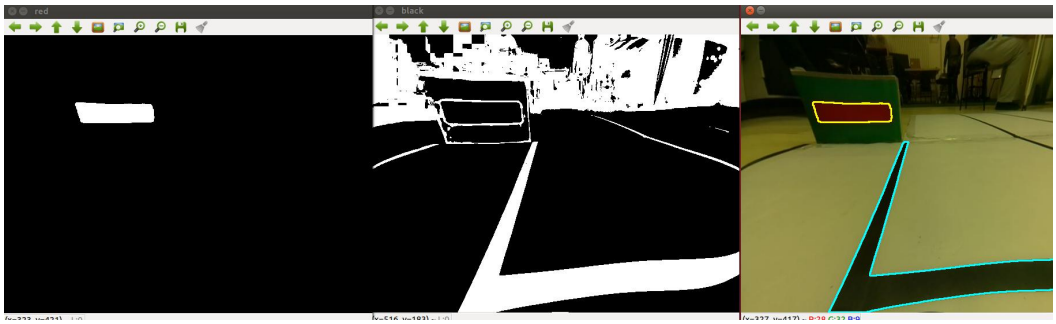


树莓派运行摄像头

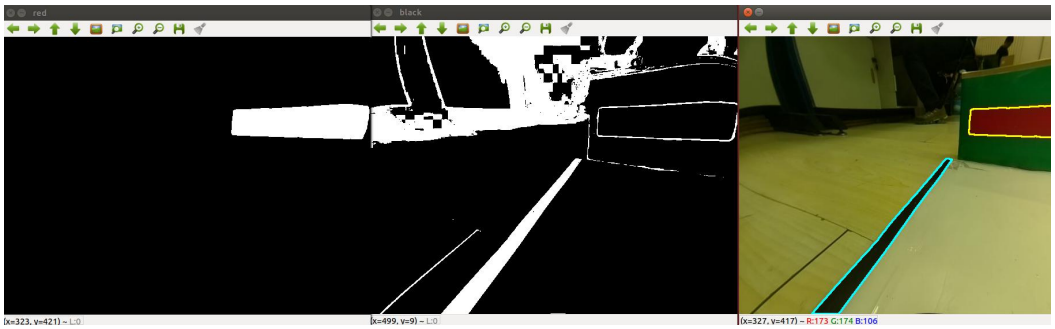


树莓派运行运动控制

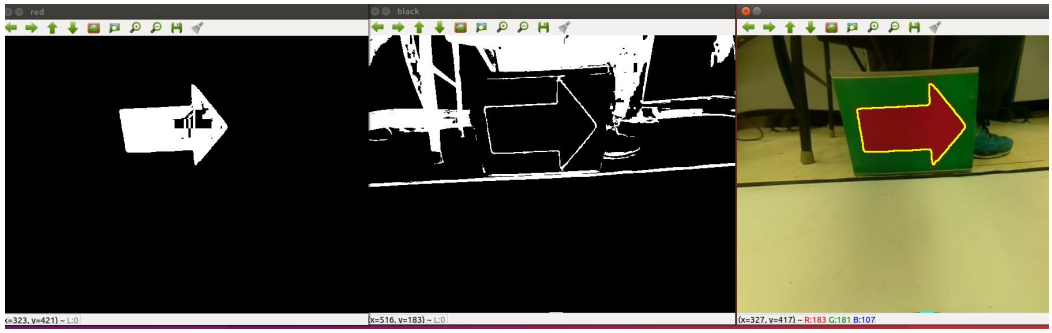
（以下图片中左图为图片中红色区域，中图为图片中黑色区域，右图为处理后画面）
（黄色圈出箭头，蓝色圈出边线）



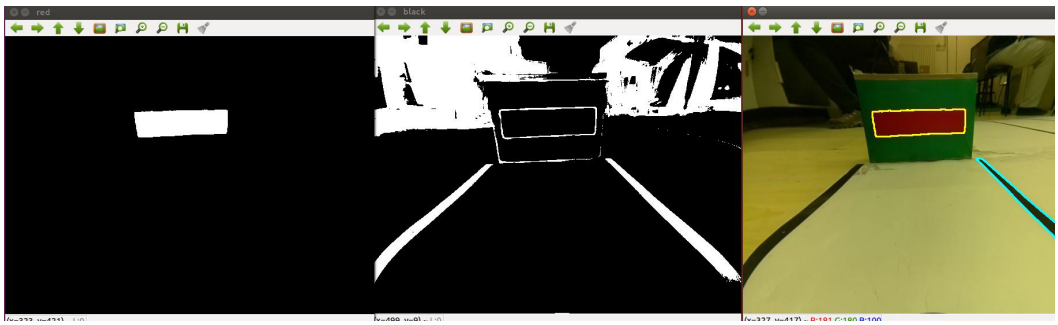
转弯处边线识别



直行时边线识别



箭头识别



停止标志识别

9 问题与解决

(1) 控制小车运动的消息类型是 duckietown 包中规定的特定消息格式，而我的主机中没有。我曾尝试在 duckietown 官网以及官方的 Github 中下载相应的包，但没有找到。

解决：在助教的帮助下我拷贝了一份 duckietown 中的消息文件到我的电脑中，并在我的工作空间进行了 rosmake，便可正常使用。另外，助教也帮我尝试自定义了一个新的消息类型，使其格式与 duckietown 中的兼容，经测试，也可以进行消息发送。两个方法均是可行的。

(2) 最初判断箭头时我使用的方法是用 Canny 算法提取所有边缘再去寻找边缘中心点颜色为红色的那个。这个算法用我的手机上显示的图片测试时效果非常好，但在实际场地中由于光线亮度、环境中的干扰物体等等导致鲁棒性变差。

解决：首先我尝试在执行 Canny 算法前进行滤波处理，但效果没有改善。经过各种尝试后决定修改算法，改用色块提取再边缘检测的方法，效果有了明显提高，而且也不需要后续再判断中心点颜色是否为红色。

(3) 其他问题在报告中已经有所提及，不再赘述。

10 小结

这次大作业对于我来说是一个全新的尝试，在时间紧张、毫无基础的情况下完成对于我来说是一个很大的挑战，也给我带来了不少成就感。我这个学期没有选数图，opencv 等图像处理方法都是通过自己查资料一点点学起，Linux 网络配置、ROS 通信的调试更是在刚刚上手时毫无头绪，在助教的帮助和一次次是试错后才逐步探索出来。环境因素对小车控制的影响也给参数的修正带来了重重困难，前几次调试场地开放时我都全程参加，一次次反复测试才逐步将算法完善。也因此我最早完成了任务要求。

大作业要求之外的 Android APP 开发让我进一步巩固了移动端应用的开发，也熟悉了 socket 通信，让大作业更有计网的味道。我以为有之前的移动端开发经验可以很快拿下，但实际开发才发现移动端网络通信开发远比想像中复杂，用了两天时间才完成了应用开发和 server 的搭建。这也算是这次大作业的一个额

外收获。

最后真的很感谢助教在我刚刚接触 ROS 时给我提供的帮助，让我少走了很多弯路；也一直在调试现场同我们一起解决问题，帮助我们搭建所需环境。谢谢！

11 参考文献

[1] The Duckiebook

[2] 基于 Java 的 TCP Socket 通信

https://github.com/jzj1993/JavaTcpSocket

[3] 虚拟机下桥接模式的配置

https://blog.csdn.net/qq_37941471/article/details/80639937

[4] ROS 机器人开发实践，胡春旭，机械工业出版社，2018.