第十七届全国大学生智能汽车竞赛

室外 ROS 无人车竞速赛（高教组）

技 术 手 册

关于技术手册使用授权的说明

本人完全了解全国大学生智能汽车竞赛关于保留、使用技术手册和研究论文的规定，即：参赛技术手册内容著作权归参赛者本人，比赛组委会和赞助公司可以在相关主页上收录并公开参赛作品的设计方案、技术手册、源码以及参赛模型车的视频、图像资料，并将相关内容编纂收录在组委会出版论文集中。

参赛队员签名：_____

带队教师签名：_____

日　　期：2022.12.8

目录

实验一　底盘控制实验

实验目的：

1、熟悉 ROS 话题通信；

2、为后续视觉导航提供电控支持；

3、熟悉 Twist 信息。

实验内容：

创建控制节点，向“~/car/cmd_vel”话题发布 Twist 信息。

实验仪器：

ROS 智能车、电脑。

实验原理：

修改 Twist 信息，并将其发布至“~/car/cmd_vel”话题。

实验步骤：

1. Twist.msg

Twist.msg

```
#定义空间中物体运动的线速度和角速度

#文件位置：geometry_msgs/Twist.msg

Vector3 linear

Vector3 angular
```

```
geometry_msgs/Vector3 linear

  float64 x

  float64 y
```

```
  float64 z
geometry_msgs/Vector3 angular
  float64 x
  float64 y
  float64 z
```

linear(线速度）下的 xyz 分别对应在 x、y 和 z 方向上的速度(单位是 m/s)；angular(角速度)下的 xyz 分别对应 x 轴上的翻滚、y 轴上俯仰和 z 轴上偏航的速度(单位是 rad/s)。

在智能小车中，只用到线速度 x 分量和角速度 z 分量（偏航角）

2. 代码实现

```python
#导包
import rospy
#导入 Twist 信息
from geometry_msgs.msg import Twist


#初始化控制节点
rospy.init_node('racecar_control')
```

```python
#创建发布者
pub = rospy.Publisher('~/car/cmd_vel', Twist, queue_size=5)
```

```
twist = Twist()
```

```
#设置线速度和角度
twist.linear.x = 1700
twist.linear.y = 0
twist.linear.z = 0


twist.angular.x = 0
twist.angular.y = 0
twist.angular.z = 90
while(True):
    #一直发布该信息，使智能小车以 1700 的线速度向前运动
    pub.publish(twist)
```

代码文件：control_car.py

使用方法

打开终端先运行

roslaunch racecar Run_car.launch

再打开另一个终端，到代码文件夹下

python3 control_car.py

打开电调

实验二　tf 静态坐标变换

实验目的：

实现不同坐标系下的点或者物体之间位置的转换。

实验内容：

静态坐标变换。

实验仪器：

ROS 智能车、电脑。

实验原理：

通过调用 ros 系统中封装好的包（把旋转量矩阵转换为四元数），将当前坐标进行变换。

实验步骤：

1、获取坐标参数

通过打印坐标相关信息，获取坐标的参数，其中较为常用的消息类型是：

geometry_msgs/TransformStamped 和

geometry_msgs/PointStamped,本次实验使用的是

geometry_msgs/TransformStamped,在终端 terminator 中输入 rosmsg info geometry_msgs/TransformStamped

打印输出后得到结果为:

```
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
string child_frame_id
geometry_msgs/Transform transform
  geometry_msgs/Vector3 translation
    float64 x
    float64 y
    float64 z
  geometry_msgs/Quaternion rotation
    float64 x
    float64 y
    float64 z
    float64 w
```

2、 实现静态坐标变换

在智能车工作空间下建立一个 cpp 文件以建立发布者

需要包含的头文件：

```
#include "ros/ros.h"
#include "tf2_ros/static_transform_broadcaster.h"
#include "geometry_msgs/TransformStamped.h"
#include "tf2/LinearMath/Quaternion.h"
```

具体代码：

```cpp
int main(int argc, char *argv[])
{
    setlocale(LC_ALL,"");
    // 2.初始化 ROS 节点
    ros::init(argc,argv,"static_brocast");
    // 3.创建静态坐标转换广播器
    tf2_ros::StaticTransformBroadcaster broadcaster;
    // 4.创建坐标系信息
    geometry_msgs::TransformStamped ts;
    //----设置头信息
    ts.header.seq = 100;
    ts.header.stamp = ros::Time::now();
    ts.header.frame_id = "base_link";
    //----设置子级坐标系
    ts.child_frame_id = "laser";
    //----设置子级相对于父级的偏移量
    ts.transform.translation.x = 0.2;
    ts.transform.translation.y = 0.0;
    ts.transform.translation.z = 0.5;
    //----设置四元数:将 欧拉角数据转换成四元数
    tf2::Quaternion qtn;
    qtn.setRPY(0,0,0);
    ts.transform.rotation.x = qtn.getX();
    ts.transform.rotation.y = qtn.getY();
    ts.transform.rotation.z = qtn.getZ();
    ts.transform.rotation.w = qtn.getW();
    // 5.广播器发布坐标系信息
    broadcaster.sendTransform(ts);
    ros::spin();
    return 0;
```

另外建立一个 cpp 文件用于作为接收者

需要包含的头文件:

```cpp
#include "ros/ros.h"
#include "tf2_ros/transform_listener.h"
#include "tf2_ros/buffer.h"
#include "geometry_msgs/PointStamped.h"
#include "tf2_geometry_msgs/tf2_geometry_msgs.h" //注意: 调用 transform 必须包含该头文件
```

具体代码:

```cpp
int main(int argc, char *argv[])
{
    setlocale(LC_ALL,"");
    // 2.初始化 ROS 节点
    ros::init(argc,argv,"tf_sub");
    ros::NodeHandle nh;
    // 3.创建 TF 订阅节点
    tf2_ros::Buffer buffer;
    tf2_ros::TransformListener listener(buffer);

    ros::Rate r(1);
    while (ros::ok())
    {
    // 4.生成一个坐标点(相对于子级坐标系)
        geometry_msgs::PointStamped point_laser;
        point_laser.header.frame_id = "laser";
        point_laser.header.stamp = ros::Time::now();
        point_laser.point.x = 1;
        point_laser.point.y = 2;
        point_laser.point.z = 7.3;
        // 5.转换坐标点(相对于父级坐标系)
        // 新建一个坐标点，用于接收转换结果
        //--------------使用 try 语句或休眠，否则可能由于缓存接收延迟而导致坐标转换失败------------
        try
        {
            geometry_msgs::PointStamped point_base;
            point_base = buffer.transform(point_laser,"base_link");
            ROS_INFO("转换后的数据:(%.2f,%.2f,%.2f),参考的坐标系是:%s",point_base.point.x,
                     point_base.point.y,point_base.point.z,point_base.header.frame_id.c_str());
        }
        catch(const std::exception& e)
        {
            // std::cerr << e.what() << '\n';
            ROS_INFO("程序异常.....");
        }
        r.sleep();
        ros::spinOnce();
    }
    return 0;
```

将这两个文件加入编译文件中生成可执行文件后进行调用，这样就能输出每个相对与子坐标系的坐标现在在父坐标系下的坐标了。

实验三　锥桶识别实验

实验目的：

1、熟悉视觉识别红蓝锥桶的原理；

2、掌握视觉识别锥桶的代码实现；

3、根据识别的锥桶进行视觉导航。

实验内容：

1.搭建实验环境：Python3 + OpenCV 4.6.0 + Numpy；

2.认识 HSV 颜色空间；

3.基本的图像处理算法；

4.识别锥桶并框选。

实验仪器：

ROS 智能车、摄像头、电脑 。

实验原理：

利用 OpenCV 对摄像头图像进行 HSV 颜色空间转换、二值化、腐蚀、膨胀、查看轮廓等处理，识别除出离智能车最近的红蓝色锥桶。

实验步骤：

1. 搭建实验环境

实验环境为 Python3 + OpenCV 4.6.0 + Numpy。在 Ubuntu 系统下，安装 ROS 后，上述环境均同时被安装；

OpenCV 是很强大的图像处理工具，它给我提供了很多现成的通用图像

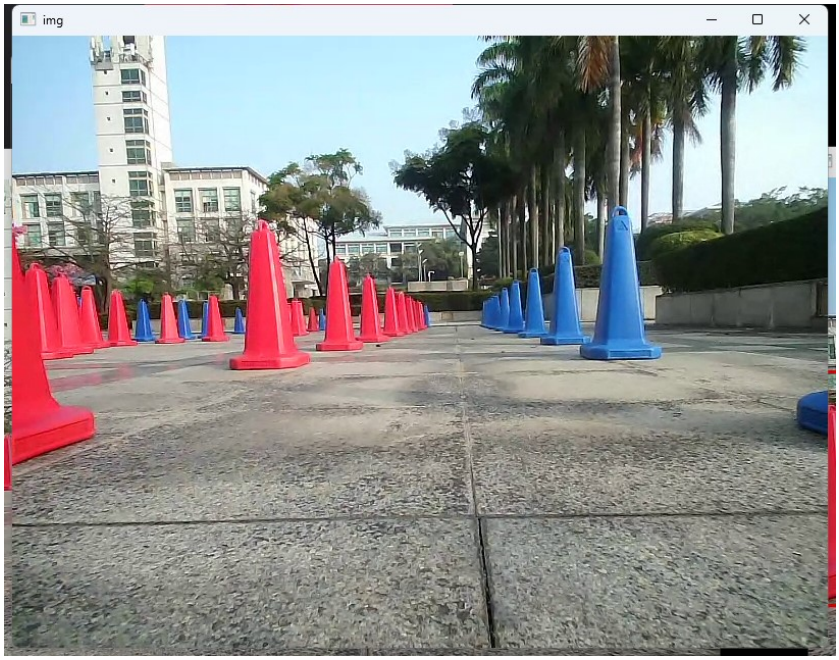
处理算法。

在 Python 环境下，使用 OpenCV 十分方便快捷。

2. 识别原理

- HSV 颜色空间

摄像头获取到的图像是 RGB 图像。



原图

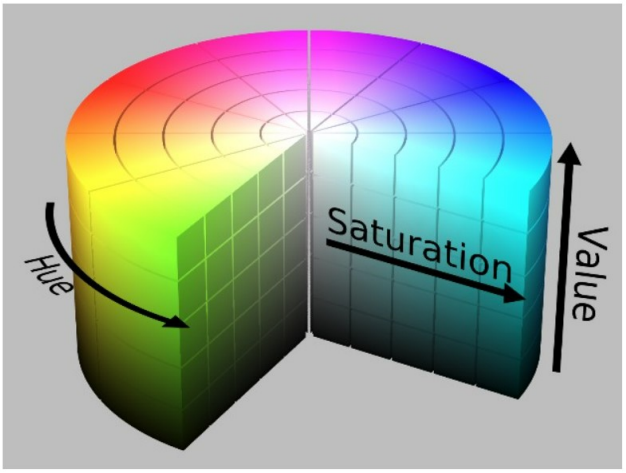
我们不难看出，红蓝锥桶最明显的特征就是其颜色——红色和蓝色。从 RGB 空间思考，显现蓝色的像素有什么特点？显然它的蓝色分量要比绿色分量和红色分量大。如果用拆分通道后的蓝色通道去其他颜色通道还有比较大的余量，说明该物体在图像中足够“蓝”。白色的物体三个通道分量都很大，显然相减之后会得到接近零的值。但是 RGB 空间分离效果不佳，受光线影响大。更好的方法是将图片转换到 HSV 颜色空间。

其中 H 代表色调、色相。取值范围 0-360° Hue = 0 表示红色 H = 120 表示绿色 H= 240 表示蓝色，连续性很好。

S 表示饱和度、色彩纯净度饱和度越高，颜色越深　0 表示纯白色　　值越大，越饱和。

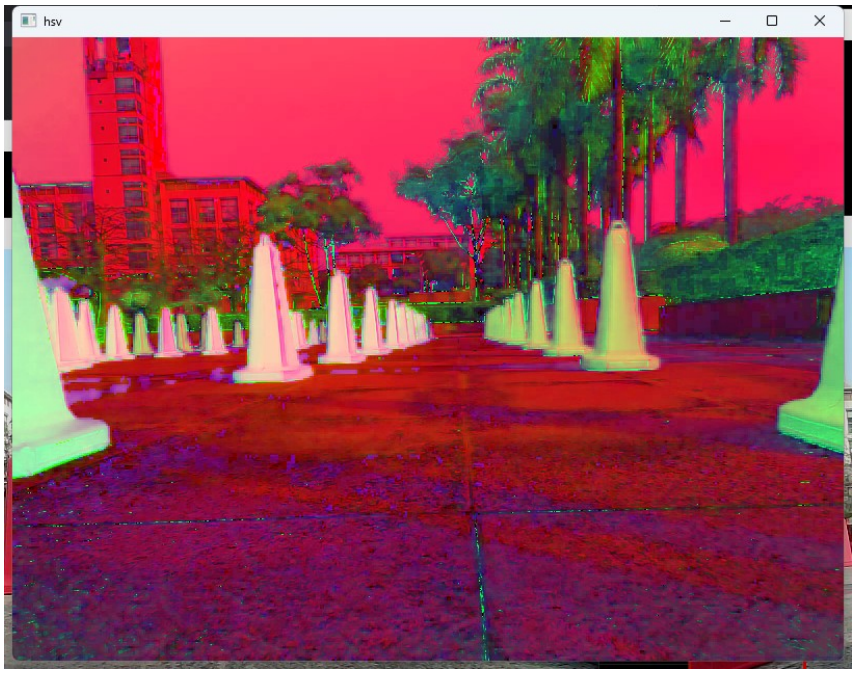
V 表示明度。取值范围 0-100% 明度越高，颜色越亮 0 表示纯黑色

用下面这个圆柱体来表示 HSV 颜色空间，圆柱体的横截面可以看做是一个极坐标系，H 用极坐标的极角表示，S 用极坐标的极轴长度表示，V 用圆柱中轴的高度表示。



代码很简单：

```
#RGB 颜色空间转换为 HSV 颜色空间
        hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
        cv2.imshow('hsv',hsv)
```



看着会比较奇怪

- inRange( )函数

函数作用：根据设定的阈值去除阈值之外的背景部分具体参数： mask = cv2.inRange (hsv, lower_red, upper_red)

```
# 提取蓝色区域的阈值设置
        blue_lower = np.array([100, 100, 100])
        blue_upper = np.array([124, 255, 255])
        #提取蓝色区域
        mask_blue = cv2.inRange(hsv.copy(), blue_lower,
blue_upper)
        cv2.imshow('mask_blue', mask_blue)
```

第一个参数：hsv 指的是原图

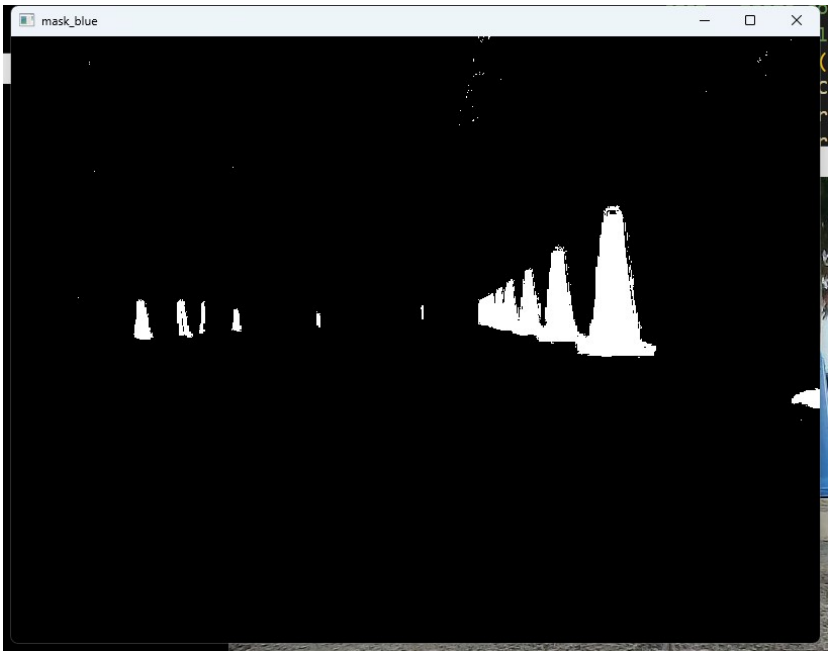
第二个参数：lower_red 指的是图像中低于这个 lower_red 的值，图像值变为 0

第三个参数：upper_red 指 的 是图像中高于这个 upper_red 的值，图像值变为 0

图象值变为 0，即变为黑色

提取蓝色锥桶为例：

```
blue_lower = np.array([100, 100, 100])
blue_upper = np.array([124, 255, 255])
mask1 = cv2.inRange(hsv.copy(), blue_lower, blue_upper)
```



已将除蓝色以外的颜色变为黑色

- 模糊处理，二值化处理

```
#模糊处理
blurred=cv2.blur(mask,(9,9))
```

cv2.imshow('blurred',blurred)

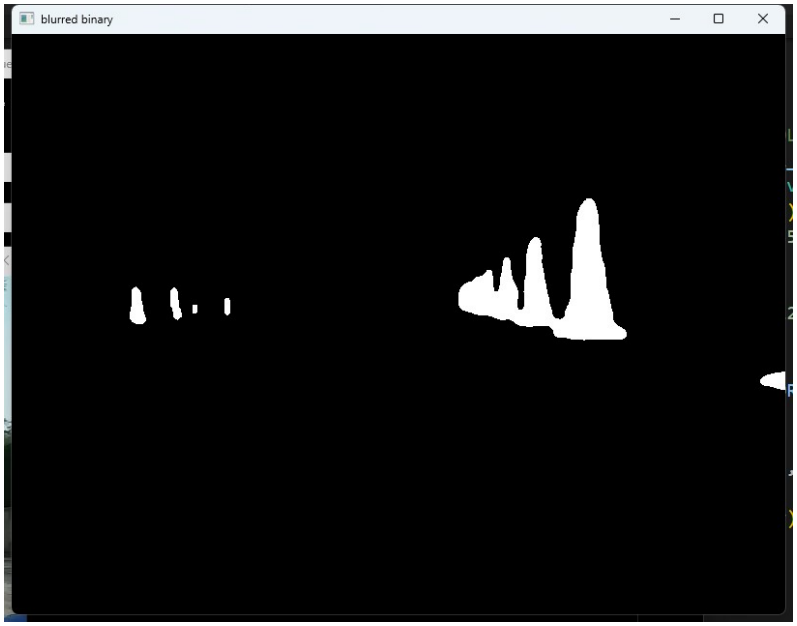
采用均值滤波，它只取内核区域下所有像素的平均值并替换中心元素。我们使用 9x9 的盒式过滤器

#二值化处理

ret,binary=cv2.threshold(blurred,127,255,cv2.THRESH_BINARY)

cv2.imshow('blurred binary',binary)

这两步使图像噪点减少，得到干净的图像

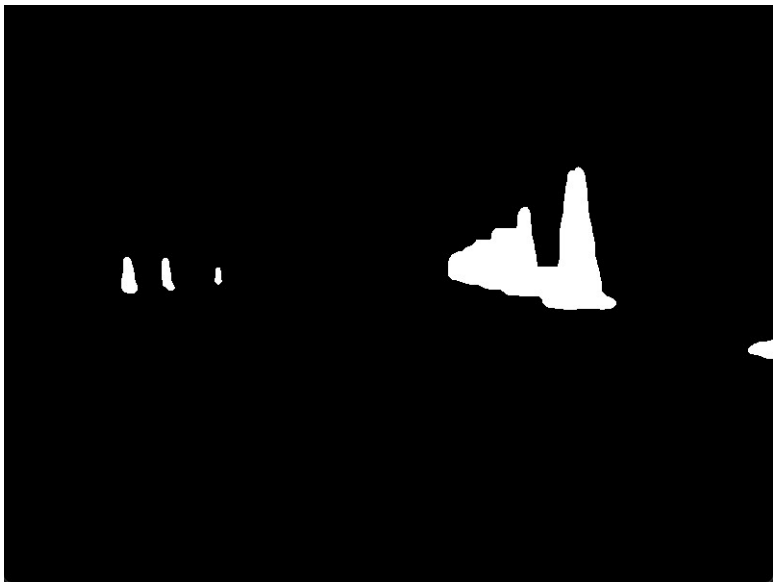


- 使区域闭合无空隙，腐蚀和膨胀

#使区域闭合无空隙

kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (21, 7))

closed = cv2.morphologyEx(binary, cv2.MORPH_CLOSE, kernel)



腐蚀和膨胀

腐蚀操作将会腐蚀图像中白色像素，以此来消除小斑点，

而膨胀操作将使剩余的白色像素扩张并重新增长回去。

```
erode = cv2.erode(closed, None, iterations=4)
cv2.imshow('erode', erode)
dilate = cv2.dilate(erode, None, iterations=4)
cv2.imshow('dilate', dilate)
```

上述操作均为了去除不间断点，利于后续识别除离智能车最近的锥桶

- 寻找轮廓

利用 OpenCV 的 findContours( )函数，将轮廓提取出来

```
contours,hierarchy=cv2.findContours(dilate.copy(),
```

cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

- 遍历轮廓，利用约束条件筛选出锥桶

```
for con in contours:
    # 轮廓转换为矩形
    rect = cv2.minAreaRect(con)
    # 矩形转换为 box
    box = np.int0(cv2.boxPoints(rect))
    # 在原图画出目标区域
    h1 = max([box][0][0][1], [box][0][1][1], [box][0][2][1],
[box][0][3][1])
    h2 = min([box][0][0][1], [box][0][1][1], [box][0][2][1],
[box][0][3][1])
    l1 = max([box][0][0][0], [box][0][1][0], [box][0][2][0],
[box][0][3][0])
    l2 = min([box][0][0][0], [box][0][1][0], [box][0][2][0],
[box][0][3][0])
    #得到矩形轮廓高和宽
    h = h1 - h2
    l = l1 - l2
```

```
if h > 100 and l > 90:
    #筛选条件，只有像素块较大的轮廓才处理
    if h/l>1.1:
        #进一步处理，可知锥桶的高宽比是大于 1.1 的
        #在图像上画出矩形框
        cv2.drawContours(res, [box], -1, (0, 0, 255), 2)
```

重复以上步骤，对红桶进行处理，最后叠加再一起，识别完毕。

识别结果

为了方便后续使用，我们只选择离车最近的锥桶。

3.代码复现

代码文件：findBucket_test.py

附件：test.webm

附件为录制的图像文件

使用方法，将文件复制到某文件夹，然后在当前文件夹打开终端，输入命令：python3 findBucket_test.py

注意：如果不能成功运行，请将 test.webm

```
#获取图像文件
cap = cv2.VideoCapture("test.webm")
```

修改为绝对路径

实验四　停车点识别实验

实验目的：

1、熟悉视觉识别原理；

2、熟悉不同图像特征的筛选条件；

3、根据识别到的停车点在第二圈进行停车。

实验内容：

1.搭建实验环境：Python3 + OpenCV 4.6.0 + Numpy；

2.认识 HSV 颜色空间；

3.基本的图像处理算法；

4.确认筛选条件，识别停车点。

实验仪器：

ROS 智能车、摄像头、电脑。

实验原理：

利用 OpenCV 对摄像头图像进行 HSV 颜色空间转换、二值化、腐蚀、膨胀、查看轮廓等处理，识别停车点。

实验步骤：

在识别红蓝锥桶实验的基础上，添加新的筛选条件

注意到停车标志也是蓝色的

思路 1：将识别锥桶二值化处理后的图像减去最后得到的图像将锥桶轮廓去除，

只剩下停车标志，实践后发现鲁棒性极低，很容易误判

思路 2：只处理蓝色通道，进行模板匹配，实践后发现，鲁棒性极低

思路 3：智能车靠近停车标识时，会产生严重的畸变（长宽比改变）且在图像中元素占比大，我们可以利用这一特性，修改筛选条件，符合条件后触发停车节点

以红蓝锥桶识别实验为基础

修改筛选条件如下（需要调参，视情况而定）：

```
#添加帅选条件，高大于等于 200 个像素点，宽大于等于 500 个像素点
            if h >= 200 and l >= 500:
                #画出黄色边框
                cv2.drawContours(res, [box], -1, (0, 255, 255), 2)
```

代码文件：parking.py

附件：parking_test.webm

使用方法：打开文件夹，当前文件夹下打开终端，输入命令

python3 parking.py

注意：如果不能成功运行，请将 parking_test.webm

```
#获取图像文件
cap = cv2.VideoCapture("parking_test.webm")
```

修改为绝对路径

实验五　PID 算法实验

实验目的：

1、熟悉 PID 算法的原理；

2、掌握 PID 算法代码实现；

3、熟悉位置式 PID 与增量式 PID 的优缺点。

实验内容：

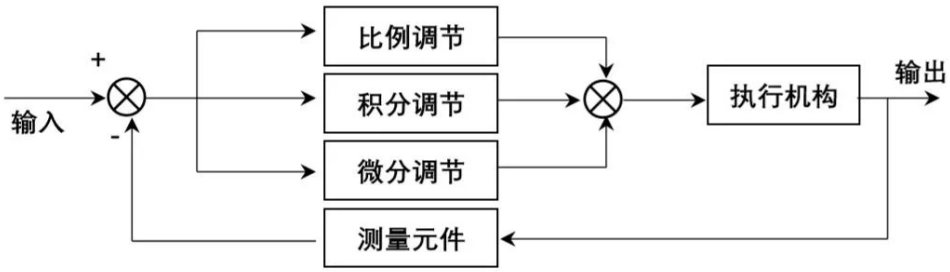
1. PID 算法的基本原理；
2. 位置式 PID 与增量式 PID；
3. PID 算法的 python 代码实现；
4. PID 算法的可视化演示。

实验仪器：

电脑。

实验原理：

PID 控制器（Proportional-Integral-Derivative Controller）是一种控制系统中常用的控制方式，它由三部分构成：比例 P（proportional）、积分 I（integral）和微分 D（derivative）。它能够控制系统达到目标状态，并使系统在目标状态附近稳定运行。



PID 算法的公式：

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt}$$

将其离散化，得到：

- 比例项：$K_p e(t) \xrightarrow{\text{离散化}} K_p e_k$
- 积分项：$K_i \int_0^{t_k} e(\tau)d\tau \xrightarrow{\text{离散化}} K_i \sum_{i=1}^{k} e(i)\Delta t$
- 微分项：$K_d \frac{de(t_k)}{dt} \xrightarrow{\text{离散化}} K_d \frac{e(k) - e(k-1)}{\Delta t}$

位置式 PID：

$$u(k) = K_p e_k + K_i \sum_{i=1}^{k} e(i)\Delta t + K_d \frac{e(k) - e(k-1)}{\Delta t}$$

求增量

$$\Delta u(k) = u(k) - u(k-1)$$

增量式 PID：

$$\Delta u(k) = K_p(e(k) - e(k-1)) + K_i e(k) + K_d\Big(e(k) - 2e(k-1) + e(k-2)\Big)$$

增量式与位置式各自的优缺点

位置式 PID 优缺点

优点：位置式 PID 是一种非递推式算法，可直接控制执行机构（如平衡小车），u(k)的值和执行机构的实际位置（如小车当前角度）是一一对应的，因此在执行机构不带积分部件的对象中可以很好应用

缺点：每次输出均与过去的状态有关，计算时要对 e(k)进行累加，运算工作量大。

增量式 PID 优缺点

优点：① 误动作时影响小，必要时可用逻辑判断的方法去掉出错数据。② 手动/自动切换时冲击小，便于实现无扰动切换。当计算机故障时，仍能保持原值。③ 算式中不需要累加。控制增量Δu(k)的确定仅与最近 3 次的采样值有关。

缺点：① 积分截断效应大，有稳态误差；② 溢出的影响大。有的被控对象用增量式则不太好；

实验步骤：

1. 搭建实验环境

实验环境为 Python3 +Matplotlib

其中 Matplotlib 是 Python 中类似 MATLAB 的绘图工具，可实现 PID 可视化功能，协助 PID 算法调参。

2. 位置式 PID 与增量式 PID 的 python 算法实现

位置式 PID

```python
# 定义位置式PID类
class PositionalPID:
    def __init__(self, kp, ki, kd):
        # 初始化比例、积分和微分系数
        self.kp = kp
        self.ki = ki
        self.kd = kd
        # 初始化误差
        self.current_error = 0
        self.add_error = 0  # 误差累计
        self.last_error = 0  # 上次误差

    def control(self, target, measured_value):
        # 计算当前误差
        self.current_error = target - measured_value

        # 输出计算
        output = self.kp * self.current_error + self.ki * self.add_error + self.kd * (
                    self.current_error - self.last_error)

        # 保存误差
        self.add_error += self.current_error
        self.last_error = self.current_error

        # 返回输出
        return output
```

增量式 PID

```python
class IncrementalPID:
    def __init__(self, kp, ki, kd):
        # 初始化比例、积分和微分系数
        self.kp = kp
        self.ki = ki
        self.kd = kd
        # 初始化误差
        self.current_error = 0
        self.last_error = 0  # 上次误差
        self.previous_error = 0  # 上上次误差

    def control(self, target, measured_value):
        # 计算当前误差
        self.current_error = target - measured_value

        # 增量计算
        increment = self.kp * (self.current_error - self.last_error) + self.ki * self.current_error + self.kd * (
                    self.current_error - 2 * self.last_error + self.previous_error)

        # 保存误差
        self.previous_error = self.last_error
        self.last_error = self.current_error

        # 返回输出
        return increment
```

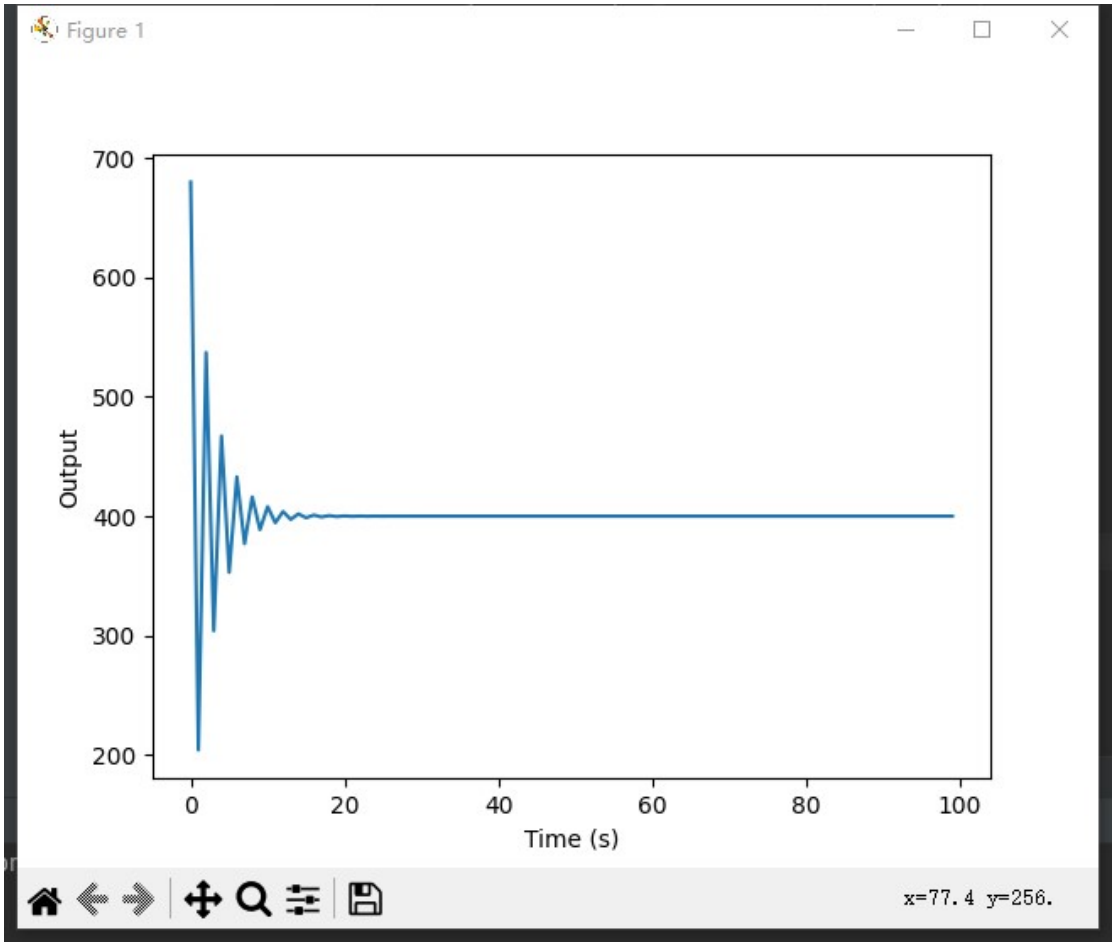
3. 利用 Matplotlib 对 PID 算法可视化（以增量式 pid 为例）

由于用 python 写的话不像是在单片机上，单片机编程有一个重要的函数中断函

数。Python 里面没有内置这个函数，所以我们用 time 库来控制函数的运行时间的间隔。下述代码是一个增量式 PID 算法可视化程序。

```python
from pid import IncrementalPID
import matplotlib.pyplot as plt

# 系统当前的实际输出
output = 0

# 系统的目标输出
target_output = 400

# 初始化可视化数据
time = []
errors = []
outputs = []

# 使用 PID 算法来控制系统
for i in range(100):
    # 计算调整量
    pid_inc = IncrementalPID(kp=0.5, ki=0.1, kd=0.1)
    adjustment = pid_inc.control(target_output, output)

    # 应用调整量，并更新系统的输出
    output += adjustment

    # 保存可视化数据
    time.append(i)
    errors.append(pid_inc.current_error)
    outputs.append(output)

# 绘制输出曲线
plt.plot(time, outputs)
plt.xlabel("Time (s)")
plt.ylabel("Output")
plt.show()
```

当 kp=1，ki=0.5，kd=0.2 时编译输出下述图像



通过可视化可以清晰观察目标数到达期望值过程中产生了震荡，我们需要调参来使目标值顺滑地到达期望值，下面是通过调整参数后的图像。

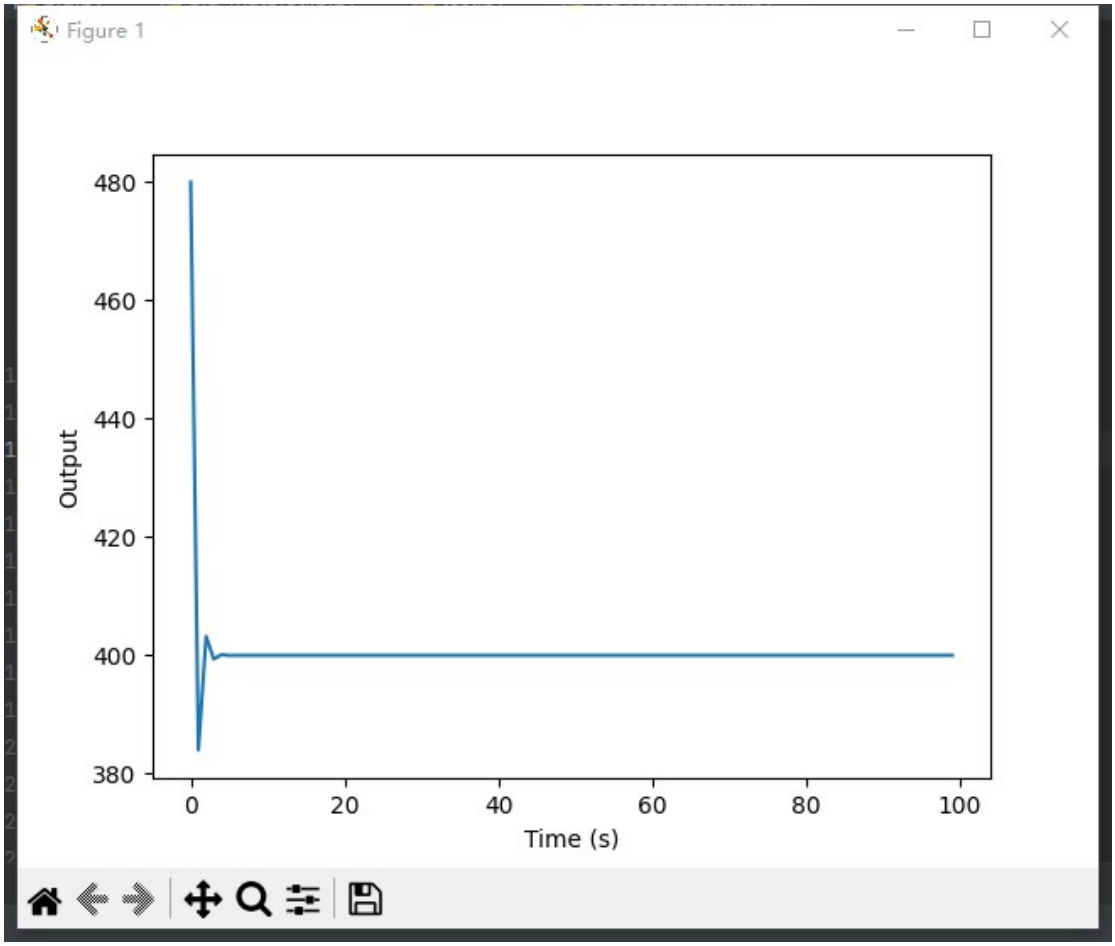
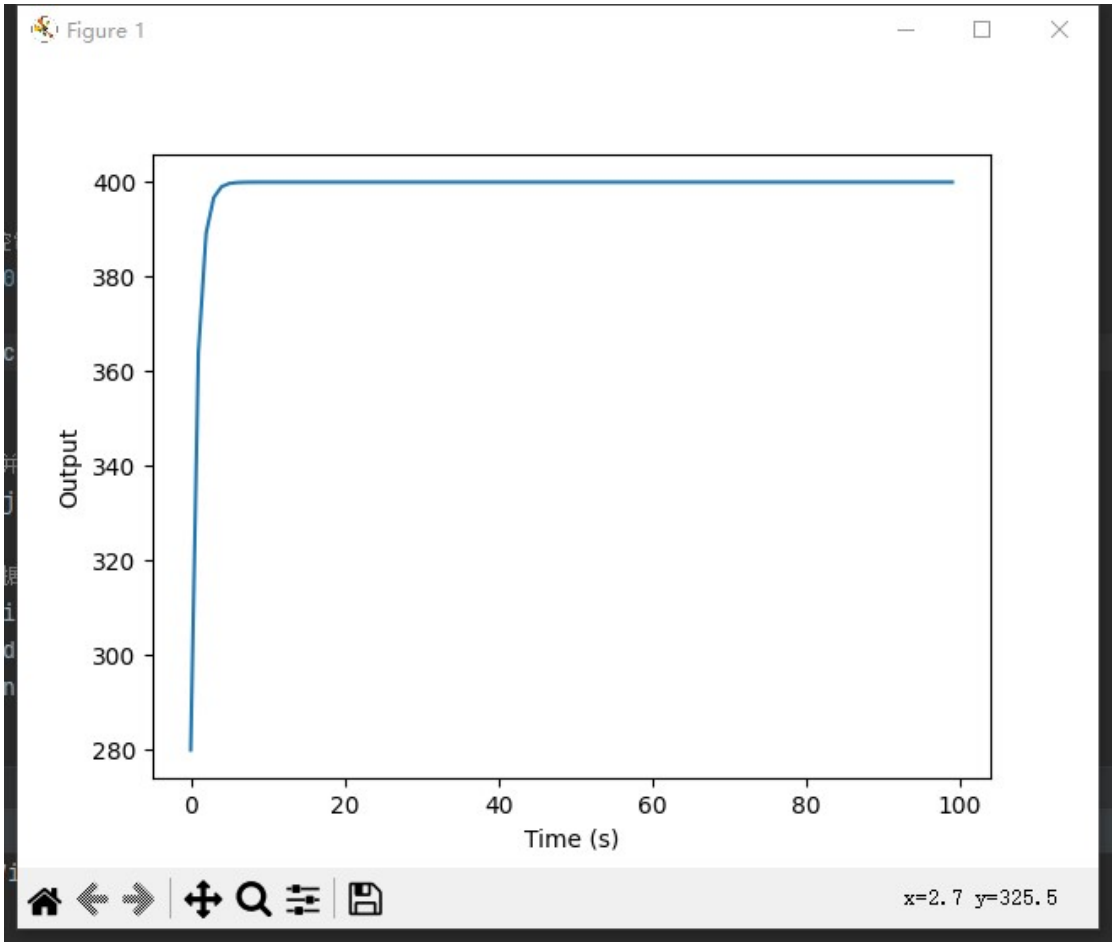


Figure 1
480
460
440
420
400
380
Output
0
20
40
60
80
100
Time (s)



代码文件：pid.py,

PIDVisualization.py

使用方法：使用 Visual Studio Code 或 PyCharm 将两个文件放置在同一工程下，直接编译运行，在此过程可通过调整期望值（参数 target_output）和 PID 算法的三个参数进行编译输出可视化，实现 PID 算法调参。

4. 熟悉 PID 算法在小车上实际应用时参数的调节方法

·kp

该参数在 PID 算法中的作用最主要体现在反应速度方面，提高该参数的值可

以显著提高小车的反应速度，同时也能够起到消除静态误差的作用。然而，该参数过大时，将会造成小车在寻路时明显摆动。

当发现小车反应速度过慢时，应增大该参数；当发现小车左右摆动明显时，应减小该参数。

·ki

该参数在 PID 算法中起到消除静态误差的作用，主要在转弯时体现。但在实际应用中，累积的误差可能导致小车反应速度大幅降低，因此，在实际应用中，必须对积分值进行限幅，减小累积误差对反应速度的影响。

当发现小车转弯时容易碰撞外圈时，应增大该参数；转弯时碰撞内圈，或是转弯后反应速度明显降低，则应减小该参数。

·kd

该参数在 PID 算法中主要起到减少抖动的作用，在小车左右晃动明显时，可以适当增大该参数；当小车出现转弯困难等现象时，可以适当减小该参数。

实验六　基于视觉的寻路实验

实验目的：

1、熟悉视觉识别红蓝锥桶的原理；

2、熟悉机器人的底盘控制；

3、将地盘控制实验与锥桶识别有机结合，用于机器人自主运动；

4、掌握 PID 算法的实机运用

实验内容：

1.寻路的思路

2.代码实现

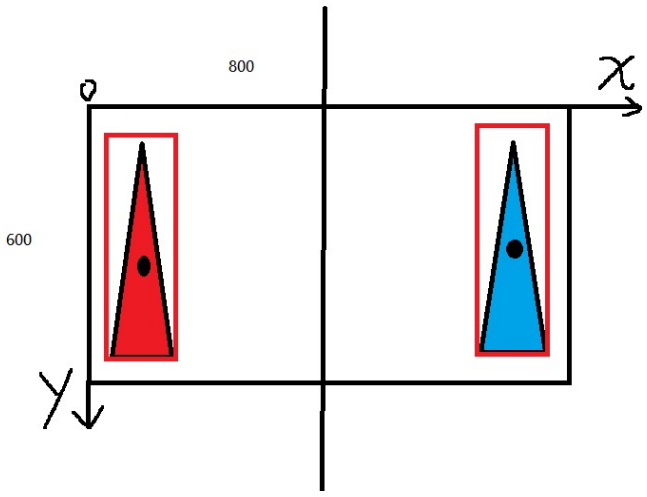
实验仪器：

ROS 智能车、摄像头、电脑

实验原理：

基于锥桶识别实验，我们可以得到离智能车最近的红蓝色锥桶，可以计算出它们的中心坐标。

摄像头获取到的图像尺寸为 800×600



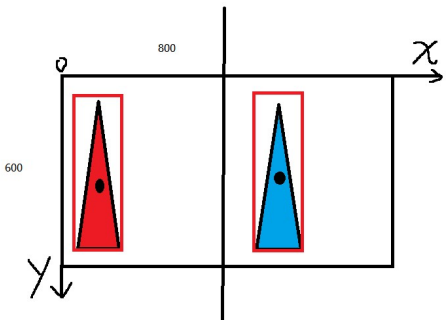
在 OpenCV 中坐标系的坐标原点在图像的左上角，X 轴为图像矩形的上面那条水平线，从左往右；Y 轴为图像矩形左边的那条垂直线，从上往下。

根据计算 x 方向上的红蓝锥桶中点坐标，假设红桶中点为 P1,蓝桶中点为 P2,我们可以进一步算出 P1、P2 的中点 P，再去与图像 x 轴中点作比较。

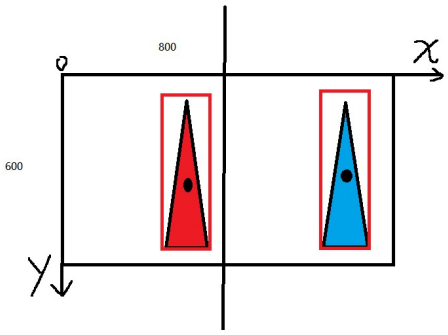
P = (P1+P2)/2　　图像 x 轴中点值为 400

如果 P>400 则智能车需要右偏，才能走在两桶之间

如果 P<400 则智能车需要左偏，才能走在两桶之间



P<400 车左偏



P>400 车右偏

实验步骤：

```
#获取图像文件
cap = cv2.VideoCapture("test.webm")
# cap = cv2.VideoCapture(0)  #实机时使用
```

1．定义图像处理函数，返回距离最近的锥桶的中心坐标

```
#图像处理，并返回距离最近的锥桶的坐标
#color = 1 时为红桶，color=2 时为蓝桶
#这里的 img 用于画图
def process(img, color):
    global p_blue_last, p_red_last
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    if color == 2:
        #提取蓝色
        mask_blue = cv2.inRange(hsv.copy(), blue_lower,
blue_upper)
        #模糊
        blurred = cv2.blur(mask_blue, (9, 9))
        p = p_blue_last
    else:
        mask_red = cv2.inRange(hsv.copy(), red_lower,
red_upper)
        blurred = cv2.blur(mask_red, (9, 9))
        p = p_red_last
    # 模糊
```

```
    # blurred = cv2.blur(mask, (9, 9))
    # 二值化
    _, binary = cv2.threshold(blurred, 127, 255,
cv2.THRESH_BINARY)
    # cv2.imshow('blurred binary', binary)
    # 使区域闭合无空隙
    closed = cv2.morphologyEx(binary, cv2.MORPH_CLOSE,
kernel)
    # cv2.imshow('closed', closed)

    erode = cv2.erode(closed, None, iterations=4)
    # cv2.imshow('erode', erode)
    dilate = cv2.dilate(erode, None, iterations=4)
    # cv2.imshow('dilate', dilate)
    # 查找轮廓
    contours, hierarchy = cv2.findContours(
        dilate, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    # print('轮廓个数：', len(contours))

    s_max = 0
    i = 0 #用于记录是否找到 P 点
```

```
    for con in contours:
        # 轮廓转换为矩形
        rect = cv2.minAreaRect(con)
        # 矩形转换为 box
        box = np.int0(cv2.boxPoints(rect))
        # 在原图画出目标区域
        h1 = max([box][0][0][1], [box][0][1][1],
[box][0][2][1], [box][0][3][1])
        h2 = min([box][0][0][1], [box][0][1][1],
[box][0][2][1], [box][0][3][1])
        l1 = max([box][0][0][0], [box][0][1][0],
[box][0][2][0], [box][0][3][0])
        l2 = min([box][0][0][0], [box][0][1][0],
[box][0][2][0], [box][0][3][0])

        h = h1 - h2
        l = l1 - l2
        if h > 50 and l > 50:
            if h/l>1.1:
                if h*l >= s_max: #找出面积最大的一块
                    s_max = h*l
```

```
                    #找到了
                    p = (l1 + l2)/2 # 获取中点(准确来说是 x
轴的)
                    i += 1
                    #用于记录当前 p 值,防止下一帧识别不到 p 点
                    if color == 1:
                        p_red_last = p
                    else:
                        p_blue_last = p
                cv2.drawContours(img, [box], -1, (0, 0,
255), 2)
        if i == 0 and color == 2: #说明没有找到合适的蓝桶
            p = 1000
        # cv2.drawContours(img, [box], -1, (0, 0, 255), 2)

    return img, p
```

2. 调用 PID 控制器，并得出偏差

```
from pid import IncrementalPID
```

初始化实例对象，设置 kp,ki,kd

```
pid_inc = IncrementalPID(kp=2, ki=0.5, kd=0.2)
```

调用 process（）函数两次，分别得到红桶蓝桶的中心坐标

```
img_red, p_red =  process(img, 1)
img_blue, p_blue = process(img, 2)
```

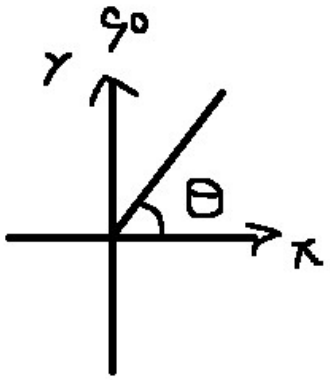
取红桶蓝桶中心坐标连线的中点，减去 400，得出转向偏差

```
value = (p_blue+p_red)/2 - 400
```

将偏差放进 PID 控制器，让 PID 控制器计算出调整量。Control 的第一个参数为目标值，我们希望智能车行驶在道路中央，所以我们希望目标值为 0，第二个参数为测量量。

```
adjustment = pid_inc.control(0, value)
```

将调整量转换成转向角度，智能车的转向角度定义如下：




```
# 转换成角度
            turn_bias = adjustment/400*90
            #turn_bias 为负时，右转，为正时，左转
            control_turn =  90 + turn_bias
```

进阶：

我们只需要将控制量通过“~/car/cmd_vel”话题发布出去即可实现寻路。

3. 代码复现

代码文件：visualDrivingDemo.py

drivingOnCar.py

附件：test.webm

visualDrivingDemo.py 这个文件是用来调试时使用的，我们可以利用图像直观地看到控制器输出的调试效果。

drivingOnCar.py 这文件是完整的视觉驾驶代码，可以部署在智能车上运行。

附件为录制的图像文件

visualDrivingDemo.py 使用方法，将文件复制到某文件夹，然后在当前文件夹打开终端，输入命令：python3 visualDrivingDemo.py

注意：如果不能成功运行，请将 `test.webm`

```
#获取图像文件
cap = cv2.VideoCapture("test.webm")
```

修改为绝对路径

drivingOnCar.py 使用方法，将文件复制到某文件夹，现在终端打开底盘控制节点:

roslaunch racecar Run_car.launch

然后在 drivingOnCar.py 所在文件夹打开终端，输入 python3 drivingOnCar.py 即可运行。按 q 键结束运行。

实验七　激光雷达里程计信息获取

实验目的：

1、熟练掌握 rf2o_laser_odometry 功能包的使用；

2、在配置好激光雷达的基础上，获取激光雷达里程计信息。

实验内容：

1. 下载激光里程计源码并修改；

2. 配置参数并运行；

3. 了解 rf2o_laser_odometry 功能包的重要参数。

实验仪器：

ROS 智能车、激光雷达。

实验原理：

通过算法将激光雷达传入的距离信息转化为里程计信息。

实验步骤：

1. 下载激光里程计源码并修改

具体步骤如下：

- 在终端处输入命令：

  git clone https://github.com/MAPIRlab/rf2o_laser_odometry.git

在 github 上下载源码，下载的源码不能直接使用，需要做一些修改：

在 CLaserOdometry2DNode.cpp 文件中

第 75 行改为：

pn.param<std::string>("laser_scan_topic",laser_scan_topic,"/scan");

使程序订阅激光雷达话题

第 512 行改为：

tf_listener.waitForTransform(base_frame_id,"/laser_link",ros::Time(),ros::Duration(5.0));

tf_listener.lookupTransform(base_frame_id, "/laser_link", ros::Time(0), transform);

使程序能正常订阅到/tf 话题，不会因为订阅不到而报错

第 221 行 if (publish_tf)前添加：

publish_tf=1;

在 CLaserOdometry2D.cpp 文件中

第 296 行 if (std::isfinite(dcenter) && dcenter > 0.f) 前、第 321 行

if (std::isfinite(dcenter) && dcenter > 0.f) 前添加：

if (dcenter > 0.f)

2. 编写 launch 并配置参数

- 具体程序如下：

<launch>

<node pkg="rf2o_laser_odometry" type="rf2o_laser_odometry_node"

```
name="rf2o_laser_odometry" output="screen">
        <param name="odom_topic" value="/odom_rf2o" />
        <param name="laser_scan_topic" value="/scan"/>
        <param name="base_frame_id" value="/base_link"/>
        <param name="odom_frame_id" value="/odom" />
        <param name="publish_tf" value="false" />
        <param name="init_pose_from_topic"
value="/base_pose_ground_truth" />
        <param name="freq" value="6.0"/>
        <param name="verbose" value="true" />
    </node>
</launch>
```

3. 运行激光里程计

该步骤需先配置好激光雷达，使其能正常传入测量值。

- 先运行激光雷达，获取其传入的距离信息；
- 运行编写好的 launch 文件；
- 获取到激光里程计信息，可在 rviz 中查看其可视化图像
- 打开 rviz 界面，添加 rf2o_laser_odometry,即可查看到获取到里程计信息。

结果如上图，红色箭头为里程计信息，控制小车运动即可看到里程计变化

4. 了解 rf2o_laser_odometry 的重要参数

laser_scan_topic(string,default:”laser_scan”)，订阅的雷达话题

odom_topic(string,default:”odom”)，发布里程计信息的话题

base_frame_id(string,defaule:”base_link”)，小车底盘坐标

odom_frame_id(string,defaule:”odom”)，里程计坐标系

freq(double)，发布信息频率

实验八　激光 SLAM 建图实验

实验目的：

1、了解采用 gmapping 算法进行建图的原理；

2、掌握 slam_gmapping 中各参数的含义。

实验内容：

1、简单了解 gmapping 算法的原理；

2、编写 launch 文件配置 slam_gmapping 功能包的参数并运行；

3、了解 slam_gmapping 的参数，掌握其重要参数与调节方法。

实验仪器：

ROS 智能车、激光雷达。

实验原理：

利用 ROS 中的 slam_gmapping 功能包分析激光雷达传入的信息并进行建图。

实验步骤：

1. 了解 gmapping 算法

在 SLAM 算法中，我们需要采用机器人获取到的信息，对机器人的位姿及地图状况进行估计。gmapping 算法是基于粒子滤波的，利用 2D 激光雷达传入的信息完成二维栅格地图构建的 SLAM 算法，其大致流程为：

- 利用上一次激光雷达获取的信息估计当前机器人位姿；
- 根据激光雷达传入的数值计算粒子权重；

- 重采样并更新地图；
- 循环。

在 ROS 中，gmapping 算法被封装在 slam_gmapping 功能包中，通过调用和传参即可实现地图构建，并可在 rviz 中看到其可视化显示。

2. 在 ROS 中使用 gmapping 算法

在 ROS 中，可提供运行 slam_gmapping 节点运行 gmapping 算法，该节点订阅话题为：

- tf，用于各种坐标系变换；
- scan，用于获取激光雷达数据；

发布话题为：

- map_metadata；
- map；
- ~entropy；

其中地图数据通过 map 发布。

下面是使用 slam_gmapping 功能包的具体方式：

- 首先配置好各传感器，包括激光雷达、编码器、IMU 等，使其能正常通过 USB 接口向电脑传输测量值，并可以提供里程计数据；
- 编写一个 launch 文件，在其中加入激光雷达、编码器和 IMU 等传感器的启动节点，同时加入 slam_gmapping 节点，配置参数如下：

<nodepkg=”gmapping” type=”slam_gmapping” name=”
slam_gmapping” output=”screen” claer_params=”ture” >

<param name=" map_frame" value=" map" />

<param name=" odom_frame" value=" odom" />

<param name=" base_frame" value=" base_link" />

<param name=" map_update_interval" value=" 0.1" />

<param name=" max_Urange" value=" 30.0" />

<param name=" map_MaxRange" value=" 20.0" />

<param name=" sigma" value=" 0.05" />

<param name=" kernelSize" value=" 1" />

<param name=" lstep" value=" 0.05" />

<param name=" astep" value=" 0.05" />

<param name=" iterations" value=" 5" />

<param name=" lsigma" value=" 0.075" />

<param name=" ogain" value=" 3.0" />

<param name=" lskip" value=" 0" />

<param name=" srr" value=" 0.01" />

<param name=" srt" value=" 0.02" />

<param name=" str" value=" 0.01" />

<param name=" stt" value=" 0.02" />

<param name=" linearUpdate" value=" 0.5" />

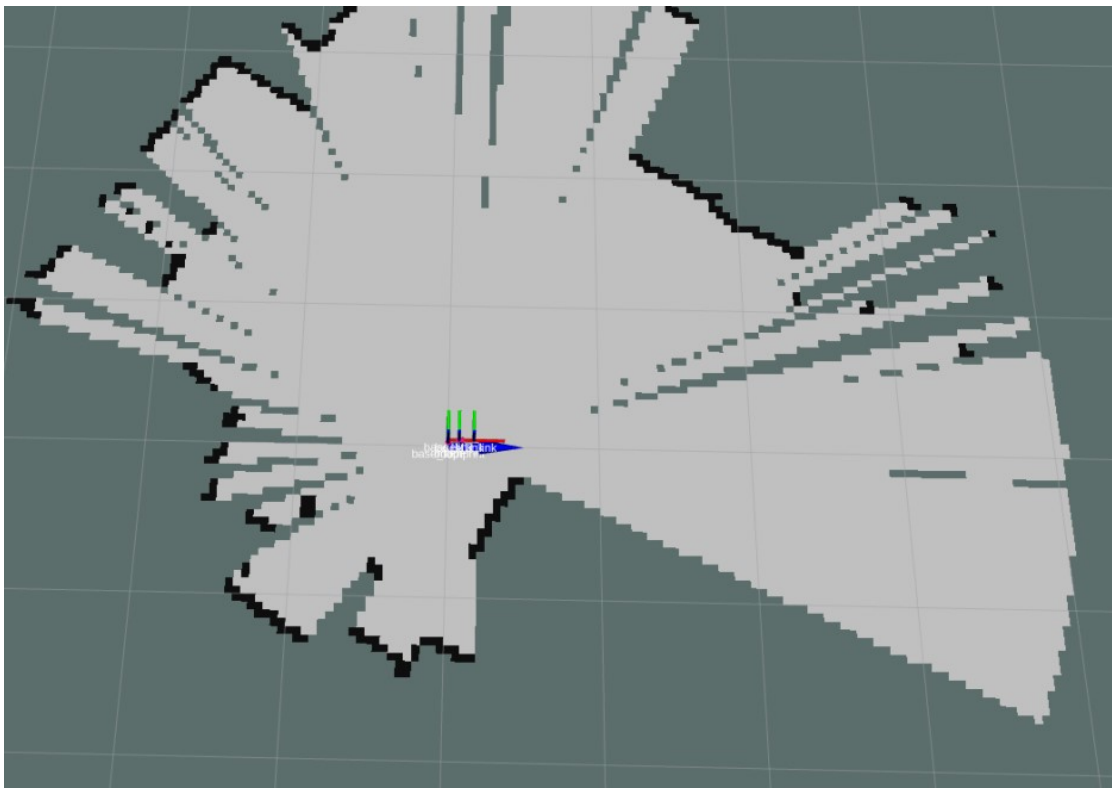
<param name=" angularUpdate" value=" 0.436" />

<param name=" temporalUpdate" value=" -1.0" />

<param name=" resampleThreshold" value=" 0.5" />

```
    <param name="particles"  value="30"/>
    <param name="xmin"  value="-50"/>
    <param name="ymin"  value="-50"/>
    <param name="xmax"  value="50"/>
    <param name="ymax"  value="50"/>
    <param name="delta"  value="0.05"/>
    <param name="llsamplerange"  value="0.01"/>
    <param name="llsamplestep"  value="0.01"/>
    <param name="lasamplerange"  value="0.005"/>
    <param name="lasamplestep"  value="0.005"/>
    <remap from="/scan"  to="/scan"/>
  </node>
```

- 随后运行该 launch 文件，即可开始建图，同时可在 rviz 中查看其可视化图像，具体方式如下：
- 在终端处输入 rviz；
- 添加 map，并将其订阅的话题设置为 map；
- 地图显示成功，可以添加 tf 查看当前小车位姿，添加 odom 显示里程计。



运行结果

3. 了解 ROS 中 slam_gmapping 功能包各参数的意义

throttle_scans (int, default: 1)，处理的扫描数据门限，默认每次处理 1 个扫描数据

base_frame (string, default:"base_link")，机器人底盘坐标系

map_frame (string, default:"map")，地图坐标系

odom_frame (string, default:"odom")，里程计坐标系

maxRange (float)，传感器效范围。

maxUrange(float)，传感器最大范围

map_update_interval (float, default: 5.0)，地图更新频率

maxUrange (float, default: 80.0)，探测最大可用范围，即光束能到达的范围。

sigma (float, default: 0.05)，endpoint 匹配标准差

kernelSize (int, default: 1)，用于查找对应的 kernel size

lstep (float, default: 0.05)，平移优化步长

astep (float, default: 0.05)，旋转优化步长

iterations (int, default: 5)，扫描匹配迭代步数

lsigma (float, default: 0.075)，用于扫描匹配概率的激光标准差

ogain (float, default: 3.0)，似然估计为平滑重采样影响使用的 gain

lskip (int, default: 0)，每次扫描跳过的光束数.

minimumScore (float, default: 0.0)，激光雷达信息置信度，越高则越依赖激光雷达进行建图

srr (float, default: 0.1)，平移时里程误差作为平移函数

srt (float, default: 0.2)，平移时的里程误差作为旋转函数

str (float, default: 0.1)，旋转时的里程误差作为平移函数

stt (float, default: 0.2)，旋转时的里程误差作为旋转函数

linearUpdate (float, default: 1.0)，机器人移动某个距离处理一次扫描数据

angularUpdate (float, default: 0.5)，机器人每旋转某个角度处理一次扫描数据

temporalUpdate (float, default: -1.0)，如果最新扫描处理比更新慢，则处理 1 次扫描，为负数时候关闭基于时间的更新

resampleThreshold (float, default: 0.5)，基于重采样门限的 Neff

particles (int, default: 30)，滤波器中粒子数目

xmin (float, default: -100.0)，地图初始尺寸

ymin (float, default: -100.0)，地图初始尺寸

xmax (float, default: 100.0)，地图初始尺寸

ymax (float, default: 100.0)，地图初始尺寸

delta (float, default: 0.05)，地图分辨

llsamplerange (float, default: 0.01)，于似然计算的平移采样距离

llsamplestep (float, default: 0.01)，用于似然计算的平移采样步长

lasamplerange (float, default: 0.005)，用于似然计算的角度采样距离

lasamplestep (float, default: 0.005)，用于似然计算的角度采样步长

transform_publish_period (float, default: 0.05)，变换发布时间间隔.

occ_thresh (float, default: 0.25)，栅格地图栅格值

4. 重要的参数与调试方法

在 slam_gmapping 节点中，有两个参数较为重要，分别是：

minimumScore（最小匹配得分）和 particles（滤波粒子数）。

- minimunScore

该参数决定了建图时对激光雷达的置信度，该参数越大，在进行激光匹配时越有可能失败，转而使用里程计的数据。在很多情况下，该参数的选择会直接决定建图能否成功。

通常，该参数选择 100-600，在室内等容易识别到特征点的地方，该参数可适当调小，在空旷的室外等较难识别到特征点的地方，该参数应适当调大。如果发现建图过程中激光匹配明显不稳定，则需调大。

- particles

该参数是 gmapping 算法中粒子滤波部分的粒子数。该参数的选取，将会在很大程度上影响粒子滤波的准确性，粒子数越多，滤波的结果越稳定；然而，过大的粒子数会导致建图速度过慢、耗费资源过多等问题。

当发现建图过程中坐标系出现漂移，且经过调节 minimunScore 无改善时，说明此时滤波粒子数过少，应加大；当发现建图时地图更新速度缓慢，则应减小。

实验九　激光雷达寻路实验（C++实现）

实验目的：

掌握采用激光雷达识别锥桶并寻路的方法。

实验内容：

1、获取激光雷达的测量数据并处理；

2、使用获取到的数据进行寻路操作。

实验仪器：

ROS 智能车、激光雷达。

实验原理：

利用激光雷达的测量数据识别锥桶，判断锥桶所处的位置，并通过锥桶位置进行寻路。

实验步骤：

1．了解激光雷达测量数据

激光雷达传入的测量数据消息类型为 sensor_msgs 下的 Laserscan，其主要由以下几部分构成：

- Header　header，包含 seq、stamp、frame_id，seq，该内容非本实验重要内容，故不详述；
- float32　angle_increment，每次扫描增加的角度
- float32　time_increment，扫描的时间间隔
- float32　scan_time，扫描的时间间隔
- float32　range_min，距离最小值

- float32　range_max，距离最大值
- float32　angle_min，开始扫描时的角度
- float32　angle_max，扫描结束时的角度
- float32[]　ranges，距离信息
- float32[]　intensities，强度数据

其中，我们需要的信息是 angle_min、angle_max、angle_increment 和 ranges。

通过 angle_min、angle_max、angle_increment 可获知数组长度，通过 ranges 中各数据的位置、angle_min 和 angle_increment 即可获知周围各角度的距离信息，即可获得周围任意边界点的极坐标信息。

例：设激光雷达所处位置为原点，其正前方为 x 轴正方向，正右方为 y 轴正方向，由上往下看时激光雷达顺时针旋转，则可根据 ranges[i]的信息获知，此时由激光雷达正前方顺时针旋转 theta=i*angle_increment + angle_min（弧度）的方向上，边界与原点的距离为 ranges[i]。

由此，我们便可得到激光雷达获取到的任意边界点的极坐标信息。

2. 代码实现获取激光雷达信息并处理

首先输入以下命令，并输入密码，安装 sensor_msgs 依赖包：

```
sudo apt install ros-sensor-msgs
```

安装完成后，编写 C++代码

由于本实验中需实现在回调函数中发布数据，因此需定义一个类如下：

```
class PubAndSub//发布+订阅对象
{
private:
    ros::NodeHandle n_;
    ros::Publisher pub_;
    ros::Subscriber sub_;
public:
    PubAndSub()
    {
        pub_ = n_.advertise<geometry_msgs::Twist>("/car/cmd_vel",5);
        sub_ = n_.subscribe("/scan",5,&PubAndSub::callback,this);
    }
    void callback(const sensor_msgs::LaserScan::ConstPtr &laser);
};
```

其中，/scan 为雷达话题，/car/cmd_vel 为小车底盘控制话题

随后包含头文件、定义所需变量并编写 main 函数如下：

```
#include "ros/ros.h"
#include "sensor_msgs/LaserScan.h"
#include "geometry_msgs/Twist.h"
#include <math.h>
#include "Control.h"


#define freq 1440//每转一圈雷达扫描次数
#define speed 1825

typedef struct //极坐标系下的点
{
    double range;
    double theta;
}Point_polar;

typedef struct //直角坐标系下的点
{
    double x;
    double y;
}Point_rectangular;
```

```
int main(int argc, char *argv[])
{
    pid.Init();
    ros::init(argc,argv,"laser_go");
    PubAndSub PAS;
    ros::spin();
    return 0;
}
```

其中 Control.h 为编写的 PID 文件，内容为 PID 控制器（如下），在此处不作细讲。

```
namespace Control
{
    class PID
    {
    public:
        double kp;
        double ki;
        double kd;
        double error_out;
        double last_error;
        double integral;
        double inte_max;//积分限副
        double last_diff;//上一次微分值

        double PIDPositional(double error);//位置式PID控制器
        double PIDIncremental(double error);
        void Init();

    };

    double PID::PIDPositional(double error)//位置式PID控制器
    {
        integral += error;

        if(integral>inte_max)
            integral = inte_max;

        error_out = (kp*error) + (ki*integral) + (kd*(error-last_error));
        last_error = error;
        return error_out;
    }
```

```
    double PID::PIDIncremental(double error)//增量式PID控制器
    {
        error_out = kp*(error-last_error) + ki*error + kd*((error-last_error)-last_diff);

        last_diff = error - last_error;
        last_error = error;

        return error_out;
    }

    void PID::Init()//PID初始化,参数在此调节
    {
        kp =15.0;
        ki = 0.1;
        kd = 3.0;
        error_out = 0.0;
        last_error = 0.0;
        integral = 0.0;
        inte_max = 8.0;
        last_diff = 0.0;
        // kp =18.0;
        // ki = 0.1;
        // kd = 3.0;
        // error_out = 0.0;
        // last_error = 0.0;
        // integral = 0.0;
        // inte_max = 8.0;
        // last_diff = 0.0;
    }
}
```

随后进行回调函数的内容编写，如下：

```
void PubAndSub::callback(const sensor_msgs::LaserScan::ConstPtr &laser)
{
    char negetiveNum = 0,positiveNum = 0;
    int i,j=0;
    double range = 0,error = 0,negetiveSum = 0,positiveSum = 0;
    geometry_msgs::Twist twist;
    Point_polar pp[30] = {0,0};
    Point_rectangular pr[30] = {0,0};

    for (i = 1;i < freq;i++)
    {
        if(laser->ranges[i-1] - laser->ranges[i] >= 2.0)//若变化大于2米则识别为锥桶
            if(laser->ranges[i] > 0.2 && laser->ranges[i] < 2.5)
            {
                pp[j].range = laser->ranges[i];
                pp[j].theta = i*laser->angle_increment + laser->angle_min;//获得2.5米范围内各锥桶的极坐标
                j++;
            }
    }
    for(i = 0;i < 30;i++)
    {
        if(pp[i].range)
        {
            pr[i].x = pp[i].range*sin(pp[i].theta);//获得2.5米范围内各锥桶的直角坐标(激光雷达为原点)
            pr[i].y = pp[i].range*cos(pp[i].theta);//为方便使用，此处x轴与y轴反置
            if(pr[i].y >= -0.25)//排除后方距离超过0.25米的锥桶
            ROS_INFO("found bucket point:(%.2f,%.2f)\n",pr[i].x,pr[i].y);
            else {
                pr[i].x = 0;
                pr[i].y = 0;
            }
```

关于识别锥桶，在无外物干扰的情况下，对 ranges 进行遍历，若距离产生突变，则说明此处极有可能是锥桶，本实验采取该方法，当距离的变化大于 2 米时识别为锥桶。运行节点，能够正常输出锥桶坐标如下，则证明代码无误：

```
[ INFO] [1670587547.587941937]: found bucket point:(-1.71,0.11)
[ INFO] [1670587547.587989417]: found bucket point:(-1.98,0.94)
[ INFO] [1670587547.588024582]: found bucket point:(-1.09,0.63)
[ INFO] [1670587547.588063578]: found bucket point:(-0.79,1.55)
```

至此，我们已经可以获取周围 2.5 米内的锥桶的直角坐标信息。

3. 代码实现激光雷达寻路

通过锥桶的坐标信息实现寻路的方法有多种，以下是一种极其简单且实用的方法：

在进行一些筛选项的前提下，将锥桶的 x 坐标相加，得出偏差，再通过 PID 控制器处理并输出。

具体步骤如下：

首先初步处理数据，对上面编写的**回调函数**进行修改和补充，排除距离激

光雷达 0.5 米以内的激光点云信息，以使小车不会过于靠近锥桶（当发现小车时常碰撞锥桶时，应增大该参数），并使锥桶的 x 坐标相加

```
void PubAndSub::callback(const sensor_msgs::LaserScan::ConstPtr &laser)
{
    char negetiveNum = 0,positiveNum = 0;
    int i,j=0;
    double range = 0,error = 0,negetiveSum = 0,positiveSum = 0;
    geometry_msgs::Twist twist;
    Point_polar pp[30] = {0,0};
    Point_rectangular pr[30] = {0,0};

    for (i = 1;i < freq;i++)
    {
        if(laser->ranges[i-1] - laser->ranges[i] >= 2.0)
            if(laser->ranges[i] > 0.5 && laser->ranges[i] < 2.5)
            {
                pp[j].range = laser->ranges[i];
                pp[j].theta = i*laser->angle_increment + laser->angle_min;//获得2.5米范围内各锥桶的极坐标
                j++;
            }
    }
    for(i = 0;i < 30;i++)
    {
        if(pp[i].range)
        {
            pr[i].x = pp[i].range*sin(pp[i].theta);
            pr[i].y = pp[i].range*cos(pp[i].theta);//获得2.5米范围内各锥桶的直角坐标(激光雷达为原点)
            if(pr[i].y >= -0.25)//排除后方距离超过0.25米的锥桶
            ROS_INFO("found bucket point:(%.2f,%.2f)\n",pr[i].x,pr[i].y);
            else {
                pr[i].x = 0;
                pr[i].y = 0;
            }
```

修改内容为：

```
if(laser->ranges[i] > 0.5 && laser->ranges[i] < 2.5)
```

同时在第二个 for 循环内第一个 if 语句（即：if（pp[i].range))中加入以下代码：

```
        if(pr[i].x>0)
        {
            positiveNum ++;
            positiveSum += pr[i].x;//左边的锥桶x坐标相加
        }
        else if(pr[i].x)
        {
            negetiveNum ++;
            negetiveSum += pr[i].x;//右边的相加
        }
```

此时我们分别得到了左右锥桶的 x 坐标之和与其数量，但此时数据中被远端的锥桶影响较多，容易出现直角弯转不过的问题。因此，我们还需滤除部分信息，本实验采用的是一种简单的方式：将 x 坐标超过平均值 1.75 倍的锥桶信息滤除。代码实现如下：

```
for(i = 0;i< 30;i++)
{
    if(pr[i].x>0)
        if(pr[i].x >= 1.75*positiveSum/positiveNum)
        {
            positiveSum -= pr[i].x;
            positiveNum --;
        }
    else if(pr[i].x<0)
        if(pr[i].x <= 1.75*negetiveSum/negetiveNum)
        {
            negetiveSum -= pr[i].x;
            negetiveNum --;
        }

}
error = negetiveSum + positiveSum;//通过锥桶的坐标算出误差

//error = pid.PIDIncremental(error);//增量式PID控制器，参数在/src/Control.h Init函数中调节
error = pid.PIDPositional(error);//位置式PID控制器
```

该参数可调节，具体视赛道状况而定。经过实验，该参数在 1.65-1.85 范围内均可，过小可能会导致数据丢失，过大可能会造成干扰。

最后，在回调函数中将数据组织并发布即可，代码如下：

```
twist.angular.z = 90 + error;//将误差换算成角度并发布
twist.linear.x = speed;//速度
if(twist.angular.z > 180)
    twist.angular.z = 180;
if(twist.angular.z < 0)
    twist.angular.z = 0;

twist.angular.y = 0;
twist.angular.x = 0;
twist.linear.y = 0;
twist.linear.z = 0;

pub_.publish(twist);
ROS_INFO("error:%.2f",error);
```

将小车放在赛道中，打开 Run_car.launch 和本实验编写的节点，即可实现激光雷达自主寻路。

实验十　坐标获取与自主导航

实验目的：

1、获取智能车在建图时途径的坐标点；

2、根据坐标的进行导航。

实验内容：

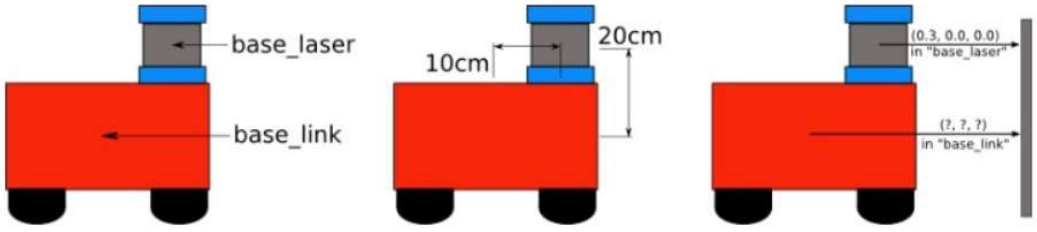
5. 获取/base_link 坐标系下当前坐标；
6. 发布并将坐标点保存至文件；
7. 读取文件并进行导航。

实验仪器：

ROS 智能车、电脑 。

实验原理：

已知激光雷达与地盘坐标关系，创建监听器，通过路线监听 tf 转换，将监听得到的四元数与 trans 值进行参数传送，然后将参数进行函数处理后存入文件。



需要进行导航时，将途径路径的坐标点进行读取后进行定点导航。

实验步骤：

1. 创建监听器和发布者对象

- 初始化时创建监听器

```
def __init__(self):
    self.tf_listener = tf.TransformListener()
    #创建了监听器，它通过线路接收tf转换，并将其缓冲10s。若用C++写，需设置等待10s缓冲。
    try:
        self.tf_listener.waitForTransform('/map', '/base_link', rospy.Time(), rospy.Duration(1.0))
        #猜测：等待Duration=1s，判断map是否转换为base_link
    except (tf.Exception, tf.ConnectivityException, tf.LookupException):
        return
    #try ... except ... 进行异常处理，若 try... 出现异常，则将错误直接输出打印，而不是以报错的形式显示。
```

- 通过监听器获取四元数与坐标点并取四元数的 y、w 与坐标点的 x、y 作为参数传出。

```
def get_pos(self):
    try:
        (trans, rot) = self.tf_listener.lookupTransform('/map', '/base_link', rospy.Time(0))
    #rospy_Time(0)指最近时刻存储的数据
    #得到从 '/map' 到 '/base_link' 的变换，在实际使用时，转换得出的坐标是在 '/base_link' 坐标系下的。
    except (tf.LookupException, tf.ConnectivityException, tf.ExtrapolationException):
        rospy.loginfo("tf Error")
        return None

    data = PoseStamped()            #定义四元数类型
    #对应getpoints.py中的订阅函数赋值
    data.pose.position.x = trans[0]
    data.pose.position.y = trans[1]
    data.pose.orientation.z = rot[2]
    data.pose.orientation.w = rot[3]
    return (data)
```

- 创建发布者对象，将获取的坐标参数发布给订阅者

```
if __name__ == "__main__":
    rospy.init_node('get_pos_demo',anonymous=True)
        #启动节点get_pos_demo，同时为节点命名， 若anonymous为真则节点会自动补充名字，实际名字以get_pos_demo_12345等表示
        #若为假，则系统不会补充名字，采用用户命名。如果有重名，则最新启动的节点会注销掉之前的同名节点。
    pub = rospy.Publisher("position",PoseStamped,queue_size=10)

    robot = Robot()
    r = rospy.Rate(1)    #设置速率，每秒发1次
    while not rospy.is_shutdown():    #如果节点已经关闭则is_shutdown()函数返回一个True，反之返回False
        pub.publish(robot.get_pos())  #发布消息
        r.sleep()
```

2. 创建接受者节点

- 创建接收者对象，将接收到的参数写入文件

```
if __name__ == '__main__':
    try:
        rospy.init_node("getPoint",anonymous=True)
        rospy.Subscriber('position',PoseStamped,PoseStampedCB,queue_size=10)
        settings = termios.tcgetattr(sys.stdin)
        rospy.spin()
    except rospy.ROSInterruptException:
        pass
```

```python
def PoseStampedCB(data):
    global count
    gx = data.pose.position.x
    gy = data.pose.position.y
    gz = data.pose.orientation.z
    gw = data.pose.orientation.w
    #count = count + 1
    point.append([gx,gy,gz,gw])
    data_write_csv('test.csv',point)
    # with open('test.csv','wb') as csvfile:
    #     pointwriter = csv.writer(csvfile,delimiter=' ',quotechar='|',quoting=csv.QUOTE_MINIMAL)
    #     pointwriter.writerow([gx,gy,gz,gw])
    #     print("write succ!!")


def data_write_csv(file_name, datas):
    file_csv = codecs.open(file_name,'w+','utf-8')
    writer = csv.writer(file_csv, delimiter=',', quoting=csv.QUOTE_MINIMAL)
    for data in datas:
        writer.writerow(data)
    print("write succ!!")
```

获取到的导航的存储在接收者节点源码所在文件夹的 test.csv 文件中

3. 编写 launch 文件

文件放在工作空间下的 racecar 功能包的 launch 文件夹中

```
<launch>
  <arg name="use_rviz" default="true" />

  <!-- Navstack -->
  <node pkg="move_base" type="move_base" respawn="false" name="move_base">
    <!-- local planner -->

    <param name="base_global_planner" value="navfn/NavfnROS"/>
    <param name="base_local_planner" value="dwa_local_planner/DWAPlannerROS"/>
    <rosparam file="$(find racecar)/param/dwa_local_planner_params.yaml" command="load"/>

    <!-- costmap layers -->
    <rosparam file="$(find racecar)/param/local_costmap_params.yaml" command="load"/>
    <rosparam file="$(find racecar)/param/global_costmap_params.yaml" command="load"/>
    <!-- move_base params -->
    <rosparam file="$(find racecar)/param/base_global_planner_params.yaml" command="load"/>
    <rosparam file="$(find racecar)/param/move_base_params.yaml" command="load"/>
  </node>

  <node pkg="racecar" type="car_controller_new" respawn="false" name="car_controller" output="screen">

    <param name="Vcmd" value="1.5" /> <!--speed of car m/s       -->
    <!-- ESC -->
    <param name="baseSpeed" value="150"/>
    <param name="baseAngle" value="0.0"/>
    <param name="Angle_gain_p" value="-3.0"/>
    <param name="Angle_gain_d" value="-3.0"/>
    <param name="Lfw" value="1.5"/>
    <param name="vp_max_base" value="200"/>
    <param name="vp_min" value="200"/>

  </node>

  <!-- Rviz -->
  <node pkg="rviz" type="rviz" name="rviz" args="-d $(find racecar)/rviz/amcl.rviz" if="$(arg use_rviz)" />
</launch>
```

4. 运行

运行上面编写的 launch 文件，以及 Run_car.launch 、Run_gmapping.launch 和键盘控制节点，随后运行本实验中编写的发布与订阅节点，并使用键盘控制小车运动，待建图完毕后关闭发布、订阅和键盘控制节点，并运行源码中的 goal_loop.py，即可实现小车的循环导航，小车将按键盘控制时的轨迹进行运动。

若不想在开启 gmapping 建图的同时进行导航，则还需开启 amcl 节点进行定位。