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。