ROS-PID实验

211250175 王艺羲

1. 基本功能实现

在f1tenth Simulator中实现基于pid算法控制汽车运动功能。

在本实验中，实现一个PID控制器，使汽车在固定距离上平行于走廊的墙壁行驶。通过从Hokuyo激光雷达获取激光扫描距离，计算所需的转向角度和速度(驱动参数)，以此来驱动汽车。

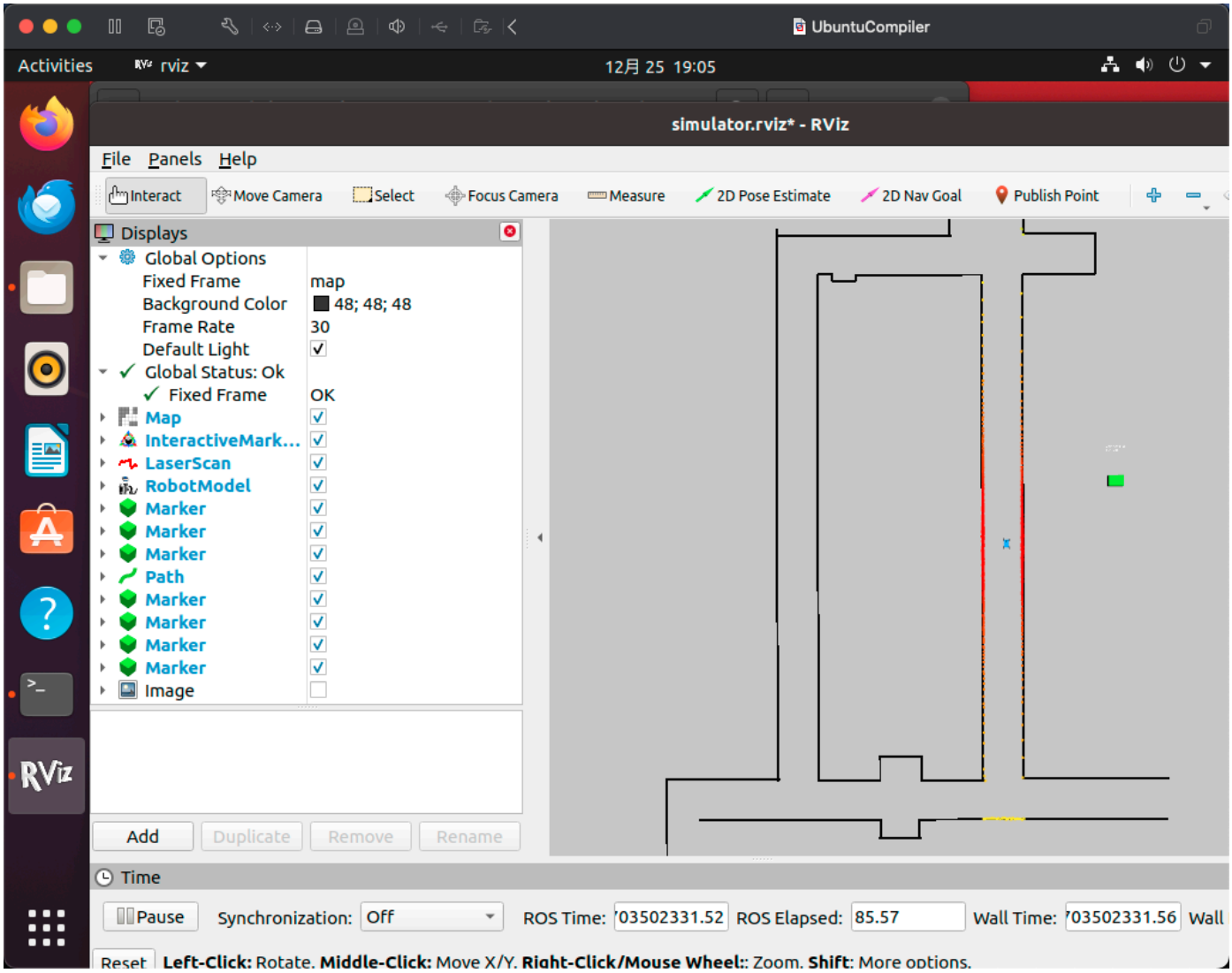
具体实现代码在/catkin_ws/src/f1tenth_simulator/node/safety_node.cpp

2. 速度为1.5m/s的实现

在代码中将小车速度设为1.5m/即可

```
ackermann_drive_result.drive.speed = 1.5;
```

运行结果见video1

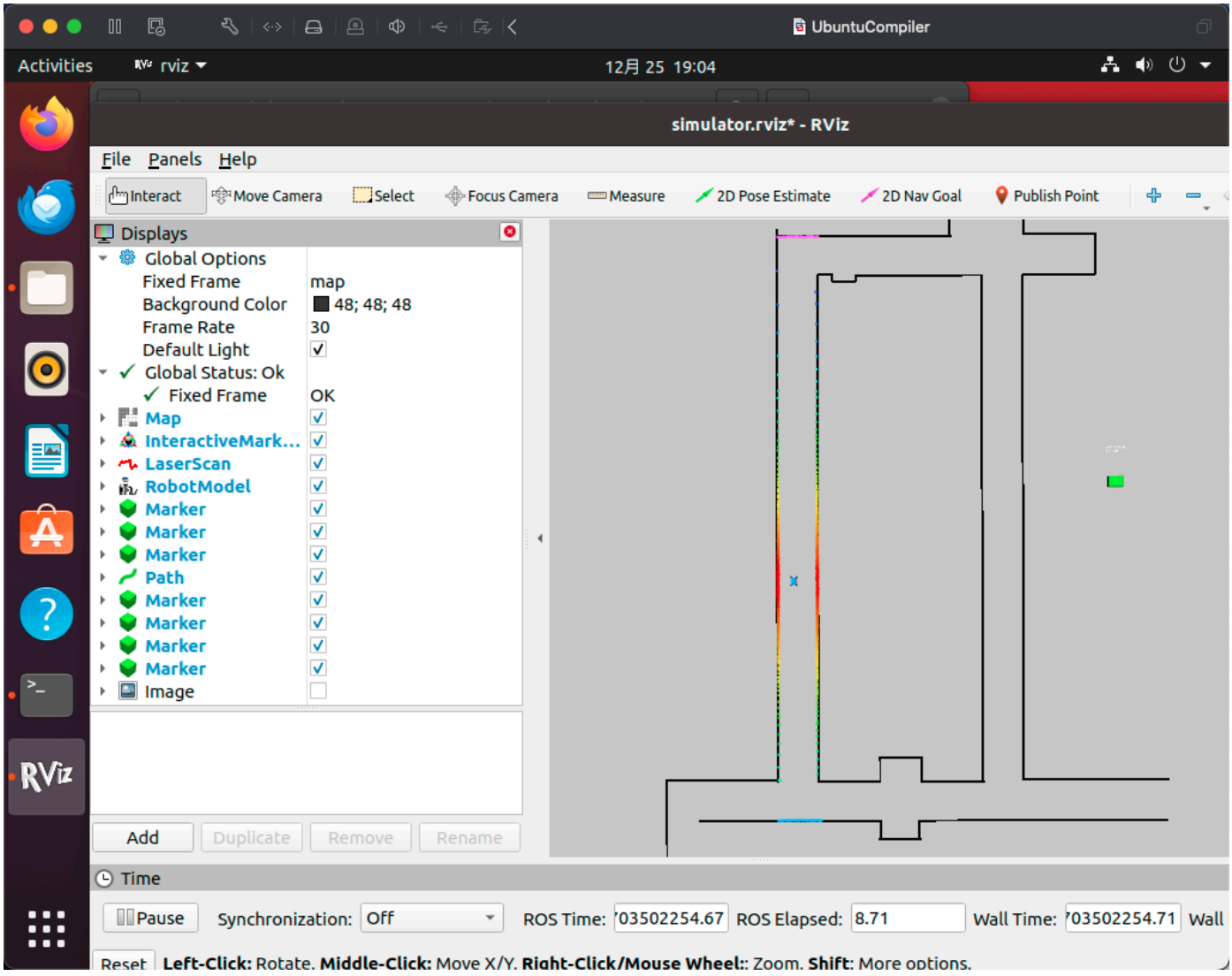


3. 速度为4m/s的实现

若直接将小车速度设为4.0m/s会导致小车与墙发生碰撞，所以我们在转弯处对小车进行制动处理

```
if (abs(ackermann_drive_result.drive.steering_angle) > 20.0 / 180.0 * PI) {
  ackermann_drive_result.drive.speed = 2.0;
} else if (abs(ackermann_drive_result.drive.steering_angle) > 10.0 / 180.0 * PI) {
  ackermann_drive_result.drive.speed = 3.0;
} else {
  ackermann_drive_result.drive.speed = 4.0;
}
```

运行结果见video2



4. 代码功能说明

4.1 参数整定

KP、KD和KI是比例-积分-微分(PID)控制器的三个参数，它们分别代表比例、积分和微分项：

KP（比例系数）

作用：KP是比例项的系数，它决定了控制器响应对当前误差的强度。较高的KP值会使系统响应更快，但也可能导致过冲（即超过目标点然后再回来），甚至系统不稳定。较低的KP值会使系统响应变慢，但能提高稳定性。

KD（微分系数）

作用：KD是微分项的系数，它基于误差的变化率（即误差的导数）来进行控制。微分控制考虑的是误差的变化速度, 微分项有助于减少系统的过冲和振荡，使系统更快地稳定下来。高的KD值可以使系统对快速变化更敏感，但过高可能导致噪声放大，引起不必要的控制动作。

KI（积分系数）

作用：KI是积分项的系数，它基于误差随时间的累积总和进行控制。积分控制的目的是消除稳态误差，积分项有助于确保系统最终能够到达所需的设定点，消除稳态误差。但是，过高的KI值可能导致系统变得过于敏感和不稳定，引起持续的振荡。

综合使用KP、KD、KI

综合来考虑，最终确定下三者数值

```
#define KP 1.00
#define KD 0.001
#define KI 0.005
```

4.2 运行过程数据的获取

以下为小车运行过程中环境数据的获取，为小车的运行状态的处理提供实时数据。

```
ackermann_msgs::AckermannDriveStamped ackermann_drive_result;
double tmoment = ros::Time::now().toSec();
del_time = tmoment - prev_tmoment;    //当前时刻-上一个时刻 = 间隔时刻
integral += prev_error * del_time;    //对误差积分，也就是误差的无限和  积分=积分+累计误差
ackermann_drive_result.drive.steering_angle = -(KP * error + KD * (error - prev_error) /
del_time + KI * integral);
prev_tmoment = tmoment;     //时间的迭代
```

4.3 转弯时进行制动

通过获取小车与障碍的角度控制小车的速度实现速度的发布，让小车实现顺利转弯。

```
if (abs(ackermann_drive_result.drive.steering_angle) > 20.0 / 180.0 * PI)
{ackermann_drive_result.drive.speed = 2.0;}
else if (abs(ackermann_drive_result.drive.steering_angle) > 10.0 / 180.0 * PI)
{ackermann_drive_result.drive.speed = 3.0;}
else {ackermann_drive_result.drive.speed = 4.0;}
drive_pub.publish(ackermann_drive_result);
```

但是这样的处理方案不一定是最优解，通过video2可以看出小车在转弯时与墙距离过近，需要对转弯方案进行改良。

5. 算法提高与实验结果比较分析

由于小车转弯时速度的变化区间太过剧烈，导致小车转弯时车身不稳，与墙距离过近，现在对小车制动的方案进行改良。

经过测量，采用2中的方案，小车单圈用时约为**41s**，采用4.3中的方案，小车单圈用时约为**24.5s**，以单圈用时最短作为目标，可以将小车直行速度加快、增加转弯角度判断区间等方案进行优化。

5.1 增加直行速度

根据实验，小车直行速度过快时无法有足够时间判断是否转弯进行制动，从而与障碍碰撞，故将速度微增至5m/s

```
ackermann_drive_result.drive.speed = 5.0;
```

5.2 增加转弯角度判断区间

修改小车转弯角度时的速度，减小小车速度变化粒度，使得小车能够在转弯时以更大的平均速度实现转弯。

经过多次测量，综合减少小车直行震荡和转弯撞墙风险之后，最终制动方案如下：

```
if (abs(ackermann_drive_result.drive.steering_angle) > 30.0 / 180.0 * PI)
{ackermann_drive_result.drive.speed = 2.0;}
else if (abs(ackermann_drive_result.drive.steering_angle) > 25.0 / 180.0 * PI)
{ackermann_drive_result.drive.speed = 2.5;}
else if (abs(ackermann_drive_result.drive.steering_angle) > 20.0 / 180.0 * PI)
{ackermann_drive_result.drive.speed = 3.5;}
else if (abs(ackermann_drive_result.drive.steering_angle) > 15.0 / 180.0 * PI)
{ackermann_drive_result.drive.speed = 4.0;}
else if (abs(ackermann_drive_result.drive.steering_angle) > 10.0 / 180.0 * PI)
{ackermann_drive_result.drive.speed = 4.5;}
else {ackermann_drive_result.drive.speed = 5.0;}
drive_pub.publish(ackermann_drive_result);
```

经测量，小车单圈用时**提高至21s**，此为改良后的最佳方案。

6. 代码

```
#include <ros/ros.h>
#include <ackermann_msgs/AckermannDriveStamped.h>
#include <sensor_msgs/LaserScan.h>
#include <std_msgs/Float32.h>
#include <std_msgs/Float64.h>
#include "math.h"

#define KP 1.00
#define KD 0.001
#define KI 0.005
```

```
#define DESIRED_DISTANCE_RIGHT 1.0
#define DESIRED_DISTANCE_LEFT 1.0
#define LOOK_AHEAD_DIS 1.0
#define PI 3.1415927

class SubscribeAndPublish {
public:
    SubscribeAndPublish() {
        //1、发布与订阅的话题
        drive_pub = nh.advertise<ackermann_msgs::AckermannDriveStamped>("/drive", 1000);
        scan_sub = nh.subscribe("/scan", 1000, &SubscribeAndPublish::callback, this);
    }

    void callback(const sensor_msgs::LaserScan& lidar_info) {
        //2、获取激光雷达测量距离 getRange
        unsigned int b_index = (unsigned int)(floor((90.0 / 180.0 * PI -
lidar_info.angle_min) / lidar_info.angle_increment));  //dist = data.ranges[int(index)]
        double b_angle = 90.0 / 180.0 * PI;    //两条射线之间的角度
        double a_angle = 45.0 / 180.0 * PI;
        unsigned int a_index;
        if (lidar_info.angle_min > 45.0 / 180.0 * PI) {
            a_angle = lidar_info.angle_min;
            a_index = 0;
        } else {
            a_index = (unsigned int)(floor((45.0 / 180.0 * PI - lidar_info.angle_min) /
lidar_info.angle_increment));
        }
        double a_range = 0.0;
        double b_range = 0.0;
        if (!std::isinf(lidar_info.ranges[a_index]) &&
!std::isnan(lidar_info.ranges[a_index])) {
            a_range = lidar_info.ranges[a_index];   //得到a的长度
        } else {
            a_range = 100.0;
        }
        if (!std::isinf(lidar_info.ranges[b_index]) &&
!std::isnan(lidar_info.ranges[b_index])) {
            b_range = lidar_info.ranges[b_index];   //得到b的长度
        } else {
            b_range = 100.0;
        }

        //3、计算公式，参考pdf
        //在车的右边得到两条射线a、b来确定车到右墙的距离AB和相对于AB的方向
        double alpha = atan((a_range * cos(b_angle - a_angle) - b_range) / (a_range *
sin(b_angle - a_angle)));
        double AB = b_range * cos(alpha);    //实际离右墙距离
        double projected_dis = AB + LOOK_AHEAD_DIS * sin(alpha);
        error = DESIRED_DISTANCE_RIGHT - projected_dis;   //求出误差
        ROS_INFO("projected_dis = %f",projected_dis);
        ROS_INFO("error = %f",error);
        ROS_INFO("del_time = %f\n",del_time);
```

```
        SubscribeAndPublish::pid_control();
    }
        //4、PID控制器
    void pid_control() {
        ackermann_msgs::AckermannDriveStamped ackermann_drive_result;
        double tmoment = ros::Time::now().toSec();
        del_time = tmoment - prev_tmoment;    //当前时刻-上一个时刻 = 间隔时刻
        integral += prev_error * del_time;    //对误差积分，也就是误差的无限和  积分=积分+累计误差
        ackermann_drive_result.drive.steering_angle = -(KP * error + KD * (error -
prev_error) / del_time + KI * integral);
        prev_tmoment = tmoment;     //时间的迭代

        //不同情况下的速度调整，转弯时速度降低，直行时速度加快
        if (abs(ackermann_drive_result.drive.steering_angle) > 30.0 / 180.0 * PI) {
            ackermann_drive_result.drive.speed = 2.0;
        }else if (abs(ackermann_drive_result.drive.steering_angle) > 25.0 / 180.0 * PI) {
            ackermann_drive_result.drive.speed = 2.5;
        } else if (abs(ackermann_drive_result.drive.steering_angle) > 20.0 / 180.0 * PI) {
            ackermann_drive_result.drive.speed = 3.5;
        }else if (abs(ackermann_drive_result.drive.steering_angle) > 15.0 / 180.0 * PI) {
            ackermann_drive_result.drive.speed = 4.0;
        }else if (abs(ackermann_drive_result.drive.steering_angle) > 10.0 / 180.0 * PI) {
            ackermann_drive_result.drive.speed = 4.5;
        }else {
            ackermann_drive_result.drive.speed = 5.0;
        }
        drive_pub.publish(ackermann_drive_result);
    }
private:
    ros::NodeHandle nh;
    ros::Publisher drive_pub;
    ros::Subscriber scan_sub;
    double prev_error = 0.0;     //前一个误差
    double prev_tmoment = ros::Time::now().toSec();     //当前时间，单位s
    double error = 0.0;
    double integral = 0.0;
    double speed = 0.0;
    double del_time = 0.0;

};
int main(int argc, char** argv) {
    ros::init(argc, argv, "wall_following_pid");
    SubscribeAndPublish SAPObject;
    ros::spin();
    return 0;
}
```