2023-HW-Fundamentals-of-autopilot-project

本项目已经全部托管在了GitHub上，诚邀您浏览并给予宝贵的意见

https://github.com/pbcn2/2023-HW-Fundamentals-of-autopilot-project

算法库简述

目前本项目使用到的算法是由两个部分整合而成

将所有算法的单独实现并收集在了两个文件夹中

- 前端-规划（`Front End - Planning`）
- 后端-控制（`Back End - Control`）

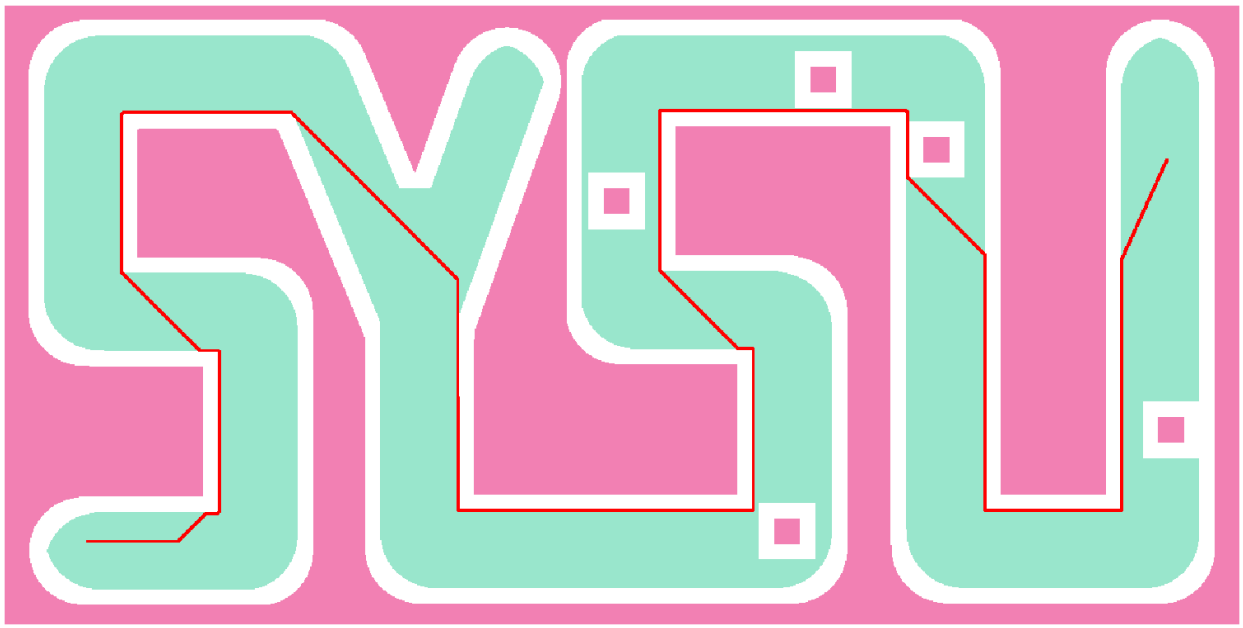
前端

前端算法库位于`.\Algorithm library\Front End - Planning`文件夹，多种函数实现了主体的路径规划功能

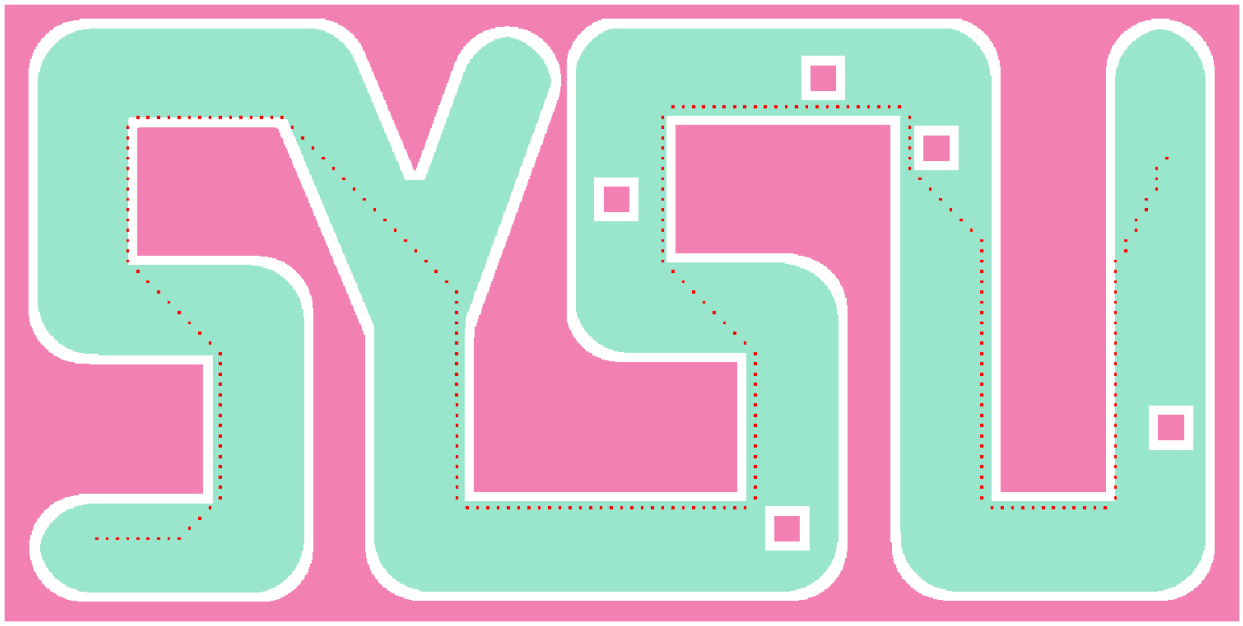
Astar算法

A*文件夹中的`v1.m` `v2.m` `v3.m`三个函数实现了手动的A*路径查找算法，具体如下：

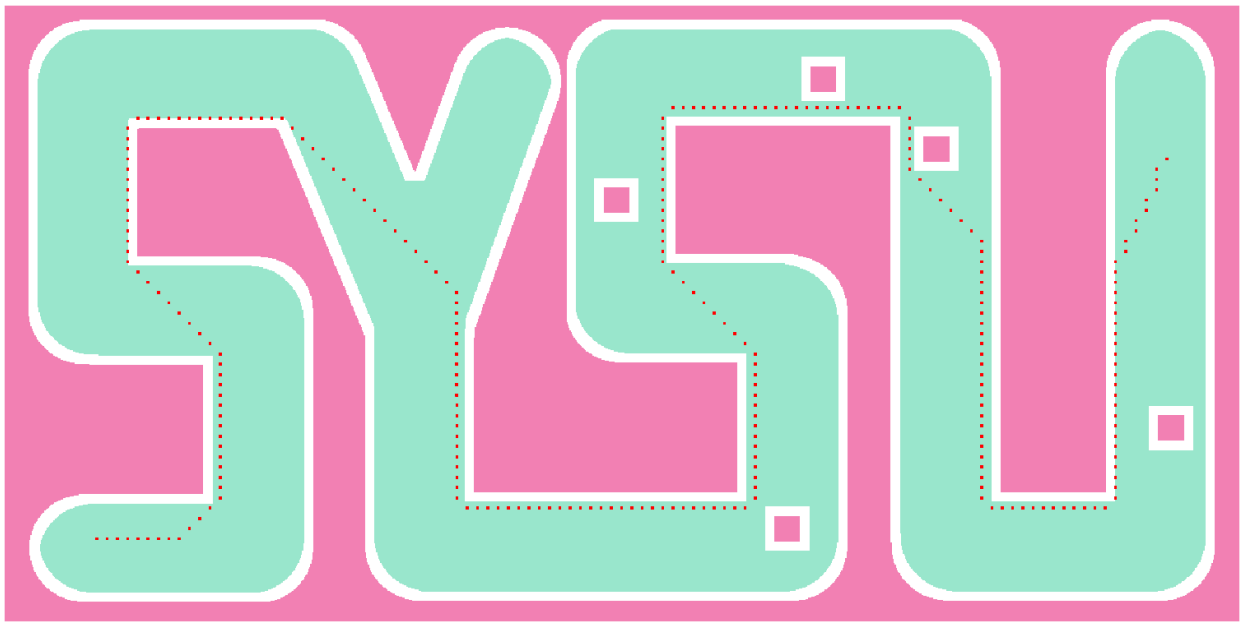
`v1.m`文件实现了A*算法，对路径进行规划，同时引入障碍物膨胀机制，对障碍物进行膨胀，使得生成的路径远离障碍物，降低小车碰撞的可能性



`v2.m`在此基础上对A*进行了步长放大，使得搜索速度显著加快，并且能够为后来引入三次、五次插值曲线做好准备

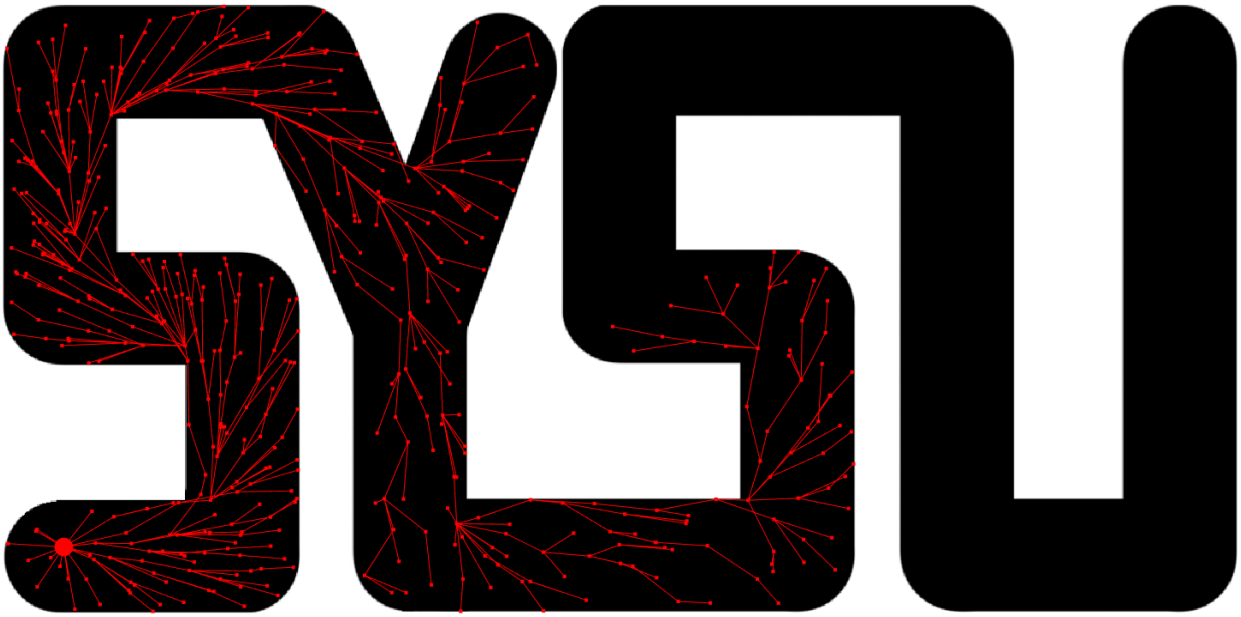


`v3.m`由对搜索步长进行了调整，使得在转角处的步长更小，意味着能够使得路径点更加密集，以防止出现相邻两点连线穿过障碍物的情况



RRTstar算法

`RRT_star.m`实现了对图片调用RRT_star算法进行路径规划，在实验过程中能够有效找到路径。但是路径转折点过多，且迭代生成的时间过长，尤其是寻路过程执行到后期的时候有效点生成效率越来越低，时间开销难以接受。因此最终放弃此种算法。



辅助函数

`compute_psi.m`函数实现了对当前路径的航向角的计算，具体来讲，以水平向右的角度为零点，向竖直上方偏转记作正数，向下偏转记作负数

`compute_curvature.m`函数实现了对航线曲率的计算，因为是离散点，所以无法直接通过求导等方法实现，在此使用了相邻三个点求曲率的算法，具体见代码。特别的是，面向航行方向，向左弯曲为正，向右弯曲为负。

具体实现：Python 版本

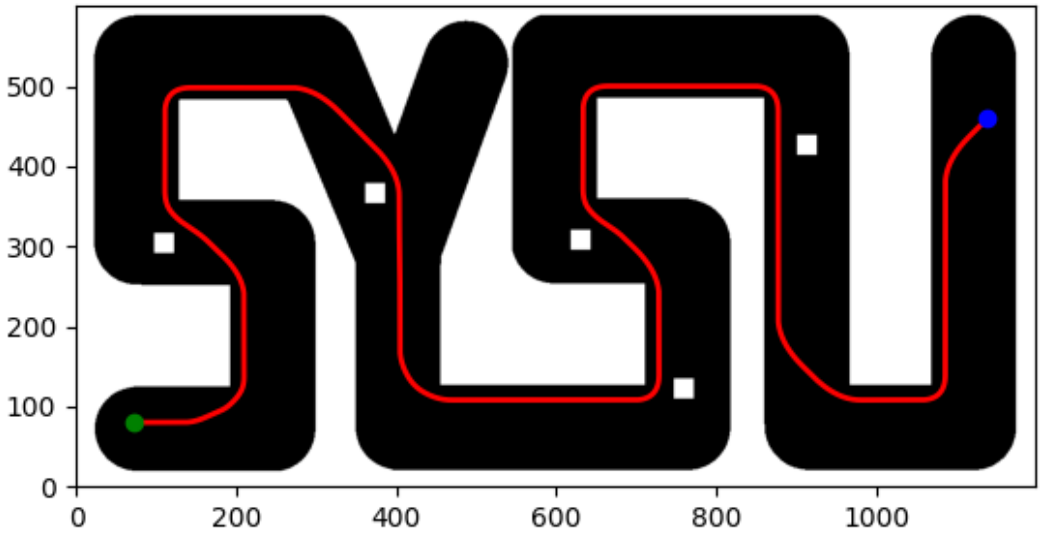
`py_v2.py`是具体使用的函数，在这个函数中，基础的路径搜索算法使用的是Astar

使用障碍物膨胀的方法解决离障碍物太近的问题

在算出路径以后，使用 滑动窗口平均 的方法对路径进行平滑化，效果较好。

且本人使用python较matlab熟练很多，因此最后综合考量之后采用了这一版的路径规划。

同时在python中集成了两个函数用于计算航向角和航向曲率，可以实现在一个脚本中对所有需要的参数进行计算。



后端

后端算法实现位于`"Integrate\car_sim_for_student_r2015.slx"`文件，

- 横向控制——离线lqr控制
- 纵向控制——pid控制

核心算法——离线lqr

```
m = 0.041;  % 车的质量
a = 0.4;  % 质心到前悬挂的距离
b = 0.4;  % 质心到后悬挂的距离
L = a + b;  % 轴距
Iz = 27.8e-6;  % 绕z轴的转动惯量
k1 = -112600;
k2 = -89500;
vx = 4;  % 车辆车速（由于是离线lqr所以在这里取一个和平均车速差不多的近似值即可）
A = [0, vx, 1, 0;
    0, 0, 0, 1;
    0, 0, (k1+k2)/m/vx, (a*k1-b*k2)/m/vx-vx;
    0, 0, (a*k1 - b*k2)/Iz/vx, (a^2*k1+b^2*k2)/Iz/vx];
B = [0; 0; -k1/m; -a*k1/Iz];
Q = [1, 0, 0, 0;
    0, 1, 0, 0;
```

```
    0, 0, 1, 0;
    0, 0, 0, 1;];
R = 1;
k = lqr(A,B,Q,R);
%setenv('MW_MINGW64_LOC', 'D:\Dev-Cpp\MinGW64');
```

项目实现方案

本项目的完整实现部分全部位于`Integrate`文件夹中

前端

使用*python 3.11*运行py_v2.py进行路径规划,将结果输出到

`".\Integrate\traj\traj_diySYSU.mat"`中供后续模块调用

后端

使用*simulink*运行`".\Integrate\car_sim_for_student_r2015.slx"`

后端模块详解

横向控制部分——lqr控制器

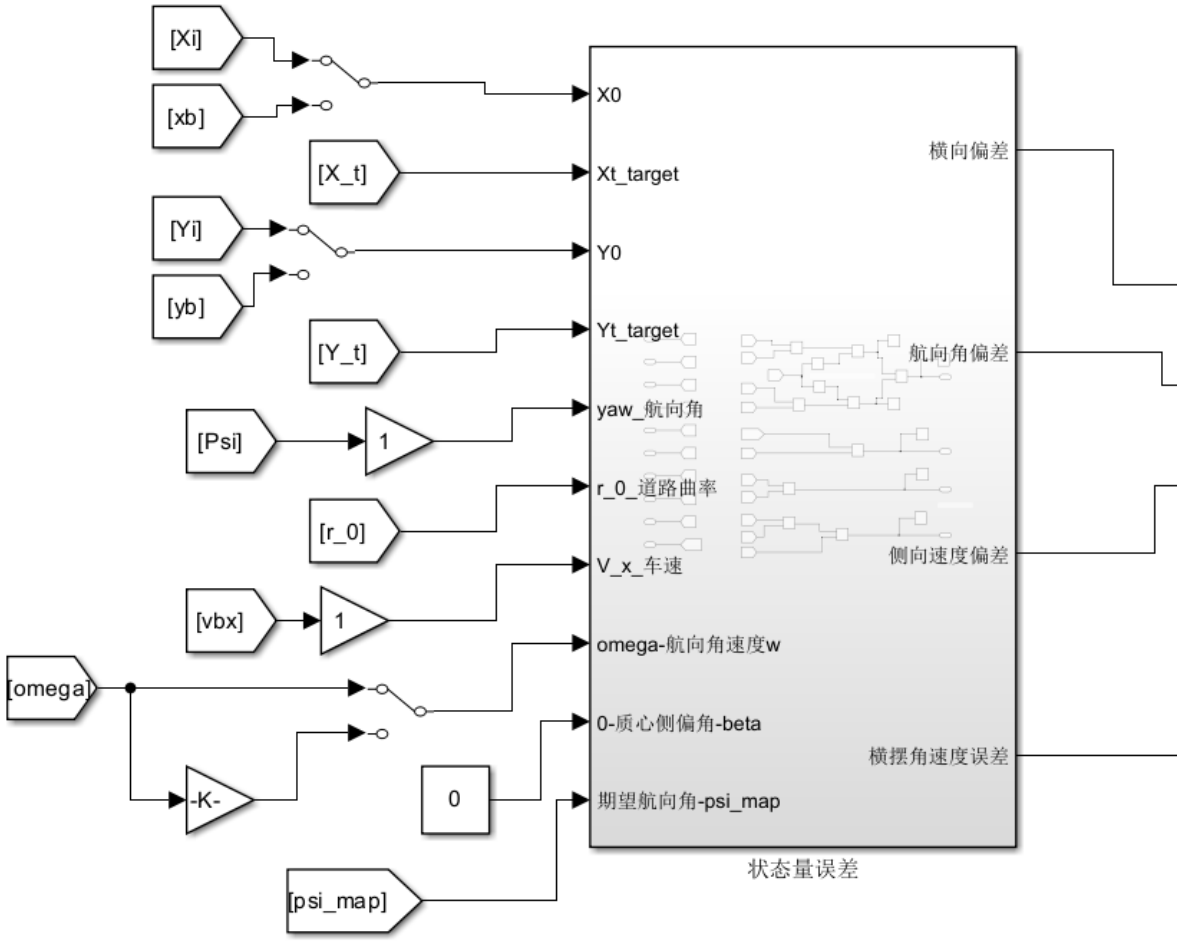
在模型初始化的时候在模型参数中设置了先运行`lqr_nums.m`计算出lqr向量的值，以备使用

状态量误差计算单元

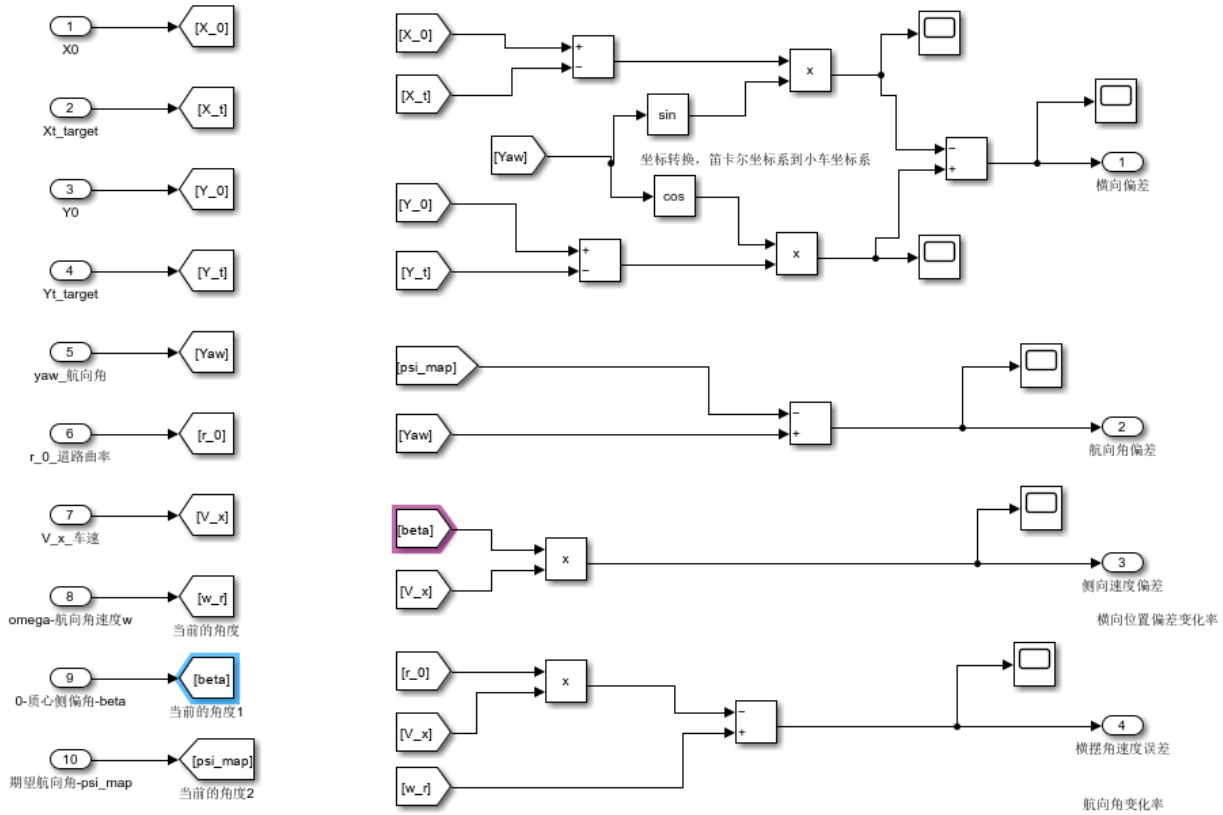
lqr控制器需要利用四个状态量误差配合lqr向量中的元素对整体进行控制，分别是：

- 横向偏差——小车与路径间的横向距离偏差
- 航向角偏差
- 横向偏差变化率
- 航向角变化率

整体输入效果如图：

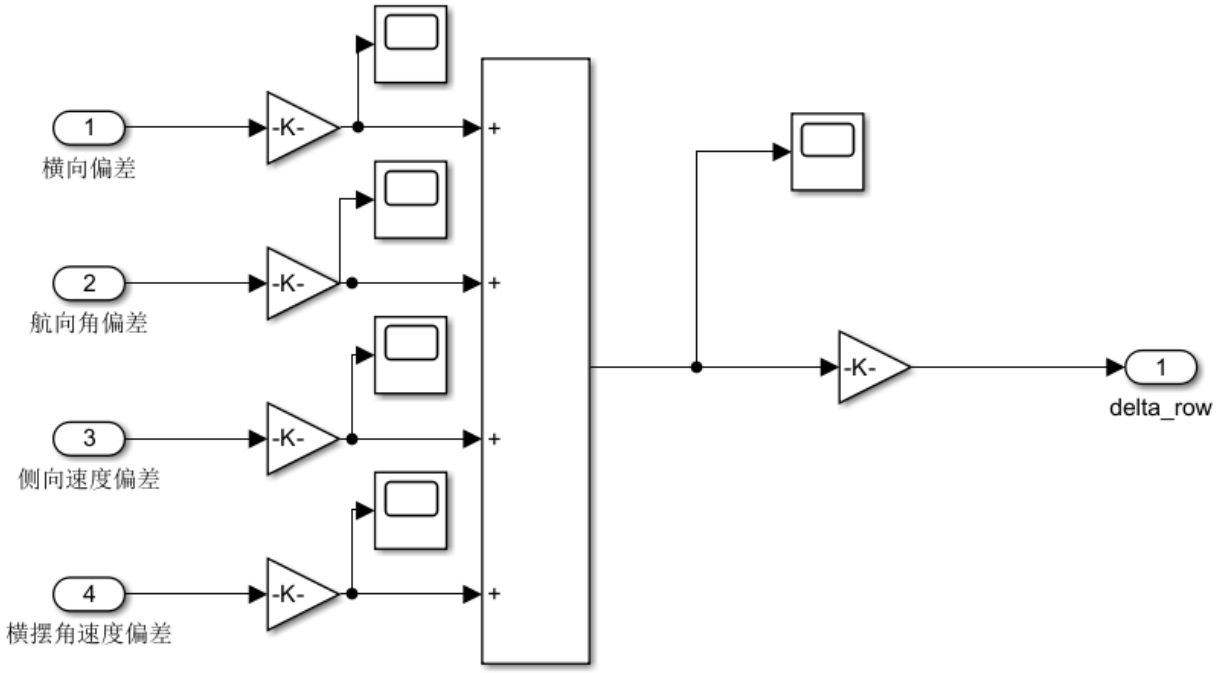

[Xi]
[xb]
[X_t]
[Yi]
[yb]
[Y_t]
[Psi]
1
[r_0]
[vbx]
1
[omega]
-K-
0
[psi_map]
X0
Xt_target
Y0
Yt_target
yaw_航向角
r_0_道路曲率
V_x_车速
omega-航向角速度w
0-质心侧偏角-beta
期望航向角-psi_map
横向偏差
航向角偏差
侧向速度偏差
横摆角速度误差
状态量误差

具体的计算过程：

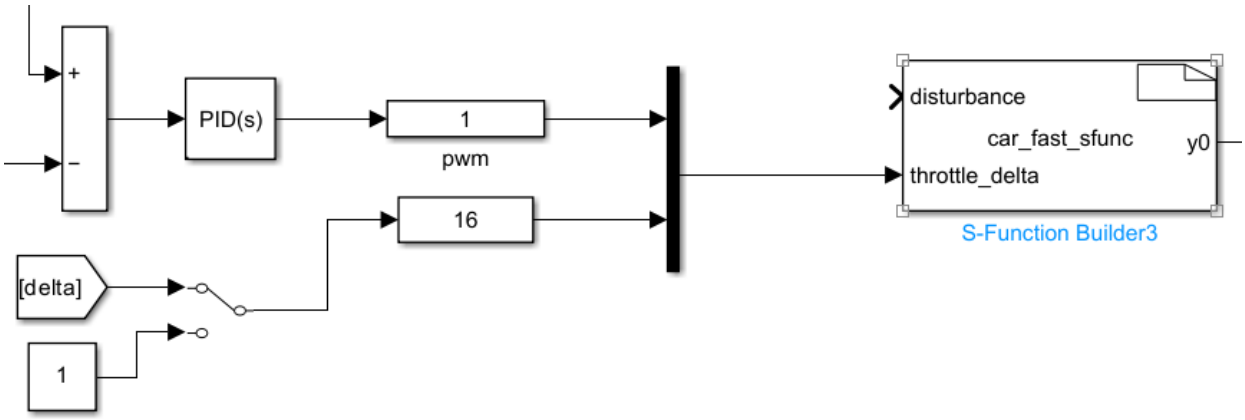

1
X0
[X_0]
2
Xt_target
[X_t]
3
Y0
[Y_0]
4
Yt_target
[Y_t]
5
yaw_航向角
[Yaw]
6
r_0_道路曲率
[r_0]
7
V_x_车速
[V_x]
8
omega-航向角速度w
[w_r]
当前的角度
9
0-质心侧偏角-beta
[beta]
当前的角度1
10
期望航向角-psi_map
[psi_map]
当前的角度2
[X_0]
[X_t]
[Yaw]
[Y_0]
[Y_t]
sin
cos
坐标转换，笛卡尔坐标系到小车坐标系
横向偏差
[psi_map]
[Yaw]
航向角偏差
[beta]
[V_x]
侧向速度偏差
横向位置偏差变化率
[r_0]
[V_x]
[w_r]
横摆角速度误差
航向角变化率

状态量->控制量

将计算出来的状态量与lqr矩阵混合以后转换成控制量



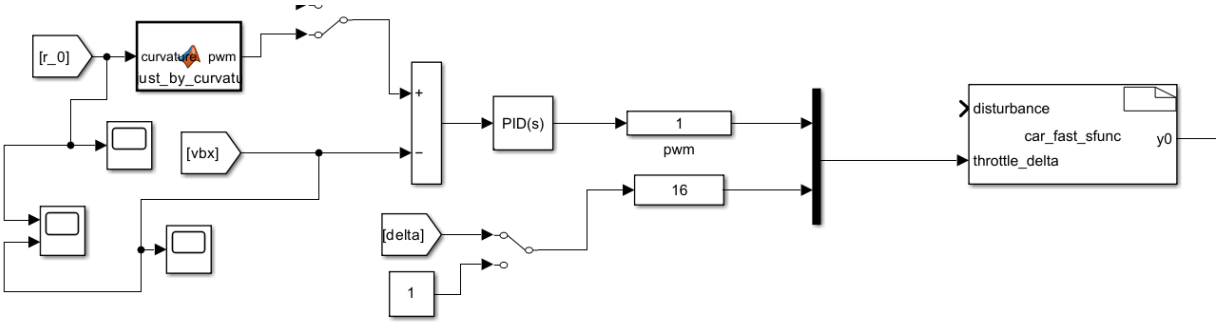
将生成的delta_row做一个0~1的限幅以后，乘一个倍增系数转换到delta，将其作为方向的控制量输入到小车的控制器中去，就完成了小车的横向控制，如下：



纵向控制部分——pid控制器

本模块使用simulink自建的pid控制器，跟踪设定的速度

模型概略图如下：



输入量为当前的曲率*r_0*，设定为曲率越大速度越小，函数如下：

```
function pwm = adjust_by_curvature(curvature)
    % 设定一些常数
    max_output = 3.3;  % 最大速度
    min_output = 0.3;  % 最小速度
    max_curvature = 0.6;  % 最小速度对应曲率阈值

    % 计算反比例函数的值
    inv_proportional = 1 / abs(curvature);

    % 将反比例函数的值映射到0~1的范围内
    inv_proportional = inv_proportional / (1 / max_curvature);
    inv_proportional = min(max(inv_proportional, 0), 1);

    % 将0~1的值映射到0.3~3.3的范围内
    pwm = min_output + (max_output - min_output) * (inv_proportional);
end
```

根据计算出来的速度的设定值，pid控制器会自动控制油门开度使得*Vbx*紧紧跟随设定的速度。实现直线的时候加速狂奔，弯道的时候及时减速。

运行效果展示

运行方式：

- step1: 将matlab工作区文件夹设置为`".\Integrate\traj"`，运行*processing.m*生成一个随机地图
- step2: 运行*py_v2.py*生成路径
- step3: 将matlab工作区文件夹设置为`".\Integrate"`，打开simulink，配置MinGW64后，运行`".\Integrate\car_sim_for_student_r2015.slx"`进行仿真。
- step4: 仿真结束以后会自动调用`".\Integrate\stopFunc.m"`生成一个.gif图像记录轨迹，位于`".\Integrate\trajectory.gif"`

运行结果：

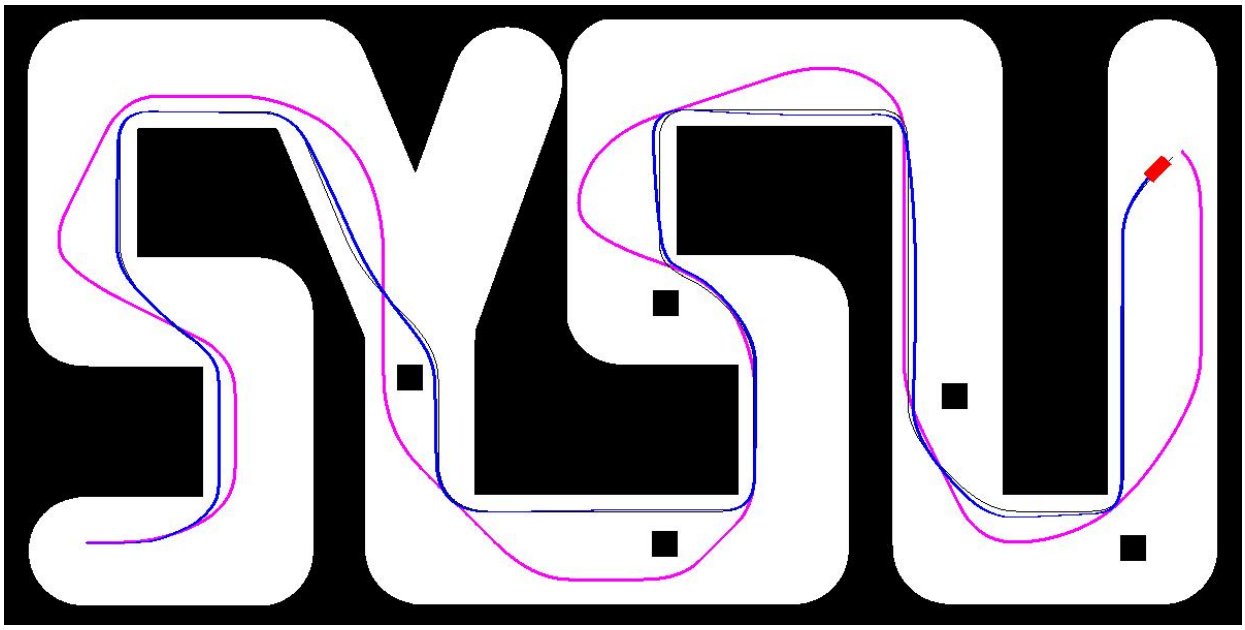
特别声明：所有路线经过本人观测均无碰撞发生

给定地图

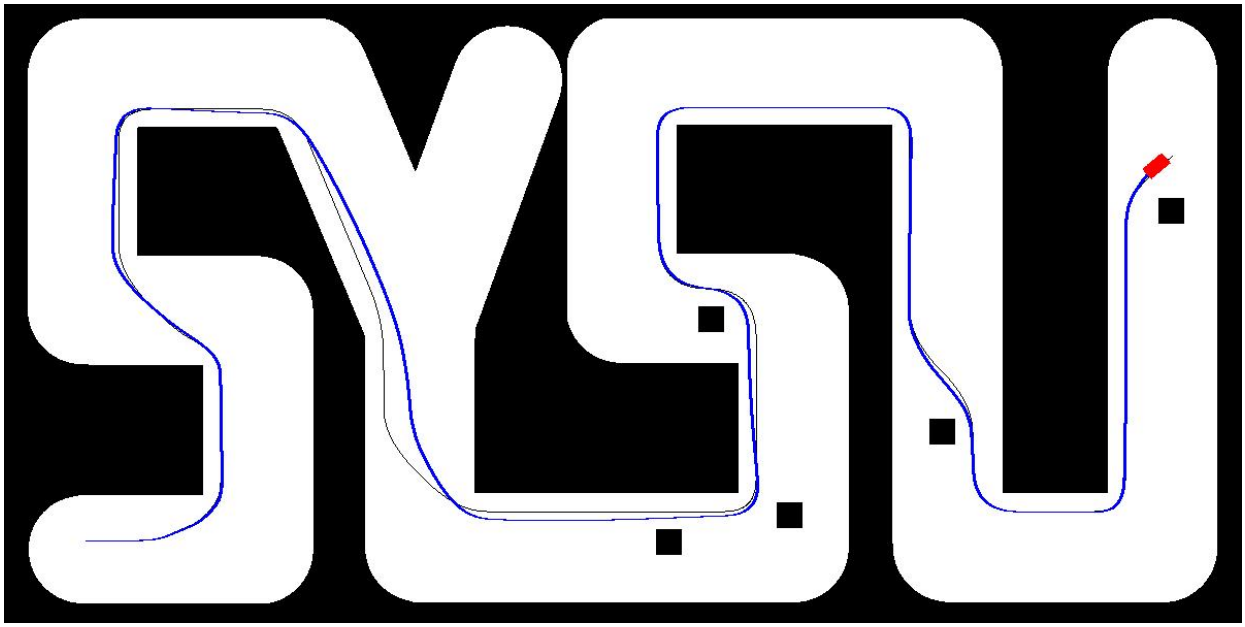
红色线表示助教规划的轨迹

黑色细线是本人规划的轨迹

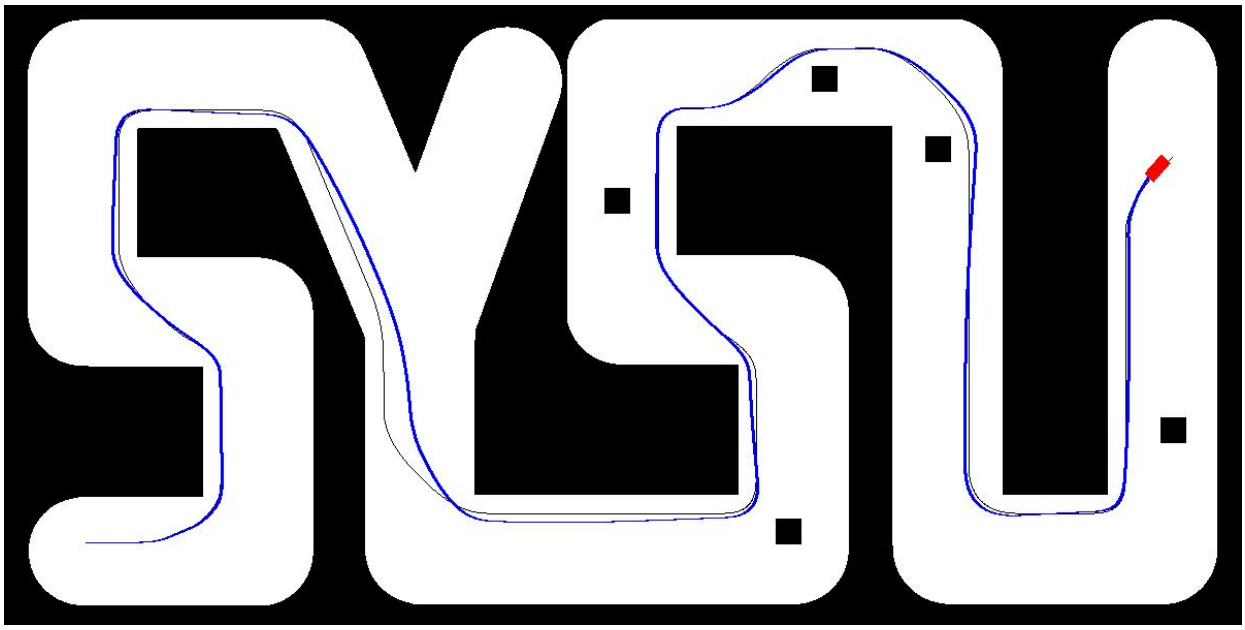
蓝色线是小车的实际轨迹



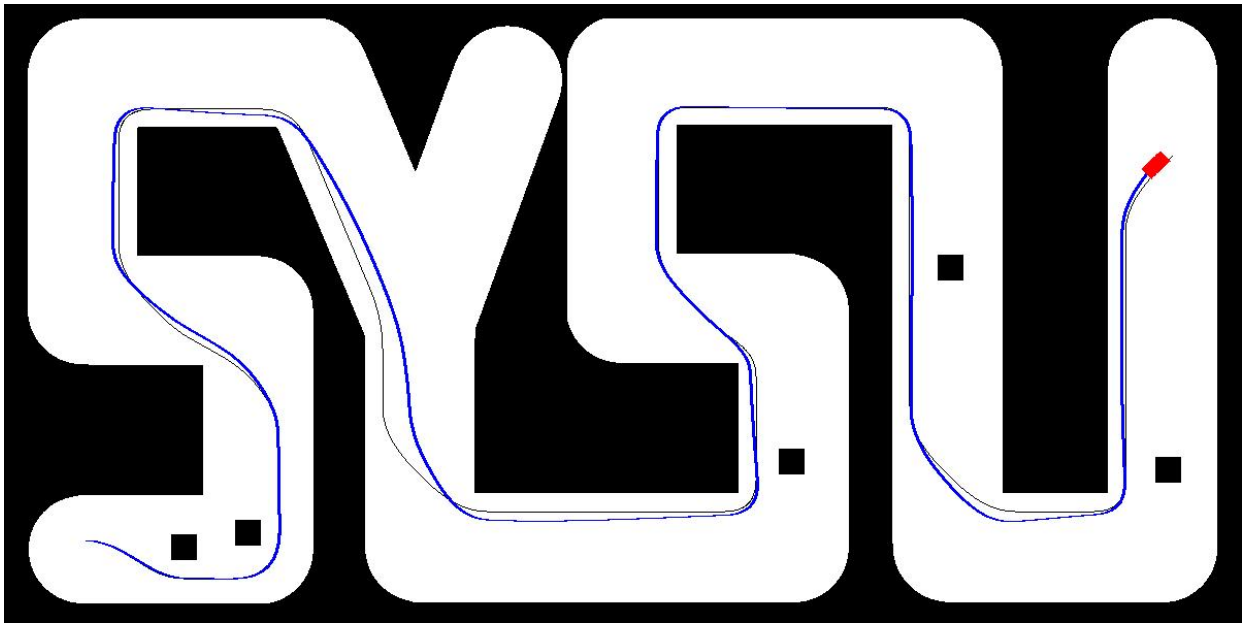
随机地图1



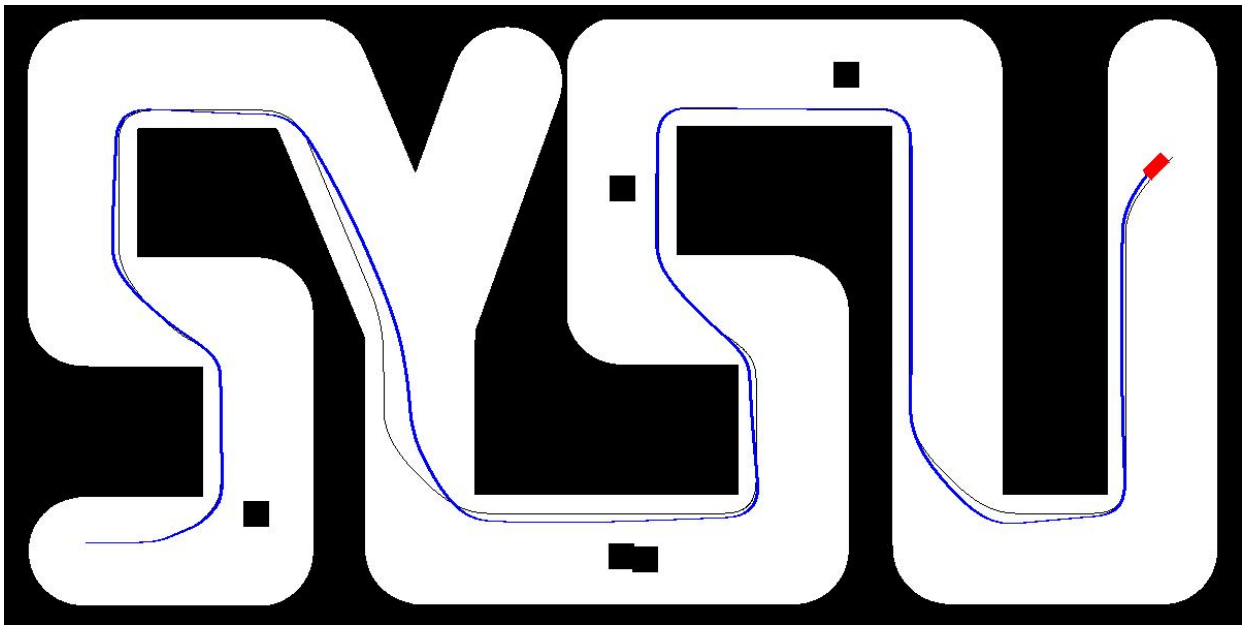
随机地图2



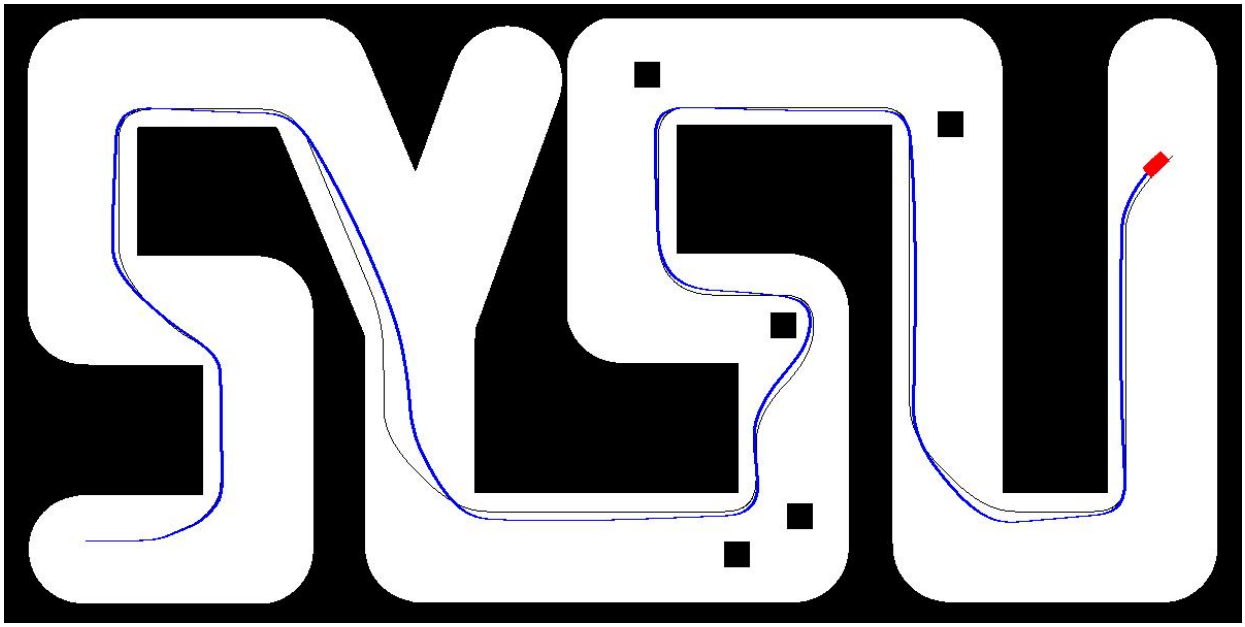
随机地图3



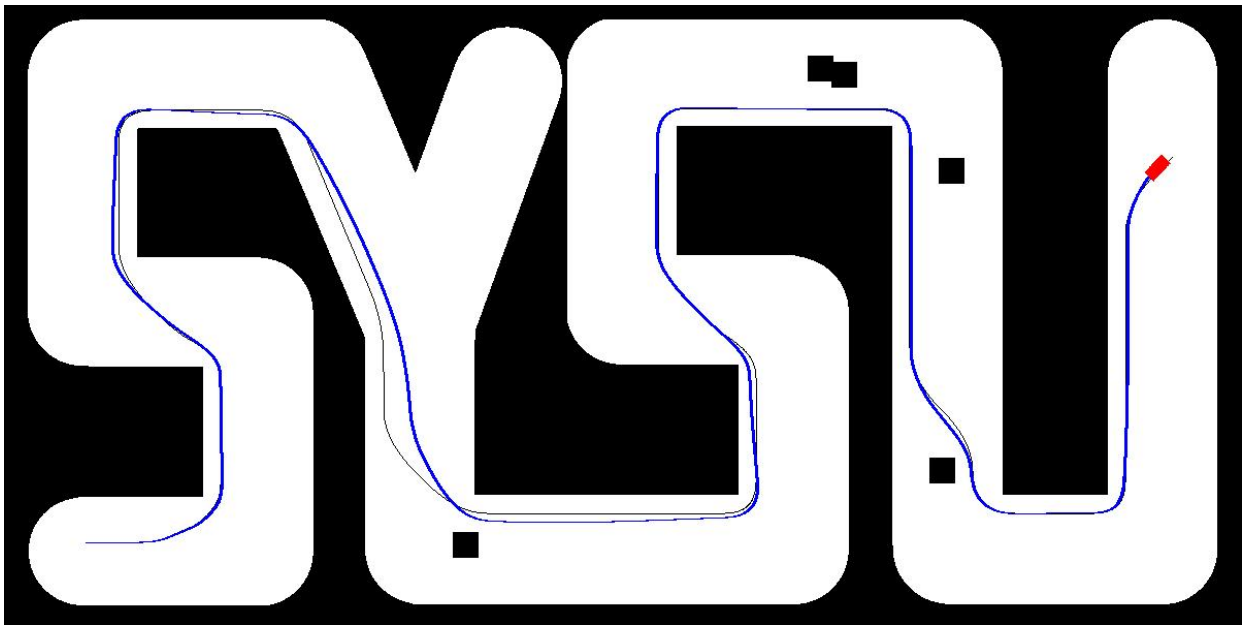
随机地图4



随机地图5



随机地图6



!!! Attention:

总计7次的实验结果均保存在了`.\result`文件夹中，每个结果文件夹均包含了以下数据：

- 运行时的地图数据——`sysu_standard.mat`
- 对应地图规划生成的轨迹数据——`traj_diySYSU.mat`
- 运行窗口截图——`r_x.jpg`
- 运行效果的动态图——`rx_trajectory.gif`

运行时间在gif中都可以看得到

REFERENCE

自动驾驶控制算法：Carsim和Simulink联合仿真实现LQR最优控制轨迹跟踪策略(Greatest contribution)

横向控制器LQR算法详解

横向控制算法与流程图（基于动力学模型的LQR）

【Carsim Simulink自动驾驶仿真】基于MPC的动力学控制

详细介绍用MATLAB实现基于A*算法的路径规划

全局路径规划A star的Matlab实现