第三届 OnSite 自动驾驶算法挑战赛

用户操作指南

（第三赛道：泊车场景）

指导单位：　国家自然科学基金委员会工程与材料科学部

　　中国汽车工程学会

主办单位：　同济大学

　　国家智能网联汽车创新中心

协办单位：　浙江大学

　　上海交通大学

　　清华大学

　　武汉理工大学

二〇二五年三月

Version 2.0 (updated 2025/04/10)

目 录

智能泊车系统由环境感知、中央控制、执行和人机交互等部分组成，其中算法是决定智能泊车系统可靠性的关键。因此，本赛道重点聚焦 AVP 技术，设置与 AVP 技术相关的测试场景，旨在通过系统算法性能分析，识别和改进自动泊车算法的不足，通过对泊车算法的测评、排序及诊断，提出改进策略，制定符合中国国情的泊车规程，并统一测评标准，丰富并完善泊车测试场景库，实现产-学-研精准快速对接。本赛道由清华大学苏州汽车研究院承办。

1 测试场景及赛题

1.1 静态高精度地图介绍

静态停车场地图由表示停车位位置的语义地图和表示可行驶区域的栅格地图两部分组成。对于地图文件 xxxx.json, 按照图层分别进行介绍。

1.1.1 地图基础信息

- 地图基础相关信息，包括版本号、创建日期、地图尺寸及经纬度等。

```
"version": "1.0.7"                    //地图版本号
"map_make_date": "2025-02-28"          //地图创建日期
"map_size": "5100*5600"                //地图大小，单位 cm
```

1.1.2 停车场出入口信息

- 该停车场入口位置和角度，也是泊车测试阶段被测车辆起始点。地图中各图层遵循统一的笛卡尔坐标系（o-xy）定义——
- 原点 O：为(x=0.0, y=0.0)；
- x 轴：与地理的正东方向对应，单位厘米；
- y 轴：与地理的正北方向对应，单位厘米；
- 航向：表示车辆、车位等含有位姿信息的航向统一定义为东偏北，范围为[0,2pi)，单位，rad；

```
"start_position":
    {
        "x": 500,
        "y": 200,
        "yaw": 1.5707963267948966
    }
```

1.1.3 障碍物信息

该停车场地图中静态障碍物（如墙壁、柱子、护角、减速带、标志标牌、限位块、消防栓等）的横纵坐标等位置信息，单位 cm。

```
"obstacles":
    {
        "ox": [...],                    //地图中静态障碍物的横坐标
        "oy": [...]                     //图中静态障碍物的纵坐标
     }
```

1.1.4 停车位信息

车位编号，车位尺寸形态及其对应的横纵坐标、角度、车位出口横纵坐标等信息。

```
"parking_sport": {
      "1":                              //车位编号
        {
            "x_end": 3400,              //车位横坐标
            "y_end": 1850,              //车位纵坐标
            "yaw": 0.0,                 //车位角度
            "x_out": 3700,              //车位出口横坐标
            "y_out": 1650,              //车位出口纵坐标
           "dis": 4400,                 //参考标准距离
            "pos": [                    //车位角点
                5300,
                5900,
                6000,
                5900,
                6000,
                5400,
                5300,
                5400
```

}//(左上角车位点顺时针起，分别对应 x1,y1,x2,y2,…)

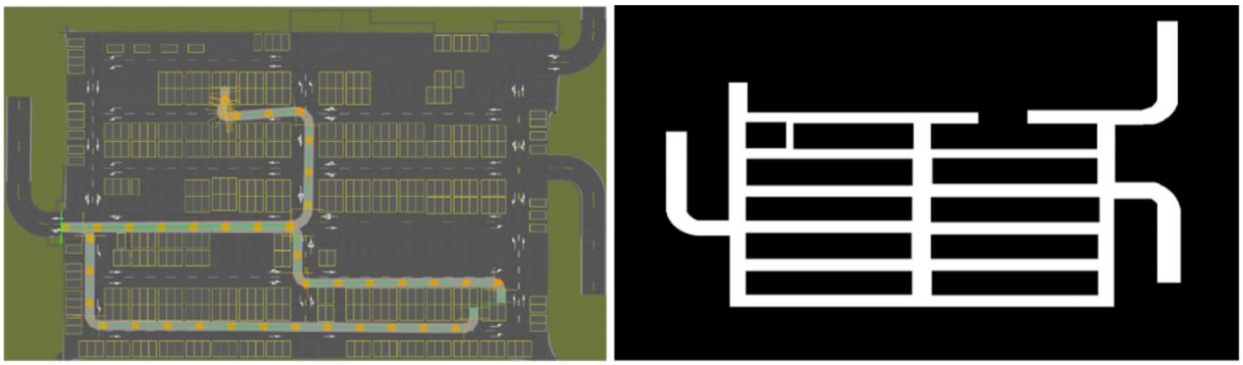


图 1 停车场地图示例

1.2 动态场景格式介绍

动态场景文件记录场景中动态车辆、行人的运动状态信息，用于泊车阶段对被测车辆进行环境干扰。数据包括每个交通目标在离散周期内 0.1s 的轨迹信息，即与选手输出的轨迹一致。动态障碍物每一帧与选手输出轨迹的每一帧进行碰撞检测。数据如下图。

动态场景文件：B01_dyobs.json

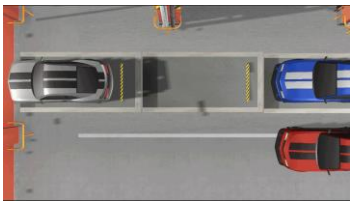
图 2 动态交互车辆运动状态及轨迹的可视化

注：动态场景循环播放。动态场景一次执行完毕，将重复执行第二次，第三次……。直至选手算法执行结束。

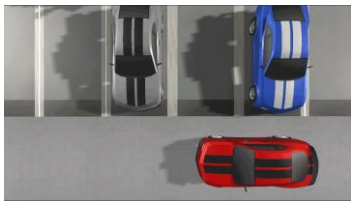
1.3 赛题情况介绍

（1）场景任务介绍

场景类型为停车场不同车位类型组合，为适当提高赛题难度，赛题场景均具有一定风险程度，涉及到的泊入车位类型包括但不限于：双边界车辆平行车位、双边界车辆垂直车位、方柱垂直车位（右）、方柱垂直车位（左）、双边界车辆斜向车位、前方有立柱车位、弯道平行车位、墙角车位，如下图所示。



双边界车辆平行车位



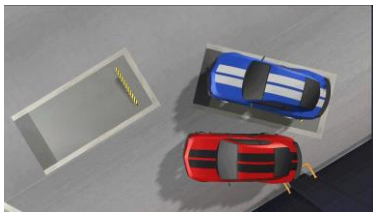
双边界车辆垂直车位

方柱垂直车位（右）

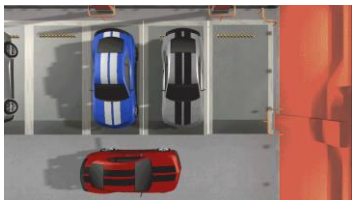
方柱垂直车位（左）

双边界车辆斜向车位

前方有立柱车位



弯道平行车位



墙角车位

图 3 竞赛场景示意图（实际背景车数量以发布场景为准）

（1）双边界车辆平行车位：车辆需要在有限空间内进行泊入，确保与边界之间的距离适当，同时避免与其他车辆相撞，考察车辆在平行车位的规划和控制能力。

（2）双边界车辆垂直车位：车辆需要在狭窄的垂直车位内进行泊入，确保与两侧边界保持适当的距离，测试系统在有限空间内的精准控制能力。

（3）方柱垂直车位-左方/右方：车辆需要在有限的空间内与方柱保持足够的距离进行泊入，同时避免碰撞，考察系统对于障碍物的感知和避障能力，在实际停车场常见且具有一定挑战性。

（4）双边界车辆斜向车位：车辆需要在斜向停车位内进行泊入，避免与两侧边界碰撞，考察系统在非传统停车位布局下的适应能力且具有一定的挑战性。

（5）车位前方有立柱的车位：车辆需要在有立柱阻挡的情况下进行泊入，确保不会与立柱碰撞，模拟现实停车场中常见的情况，考察系统对于障碍物规避的能力。

（6）弯道平行车位：车辆需要在弯道上进行泊入，确保与边界保持适当的距离，考察系统在复杂道路结构下的规划和控制能力。

（7）墙角车位：车辆需要在墙角进行泊入，避免与墙壁碰撞，考察系统在有限空间下的规划和控制能力。

（2）赛题数量及组成介绍

重点面向自主代客泊车 AVP，泊车赛道共设计三套试卷（A 卷/B 卷/C 卷），总计 330 道赛题(以实际发布为准)，在仿真环境中还原真实停车场环境，考察自动泊车系统的规控算法能力。

赛题	作用	数量	组成
A 卷	**供参赛选手训练并校验算法，其成绩不参与排名**	**50**	**含所有车位类型，不包含动态交通目标物，供参赛团队训练并校验算法**
B 卷	**公开考题，供参赛选手提交作品，其成绩参与排名**	**200**	**含所有车位类型，部分试题包含动态车辆/行人，供参赛团队提交作品，其成绩参与排名**
C 卷	**非公开考题，共测试工作人**	**80**	**含所有车位类型，所有**

	员使用，在赛后公开		试题包含动态车辆，作为保留加试题目

2 测试环境安装

2.1 安装 python 编程环境

为了保证参赛队的自动驾驶算法能够在 OnSite-Parking 测试环境中顺利运行，参赛队应安装统一提供的测试环境，并在之中测试自己的算法，保证其能够顺利运行。请注意第三届挑战赛泊车赛道仅面向自动驾驶规划决策控制算法开展。

使用 conda 建立虚拟环境（过 python3.9 版本测试通）

```
conda create -n onsite_parking_py39 python=3.9
```

激活虚拟环境

```
conda activate onsite_parking_py39
```

加载依赖项

```
pip install -r requirements.txt
```

2.2 安装下载 OnSite-Parking

下载 OnSite-Parking 压缩包，解压至本地文件夹，使用任意 IDE 打开，并设置 python 解释器。下载链接：
https://github.com/17863625206/OnSite_Parking/releases/tag/onsiteparking

2.3 测试安装是否成功

为了便于各位参赛者快速上手测试环境，在此提供简单的样例文件(OnSite-Parking)。样例文件的文件结构如下所示

```
└─OnSite-Parking
        ├─ image_save                  # 场景图片存放在位置
        ├─ input
        │    ├─ Atest.json             # 测试地图
```

```
│   ├─Atest_dyobs.json              # 测试动态地图
│   ├─Atest.jpg
│   ├─make_car.py                  # 生成车辆模型
│   └─make_map.py                  # 生成地图模型
├─out
│   result_Atest_01.json           # 输出轨迹存放位置
├─planner
│   ├─hybridastar                  # 规控算法
│   │   ├─__init__.py
│   │   ├─astar.py
│   │   ├─planner.py               # 生成轨迹
│   │   └─planer_reeds_shepp.py
│   ├─example_run.py               # 示例函数
│   └─user_run.py                  # 用户程序运行主函数
└─utils                            # 工具包
    ├─carscoremoudle               # 打分模块
    ├─map_display.py               # 地图展示模块
    ├─draw.py                      # 绘制地图
    ├─drawcar.py                   # 绘制车辆
    └─replay.py                    # 回放方法
```

激活 OnSite-Parking 虚拟环境后，运行 planner 文件下的 test_run.py 文件。如果 outputs 文件夹中，出现测试结果文件，则代表测试环境安装成功。

3 测试运行说明

用户可以仿照我们提供的测试用例，进行算法编写

```
import json
import input.make_map as mp
from planner.hybridastar import planner as planner
from input import make_car
from utils import replay
from utils import map_display as mdp
import matplotlib.pyplot as plt
def  main():
    # 读取路径信息
    map_path = '../input/B01_test.json'          # 读取静态地图
    map_path_dyobs='../input/B01_dyobs.json' # 读取动态地图
    # mdp.map_display(map_path)                 #仅绘制地图
```

```
            park ='3'                                    #选择测试停车位
            ox, oy,sp,gp = mp.make_map(map_path)
            sx, sy, syaw0 = sp['x'], sp['y'], sp['yaw']
            gx, gy, gyaw0 = gp[park]['x_end'], gp[park]['y_end'], gp[park]['yaw']
            # 获取动态障碍物
            dyobsx_x,dyobs_y,dyobs_yaw=mp.get_obs(map_path_dyobs)
            C = make_car.C

            # 规划算法
            path = planner.hybrid_astar_planning(sx, sy, syaw0, gx, gy, gyaw0, ox, oy, C.XY_RESO,
C.YAW_RESO, dyobsx_x,dyobs_y,dyobs_yaw)
            # 算法测试结果保存
            if not path:
                print("Searching failed!")
                return
            output_dit={
                "parking":park,        #目标停车位
                "output_x":path.x,
                "output_y": path.y,
                "output_yaw": path.yaw,
                "output_dir": path.direction,
            }
        with open(f"../Onsite_Parking/output/result_{map_path.split('/')[-1].split('.json')[0]}_{str(i)}.json", "w") as
file:
        json.dump(output_dit, file)
            # 仿真回放
             replay.replay(map_path, parking, path)

    if __name__ == '__main__':
        main()
```

案例解析:

1.读取场景信息

```
    import input.make_map as mp
    map_path = '../input/A_test.json' # 读取地图文件
    # mdp.map_display(map_path) #   仅绘制地图
    park ='3'    # 选择测试停车位
    ox, oy, sp, gp = mp.make_map(map_path)    #读取地图信息
    sx, sy, syaw0 = sp['x'], sp['y'], sp['yaw']
    gx, gy, gyaw0 = gp[park]['x_end'], gp[park]['y_end'], gp[park]['yaw']
    dyobsx_x,dyobs_y,dyobs_yaw=mp.get_obs(map_path_dyobs)# 获取动态障碍物
```

其中 make_map()为解析地图工具，返回内容中 ox，oy，为一维数组，代表所有障碍物的位置；sp 为泊车起点位置，由车辆起点横坐标 x，纵坐标 y，偏航角 yaw 三个属性组成；gp 为所有停车位信息，包括车位中心点的横坐标、纵坐标，朝向，车位四个顶点坐标，车位出口横坐标，车位出口纵坐标，以及从入口到该停车位的标准距离信息，具体可参考场景格式介绍。dyobs_x, dyobs_y, dyobs_yaw 分别为动态障碍物运行的横坐标，纵坐标，朝向。make_map()完整程序如下所示：

```
import json
  def make_map(map_path):
      with open(map_path,'r',encoding='UTF-8') as f:
          map=json.load(f)
      ox = map['obstacles']['ox']
      oy = map['obstacles']['oy']
      sp = map['start_position']
      gp = map['parking_sport']
      return ox, oy, sp, gp
```

make_car.C 主要用于获取车辆属性，C 为车辆属性设置详细参考车辆参数介绍，具体程序如下：

```
C = make_car.C
import math
import numpy as np
# Parameter config
class C:
      PI = math.pi
      XY_RESO = 1000    # [cm]
      YAW_RESO = np.deg2rad(5.0)    # [rad]
      MOVE_STEP = 20    # [cm] path interporate resolution
      N_STEER = 20.0    # steer command number
      COLLISION_CHECK_STEP = 5    # skip number for collision check
      EXTEND_BOUND = 20    # collision check range extended
      GEAR_COST = 100.0    # switch back penalty cost
      BACKWARD_COST = 5.0    # backward penalty cost
      STEER_CHANGE_COST = 5.0    # steer angle change penalty cost
      STEER_ANGLE_COST = 1.0    # steer angle penalty cost
      H_COST = 15.0    # Heuristic cost penalty cost
      RF = 450    # [cm] distance from rear to vehicle front end of vehicle
      RB = 100    # [cm] distance from rear to vehicle back end of vehicle
```

```
        W = 300  # [cm] width of vehicle
        WD = 0.7 * W  # [cm] distance between left-right wheels
        WB = 350  # [cm] Wheel base
        TR = 50  # [cm] Tyre radius
        TW = 100  # [cm] Tyre width
        MAX_STEER = 0.6  # [rad] maximum steering angle
```

规划算法，当获取障碍物信息，车辆起点，车辆目标车位，以及车辆属性信息后；用户需要自行编写规控算法，规控算法最后输出内容为相同时间间隔（0.1s）下的每个位置点信息（包括横坐标（x）,纵坐标（y）,车辆偏航角（yaw），以及车辆行驶方向（direction，前进为 1，后退为-1））

```
path = planner.hybrid_astar_planning(sx, sy, syaw0, gx, gy, gyaw0, ox, oy, C.XY_RESO, C.YAW_RESO)
```

```
class Path:
    def __init__(self, x, y, yaw, direction):
        self.x = x
        self.y = y
        self.yaw = yaw
        self.direction = direction
```

2.算法测试结果保存至 output 文件夹下

注意：输出文件命名格式为 result_试卷名_车位.json,如 result_B01_3.json(B01 卷 3 号停车位规划结果值)

```
if not path:
    print("Searching failed!")
    return
output_dit={
    "parking":park,              #目标停车位
    "output_x":path.x,
    "output_y": path.y,
    "output_yaw": path.yaw,
    "output_dir": path.direction,
    }
with open(f"../Onsite_Parking/output/result_{map_path.split('/')[-1].split('.json')[0]}_{str(i)}.json", "w") as file:
json.dump(output_dit, file)
# 仿真回放
# 调用 replay 场景回放方法，输入参数为地图场景路径、目标停车位、算法测试结果保存路径
```

```
from utils import replay
replay.replay(map_path, parking, path)
```

3.打分工具

运行我们提供的 calscoremodel 模块进行打分，输入场景文件、规划轨迹结果，点击计算即可看到单项场景分数（打分时间可能较长）。

4.测试更新逻辑

目前采用基础的自行车模型。当 t→t+Δt 时刻，得到本车加速度α与前轮转角δ作为输入，按照以下步骤更新车辆状态：

（1）更新本车位置

$$x_{t+\Delta t} = x_t + v_t \times cos(\theta_t) \times \Delta t$$

$$y_{t+\Delta t} = y_t + v_t \times sin(\theta_t) \times \Delta t$$

其中：

x_t、y_t为当前时刻车辆的 x 坐标与 y 坐标

$x_{t+\Delta t}$、$y_{t+\Delta t}$为下一时刻车辆的 x 坐标与 y 坐标

v_t为当前时刻的车辆速度

θ_t为当前时刻的车辆偏航角

Δt为一个步长的时间长度

（2）更新本车偏航角

$$\theta_{t+\Delta t} = \theta_t + v_t/l \times tan(\delta_t) \times \Delta t$$

其中：

$\theta_{t+\Delta t}$为下一时刻的车辆偏航角
δ_t为前轮转角
l为车辆轴距
v_t为当前时刻的车辆速度
未说明的符号，含义与前文一致

（3）更新本车速度

$$v_{t+\Delta t} = v_t + \alpha \times \Delta t$$

其中：

$v_{t+\Delta t}$为下一时刻的车辆速度
α为加速度
v_t为当前时刻的车辆速度
未说明的符号，含义与前文一致

注：自车速度阈值：最大为 13.5m/s(约 48Km/h)。

4 上传步骤

参赛队应采取 Docker 镜像(docker image)作为结果上传格式。Docker 是一个虚拟环境容器，可以将开发环境、代码、配置文件打包，并发布和应用到任意平台中，保证了测试环境的通用性和一致性。

Docker 的基本教程和安装方式可以参考官方网站 (Orientation and setup | Docker Documentation)。本章将以样例文件(onsite-parking)为例，介绍如何发布与上传最基本的 Docker 镜像。

本届赛事采用阿里云服务器提交 Docker，具体请访问：https://www.aliyun.com/product/acr。

4.1 获取阿里云服务器账号

（1）登录账号

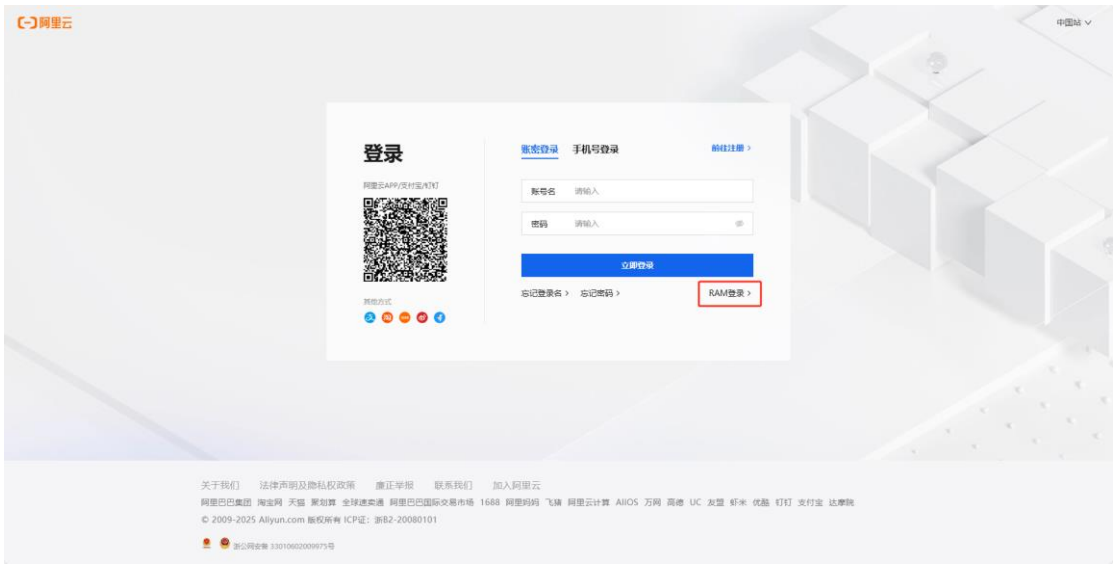
首先在 OnSite 官网进入个人中心，点击“我的团队”获取阿里云 ACR 账号

和密码。

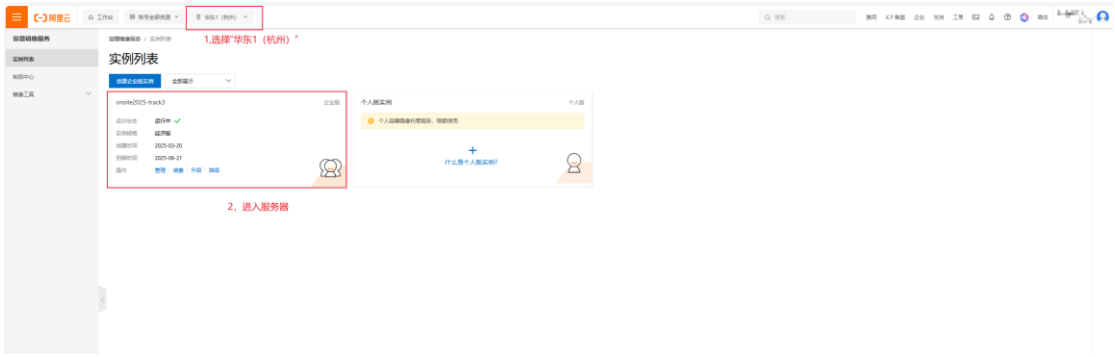
账号名：<ACR-name>@1432801272907066.onaliyun.com

密码：<ACR-key>

之后，进入阿里云 ACR 网站：https://www.aliyun.com/product/acr 登录账号选择“RAM 登录”，然后点击“管理控制台”。

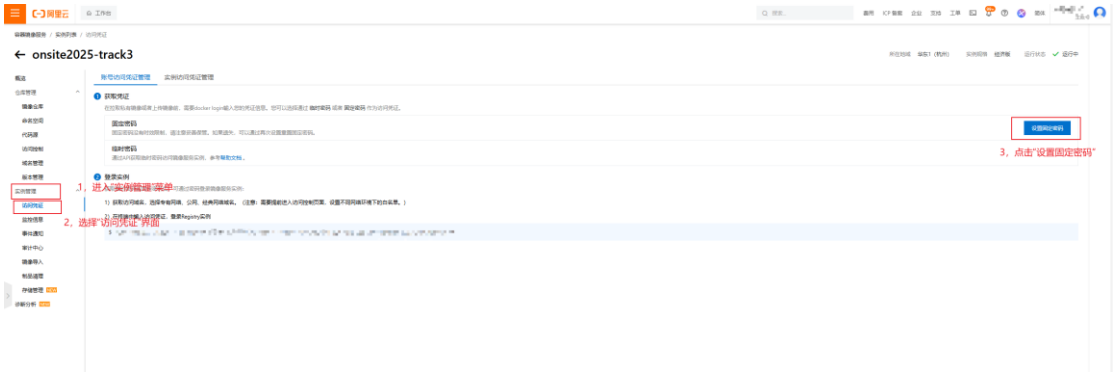


确认节点为“华东 1（杭州）”后点击进入“onsite2025-track3”服务器。



（2）设置访问凭证密码

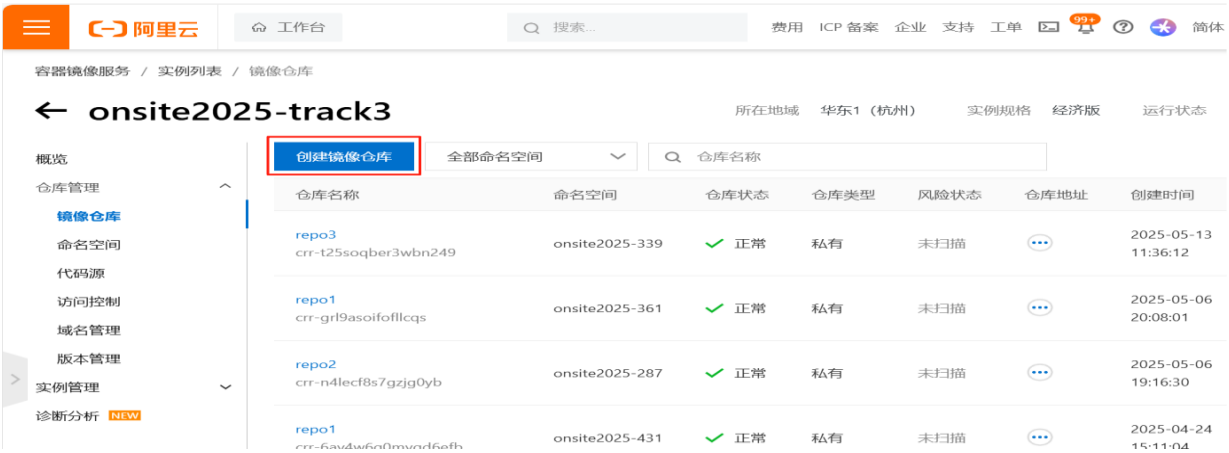
在左侧菜单栏选择“实例管理”-“访问凭证”,之后点击“设置固定密码”。注：此步非常重要，该操作设置上传 Docker 时链接服务器的密码，与 5.2（4）相关。

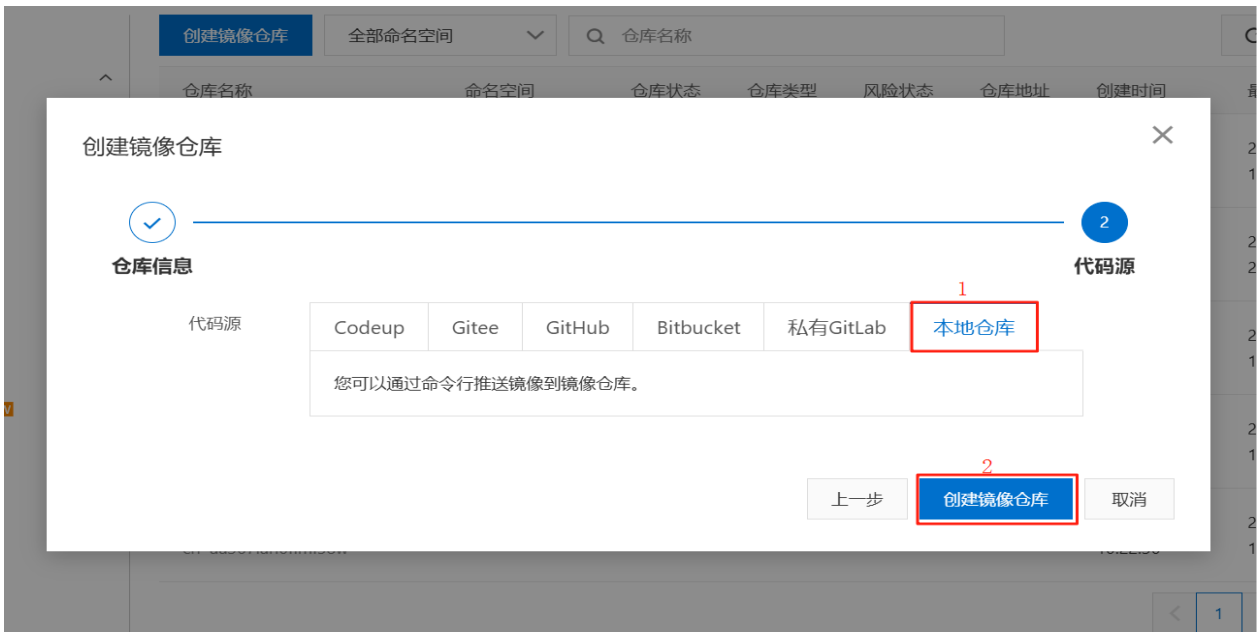


（3）创建镜像仓库

在左侧菜单选择“镜像仓库”，之后点击“创建镜像仓库”，选择自己组

作为命名空间，仓库名称(如 repo1,repo2),点击下一步，之后点击“本地仓库”，最后点击“创建镜像仓库”。





4.2 制作 Docker 镜像

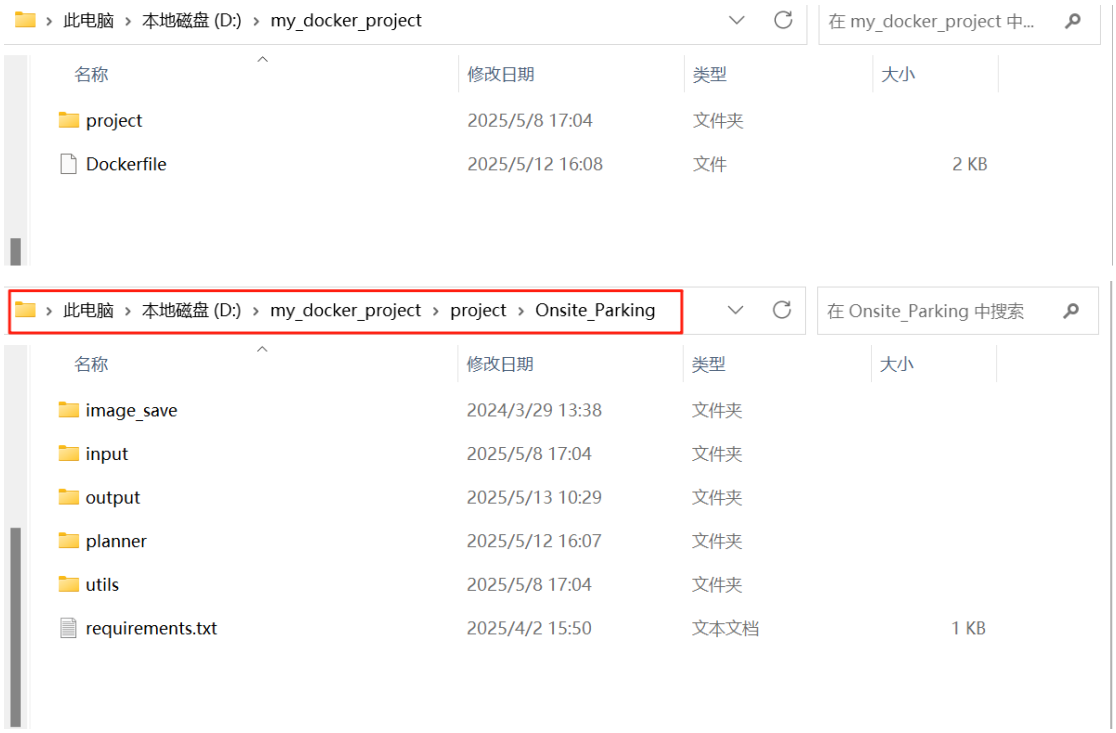
1，确保你的系统已安装 Docker，可以通过 docker --version 命令来检查

2，在本地创建一个新的目录，在该目录下创建 Dockerfile 文件，内容如下：

```
# 使用 Ubuntu 20.04 作为基础镜像
FROM ubuntu:20.04

# 设置环境变量，避免交互提示
ENV DEBIAN_FRONTEND=noninteractive

# 更新软件源并安装必要的软件包
RUN apt-get update && apt-get install -y \
    software-properties-common \
    && add-apt-repository ppa:deadsnakes/ppa \
    && apt-get update \
    && apt-get install -y python3.9 python3.9-distutils wget\
    && apt-get clean \
    && rm -rf /var/lib/apt/lists/*

# 安装 setuptools
RUN wget https://bootstrap.pypa.io/ez_setup.py -O - | /usr/bin/python3.9

RUN update-alternatives --install /usr/bin/python3 python3 /usr/bin/python3.9 1

# 安装 pip，设置超时时间和使用国内镜像源
```

```
RUN wget https://bootstrap.pypa.io/get-pip.py
RUN /usr/bin/python3.9 get-pip.py --default-timeout=100 -i https://pypi.tuna.tsinghua.edu.cn/simple
RUN rm get-pip.py

# 复制本地的 project 目录到镜像的工作目录
COPY project/Onsite_Parking .

# 安装依赖 requirements.txt，使用国内镜像源
RUN if [ -f requirements.txt ]; then /usr/bin/python3.9 -m pip install -r requirements.txt -i https://pypi.tuna.tsinghua.edu.cn/simple; fi
```

3，将本地的 project 目录放置在与 Dockerfile 相同的目录下，确保 project 目录包含你的 Onsite_Parking 中项目所需的所有文件。



4，在包含 Dockerfile 和 project 目录的目录下，打开终端，执行以下命令构建镜像（注意后面的“.”）：

```
docker build -t myonsiteparkingproject:v0321 .
```

-t 选项用于为镜像指定标签，最后的.表示使用当前目录下的 Dockerfile 进行构建

运行下面命令，如果可以找到 myonsiteparkingproject,则证明镜像创建成功

```
docker images
```

5，构建完成后，通过以下命令试运行容器：

```
docker run -it --rm myonsiteparkingproject:v0321
```

输入指令“ls”查看其中文件，确保 input,output,planner,utils 等项目文件在镜像的主目录文件下。

```
(base) PS C:\Users\97301> docker run -it --rm  myonsiteparkingproject:v0324
root@dc7a233d6f4b:/# ls
bin   etc         input  lib64   mnt     planner           root  setuptools-33.1.1.zip  tmp    var
boot  home        lib    libx32  opt     proc              run   srv                    usr
dev   image_save  lib32  media   output  requirements.txt  sbin  sys                    utils
root@dc7a233d6f4b:/#
```

4.3 推送镜像至阿里云仓库

1，在终端执行以下命令，使用阿里云账号登录镜像仓库：

```
docker login --username=你的阿里云账号 onsite2025-track3-registry.cn-hangzhou.cr.aliyuncs.com
```

输入密码完成登录。

2，将本地构建的镜像打上阿里云镜像仓库的标签

```
docker tag [ImageId] onsite2025-track3-registry.cn-hangzhou.cr.aliyuncs.com/ [命名空间]/repo2:[镜像版本号]
```

- 将[ImageId]替换为镜像 ID 如 myonsiteparkingproject:v0321
- 修改命名空间如 onsite2025-431
- 仓库名称，须在阿里云 ACR 中创建(步骤 4.2)，名称规定为 repo1、repo2、repo3……
- 镜像标签，标注镜像的版本信息。

注：为了防止他人抄袭或直接上传您编写的代码，请在 tag 命名镜像标签时采用复杂名称，并且避免在名称中体现出 OnSite 等比赛相关的内容：如队伍名称或算法名称。

3，测试镜像文件运行

```
docker run  --rm -w / -v [output_path]:/output -v [input]:/ input [镜像 ID] python3 planner/user_run.py
```

output_path:在本地创建空的文件夹，目的是将镜像输出结果挂载到本地文件

input:为本地的场景文件，即 Onsite_Parking 项目下的 input 目录

镜像 ID: 如 myonsiteparkingproject:v0321

当 output_path 中有输出结果，即代表运行成功。

注：选择挂载的目的是因为需要在服务器运算中使用挂载的方式测试镜像

4，执行以下命令将镜像推送到阿里云镜像仓库：

```
docker push onsite2025-track3-registry.cn-hangzhou.cr.aliyuncs.com/[命名空间]/repo2:[镜像版本号]
```

4.4 上传至 OnSite 网站

（1）进入 OnSite 官网，登录选手账号，点击进入作品管理界面

（2）点击作品上传，进入作品上传界面

（3）填入打包好的镜像 ID 并选择相应的赛道，镜像 ID 如：onsite2025-track3-registry.cn-hangzhou.cr.aliyuncs.com/onsite2025-431/repo2:test

（4）点击确定后，作品栏显示最新上传记录即为上传完成

5 评价体系

排名标准以试题完成率为第一判断指标，试题完成率越高，排名越高（例如 B 卷共 100 套试题，完成试题数目越多，排名越高），在相同完成率的情况下，以完成效果得分进行二次排名，完成效果得分为已完成试题的总得分（例如共 5 题，完成了其中三道题目，每题得分为 80，90，100，则完成效果得分为 270 分）。

1. 试题是否完成判断

（1）是否发生碰撞冲突行为，若发生视为未完成

（2）是否停入目标停车位内，若未停入视为未完成

2. 单个试题得分标准

（1）效率得分（60 分）

■ 泊车路程（30 分）：

计分公式：参考路程/行驶路程*20（最高 30 分，最低 0 分）

行驶路程：被测车辆从出发点位置到终止点位置的行驶总路程

参考路程：出发点位置到目标车库中心位置的参考路程距离

■ 揉库次数（30 分）：

揉库次数以每次切换行驶方向为判断依据，揉库一次不扣分，每多一次，扣 3 分，30 分扣完为止

（2）停车位置姿态得分（40 分）

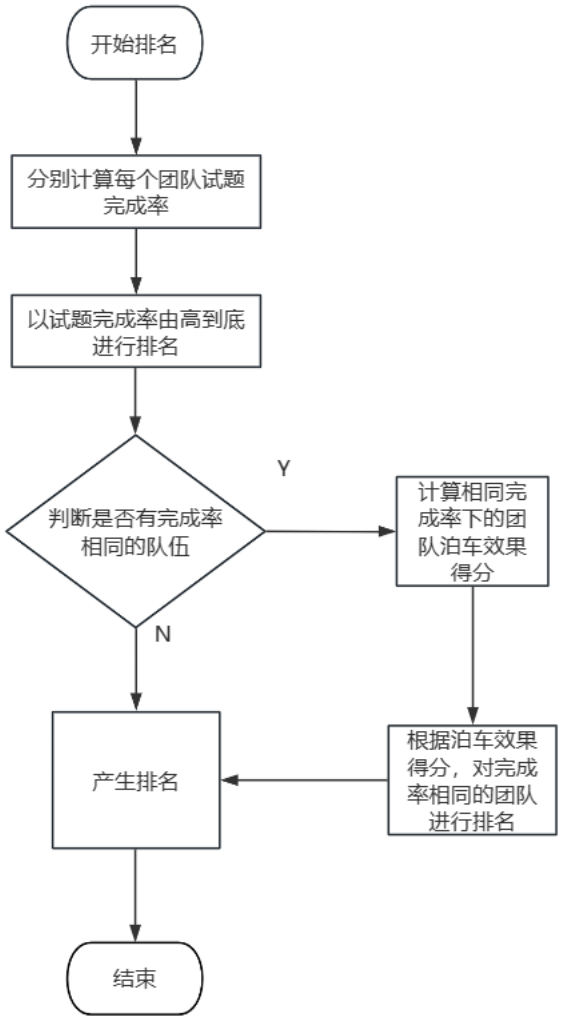
■ 停车位置（20 分）：

车辆中心点距离停车位中心点距离，每差 2cm，扣 3 分,（共 20 分，扣完为止）

■ 停车姿态（20 分）：

车辆最终朝向与停车位朝向偏差，每差 0.5 度，扣 3 分，（共 20 分，扣完为止）

具体操作步骤如图所示：



评价体系流程图

6 具体操作案例

本案例以样例文件(OnSite-parking.zip)为例，本案例中所有代码均已公开，参

赛选手可自行下载体验，详细过程说明如下。

部署环境：windows 11

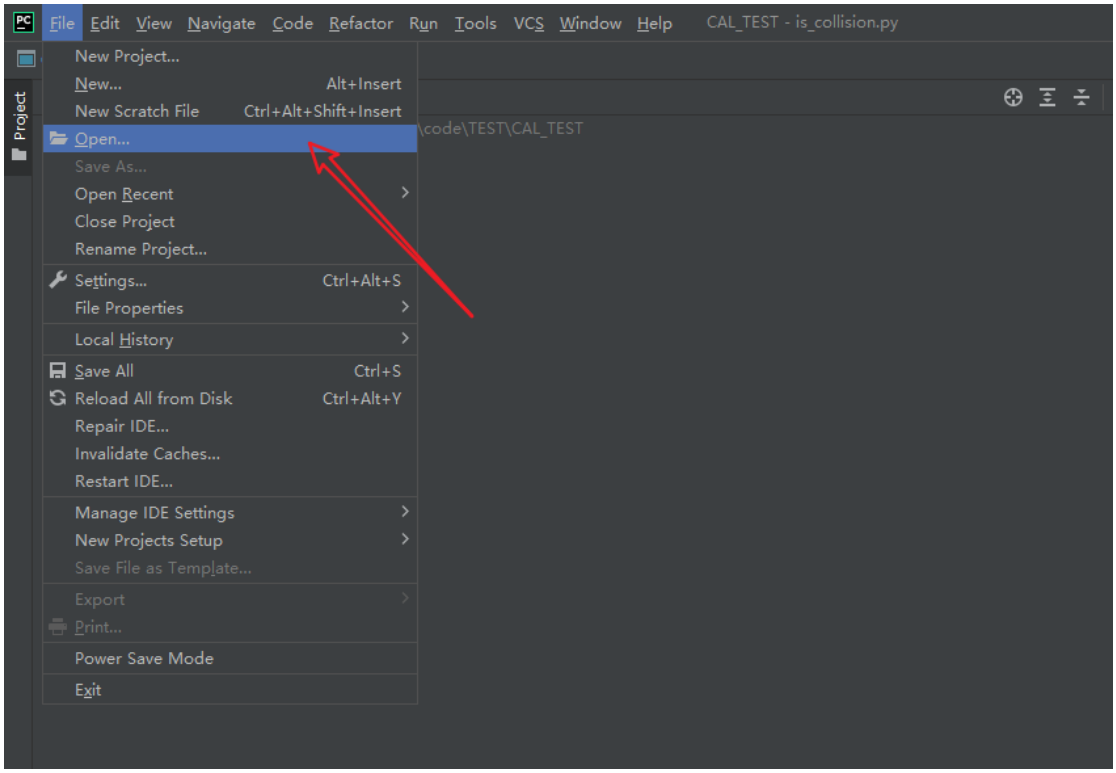
样例文件：OnSite-parking.zip

（1）按第 2 章的说明安装 python 环境

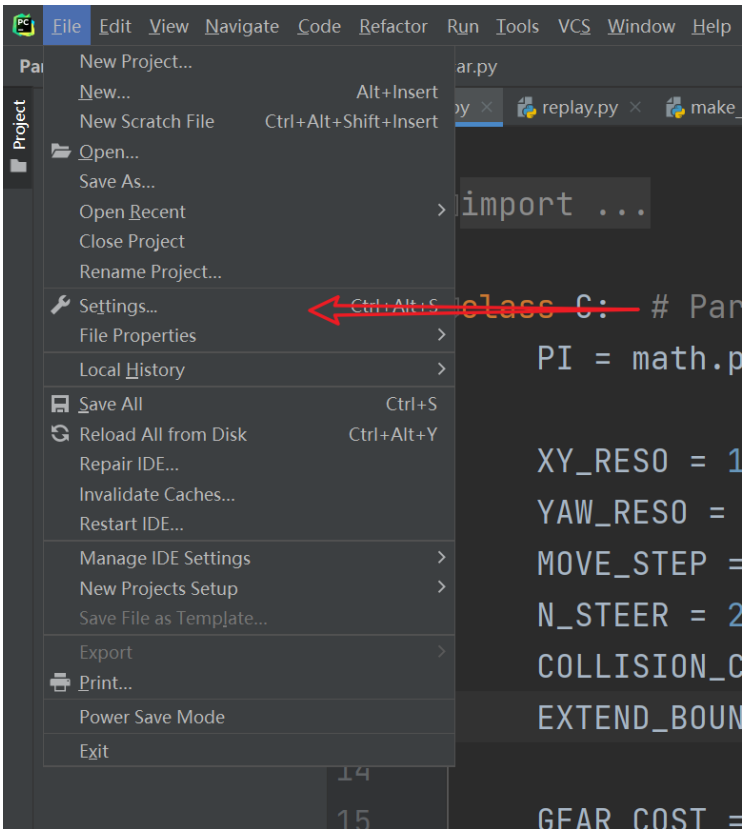
（2）下载获取样例文件，并解压

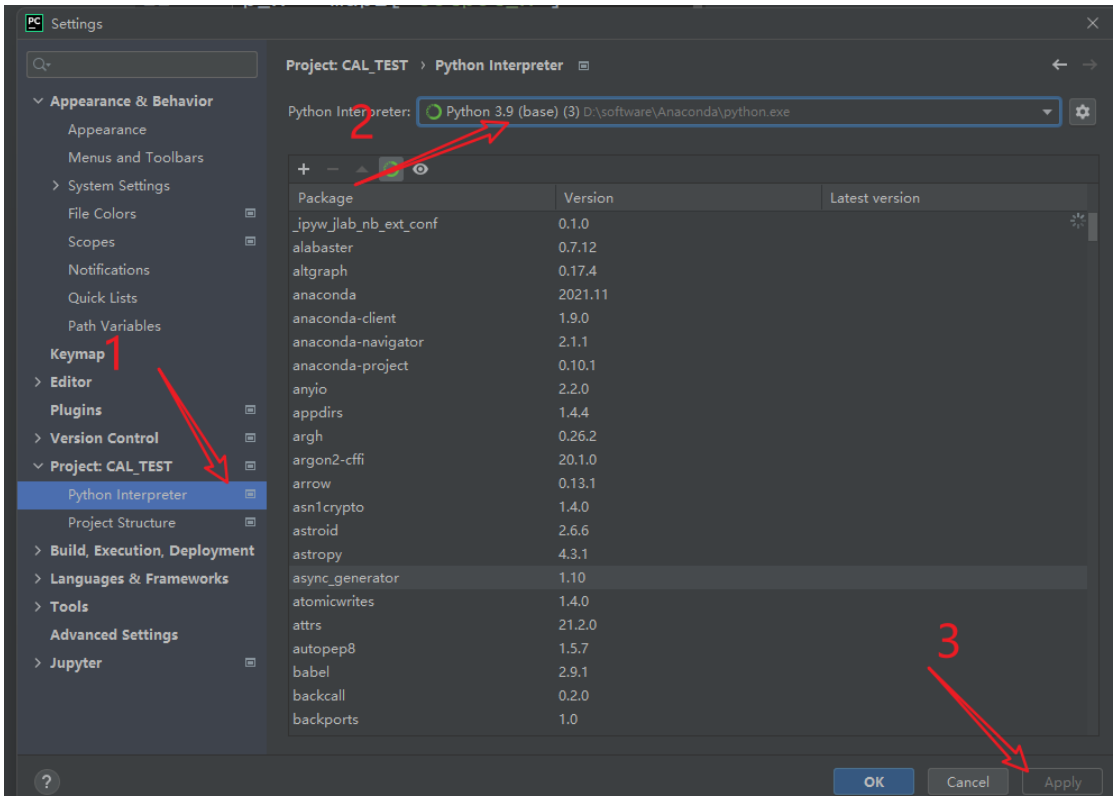
（3）运行样例文件

- 打开 Onsite_Parking 文件



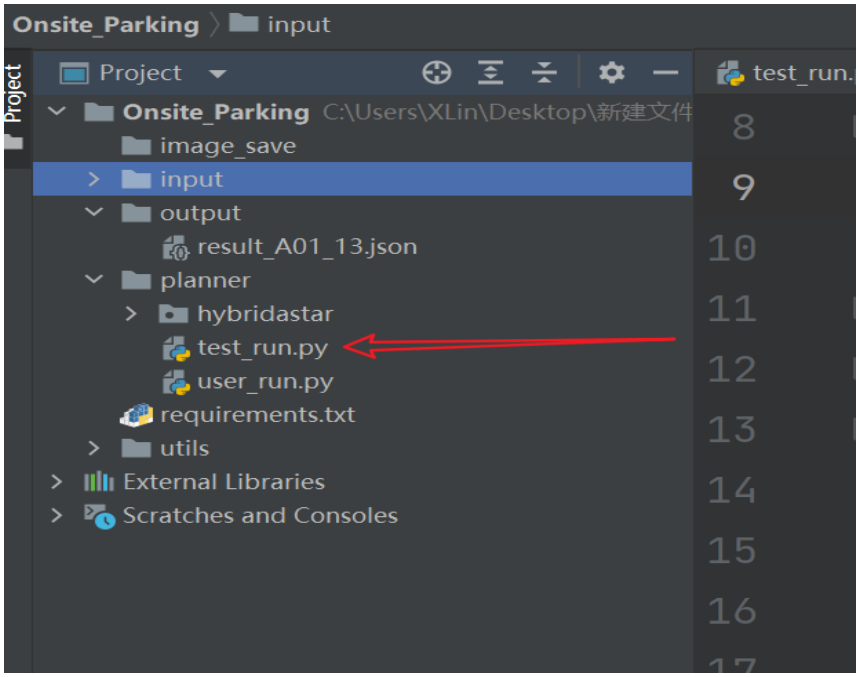
- 设置 python 运行环境





- 安装 requirements.txt，运行/planner/user_run.py，样例文件开始运行，如若

output 文件夹内又内容输出，说明安装成功。



（4）备注

- 替换 inputs 文件夹中的文件，即可更改测试场景。
- 样例文件(test_run.py)仅为示例性的工具，他仅仅选择了一张地图里面的一个停车位作为演示
- 用户需要在 user_run 里面书写运行程序，切记输出结果内容格式。
- **用户需在打包镜像前将 user_run 中的仿真回放代码进行注释，以防止在打分时运行**

```
46
47      # 仿真回放
48      # result_path = f"./output/result_{map_path.split('/')[-1].split('.json')[0]}_{park}.json"
49      # replay.replay(map_path, result_path,map_path_dyobs)
50  if __name__ == '__main__':
51      main()
```

（5）打包上传镜像

参考第四章内容