第二届低空经济智能飞行管理挑战赛
性能赛比赛平台使用说明

（20240822）

1. 比赛概述

1.1. 比赛内容

比赛设置了三个主要作业区：航空作业区、地勤作业区和用户机场。参赛者需要利用小车和仿真机来完成送餐任务。具体内容如下：

航空作业区

仿真机的起飞和降落将在航空作业区的小车上进行。

参赛者需要协调小车和仿真机的起降操作。

地勤作业区

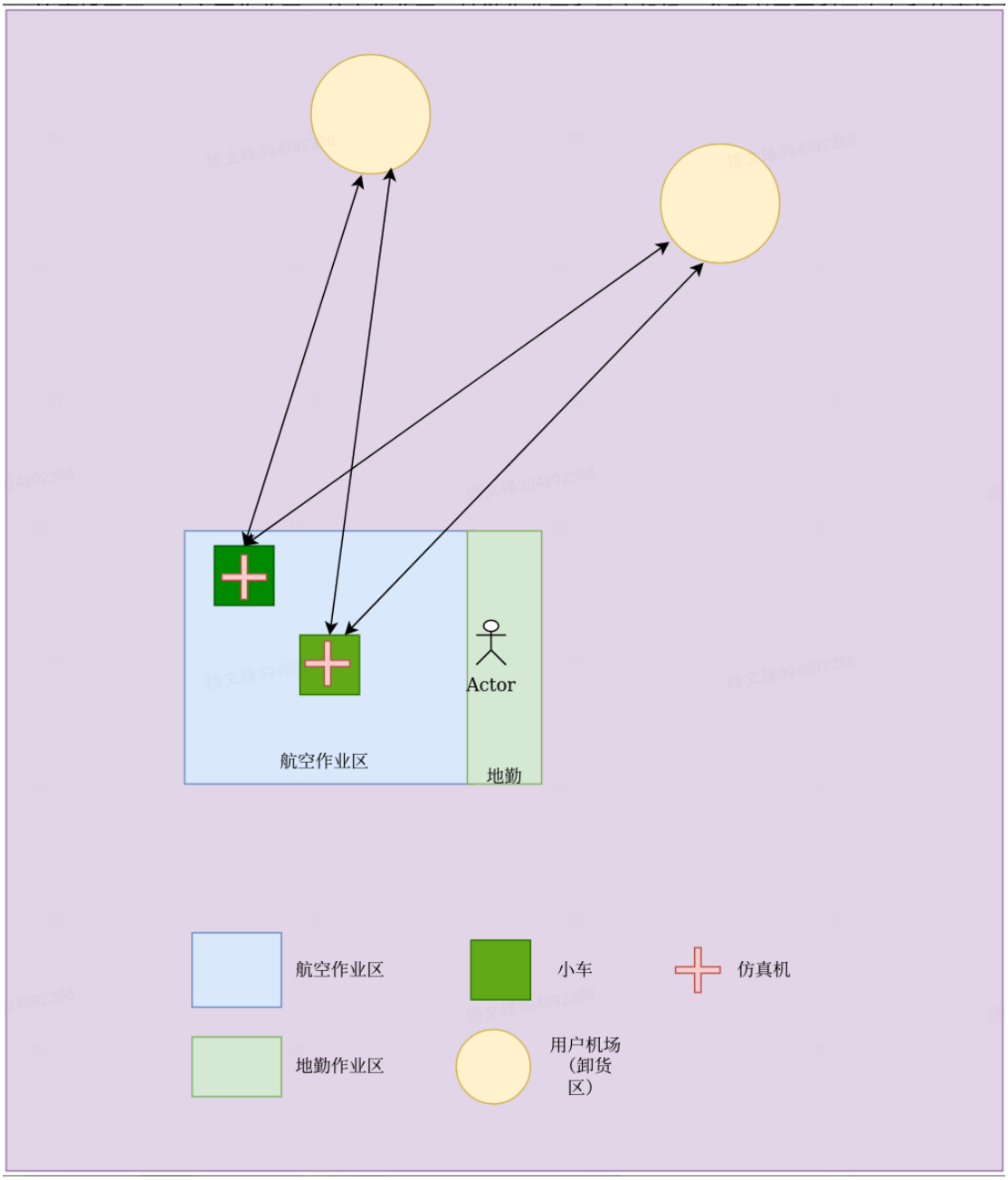
给飞机装货、换电等操作将在地勤作业区内执行。

参赛者需要安排小车和地勤设备进行高效的地面服务。

用户机场

订单的目的地，卸货之后完成订单。

比赛的核心目标是合理协调机场内的小车和飞机操作，在限定的时间内完成尽可能多的订单配送。参赛者需要展示出高效的调度和操作能力，以最大化送餐任务的完成数量。



1.2. 比赛系统架构

比赛系统启动后，将会启动四个 Docker 容器，分别是：

参赛选手 Docker

提供 ROS（机器人操作系统）开发套件。

参赛选手需要在此容器内编写算法以完成比赛任务。

场景管理 Docker

负责比赛场景的整体管理和协调，完成算分等操作

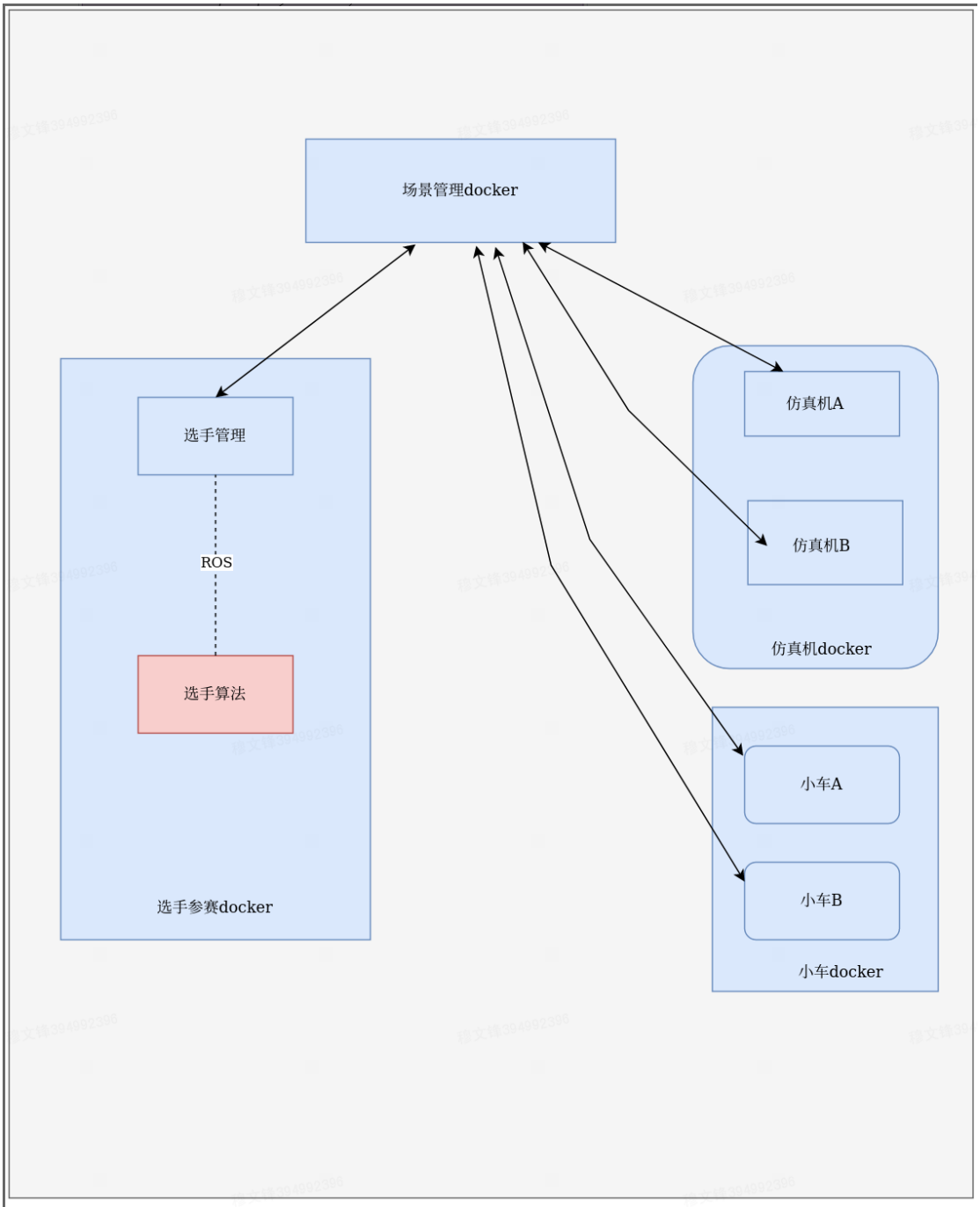
仿真机 Docker

模拟仿真机的操作，包括起飞、降落和飞行过程。

小车 Docker

模拟小车的操作，用于地面运输和服务。

通过这四个 Docker 容器的协同工作，比赛系统能够高效地模拟和管理整个比赛过程，参赛选手需要在参赛选手 Docker 中开发和运行算法，以实现最佳的比赛表现。


场景管理docker
选手管理
ROS
选手算法
选手参赛docker
仿真机A
仿真机B
仿真机docker
小车A
小车B
小车docker

服务器

2. 系统与环境配置

2.1. 系统要求

本地推荐配置

操作系统：Ubuntu20.04

硬件要求

最低要求

CPU：四核 Intel 或 AMD 移动处理器或更好（即 Intel i5-8550U ，主频 1.8GHz 或 AMD Ryzen 5 3500U，主频 2.1GHz）

GPU：Intel(R) UHD Graphics

内存：8GB（本机 Ubuntu）/16GB（虚拟机）

存储：100GB（强烈推荐 SSD）

推荐要求

CPU：8 核 （主频 2.5GHz）

内存：16GB

存储：100GB（强烈推荐 SSD）

对于本地算法开发，配置越高越好

线上运行环境

操作系统：Ubuntu20.04

CPU：32 核 （主频 2.5GHz）

内存：48GB

存储：500GB SSD

2.2. 环境设置

建议使用官方脚本自动配置 Docker 环境

```
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
```

也可自行安装（需谷歌）

验证安装是否成功

```
docker --version   # 检查 Docker 版本
```

输出如下表示成功

```
docker run hello-world
```

正常情况下，输出如下

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
1b930d010525: Pull complete
Digest: sha256:fc6cf906cbfa4b9daeea7b6e0b8a0e3c4e5d7a7f2a9f1a1a1b1a1b1a1b1a1b1a
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.
```

2.3. 单机版下载及运行

2.3.1. 下载 SDK 镜像

新建脚本 race_images.sh

```
#!/bin/bash


# 登陆到腾讯云 docker 服务
docker login uav-challenge.tencentcloudcr.com --username 'tcr$user' --password
gXWWpxhO9igRnXzYYV58UexxS1Gw8VQY


# 要拉取的镜像列表
images=(
    "uav-challenge.tencentcloudcr.com/uav_challenge_2024/sdk:cars"
    "uav-challenge.tencentcloudcr.com/uav_challenge_2024/sdk:drones"
    "uav-challenge.tencentcloudcr.com/uav_challenge_2024/sdk:scene"
    "uav-challenge.tencentcloudcr.com/uav_challenge_2024/sdk:user"
)


# 循环拉取每个镜像
for image in "${images[@]}"; do
    echo "Pulling $image..."
    docker pull "$image"
    if [ $? -ne 0 ]; then
        echo "Failed to pull $image"
```

```
        exit 1
    fi
done


echo "All images pulled successfully!"
```

将以上内容复制到文件中

运行脚本

```
chmod +x race_images.sh
./race_images.sh
```

```
mu@mu-System-Product-Name:~/workspace/uav_competition_2$ ./race_images.sh
Pulling registryonline-hulk.sankuai.com/custom_prod/com.sankuai.udm.udss/race:race_user_sdk...
race_user_sdk: Pulling from custom_prod/com.sankuai.udm.udss/race
560c024910be: Already exists
e541596f8b35: Already exists
904313c44f3c: Already exists
92a0710e8ce1: Already exists
3ab8426a021e: Already exists
9eb19495c817: Already exists
7df37cf25808: Already exists
1186322a3cea: Already exists
456a721c0061: Already exists
53e4b749804c: Already exists
8135187a8f5c: Already exists
0158d6afa5ef: Already exists
cb484d58aad1: Already exists
d39829599b14: Pull complete
46143c947101: Pull complete
Digest: sha256:ff56da2414a30c858a935dcbab4730f78882659378063f3eed3f307f4db76d98
Status: Downloaded newer image for registryonline-hulk.sankuai.com/custom_prod/com.sankuai.udm.udss/race:race_user_sdk
registryonline-hulk.sankuai.com/custom_prod/com.sankuai.udm.udss/race:race_user_sdk
Pulling registryonline-hulk.sankuai.com/custom_prod/com.sankuai.udm.udss/race:race_drone_sdk...
race_drone_sdk: Pulling from custom_prod/com.sankuai.udm.udss/race
560c024910be: Already exists
1b45f09fb1ef: Pull complete
519df1d68fa8: Pull complete
ea5cddd884b4: Pull complete
f6f98478de1e: Pull complete
e755eef3619b: Pull complete
f162631af00d: Pull complete
Digest: sha256:997e1b31aafa91c723a1a9588aaeecb0ce7754c4cc01406e4b882fd040afa2c6
Status: Downloaded newer image for registryonline-hulk.sankuai.com/custom_prod/com.sankuai.udm.udss/race:race_drone_sdk
registryonline-hulk.sankuai.com/custom_prod/com.sankuai.udm.udss/race:race_drone_sdk
Pulling registryonline-hulk.sankuai.com/custom_prod/com.sankuai.udm.udss/race:race_car_sdk...
race_car_sdk: Pulling from custom_prod/com.sankuai.udm.udss/race
41af1b5f0f51: Already exists
e384ee976924: Already exists
fc85c6d8f29e: Already exists
9964396126f3: Already exists
83c28351996c: Pull complete
fd5525071028: Pull complete
c210fd778b46: Pull complete
26f93e9b3297: Pull complete
14ae0962de80: Pull complete
38cd20397512: Pull complete
3efa4585a35f: Pull complete
eebaf90c732a: Pull complete
ea2912091051: Pull complete
1c1b8f256c51: Pull complete
82e277d9e9dc: Pull complete
924a1c38c9dd: Pull complete
4f4fb700ef54: Pull complete
88d0d3451c77: Pull complete
b0bf333db92c: Pull complete
d209f1c65c71: Pull complete
793aca09ae05: Pull complete
Digest: sha256:7dcfc1a873bd69c70c6bd3f025092d79dedb3704c2b79fb9c8b801fb30a4549d
Status: Downloaded newer image for registryonline-hulk.sankuai.com/custom_prod/com.sankuai.udm.udss/race:race_car_sdk
registryonline-hulk.sankuai.com/custom_prod/com.sankuai.udm.udss/race:race_car_sdk
Pulling registryonline-hulk.sankuai.com/custom_prod/com.sankuai.udm.udss/race:race_scene_sdk...
race_scene_sdk: Pulling from custom_prod/com.sankuai.udm.udss/race
08c01a0ec47e: Pull complete
c3392fe4e97a: Pull complete
ced61376d396: Pull complete
92b8f8253df7: Pull complete
212db4fefa93: Pull complete
2c1a0693f7f0: Pull complete
dad699b28826: Pull complete
d7c1a9b398eb: Pull complete
26ce28477fec: Pull complete
5479cbb6ea4d: Pull complete
d64cfd4157c4: Pull complete
def0694e35a2: Pull complete
4ac89910bc3d: Pull complete
776510ff9a7f: Pull complete
5e485c24ed7f: Pull complete
d914e11d6c56: Pull complete
022cd2b87d1b: Pull complete
70971687c897: Pull complete
c16ea4ebe165: Pull complete
9bcfa185017a: Pull complete
663f20193f72: Pull complete
3c2c02fb070f: Pull complete
Digest: sha256:66508e55f02ee5f048284dc8a3894083b0cef4e706b4f536a8c0ead31f523ad1
Status: Downloaded newer image for registryonline-hulk.sankuai.com/custom_prod/com.sankuai.udm.udss/race:race_scene_sdk
registryonline-hulk.sankuai.com/custom_prod/com.sankuai.udm.udss/race:race_scene_sdk
All images pulled successfully!
```

2.3.2. 创建局域网

创建自定义网络

```
docker network create --subnet=192.168.100.0/24 race_net
```

```
mu@mu-System-Product-Name:~/workspace/uav_competition_2$ docker network create --subnet=192.168.100.0/24 race_net
1dbda1807ce75f5ab7487fd3b34cc0251822737a1b4aca70d7d0151ced4aef3f
```

查看子网络详细信息

```
docker network inspect race_net
```

```
mu@mu-System-Product-Name:~/workspace/uav_competition_2$ docker network inspect race_net
[
    {
        "Name": "race_net",
        "Id": "1dbda1807ce75f5ab7487fd3b34cc0251822737a1b4aca70d7d0151ced4aef3f",
        "Created": "2024-08-15T15:54:47.916562322+08:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "192.168.100.0/24"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]
```

2.3.3. 启动 SDK 服务

新建启动脚本 start_race.sh

```
#!/bin/bash


# Function to check the last command status and exit if it failed

check_status() {

   if [ $? -ne 0 ]; then

      echo "Error: $1 failed"

      exit 1

   fi

}


# Run the first container

docker run -d --entrypoint /manager/run.sh --name race_scene_sdk_container
uav-challenge.tencentcloudcr.com/uav_challenge_2024/sdk:scene

check_status "docker run race_scene_sdk_container"


# Connect the first container to the network

docker network connect race_net race_scene_sdk_container --ip 192.168.100.5
```

```
check_status "docker network connect race_scene_sdk_container"

# Run the second container
docker run -d --name race_car_sdk_container
uav-challenge.tencentcloudcr.com/uav_challenge_2024/sdk:cars
check_status "docker run race_car_sdk_container"

# Connect the second container to the network
docker network connect race_net race_car_sdk_container --ip 192.168.100.2
check_status "docker network connect race_car_sdk_container"

# Run the third container
docker run -d --name race_drone_sdk_container
uav-challenge.tencentcloudcr.com/uav_challenge_2024/sdk:drones
check_status "docker run race_drone_sdk_container"

# Connect the third container to the network
docker network connect race_net race_drone_sdk_container --ip 192.168.100.3
check_status "docker network connect race_drone_sdk_container"

# 如果有权限问题，可以手动去创建
if [ ! -d "/home/race_log" ]; then
    echo "Directory /home/race_log does not exist. Creating it..."
    mkdir -p /home/race_log
fi

# Run the fourth container
#docker run -d --name race_user_sdk_container -v /etc/localtime:/etc/localtime:ro -v
/etc/timezone:/etc/timezone:ro
uav-challenge.tencentcloudcr.com/uav_challenge_2024/sdk:user
docker run -d -p 8888:8888    --name race_user_sdk_container \
    --network race_net --ip 192.168.100.4 \
    -e ROS_MASTER_URI=http://192.168.100.4:11311 \
    -e ROS_IP=192.168.100.4 \
    -v /home/race_log:/config \
```

```
    -v /etc/localtime:/etc/localtime:ro \
    -v /etc/timezone:/etc/timezone:ro \
    marcobright2023/mtuav-competition-2024:user
check_status "docker run race_user_sdk_container"
# Connect the fourth container to the network
#docker network connect race_net race_user_sdk_container --ip 192.168.100.4
#check_status "docker network connect race_user_sdk_container"


echo "All commands executed successfully"
```

将以上内容复制到文件

执行脚本

```
chmod +x start_race.sh
./start_race.sh


#可以使用 docker ps 查看当前启用的容器
```

```
mu@mu-System-Product-Name:~/workspace/uav_competition_2$ ./start_race.sh
e5d0beea3da0a22875ec11d11bae3598d57f93dcb2c239a10d46e6e9cc1c8dc4
228d10c563dc0b77d268d3a635246228384f0aabb88069ec38eecd507e6da8f5
b8a9708a39857694375d64e885d05a8e2219ddce6e10b3fe3c0208bae2fb5417
58cc64a809fc80b455c9f43df3acf50c1792b76b2564657d7627eec79e3ef27e
All commands executed successfully
```

2.3.4. 关闭 SDK 服务

新建脚本 stop_race.sh

```
#!/bin/bash


# Function to check the last command status and exit if it failed

check_status() {

    if [ $? -ne 0 ]; then

        echo "Error: $1 failed"
```

```
        exit 1
    fi
}

# Stop and remove the first container
docker stop race_scene_sdk_container
check_status "docker stop race_scene_sdk_container"

docker rm race_scene_sdk_container
check_status "docker rm race_scene_sdk_container"

# Stop and remove the second container
docker stop race_car_sdk_container
check_status "docker stop race_car_sdk_container"

docker rm race_car_sdk_container
check_status "docker rm race_car_sdk_container"

# Stop and remove the third container
docker stop race_drone_sdk_container
check_status "docker stop race_drone_sdk_container"

docker rm race_drone_sdk_container
check_status "docker rm race_drone_sdk_container"

# Stop and remove the fourth container
docker stop race_user_sdk_container
check_status "docker stop race_user_sdk_container"

docker rm race_user_sdk_container
check_status "docker rm race_user_sdk_container"

echo "All containers have been stopped and removed successfully."
```

将以上内容复制到文件

执行脚本

```
chmod +x start_race.sh

chmod +x stop_race.sh

./stop_race.sh


#可以自行设置是否删除容器，正常来讲 race_user_sdk_container 镜像可以重复使用，其他三个镜像每次使用都需要新建进行初始化
```

```
mu@mu-System-Product-Name:~/workspace/uav_competition_2$ ./stop_race.sh
race_scene_sdk_container
race_scene_sdk_container
race_car_sdk_container
race_car_sdk_container
race_drone_sdk_container
race_drone_sdk_container
race_user_sdk_container
race_user_sdk_container
All containers have been stopped and removed.
```

2.3.5. 验证 SDK 服务是否成功

查看并进入容器

```
docker ps
```

```
mu@mu-System-Product-Name:~/workspace/uav_competition_2$ docker ps
CONTAINER ID   IMAGE                                                                    COMMAND                  CREATE
D         STATUS         PORTS       NAMES
58cc64a809fc   registryonline-hulk.sankuai.com/custom_prod/com.sankuai.udm.udss/race:race_user_sdk    "/ros_entrypoint.sh"     9 minu
tes ago   Up 9 minutes               race_user_sdk_container
b8a9708a3985   registryonline-hulk.sankuai.com/custom_prod/com.sankuai.udm.udss/race:race_drone_sdk   "/drone/run.sh /bin/…"   9 minu
tes ago   Up 9 minutes               race_drone_sdk_container
228d10c563dc   registryonline-hulk.sankuai.com/custom_prod/com.sankuai.udm.udss/race:race_car_sdk     "/ros_entrypoint.sh"     9 minu
tes ago   Up 9 minutes               race_car_sdk_container
e5d0beea3da0   registryonline-hulk.sankuai.com/custom_prod/com.sankuai.udm.udss/race:race_scene_sdk   "/manager/run.sh"        9 minu
tes ago   Up 9 minutes               race_scene_sdk_container
```

```
docker exec -it race_user_sdk_container bash
```

```
mu@mu-System-Product-Name:~/workspace/uav_competition_2$ docker exec -it race_user_sdk_container bash
root@58cc64a809fc:/# ls
bin   config  etc   lib    lib64   media  opt   root             run   srv  tmp  var
boot  dev     home  lib32  libx32  mnt    proc  ros_entrypoint.sh  sbin  sys  usr
```

查看 SDK 服务是否启动

```
ps aux

# 如果进程中包含
/home/sim_competition_sdk/devel/lib/user_pkg/competition_msg_handler 与

#/home/sim_competition_sdk/devel/lib/user_pkg/map_client_node __name:，表示 SDK 服务启动成功

# 也可以使用 ros 指令查看

rosnode list
```

```
root@91e7a37db000:/# rosnode list
/competition_msg_handler_node
/map_client_node
/race_monitor_node
/rosout
root@91e7a37db000:/#
```

2.4. 联网版说明

联网版是选手最终提交的版本，选手需要提交包含其自身代码和程序的镜像。

- 选手必须将比赛代码拷贝到容器 **race_user_sdk_container 的/home/**目录下进行调试，该容器由脚本 start_race.sh 启动的。调试完成后将比赛程序以及所需的库文件，配置文件等，保留在/home 目录下（也可以在 home 下新建目录）
- 在 home 目录下我们提供了名为 **run.sh** 的文件，选手必须将比赛程序的启动方式写入到 run.sh 文件中，这样才能保证下次启动时，会自动执行比赛程序。

```
root@91e7a37db000:/home# ll
total 16
drwxr-xr-x 1 root root 4096 Aug 22 14:52 ./
drwxr-xr-x 1 root root 4096 Aug 22 14:47 ../
-rwxr-xr-x 1 root root  116 Aug 22 14:52 run.sh*
drwxr-xr-x 6 root root 4096 Aug 21 15:31 sdk_for_use/
root@91e7a37db000:/home#
```

run.sh 内容如下

```
#!/bin/bash
echo "正在执行比赛程序"

# 在这里添加您的比赛程序代码

echo "程序执行完毕"
~
```

- 将 user 打包为镜像（详情见 **5.提交方法**），然后提交

3. 协议与接口

3.1. 数据结构

3.1.1. 地图数据

用途	类型 ROS service	QueryVoxel.srv 定义	说明
地图查询	QueryVoxel.srv	float32 x float32 y float32 z --- user_pkg/Voxel voxel bool success Voxel.msg: float32 distance float32 cur_height uint16 height uint8 semantic	**主要功能：** ● 客户端发送一个三维坐标（x，y，z）。 ● 服务器根据这个坐标查询相应的体素数据，并返回体素数据和操作成功状态。 注：具体的参数 srv 类型定义，可在容器 **race_user_sdk_container 的 /home/sdk_for_user/srv** 自行查看

3.1.2. 用户控制指令

发布指令	/cmd_exec	user_pkg::UserCmdRequest	UserCmdRequest.msg
			uint32 peer_id
			string task_guid
			
			# UserCmdType
			uint8 type
			uint8 USER_CMD_NONE = 0
			uint8 USER_CMD_DRONE_EXEC_ROUTE = 1
			uint8 USER_CMD_CAR_EXEC_ROUTE = 2
			uint8 USER_CMD_MOVE_DRONE_ON_CAR = 3
			uint8 USER_CMD_MOVE_DRONE_ON_BIRTHPLACE = 4
			uint8 USER_CMD_MOVE_CARGO_IN_DRONE = 5
			uint8 USER_CMD_DRONE_RELEASE_CARGO = 6
			uint8
				peer_id：选手端 id
				task_guid：该次任务的全局唯一标识符
				二者都可从 json 文件（需进入容器 **race_user_sdk_container，配置文件路径为 /config/config.json**）中读取：
				msg.peer_id = config.getPeerId();
				msg.task_guid = config.getGuid();
				命令类型 type：
				1：飞机航线下发
				2：小车轨迹下发
				3：将飞机移动到小车上，飞机必须在出生地才能执行
				4：将飞机移动到出生地，飞机必须在小车上才能执行

			USER_CMD_DRONE_BATTERY_REPLACEMENT = 7 DroneWayPointInfo drone_way_point_info CarRouteInfo car_route_info BindDrone binding_drone UnBindInfo unbind_info BindCargo binding_cargo DroneMsg drone_msg	5: 将货物装到飞机里，飞机必须在上货点且在小车上 6: 飞机释放货物，飞机必须在卸货点 且飞机上装载有货物 7: 飞机换电指令 每一种类型对应下方的一个参数(oneof args): 前五个参数易于匹配，最后一个 drone_msg 参数绑定于 释放货物 6 和 飞机换电 7 指令。 注：具体的参数 msg 类型定义，可在容器 **race_user_sdk_container 的 /home/sdk_for_user/msg** 自行查看
返回指令结果	/cmd_resp	user_pkg::UserCmdResponse	user_pkg::UserCmdResponse # CmdResponseType uint8 type uint8 CMD_RESP_SUCCESS = 0 uint8 ERROR_TYPE_DRONE_NOT_READY = 1 # 飞机不是 ready 状态 uint8 ERROR_TYPE_DRONE_PLAN_ROUTE_VERIFICATION_FAILED = 2 # 飞机航线校验失败 uint8 ERROR_TYPE_CAR_CANNOT_EXECUTE_RELEASE_OPERATION = 3 # 飞机无法执行释放动作 uint8 ERROR_TYPE_DRONE_NOT_ON_AVIATION_OPERATION_POINT = 4 # 飞机不在航空作业点 uint8 ERROR_TYPE_DRONE_NOT_ON_BI	详细信息 uint8 type 0：成功 1 ： 飞机不是 ready 状态 2 ： 飞机航线校验失败 3 ： 飞机无法执行释放动作 4 ： 飞机不在航空作业点 5 ： 飞机不在出生地 6 ： 飞机不在上货点 7 ： 飞机不在卸货站 8 ： 飞机绑定了其他货物 9 ： 飞机绑定了其他小车 10 ： 飞机未绑定货物 11 ： 飞机无法执行换电操作 12 ：小车不是 ready 状态 13 ： 小车规划路径校验

			RTHPLACE = 5 # 飞机不在出生地 uint8 ERROR_TYPE_DRONE_NOT_ON_LOADING_CARGO_STATION = 6 # 飞机不在上货点 uint8 ERROR_TYPE_DRONE_NOT_ON_UNLOADING_CARGO_STATION = 7 # 飞机不在卸货站 uint8 ERROR_TYPE_DRONE_BINDING_OTHER_CARGO = 8 # 飞机绑定了其他货物 uint8 ERROR_TYPE_DRONE_BINDING_OTHER_CAR = 9 # 飞机绑定了其他小车 uint8 ERROR_TYPE_DRONE_NO_BINDING_CARGO = 10 # 飞机未绑定货物 uint8 ERROR_TYPE_DRONE_CANNOT_EXECUTE_BATTERY_REPLACEMENT = 11 # 飞机无法执行换电操作 uint8 ERROR_TYPE_CAR_NOT_READY = 12 # 小车不是 ready 状态 uint8 ERROR_TYPE_CAR_PLAN_ROUTE_VERIFICATION_FAILED = 13 # 小车规划路径校验失败 uint8 ERROR_TYPE_CAR_HAVE_OTHER_DRONE = 14 # 小车上有其他飞机 uint8 ERROR_TYPE_CAR_NOT_ON_LOADING_CARGO_STATION = 15 # 小车不在上货点 uint8 ERROR_TYPE_NO_BINDING_CAR_AND_DRONE = 16 # 飞	失败 14 ： 小车上有其他飞机 15 ： 小车不在上货点 16 ： 飞机和小车不是绑定关系 17 ： 货物不是 NOT_STARTED 状态 18 ： 货物不是配送状态 19 ： 货物被送至错误的目的地 20 ： 上货点没有飞机 21 ： 上货点没有小车 22 ： 飞机在飞行中释放了货物 string description

			机和小车不是绑定关系 uint8 ERROR_TYPE_CARGO_NO_NOT_STARTED = 17 # 货物不是 NOT_STARTED 状态 uint8 ERROR_TYPE_CARGO_NO_DELIVERY = 18 # 货物不是配送状态 uint8 ERROR_TYPE_CARGO_WRONG_DESTINATION = 19 # 货物被送至错误的目的地 uint8 ERROR_TYPE_LOADING_CARGO_STATION_NO_DRONE = 20 # 上货点没有飞机 uint8 ERROR_TYPE_LOADING_CARGO_STATION_NO_CAR = 21 # 上货点没有小车 uint8 ERROR_TYPE_DRONE_IN_FLYING_RELEASE_CARGO = 22 # 飞机在飞行中释放了货物 string description	

3.1.3. 全局信息数据

用途	对应的 ROS topic	类型 ROS message	PanoramicInfo.msg 定义	说明
订阅全局信息	/panoramic_info	user_pkg::PanoramicInfo	DronePhysicalStatus[] drones CarPhysicalStatus[] cars BillStatus[] bills EventMsg[] events float64 score	DronePhysicalStatus：数组包含所有飞机信息，单个飞机信息如下： string sn uint64 timestamp uint32 peer_id string task_guid DynamicPosition pos #飞机位置 # DroneWorkState

```
uint8 drone_work_state      #飞机状态
uint8 UNKOWN = 0
uint8 READY = 1
uint8 TAKEOFF = 2
uint8 FLYING = 3
uint8 LANDING = 4
uint8 CHARGING_BATTERY = 5
uint8 LOADING_CARGO = 6
uint8 ERROR = 10

float64 remaining_capacity   #剩余电量
string description
int32 bind_cargo_id     #绑定的货物 id
```

CarPhysicalStatus：数组包含所有小车信息，单个小车信息如下：

```
string sn
uint64 timestamp
uint32 peer_id
string task_guid
DynamicPosition pos      #小车位置

# CarWorkState
uint8 car_work_state     #小车状态
uint8 CAR_UNKOWN = 0
uint8 CAR_READY = 1
uint8 CAR_RUNNING = 2
uint8 CAR_ERROR = 10

string description
string drone_sn        #绑定在小车上的飞机 sn 若无 则为 none
```

BillStatus：数组包含所有订单信息，单个订单信息如下：

```
uint64 index     #订单编号
Position target_pos   #目标位置

# WaybillStatus
uint8 status
uint8 BILL_UNKOWN = 0
```

				uint8 NOT_STARTED = 1 uint8 DELIVERY = 2 uint8 OVER = 3 uint8 BILL_ERROR = 5 DynamicPosition dynamic_position EventMsg：数组包含当前发生的事件信息，单个事件信息如下： float64 score #当前事件的得分 如订单送达+1 飞机碰撞-50 string event_type #事件类型 string description #具体描述 int64 timestamp score：从任务开始到当前时刻的总得分

3.2. API 说明

比赛所使用的 SDK 接口采用了通过 ROS Topic 与 Service 两种方式来实现，因此选手需要自行实现 ROS Node 来完成交互的过程。

3.2.1. Topic 列表

Topic 名称	消息类型	发布者	订阅者	简要描述
/cmd_exec	user_pkg::UserCmdRequest	选手自定义	competition_msg_handler_node	下发无人机，小车运动轨迹以及其他指令【详情见 3.1.2】
/cmd_resp	user_pkg::UserCmdResponse	competition_msg_handler_node	选手自定义	返回选手指令的执行结果【详情见 3.1.2】

/panoramic_info	user_pkg::PanoramicInfo	competition_msg_handler_node	选手自定义	获取比赛所需全局信息【详情见 3.1.3】

3.2.2. Service 列表

Service 名称	服务类型	提供者	请求者	简要描述
query_voxel	QueryVoxel.srv	map_client_node	选手自定义	查询体素等数据【详情见 3.1.1】

3.2.3. 创建 ROS NodeHandler 示例

包括发布者，订阅者，service client。

```
ros::NodeHandle nh;

ros::Publisher cmd_pub = nh.advertise<user_pkg::UserCmdRequest>("/cmd_exec", 10000);    //建立一个发布者，用于发布/cmd_exec

ros::Subscriber info_sub = nh.subscribe("/panoramic_info", 10, panoramicInfoCallback);    //建立一个订阅者，用于订阅/panoramic_info

ros::Subscriber cmd_resp_sub = nh.subscribe("/cmd_resp", 10, cmdResponseCallback);    // 建立一个订阅者，用于订阅/cmd_resp

ros::ServiceClient map_client = nh.serviceClient<user_pkg::QueryVoxel>("query_voxel");  //建立一个 service client，发送 query_voxel 请求
```

3.2.4. 控制指令示例

发布命令需填充 user_pkg::UserCmdRequest msg 字段

序号	指令	描述	举例说明

1	*USER_CMD_CAR_EXEC_ROUTE*	将小车移动到上货点	user_pkg::UserCmdRequest msg
			msg.peer_id = config.getPeerId();
			msg.task_guid = config.getGuid();
			msg.type = user_pkg::UserCmdRequest::USER_CMD_CAR_EXEC_ROUTE;
			msg.car_route_info.carSn = car_sn;
			
			user_pkg::Position load_cargo_pos;

			load_cargo_pos.x = config.getLoadingCargoPoint().x;
			load_cargo_pos.y = config.getLoadingCargoPoint().y;
			load_cargo_pos.z = config.getLoadingCargoPoint().z;
			
			msg.car_route_info.way_point.push_back(car_start_pos);
			msg.car_route_info.way_point.push_back(load_cargo_pos);
			msg.car_route_info.yaw = 0.0;
			LOG(INFO) << "Publishing UserCmdRequest message for car to load cargo point";
			LOG(INFO) << " car_sn: " << car_sn;
			LOG(INFO) << " car route info:" << msg.car_route_info;
			cmd_pub.publish(msg);

2	*USER_CMD_MOVE_DRONE_ON_CAR*	将飞机移动到小车上
		小车必须在上货点
			user_pkg::UserCmdRequest msg
			msg.peer_id = config.getPeerId();
			msg.task_guid = config.getGuid();
			msg.type = user_pkg::UserCmdRequest::USER_CMD_MOVE_DRONE_ON_CAR;
			msg.binding_drone.car_sn = car_sn;
			msg.binding_drone.drone_sn = drone_sn;
			LOG(INFO) << "Publishing UserCmdRequest message for bind drone to car";
			LOG(INFO) << " car_sn: " << car_sn;
			LOG(INFO) << " drone_sn: " << drone_sn;
			cmd_pub.publish(msg);

3	*USER_CMD_MOVE_CARGO_IN_DRONE*	将货物绑定到飞机上
		飞机和小车必须在上货点，并且飞机和小车已绑定
			user_pkg::UserCmdRequest msg
			msg.peer_id = config.getPeerId();
			msg.task_guid = config.getGuid();
			msg.type = user_pkg::UserCmdRequest::USER_CMD_MOVE_CARGO_IN_DRONE;
			msg.binding_cargo.cargo_id = config.getWaybillParamList()[0].cargoParam.index;
			msg.binding_cargo.drone_sn = drone_sn;
			LOG(INFO) << "Publishing UserCmdRequest message for bind cargo to drone";
			LOG(INFO) << " cargo_id: " << msg.binding_cargo.cargo_id;
			LOG(INFO) << " drone_sn: " << drone_sn;
			cmd_pub.publish(msg);

4	*USER_CMD_CAR_EXEC_ROUTE*	将小车移动到航空作业区	user_pkg::UserCmdRequest msg
			msg.peer_id = config.getPeerId();
			msg.task_guid = config.getGuid();
			msg.type = user_pkg::UserCmdRequest::USER_CMD_CA

			R_EXEC_ROUTE;
			msg.car_route_info.carSn = car_sn;
			
			user_pkg::Position load_cargo_pos;
			load_cargo_pos.x = config.getLoadingCargoPoint().x;
			load_cargo_pos.y = config.getLoadingCargoPoint().y;
			load_cargo_pos.z = config.getLoadingCargoPoint().z;
			
			msg.car_route_info.way_point.push_back(load_cargo_pos);
			msg.car_route_info.way_point.push_back(car_start_pos);
			msg.car_route_info.yaw = 0.0;
			LOG(INFO) << "Publishing UserCmdRequest message for car to return to takeoff point";
			LOG(INFO) << " car_sn: " << car_sn;
			LOG(INFO) << " car route info:" << msg.car_route_info;
			cmd_pub.publish(msg);

5	*USER_CMD_DRONE_EXEC_ROUTE*	给飞机下发航线送货	user_pkg::UserCmdRequest msg
			msg.peer_id = config.getPeerId();
			msg.task_guid = config.getGuid();
			msg.type = user_pkg::UserCmdRequest::USER_CMD_DRONE_EXEC_ROUTE;
			msg.drone_way_point_info.droneSn = drone_sn;
			
			user_pkg::DroneWayPoint takeoff_point;
			takeoff_point.type = user_pkg::DroneWayPoint::POINT_TAKEOFF;
			takeoff_point.timeoutsec = 1000;
			msg.drone_way_point_info.way_point.push_back(takeoff_point);
			
			user_pkg::DroneWayPoint takeoff_air_point;
			takeoff_air_point.type = user_pkg::DroneWayPoint::POINT_FLYING;
			takeoff_air_point.pos.x = car_start_pos.x;
			takeoff_air_point.pos.y = car_start_pos.y;
			takeoff_air_point.pos.z = -145;
			takeoff_air_point.v = 10.0;
			takeoff_air_point.timeoutsec = 1000;
			msg.drone_way_point_info.way_point.push_back(takeoff_air_point);
			
			user_pkg::DroneWayPoint last_flying_point;

```
last_flying_point.type = user_pkg::DroneWayPoint::POINT_FLYING;
last_flying_point.pos.x = config.getWaybillParamList()[0].targetPosition.x;
last_flying_point.pos.y = config.getWaybillParamList()[0].targetPosition.y;
last_flying_point.pos.z = config.getWaybillParamList()[0].targetPosition.z - 5;
last_flying_point.v = 10.0;
last_flying_point.timeoutsec = 1000;

user_pkg::DroneWayPoint flying_point1;
flying_point1.type = user_pkg::DroneWayPoint::POINT_FLYING;
flying_point1.pos.x = takeoff_air_point.pos.x + (last_flying_point.pos.x - takeoff_air_point.pos.x) / 3;
flying_point1.pos.y = takeoff_air_point.pos.y + (last_flying_point.pos.y - takeoff_air_point.pos.y) / 3;
flying_point1.pos.z = -145;
flying_point1.v = 10.0;
flying_point1.timeoutsec = 1000;
msg.drone_way_point_info.way_point.push_back(flying_point1);

user_pkg::DroneWayPoint flying_point2;
flying_point2.type = user_pkg::DroneWayPoint::POINT_FLYING;
flying_point2.pos.x = flying_point1.pos.x + (last_flying_point.pos.x - takeoff_air_point.pos.x) / 3;
flying_point2.pos.y = flying_point1.pos.y + (last_flying_point.pos.y - takeoff_air_point.pos.y) / 3;
flying_point2.pos.z = -145;
flying_point2.v = 10.0;
flying_point2.timeoutsec = 1000;
msg.drone_way_point_info.way_point.push_back(flying_point2);

user_pkg::DroneWayPoint flying_point3;
flying_point3.type = user_pkg::DroneWayPoint::POINT_FLYING;
flying_point3.pos.x = flying_point2.pos.x + (last_flying_point.pos.x - flying_point2.pos.x) * 1;
flying_point3.pos.y = flying_point2.pos.y + (last_flying_point.pos.y - flying_point2.pos.y) * 1;
```

			flying_point3.pos.z = -145;
			flying_point3.v = 10.0;
			flying_point3.timeoutsec = 1000;
			msg.drone_way_point_info.way_point.push_back(flying_point3);
			
			msg.drone_way_point_info.way_point.push_back(last_flying_point);
			
			user_pkg::DroneWayPoint landing_point;
			landing_point.type = user_pkg::DroneWayPoint::POINT_LANDING;
			landing_point.timeoutsec = 1000;
			msg.drone_way_point_info.way_point.push_back(landing_point);
			LOG(INFO) << "Publishing UserCmdRequest message for drone to takeoff and flying and land";
			LOG(INFO) << " drone_sn: " << msg.drone_way_point_info.droneSn;
			LOG(INFO) << " drone way point info:" << msg.drone_way_point_info;
			cmd_pub.publish(msg);

6	*USER_CMD_DRONE_RELEASE_CARGO*	飞机卸货
		飞机已经降落在目的地
			user_pkg::UserCmdRequest msg
			msg.peer_id = config.getPeerId();
			msg.task_guid = config.getGuid();
			msg.type = user_pkg::UserCmdRequest::USER_CMD_DRONE_RELEASE_CARGO;
			msg.drone_msg.drone_sn = drone_sn;
			LOG(INFO) << "Publishing UserCmdRequest message for drone to release cargo";
			LOG(INFO) << " drone_sn: " << msg.drone_msg.drone_sn;
			cmd_pub.publish(msg);

7	*USER_CMD_DRONE_EXEC_ROUTE*	给飞机下发航线返航	user_pkg::UserCmdRequest msg
			msg.peer_id = config.getPeerId();
			msg.task_guid = config.getGuid();
			msg.type = user_pkg::UserCmdRequest::USER_CMD_DRONE_EXEC_ROUTE;
			msg.drone_way_point_info.droneSn = drone_sn;
			
			user_pkg::DroneWayPoint takeoff_point;
			takeoff_point.type = user_pkg::DroneWayPoint::POINT_TAKEOFF;
			takeoff_point.timeoutsec = 1000;

```
msg.drone_way_point_info.way_point.push_back(takeoff
_point);

user_pkg::DroneWayPoint takeoff_air_point;
takeoff_air_point.type = user_pkg::DroneWayPoint::POI
NT_FLYING;
takeoff_air_point.pos.x = config.getWaybillParamList()
[0].targetPosition.x;
takeoff_air_point.pos.y = config.getWaybillParamList()
[0].targetPosition.y;
takeoff_air_point.pos.z = -145;
takeoff_air_point.v = 10.0;
takeoff_air_point.timeoutsec = 1000;
msg.drone_way_point_info.way_point.push_back(takeoff
_air_point);

user_pkg::DroneWayPoint last_flying_point;
last_flying_point.type = user_pkg::DroneWayPoint::POI
NT_FLYING;
last_flying_point.pos.x = car_start_pos.x;
last_flying_point.pos.y = car_start_pos.y;
last_flying_point.pos.z = -20;
last_flying_point.v = 10.0;
last_flying_point.timeoutsec = 1000;

user_pkg::DroneWayPoint flying_point1;
flying_point1.type = user_pkg::DroneWayPoint::POINT
_FLYING;
flying_point1.pos.x = takeoff_air_point.pos.x + (last_flyi
ng_point.pos.x - takeoff_air_point.pos.x) / 3;
flying_point1.pos.y = takeoff_air_point.pos.y + (last_flyi
ng_point.pos.y - takeoff_air_point.pos.y) / 3;
flying_point1.pos.z = -145;
flying_point1.v = 10.0;
flying_point1.timeoutsec = 1000;
msg.drone_way_point_info.way_point.push_back(flying_
point1);

user_pkg::DroneWayPoint flying_point2;
flying_point2.type = user_pkg::DroneWayPoint::POINT
_FLYING;
flying_point2.pos.x = flying_point1.pos.x + (last_flying_
point.pos.x - takeoff_air_point.pos.x) / 3;
flying_point2.pos.y = flying_point1.pos.y + (last_flying_
```

			point.pos.y - takeoff_air_point.pos.y) / 3;
			flying_point2.pos.z = -145;
			flying_point2.v = 10.0;
			flying_point2.timeoutsec = 1000;
			msg.drone_way_point_info.way_point.push_back(flying_point2);
			
			user_pkg::DroneWayPoint flying_point3;
			flying_point3.type = user_pkg::DroneWayPoint::POINT_FLYING;
			flying_point3.pos.x = flying_point2.pos.x + (last_flying_point.pos.x - flying_point2.pos.x) * 1;
			flying_point3.pos.y = flying_point2.pos.y + (last_flying_point.pos.y - flying_point2.pos.y) * 1;
			flying_point3.pos.z = -145;
			flying_point3.v = 10.0;
			flying_point3.timeoutsec = 1000;
			msg.drone_way_point_info.way_point.push_back(flying_point3);
			
			msg.drone_way_point_info.way_point.push_back(last_flying_point);
			
			user_pkg::DroneWayPoint landing_point;
			landing_point.type = user_pkg::DroneWayPoint::POINT_LANDING;
			landing_point.timeoutsec = 1000;
			msg.drone_way_point_info.way_point.push_back(landing_point);
			LOG(INFO) << "Publishing UserCmdRequest message for drone to return";
			LOG(INFO) << " drone_sn: " << msg.drone_way_point_info.droneSn;
			LOG(INFO) << " drone way point info:" << msg.drone_way_point_info;
			cmd_pub.publish(msg);

8	*USER_CMD_CAR_EXEC_ROUTE*	给小车下发轨迹移动到上货点	user_pkg::UserCmdRequest msg
			msg.peer_id = config.getPeerId();
			msg.task_guid = config.getGuid();
			msg.type = user_pkg::UserCmdRequest::USER_CMD_CAR_EXEC_ROUTE;
			msg.car_route_info.carSn = car_sn;
			
			user_pkg::Position load_cargo_pos;

			load_cargo_pos.x = config.getLoadingCargoPoint().x;
			load_cargo_pos.y = config.getLoadingCargoPoint().y;
			load_cargo_pos.z = config.getLoadingCargoPoint().z;
			
			msg.car_route_info.way_point.push_back(car_start_pos);
			msg.car_route_info.way_point.push_back(load_cargo_pos);
			msg.car_route_info.yaw = 0.0;
			LOG(INFO) << "Publishing UserCmdRequest message for car to load cargo point";
			LOG(INFO) << " car_sn: " << msg.car_route_info.carSn;
			LOG(INFO) << " car route info:" << msg.car_route_info;
			cmd_pub.publish(msg);

9	*USER_CMD_DRONE_BATTERY_REPLACEMENT*	给飞机换电	user_pkg::UserCmdRequest msg
			msg.peer_id = config.getPeerId();
			msg.task_guid = config.getGuid();
			msg.type = user_pkg::UserCmdRequest::USER_CMD_DRONE_BATTERY_REPLACEMENT;
			msg.drone_msg.drone_sn = drone_sn;
			LOG(INFO) << "Publishing UserCmdRequest message for drone to battery replacement";
			LOG(INFO) << " drone_sn: " << msg.drone_msg.drone_sn;
			cmd_pub.publish(msg);

10	*USER_CMD_MOVE_DRONE_ON_BIRTHPLACE*	飞机与小车解绑，回到出生点	user_pkg::UserCmdRequest msg
			msg.peer_id = config.getPeerId();
			msg.task_guid = config.getGuid();
			msg.type = user_pkg::UserCmdRequest::USER_CMD_MOVE_DRONE_ON_BIRTHPLACE;
			msg.unbind_info.drone_sn = drone_sn;
			msg.unbind_info.car_sn = car_sn;
			LOG(INFO) << "Publishing UserCmdRequest message for drone to move on birthplace";
			LOG(INFO) << " drone_sn: " << msg.unbind_info.drone_sn;
			LOG(INFO) << " car_sn: " << msg.unbind_info.car_sn;
			cmd_pub.publish(msg);

订阅指令的执行结果 user_pkg::UserCmdResponse

序号	结果	描述	举例说明

1	uint8 type	获取指令的执行结果以及详细的描述信息	void cmdResponseCallback(const user_pkg::UserCmdResponse::ConstPtr& msg)
			{
			cmd_response_type = msg->type;
			LOG(INFO) << "Ros Received cmd response:" << cmd_response_type;
			LOG(INFO) << " description: " << msg->description;
			}

3.2.5. 获取全局信息示例

订阅全景消息 user_pkg::PanoramicInfo，全景消息会定时向用户发布

序号	结果	描述	举例说明

1	user_pkg::PanoramicInfo	获取小车、无人机、订单状态以及当前事件和总分信息	user_pkg::PanoramicInfo::ConstPtr current_panoramic_info;
			bool recevie_panoramic_info = false;
			void panoramicInfoCallback(const user_pkg::PanoramicInfo::ConstPtr& msg)
			{
			// LOG(INFO) << "Ros Received panoramic info";
			recevie_panoramic_info = true;
			current_panoramic_info = msg; //所有消息都更新在该变量
			car_physical_status = msg->cars; //获取所有小车信息
			drone_sn = msg->drones[0].sn; //获取数组中第一架飞机 sn
			car_work_state = msg->cars[findCarIndexBySN(car_physical_status, car_sn[0])].car_work_state;
			drone_current_pos = msg->drones[0].pos.position;
			drone_work_state = msg->drones[0].drone_work_state;
			}

3.2.6. 查询体素示例

通过服务 query_voxel 查询体素信息

序号	指令	描述	举例说明

1	query_voxel	查询体素	user_pkg::QueryVoxel srv;
			srv.request.x = 1.0;
			srv.request.y = 2.0;
			srv.request.z = -3.0;
			
			if (map_client.call(srv))
			{
			if (srv.response.success)
			{
			LOG(INFO) << "Query successful:";
			LOG(INFO) << "Distance: " << srv.response.voxel.distance;
			LOG(INFO) << "Current Height: " << srv.response.voxel.cur_height;
			LOG(INFO) << "Height: " << srv.response.voxel.height;
			LOG(INFO) << "Semantic: " << srv.response.voxel.semantic;
			}
			else
			{
			LOG(ERROR) << "Query failed.";
			}
			}
			else
			{
			LOG(ERROR) << "Failed to call service query_voxel";
			}

通过 SDK 查询体素信息

SDK 存放的位置：运行容器后，在 race_user_sdk_container 容器的 /home/sdk_for_user/map_client_sdk/下

```
root@d56eabd5e277:/home/sdk_for_user/map_client_sdk# ls
for_cpp  for_py
```

选手在开发时需自行将这些 SDK 文件拷贝到本地，可以使用 docker cp 指令（详细用法可自行谷歌）

- Python 方法

```
root@d56eabd5e277:/home/sdk_for_user/map_client_sdk/for_py# ll
total 138176
drwxr-xr-x 2 root root      4096 Aug 20 22:31 ./
drwxrwxr-x 4 1000 1000      4096 Aug 20 22:32 ../
-rwxr-xr-x 1 root root       378 Aug 20 22:31 list_functions.py*
-rwxr-xr-x 1 root root  11992720 Aug 20 22:31 pymtmap.cpython-38-x86_64-linux-gnu.so*
-rwxr-xr-x 1 root root      1078 Aug 20 22:31 test_map_sdk.py*
-rw-r--r-- 1 root root 129482525 Aug 20 22:31 voxel_map.bin
```

选手可参考 test_map_sdk.py 设计查询体素的方法

- C++方法

```
root@d56eabd5e277:/home/sdk_for_user/map_client_sdk/for_cpp# ll
total 133440
drwxr-xr-x 2 root root      4096 Aug 21 10:52 ./
drwxrwxr-x 4 1000 1000      4096 Aug 20 22:32 ../
-rw-r--r-- 1 root root      3289 Aug 20 22:31 libmap_client_sdk.go.h
-rw-r--r-- 1 root root   7136696 Aug 20 22:31 libmap_client_sdk.go.so
-rwxr-xr-x 1 root root      1448 Aug 21 10:51 test_map.cpp*
-rw-r--r-- 1 root root 129482525 Aug 21 10:16 voxel_map.bin
```

.h 文件中提供了数据结构和方法，选手可以参考 test_map.cpp 设计查询体素的方法

3.2.7. 可视化界面使用示例

- 可视化服务器已经跟随 docker 镜像的启动而自动运行
- docker 容器在本机运行，选手只需要在浏览器输入 http://localhost:8888 即可以打开可视化界面
- docker 容器远端主机运行，浏览器在本地主机上，**假设远端主机 ip 为 203.0.113.4**
  - 远端主机和本地主机在同一个网段，选手需要在浏览器输入 **http://203.0.113.4:8888**，即可打开可视化界面
  - 远端主机和本地主机不在同一个网段，比如远端主机为学校某服务器（**假设其地址为 10.232.149.195:8080**），需要使用端口转发（SSH 隧道）实现跨子网访问

    ```
    ssh -L 8888:192.168.100.4:8888 username@10.232.149.195 -p 8088
    ```

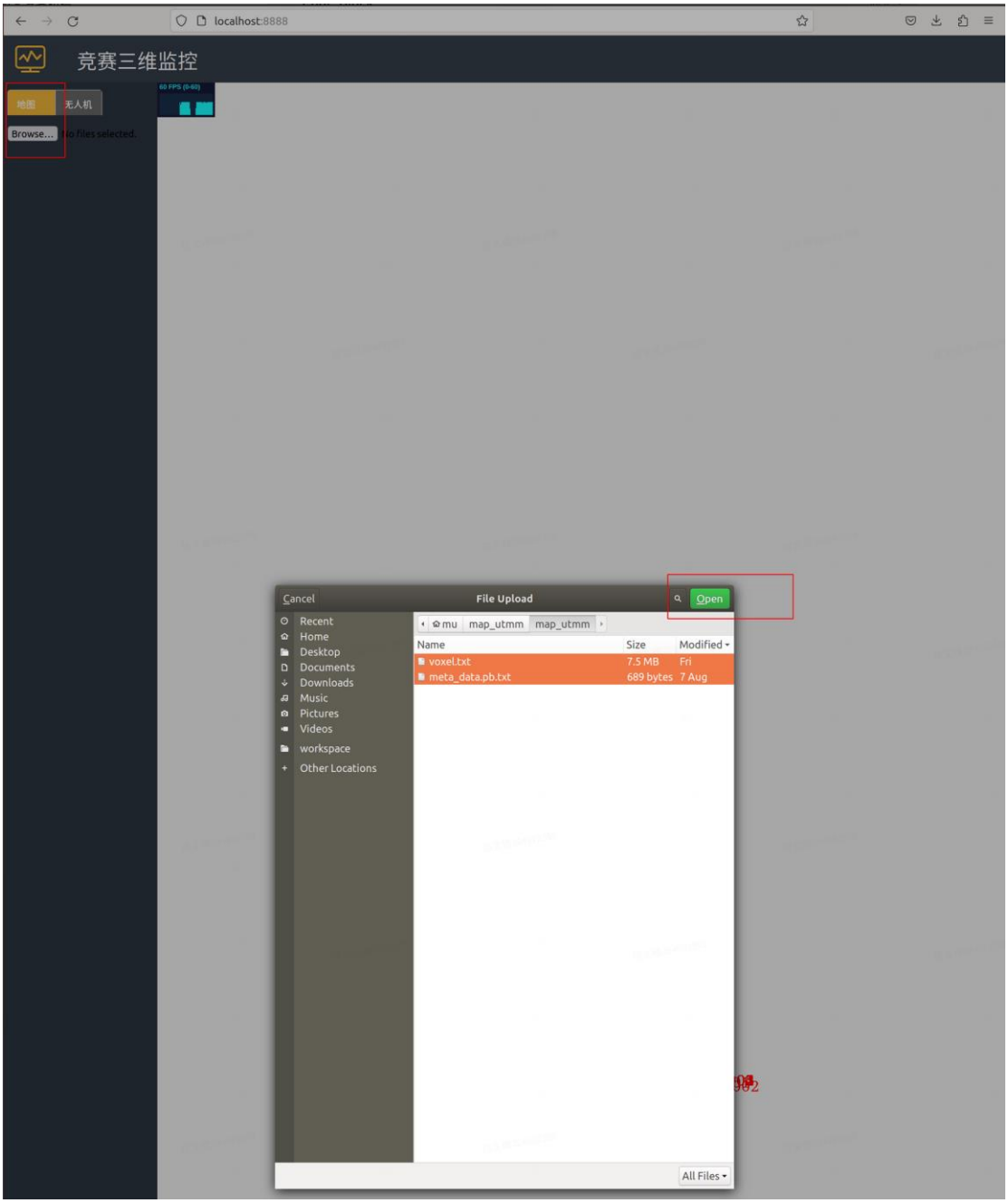
    Username 为 ssh 登陆的用户名，执行上述指令成功后，在本地浏览器输入 http://localhost:8888 即可打开可视化界面。此时的可视化界面中并没有导入地图。
- 下载并导入地图包

地图文件存放位置：运行容器后，在 race_user_sdk_container 容器的 /home/sdk_for_user/map_utmm/下

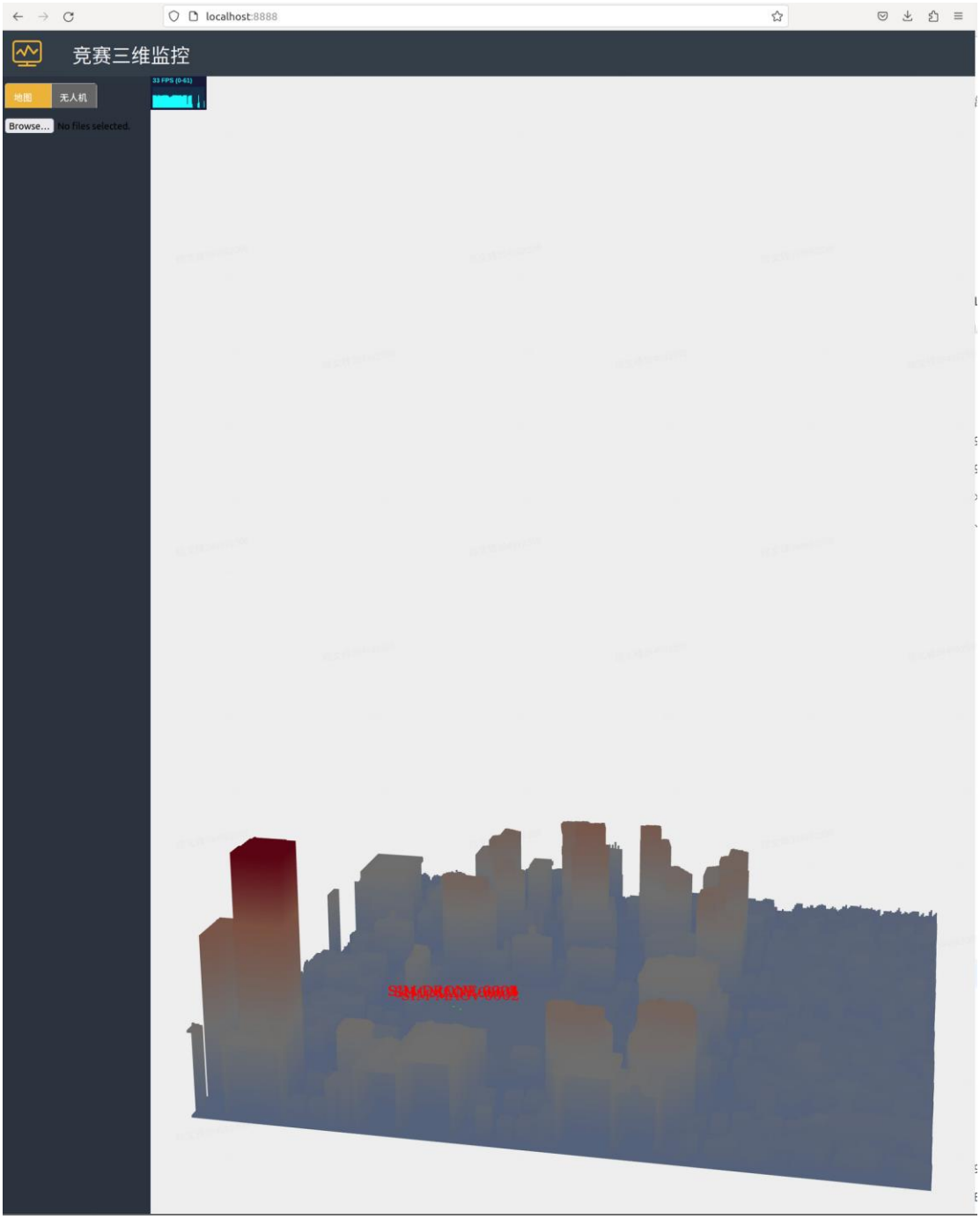
```
root@d56eabd5e277:/home/sdk_for_user/map_utmm# ls
meta_data.pb.txt  voxel.txt
```

选手需要将这两个文件下载到浏览器的同一主机上，然后按照下面的方式加载这两个文件

- 在浏览器打开可视化界面
- 点击左上角“地图按钮” --> “Browse”

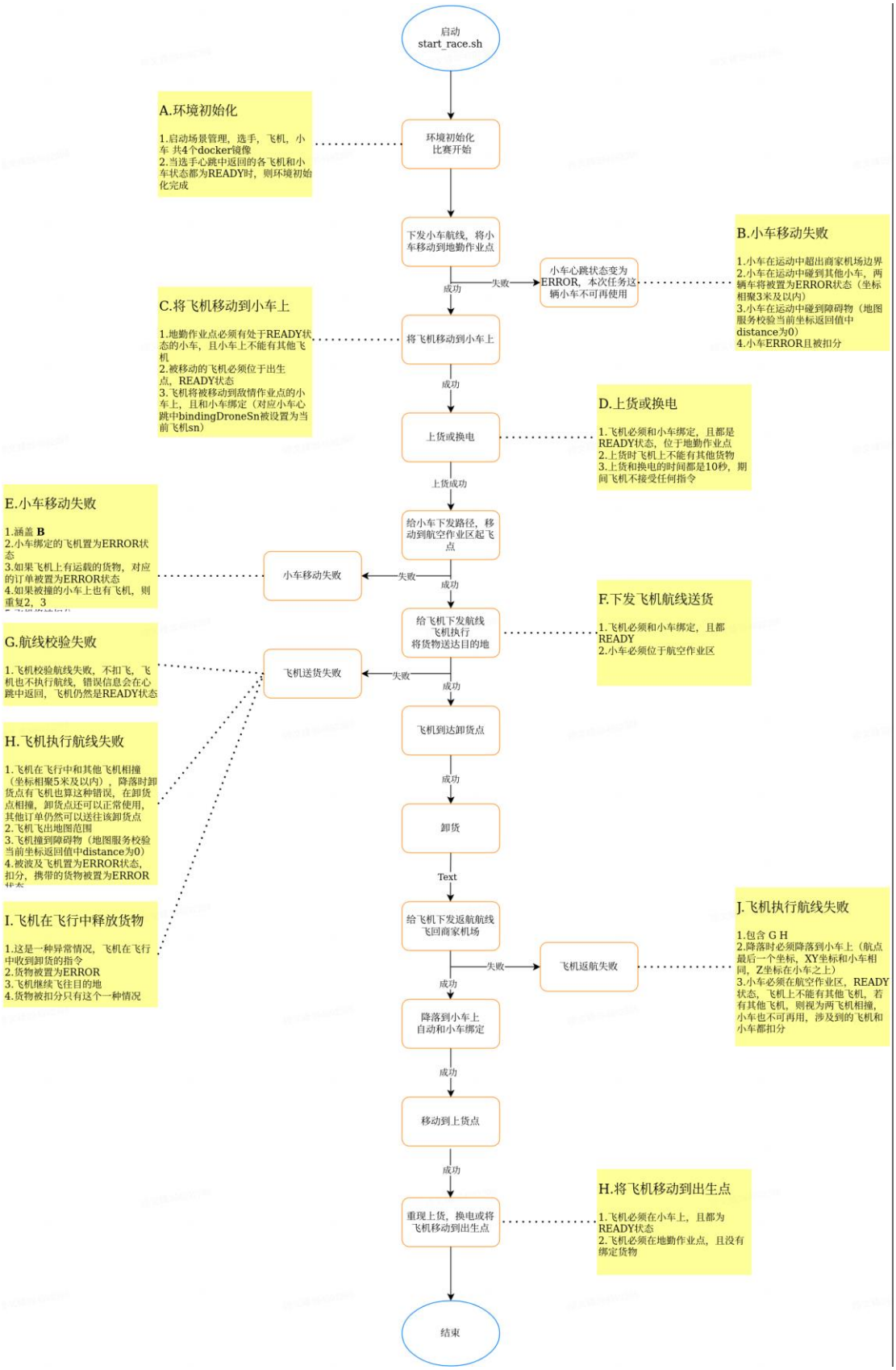


- 选择所有文件，点击“Open”，效果如下


localhost:8888
竞赛三维监控
地图
无人机
Browse...
No files selected.
33 FPS (0-61)

4. 业务流程与注意事项

4.1. 业务流程示意图


启动
start_race.sh
环境初始化
比赛开始
A.环境初始化
1.启动场景管理，选手，飞机，小车 共4个docker镜像
2.当选手心跳中返回的各飞机和小车状态都为READY时，则环境初始化完成
下发小车航线，将小车移动到地勤作业点
失败
小车心跳状态变为ERROR，本次任务这辆小车不可再使用
B.小车移动失败
1.小车在运动中超出商家机场边界
2.小车在运动中碰到其他小车，两辆车将被置为ERROR状态（坐标相聚3米及以内）
3.小车在运动中碰到障碍物（地图服务校验当前坐标返回值中distance为0）
4.小车ERROR且被扣分
成功
C.将飞机移动到小车上
1.地勤作业点必须有处于READY状态的小车，且小车上不能有其他飞机
2.被移动的飞机必须位于出生点，READY状态
3.飞机将被移动到敌情作业点的小车上，且和小车绑定（对应小车心跳中bindingDroneSn被设置为当前飞机sn）
将飞机移动到小车上
成功
上货或换电
D.上货或换电
1.飞机必须和小车绑定，且都是READY状态，位于地勤作业点
2.上货时飞机上不能有其他货物
3.上货和换电的时间都是10秒，期间飞机不接受任何指令
上货成功
E.小车移动失败
1.涵盖 B
2.小车绑定的飞机置为ERROR状态
3.如果飞机上有运载的货物，对应的订单被置为ERROR状态
4.如果被撞的小车上也有飞机，则重复2，3
给小车下发路径，移动到航空作业区起飞点
小车移动失败
失败
成功
给飞机下发航线
飞机执行
将货物送达目的地
F.下发飞机航线送货
1.飞机必须和小车绑定，且都READY
2.小车必须位于航空作业区
G.航线校验失败
1.飞机校验航线失败，不扣飞，飞机也不执行航线，错误信息会在心跳中返回，飞机仍然是READY状态
飞机送货失败
失败
成功
H.飞机执行航线失败
1.飞机在飞行中和其他飞机相撞（坐标相聚5米及以内），降落时卸货点有飞机也算这种错误，在卸货点相撞，卸货点还可以正常使用，其他订单仍然可以送往该卸货点
2.飞机飞出地图范围
3.飞机撞到障碍物（地图服务校验当前坐标返回值中distance为0）
4.被波及飞机置为ERROR状态，扣分，携带的货物被置为ERROR
飞机到达卸货点
成功
卸货
Text
I.飞机在飞行中释放货物
1.这是一种异常情况，飞机在飞行中收到卸货的指令
2.货物被置为ERROR
3.飞机继续飞往目的地
4.货物被扣分只有这个一种情况
给飞机下发返航航线
飞回商家机场
失败
飞机返航失败
J.飞机执行航线失败
1.包含 G H
2.降落时必须降落到小车上（航点最后一个坐标，XY坐标和小车相同，Z坐标在小车之上）
3.小车必须在航空作业区，READY状态，飞机上不能有其他飞机，若有其他飞机，则视为两飞机相撞，小车也不可再用，涉及到的飞机和小车都扣分
成功
降落到小车上
自动和小车绑定
成功
移动到上货点
成功
重现上货，换电或将飞机移动到出生点
H.将飞机移动到出生点
1.飞机必须在小车上，且都为READY状态
2.飞机必须在地勤作业点，且没有绑定货物
结束

4.2. 飞机操作注意事项

- 在飞机故障、换电、上货时，飞机无法执行其他指令
- 选手向飞机下发指令时，必须检查是不是完整的指令，必须有起飞和降落的操作指令，否则飞机无法正常飞行，例如

  起飞、飞行点 1，飞行点 2.....飞行点 n、降落
- 选手应该合理规划飞机航线，飞机如果撞机，将不再执行其他指令
- 选手规划航线，应该关注飞机电量，避免飞机没电，出现坠机的情况。PS：飞机飞行中每 10 秒损耗电量 1%；
- 飞机换电、上货时间为 10 秒；
- 飞机最大速度 10m/s;
- 选手应该合理设置飞机速度，避免在较短的航程内设置较大的速度，例如（10,10,-10）--->(9,9,-10) 速度应该设置在 1m/s 左右，避免将速度设置为 10m/s 等情况；

4.3. 如何查看日志

- 所有的日志都存放在容器 race_user_sdk_container 的/home/race_log 目录下，选手需要进入该容器查看
- 选手也可以在本地（宿主机）的/home/race_log/目录下查看日志

5. 提交方法

使用单机版本调试好代码后，根据文档打包成 Docker 镜像，然后使用工具提交给比赛系统。步骤如下：

5.1. 安装 Docker（参考 2.2)

5.2. 创建 Docker 镜像

保证 race_user_sdk_container 正在运行，然后执行以下指令

```
docker  commit  race_user_sdk_container race_user:xxxx
```

建议使用 race_user，xxxx 选手可自定义

5.3. 登录腾讯云 Docker 服务

```
docker  login uav-challenge.tencentcloudcr.com --username  'tcr$user'  --password  gXWWpxhO9igRnXzYYV58UexxS1Gw8VQY
```

5.4. 提交镜像到 Docker Hub

```
docker tag race_user:appkeyuav-challenge.tencentcloudcr.com/uav_challenge_2024/appkey:tag
docker push uav-challenge.tencentcloudcr.com/uav_challenge_2024/appkey:tag
```

- appkey 需要从邮件中获取，**请各个团队妥善保存 appkey（不可共享），**tag 选手可自定义。
- 比赛系统会根据提交顺序运行镜像，并且计分。

5.5. 向竞赛系统提交比赛结果

提交工具存放的位置：运行容器后，在 **race_user_sdk_container 容器的 /home/sdk_for_user/docker_submit_tool/**下

- 将 submit_client 与 submit.sh 两个文件拷贝到本地
- 在命令行执行./submit.sh，可以查看到提交方法

```
test@MT-WS:/data1/map_client_sdk$ ./submit.sh
submit tools
usage:
    [IMPORTANT] Before using this script, edit with it text editor, and put your app-key and secret-key in AK and SK
    [IMPORTANT] Before submitting, you must commit your IMAGE to docker hub [https://hub.docker.com/], and make it public
    To query result: ./submit.sh query
    To submit image: ./submit.sh submit ImageName:ImageTag
```

5.6. 提交任务状态如下

WAITING：等待执行，用户提交任务后，任务会进入等待队列，此时任务状态为 WAITING，在 WAITING 期间，用户再次提交任务，新的任务会覆盖掉之前的任务，在等待队列中的位置并不发生变化

RUNNING：运行状态，当有空余的资源可供任务运行时，系统会运行 等待队列 中等待最久的任务，此时这个任务的状态变为 RUNNING

OVER：运行完成，状态变为 OVER

ERROR：运行异常，当选手提交的镜像有问题，导致任务无法正常运行时，任务结束，并变为 ERROR 状态

6. 常见问题

6.1. SDK 只有 C++版本吗?

比赛使用 ROS 进行了 API 的封装，选手可以使用 ROS 支持的任意语言进行开发，比较推荐 C++和 Python。

6.2. 飞机可以降落/停靠的位置有哪些?

飞机的起始位置、地勤作业区，卸货区

6.3. 一次下发所有航线，还是得分段下发航线?

为了模拟真实运营，每次只能下发一架飞机的一条航线。多架飞机的航线下发需要调用多次下发函数。一架飞机的后续航线，需要根据飞机状态确认下发时机。

6.4. 如何知道在哪里取餐，以及送往何处?

开发者需要从配置文件读取取餐位置；送货点则是从订单信息中获取

6.5. 下发航线后，判断航线是否有效的标准是什么?

指令下发后，用户需要根据获取到的 PanoramicInfo 以及 UserCmdResponse 综合判断指令的执行结果。

6.6. 如何换电?

只有当飞机运动到装货点时，通过发布换电指令才能触发换电操作。

6.7. 装载货物与卸载货物的条件是什么?

装载货物：小车当前位置处于 LoadingCargoPoint，且需将飞机绑定在小车上，否则无法装货。

卸载货物：飞机当前位置处于货物对应的 targetPosition，且货物绑定在飞机上，否则无法卸货。

6.8. 如何判断无人机相撞?

本次比赛统一定义无人机的安全间距为 5 米，如果两架无人机最短距离小于 5 米，会被判定为相撞，飞机状态转换为 CRASHED。注意：降落到距离很近的位置也会判定为相撞。

6.9. 如何判断小车相撞?

两车坐标点相距小于 3 米，会被判定为相撞。

6.10. 比赛时间说明

初赛 30min; 决赛 60min。正式比赛时间是 30min/60min，但是实际执行时间可能会超过这个数值，因此结果生成并反馈的时候可能大于 30min/60min

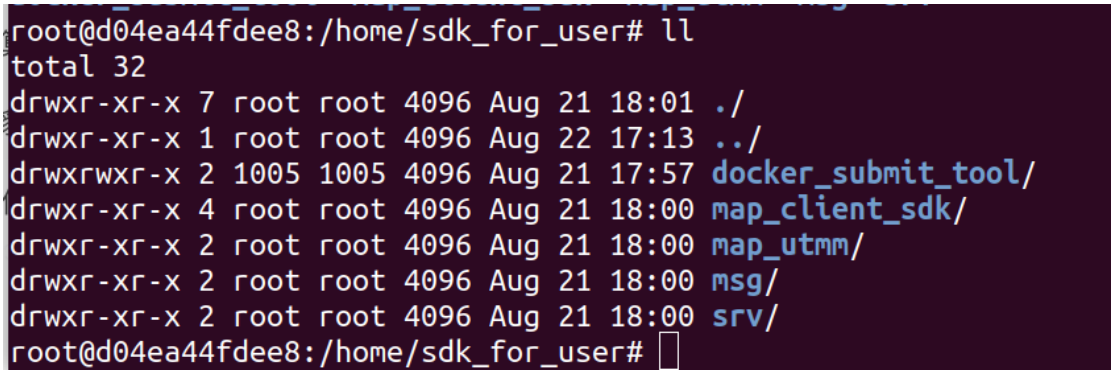
6.11. 决赛相对于初赛内容的变化有哪些?

1、比赛时间：60min

2、比赛内容：地图、货物等会有变化

6.12. 除了 ROS 外，是否提供了其他 api 或者文件可以使用?

在容器 **race_user_sdk_container 的/home/sdk_for_user/**目录下提供了供选手使用的其他文件



- docker_submit_tool：用于提交最终的比赛镜像
- map_client_sdk：提供了查询体素的 C++方法和 python 方法
- msg：比赛所需的 ROS topic 定义
- srv：比赛所需的 ROS servic 定义

6.13. 比赛使用的 NED 坐标系介绍

NED（North-East-Down）坐标系是一种常用于航空航天和导航领域的地理坐标系。该坐标系基于地球表面定义，具有以下特点：

原点：通常选择在地球表面某一特定点，例如飞机或无人机的当前位置。

轴方向：

N 轴（North）：指向地理北极，即地球的北方向。

E 轴（East）：指向地理东极，即地球的东方向，与 N 轴和 D 轴垂直。

D 轴（Down）：指向地心方向，即垂直向下，与 N 轴和 E 轴垂直。

这种坐标系的优点在于其直观性，特别适合描述飞行器或移动物体在地球表面的运动和位置。NED 坐标系中的位置和运动描述通常包括三个分量：北向分量（N）、东向分量（E）和下向分量（D）。

无人机中 NED 坐标理解

机体坐标系：机体坐标系固连飞机，其原点 取在多旋翼的重心位置上。 x 轴在多旋翼对称平面内指向机头（机头方向与多旋+字形或 X 字形相关）。 z 轴在飞机对称平面内，垂直轴向下。然后，按右手定则确定 y 轴 。

地球固联坐标系：通常以多旋翼起飞位置作为坐标原点 。先让 x 轴在水平面内指向某一方向，z 轴垂直于地面向下。然后，按右手定则确定 y 轴，坐标原点还有用地心的？**比如 NED 坐标系为 x 轴为正北方向，y 轴为正东方向，z 轴指向下**。

本次比赛中用到的地球固联坐标轴是 NED 坐标系，即 x，y，z 的方向固定不变