配置说明

（一） 环境部署

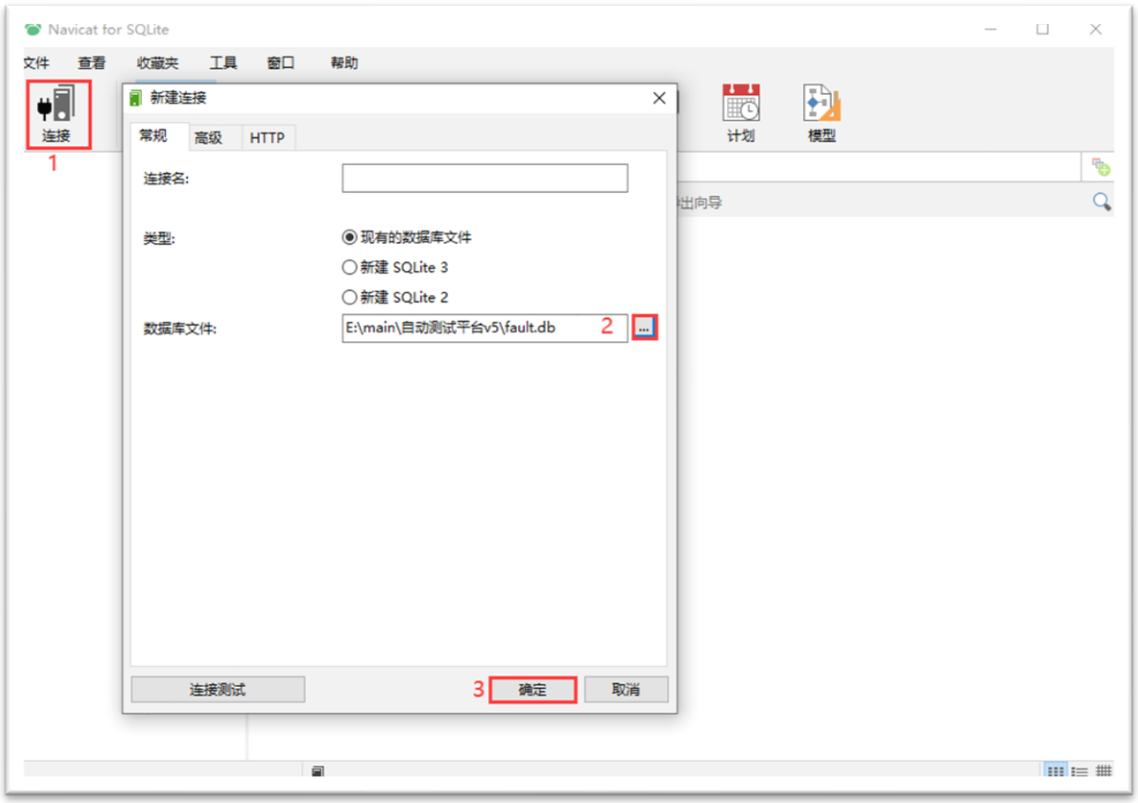
本平台架构为 RflySim+SQLite。

1、 升级最新版平台

2、 将 Model******Model.dll 复制到 PX4PSP\CopterSim\external\model 文件夹里

3、 压缩 Navicat_for_SQLite_11.0.10.rar 到当前文件夹

4、 打开 Navicat_for_SQLite_11.0.10 中的 navicat.exe，连接数据库



点击连接，在弹出的界面选择数据库文件右侧的三个点按钮，选择****AutoTestPlatForm\fault.db，再点击确定即可

使用说明

注：

（PX4PSP 默认在 c 盘，如果装在别的盘，需要改此文件： Model*******.bat（用 vscode 或记事本打开），将 C（见下图）改为自己的盘

```
6    SET PSP_PATH=C:\PX4PSP
7    SET PSP_PATH_LINUX=/mnt/c/PX4PSP
8    C:
```

1、 打开 vscode，点击文件->打开文件夹，选择 AutoTestPlatForm

2、 配置编译环境：打开 AutoTestAPI.py，将编译环境设置成 PX4PSP 文件中的编辑环境

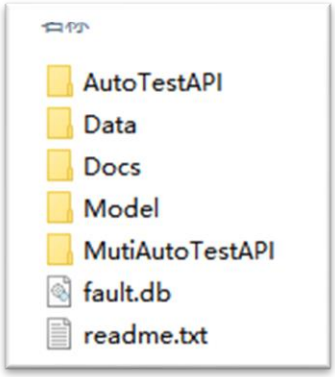
```
import AutoTestAPI
import PX4MavCtrlV4 as PX4MavCtrl


# json、bat文件所在目录
'''
Frame:
    1:四旋翼
    2:固定翼
    3:无人船
TestBatPath:
    [1,'Quadrotor','QuadModelSITL.bat']
    [2,'Fixedwing','FixedwingModelSITL.bat']
    [3,'USV','USVModelSITL.bat']
'''
TestBatPath=[2,'Fixedwing','FixedwingModelSITL.bat']

mav = PX4MavCtrl.PX4MavCtrler(20100)
mavobj = AutoTestAPI.AutoMavCtrler(mav,TestBatPath)
mavobj.AutoMavLoopStart()
```

3、 运行 AutoTestAPI.py。

4、 如需改测试机型，将 TestBatPath 换成对应的路径

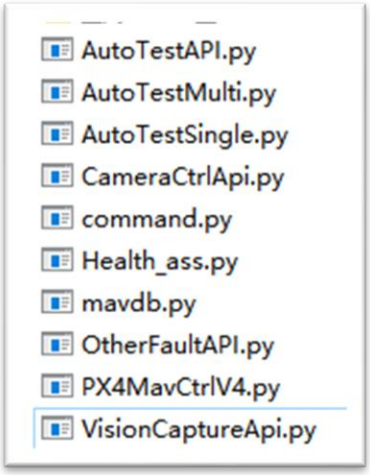
操作手册

（一） 文件结构说明



平台包含 6 个主要文件:

1）AutoTestAPI：主测试程序



- **AutoTestAPI 自动测试接口程序**
- **VisionCaptureApi 和 CameraCtrlApi 视觉接口**
- **PX4MavCtrlV4 通信接口**
- **Mavdb 数据接口**
- **Command 控制命令解析接口**
- **Health_Ass 安全评估接口**
- **AutoTestSingle.Py 单机测试实例**

- **AutoTestMulti.py 多机测试实例**

2) Model: 模型文件夹

- **Fixedwing 固定翼模型**
- **Quadrotor 四旋翼**
- **USV 无人船**

每个模型文件夹主要包括：仿真模型、测试脚本、测试 json 文件

3）Data 文件夹

每次仿真结束，会针对不同的机型，生成该机型仿真的数据

4）fault.db

数据库

（二） 机型

现有机型：四旋翼、固定翼、无人船

1）四旋翼功能含义解析如下：
1：时间类：
    1：等待时间：Wait（times）
    2： 等待复位：WaitReset (targetpos)
2：控制类：
    1：解锁：Arm（void）
    2：上锁：DisArm（void）
    3：位置控制：QuadPos（pos）
    4：速度控制：QuadVel（vel）
    5：降落：Land（pos）
    6：故障注入：FaultInject（param）

2）固定翼功能含义解析如下：
1：时间类：
    1：等待时间：Wait（times）

2: 等待复位：WaitReset (targetpos)

2：控制类：

1：解锁：Arm（void）

2：上锁：DisArm（void）

3：起飞：FixedTakeoff（pos）

4：位置控制：FixedPos（pos）

5：降落：Land（pos）

6：故障注入：FaultInject（param）

3）无人船功能含义解析如下：

1：时间类：

1：等待时间：Wait（times）

2: 等待复位：WaitReset (targetpos)

2：控制类：

1：解锁：Arm（void）

2：上锁：DisArm（void）

3：位置控制：USVPos（pos）

4：速度控制：USVVel（pos）

5：设置速度：GroundSpeed（speed）

6：故障注入：FaultInject（param）

控制序列说明：

具体的含义如下（********）：**

测试脚本会解析数据库中的控制序列，从而实现一次测试的控制逻辑。

测试序列由不同的指令组成，其中，每一条指令由分号结束（;），每条指令又由不同的字符串组成。（所有的字符都是英文符号）

其中每个指令的第一位代表一个操控类。现有的操控类含义解析如下：

1：时间类（包含等待时间、等待复位功能）

2：控制类（包含解锁、上锁、位置控制、速度控制、降落、故障注功能）

每条指令的第二位代表具体的功能，后面的位数代表对应函数的参数。现有的功能含义解析如下（以无人船为例）：

1：时间类：

1：等待时间：Wait（times）

2: 等待复位：WaitReset (targetpos)

2：控制类：

1：解锁：Arm（void）

2：上锁：DisArm（void）

3：位置控制：USVPos（pos）

4：速度控制：USVVel（pos）

5：设置速度：GroundSpeed（speed）

6：故障注入：FaultInject（param）

一个无人船例子(吊舱故障)如下：

2,1;1,1,5;2,3,150,0,30;1,2,150,0,30;2,5,500,100,60;1,1,5;2,7,123450,123450,123450,1,1,1,1,1,0.5;1,1,10;

解析如下：（每条指令的第一位代表一个类，第二位代表此类对应的函数，之后的位数都是函数的参数，没有参数则不用设置）

解锁（2,1）->等待 5s（1,1,5）->发送起飞命令，飞至[150,0,-30]处（2,3,150,0,-30）->等待复位[150,0,-30]（1,2,150,0,-30）->发送目标航点[200,100,-60]（2,5,500,100,-60）->等待 5s,（1,1,5）-> 故障注入，注入舵机故障（123450 为舵机故障）（123450,123450,123450,1,1,1,1,1,0.5），->等待 10s（1,1,10）

用户可以根据自己的需要任意组合

注：在故障注入时，模型采用的是标准化故障注入模块实现，具体是由 8 维整型 InSILInts 和 20 维 InSILFloats 参数实现。在数据库的控制序列中，对于 8 位的 InSILInts 和 20 位的 InSILFloats，只需要设置对应的故障 ID 和参数即可，剩余的位数不用补齐。

修改仿真机型在 AutoTestAPI\AutoTestSingle.py

```
'''
Frame:
    1:四旋翼
    2:固定翼
    3:无人船
TestBatPath:
    [1,'Quadrotor','QuadModelSITL.bat']
    [2,'Fixedwing','FixedwingModelSITL.bat']
    [3,'USV','USVModelSITL.bat']
'''
TestBatPath=[1,'Quadrotor','QuadModelSITL.bat']
```

在给出的例子中，其中的故障注入参数中，123451 为 inSILInts 的故障 ID（一位 ID 对应 2 位故障参数）。其他的类似。

现有的

现有的无人船故障 ID 和参数如下：

1）电机故障 ID：123451

参数：2 个，分别为两个电机的故障系数，范围为 0~1 (0 为完全损坏)

2）风故障（常风）ID：123459

参数：x、y、z 轴的风速，共三个，单位 m/s

3）风故障（阵风）ID：123540

参数：参数 1 为风的强度（即风速），参数 2 为风的频率

4）风故障（紊流风）ID：123541

参数：风速强度（1 个）

5）风故障（切向风）ID：123542

参数：风速强度（1 个）

6）传感器故障（加速度计）ID：123544

参数：叠加的噪声大小

7）传感器故障（陀螺仪）ID：123545

参数：叠加的噪声大小

8）传感器故障（磁力计）ID：123546

参数：叠加的噪声大小

9）传感器故障（气压计）ID：123547

参数：叠加的噪声大小

10）GPS 故障 ID：123548

参数：三个：噪声叠加大小、3Dlock 数、搜星数量

11）吊舱故障（无图像）ID：123549

参数：无

12）吊舱故障（云台乱转）ID：125340

参数：无

11）吊舱故障（图像噪声）ID：125341

固定翼故障 ID 和参数如下：

1）舵机故障 ID：123450

参数：6 个，分别为 5 个舵机的故障系数，第六个参数为总的故障增益

2）风故障（常风）ID：123459

参数：x、y、z 轴的风速，共三个，单位 m/s

3）风故障（阵风）ID：123540

参数：参数 1 为风的强度（即风速），参数 2 为风的频率

4）风故障（紊流风）ID：123541

参数：风速强度（1 个）

5）风故障（切向风）ID：123542

参数：风速强度（1 个）

6）传感器故障（加速度计）ID：123544

参数：叠加的噪声大小

7）传感器故障（陀螺仪）ID：123545

参数：叠加的噪声大小

8）传感器故障（磁力计）ID：123546

参数：叠加的噪声大小

9）传感器故障（气压计）ID：123547

参数：叠加的噪声大小

10）GPS 故障 ID：123548

参数：三个：噪声叠加大小、3Dlock 数、搜星数量

11）吊舱故障（无图像）ID：123549

参数：无

12）吊舱故障（云台乱转）ID：125340

参数：无

11）吊舱故障（图像噪声）ID：125341
参数：无

现有的四旋翼故障 ID 和参数如下：

1、电机故障（123450）
故障参数为 4 个，范围为[0,1]。0 为彻底损坏，1 为正常
2、螺旋桨故障（123451）
故障参数为 4 个，范围为[0,1]。0 为彻底损坏，1 为正常
3、用户自定义悬停时间（123452）
故障参数为 1 个，即自定义悬停时间
4、电池失效故障（123453）
无故障参数，直接触发
5、低电压故障（123454）
故障参数为 1 个，剩余电压比，范围[0,1]
6、低电量故障（123455）
故障参数 1 个，剩余电量比，范围[0,1]
7、负载掉落故障（123456）
故障参数 1 个，重量泄露比，范围[0,1]
8、负载漂移故障（123457）
故障参数 4 个，重量泄露比，以及 x，y，z 轴的漂移因子，范围[0,1]
9、负载泄露故障（123458）
故障参数 2 个，重量泄露比,以及泄露因子，范围[0,1]
10、　常风故障（123459）
故障参数 3 个，x,y,z 轴的风速
11、　阵风故障（123540）
故障参数 2 个，阵风强度（风速）以及阵风方向
12、紊流风故障（123542）
故障参数 1 个，风强度（风速）
12、　切向风故障（123543）
故障参数 1 个，风强度（风速）
13、　风噪声（123543）
故障参数 2 个，风振幅扰动因子（范围[0,1]）,风增益水平
14、　加速度计故障（123544）
故障参数 1 个，噪声增益水平
15、　陀螺仪故障（123545）
故障参数 1 个，噪声增益水平
16、　磁力计故障（123546）
故障参数 1 个，噪声增益水平
17、　气压计故障（123547）
故障参数 1 个，噪声增益水平
18、　GPS 故障（123548）
故障参数 1 个，噪声增益水平
19、　吊舱故障（无图像）ID：123549

参数：无

20、 吊舱故障（云台乱转）ID：125340

参数：无

21、 吊舱故障（图像噪声）ID：125341

Json 文件说明

其中：

1） faultcase 为测试用例，会同步到数据库中，展开 faultcase 可见下述内容：


```
"faultcase": [
    {
        "CaseID": 1,
        "Subsystem": "Sensor subsystem",
        "FaultType": "123451",
        "ControlSequence": "2,1;1,1,5;2,3,0,150,0;1,1,10;2,6,123451,1,0;1,1,10",
        "InterestedLog": {
            "*actuator_outputs_0.csv": [
                "timestamp",
                "noutputs",
                "output[0]",
                "output[1]"
            ],
            "*vehicle_command_0.csv": [
                "timestamp",
                "param1"
            ]
        },
        "TestStatus": "Finished"
    },
    {
        "CaseID": 2,
        "Subsystem": "Power subsystem",
        "FaultType": "123548",
        "ControlSequence": "2,1;1,1,5;2,3,0,150,0;1,1,10;2,6,123548,123548,0,3,0;1,1,10",
        "InterestedLog": {
            "*vehicle_gps_position_0.csv": [
                "timestamp",
                "lat",
                "lon",
                "alt",
                "satellites_used"
            ]
        },
        "TestStatus": "Finished"
    }
],
```


按照其格式，可以加你自己的用例，。

其中：

Caseid 为用例 ID

ControlSequence 为控制序列，控制序列见平台配置说明的控制序列说明部分

InterestedLog 为感兴趣的 log 数据，每次测试完毕后会将 InterestedLog 的数据单独提取出来放在一个表格里用于观察

2） testcase 为自动测试的顺序：

a) 单机模式：上图中”1,2”意为测试 1 号和 2 号用例（注：单机模式每条测试用例以逗号结尾），如果想测试所有的用例，将其改为:”testcase”:”all

b) 多机模式：如选择多机测试，每条测试 ID 以“;”（分号）结尾，例：如一次测试两个飞机，testcase 改为：”1,2;1,3;”此意为测试两轮，第一轮一号飞机选择测试用例 1，二号飞机选择测试用例 2、第二轮一号飞机选择测试用例 1，二号飞机选择测试用例 3。（注：每条测试用例要以;结尾，否则会检测不到会误测或报错）

3）testdata 为测试完后从 log 文件提取的数据

4）Vision 为视觉接口，改为“on”即打开摄像头，“off”为不打开

原理：将 db.json 中的用例添加到数据库中，然后再从数据库中提取出所需的用例进行测试。由于 CaseID 为数据库中的主键，不能存在相同的值，故当 db.json 中的 CaseID 和数据库中的 CaseID 相同时（即存在冲突），此时如果其他项完全相同，则 json 中的用例不会覆盖数据库中的用例（即 json 中的用例不会上传到数据库），否则，json 中的用例会覆盖数据库中的用例。

基于此，可以在 json 中自定义数据库用例以外的其他用例，做到一个补充用例的作用（即 json 中的 CaseID 不同于数据库中的 CaseID）；也可以在 json 中修改原来数据库中的用例（即 json 中的 CaseID 等于数据库中的 CaseID，再改其他项）

上述操作完全可以只在数据库中进行，方便之处在于，testcase 可以自定义测试的顺序和用例。