故障仿真数据收集实验

1. 实验目的

本实验旨在通过自动化脚本收集无人机在不同故障情况下的飞行数据，为后续的故障诊断、预测性维护以及智能决策系统提供训练和验证数据。通过软件在环(SITL)或硬件在环(HITL)仿真环境，模拟各种传感器和执行器故障场景，记录无人机在这些异常条件下的响应行为和状态变化。

该项目可以帮助研究人员和工程师：

- 收集标准化的故障数据集用于机器学习模型训练
- 自动化测试无人机在各种故障场景下的表现
- 分析无人机系统的可靠性和容错能力
- 建立预测性维护模型，提高飞行安全性

2. 实验要求

- 软件要求：Windows 10及以上版本；RflySim工具链[1]；MATLAB2022B以上版本。
  > 工具链安装时的编译命令为：px4_fmu-v6x_default，推荐PX4固件版本为：1.14.4。
- 硬件要求：笔记本/台式电脑1台；Pixhawk 6X[2]。

3.实验地址

例程目录：[安装目录]\RflySimAPIs\7.RflySimPHM\0.ApiExps\e9_data_collect

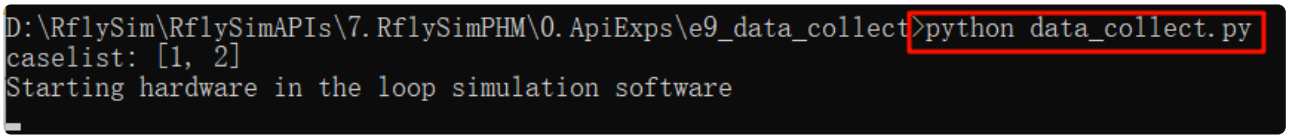
- data_collect.py：主控制程序，负责协调整个数据采集流程，包括仿真环境启动、指令解析执行、数据记录和存储等。
- command.py：定义各种飞行控制指令和故障注入方法，封装了诸如起飞、降落、位置控制、速度控制和故障注入等功能。

- mavdb.py：负责测试用例管理和测试结果存储，通过SQLite数据库和JSON配置文件协同工作，实现测试用例的动态加载和更新。
- Off_ctrl_connect_copter.py：负责建立与仿真器或真实飞控的连接，支持软件在环和硬件在环两种仿真模式。
- db.json：配置文件，存储测试用例信息，包括故障子系统、故障类型、控制序列和测试状态等。
- fault.db：SQLite数据库文件，存储测试用例和测试结果数据，包含faultcase表和testresult表。
- Python38AdminRun.bat：批处理脚本，用于启动预配置的Python环境。
- FaultModelV5SITL.bat：软件在环(SITL)仿真启动脚本。
- FaultModelV5HITL.bat：硬件在环(HITL)仿真启动脚本。

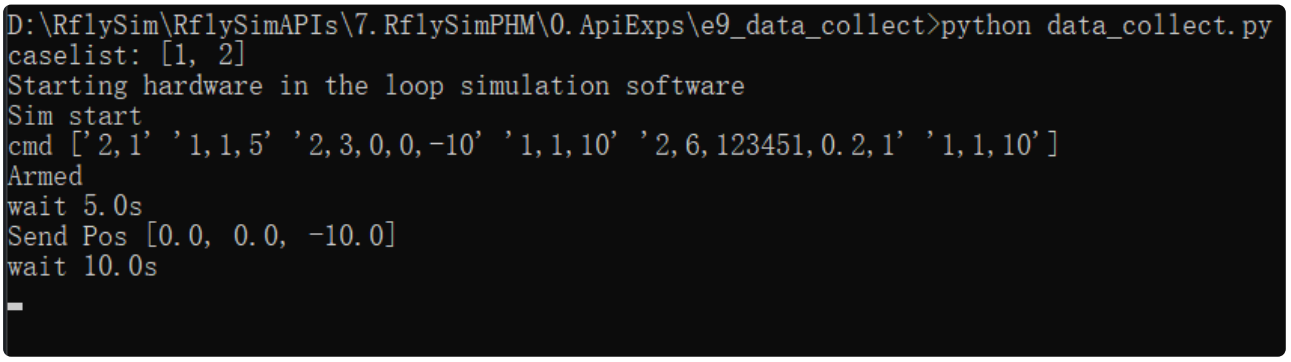
4. 实验内容或步骤

4.1 步骤1：软件在环仿真收集数据

双击运行Python38AdminRun.bat，打开集成好的python环境，在该环境下运行data_collect.py文件，输入：`python data_collect.py`.



运行成功后，在控制台输出如下信息：



该程序将会自动启动仿真并开始运行，运行结束后(如下图所示)，会将收集到的数据存储在`fault.db`文件中。

```
640 has been terminated.
                    All closed
Starting hardware in the loop simulation software
Sim start
cmd ['2,1' '1,1,5' '2,3,0,0,-10' '1,1,10' '2,6,123451,0.2,1' '1,1,10']
Armed
wait 5.0s
Send Pos [0.0, 0.0, -10.0]
wait 10.0s
Start Inject Fault
wait 10.0s
CaseID 2 test completed
1
Sim end
Exit hardware in the loop simulation software
CopterSim.exe                67360 Console                    1    265,268 K
SUCCESS: Sent termination signal to the process "CopterSim.exe" with PID 67360.
                                                                    QGroundControl.exe           17696 Consol
e                    1    336,452 K
SUCCESS: The process "QGroundControl.exe" with PID 17696 has been terminated.
                                                                    RflySim3D.exe                51892 Console
                   1      8,168 K
RflySim3D.exe                55132 Console                    1  1,038,008 K
SUCCESS: The process "RflySim3D.exe" with PID 51892 has been terminated.
                                                                    SUCCESS: The process "RflySim3D.exe" with PID 55
132 has been terminated.
                    All closed
All case test finish!

D:\RflySim\RflySimAPIs\7.RflySimPHM\0.ApiExps\e9_data_collect>
```

4.2 步骤2：硬件在环仿真收集数据

首先，需要修改`Off_ctrl_connect_copter.py`程序中如下程序段为自动启动硬件在环仿真来收集数据。

Off_ctrl_connect_copter.py > HIL_connect

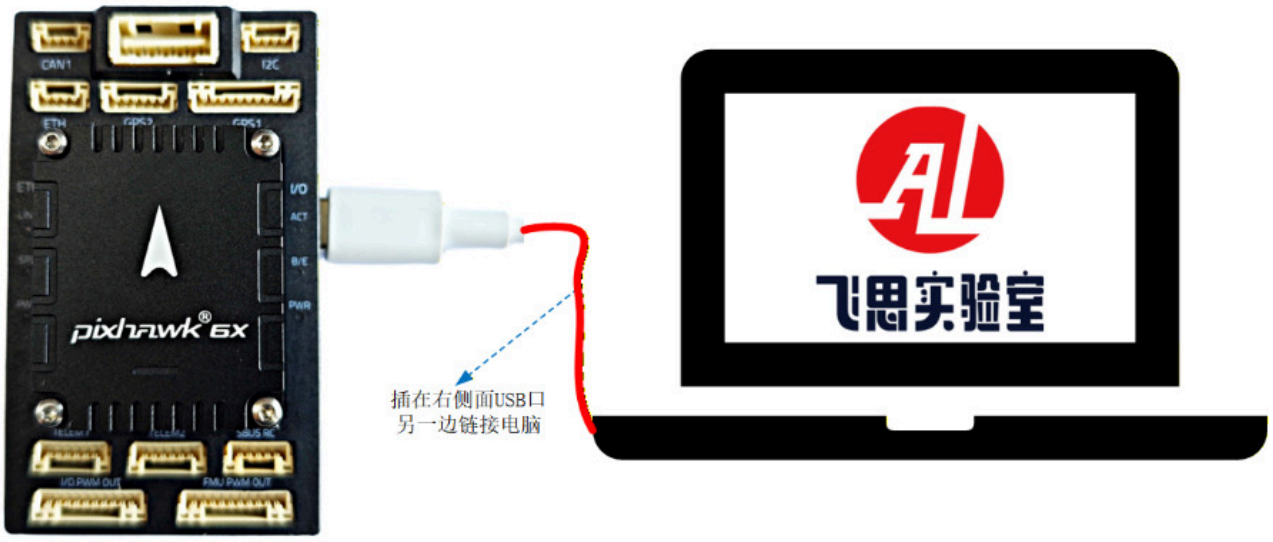
```python
import importlib.util
import PX4MavCtrlV4 as PX4MavCtrl
import time
import subprocess
import os
import sys

path = os.getcwd()

# 启动HIL软件
def HIL_connect(mav):
    # cmdStr = '{}//bat//FaultModelV5SITL.bat'.format(path)
    cmdStr = '{}//bat//FaultModelV5HITL.bat'.format(path)

    # 启动HIL软件
    global child
    child = subprocess.Popen(cmdStr,shell=True,stdout=subprocess.PIPE)
    print('Starting hardware in the loop simulation software')
    time.sleep(25)
    # print('20s,启动成功')
```

根据`[RflySim安装目录]\RflySimAPIs\1.RflySimIntro\2.AdvExps\e2.FCUIntro`将飞控进行还原处理。然后插入飞控到电脑中。如下图所示。



然后，双击运行Python38AdminRun.bat，打开集成好的python环境，在该环境下运行data_collect.py文件，输入：`python data_collect.py`.

```
D:\RflySim\RflySimAPIs\7.RflySimPHM\0.ApiExps\e9_data_collect>python data_collect.py
caselist: [1, 2]
Starting hardware in the loop simulation software
```

后续步骤与 步骤1 类似。

5. 关键知识点

关键知识点主要涉及实验的整体框架和核心代码实现，包括实验的整体架构设计思路和关键代码模块的解析。

5.1 关键知识点1：实验整体框架和思路

本实验采用模块化设计思想，主要包括以下几个核心组件：

1. **数据采集模块(data_collect.py)**: 主控制程序，负责协调整个数据采集流程，包括仿真环境启动、指令解析执行、数据记录和存储等。
2. **指令控制模块(command.py)**: 定义各种飞行控制指令和故障注入方法，封装了诸如起飞、降落、位置控制、速度控制和故障注入等功能。
3. **数据库管理模块(mavdb.py)**: 负责测试用例管理和测试结果存储，通过SQLite数据库和JSON配置文件协同工作，实现测试用例的动态加载和更新。

4. **连接控制模块(Off_ctrl_connect_copter.py)**: 负责建立与仿真器或真实飞控的连接，支持软件在环和硬件在环两种仿真模式。

实验流程如下：

1. 从db.json配置文件或fault.db数据库中读取测试用例
2. 启动仿真环境（SITL或HITL）
3. 按照测试用例中的控制序列逐步执行指令
4. 在故障注入后收集飞行数据
5. 将测试结果保存到数据库中

5.2 关键知识点2：关键代码解析

data_collect.py 核心逻辑分析

```
# 控制序列预处理
def CmdPro(seq):
    case = re.split(';',seq)
    cmd = np.array([])
    for i in range(len(case)):
        cmd = np.append(cmd,case[i])
    return cmd
```

该函数用于将分号分隔的控制指令序列拆分成数组形式，例如
将 `"2,1;1,1,5;2,3,0,0,-10"` 拆分为 `['2,1' '1,1,5' '2,3,0,0,-10']` 。

```
def DoCmd(ctrlseq):
    cmdseq = ctrlseq # '2,3,0,0,-20'
    cmdseq = re.findall(r'-?\d+\.?[0-9]*',cmdseq) # ['2', '3', '0', '0', '-20']
    cmdCID = cmdseq[0]
    if  cmdCID in CID:
        FID = FIDPro(cmdCID)
        # 有参数输入
        if len(cmdseq) > 2:
            # 提取参数
            param = cmdseq[2:len(cmdseq)]
            param = [float(val) for val in param]
            FID[cmdseq[1]](param)

        else:
            FID[cmdseq[1]]()
    else:
        print('Command input error, please re-enter')
```

该函数用于解析并执行具体的控制指令。指令格式为`类别ID,指令ID,参数1,参数2,...`，其中类别1为延时相关指令，类别2为飞行控制相关指令。

command.py 指令系统分析

```python
class Command:
    def __init__(self,mav):
        self.CID = 2
        self.mav = mav
        self.isArm = 0 # 解锁标志(解锁意味着开始记录数据)
        self.isDone = 0 # 任务完成标志
        self.LandFlag = 0
        self.LandFlagtag = 0
        # 初始化故障注入参数
        self.silInt = np.zeros(8).astype(int).tolist()
        self.silFloats = np.zeros(20).astype(float).tolist()
        self.isRecord = 0
        self.isRecord = 0
        self.isInject = 0

    def FaultInject(self,param): # ID = 6
        self.isDone = 0
        inInts = np.array([])
        inFloats = np.array([])
        for i in range(len(param)):
            if param[i] >= 123450:
                inInts = np.append(inInts,param[i])
            else:
                inFloats = np.append(inFloats,param[i])

        for i in range(len(inInts)):
            self.silInt[i] = inInts[i].astype(int)
        for i in range(len(inFloats)):
            self.silFloats[i] = inFloats[i].astype(np.double)

        print('Start Inject Fault')
        self.mav.SendMavCmdLong(183,16,568,1,1,1,1,1)
        # 作为一个故障输入的标志，方便后续对故障输入时间的判断
        self.mav.sendSILIntFloat(self.silInt,self.silFloats)
        self.isDone = 1
        self.isRecord = 1 # 故障注入之后开始收集数据
        self.isInject = 1
```

该类实现了飞行控制相关的各类指令，其中FaultInject方法用于注入故障，通过区分整数和浮点数参数来分别配置故障类型和故障强度。故障注入后设置相应的标记位，以便主程序开始记录数据。

mavdb.py 数据库管理分析

```
def dict_factory(cursor, row):
    d = {}
    for idx, col in enumerate(cursor.description):
        d[col[0]] = row[idx]
    return d

class mavdb:
    def __init__(self):
        self.cursor = 0
        self.mydb = 0
        self.is_tested = 0
        self.dbpath = os.getcwd()
        self.jsonpath = sys.path[0]
        mavdb.json_loadto_sql(self)
```

该类负责测试用例和测试结果的管理，使用SQLite数据库存储持久化数据，并通过dict_factory函数使查询结果以字典形式返回，便于访问。初始化时会调用json_loadto_sql方法同步JSON配置文件和数据库中的测试用例。

```
# 处理控制序列
def ctrl_seq_pro(self,case_id):
    # 获取游标
    mavdb.get_cursor(self)
    sql = '''
    select * from faultcase
    where CaseID = {}
    '''.format(case_id)
    self.cursor.execute(sql)
    data = self.cursor.fetchall()
    case_sequence = data[0].get('ControlSequence')
    case = re.split(';',case_sequence)
    cmd = np.array([])
    for i in range(len(case)):
        cmd = np.append(cmd,case[i])
    return cmd
```

该方法用于从数据库中获取特定测试用例的控制序列，并将其分割成指令数组。

6.参考资料

1. PX4 Autopilot官方文档
2. RflySim用户手册
3. Mavlink通信协议

4. Python MAVLink库使用指南
5. SQLite数据库Python接口文档

7.常见问题

Q1：data_collect.py运行时出现"ModuleNotFoundError: No module named 'PX4MavCtrlV4'"错误

A1：这是因为缺少PX4MavCtrlV4模块导致的错误。请确保已经正确安装RflySim工具链，并且在正确的Python环境中运行程序。推荐使用项目提供的Python38AdminRun.bat批处理文件启动Python环境。

Q2：提示"Failed to connect to simulator"

A2：这通常是因为仿真器没有正确启动或者连接配置有问题。请检查以下几点：

1. 确保正确选择了软件在环(SITL)或硬件在环(HITL)模式
2. 检查对应的批处理文件(FaultModelV5SITL.bat或FaultModelV5HITL.bat)路径是否正确
3. 确认防火墙没有阻止相关进程的网络连接
4. 如果是HITL模式，确认飞控已正确连接到电脑

Q3：数据收集完成后，fault.db数据库中没有新数据

A3：可能的原因包括：

1. 测试用例已经标记为已完成，可以检查testcase字段配置是否正确
2. 故障未成功注入，检查故障注入指令格式是否正确
3. 程序异常退出，查看控制台输出是否有错误信息

---

1. https://rflysim.com/ ↩

2. 推荐配置请见： https://rflysim.com/doc/zh/HowToInstall.pdf ↩