

目录

1. RflySim 故障注入构架简介.....	6
2. 开发环境配置	8
2.1 Windows 开发环境.....	8
2.1.1 VS Code 开发工具.....	8
2.1.2 Matlab 开发工具.....	9
3. 开发预备知识简介.....	10
3.1 Matlab-Simulink 代码生成 Visual studio 编译环境配置.....	10
3.2 Matlab-Simulink 代码生成操作步骤.....	11
3.2.1 模型编译参数设置.....	11
3.2.2 动态库文件的生成（GenerateModelDLLFile.p 的使用）	12
3.3 Matlab-Simulink 常用建模模块（UDP、Goto-From 标签）	12
3.3.1 Simulink 中 UDP 通信模型使用.....	12
3.3.2 Simulink 中 Goto-From 标签模型使用.....	12
3.4 bat 一键启动脚本的修改与使用（.bat 文件）	13
3.4.1 PX4PSP 启动路径修改.....	13
3.4.2 动态库加载路径修改.....	13
4. simulink 故障模块封装库（MulticopterModelLib.slx）的搭建与使用	15
4.1 电机故障（MotorFault）模块.....	15
4.1.1 电机故障注入 ID 和参数配置.....	15
MotorFaultTemp.FaultID=123450;.....	15
MotorFaultTemp.NoiseFaultID=111111;.....	15
MotorFaultTemp.MotorNum=int32(4);.....	15
4.1.2 故障模块中故障参数（FaultParamAPI）的封装传递.....	15
4.1.3 故障消息的订阅与触发.....	19
Goto 模块 FaultIn 标记-发布故障消息.....	19
From 模块 FaultIn 标记-订阅故障消息.....	19
FaultParamsExtract 自定义模块-故障触发与处理.....	20
4.2 螺旋桨故障（PropFault）模块.....	21
4.2.1 螺旋桨故障注入 ID 和参数配置.....	21
PropFault.FaultID = 123451;.....	21
PropFault.PropNum = int32(4);.....	21
4.2.2 故障模块中故障参数（FaultParamAPI）的封装传递.....	21
4.2.3 故障消息的订阅与触发.....	24

Goto 模块 FaultIn 标记-发布故障消息	24
From 模块 FaultIn 标记-订阅故障消息.....	24
FaultParamsExtract 自定义模块-故障触发与处理.....	25
4.3 电池故障（BatteryFault）模块.....	25
4.3.1 电池故障注入 ID 和参数配置	25
BatteryFault.PowOffFaultID = 123452;.....	25
BatteryFault.LowVoltageFaultID = 123453;.....	25
BatteryFault.LowCapacityFaultID = 123454;.....	25
4.3.2 故障模块中故障参数（FaultParamAPI）的封装传递.....	26
4.3.3 故障消息的订阅与触发	29
Goto 模块 FaultIn 标记-发布故障消息	29
From 模块 FaultIn 标记-订阅故障消息.....	29
FaultParamsExtract 自定义模块-故障触发与处理.....	30
4.4 负载故障（LoadFault）模块.....	30
4.4.1 负载故障注入 ID 和参数配置	30
LoadFault.LoadFallFaultID = 123455;.....	30
LoadFault.LoadShiftFaultID = 123456;.....	31
LoadFault.LoadLeakFaultID = 123457;.....	31
4.4.2 故障模块中故障参数（FaultParamAPI）的封装传递.....	31
4.4.3 故障消息的订阅与触发	31
Goto 模块 FaultIn 标记-发布故障消息	31
From 模块 FaultIn 标记-订阅故障消息.....	31
FaultParamsExtract 自定义模块-故障触发与处理.....	31
4.5 环境风故障（WindFault）模块.....	31
4.5.1 环境风故障注入 ID 和参数配置	31
WindFault.ConstWindFaultID = 123458;.....	31
WindFault.GustWindFaultID = 123459;.....	31
WindFault.TurbWindFaultID = 123540;.....	31
WindFault.SheerWindFaultID = 123541;.....	31
4.5.2 故障模块中故障参数（FaultParamAPI）的封装传递.....	32
4.5.3 故障消息的订阅与触发	32
Goto 模块 FaultIn 标记-发布故障消息	32
From 模块 FaultIn 标记-订阅故障消息.....	32
FaultParamsExtract 自定义模块-故障触发与处理.....	32
4.6 传感器故障（SensorFault）模块.....	32

4.6.1 传感器故障注入 ID 和参数配置.....	32
加速度计 SensorFault.AccNoiseFaultID = 123542;.....	32
陀螺仪 SensorFault.GyroNoiseFaultID = 123543;.....	32
磁罗盘 SensorFault.MagNoiseFaultID = 123544;.....	32
气压计 SensorFault.BaroNoiseFaultID = 123545;.....	32
GPS SensorFault.GPSNoiseFaultID = 123546;.....	32
4.6.2 故障模块中故障参数 (FaultParamAPI) 的封装传递.....	33
4.6.3 故障消息的订阅与触发.....	33
Goto 模块 FaultIn 标记-发布故障消息.....	33
From 模块 FaultIn 标记-订阅故障消息.....	33
FaultParamsExtract 自定义模块-故障触发与处理.....	33
5. simulink 模型消息接口.....	34
5.1 CopterSim 输入输出接口.....	34
5.1.1 消息输出接口.....	34
通过 udp 模块 (30101) 端口输出 DLL 模型消息, 使用 32 维数组接收	34
5.1.2 消息输入接口.....	35
通过 udp 模块 (30100) 端口接收外部输入消息.....	35
5.2 Rflysim 接口协议文件 Python-PX4MavCtrlV4.py.....	35
5.2.1 基于 udp 故障注入接口.....	35
5.2.2 基于串口故障注入接口.....	36
5.3 DLL 模型内部状态消息输入输出接口.....	37
5.3.1 消息的构建.....	37
通过 goto from 标签收集 (共 32 维数组).....	37
5.3.2 消息的输出.....	37
通过 outCopterData 输出端口输出.....	37
5.4 Python-飞控硬件信息交互输入输出接口.....	38
5.4.1 基于串口连接的串口传输.....	38
5.4.2 基于 usb 连接的 udp 传输.....	38
6. 自动化故障注入平台的搭建与使用.....	40
6.1 平台配置文件.....	40
6.1.1 测试用例配置 db.json.....	40
6.1.2 吊舱参数配置 Config.json.....	40
6.2 Rflysim 接口协议文件 PX4MavCtrlV4.py.....	41
6.2.1 故障注入协议类 PX4SILIntFloat.....	41
6.2.2 解锁/未解锁接口 SendMavArm.....	41

6.2.3 无人机目标位置接口 SendPosNED.....	42
6.2.4 无人机飞行速度接口 FlyVel.....	44
6.3 率模可靠度的安全评估算法 Health_ass.py.....	44
6.3.1 率模健康度的计算接口 Rate_Model_rank.....	44
6.3.2 记录差值序列数据接口 fly_log_record_allan.....	44
6.4 平台指令控制接口 command.py.....	44
6.4.1 数据库故障命令协议说明.....	44
6.4.2 未解锁命令接口 DisArm(self).....	44
6.4.3 解锁命令接口 Arm(self).....	44
6.4.4 飞行目标接口 FlyPos(self,pos).....	44
6.4.5 飞行速度接口 FlyVel(self,vel).....	44
6.4.6 着陆接口 Land(self):.....	44
6.4.7 故障注入参数接口 FaultInject(self,param).....	44
6.5 吊舱视觉 API VisionCaptureApi.py.....	44
6.5.1 开始视觉图像捕捉 startImgCap.....	44
6.5.2 更新视觉图像 sendUpdateUIImage.....	46
6.5.3 吊舱参数配置文件加载接口 jsonLoad.....	47
6.5.4 添加视觉传感器 addVisSensor.....	51
6.6 平台自动化测试 API AutoTest.py.....	52
6.6.1 自动化测试 TestcasePro().....	52
6.6.2 控制指令接口 DoCmd(ctrlseq).....	52
6.6.3 获取指令接口 FIDPro(cmdCID).....	52
6.6.4 指令序列控制接口 CmdPro(seq).....	52
6.7 数据库故障用例读写 mavdb.py.....	52
6.7.1 获取数据库游标接口 get_cursor(self).....	52
6.7.2 获取故障用例接口 get_fault_case(self).....	52
6.8 故障注入测试用例集的编写与使用	52
7. 软件在环 simulink 模型故障注入接口与使用	53
7.1 电机故障 (MotorFault) 注入与使用	53
7.2 螺旋桨故障 (PropFault) 注入与使用	55
7.3 电池故障 (BatteryFault) 注入与使用.....	57
7.4 负载故障 (LoadFault) 注入与使用	60
7.5 环境风故障 (WindFault) 注入与使用	63
7.6 传感器故障 (SensorFault) 注入与使用.....	66
7.7 GPS 故障 (GPSFault) 注入与使用	70

8. 软件在环可视化故障注入 APP (GUI)	74
8.1 触发按钮回调逻辑.....	74
8.2 udp 故障发送模块编写	75
8.3 故障注入协议编写	75
9. 硬件在环 PX4 飞控故障模块的编写与使用	76
9.1 GPS 故障模块的编写与使用	76
9.1.1 外部注入的 msg 文件 (消息格式)	76
9.1.2 msg 的头文件引用.....	76
9.1.3 故障消息的订阅与触发故障注入.....	76
9.2 电机故障模块的编写与使用	77
9.2.1 外部注入的 msg 文件 (消息格式)	77
9.2.2 msg 的头文件引用.....	78
9.2.3 故障消息的订阅与触发故障注入.....	78
9.3 遥控故障模块的编写与使用	79
9.3.1 外部注入的 msg 文件 (消息格式)	79
9.3.2 msg 的头文件引用.....	79
9.3.3 故障消息的订阅与触发故障注入.....	79
9.4 地磁故障模块的编写与使用	80
9.4.1 外部注入的 msg 文件 (消息格式)	80
9.4.2 msg 的头文件引用.....	81
9.4.3 故障消息的订阅与触发故障注入.....	81
10. 飞控日志的收集与处理.....	83
10.1 数据收集.....	83
10.2 数据实时获取	83
10.3 数据分析	83
10.4 数据标注.....	83
11. 安全评估算法设计与使用 Health_ass.py	83
11.1 数据筛选.....	83
11.2 数据方差值.....	83
11.3 率模加权值.....	83
11.4 安全评估	83
12. 基于神经网络的健康评估算法的设计与使用	83
12.1 故障数据的获取 AutoTestAPI.py	83
12.1.1 自启动脚本 FixedwingModelHITL	83
12.1.2 故障用例读取 caselist.....	83

12.2 数据集制作 data_handle.py.....	83
12.2.1 选取关键维度 fnmatch.....	83
12.2.2 关键数据合成（合成大表） join.....	83
12.3 模型训练 train.py.....	83
12.3.1 定义模型 DNN.....	83
12.3.2 训练 train_accuracy.....	83
12.4 在线评估 AutoTestAPI.py.....	83
12.4.1 引入模型 load_model.....	83
12.4.2 实时评估 model.predict.....	84

1. RflySim 故障注入构架简介

任何无人系统可以分为若干个组件（或子系统），其中包括电机、螺旋桨、陀螺仪等实体组件，也包含风、气压、障碍等虚拟组件。如图 6-1 所示，任何一个组件都可以认为包含三类模型：能耗模型、运动模型和故障模型。首先是运动模型负责描述组件的瞬态规律，其次是能耗模型用于描述组件的长态规律。运动模型和能耗模型共同描述了组件的正常的短期和长期运行规律，而故障模型则描述了组件因各种内外因素导致的偏离正常运行状态的规律。

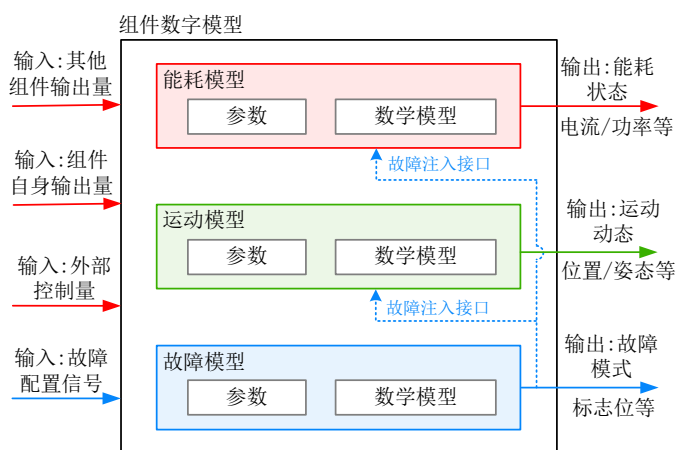


图 6-1 载具组件统一建模框架

每个组件可能包含三个基本模型中的一个或多个，只有在考虑进行故障测试时，故障模型才是必须的。例如，电池正常情况下主要由能耗模型描述，而电机则主要由运动模型描述。但是当需要考虑故障注入时，则需要将故障模型通过合适的方式嵌入到组件模型中。如图 6-2 所示的电池模型为例，电池模型主要包含了能耗模型和故障模型，其中能耗模型主要对应了电池的压降曲线，而故障模型主要对应了容量损耗曲线。电池的容量损耗主要是因为电池充放电次数的增加而产生的电解质和原件老化，带来的电量的衰减，最终体现在电池的压降速度上，而电压的下降，最终会影响载具的整体运动规律。

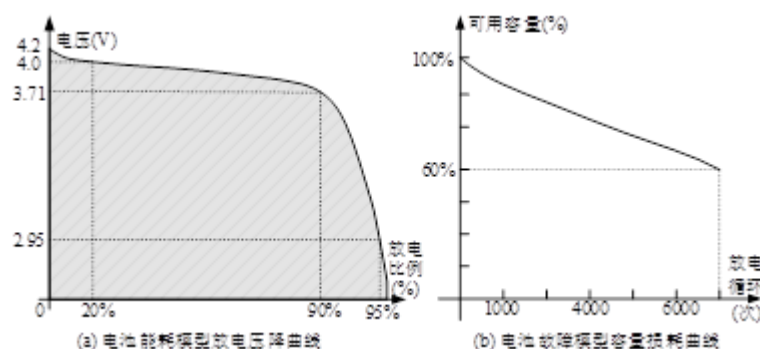


图 6-2 电池模型曲线

如图 6-1 所示的统一建模框架中，故障模型的输入主要是故障配置信号（包括是否使能、是否触发、故障参数等），当然也会根据需求引入自身组件状态量、外部组件状态量和外部控制量等信息。例如，传感器高温失效的故障，需要输入故障配置信号（是否使能故障和触发温度等）和外部组件状态量（主要是温度模型组件的输出）。故障模型的输出主要是一些标志量，例如是否已经触发故障、触发时间戳、故障参数等，用于反馈给自动测试程序。故障模型内部本身包含了数学模型和参数两部分，其中故障模型参数主要是为了增加模型的扩展性，将来通过修改参数，可以将本故障模块应用到其他系统；而数学模型则描述了故障的触发机制，以及对其他模型的影响规律。此外，由于每个组件都可能会有多种不同的故障（高温失效、电磁干扰等），那么在传入故障触发信号时，需要具备编号功能，故障模型需要先识别出是哪一类故障，然后产生对应的故障信号，注入到能耗模型或运动模型中。

故障模型主要是识别出故障源和触发时间，然后将故障信号传给运动模型或能耗模型，最终才能体现在整机的运行效果上。故障模型对运动模型和能耗模型的影响主要体现在三个方面：（1）参数影响。改变原有运动模型与能耗模型的参数，例如传感器故障时，可能导致数据噪声变大，也就是传感器输出的噪声方差增大；（2）模态影响。直接改变了原有运动模型与能耗模型的数学表达式，可以认为是发送了模态切换，例如，传感器失效的故障，直接让传感器变成全 0 输出模态；（3）叠加影响。直接生成干扰量，叠加在运动模型和能耗模型的信号量（可能是输入、中间状态或输出信号）上，例如外部振动故障，可能导致传感器的输出上，叠加其他的噪声。

在仿真系统中，故障的触发机制还可以分为确定性触发和不确定性触发（概率触发）。确定性触发是指明确已知在什么时间或状态下触发故障，它可以用各种逻辑判断函数直接描述。概率（不确定性）触发是指在某时间或状态下以一定的概率触发故障，它通常需要结合随机过程的概率模型来进行描述。不确定性故障触发通常在整机最终的安全测试阶段非常有效，需要设置好每个部件的故障概率（在仿真中可以将故障出现概率统一合理地调大，以保证在尽量短时间内能有故障出现），然后进行大量的任务模拟，以便测试在单个或多个故障突发时，对整体任务可能产生的影响。

2. 开发环境配置

2.1 Windows 开发环境

2.1.1 VS Code 开发工具

打开 Visual Studio Code，选择打开文件夹，打开文件夹 RflySimAPIs3.0\6.RflySimPHM\1.BasicExps\e4_FaultInjectAPITest_py。



对 FaultInjectAPITest.py 其中的故障注入代码按照 RflySimAPIs3.0\6.RflySimPHM\1.BasicExps\e4_FaultInjectAPITest_py 中的 FaultInjectAPITest_py 中的故障注入代码更改为螺旋桨模块故障（螺旋桨模块故障注入代码可以查看参考文献），并对故障参数进行修改。

```
silInt=np.zeros(8).astype(int).tolist()
silFloat=np.zeros(20).astype(float).tolist()
silInt[0:2]=[123450,123450]
silFloat[0:4]=[0,0,0,0]
# silInt[0:1]=[123540]
# silFloat[0:2]=[15,20]
mav1.sendSILIntFloat(silInt,silFloat)
print('Inject a fault, and start logging')
flag=2
```

对 FaultInjectAPITest.py 进行调试，即可在 RflySim3D 中观察到无人机起飞，并发生故

障。



2.1.2 Matlab 开发工具

MATLAB 安装包下载路径: [MATLAB - 技术计算语言 产品信息 \(mathworks.cn\)](https://www.mathworks.cn/products/matlab/)

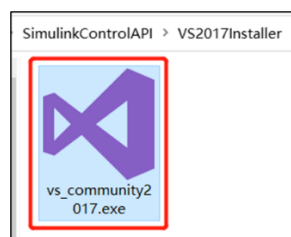
3. 开发预备知识简介

3.1 Matlab-Simulink 代码生成 Visual studio 编译环境配置

这里推荐安装 Visual Studio 2017，在线安装步骤（需联网）如下：

双击“RflySimAPIs\SimulinkControlAPI\VS2017Installer\vs_community2017.exe”

本课程内容只需勾选右图的“C++的桌面开发”即可。



注意：高版本 MATLAB 也可安装 VS2019，但是 MATLAB 只能识别到低于自己版本的 Visual Studio，因此 MATLAB 2017b 无法识别 VS 2019。

注意：请不要更改 VS 默认安装目录（例如装到 D 盘），会导致 MATLAB 无法识别。不能使用 Mingw 编译器，需 VS。

MATLAB 编译器安装确认：

在 MATLAB 的命令行窗口中输入指令“mex -setup”。

```
命令窗口
>> mex -setup
MEX 配置为使用 'Microsoft Visual C++ 2017 (C)' 以进行 C 语言编译。
警告：MATLAB C 和 Fortran API 已更改，现可支持
包含 2^32-1 个以上元素的 MATLAB 变量。您需要
更新代码以利用新的 API。
您可以在以下网址找到更多的相关信息：
http://www.mathworks.com/help/matlab/matlab\_external/upgrading-mex-files-to-use-64-bit-
要选择不同的 C 编译器，请从以下选项中选择一种命令：
Microsoft Visual C++ 2013 (C) mex -setup:D:\MATLAB\R2017b\bin\win64\mexopts\msvc2013.xml C
Microsoft Visual C++ 2015 (C) mex -setup:D:\MATLAB\R2017b\bin\win64\mexopts\msvc2015.xml C
Microsoft Visual C++ 2017 (C) mex -setup:C:\Users\dream\AppData\Roaming\MathWorks\MATLAB\R2
要选择不同的语言，请从以下选项中选择一种命令：
mex -setup C++
mex -setup FORTRAN
fx >>
```

一般来说会自动识别并安装上 VS 2017 编译器，如右图所示显示“MEX 配置使用 ‘Microsoft Visual C++ 2017’以进行编译”说明安装正确。

若有其他编译器，本页面还可以切换选择 VS 2013/2015 等其他编译器。

3.2 Matlab-Simulink 代码生成操作步骤

3.2.1 模型编译参数设置

常见求解器分类：

(1) 定步长与变步长求解器

定步长求解器的仿真步长为定制，没有误差控制机制；变步长求解器在仿真过程中需要计算仿真步长，通过增加/减小步长来满足所设定的误差宽容限。在生成实时运算代码时，必须使用定步长求解器，若不打算配置模型代码生成，求解器的选择根据建立模型而定。通常，变步长求解器可以减少仿真时间，定步长求解步长越小，仿真精度越高，故在同样仿真精度要求下，在采用定步长求解器进行仿真时，整个仿真过程必须采用变步长求解器中的最小步长。

(2) 连续与离散求解器

在定步长与变步长求解器中均有连续与离散求解器。连续与离散求解器都是依靠模块来计算所有离散状态值。定义离散状态的模块负责在每个步长的时间点计算离散状态值，连续求解器是通过数值积分来计算定义连续状态的模块的状态值。在选择求解器时，必须先确定模型中是否需要离散求解器。在模型中若没有连续状态模块，求解器采用连续、离散均可，若有连续状态模型必须采用连续求解器。

(3) 显式与隐式求解器

隐式求解器的应用主要解决模型中的刚性问题，显式求解器应用解决非刚性问题。譬如，在控制系统中，控制部件反应灵敏是快变的，具有小的时间常数，而受控对象一般惯性大，慢变的，具有大的时间常数。通常将具有非常不同时间尺度的系统称之为刚性系统，通俗讲，就是系统中含有时间快变和慢变分量（同时含有小时间常数和大时间常数的系统）。刚性系统有非常大的恢复能力使得快变化分量的扰动很快就衰减，当数值积分这样一个系统时，一旦快变分量消失时期望选取合适的时间步长用于计算慢变分量。故刚性系统的实质是要计算的解是慢变化，但存在迅速衰减的扰动，这样的扰动出现使得慢变解的数值计算复杂化。故，对系统中的震荡现象，隐式求解远比显式求解稳定，但计算的消耗比显式求解大，它需要在仿真的每个步长利用 Newton-like 方法计算所产生的雅克比矩阵和代数方程组。为了减少计算消耗，Simulink 提供了计算雅克比方法的参数，提高仿真性能。

(4) 单步与多步求解器

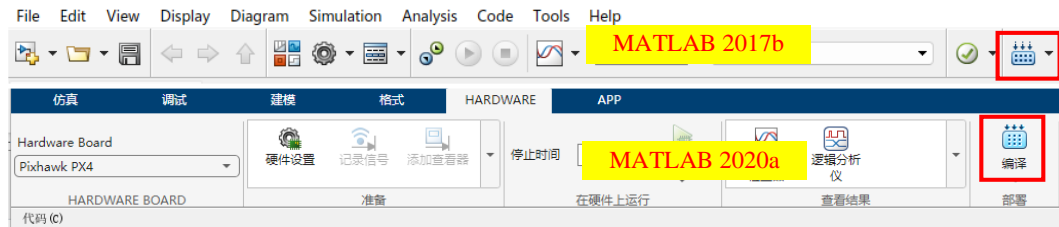
在 Simulink 求解库中提供了单步与多步求解器。单步求解就是在计算系统当前时刻 $y(tn)$ ，需要利用前一时刻 $y(tn-1)$ 以及在 $tn-1$ 与 tn 之间多个时间点的微分量（这些时间点称为微步长）；多步求解器就是利用系统前多个时刻的值计算当前时刻的值。Simulink 提供了一个显式多步求解器 `ode113` 和一个隐式多步求解器 `ode15s`，这两个都是变步长求解器。

(5) 变阶式求解器

Simulink 提供两种变阶式求解器，ode15s 求解器利用 1 阶到 5 阶仿真；ode113 应用 1 阶到 13 阶。对于 ode15s 可以设置最高阶次。

3.2.2 动态库文件的生成 (GenerateModelDLLFile.p 的使用)

- (1) 首先，使用例程中的初始化文件，并点击运行。
- (2) 打开想要使用的文件，并对其进行编译。



在完成编译后，会生成.slx 的 Simulink 组件文件，一个压缩包文件以及其余的一系列的编译文件。

- (3) 之后，我们可以点击运行 GenerateModelDLLFile.p 文件，从而生成.dll 动态库文件。

3.3 Matlab-Simulink 常用建模模块 (UDP、Goto-From 标签)

3.3.1 Simulink 中 UDP 通信模型使用

1. 创建 UDP 发送器和接收器 block:

在 Simulink 库中，可以找到“UDP Send”和“UDP Receive”两个 Block。将这些 Block 添加到模型中，并连接他们以便进行通信。

2. 配置 UDP 参数:

在 UDP Send 和 UDP Receive Block 中，需要配置相应的 UDP 参数，例如地址、端口号、数据包大小等。还可以选择使用自定义的 UDP 头部或 payload。

3. 生成测试数据:

在 UDP Send Block 中，可以使用“Generate Test Data”功能来生成测试数据。这些数据将被发送给 UDP Receive Block。

4. 验证结果:

在 UDP Receive Block 中，可以使用“Result”PORT 来验证接收到的数据。还可以使用“Plot”Portail 来 visualize Received 数据。

3.3.2 Simulink 中 Goto-From 标签模型使用

Goto-From 标记是一对标记，用于指定状态机中两个状态之间的转换。第一个标记“Goto”指示系统应转换到的目标状态，而第二个标记“From”指定应从中转换的当前状态。它们共同定义了两种状态之间的有向边，允许系统根据某些条件更改其状态。

要在 Simulink 中使用 Goto-From 标签，请执行以下步骤:

1. 创建新的状态机: 首先创建一个新的 Simulink 模型, 然后从库中选择 “State Machine” 模块。这将创建一个具有两种状态的基本状态机, 即“初始”和“最终”。

2. 添加状态: 单击状态机块, 然后按 “添加状态” 按钮向计算机添加新状态。还可以根据需要删除或重命名现有状态。

3. 添加转换: 要添加两种状态之间的转换, 请单击状态机块中的 “转换” 选项卡, 然后单击 “新建转换” 按钮。这将创建一个连接两个状态的新过渡箭头。

4. 分配 Goto-From 标签: 选择过渡箭头, 然后单击属性检查器中的 “标签” 选项卡。在这里, 可以分配一个 “Goto” 标签来指示目标状态, 并分配一个 “From” 标签来指示当前状态。例如, 如果希望系统从状态 A 转换到状态 B, 则应将 “Goto” 标记设置为 “B”, 将 “From” 标记设置为 “A”。

5. 设置条件: 还可以设置转换的条件。单击过渡箭头, 然后在属性检查器中选择 “条件” 选项卡。在这里, 可以指定必须为 true 才能进行转换的逻辑表达式。

6. 运行仿真: 定义状态机和转换后, 可以通过单击 Simulink 工具栏中的 “运行” 按钮来运行仿真。系统将以初始状态启动, 并根据指定的条件在状态之间转换。

3.4 bat 一键启动脚本的修改与使用 (.bat 文件)

3.4.1 PX4PSP 启动路径修改

在例程中, 我们经常会使用到一些软件在环一键启动脚本。但是, 在平台进行安装时, 由于安装的位置不同, 可能会发现脚本无法运行的情况, 这是我们需要对脚本中的 PXP 启动的路径进行修改。

我们可以右键点击脚本, 选择 “显示更多选项”, 点击 “编辑” 对脚本进行修改。我们可以在最开始的地方看到如下的几行代码:

```
REM Set the path of the RflySim tools
SET PSP_PATH=C:\PX4PSP
SET PSP_PATH_LINUX=/mnt/c/PX4PSP
C:
```

它们与设置 RflySim 工具的路径有关。其中的 C 表示 PX4PSP 安装在 C 盘中, 可以根据自己平台安装的位置进行修改。

3.4.2 动态库加载路径修改

动态库加载路径修改是同样是基于 bat 脚本, 在指定 PX4PSP 的路径后, 在后续的 bat 脚本中, 我们可以看到如下的代码:

```
REM Set use DLL model name or not, use number index or name string
REM This option is useful for simulation with other types of vehicles instead of multicopters
QuadModel FaultModel QuadModelv
set DLLModel=MulticopterModel
```

```
REM Check if DLLModel is a name string, if yes, copy the DLL file to CopterSim folder
SET /A DLLModelVal=DLLModel
if %DLLModelVal% NEQ %DLLModel% (
    REM Copy the latest dll file to CopterSim folder
    Copy                                                    /Y
    "%~dp0"%DLLModel%.dll %PSP_PATH%\CopterSim\external\model\%DLLModel%.dll
)
```

其中，最后一行代码表示是从当前目录下将 dll 文件拷贝到指定的 C 盘中 PX4PSP 的文件下。第四行的代码则是需要进行拷贝的 dll 文件名称，如果我们生成的 dll 文件名称与脚本中不一致，便无法进行拷贝，这是便需要我们对它进行修改。

4. simulink 故障模块封装库（MulticopterModelLib.slx）的搭建与使用

4.1 电机故障（MotorFault）模块

```
% Define the 32-D ModelInParams vector for external modification
FaultParamAPI.FaultInParams = zeros(32,1);

MotorFaultTemp.FaultID=123450;
MotorFaultTemp.NoiseFaultID=111111;
MotorFaultTemp.MotorNum=int32(4);
```

4.1.1 电机故障注入 ID 和参数配置

```
MotorFaultTemp.FaultID=123450;
```

字段 ▲	值
FaultID	123450
MotorNum	4

```
MotorFaultTemp.NoiseFaultID=111111;
```

字段 ▲	值
FaultID	123450
NoiseFaultID	111111
MotorNum	4

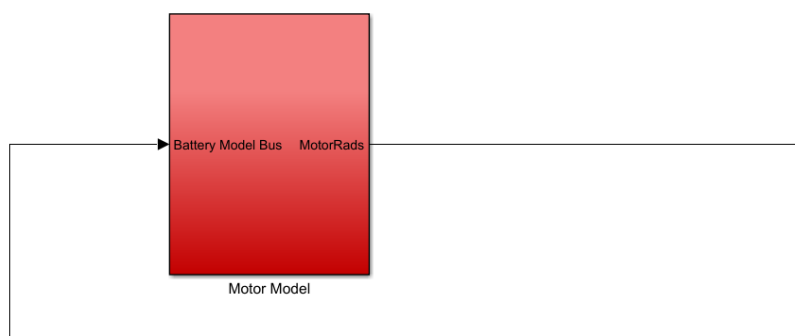
```
MotorFaultTemp.MotorNum=int32(4);
```

此处表示电机数量有四个。

4.1.2 故障模块中故障参数（FaultParamAPI）的封装传递

model_param_api_type	
MotorFault	1x1 struct
MotorFault1	1x1 struct
MotorFaultTemp	1x1 struct
PropFault	1x1 struct
SensorFault	1x1 struct
WindFault	1x1 struct

我们可以通过工作区对封装模块引用参数进行查看。
双击打开初始化脚本，单机运行。打开例程文件，我们可以对封装模块进行查看。

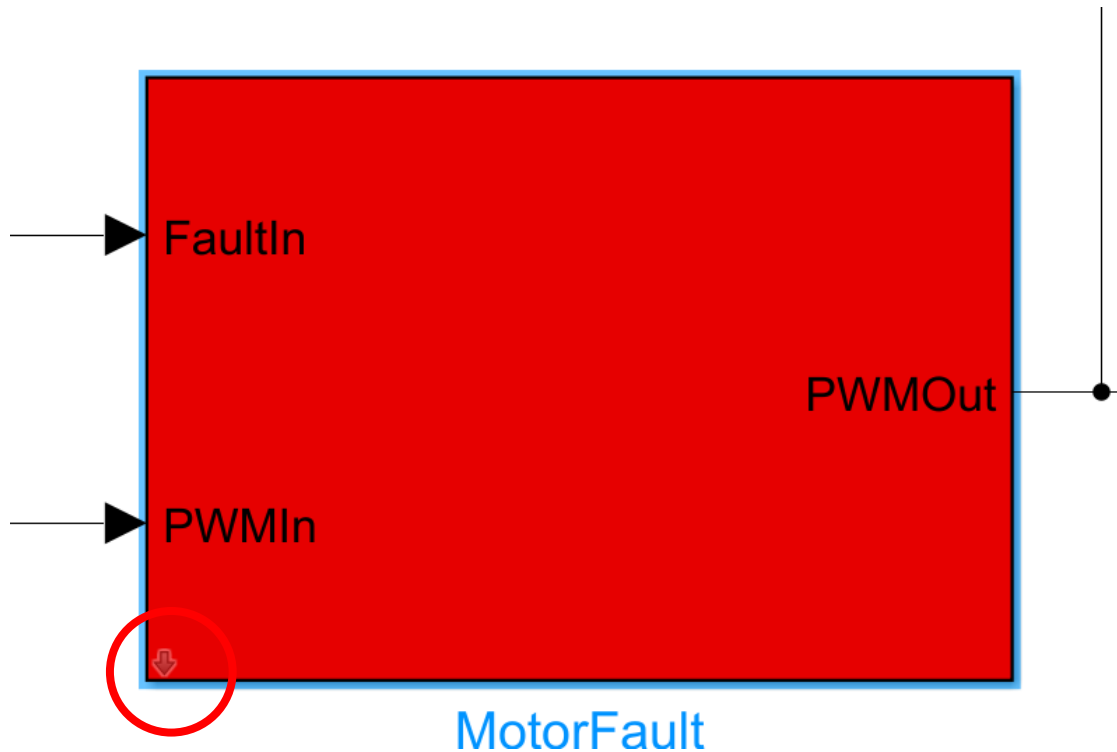
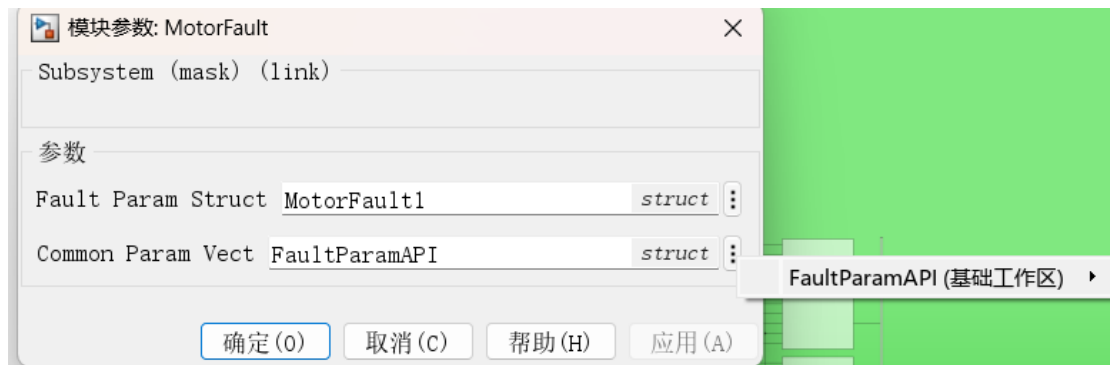


此模块中参数由工作区导入。我们可以右键点击，查看封装。

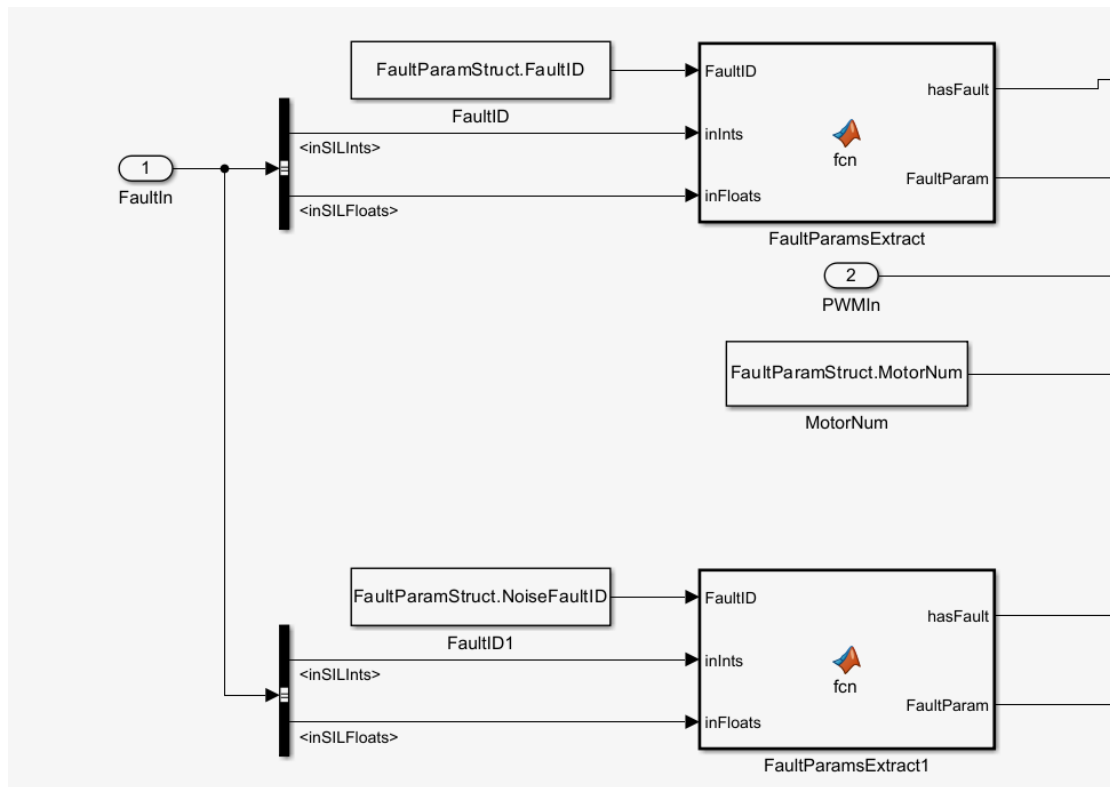
类型	提示	名称	属性编辑器
A	%<MaskType>	DescGroupVar	名称 ComParamVect
	%<MaskDescription>	DescTextVar	
#1	参数	ParameterGroupVar	别名 FaultParamAPI
	Fault Param Struct	FaultParamStruct	值 Common Param Vect
#2	Common Param Vect	ComParamVect	提示 edit
			类型

我们可以看到,此处应用的值跟上面相同,导入了工作区中的值,只是对其进行了命名。之后,我们进入封装内部,进行观察。

双击可查看模块参数。



点击红圈中的箭头，我们便可以进入封装。



我们可以看到两个电机模块，其中所使用的参数，我们都可以从工作区中看到，前方故障 ID 模块命名方法为 32 位的故障参数名加上故障 ID 组成。双击点击模块，我们可以看到其中的代码逻辑。

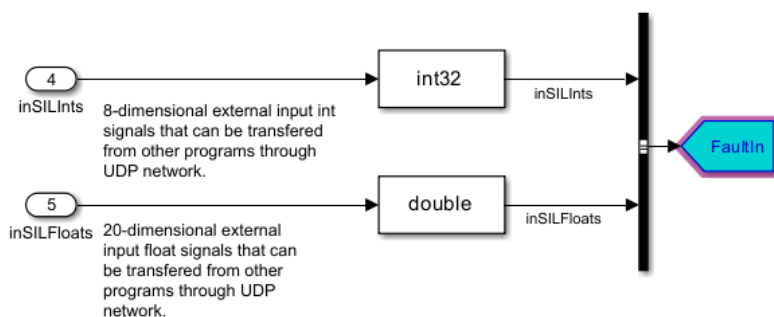
```
function [hasFault, FaultParam] = fcn(FaultID,inInts,inFloats)
persistent hFault;
persistent fParam;

if isempty(hFault)
    hFault=false;
end
if isempty(fParam)
    fParam=zeros(20,1);
end
```

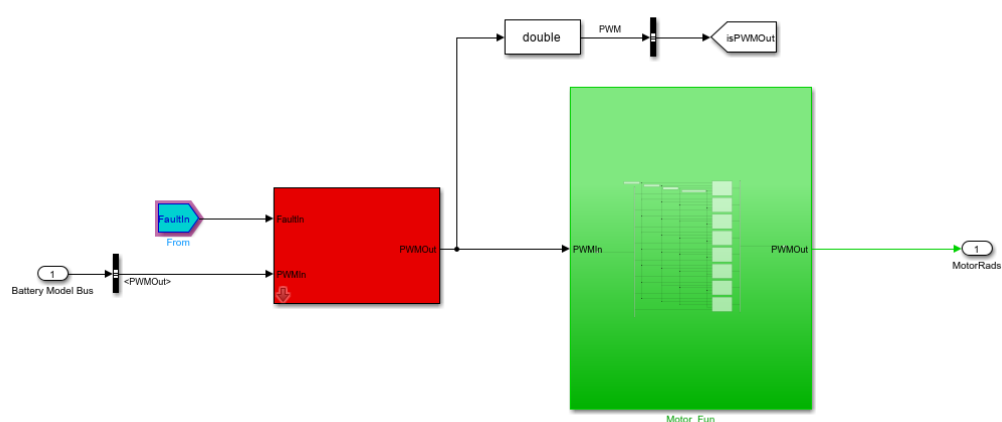
4.1.3 故障消息的订阅与触发

Goto 模块 FaultIn 标记-发布故障消息

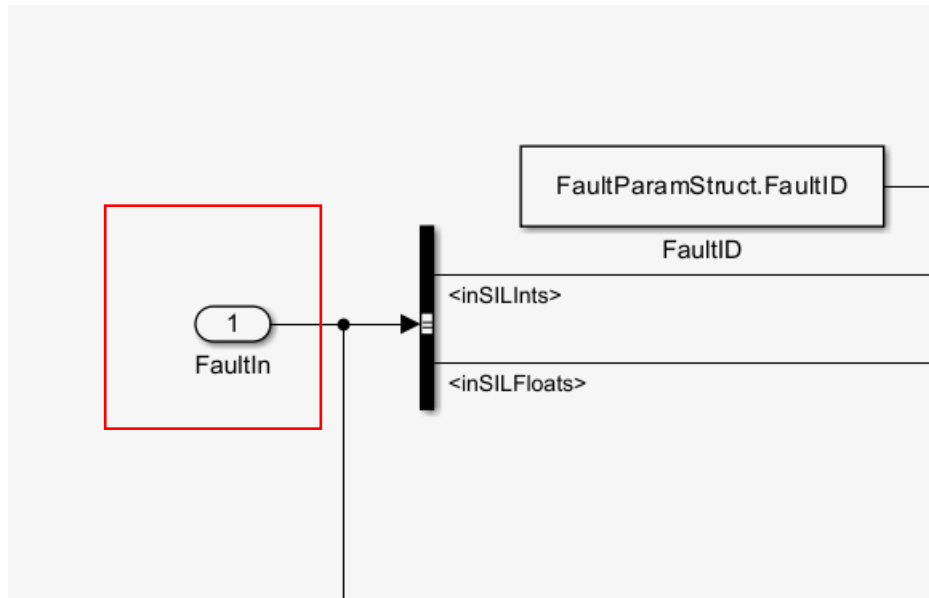
From 模块 FaultIn 标记-订阅故障消息



通过初始化文件对设置的参数进行导入，再通过 GOTO—FROM 模块进行数据传输。



上层从工作区导入的数据传输的此处，在注入到故障模块中。



此处同样为 FROM 模块，通过模块层层注入，从而达到故障注入的效果。

FaultParamsExtract 自定义模块-故障触发与处理

```

1  function [hasFault, FaultParam] = fcn(FaultID,inInts,inFloats)
2      persistent hFault;
3      persistent fParam;
4
5      if isempty(hFault)
6          hFault=false;
7      end
8      if isempty(fParam)
9          fParam=zeros(20,1);
10     end
11
12     hFaultTmp=false;
13     fParamTmp=zeros(20,1);
14     j=1;
15     for i=1:8
16         if inInts(i) == FaultID
17             hFaultTmp=true;
18             fParamTmp(2*j-1)=inFloats(2*i-1);
19             fParamTmp(2*j)=inFloats(2*i);
20             j=j+1;
21         end
22     end
23     if hFaultTmp
24         hFault=hFaultTmp;
25         fParamTmp(17:20) = inFloats(17:20);
26         fParam=fParamTmp;
27     end
28
29     hasFault=hFault;
30     FaultParam=fParam;
31

```



这里代码显示的故障处理与触发的过程。

4.2 螺旋桨故障（PropFault）模块

```
MotorFault.MotorNum=int32(4);  
%Prop Fault Struct  
PropFault.FaultID = 123451;  
PropFault.PropNum = int32(4);  
%
```

4.2.1 螺旋桨故障注入 ID 和参数配置








PropFault.FaultID = 123451;



字段 ▲	值	
 FaultID	123451	
 PropNum	4	

PropFault.PropNum = int32(4);

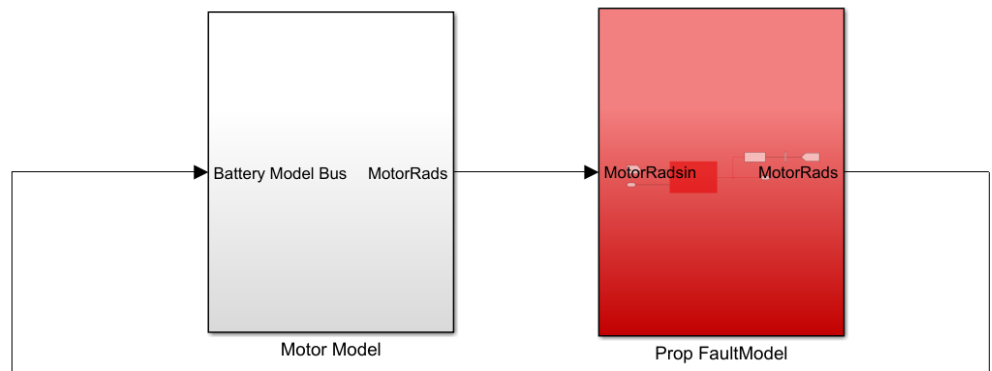
此处表示螺旋桨数量为 4 个。

4.2.2 故障模块中故障参数（FaultParamAPI）的封装传递

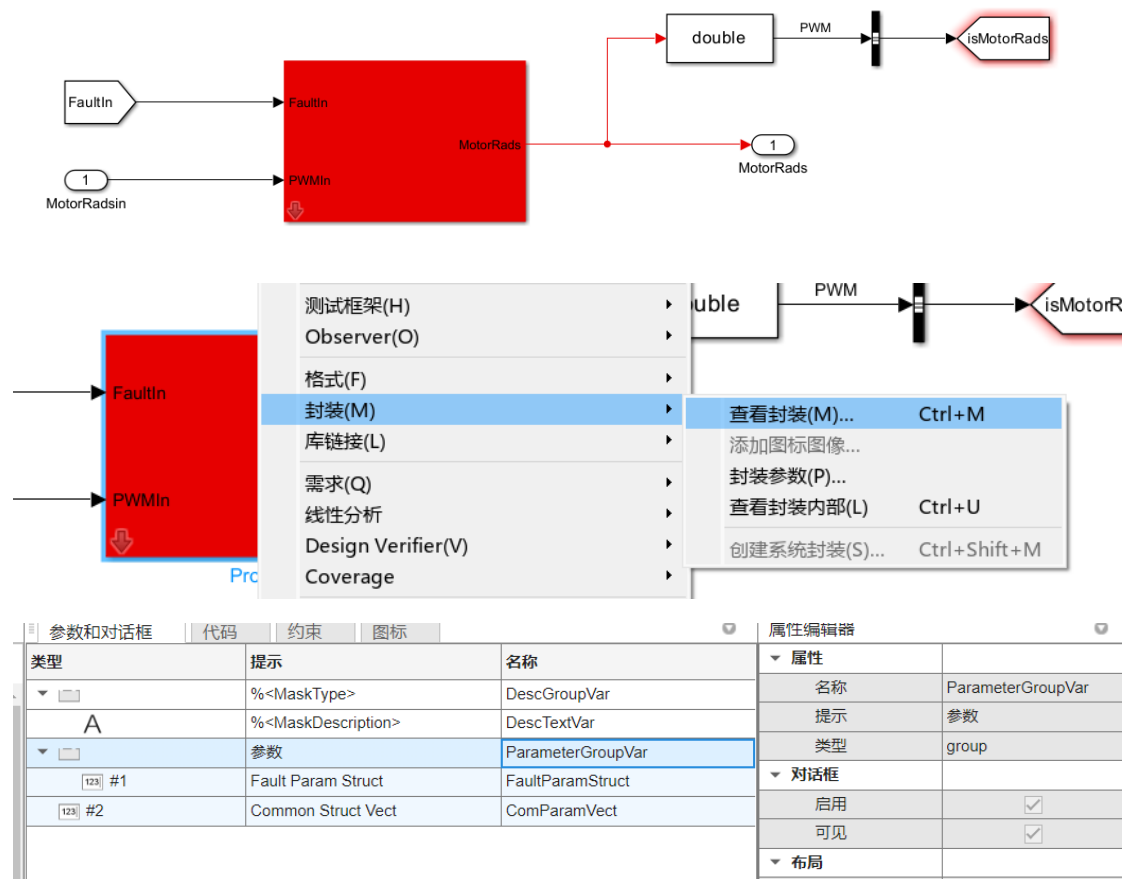
 module_param_api
 MotorFault 1x1 struct
 MotorFault1 1x1 struct
 MotorFaultTemp 1x1 struct
 **PropFault** 1x1 struct
 SensorFault 1x1 struct
 WindFault 1x1 struct

字段 ▲	值	
 FaultID	123451	
 PropNum	4	

我们可以通过工作区对封装模块引用参数进行查看。
双击打开初始化脚本，单机运行。打开例程文件，我们可以对封装模块进行查看。



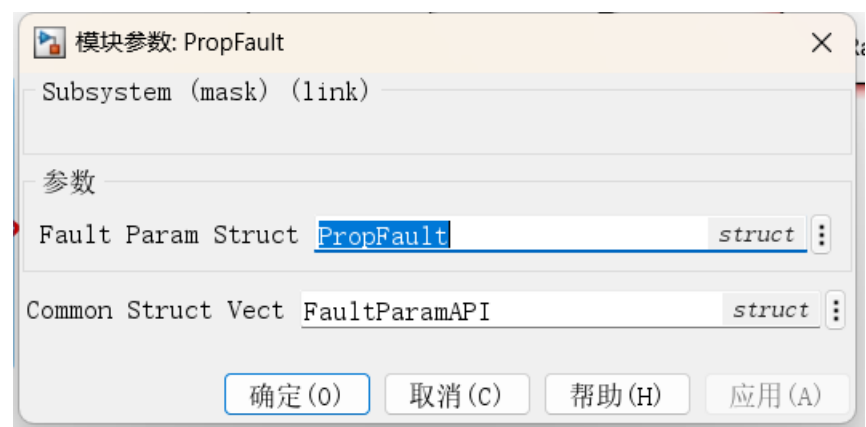
此模块中参数由工作区导入。我们可以右键点击，查看封装。

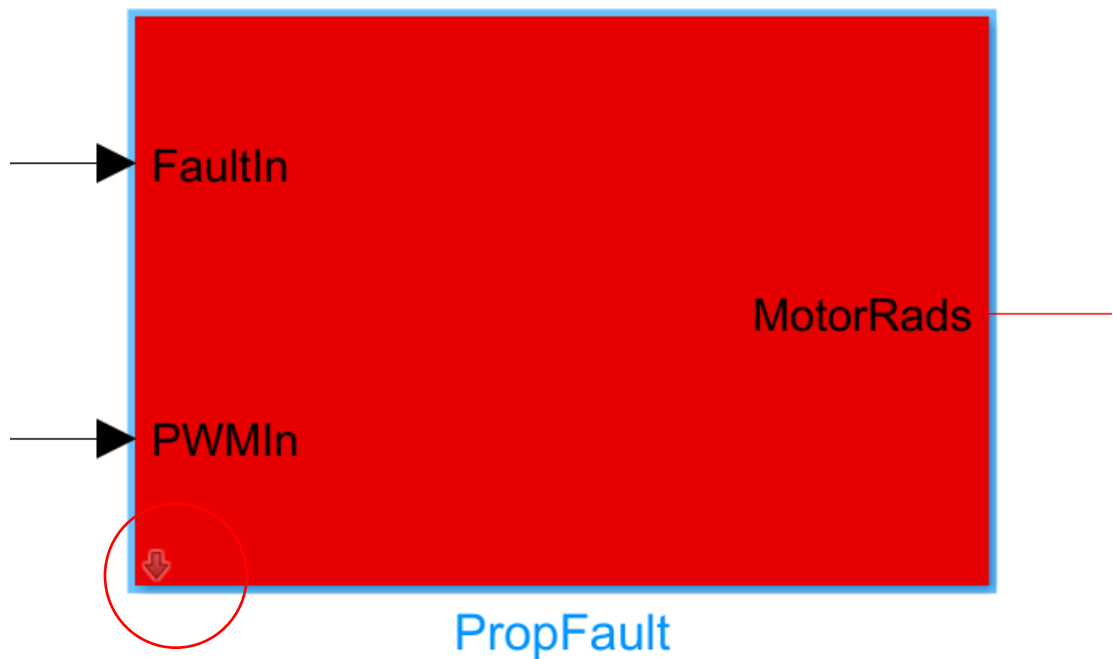


我们可以看到，此处应用的值跟上面相同，导入了工作区中的值，只是对其进行了命名。

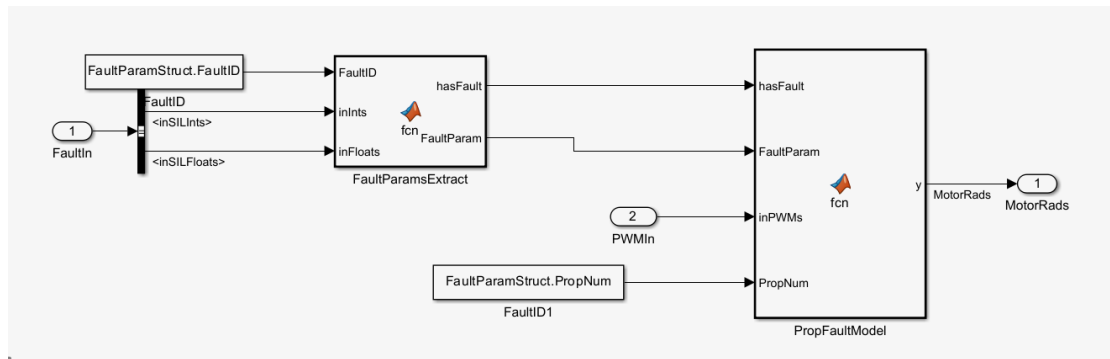
之后，我们进入封装内部，进行观察。

双击可查看模块参数。





点击红圈中的箭头，我们便可以进入封装。

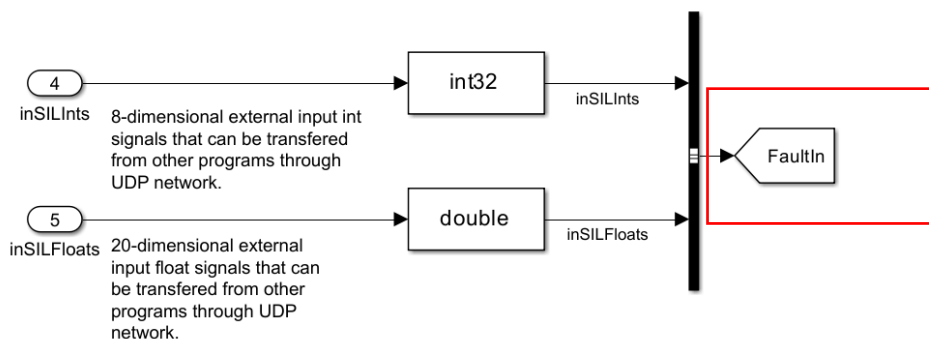


我们可以看到螺旋桨故障注入模块，双击进入 **FaultParamsExtract** 模块，我们可以看到其故障注入的方法。

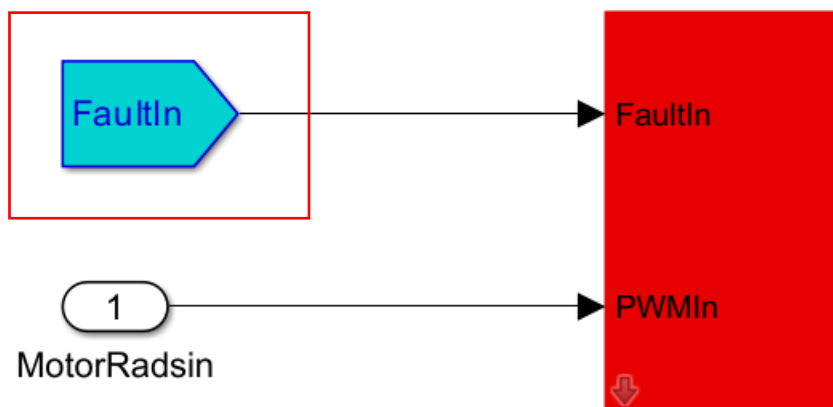
4.2.3 故障消息的订阅与触发

Goto 模块 FaultIn 标记-发布故障消息

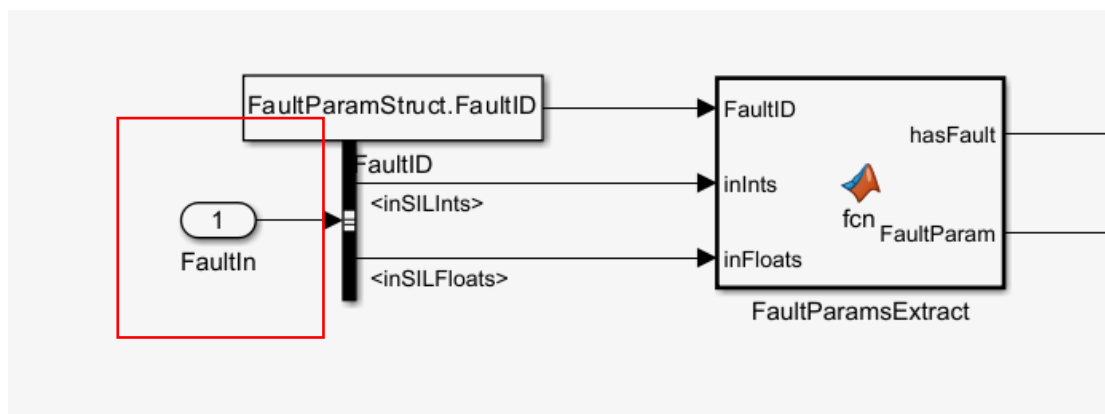
From 模块 FaultIn 标记-订阅故障消息



同样，例程文件通过从工作区读取到所需的参数后，通过 **GOTO** 模块进行故障参数的发送，传输到螺旋桨故障模块中。



之后，通过同样的方式，将故障参数传输到封装模块之中。



FaultParamsExtract 自定义模块-故障触发与处理

```
1 function [hasFault, FaultParam] = fcn(FaultID,inInts,inFloats)
2 persistent hFault;
3 persistent fParam;
4
5 if isempty(hFault)
6     hFault=false;
7 end
8 if isempty(fParam)
9     fParam=zeros(20,1);
10 end
11
12 hFaultTmp=false;
13 fParamTmp=zeros(20,1);
14 j=1;
15 for i=1:8
16     if inInts(i) == FaultID
17         hFaultTmp=true;
18         fParamTmp(2*j-1)=inFloats(2*i-1);
19         fParamTmp(2*j)=inFloats(2*i);
20         j=j+1;
21     end
22 end
23 if hFaultTmp
24     hFault=hFaultTmp;
25     fParamTmp(17:20) = inFloats(17:20);
26     fParam=fParamTmp;
27 end
28
29 hasFault=hFault;
30 FaultParam=fParam;
31
```

4.3 电池故障（BatteryFault）模块

4.3.1 电池故障注入 ID 和参数配置

字段 ▲	值
PowOffFaultID	123452
LowVoltageFaultID	123453
LowCapacityFaultID	123454

BatteryFault.PowOffFaultID = 123452;

电池失效故障 ID 为 123452。

BatteryFault.LowVoltageFaultID = 123453;

低电压故障 ID 为 123453。

BatteryFault.LowCapacityFaultID = 123454;

低电量故障 ID 为 123454。

4.3.2 故障模块中故障参数（FaultParamAPI）的封装传递

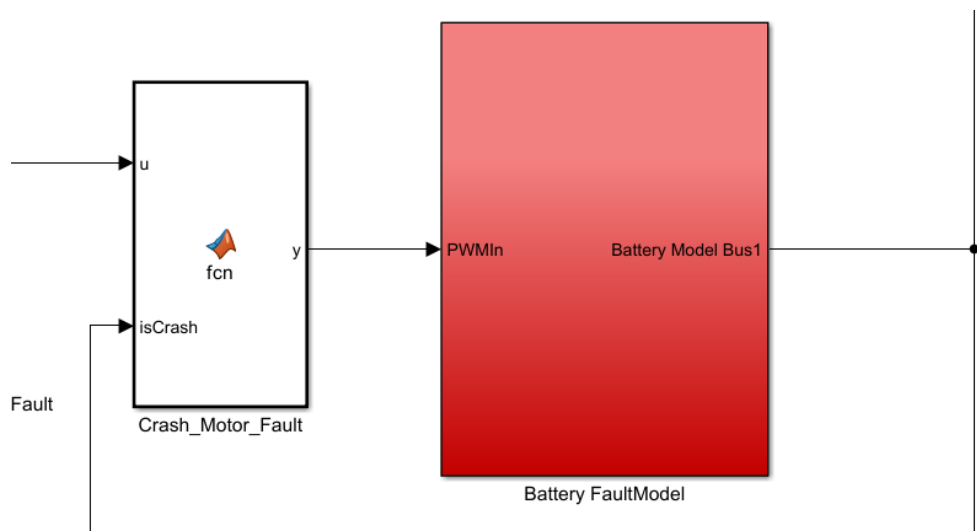
工作区

名称	值
BatteryFault	1x1 struct
FaultParamAPI	1x1 struct
HILGPS	1x1 Bus
LoadFault	1x1 struct
MavlinkGDS	1x1 Bus

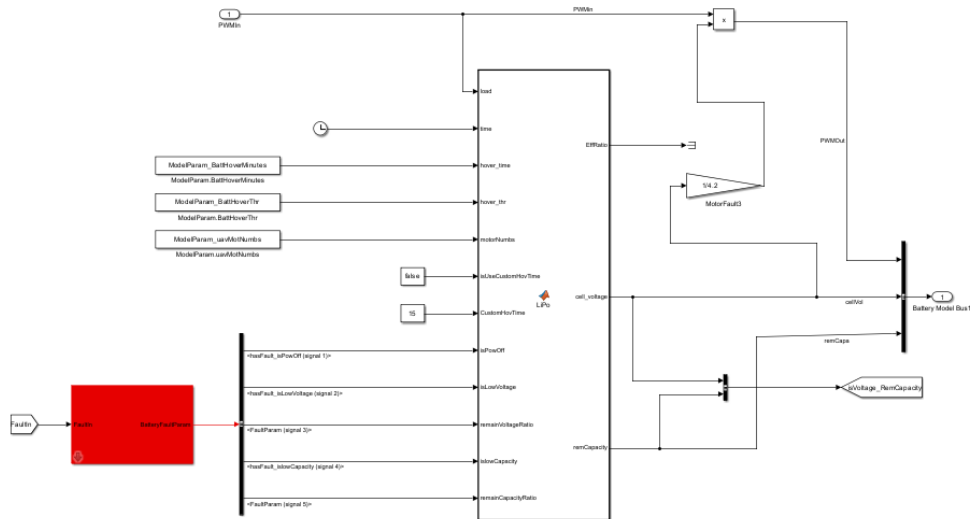
1x1 struct 包含 3 个字段

字段	值
PowOffFaultID	123452
LowVoltageFaultID	123453
LowCapacityFaultID	123454

我们可以通过工作区对封装模块引用参数进行查看。
双击打开初始化脚本，单机运行。打开例程文件，我们可以对封装模块进行查看。



此模块中参数由工作区导入。我们可以右键点击，查看封装。

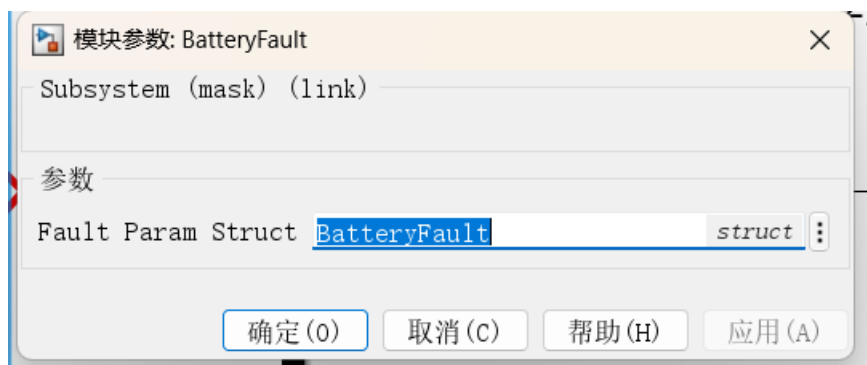


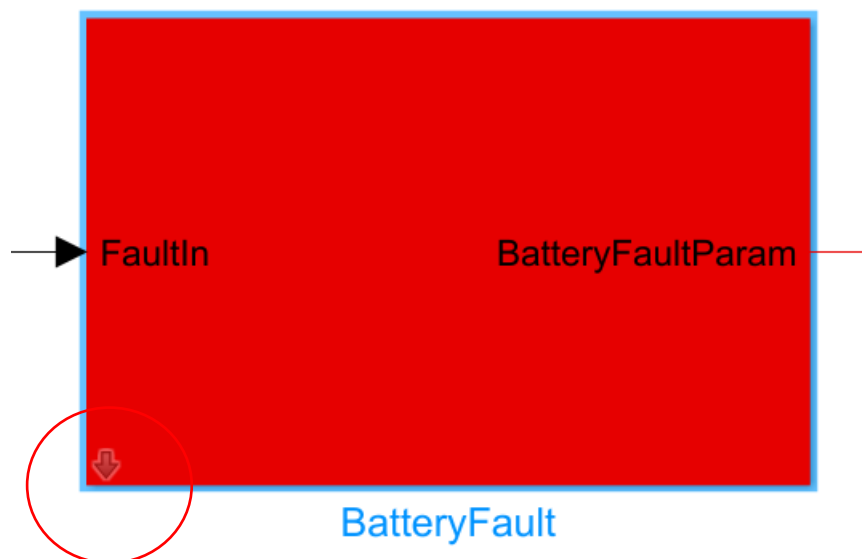
参数和对话框	代码	约束	图标	属性编辑器
类型	提示	名称		属性
▼	%<MaskType>	DescGroupVar		名称
A	%<MaskDescription>	DescTextVar		提示
▼	参数	ParameterGroupVar		类型
123 #1	Fault Param Struct	FaultParamStruct		对话框
				启用
				可见

我们可以看到,此处应用的值跟上面相同,导入了工作区中的值,只是对其进行了命名。

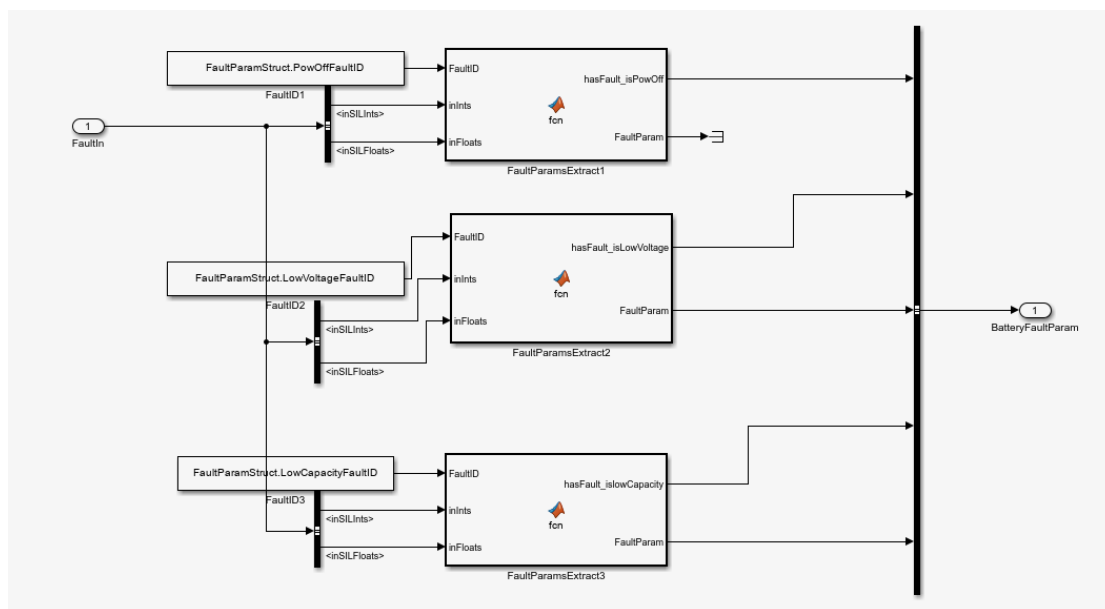
之后,我们进入封装内部,进行观察。

双击可查看模块参数。





点击红圈中的箭头，我们便可以进入封装。

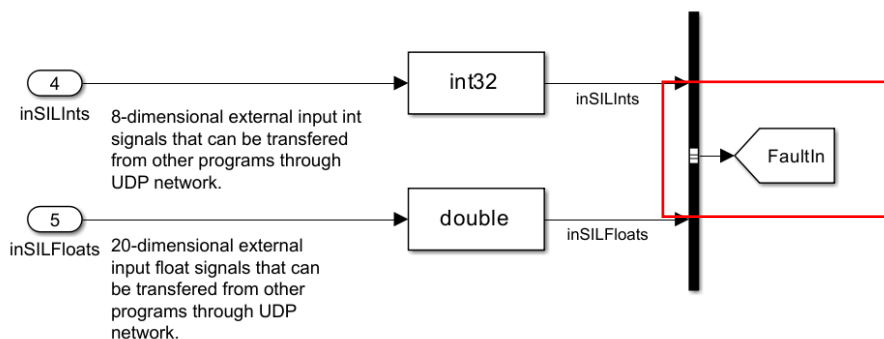


我们可以看到三个电池故障注入模块，这是因为有三种故障可以选择注入，根据注入故障 ID 的不同，根据内部的代码逻辑，会选择正确的模块进行故障注入。

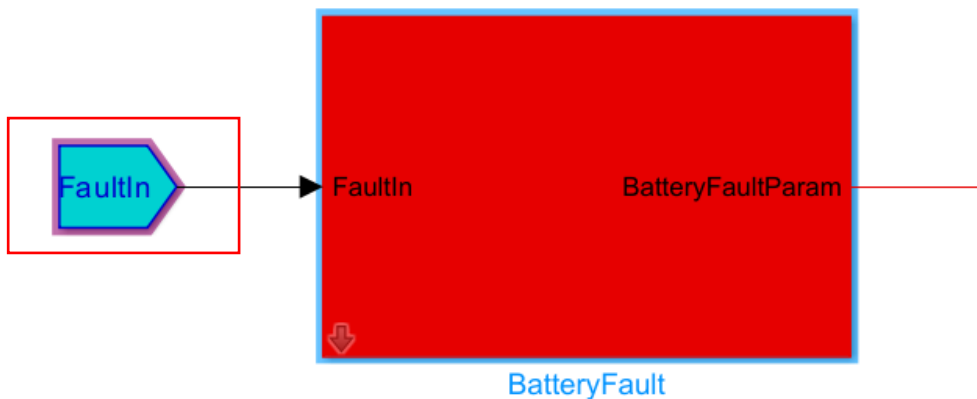
4.3.3 故障消息的订阅与触发

Goto 模块 FaultIn 标记-发布故障消息

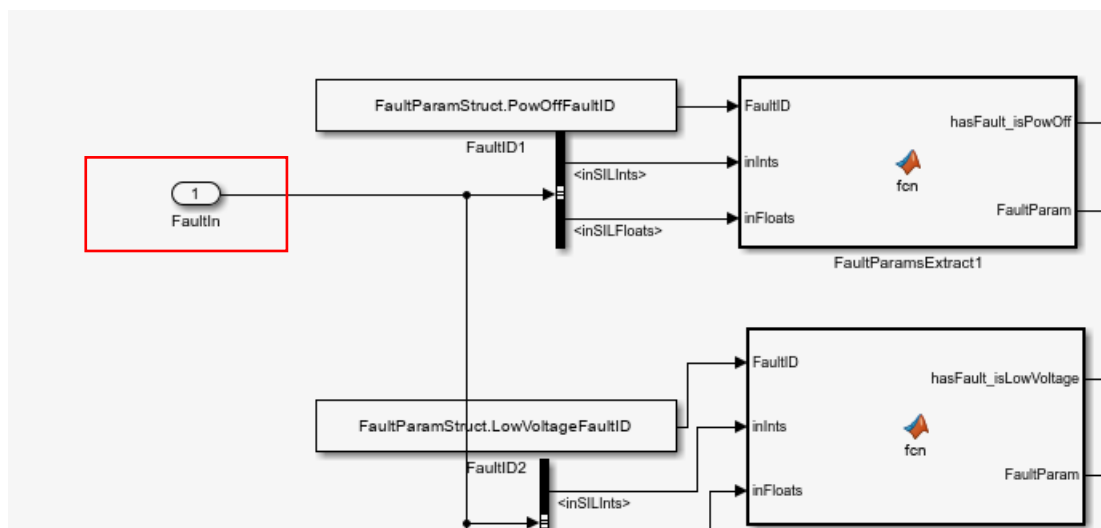
From 模块 FaultIn 标记-订阅故障消息



同样，例程文件通过从工作区读取到所需的参数后，通过 GOTO 模块进行故障参数的发送，传输到电池故障模块中。



之后，通过同样的方式，将故障参数传输到封装模块之中。



FaultParamsExtract 自定义模块-故障触发与处理

```

if isempty(hFault)
    hFault=false;
end
if isempty(fParam)
    fParam=zeros(20,1);
end

hFaultTmp=false;
fParamTmp=zeros(20,1);
j=1;
for i=1:8
    if inInts(i) == FaultID
        hFaultTmp=true;
        fParamTmp(2*j-1)=inFloats(2*i-1);
        fParamTmp(2*j)=inFloats(2*i);
        j=j+1;
    end
end

```

4.4 负载故障（LoadFault）模块

4.4.1 负载故障注入 ID 和参数配置

字段 ▲	值
LoadFallFaultID	123455
LoadShiftFaultID	123456
LoadLeakFaultID	123457

LoadFault.LoadFallFaultID = 123455;

负载故障故障 ID 为 123455。

LoadFault.LoadShiftFaultID = 123456;

负载漂移故障故障 ID 为 123456。

LoadFault.LoadLeakFaultID = 123457;

负载泄露故障故障 ID 为 123457。

4.4.2 故障模块中故障参数（FaultParamAPI）的封装传递

此处故障参数的封装传递可以参考上文 [4.1.2 故障模块中故障参数（FaultParamAPI）的封装传递](#)

4.4.3 故障消息的订阅与触发

Goto 模块 FaultIn 标记-发布故障消息







From 模块 FaultIn 标记-订阅故障消息





FaultParamsExtract 自定义模块-故障触发与处理

此处可参考上文中 [4.1.3 故障消息的订阅与触发](#)

4.5 环境风故障（WindFault）模块

4.5.1 环境风故障注入 ID 和参数配置

	MotorFault	1x1 struct
	MotorFault1	1x1 struct
	MotorFaultTemp	1x1 struct
	PropFault	1x1 struct
	SensorFault	1x1 struct
	WindFault	1x1 struct

字段 ▲	值	
	ConstWindFaultID	123458
	GustWindFaultID	123459
	TurbWindFaultID	123540
	SheerWindFaultID	123541

WindFault.ConstWindFaultID = 123458;

常风故障故障 ID 为 123458。

WindFault.GustWindFaultID = 123459;

阵风故障故障 ID 为 123459。

WindFault.TurbWindFaultID = 123540;

紊流风故障故障 ID 为 123540。

WindFault.SheerWindFaultID = 123541;

切向风故障故障 ID 为 123541。

4.5.2 故障模块中故障参数（FaultParamAPI）的封装传递

此处故障参数的封装传递可以参考上文 [4.1.2 故障模块中故障参数（FaultParamAPI）的封装传递](#)

4.5.3 故障消息的订阅与触发

Goto 模块 FaultIn 标记-发布故障消息

From 模块 FaultIn 标记-订阅故障消息

FaultParamsExtract 自定义模块-故障触发与处理

此处可参考上文中 [4.1.3 故障消息的订阅与触发](#)

4.6 传感器故障（SensorFault）模块

4.6.1 传感器故障注入 ID 和参数配置

MotorFault	1x1 struct
MotorFault1	1x1 struct
MotorFaultTemp	1x1 struct
PropFault	1x1 struct
SensorFault	1x1 struct
WindFault	1x1 struct

字段	值
AccNoiseFaultID	123542
AccBaisFaultID	1235421
GyroNoiseFaultID	123543
GyroBaisFaultID	1235431
MagNoiseFaultID	123544
MagBaisFaultID	1235441
BaroNoiseFaultID	123545
GPSNoiseFaultID	123546

加速度计 **SensorFault.AccNoiseFaultID = 123542;**

加速度计噪声干扰故障 ID 为 123542。

陀螺仪 **SensorFault.GyroNoiseFaultID = 123543;**

陀螺仪噪声干扰故障 ID 为 123543。

磁罗盘 **SensorFault.MagNoiseFaultID = 123544;**

磁力计噪声干扰故障 ID 为 123544。

气压计 **SensorFault.BaroNoiseFaultID = 123545;**

气压计噪声干扰故障 ID 为 123545。

GPS **SensorFault.GPSNoiseFaultID = 123546;**

GPS 故障故障 ID 为 123546。

4.6.2 故障模块中故障参数（FaultParamAPI）的封装传递

此处故障参数的封装传递可以参考上文 [4.1.2 故障模块中故障参数（FaultParamAPI）的封装传递](#)

4.6.3 故障消息的订阅与触发

Goto 模块 FaultIn 标记-发布故障消息

From 模块 FaultIn 标记-订阅故障消息

FaultParamsExtract 自定义模块-故障触发与处理

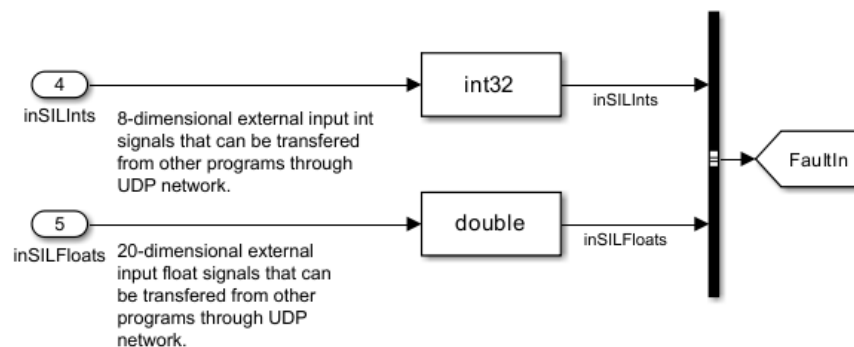
此处可参考上文中 [4.1.3 故障消息的订阅与触发](#)

5. simulink 模型消息接口

5.1 CopterSim 输入输出接口

5.1.1 消息输出接口

通过 udp 模块（30101）端口输出 DLL 模型消息，使用 32 维数组接收



5.1.2 消息输入接口

通过 `udp` 模块（30100）端口接收外部输入消息

```
1  # import required libraries
2  import time
3  import math
4  import numpy as np
5  # import RflySim APIs
6  import PX4MavCtrlV4 as PX4MavCtrl
7
8  # Create MAVLink control API instance
9  mav1 = PX4MavCtrl.PX4MavCtrl(1)
10 # mav2 = PX4MavCtrl.PX4MavCtrl(2)
11 # mav2 = PX4MavCtrl.PX4MavCtrl(3)
12 # mavN --> 20100 + (N-1)*2
13
14 # Init MAVLink data receiving loop
15 mav1.InitMavLoop()
16 #mav2.InitMavLoop(), ...
17
18 time.sleep(0.5)
19 mav1.InitTrueDataLoop()
20 time.sleep(0.5)
21
22 mav1.initOffboard()
23
24 lastTime = time.time()
25 startTime = time.time()
26 # time interval of the timer
27 timeInterval = 1/30.0 #here is 0.0333s (30Hz)
28
```

5.2 Rflysim 接口协议文件 Python-PX4MavCtrlV4.py

5.2.1 基于 `udp` 故障注入接口

```
auditInjectAPItest.py > ...
# import required libraries
import time
import math
import numpy as np
# import RflySim APIs
import PX4MavCtrlV4 as PX4MavCtrl

# Create MAVLink control API instance
mav1 = PX4MavCtrl.PX4MavCtrl(1)
# mav2 = PX4MavCtrl.PX4MavCtrl(2)
# mav2 = PX4MavCtrl.PX4MavCtrl(3)
# mavN --> 20100 + (N-1)*2
```

```
查看(V) 转到(G) 运行(R) ... 64_faultInjectAPItest_py
FaultInjectAPITest.py PX4MavCtrlV4.py x
C:\> PX4PSP > RflySimAPIs > RflySimSDK > ctrl > PX4MavCtrlV4.py > PX4MavCtrl > __init__
149 self.port = 20100+self.CopterID*2-2
150
151
152 # UDP模式解析
153 if (Com=='udp' or Com=='UDP' or Com=='Udp') and ID>10000: # 如果是UDP通信模式
154     # 兼容旧版协议, 如果ID是20100等端口输入, 则自动计算CopterID
155     self.port=ID
156     self.CopterID = int((ID-20100)/2)+1
157
```

5.2.2 基于串口故障注入接口

```
import time
import math
import sys

import PX4MavCtrlV4 as PX4MavCtrl

# For hardware connection
#Windows use format PX4MavCtrl(ID,ip,'COM3',baud) for Pixhawk USB port connection
#Windows use format 'COM4' for Pixhawk serial port connection
#Linux use format '/dev/ttyUSB0' for USB, or '/dev/ttyAMA0' for Serial port (RaspberryPi)
# PX4MavCtrl(1,'127.0.0.1','COM3',57600)
# PX4MavCtrl(1,'127.0.0.1','/dev/ttyS0',57600)
# constructor function
mav = PX4MavCtrl.PX4MavCtrl(Com = 'COM10:57600')

#mav.InitMavLoop(UDPMODE), where UDPMODE=0,1,2,3,4

...

# constructor function
def __init__(self, ID=1, ip='127.0.0.1',Com='udp',port=0, simulinkDLL=False):
    global isEnabledRdis
    self.isInPointMode = False
    self.isCom = False
    self.Com = Com
    self.baud = 115200
    self.isRealFly = 0
    self.ip = ip
    self.isRedis = False
    self.simulinkDLL = simulinkDLL

    # 这里是为了兼容之前的PX4MavCtrl('COM3:115200')串口协议, 将来会取消

    self.ComName = 'COM3' # 默认值: 串口3

    if Com[0:3]=='COM' or Com[0:3]=='com' or Com[0:3]=='Com' or Com[0:3]=='/de': # 如果是串口连接方式
        self.isCom = True # 串口通信模式
        strlist = Com.split(':')
        if port==0: # 默认值57600
            self.baud = 57600
        if(len(strlist) >= 2): # 串口号:波特率 协议解析, 为了兼容旧接口
            if strlist[1].isdigit():
                self.baud = int(strlist[1])
            self.ComName = strlist[0] # 串口名字
```

串口故障注入中所使用的波特率有两种, 为 115200 以及 57600, 如果设置为 0, 即默认 57600。

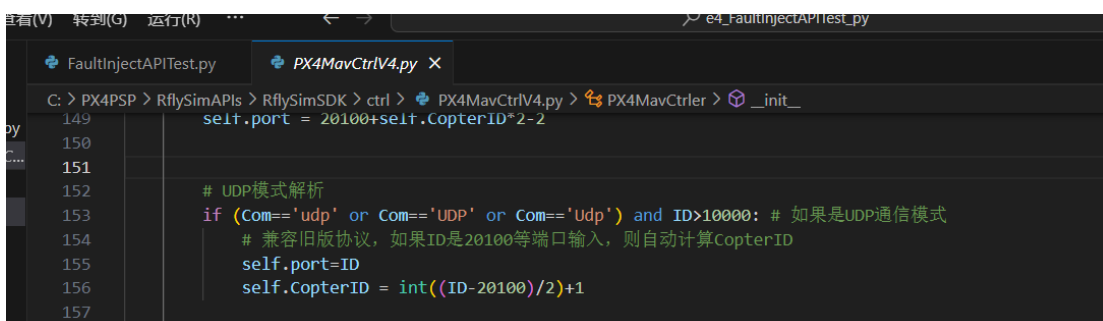
The data will also sent to 30100 + (2*i-2) series port, which can be listened through Simulink or Python. The struct is

5.4 Python-飞控硬件信息交互输入输出接口

5.4.1 基于串口连接的串口传输

```
faultInjectAPItest.py > ...
# import required libraries
import time
import math
import numpy as np
# import RflySim APIs
import PX4MavCtrlV4 as PX4MavCtrl

# Create MAVLink control API instance
mav1 = PX4MavCtrl.PX4MavCtrlr(1)
# mav2 = PX4MavCtrl.PX4MavCtrlr(2)
# mav2 = PX4MavCtrl.PX4MavCtrlr(3)
# mavN --> 20100 + (N-1)*2
```



```
149 self.port = 20100+self.CopterID*2-2
150
151
152 # UDP模式解析
153 if (Com=='udp' or Com=='UDP' or Com=='Udp') and ID>10000: # 如果是UDP通信模式
154     # 兼容旧版协议, 如果ID是20100等端口输入, 则自动计算CopterID
155     self.port=ID
156     self.CopterID = int((ID-20100)/2)+1
157
```

5.4.2 基于 usb 连接的 udp 传输

```
import time
import math
import sys

import PX4MavCtrlV4 as PX4MavCtrl

# For hardware connection
#Windows use format PX4MavCtrlr(ID,ip,'COM3',baud) for Pixhawk USB port connection
#Windows use format 'COM4' for Pixhawk serial port connection
#Linux use format '/dev/ttyUSB0' for USB, or '/dev/ttyAMA0' for Serial port (RaspberryPi)
# PX4MavCtrlr(1,'127.0.0.1','COM3',57600)
# PX4MavCtrlr(1,'127.0.0.1','/dev/ttyS0',57600)
# constructor function
mav = PX4MavCtrl.PX4MavCtrlr(Com = 'COM10:57600')

#mav.InitMavLoop(UDPMODE), where UDPMODE=0,1,2,3,4
```

```

"""
# constructor function
def __init__(self, ID=1, ip='127.0.0.1', Com='udp', port=0, simulinkDLL=False):
    global isEnabledRdis
    self.isInPointMode = False
    self.isCom = False
    self.Com = Com
    self.baud = 115200
    self.isRealFly = 0
    self.ip = ip
    self.isRedis = False
    self.simulinkDLL = simulinkDLL

```

这里是为了兼容之前的PX4MavCtrlr('COM3:115200')串口协议，将来会取消

self.ComName = 'COM3' # 默认值：串口名

```

if Com[0:3]=='COM' or Com[0:3]=='com' or Com[0:3]=='Com' or Com[0:3]=='/de': # 如果是串口连接方式
    self.isCom = True # 串口通信模式
    strlist = Com.split(':')
    if port==0: # 默认值57600
        self.baud = 57600
    if (len(strlist) >= 2): # 串口号:波特率 协议解析，为了兼容旧接口
        if strlist[1].isdigit():
            self.baud = int(strlist[1])
    self.ComName = strlist[0] # 串口名字

```

6. 自动化故障注入平台的搭建与使用

6.1 平台配置文件

6.1.1 测试用例配置 db.json

```
{
  "faultcase": [
    {
      "CaseID": "RT01",
      "Subsystem": "Power",
      "Component": "Motor",
      "FaultID": "123450",
      "FaultType": "Elevator Steering Fault",
      "FaultMode": "Decreased efficiency of actuator execution",
      "CaseDescription": "Faulty steering gear for fixed-point navigation",
      "FaultParams": "FaultParam",
      "ControlSequence": "2, 1; 1, 1, 5; 2, 3, 100, 0, 0; 1, 1, 15; 2, 6, 123540, 5, 10; 1, 1, 10",
      "DataRequired": "Simulator ground truth data (pose and pose output)",
      "TestStatus": "Finished"
    }
  ],
  "testcase": "all",
  "Vision": "off"
}
```

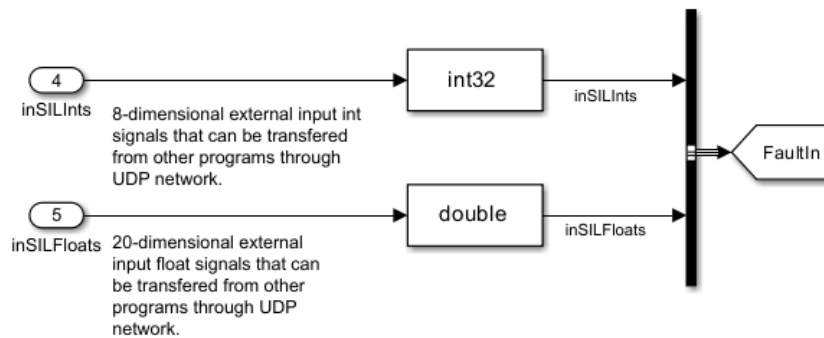
6.1.2 吊舱参数配置 Config.json

```
1 |
2 | {
3 |   "VisionSensors": [
4 |     {
5 |       "SeqID": 0,
6 |       "TypeID": 1,
7 |       "TargetCopter": 1,
8 |       "TargetMountType": 0,
9 |       "DataWidth": 640,
10 |      "DataHeight": 480,
11 |      "DataCheckFreq": 200,
12 |      "SendProtocol": [0, 127, 0, 0, 1, 9999, 0, 0],
13 |      "CameraFOV": 90,
14 |      "SensorPosXYZ": [7, 0, -2.8],
15 |      "SensorAngEuler": [0, 0, 0],
16 |      "otherParams": [0, 0, 0, 0, 0, 0, 0, 0]
17 |    }
18 |   ]
19 | }
```


6.2 Rflsim 接口协议文件 PX4MavCtrlV4.py

6.2.1 故障注入协议类 PX4SILIntFloat

```
# //输出到CopterSim DLL模型的SILints和SILFloats数据
# struct PX4SILIntFloat{
#     int checksum;//1234567897
#     int CopterID;
#     int inSILInts[8];
#     float inSILFloats[20];
# };
#struct.pack 10i20f
```



此处是对故障进行注入，其注入分为 8 位的故障 ID 注入以及 20 位的故障参数注入。

6.2.2 解锁/未解锁接口 SendMavArm

```
# send MAVLink command to Pixhawk to Arm/Disarm the drone
def SendMavArm(self, isArm=0):
    """ Send command to PX4 to arm or disarm the drone
    """
    if self.UDPMode>1.5:
        if (isArm):
            self.SendMavCmdLong(mavlink2.MAV_CMD_COMPONENT_ARM_DISARM, 1)
        else:
            self.SendMavCmdLong(mavlink2.MAV_CMD_COMPONENT_ARM_DISARM, 0, 21196.0)
    else:
        ctrls=[isArm,0,0,0]
        self.sendUDPSimpData(9,ctrls)
```

6.2.3 无人机目标位置接口 SendPosNED

```
# send target position in earth NED frame
def SendPosNED(self,x=0,y=0,z=0,yaw=0):
    """ Send vehicle target position (m) to PX4 in the earth north-east-down (NED) frame with yaw control (rad)
    when the vehicle fly above the ground, then z < 0
    """

    self.offMode=0 # SET_POSITION_TARGET_LOCAL_NED
    self.ctrlMode=2 #地球位置控制模式
    self.EnList = [1,0,0,0,1,0]
    self.type_mask=self.TypeMask(self.EnList)
    self.coordinate_frame = mavlink2.MAV_FRAME_LOCAL_NED
    self.pos=[x,y,z]
    self.vel = [0,0,0]
    self.acc = [0, 0, 0]
    self.yawrate = 0
    self.yaw = yaw

# send target position in earth NED frame
def SendVelYawAlt(self,vel=10,yaw=6.28,alt=-100):
    """ Send vehicle target position (m) to PX4 in the earth north-east-down (NED) frame with yaw control (rad)
    when the vehicle fly above the ground, then z < 0
    """

    if abs(yaw)<0.00001:
        yaw = 6.28
    self.offMode=0 # SET_POSITION_TARGET_LOCAL_NED
    self.ctrlMode=13 #速度高度偏航控制模式
    self.type_mask=int("000111000000", 2)
    self.coordinate_frame = 1
    self.pos=[0,0,alt]
    self.vel = [yaw,vel,0]
    self.acc = [0, 0, 0]
    self.yawrate = 0
    self.yaw = yaw

# send target position in earth NED frame
def SendPosNEDNoYaw(self,x=0,y=0,z=0):
    """ Send vehicle target position (m) to PX4 in the earth north-east-down (NED) frame without yaw control (rad)
    when the vehicle fly above the ground, then z < 0
    """

    self.offMode=0 # SET_POSITION_TARGET_LOCAL_NED
    self.ctrlMode=2 #地球位置控制模式
    self.EnList = [1,0,0,0,0,0]
    self.type_mask=self.TypeMask(self.EnList)
    self.coordinate_frame = mavlink2.MAV_FRAME_LOCAL_NED
    self.pos=[x,y,z]
    self.vel = [0,0,0]
    self.acc = [0, 0, 0]
    self.yawrate = 0
    self.yaw = 0
```

```

# send target position in body FRD frame
def SendPosFRD(self,x=0,y=0,z=0,yaw=0):
    """ Send vehicle targe position (m) to PX4 in the body forward-rightward-downward (FRD) frame with yaw control (rad)
    when the vehicle fly above the ground, then z < 0
    """
    self.offMode=0 # SET_POSITION_TARGET_LOCAL_NED
    self.ctrlMode=3 #机体位置控制模式
    self.EnList = [1,0,0,0,1,0]
    self.type_mask=self.TypeMask(self.EnList)
    self.coordinate_frame = mavlink2.MAV_FRAME_BODY_NED
    self.pos=[x,y,z]
    self.vel = [0,0,0]
    self.acc = [0, 0, 0]
    self.yawrate = 0
    self.yaw = yaw

# send target position in body FRD frame
def SendPosFRDNoYaw(self,x=0,y=0,z=0):
    """ Send vehicle targe position (m) to PX4 in the body forward-rightward-downward (FRD) frame without yaw control (rad)
    when the vehicle fly above the ground, then z < 0
    """
    self.offMode=0 # SET_POSITION_TARGET_LOCAL_NED
    self.ctrlMode=3 #机体位置控制模式
    self.EnList = [1,0,0,0,0,0]
    self.type_mask=self.TypeMask(self.EnList)
    self.coordinate_frame = mavlink2.MAV_FRAME_BODY_NED
    self.pos=[x,y,z]
    self.vel = [0,0,0]
    self.acc = [0, 0, 0]
    self.yawrate = 0
    self.yaw = 0

```

```

def SendPosNEDExt(self,x=0,y=0,z=0,mode=3,isNED=True):
    """ Send vehicle targe position (m) to PX4
    when the vehicle fly above the ground, then z < 0
    """
    self.offMode=0 # SET_POSITION_TARGET_LOCAL_NED
    self.EnList = [1,0,0,0,1,0]
    self.type_mask=self.TypeMask(self.EnList)
    if mode==0:
        # Gliding setpoint
        self.type_mask=int(292) # only for fixed Wing
    elif mode==1:
        # Takeoff setpoints
        self.type_mask=int(4096) # only for fixed Wing
    elif mode==2:
        # Land setpoints
        self.type_mask=int(8192) # only for fixed Wing
    elif mode==3:
        # Loiter setpoints
        # for Rover: Loiter setpoint (vehicle stops when close enough to setpoint).
        # for fixed wing: Loiter setpoint (fly a circle centred on setpoint).
        self.type_mask=int(12288)
    elif mode==4:
        # Idle setpoint
        # only for fixed wing
        # Idle setpoint (zero throttle, zero roll / pitch).
        self.type_mask=int(16384)
    if isNED:
        self.coordinate_frame = mavlink2.MAV_FRAME_LOCAL_NED
    else:
        self.coordinate_frame = mavlink2.MAV_FRAME_BODY_NED
    self.pos=[x,y,z]
    self.vel = [0,0,0]
    self.acc = [0, 0, 0]
    self.yawrate = 0
    self.yaw = 0

```

6.2.4 无人机飞行速度接口 FlyVel

```
self.uavVelNED = [0, 0, 0] # Estimated local velocity from PX4 in NED
frame
        self.trueVelNED = [0, 0, 0] # True simulated speed from
CopterSim's DLL model in NED frame
```

uavVel 为例程运行合成的飞行速度，trueVel 为没有经过合成，真实的飞行速度。

6.3 率模可靠度的安全评估算法 Health_ass.py

6.3.1 率模健康度的计算接口 Rate_Model_rank

6.3.2 记录差值序列数据接口 fly_log_record_allan

6.4 平台指令控制接口 command.py

6.4.1 数据库故障命令协议说明

6.4.2 未解锁命令接口 DisArm(self)

6.4.3 解锁命令接口 Arm(self)

6.4.4 飞行目标接口 FlyPos(self,pos)

6.4.5 飞行速度接口 FlyVel(self,vel)

6.4.6 着陆接口 Land(self):

6.4.7 故障注入参数接口 FaultInject(self,param)

6.5 吊舱视觉 API VisionCaptureApi.py

6.5.1 开始视觉图像捕捉 startImgCap

```
def startImgCap(self, isRemoteSend=False):
    """start loop to receive image from UE4,
    isRemoteSend=true will forward image from memory to UDP port
    """
    self.isRemoteSend = isRemoteSend
    global isEnabledRosTrans
    memList = []
    udpList = []
    if isEnabledRosTrans:
        self.time_record = np.zeros(len(self.VisSensor))
        if is_use_ros1:
```

```

        self.rostime = np.ndarray(len(self.time_record),
dtype=rospy.Time)
    else:
        self.rostime = np.ndarray(len(self.time_record),
dtype=rclpy.time.Time)

    for i in range(len(self.VisSensor)):
        self.Img = self.Img + [0]
        self.Img_lock = self.Img_lock + [
            threading.Lock()
        ] # 每个传感器都是一个独立的线程，应时使用独立的锁
        self.ImgData = self.ImgData + [0]
        self.hasData = self.hasData + [False]
        self.timeStmp = self.timeStmp + [0]
        self.imgStmp = self.imgStmp + [0]

        TarCopt = self.VisSensor[i].TargetCopter
        starTime=0
        for j in range(len(self.RflyTimeVect)):
            if self.RflyTimeVect[j].copterID == TarCopt:
                if isEnabledRosTrans:
                    starTime=self.RflyTimeVect[j].rosStartTimeStmp
                else:
                    starTime=self.RflyTimeVect[j].pyStartTimeStmp
                print('Got start time for SeqID
#',self.VisSensor[i].SeqID)
            self.rflyStartStmp = self.rflyStartStmp + [starTime]
        IP = (
            str(self.VisSensor[i].SendProtocol[1])
            + "."
            + str(self.VisSensor[i].SendProtocol[2])
            + "."
            + str(self.VisSensor[i].SendProtocol[3])
            + "."
            + str(self.VisSensor[i].SendProtocol[4])
        )
        if IP == "0.0.0.0":
            IP = "127.0.0.1"
        if self.RemotSendIP != "":
            IP = self.RemotSendIP
        self.IpList = self.IpList + [IP]
        self.portList = self.portList +
[self.VisSensor[i].SendProtocol[5]]
        if self.VisSensor[i].SendProtocol[0] == 0:

```

```

        memList = memList + [i]
    else:
        udpList = udpList + [i]

    if len(memList) > 0:
        self.t_memRec = threading.Thread(target=self.img_mem_thrd,
args=(memList,))
        self.t_memRec.start()

    if len(udpList) > 0:
        # print('Enter UDP capture')
        for i in range(len(udpList)):
            udp = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
            udp.setsockopt(socket.SOL_SOCKET, socket.SO_RCVBUF,
60000 * 100)
            udp.bind(("0.0.0.0", self.portList[udpList[i]]))
            typeId = self.VisSensor[udpList[i]].TypeID
            t_udpRec = threading.Thread(
                target=self.img_udp_thrdNew,
                args=(
                    udp,
                    udpList[i],
                    typeId,
                ),
            )
            t_udpRec.start()

```

6.5.2 更新视觉图像 sendUpdateUEImage

```

def sendUpdateUEImage(self, vs=VisionSensorReq(), windID=0,
IP="127.0.0.1"):
    if not isinstance(vs, VisionSensorReq):
        raise Exception("Wrong data input to addVisSensor()")
    intValue = [
        vs.checksum,
        vs.SeqID,
        vs.TypeID,
        vs.TargetCopter,
        vs.TargetMountType,
        vs.DataWidth,
        vs.DataHeight,
        vs.DataCheckFreq,
    ] + vs.SendProtocol
    if self.isNewJson: # 使用新版协议发送

```

```

        floValue = [vs.CameraFOV] + vs.SensorPosXYZ +
[vs.EulerOrQuat] + vs.SensorAngEular + vs.SensorAngQuat +
vs.otherParams
        buf = struct.pack("16H28f", *intValue, *floValue)
    else: # 使用旧版协议发送
        floValue = [vs.CameraFOV] + vs.SensorPosXYZ +
vs.SensorAngEular + vs.otherParams[0:8]
        buf = struct.pack("16H15f", *intValue, *floValue)

    self.udp_socket.sendto(buf, (IP, 20010 + windID))
    if self.RemotSendIP != "" and self.RemotSendIP != "127.0.0.1":
        self.udp_socket.sendto(buf, (self.RemotSendIP, 20010 +
windID))

```

6.5.3 吊舱参数配置文件加载接口 jsonLoad

```

def jsonLoad(self, ChangeMode=-1, jsonPath=""):
    """load config.json file to create camera list for image
capture,
    if ChangeMode>=0, then the SendProtocol[0] will be set to
ChangeMode to change the transfer style
    """
    #print(sys.path[0])
    if os.path.isabs(jsonPath):
        print('Json use absolute path mode')
    else:
        print('Json use relative path mode')
        if len(jsonPath) == 0:
            jsonPath = sys.path[0] + "/Config.json"
        else:
            jsonPath = sys.path[0] + "/" + jsonPath

    print("jsonPath=", jsonPath)

    if not os.path.exists(jsonPath):
        print("The json file does not exist!")
        return False
    self.isNewJson=False
    with open(jsonPath, "r", encoding="utf-8") as f:
        jsData = json.loads(f.read())
        if len(jsData["VisionSensors"]) <= 0:
            print("No sensor data is found!")
            return False
        for i in range(len(jsData["VisionSensors"])):

```

```

        visSenStruct = VisionSensorReq()
        if isinstance(jsData["VisionSensors"][i]["SeqID"], int):
            visSenStruct.SeqID =
jsData["VisionSensors"][i]["SeqID"]
        else:
            print("Json data format is wrong!")
            continue

        if isinstance(jsData["VisionSensors"][i]["TypeID"],
int):
            visSenStruct.TypeID =
jsData["VisionSensors"][i]["TypeID"]
        else:
            print("Json data format is wrong!")
            continue

        if
isinstance(jsData["VisionSensors"][i]["TargetCopter"], int):
            visSenStruct.TargetCopter =
jsData["VisionSensors"][i][
                "TargetCopter"
            ]
        else:
            print("Json data format is wrong!")
            continue

        if
isinstance(jsData["VisionSensors"][i]["TargetMountType"], int):
            visSenStruct.TargetMountType =
jsData["VisionSensors"][i][
                "TargetMountType"
            ]
        else:
            print("Json data format is wrong!")
            continue

        if isinstance(jsData["VisionSensors"][i]["DataWidth"],
int):
            visSenStruct.DataWidth =
jsData["VisionSensors"][i]["DataWidth"]
        else:
            print("Json data format is wrong!")
            continue

```



```

        if isinstance(jsData["VisionSensors"][i]["DataHeight"],
int):
            visSenStruct.DataHeight =
jsData["VisionSensors"][i]["DataHeight"]
        else:
            print("Json data format is wrong!")
            continue

        if
isinstance(jsData["VisionSensors"][i]["DataCheckFreq"], int):
            visSenStruct.DataCheckFreq =
jsData["VisionSensors"][i][
                "DataCheckFreq"
            ]
        else:
            print("Json data format is wrong!")
            continue

        if isinstance(
            jsData["VisionSensors"][i]["CameraFOV"], float
        ) or isinstance(jsData["VisionSensors"][i]["CameraFOV"],
int):
            visSenStruct.CameraFOV =
jsData["VisionSensors"][i]["CameraFOV"]
        else:
            print("Json data format is wrong!")
            continue

        if len(jsData["VisionSensors"][i]["SendProtocol"]) == 8:
            visSenStruct.SendProtocol =
jsData["VisionSensors"][i][
                "SendProtocol"
            ]
            if ChangeMode != -1:
                # 如果是远程接收模式，那么读图这里需要配置为 UDP 接收
                visSenStruct.SendProtocol[0] = ChangeMode
            else:
                print("Json data format is wrong!")
                continue

        if len(jsData["VisionSensors"][i]["SensorPosXYZ"]) == 3:
            visSenStruct.SensorPosXYZ =
jsData["VisionSensors"][i][
                "SensorPosXYZ"
            ]

```

```

        ]
    else:
        print("Json data format is wrong!")
        continue

    isNewProt=False

    if 'EularOrQuat' in jsData["VisionSensors"][i]:
        isNewProt=True
        visSenStruct.EularOrQuat =
jsData["VisionSensors"][i][
            "EularOrQuat"
        ]
    else:
        visSenStruct.EularOrQuat=0

    if len(jsData["VisionSensors"][i]["SensorAngEular"]) ==
3:
        visSenStruct.SensorAngEular =
jsData["VisionSensors"][i][
            "SensorAngEular"
        ]
    else:
        print("Json data format is wrong!")
        continue

    if isNewProt:
        if len(jsData["VisionSensors"][i]["SensorAngQuat"])
== 4:
            visSenStruct.SensorAngQuat =
jsData["VisionSensors"][i][
                "SensorAngQuat"
            ]
        else:
            print("Json data format is wrong!")
            continue

    if isNewProt: # 新协议使用 16 维的 otherParams
        if len(jsData["VisionSensors"][i]["otherParams"]) ==
16:
            visSenStruct.otherParams =
jsData["VisionSensors"][i]["otherParams"]
        else:
            print("Json data format is wrong!")

```

```

            continue
        else:
            if len(jsData["VisionSensors"][i]["otherParams"]) ==
8:
                visSenStruct.otherParams =
jsData["VisionSensors"][i]["otherParams"]+[0]*8 # 扩展到 16 维
            else:
                print("Json data format is wrong!")
                continue
            self.VisSensor = self.VisSensor + [visSenStruct]

        if ~self.isNewJson and isNewProt:
            self.isNewJson=True

    if (len(self.VisSensor)) <= 0:
        print("No sensor is obtained.")
        return False
    print("Got", len(self.VisSensor), "vision sensors from json")

    if len(self.RflyTimeVect)==0 and ~self.tTimeStmpFlag:
        #print('Start listening CopterSim time Data')
        self.StartTimeStmplisten()
        time.sleep(2)
        self.endTimeStmplisten()
    if len(self.RflyTimeVect)>0:
        print('Got CopterSim time Data for img')
    else:
        print('No CopterSim time Data for img')

    return True

```

6.5.4 添加视觉传感器 addVisSensor

```

def addVisSensor(self, vsr=VisionSensorReq()):
    """Add a new VisionSensorReq struct to the list"""
    if isinstance(vsr, VisionSensorReq):
        self.VisSensor = self.VisSensor + [copy.deepcopy(vsr)]
    else:
        raise Exception("Wrong data input to addVisSensor()")

```

6.6 平台自动化测试 API AutoTest.py

6.6.1 自动化测试 TestcasePro()

6.6.2 控制指令接口 DoCmd(ctrlseq)

6.6.3 获取指令接口 FIDPro(cmdCID)

6.6.4 指令序列控制接口 CmdPro(seq)

6.7 数据库故障用例读写 mavdb.py

6.7.1 获取数据库游标接口 get_cursor(self)

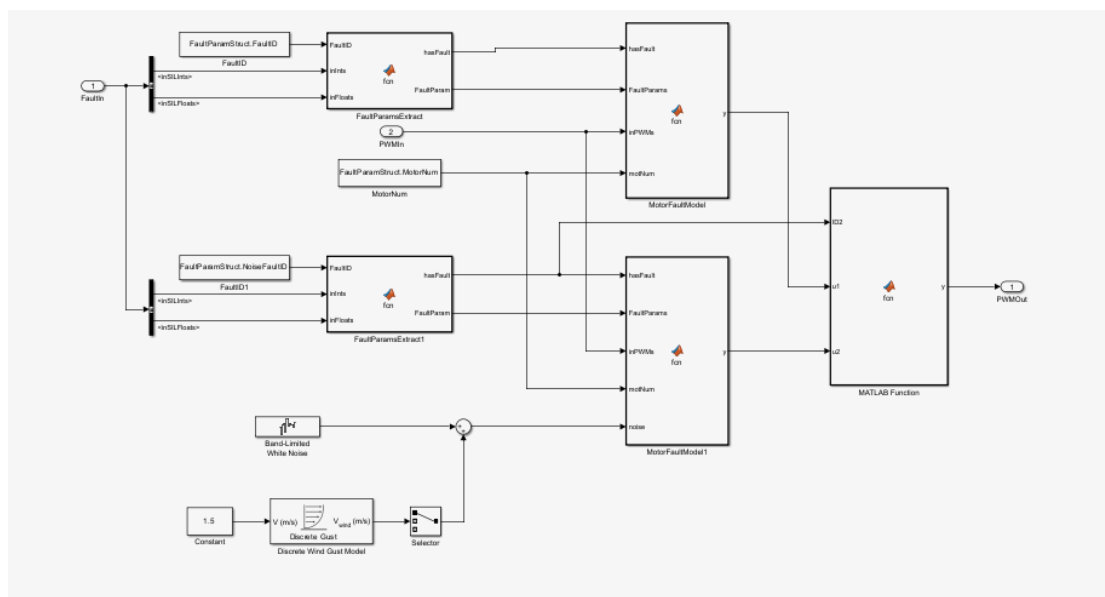
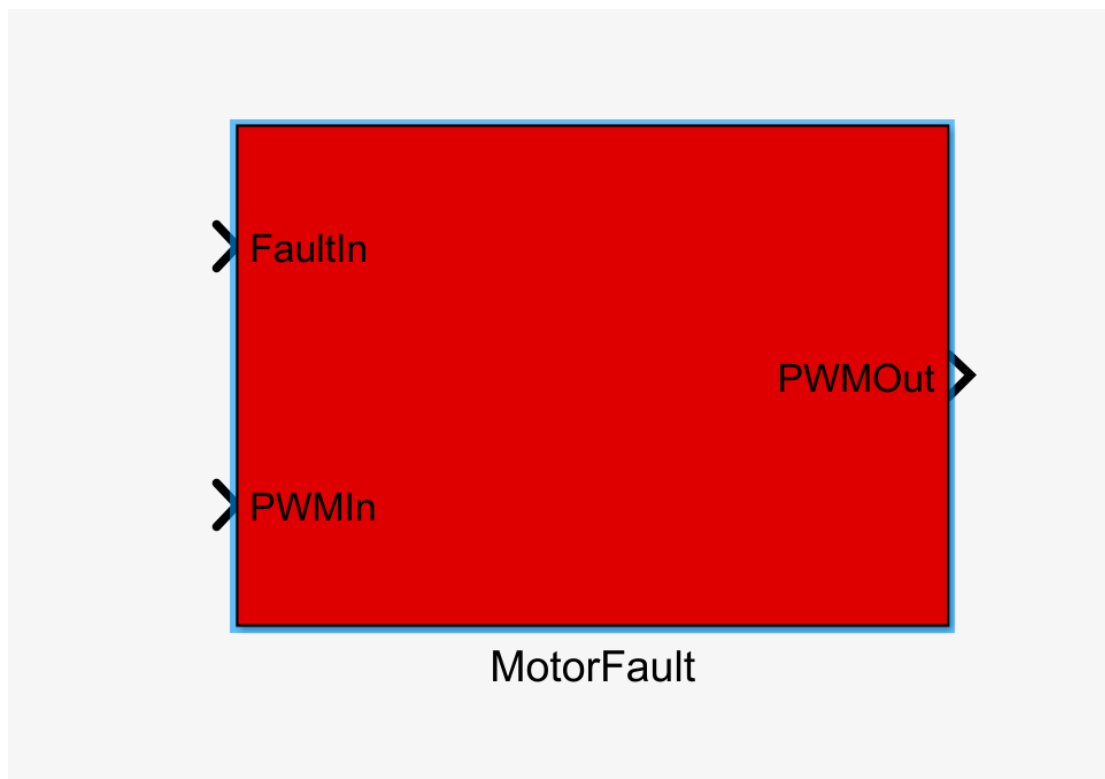
6.7.2 获取故障用例接口 get_fault_case(self)

6.8 故障注入测试用例集的编写与使用

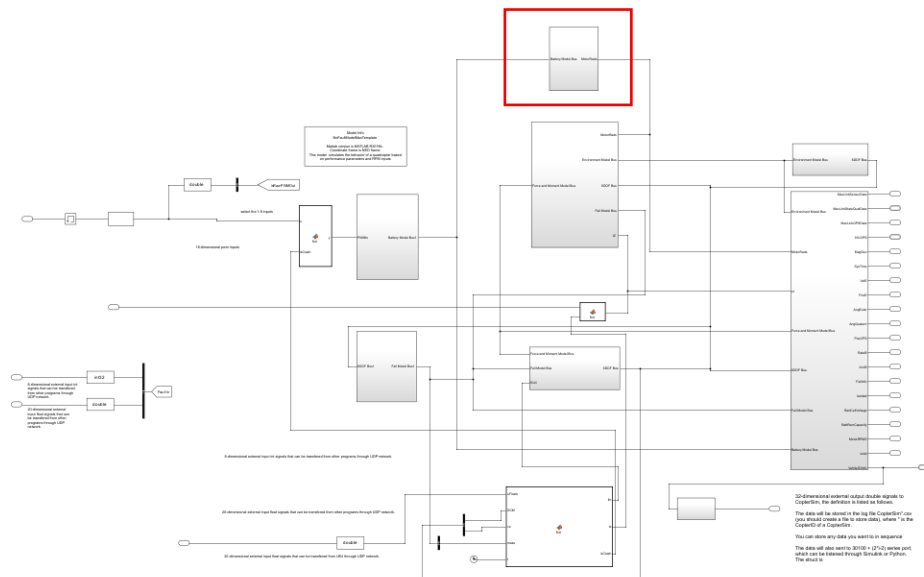
7. 软件在环 simulink 模型故障注入接口与使用

首先，在本章的大部分例程中，都有 MulticopterModelLib.slx 的一个故障模块库，其中包含有各种故障的故障注入模块。

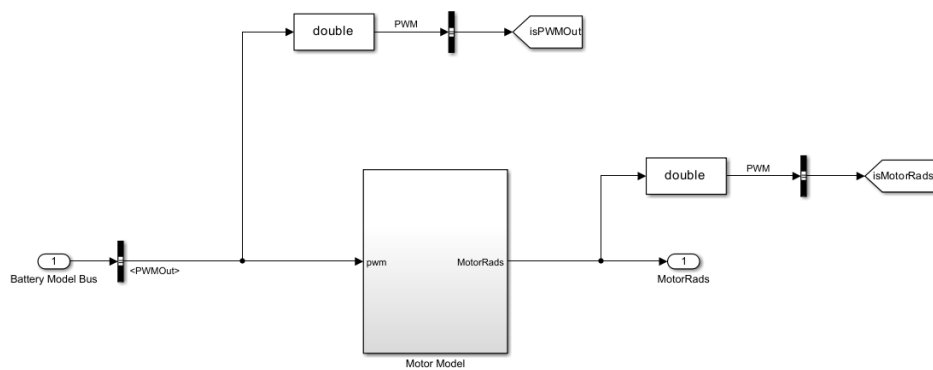
7.1 电机故障（MotorFault）注入与使用



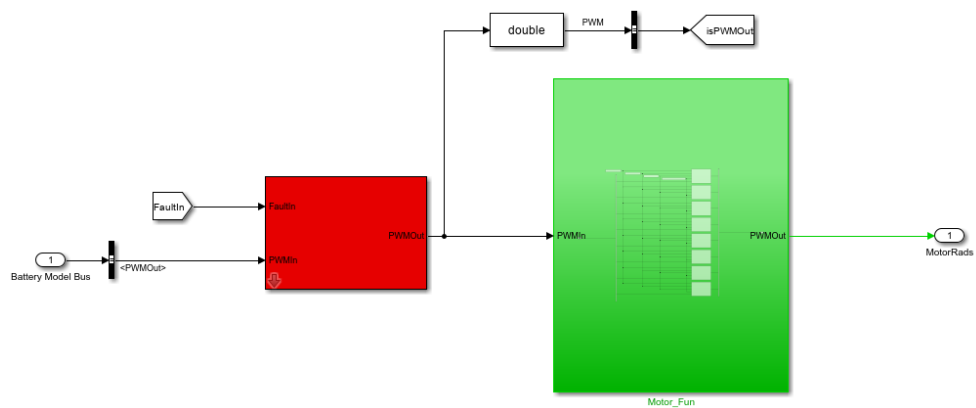
例程文件中的 **MotorFault** 模块是电机故障封装库，我们可以对封装内部进行查看，此模块需要联系没有故障注入的最大模板和没有故障注入的最小模板同时使用，需要将带有故障注入的模板将原本没有故障的模块进行替换使用。



该例程文件为没有故障注入的最大模板，我们需要从中找到电机模块的对应位置。

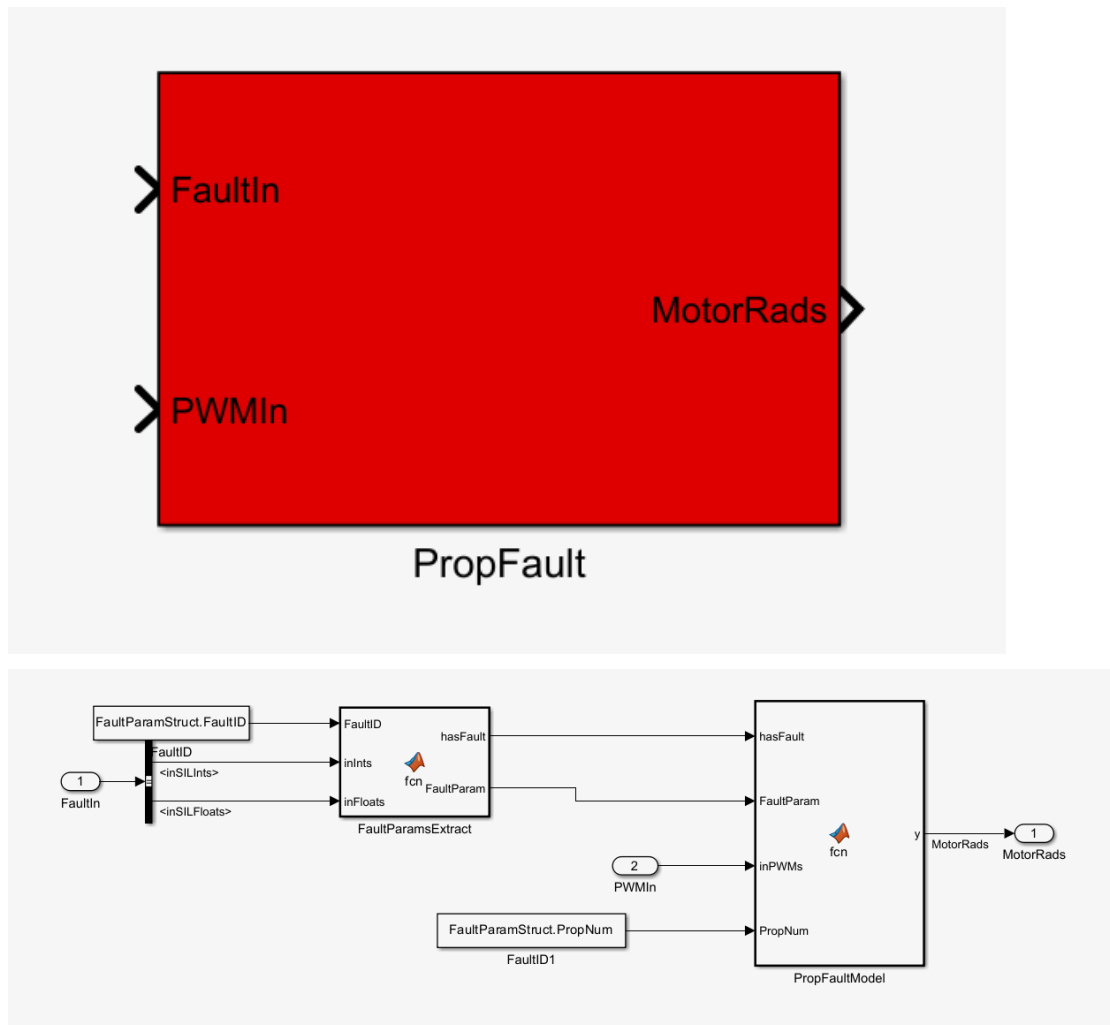


这是需要从封装库中将对应的模块进行更改。

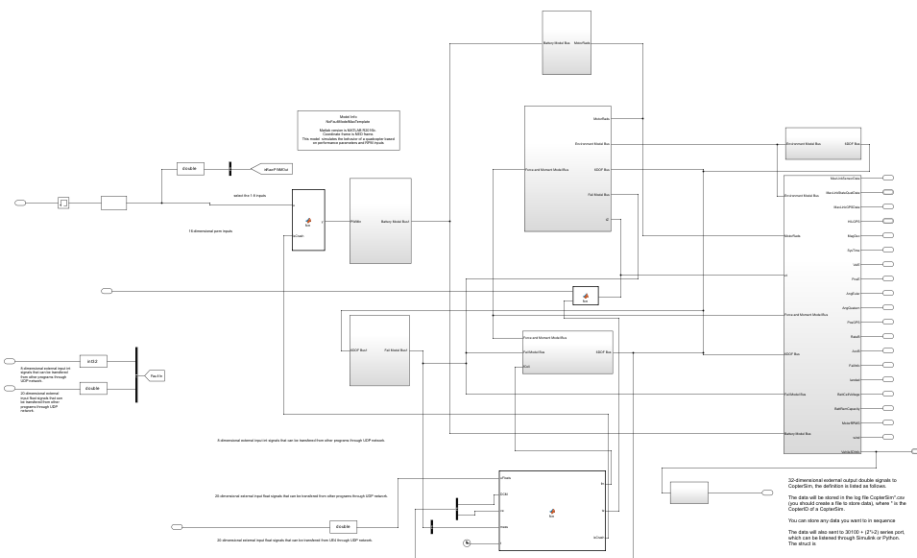


之后，我们便可以对模型进行电机故障注入实验。

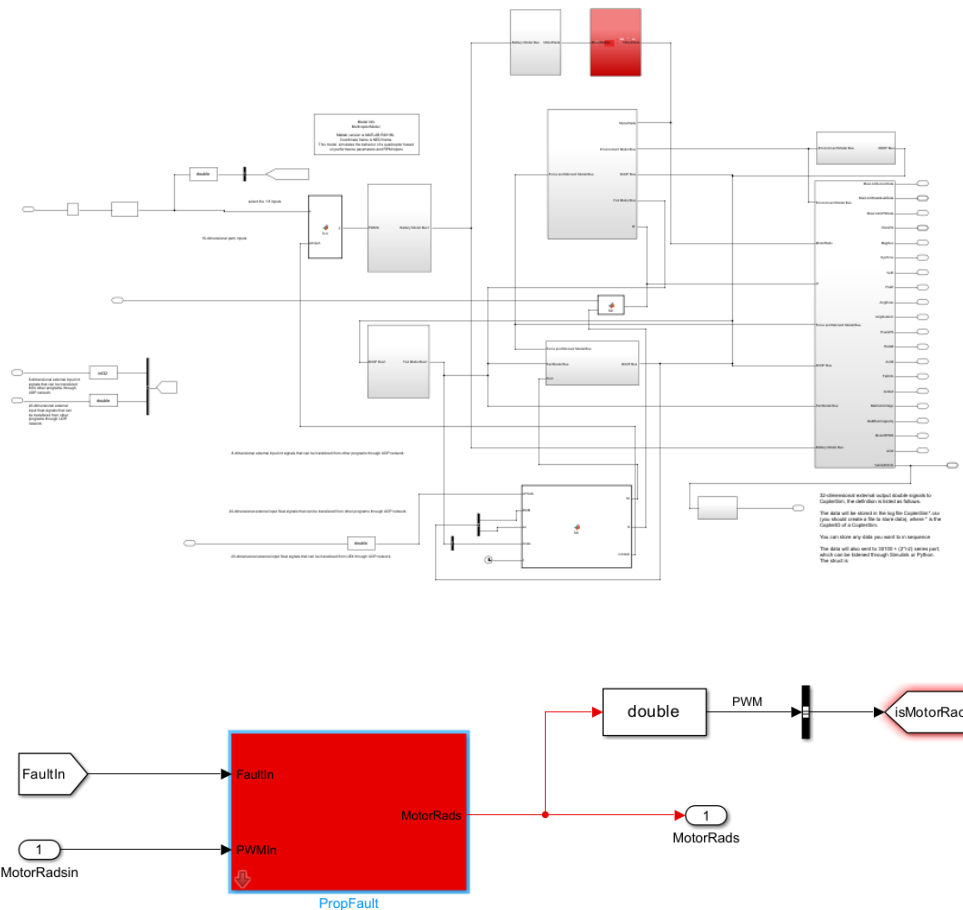
7.2 螺旋桨故障（PropFault）注入与使用



例程文件中的 **PropFault** 模块是螺旋桨故障封装库，我们可以对封装内部进行查看，此模块需要联系没有故障注入的最大模板和没有故障注入的最小模板同时使用，需要将带有故障注入的模板将原本没有故障的模块进行替换使用。

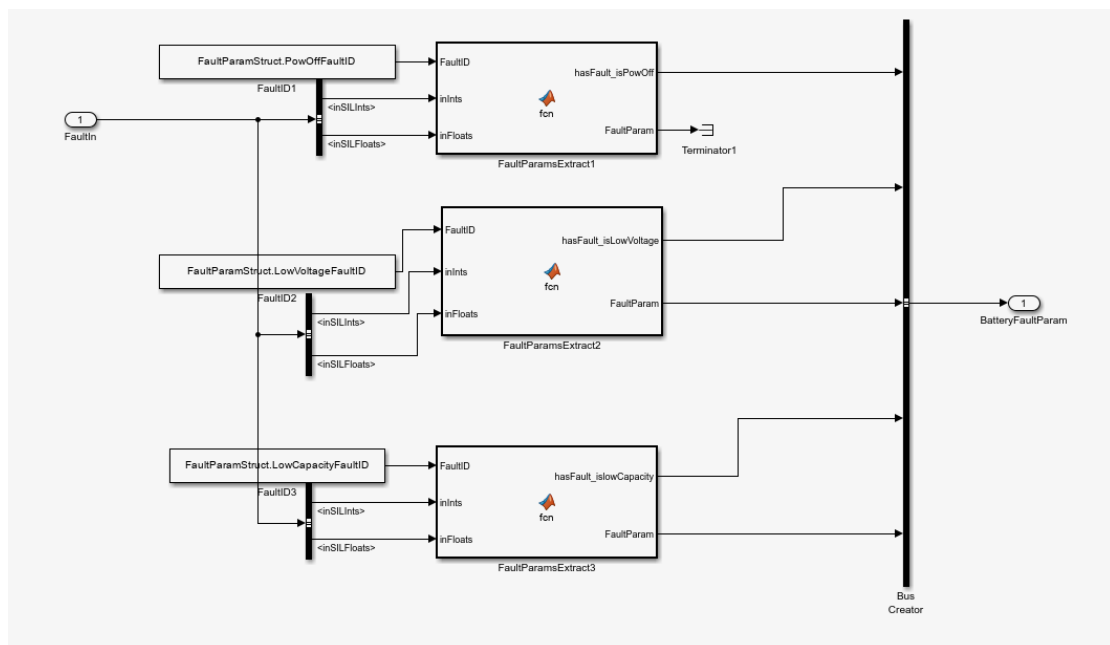
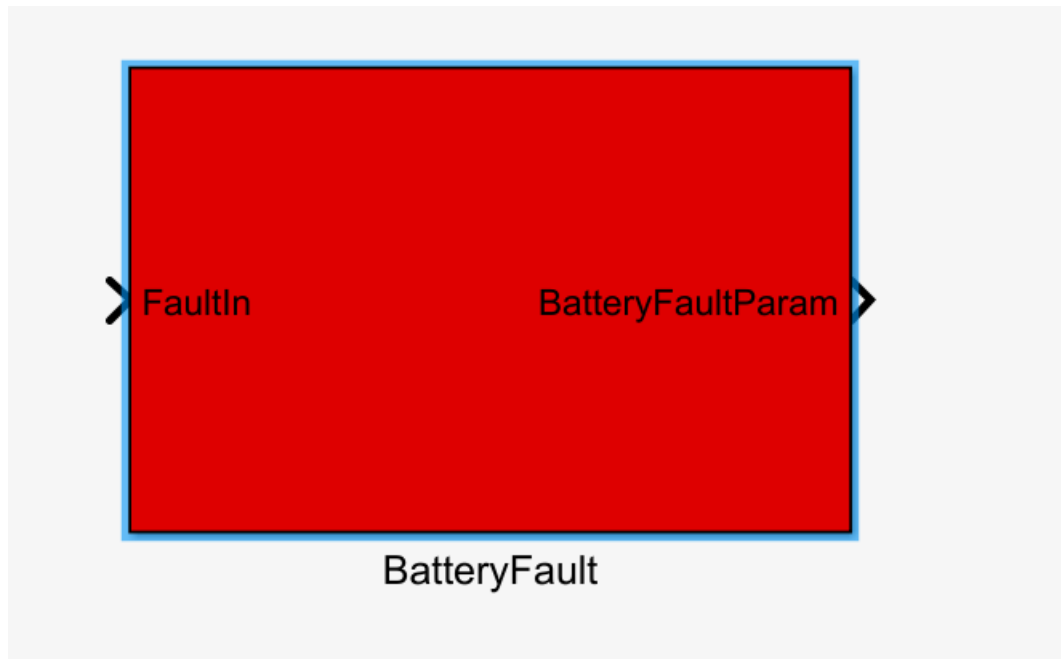


该例程文件为没有故障注入的最大模板，我们需要在其中添加上一个螺旋桨故障注入模块。

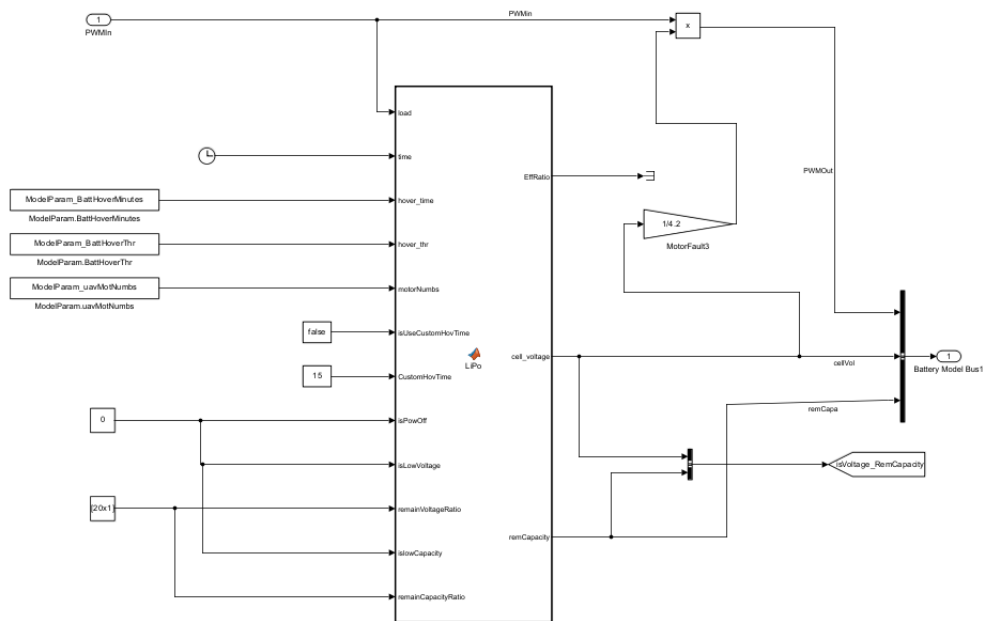


之后，我们便可以对模型进行螺旋桨故障注入实验。

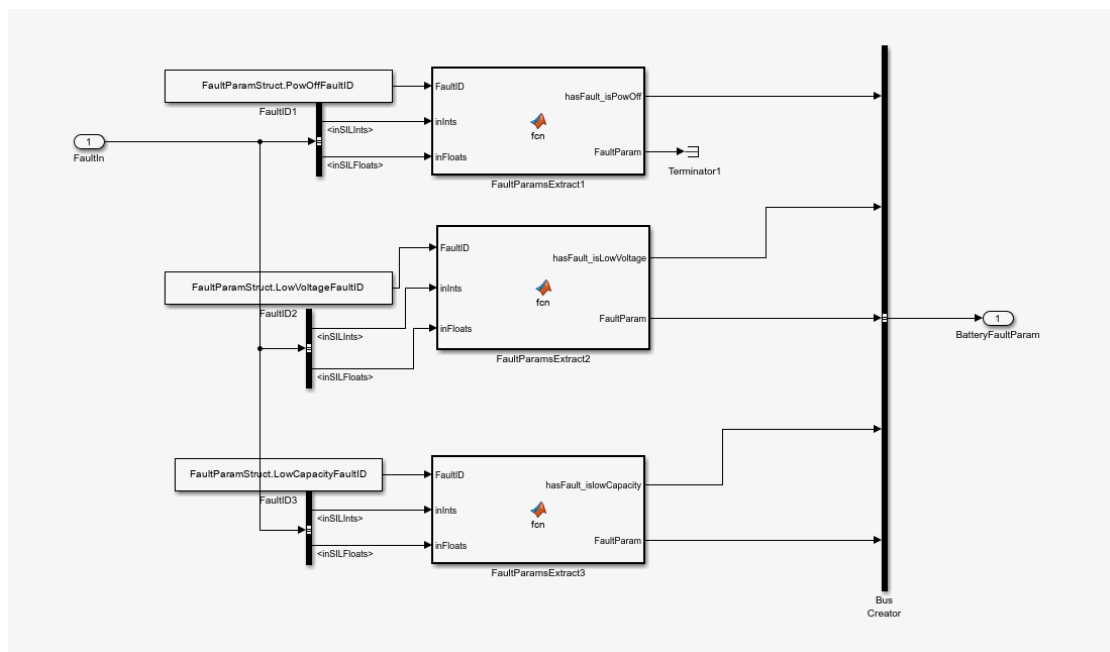
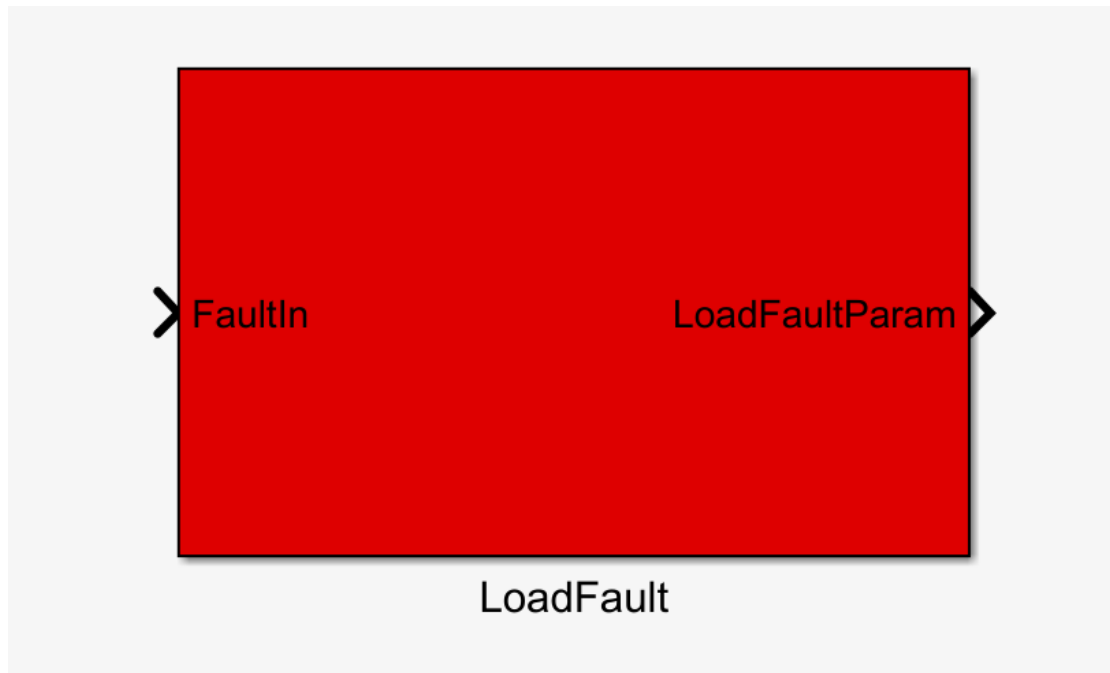
7.3 电池故障（BatteryFault）注入与使用



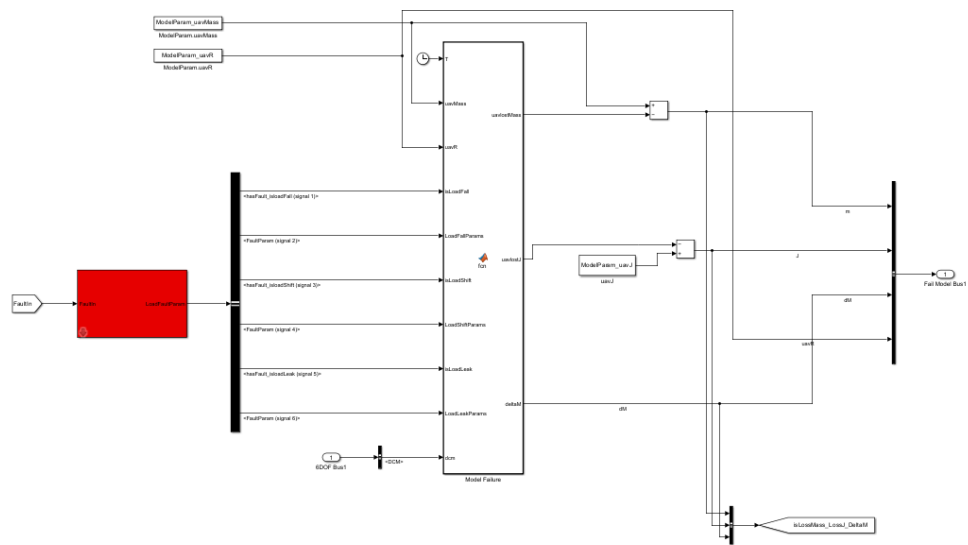
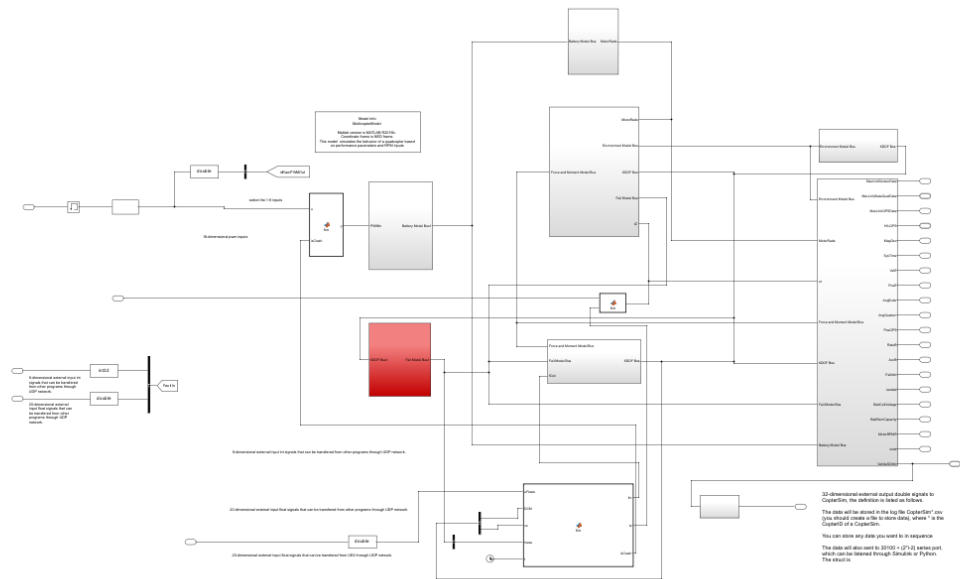
例程文件中的 **BatteryFault** 模块是电池故障封装库，我们可以对封装内部进行查看，此模块需要联系没有故障注入的最大模板和没有故障注入的最小模板同时使用，需要将带有故障注入的模板将原本没有故障的模块进行替换使用。



7.4 负载故障（LoadFault）注入与使用

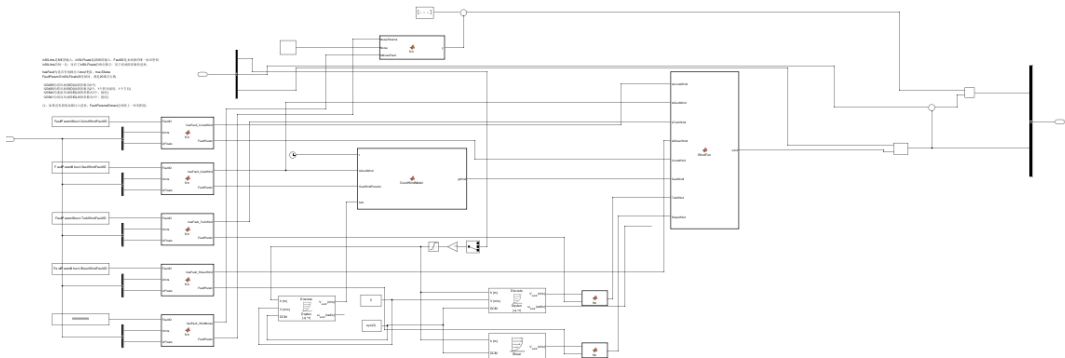
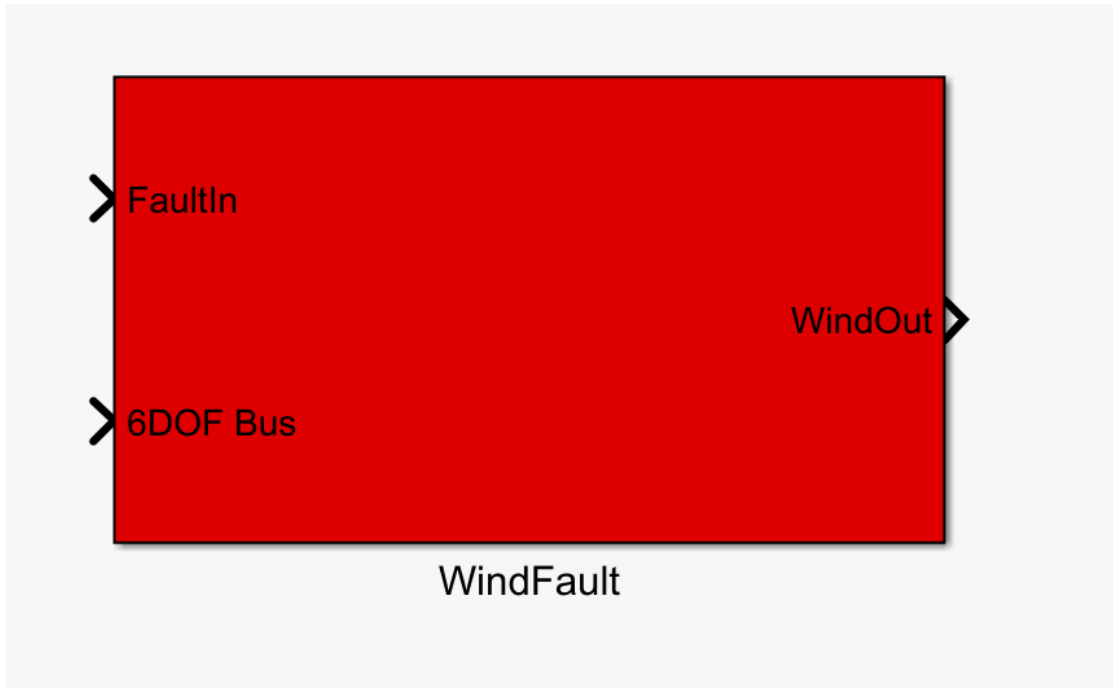


例程文件中的 **LoadFault** 模块是负载故障封装库，我们可以对封装内部进行查看，此模块需要联系没有故障注入的最大模板和没有故障注入的最小模板同时使用，需要将带有故障注入的模板将原本没有故障的模块进行替换使用。

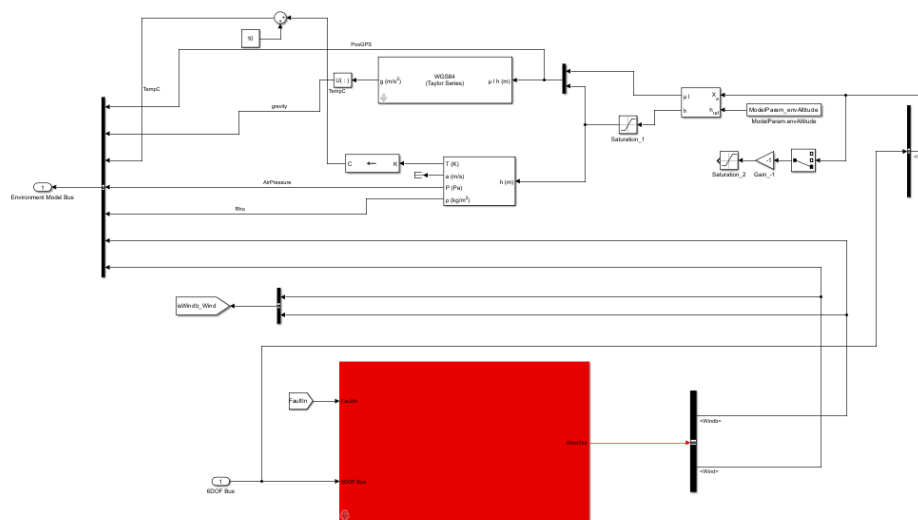


之后，我们便可以对模型进行电池故障注入实验。

7.5 环境风故障（WindFault）注入与使用

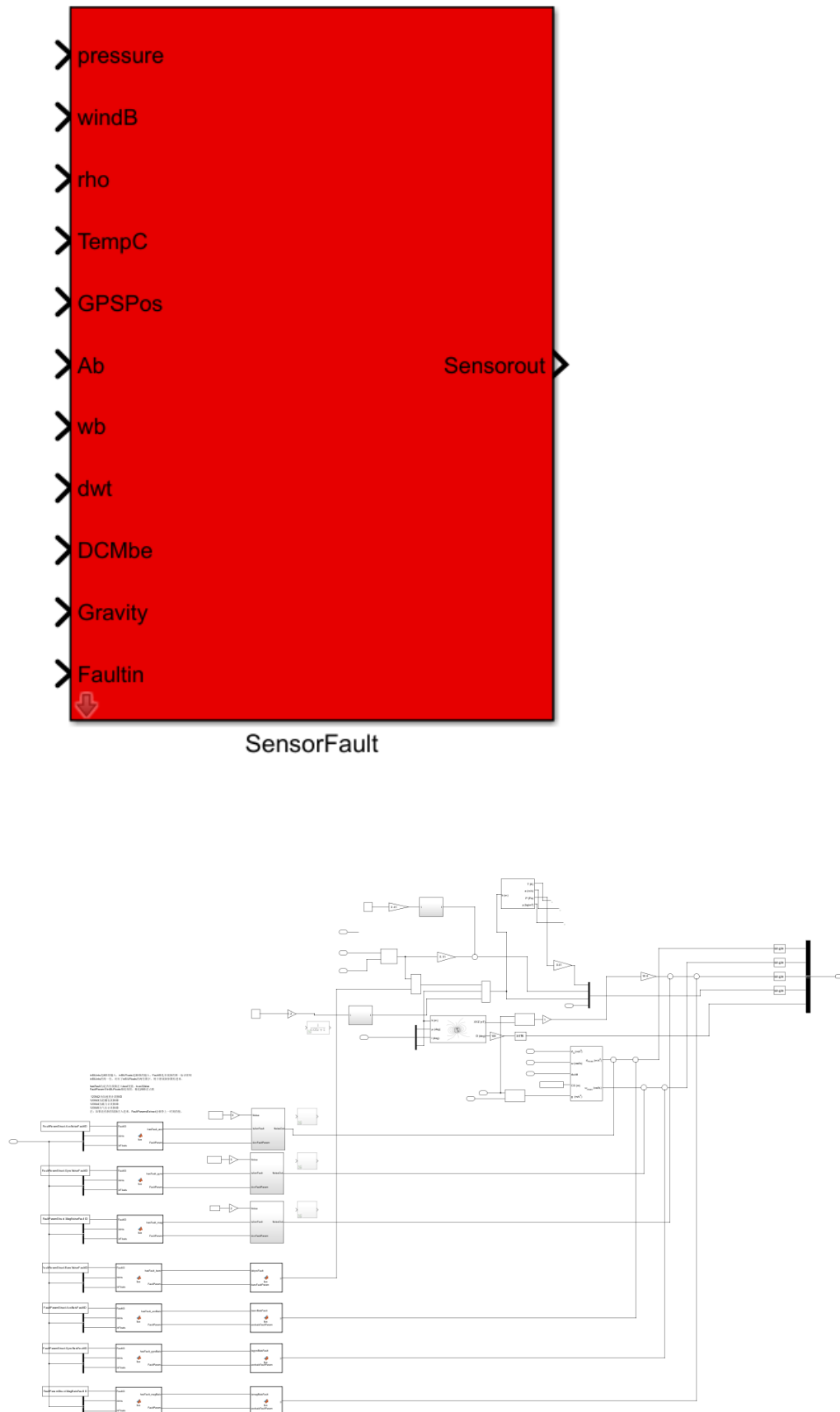


例程文件中的 **WindFault** 模块是环境风故障封装库，我们可以对封装内部进行查看，此模块需要联系没有故障注入的最大模板和没有故障注入的最小模板同时使用，需要将带有故障注入的模板将原本没有故障的模块进行替换使用。

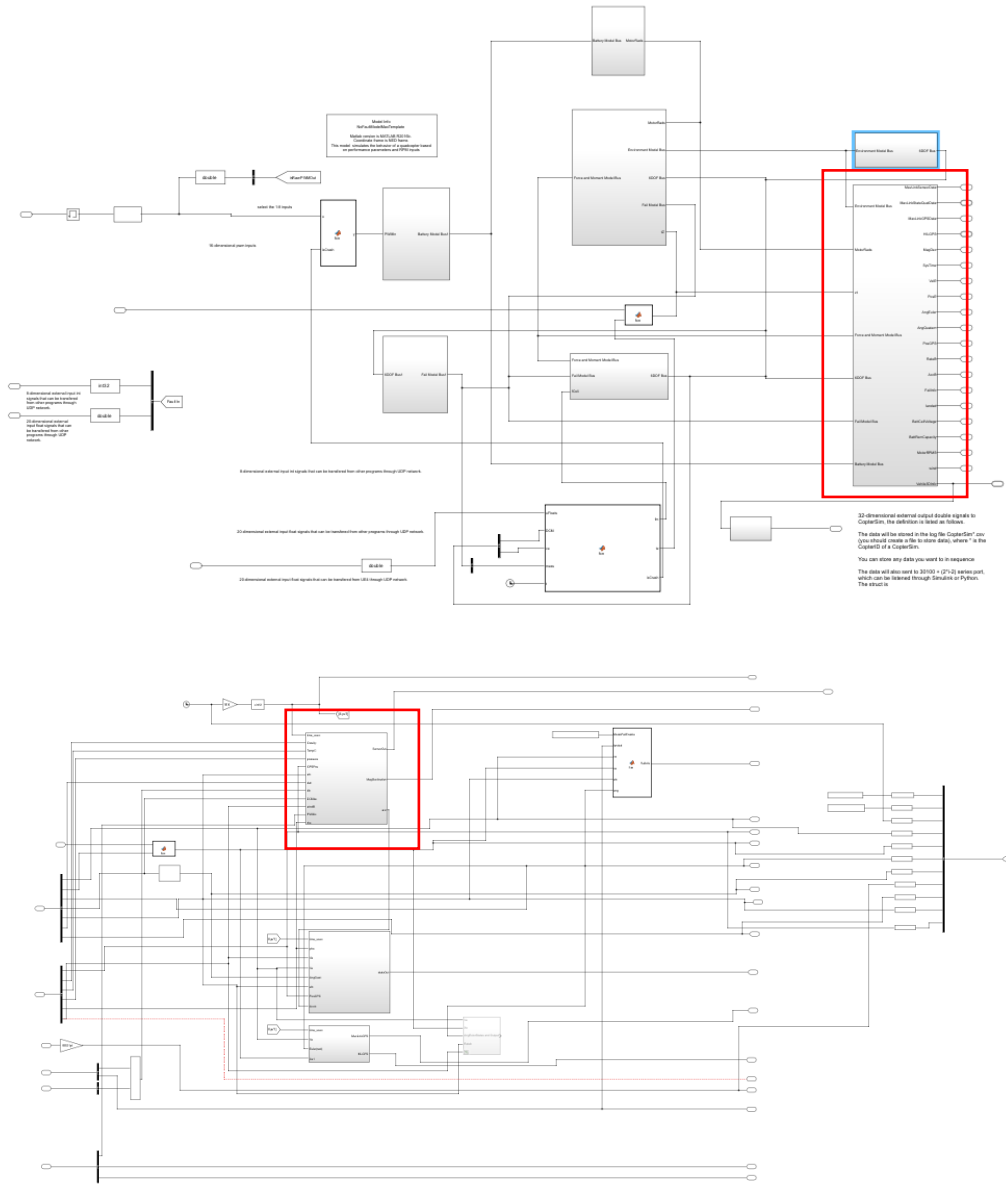


之后，我们便可以对模型进行环境风故障注入实验。

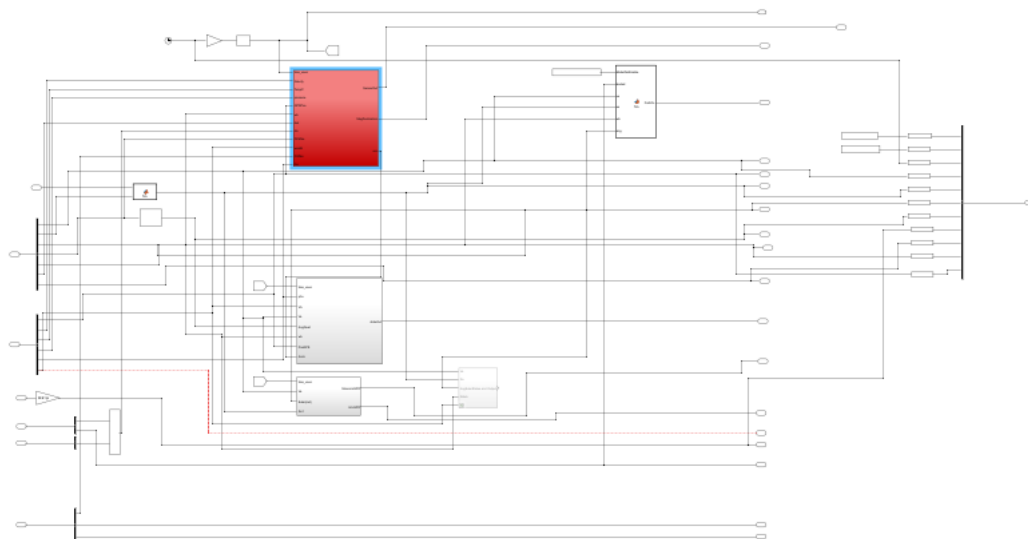
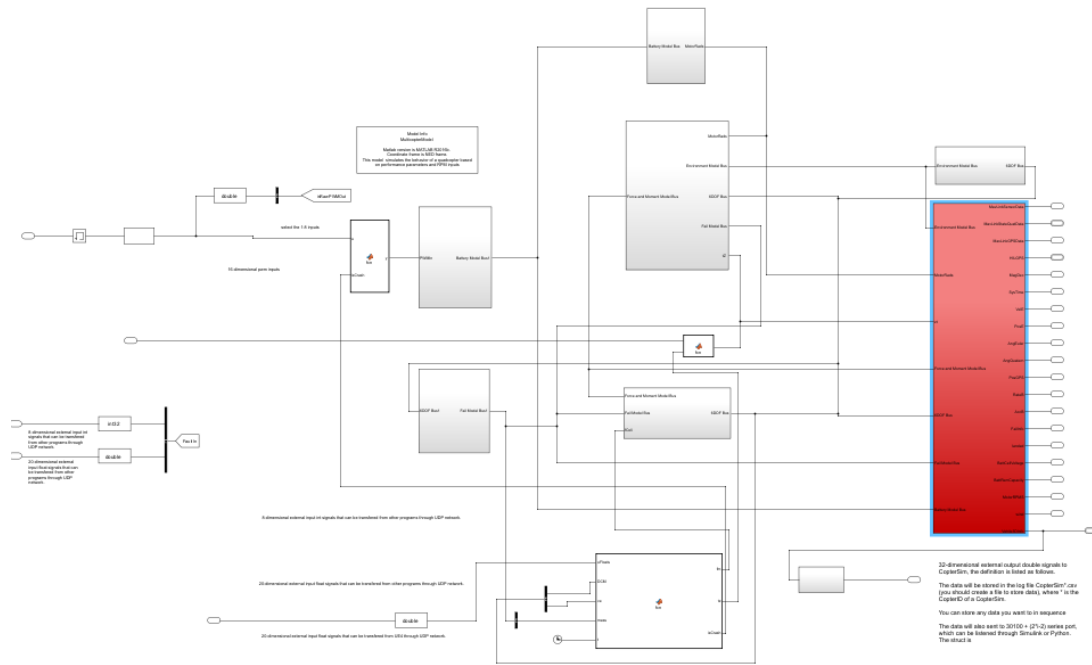
7.6 传感器故障（SensorFault）注入与使用

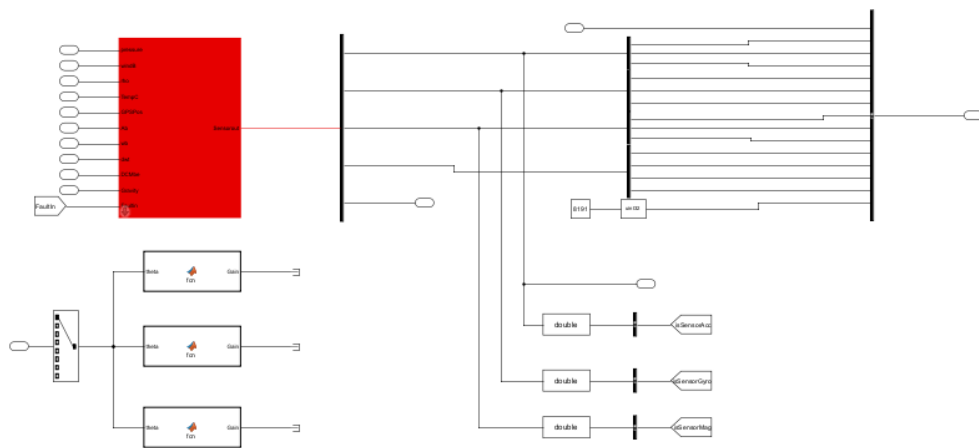
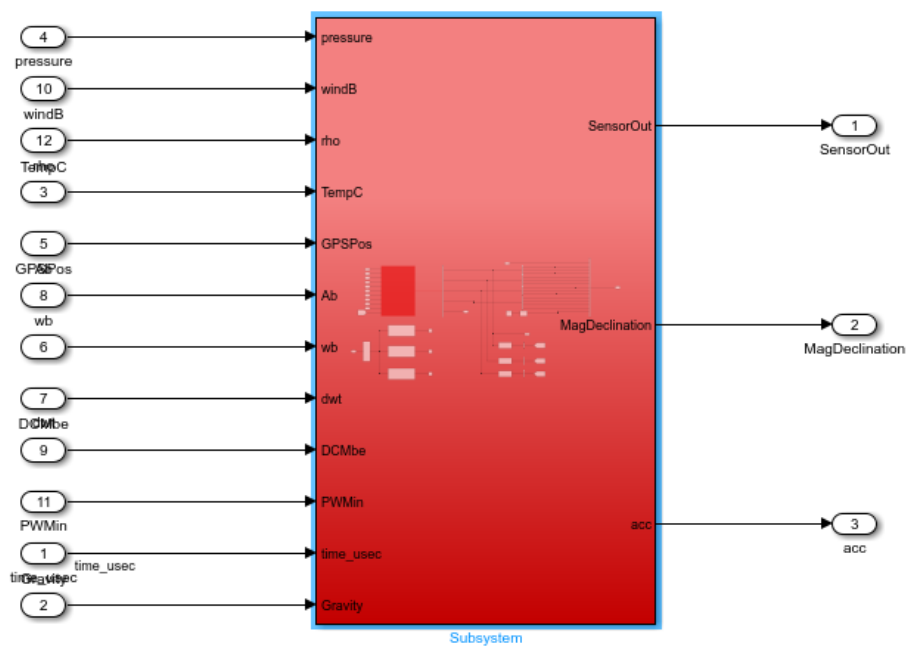


例程文件中的 **SensorFault** 模块是传感器故障封装库，我们可以对封装内部进行查看，此模块需要联系没有故障注入的最大模板和没有故障注入的最小模板同时使用，需要将带有故障注入的模板将原本没有故障的模块进行替换使用。



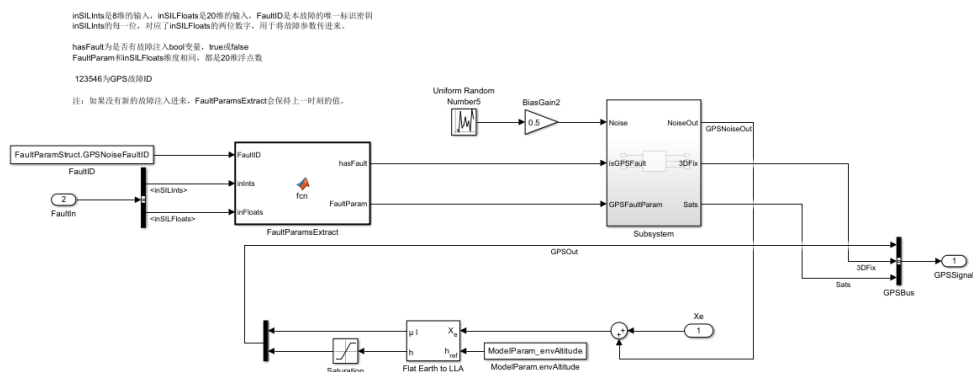
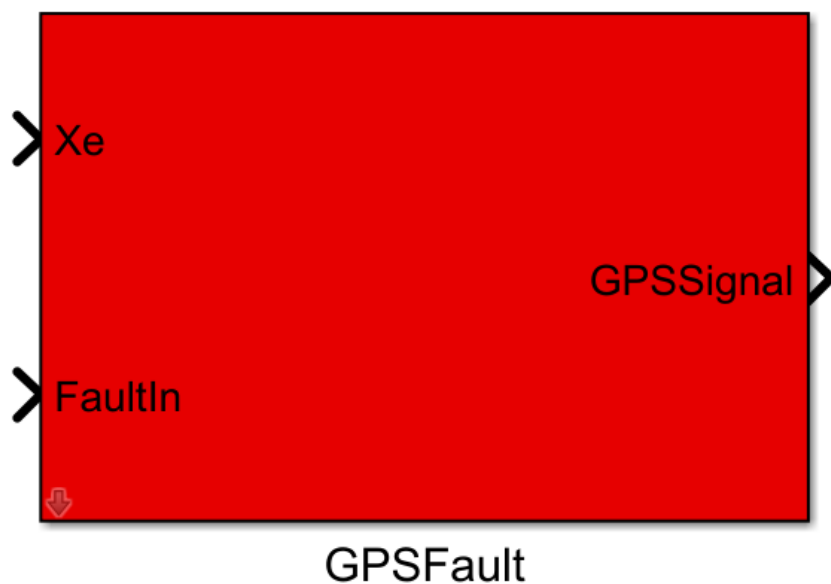
该例程文件为没有故障注入的最大模板，我们需要在其中添加传感器模块的对应位置。



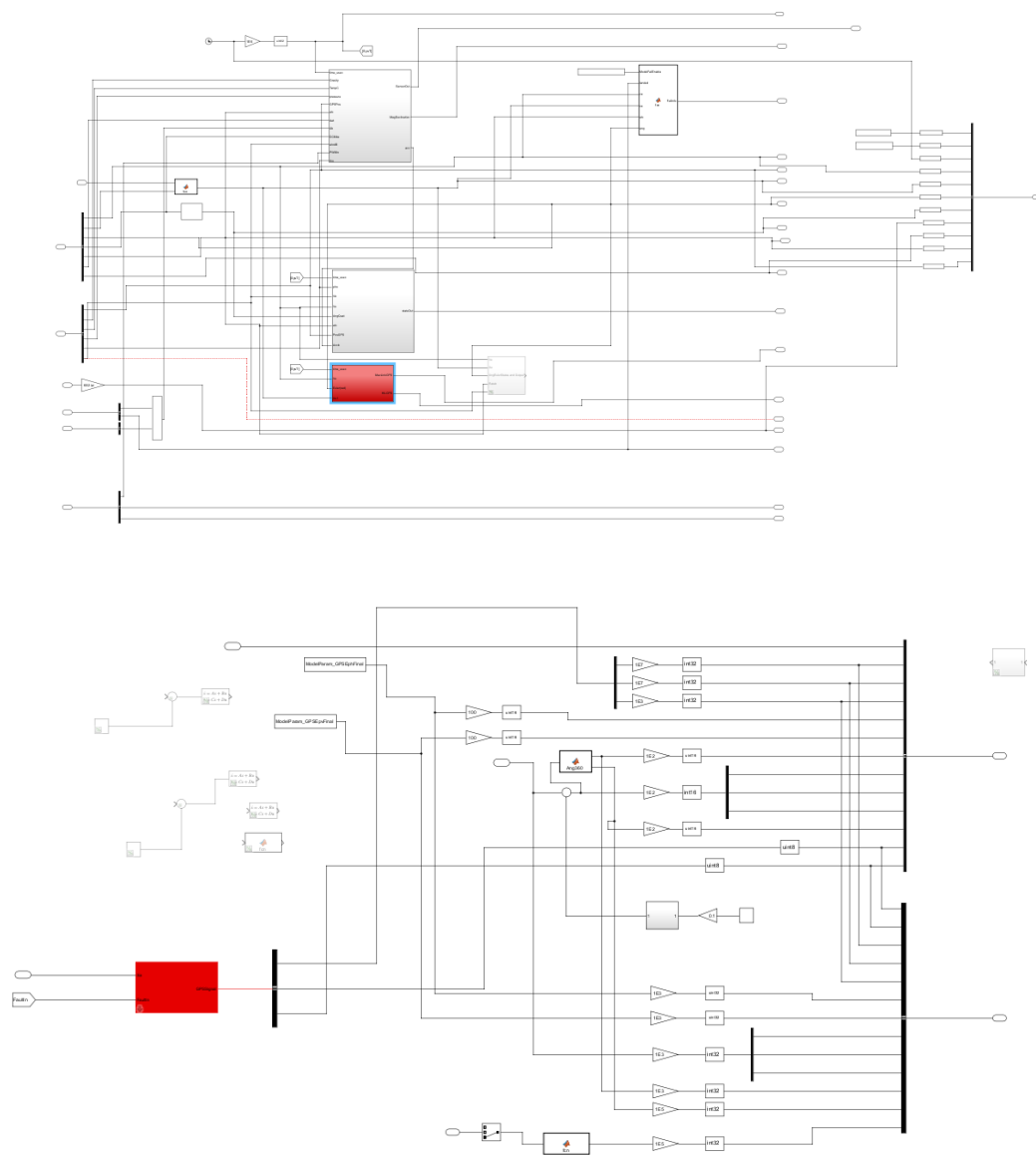


之后，我们便可以对模型进行传感器故障注入实验。

7.7 GPS 故障（GPSFault）注入与使用



例程文件中的 **GPSFault** 模块是 **GPS** 模块故障封装库，我们可以对封装内部进行查看，此模块需要联系没有故障注入的最大模板和没有故障注入的最小模板同时使用，需要将带有故障注入的模板将原本没有故障的模块进行替换使用。



之后，我们便可以对模型进行 **GPS** 故障注入实验。

8. 软件在环可视化故障注入 APP（GUI）

8.1 触发按钮回调逻辑

```
407 % Callbacks that handle component events
408 methods (Access = private)
409
410 % Button pushed function: Button_6
411 function Button_6Pushed(app, event)
412     Initialize(app);
413 end
414
415 % Button pushed function: Button_7
416 function Button_7Pushed(app, event)
417     Stop(app);
418 end
419
420 |
421 % Button pushed function: Button_10
422 function Button_10Pushed(app, event)
423     FaultSend(app);
424 end
425
426 % Button pushed function: Button_8
427 function Button_8Pushed(app, event)
428     StartLog(app);
429 end
430
431 % Button pushed function: Button_9
432 function Button_9Pushed(app, event)
433     SaveParam(app);
434 end
435
436 % Value changed function: CheckBox_31, CheckBox_32, CheckBox_33,
437 % ...and 15 other components
438 function CheckBox_ValueChanged(app, event)
439     CheckBox_ValueChangedUpdate(app, event);
440 end
441
442 % Value changed function: EditField_191, EditField_192,
443 % ...and 33 other components
444 function EditField_ValueChanged(app, event)
445     EditField_ValueChangedUpdate(app, event);
446 end
447
448 % Value changed function: DropDown_3
449 function DropDown_3ValueChanged(app, event)
450     value = app.DropDown_3.Value;
451     RestoreParam(app,value)
452 end
453
454 % Drop down opening function: DropDown_3
455 function DropDown_3Opening(app, event)
456     UpdateRestoreParam(app);
457 end
458 end
459
460 % Component initialization
461 methods (Access = private)
462
463 % Create UIFigure and components
464 function createComponents(app) ***
465 end
466
467 % App creation and deletion
468 methods (Access = public)
469
470 % Construct app
471 function app = Demo ***
472
473 % Code that executes before app deletion
474 function delete(app) ***
475 end
476
```

8.2 udp 故障发送模块编写

```
%udp 发送器
function udp_cmd_sender(app,RemoteIP, RemoteIPPort,Checksum,TargetID, inSILInts,inSILFloats)
persistent remoteip remoteport udps

    if isempty(udps)
        remoteip = RemoteIP;
        remoteport = RemoteIPPort;
        udps = dsp.UDPSender('RemoteIPAddress', RemoteIP, 'RemoteIPPort', RemoteIPPort);
    end

    inSILInts = inSILInts(1:8);
    inSILFloats = inSILFloats(1:20);

    CheckSum = uint32(CheckSum);
    TargetID = uint32(TargetID);
    inSILIntsLen = length(inSILInts);
    inSILFloatsLen = length(inSILFloats);

    inSILInts = int32(inSILInts);
    inSILFloats = single(inSILFloats);

    dataSend = uint8(zeros((inSILIntsLen + inSILFloatsLen) * 4, 1));

    dataSend(1:4) = typecast(CheckSum, 'uint8');
    dataSend(5:8) = typecast(TargetID, 'uint8');
    dataSend(9:40) = typecast(inSILInts, 'uint8');|
    dataSend(41:120) = typecast(inSILFloats, 'uint8');
    udps(dataSend);
    % release(udps);
end
end
```

8.3 故障注入协议编写

```
%之所以分成4列 是因为故障参数id太多
FaultSILIntsParams = zeros(4,8);
FaultSILFloatsParams = zeros(4,20);
FaultCountParams = 0;
systemParams ;%count;
timer1 ;
end
```

9. 硬件在环 PX4 飞控故障模块的编写与使用

9.1 GPS 故障模块的编写与使用

9.1.1 外部注入的 msg 文件（消息格式）

```
uint64 timestamp          # time since system start
(microseconds)           //时间戳
uint32 flags              # control flag
//控制flag
uint8 modes               # mode flag
//模式flag
float32[16] controls      # 16D control signals
//16位控制参数
```

9.1.2 msg 的头文件引用

```
//省略部分代码
#include <uORB/topics/rfly_ctrl.h> //msg格式的头文件
#include <uORB/Subscription.hpp> //订阅操作相关的头文件
//省略部分代码
private:
    /*订阅外部uorb消息rfly_ctrl_s用于触发故障*/
    // 1、声明结构体参数
    rfly_ctrl_s rflydata;
    //2、订阅rfly_ctrl的uorb消息
    uORB::Subscription _rfly_ctrl_sub{ORB_ID(rfly_ctrl)};
//省略部分代码
```

9.1.3 故障消息的订阅与触发故障注入

```
_rfly_ctrl_sub.copy(&rflydata);
// if ( abs(rflydata.controls[0] - 123456) < 0.01f){
if (int(rflydata.controls[0] - 123456) == 0)
{

    if (int(rflydata.controls[1] - 1) == 0)
    {
        _report_gps_pos.lat = (int32_t)rflydata.controls[2];
        _report_gps_pos.lon = (int32_t)rflydata.controls[3];
        _report_gps_pos.alt = (int32_t)rflydata.controls[4];
    }
}
```

```

    if (int(rflydata.controls[1] - 2) == 0)
    {
        _report_gps_pos.lat = (int32_t)(_report_gps_pos.lat +
rflydata.controls[2]);
        _report_gps_pos.lon = (int32_t)(_report_gps_pos.lon +
rflydata.controls[3]);
        _report_gps_pos.alt = (int32_t)(_report_gps_pos.alt +
rflydata.controls[4]);
    }

    if (int(rflydata.controls[1] - 0) != 0)
        _report_gps_pos_pub.publish(_report_gps_pos);
}
else
{
    _report_gps_pos_pub.publish(_report_gps_pos);
}

```

9.2 电机故障模块的编写与使用

9.2.1 外部注入的 msg 文件（消息格式）

```

uint64 timestamp           # time since system start
(microseconds)           //时间戳
uint32 flags              # control flag
//控制flag
uint8 modes               # mode flag
//模式flag
float32[16] controls      # 16D control signals
//16位控制参数

```

9.2.2 msg 的头文件引用

```
//省略部分代码
#include <uORB/topics/rfly_ctrl.h> //msg格式的头文件
#include <uORB/Subscription.hpp> //订阅操作相关的头文件
//省略部分代码
private:
    /*订阅外部uorb消息rfly_ctrl_s用于触发故障*/
    // 1、声明结构体参数
    rfly_ctrl_s rflydata;
    //2、订阅rfly_ctrl的uorb消息
    uORB::Subscription _rfly_ctrl_sub{ORB_ID(rfly_ctrl)};
//省略部分代码
```

9.2.3 故障消息的订阅与触发故障注入

```
/* output to the servos */
if (_pwm_initialized) {
    for (size_t i = 0; i < math::min(_num_outputs, num_outputs); i++) {
        up_pwm_servo_set(_output_base + i, outputs[i]);
    }

    if (int(rflydata.controls[0] - 123450) == 0)
    {
        for (size_t i = 0; i < math::min(_num_outputs, num_outputs); i++)
        {
            if (int(rflydata.controls[1] - 1) == 0)
                up_pwm_servo_set(_output_base + i, rflydata.controls[i +
2]);

            else if(int(rflydata.controls[1] - 2) == 0)
                up_pwm_servo_set(_output_base + i, outputs[i] +
rflydata.controls[i + 2]);
        }
    }
}
```

9.3 遥控故障模块的编写与使用

9.3.1 外部注入的 msg 文件（消息格式）

```
uint64 timestamp          # time since system start
(microseconds)           //时间戳
uint32 flags              # control flag
//控制flag
uint8 modes               # mode flag
//模式flag
float32[16] controls      # 16D control signals
//16位控制参数
```

9.3.2 msg 的头文件引用

```
//省略部分代码
#include <uORB/topics/rfly_ctrl.h> //msg格式的头文件
#include <uORB/Subscription.hpp> //订阅操作相关的头文件
//省略部分代码
private:
    /*订阅外部uorb消息rfly_ctrl_s用于触发故障*/
    // 1、声明结构体参数
    rfly_ctrl_s rflydata;
    //2、订阅rfly_ctrl的uorb消息
    uORB::Subscription _rfly_ctrl_sub{ORB_ID(rfly_ctrl)};
//省略部分代码
```

9.3.3 故障消息的订阅与触发故障注入

```
// 3、取出uorb的值
_rfly_ctrl_sub.copy(&rflydata);

// receive rflysim msg
// if(abs(rflydata.controls[0]-123457)<0.01){
if (int(rflydata.controls[0] - 123457) == 0)
{

    if (int(rflydata.controls[1] - 1) == 0)
    {
        _rc_in.values[i] = rflydata.controls[i + 2];
    }
}
```

```

        if (int(rflydata.controls[1] - 2) == 0)
        {
            _rc_in.values[i] = raw_rc_values_local[i] + rflydata.controls[i + 2];
        }
    }
    else
    {
        _rc_in.values[i] = raw_rc_values_local[i];
    }

    if (raw_rc_values_local[i] != UINT16_MAX)
    {
        valid_chans++;
    }

    // once filled, reset values back to default
    _raw_rc_values[i] = UINT16_MAX;
}

```

9.4 地磁故障模块的编写与使用

9.4.1 外部注入的 msg 文件（消息格式）

```

uint64 timestamp           # time since system start
(microseconds)           //时间戳
uint32 flags              # control flag
//控制flag
uint8 modes               # mode flag
//模式flag
float32[16] controls      # 16D control signals
//16位控制参数

```


9.4.2 msg 的头文件引用

```
//省略部分代码
#include <uORB/topics/rfly_ctrl.h> //msg格式的头文件
#include <uORB/Subscription.hpp> //订阅操作相关的头文件
//省略部分代码
private:
    /*订阅外部uorb消息rfly_ctrl_s用于触发故障*/
    // 1、声明结构体参数
    rfly_ctrl_s rflydata;
    //2、订阅rfly_ctrl的uorb消息
    uORB::Subscription _rfly_ctrl_sub{ORB_ID(rfly_ctrl)};
//省略部分代码
```

9.4.3 故障消息的订阅与触发故障注入

```
//省略部分代码

    _rfly_ctrl_sub.copy(&rflydata); // 取出uorb的值，取出消息订阅的值

    // if(abs(rflydata.controls[0]-123455)<0.01 )
    if (int(rflydata.controls[0] - 123455) == 0) // 判断故障ID，符合
进入故障
    {
```

//如果故障模式为1，则为覆盖模式，直接将输出值替换成故障注入中的值

```
if (int(rflydata.controls[1] - 1) == 0)
{
    report.x = rflydata.controls[2];
    report.y = rflydata.controls[3];
    report.z = rflydata.controls[4];
}
```

//如果故障模式为2，则为叠加模式，直接将输出值替换成故障注入中的值和传感器自身值的和

```
if (int(rflydata.controls[1] - 2) == 0)
{
    report.x = report.x + rflydata.controls[2];
    report.y = report.y + rflydata.controls[3];
    report.z = report.z + rflydata.controls[4];
}
```

//如果故障模式为0，则为拦截状态，即直接拦截传感器的值，即传感器的状态不更新，默认丢失

```
if (int(rflydata.controls[1] - 0) != 0)
{
    _sensor_pub.publish(report);
}
else
{
    /// 如果故障模式输入其它的值则为正常模式，不做处理
    _sensor_pub.publish(report);
}
```

//省略部分代码

10. 飞控日志的收集与处理

10.1 数据收集

10.2 数据实时获取

10.3 数据分析

10.4 数据标注

11. 安全评估算法设计与使用 Health_ass.py

11.1 数据筛选

11.2 数据方差值

11.3 率模加权重

11.4 安全评估

12. 基于神经网络的健康评估算法的设计与使用

12.1 故障数据的获取 AutoTestAPI.py

12.1.1 自启动脚本 FixedwingModelHITL

12.1.2 故障用例读取 caselist

12.2 数据集制作 data_handle.py

12.2.1 选取关键维度 fnmatch

12.2.2 关键数据合成（合成大表） join

12.3 模型训练 train.py

12.3.1 定义模型 DNN

12.3.2 训练 train_accuracy

12.4 在线评估 AutoTestAPI.py

12.4.1 引入模型 load_model

12.4.2 实时评估 `model.predict`