1. 实验名称及目的

1.1. 实验名称

基于 binary_logger 模块的日志写入和读取验证实验

1.2. 实验目的

使用二进制日志记录模块：binary_logger，完成飞行数据写入与读取，log 数据记录，以 RflySim 平台设定了 20s 的四维随机数据，数据存储位置飞控板内的片上外设存储卡内（路径为/fs/microsd/log/pixhawk），熟悉 PX4 飞控的底层运行逻辑。

1.3. 关键知识点

Counter Free-Running—自由运行计数器

如下图所示，Counter Free-Running（自由运行计数器）模块是一个用于生成连续递增或递减计数序列的模块。它可以用于模拟时序信号、控制循环次数等应用场景。

主要特点和用法包括：

计数方向：Counter Free-Running 模块可以配置为递增或递减计数器，具体取决于自己需求。可以设置递增或递减的步长和初始值。

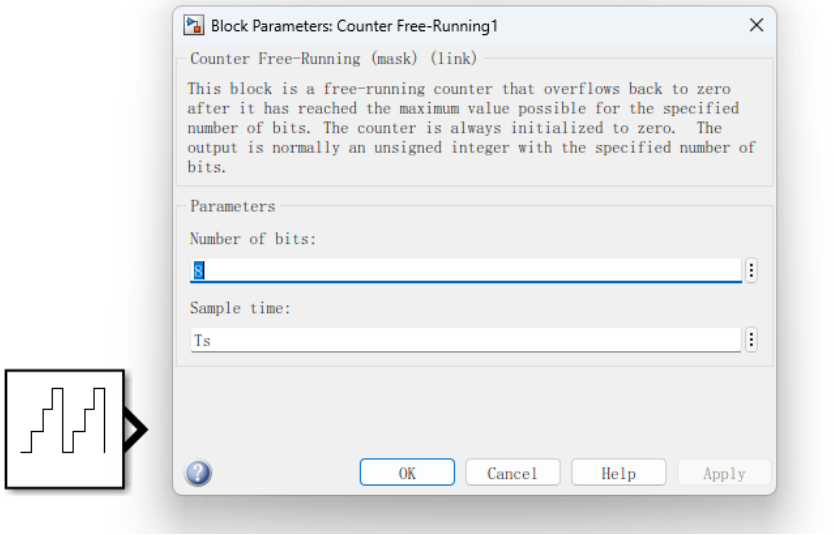
计数范围：可以指定计数器的范围，即计数器在何时回到初始值或停止计数。这对于模拟循环计数或者控制计数范围非常有用。

输出信号：Counter Free-Running 模块会生成一个连续的计数序列作为输出信号。可以将该信号连接到其他模块，用于模拟时序信号、触发事件或者控制其他模块的行为。

触发方式：可以选择不同的触发方式来启动或停止计数器。例如，可以基于外部触发信号来启动计数器，也可以选择模拟时钟信号来控制计数器的运行。

灵活性：Counter Free-Running 模块提供了灵活的参数设置，能够根据具体需求对计数器进行配置，并根据需要调整计数器的行为。

在达到指定位数的最大值后溢出回零。计数器总是初始化为零，输出通常是一个具有指定位数的无符号整数。



Band-Limited White Noise—带限白噪声

如下图所示， Band-Limited White Noise（带限白噪声）模块用于生成具有特定频率范围内的白噪声信号。白噪声是一种具有均匀功率谱密度的随机信号，在所有频率上具有相等的能量。

Band-Limited White Noise 模块的主要特点和用法包括：

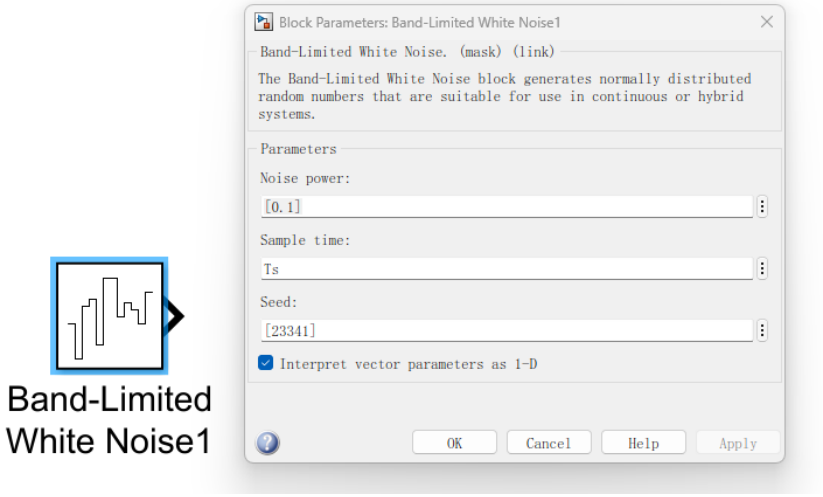
频率范围：可以通过设置模块参数来指定白噪声信号的频率范围，即带宽。允许限制白噪声信号的频率范围，使其只包含特定频率范围内的随机成分。

功率谱密度：带限白噪声模块生成的信号具有均匀的功率谱密度，这意味着在指定的频率范围内，每个频率上的功率相等。这使得该模块适用于模拟特定频率范围内的噪声。

采样时间：可以设置模块的采样时间，以确定每个时间步长内生成的噪声样本数。这影响了生成的白噪声信号的时间分辨率和精度。

输出信号：Band-Limited White Noise 模块生成的信号可以作为模型中的输入信号，用于模拟系统的随机噪声成分。可以将该信号连接到其他模块，例如控制系统、滤波器或传感器模型中，以评估系统在噪声环境下的性能。

参数配置：该模块提供了丰富的参数配置选项，包括频率范围、采样时间、随机种子等，能够根据具体需求调整白噪声信号的生成方式。



Repeating Sequence Stair—重复序列阶梯

如下图所示，Repeating Sequence Stair（重复序列阶梯）模块用于生成重复的阶梯状序列信号。这种信号通常用于模拟周期性变化或周期性行为，例如周期性控制、测试或仿真应用中。

Repeating Sequence Stair 模块的主要特点和用法包括：

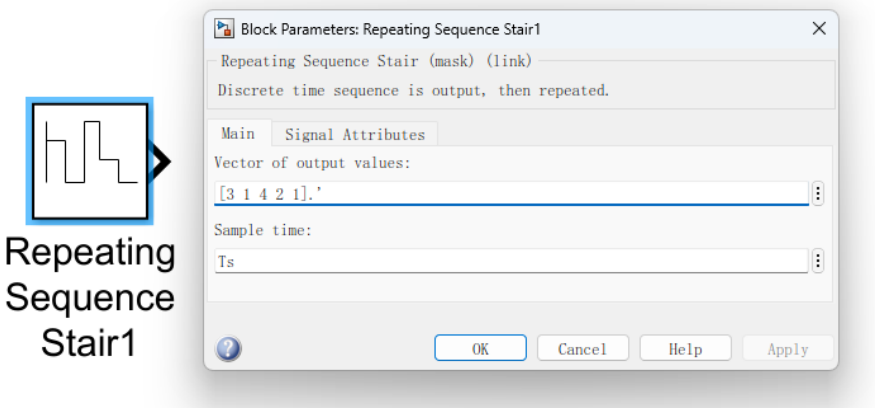
阶梯序列：该模块生成一个具有阶梯状变化的序列信号，其中每个阶梯段的持续时间和幅度可以自定义。这使得能够模拟周期性的变化或离散事件。

重复周期：可以设置重复序列的周期，即序列在重复周期内重复出现。这使得能够模拟周期性行为，例如周期性控制信号或测试序列。

阶梯数量：可以指定生成的阶梯数量，并为每个阶梯段指定持续时间和幅度。这使得能够自定义阶梯序列的形状和特性。

输出信号：Repeating Sequence Stair 模块生成的信号可以作为模型中的输入信号，用于模拟系统的周期性行为或周期性控制信号。可以将该信号连接到其他模块，例如控制系统、传感器模型或仿真环境中，以评估系统在周期性条件下的性能。

参数配置：该模块提供了丰富的参数配置选项，包括重复周期、阶梯数量、每个阶梯段的持续时间和幅度等，使能够根据具体需求调整重复序列的生成方式。



Random Number—随机数

如下图所示，Random Number（随机数）模块用于生成各种类型的随机数信号，例如均匀分布随机数、高斯（正态）分布随机数等。这些随机数信号可以用于模拟系统中的随机性或噪声，进行概率分析，或者用于生成测试数据。

Random Number 模块的主要特点和用法包括：

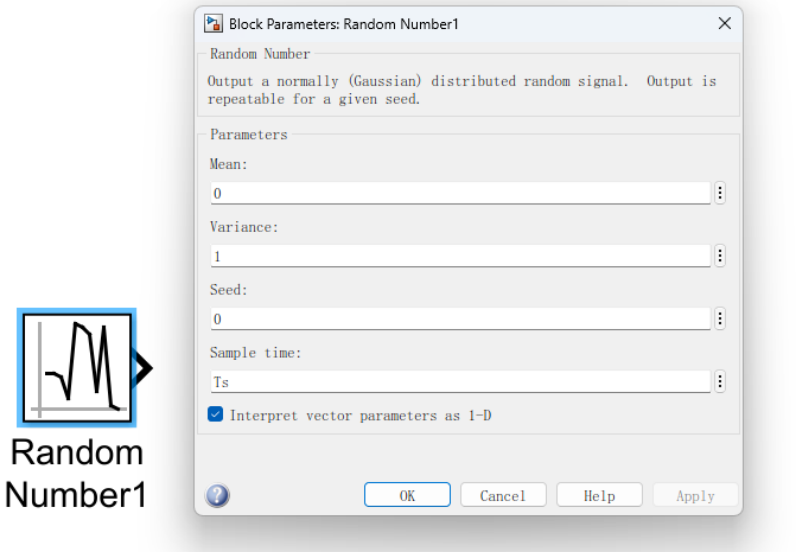
随机数类型：该模块支持生成多种类型的随机数，包括均匀分布随机数、高斯（正态）分布随机数、二项分布随机数等。可以根据需要选择合适的随机数类型。

参数设置：可以通过设置模块的参数来调整随机数的生成方式，例如设置随机数的范围、分布参数、种子等。这使得能够根据具体需求生成符合特定要求的随机数信号。

输出信号：Random Number 模块生成的随机数信号可以作为模型中的输入信号，用于模拟系统中的随机性或噪声。可以将该信号连接到其他模块，例如控制系统、滤波器模型或传感器模型中，以评估系统在随机环境下的性能。

统计分析：生成的随机数信号可以用于进行概率分析和统计分析，例如计算随机变量的均值、方差、概率密度函数等，以评估系统的性能和可靠性。

通过 Random Number 模块，可以方便地在 Simulink 中生成各种类型的随机数信号，用于模拟系统中的随机性或噪声，进行概率分析，以及生成测试数据。



Constant—常数如下图所示，Constant（常数）模块用于生成一个固定值的信号。这个固定值可以是任何你想要的数值，可以是标量、向量或矩阵形式，用于提供模型中的常量输入或参数。

Constant 模块的主要特点和用法包括：

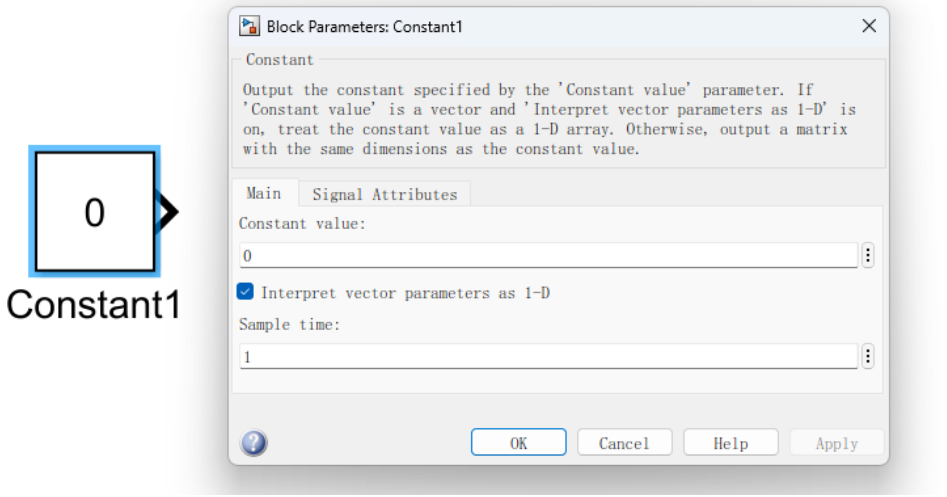
固定数值：该模块生成的信号具有固定的数值，不会随时间变化。可以在模块的参数设置中指定这个固定的数值。

参数设置：可以在 Constant 模块的参数设置中指定生成的固定数值。这使得能够轻松地在模型中引入常量值，用于初始化参数、提供固定输入信号等。

输出信号：Constant 模块生成的固定值信号可以作为模型中的输入信号，连接到其他模块的输入端口，用于模拟系统中的固定条件或参数。例如，可以将该信号连接到控制系统中的某些参数，以评估系统在固定条件下的行为。

多维数组支持：除了标量值外，Constant 模块还支持生成向量或矩阵形式的固定值信号。这使得能够在模型中引入多维数组的常量值，用于初始化参数或提供多维输入信号。

通过 Constant 模块，可以方便地在 Simulink 中生成固定的数值信号，用于提供常量输入或参数，初始化系统参数，或者用作模型中的固定条件。



Delay—延迟

如下图所示，Delay（延迟）模块用于在信号路径中引入延迟。这个延迟可以是固定的时间延迟，也可以是可变的延迟，取决于如何配置模块。

Delay 模块的主要特点和用法包括：

固定延迟：可以在模块的参数设置中指定一个固定的延迟时间。这意味着模块将输入信号延迟指定的时间后输出。这种固定延迟通常用于模拟系统中的信号传播延迟或者处理器执行延迟等情况。

可变延迟：除了固定延迟外，Delay 模块还支持通过输入端口动态地指定延迟时间。这使得能够根据系统状态或其他信号的值来调整延迟时间，从而实现更灵活的控制。

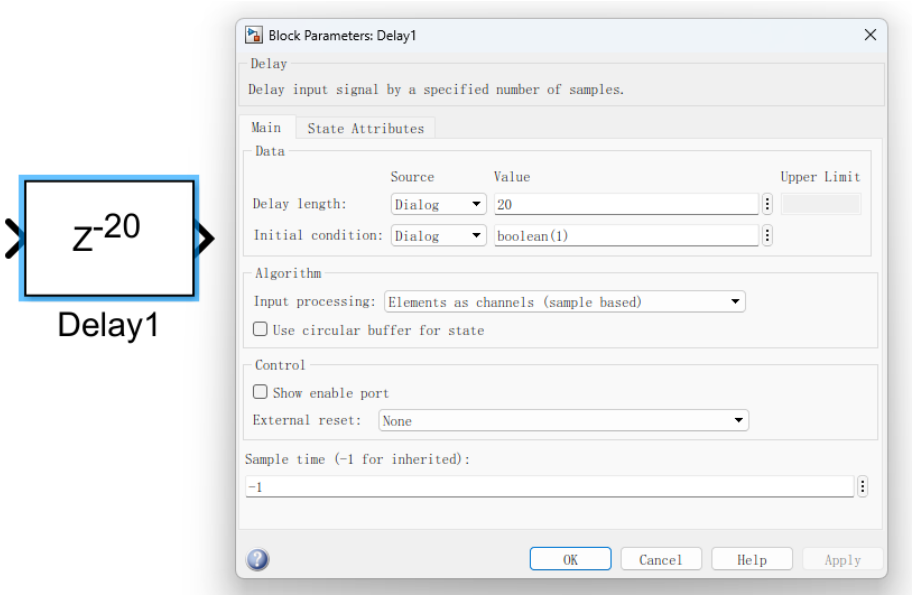
离散和连续延迟：Delay 模块可以用于引入离散或连续的延迟，具体取决于信号的采样时间。对于连续信号，Delay 模块会根据模拟仿真的步长来引入延迟；对于离散信号，Delay 模块会根据离散采样时间来引入延迟。

参数设置：除了延迟时间外，Delay 模块还可以设置其他参数，例如初始化延迟、延迟类型（单位延迟或整数延迟）等。这些参数可以根据具体需求进行调整，以满足系统建模和仿真的要求。

缓冲：在引入延迟的同时，Delay 模块还会在内部使用缓冲来存储延迟的输入信号。这确保了延迟信号的稳定性和正确性，避免了信号数据的丢失或损坏。

通过 Delay 模块，你可以在 Simulink 中方便地引入固定或可变的延迟，用于模拟系统中的信号传播延迟、处理器执行延迟等情况，从而更准确地建模和仿真系统的行为。

本例程中，引用了 20 个单位延迟，表示信号将被延迟 20s 的时间后输出。



Data Type Conversion—数据类型转换

如下图所示，Data Type Conversion（数据类型转换）模块用于将输入信号的数据类型转换为所需的输出数据类型。这个模块在处理不同数据类型之间的转换时非常有用，例如将整数转换为浮点数，或者将固定点数转换为双精度浮点数等。

主要特点和用法包括：

数据类型转换：Data Type Conversion 模块可以将输入信号的数据类型转换为指定的输出数据类型。常见的数据类型包括整数、浮点数、固定点数等。

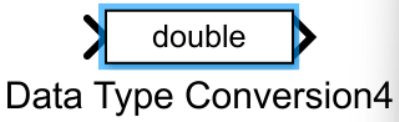
精度控制：通过 Data Type Conversion 模块，可以控制信号的精度，例如从高精度转换到低精度，或者从低精度转换到高精度。这对于在系统中平衡性能和计算成本非常有用。

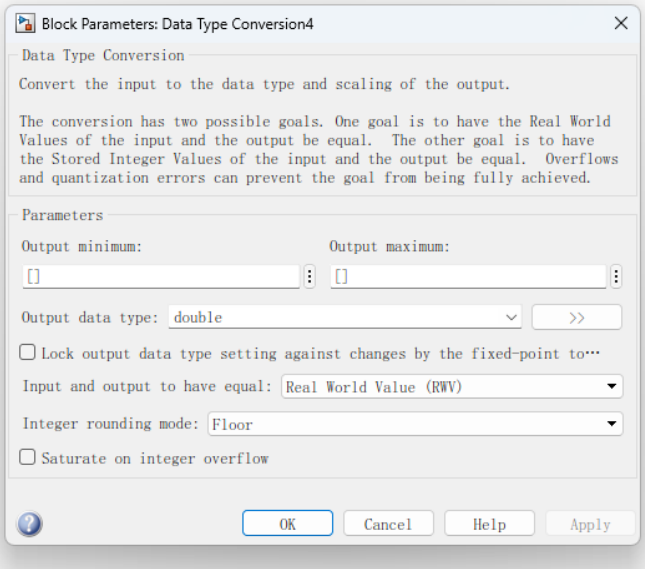
饱和和截断：在进行数据类型转换时，可以选择是否进行饱和或截断操作。饱和操作会将超出目标数据类型范围的值限制在目标范围内，而截断操作则会直接舍弃超出目标数据类型范围的部分。

舍入模式：在进行浮点数或固定点数转换时，可以选择不同的舍入模式，包括向上舍入、向下舍入、向零舍入等。这有助于控制在转换过程中的舍入误差。

输出端口数目：Data Type Conversion 模块可以有一个或多个输出端口，取决于输入信号的数量和转换类型。你可以根据需要选择合适的输出端口。

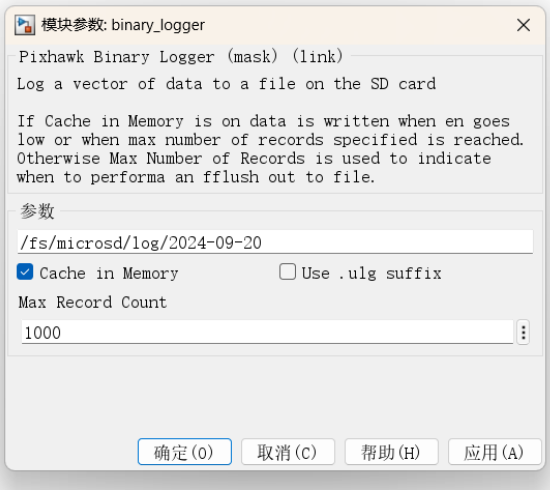
通过 Data Type Conversion 模块，可以灵活地控制信号的数据类型和精度，以满足系统建模和仿真的需求。这在处理不同数据类型的信号、接口以及在不同精度下进行计算时非常有用。





Binary_logger—数据记录模块

如下图所示，这个模块将数据记录为 double 型.bin 或.ulg 格式文件存储到 SD 卡中，如果在内存中缓存打开，则当 en 变低或达到指定的最大记录数时写入数据。在程序执行过程中第一个使能输入信号必须先触发高电平再降为低电平才能成功地记录数据，如果在代码结束运行之前没有降为低电平，那么记录文件将被认为处于打开状态，不可访问。此外，日志必须存储在目录“/fs/microsd/”下。并且可以设置日志存储路径及最大记录条数。



Signal Specification—信号规范

如下图所示，在 Simulink 中，Signal Specification（信号规范）模块用于指定信号的特定属性，如数据类型、尺寸和采样时间等。这个模块通常用于定义输入信号的属性，以确保与其他模块的输入端口匹配。

Signal Specification 模块的主要特点和用法包括：

数据类型指定：可以通过 Signal Specification 模块指定输入信号的数据类型，例如整数、浮点数、布尔值等。这有助于确保信号类型与系统中其他模块的要求相匹配。

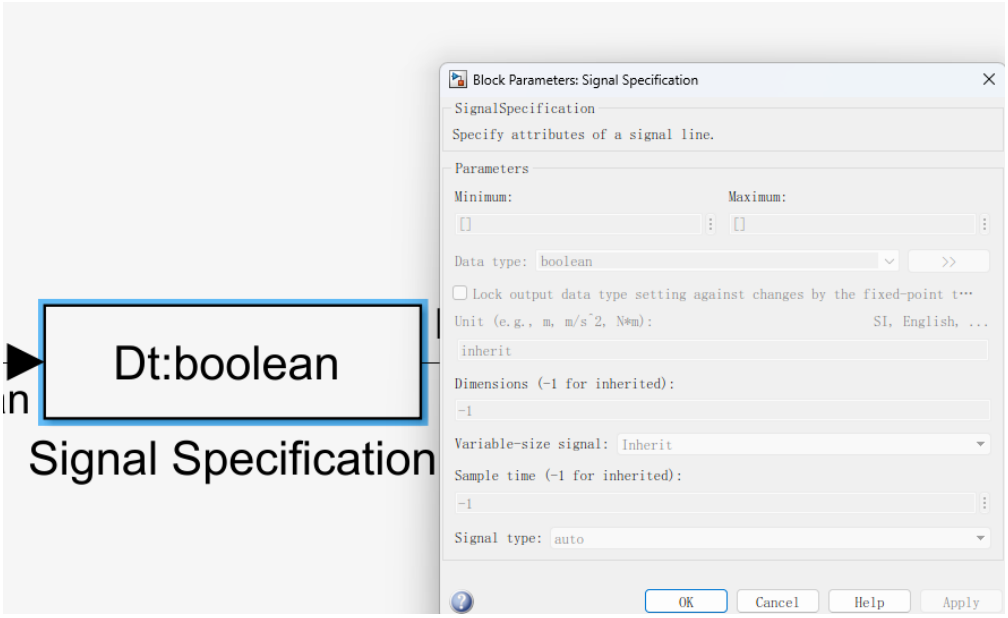
信号尺寸指定：可以定义输入信号的尺寸，包括向量或矩阵的维度。这允许在模型中使用多维信号，并确保与其他模块的输入端口匹配。

采样时间指定：可以设置输入信号的采样时间，以指定信号的更新频率。这对于模拟连续系统时非常有用，可以确保模拟器按照指定的时间步长执行仿真。

单位指定：可以指定信号的单位，例如长度单位、时间单位等。这有助于确保模型中的物理量符合预期的单位标准。

数据范围指定：有时也可以指定输入信号的数据范围，例如最小值和最大值。这对于确保输入信号的值在可接受范围内非常有用。

Signal Specification 模块通常用于模型的输入端口，以确保输入信号的属性与系统中其他模块的要求相匹配。通过指定信号的数据类型、尺寸、采样时间等属性，可以有效地管理模型中的信号流，并确保模型的准确性和一致性。



其他模块

如下图所示，更多 Simulink 官方模块的详细用法和功能，请到 Simulink 界面的库浏览器中搜索，找到对应的模块，双击该模块，点“帮助”即可查看该模块的详细用法和功能。

2. 实验效果

实现飞行日志的写入与读取。

3. 文件目录

文件夹/文件名称	说明
pixhawk_A.bin	*.bin 格式日志文件。
log_0_2024-9-23-19-34-50.ulg	*.ulg 格式日志文件。
px4demo_log.slx	Simulink 飞行日志写入*.bin 格式日志模型。
px4demo_log_qgc.slx	Simulink 飞行日志写入*.ulg 格式日志模型。

4. 运行环境

序号	软件要求	硬件要求	
		名称	数量
1	Windows 10 及以上版本	笔记本/台式电脑①	1
2	RflySim 工具链	Pixhawk 6x 或 Pixhawk 6x mini②	1
3		遥控器③	1
4		遥控器接收器	1
5		数据线、杜邦线等	若干

① ：推荐配置请见：https://doc.rflysim.com

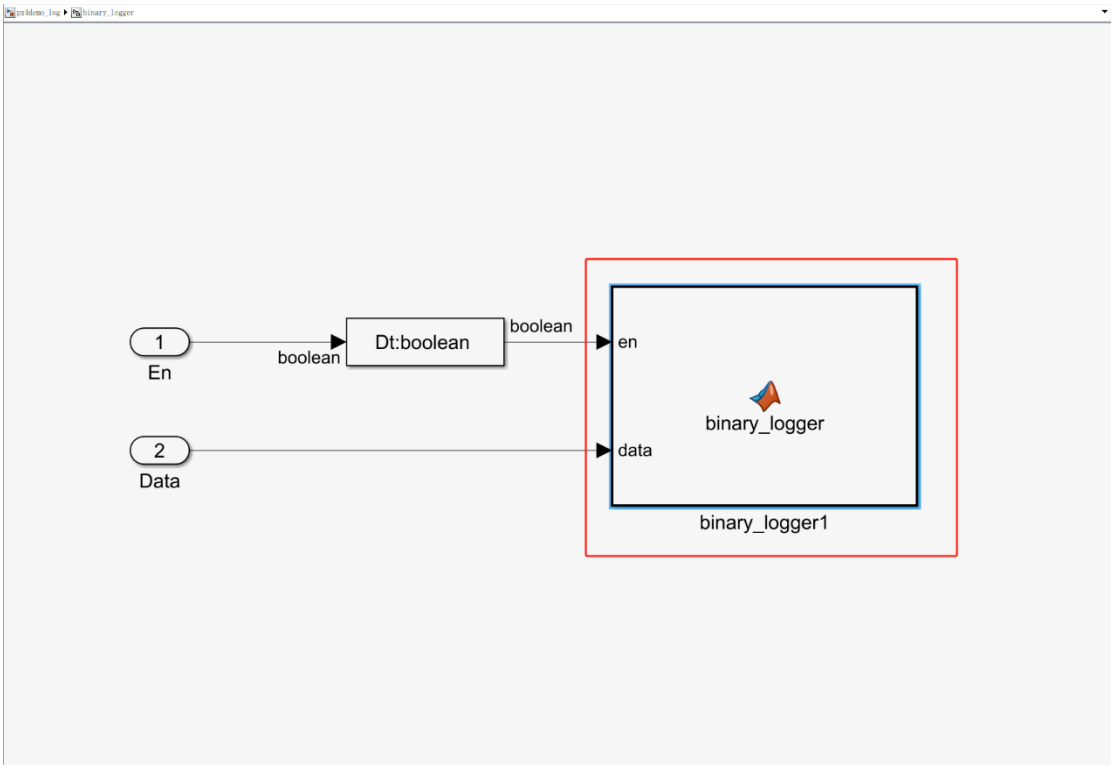
② ：须保证平台安装时的编译命令为：px4_fmu-v6x_default，固件版本为：1.12.3。其他配套飞控请见：http://rflysim.com

③：本实验演示所使用的遥控器为：天地飞 ET10、配套接收器为：WFLY RF209S。遥控器相关配置见： https://rflysim.com/doc/zh/B/3.1ET10.html

5. 实验步骤

5.1. Binary_logger—数据记录模块-实验原理

在 MATLAB 中打开 px4demo_log.slx 文件，点击 binary_logger 查看其内部封装模块。然后点击 binary_logger 即可查看该模块的函数代码。



代码及解析如下：

函数 binary_logger 接受五个输入参数：en 表示是否启用记录器，data 是要记录的数据，logname 是日志文件的名称，cacheflag 表示是否缓存数据，numRecs 是要记录的记录数。

```
function binary_logger(en, data, logname, cacheflag, numRecs)
```

函数中定义了一些持久变量（persistent）来保存状态信息，如文件指针 fp、记录计数器 count、文件是否已打开 isopen 等。

```
    persistent fp isflushed isopen count lognumstr elem_size vect_size datastore
    coder.cinclude('<stdio.h>');
    coder.cinclude('<time.h>');

    if isempty(count)
        count = uint32(0);
        lognumstr = 65;
        isopen = false;
        fp = coder.opaque('FILE*', 'NULL');
        isflushed = false;
        elem_size = uint16(1);
        vect_size = coder.const(uint8(numel(data)));
        if cacheflag == true
            datastore = zeros(numRecs*uint32(coder.const(numel(data))), 1, 'like', data);
        end
```

```
        end

        if (en)
            if isopen == false
                [elsize, eltype] = get_elem_size(data);
                elem_size = coder.const(uint16(elsize));
                elem_type = coder.const(uint8(eltype));
                vect_size = coder.const(uint8(numel(data)));
                coder.ceval('printf',['OPENING file %s vect_size %d elem_size %d numRecs %d %
c' uint8(0)], ...
                    [logname, '_', char(lognumstr), '.bin', int8(0)], uint32(vect_size), uint
32(elem_size), uint32(numRecs), uint8(10));
                fp = coder.ceval('fopen', [logname, '_', char(lognumstr), '.bin', int8(0)], ['
wb', int8(0)]);
                lognumstr = lognumstr + 1;
                isflushed = false;
                count = uint32(0);
                isopen = true;
                % Write header information here
                % header token + version
                mlStrVer = 'MWLOGV01';
                coder.ceval('fwrite', coder.rref(mlStrVer), uint32(1), length(mlStrVer), fp);
                % date/time
                tm = uint32(0);
                tmptr = uint32(0);
                tm = coder.ceval('time',coder.wref(tmptr));
                coder.ceval('fwrite', coder.rref(tm), uint32(4), uint32(1), fp);
                % number of elements in each record
                coder.ceval('fwrite', coder.rref(vect_size), uint32(1), uint32(1), fp);
                % data type of the elements
                coder.ceval('fwrite', coder.rref(elem_type), uint32(1), uint32(1), fp);
                % size of each record in bytes
                recSize = elem_size * uint16(vect_size);
                coder.ceval('fwrite', coder.rref(recSize), uint32(2), uint32(1), fp);
            end
            % cache data in memory or use fwrite
            if cacheflag == true
                addr = (count*uint32(vect_size))+1;
                for idx = 1:vect_size
                    datastore(addr) = data(idx);
                    addr = addr + 1;
                end
            else
                coder.ceval('fwrite', coder.rref(data), elem_size, vect_size, fp);
            end
            count = count + uint32(1);
            % check for data count limit
            if count >= numRecs
                if cacheflag
                    for idx = 1:count*uint32(vect_size)
                        coder.ceval('fwrite', coder.rref(datastore(idx)), elem_size, uint32
(1), fp);
                    end
                end
                coder.ceval('fflush', fp);
```

```
                isflushed = false;
                count = uint32(0);
            end
        else
            if isopen == true
                if ~isflushed
                    coder.ceval('printf',['CLOSING file. count = %d %c' uint8(0)], uint32(cou
nt), uint8(10));
                    if cacheflag
                        count = count + uint32(1);
                        for idx = 1:count*uint32(vect_size)
                            coder.ceval('fwrite', coder.rref(datastore(idx)), elem_size, uint
32(1), fp);
                        end
                    end
                    coder.ceval('fflush', fp);
                    isflushed = true;
                end
                coder.ceval('fclose', fp);
            end
            isopen = false;
            count = uint32(0);
        end
    end

    % return the size of the element in bytes
```

这个函数 get_elem_size 根据输入数据的类型返回相应的数据元素大小和类型编码。对于不同的数据类型，分别设置对应的元素大小 elem_size 和类型编码 dtype。

帮助 binary_logger 函数确定输入数据的大小和类型，以便正确地将数据写入文件。

```
    function [elem_size, dtype] = get_elem_size(data)
```

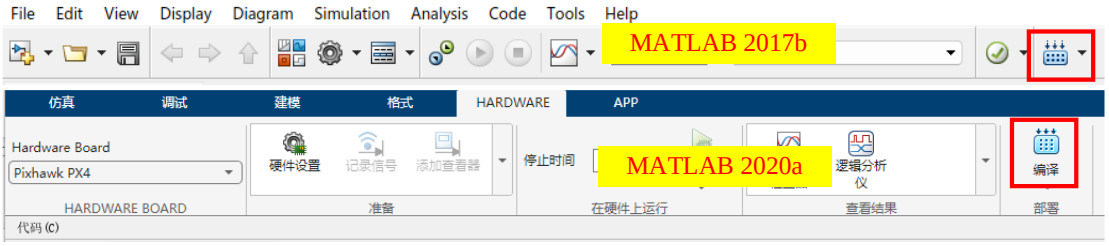
使用 switch 语句根据输入数据的类型（使用 class(data) 获取）进行不同情况的处理。

```
        switch(class(data))
            case 'double'
                elem_size = 8;
                dtype = 1;
            case 'single'
                elem_size = 4;
                dtype = 2;
            case 'int32'
                elem_size = 4;
                dtype = 3;
            case 'uint32'
                elem_size = 4;
                dtype = 4;
            case 'int16'
                elem_size = 2;
                dtype = 5;
            case 'uint16'
                elem_size = 2;
                dtype = 6;
            case 'int8'
                elem_size = 1;
                dtype = 7;
```

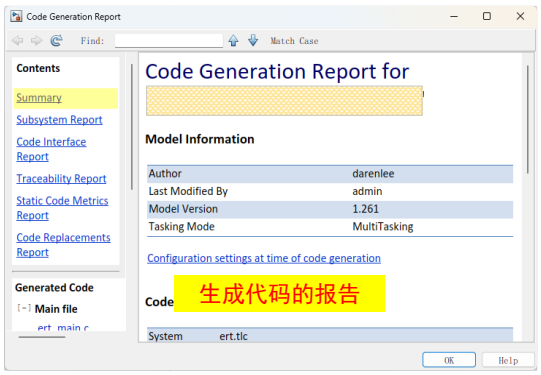
```
            case 'uint8'
                elem_size = 1;
                dtype = 8;
            case 'logical'
                elem_size = 1;
                dtype = 9;
            case 'embedded.fi'
                nt = numerictype(data);
                elem_size = 2^ceil(log2(ceil(nt.WordLength / 8)));
                dtype = 10;
        end
        if ~isreal(data)
            elem_size = elem_size*2;
        end

        elem_size = uint16(elem_size);
        dtype = uint8(dtype);
    end
```

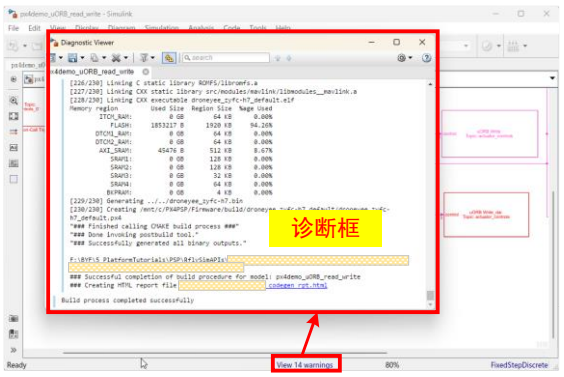
5.2. *.bin 格式日志数据记录与读取实验

打开 MATLAB 软件，在 MATLAB 中打开 px4demo_log.slx 文件，在 Simulink 中，点击编译命令。

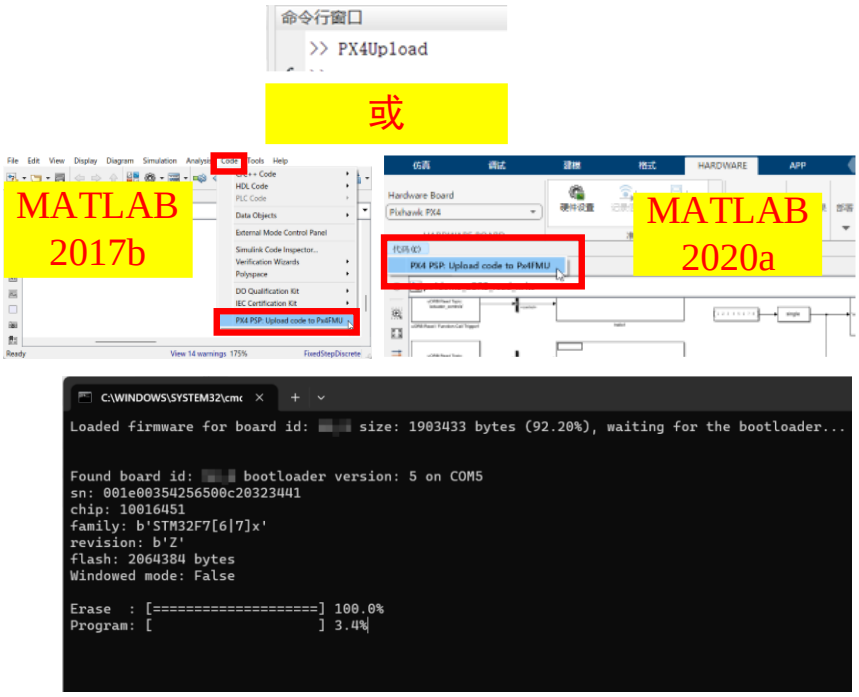


在 Simulink 的下方点击 View diagnostics 指令，即可弹出诊断对话框，可查看编译过程。在诊断框中弹出 Build process completed successfully，即可表示编译成功，左图为生成的编译报告。

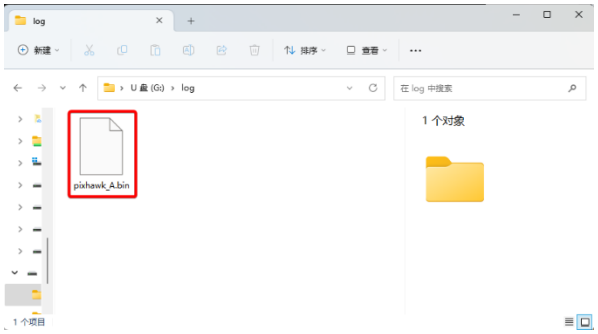




用 USB 数据线链接飞控与电脑。在 MATLAB 命令行窗口输入：PX4Upload 并运行或点击 PX4 PSP：Upload code to Px4FMU，弹出 CMD 对话框，显示正在上传固件至飞控中，等待上传成功。



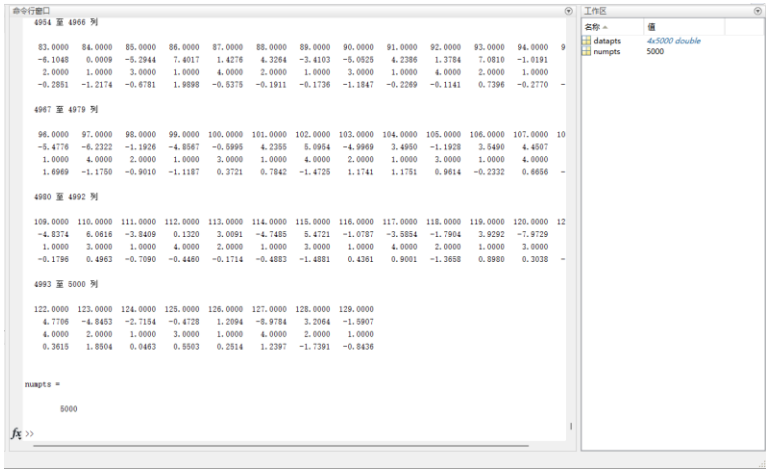
上传成功后，等待 20s 之后取出飞控中的 SD 卡，插入电脑后，在 SD 卡文件中找到 log 文件夹，复制 pixhawk_A.bin 文件，到实验的文件夹下，可修改文件名避免冲突！



在 MATLAB 命令行中输入如下程序：

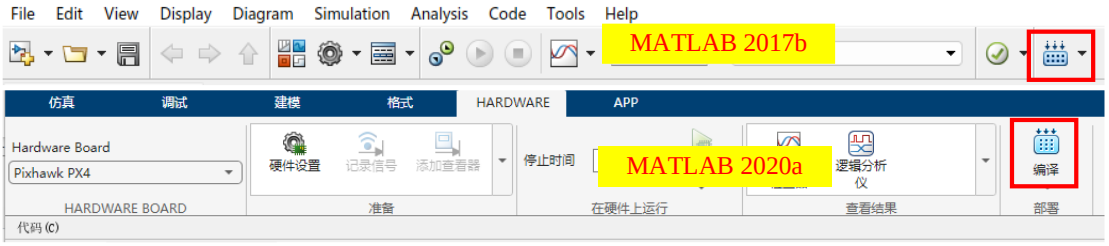
```
PX4ReadLog('pixhawk_A.bin')
或
PX4ReadLog 'pixhawk_A.bin'
```

运行后，datapts 为记录的 4*5000 矩阵数据，同时也将全部打印命令行中，numpts 为采集到的数据点数量。

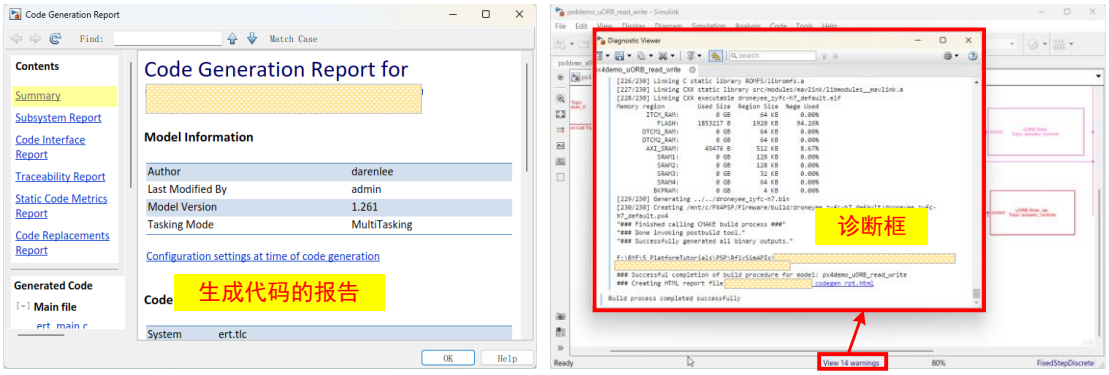


5.3. *.ulg 格式日志数据记录与读取实验

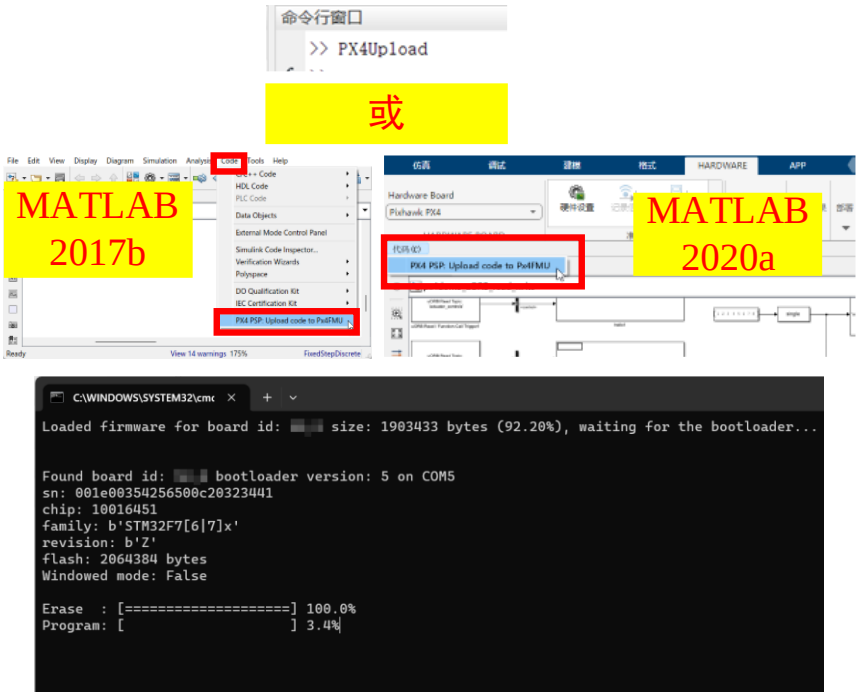
打开 MATLAB 软件，在 MATLAB 中打开 px4demo_log_qgc.slx 文件，在 Simulink 中，点击编译命令。



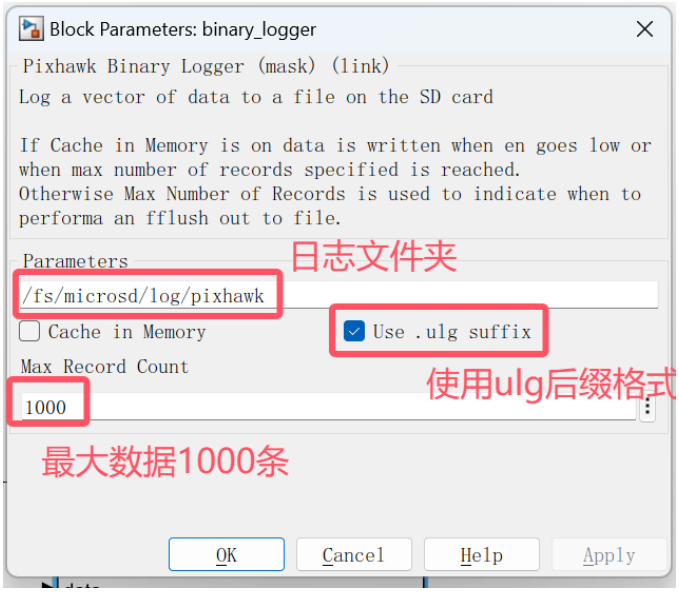
在 Simulink 的下方点击 View diagnostics 指令，即可弹出诊断对话框，可查看编译过程。在诊断框中弹出 Build process completed successfully，即可表示编译成功，左图为生成的编译报告。



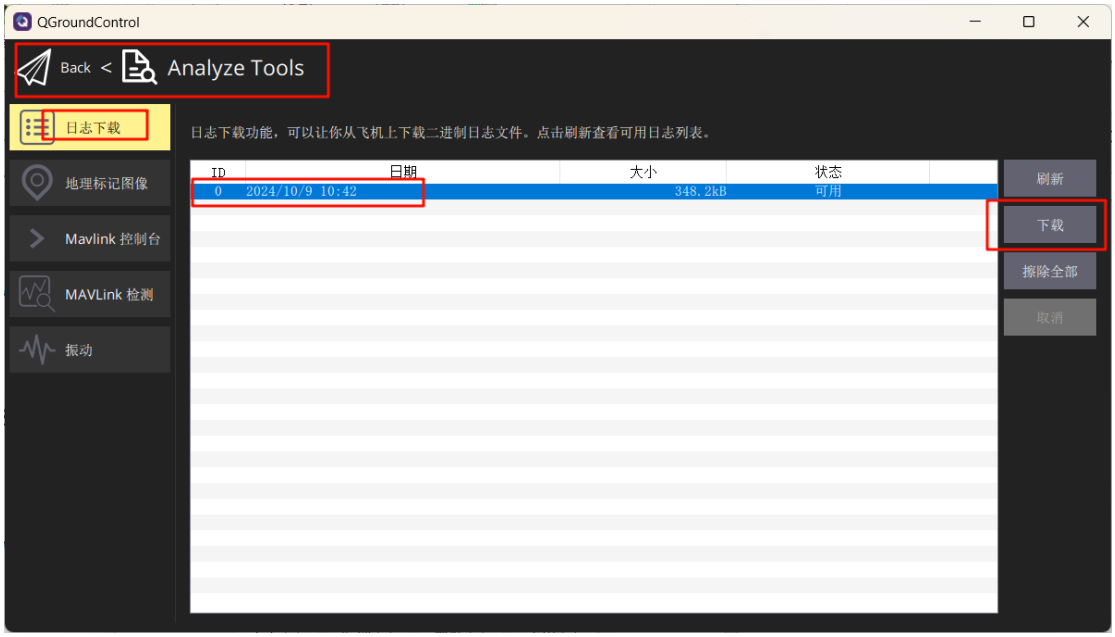
用 USB 数据线链接飞控与电脑。在 MATLAB 命令行窗口输入：PX4Upload 并运行或点击 PX4 PSP：Upload code to Px4FMU，弹出 CMD 对话框，显示正在上传固件至飞控中，等待上传成功。



该例程中相比于 px4demo_log.slx 程序，自定义设置了 binary_logger 模块记录的数据为.ulg 格式文件，用户可通过 QGC 地面站下载该文件。

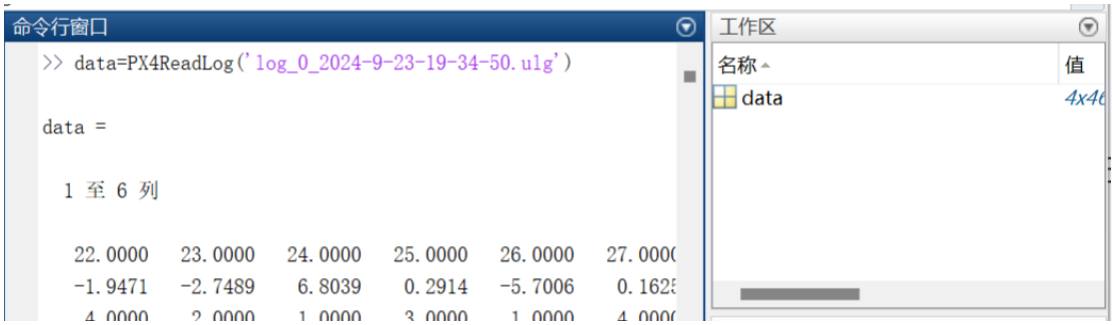


模型中定义了通过遥控器的 CH5 通道使能开始记录日志，在拨动 CH5 通道后等待 20s 之后，再次拨动 CH5 通道到低电位。即可完成数据的记录。然后打开 QGC 地面站点击左上角“Q”字样 Logo—>“Analyze Tools”—>“日志下载”—>“下载”即可下载该日志文件。



在 MATLAB 命令行中输入如下程序：

```
PX4ReadLog('log_0_2024-9-23-19-34-50.ulg')
或
PX4ReadLog 'log_0_2024-9-23-19-34-50.ulg'
```



即可将记录的数据加载到 MATLAB 的工作区中。

6. 参考资料

[1] 无。

7. 常见问题

Q1：***

A1：***