

# **PX4 上层源码架构分析及双系统飞控算法开发 指南**

**2023 年 06 月于中山大学·深圳**

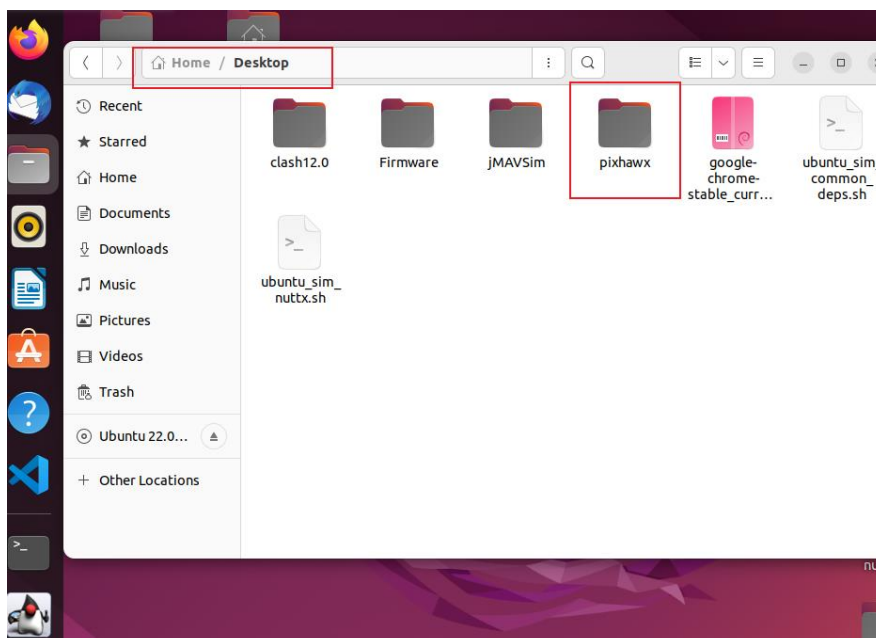
**BY REZ**

## 内 容 概 要

- 1) 梳理 PX4 开源飞控上层源码目录结构;
- 2) 编译及修改 PX4 源码步骤;
- 3) PX4 源码移植梳理
- 4) Windows 系统下基于 MATLAB/Simulink 硬件支持包的 PX4 飞控算法自主实践步骤
- 5) PX4 底层结构系统梳理

文件编号		页码	1	版次	A
目 录					
1	PX4 源码的下载及编译上传操作.....	1			
2	PX4-AUTOPILOT 文件夹下的子目录架构 .....	12			
3	PX4 源码中的传感器接口 .....	15			
3.1	传感器相关程序的路径位置 .....	15			
3.2	传感器相关程序的功能说明 .....	16			
4	PX4 源码中不同模块间的 UORB 通信 .....	17			
4.1	UORB 通信框架.....	17			
4.2	不同模块通过 UORB 通信的案例.....	18			
5	PX4 源码移植至另一台 STM32 硬件系统 .....	19			
5.1	修改引脚配置示例 .....	20			
5.2	编译配置（MAKEFILE 和 CMAKELISTS.TXT） .....	22			
5.3	PX4 的 BOOTLOADER 示例 .....	24			
6	WINDOWS 系统下 PX4 飞控算法自主开发实践 .....	26			
6.1	依赖的软硬件材料（以 PIXHAWK4 飞控为示例） .....	26			
6.2	安装及配置步骤 .....	29			
7	PX4 底层结构梳理.....	34			
7.1	MAVLink 通信和 QGC 中的 UDP 无线数传通信的区别 .....	35			
7.2	PX4 固件内部 UORB 通信和跨不同物理平台间 MAVLink 通信协议区别 .....	36			
7.3	NUTTX 嵌入式操作系统中的硬件抽象层总结(打开移植思路).....	37			
7.4	编译（MAKEFILE 和 CMAKELISTS.TXT） .....	38			

文件编号		页码 1	版次 A
<div><h1>1 PX4 源码的下载及编译上传操作</h1><p>引言：PX4 是一个开源的自动驾驶系统，它的源代码托管在 GitHub 上。在 GitHub 上，有一些仓库的名称可能会引起一些混淆，这包括 Firmware 和 PX4-Autopilot。Firmware 仓库是 PX4 的主要代码库，包含了所有的核心功能。PX4-Autopilot 仓库是一个辅助仓库，包含了一些额外的工具和应用程序，使得用户和开发者更容易地使用和开发 PX4 系统。</p><p>参考链接：</p><p><a href="#">PX4 固件 CSDN 编译步骤</a></p><p><a href="#">PX4 编译原生固件和编译到虚拟仿真器</a></p><p>以 Pixhawk2.4.8 和 Pixhawk4 两款飞控编译 PX4-Autopilot 固件，需要分别克隆不同的代码库并选择相应的编译选项和配置文件。以下是针对 Pixhawk2.4.8 和 Pixhawk4 飞控源码的克隆及编译操作：</p><p>PX4 源码可以从其官方 GitHub 仓库下载，下载的方式有以下几种：</p><p>1) Github 网站</p><p><a href="#">PX4/PX4-Autopilot: PX4 Autopilot Software (github.com)</a></p><p>从仓库页面上，可以下载最新版本的源代码。选择“Clone or download”按钮，然后选择“Download ZIP”选项即可下载压缩文件。</p><p>2) Git 命令克隆仓库(Ubuntu 系统)</p><p>2.1) 新建文件夹，用于存放 PX4 源码，比如在 home/Desktop 中新建 ‘pixhawk/source’ 文件夹，然后终端中 cd 进入该目录开始 clone 源码固件：</p><pre>cd pixhawk/source</pre></div>			



2.1) 使用 Git 命令行工具克隆 PX4 的源代码仓库。在终端中，切换到您希望克隆源代码的目录，并执行以下命令：

```
git clone https://github.com/PX4/Firmware.git
```

在执行上述命令后，Git 会自动从 GitHub 上下载 PX4 源码，并保存到当前工作目录下的 `pixhawk/source` 目录中。其中，`https://github.com/PX4/Firmware.git` 表示要克隆的 PX4 源码的 Git 仓库地址，`--recursive` 表示同时克隆 PX4 子模块。

注：使用 Git 克隆 PX4 源码一般需要较好的网络环境，普通网络会导致下载失败或者下载不完全问题！（这里网络存在不确定性）

## 2.2) 切换到特定分支：(慎重切换！)

如果需要下载特定分支的 PX4 源码，可以使用 `git checkout` 命令来切换分支。例如，要下载 v1.12.0 版本的 PX4 源码，可以在终端中输入以下命令：

%%进入 PX4 源码的下载目录

```
git checkout v1.10.2
```

 %%切换到 v1.10.2 分支

2.3) 到 Firmware 目录中更新子模块：(下载后可不执行这条)

```
cd Firmware
```

```
git submodule update --init --recursive
```

2.4) 修改权限（非必须，暂不执行也无大的影响）

```
sudo usermod -a -G dialout $USER
```

该命令的作用是将当前用户（\$USER）添加到"dialout"用户组中。通过将用户添

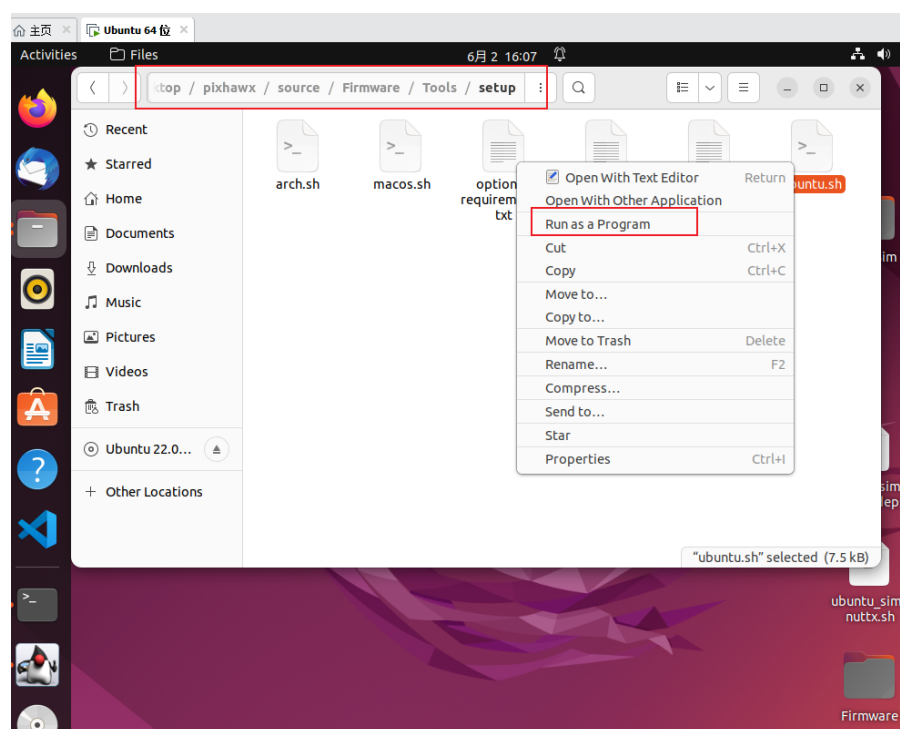
加到"dialout"组，该用户将获得对串行设备（如串口）的访问权限。

在 Linux 系统中，串行设备通常位于"/dev"目录下，而访问这些设备可能需要特定的权限。通过将用户添加到"dialout"组，用户可以获得读取和写入串行设备的权限，而无需使用 root 权限或 sudo 命令。

在某些情况下，例如使用串口与硬件设备进行通信或连接到无人机飞控系统，添加用户到"dialout"组可能是必需的。这样可以确保用户具有适当的权限来与串行设备进行通信。

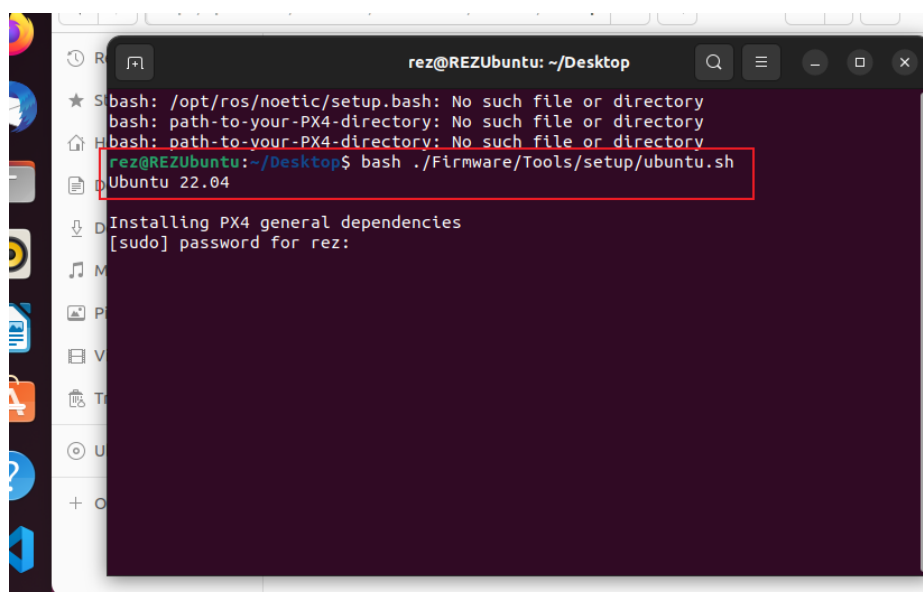
2.5) 构建 PX4 固件编译给 jmavsim 虚拟飞行仿真器前所需的依赖环境（很重要！）

打开源码所在目录下的执行脚本（推荐这种操作），如下图：



也可以在桌面打开终端输入以下命令：

```
bash ./Firmware/Tools/setup/ubuntu.sh
```

A terminal window titled 'rez@REZUbuntu: ~/Desktop' showing the execution of the 'ubuntu.sh' script. The script output includes error messages for missing files, the Ubuntu version '22.04', and the start of 'Installing PX4 general dependencies'. A red box highlights the command 'rez@REZUbuntu:~/Desktop\$ bash ./Firmware/Tools/setup/ubuntu.sh' and the first error message.

```
rez@REZUbuntu: ~/Desktop
$ bash: /opt/ros/noetic/setup.bash: No such file or directory
$ bash: path-to-your-PX4-directory: No such file or directory
$ bash: path-to-your-PX4-directory: No such file or directory
rez@REZUbuntu:~/Desktop$ bash ./Firmware/Tools/setup/ubuntu.sh
Ubuntu 22.04
Installing PX4 general dependencies
[sudo] password for rez:
```

通过执行"`bash ./Firmware/Tools/setup/ubuntu.sh`"命令，可以自动执行这些准备工作，以确保自己的 Ubuntu 系统已正确配置和准备好构建 PX4 固件。具体而言，"`ubuntu.sh`"脚本会执行以下操作：

- 安装所需的依赖项：脚本将检查并安装构建 PX4 固件所需的各种软件包和依赖项。这些依赖项可能包括编译工具链、库文件、构建工具等。
- 配置环境：脚本会设置一些环境变量和系统配置，以确保构建和运行 PX4 固件的顺利进行。这可能包括设置路径、环境变量和其他相关设置。
- 更新子模块：PX4 固件包含多个 Git 子模块，这些子模块存储了额外的源代码和资源。脚本会更新这些子模块，以获取最新的代码和资源。

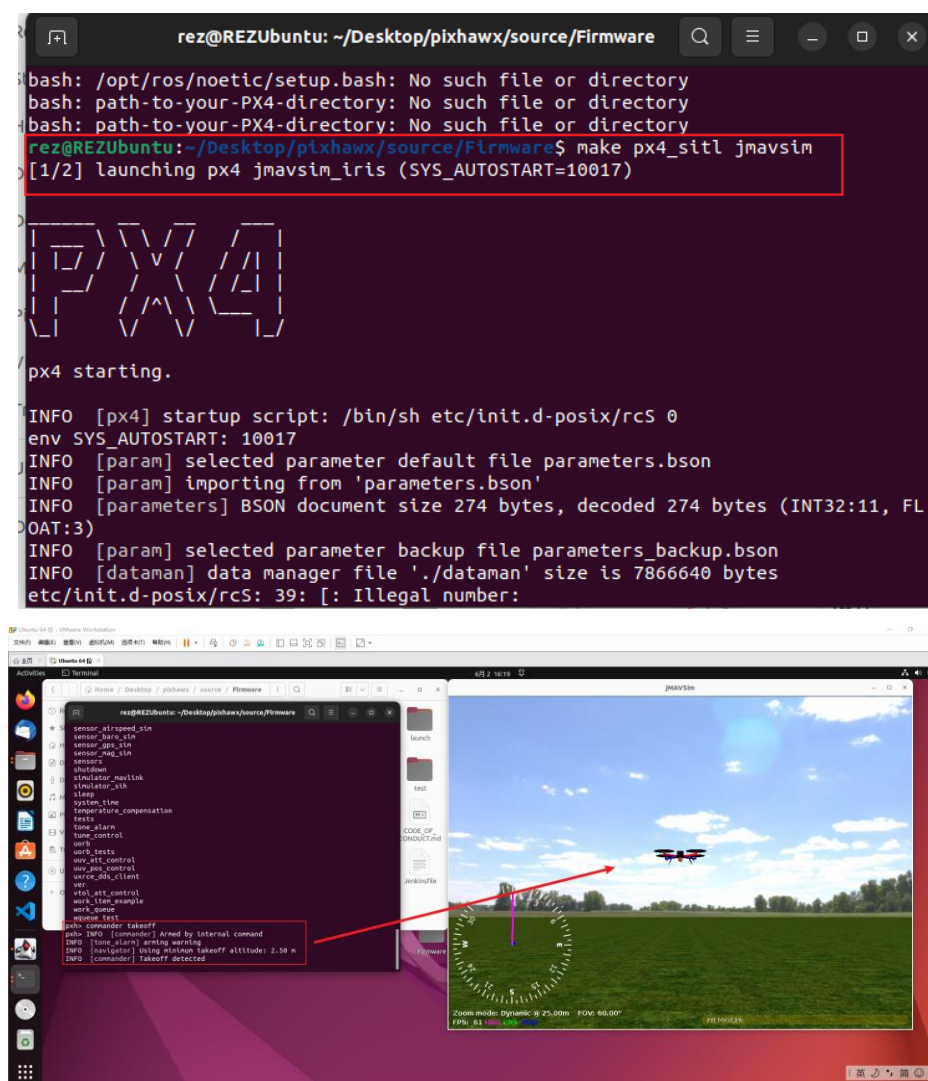
2.6) 卸载模式管理器：（未使用,待验证，暂时别执行）

```
sudo apt-get remove modemmanager
```

2.7) 编译构建 jmavsim 仿真器

```
make px4_sitl jmavsim
```

通过执行 `make px4_sitl jmavsim`，将会构建 **PX4 SITL 仿真环境**以及 **JMavSim 仿真器**，并在计算机上启动仿真环境。这样您就可以在仿真环境中测试飞控算法、进行飞行模拟以及开发和调试飞行控制软件。



```
rez@REZUbuntu: ~/Desktop/px4hax/source/Firmware
bash: /opt/ros/noetic/setup.bash: No such file or directory
bash: path-to-your-PX4-directory: No such file or directory
bash: path-to-your-PX4-directory: No such file or directory
rez@REZUbuntu:~/Desktop/px4hax/source/Firmware$ make px4_sitl jmavsim
[1/2] launching px4 jmavsim_iris (SYS_AUTOSTART=10017)

px4 starting.

INFO [px4] startup script: /bin/sh etc/init.d-posix/rcs 0
env SYS_AUTOSTART: 10017
INFO [param] selected parameter default file parameters.bson
INFO [param] importing from 'parameters.bson'
INFO [parameters] BSON document size 274 bytes, decoded 274 bytes (INT32:11, FL
OAT:3)
INFO [param] selected parameter backup file parameters_backup.bson
INFO [dataman] data manager file './dataman' size is 7866640 bytes
etc/init.d-posix/rcs: 39: [: Illegal number:

px4 starting.

INFO [px4] startup script: /bin/sh etc/init.d-posix/rcs 0
env SYS_AUTOSTART: 10017
INFO [param] selected parameter default file parameters.bson
INFO [param] importing from 'parameters.bson'
INFO [parameters] BSON document size 274 bytes, decoded 274 bytes (INT32:11, FL
OAT:3)
INFO [param] selected parameter backup file parameters_backup.bson
INFO [dataman] data manager file './dataman' size is 7866640 bytes
etc/init.d-posix/rcs: 39: [: Illegal number:
```

在 Ubuntu 系统下的 JMAVSIM 与 QGC 的联合软件在环仿真：

### [B 站参考网址链接](#)

第一次在 ubuntu 系统上安装 QGC 可按以下步骤进行：

在 Ubuntu 终端中运行以下命令以更新软件包列表并安

装依赖项：

```
sudo apt-get update
```

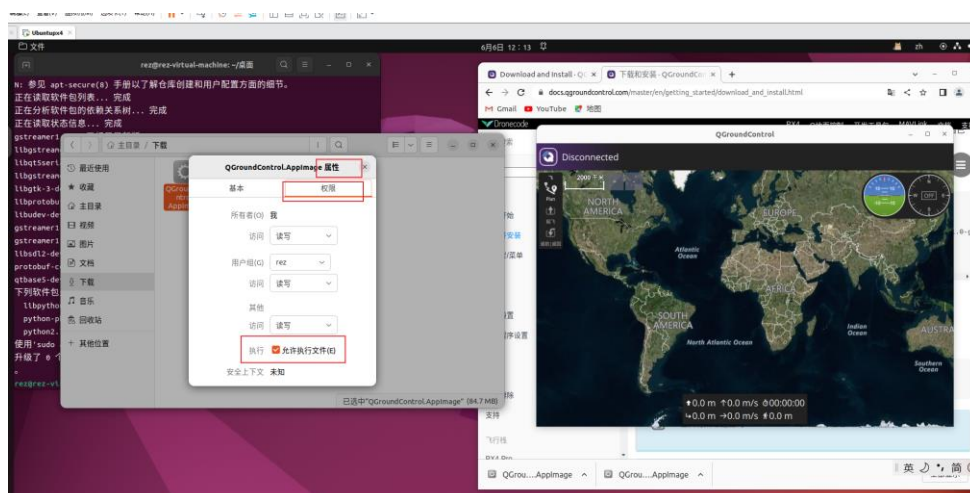
```
sudo apt-get install gstreamer1.0-plugins-bad gstreamer1.0-libav gstreamer1.0-gl
libgstreamer-plugins-base1.0-dev libgstreamer-plugins-good1.0-dev libgtk-3-dev libudev-dev
libsdl2-dev libprotobuf-dev protobuf-compiler libqt5serialport5-dev qtbase5-dev
```

在 QGC 官网上下载适用于 Ubuntu 的 QGC 安装包：

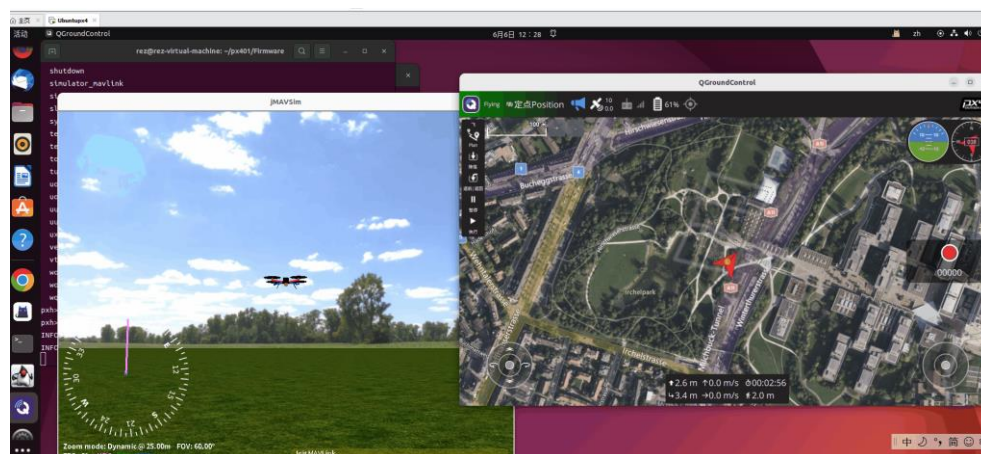
### [QGC 下载配置网址](#)

下载好 QGC 安装包后打开时无法打开，原因是没有打开【属性】中的【权限】设置，如下图：



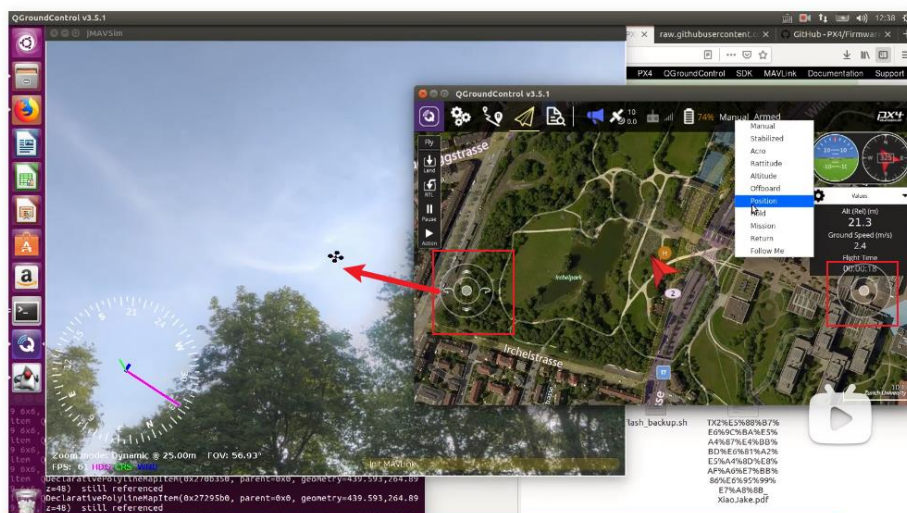


联合纯软件在环仿真如下：



开源PX4环境编译过程演示（以1.8.2固件为例）

5292 4 2019-05-28 15:14:32 未经授权，禁止转载

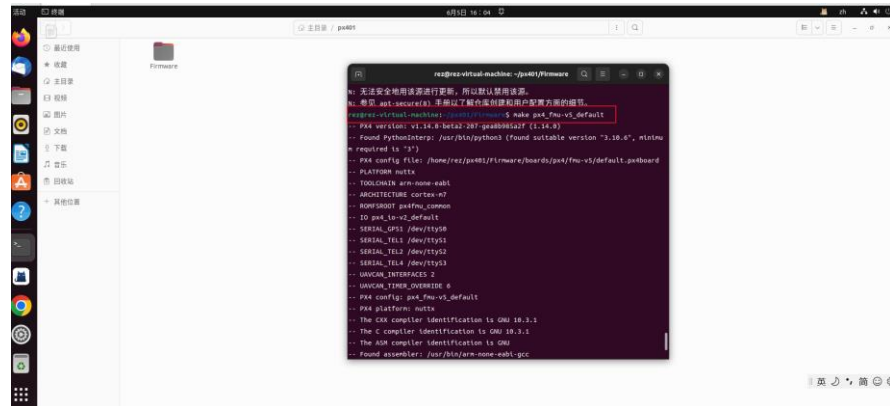


## 2.7) 编译 PX4:

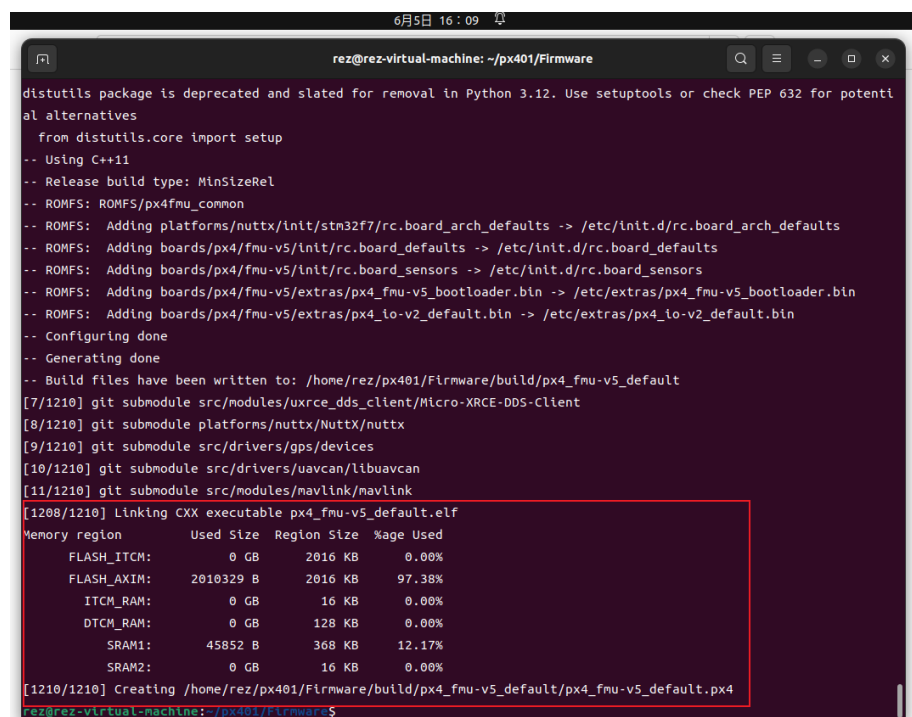
比如打开 终端，切换到 Firmware 目录，并执行以下命令进行编译，以生成适用于 Pixhawk4 飞控的固件：

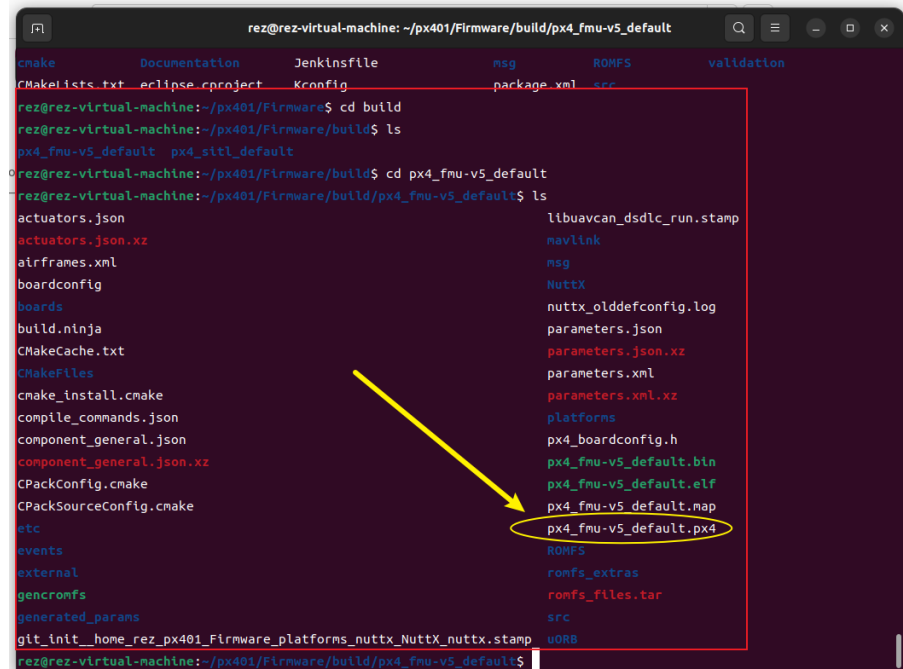
使用 make 命令编译前需要先执行几个命令，安装编译器环境并更新：

- 1) `sudo apt-get install gcc-arm-none-eabi`
- 2) `export PATH=$PATH:/path/to/gcc-arm-none-eabi/bin`
- 3) `sudo apt-get update`
- 4) `make px4_fmu-v5_default`



如果编译成功了，在 build 文件夹下会生成可执行文件，就可将生成的可执行文件上传到 PX4 飞控。





```
rez@rez-virtual-machine: ~/px401/Firmware/build/px4_fmu-v5_default
cmake Documentation Jenkinsfile msg ROMFS validation
CMakeLists.txt eclipse_cproject Kconfig package.xml src
rez@rez-virtual-machine:~/px401/Firmware$ cd build
rez@rez-virtual-machine:~/px401/Firmware/build$ ls
px4_fmu-v5_default px4_sitl_default
rez@rez-virtual-machine:~/px401/Firmware/build$ cd px4_fmu-v5_default
rez@rez-virtual-machine:~/px401/Firmware/build/px4_fmu-v5_default$ ls
actuators.json libuavcan_dsdlc_run.stamp
actuators.json.xz navlink
airframes.xml msg
boardconfig NuttX
boards nuttx_olddefconfig.log
build.ninja parameters.json
CMakeCache.txt parameters.json.xz
CMakeFiles parameters.xml
cmake_install.cmake parameters.xml.xz
compile_commands.json platforms
component_general.json px4_boardconfig.h
component_general.json.xz px4_fmu-v5_default.bin
CPackConfig.cmake px4_fmu-v5_default.elf
CPackSourceConfig.cmake px4_fmu-v5_default.map
etc px4_fmu-v5_default.px4
events ROMFS
external romfs_extras
gendromfs romfs_files.tar
generated_params src
git_init_home_rez_px401_Firmware_platforms_nuttX_NuttX_nuttX.stamp uORB
rez@rez-virtual-machine:~/px401/Firmware/build/px4_fmu-v5_default$
```

安装一些依赖的库:

sudo apt-get update

sudo apt-get install git zip cmake build-essential python-empy

sudo apt-get install python-matplotlib python-numpy python-pexpect

sudo apt-get install python-pyqtgraph python-scipy python-serial python-serial

sudo apt-get install ant protobuf-compiler libeigen3-dev libopencv-dev openocd

其中, **openocd** 为开源调试工具, 用于与 **PX4-Autopilot** 中的调试器和硬件进行交互。

```
rez@rez-virtual-machine: ~/px401/Firmware
准备解压 .../6-openocd-0.11.0-1_amd64.deb ...
正在解压 openocd (0.11.0-1) ...
正在设置 libjim0.79:amd64 (0.79+dfsg0-3) ...
正在设置 libjaylink0:amd64 (0.2.0-1) ...
正在设置 libhidapi-hidraw0:amd64 (0.11.2-1) ...
正在设置 libusb-0.1-4:amd64 (2:0.1.12-32build3) ...
正在设置 libcapstone4:amd64 (4.0.2-5) ...
正在设置 libgpod2:amd64 (1.6.3-1build1) ...
正在设置 openocd (0.11.0-1) ...
正在处理用于 libc-bin (2.35-0ubuntu3.1) 的触发器 ...
正在处理用于 man-db (2.10.2-1) 的触发器 ...
正在处理用于 install-info (6.8-4build1) 的触发器 ...
rez@rez-virtual-machine:~/px401/Firmware$ make px4_fmu-v5_default upload
[905/908] Linking CXX executable px4_fmu-v5_default.elf
Memory region      Used Size  Region Size  %age Used
FLASH_ITCM:         0 GB      2016 KB      0.00%
FLASH_AXIM:    2010617 B      2016 KB     97.40%
ITCM_RAM:          0 GB         16 KB      0.00%
DTCM_RAM:          0 GB        128 KB      0.00%
SRAM1:           45852 B        368 KB     12.17%
SRAM2:            0 GB         16 KB      0.00%
[907/908] uploading px4
Waiting for bootloader...
```

make px4\_fmu-v3\_default upload

```
rez@rez-virtual-machine: ~/px4v302/PX4-Autopilot
z/px4v302/PX4-Autopilot/build/px4_fmu-v3_default/px4_fmu-v3_default.px4
ninja: build stopped: subcommand failed.
make: *** [Makefile:227: px4_fmu-v3_default] 错误 1
rez@rez-virtual-machine:~/px4v302/PX4-Autopilot$ make px4_fmu-v3_default upload
[0/1] uploading px4
Waiting for bootloader...
Found board id: 9,0 bootloader version: 5 on /dev/serial/by-id/usb-Hex_ProfiCNC_CubeBlack-BL_34001D001151303239353934-if00
Loaded firmware for board id: 9,0 size: 2043913 bytes (98.23%)
sn: 001d00343230511134393539
chip: 20036419
family: b'STM32F42x'
revision: b'?'
flash: 2080768 bytes
Windowed mode: False
Erase : [=====] 100.0% (timeout: 14 seconds)
Program: [=====] 100.0%
Verify : [=====] 100.0%
Rebooting. Elapsed Time 57.655
rez@rez-virtual-machine:~/px4v302/PX4-Autopilot$
```

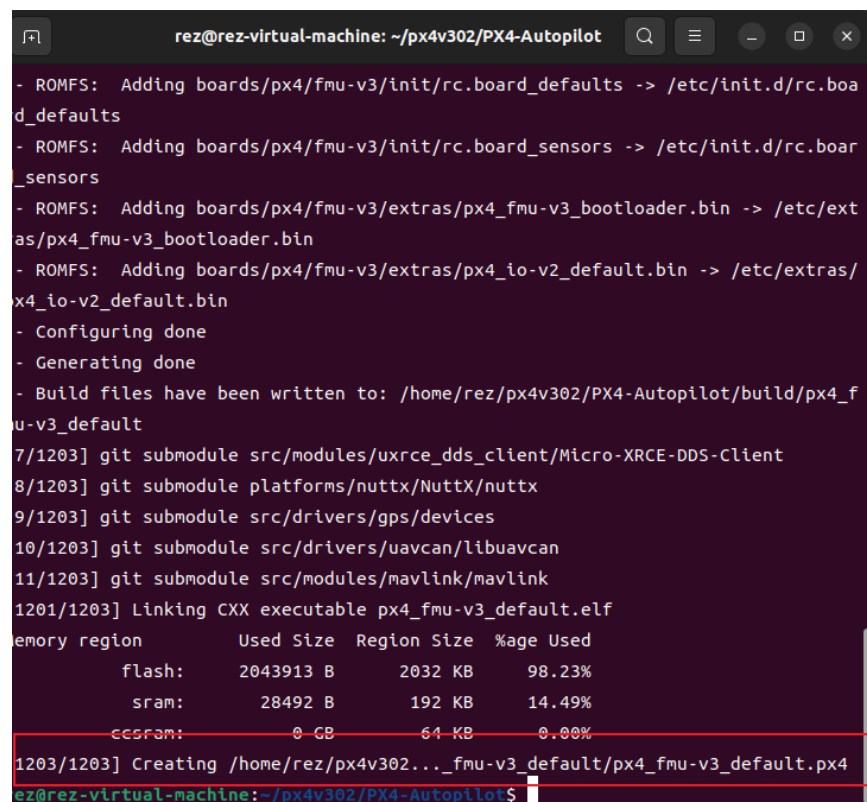
上传至Pixhawk2.4.8

1.2

要为 Pixhawk 2.4.8 和 Pixhawk 4 飞控正确编译 PX4 Autopilot 固件，您需要分别克隆不同的代码库并选择相应的编译选项和配置文件。以下是针对 Pixhawk 2.4.8 和 Pixhawk 4 飞控的建议：

文件编号		页码 10	版次 A
<p>1. Pixhawk 2.4.8 飞控</p> <ul style="list-style-type: none"> <li>- 克隆 PX4-Autopilot 代码库：  <pre>git clone https://github.com/PX4/PX4-Autopilot.git</pre> </li> <li>- 进入代码库目录，并切换到适用于 Pixhawk 2.4.8 的固件分支：  <pre>cd PX4-Autopilot</pre> <pre>git checkout v1.12.3 （v1.13.0 默认—fmu-3）</pre> </li> <li>- 编译前最好更新一下环境配置：  <pre>cd PX4-Autopilot/src/lib</pre> <pre>git submodule update --init --recursive</pre> </li> <li>- 选择适用于 Pixhawk 2.4.8 的编译选项：  <pre>make px4_fmu-v3_default</pre> </li> </ul> <p>这将使用 GCC 来编译适用于 Pixhawk 2.4.8 的固件。如果想要使用其他编译工具，如 Clang 或 Ninja，请参考官方文档。</p> <p>2. Pixhawk 4 飞控</p> <ul style="list-style-type: none"> <li>- 克隆 PX4-Autopilot 代码库：  <pre>git clone https://github.com/PX4/PX4-Autopilot.git</pre> </li> <li>- 进入代码库目录，并切换到适用于 Pixhawk 4 的固件分支：  <pre>cd PX4-Autopilot</pre> <pre>git checkout v1.12.3</pre> </li> <li>- 选择适用于 Pixhawk 4 的编译选项：  <pre>make px4_fmu-v5_default</pre> </li> </ul> <p>这将使用 GCC 来编译适用于 Pixhawk 4 的固件。如果您想要使用其他编译工具，如 Clang 或 Ninja，请参考官方文档。</p> <p>请注意，针对不同飞控的代码库和编译选项之间的主要区别在于飞控硬件的不</p>			

同。Pixhawk 2.4.8 飞控使用的是 STM32F427 处理器，而 Pixhawk 4 飞控使用的是 STM32F765 处理器。因此，针对不同硬件的固件需要使用不同的编译选项和配置文件，以确保生成的固件可以正确运行在目标硬件上。



```
rez@rez-virtual-machine: ~/px4v302/PX4-Autopilot
- ROMFS: Adding boards/px4/fmu-v3/init/rc.board_defaults -> /etc/init.d/rc.boa
d_defaults
- ROMFS: Adding boards/px4/fmu-v3/init/rc.board_sensors -> /etc/init.d/rc.boar
_sensors
- ROMFS: Adding boards/px4/fmu-v3/extras/px4_fm-v3_bootloader.bin -> /etc/ext
as/px4_fm-v3_bootloader.bin
- ROMFS: Adding boards/px4/fmu-v3/extras/px4_io-v2_default.bin -> /etc/extras/
x4_io-v2_default.bin
- Configuring done
- Generating done
- Build files have been written to: /home/rez/px4v302/PX4-Autopilot/build/px4_f
u-v3_default
7/1203] git submodule src/modules/uxrce_dds_client/Micro-XRCE-DDS-Client
8/1203] git submodule platforms/nuttx/Nuttx/nuttx
9/1203] git submodule src/drivers/gps/devices
10/1203] git submodule src/drivers/uavcan/libuavcan
11/1203] git submodule src/modules/mavlink/mavlink
1201/1203] Linking CXX executable px4_fm-v3_default.elf
memory region      Used Size  Region Size  %age Used
      flash:      2043913 B      2032 KB      98.23%
      sram:        28492 B       192 KB      14.49%
      ccsram:         0 B         64 KB       0.00%
1203/1203] Creating /home/rez/px4v302..._fm-v3_default/px4_fm-v3_default.px4
rez@rez-virtual-machine:~/px4v302/PX4-Autopilot$
```

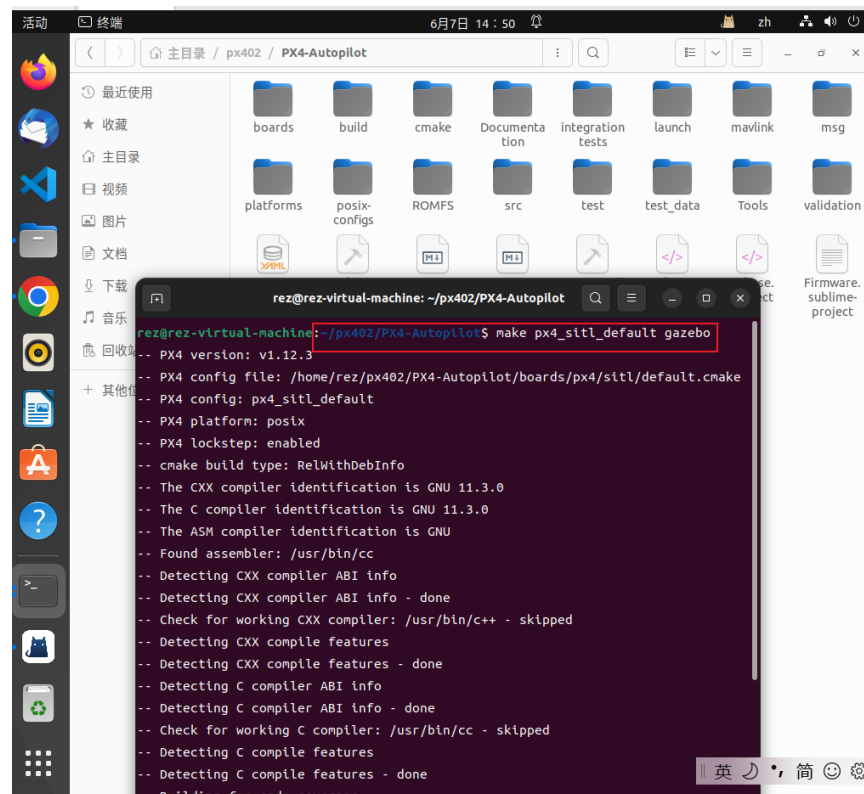
编译到虚拟仿真器前，先安装编译需要依赖的库：

`sudo apt-get update`

`sudo apt-get install libboost-all-dev`

这将从 Ubuntu 软件源中安装最新的 Boost 库，并将其安装在系统的默认位置。





虚拟仿真器与 QGC 的联合仿真：

打开 QGroundControl 地面站：打开 QGroundControl 地面站，并连接到本地 P 地址 127.0.0.1 和端口 14550。这将使您可以监视飞行器的状态并发送控制指令。

## 2 PX4-Autopilot 文件夹下的子目录架构

使用 Git 命令克隆 PX4 源码后的 PX4-Autopilot 文件夹如下图：

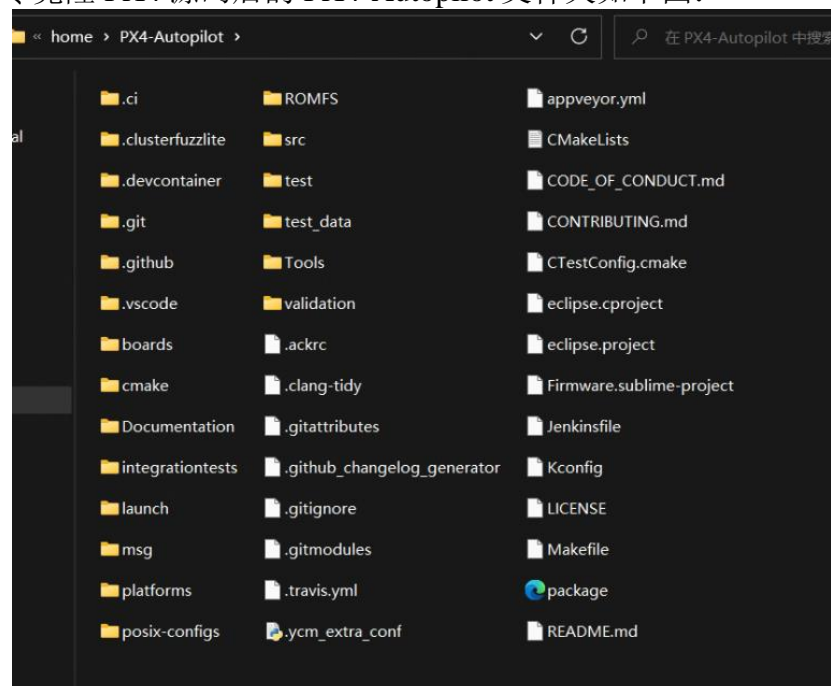


图 1 使用 Git 命令下载的 PX4 源码文件夹 PX4-Autopilot

文件编号		页码 13	版次 A
<p>PX4-Autopilot 源代码总目录下的子文件夹包括：</p> <ol style="list-style-type: none"><li>1. <b>.github</b>：该文件夹包含 GitHub 存储库的配置文件。</li><li>2. <b>boards</b>：该文件夹包含各种支持的硬件平台和设备的配置文件，例如飞控硬件、传感器和电机控制器等。这些配置文件定义了硬件接口和设备驱动程序等。</li></ol> <p>其中，boards/px4 目录包含了 PX4 所支持的各种飞控硬件及其相关配置文件。这些配置文件定义了硬件接口和设备驱动程序等，使得 PX4 系统能够与不同的飞控硬件平台进行通信。每个子目录都代表一个特定的飞控硬件平台。这些配置文件和代码使得 PX4 系统能够与不同的飞控硬件平台进行通信，并提供各种功能，例如姿态控制、导航、传感器数据处理、通信等。如果需要将 PX4 系统移植到新的飞控硬件平台上，可以参考这些配置文件和代码，并在其基础上进行修改和扩展。</p> <ol style="list-style-type: none"><li>3. <b>build</b>：该文件夹包含编译过程中生成的所有中间文件和目标文件。这些文件包括可执行文件、库文件、对象文件、符号文件等。在执行“make”命令时，编译输出将生成在该目录中。</li><li>4. <b>cmake</b>：该文件夹包含 CMake 构建系统的配置文件。这些文件定义了 PX4 的构建过程和依赖关系，以及如何生成可执行文件和库文件等。</li><li>5. <b>drivers</b>：位于 src 中。该文件夹包含各种传感器和设备的驱动程序，例如惯性测量单元（IMU）、GPS、气压计、电机控制器等。这些驱动程序负责与硬件交互，并将传感器数据和控制信号传递给 PX4 系统。</li></ol> <p>drivers 目录包含了 PX4 所支持的各种传感器和设备的驱动程序。这些驱动程序负责与硬件交互，并将传感器数据和控制信号传递给 PX4 系统。该目录下的驱动程序与硬件平台无关，可以在不同的飞控硬件平台上通用。</p> <ol style="list-style-type: none"><li>6. <b>msg</b>：msg 文件夹是存放 MAVLink 协议中使用的消息定义文件的地方。这些消息定义文件描述了各种飞行器状态、传感器数据、控制命令等，用于与地面站通信并传输各种消息。PX4 系统使用 MAVLink 协议与地面站通信，通过该协议传输各种消息，例如姿态、位置、速度、控制指令等。在 msg 文件夹中，每个子文件夹表示一个特定的 MAVLink 消息类型，例如：</li></ol> <p>每个子文件夹中都包含一个 XML 文件，该文件定义了该消息类型的字段和数据类型等。PX4 使用工具自动生成源代码，用于解析和生成这些消息。</p>			



- 7. platforms: 系统平台实现的文件, 包括 PX4 采用的 Nuttx 操作系统的源代码。
- 8. src: 该文件夹包含 PX4 的核心源代码。这些源代码实现了 PX4 的各种功能, 例如姿态控制、导航、传感器数据处理、通信、数据存储等。该文件夹中的代码被组织成多个子文件夹, 每个子文件夹都包含特定功能的源代码。

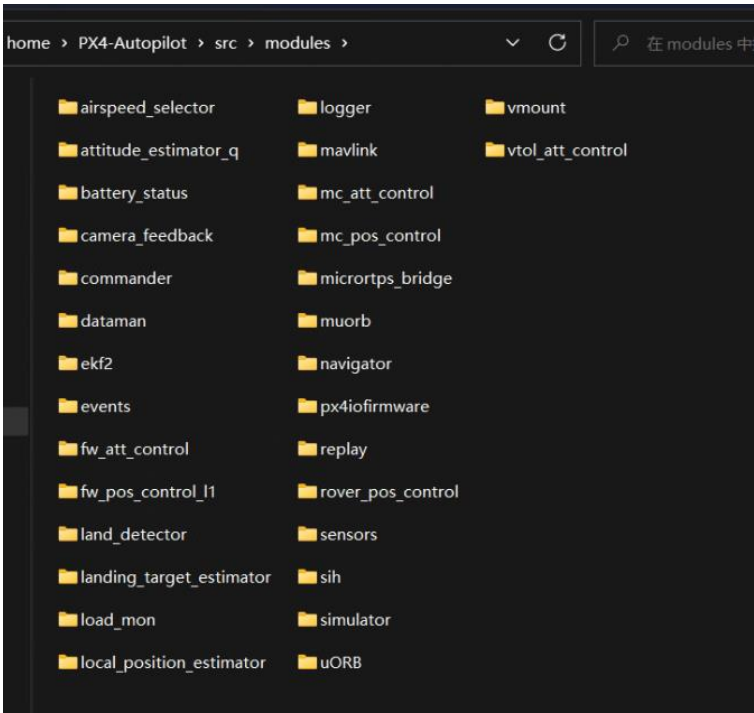


图 2 PX4 的核心源代码

以下分析粘自 CSDN:

[\(26 条消息\)【PX4-AutoPilot 教程-1】PX4 源码文件目录架构分析\\_啾音攻城狮的博客-CSDN 博客](#)

modules 文件夹是功能模块文件夹, 也是二次开发主要的文件夹, 包括姿态解算, 姿态控制, 位置估计, 位置控制, 指令控制, 落地检测, 传感器初始化等。

attitude\_estimator\_q: 使用 mahony 互补滤波算法实现姿态结算。

commander: 整个系统的过程实现, 包括起飞前各传感器的校准、安全开关是否使能、飞行模式切换、pixhawk 硬件上指示灯颜色定义等。

ekf2: 使用扩展卡尔曼滤波器算法实现姿态和位置结算。

fw\_att\_control: 固定翼飞机的姿态控制。

fw\_pos\_control\_l1: 固定翼飞机的位置控制器。

land\_detector: 使用 land 飞行模式降落的落地监测部分。

local\_position\_estimator: LPE 算法实现位置解算。

logger: 关于 log 日志的读写函数。

文件编号		页码 15	版次 A
<p><b>mavlink:</b> 和地面站通信的通信协议。</p> <p><b>mc_att_control:</b> 姿态控制的算法实现，主要就是姿态的内外环 PID 控制，外环角度控制、内环角速度控制。</p> <p><b>mc_pos_control:</b> 位置控制的算法实现，主要就是位置的内外环 PID 控制，外环速度控制、内环加速度控制。</p> <p><b>navigator:</b> 任务，失效保护和 RTL 导航仪。</p> <p><b>sensors:</b> 关于各种传感器的相关函数。</p> <p><b>vtol_att_control:</b> 垂直起降姿态控制器。</p> <p>9. <b>test:</b> 该文件夹包含 PX4 的各种测试程序。这些测试程序用于测试 PX4 的各种功能和组件，例如传感器数据处理、控制逻辑、通信等。测试程序通常在仿真环境中运行，并且可以通过 CI/CD 管道自动运行。</p> <p>10. <b>Tools:</b> 该文件夹包含各种辅助工具，例如日志查看器、参数编辑器、仿真环境等。这些工具可以帮助您调试和测试 PX4 系统，并管理 PX4 系统的配置和参数。</p> <p>11. <b>.gitignore:</b> 该文件是 Git 版本控制系统的配置文件，用于指定哪些文件和目录应该被忽略，不被纳入版本控制。</p> <p>12. <b>.travis.yml:</b> 该文件是 Travis CI 的配置文件，用于定义 CI/CD 工作流程和构建过程。</p> <p>13. <b>CMakeLists.txt:</b> 该文件是 CMake 构建系统的顶级配置文件，定义了整个 PX4 系统的构建过程和依赖关系。</p> <p>14. <b>Makefile:</b> 该文件是 GNU Make 构建工具的顶级配置文件，定义了 PX4 的编译过程和依赖关系。</p> <p>15. <b>README.md:</b> 该文件是项目的说明文档，包含了 PX4 的概述、功能、安装和使用方法等。</p> <h3>3 PX4 源码中的传感器接口</h3> <h4>3.1 传感器相关程序的路径位置</h4> <p>PX4 源码中的传感器接口部分实现了对多种不同类型的传感器的支持和管理，通过通用的传感器驱动接口和传感器数据处理模块，实现了对传感器数据的处理和发布，使得其他组件可以方便地获取和使用传感器数据。传感器接口程序在 PX4 源代码中对</p>			

文件编号		页码 16	版次 A
<p>应的位置如下：</p> <ol style="list-style-type: none"><li>1. 传感器驱动程序：在`src/drivers`目录下，包括多个子目录，每个子目录对应一个特定的传感器类型。</li><li>2. 传感器数据处理：在`src/modules/sensors`目录下，包括多个子目录，每个子目录对应一个特定的传感器类型的数据处理模块。</li><li>3. 传感器驱动接口：在`src/drivers`目录下，包括`SensorDriver`类、`SensorData`类和`SensorFactory`类等。</li><li>4. 传感器数据发布：在`msg`目录下，包括多个子目录，每个子目录对应一个特定的传感器数据类型的 uORB 消息定义文件。</li></ol> <p>以上部分并非完全独立，它们之间存在着相互依赖和交互。例如，传感器驱动程序需要与传感器数据处理模块进行交互，将读取的传感器数据发送给数据处理模块；传感器数据处理模块需要订阅传感器数据主题来获取传感器数据，并将处理后的数据发布到对应的 uORB 消息主题上；传感器驱动接口则是传感器驱动程序和传感器数据处理模块之间的桥梁，提供了标准化的接口和协议，使得它们可以方便地进行交互和通信。因此，在理解 PX4 源代码中的传感器接口部分时，需要综合考虑这些部分之间的关系和交互。</p> <h3>3.2 传感器相关程序的功能说明</h3> <p>针对 PX4 源码中的传感器接口部分的功能介绍如下：</p> <ol style="list-style-type: none"><li>1. 传感器驱动程序：PX4 支持多种不同类型的传感器，例如加速度计、陀螺仪、磁力计、气压计、GPS 等。每种传感器都需要对应的驱动程序来读取传感器数据并将其传输到 PX4 系统中。传感器驱动程序通常包括硬件抽象层（HAL）和传感器驱动层（SDL）两部分，其中 HAL 负责与具体的硬件平台进行交互，SDL 负责解析传感器数据并将其发送到 PX4 系统中。</li><li>2. 传感器数据处理：PX4 系统中有一个传感器数据处理模块，用于对传感器数据进行处理和校准。该模块负责读取传感器数据并进行滤波、校准、转换等操作，以提高传感器数据的信噪比和精度。</li><li>3. 传感器驱动接口：PX4 系统中定义了一套通用的传感器驱动接口，用于将不同类型的传感器驱动程序和数据处理模块进行交互。传感器驱动接口包括传感器驱动接口类（SensorDriver）、传感器数据接口类（SensorData）和传感器工厂类（SensorFactory）等。传感器驱动接口类定义了一组标准的接口函数，传感器驱动程序需要实现这些接口函数以与传感器数据处理模块进行交互。传感器数据接口类用于封装传感器数据，并提供了一组标准的接口函数，传感器数据处理模块需要实现这些接口函数以处理传感器数据。传感器工厂类用于创建和管理传感器驱动程序和传感器数据处理模块，以实现对外不同类型的传感器的支持和管理。</li><li>4. 传感器数据发布：PX4 系统中使用 uORB 消息总线进行内部组件之间的通信和数据交换。传感器数据处理模块会将处理后的传感器数据发布到对应的 uORB 消息主题上，其他组件可以订阅这些消息来获取传感器数据。PX4 系统中定义了一套标准的 uORB 消息主题类型，用于存储和传输传感器数据。每个传感器数据主题都有一个对应的消息类型，用于描述传感器数据的格式和字段。</li></ol>			

文件编号		页码 17	版次 A
<h2>4 PX4 源码中不同模块间的 uORB 通信</h2> <p>PX4 源码包括了从底层硬件驱动到上层功能模块的各个部分，提供了一个完整的无人机控制系统的开发框架。PX4 源码中不同功能模块之间是通过 uORB 发布(Publish)和订阅(Subscribe)消息。<a href="#">模块通过发布消息将数据发送到 uORB 中，其他模块可以订阅该消息，以便在需要时获取相关数据。</a>uORB 通过主题(Topic)来描述不同类型的消息，在发布和订阅消息时都需要指定相应的主题。</p> <h3>4.1 uORB 通信框架</h3> <p>在 PX4 源代码中，每个不同程序之间通过 UORB 进行通信。UORB 是一种用于高效、实时数据交换的发布-订阅消息机制，是 PX4 中的通信框架。</p> <p>以下是 PX4 中使用 UORB 进行通信的基本步骤：</p> <ol style="list-style-type: none"><li>1)定义消息：每个程序需要定义其所需的消息格式。消息定义使用一个结构体，并包含消息字段的数据类型和名称。</li><li>2)注册话题：在程序中，使用 ORB_DEFINE 宏将消息定义为一个话题（topic）。每个话题具有唯一的名称和消息类型。话题名称用于在发布者和订阅者之间进行匹配。</li><li>3)发布消息：在发布者程序中，可以创建消息实例，并使用 orb_publish()函数将其发布到特定的话题。这样，其他订阅者就可以接收到该消息。</li><li>4)订阅消息：在订阅者程序中，可以使用 orb_subscribe()函数订阅特定的话题。通过调用 orb_copy()函数，可以将最新的消息副本复制到本地变量中，以进行进一步处理和分析。</li><li>5)通信循环：程序通过在循环中重复发布和订阅消息来实现持续的通信。在每个循环迭代中，发布者发布消息，而订阅者订阅并接收最新的消息。</li></ol> <p>通过使用 UORB 进行消息发布和订阅，不同的程序可以以异步、实时的方式交换数据。这种松耦合的通信机制使得 PX4 的各个组件能够以高效、可靠的方式共同协作，实现复杂的飞行控制和任务执行。需要注意的是，UORB 的实现细节可能因 PX4 版本和具体组件而有所不同。建议参考 PX4 的官方文档和源代码以获取更详细的信息和实现细节。</p> <p>在 PX4 源代码中，与 uORB 相关的程序可以在以下位置找到：</p> <ol style="list-style-type: none"><li>1)uORB 消息定义：uORB 消息的定义位于 msg 目录中。每个消息都有一个对应的.msg 文件，定义了消息的结构和字段。 这些文件通常位于 src/modules/&lt;module_name&gt;/msg/目录下，其中&lt;module_name&gt;表示相应模块的名称。</li><li>2)uORB 话题注册：uORB 话题的注册位于 src/modules/&lt;module_name&gt;/module.cpp 文件中。在该文件中，使用 ORB_DEFINE 宏来注册和定义话题。每个模块的 module.cpp 文件负责定义并注册该模块所需的所有话题。</li><li>3)uORB 发布者：uORB 发布者的实现位于各个模块的源代码中。可以在 src/modules/&lt;module_name&gt;/目录下的相关源代码文件中找到发布者的实现。发布者使用 orb_publish()函数将消息发布到相应的话题。</li><li>4)uORB 订阅者：uORB 订阅者的实现也位于各个模块的源代码中。可以在 src/modules/&lt;module_name&gt;/目录下的相关源代码文件中找到订阅者的实现。订阅者使</li></ol>			

文件编号		页码 18	版次 A
<p>用 <code>orb_subscribe()</code> 函数订阅相应的话题，并使用 <code>orb_copy()</code> 函数获取最新的消息。</p> <p>总体而言，uORB 相关的程序分布在不同的模块目录下，每个模块负责定义和处理自己所需的消息和通信。您可以根据具体的模块和功能需求查找相应的源代码文件，以深入了解 uORB 在 PX4 中的实现和使用。</p> <h2>4.2 不同模块通过 uORB 通信的案例</h2> <hr/> <p>例如，假设模块 A 需要将姿态角数据发送到模块 B，可以按照以下步骤进行操作：</p> <ol style="list-style-type: none"><li>在模块 A 中定义一个姿态角消息类型，如下所示： ... <pre>struct attitude_angles_s {     float roll;     float pitch;     float yaw; };</pre> ...</li><li>在模块 A 中发布该消息，<b>将数据发送到 UORB 中</b>，可以使用以下代码： ... <pre>orb_advert_t pub = orb_advertise(ORB_ID(attitude_angles), &amp;msg);</pre> ... <p>其中，<code>ORB_ID(attitude_angles)</code> 指定了该消息的主题，<code>&amp;msg</code> 是姿态角消息的指针。</p></li><li>在模块 B 中订阅该消息，以获取姿态角数据，可以使用以下代码： ... <pre>int sub = orb_subscribe(ORB_ID(attitude_angles));</pre> ... <p>其中，<code>ORB_ID(attitude_angles)</code> 指定了要订阅的主题。</p></li><li>在模块 B 中通过 <code>orb_copy</code> 函数获取姿态角数据，如下所示： ... <pre>struct attitude_angles_s msg; orb_copy(ORB_ID(attitude_angles), sub, &amp;msg);</pre> ... <p>其中，<code>ORB_ID(attitude_angles)</code> 指定了要获取数据的主题，<code>sub</code> 是订阅该主题的句柄，<code>&amp;msg</code> 是用于存储姿态角消息数据的指针。</p></li></ol>			

文件编号		页码 19	版次 A
<p>通过这种方式，不同模块之间可以进行消息传递和数据共享，实现不同模块之间的协同工作。UORB 作为 PX4 系统中的核心组件之一，为 PX4 系统提供了一种高效、可靠的消息传递机制。</p> <p>我的理解如下：在 PX4 系统中，uORB 消息机制相当于一个中介模块，用于联系不同的功能模块，使它们之间可以进行数据共享和通信。uORB 提供了一种轻量级的消息传递机制，使得不同功能模块之间可以进行高效的数据传递和消息交换。</p> <h2>5 PX4 源码移植至另一台 STM32 硬件系统</h2> <p><b>概述：</b></p> <p>要将编译好的 PX4 可执行文件移植到另一台 STM32 硬件系统上，需要执行以下步骤：</p> <ul style="list-style-type: none"><li>➤ 确定目标 STM32 硬件系统的特性： 获取目标硬件的芯片型号和技术规格。 确定硬件的处理器架构、时钟频率、存储器大小等关键参数。 了解目标硬件上的外设配置和引脚映射。</li><li>➤ 配置 PX4 源码中的硬件相关文件： 在 PX4 源码中，找到与目标 STM32 芯片型号对应的配置文件，一般位于 boards 目录下。 修改配置文件中的引脚映射、时钟设置和外设配置等，以适配目标 STM32 硬件系统。</li><li>➤ 配置引脚映射： 根据目标硬件系统的引脚映射和外设配置，修改 PX4 源码中的硬件配置文件，例如 board_config.h 或类似的文件。在这些文件中，你可以定义与目标硬件系统相关的引脚映射、外设配置和时钟设置等。</li><li>➤ 配置编译选项： 修改 PX4 源码根目录下的.config 文件或使用 CMake 进行配置，确保编译选项正确设置为目标硬件系统。在编译过程中，确保正确选择目标硬件系统的编译选项。这通常需要在 Makefile 或 CMakeLists.txt 中进行相应的配置，包括选择正确的目标芯片型号和设置编译器参数等。</li></ul>			

文件编号		页码 20	版次 A
<p>配置编译器选项、处理器架构和优化级别等。</p> <p>➤ 编译 PX4 固件：</p> <p>执行编译命令，根据目标硬件系统的配置编译 PX4 源码。具体的编译命令可以是 <code>make px4_fmu-vX_default</code>，其中 X 表示目标硬件版本号。</p> <p>编译过程可能需要较长时间，因此请耐心等待。</p> <p>➤ 烧录固件到目标硬件系统：</p> <p>将编译生成的可执行文件（通常是 .px4 或 .bin 文件）烧录到目标 STM32 硬件系统的存储器中。</p> <p>可以使用适当的烧录工具（例如 ST-Link、J-Link 等）来执行烧录操作。具体的烧录步骤和工具使用方法会根据硬件和开发环境而有所不同。</p> <p>➤ 涉及 Bootloader 配置，步骤可能会略有不同。</p> <p>在这种情况下，就需要了解目标 STM32 硬件系统的 Bootloader 配置方式，并根据其要求进行操作。一般情况下，需要将 PX4 固件烧录到硬件系统的特定存储器区域，并配置 Bootloader 以正确引导 PX4 固件。具体的 Bootloader 配置方法取决于硬件系统和使用的 Bootloader。</p> <p>以上步骤仅为概述，具体的移植过程可能因硬件系统、PX4 版本和开发环境而有所不同。</p> <h3>5.1 修改引脚配置示例</h3> <p>PX4 是一款开源的飞控软件，支持多种不同的硬件平台。如果需要修改硬件平台的引脚配置，可以通过修改对应的 <code>board_config.h</code> 文件来实现。下面是一个简单的例子来说明如何修改引脚配置：</p> <p>假设我们有一款基于 STM32F4xx 芯片的飞控板，现在需要将其上的 SPI1 接口的 SCK 引脚从默认的 PB3 修改为 PC10。可以按照以下步骤进行修改：</p> <ol style="list-style-type: none"><li>找到对应的 <code>board_config.h</code> 文件。</li></ol> <p>在 PX4 源码中，不同的硬件平台使用不同的 <code>board_config.h</code> 文件来描述硬件的引脚配置。通常位于路径 <code>`Firmware/src/boards/px4/&lt;board_name&gt;/board_config.h`</code> 下，其中 <code>`&lt;board_name&gt;`</code> 是硬件平台的名称（比如 <code>fmu-v5</code>）。找到对应的 <code>board_config.h</code> 文件后，你可以在其中进行修改。</p> <ol style="list-style-type: none"><li>找到 SPI1 的引脚配置。</li></ol> <p>在 <code>board_config.h</code> 文件中，可以找到关于 SPI1 引脚配置的代码段。你可以在其中</p>			



找到 SCK 引脚的定义，通常是类似下面这样的代码：

```
```c
#define GPIO_SPI1_SCK                /* default: PB3 */ \
    (GPIO_OUTPUT|GPIO_PUSHPULL|GPIO_SPEED_50MHz|GPIO_OUTPUT_CLEAR|GPIO_PORTB|GPIO_PIN3)
```
```

3. 修改 SCK 引脚的定义。

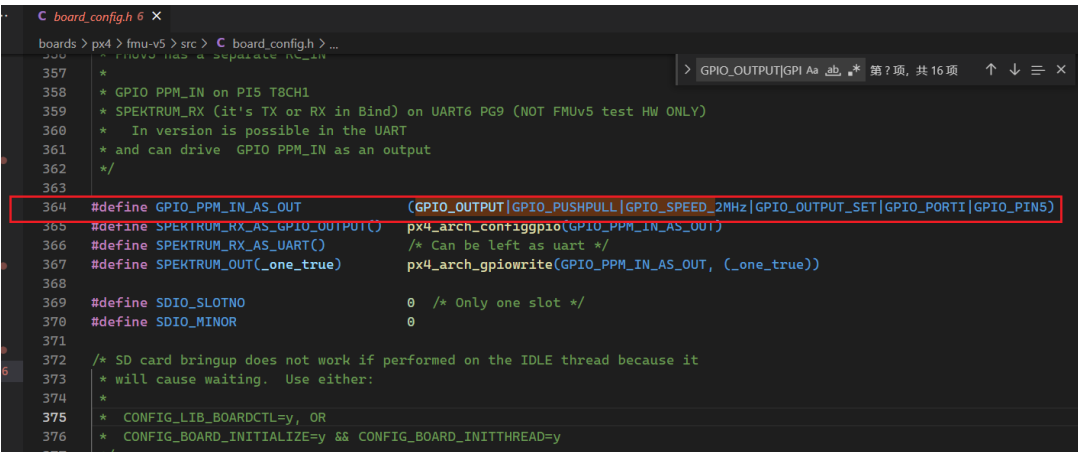
将上述代码中的 GPIO\_PORTB 和 GPIO\_PIN3 分别改为 GPIO\_PORTC 和 GPIO\_PIN10，即可将 SCK 引脚从 PB3 修改为 PC10。修改后的代码如下：

```
```c
#define GPIO_SPI1_SCK                /* modified: PC10 */ \
    (GPIO_OUTPUT|GPIO_PUSHPULL|GPIO_SPEED_50MHz|GPIO_OUTPUT_CLEAR|GPIO_PORTC|GPIO_PIN10)
```
```

4. 保存并编译代码。

在完成修改后，保存 board\_config.h 文件并重新编译 PX4 代码，即可将修改后的引脚配置应用到硬件平台上。

需要注意的是，修改硬件平台的引脚配置需要谨慎操作，以免出现硬件损坏或软件异常等问题。



逐步解释上图这段代码，并演示如何修改引脚配置。

- GPIO\_PPM\_IN\_AS\_OUT: 这是宏定义的名称，它表示将 PPM 输入引脚配置为输出引脚。

(GPIO\_OUTPUT|GPIO\_PUSHPULL|GPIO\_SPEED\_2MHz|GPIO\_OUTPUT\_SET|GPIO\_PORTI|GPIO\_PIN5): 这部分是具体的引脚配置参数。它使用了位操作符 "|" 来将多个



|  |  |       |      |
|--|--|-------|------|
| 文件编号   |  | 页码 22 | 版次 A |
| <p>参数组合在一起。</p> <ul style="list-style-type: none"><li>➤ GPIO_OUTPUT: 将引脚配置为输出模式。</li><li>➤ GPIO_PUSH_PULL: 配置输出引脚为推挽输出。</li><li>➤ GPIO_SPEED_2MHz: 设置引脚输出速度为 2MHz。</li><li>➤ GPIO_OUTPUT_SET: 将引脚初始化为高电平状态。</li><li>➤ GPIO_PORTI: 引脚所在的 GPIO 端口为 Port I。</li><li>➤ GPIO_PIN5: 引脚的编号为 5。</li></ul> <p>现在，将演示如何修改引脚配置。假设我们要将该引脚配置为另一个引脚，例如 Port C 的引脚号为 8。在 board_config.h 文件中找到该宏定义并修改如下：</p> <pre>#define GPIO_PPM_IN_AS_OUT<br/>(GPIO_OUTPUT GPIO_PUSH_PULL GPIO_SPEED_2MHz GPIO_OUTPUT_SET GPIO_P<br/>ORTC GPIO_PIN8)</pre> <p>根据目标硬件系统的引脚配置，修改 GPIO_PORTC 为正确的 GPIO 端口，修改 GPIO_PIN8 为正确的引脚编号。</p> <p>完成修改后，保存文件并重新编译整个 PX4 固件。确保在编译时选择了适合目标硬件的编译选项，并将修改后的固件烧录到目标 STM32 硬件系统上。</p> <p>注意，以上演示只是一个示例，具体的引脚配置和修改方法取决于目标硬件系统和相关的硬件文档。</p> |  |       |      |
| <h2>5.2 编译配置（Makefile 和 CMakeLists.tx）</h2>  |  |       |      |
| <p>1. Makefile 和 CMakeLists.txt 文件在 PX4 源码中扮演不同的角色，并使用不同的构建系统。</p>   |  |       |      |
| <p>1.1 Makefile:</p>   |  |       |      |
| <p>Makefile 是 GNU make 工具使用的构建系统文件，它使用 Makefile 语法描述了构建过程。</p>   |  |       |      |
| <p>Makefile 文件包含了一系列规则和命令，用于定义编译、链接和构建的步骤。</p>   |  |       |      |
| <p>Makefile 使用依赖关系来确定哪些文件需要重新编译以及它们之间的依赖关系。</p>  |  |       |      |
| <p>Makefile 通常以递归方式构建目标，即根据依赖关系，自动执行所需的操作。</p>   |  |       |      |
| <p>1.2 CMakeLists.txt:</p>   |  |       |      |
| <p>CMakeLists.txt 是 CMake 构建系统使用的配置文件，它使用 CMake 语言描述了构</p>   |  |       |      |

建过程。

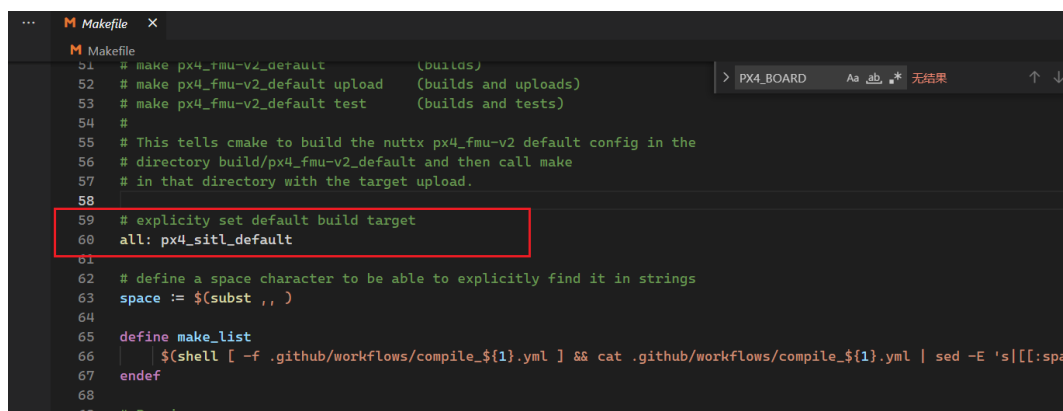
CMakeLists.txt 文件更高级，提供了更灵活和可移植的配置选项。

CMakeLists.txt 文件定义了项目的源代码、依赖关系、编译选项、目标等。

CMake 工具可以根据 CMakeLists.txt 文件生成特定构建系统所需的 Makefile、Visual Studio 项目文件或其他构建脚本。

总结：Makefile 是 GNU make 工具使用的构建系统文件，使用 Makefile 语法描述构建过程，而 CMakeLists.txt 是 CMake 构建系统使用的配置文件，使用 CMake 语言描述构建过程。CMakeLists.txt 提供了更高级和可移植的配置选项，可以生成不同构建系统所需的文件。在 PX4 源码中，通常推荐使用 CMake 作为构建系统，并使用 CMakeLists.txt 文件进行配置和管理。

## 2. Makefile 文件中的示例解释：



# explicitly set default build target

all: px4\_sitl\_default

在 Makefile 中，all: px4\_sitl\_default 表示将 px4\_sitl\_default 设置为默认的构建目标。

这意味着当运行 make 命令时，Makefile 会自动构建 px4\_sitl\_default 目标，除非在命令行中显式指定其他目标。如果想修改默认构建目标，可以修改该行代码为所需的目标名称。例如，假设你想将 px4\_sitl\_default 修改为 my\_custom\_target，则将该行代码修改为：

all: my\_custom\_target

然后保存 Makefile 并重新运行 make 命令，它将构建我们所指定的新目标作为默认构建目标。

|  |  |       |      |
|--|--|-------|------|
| 文件编号   |  | 页码 24 | 版次 A |
| <div>5.3 PX4 的 Bootloader 示例</div>   |  |       |      |
| <div><p>PX4 源码中的 Bootloader 文件（通常是位于 <code>src/drivers/bootloaders</code> 目录下的 bootloader 源码文件）负责启动和引导 PX4 固件。它的工作流程和原理如下：</p><div><div>➤ Bootloader 加载和初始化：</div><p>在目标硬件上电或复位时，Bootloader 会首先执行。</p><p>Bootloader 的目标是加载和运行 PX4 固件。它通常位于芯片的内部 ROM 或 Flash 存储器的固定地址处。</p></div><div><div>➤ Bootloader 的主要任务：</div><p>配置硬件和外设：Bootloader 会初始化和配置硬件系统，包括时钟设置、引脚映射和外设初始化等。</p></div><div><div>➤ 加载固件：</div><p>Bootloader 会从特定的存储器位置（例如 Flash）加载 PX4 固件到指定的内存地址中。</p></div><div><div>➤ 校验固件：</div><p>Bootloader 会验证加载的固件的完整性和正确性，以确保其有效性。</p></div><div><div>➤ 启动固件：</div><p>一旦固件校验通过，Bootloader 会跳转到固件的入口点，将控制权转移到 PX4 固件中。</p></div><div><div>➤ 引导过程：</div><p>当 Bootloader 启动并加载固件后，控制权会转移到 PX4 固件中。</p><p>PX4 固件会继续执行剩余的初始化步骤和任务，如传感器校准、任务调度和飞行控制等。</p></div><div><div>➤ Bootloader 的优点和用途：</div><p>可靠性：Bootloader 提供了启动和加载固件的可靠机制，可以确保固件的正确加载和运行。</p><p>灵活性：通过 Bootloader，可以支持不同的启动和引导方式，如串口下载、SD 卡引导等。</p></div><div><div>➤ 更新固件：</div><p>Bootloader 使得可以轻松更新 PX4 固件，通过特定的方式和协议将新固件烧录到设备中。</p></div></div> |  |       |      |

|  |  |       |      |
|--|--|-------|------|
| 文件编号   |  | 页码 25 | 版次 A |
| <p>总的来说，PX4 源码中的 Bootloader 文件负责硬件初始化、加载和校验固件，以及引导控制权到 PX4 固件。它是整个 PX4 系统启动和运行的关键组件，确保固件的可靠性和正确性。理解 Bootloader 的工作流程和原理对于了解 PX4 的启动过程和固件加载机制非常重要。</p>  |  |       |      |
| <p>PX4 的 Bootloader 官网链接：<br/><a href="#">PX4 官网的 Bootloader 链接</a></p>  |  |       |      |
| <div><div>1) 构建下载 Bootloader</div><div><div>STM32 Bootloader</div><div><div>PX4 引导加载程序的代码可从 Github <a href="#">Bootloader</a> 存储库获得。</div><div>支持的飞控板<ul style="list-style-type: none"><li>• FMUv2 (Pixhawk 1, STM32F4)</li><li>• FMUv3 (Pixhawk 2, STM32F4)</li><li>• FMUv4 (Pixracer 3 and Pixhawk 3 Pro, STM32F4)</li><li>• FMUv5 (Pixhawk 4, STM32F7)</li><li>• TAPv1 (TBA, STM32F4)</li><li>• ASCv1 (TBA, STM32F4)</li></ul></div><div>构建 Bootloader<pre>git clone https://github.com/PX4/Bootloader.git cd Bootloader git submodule init git submodule update make</pre></div><div><div>在此步骤之后，所有支持的主板的 elf 文件范围都出现在引导 Bootloader 目录中</div><div>2) 刷写 Bootloader</div></div></div></div></div> |  |       |      |

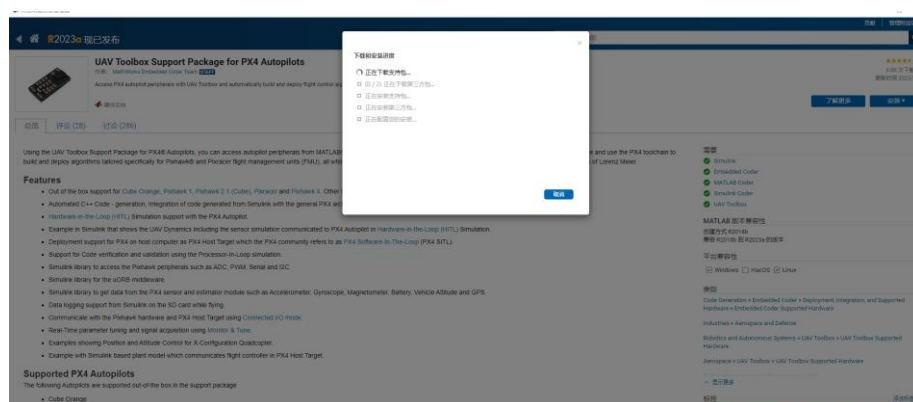
## 6 Windows 系统下 PX4 飞控算法自主开发实践

### 6.1 依赖的软硬件材料（以 Pixhawk4 飞控为例）



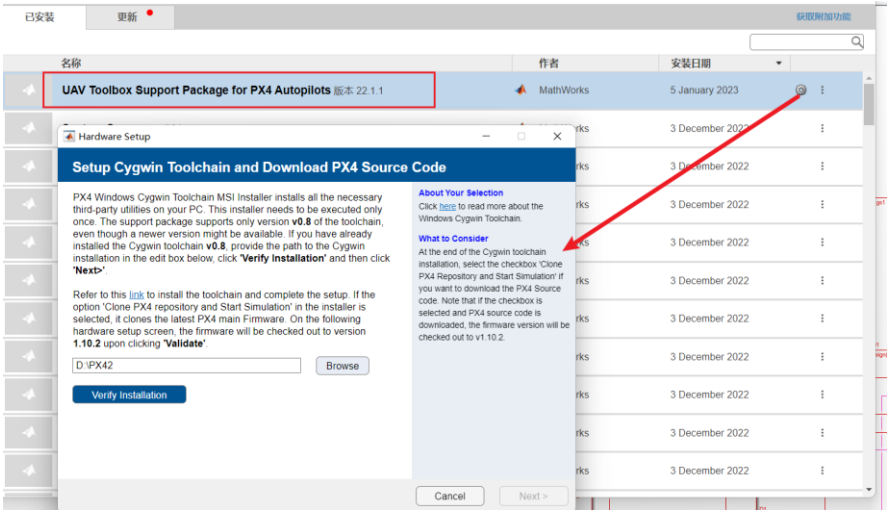
依赖的文件材料如下：

#### 1. 硬件支持包



【UAV Toolbox Support Package for PX4 Autopilots】硬件支持包是针对使用 PX4 飞控的无人机开发者和研究人员设计的一个 MATLAB 工具包。该支持包旨在简化无人机开发过程中的建模、控制和仿真任务，并提供一种便捷的方式将 MATLAB 和 Simulink 与 PX4 飞控集成起来。

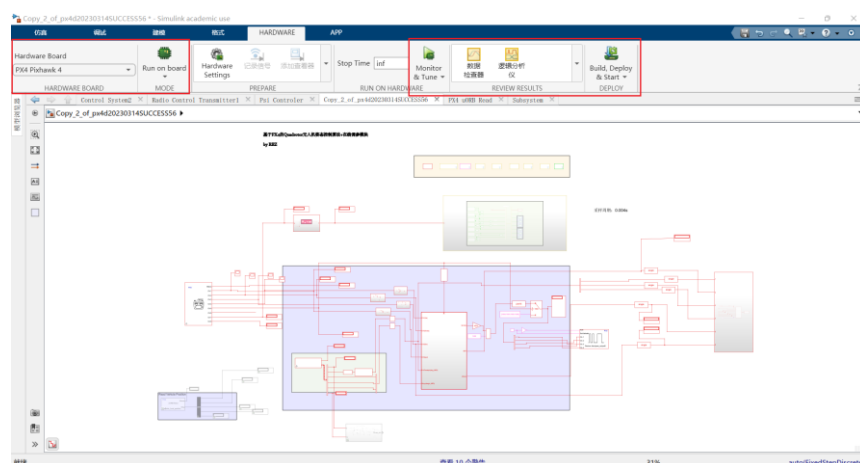


|   |  |       |      |
|---|--|-------|------|
| 文件编号  |  | 页码 27 | 版次 A |
| <div data-bbox="422 188 1313 696"></div>  |  |       |      |
| <p>使用 UAV Toolbox Support Package for PX4 Autopilots 硬件支持包，可以执行以下任务：</p> <ol style="list-style-type: none"><li>1）建立无人机的动力学模型和控制器模型，并进行数学仿真和测试、PX4 飞控硬件在环仿真测试。</li><li>2）使用 Simulink 进行无人机新型控制算法的设计和验证。</li><li>3）与 PX4 飞控硬件板通信，直接将 MATLAB 和 Simulink 生成的控制指令发送到无人机上，实现无人机的自主飞行任务。</li><li>4）通过无线电通信将数据从飞行器传输到 MATLAB，以进行数据分析和后处理。</li><li>5）使用 UAV Toolbox Support Package for PX4 Autopilots 硬件支持包提供的工具和函数，直接访问 PX4 飞控的传感器（加速度计+陀螺仪等）和执行器（PWM）数据，以及其他系统状态信息。</li></ol> <p>总结：UAV Toolbox Support Package for PX4 Autopilots 硬件支持包为使用 MATLAB 和 Simulink 进行无人机开发和研究的用户提供了一个全面的工具集，使他们可以更加快速地构建、测试和部署无人机控制算法等。</p> <p>2.PX4 固件：</p> <p>UAV Toolbox Support Package for PX4 Autopilots 硬件支持包与开源的 PX4 固件相比，提供了更加方便和高效的无人机开发工具集。UAV Toolbox Support Package for PX4 Autopilots 硬件支持包中调用的 PX4 固件与开源的 PX4 固件在本质上没有区别。</p> <p>PX4 固件本身是一个开源的飞控固件，由 PX4 开源社区维护和更新。UAV Toolbox Support Package for PX4 Autopilots 硬件支持包中调用的 PX4 固件实际上就是从 PX4 开源社区获取的固件。</p> |  |       |      |

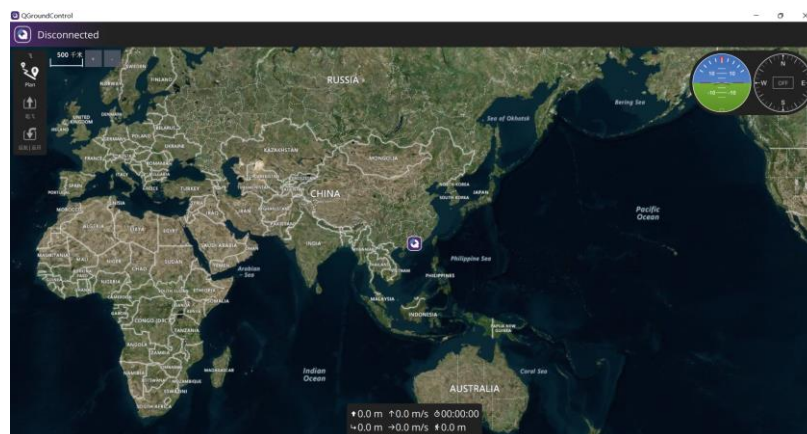
UAV Toolbox Support Package for PX4 Autopilots 硬件支持包中调用的 PX4 固件与开源的 PX4 固件之间的区别在于它们的使用方式和集成方式。UAV Toolbox Support Package for PX4 Autopilots 硬件支持包将 PX4 固件与 MATLAB 和 Simulink 环境进行了集成，使用户可以使用 MATLAB 和 Simulink 进行无人机建模、控制和仿真，并将生成的代码和指令直接发送到 PX4 飞控硬件板上，从而实现无人机的自主飞行任务。

### 3. 基于硬件支持包的 Simulink 模型界面

优点：硬件飞控在环仿真、数据可视化分析



### 4. QGC 地面站



用于刷新 Pixhawk 飞控的固件、刷新传感器、实现控制参数的在线调参等功能。

### 5. 无线数传和蓝牙模块

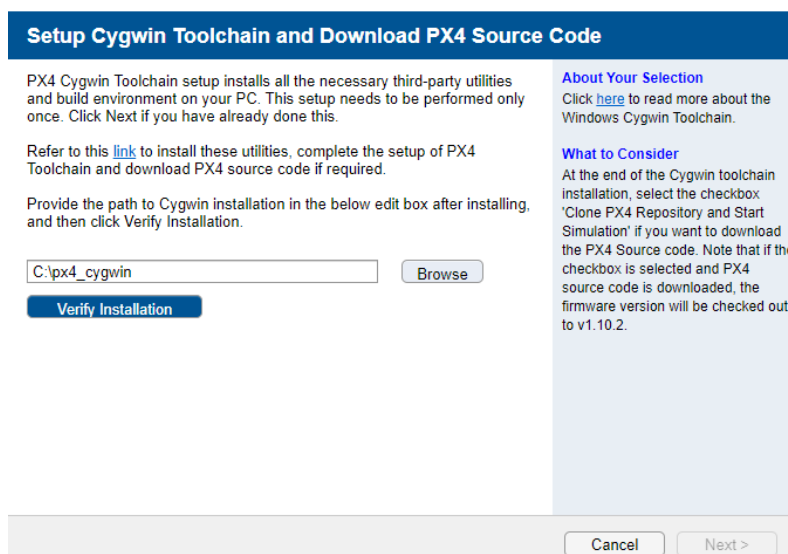
用于在飞控和地面站之间实现无线传输数据和通信。

在 PX4 硬件支持包中，px4\_cygwin 是指针对 Cygwin 环境的编译工具链。Cygwin 是一个在 Windows 操作系统上运行的开源软件，它提供了类似于 Unix 的环境和命令行工具，使得在 Windows 上进行类 Unix 开发成为可能。

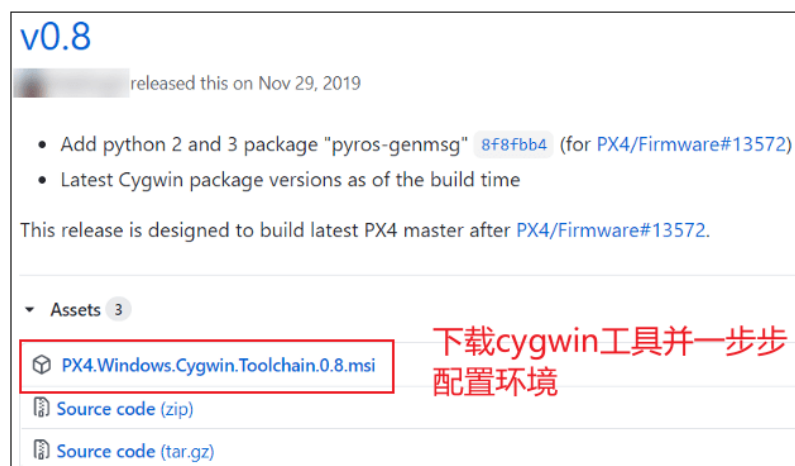


## 6.2 安装及配置步骤

- 1) px4\_cygwin 工具链的作用是为 PX4 飞控固件的编译和构建提供支持。它包括一系列编译工具、库文件和链接器等，用于将 C/C++ 代码编译成可在 PX4 飞控硬件上运行的固件。

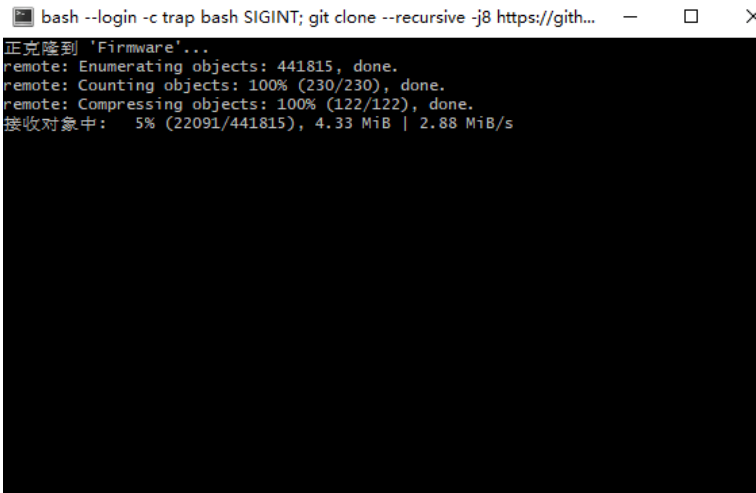


要设置 Cygwin 工具链并下载 PX4®自动驾驶仪 UAV 工具箱支持包中使用的 PX4 源代码，请点击上图的 [link](#) 执行相应安装步骤。



- 2) PX4.Windows.Cygwin.Toolchains 工具箱界面中下载 PX4 源码：



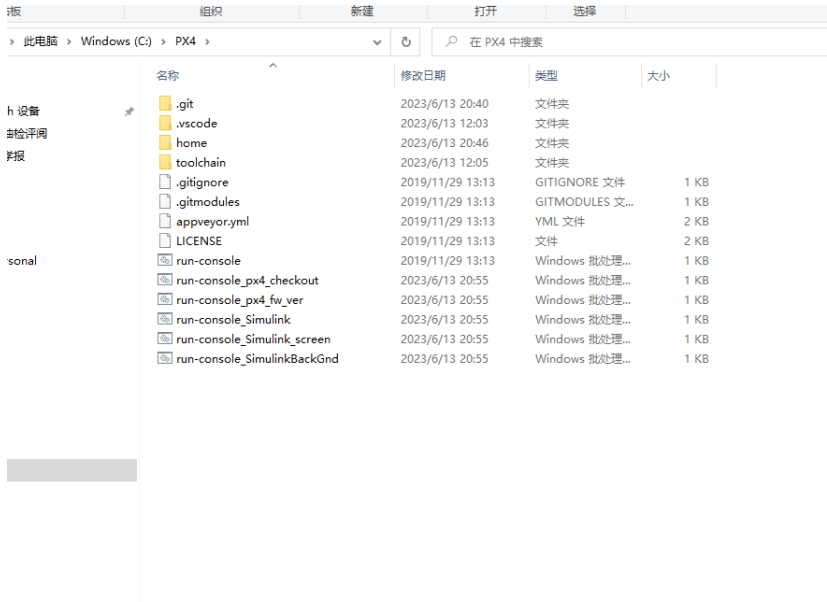


这里克隆 PX4 源码如果因为网络问题始终下载不全，可用别人之前克隆完整可行的 Firmware 源码，然后直接替换 home/Firmware 即可。

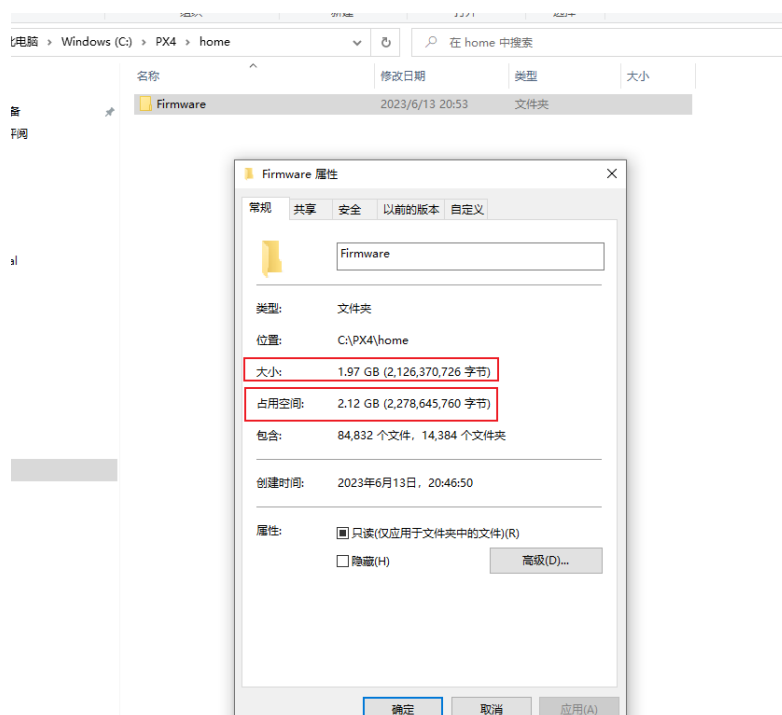
3）切换分支：（一般默认不用操作也可，如果出错最好返回来这里切换分支）

上述 Firmware 下载完整没出错后，需要切换源码分支到与硬件支持包相匹配的一版本，在 cygwin 安装好的文件夹下的 run-console 文件中输入下述命令：

```
git checkout v.10.2
```

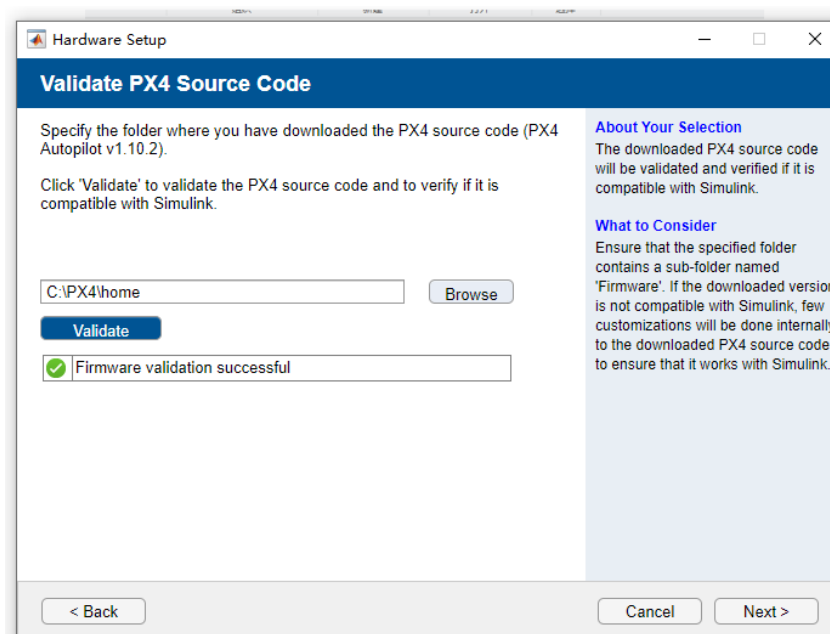


4）检查一下固件是否下载完整全面：



下载全面的 Firmware 文件夹大小如上图，下不全的话就会在校验时报错。

#### 5) 校验 PX4 源码固件：




#### 6) 选择硬件型号：

### Select a PX4 Autopilot and Build Target

PX4 flight stack can be loaded on several Autopilot hardware. Please select a Autopilot hardware and corresponding Build Target Configuration from the drop-down list to start the setup process.

PX4 Autopilot board: PX4 Pixhawk 4

Select Build Target: px4\_fmu-v5\_default



#### About Your Selection

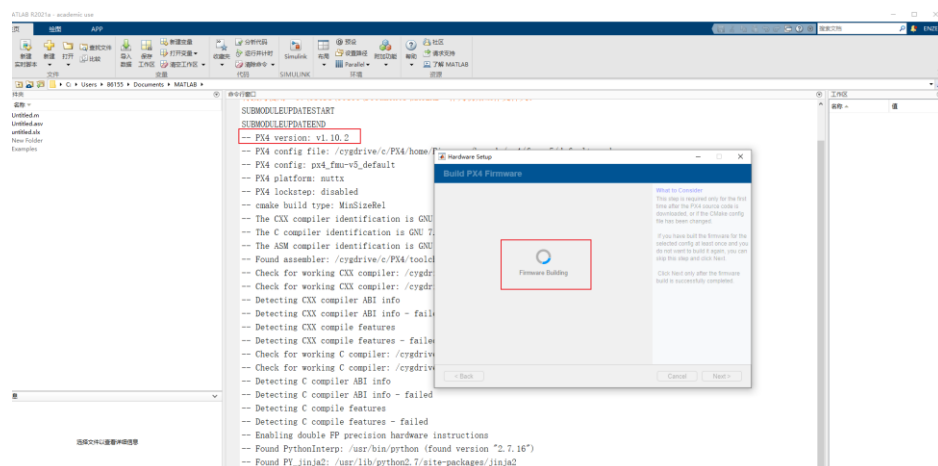
The Pixhawk 4 Autopilot is based on the FMUv5 open hardware design. It has a STM32F765 as the main FMU processor. For more information about the selected Autopilot board, click [here](#).

#### What to Consider

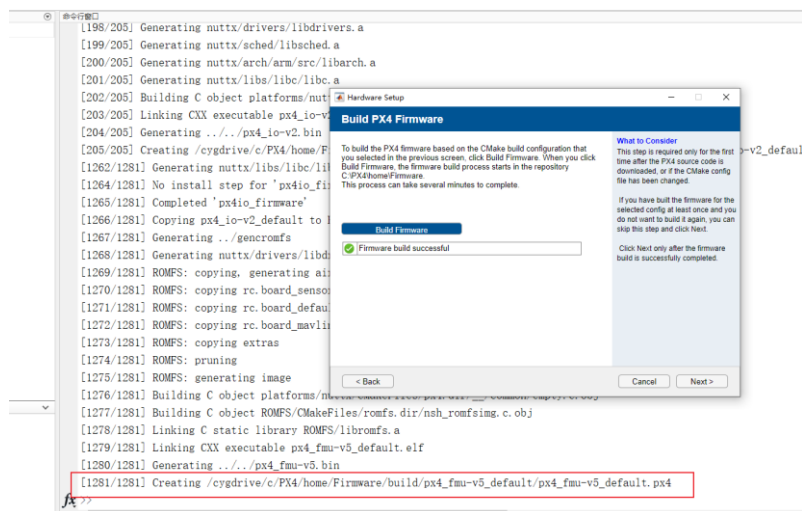
Ensure that the Autopilot board that you select in this list is the same as the one you want to connect to the PC and upload the PX4 firmware.

< BackCancelNext >

7) 编译之前下载好的固件:

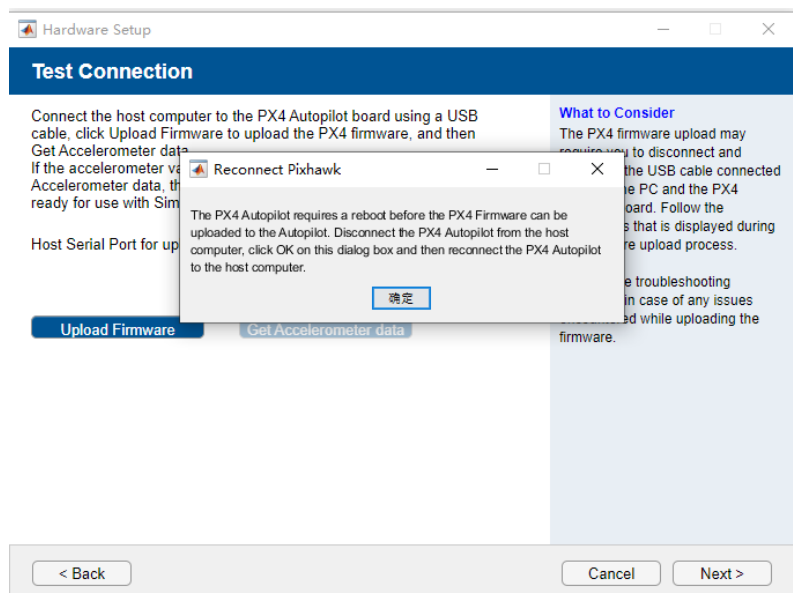


编译成功，如下图:

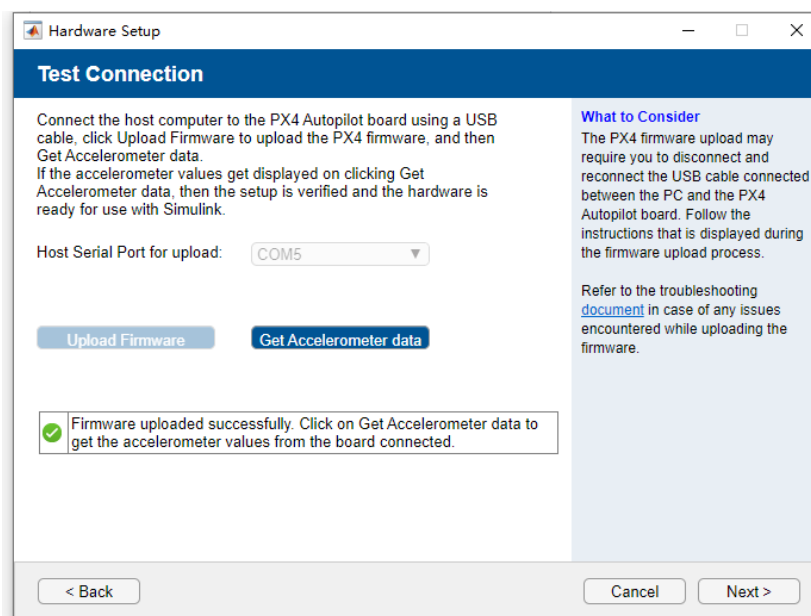


8) 长传编译好的可执行文件给硬件飞控:

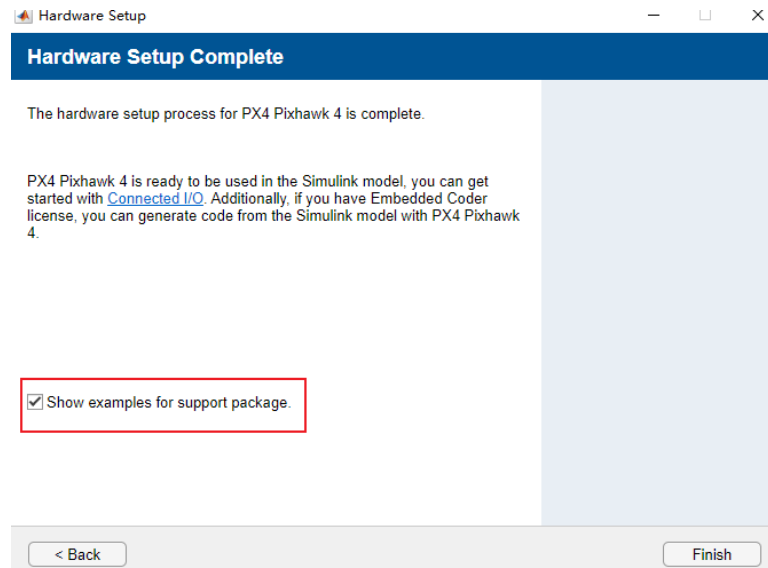
重新插拔一下 USB 数据线后，再点击【确定】。



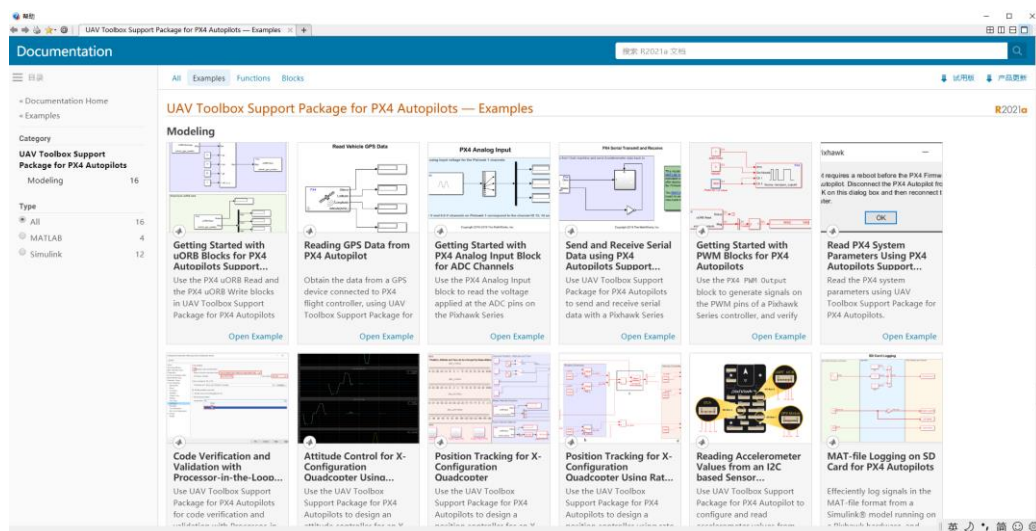
上传过程中如果 chan 长时间卡住，可快速重新插拔一下 USB 线，成功如下图：



9) 在硬件支持包下的 demo 模型上可开发自己的模型算法：



有许多个案例 demo 如下：



## 7 PX4 底层结构梳理

根据 ChatGPT 概述：

PX4 是一个开源的自动驾驶飞行控制系统，旨在为各种无人机和自动驾驶飞行器提供一个灵活、可扩展的平台。PX4 的底层结构包括以下几个主要组件：

- 1) **NuttX**: NuttX 是 PX4 的实时操作系统内核，它提供了对硬件的底层访问和任务管理。NuttX 是一个嵌入式操作系统，可以运行在各种处理器架构上，并提供了实时调度和硬件抽象层。
- 2) **Firmware**: PX4 的固件是在 NuttX 上运行的飞行控制软件。它包括飞行控制算法、传感器驱动程序、状态估计和控制律等功能。PX4 固件支持多种飞行控制器硬件，如 Pixhawk 系列、FMU（飞行管理单元）以及其他兼容的硬件平

|   |  |       |      |
|---|--|-------|------|
| 文件编号  |  | 页码 35 | 版次 A |
| <p>台。</p> <p>3) <b>Middleware:</b> PX4 使用 Robot Operating System (ROS) 作为其通信和消息传递中间件。ROS 提供了一种灵活的机制，用于不同模块之间的数据交换和通信。PX4 使用 ROS 的消息格式来传递传感器数据、状态信息、控制指令等。</p> <p>4) <b>Drivers:</b> PX4 提供了对各种传感器和执行器的驱动程序支持，包括惯性测量单元 (IMU)、全球定位系统 (GPS)、磁力计、遥控器、电调等。<b>这些驱动程序通过底层硬件接口与硬件设备进行通信</b>，并将数据传递给飞行控制算法进行处理。</p> <p>5) <b>Command Stack:</b> PX4 的命令堆栈负责接收和处理来自用户或上层系统的指令。它可以执行自主飞行任务，如航点导航、路径规划和姿态控制。命令堆栈还包括飞行模式管理、飞行任务调度和决策逻辑等功能。</p> <p>6) <b>MAVLink:</b> <b>MAVLink 是一种轻量级的高层抽象的结构化的通信协议</b>，用于在 PX4 和地面站（如 QGroundControl）之间进行通信。它定义了一组消息和命令，用于传递飞行状态、控制指令、传感器数据等信息。PX4 使用 MAVLink 协议与地面站进行实时数据交换和参数配置。</p> <p><b>这些组件共同构成了 PX4 的底层结构。通过它们的协同工作，PX4 能够实现飞行控制、传感器数据获取、任务管理和通信等功能</b>，为无人机和自动驾驶飞行器提供稳定可靠的控制平台。</p> <h3>7.1 MAVLink 通信和 QGC 中的 UDP 无线数传通信的区别</h3> <p>MAVLink 通信和 UDP 无线数传通信是两种不同的通信方式，它们有以下区别：</p> <ul style="list-style-type: none"><li>➤ <b>协议:</b> MAVLink 是一种通信协议，用于定义消息和命令的格式和语义。<b>它是一种高层抽象的协议，可以在不同的物理传输层上运行，如串口、UDP、TCP 等。</b>MAVLink 提供了一种结构化的方式来传递飞行器状态、控制指令、传感器数据等信息。</li><li>➤ <b>QGC:</b> QGC (QGroundControl) 是一种地面站软件，用于与无人机或自动驾驶飞行器进行交互和控制。<b>它(QGC)是基于 MAVLink 协议实现的</b>，并提供了用户界面来显示飞行器状态、配置参数、执行任务等。QGC 可以通过多种物理传输层与飞行器进行通信，包括 UDP 无线数传通信。</li><li>➤ <b>UDP 无线数传通信:</b> UDP (User Datagram Protocol) 是一种传输层协议，提供</li></ul> |  |       |      |

|  |  |       |      |
|--|--|-------|------|
| 文件编号   |  | 页码 36 | 版次 A |
| <p>了无连接的数据传输服务。<b>UDP 无线数传通信指的是使用 UDP 协议在无线网络中进行数据传输。</b>在这种方式下，飞行器和地面站之间通过无线网络建立连接，并使用 UDP 协议进行数据交换。这种通信方式可以实现无线遥控、传输传感器数据等功能。</p> <p>因此，MAVLink 通信是一种基于消息和命令定义的<b>高层</b>抽象通信协议，而 UDP 无线数传通信是一种基于 UDP 协议的无线数据传输方式。<b>MAVLink 通信可以在不同的物理传输层上运行，而 UDP 无线数传通信是其中一种可能的方式</b>，可以用于在无线网络中进行数据传输。QGC 是一个使用 MAVLink 协议进行通信的地面站软件，可以与飞行器进行交互和控制，并支持通过 UDP 无线数传通信与飞行器进行连接。</p> <h2>7.2 PX4 固件内部 uORB 通信和跨不同物理平台间 Mavlink 通信协议区别</h2> <p>PX4 的 uORB（Micro Object Request Broker）是一种用于消息传输的发布-订阅机制，它是 PX4 中用于实现内部模块之间通信的关键组件。</p> <p>uORB 的消息传输机制与 MAVLink 有以下区别：</p> <ol style="list-style-type: none"><li>1.发布-订阅模型：<b><u>uORB 使用发布-订阅模型进行消息传输。</u></b>模块可以发布消息到 uORB 中的话题（Topic），而其他模块可以订阅这些话题以接收消息。这种模型支持一对多的消息传输，即一个发布者可以同时将消息发送给多个订阅者。</li><li>2.实时性：<b>uORB 在设计上注重实时性</b>，它使用了专门的实时内存管理和事件驱动机制，以提供低延迟和可靠的消息传输。这使得 PX4 的各个模块能够在实时系统的要求下进行高效的数据交换和协同工作。</li><li>3.数据结构：<b>uORB 使用定义良好的数据结构描述消息的格式和语义，类似于 ROS 中的消息类型。每个消息类型都有一个对应的数据结构，描述了消息的字段和数据类型。</b>这样，在发布者和订阅者之间传递消息时，可以保持一致的消息格式，减少数据解析和转换的开销。</li><li>4.高度集成：<b><u>uORB 是 PX4 的内部通信机制，紧密集成在 PX4 固件中。</u></b>它与其他 PX4 模块（如姿态估计、控制律、任务调度等）无缝协作，实现了高度集成和高性能的消息传输。</li></ol> <p>相比之下，MAVLink 是一种更通用的通信协议，用于飞行器与地面站之间的数据交换。MAVLink 使用了一种序列化的消息格式，并提供了一组标准的消息和命令定</p> |  |       |      |



|  |  |       |      |
|--|--|-------|------|
| 文件编号   |  | 页码 37 | 版次 A |
| <p>义。MAVLink 主要用于飞行器与地面站之间的通信，支持多种传输方式（如串口、UDP 等），并提供了地面站软件（如 QGroundControl）与飞行控制系统进行交互。<b>总结而言，PX4 的 uORB 是用于 PX4 内部模块之间实时消息传输的机制，采用发布-订阅模型，注重实时性和高度集成。而 MAVLink 是一种通用的通信协议，主要用于飞行器与地面站之间的数据交换，支持多种传输方式(如串口、UDP 等方式)</b>，并提供了地面站软件与飞行控制系统进行交互。</p> |  |       |      |
| <p><b>7.3 NuttX 嵌入式操作系统中的硬件抽象层总结(打开移植思路)</b></p>   |  |       |      |
| <p>1) NuttX 是一个嵌入式操作系统，它为各种处理器架构提供了一个可运行的平台，并提供了实时调度和硬件抽象层。这句话可以通过以下方式理解：</p>  |  |       |      |
| <p>✓ 嵌入式操作系统：NuttX 是专门设计用于嵌入式系统的操作系统。嵌入式系统是指嵌入在其他设备或系统中，用于控制和管理硬件资源的计算机系统。NuttX 提供了一种轻量级、高效的操作系统解决方案，适用于资源受限的嵌入式设备。</p>  |  |       |      |
| <p>✓ 处理器架构：NuttX 可以在各种不同的处理器架构上运行。处理器架构是指计算机系统中处理器的指令集和硬件设计。常见的处理器架构包括 ARM、x86、MIPS 等。<b>NuttX 具有可移植性，可以在不同的处理器架构上进行编译和运行</b>，以满足不同设备的需求。</p>  |  |       |      |
| <p>✓ 实时调度：NuttX 提供了实时调度功能，使得多个任务可以按照一定的优先级和调度策略进行并发执行。实时调度意味着操作系统能够对任务的执行时间进行严格的控制和管理，以满足实时性要求。这对于嵌入式系统中需要及时响应外部事件的应用非常重要。</p>   |  |       |      |
| <p>✓ <b>硬件抽象层：</b>NuttX 提供了硬件抽象层（Hardware Abstraction Layer，HAL），用于将操作系统与底层硬件之间进行解耦。硬件抽象层提供了一组标准的接口和函数，用于访问和控制硬件设备，如 GPIO（通用输入输出）、定时器、串口等。通过硬件抽象层，应用程序可以与底层硬件进行交互，而不需要关心具体的硬件细节和底层驱动程序的实现。</p>   |  |       |      |
| <p>综上所述，<u>NuttX 作为嵌入式操作系统，具备可移植性、实时调度和硬件抽象层的特性，可以在各种处理器架构上运行</u>，并为嵌入式设备提供了一个灵活、可靠的操作系统平台。</p>  |  |       |      |



|   |  |       |      |
|---|--|-------|------|
| 文件编号  |  | 页码 38 | 版次 A |
| <p>2) 可将 NuttX 的硬件抽象层理解为一种中间层，它类似于软硬件驱动程序，但提供了更高级、更抽象的接口，用于解耦操作系统和底层硬件之间的关系，提供了一致性和可移植性。</p> <p>硬件抽象层在 NuttX 中充当了一个中间层，它隐藏了底层硬件的具体实现细节，为操作系统和上层应用程序提供了一组标准的接口和函数。这些接口和函数允许开发者编写与硬件无关的代码，而不必关心底层硬件的具体差异。</p> <p>通过硬件抽象层，开发者可以使用统一的方式访问和控制各种硬件设备，如 GPIO、定时器、串口等。硬件抽象层提供了一系列的函数和数据结构，允许开发者执行与硬件相关的操作，如读写寄存器、配置硬件参数、发送接收数据等。</p> <p>硬件抽象层的目的是提供一种一致的编程接口，使得应用程序的开发更加简化和可移植。通过使用硬件抽象层，开发者可以编写与硬件无关的代码，而不必关注底层硬件的具体细节。这样，在不同的硬件平台上，只需要实现适配该平台的硬件抽象层即可（PX4 源码移植至另外的 STM32 载板上），而无需修改应用程序的代码。</p> |  |       |      |
| <h2>7.4 编译（Makefile 和 CMakeLists.tx）</h2>   |  |       |      |
|   |  |       |      |