# PX4说明文档

## PX4开发文档适用范围

* 获得[最低限度的开发人员设置](https://docs.px4.io/main/en/dev_setup/config_initial.html)，[从源代码构建 PX4](https://docs.px4.io/main/en/dev_setup/building_px4.html)并部署在[众多受支持的自动驾驶仪](https://docs.px4.io/main/en/flight_controller/)上。
* 了解[PX4 系统架构](https://docs.px4.io/main/en/concept/architecture.html)和其他核心概念。【重要】
* 了解如何修改飞行堆栈和中间件：【重要】
  + 修改飞行算法并添加新的[飞行模式](https://docs.px4.io/main/en/concept/flight_modes.html)。
  + 支持新[机身](https://docs.px4.io/main/en/dev_airframes/)。
* 了解如何将 PX4 与新硬件集成：【重要】
  + 支持新的传感器和执行器，包括摄像头、测距仪等。
  + 修改 PX4 以在新的自动驾驶仪硬件上运行。
* [模拟](https://docs.px4.io/main/en/simulation/)、[测试](https://docs.px4.io/main/en/test_and_ci/)和[调试/记录](https://docs.px4.io/main/en/debug/)PX4。【重要】
* 与外部机器人 API 进行通信/集成。

## 基本设备

* 安全飞行员的 **远程控制**
  + Taranis Plus 遥控器（或同等产品）
* **开发计算机**
  + 配备 OSX 10.15 或更高版本的 MacBook Pro（2015 年初及更新机型）
  + 配备 Ubuntu Linux 18.04 或更高版本的 Lenovo Thinkpad 450 (i5)
* **地面控制站**（电脑或平板电脑）：
  + iPad（需要 Wifi 遥测适配器）
  + 任何 MacBook 或 Ubuntu Linux 笔记本电脑（可以是开发计算机）
  + Samsung Note 4 或同等设备（任何最新的 Android 平板电脑或手机，具有足够大的屏幕以有效运行*QGroundControl*）。
* **能够运行 PX4 的车辆**：
  + [获得预制车辆](https://docs.px4.io/main/en/complete_vehicles/)
  + [建立你自己的](https://docs.px4.io/main/en/airframes/)
* **护目镜**
* **Tether**（仅限多旋翼飞行器 - 用于更具风险的测试）

## 车辆配置

安装[QGroundControl 每日构建](https://docs.qgroundcontrol.com/master/en/qgc-user-guide/releases/daily_builds.html)对于**桌面操作系统**。

配置车辆：

1. [安装 PX4 固件](https://docs.px4.io/main/en/config/firmware.html#installing-px4-main-beta-or-custom-firmware)（包括您自己更改的“自定义”固件）。
2. [从机身参考](https://docs.px4.io/main/en/airframes/airframe_reference.html)中选择与您的车辆最匹配的[机身开始](https://docs.px4.io/main/en/config/airframe.html)。
3. [基本配置](https://docs.px4.io/main/en/config/)解释了如何执行基本配置。【重要】
4. [参数配置](https://docs.px4.io/main/en/advanced_config/parameters.html)解释了如何查找和修改各个参数。【重要】

## 构建PX4软件

1. 下载源代码

PX4 源代码存储在[PX4/PX4-Autopilot 的Github 上](https://github.com/PX4/PX4-Autopilot)存储库。

要将*最新*（“主”）版本安装到您​​的计算机上，请在终端中输入以下命令：

|  |
| --- |
| git clone https://github.com/PX4/PX4-Autopilot.git --recursive |

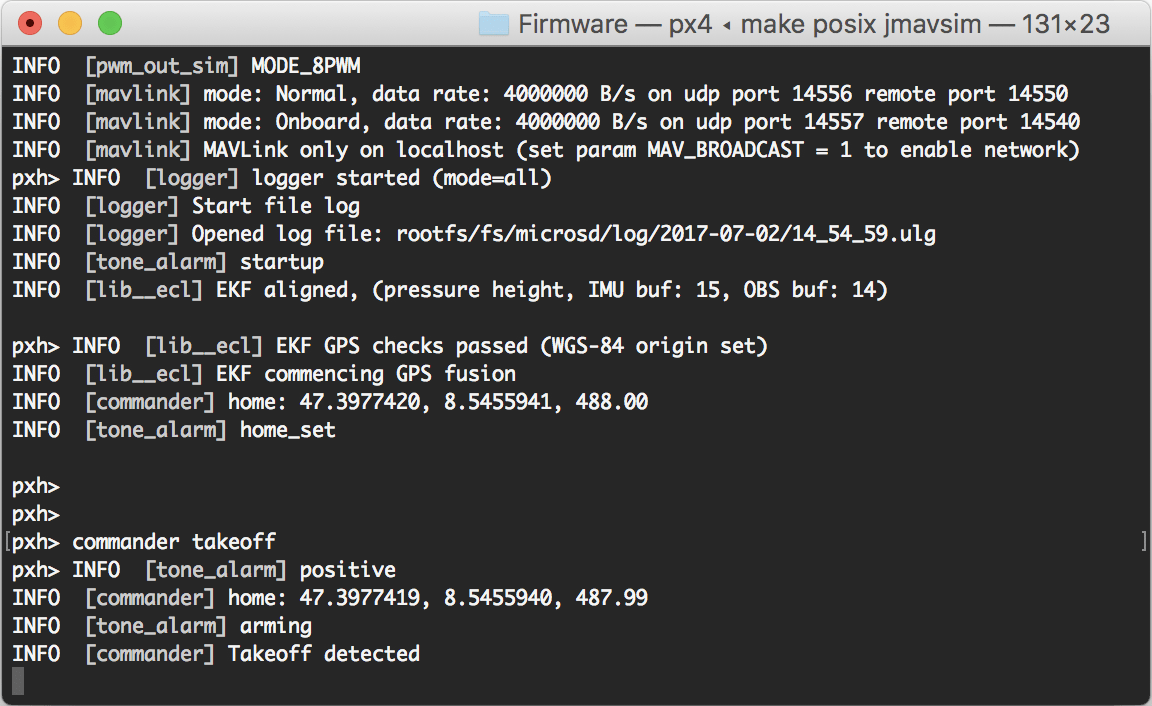
1. 第一次构建（使用jMAVSim模拟器）

首先，我们将使用控制台环境构建一个模拟目标。这使我们能够在转向实际硬件和 IDE 之前验证系统设置。

导航到**PX4-Autopilot**目录并使用以下命令启动[jMAVSim ：](https://docs.px4.io/main/en/sim_jmavsim/)

|  |
| --- |
| make px4\_sitl jmavsim |

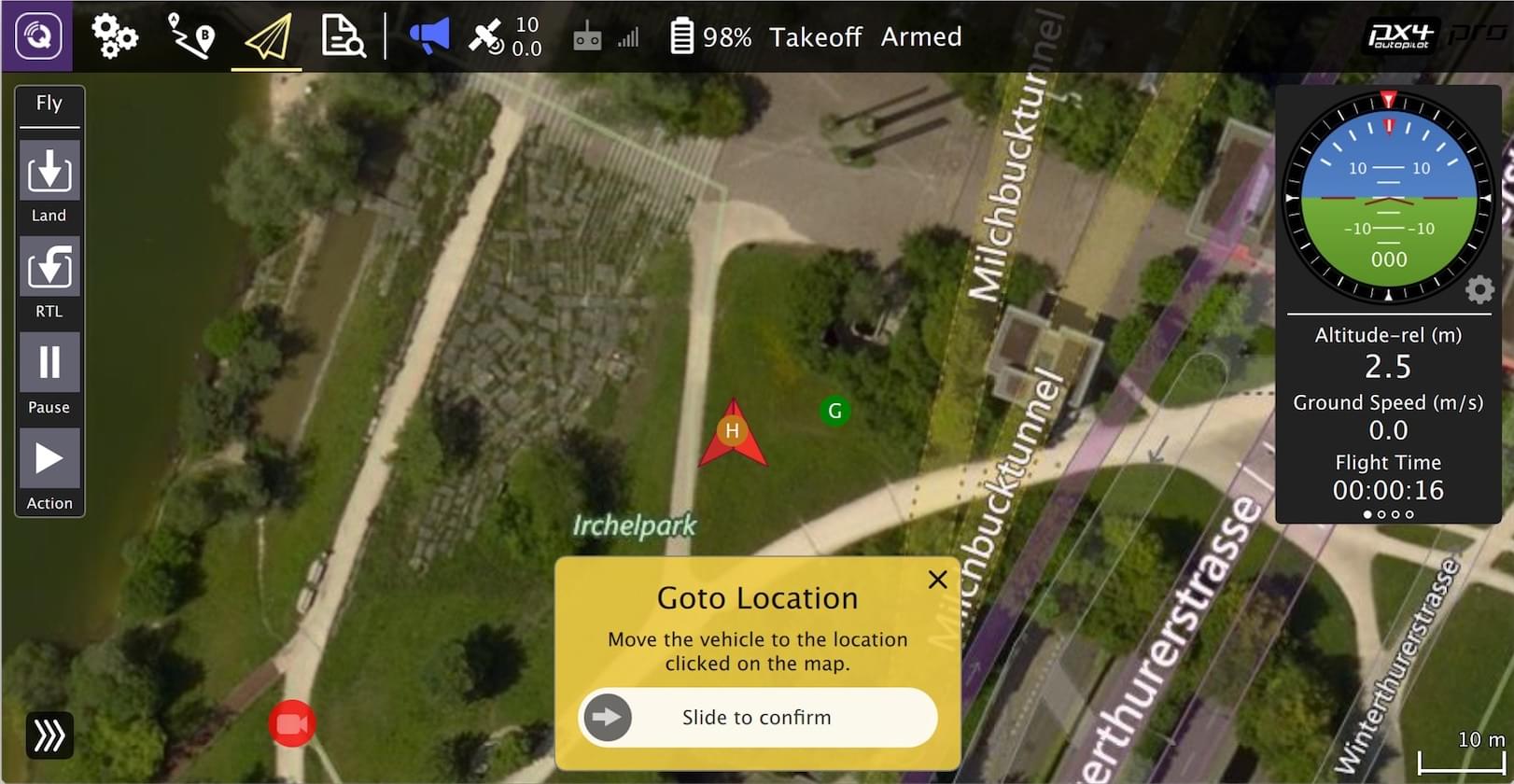
这将打开下面的 PX4 控制台：



【提示：您可能需要在继续之前启动*QGroundControl*，因为默认 PX4 配置需要在起飞前建立地面控制连接。可以[从这里下载](https://docs.qgroundcontrol.com/master/en/getting_started/download_and_install.html)。】

无人机可以通过打字着陆，并且可以通过执行**CTRL+C**（或输入）commander land来停止整个模拟，输入shutdown命令即可。

通过地面控制站进行模拟飞行更接近车辆的真实操作。在车辆飞行时（起飞飞行模式）单击地图中的某个位置并启用滑块。这将重新定位车辆。



【提示：PX4 可与许多其他[模拟器](https://docs.px4.io/main/en/simulation/)一起使用，包括[Gazebo](https://docs.px4.io/main/en/sim_gazebo_gz/)、[Gazebo Classic](https://docs.px4.io/main/en/sim_gazebo_classic/)和[AirSim](https://docs.px4.io/main/en/sim_airsim/)。这些也是从*make*开始的，例如构建时使用如下代码：

|  |
| --- |
| make px4\_sitl gazebo-classic |

】

1. 基于NuttX/Pixhawk的主板

3,1为NuttX构建

要构建基于 NuttX 或 Pixhawk 的板，请导航到**PX4-Autopilot**目录，然后make使用板的构建目标进行调用。例如，要构建[Pixhawk 4](https://docs.px4.io/main/en/flight_controller/pixhawk4.html)硬件，您可以使用以下命令：

|  |
| --- |
| cd PX4-Autopilot  make px4\_fmu-v5\_default |

成功的运行将以类似的输出结束：

|  |
| --- |
| -- Build files have been written to: /home/youruser/src/PX4-Autopilot/build/px4\_fmu-v4\_default  [954/954] Creating /home/youruser/src/PX4-Autopilot/build/px4\_fmu-v4\_default/px4\_fmu-v4\_default.px4 |

【笔记：构建得到的固件将存储在PX4-Autopilot/build/文件夹中】

构建目标的第一部分px4\_fmu-v4指示固件的目标飞行控制器硬件。在这种情况下，后缀\_default表示固件*配置*，例如支持或省略特定功能。

【笔记：后缀\_default是可选的。例如，make px4\_fmu-v5并px4\_fmu-v5\_default产生相同的固件。】

以下列表显示了[Pixhawk 标准](https://docs.px4.io/main/en/flight_controller/autopilot_pixhawk_standard.html)板的构建命令：

* [Holybro Pixhawk 6X (FMUv6X)](https://docs.px4.io/main/en/flight_controller/pixhawk6x.html) :make px4\_fmu-v6x\_default
* [Holybro Pixhawk 6C (FMUv6C)](https://docs.px4.io/main/en/flight_controller/pixhawk6c.html) :make px4\_fmu-v6c\_default
* [Holybro Pixhawk 6C 迷你 (FMUv6C)](https://docs.px4.io/main/en/flight_controller/pixhawk6c_mini.html)：make px4\_fmu-v6c\_default
* [Holybro Pix32 v6 (FMUv6C)](https://docs.px4.io/main/en/flight_controller/holybro_pix32_v6.html) :make px4\_fmu-v6c\_default
* [Holybro Pixhawk 5X (FMUv5X)](https://docs.px4.io/main/en/flight_controller/pixhawk5x.html) :make px4\_fmu-v5x\_default
* [Pixhawk 4 (FMUv5)](https://docs.px4.io/main/en/flight_controller/pixhawk4.html)：make px4\_fmu-v5\_default
* [Pixhawk 4 迷你 (FMUv5)](https://docs.px4.io/main/en/flight_controller/pixhawk4_mini.html)：make px4\_fmu-v5\_default
* [CUAV V5+ (FMUv5)](https://docs.px4.io/main/en/flight_controller/cuav_v5_plus.html) :make px4\_fmu-v5\_default
* [CUAV V5 纳米（FMUv5）](https://docs.px4.io/main/en/flight_controller/cuav_v5_nano.html)：make px4\_fmu-v5\_default
* [Pixracer（FMUv4）](https://docs.px4.io/main/en/flight_controller/pixracer.html)：make px4\_fmu-v4\_default
* [Pixhawk 3 Pro](https://docs.px4.io/main/en/flight_controller/pixhawk3_pro.html)：make px4\_fmu-v4pro\_default
* [迷你皮克斯霍克](https://docs.px4.io/main/en/flight_controller/pixhawk_mini.html)：make px4\_fmu-v3\_default
* [Pixhawk 2（立方体黑色）（FMUv3）](https://docs.px4.io/main/en/flight_controller/pixhawk-2.html)：make px4\_fmu-v3\_default
* [mRo Pixhawk (FMUv3)](https://docs.px4.io/main/en/flight_controller/mro_pixhawk.html) : make px4\_fmu-v3\_default(支持 2MB 闪存)
* [Holybro pix32 (FMUv2)](https://docs.px4.io/main/en/flight_controller/holybro_pix32.html) :make px4\_fmu-v2\_default
* [Pixfalcon（FMUv2）](https://docs.px4.io/main/en/flight_controller/pixfalcon.html)：make px4\_fmu-v2\_default
* [Dropix（FMUv2）](https://docs.px4.io/main/en/flight_controller/dropix.html)：make px4\_fmu-v2\_default
* [Pixhawk 1 (FMUv2)](https://docs.px4.io/main/en/flight_controller/pixhawk.html)：make px4\_fmu-v2\_default
* 具有 2 MB 闪存的 Pixhawk 1：make px4\_fmu-v3\_default

[各个飞行控制器板](https://docs.px4.io/main/en/flight_controller/)的文档中提供了非 Pixhawk NuttX 战斗控制器（以及所有其他板）的构建命令。

3.2上传固件

附加upload到 make 命令以通过 USB 将编译的二进制文件上传到自动驾驶仪硬件。例如：

|  |
| --- |
| make px4\_fmu-v4\_default upload |

成功运行将以以下输出结束：

|  |
| --- |
| Erase : [====================] 100.0%  Program: [====================] 100.0%  Verify : [====================] 100.0%  Rebooting.  [100%] Built target upload |

1. 故障排除

4.1一般构建错误

许多构建问题是由不匹配的子模块或不完全清理的构建环境引起的。更新子模块并执行以下操作distclean可以修复这些类型的错误：

|  |
| --- |
| git submodule update --recursive  make distclean |

4.2其他错误处理方式

见如下网址罗列项目：

|  |
| --- |
| https://docs.px4.io/main/en/dev\_setup/building\_px4.html |

## 首次应用教程（Hello Sky）

1. 先决条件

【提示：如果能够完成之前的安装步骤，此先决条件以满足，可以跳过】

您将需要以下内容：

* [PX4 SITL 模拟器](https://docs.px4.io/main/en/simulation/) *或*PX4[兼容的飞行控制器](https://docs.px4.io/main/en/flight_controller/)。
* 适用于所需目标的[PX4 开发工具链。](https://docs.px4.io/main/en/dev_setup/dev_env.html)
* 从 Github[下载 PX4 源代码](https://docs.px4.io/main/en/dev_setup/building_px4.html#download-the-px4-source-code)

PX4-Autopilot源代码[/src/examples/px4\_simple\_app](https://github.com/PX4/PX4-Autopilot/tree/main/src/examples/px4_simple_app)目录包含本教程的完整版本，如果您遇到困难，可以查看该版本。

* 重命名（或删除）**px4\_simple\_app**目录。

1. 最小应用构建步骤

在本节中，我们创建一个仅打印的最小应用程序Hello Sky!。它由一个*C*文件和一个*cmake*定义（告诉工具链如何构建应用程序）组成。如下是完整的构建步骤：

**Step1：**创建一个新目录PX4-Autopilot/src/examples/px4\_simple\_app。

**Step2：**在该目录中创建一个名为px4\_simple\_app.c的新 C 文件：

将如下默认标题复制到页面顶部。这应该出现在所有贡献的文件中！

|  |
| --- |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*  \* Copyright (c) 2012-2022 PX4 Development Team. All rights reserved.  \*  \* Redistribution and use in source and binary forms, with or without  \* modification, are permitted provided that the following conditions  \* are met:  \*  \* 1. Redistributions of source code must retain the above copyright  \* notice, this list of conditions and the following disclaimer.  \* 2. Redistributions in binary form must reproduce the above copyright  \* notice, this list of conditions and the following disclaimer in  \* the documentation and/or other materials provided with the  \* distribution.  \* 3. Neither the name PX4 nor the names of its contributors may be  \* used to endorse or promote products derived from this software  \* without specific prior written permission.  \*  \* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS  \* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT  \* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS  \* FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE  \* COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,  \* INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,  \* BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS  \* OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED  \* AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT  \* LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN  \* ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE  \* POSSIBILITY OF SUCH DAMAGE.  \*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |

将以下代码复制到默认标头下方。这应该出现在所有贡献的文件中！

|  |
| --- |
| /\*\*  \* @file px4\_simple\_app.c  \* Minimal application example for PX4 autopilot  \*  \* @author Example User <mail@example.com>  \*/  #include <px4\_platform\_common/log.h>  \_\_EXPORT int px4\_simple\_app\_main(int argc, char \*argv[]);  int px4\_simple\_app\_main(int argc, char \*argv[])  {  PX4\_INFO("Hello Sky!");  return OK;  } |

【提示：主函数必须命名<module\_name>\_main并从模块中导出，如上面代码所示。】

【提示：PX4\_INFO相当于printf（PX4 shell内）（包含在px4\_platform\_common/log.h中）。有不同的日志级别：PX4\_INFO、PX4\_WARN、PX4\_ERR、PX4\_DEBUG。警告和错误会另外添加到[ULog](https://docs.px4.io/main/en/dev_log/ulog_file_format.html)中并显示在[Flight Review上](https://logs.px4.io/)。】

**Step3：**创建并打开一个名为CMakeLists.txt的新*cmake*定义文件。复制以下文本：

|  |
| --- |
| ############################################################################  #  # Copyright (c) 2015 PX4 Development Team. All rights reserved.  #  # Redistribution and use in source and binary forms, with or without  # modification, are permitted provided that the following conditions  # are met:  #  # 1. Redistributions of source code must retain the above copyright  # notice, this list of conditions and the following disclaimer.  # 2. Redistributions in binary form must reproduce the above copyright  # notice, this list of conditions and the following disclaimer in  # the documentation and/or other materials provided with the  # distribution.  # 3. Neither the name PX4 nor the names of its contributors may be  # used to endorse or promote products derived from this software  # without specific prior written permission.  #  # THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS  # "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT  # LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS  # FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE  # COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,  # INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,  # BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS  # OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED  # AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT  # LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN  # ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE  # POSSIBILITY OF SUCH DAMAGE.  #  ############################################################################  px4\_add\_module(  MODULE examples\_\_px4\_simple\_app  MAIN px4\_simple\_app  STACK\_MAIN 2000  SRCS  px4\_simple\_app.c  DEPENDS  ) |

该px4\_add\_module()方法根据模块描述构建静态库。

* 该MODULE块是模块的固件唯一名称（按照惯例，模块名称的父目录前缀为src）。
* 该MAIN块列出了模块的入口点，该入口点向 NuttX 注册命令，以便可以从 PX4 shell 或 SITL 控制台调用它。

【提示：该px4\_add\_module()格式记录在[PX4-Autopilot/cmake/px4\_add\_module.cmake中](https://github.com/PX4/PX4-Autopilot/blob/main/cmake/px4_add_module.cmake)。】

【笔记：如果您指定DYNAMIC为 的选项px4\_add\_module，则会在 POSIX 平台上创建*共享库*而不是静态库（无需重新编译 PX4 即可加载这些库，并作为二进制文件而不是源代码共享给其他人）。您的应用程序不会成为内置命令，但最终会出现在一个名为examples\_\_px4\_simple\_app.px4mod. 然后，您可以使用以下命令在运行时加载文件来运行命令dyn：dyn ./examples\_\_px4\_simple\_app.px4mod】

**Step4：**创建并打开一个名为Kconfig的新*Kconfig*定义文件，并定义命名符号（请参阅[Kconfig 命名约定](https://docs.px4.io/main/en/hardware/porting_guide_config.html#px4-kconfig-symbol-naming-convention)）。复制以下文本：

|  |
| --- |
| menuconfig EXAMPLES\_PX4\_SIMPLE\_APP  bool “px4\_simple\_app”  default n  ---help---  Enable support for px4\_simple\_app |

**Step5：**构建应用程序/固件：申请现已完成。为了运行它，您首先需要确保它是作为 PX4 的一部分构建的。*应用程序将添加到适合您的目标的板级px4board*文件中的构建/固件中：

* PX4 SITL（模拟器）：[PX4-Autopilot/boards/px4/sitl/default.px4board](https://github.com/PX4/PX4-Autopilot/blob/main/boards/px4/sitl/default.px4board)
* Pixhawk v1/2：[PX4-Autopilot/boards/px4/fmu-v2/default.px4board](https://github.com/PX4/PX4-Autopilot/blob/main/boards/px4/fmu-v2/default.px4board)
* Pixracer (px4/fmu-v4): [PX4-Autopilot/boards/px4/fmu-v4/default.px4board](https://github.com/PX4/PX4-Autopilot/blob/main/boards/px4/fmu-v4/default.px4board)
* 其他板的px4board文件可以在[PX4-Autopilot/boards/](https://github.com/PX4/PX4-Autopilot/tree/main/boards)中找到

要将应用程序编译到固件中，请把以下代码添加到px4board类型文件中最后一行：

|  |
| --- |
| CONFIG\_EXAMPLES\_PX4\_SIMPLE\_APP=y |

目的是在*px4board*文件中添加相应的 Kconfig 键。或者运行[boardconfig](https://docs.px4.io/main/en/hardware/porting_guide_config.html#px4-menuconfig-setup) make px4\_fmu-v4\_default boardconfig（在*px4board*文件中添加相应的 Kconfig 键【提示：不推荐使用，主要是我也不会操作这个界面】）：

|  |
| --- |
| examples --->  [x] PX4 Simple app ---- |

**Step6：**测试应用程序（硬件）【提示：最开始的用的时候不建议去弄硬件测试，后面有仿真软件测试部分】

**Step6.1：**将固件上传到您的主板

启用上传器，然后重置开发板：

* Pixhawk v1/2：make px4\_fmu-v2\_default upload
* 皮克斯霍克 v3：make px4\_fmu-v4\_default upload

在重置板之前，它应该打印一些编译消息，并在最后打印：

|  |
| --- |
| Loaded firmware for X,X, waiting for the bootloader... |

一旦板重置并上传，它就会打印：

|  |
| --- |
| Erase : [====================] 100.0%  Program: [====================] 100.0%  Verify : [====================] 100.0%  Rebooting.  [100%] Built target upload |

**Step6.2：**连接控制台

现在通过串行或 USB连接到[系统控制台。](https://docs.px4.io/main/en/debug/system_console.html)按**ENTER**键将弹出 shell 提示符：

|  |
| --- |
| nsh> |

输入“help”并按 ENTER 键，它就会打印如下结果：

|  |
| --- |
| nsh> help  help usage: help [-v] [<cmd>]  [ df kill mkfifo ps sleep  ? echo losetup mkrd pwd test  cat exec ls mh rm umount  cd exit mb mount rmdir unset  cp free mkdir mv set usleep  dd help mkfatfs mw sh xd  Builtin Apps:  reboot  perf  top  ..  px4\_simple\_app  ..  sercon  serdis |

请注意，它px4\_simple\_app现在是可用命令的一部分。通过键入px4\_simple\_app并 ENTER 启动它，将得到如下结果：

|  |
| --- |
| nsh> px4\_simple\_app  Hello Sky! |

该应用程序现在已在系统中正确注册，并且可以扩展以实际执行有用的任务。

**Step7：**测试应用程序（SITL）【提示：仿真软件测试部分】

如果您使用 SITL，则*PX4 控制台*会自动启动（请参阅[构建代码 > 首次构建（使用 jMAVSim 模拟器）](https://docs.px4.io/main/en/dev_setup/building_px4.html#first-build-using-the-jmavsim-simulator)）。与*nsh 控制台*（请参阅上一节）一样，您可以键入help以查看内置应用程序的列表。

输入px4\_simple\_app以运行最小应用程序，将得到如下结果：

|  |
| --- |
| pxh> px4\_simple\_app  INFO [px4\_simple\_app] Hello Sky! |

现在可以扩展该应用程序以实际执行有用的任务。

## 订阅传感器数据

为了做一些有用的事情，应用程序需要订阅输入并发布输出（例如电机或伺服命令）。

【提示：请先完成最小应用程序px4\_simple\_app，然后在此基础上修改px4\_simple\_app.c中的代码（或者不修改，原先的代码就是这节所用到的代码），完成该节内容，】

【提示：PX4 硬件抽象的优势在这里发挥出来！无需以任何方式与传感器驱动程序交互，如果板或传感器更新，也无需更新您的应用程序。】

应用程序之间的各个消息通道称为[topics](https://docs.px4.io/main/en/middleware/uorb.html)（主题）。[在本教程中，我们对topics中的SensorCombined](https://github.com/PX4/PX4-Autopilot/blob/main/msg/SensorCombined.msg)感兴趣 ，它保存整个系统的同步传感器数据。

**Step1：**订阅主题

订阅主题很简单，如下代码所示：

|  |
| --- |
| #include <uORB/topics/sensor\_combined.h>  ..  int sensor\_sub\_fd = orb\_subscribe(ORB\_ID(sensor\_combined)); |

这sensor\_sub\_fd是一个主题句柄，可用于非常有效地执行对新数据的阻塞等待。当前线程进入睡眠状态，一旦有新数据可用，就会被调度程序自动唤醒，等待时不会消耗任何 CPU 周期。为此，我们使用POSIX 系统调用[poll()](http://pubs.opengroup.org/onlinepubs/007908799/xsh/poll.html)。

添加poll()到订阅看起来像（*伪代码，在下面查找完整的实现*）：

|  |
| --- |
| #include <poll.h>  #include <uORB/topics/sensor\_combined.h>  ..  int sensor\_sub\_fd = orb\_subscribe(ORB\_ID(sensor\_combined));  /\* one could wait for multiple topics with this technique, just using one here \*/  px4\_pollfd\_struct\_t fds[] = {  { .fd = sensor\_sub\_fd, .events = POLLIN },  };  while (true) {  /\* wait for sensor update of 1 file descriptor for 1000 ms (1 second) \*/  int poll\_ret = px4\_poll(fds, 1, 1000);  ..  if (fds[0].revents & POLLIN) {  /\* obtained data for the first file descriptor \*/  struct sensor\_combined\_s raw;  /\* copy sensors raw data into local buffer \*/  orb\_copy(ORB\_ID(sensor\_combined), sensor\_sub\_fd, &raw);  PX4\_INFO("Accelerometer:\t%8.4f\t%8.4f\t%8.4f",  (double)raw.accelerometer\_m\_s2[0],  (double)raw.accelerometer\_m\_s2[1],  (double)raw.accelerometer\_m\_s2[2]);  }  } |

输入以下内容再次编译应用程序：

|  |
| --- |
| make |

**Step2：**测试uORB订阅

最后一步是通过在 nsh shell 中键入以下内容将应用程序作为后台进程/任务启动：

|  |
| --- |
| px4\_simple\_app & |

您的应用程序将在控制台中显示 5 个传感器值，然后退出：

|  |
| --- |
| [px4\_simple\_app] Accelerometer: 0.0483 0.0821 0.0332  [px4\_simple\_app] Accelerometer: 0.0486 0.0820 0.0336  [px4\_simple\_app] Accelerometer: 0.0487 0.0819 0.0327  [px4\_simple\_app] Accelerometer: 0.0482 0.0818 0.0323  [px4\_simple\_app] Accelerometer: 0.0482 0.0827 0.0331  [px4\_simple\_app] Accelerometer: 0.0489 0.0804 0.0328 |

【提示：[完整应用程序的模块模板](https://docs.px4.io/main/en/modules/module_template.html)可用于编写可从命令行控制的后台进程。】

**Step3：**发布数据

要使用计算的输出，下一步是发布结果。下面我们展示如何发布attitude主题。

【笔记：我们之所以选择attitude主题，是因为我们知道mavlink应用程序会将其转发到地面控制站，这提供了一种查看结果的简单方法。】

界面非常简单：初始化要发布的主题数据结构体，并宣传该主题：

|  |
| --- |
| #include <uORB/topics/vehicle\_attitude.h>  ..  /\* advertise attitude topic \*/  struct vehicle\_attitude\_s att;  memset(&att, 0, sizeof(att));  orb\_advert\_t att\_pub\_fd = orb\_advertise(ORB\_ID(vehicle\_attitude), &att); |

在主循环中，只要准备好就发布信息：

|  |
| --- |
| orb\_publish(ORB\_ID(vehicle\_attitude), att\_pub\_fd, &att); |

**Step4：**运行完整示例

最后使用下面代码运行你的应用程序：

|  |
| --- |
| px4\_simple\_app |

如果启动QGroundControl，您可以检查实时图中的传感器值([Analyze > MAVLink Inspector](https://docs.qgroundcontrol.com/master/en/analyze_view/mavlink_inspector.html))。【提示：与此程序无关，QGroundControl自带有的功能】

**附录**（完整的示例代码）

|  |
| --- |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*  \* Copyright (c) 2012-2019 PX4 Development Team. All rights reserved.  \*  \* Redistribution and use in source and binary forms, with or without  \* modification, are permitted provided that the following conditions  \* are met:  \*  \* 1. Redistributions of source code must retain the above copyright  \* notice, this list of conditions and the following disclaimer.  \* 2. Redistributions in binary form must reproduce the above copyright  \* notice, this list of conditions and the following disclaimer in  \* the documentation and/or other materials provided with the  \* distribution.  \* 3. Neither the name PX4 nor the names of its contributors may be  \* used to endorse or promote products derived from this software  \* without specific prior written permission.  \*  \* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS  \* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT  \* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS  \* FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE  \* COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,  \* INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,  \* BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS  \* OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED  \* AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT  \* LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN  \* ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE  \* POSSIBILITY OF SUCH DAMAGE.  \*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*\*  \* @file px4\_simple\_app.c  \* Minimal application example for PX4 autopilot  \*  \* @author Example User <mail@example.com>  \*/  #include <px4\_platform\_common/px4\_config.h>  #include <px4\_platform\_common/tasks.h>  #include <px4\_platform\_common/posix.h>  #include <unistd.h>  #include <stdio.h>  #include <poll.h>  #include <string.h>  #include <math.h>  #include <uORB/uORB.h>  #include <uORB/topics/sensor\_combined.h>  #include <uORB/topics/vehicle\_attitude.h>  \_\_EXPORT int px4\_simple\_app\_main(int argc, char \*argv[]);  int px4\_simple\_app\_main(int argc, char \*argv[])  {  PX4\_INFO("Hello Sky!");  /\* subscribe to sensor\_combined topic \*/  int sensor\_sub\_fd = orb\_subscribe(ORB\_ID(sensor\_combined));  /\* limit the update rate to 5 Hz \*/  orb\_set\_interval(sensor\_sub\_fd, 200);  /\* advertise attitude topic \*/  struct vehicle\_attitude\_s att;  memset(&att, 0, sizeof(att));  orb\_advert\_t att\_pub = orb\_advertise(ORB\_ID(vehicle\_attitude), &att);  /\* one could wait for multiple topics with this technique, just using one here \*/  px4\_pollfd\_struct\_t fds[] = {  { .fd = sensor\_sub\_fd, .events = POLLIN },  /\* there could be more file descriptors here, in the form like:  \* { .fd = other\_sub\_fd, .events = POLLIN },  \*/  };  int error\_counter = 0;  for (int i = 0; i < 5; i++) {  /\* wait for sensor update of 1 file descriptor for 1000 ms (1 second) \*/  int poll\_ret = px4\_poll(fds, 1, 1000);  /\* handle the poll result \*/  if (poll\_ret == 0) {  /\* this means none of our providers is giving us data \*/  PX4\_ERR("Got no data within a second");  } else if (poll\_ret < 0) {  /\* this is seriously bad - should be an emergency \*/  if (error\_counter < 10 || error\_counter % 50 == 0) {  /\* use a counter to prevent flooding (and slowing us down) \*/  PX4\_ERR("ERROR return value from poll(): %d", poll\_ret);  }  error\_counter++;  } else {  if (fds[0].revents & POLLIN) {  /\* obtained data for the first file descriptor \*/  struct sensor\_combined\_s raw;  /\* copy sensors raw data into local buffer \*/  orb\_copy(ORB\_ID(sensor\_combined), sensor\_sub\_fd, &raw);  PX4\_INFO("Accelerometer:\t%8.4f\t%8.4f\t%8.4f",  (double)raw.accelerometer\_m\_s2[0],  (double)raw.accelerometer\_m\_s2[1],  (double)raw.accelerometer\_m\_s2[2]);  /\* set att and publish this information for other apps  the following does not have any meaning, it's just an example  \*/  att.q[0] = raw.accelerometer\_m\_s2[0];  att.q[1] = raw.accelerometer\_m\_s2[1];  att.q[2] = raw.accelerometer\_m\_s2[2];  orb\_publish(ORB\_ID(vehicle\_attitude), att\_pub, &att);  }  /\* there could be more file descriptors here, in the form like:  \* if (fds[1..n].revents & POLLIN) {}  \*/  }  }  PX4\_INFO("exiting");  return 0;  } |

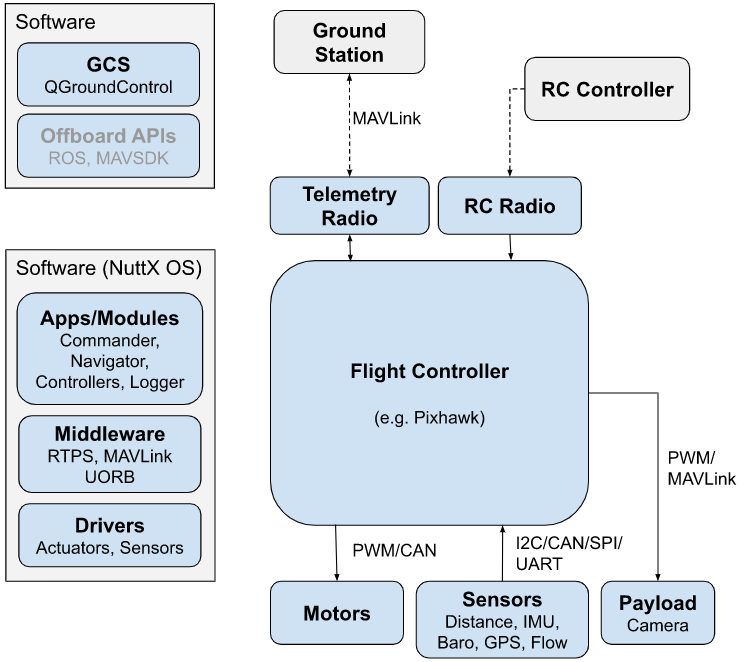
## PX4系统架构

以下部分提供了两个“典型”PX4 系统的 PX4 硬件和软件堆栈的高级概述；一个只有一个飞行控制器，另一个有一个飞行控制器和一台配套计算机（也称为“任务计算机”）。

【笔记：PX4架构[概述](https://docs.px4.io/main/en/concept/architecture.html)提供了有关飞行堆栈和中间件的信息。[ROS](https://docs.px4.io/main/en/ros/)和[MAVSDK](https://mavsdk.mavlink.io/main/en/)涵盖了 Offboard API 。】

**Part1：**飞行控制器（仅限）

下图提供了基于飞行控制器的典型“简单”PX4 系统的高级概述。



硬件包括：

* [飞行控制器](https://docs.px4.io/main/en/flight_controller/)（运行 PX4 飞行堆栈）。这通常包括内部 IMU、指南针和气压计。
* [Motor ESCs](https://docs.px4.io/main/en/peripherals/esc_motors.html) 连接到[PWM 输出](https://docs.px4.io/main/en/peripherals/pwm_escs_and_servo.html)、[DroneCAN](https://docs.px4.io/main/en/dronecan/escs.html)（DroneCAN 允许双向通信，而不是如图所示的单向通信）或其他总线的
* 通过 I2C、SPI、CAN、UART 等连接的传感器（[GPS](https://docs.px4.io/main/en/gps_compass/)、[指南针](https://docs.px4.io/main/en/gps_compass/)、距离传感器、气压计、光流、气压计、ADSB 应答器等）。
* [相机](https://docs.px4.io/main/en/peripherals/camera.html)或其他有效负载。相机可以连接到 PWM 输出或通过 MAVLink进行连接。
* 用于连接地面站计算机/软件的遥测无线电[Telemetry radios](https://docs.px4.io/main/en/telemetry/)。
* 用于手动控制的RC控制系统[RC Control System](https://docs.px4.io/main/en/getting_started/rc_transmitter_receiver.html)。

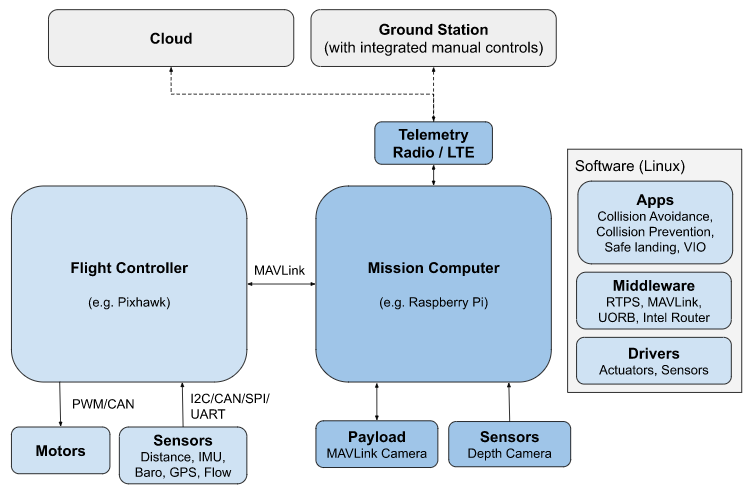
软件包括：

上图的左侧显示了软件堆栈，它与该图的硬件部分（大约）水平对齐。

* 地面站计算机通常运行[QGroundControl](https://docs.px4.io/main/en/getting_started/px4_basic_concepts.html#qgroundcontrol)（或一些其他地面站软件）。它还可以运行[MAVSDK](https://mavsdk.mavlink.io/)等机器人软件或 [ROS](https://docs.px4.io/main/en/ros/)。
* 飞行控制器上运行的 PX4 飞行堆栈包括[驱动程序](https://docs.px4.io/main/en/modules/modules_driver.html)、[通信模块](https://docs.px4.io/main/en/modules/modules_communication.html)、[控制器](https://docs.px4.io/main/en/modules/modules_controller.html)、[估算器](https://docs.px4.io/main/en/modules/modules_controller.html)以及其他[中间件和系统模块](https://docs.px4.io/main/en/modules/modules_main.html)。

**Part2：**飞行控制器和配套计算机

下图显示了一个 PX4 系统，其中包括飞行控制器和配套计算机（此处称为“任务计算机”）。



飞行控制器运行正常的 PX4 飞行堆栈，而配套计算机则提供[物体回避](https://docs.px4.io/main/en/computer_vision/obstacle_avoidance.html)和[碰撞预防](https://docs.px4.io/main/en/computer_vision/collision_prevention.html)等高级功能。这两个系统使用快速串行或 IP 链路连接，通常使用[MAVLink 协议进行通信](https://mavlink.io/en/)。与地面站和云端的通信通常通过配套计算机进行路由（例如使用[MAVLink 路由器](https://github.com/mavlink-router/mavlink-router)（来自英特尔））。

PX4 系统通常在配套计算机上运行 Linux 操作系统（因为[PX4/PX4-Avoidance](https://github.com/PX4/PX4-Avoidance)项目提供了专为 Linux 设计的基于 ROS 的回避库）。Linux 是一个比 NuttX 更好的“通用”软件开发平台；还有更多的 Linux 开发人员，并且已经编写了许多有用的软件（例如计算机视觉、通信、云集成、硬件驱动程序）。出于同样的原因，配套计算机有时也会运行 Android。

【笔记：该图显示了通过 LTE 进行的云或地面站连接，这种方法已在许多基于 PX4 的系统中使用。PX4 不提供专门用于 LTE 和/或云集成的软件（这需要定制开发）。】

## PX4飞行堆栈架构

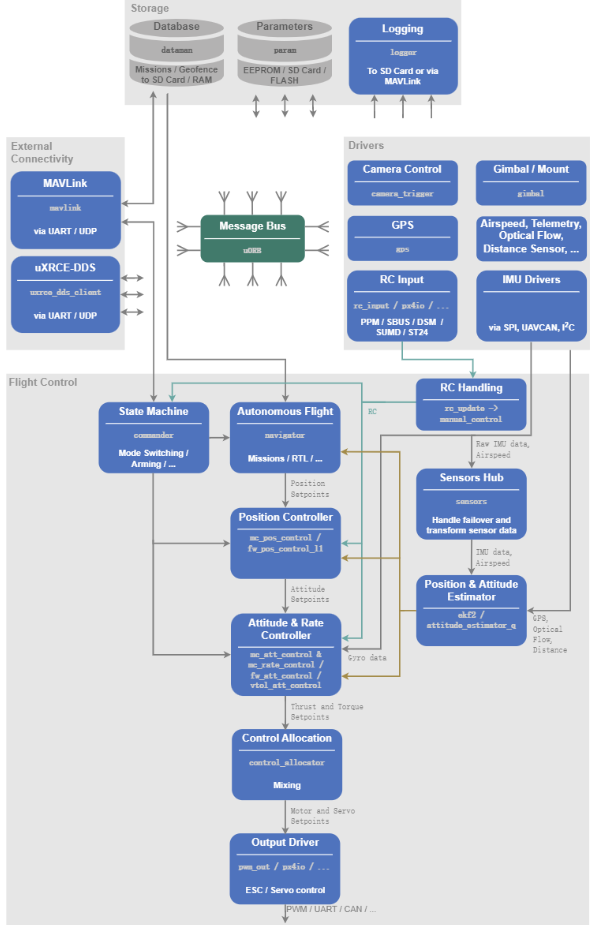
PX4由两个主要层组成：[飞行堆栈](https://docs.px4.io/main/en/concept/architecture.html#flight-stack)是一个估计器和飞行控制系统，[中间件](https://docs.px4.io/main/en/concept/architecture.html#middleware)是通用机器人层，可以支持任何类型的自主机器人，提供内部/外部通信和硬件集成。

所有 PX4 [airframes](https://docs.px4.io/main/en/airframes/)（机身）共享一个代码库（这包括其他机器人系统，如船只、漫游车、潜艇等）。整个系统设计是[反应式的](http://www.reactivemanifesto.org/)， 意思就是：

* 所有功能都分为可交换和可重用的组件
* 通信是通过异步消息传递完成的
* 系统可以处理不同的工作负载

1. 高级软件架构

下图详细概述了 PX4 的构建模块。该图的顶部包含中间件块，而下部显示飞行堆栈的组件。



源代码被分成独立的模块/程序（使用monospace字体，如上图所示）。通常，一个构建块恰好对应于一个模块。

【提示：在运行时，您可以使用topshell 中的命令检查哪些模块被执行，并且每个模块可以通过 单独启动/停止<module\_name> start/stop。虽然top命令特定于 NuttX shell，但其他命令也可以在 SITL shell (pxh>) 中使用。有关每个模块的更多信息，请参阅[模块和命令参考](https://docs.px4.io/main/en/modules/modules_main.html)。】

箭头显示模块之间最重要连接的信息流*。*实际上，连接比所示的要多得多，并且大多数模块都会访问一些数据（例如参数）。

[模块通过名为uORB](https://docs.px4.io/main/en/middleware/uorb.html)的发布-订阅消息总线相互通信。使用发布-订阅方案意味着：

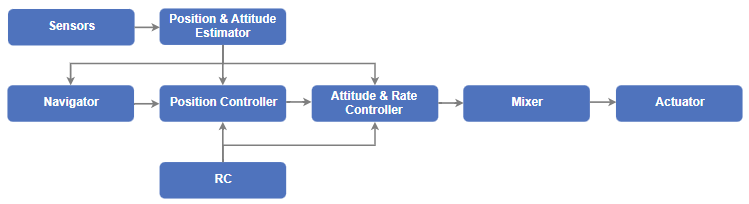
* 该系统是反应式的——它是异步的，当新数据可用时会立即更新
* 所有操作和通信都是完全并行的
* 系统组件可以以线程安全的方式使用来自任何地方的数据

【笔记：这种架构允许快速、轻松地替换这些块中的每一个，甚至在运行时也是如此。】

1. 飞行堆栈

飞行堆栈是自主无人机的制导、导航和控制算法的集合。它包括固定翼、多旋翼和垂直起降机身的控制器以及姿态和位置估计器。

下图显示了飞行堆栈构建块的概述。它包含从传感器、RC 输入和自主飞行控制（导航器）到电机或伺服控制（执行器）的完整管道。



估计器获取一个或多个传感器输入，将它们组合起来，并计算车辆状态（例如来自 IMU 传感器数据的姿态）。

控制器是将设定值和测量值或估计状态（过程变量）作为输入**的**组件。其目标是调整过程变量的值，使其与设定值匹配。输出是最终达到该设定点的修正。例如，位置控制器将位置设定点作为输入，过程变量是当前估计的位置，输出是将车辆移向期望位置的姿态和推力设定点。

混合器接受力命令（例如“右转”）并将其转换为单独的电机命令，同时确保不超过某些限制**。**这种转换特定于车辆类型并且取决于各种因素，例如相对于重心的电机布置或车辆的旋转惯量。

1. 中间件

中间件主要由嵌入式[传感器](https://docs.px4.io/main/en/middleware/)的设备驱动程序、与外部世界（配套计算机、GCS 等）的通信以及 uORB 发布-订阅消息总线组成。

此外，中间件还包括一个[模拟层](https://docs.px4.io/main/en/simulation/)，允许 PX4 飞行代码在桌面操作系统上运行，并在模拟“世界”中控制计算机建模的车辆。

1. 更新率

由于模块等待消息更新，因此驱动程序通常定义模块更新的速度。大多数 IMU 驱动程序以 1kHz 采样数据、集成并以 250Hz 发布。系统的其他部分（例如 ）navigator不需要如此高的更新率，因此运行速度要慢得多。

可以通过运行在系统上实时[检查](https://docs.px4.io/main/en/middleware/uorb.html)消息更新率uorb top。

1. 运行环境

PX4 可以在提供 POSIX-API 的各种操作系统（例如 Linux、macOS、NuttX 或 QuRT）上运行。它还应该具有某种形式的实时调度（例如 FIFO）。

模块间通信（使用[uORB](https://docs.px4.io/main/en/middleware/uorb.html)）基于共享内存。整个PX4中间件运行在单个地址空间中，即内存在所有模块之间共享。

【笔记：该系统的设计使得只需最少的努力就可以在单独的地址空间中运行每个模块（需要更改的部分包括uORB、parameter interface、dataman 和perf。）】

模块有两种不同的执行方式：

* **任务**：模块在自己的任务中运行，具有自己的堆栈和进程优先级。
* **工作队列任务**：模块在共享工作队列上运行，与队列上的其他模块共享相同的堆栈和工作队列线程优先级。
  + 所有任务必须协同工作，因为它们不能互相干扰。
  + 一个队列上可以运行多个*工作队列任务*，并且可以有多个队列。
  + 通过指定未来的固定时间或通过 uORB 主题更新回调来安排工作队列任务。

在工作队列上运行模块的优点是它使用更少的 RAM，并且可能导致更少的任务切换。缺点是工作队列任务不允许休眠或轮询消息，或执行阻塞 IO（例如从文件中读取）。长时间运行的任务（进行大量计算）也应该在单独的任务或至少单独的工作队列中运行。

【笔记：在工作队列上运行的任务不会显示在[top](https://docs.px4.io/main/en/modules/modules_command.html#top)（只能看到工作队列本身 - 例如作为wq:lp\_default）。用于[work\_queue status](https://docs.px4.io/main/en/modules/modules_system.html#work-queue)显示所有活动的工作队列项目。】

1. 后台任务

px4\_task\_spawn\_cmd()用于启动独立于调用（父）任务运行的新任务（NuttX）或线程（POSIX - Linux/macOS）：

|  |
| --- |
| independent\_task = px4\_task\_spawn\_cmd(  "commander", // Process name  SCHED\_DEFAULT, // Scheduling type (RR or FIFO)  SCHED\_PRIORITY\_DEFAULT + 40, // Scheduling priority  3600, // Stack size of the new task or thread  commander\_thread\_main, // Task (or thread) main function  (char \* const \*)&argv[0] // Void pointer to pass to the new task  // (here the commandline arguments).  ); |

1. 操作系统特定信息

**NuttX**

[NuttX](https://nuttx.apache.org/)是在飞行控制板上运行 PX4 的主要 RTOS。它是开源的（BSD许可证）、轻量级、高效且非常稳定。

模块作为任务执行：它们有自己的文件描述符列表，但它们共享单个地址空间。一项任务仍然可以启动一个或多个共享文件描述符列表的线程。

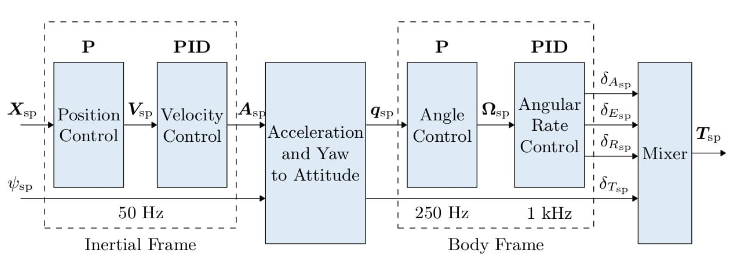
每个任务/线程都有一个固定大小的堆栈，并且有一个定期任务检查所有堆栈是否有足够的可用空间（基于堆栈着色）。

**Linux/macOS**

在 Linux 或 macOS 上，PX4 在单个进程中运行，模块在自己的线程中运行（与 NuttX 上一样，任务和线程之间没有区别）。

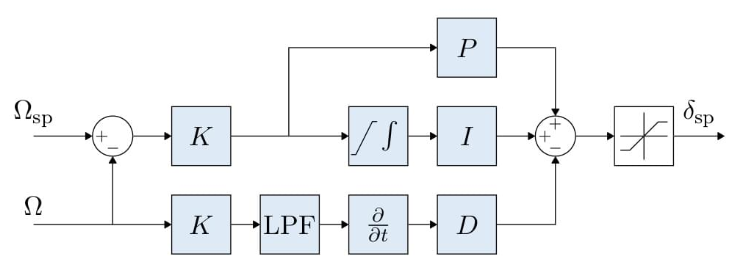
## 多旋翼控制器说明

**多旋翼控制架构**



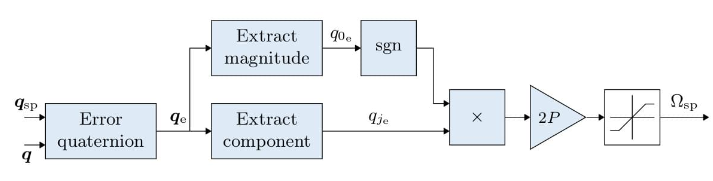
* 这是标准的级联控制架构。
* 控制器是 P 控制器和 PID 控制器的混合。
* 估计值来自[EKF2](https://docs.px4.io/main/en/advanced_config/tuning_the_ecl_ekf.html)。
* 根据模式的不同，外部（位置）环路会被旁路（在外部环路之后显示为多路复用器）。位置环仅在保持位置或轴中请求的速度为空时使用。

**多旋翼角速率控制器**



* K-PID控制器。有关详细信息，请参阅[速率控制器。](https://docs.px4.io/main/en/config_mc/pid_tuning_guide_multicopter.html#rate-controller)
* 积分权限受到限制，防止清盘。
* 输出受到限制（在控制分配模块中），通常为 -1 和 1。
* 低通滤波器 (LPF) 用于导数路径以减少噪声（陀螺仪驱动器向控制器提供滤波后的导数）。

多旋翼姿态控制器



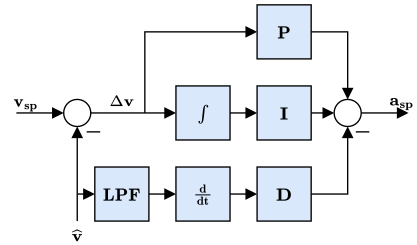
* 姿态控制器利用[四元数](https://en.wikipedia.org/wiki/Quaternion)。
* 控制器是从这篇[文章中实现的](https://www.research-collection.ethz.ch/bitstream/handle/20.500.11850/154099/eth-7387-01.pdf)。
* 调整该控制器时，唯一需要关注的参数是 P 增益。
* 速率命令已饱和。

**多旋翼加速度到推力和姿态设定点的转换**

* 速度控制器生成的加速度设定值将转换为推力和姿态设定值。
* 转换后的加速度设定值将饱和并优先考虑垂直和水平推力。
* 推力饱和是在计算相应的推力后完成的：
  1. 计算所需的垂直推力 ( thrust\_z)
  2. 饱和thrust\_z\_MPC\_THR\_MAX
  3. 饱和thrust\_xy\_(MPC\_THR\_MAX^2 - thrust\_z^2)^0.5

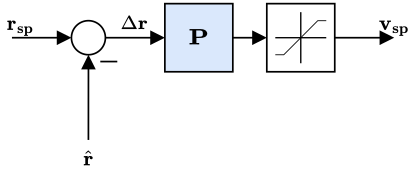
PositionControl.cpp实现细节可以在和中找到ControlMath.cpp。

**多旋翼速度控制器**



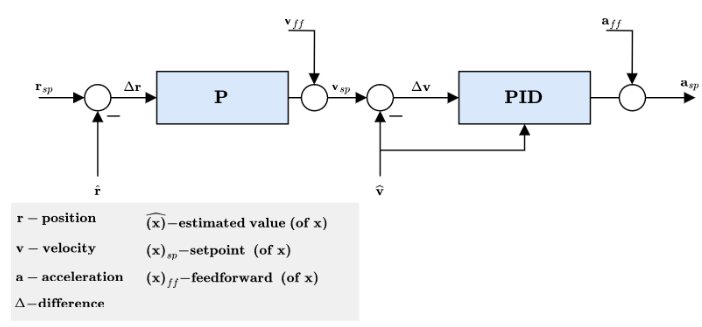
* PID控制器稳定速度。命令加速。
* 该积分器包括使用钳位方法的防复位饱和 (ARW)。
* 命令加速度未饱和 - 饱和度将与最大倾斜角结合应用于转换后的推力设定值。
* 水平增益通过参数MPC\_XY\_VEL\_P\_ACC、MPC\_XY\_VEL\_I\_ACC和设定MPC\_XY\_VEL\_D\_ACC。
* 垂直增益通过参数MPC\_Z\_VEL\_P\_ACC、MPC\_Z\_VEL\_I\_ACC和进行设置MPC\_Z\_VEL\_D\_ACC。

**多旋翼位置控制器**



* 命令速度的简单 P 控制器。
* 命令速度饱和以将速度保持在一定限度内。参见参数MPC\_XY\_VEL\_MAX。该参数设置最大可能的水平速度。这与最大**所需**速度MPC\_XY\_CRUISE（自主模式）和MPC\_VEL\_MANUAL（手动位置控制模式）不同。
* 水平 P 增益通过参数设置MPC\_XY\_P。
* 垂直 P 增益通过参数设置MPC\_Z\_P。

位置和速度组合控制器图



* 模式相关前馈 (ff) - 例如任务模式轨迹生成器（加加速度限制轨迹）计算位置、速度和加速度设定点。
* 加速度设定点（惯性系）将（带有偏航设定点）转换为姿态设定点（四元数）和集体推力设定点。

## 参数及配置

PX4 使用参数子系统（一个平面表float和int32\_t值）和文本文件（用于启动脚本）来存储其配置。

本节详细讨论param子系统*。*它涵盖了如何列出、保存和加载参数，以及如何定义它们并使它们可供地面站使用。

【提示：[系统启动和](https://docs.px4.io/main/en/concept/system_startup.html)[框架配置](https://docs.px4.io/main/en/dev_airframes/adding_a_new_frame.html)启动脚本的工作方式在其他页面上有详细介绍。】

**命令行使用**

PX4[系统控制台](https://docs.px4.io/main/en/debug/system_console.html)提供[param](https://docs.px4.io/main/en/modules/modules_command.html#param)工具，可用于设置参数、读取参数值、保存参数以及导出文件和从文件恢复。

1. 获取和设置参数

以下命令列出所有系统参数：

|  |
| --- |
| param show |

为了更具选择性，可以使用带有通配符“\*”的部分参数名称，示例如下：

|  |
| --- |
| nsh> param show RC\_MAP\_A\*  Symbols: x = used, + = saved, \* = unsaved  x RC\_MAP\_AUX1 [359,498] : 0  x RC\_MAP\_AUX2 [360,499] : 0  x RC\_MAP\_AUX3 [361,500] : 0  x RC\_MAP\_ACRO\_SW [375,514] : 0  723 parameters total, 532 used. |

您可以使用该-c标志来显示所有已更改的参数（相对于默认值）：

|  |
| --- |
| param show -c |

您可以用于param show-for-airframe显示当前机身定义文件（及其导入的默认值）的默认值已更改的所有参数。

1. 导出和加载参数

您可以保存已*更改的*任何参数（与机身默认值不同）。

标准param save命令会将参数存储在当前默认文件中：

|  |
| --- |
| param save |

如果提供了参数，它将把参数存储到这个新位置：

|  |
| --- |
| param save /fs/microsd/vtol\_param\_backup |

有两种不同的命令来加载参数：

* param load首先将所有参数完全重置为其默认值，然后用文件中存储的任何值覆盖参数值。
* param import只是用文件中的值覆盖参数值，然后保存结果（即有效地调用param save）。

有效地load将参数重置为保存参数时的状态（我们说“有效”是因为文件中保存的任何参数都将被更新，但其他参数可能具有与创建参数文件时不同的固件定义的默认值）。

相比之下，import将文件中的参数与车辆的当前状态合并。例如，这可用于仅导入包含校准数据的参数文件，而不覆盖系统配置的其余部分。

两种情况的示例如下所示：

|  |
| --- |
| # Reset the parameters to when file was saved  param load /fs/microsd/vtol\_param\_backup  # Optionally save params (not done automatically with load)  param save |

|  |
| --- |
| # Merge the saved parameters with current parameters  param import /fs/microsd/vtol\_param\_backup |

**创建/定义参数**

参数定义有两部分：

* [参数元数据](https://docs.px4.io/main/en/advanced/parameters_and_configurations.html#parameter-metadata)指定固件中每个参数的默认值以及用于地面控制站和文档中参数的呈现（和编辑）的其他元数据。
* [C/C++ 代码](https://docs.px4.io/main/en/advanced/parameters_and_configurations.html#c-c-api)，提供从 PX4 模块和驱动程序中获取和/或订阅参数值的访问权限。

下面描述了用于编写元数据和代码的几种方法。在可能的情况下，代码应使用较新的[YAML 元数据](https://docs.px4.io/main/en/advanced/parameters_and_configurations.html#yaml-metadata)和[C++ API](https://docs.px4.io/main/en/advanced/parameters_and_configurations.html#c-api)，而不是较旧的 C 参数/代码定义，因为它们更加灵活和健壮。

参数元数据被[编译到固件中，并通过](https://docs.px4.io/main/en/advanced/parameters_and_configurations.html#publishing-parameter-metadata-to-a-gcs)[MAVLink组件信息服务](https://mavlink.io/en/services/component_information.html)提供给地面站。

1. 参数名称

参数名称不得超过 16 个 ASCII 字符。

按照惯例，组中的每个参数应共享相同的（有意义的）字符串前缀，后跟下划线，并且MC\_和FW\_用于与多旋翼或固定翼系统专门相关的参数。该公约并未得到执行。

[代码和参数元数据](https://docs.px4.io/main/en/advanced/parameters_and_configurations.html#parameter-metadata)中的名称必须匹配，才能正确地将参数与其元数据（包括固件中的默认值）关联起来。

1. C/C++API

有单独的 C 和 C++ API 可用于从 PX4 模块和驱动程序中访问参数值。

API 之间的一个重要区别是 C++ 版本具有更有效的标准化机制来同步参数值的更改（即来自 GCS）。

同步很重要，因为参数可以随时更改为另一个值。您的代码应*始终*使用参数存储中的当前值。如果无法获取最新版本，则更改参数后将需要重新启动（使用@reboot\_required元数据设置此要求）。

此外，C++ 版本还具有更好的类型安全性和更少的 RAM 开销。缺点是参数名称必须在编译时已知，而 C API 可以将动态创建的名称作为字符串。