怕什么真理无穷，进一寸有一寸的欢喜

学习笔记

李冰

**目 录**

[**目 录** 2](#_Toc474697322)

[**无人机学习笔记** 5](#_Toc474697323)

[1 参与ardupilot项目编程--概述 5](#_Toc474697324)

[1.1](#_Toc474697325) 概述 [5](#_Toc474697325)

[1.2 须知 6](#_Toc474697326)

[2 参与ardupilot项目编程——git实践 6](#_Toc474697327)

[1.1](#_Toc474697328) 学习git [7](#_Toc474697328)

[2.1](#_Toc474697329) 复制主要资源库 [7](#_Toc474697329)

[2.2](#_Toc474697330) 克隆资源库 [9](#_Toc474697330)

[2.3](#_Toc474697331) 编译代码（程序） [10](#_Toc474697331)

[2.4](#_Toc474697332) 生成分支和更新一些代码 [10](#_Toc474697332)

[3 （3）参与ardupilot项目编程——git实践 13](#_Toc474697333)

[3.1](#_Toc474697334) 变基工作流：保持你的代码最新 [13](#_Toc474697334)

[3.2](#_Toc474697335) 研究子模块 [15](#_Toc474697335)

[3.3](#_Toc474697336) 小结 [15](#_Toc474697336)

[4 学习ArduPilot源码——引言和代码架构 15](#_Toc474697337)

[4.1](#_Toc474697338) 介绍 [15](#_Toc474697338)

[4.2](#_Toc474697339) 内容 [15](#_Toc474697339)

[4.2.1 引言 16](#_Toc474697340)

[4.2.2 基本架构 16](#_Toc474697341)

[4.2.3 编译系统 17](#_Toc474697342)

[5 （2）学习ArduPilot源码——shetch范例 18](#_Toc474697343)

[5.1](#_Toc474697344) 理解范例代码 [22](#_Toc474697344)

[5.1.1 hal引用 22](#_Toc474697345)

[5.1.2 setup（）和loop（）函数 23](#_Toc474697346)

[5.1.3 AP\_HAL\_MAIN()宏 23](#_Toc474697347)

[5.1.4 粗糙的示例代码 23](#_Toc474697348)

[6 （3）学习ArduPilot源码——理解Ardupilot线程 24](#_Toc474697349)

[6.1](#_Toc474697350) 计时器回调 [24](#_Toc474697350)

[**PX4开发者指南** 26](#_Toc474697351)

[1 引言 26](#_Toc474697352)

[1.1](#_Toc474697353) 更新日志 [26](#_Toc474697353)

[1.2](#_Toc474697354) 许可证Licence [26](#_Toc474697354)

[2 入门 26](#_Toc474697355)

[2.1](#_Toc474697356) 初始配置 [26](#_Toc474697356)

[2.2](#_Toc474697357) 工具链安装 [26](#_Toc474697357)

[2.3 开发环境 26](#_Toc474697358)

[1.2.1 Linux开发环境搭建（windows等系统的就不翻译了，自己看官网） 27](#_Toc474697359)

[2.3.1 权限设置 27](#_Toc474697360)

[2.3.2 安装 27](#_Toc474697361)

[2.3.3 NuttX的基本硬件 27](#_Toc474697362)

[2.4](#_Toc474697363) 编译代码 [28](#_Toc474697363)

[2.4.1 在控制台编译 28](#_Toc474697364)

[2.4.2 基于NuttX / Pixhawk的硬件板 28](#_Toc474697365)

[2.5](#_Toc474697366) 投稿&开发 [29](#_Toc474697366)

[2.5.1 测试飞行结果 30](#_Toc474697367)

[2.5.2 论坛和聊天 30](#_Toc474697368)

[2.6](#_Toc474697369) 总结 [30](#_Toc474697369)

[3 相关概念 30](#_Toc474697370)

[3.1](#_Toc474697371) 飞行模式/操控 [30](#_Toc474697371)

[飞行模式概述 31](#_Toc474697372)

[1手动模式（*MANUAL*](#_Toc474697373) ） [31](#_Toc474697373)

[2辅助模式 31](#_Toc474697374)

[3.自动 31](#_Toc474697375)

[3.2](#_Toc474697376) 架构概述 [33](#_Toc474697376)

[上层软件结构 33](#_Toc474697377)

[与地面站的通信架构 34](#_Toc474697378)

[3.3 PX4](#_Toc474697379)飞行栈 [35](#_Toc474697379)

[3.4 PX4](#_Toc474697380)中间件 [35](#_Toc474697380)

[3.5](#_Toc474697381) 混合驱动器 [35](#_Toc474697381)

[关键概念 36](#_Toc474697382)

[映射 36](#_Toc474697383)

[PX4混合器的定义 36](#_Toc474697384)

[空混频器 36](#_Toc474697385)

[4 理论学习部分 38](#_Toc474697386)

[4.1](#_Toc474697387) 坐标系 [38](#_Toc474697387)

[4.1.1 右手定则 38](#_Toc474697388)

[4.1.2](#_Toc474697389) 惯性坐标系与机体坐标系定义 [38](#_Toc474697389)

[4.2](#_Toc474697390) 姿态表示 [39](#_Toc474697390)

[4.2.1](#_Toc474697391) 欧拉角的定义 [39](#_Toc474697391)

[4.2.2 欧拉角变化率与机体角速度的关系 40](#_Toc474697392)

[4.2.3 旋转矩阵 41](#_Toc474697393)

[4.3](#_Toc474697394) 多旋翼的控制模型 [42](#_Toc474697394)

[4.3.1 刚体运动学模型 42](#_Toc474697395)

[4.3.2 多旋翼的动力学模型 43](#_Toc474697396)

[4.4](#_Toc474697397) 控制分配模型 [44](#_Toc474697397)

[4.4.1 动力系统模型 46](#_Toc474697398)

[4.4.2 多旋翼的气动阻力模型 46](#_Toc474697399)

[4.4.3 多旋翼模型参数测量 48](#_Toc474697400)

[《人类简史》 52](#_Toc474697401)

[《机器人学导论》--John J. Craig 53](#_Toc474697402)

[1 空间描述和变换 53](#_Toc474697403)

[1.1](#_Toc474697404) 位置描述 [53](#_Toc474697404)

[1.2](#_Toc474697405) 姿态描述 [53](#_Toc474697405)

[1.3](#_Toc474697406) 坐标系的描述 [54](#_Toc474697406)

[1.4](#_Toc474697407) 坐标系的一般映射：齐次矩阵变换 [54](#_Toc474697407)

[1.5](#_Toc474697408) 变换算法 [55](#_Toc474697408)

[1.5.1 混合变换 55](#_Toc474697409)

[1.5.2 逆变换 55](#_Toc474697410)

[1.6](#_Toc474697411) 变换方程 [56](#_Toc474697411)

[1.7](#_Toc474697412) 姿态的其他描述方法 [56](#_Toc474697412)

[1.7.1 X-Y-Z固定角坐标系 56](#_Toc474697413)

[1.7.2 X-Y-Z欧拉角 57](#_Toc474697414)

[1.7.3 Z-Y-Z欧拉角 58](#_Toc474697415)

[1.7.4 等效轴坐标表示法 58](#_Toc474697416)

[1.7.5 欧拉参数 59](#_Toc474697417)

[2 操作臂的运行学 59](#_Toc474697418)

[2.1](#_Toc474697419) 连杆的描述 [59](#_Toc474697419)

[2.2](#_Toc474697420) 操作臂运动的运行学 [61](#_Toc474697420)

[2.2.1 连杆变换的推导 61](#_Toc474697421)

**无人机学习笔记**

1. 参与ardupilot项目编程--概述

本文介绍如何获取ArduPilot代码和如何把改变的代码提交给项目。

* 1. 概述

ArduPilot项目使用[git](http://git-scm.com/)管理源代码和使用[GitHub](https://github.com/)托管源代码。

开发者想为ArduPilot项目贡献自己力量时会创建分叉，在分叉上创建一个带有新特性的分支，然后提交一个拉的请求来把开发者所做的改变合并到“主”项目。开发者只是使用代码或者创建最新的代码时，可以通过克隆和创建“主”代码库实现。

ArduPilot项目对飞机、直升机、小车和天线跟踪器的源代码可以在GitHub的 <https://github.com/ArduPilot/ardupilot> 代码库上取得。另外几个项目是用于px4的基本平台（比如PX4v1和Pixhawk）： [PX4Firmware](https://github.com/ArduPilot/PX4Firmware), [PX4NuttX](https://github.com/ArduPilot/PX4NuttX), [uavcan](https://github.com/ArduPilot/uavcan)——当你建立项目的时候这些会被导入成为 [Git Submodules](http://www.ardupilot.org/dev/docs/git-submodules.html#git-submodules) 。

MissionPlanner是在 [diydrones/MissionPlanner](https://github.com/ArduPilot/MissionPlanner)的库。

注意

一个旧的谷歌代码库由于遗留原因仍然在线，但是除非你特别需要旧的（APM1.x）资源，否则你是不需要它的。

* 1. 须知

Ardupilot项目使用 [git](http://git-scm.com/) 管理源代码。

Git可以在主流的操作系统（OS）上使用，也有各种各样的工具使得很容易地掌握它。首先，你需要在你的操作系统[下载和安装客户端](http://git-scm.com/downloads)。如果你是源码控制系统的新手， [GitHub for Windows](https://windows.github.com/) 或者 [GitHub for Mac](https://mac.github.com/)的客户端 会有很好的说明文档和与GitHub的集成性，是一个开始的好方法。这个指南会同时在windows用户界面和OSX/Linux终端命令行使用GitHub。

如果你正在准备提交代码到官方APM源码库，你需要[注册一个免费的](https://github.com/join)Github账户。

1. 参与ardupilot项目编程——git实践

我的tip：（安装git客户端在windows很简单不必多说，在linux上安装也很简单。

Debian/Ubuntu

$ apt-get install git

Fedora

$ yum install git (up to Fedora 21)  
$ dnf install git (Fedora 22 and later)

Gentoo

$ emerge --ask --verbose dev-vcs/git

Arch Linux

$ pacman -S git

openSUSE

$ zypper install git

FreeBSD

$ cd /usr/ports/devel/git  
$ make install

Solaris 11 Express

$ pkg install developer/versioning/git

OpenBSD

$ pkg\_add git

* 1. 学习git

这个指南涉及进行项目研究：克隆、分支、提交、推送的基本的git 命令/概念。

如果你需要了解更多关于git的信息，这里有很多的在线资源。下面是一些可能对你有用的：

* [Try Git](http://try.github.io/levels/1/challenges/1): 基于浏览器的学习Git的互动教程
* [Git Ready](http://gitready.com/): 不同难度级别的教程
* [Git SCM Book](http://git-scm.com/book/en/Getting-Started): 介绍和完整文档
  1. 复制主要资源库

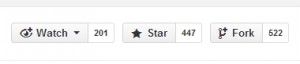
注意：如果你想创建和测试源码（不进行修改）你可以跳过这个步骤，只需克隆主要的项目资源库（下一部分）。

“Forking”是GitHub复制一个资源库到自己账户的术语。复制的资源库保存了原始项目的信息，这样你就可以通过它获取更新（并且可以反过来给它提交更新）。如果你想提交更新到主项目你首先需要创建你自己复制的主要ArduPilot资源库。

关于forking的更深入工作原理请看下面这个中文博客的讲解https://github.com/oldratlee/translations/blob/master/git-workflows-and-tutorials/workflow-forking.md

复制主要的资源库

* 登录Github，去到网站 <https://github.com/ArduPilot/ardupilot>.
* 在右上角有一个按钮叫, “Fork”:



点击**Fork**按钮，然后跟着向导操作。

当你完成复制完成后，会在你的账户下面有一个新目录，如下图所示

//github.com/your-github-account-name/ardupilot

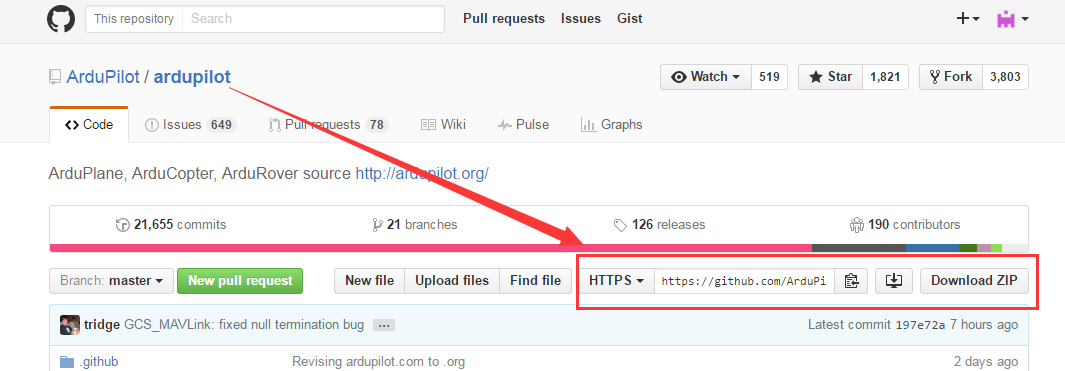


当你更新代码时，这个复制的资源库就是你将要克隆和在本地进行工作用的资源库。

* 1. 克隆资源库

“克隆”是git复制任何资源库到你的电脑的简称。你可以克隆你自己复制好的资源库（如果你需要更改源代码）或者主要的ArduPilot资源库。

你克隆的项目所需要的的信息在每个Github资源库首页的屏幕的右边。（可能随着网站的更新，图标位置会有所变化）



在Github对话框克隆资源库

**OSX/linux终端：**

* 打开终端，并去到你想存放资源库的目录位置
* 克隆你的资源库：

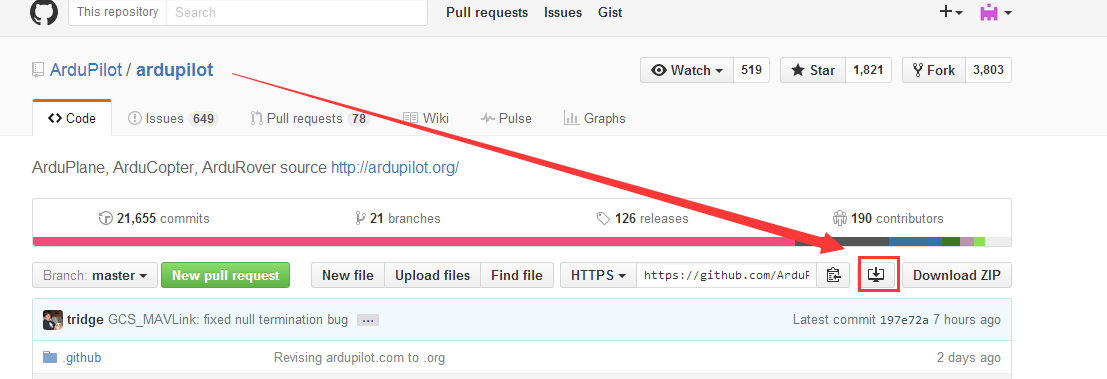
git clone https:**//**github**.**com**/**your**-**github**-**account**-**name**/**ardupilot

或者主项目

git clone https:**//**github**.**com**/**ArduPilot**/**ardupilot

**Windows (GitHub GUI):**

* **浏览器打开** [diydrones/ardupilot](https://github.com/ArduPilot/ardupilot) 资源库所在页面
* 单击“Clone in Desktop”按钮（随着网站更新，按钮可能会有变化）



* 1. 编译代码（程序）

Ardupilot支持在Linux、Windows和Mac OSX编译很多不同的目标代码（车辆和autopilot硬件）。关于怎样编译特定的目标代码，请看 [Building the code](http://ardupilot.org/dev/docs/building-the-code.html#building-the-code)。（中文博客更新中。。。。。。）

* 1. 生成分支和更新一些代码

分支是是不同路径的开发相互独立，也可以合并为一个简单（经常命名为“mashter”）的分支。参考这个[简短的指南](http://gitready.com/beginner/2009/01/25/branching-and-merging.html)以获得跟多信息，它在[Learning Git](http://ardupilot.org/dev/docs/where-to-get-the-code.html#where-to-get-the-code-learning-git)部分下的某个资源。在那部分教程，你可以创建一个分支和改变一些代码。

分支的名字由你自己来决定，但是使用描述性的短语会有帮助，用于这个教程中的分支名字用 “apm\_git\_tutorial”

**OSX/Linux终端命令**

**这些命令假设你目前的工作目录在你克隆的资源库的根目录下。**

**注意：这些命令也可以在windows如果你使用git客户端（比如**“Git Shell”实用程序和GitHub在windows进行了安装**）。**

**1.创建一个分支**

git checkout **-**b apm\_git\_tutorial

2.改变一些代码。在本教程，通过编辑器打开**Tools/GIT\_Test/GIT\_Success.txt，在文件末尾加入你的名字，保存文件。**

**3.通过检查状态查看你对一些文件的改动：**

git status

4.提交你的工作到分支，加入你在git history所做改变：

git add Tools**/**GIT\_Test**/**GIT\_Success**.**txt

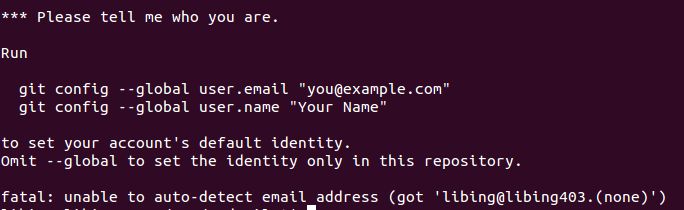
git commit **-**m 'Added name to GIT\_Success.txt'

我的tip：#上面命令中官网中命令为git add Tools**/**GIT\_Test**/**Git\_Success**.**txt

文件名Git\_Success**.**txt中git应该大写，下载的资源库中Git\_Success**.**txt文件名可能与官网的命令不一样

第一次执行git commit **-**m 'Added name to GIT\_Success.txt'

可能会报下面的错误，按照提示进行配置即可



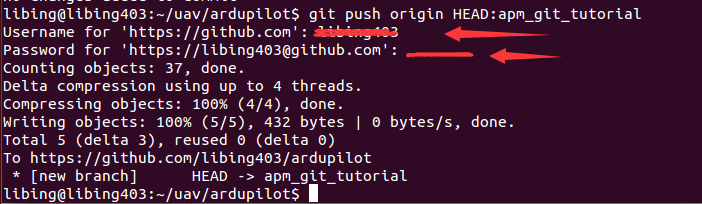
提交成功可能显示如下面所示



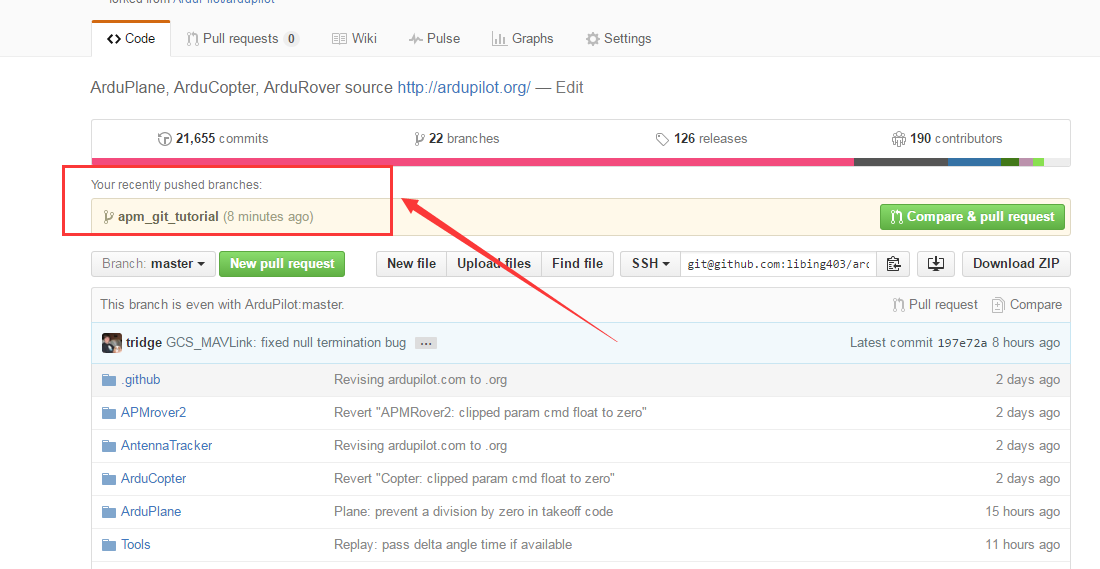
5.推送你的分支到GitHub。这个操作会复制你本地的分支到一个GitHub的一个新分支。推送分支是在GitHub和其他人合作的先决条件，或者为了提交补丁给官方版本。假设你的源是你复制的github资源库的远程端名字（）。

git push origin HEAD:apm\_git\_tutorial

运行命令后会要求你输入github网站账号的用户名和密码。成功推送结果如下，



再次登录或刷新gihub就会发现在网页中多了你刚刚推送的分支以及所做的改变，如下图所示



1. （3）参与ardupilot项目编程——git实践

git在windows中的用法我就不进行翻译了，因为我用linux平台。对于用windows的伙伴可以百度一下，或者到[官网看资料](http://ardupilot.org/dev/docs/where-to-get-the-code.html)

* 1. 变基工作流：保持你的代码最新

当你进行开发的时候，（原始的）ArduPilot资源库主分支很可能会更新，你应该[保持你复制的本地分支是目前最新的](http://robots.thoughtbot.com/keeping-a-github-fork-updated)。变基允许最新的原始资源库再次应用你的修改，这使得项目更容易合并。

我的tip：这里把Rebasing 翻译为变基，我的理解是改变原来的基本，基础。也就是说拟开发所依赖的原始资源库（基本的，基础的）发生了改变、更新，而变基是更新依赖的原始库，使你开发所依赖的库是最新的。

下面的命令可以用来对你的复制资源库变基。你可以在linux/OSX终端输入这些命令，或者windows上安装的“Git Shell”程序使用这些命令。

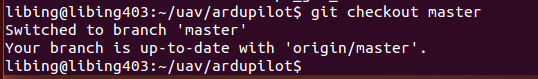
1.去到ardupilot 目录

cd **<**ardupilot**-**path**>**

2.确保你在主分支

git checkout master

如下图转到主分支了



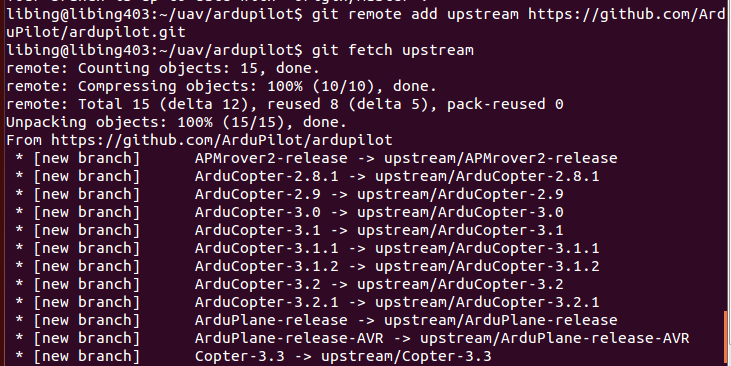
3.保证你保证你复制的资源库与顶层资源库（原始的）连接上

git remote add upstream https:**//**github**.**com**/**ArduPilot**/**ardupilot**.**git

4.从顶层资源库获取更新

git fetch upstream

执行这个命令你需要等待一段时间，更新可能要花好几分钟，具体看电脑和网络环境，更新结果可能会如下图那样



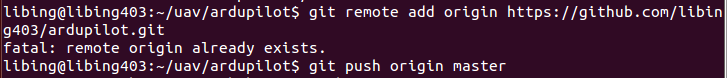
5.从当前顶层资源库对你的当前分支变基

git rebase upstream**/**master

6.保证你的资源库与你在github的资源库保持连接。

git remote add origin https:**//**github**.**com**/**your**-**github**-**account**-**name**/**ardupilot**.**git

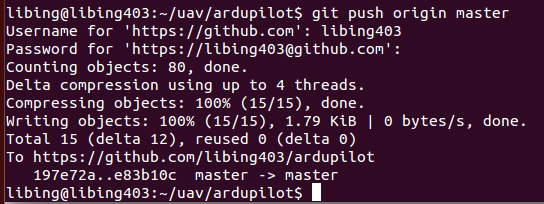
结果如下图，origin库确认存在



7.推送新的顶层库（master）到你的github资源库

git push origin master

提示输出gihub账户的用户名和密码，更新后结果如下图所示



* 1. 研究子模块

Ardupilot 对PX4的开发平台是基于的使用下面三个资源库的平台：

* [PX4Firmware](https://github.com/ArduPilot/PX4Firmware)
* [PX4NuttX](https://github.com/ArduPilot/PX4NuttX)
* [uavcan](https://github.com/ArduPilot/uavcan)

这些Ardupilot项目的Git 子模块，但需要时会被自动获取成为编译的一部分。要了解研究这些项目的更多信息请参考[Git](http://ardupilot.org/dev/docs/git-submodules.html#git-submodules) 子模块。（后续将会在Ardupilot源码学习的博客中更新）

* 1. 小结

本文重点是介绍了变基的实现过程。个人理解，变基就是把工作的基础资源库进行更新。因为我们复制了项目的主资源库到自己的账户下，并且克隆到了本地电脑，自己账户下的资源库和本地资源库除了自己进行的修改和更新外，没有其他变化。而项目的主资源库由世界各地的开发者不断开发，还不断地被更新和修改。因此为了我们自己的开发跟上项目的进展，需要对我们复制的和克隆的原始主资源库进行更新，也就是我们会在新的基础资源库上继续开发。我把rebaseing翻译为“变基”不知道是否合适？

1. 学习ArduPilot源码——引言和代码架构
   1. 介绍

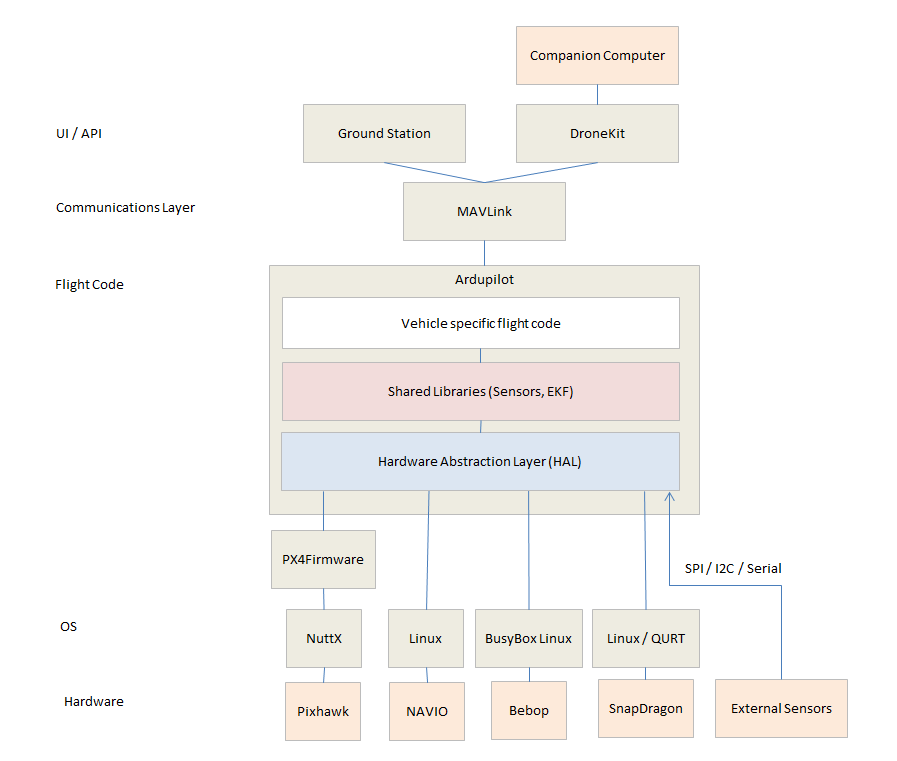
Ardupilot项目的代码库是相当大的（在ardupilot git tree大约有70万行），对于初学者来说是很吓人的。这里打算给一些建议，加快理解代码的速度。这里假设你对c++的关键概念熟悉，很多例子都假设你可以使用linux系统研究代码。

* 1. 内容
* 引言和代码架构
* 范例sketch
* Ardupilot进程
* UARTs和控制台
* RC输入和RC输出
* 存储和EEPROM管理
* 飞行器代码
  + 1. 引言

本节介绍Ardupilot的基本架构。在开始之前你需要确定你使用的系统。你可以使用浏览器看代码或者查看<https://github.com/ArduPilot/ardupilot/>，但是如果使用一个好的编程IDE你很可能会得到更多信息，可以让你找到函数、结构和类的定义，通过结构的方法展示代码。

一些建议如下：

* Eclipse on Windows, Linux or MacOS
* Emacs on Linux, Windows or MacOS, with etags for finding code elements
* Vim on emacs with ctags
  + 1. 基本架构



Ardupilot的基本架构可以分为5个主要部分：

* 飞行器目录 vehicle directories
* 飞控硬件抽象层 AP\_HAL
* 运行库 libraries
* 工具目录 tool directories
* 外部支持代码 external support code

下面详细会详细介绍，但是在移动前要确认你[克隆了你所需的所有的](http://www.ardupilot.org/dev/docs/where-to-get-the-code.html#where-to-get-the-code)git库

**飞行器目录 vehicle directories**

飞行器目录是在上层水平定义了固件对应的飞行器类型，目前有四种——飞机、直升机、APM车和天线跟踪器。

和\*.cpp文件一起，每个飞行器目录包含一个make.inc 文件列出了依赖库。Makefiles读取这个文件为生成程序来创建-l和-L标志。

**AP\_HAL**

AP\_HAL（Hardware Abstraction Layer）使得Ardupilot可以移植到大量不同的平台。运行库的顶层AP\_HAL定义了接口，代码需要指定板载的特征，然后是一个关于每种板载的AP\_HAL\_XXX 子目录，比如AP\_HAL\_AVR用于基于AVR的控制板，AP\_HAL\_PX4用于PX4控制板， AP\_HAL\_Linux用于基于linux的控制板。

**工具目录Tools directories**

工具目录是杂项支持目录。比如tools/autotes指 [autotest.ardupilot.org](http://autotest.ardupilot.org/)网站autotest基础设施，tools/Replay提供我们日志重放使用程序。

**外部支持代码 external support code**

在一些平台我们需要外部代码提供附加的特征或者控制板支持，目前外部代码有：

* [PX4NuttX](https://github.com/ArduPilot/PX4NuttX) - the core NuttX RTOS used on PX4 boards
* [PX4Firmware](https://github.com/ArduPilot/PX4Firmware) - the base PX4 middleware and drivers used on PX4 boards
* [uavcan](https://github.com/ArduPilot/uavcan) - the uavcan CANBUS implementation used in ArduPilot
* [mavlink](https://github.com/mavlink/mavlink) - the mavlink protocol and code generator

注意：当你为PX4/Pixhawk创建ArduPilot这些代码大多数可以导入为 [Git Submodules](http://www.ardupilot.org/dev/docs/git-submodules.html#git-submodules)。

* + 1. 编译系统

编译系统是基于linux的make编译工具，但是也支持旧的arduino IDE 对AVR编译。makefiles文件在 [mk/ directory](https://github.com/ArduPilot/ardupilot/tree/master/mk)（mk目录下），定义了支持的每一种类型控制板的编译规则。

为特定的控制板目标编译一个飞行器系统或“sketch”，需要输入“make TARGET”，TARGET就是控制板的类型，目前可以使用的控制板类型如下：

* make apm1 - the APM1 board
* make apm2 - the APM2 board
* make px4-v1 - the PX4v1
* make px4-v2 - the Pixhawk (and [Arsov AUAV-X2](http://www.auav.co/product-p/auavx2.htm))
* make pxf - the BBB+PXF cape combination
* make navio - the RaspberryPi+NavIO cape combination
* make linux - a generic Linux build
* make flymaple - the FlyMaple board
* make vrbain - the VRBrain boards
* make sitl - the SITL software in the loop simulation

更多的接口在不断被添加，因此查看“make help”文件了解新的目标板。

对于这些编译，你可以添加额外的选项，你可以做并行编译提高速度。比如在copter目录下你可以执行：

make apm2**-**octa **-**j8

意思是使用8通道并行编译 APM2硬件板上的OctaCopter程序，你也可以研究可供使用[ccache](http://ccache.samba.org/)进行快速编译。

一些硬件板也支持从固件目录上传，比如：

make px4**-**v2**-**upload

可以编译并上传一个sketch到pixhawk硬件板上。

特定的硬件板会有辅助编译，比如：

* make clean - clean the build for non-px4 targets
* make px4-clean - completely clean the build for PX4 targets
* make px4-cleandep - cleanup just dependencies for PX4 targets

我的tip:对于上述的命令我也不了解，官网There are also helper make targets for specific boards的说法还没搞清楚是什么意思，我运行命令的时候通常会有下面的提示



1. （2）学习ArduPilot源码——shetch范例

我的tip：在了解了ArduPilot源码的基本信息和架构后，下面开始实战，下面开始学习第一个范例。

你自己探索这些代码的第一步是使用libraries的例程。根据arduino的传统，我们的大部分libraies都会有范例。范例（“sketch”）是写成cpp文件的主程序。

认识Ardupilot使用的API库和协议是理解代码的前提，因此使用范例是入门的好方法。刚开始你应该阅读、编译和运行下面的库的范例：

* libraries/AP\_GPS/examples/GPS\_AUTO\_test
* libraries/AP\_InertialSensor/examples/INS\_generic
* libraries/AP\_Compass/examples/AP\_Compass\_test
* libraries/AP\_Baro/examples/BARO\_generic
* libraries/AP\_AHRS/examples/AHRS\_Test

例如， 下面命令可以编译和安装AP\_GPS范例到Pixhawk上：

cd libraries**/**AP\_GPS**/**examples**/**GPS\_AUTO\_test

make px4**-**clean

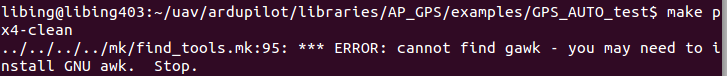
make px4**-**v2

make px4**-**v2**-**upload

我的tip（自己实践的经历）：首次运行第二个命令时

make px4**-**clean

可能会出现下面的错误（我在ubuntu14.04上运行出现了下面的错误），提示找不到gawk。



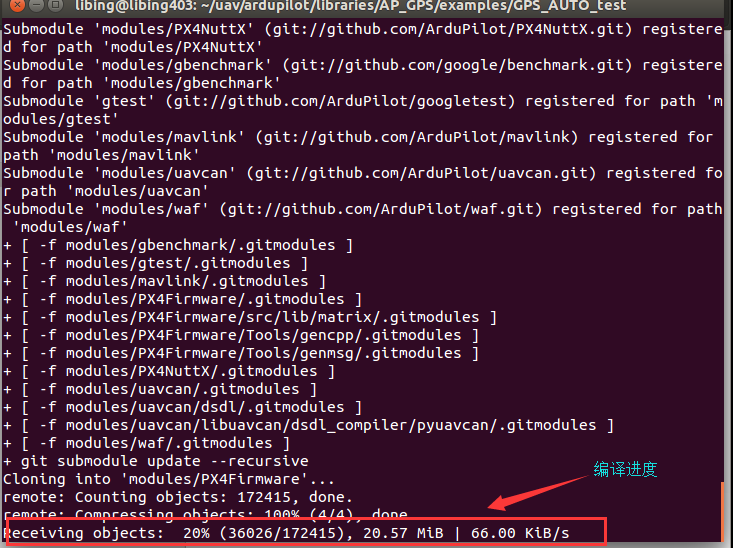
这是因为缺少一个虚拟包，在联网的情况下执行下面的安装命令

sudo apt-get install gawk

安装好gawk后，继续执行

make px4**-**clean

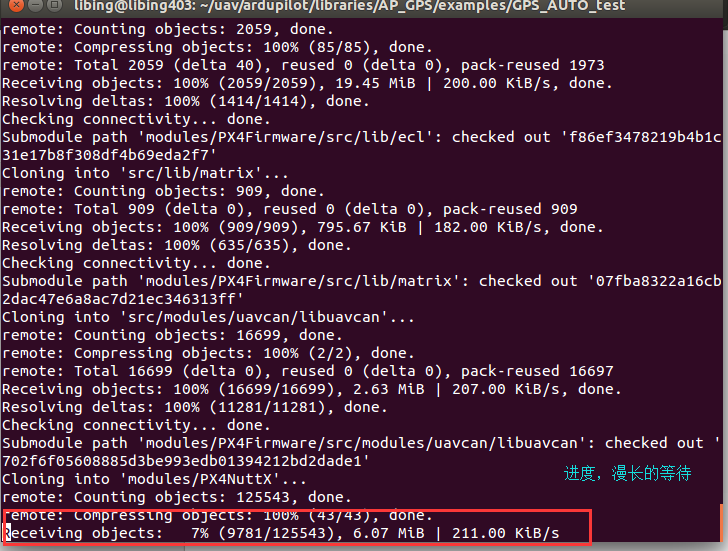
就可以正常编译了，如下图所示，会显示编译进度，编译可能会花挺长一段时间



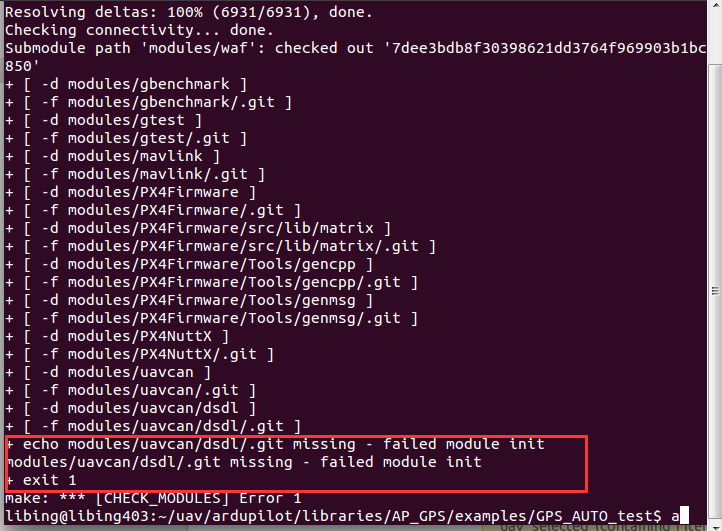
编译到后面可能会提示一些错误和失败，可以先不用管，继续执行下一步命令

make px4**-**v2

接下来又是漫长的等待，真的要很久。



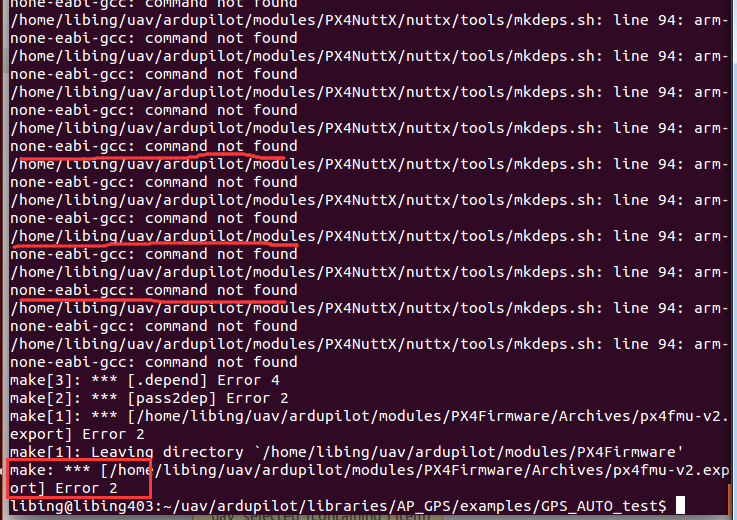
经过漫长的等待，最后发现有一些错误提示，这到底怎么解决？这些错误提示会影响到后面的操作吗？到目前的学习为止或许我们还没能力去解决，我们先带着这样的问题继续学习，或许后面积累的东西多了就会柳暗花明又一村。



电脑连接好飞控pixhawk，继续执行命令：

make px4**-**v2**-**upload

最后发现又是一堆错误，很正常，编译总会遇到这样那样的错误，根据提示试着解决，要是解决不了，后面继续学习想办法解决。



一旦你上传范例你就可以通过连接控制台查看输出。使用什么样的控制台由硬件板决定。在PX4板（比如PX4v1和pixhawk）就是USB连接器。

比如，如果你安装了mavproxy代理，你可以这样操作，在linux上连接pixhawk：

mavproxy**.**py **--**setup **--**master **/**dev**/**serial**/**by**-**id**/**usb**-**3D\_Robotics\_PX4\_FMU\_v2**.**x\_0**-**if00

使用--setup选项使mavproxy进入原串行模式，而不是执行MAVlink模式。这就是范例中你需要做的。

* 1. 理解范例代码

当你阅读范例代码（比如GPS\_AUTO\_test代码）就会留意到一些奇怪的事情：

* 它声明了一个‘hal’变量作为引用
* 这些代码很粗糙，没有好评
* setup（）函数和loop（）函数
  + 1. hal引用

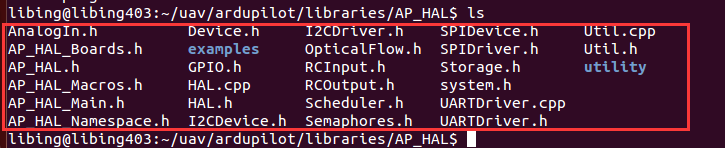
每个使用AP\_HAL特征的文件需要声明一个hal引用。这给了进入AP\_HAL的入口，AP\_HAL提供了所有硬件特定函数的入口，包括打印消息到控制台，休眠和使用I2C和SPI总线。

真实的hal变量是隐藏在特定的AP\_HAL\_XXX 库里，在每个文件的引用只是提供了获取hal的便利方法。

最常用的hal函数是：

* hal.console->printf() and hal.console->printf\_P() to print strings (use the \_P to use less memory on AVR)
* hal.scheduler->millis() and hal.scheduler->micros() to get the time since boot
* hal.scheduler->delay() and hal.scheduler->delay\_microseconds() to sleep for a short time
* hal.gpio->pinMode(), hal.gpio->read() and hal.gpio->write() for accessing GPIO pins
* I2C access via hal.i2c
* SPI access via hal.spi

继续去看一下libraries/AP\_HAL目录，了解在HAL上可用的所有函数列表。如下图所示



* + 1. setup（）和loop（）函数

你或许会发现每个范例都会有setup（）函数和loop（）函数。在硬件板启动时就会调用setup函数，真实的调用来自每个板的HAL，因此main（）函数是隐藏在HAL里的，硬件板完全启动后，main（）函数就会调用setup（）。

setup（）函数只会调用一次，它初始化libraries，在启动时还可能答应一个“hello”来显示。

在setup（）运行完成后，loop（）函数接着被调用（被AP\_HAL的主代码调用）范例的主要工作通常在loop（）函数里进行。

要注意，在复杂的硬件板上，setup（）/loop（）函数只是冰山一角。这会使得Ardupilot看起来好像是单线程的，但是实际上有很多在底层运行的线程，硬件板（如PX4和基于linux的板）实际上有大量实时线程正在启动，具体请看后面的内容——理解ArduPilot线程。

* + 1. AP\_HAL\_MAIN()宏

你会注意到在每个范例的 后面会有额外的一行代码：

AP\_HAL\_MAIN();

这个就是HAL宏，它会根据硬件板为初始化HAL的代码，生成一些必要的代码来声明C++main（）函数，你几乎不用担心它是怎样工作的，如果你对此感到好奇，可以查看AP\_HAL\_XXX目录里每个HAL的关于#define的内容，它通常在AP\_HAL\_XXX\_Main.h.。

* + 1. 粗糙的示例代码

你会注意到范例的代码是相当粗糙的，且缺少注释。这也是你为代码贡献力量的机会！但你阅读过这些示例代码和知道它们怎样工作后，就可以添加注释在代码里，解释APIs然后[提交一个推送](http://ardupilot.org/dev/docs/submitting-patches-back-to-master.html#submitting-patches-back-to-master)，这样别人也会从你的学习中受益。

1. （3）学习ArduPilot源码——理解Ardupilot线程

当你学习了Ardupilot代码库的基本知识后，就该去理解Ardupilot是怎样处理线程的。setup（）/loop（）结构从Arduino继承过来使得Ardupilot看起来是单线程系统，但实际上它并不是。

Ardupilot线程的方法由硬件板决定，它在一些板上编译（如APM1和APM2）是不支持线程的，只是编译成简单的计时器和回调。一些硬件板（如px4和linux）支持大量的Posix进程模型，带有实时优先级。这些都被Ardupilot广泛使用。

涉及线程时Ardupilot有大量关键概念你需要理解：

* 定时回调
* HAL专属线程
* 驱动专属线程
* APM驱动与板级驱动
* 板级专属线程、任务
* AP\_Scheduler任务系统
* 信号
* 解锁数据结构
  1. 计时器回调

每个平台都由AP\_HAL提供一个1KHZ的计时器，Ardupilot里的任何代码都可以注册一个定时器函数，然后以1KHZ调用。所有注册过的计时器会按顺序调用。使用这种非常原始的机制是因为它高度可移植，而且非常有用。你可以通过调用hal.scheduler->register\_timer\_process()来注册定时器回调。

hal**.**scheduler**->**register\_timer\_process(AP\_HAL\_MEMBERPROC(**&**AP\_Baro\_MS5611::\_update));

这个具体的例子来自于MS5611气压计的驱动。AP\_HAL\_MEMBERPROC()宏提供了一种方法封装C++成员函作为一个回调参数（用函数指针将上下文捆绑）。

当一段代码需要一些事情以小于1KHZ执行时，如果没有经过足够的时间，它应该保持他自己的“last\_called”可用，并且立即返回。当启动毫秒和微秒，你可以使用hal.scheduler->millis() 和hal.scheduler->micros() 函数获得时间来实现。

现在你可以去修改一个存在范例（或者创建一个新的），然后添加一个计时器。是计时器增加计数，每秒钟在loop（）函数里打印计数的数值。修改你的函数是它每隔25毫秒增加。

HAL特定线程

在支持真实线程的平台，AP\_HAL会为该平台创建大量的线程来支持基本的操作。比如，在PX4中下面的HAL特定线程会被创建：

* The UART thread, for reading and writing UARTs (and USB)
* The timer thread, which supports the 1kHz timer functionality described above
* The IO thread, which supports writing to the microSD card, EEPROM and FRAM

查看一下每个AP\_HAL执行操作里面的Scheduler.cpp文件，就知道创建了什么线程和线程的实时优先级。

如果你有一个Pixhawk，你现在就可以设置一个调试控制台的连线连接到nsh控制台（串行端口5）。用57600波特率连接。但你连接上后，用“ps”命令获取 下面的一些信息：

PID PRI SCHD TYPE NP STATE NAME

0 0 FIFO TASK READY Idle Task()

1 192 FIFO KTHREAD WAITSIG hpwork()

2 50 FIFO KTHREAD WAITSIG lpwork()

3 100 FIFO TASK RUNNING init()

37 180 FIFO TASK WAITSEM AHRS\_Test()

38 181 FIFO PTHREAD WAITSEM **<**pthread**>**(20005400)

39 60 FIFO PTHREAD READY **<**pthread**>**(20005400)

40 59 FIFO PTHREAD WAITSEM **<**pthread**>**(20005400)

10 240 FIFO TASK WAITSEM px4io()

13 100 FIFO TASK WAITSEM fmuservo()

30 240 FIFO TASK WAITSEM uavcan()

在这个例子中

**PX4开发者指南**

1. 引言

这文档大部分是翻译PX4官方开发指南，但是其中的的绿色字体就是我自己的实践经验或者案例。

注意：仅限于开发者！这个指南旨在推动开发而不是为消费者准备。

这个指南介绍了如何在PX4系统架构下开发，它使开发人员能够：

1. [了解系统的概况](http://dev.px4.io/starting-initial-config.html)
2. 获取和修改PX4飞行栈和PX4中间件。
3. 在骁龙飞控、Pixhawk和Pixfalcon上部署PX4
   1. 更新日志

2015-10-06

初始发布

* 1. 许可证Licence

PX4开发者文档发布为 [CC BY 4.0](https://creativecommons.org/licenses/by/4.0/).。具体请查看细节[Github Repository](https://github.com/PX4/Devguide)。

1. 入门
   1. 初始配置

在开始开发PX4之前，系统需要按照默认配置进行初始配置，以保证硬件设置合适，被检测到。下面的一个视频介绍了Pixhawk硬件和地面站设置过程，有支持的飞机类型的列表。

* 1. 工具链安装

PX4代码可以在 [Mac OS](http://dev.px4.io/starting-installing-mac.html), [Linux](http://dev.px4.io/starting-installing-linux.html) or [Windows](http://dev.px4.io/starting-installing-windows.html).上进行开发，建议在Mac OS和Linux上进行开发，因为图像处理和高级导航在windows上不容易开发。如果不确定，新的开发者应默认用Linux和当前Ubuntu长期支持版本（[Ubuntu LTS edition](https://wiki.ubuntu.com/LTS)）。

* 1. 开发环境

开发环境安装涉及到下面几种：

* [Mac OS](http://dev.px4.io/starting-installing-mac.html)
* [Linux](http://dev.px4.io/starting-installing-linux.html)
* [Windows](http://dev.px4.io/starting-installing-windows.html)

如果你对Docker熟悉，你可以使用其中一个容器：[Docker Containers](http://dev.px4.io/advanced-docker.html)。

（这里只翻译了Linux下开发环境的搭建）

### 1.2.1 Linux开发环境搭建（windows等系统的就不翻译了，[自己看官网](http://dev.px4.io/starting-installing.html)）

我们使用Debian / Ubuntu LTS 作为Linux的标准支持版本。，但是也支持Cent OS and Arch Linux的发行版本。

* + 1. 权限设置

注意：永远不要使用“sudo”来修复权限问题，否则会带来更多的权限问题，需要重装系统来解决。

把用户添加到用户组“dialout”

sudo usermod -a -G dialout $USER

然后注销后，重新登录，因为重新登录后所做的改变才会有效。

* + 1. 安装

更新包列表，安装下面编译PX4的依赖包。PX4主要支持的家族：

* NuttX based hardware: [Pixhawk](http://dev.px4.io/hardware-pixhawk.html), [Pixfalcon](http://dev.px4.io/hardware-pixfalcon.html)
* Snapdragon Flight hardware: [Snapdragon](http://dev.px4.io/hardware-snapdragon.html)
* Raspberry Pi hardware: [Raspberry Pi 2](http://dev.px4.io/hardware-pi2.html)
* Host simulation: [jMAVSim SITL](http://dev.px4.io/simulation-sitl.html) and [Gazebo SITL](http://dev.px4.io/simulation-gazebo.html)

注意：安装 [Ninja Build System](http://dev.px4.io/starting-installing-linux-boutique.html#ninja-build-system)可以比make更快进行编译。如果安装了它就会自动选择使用它进行编译。

sudo add-apt-repository ppa:george-edison55/cmake-3.x -y

sudo apt-get update

sudo apt-get install python-argparse git-core wget zip \

python-empy qtcreator cmake build-essential genromfs -y

# simulation tools

sudo apt-get install ant protobuf-compiler libeigen3-dev libopencv-dev openjdk-7-jdk openjdk-7-jre clang-3.5 lldb-3.5 -y

* + 1. NuttX的基本硬件

Ubuntu配备了一系列代理管理，这会严重干扰任何机器人相关的串口（或usb串口），卸载掉它也不会有什么影响：

sudo apt-get remove modemmanager

更新包列表和安装下面的依赖包。指明版本的包需要安装指定的版本的包。

sudo add-apt-repository ppa:terry.guo/gcc-arm-embedded -y

sudo apt-get update

sudo apt-get install python-serial openocd \

flex bison libncurses5-dev autoconf texinfo build-essential \

libftdi-dev libtool zlib1g-dev \

python-empy gcc-arm-none-eabi -y

如果gcc-arm-none-eabi版本导致PX4/Firmware编译错误，请参考 [the bare metal installation instructions](http://dev.px4.io/starting-installing-linux-boutique.html#Toolchain Installation)手动安装4.8版本。

* 1. 编译代码

PX4可以在控制台或者图形界面/IDE开发

* + 1. 在控制台编译

在去到图形界面或者IDE前，验证系统设置的正确性非常重要，因此打开控制台。在Ubuntu，单击启动栏，搜索“terminal”（或者trl+alt+T）。

终端在家目录启动，我们默认去到'~/src/Firmware' 然后，克隆顶层资源库。有经验的开发者可以克隆自己的复制的资源库。

mkdir -p ~/src

cd ~/src

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

cd Firmware

git submodule update --init --recursive

cd ..

注意：没有安装git的朋友可以先安装git再执行 的命令

sudo apt-get install git

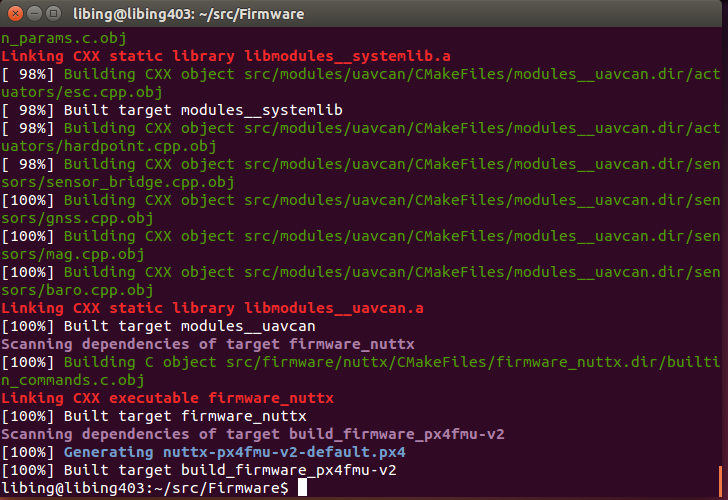
现在到时候由源代码编译二进制代码了。但在直接使用硬件前，推荐先进行仿真。喜欢在图形界面开发环境工作的用户也应该继续完成下面部分。

* + 1. 基于NuttX / Pixhawk的硬件板

cd Firmware

make px4fmu-v2\_default

大概10分钟那样就会编译成功，我实在虚拟机的Ubuntu上编译的。编译成功如下图所示



注意到“make”是一个字符命令编译工具，“px4fmu-v2”是硬件/ardupilot版本，“default”是默认配置，所有的PX4编译目标遵循这个规则。

成功编译的最后输出是这样的：

[100%] Linking CXX executable firmware\_nuttx

[100%] Built target firmware\_nuttx

Scanning dependencies of target build\_firmware\_px4fmu-v2

[100%] Generating nuttx-px4fmu-v2-default.px4

[100%] Built target build\_firmware\_px4fmu-v2

通过在命令后面添加‘upload’，编译的二进制程序就会通过USB上传到飞控硬件：

make px4fmu-v2\_default upload

成功上传的最后输出是这样的：

Erase : [====================] 100.0%

Program: [====================] 100.0%

Verify : [====================] 100.0%

Rebooting.

[100%] Built target upload

我使用的是linux开发环境和pixhawk硬件，其他的开发环境和pixhawk的指南我就不翻译了，详细可以参考官网<http://dev.px4.io/index.html>

* 1. 投稿&开发

核心开发团队和社区的联系信息可以在下面找到。PX4项目使用了三个分支Git branching model:

* [master](https://github.com/px4/firmware/tree/master) 默认情况下不稳定，可以看到快速开发。
* [beta](https://github.com/px4/firmware/tree/beta) 已充分测试，面向飞行测试者。
* [stable](https://github.com/px4/firmware/tree/stable) 指向最新的发布分支。.

我们尝试通过[rebases保持一个线性的历史](https://www.atlassian.com/git/tutorials/rewriting-history)，[避免Github flow](https://guides.github.com/introduction/flow/)。但是由于全球的开发队伍和快速的开发转移，我们会定期分类合并。

可以这样为代码贡献新功能，通过[注册Github](https://help.github.com/articles/signing-up-for-a-new-github-account/)，[复制资源库](https://help.github.com/articles/fork-a-repo/)，[创建分支](https://help.github.com/articles/creating-and-deleting-branches-within-your-repository/)，加入你的改变，最后[推送请求](https://help.github.com/articles/using-pull-requests/)。当它们通过我们的[持续的综合测试](https://en.wikipedia.org/wiki/Continuous_integration)后，更新就会合并。

所有的投稿（贡献）必须在 [BSD 3-clause license](https://opensource.org/licenses/BSD-3-Clause)许可下进行，所有的代码在使用上不能提出任何的，进一步的限制。

* + 1. 测试飞行结果

飞行测试对于保证质量非常重要，请从microSD卡上传飞行日志到 [Log Muncher](http://logs.uaventure.com/)，并在[论坛](http://groups.google.com/group/px4users)分享一个的连接，附带有书面飞行报告。

* + 1. 论坛和聊天
* [Google+](https://plus.google.com/117509651030855307398)
* [Gitter](https://gitter.im/PX4/Firmware?utm_source=badge&utm_medium=badge&utm_campaign=pr-badge&utm_content=badge)
* [PX4 Users Forum](http://groups.google.com/group/px4users)

每周开发电话

PX4团队在每周同时进行电话联系（通过[Mumble](http://mumble.info/)客户端连接）。

* TIME: 19:00h Zurich time, 1 p.m. Eastern Time, 10 a.m. Pacific Standard Time
* Server: mumble.dronecode.org
* Port: 10028
* Channel: PX4 Channel
* The agenda is announced the same day on the [px4users forum](http://groups.google.com/group/px4users)
  1. 总结

本文介绍了linux ubuntu下PX4开发环境的搭建，PX4编译和上传到飞控的命令。介绍了PX4团队如何进行开源项目的开发，介绍了其运行的一些概况，有兴趣可以根据这些信息和相关链接深入了解，为开源项目贡献自己的代码。甚至参与到PX4的开发中来，成为PX4团队的一员。

1. 相关概念
   1. 飞行模式/操控

飞行模式定义了系统在任意时间的状态。用户可以通过遥控器的开关或者[地面控制站](http://dev.px4.io/qgroundcontrol-intro.html)切换飞行模式。

### 飞行模式概述

### 1手动模式（MANUAL ）

**1.1固定翼飞机/多旋翼/船：**飞行员的控制输入直接传给输出混频器。

**1.2多旋翼：**

**（1）特技（ACRO）：飞行员的输入作为滚转、俯仰和偏航的指令传送给飞控。这允许多旋翼可以完全颠倒。油门被直接传送到输出混合器。**

**（2）（RATTITUDE）：飞行员的输入作为滚转、俯仰和偏航的命令输入到飞控。如果输入不作为滚转和俯仰角度的指令和偏航速率指令。油门直接输入到混频器。（3） （ANGLE）：飞行员的输入作为滚转和俯仰角度指令和偏航率指令传给飞控。油门直接输出到混频器**

### **2**辅助模式

**2.1 高度控制**

**（1）固定翼飞机：**当滚转、俯仰和偏航输入都在中间位置（小于一定的死区范围），飞机会返回到直线和水平飞行姿态，并保持当前高度。它会由于风的作用发生漂移。

（2）**多旋翼**：当滚转、俯仰和偏航输入作为手动模式，油门输入会以预定义的最大速率爬升或者下沉。油门有最大的死区。

**2.2位置控制POSCTL**

**（1）固定翼飞机:** 在空档位置的输入会给定水平飞行，如果需要保持直线，它会对风进行对抗。

（2）**多旋翼**：相对于地面，滚转控制左右的速度，俯仰控制前后的速度。当滚转和俯仰都在中心时（在死区内），多旋翼会保持原来的位置。偏航控制以手动模式控制偏航速率。在ALTCTLM模式，油门控制爬升或下降速率。

### ****3.自动****

**3.1自动悬停（AUTO\_LOITER）**

（1）**固定翼飞机**：飞机悬停在当前位置的当前高度（或者可能在当前位置稍微上面的位置）

（2）**多旋翼**：多旋翼悬停在当位置和当前高度。

**3.2自动返航（AUTO\_RTL）:**

**（1）固定翼飞机：**飞机返回到家位置并在家的上空以圆形方式盘旋。

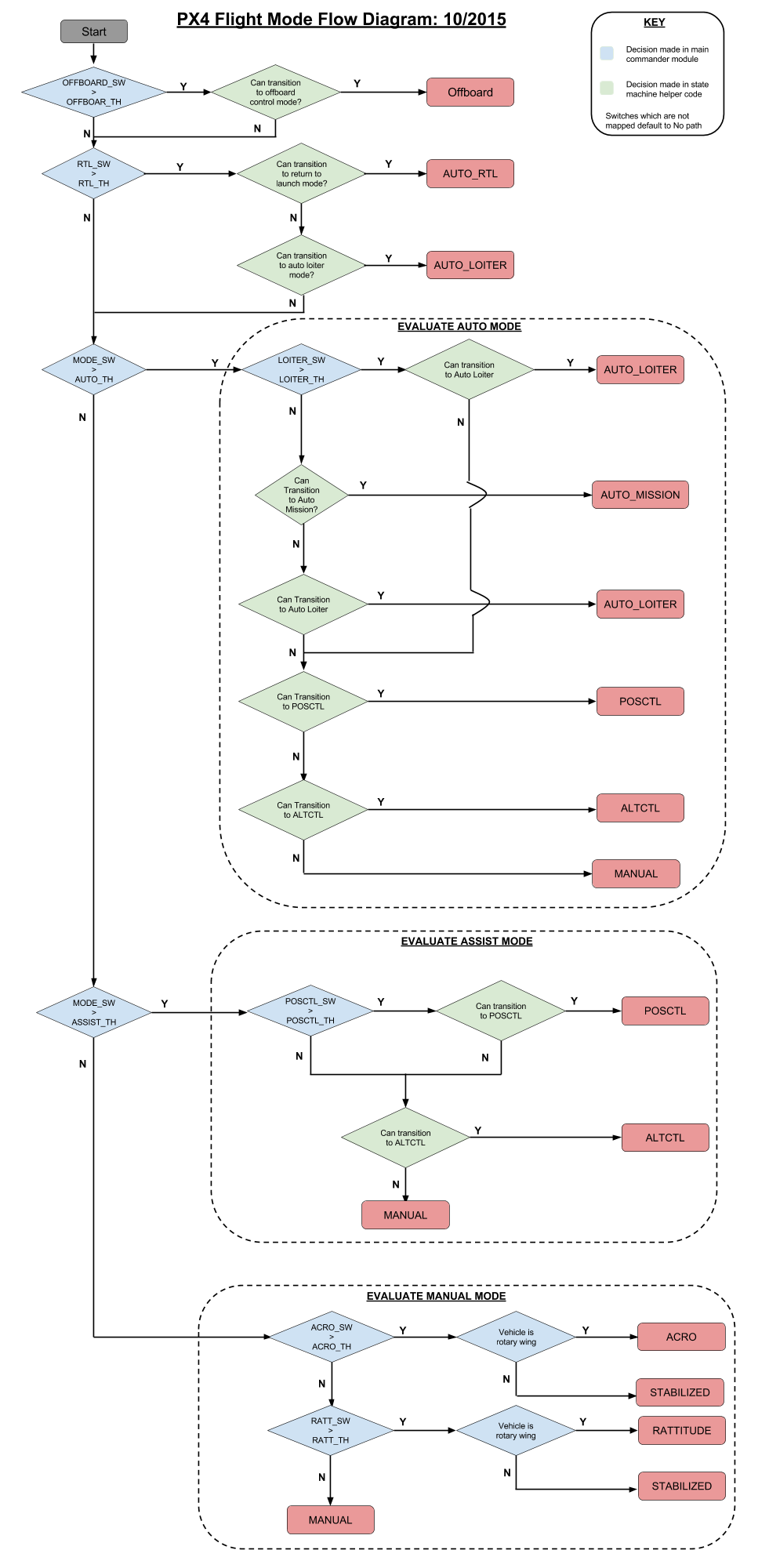
**（2）多旋翼**：多旋翼飞机以当前高度（如果高于家的位置+悬停的高度）或者以悬停高度（如果比当前高度高）直线返回，然后自动着陆。

**3.3自动任务**

所有的系统类型：飞行器执行从地面控制站（GCS）发出的程序任务。如果没有收到任务，飞机会悬停在当前位置。

**4 机外**：在这个模式下，位置、速度或者姿态参考/目标/设定值是由一个通过串行线缆和MAVlink连接的伴随计算机提供。机外设定值可以由APIs比如 [MAVROS](https://github.com/mavlink/mavros) 或者 [Dronekit](http://dronekit.io/)提供。

飞行模式调整图



* 1. 架构概述

PX4由主要的两层组成：[PX4飞行栈](http://dev.px4.io/concept-flight-stack.html)，自动驾驶软件和[PX4中间件](http://dev.px4.io/concept-middleware.html)，这是一种通用的机器人中间件，可以支持任何类型的自主机器人。

所有的飞机类型，实际上所有机器人系统包括船，都共享一个代码库。整个系统的设计是[反应型](http://www.reactivemanifesto.org/)的，这一意味着：

（1）所有的功能被分为可以交换的组件。

（2）通过异步消息传递通信。

（3）系统可以处理各种任务载荷。

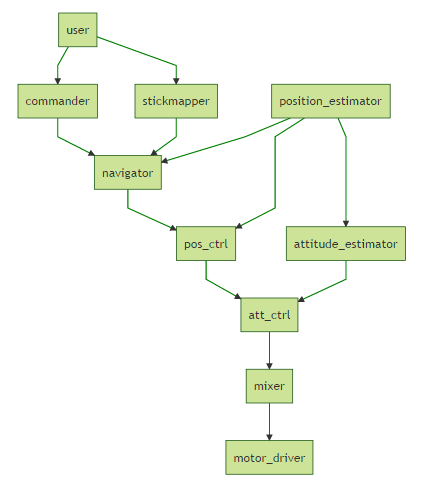
除了考虑这些的运行外，它的模块化最大限度提高可重用性。

### 上层软件结构

下面的每个板块是一个独立的模块，它在代码，依赖关系和甚至运行是独立的。每个箭头都是通过uORB发布/订阅实现连接。

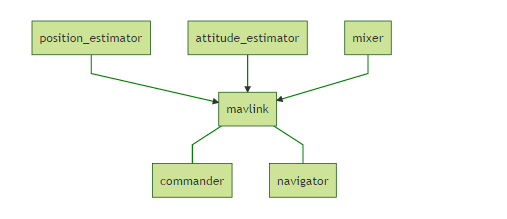
tip：PX4架构可以非常快速、方便地交换这些单一的模块，甚至在运行时。

控制器/混频器由特定的飞机类型指定（比如直升机，垂直升降机或飞机），但是在更高层的任务管理模块，比如指令和导航器是在平台间是共享的。



### 与地面站的通信架构

飞机与地面站的交互通过“业务逻辑”应用，包括指令（通用指令和控制，比如arming），导航器（接收任务，把任务转给下一层导航原语）和mavlik应用程序接收MAVLink数据包并转化为板上的uORB数据结构。架构采用这种分离明确地减少了MAVLink对系统的深度依赖。MAVLink应用程序也使用大量的传感器数据和状态估计，并传送给地面控制站。

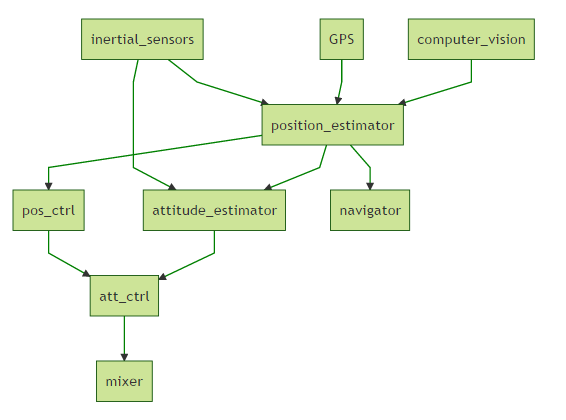


* 1. PX4飞行栈

PX4飞行栈是自主无人的一个引导集合，导航和控制算法。包括固定翼、多旋翼和垂直起降飞机，以及姿态和位置估测器。

估算和控制架构

下图表示了一个典型模块的简单实现。根据飞机，其中一些可以组合成一个简单的应用程序（特定想要一个特定飞机模型预测控制器）。



* 1. PX4中间件

PX4中间件组成嵌入式传感器和发布/订阅基本中间件的重要的设备驱动，这些驱动连接传感器和应用程序，运行[飞行控制](http://dev.px4.io/concept-flight-stack.html)。

使用发布/订阅方案的意思是：

（1）系统是反应型的：担忧新数据可用时，它会不断更新。

（2）它是完全并行运行的。

（3）系统部件可以使用来自任何安全线程方式的数据

* 1. 混合驱动器

PX4的架构保证了不同机身布局，核心控制器不需要进行特定情况的处理。

混合器意味着采取强制命令（比如右转），把它们转化为控制电机或者伺服驱动器的指令。

对于带有一个伺服/副翼，这意味着它们一个高，另一个就低。同样，对于直升机：俯仰向前需要改变所有电机的转速。

混合器逻辑从实际姿态控制器中分离开来，极大提高了可重用性。

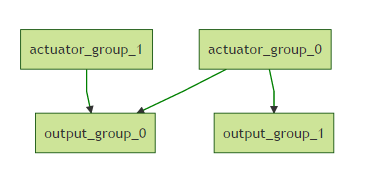
### 关键概念

PX4使用控制组（输入）和输出组。概念上非常简单：一个控制组是对于核心飞行控制说的，比如姿态，或者对于有效载荷来说，是万向节。一个输出组是一个物理总线，比如前8个伺服器PWM输出。每个组有8个标准化（-1…+1）命令端口，通过混合器可以映射和缩放。一个混合器定义了8个控制信号如何连接到8个输出端。

对于一个简单飞机控制0（roll）直接连接到输出0（elevator）。对于一个直升机，情况会有所不同：控制0（roll）连接到所有四个电机和油门。

### 映射

因为这里多种控制组（比如飞行控制、有效载荷等）和多种输出组（前8个PWM输出、UAVCAN等）。一个控制组可以给多个输出组发送指令。



### PX4混合器的定义

在目录ROMFS/px4fmu\_common/mixers implement mixers的文件用来预定义机身类型。

它们可以用作为定制或者一般测试目的的基础。

语法

混频器的定义是一个文本文件：每行以一个简单的大写字母开头，后面跟着一个重要的冒号。其他的所有行都被忽略，意味着解释性文本可以自由地与定义混合。

每个文件可能定义不止一个混频器。混频器给执行器的分配就是指定读取混频器定义的设备

一个混频器以下面这样一行的形式开始

<tag>: <mixer arguments>

标签选择了混频器的类型，‘M’表示一个简单的加法混频器，‘R’表示多旋翼混频器等。

### 空混频器

一个空混频器不会使用控制信号，也不会产生简单的驱动输出，它的值一直是0。在混频器的集合里，典型的空混频器作为一个占位符来取得特定驱动器的输出模式。

空混频器的定义形式是

Z:

简单的混频器

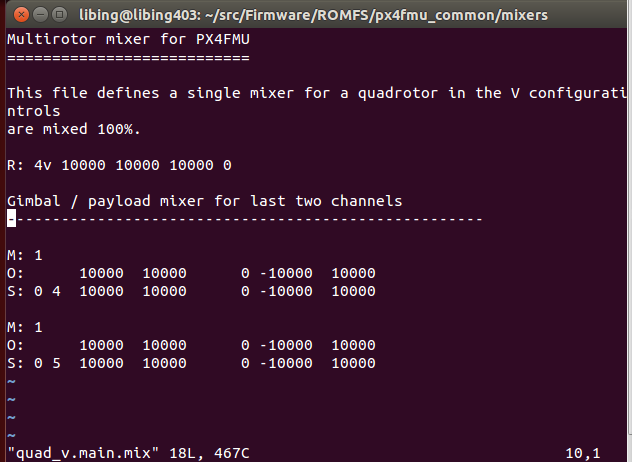
一个简单的混频器把0或者跟多输入结合成一个简单的驱动器输出。输入被缩放，混频函数在使用输出因子前累加结果。

一个简单的混频器可以这样开始定义

M: <control count>

O: <-ve scale> <+ve scale> <offset> <lower limit> <upper limit>

比如打开quad\_v.main.mix的内容如下



如果<control count>是0，累加的和肯定是0，混合器会输出一个固定的值，这个值是<offset>，在<lower limit> 和 <upper limit>的约束的范围。

第二行定义输出缩放因子和缩放参数，和上面讨论的一样。虽然计算进行浮点运算，但是存储在定义文件里以10000的因子进行了缩放，比如-0.5编码成-5000。

定义在<control count>入口继续描述控制输入和他们的缩放，形式为

S: <group> <index> <-ve scale> <+ve scale> <offset> <lower limit> <upper limit>

其中<group>的值确定控制组会读取哪个缩放因子，<index>值是在组内进行偏置。这些值指定读取混合器定义的设备。

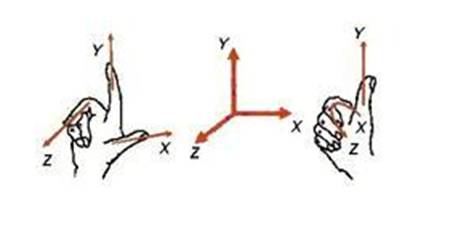
当使用混频器飞行器，混频器组的0是飞行器姿态控制组，索引值0分别通过滚转、俯仰和油门。

1. 理论学习部分

根据学习北航《多旋翼飞行器设计与控制》的课件，结合自己理解、推导写的笔记，以此加深理解，与大家交流。

* 1. 坐标系
     1. 右手定则

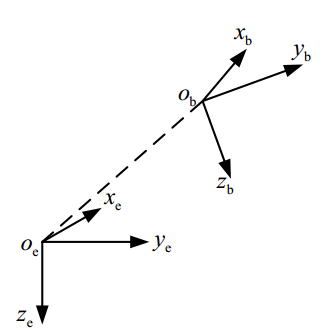
采用的坐标系和定义的角度正方向沿用右手定则。



* + 1. 惯性坐标系与机体坐标系定义

地球表面惯性坐标系（下标e）用于研究多旋翼飞行器相对于地面的运动状态，确定机体的空间位置坐标。它忽略地球曲率，即将地球表面假设成一张平面。在地面上选一点作为多旋翼飞行器起飞位置。

机体坐标系（下标b），其原点 取在多旋翼的重心上，坐标系与多旋翼固连。 轴在多旋翼对称平面内指向机头。



定义三个单位向量

在惯性坐标系中，沿着坐标轴的单位向量可表示为

在机体坐标系下，沿的坐标轴的单位向量满足（注：左上标b表示向量在机体坐标系的表示）

在地球惯性坐标系中，沿的坐标轴的单位向量可表示为（注：左上标e表示向量在惯性坐标系的表示）

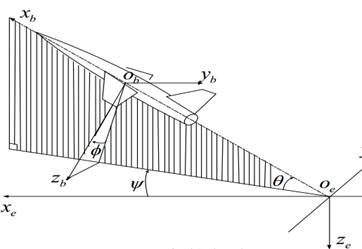
* 1. 姿态表示
     1. 欧拉角的定义

机体坐标系与地面惯性坐标系之间的夹角就是飞机的姿态角，又称欧拉角。

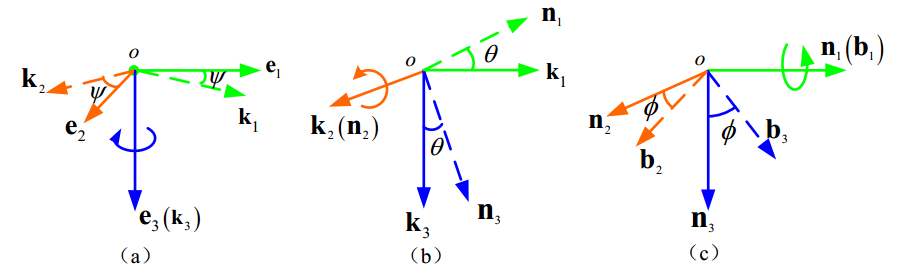
（1）俯仰角： 机体轴与地平面（水平面 ）之间的夹角，飞机抬头为正。

（2）偏航角（方位角）：机体轴在水平面上的投影与地轴之间的夹角，以机头右偏为  
正。

（3）滚转角（倾斜角）：飞机对称面绕机体轴 转过的角度，右滚为正。



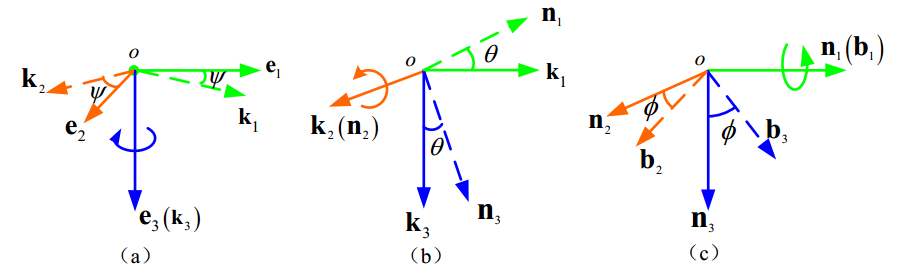
可以通过绕轴分别旋转欧拉角 将地球表面惯性坐标系转动到机体坐标系。



* + 1. 欧拉角变化率与机体角速度的关系

若机体旋转的角速度为

那么有（注：上标b表示向量在机体坐标系下的坐标表示，这里的机体坐标系当然指的是经过三次旋转后的机体坐标系，如图所示）。



在最新的机体坐标系下，显然有

如图（c），可以由绕轴转过而得到，因此

如图（c），（b），（d）所示，可以由经过旋转角度得到，因此

因此

因此有

进一步有

其中

当时，出现奇异问题。

当时，可以认为

* + 1. 旋转矩阵

（1）定义旋转矩阵为

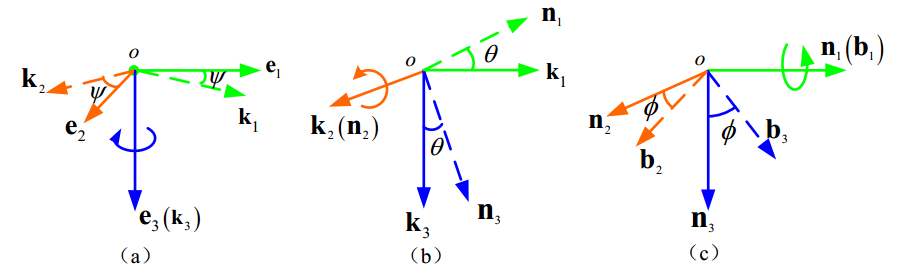
右侧的上下标表示从机体坐标系b到旋转到惯性坐标系e的旋转矩阵。

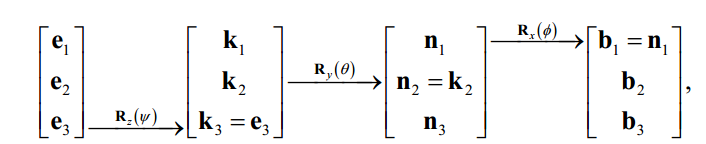
其中满足

旋转矩阵中的矢量满足

左上标e表示向量在惯性坐标系中的表示，左上标b表示向量在机体坐标系中的表示。

从地球惯性坐标系到机体坐标系的旋转可以通过三步来完成。





其中

旋转矩阵是正交反对称矩阵，因此有

同时注意观察,,,会发现

因此有

令

反求欧拉角

时出现奇异问题，在奇异的情况下，人为设定

（2）旋转矩阵的导数与机体角速度的关系

仅考虑刚体旋转（不考虑平动），由动力学知识可知，对任意向量

* 1. 多旋翼的控制模型
     1. 刚体运动学模型

多旋翼的建模流程如下图所示

假设1：飞行器是刚体；

假设2：质量和转动惯量是不变的；

假设3：飞行器重心和中心一致；

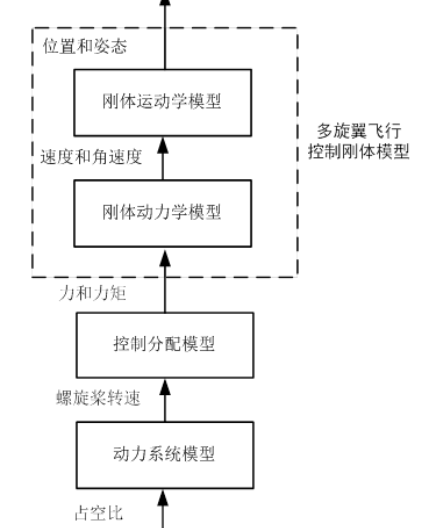
假设4：飞行器只受重力和螺旋桨的升力，其中螺旋桨拉力沿，而重力沿方向。

假设5：奇数标号的螺旋桨逆时针旋转，偶数标号的螺旋桨顺时针旋转。

（1）基于欧拉角的模型

（2）基于旋转矩阵的模型

（3）基于四元数的模型



* + 1. 多旋翼的动力学模型

（1）惯性坐标系下的位置动力学模型

（2）机体坐标系下的位置动力学模型

两边求导得

因此

其中用到了

（3）姿态动力学模型

基于假设1-3，机体坐标系下多旋翼姿态动力学方程

其中，是陀螺力矩，是螺旋桨在机体上产生的力矩，，表示多旋翼的转动惯量。

（4）综合多旋翼飞行控制刚体模型

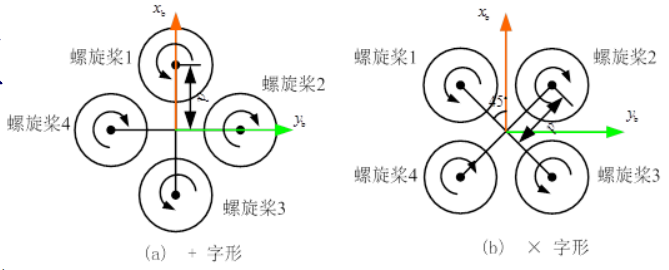
模型1（基于欧拉角）

模型2（基于旋转矩阵）

模型3（基于四元数）

* 1. 控制分配模型

（1）标准四旋翼



对于标准四旋翼，螺旋桨作用在机体上的拉力为

对于十字形四旋翼，螺旋桨产生的力矩为

其中，（参考空气空力学等）

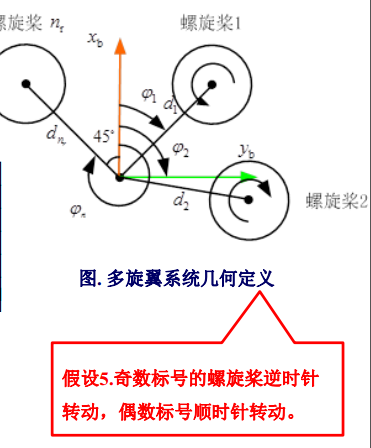
因此有

对于x字形四旋翼，螺旋桨产生的力矩为

其中，（参考空气空力学等）

因此有

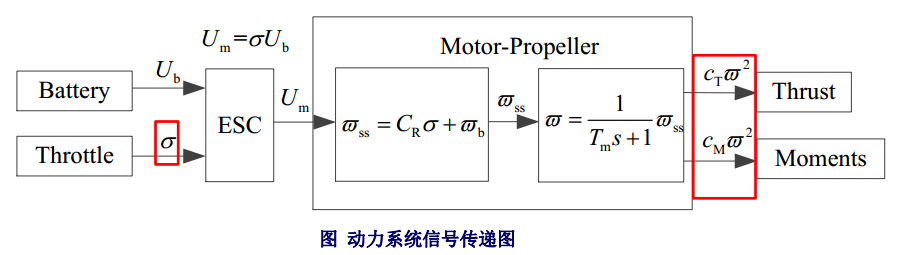
（2）多旋翼



其中，

特别注意，要把上述公式套用到四轴、六轴等多旋翼，对电机的标号和旋转方向要和上图多旋翼的标记一致，即从x轴右边开始顺时针标记1,2,3,...,电机，奇数号的螺旋桨逆时针转动，偶数号的螺旋桨顺时针旋转。

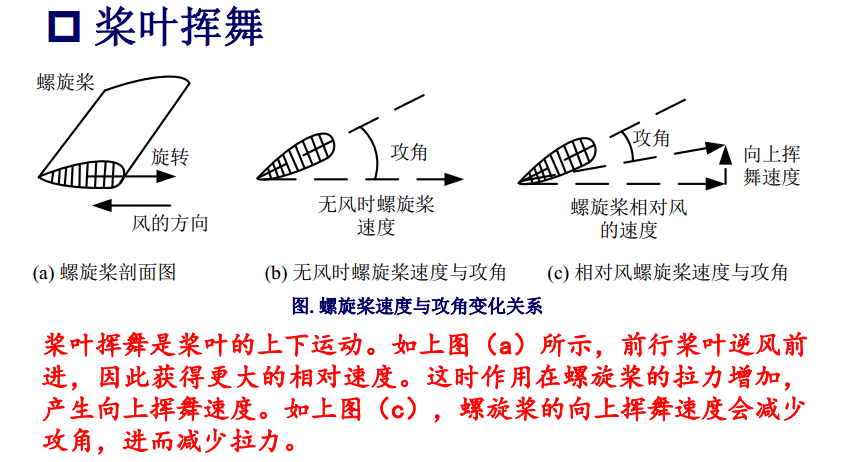
* + 1. 动力系统模型

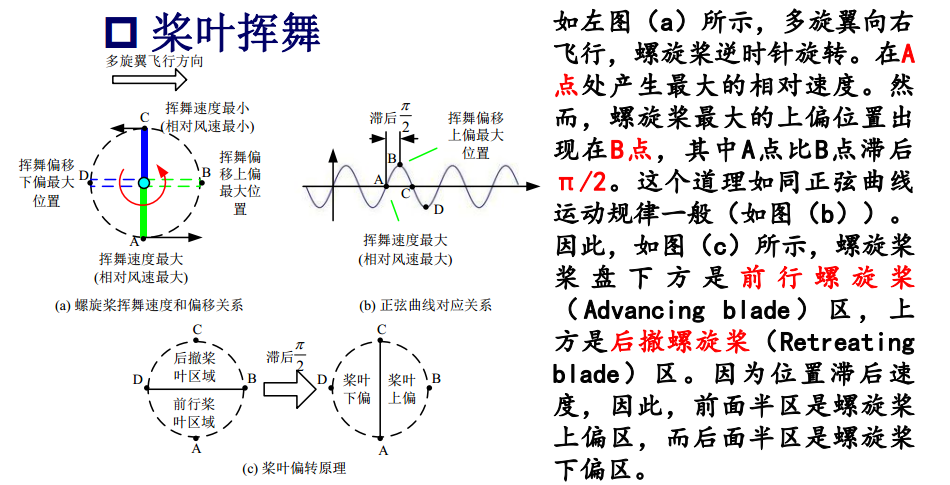


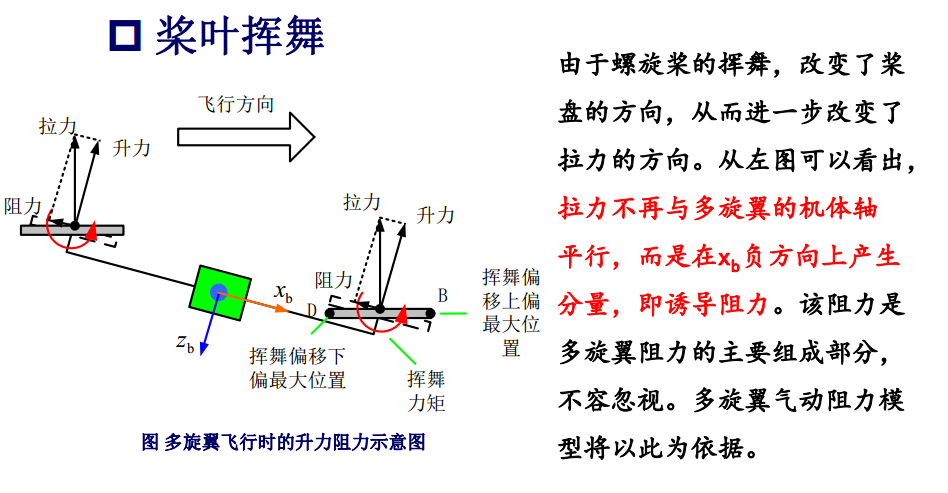
模型

其中电机油门为输入，电机转速为输出，为电动机动态响应常数。

* + 1. 多旋翼的气动阻力模型







**气动阻力模型**

前面已经推导过多旋翼机体坐标系下的位置动力学模型

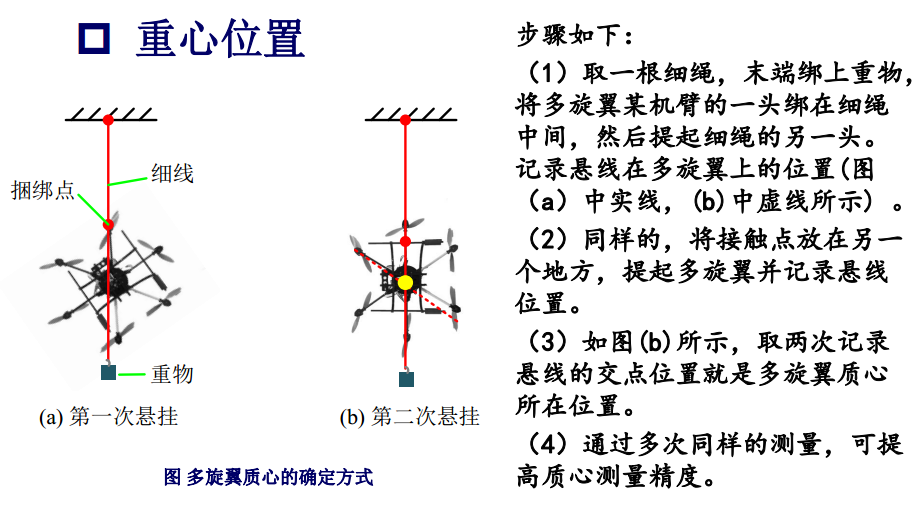
把上式展开就有，多旋翼在机体轴，上的速度为

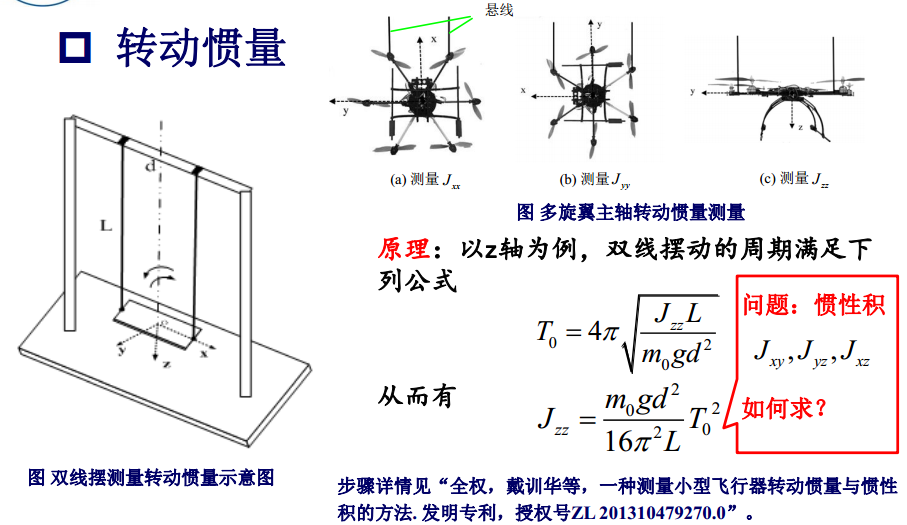
阻力表示如下

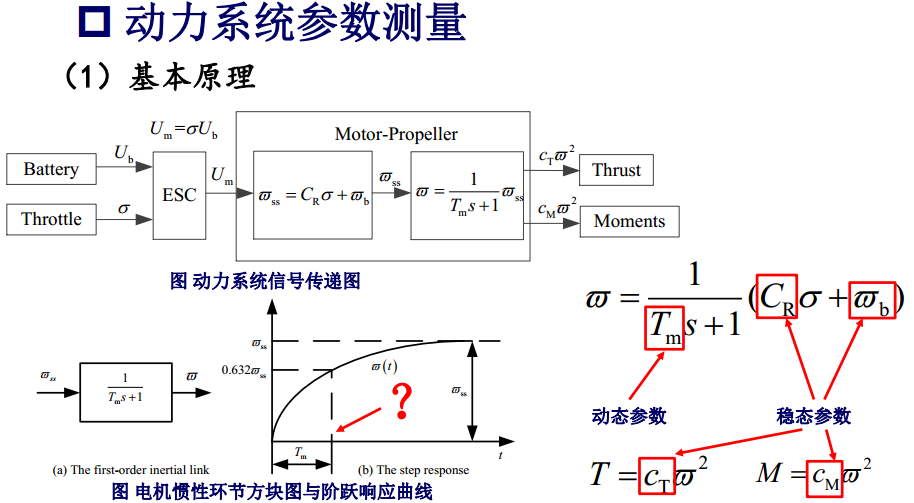
其中，分别表示在机体轴，上的阻力，而表示阻力系数。

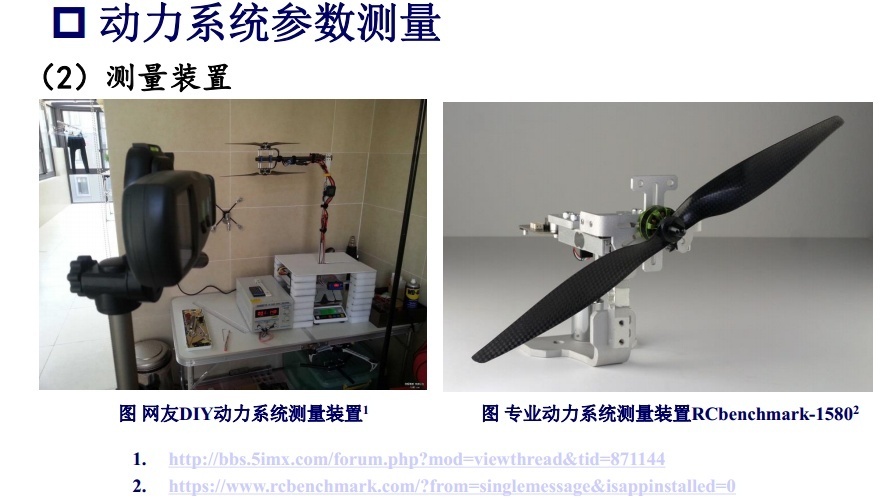
因此，多旋翼的气动阻力模型如下

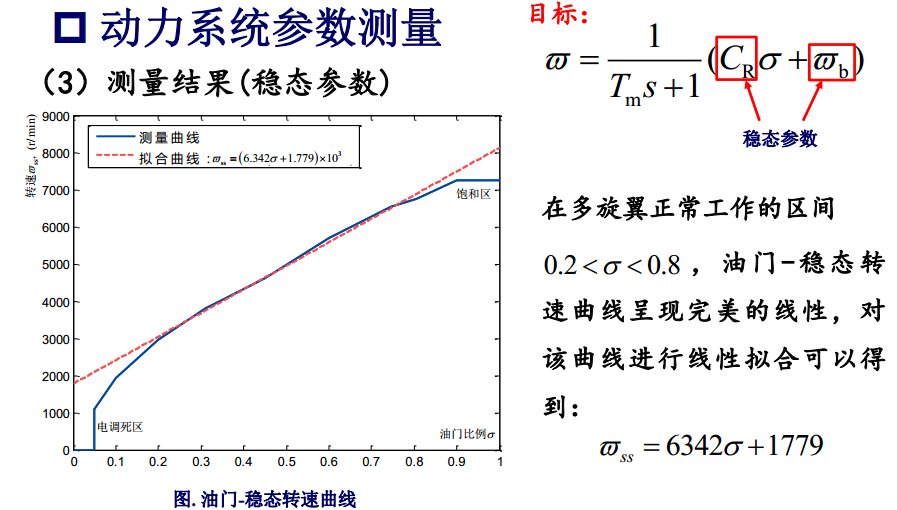
* + 1. 多旋翼模型参数测量

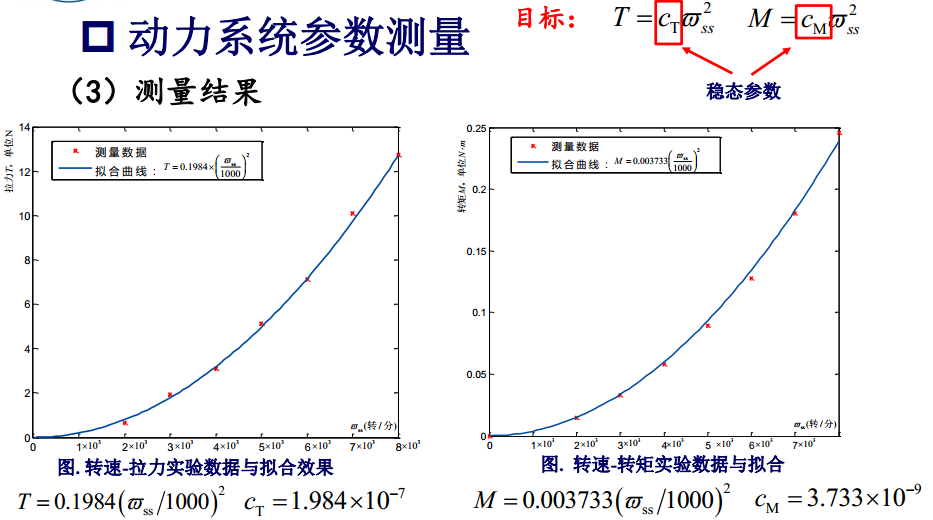


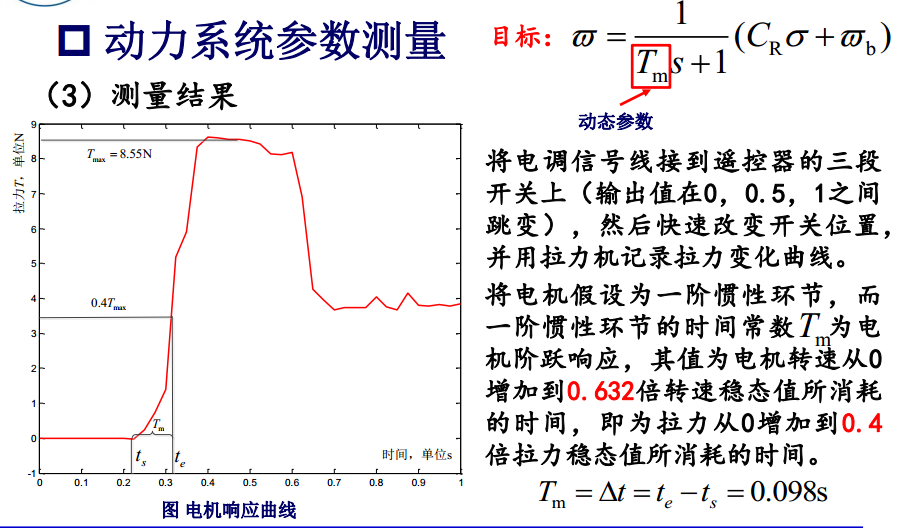












# 《人类简史》

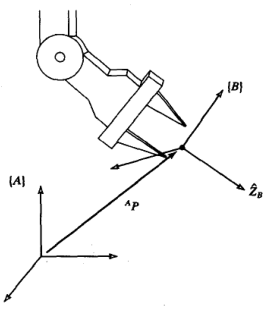
《机器人学导论》--John J. Craig

1. 000Equation Chapter 1 Section 1空间描述和变换
   1. 位置描述

在坐标系{A}中，空间中一个点的位置用一个3×1的矢量描述

000\\* MERGEFORMAT (-)

* 1. 姿态描述



点的位置可以用矢量描述，物体的姿态可以用固定在其上的坐标系描述。描述坐标系{B}的一种方法是用{A}坐标系的三个主轴单位矢量表示。

用来表示{B}主轴方向的单位矢量，当用坐标系{A}的坐标来表达时，被写成 ，{B}相对于{A}的表达为

000\\* MERGEFORMAT (-)

标量可用每个矢量在其参考坐标系中单位方向上的投影来表示，投影可通过点积得到，因此

000\\* MERGEFORMAT (-)

上式中矢量的右上标被省略了。实际上，只要点积的对矢量是在同一坐标系中描述，那么坐标系的选择是任意的。观察上式，也可发现矩阵中行向量是单位矢量{A}在{B}中的表达。即

. 000\\* MERGEFORMAT (-)

因此，坐标系{A}相对于{B}的描述，可由转置得到，即

000\\* MERGEFORMAT (-)

这表明旋转矩阵的逆可由它的转置得到，实际上旋转矩阵也是正交矩阵。

* 1. 坐标系的描述

在机器人学中，位置和姿态经常成对出现，我们将此组合称作坐标系。因此一个坐标系可等价的用一个位置矢量和一个旋转矩阵来描述，例如

000\\* MERGEFORMAT (-)

* 1. 坐标系的一般映射：齐次矩阵变换

齐次变换矩阵有三方面的含义：

（1）它是坐标系的描述。表示相对于坐标系{A}的坐标系{B}。特别是，的各列是坐标系{B}主轴方向的单位矢量。确定了{B}的原点。

（2）它是变换映射。是映射。

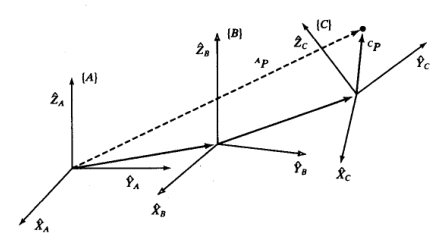
（3）它是变换算子。将 变换为。

其次变换矩阵是旋转和平移的组合，具体可写成

000\\* MERGEFORMAT (-)

由此可见，坐标系和变换都可以通过位置矢量和姿态来描述。一般坐标系主要用于描述，而变换用来表示映射或算子。

* 1. 变换算法



* + 1. 混合变换

000\\* MERGEFORMAT (-)

由此定义

000\\* MERGEFORMAT (-)

由变换的定义，可得

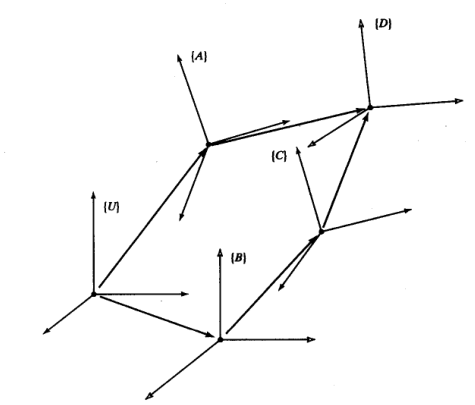
000\\* MERGEFORMAT (-)

* + 1. 逆变换

通过一般的方法求矩阵的逆可以得到逆变换，但是结合旋转矩阵的性质，可以更简便得到逆变换，编程时分开计算可降低维数，减少求逆的计算时间。

000\\* MERGEFORMAT (-)

* 1. 变换方程



坐标系{D}可以通过两个不同的变换得到

000\\* MERGEFORMAT (-)

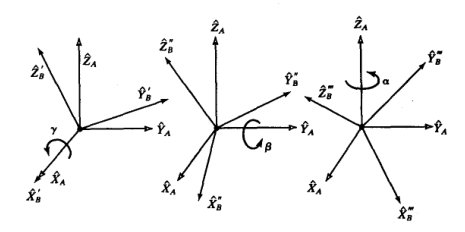
000\\* MERGEFORMAT (-)

两个表达式构成一个方程

000\\* MERGEFORMAT (-)

如果有n个未知变换和n个变换方程，这个变换可以从变换方程解出。

* 1. 姿态的其他描述方法
     1. X-Y-Z固定角坐标系



描述坐标系{B}的姿态：

首先将坐标系{B}和一个已知的参考坐标系{A}重合，先将{B}绕轴旋转角，再绕轴旋转，最后绕轴旋转角。每次旋转都是绕着固定参考坐标系{A}的轴。

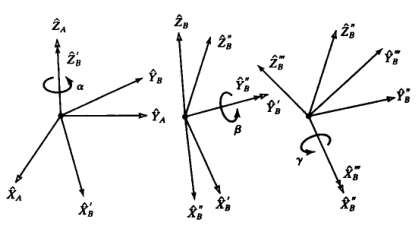
把每个旋转看做算子，等价旋转矩阵为

000\\* MERGEFORMAT (-)

计算得到

000\\* MERGEFORMAT (-)

* + 1. X-Y-Z欧拉角



坐标系{B}的描述：

首先将坐标系{B}与一个已知坐标系{A}重合，先将{B}绕旋转角，再将{B}绕旋转角，最后将{B}绕旋转角。每次都是绕运行坐标系{B}的各轴旋转。

把每个旋转看成坐标系的描述，可立即写出

000\\* MERGEFORMAT (-)

即{B}相对于{A}的最终姿态为

000\\* MERGEFORMAT (-)

计算得到

000\\* MERGEFORMAT (-)

带有“撇”号表示是由X-Y-Z欧拉角参数化的旋转矩阵。发现和X-Y-Z固定轴旋转矩阵是完全相同的。

这两种变换的矩阵一样，因此求解欧拉角的方法一样。由旋转矩阵求欧拉角，设

000\\* MERGEFORMAT (-)

当时

000\\* MERGEFORMAT (-)

当时，一般取，结果如下

若，

000\\* MERGEFORMAT (-)

若，

000\\* MERGEFORMAT (-)

* + 1. Z-Y-Z欧拉角

坐标系{B}的描述：

首先将坐标系{B}与一个已知坐标系{A}重合，先将{B}绕旋转角，再将{B}绕旋转角，最后将{B}绕旋转角。每次都是绕运行坐标系{B}的各轴旋转。

等效旋转矩阵为

000\\* MERGEFORMAT (-)

* + 1. 等效轴坐标表示法

坐标系{B}的描述：

首先将坐标系{B}与一个已知的参考坐标系{A}重合，将{B}绕矢量 按右手定则旋转角。

则{B}相对于{A}的姿态可用或表示，称作等效轴角坐标表示法。

* + 1. 欧拉参数

另一种姿态表示法通过四个数值来表示，称为欧拉参数法。根据等效旋转轴和等效旋转角，可以得到欧拉参数如下

000\\* MERGEFORMAT (-)

这四个参数不是相互独立的

000\\* MERGEFORMAT (-)

使用欧拉参数表示的旋转矩阵为

000\\* MERGEFORMAT (-)

已知一个矩阵，得到对应的欧拉参数为

000\\* MERGEFORMAT (-)

1. 000Equation Chapter (Next) Section 1操作臂的运行学
   1. 连杆的描述

机器人中每个连杆都可以用四个运行学参数来描述，其中两个描述连杆本身，另外两个用于描述连杆之间的连接关系。为了是问题简化，通常设定沿关节轴1的方向，并且当关节变量1为0时，设定关节参考坐标系{0}和坐标系{1}重合，按照这个规定总会有，。另外当关节1为转动关节时，；当 关节1位移动关节时， 。

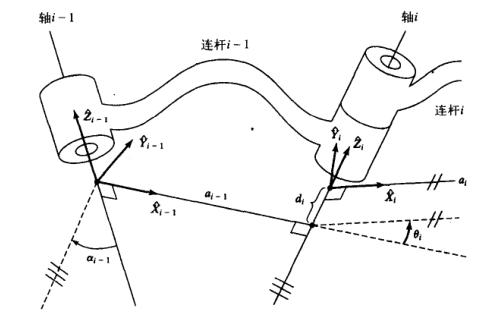
各个参数的定义

=沿轴，从轴移动到的距离；

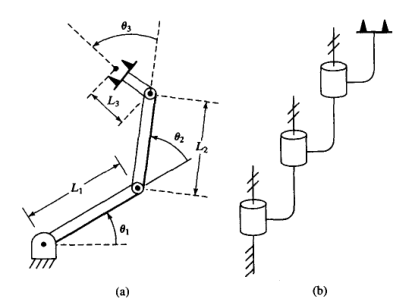
=沿轴，从轴旋转到的角度；

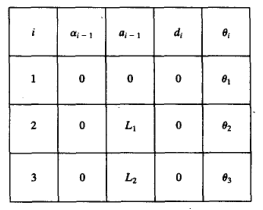
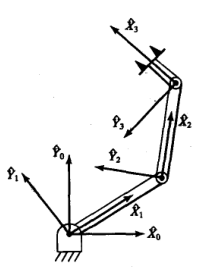
=沿轴，从移动到轴的距离；

=沿轴，从旋转到轴的角度。



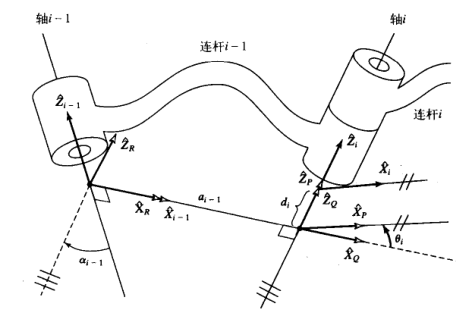
例子：3R机械手的连杆参数





* 1. 操作臂的运行学
     1. 连杆变换的推导

连杆两个关节轴用四个参数描述，那么每个参数可看成是坐标系的一次简单变换，从而推导出坐标系{i}相对于坐标系{i-1}的变换。



建立中间坐标系{R}，{Q}，{P}，把在坐标系{i}中定义的矢量变换为在坐标系{i-1}中的描述，对应的变换可以写成

000\\* MERGEFORMAT (-)

变换矩阵为

000\\* MERGEFORMAT (-)

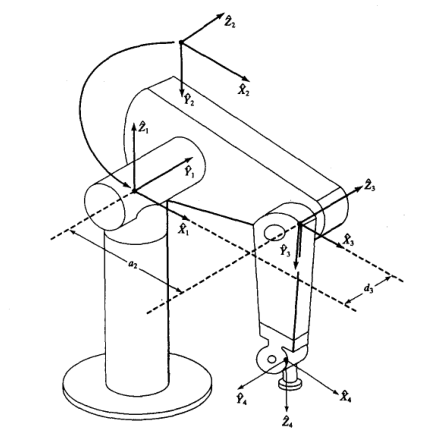
即

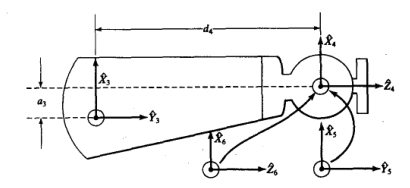
000\\* MERGEFORMAT (-)

计算得到\[{}\_{i-1}^{i}\mathbf{T}\]的一般表达式为

000\\* MERGEFORMAT (-)

* + 1. 例子：PUMA560连杆参数





|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| i |  |  |  |  |
| 1 | 0 | 0 | 0 |  |
| 2 | 0 | -90 | 0 |  |
| 3 |  | 0 |  |  |
| 4 |  | -90 |  |  |
| 5 | 0 | 90 | 0 |  |
| 6 | 0 | -90 | 0 |  |

1. 000Equation Chapter (Next) Section 1操作臂逆运动学

C Primer Plus--Stephen Prata

1. 初识C语言
2. C语言概述
3. 数据和C
4. 字符串和格式化输入/输出
5. 运算符、表达式和语句
6. C控制语句：循环
7. C控制语句：分支和跳转
8. 字符串输入/输出和输入验证
9. 函数
10. 数组和指针
11. 字符串和字符串函数
12. 存储类别、连接和内存管理
13. 文件输入/输出
14. 数据结构和其他数据形式
    1. 建立结构声明

结构声明（struct declaration）描述了一个结构的组织布局。有时把结构声明称为模板。

struct book {

char title[MAXTITLE];

char author[MAXAUTL];

float value;

};

* 1. 定义结构变量

struct book library;

struct book 的作用相当于一般声明的int或float，因此可以定义多个个struct book 类型的变量，或指向struct book的指针：

struct book doyle, panshin, \*ptbook;

* + 1. 初始化结构

初始化结构与初始化数组类似

structure book library={

"The Pious Pirate and the Devious Damel",

"Renee Vivote",

1.95

};

* + 1. 访问结构成员

使用结构运算符——点（）访问结构中的成员，如libaray.value，可以像任何float类型变量那样使用library.value。比&的优先级高，因此这个表达式和&（library.float）一样。

* + 1. 结构的初始化器

C99和C11为结构体提供了指定初始化器（designated initializer），可以按任意顺序初始化任意的某个或某些或全部结构体成员

struct book gitf = {.value = 25.99,

.author = " James Broadfool",

.title = " Rue for the Toad"};

* 1. 结构数组
     1. 声明结构体数组

声明结构数组和声明其他类型的数组类似：

struct book library[MAXBKS];

数组名library本身不是结构名，它是一个数组名，该数组中的每个元素都是struct book类型的结构变量。

* + 1. 标识结构数组的成员

注意下标的意义：

library //一个book结构的数组

library[2] //一个数组元素（这是一个结构变量名）， 该元素是book结构。

library[2].title //一个char数组（library[2]的title成员）

library[2].title[4] //数组中library[2]元素的title成员的一个 字符

* 1. 嵌套结构

在一个结构中包含另一个结构：

struct names { //第一个结构

char first [LEN];

char last [LEN];

};

struct guy { //第二个结构

struct names handle; //嵌套结构

char favfood [LEN];

char job [LEN];

float income;

};

嵌套结构成员的访问，

定义结构变量fellow

struct guy fellow；

访问嵌套结构的成员

fellow.handle.fist, fellow.handle.last

* 1. 指向结构的指针

使用指向结构的指针有四个好处

（1）就像数组的指针比数组本身更容易操控（如，排序问题）一样，指向结构的指针通常比结构本身更容易操作。

（2）在一些早期的C实现中，结构不能作为参数传递给函数，但是可以传递指向结构的指针。

（3）即使能传递一个结构，传递指针通常更有效率

（4）一些用于表示数据的结构中包含指向其他结构的指针。

* + 1. 声明和初始化结构指针

声明结构指针

struct guy \* him;

指针him可以指向任意现有的guy类型的结构。

若barney是一个guy类型的结构，可以这样指向它

him=&barney；

和数组不同的是，结构名并不是结构的地址，因此要加&。

若fellow是一个结构数组，则fellow [0]是一个结构，因此，可以这样指向它

him=&fellow [0];

* + 1. 用指针访问成员

如果him==&barney，那么barney.income == (\*him).income == him->income恒成立。注意，.运算符比\*运算符优先级高，因此必须加圆括号。

* 1. 向函数传递结构的信息

ANSI C允许把结构作为参数使用，但是对于复杂的结构，传递指向结构的指针效率会更高。

* + 1. 传递结构成员

只要结构成员是一个具有单个值的数据类型，便可把它看作为参数传递给接受给特定类型的函数。如把stan结构中的两个成员相加

sum(stan.bankfund,stan.savefund);

如果需要在被调函数中修改主调函数中成员的值，就要传递成员的地址，如

modify(&stan.bankfund);

* + 1. 传递结构的地址

函数声明

double sum(const struct funds \*);

调用

sum(&stan);

* + 1. 传递结构

函数声明

double sum(struct funds moolah);

调用

sum(stan);

* + 1. 其他结构特性

（1）把一个结构赋值给另一个结构

o\_data = n\_data;

（2）把一个结构初始化为另一个结构

struct names right\_field={"Ruthie","George"};

struct names captain = right\_field;

* + 1. 复合字面量和结构（C99）

C99复合字面量特性可以用于结构和数组。struck book类型的复合字面量：

(struct book) {"The Idiot", "Fyodor Dostoyvesky", 6.99}

* + 1. 伸缩型数组成员（C99）

声明一个伸缩型数组成员有如下规则：

（1）伸缩型数组成员必须是结构的最后一个成员；

（2）结构中必须至少有一个成员；

（3）伸缩数组的声明类似于普通数组，只是他的方括号中是空的。

struct flex

{

int count;

double average;

double scores []; //伸缩型数组

}

声明一个struct flex结构变量时，不能用scores做任何事，因为没有给这个数组预留存储空间。实际上，C99的意图不是让你声明struct flex类型的变量，而是希望你声明一个指向struct flex类型的指针，然后用malloc（）来分配足够的存储空间。

struct flex \* pf; //声明一个指针

//请求一个结构和一个数组分配存储空间

pf=malloc(sizeof (struct flex) + 5 \* sizeof(double));

带伸缩型数组成员的结构有一些特殊的处理要求。

第一，不能用结构结构进行赋值或拷贝。

第二，不要以按值方式把这种结构传递给结构。

第三，不要使用带伸缩型数组成员的结构作为数组成员或另一个结构的成员。

* + 1. 匿名结构（C11）
    2. 使用结构数组的函数

数组名就是该数组的地址。

* 1. 联合简介

联合（union）是一种数据类型。它能够在同一个内存空间中存储不同的数据类型。

RTEMS linux 下开发环境搭建

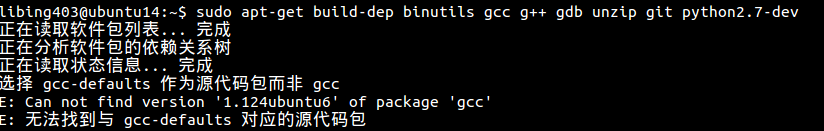
RTEMS官网推荐使用RSB搭建GNU交叉编译环境工具链。这里我以ubuntu14.04为例子，说明流程。

(1)通过RSB编译搭建交叉编译环境

首先安装RSB需要的软件包

$ sudo apt-get build-dep binutils gcc g++ gdb unzip git python2.7-dev

安装这个软件包的时候，我试过出现以下找不到某个软件包源码的错误，原因有可能是选择的软件更新服务器不包含该软件包，换个一个服务器地址可以解决，用主服务器是没有问题的。



设置目录等(注意以下命令需要在特定的目录运行，不要随便切换到其他目录)

$ cd

$ mkdir -p development/rtems/src

$ cd development/rtems/src

从github获取工具集的源码

$ git clone git://git.rtems.org/rtems-source-builder.git

$ cd rtems-source-builder

检验，检验设置是否正确，正确和不正确的结果如下

若设置不正确，显示结果可能如下

$ source-builder/sb-check

warning: exe: absolute exe found in path: (\_\_objcopy) /usr/local/bin/objcopy

warning: exe: absolute exe found in path: (\_\_objdump) /usr/local/bin/objdump

error: exe: not found: (\_xz) /usr/local/bin/xz

RTEMS Source Builder environment is not correctly set up

若设置正确，显示结果为

$ source-builder/sb-check

RTEMS Source Builder environment is ok

编译工具集，先进入RTEMS目录结构的正确位置

$ cd rtems

此时，所在的目录应该为

~/development/rtems/src/rtems-source-builder/rtems

运行命令检查可以用于编译的配置，

$ ../source-builder/sb-set-builder --list-bsets

若出现不允许读写/usr目录,加上sudo.

我的运行结果为

RTEMS Source Builder - Set Builder, 4.12 (08f709966f97 modified)

Examining: config

Examining: ../source-builder/config

Examining: ../bare/config

4.12/rtems-aarch64.bset

4.12/rtems-all.bset

4.12/rtems-arm.bset

4.12/rtems-autotools-base.bset

4.12/rtems-autotools-internal.bset

4.12/rtems-autotools.bset

4.12/rtems-bfin.bset

4.12/rtems-default.bset

4.12/rtems-epiphany.bset

4.12/rtems-i386.bset

4.12/rtems-lm32.bset

4.12/rtems-m32c.bset

4.12/rtems-m68k.bset

4.12/rtems-microblaze.bset

4.12/rtems-mips.bset

4.12/rtems-moxie.bset

4.12/rtems-nios2.bset

4.12/rtems-or1k.bset

4.12/rtems-powerpc.bset

4.12/rtems-sh.bset

4.12/rtems-sparc.bset

4.12/rtems-sparc64.bset

4.12/rtems-tools.bset

4.12/rtems-v850.bset

4.12/rtems-x86\_64.bset

databases/sqlite.bset

devel/autotools-base.bset

devel/autotools-internal.bset

devel/autotools.bset

devel/dtc.bset

devel/libtool.bset

devel/libusb.bset

devel/or1ksim.bset

devel/qemu.bset

gnu-tools-4.6.bset

gnu-tools-4.8.2.bset

graphics/freetype2.bset

graphics/graphics-all.bset

graphics/libjpeg.bset

graphics/libpng.bset

graphics/libtiff.bset

graphics/microwindows.bset

graphics/nxlib.bset

graphics/t1lib.bset

lang/gcc491.bset

net-mgmt/net-snmp.bset

net/lwip.bset

net/ntp.bset

net/protobuf.bset

rtems-base.bset

rtems-package.bset

rtems-urls.bset

rtems-version.bset

指定编译器安装路径，log文件，其中 4.12/rtems-arm是系统支持的编译配置文件.可以从上一步的显示的配置中选择一个，我这里选择为4.12/rtems-arm，每个配置文件有不同的配置，如4.12/rtems-all.bset 表示rtem4.12支持的所有处理器架构，4.12/rtems-all.bset 表示rtems4.12支持的arm处理器架构。

$ ../source-builder/sb-set-builder --log=l-arm.txt \

--prefix=$HOME/development/rtems/4.12　4.12/rtems-arm

编译需要比较长时间，我的笔记本电脑编译花了一个半小时。

(2)编译 rtems

$ cd ~/development/rtems/src

下载rtems源码

$ git clone git://git.rtems.org/rtems.git

$ export PATH=$HOME/development/rtems/4.12/bin:$PATH

$ cd rtems  
$ ./bootstrap

$ cd ..  
$ mkdir rtems-build  
$ cd rtems-build

其中 arm-rtems4.12 为工具链的版本，enable-rtemsbsp 选择对应的bsp参与编译  
其他为一些功能选项，根据需要打开  
../rtems/configure \  
--prefix=`pwd`/../rtems\_stm32f4xx \  
--target=arm-rtems4.12 \  
--disable-maintainer-mode \  
--disable-multiprocessing \  
--disable-posix \  
--disable-networking \  
--disable-cxx \  
--disable-tests \  
--disable-rtems-debug \  
--disable-multilib \  
--disable-paravirt \  
--disable-drvmgr \  
--disable-docs \  
--enable-rtemsbsp=stm32f4

make  
make install

编译完成.

编译后安装的文件在 ~/development/rtems/src/rtems\_stm32f4xx

(3)编译例子

下载  
git clone git://git.rtems.org/examples-v2.git examples-v2  
添加makefile路径变量  
export RTEMS\_MAKEFILE\_PATH=~/development/rtems/src/rtems\_stm32f4xx/arm-rtems4.12/stm32f4/  
  
  
cd examples-v2  
make

编译完成

Tip:为了以后启动终端不用重复设置环境变量,可以把它们添加到家目录下的配置文件.bashrc中,在里面加入两行.如果没设置好这两个环境变量,在编译例子的时候会出现错误.

export PATH=$HOME/development/rtems/4.12/bin:$PATH

export RTEMS\_MAKEFILE\_PATH=~/development/rtems/src/rtems\_stm32f4xx/arm-rtems4.12/stm32f4/

然后更新一下

$ source .bashrc

(3)RSB编译的bset文件的配置。

在目录~/development/rtems/src/rtems-source-builder/rtems/config/4.12下面有很多以.bset结尾的配置文件，这指定了所编译的工具集对应的处理器架构（architecture），当一次性编译多个不同架构的工具集时，后面编译RTEMS的命令和相关路径的设置会不一样。

例如采用rtems-all.bset配置文件编辑工具集

rtems-all.bset包含了以下17种架构的工具集配置，若执行下面命令

$ ../source-builder/sb-set-builder --log=l-all.txt \

--prefix=$HOME/development/rtems/4.12　4.12/rtems-all

则意味着会同时编译17种处理器架构的工具集，这会花费大约十几个小时的时间来编译完成（用笔记本亲测12小时左右），占用14GB左右的空间。当然你也可以从rtems-all.bset里删除掉一些你不需要的，只编译几个architecture的工具集。

若一次性编译多个不同架构的工具集，后面编译RTEMS的时候有两个地方需要做相应的变化。

（1）关于PATH环境变量的设置

$ export PATH=$HOME/development/rtems/4.12:$PATH

即去掉bin。

（2）make命令

原来只需简单执行

$make

编译多个工具集后，在make时需要再次指明target，即BSP对应的架构

$make RTEMS\_BSP="BSP1 BSP2 ..."

通常只编译一个BSP，因为前面的选项通常指定一个target，例如

--enable-rtemsbsp=stm32f4

对应的make命令是

$make RTEMS\_BSP=" stm32f4"

最后运行

$make install

编译成功。

小技巧：

我们用linux命令的时候通常会觉得要打很长的命令和参数选项，比较费时，对于经常用的命令可以用命令别名alias，提高效率。

命令别名功能  
在管理和维护Linux系统的过程中，将会使用到大量命令，有一些很长的命令或用法经常被用到，重复而频繁地输入某个很长命令或用法是不可取的。这时可以使用命令别名功能将这个过程简单化。更具体的可以参考http://blog.csdn.net/wangjunjun2008/article/details/20618023

http://blog.csdn.net/zyz511919766/article/details/22664745

例子：例如你经常要去目录修改源码~/development/rtems/src，可以定义别名

alias cd=''cd-src"

而且命令别名也支持table键补齐功能。

例子：例如你要在编译不同的BSP的rtems程序，要改变RTEMS\_MAKEFILE\_PATH环境变量，使用不同的工具集，那可以定义几个命令别名

alias set-arm='export RTEMS\_MAKEFILE\_PATH=~/development/rtems/src/rtems\_stm32f4xx/arm-rtems4.12/stm32f4/'

alias set-386=' export RTEMS\_MAKEFILE\_PATH=~/development/rtems/src/rtems\_i386/i386-rtems4.12/pc386/'

只需运行

$set-arm

$set-i386

就可以切换不同BSP的工具集了。

举一反三，其他很多经常用的命令可以这样做。

'

搭建QEMU虚拟仿真平台

Ubuntu14.04安装QEMU

(其他平台参照官网http://www.qemu-project.org/download/)

$ sudo apt-get install qemu