**第一讲：PX4二次开发基础**

飞控二次开发所需的知识体系

基础知识：模拟电子基础，数字电子基础，C/C++，C#，单片机(STM32系列)，嵌入式linux, ubuntu操作系统的使用。

高阶知识：经典控制理论，PID算法，卡尔曼滤波，图像处理，图像传输，matlab的使用。

1：C语言网上很多推荐课程：我们推荐如下

<http://study.163.com/course/courseMain.htm?courseId=1003425004>

网易云课堂的零基础学编程系列之C语言教程

2：模拟电子，数字电子：那就推荐华成英的系视频教程。

3: 单片机，推荐直接买一套单片机开发板回来学习，学习单片机的同时可以学会C语言，（课程中说入门可能需要2~3月，时间成本太高）

<https://item.taobao.com/item.htm?spm=a1z10.1-c.w4004-761023111.2.YuInBP&id=13951573558>

4：C# 地面站开发需要，在会C或者Java的情况下大概需要2月左右

5：基本的控制理论和方法，团队里只有袁有相关理论基础，课程中推荐要多在单片机开发中实践

6：Ubuntu和matlab等工具的掌握。嵌入式LINUX也有必要掌握，实际上很多飞控系统就实际基于嵌入式LINUX之上，飞控的操作系统NUTTX也是一种类LINUX系统。可以从树莓派来入手嵌入式Linux的开发。

7：TCP/IP网络通信。

可以入手开发无人机或者机器人，一整套课程体系学习需要3年时间左右，要大量的实践操作和理论相结合。

学习路径建议： 飞控应用开发》地面站开发》控制算法开发》视觉算法开发。

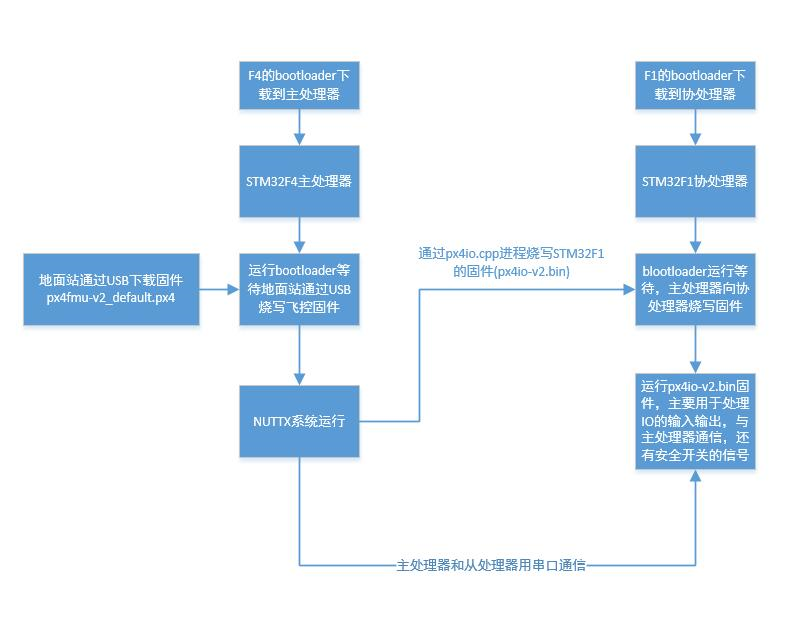
**第四讲：启动脚本**

Px4相比单片机，多了Nuttx操作系统，多个进程平行运行，使用uORB进行进程间通信

主处理器（PX4FMU）+协处理器（PX4IO）：都有bootloader，有各自的固件，两者间通过串口通行

主处理器PX4FMU ：各种传感器数据读取、姿态解算、PWM控制量的计算、与PX4IO通信。负责飞控最主要的工作

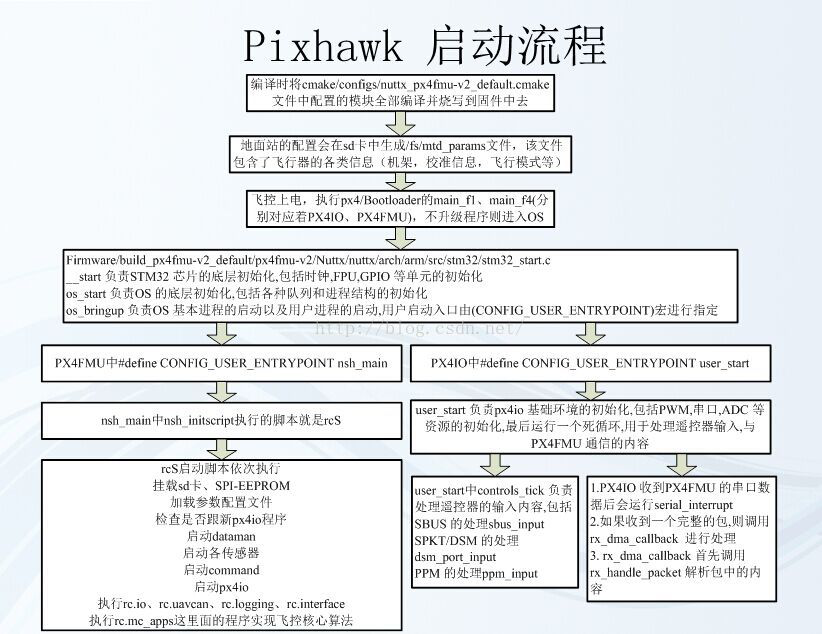
协处理器PX4IO：PIXHAWK 中专用于处理输入输出的部分,输入为支持的各类遥控器(PPM,SPKT/DSM,SBUS)信号, 输出为电调的PWM 驱动信号, 它与PX4FMU通过串口进行通信。



上电运行blootloader以后，主处理器会等待下载飞控固件（所以下载固件时要插拔电源），用地面站通过USB向飞控固件下载完以后，主处理器的飞控固件启动，这时候通过串口给协处理器下载固件(固件只是下载一次)。这个工作完成之后，主处理器和协处理器同时开始工作。

Bootloader：是嵌入式系统在加电后执行的第一段代码，在它完成CPU和相关硬件的初始化之后，再将操作系统映像或固化的嵌入式应用程序装在到内存中然后跳转到操作系统所在的空间，启动操作系统运行。飞控板出厂前已经将相应的bootloader烧写进飞控的Flash中的特定区域，bootloader作为一个单独的工程存在，飞控烧写固件不会修改bootloader，在定制开发时可对其修改，一般的二次开发不涉及修改bootloader

px4/Bootloader中可以看到main\_f1、main\_f4，分别对应着PX4IO、PX4FMU的Bootloader



Pixhawk的启动方式：

飞控上电后，Bootloader启动执行到最后，会用汇编语句，自动跳转到FLASH中固件下载的位置，开始执行固件代码。

PX4FMU的系统启动函数为nash\_main(int argc,char argv[])，通过执行启动脚本rcs来启动想要的飞控程序进程，例如可以可以通过运行不同的启动脚本来达到不同的飞行控制（例如，多旋翼、固定翼），在二次开发时主要是对启动脚本进行更改（比如添加模块等）

PX4IO 的系统启动函数为 user\_start(int argc,char argv[]).

**第五讲：系统软件框架及代码编译**



**这是原生固件的目录视图**

**总目录结构**

Src：目录是源码目录存放所有的源码，源码的查看都应该在这里。

Mavlink：是MAvlink的库目录，源码要调用这个库，如果我们要修改和添加MAVLINLK消息ID也是在这个目录下面。

NuttX：是NuttX的系统库，如果做二次开发，很少用到这个库。

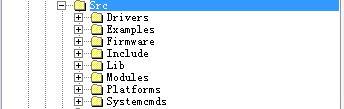
Cmake：是存放的Cmake编辑脚本文件夹，其中Cmake/Configs是存放的不同硬件的编译脚本，**nuttx\_mindpx-v2\_default**是PIXHAWK这个硬件的编辑脚本，如果要添加驱动，添加系统功能的模块函数，要修改里面的编译选项，保证添加的模块文件，能够被编译到。

Romfs：是文件系统文件夹。里面存放的飞控系统的启动脚本，我们要修改启动脚本主要在这个文件夹里面。src/Romfs/px4fmu\_common/init.d ,如rcS主要的启动脚本，还有其他的rc.XXX的脚本，传感器脚本，参数脚本等等，反正负责系统初始化的，如果自己定写了个功能模块要随系统启动那么就在这里添加启动。和linux系统的启动脚本非常相似。

Msg：是存放UORB消息主题的地方，如果要二次开发添加消息主题，就在这个文件夹里面添加修改。

**子目录结构**

Src这个文件夹比较重要，详细看看：

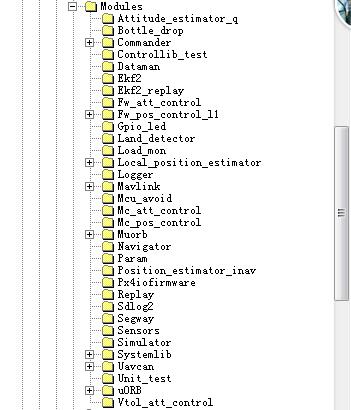


其中**Drivers**是驱动文件夹，包括了所有的传感器驱动，三轴，地磁，加速度，超声波....

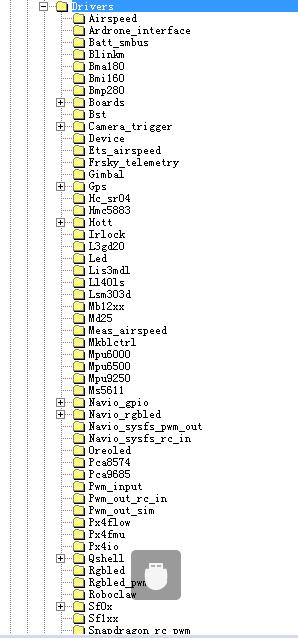
**Examples**是实例文件夹,入门PIXHAEK开发的一些很好的例子就在里面，其中最经典的是 px4\_simple\_app这个例子，要搞明白进程间通信UORB就是这个例程。

**Modules**是功能模块文件夹，什么位置估计，位置控制，指令控制....都在这个文件夹，也是二次开发主要要修改的文件夹。

**Systemcmds**是系统指令文件夹，都是飞控的支持的命令的源码。比如top命令，reboot命令等等，这些命令都可以在终端操作。



这是modules的文件夹下面的东西，很清楚，姿态估计，位置估计，位置控制，导航，UORB源码等等相当于飞控代码的核心。



这是Drivers文件夹里面的内容，可以看到所有的传感器驱动都在这里面，气压计，GPS，加速度，地磁，光流等等。

整个工程框架还是非常清楚明了的，结构清楚，对于开发也很有利。**主要就是修改moudlse,divers文件里面的内容。**

Ubuntu虚拟机的开发环境和下载地址：

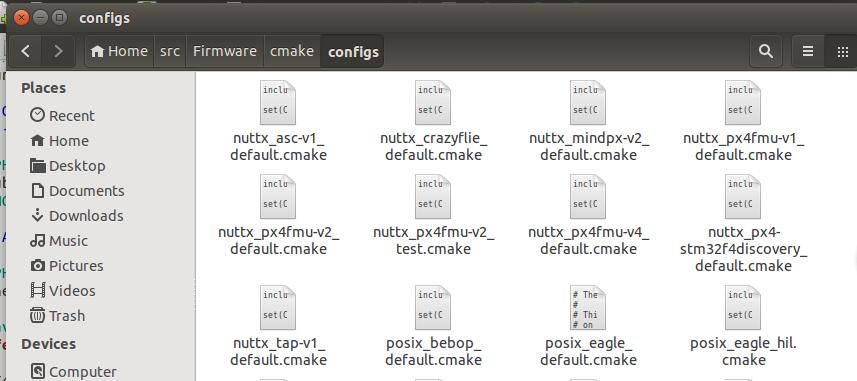
链接：http://pan.baidu.com/s/1gfJzNKv 密码：o9rx

**第六讲：PX4编译脚本与调试**

**1编译脚本和启动脚本的修改**

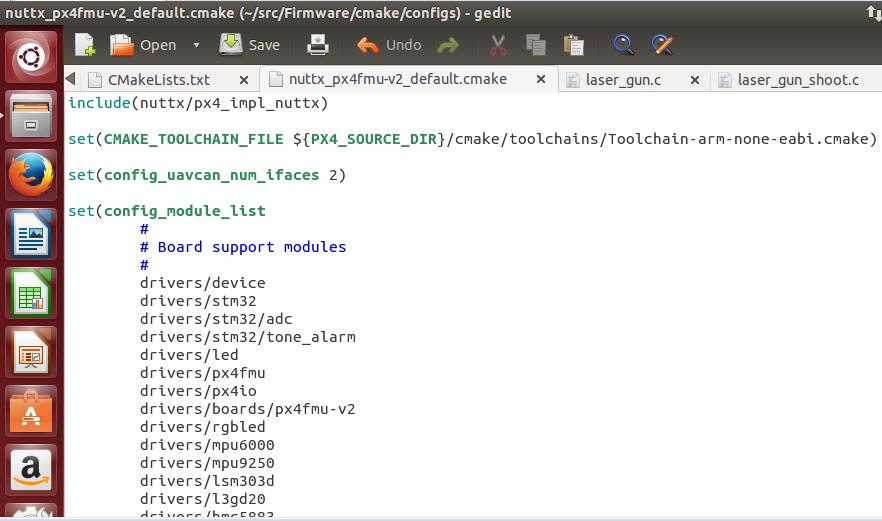
这个PIXHAWK原生固件采用的Cmake编译脚本的书写方式，降低makefile的书写难度。我们只要掌握几个修改脚本就可以掌握系统的编译。

**1 总的模块编译脚本的修改**



其中src/Firmware/cmake/configs是飞控硬件选择文件夹，比如有px4fmu-v2,px4fmu-v1的不同硬件可以选择。这个文件就是配置相应的飞控硬件，要编译什么模块，填写好正确的模块路径，系统就会编译里面的源码。

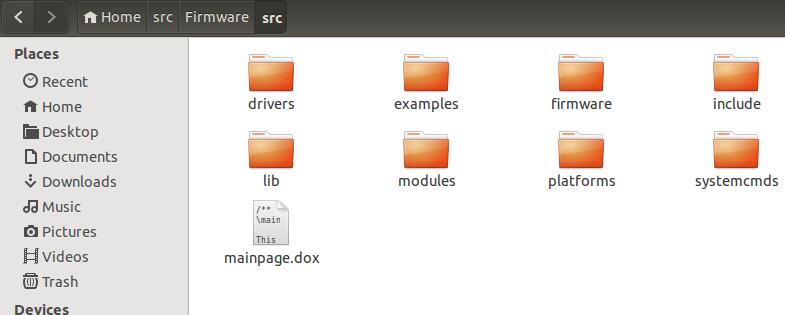
比如下图:



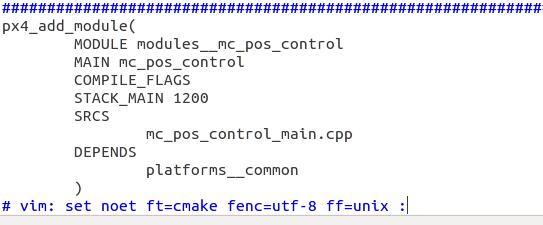
在set(config\_module\_list 中添加要编译的子模块。

**2子模块的CmakeList.txt的修改**

前面配置好的总的编译模块添加了，编译路径之后，实际上编译器是去寻找子模块下面的CmakeList.txt去进一步编译



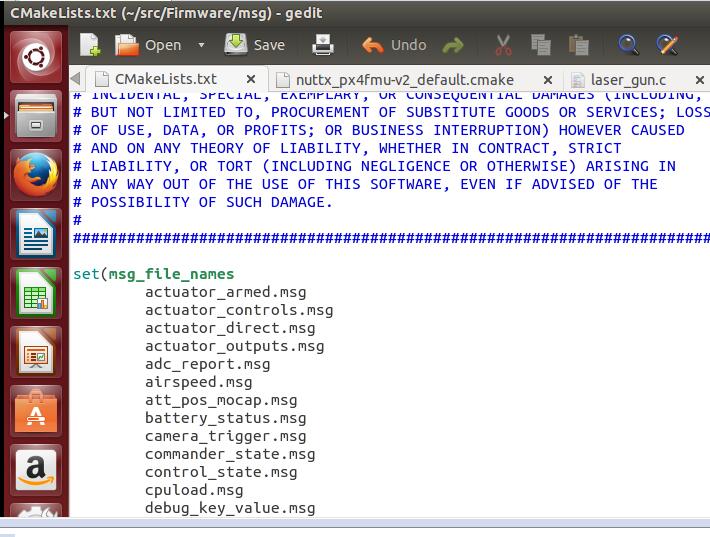
这个src目录就是源码，里面有所有的子目录都有CmakeList.txt编译脚本文件，我们一般在modules里面添加我们自定义的代码，drivers是传感器驱动目录，lib是库目录，systemcmds是系统命令。



按照如上添加修改CmakeList.txt即可。

**3 msg消息主题的添加**

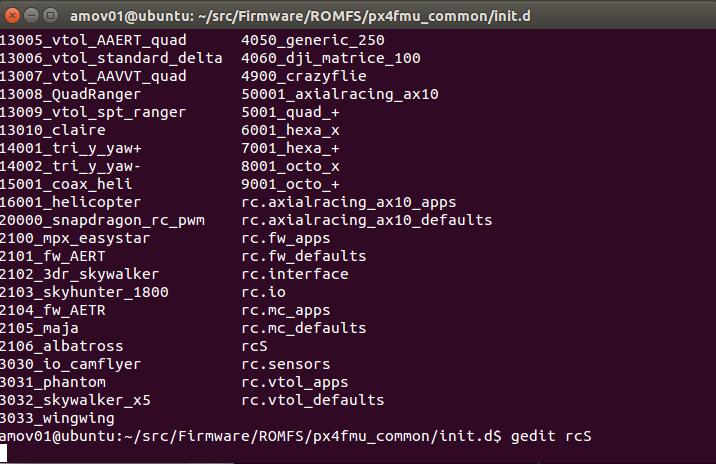
在源码目录msg目录添加，自定义的msg消息成员变量。在msg同目录的修改CmakeList.txt



把自定义的msg消息添加进去即可。这时候编译make px4fmu-v2\_default后，在

/src/Firmware/build\_px4fmu-v2\_default/src/modules/uORB/topics 这个cmake的bulid目录自动产生相应的自定义msg消息.h头文件。这是后在自己的应用代码中可以调用这个头文件，使用自定义的消息了。

**4 启动脚本修改**



以rc为开头的都是飞控的启动脚本的文件，通过这些文件决定那些应用程序应该执行，那些不执行，pixhawk支持多种类型的飞行器，四轴，固定翼，直升机等等。就是通过这些rc启动脚本指定。比如如果是rc.fw\_defaults是固定翼飞机的脚本，如果飞机是多旋翼这个rc.fw\_defaults是不会执行的，而是执行rc.mc\_defaults这个多旋翼脚本。如果我们的飞机是多旋翼，我们一般在修改rc.mc\_apps了。把相应的应用程序在这个脚本中 XXX start即可。

**第七讲：uORB消息订阅发布机制**

uORB(Micro Object Request Broker,微对象请求代理器)是PX4/Pixhawk系统中非常重要且关键的一个模块，它肩负了整个系统的数据传输任务，所有的传感器数据、GPS、PPM信号等都要从芯片获取后通过uORB进行传输到各个模块进行计算处理。实际上uORB是一套跨「进程」 的IPC通讯模块。

**1 进程**

进程：进程这个概念是针对系统而不是针对用户的，对用户来说，他面对的概念是程序。当用户敲入命令执行一个程序的时候，对系统而言，它将启动一个进程。但和程序不同的是，在这个进程中，系统可能需要再启动一个或多个进程来完成独立的多个任务。多进程编程的主要内容包括进程控制和进程间通信。

**1.1 Linux下进程的结构**

Linux下一个进程在内存里有三部分的数据，就是"代码段"、"堆栈段"和"数据段"。

1. "代码段"，顾名思义，就是存放了程序代码的数据，假如机器中有数个进程运行相同的一个程序，那么它们就可以使用相同的代码段。
2. "堆栈段"存放的就是子程序的返回地址、子程序的参数以及程序的局部变量。
3. 数据段存放程序的全局变量，常数以及动态数据分配的数据空间。

**1.2 Linux下的进程控制**

在传统的Unix环境下，有两个基本的操作用于创建和修改进程：函数fork( )用来创建一个新的进程，该进程几乎是当前进程的一个完全拷贝；函数族exec( )用来启动另外的进程以取代当前运行的进程。

**1.3 进程间通信**

Linux作为一种新兴的操作系统，几乎支持所有的Unix下常用的进程间通信方法：管道、消息队列、共享内存、信号量、套接口等等。

1. 管道是进程间通信中最古老的方式，它包括无名管道和有名管道两种，前者用于父进程和子进程间的通信，后者用于运行于同一台机器上的任意两个进程间的通信。
2. 消息队列用于运行于同一台机器上的进程间通信，它和管道很相似，事实上，它是一种正逐渐被淘汰的通信方式。
3. 共享内存是运行在同一台机器上的进程间通信最快的方式，因为数据不需要在不同的进程间复制。通常由一个进程创建一块共享内存区，其余进程对这块内存区进行读写。
4. 信号量又称为信号灯，它是用来协调不同进程间的数据对象的，而最主要的应用是共享内存方式的进程间通信。
5. 套接口（socket）编程是实现Linux系统和其他大多数操作系统中进程间通信的主要方式之一。我们熟知的WWW服务、FTP服务、TELNET服务等都是基于套接口编程来实现的。

**1.4 Linux的进程和Win32的进程/线程比较**

　　在UNIX里，只有进程的概念，但在WIN32里却还有"线程"的概念。

WIN32里的进程/线程是继承自OS/2的。在WIN32里，"进程"是指一个程序，而"线程"是一个"进程"里的一个执行"线索"。从核心上讲，WIN32的多进程与Linux并无多大的区别，在WIN32里的线程才相当于Linux的进程，是一个实际正在执行的代码。但是，WIN32里同一个进程里各个线程之间是共享数据段的。这才是与Linux的进程最大的不同。

**2 uORB使用**

**2.1 uORB常用函数功能解析**

**int poll(struct pollfd fds[], nfds\_t nfds, int timeout)**

功能：**监控文件描述符（多个）；**

说明：timemout=0,poll()函数立即返回而不阻塞；timeout=INFTIM(-1),poll()会一直阻塞下去，直到检测到return > 0；

参数：

fds:struct pollfd结构类型的数组；

nfds:用于标记数组fds中的结构体元素的总数量；

timeout:是poll函数调用阻塞的时间，单位：毫秒；

返回值：

>0：数组fds中准备好读、写或出错状态的那些socket描述符的总数量；

==0:poll()函数会阻塞timeout所指定的毫秒时间长度之后返回;

-1:poll函数调用失败；同时会自动设置全局变量errno；

**int orb\_subscribe(const struct orb\_metadata \*meta)**

功能：**订阅主题（topic）;**

说明：即使订阅的主题没有被公告，但是也能订阅成功；但是在这种情况下，却得不到数据，直到主题被公告；

参数：

meta:uORB元对象，可以认为是主题id，一般是通过ORB\_ID(主题名)来赋值；

返回值：

错误则返回ERROR;成功则返回一个可以读取数据、更新话题的句柄；如果待订阅的主题没有定义或声明则会返回-1，然后会将errno赋值为ENOENT;

eg:

int fd = orb\_subscribe(ORB\_ID(topicName));

**int orb\_copy(const struct orb\_metadata \*meta, int handle, void \*buffer)**

功能：**从订阅的主题中获取数据并将数据保存到buffer中；**

参数：

meta:uORB元对象，可以认为是主题id，一般是通过ORB\_ID(主题名)来赋值;

handle:订阅主题返回的句柄；

buffer:从主题中获取的数据；

返回值：

返回OK表示获取数据成功，错误返回ERROR;否则则有根据的去设置errno;

eg:

struct sensor\_combined\_s raw;

orb\_copy(ORB\_ID(sensor\_combined), sensor\_sub\_fd, &raw);

**orb\_advert\_t orb\_advertise(const struct orb\_metadata \*meta, const void \*data)**

功能：**公告发布者的主题；**

说明：在发布主题之前是必须的；否则订阅者虽然能订阅，但是得不到数据；

参数：

meta:uORB元对象，可以认为是主题id，一般是通过ORB\_ID(主题名)来赋值;

data:指向一个已被初始化，发布者要发布的数据存储变量的指针；

返回值：错误则返回ERROR;成功则返回一个可以发布主题的句柄；如果待发布的主题没有定义或声明则会返回-1，然后会将errno赋值为ENOENT;

eg:

struct vehicle\_attitude\_s att;

memset(&att, 0, sizeof(att));

int att\_pub\_fd = orb\_advertise(ORB\_ID(vehicle\_attitude), &att);

**int orb\_publish(const struct orb\_metadata \*meta, orb\_advert\_t handle, const void \*data)**

功能：**发布新数据到主题；**

参数：

meta:uORB元对象，可以认为是主题id，一般是通过ORB\_ID(主题名)来赋值;

handle:orb\_advertise函数返回的句柄；

data:指向待发布数据的指针；

返回值：OK表示成功；错误返回ERROR；否则则有根据的去设置errno;

eg:

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

**int orb\_set\_interval(int handle, unsigned interval)**

功能：**设置订阅的最小时间间隔；**

说明：如果设置了，则在这间隔内发布的数据将订阅不到；需要注意的是，设置后，第一次的数据订阅还是由起初设置的频率来获取，

参数：

handle:orb\_subscribe函数返回的句柄；

interval:间隔时间，单位ms;

返回值：OK表示成功；错误返回ERROR；否则则有根据的去设置errno;

eg:

orb\_set\_interval(sensor\_sub\_fd, 1000);

**orb\_advert\_t orb\_advertise\_multi(const struct orb\_metadata \*meta, const void \*data, int \*instance, int priority)**

功能：**设备/驱动器的多个实例实现公告，利用此函数可以注册多个类似的驱动程序；**

说明：例如在飞行器中有多个相同的传感器，那他们的数据类型则类似，不必要注册几个不同的话题；

参数：

meta:uORB元对象，可以认为是主题id，一般是通过ORB\_ID(主题名)来赋值;

data:指向一个已被初始化，发布者要发布的数据存储变量的指针；

instance:整型指针，指向实例的ID（从0开始）；

priority:实例的优先级。如果用户订阅多个实例，优先级的设定可以使用户使用优先级高的最优数据源；

返回值：

错误则返回ERROR;成功则返回一个可以发布主题的句柄；如果待发布的主题没有定义或声明则会返回-1，然后会将errno赋值为ENOENT;

eg:

struct orb\_test t;

t.val = 0;

int instance0;

orb\_advert\_t pfd0 = orb\_advertise\_multi(ORB\_ID(orb\_multitest), &t, &instance0, ORB\_PRIO\_MAX);

**int orb\_subscribe\_multi(const struct orb\_metadata \*meta, unsigned instance)**

功能：**订阅主题（topic）;**

说明：通过实例的ID索引来确定是主题的哪个实例；

参数：

meta:uORB元对象，可以认为是主题id，一般是通过ORB\_ID(主题名)来赋值;

instance:主题实例ID;实例ID=0与orb\_subscribe()实现相同；

返回值：

错误则返回ERROR;成功则返回一个可以读取数据、更新话题的句柄；如果待订阅的主题没有定义或声明则会返回-1，然后会将errno赋值为ENOENT;

eg:

int sfd1 = orb\_subscribe\_multi(ORB\_ID(orb\_multitest), 1);

**int orb\_unsubscribe(int handle)**

功能：**取消订阅主题；**

参数：

handle:主题句柄；

返回值：

OK表示成功；错误返回ERROR;否则则有根据的去设置errno;

eg:

ret = orb\_unsubscribe(handle);

**int orb\_check(int handle, bool \*updated)**

功能：**订阅者可以用来检查一个主题在发布者上一次更新数据后，有没有订阅者调用过ob\_copy来接收、处理过；**

说明：如果主题在在被公告前就有人订阅，那么这个API将返回“not-updated”直到主题被公告。可以不用poll，只用这个函数实现数据的获取。

参数：

handle:主题句柄；

updated:如果当最后一次更新的数据被获取了，检测到并设置updated为ture;

返回值：

OK表示检测成功；错误返回ERROR;否则则有根据的去设置errno;

eg:

if (PX4\_OK != orb\_check(sfd, &updated)) {

return printf("check(1) failed");

}

if (updated) {

return printf("spurious updated flag");

}

//or

bool updated;

struct random\_integer\_data rd;

/\* check to see whether the topic has updated since the last time we read it \*/

orb\_check(topic\_handle, &updated);

if (updated) {

/\* make a local copy of the updated data structure \*/

orb\_copy(ORB\_ID(random\_integer), topic\_handle, &rd);

printf("Random integer is now %d\n", rd.r);

}

**int orb\_stat(int handle, uint64\_t \*time)**

功能：**订阅者可以用来检查一个主题最后的发布时间；**

参数：

handle:主题句柄；

time:存放主题最后发布的时间；0表示该主题没有发布或公告；

返回值：

OK表示检测成功；错误返回ERROR;否则则有根据的去设置errno;

eg:

ret = orb\_stat(handle,time);

**int orb\_exists(const struct orb\_metadata \*meta, int instance)**

功能：**检测一个主题是否存在；**

参数：

meta:uORB元对象，可以认为是主题id，一般是通过ORB\_ID(主题名)来赋值;

instance:ORB 实例ID;

返回值：

OK表示检测成功；错误返回ERROR;否则则有根据的去设置errno;

eg:

ret = orb\_exists(ORB\_ID(vehicle\_attitude),0);

**int orb\_priority(int handle, int \*priority)**

功能：**获取主题优先级别；**

参数：

handle:主题句柄；

priority:存放获取的优先级别；

返回值：

OK表示检测成功；错误返回ERROR;否则则有根据的去设置errno;

eg:

ret = orb\_priority(handle,&priority);

**2.2 自定义uORB消息流程**

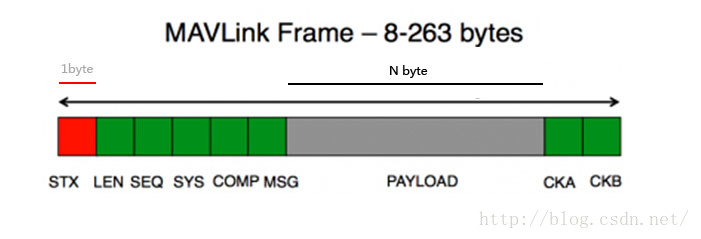
1. 在msg文件夹下添加一个具体的消息（.msg文件），在同级目录里面注册cmakelist（系统编译会自动生成这个消息主题相应的.h 头文件）
2. 新建自己的应用APP, 在同级目录下添加cmakelist.txt编译脚本
3. 修改总的编译脚本，具体的目录为: /src/Firmware/cmake/configs/nuttx\_px4fmu-v2\_default.cmake
4. 修改上电自启动脚本，参考启动脚本内容

**第八讲：Mavlink协议**

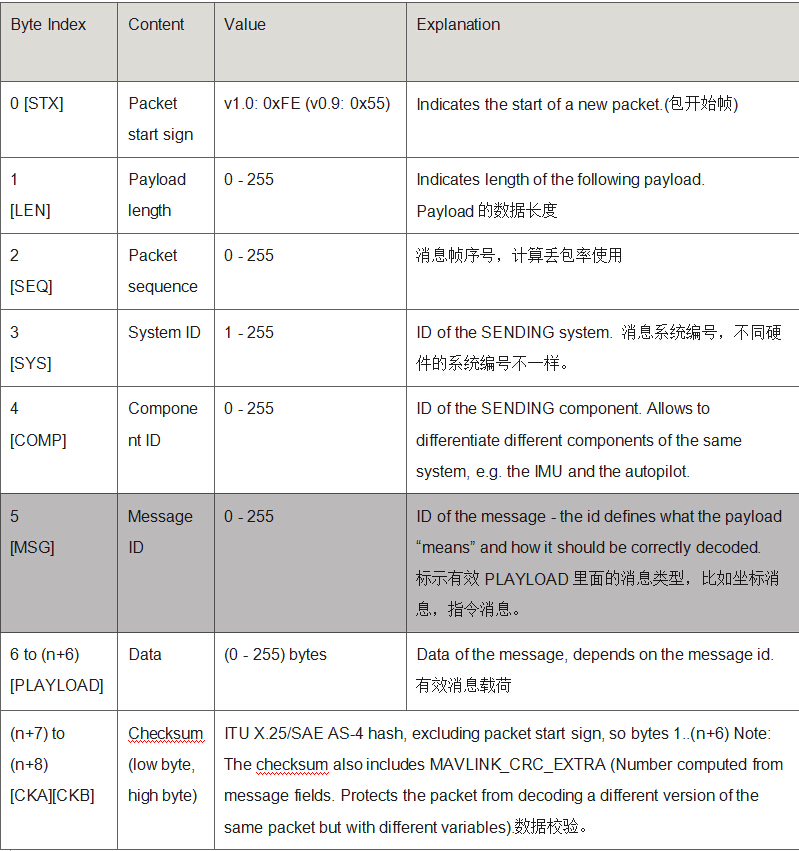
Mavlink协议最早由 苏黎世联邦理工学院 计算机视觉与几何实验组 的 Lorenz Meier于2009年发布，并遵循LGPL开源协议。Mavlink协议是在串口通讯基础上的一种更高层的开源通讯协议，主要应用在微型飞行器（micro aerial vehicle）的通讯上。Mavlink是为小型飞行器和地面站（或者其他飞行器）通讯时常常用到的那些数据制定一种发送和接收的规则并加入了校验（checksum）功能。

**1.1 mavlink消息结构分析：**

MAVLINK的传输基本单位是消息帧，一帧的数据长度8bytes到263bytes不等



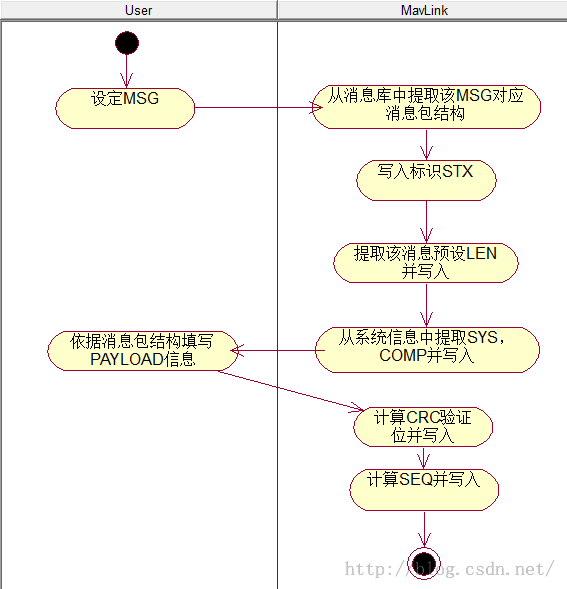
上图中除了灰色外，其他格子都代表一个字节的数据，灰色格子里面的数据长度是不固定的。下面是每个消息帧的详细解释：



**1.2消息包封装过程**

消息包内各区域信息存在关联，当使用MavLink协议提供的方法封装消息包时，会根据所使用的MSG获取到该类别MSG消息的LEN信息，同时软件（地面站或飞行控制软件）会根据自身状态信息填写SYS、COMP。信息填写完毕生成数据包时，封装方法会自动添加STX，并在上一次发送消息包所使用的SEQ上加1作为本次发送的SEQ写入，当SEQ超过255时，会回到0并重新开始计数。CKA、CKB会在PAYLOAD信息写入后、封装完成之前，根据CRC[Fe1] （CyclicRedundancy Check）循环冗余校验码算法计算得出并自动写入包内。

也就是说，设定SYS和COMP并且正确调用MavLink所提供方法后，整个消息包的生成过程中**仅有MSG和PAYLOAD两项内容需要用户关心**，消息包封装过程如活动图所示



**传感器性能及原理**

* 1. **X7PRO 内置IMU**

加速计 ADIS16470/ICM-20649/BMI088

陀螺仪 ADIS16470/ICM-20649/BMI088

电子罗盘 RM3100

气压计 MS5611\*2

**ADIS16470**

三轴数字陀螺仪，±2000°/秒动态范围

运动中偏置稳定度：8°/小时

rms速率噪声密度：0.008°/sec/√Hz

三轴数字加速度计动态范围：±40 gg

运动中偏置稳定度：13 μg

**三轴角速度和加速度输出**

校准温度范围：-10°C至+75°C

SPI兼容数据通信

可编程工作和控制

自动和手动偏置校正控制s

用于同步数据采集的数据就绪指示

外部同步模式：直接、脉冲、调整和输出

按需对惯性传感器自测

按需对闪存自测

单电源供电(VDD)：3.0 V至3.6 V

抗机械冲击能力：2000 g

工作温度范围：-25°C至+85°C

**ICM-20649**

3轴陀螺仪，可编程FSR为±500dps、±100dps、±2000dps和±4000dps

3轴加速度计，可编程FSR为±4g、±8g、±16g和±30g

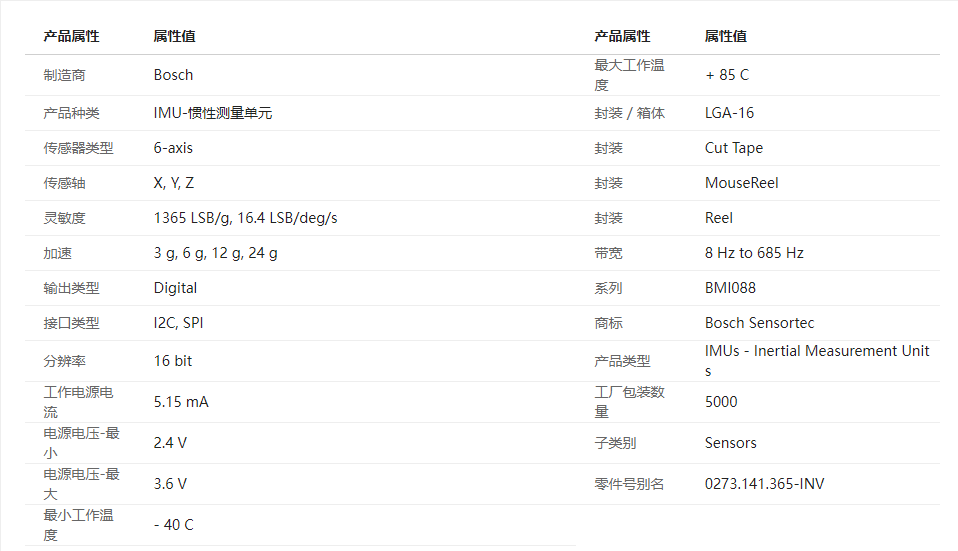
7MHz SPI或400kHz I2C主机接口

数字输出温度传感器

1.71V至3.6V VDD工作电压

10,000g耐冲击性

**BMI088**



**ms5611**

工作温度范围：-40℃～85℃

精确度：25℃,750mbar时-1.5～+1.5mbar

供电电源：1.8 V～ 3.6 V

特点：低功耗1μA，集成数字压力传感器，I²C和SPI接口，高达20 MHz，无需外部元件（内部振荡器），分辨率可达10CM

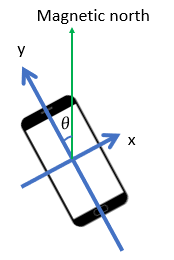
**1.2 IMU惯性测量单元测量原理**

惯性测量单元是测量物体三轴姿态角(或角速率)以及加速度的装置。一般的，一个IMU包含了**三个单轴的**[**加速度计**](https://baike.baidu.com/item/%E5%8A%A0%E9%80%9F%E5%BA%A6%E8%AE%A1/8652392)**和三个单轴的陀螺**，加速度计检测物体在载体坐标系统独立三轴的加速度信号，而陀螺检测载体相对于导航坐标系的角速度信号，测量物体在三维空间中的角速度和加速度，并以此解算出物体的姿态。

加速度计：

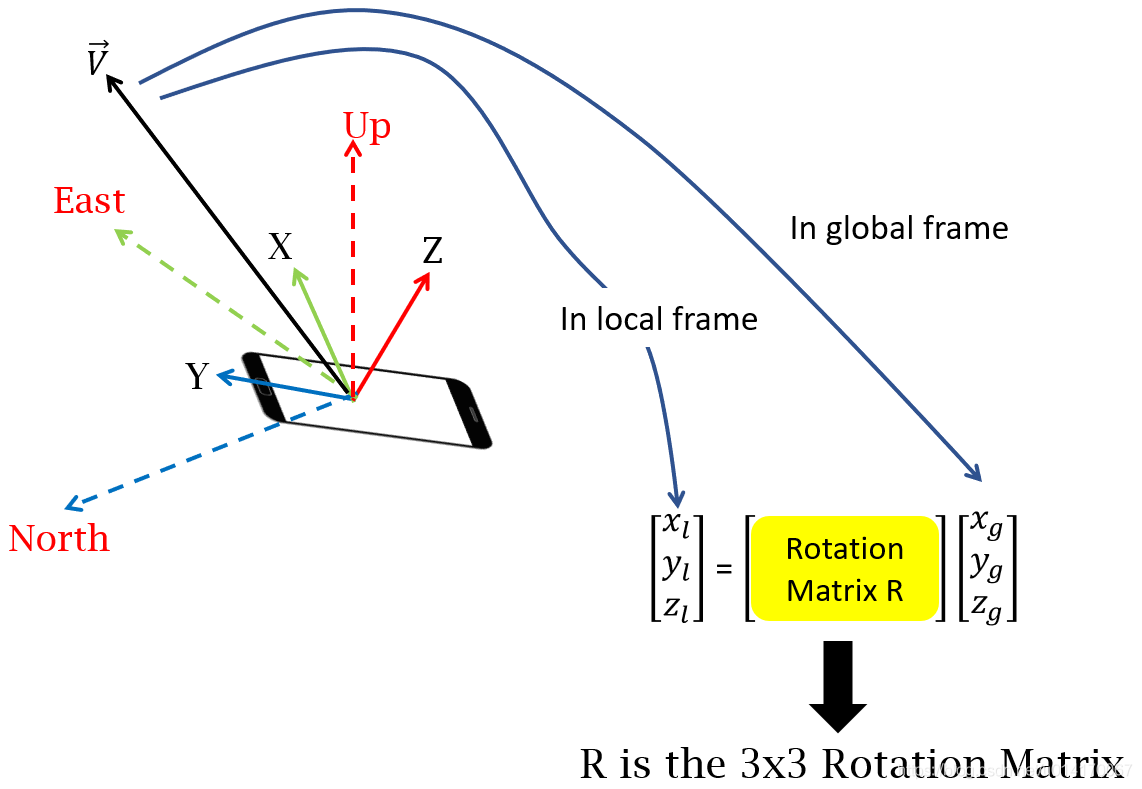
1. 测量重力加速度和直线加速度；
2. 2D情况下，**在静止状态下，**测量到的重力可以用于估算旋转角度；
3. 通过对加速度进行双重积分来估算距离，会导致误差严重积累；

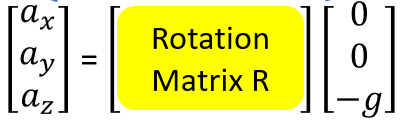
磁力计：

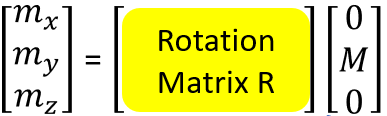
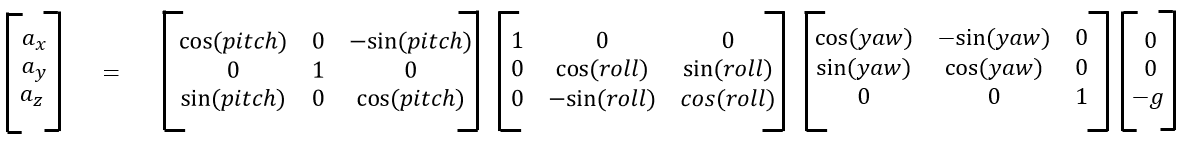
  
在3D情况下，3D磁力计的测量输出值取决于旋转角度。  
在2D情况下，磁力计可以用于估算旋转角度：  
在这里插入图片描述  
其中，Mx是磁力计x轴的测量输出值，M是磁场北极强度，是已知值。  
然而，在3D情况下，单靠磁力计不足以估算三维旋转角度，

磁力计：

如下图所示，在三维情况下，X-Y-Z表示磁力计本地坐标系，East-North-Up表示的是世界坐标系，三维旋转就是测量的世界坐标系和磁力计本地坐标系之间的旋转角度关系。也就是三维旋转矩阵的测量

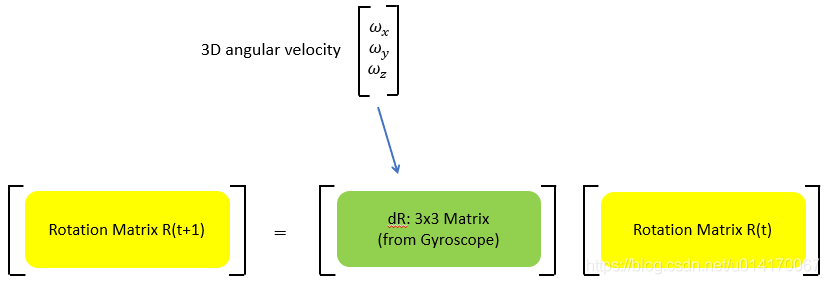


最关键的一点是要使用全局已知的参考向量，并且能在本地坐标系中测量到。有两个向量可以做到：重力和磁场北极。  
如果采用重力向量，那么根据加速度传感器可以得到如下公式：  
  
**重力测量的前置条件是飞机在静止状态，在空中如何测量？？？？**

如果采用磁场北极向量，那么根据磁力计可以得到如下公式：  
  
从上面的方程可以得出6个方程，旋转矩阵为1个正交矩阵，所以根据上面两组公式足以解算出旋转矩阵R，旋转矩阵公式如下：  
  
可由此解算出yaw, pitch, roll的值

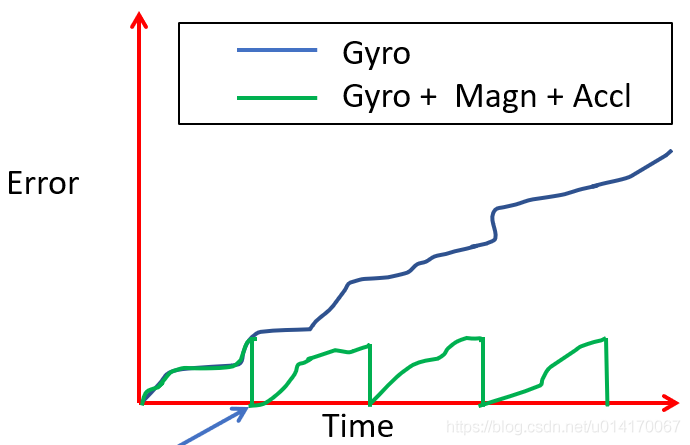
陀螺仪

陀螺仪是用来测量角速度的，通过对角速度进行积分就可以知道角度。



这种方式测量测量容易出现累计误差。

鉴于用加速度计只能在静态下测量旋转矩阵，磁力计容易受到磁场干扰，一般采用加速度计、磁力计和陀螺仪结合的方式，在静态以及没有磁场干扰的情况下，通过加速度计和磁力计将误差进行一次reset，就可以减少误差累计而导致测量结果漂移的情况，如下图：



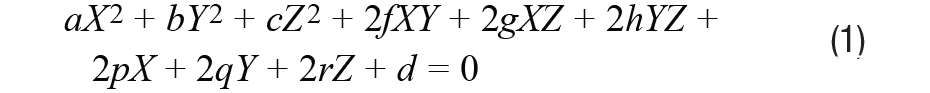
**罗盘传感器的校准**

在使用罗盘传感器之前，需要对其进行校准以消除两个主要误差。一个是**失调误差**，这原本是由传感器和电路的失调误差引起的。另一个是**标度误差**。这两种误差都容易受到周围磁环境的干扰。例如，如果有一个x轴向的外部磁场施加到传感器上，就会给出外部x轴失调误差。同时，x轴标度也将与y轴和z轴不同。

通常用于校准磁传感器的方法是在xy平面上转动传感器绕圈，然后抽取数据。一个地点的地磁场强度是一个常数值，因此绘制的数据应该是一个圆；然而，事实上，我们将看到一个椭圆形，这意味着我们需要移动椭圆并重新缩放到以零为中心的圆。

上述2D校准方法有一些缺点，并且需要用加速器来测量其倾斜度。我们使用3D球面拟合方法来校准罗盘传感器。首先，我们需要将传感器旋转到x-y-z空间中的每个方向，并在3D坐标中绘制其值。然后我们需要使用最小平方误差（MSE）方法将数据拟合为椭球面。

椭球方程可以表示为



其中，X、Y和Z是罗盘输出在三个方向上的地磁分量。将这些值拟合为椭球面意味着，我们需要得到一组最优系数解。我们将系数定义为：

基于IMU和地磁传感器的捷联惯性导航系统

在拟合时，我们定义向量：

基于IMU和地磁传感器的捷联惯性导航系统

所以我们需要计算最优σ，并使用公式2来找出最小值：

基于IMU和地磁传感器的捷联惯性导航系统

这样我们就可以得到图1所示的拟合结果。

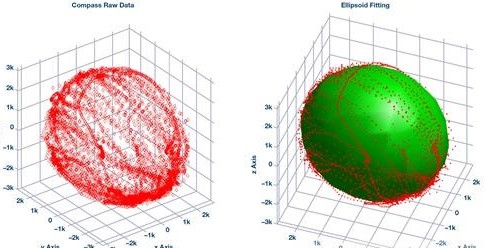


图1. 原始罗盘数据分布（左）和使用椭球拟合后的罗盘数据（右）。

为了校准传感器，我们需要拉伸或压缩拟合的椭球面并将其移至以零为中心的球面上。我们使用矩阵奇异值分解（SVD）方法来进行这种校准。校准后的球体如图2所示。1，2

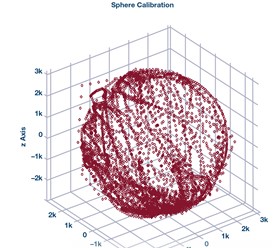


图2. 用SVD方法进行球体校准后的罗盘数据。

**其他开源飞控**

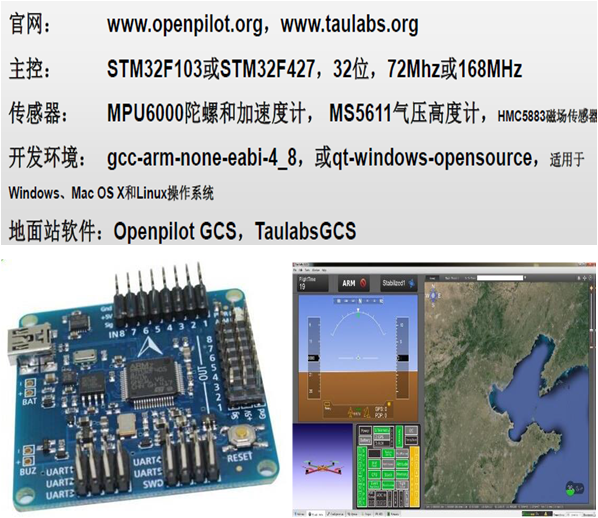
**Arduino飞控**



**APM飞控**



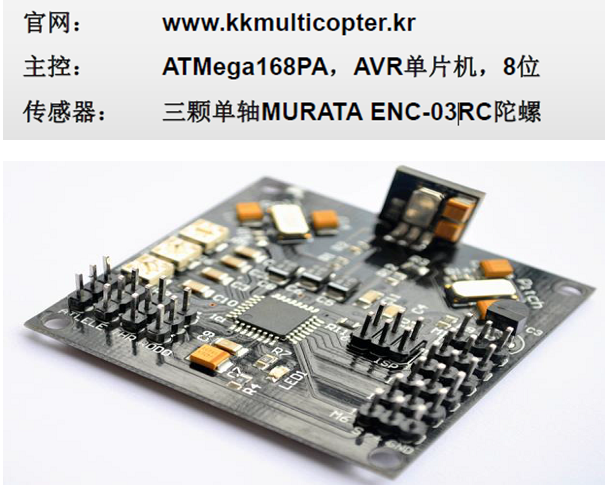
**Openpilot和Taulabs**



**Multiwiicopter**



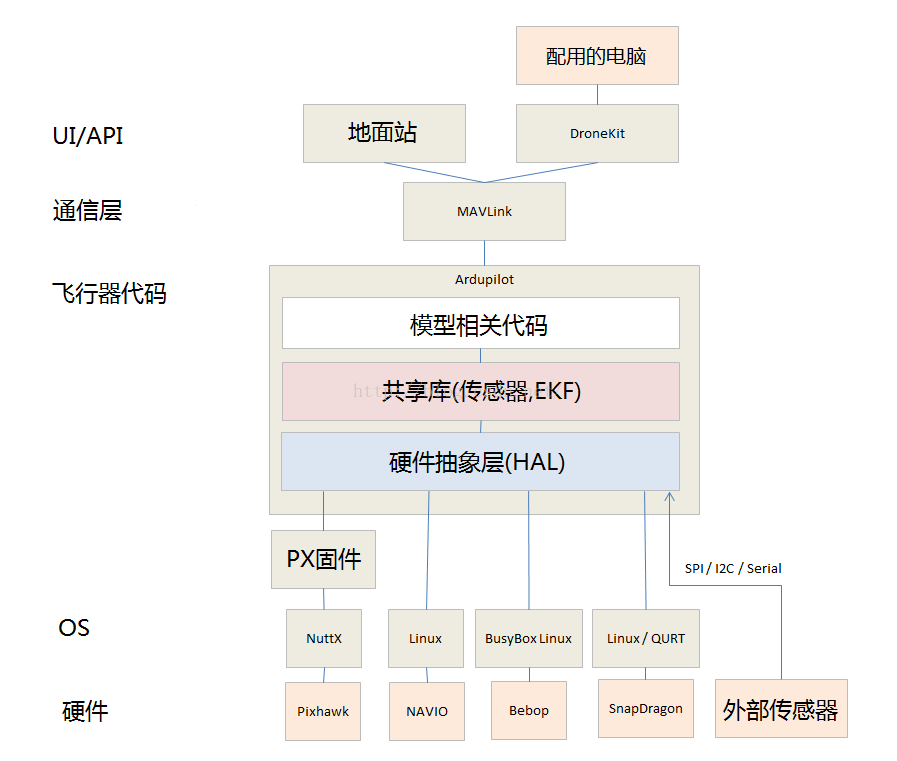
KK飞控



**Paparazzi**



**APM软件架构**



各模型目录

各模型目录是最高级目录，用于定义每一种模型的固件。

固定翼飞机 ArduPlane

旋翼飞机 ArduCopter

车辆 Rover

追踪雷达 AntennaTracker

水下机器人 ArduSub

硬件抽象层

硬件抽象层是ArduPilot能移植到各种不同平台的关键，其顶层抽象在libraries/AP\_HAL目录下，定义了其他代码所需要遵循的接口，每个板子各有一个AP\_HAL\_XXX这样的子目录。

库文件 \libraries

AP\_AHRS 姿态估计库

AP\_Common 核心库

AP\_Math

AC\_PID

AP\_InertialNav 惯性导航库

AC\_AttitudeControl 姿态控制库

AP\_WPNav 导航库

AR\_Motors

RC\_Channel

AP\_InertialSensor 陀螺仪

AP\_RangeFinder 声呐和红外距离传感器接口库

AP\_Baro 气压计接口库

AP\_GPS GPS接口库

AP\_Compass 三轴磁力计接口库

AP\_OpticalFlow 光流传感器接口库