# PX4源码结构分析

目录

[PX4源码结构分析 1](#_Toc495739571)

[第一章 PX4状态机分析 4](#_Toc495739572)

[第二章 软件架构与功能 40](#_Toc495739573)

[第三章 PX4姿态解算 41](#_Toc495739574)

[姿态解算1\_入门篇（DCM Nomalize） 41](#_Toc495739575)

[姿态解算2\_源码姿态解算算法分析 56](#_Toc495739576)

[姿态解算3\_源码姿态解算算法分析 86](#_Toc495739577)

[姿态解算4\_补充篇 109](#_Toc495739578)

[姿态解算 5\_ECF/EKF/GD介绍 116](#_Toc495739579)

[姿态解算 6\_Gradient Descent 127](#_Toc495739580)

[第四章 姿态控制 135](#_Toc495739581)

[第四章 位置控制 171](#_Toc495739582)

[整体架构 171](#_Toc495739583)

[Offboard位置控制 180](#_Toc495739584)

[第五章 源码模块关系图： 181](#_Toc495739585)

[第六章 源码流程图 183](#_Toc495739586)

[第七章 PX4源码启动分析 184](#_Toc495739587)

[第八章 PX4的sensors校准源码分析 205](#_Toc495739588)

[PX4源码分析8\_PX4的sensor校准 205](#_Toc495739589)

[sensor的校准过程分为两部分，首先需要先通过地面站进行校准设置，然后通过校准数据的更新来获取最新的校准数据。 205](#_Toc495739590)

[1.校准数据的设置： 205](#_Toc495739591)

[2.校准数据的获取： 206](#_Toc495739592)

[二.陀螺仪的校准： 206](#_Toc495739593)

[1.do\_gyro\_calibration()： 206](#_Toc495739594)

[2.初始化： 206](#_Toc495739595)

[3.gyro\_calibration\_worker()： 206](#_Toc495739596)

[4.校准： 206](#_Toc495739597)

[5.赋值： 207](#_Toc495739598)

[三.加速度计的校准： 207](#_Toc495739599)

[1.do\_accel\_calibration()： 207](#_Toc495739600)

[2.初始化： 207](#_Toc495739601)

[3.校准： 207](#_Toc495739602)

[4.计算offset： 208](#_Toc495739603)

[5.赋值： 208](#_Toc495739604)

[四.地磁的校准： 209](#_Toc495739605)

[1.do\_mag\_calibration()： 209](#_Toc495739606)

[2.校准： 209](#_Toc495739607)

[第八章 PX4源码总结 210](#_Toc495739608)

[第九章 安全决策机制 210](#_Toc495739609)

[第十章 PX4内部进程通信 210](#_Toc495739610)

[IPC性能 210](#_Toc495739611)

[Uorb原理 211](#_Toc495739612)

[节点句柄（Node Handle） 211](#_Toc495739613)

[数据结构 211](#_Toc495739614)

[发布（publication） 211](#_Toc495739615)

[订阅（Subscription） 212](#_Toc495739616)

[混合系统 212](#_Toc495739617)

[集成ROS 213](#_Toc495739618)

[第十一章 进程间通讯开发方法 214](#_Toc495739619)

[第十二章 构建后台应用程序 225](#_Toc495739620)

[第一步: 创建一个小的标准应用 225](#_Toc495739621)

[第二步: 创建后台进程管理函数 226](#_Toc495739622)

[第三步: 添加停止/状态指令以及安全保护 227](#_Toc495739623)

[第十三章 构建最小应用程序 231](#_Toc495739624)

[第一步: 准备源码文件 231](#_Toc495739625)

[第二步: 编写最小程序 232](#_Toc495739626)

[第三步: 在NuttShell中注册应用并build 232](#_Toc495739627)

[第四步:加载应用并测试 233](#_Toc495739628)

[第十五章 添加自己的传感器 235](#_Toc495739629)

[字符型设备 235](#_Toc495739630)

[传感器的硬件连接 236](#_Toc495739631)

[与SPI操作相关的函数 240](#_Toc495739632)

[spi.h 240](#_Toc495739633)

[stm32\_spi.c 241](#_Toc495739634)

[px4fmu\_spi.c 243](#_Toc495739635)

[px4fmu2\_init.c 245](#_Toc495739636)

[MPU6000驱动分析 247](#_Toc495739637)

[MPU6000类和MPU6000\_gyro类 247](#_Toc495739638)

[驱动注册过程 249](#_Toc495739639)

[MPU6000的启动过程 259](#_Toc495739640)

[数据处理过程 268](#_Toc495739641)

[数据测量 MPU6000::measure() 268](#_Toc495739642)

[read 268](#_Toc495739643)

[大小端处理 269](#_Toc495739644)

[交换XY轴 270](#_Toc495739645)

[自动测量MPU6000::start() 271](#_Toc495739646)

[寄存器检查 273](#_Toc495739647)

[为相片添加POS数据 277](#_Toc495739648)

[**为单反拍摄照片批量加入地理位置信息（POS点）和EXIF信息** 277](#_Toc495739649)

## ****第一章 PX4状态机分析****

### 简介

Commander文件夹是px4源码中的飞行控制命令切换模块（STABLE、ALTCTL、POSCTL、AUTO相互切换的过程）其次，所有模式能不能成功切换，都会在这个模块中做相应的飞行条件检查。

模式切换就是MODE\_SW （地面站选择模式切换开关，通常我们的遥控器只有3个档，因此需要3个以上的模式切换，则需要“改造”遥控器了——改装个多档位切换开关）：**但需要注意的是**——Mode\_Switch开关除了模式通道定义的模式外，还包含独立模式通道，比如空中停转（brake，return、等等）。

如此种种都是在Commander内完成的内容，最终能不能成功的切换就看各种模式需要的传感器满足条件，和切换间隙、出错问题的判断了。

### 状态机相关数据的来与去

ALTCTL STABLE MANUAL AUTO\_LOITER AUTO\_MISSION AUTO\_RETURN

首先，需要明确：commander.cpp、commander\_params.c是关于整个commander模块线程的任务调度的入口文件：

**int** **commander\_main**(**int** argc, **char** \*argv[]) commander模块线程能够接受的参数挺多

参数不能少于1个：

**如：**除了“start”“stop”命令参数外，剩下的参数命令可以跟多个参数命令

进入函数int commander\_thread\_main(int argc, char \*argv[])在一进入函数，首先是针对“函数参数”对重要“状态-环节”判断、初始化（标志初始化）：

大致骨架梳理：

while(!thread\_should\_exit)

 {

orb\_check(param\_changed\_sub,&updated);

orb\_check(sp\_man\_sub,&updated);

orb\_check(offboard\_control\_mode\_sub,&updated);...

orb\_check(sensor\_sub,&updated);

orb\_check(diff\_pres\_sub,&updated);

orb\_check(system\_power\_sub,&updated);

orb\_check(safety\_sub,&updated);

if (updated){

          bool previous\_safety\_off = safety.safety\_off;

          orb\_copy(ORB\_ID(safety), safety\_sub, &safety);

          /\* disarmif safety is now on and still armed \*/

if (status.hil\_state == vehicle\_status\_s::HIL\_STATE\_OFF &&safety.safety\_switch\_available && !safety.safety\_off&&armed.armed) {

arming\_state\_t new\_arming\_state = (status.arming\_state== vehicle\_status\_s::ARMING\_STATE\_ARMED ?

vehicle\_status\_s::ARMING\_STATE\_STANDBY : vehicle\_status\_s::ARMING\_STATE\_STANDBY\_ERROR);

if (*TRANSITION\_CHANGED* == arming\_state\_transition(&status,

&battery,

                                           &safety,

                                           new\_arming\_state,

                                           &armed,

                                           true /\*fRunPreArmChecks \*/,

                                           &mavlink\_log\_pub,

                                           &status\_flags,

                                           avionics\_power\_rail\_voltage,

                                           arm\_without\_gps,

arm\_mission\_required,

hrt\_elapsed\_time(&commander\_boot\_timestamp))){

// mavlink\_log\_info(&mavlink\_log\_pub, "DISARMEDby safety switch");

arming\_state\_changed = true;

}

              }

                     //Notify the user ifthe status of the safety switch changes

            if (safety.safety\_switch\_available &&previous\_safety\_off != safety.safety\_off) {

              if (safety.safety\_off){

               set\_tune(*TONE\_NOTIFY\_POSITIVE\_TUNE*);

              } else {

tune\_neutral(true);

}

             status\_changed = true;

          }

}

orb\_check(vtol\_vehicle\_status\_sub,&updated);

orb\_check(global\_position\_sub,&updated);

orb\_check(local\_position\_sub,&updated);

orb\_check(attitude\_sub,&updated);

orb\_check(land\_detector\_sub,&updated);

orb\_check(cpuload\_sub,&updated);

orb\_check(battery\_sub,&updated);

orb\_check(subsys\_sub,&updated);

orb\_check(pos\_sp\_triplet\_sub,&updated);

/\* 在初始化状态下——试着尝试arming\_STANDBY状态 \*/

if(!status\_flags.condition\_calibration\_enabled && status.arming\_state== vehicle\_status\_s::ARMING\_STATE\_INIT) {

                     arming\_ret = arming\_state\_transition(&status,

                                          &battery,

                                          &safety,

                                          vehicle\_status\_s::ARMING\_STATE\_STANDBY,

                                          &armed,

                                          true /\*fRunPreArmChecks \*/,

                                          &mavlink\_log\_pub,

                                          &status\_flags,

                                           avionics\_power\_rail\_voltage,

                                           arm\_without\_gps,

                                           arm\_mission\_required,

                                           hrt\_elapsed\_time(&commander\_boot\_timestamp));

                     if(arming\_ret == *TRANSITION\_CHANGED*) {

                            arming\_state\_changed= true;

                     } else if (arming\_ret== *TRANSITION\_DENIED*) {

                            /\* do notcomplain if not allowed into standby \*/

                            arming\_ret = *TRANSITION\_NOT\_CHANGED*;

                     }

              }

orb\_check(gps\_sub,&updated);

orb\_check(mission\_result\_sub,&updated);

/\* 电子围栏 \*/

orb\_check(geofence\_result\_sub,&updated);

// revertgeofence failsafe transition if sticks are moved and we werepreviously in a manual mode

          // but only if notin a low battery handling action

          if(rc\_override != 0 && !critical\_battery\_voltage\_actions\_done &&(warning\_action\_on &&

            (main\_state\_before\_rtl == commander\_state\_s::MAIN\_STATE\_MANUAL ||

                 main\_state\_before\_rtl == commander\_state\_s::MAIN\_STATE\_ALTCTL ||

                 main\_state\_before\_rtl == commander\_state\_s::MAIN\_STATE\_POSCTL ||

                 main\_state\_before\_rtl == commander\_state\_s::MAIN\_STATE\_ACRO ||

                 main\_state\_before\_rtl == commander\_state\_s::MAIN\_STATE\_RATTITUDE ||

                 main\_state\_before\_rtl == commander\_state\_s::MAIN\_STATE\_STAB))) {

                 // transition toprevious state if sticks are touched

                 if((\_last\_sp\_man.timestamp != sp\_man.timestamp) &&

                        ((fabsf(sp\_man.x -\_last\_sp\_man.x) > min\_stick\_change) ||

                         (fabsf(sp\_man.y - \_last\_sp\_man.y) >min\_stick\_change) ||

                         (fabsf(sp\_man.z - \_last\_sp\_man.z) > min\_stick\_change)||

                         (fabsf(sp\_man.r - \_last\_sp\_man.r) >min\_stick\_change))) {

                     // revert to position control in anycase

                     main\_state\_transition(&status, commander\_state\_s::MAIN\_STATE\_POSCTL,main\_state\_prev,

&status\_flags, &internal\_state);

mavlink\_log\_critical(&mavlink\_log\_pub,"Autopilotoff, returned control to pilot");

                 }

          }

          // abort landing orauto or loiter if sticks are moved significantly

          // but only if notin a low battery handling action

          if(rc\_override != 0 && !critical\_battery\_voltage\_actions\_done &&

                 (internal\_state.main\_state== commander\_state\_s::MAIN\_STATE\_AUTO\_LAND ||

                 internal\_state.main\_state== commander\_state\_s::MAIN\_STATE\_AUTO\_MISSION ||

                 internal\_state.main\_state== commander\_state\_s::MAIN\_STATE\_AUTO\_LOITER)) {

                 // transition toprevious state if sticks are touched

                 if((\_last\_sp\_man.timestamp != sp\_man.timestamp) &&

                        ((fabsf(sp\_man.x -\_last\_sp\_man.x) > min\_stick\_change) ||

                         (fabsf(sp\_man.y - \_last\_sp\_man.y) > min\_stick\_change)||

                         (fabsf(sp\_man.z - \_last\_sp\_man.z) >min\_stick\_change) ||

                         (fabsf(sp\_man.r - \_last\_sp\_man.r) >min\_stick\_change))) {

                        // revert toposition control in any case

                        main\_state\_transition(&status, commander\_state\_s::MAIN\_STATE\_POSCTL,main\_state\_prev,

&status\_flags,&internal\_state);

                        mavlink\_log\_critical(&mavlink\_log\_pub,"Autopilotoff, returned control to pilot");

                 }

          }

          /\* Check for missionflight termination \*/

          if (armed.armed&& \_mission\_result.flight\_termination &&

             !status\_flags.circuit\_breaker\_flight\_termination\_disabled) {

                 armed.force\_failsafe = true;

                 status\_changed = true;

                 static boolflight\_termination\_printed = false;

                 if(!flight\_termination\_printed) {

                        mavlink\_log\_critical(&mavlink\_log\_pub,"Geofenceviolation: flight termination");

                        flight\_termination\_printed= true;

                 }

                 if (counter %(1000000 / COMMANDER\_MONITORING\_INTERVAL) == 0) {

                        mavlink\_log\_critical(&mavlink\_log\_pub,"Flighttermination active");

                 }

          }

          /\* Only evaluatemission state if home is set,

           \* this prevents false positives for themission

           \* rejection. Back off 2 seconds to not overlay

           \* home tune.

           \*/

          if(status\_flags.condition\_home\_position\_valid &&

                 (hrt\_elapsed\_time(&\_home.timestamp)> 2000000) &&

                 \_last\_mission\_instance !=\_mission\_result.instance\_count) {

                 if(!\_mission\_result.valid) {

                        /\* the mission isinvalid \*/

                        tune\_mission\_fail(true);

                        warnx("missionfail");

                 } else if (\_mission\_result.warning){

                        /\* the mission has awarning \*/

                        tune\_mission\_fail(true);

                        warnx("missionwarning");

                 } else {

                        /\* the mission isvalid \*/

                        tune\_mission\_ok(true);

                 }

                 /\* prevent furtherfeedback until the mission changes \*/

                 \_last\_mission\_instance =\_mission\_result.instance\_count;

          }

          /\* RC input check \*/

          if(!status\_flags.rc\_input\_blocked && sp\_man.timestamp != 0 &&

             (hrt\_absolute\_time() < sp\_man.timestamp + (uint64\_t)(rc\_loss\_timeout\* 1e6f))) {

                 /\* handle the casewhere RC signal was regained \*/

                 if(!status\_flags.rc\_signal\_found\_once) {

                        status\_flags.rc\_signal\_found\_once = true;

                        status\_changed = true;

                 } else {

                        if (status.rc\_signal\_lost){

                               mavlink\_log\_info(&mavlink\_log\_pub,"(xiwei)MANUAL control regained after %llums",

                                                  (hrt\_absolute\_time() -rc\_signal\_lost\_timestamp) / 1000);

                               status\_changed = true;

                        }

                 }

if(!in\_armed\_state && status.rc\_input\_mode != vehicle\_status\_s::RC\_IN\_MODE\_OFF &&

(stick\_in\_lower\_right|| arm\_button\_pressed || arm\_switch\_to\_arm\_transition) ) {

           if ((stick\_on\_counter == rc\_arm\_hyst &&stick\_off\_counter < rc\_arm\_hyst) || arm\_switch\_to\_arm\_transition) {

if ((internal\_state.main\_state != commander\_state\_s::MAIN\_STATE\_MANUAL) &&

(internal\_state.main\_state != commander\_state\_s::MAIN\_STATE\_ACRO) &&

(internal\_state.main\_state != commander\_state\_s::MAIN\_STATE\_STAB) &&

(internal\_state.main\_state != commander\_state\_s::MAIN\_STATE\_ALTCTL) &&

(internal\_state.main\_state != commander\_state\_s::MAIN\_STATE\_POSCTL) &&

(internal\_state.main\_state != commander\_state\_s::MAIN\_STATE\_RATTITUDE) ) {

print\_reject\_arm("NOT ARMING: Switch need a manual modefirst.");

              } else if (!status\_flags.condition\_home\_position\_valid &&geofence\_action ==geofence\_result\_s::GF\_ACTION\_RTL) {        print\_reject\_arm("NOTARMING: Geofence RTL requires valid home");

} else if (status.arming\_state== vehicle\_status\_s::ARMING\_STATE\_STANDBY) {

arming\_ret = arming\_state\_transition( &status,

&battery,

&safety,

vehicle\_status\_s::ARMING\_STATE\_ARMED,

&armed,

true /\* fRunPreArmChecks \*/,

&mavlink\_log\_pub,

&status\_flags,

avionics\_power\_rail\_voltage,

arm\_without\_gps,

arm\_mission\_required,

hrt\_elapsed\_time(&commander\_boot\_timestamp));                             if (arming\_ret == *TRANSITION\_CHANGED*) {

arming\_state\_changed = true;

} else {

usleep(100000);

print\_reject\_arm("NOT ARMING: Preflight checksfailed");

}

}

}

stick\_on\_counter++;

} else if(!(arm\_switch\_is\_button == 1 &&

sp\_man.arm\_switch == manual\_control\_setpoint\_s::SWITCH\_POS\_ON)) {

        stick\_on\_counter= 0;

}

\_last\_sp\_man\_arm\_switch = sp\_man.arm\_switch;

/\* evaluate the main state machine according to mode switches \*/

bool first\_rc\_eval = (\_last\_sp\_man.timestamp== 0) && (sp\_man.timestamp > 0);

/\* 遥控模式输入    \*/

transition\_result\_t main\_res = set\_main\_state\_rc(&status);

orb\_check(cmd\_sub, &updated);

/\* checkif we are disarmed and there is a better mode to wait in \*/

if (!armed.armed){

       /\* if there is no radio control butGPS lock the user might want to fly using

 \* just a tablet. Since the RC will force itsmode switch setting on connecting

        \*we can as well just wait in a hold mode which enables tablet control.

        \*/

       if (status.rc\_signal\_lost&& (internal\_state.main\_state == commander\_state\_s::MAIN\_STATE\_MANUAL)

              &&status\_flags.condition\_home\_position\_valid) {

              (void)  main\_state\_transition (&status,commander\_state\_s::MAIN\_STATE\_AUTO\_LOITER,

main\_state\_prev, &status\_flags, &internal\_state);

                     }

       }

       /\* handle commands last, as thesystem needs to be updated to handle them \*/

       orb\_check(cmd\_sub, &updated);

       if (updated){

       /\* got command \*/

              orb\_copy(ORB\_ID(vehicle\_command),cmd\_sub, &cmd);

              /\* handle it \*/

              if(handle\_command(&status, &safety, &cmd, &armed, &\_home,&global\_position, &local\_position,

                                   &attitude,&home\_pub, &command\_ack\_pub, &command\_ack, &\_roi,&roi\_pub)) {

                     status\_changed = true;

              }

      }

/\* now set navigation state according to failsafe andmain state \*/

   bool nav\_state\_changed = set\_nav\_state(&status,    &armed,     &internal\_state,     &mavlink\_log\_pub,    (link\_loss\_actions\_t)datalink\_loss\_act,  \_mission\_result.finished,       \_mission\_result.stay\_in\_failsafe,  &status\_flags,     land\_detector.landed,   (link\_loss\_actions\_t)rc\_loss\_act,    offboard\_loss\_act,   offboard\_loss\_rc\_act);

if(main\_state\_changed || nav\_state\_changed) {

             status\_changed = true;

             main\_state\_changed = false;

      }

/\* 我们知道commander不断循环，但是在循环的过程中为20的倍数，因而commandloop的周期，以及set\_control\_mode函数调用周期也是不固定的但是对于ms运行速度来说，速度至少是5HZ是可以保证的，这使得遥控或者地面站数据对外部模式切换是完全够用的 \*/

     if (counter %(200000 / COMMANDER\_MONITORING\_INTERVAL（default:1000）) == 0 || status\_changed) {

          set\_control\_mode();

          control\_mode.timestamp = now;

          orb\_publish(ORB\_ID(vehicle\_control\_mode),control\_mode\_pub, &control\_mode);

          status.timestamp = now;

           orb\_publish(ORB\_ID(vehicle\_status),status\_pub, &status); // *包含了vtol判断、rc、DLink、arm状态、及飞行模式，*

          armed.timestamp = now;

          /\* set prearmedstate if safety is off, or safety is not present and 5 seconds passed \*/

          if (safety.safety\_switch\_available){

                 /\* safetyis off, go into prearmed \*/

             armed.prearmed = safety.safety\_off;

          } else {

          /\* safety is notpresent, go into prearmed

          \* (all output drivers should bestarted / unlocked last in the boot process

           \* when the rest of the system is fullyinitialized)

           \*/

          armed.prearmed = (hrt\_elapsed\_time(&commander\_boot\_timestamp)> 5 \* 1000 \* 1000);

          }

          orb\_publish(ORB\_ID(actuator\_armed),armed\_pub, &armed); *// 发布飞机 fmu 、及io的加锁解锁状态*

   }

counter++;

/\* publishinternal state for logging purposes \*/

       if(commander\_state\_pub != nullptr) {

            orb\_publish(ORB\_ID(commander\_state), commander\_state\_pub,&internal\_state);

       } else {

            commander\_state\_pub = orb\_advertise(ORB\_ID(commander\_state),&internal\_state);

       }

usleep(COMMANDER\_MONITORING\_INTERVAL*（default: 10000）*);

}

ret = pthread\_join(commander\_low\_prio\_thread,nullptr);

if (ret) {

  warn("join failed: %d", ret);

}

rgbled\_set\_color\_and\_mode(led\_control\_s::COLOR\_WHITE, led\_control\_s::MODE\_OFF);

...

通过上面“凸显的函数”名我们可以抓住主要“code骨架”，光知道函数骨架——那我们还是太单纯了（函数本身并不复杂，为毛是这样子的骨架么？）**（这才是关键）**因此，我们来一点点逻辑。

1、函数基本作用

|  |  |  |
| --- | --- | --- |
| 函数 | 作用 | 备注 |
| arming\_state\_transition | 解锁判断、执行函数 | 注意：3 basic state(mode) |
| set\_main\_state\_rc | RC\_input模式切换、判断 | Easy |
| main\_state\_transition | Mode切换设置 | Easy |
| set\_nav\_state | 导航模式设置 | Easy |
| set\_control\_mode() | 模式设置 | 逻辑判断 + 结论 |

上面罗列了那么多代码，就是为了说清楚这个基本逻辑。源代码中多处用到了函数arming\_state\_transition——都是按照这个思路来做事的。所以代码还是要看的，毕竟，px4的这帮人还是花了时间的，不管啦，还是老话：在想改变一个规则之前，那就先学会它，打破再制定。

有了这个基本思路，我们在往里看看( ARMING )：

transition\_result\_t arming\_state\_transition(       **struct** vehicle\_status\_s \*status,    //飞机状态数据集

**struct** battery\_status\_s \*battery,   // 电量

**const struct** safety\_s \*safety,      //

                                                     arming\_state\_tnew\_arming\_state,    // 新状态（期望值）

**struct** actuator\_armed\_s \*armed,     // 执行机构（如 电机推力）

**bool**fRunPreArmChecks,               // 是否进行前期pre-fligt-check

                                                     orb\_advert\_t\*mavlink\_log\_pub,     ///<uORB handle for mavlink log

                                   status\_flags\_s\*status\_flags,

**float**avionics\_power\_rail\_voltage,   //航电 电压等检测

**bool**arm\_without\_gps,                // 是否允许no GPS 下解锁

**bool**arm\_mission\_required,           // mission任务需求bool量标识

                                                 hrt\_abstime time\_since\_boot)         //绝对运行状态值

{

       第一步：

new\_arming\_state== current\_arming\_state  *// 状态相同。返回*

       fRunPreArmChecks                  *// 是否需要转换前的状态检查bool值*

|  |  |
| --- | --- |
| Yes | 1、try to disarm if ARMED or to STANDBY\_ERROR if ARMED\_ERROR  2、manual try to arming |
| no | 1、Go back INIT/PREFLIGTH     2、Standby |

*//需要pre\_flight check的情形：sensor failing 切非 HIL（仿真）状态*

*如果需要*preflight\_check (){ *则fRunPreArmChecks==1*

Commander::preflightCheck (mavlink\_log\_pub,**true**, **true**, **true**, **true**,checkAirspeed,

(status->rc\_input\_mode== vehicle\_status\_s::RC\_IN\_MODE\_DEFAULT),!arm\_without\_gps,

**true**,status->is\_vtol, reportFailures, prearm, time\_since\_boot);

{ **bool preflightCheck**(orb\_advert\_t\*mavlink\_log\_pub, **bool** checkMag, **bool** checkAcc, **bool** checkGyro,

**bool** checkBaro,**bool**checkAirspeed, **bool** checkRC, **bool** checkGNSS, **bool** checkDynamic, **bool** isVTOL,

**bool** reportFailures, **bool** prearm, hrt\_abstimetime\_since\_boot) }

...

}

第二步：

**新状态值（期望） 老状态值（current arming state）**

**bool** valid\_transition =arming\_transitions[new\_arming\_state][status->arming\_state];

*到这里涉及一个arm\_trans\_table的东西，其实，模式之间能不能转换，都是已经定义好的，（因为每个模式条件下，涉及的传感器，状态值有所不同，因此需要有所了解）*

*于是乎我们跳转到state\_machine\_helper.cpp下的 arming\_transition[new][cur]*

*{*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *ROW==NEW\ COL==CURR* | INIT | STANDBY | ARMED | ARMED\_ERROR | STANDBY\_ERROR | REBOOT | IN\_AIR\_RESTORE |
| ARMING\_STATE\_INIT | *T(1)* | *T(1)* | *F(0)* | *F(0)* | *T(1)* | *F(0)* | *F(0)* |
| **ARMING\_STATE\_STANDBY** | **T(1)** | **T(1)** | **T(1)** | **T(1)** | **F(0)** | **F(0)** | **F(0)** |
| **ARMING\_STATE\_ARMED** | ***F(0)*** | ***T(1)*** | ***T(1)*** | ***F(0)*** | ***F(0)*** | ***F(0)*** | ***T(1)*** |
| ARMING\_STATE\_ARMED\_ERROR | *F(0)* | *F(0)* | *T(1)* | *T(1)* | *F(0)* | *F(0)* | *F(0)* |
| ARMING\_STATE\_STANDBY\_ERROR | *T(1)* | *T(1)* | *T(1)* | *T(1)* | *T(1)* | *F(0)* | *F(0)* |
| ARMING\_STATE\_REBOOT | *T(1)* | *T(1)* | *F(0)* | *F(0)* | *T(1)* | *T(1)* | *T(1)* |
| ARMING\_STATE\_IN\_AIR\_RESTORE | *F(0)* | *F(0)* | *F(0)* | *F(0)* | *F(0)* | *F(0)* | *F(0)* |

值得注意的是：这里虽然定义了转换允许的 **Y/ N 但是有些在 True的情况下 附件条件的存在，也会使得转换不成功：**

*这也就是所谓的 second validation（二次确认）*

*a、在（row==new）***ARMING\_STATE\_ARMED && （curr !=**IN\_AIR\_RESTORE**）“***栏中***”***尽管转换允许，但如果在pre-arm（出错）时，同*

*样最终转换为失败。再入，安全开关使能但是没按情况下，依旧转换失败。*

*b、stand\_by本来就出错了（即就是：自检出错，try arming时候允许转换，但是“二次判断”后再次否决转换）*

*c、给出最后判断*

*}*

**if** (valid\_transition) {

status->arming\_state= new\_arming\_state;

mavlink（“*输出信息*”）；

}

第三步：  跟进结论——如果需要：则重置转换反馈信息；

否则：告诉地面站我转换成 后的 目标

}

做完这些总得有个结果：得到这些结果的下一步就是（特殊情况特殊对待的选项）：跟据home\_position以及可能由于mission特殊情况下，要对某些情况做特殊对待。

比如： 1、在有mission的情况下，转换到mission前是pos\_ctrl，刚起飞就碰到misssion\_finished状况了，则要直接设定值。

      2、DL、RcGPS 信号灯lost  要立马进行飞行状态的“特殊”处理了。

      3、例外，（例外的例外——就是目标了）当然cmd(RC输入命令的进入，正常切换了)则相应的处理时根本“初衷”。

因此，需要引出另一个函数：**main\_state\_transition进行模式设置**

transition\_result\_t**main\_state\_transition**(**struct** vehicle\_status\_s \*status, main\_state\_tnew\_main\_state, uint8\_t&main\_state\_prev,

                                         status\_flags\_s\*status\_flags, **struct** commander\_state\_s\*internal\_state)

{

**switch** (new\_main\_state)

{

case xxx:

case xxx:

}

}

参数：

|  |  |  |
| --- | --- | --- |
| Satuts： | 导航状态，飞行器应该进入的状态 |  |
| new\_main\_state： | 期望切换到的新状态 |  |
| main\_state\_prev： | 之前的状态 |  |
| status\_flags： | commander内部的状态标志 |  |
| status\_flags： | 主状态，用户想要的状态。由遥控器设置，也可以由地面站通过数传控制 |  |

返回值：

|  |  |  |
| --- | --- | --- |
|  |  |  |
| ret = TRANSITION\_DENIED; | // 表示不满足切换条件，拒绝状态切换 |  |
| ret = TRANSITION\_CHANGED; | // 切换到某个状态，不一定是用户想要的目标状态，会根据降级策略，切换至一个最接近的状 |  |
| ret = TRANSITION\_NOT\_CHANGED; |  |  |

解锁成功了、转换逻辑通过确认（这里应该这样称呼才对）了，——>就是该去：模式设置成功了，

**bool handle\_command**( **struct** vehicle\_status\_s\*status\_local,

**const struct** safety\_s\*safety\_local,

**struct** vehicle\_command\_s \*cmd,              //飞行器操作cmd

**struct** actuator\_armed\_s\*armed\_local,

**struct** home\_position\_s \*home,

**struct** vehicle\_global\_position\_s\*global\_pos,

**struct** vehicle\_local\_position\_s\*local\_pos,

**struct** vehicle\_attitude\_s \*attitude,orb\_advert\_t \*home\_pub,

                  orb\_advert\_t\*command\_ack\_pub,

**struct** vehicle\_command\_ack\_s\*command\_ack,

**struct** vehicle\_roi\_s \*roi, orb\_advert\_t \*roi\_pub)  //飞行器感兴趣的区域（or位置）misssion 有关

{

**unsigned** cmd\_result= vehicle\_command\_s::VEHICLE\_CMD\_RESULT\_UNSUPPORTED;

       /\* request to set different systemmode \*/

**switch** (cmd->command){

**case** vehicle\_command\_s::VEHICLE\_CMD\_DO\_REPOSITION: {

              // Just switch theflight mode here, the navigator takes care of

              // doing somethingsensible with the coordinates. Its designed

              // to not requirenavigator and command to receive / process

              // the data at theexact same time.

              // Check if a modeswitch had been requested

**if** ((((uint32\_t)cmd->param2)& 1) > 0)

{

transition\_result\_t main\_ret =main\_state\_transition(status\_local,commander\_state\_s::MAIN\_STATE\_AUTO\_LOITER,

main\_state\_prev, &status\_flags,&internal\_state);

**if** ((main\_ret!= *TRANSITION\_DENIED*)) {

                     cmd\_result = vehicle\_command\_s::VEHICLE\_CMD\_RESULT\_ACCEPTED;

                    } **else** {

                     cmd\_result = vehicle\_command\_s::VEHICLE\_CMD\_RESULT\_TEMPORARILY\_REJECTED;

                     mavlink\_log\_critical(&mavlink\_log\_pub,"Rejectingreposition command");

                    }

              } **else** {

                   cmd\_result = vehicle\_command\_s::VEHICLE\_CMD\_RESULT\_ACCEPTED;

              }

}

**break;**

case vehicle\_command\_s::VEHICLE\_CMD\_DO\_SET\_MODE

**break;**

case vehicle\_command\_s::VEHICLE\_CMD\_DO\_SET\_MODE

**break;**

**answer\_command(\*cmd,cmd\_result, \*command\_ack\_pub, \*command\_ack);**

}

模式确定最后总算在这里**肯定**了下来（这么称呼比较好说）。

到这里，我们做好了大量工作:

① Arming(允许了)——>

② 模式转换判断（Ture）——>

③ 根据RC、Mavlink（vehile\_CMD）——>

④ Mian\_state（最后确定模式）*并回答给发出CMD的控制的主体*——>

⑤ 根据Mode+Cmd/failsafe设定Set\_nav\_sate（）——>*因此我们在进入函数*

**bool set\_nav\_state**(**struct** vehicle\_status\_s \*status,     // 导航状态数据集，飞行器应该进入的状态

**struct** actuator\_armed\_s \*armed,

**struct** commander\_state\_s\*internal\_state,    //模式CMD值，用户主要状态值

            orb\_advert\_t \*mavlink\_log\_pub,

**const** link\_loss\_actions\_t data\_link\_loss\_act, //地面站数据链丢失

**const bool** mission\_finished,

**const bool**stay\_in\_failsafe,                  //故障保护数据

            status\_flags\_s \*status\_flags,                 //commander内部状态标志

**bool** landed,

**const** link\_loss\_actions\_t rc\_loss\_act,

**const int** offb\_loss\_act,

**const int** offb\_loss\_rc\_act)

{

//记录old\_参数

navigation\_state\_t nav\_state\_old = status->nav\_state;

...

**switch**(internal\_state->main\_state)

{

**case** commander\_state\_s::MAIN\_STATE\_ACRO:

**case** commander\_state\_s::MAIN\_STATE\_MANUAL:

**case** commander\_state\_s::MAIN\_STATE\_RATTITUDE:

**case** commander\_state\_s::MAIN\_STATE\_STAB:

**case** commander\_state\_s::MAIN\_STATE\_ALTCTL:

/\* require RC for all manual modes \*/

**if** (rc\_lost&& is\_armed) {

          enable\_failsafe(status, old\_failsafe,mavlink\_log\_pub, reason\_no\_rc);

          set\_rc\_loss\_nav\_state(status, armed,status\_flags, rc\_loss\_act);

    } **else** {

**switch**(internal\_state->main\_state) {

**case** commander\_state\_s::MAIN\_STATE\_ACRO:

                 status->nav\_state = vehicle\_status\_s::NAVIGATION\_STATE\_ACRO;

**break**;

**case** commander\_state\_s::MAIN\_STATE\_MANUAL:

                 status->nav\_state = vehicle\_status\_s::NAVIGATION\_STATE\_MANUAL;

**break**;

**case** commander\_state\_s::MAIN\_STATE\_RATTITUDE:

                 status->nav\_state = vehicle\_status\_s::NAVIGATION\_STATE\_RATTITUDE;

**break**;

**case** commander\_state\_s::MAIN\_STATE\_STAB:

                 status->nav\_state = vehicle\_status\_s::NAVIGATION\_STATE\_STAB;

**break**;

**case** commander\_state\_s::MAIN\_STATE\_ALTCTL:

                 status->nav\_state = vehicle\_status\_s::NAVIGATION\_STATE\_ALTCTL;

**break**;

**default**:

                 status->nav\_state = vehicle\_status\_s::NAVIGATION\_STATE\_MANUAL;

**break**;

          }

    }

**break**;

...

}

    …

*//以下如法炮制*

   …

…

**return** status->nav\_state != nav\_state\_old; //*设置成功逻辑为1*

}

返回值：

|  |  |
| --- | --- |
| *1* | *状态改变成功* |
| *0* | *状态未变（失败）* |

**一切判断，做完了最后是设置了：那就是函数：**

⑥ set\_control\_mode();

这个函数简单就是最后的模式设定了，但是值得注意的是，对于不同的模式在加速度，速度，XY平面内\Z方向上的  速度加速度，tilt、VTOL、转换过程中的有误增稳（加速度\角速度控制）等等都有相应的最后设定，即其设定control\_mode.flag\_xxx值。

好了，到这个地方基本上算是思路和路子走完了，但是很多代码细节还是要细度分析的。

剩下的就是及时的发布各种平行并列的消息给相应的模块提供指示和参考了。

orb\_publish(ORB\_ID(vehicle\_status), status\_pub, &status);

orb\_publish(ORB\_ID(actuator\_armed), armed\_pub, &armed);

orb\_publish(ORB\_ID(commander\_state), commander\_state\_pub,&internal\_state);

。。。

等等，还有件事就是，

rgbled\_set\_color\_and\_mode(led\_control\_s::COLOR\_WHITE,led\_control\_s::MODE\_OFF);

它做了—>orb\_publish(ORB\_ID(led\_control),led\_control\_pub, &led\_control);给led驱动模块参考用。

基本将commander的顺序理解完了，还有一件大事，耶，对了突然想起来还有个哥们在干活嘞，那就是Low\_prio\_thread了，你到底是干啥子的嘞。看来还不能休息哈,那我们一口气干完再说休息么。那还是继续…

**void** \***commander\_low\_prio\_loop**(**void** \*arg)*//指针函数提供posix跨平台api——创建线程入口*

{  //这个线程的函数名字为：commander\_low\_prio

订阅了一个msg // ORB\_ID(vehicle\_command)

…

}

可以看到在这个函数中忽略了high\_prio command 执行了 低优先级任务：包含：

**case** vehicle\_command\_s::VEHICLE\_CMD\_PREFLIGHT\_REBOOT\_SHUTDOWN:

**case** vehicle\_command\_s::VEHICLE\_CMD\_PREFLIGHT\_CALIBRATION:

**case** vehicle\_command\_s::VEHICLE\_CMD\_START\_RX\_PAIR:

**case**vehicle\_command\_s::VEHICLE\_CMD\_PREFLIGHT\_STORAGE:

低优先级函数到底怎么被调用执行了呢？

## 第二章 软件架构与功能

**软件的堆栈结构主要分为四层:**

**PX4 在广播消息网络内，按照一组节点（nodes ）的形式进行组织，网络之间使用像如“姿态”和“位置”之类的语义通道来传递系统状态。这四层机构如下:**

1. **应用程序接口:提供给app开发人员，使开发人员可以使用**[**ROS**](http://en.wikipedia.org/wiki/Robot%20Operating%20System)**或者**[**DroneAPI**](http://droneapi.io/)**。这一接口设计尽可能简洁和扁平，并尽量多的隐藏复杂性。**
2. **应用框架：这是一组核心缺省应用程序（节点），运行核心的飞行控制。**
3. **函数库：这一层包括针对机器核心操作的所有系统函数库和功能**
4. **操作系统：最后一层提供硬件驱动、网络通信、UAVCAN和故障安全系统。**

**PX4自动驾驶仪软件可分为三大部分：实时操作系统、中间件、飞行控制栈。如下图**



**Nuttx实时操作系统，提供posix-style的用户操作环境，如printf(), open(),write(),poll(),ioctl()等系统调用函数。**

**PX4中间件，运行于Nuttx之上，提供设备驱动和一个微对象请求代理（micro object request broker，uorb)用于驾驶仪上运行的单个任务之间的异步通信。PX4为了简化开发而采用牺牲部分效率的消息传递机制，是PX4 与Ardupilot 最本质的差别。**

**px4飞行控制栈结构：**

**决策导航部分：根据飞行器自身安全状态和接收到的命令，决定工作于什么模式，下一步应该怎么做。**

**位置姿态估计部分：根据传感器得到自身的位置和姿态信息，此部分算法含金量最高，算法也相当多。**

**位置姿态控制部分：根据期望位置和姿态设计控制结构，尽可能快、稳的达到期望位置和姿态**

**控制器输出部分：mixer和执行器，pwm限幅。**

## ****第三章 PX4姿态解算****

### 姿态解算1\_入门篇（DCM Nomalize）

**一、开篇**

需要掌握欧拉角、DCM、四元数、gyro误差、矫正、正交化等各个概念。然后就是各种转换公式。

        一定要多看论文啊，英文版的论文（也没有中文的，国人的悲哀啊），虽然看着头疼，看是看完了以后就会发现许多不了解的迷惑的就自然都解开了。

二、基本知识

1、如何实现控制

        一个无人机系统的算法主要有两类：姿态检测算法、姿态控制算法。姿态控制、被控对象、姿态检测三个部分构成一个闭环控制系统。被控对象的模型是由其物理系统决定，设计无人机的算法就是设计姿态控制算法、姿态检测算法。

        1）姿态检测算法：姿态的表示可以用欧拉角，也可以用四元数。姿态检测算法的作用就是将加速度计、陀螺仪等传感器的测量值解算成姿态，进而作为系统的反馈量。在获取sensors值之前需要对数据进行滤波，滤波算法主要是将获取到的陀螺仪和加速度计的数据进行去噪声及融合，得出正确的角度数据（欧拉角或四元数），主要采用互补滤波或者高大上的卡尔曼滤波。

        2）姿态控制算法：控制无人机姿态的三个自由度，用给定姿态与姿态检测算法得出的姿态偏差作为输入，被控对象模型的输入量作为输出（例如姿态增量），从而达到控制无人机姿态的作用。最常用的就是PID控制及其各种PID扩展（分段、模糊等，我的毕设就是模糊PID控制算法，惨了，啥都不懂，还能毕业不，哎~~~），高端点的有自适应控制（自动壁障应该就用这个）。当然，姿态控制算法里面比较常用角速度、角度双闭环控制（所谓的两级PID控制），所以常常有PID外环+PID内环等等，懂我搞懂了再细说吧。PID算法就是控制四个电机的转速来纠正欧拉角，从而使机身保持平稳。

**五、正文**

1、ardupilot到PX4Firmware的变化

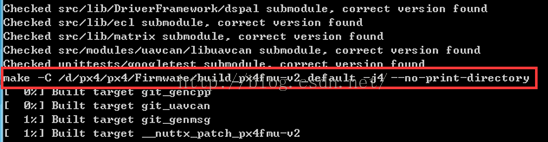
        首先说明一下，这两套代码我是结合着看的，反正ardupilot的底层也是直接调用的PX4Firmware，博文还是以ardupilot为主线展开。介绍一下ardupilot到PX4Firmware的变化吧，其实变化的不多，主要就是由原本的make转变到了现在的Cmake，原来在ardupilot里面处处可以见到.mk，现在是彻底的没有了，全部都是CmakeList.txt和.cmake了，它俩就是由Cmake写的makefile，一看就知道里面是什么意思，不做解释了。

        另外一个比较重要的变化就是关于各种库的选择编译的配置，搬移到了PX4-Firmware/cmake/config中，即nuttx\_px4fmu-v2\_default.cmake，可以自己修改这个cmake文件增加或删除某个库。该部分会在console控制台中使用命令make编译时调用，在PX4-Firmware/Makefile中有如下代码：

[在CODE上查看代码片](https://code.csdn.net/snippets/1615323)

1. px4fmu-v2\_default:
2. $(call cmake-build,nuttx\_px4fmu-v2\_default)
3. cmake-build是宏定义，nuttx\_px4fmu-v2\_default作为参数传入。
4. # Functions
5. # --------------------------------------------------------------------
6. # describe how to build a cmake config
7. define cmake-build
8. +@if [ $(PX4\_CMAKE\_GENERATOR) = "Ninja" ] && [ -e $(PWD)/build\_$@/Makefile ]; then rm -rf $(PWD)/build\_$@; fi
9. +@if [ ! -e $(PWD)/build\_$@/CMakeCache.txt ]; then mkdir -p $(PWD)/build\_$@ && cd $(PWD)/build\_$@ && cmake .. -G$(PX4\_CMAKE\_GENERATOR) -DCONFIG=$(1); fi
10. +Tools/check\_submodules.sh/\*该脚本检查是否需要更新固件源码，如不需可以直接屏蔽\*/
11. +$(PX4\_MAKE) -C $(PWD)/build\_$@ $(PX4\_MAKE\_ARGS) $(ARGS)
12. Endef
13. 其中：PX4\_MAKE = make
14. PX4\_MAKE\_ARGS = -j$(j) --no-print-directory

        解析以后的命令就是如下图所示。



        简要介绍一下Cmake：其实也没什么难的，我们不需要写多高端的，只是会用它即可，几乎每个文件里面都有.cmake，在PX4-Firmware/cmake/中的common或者nuttx中是一些基本的cmake函数（类似于C/C++语言的的库一样）以供在别处编写makefile时使用（比如在PX4-Firmware/src/modules/commander的CMakeLists.txt）。

        举一例说明问题，在PX4-Firmware/cmake/common中的px4\_base.cmake，该文件的最前端部分会介绍本file内部有哪些function以及介绍每个function的用法，使用时就按照这里面的example仿写就行。在CMakeLists.txt中不需要使用include（xxx.cmake）就可以使用需要的function，但是在xxx.cmake中使用时需要include(common/px4\_base) 以后才可以使用（和C/C++类似）。下面是Firmware/src/modules/commander的CMakeLists.txt代码。

[在CODE上查看代码片](https://code.csdn.net/snippets/1615323)

1. set(MODULE\_CFLAGS -Os)
2. if(${OS} STREQUAL "nuttx")            /\*判断OS类型\*/
3. list(APPEND MODULE\_CFLAGS -Wframe-larger-than=2450)
4. endif()
5. px4\_add\_module(                   /\*在px4\_base.cmake中定义，里面会介绍用法\*/
6. MODULE modules\_\_commander   /\*标明路径\*/
7. MAIN commander   /\* 类似于.mk中的MODULE\_COMMAND = commander\*/
8. STACK 4096
9. COMPILE\_FLAGS
10. ${MODULE\_CFLAGS}
11. -Os
12. SRCS                          /\*source files\*/
13. commander.cpp
14. state\_machine\_helper.cpp
15. commander\_helper.cpp
16. calibration\_routines.cpp
17. accelerometer\_calibration.cpp
18. gyro\_calibration.cpp
19. mag\_calibration.cpp
20. baro\_calibration.cpp
21. rc\_calibration.cpp
22. airspeed\_calibration.cpp
23. esc\_calibration.cpp
24. PreflightCheck.cpp
25. DEPENDS
26. platforms\_\_common
27. )

2、在分析代码之前首先需要了解一下arducopter/copter.h文件，内部以class类的形式定义了几乎所有的使用的函数。平时索引函数时可以先到该文件中查找一下。

3、程序的main入口（补充）

        总的来说，这里的main函数就是ArduCopter.cpp里的AP\_HAL\_MAIN\_CALLBACKS(&copter)，它实际上是一个宏定义，传进来的参数为类对象的引用，通过在AP\_HAL\_Main.h里的定义可知原型为：

[在CODE上查看代码片](https://code.csdn.net/snippets/1615323)

1. #define AP\_HAL\_MAIN\_CALLBACKS(CALLBACKS) extern "C" { \
2. int AP\_MAIN(int argc, char\* const argv[]); \
3. int AP\_MAIN(int argc, char\* const argv[]) { \
5. hal.run(argc, argv, CALLBACKS); \
6. return 0; \
7. } \
8. }

        而这里的AP\_MAIN也是一个宏，如下：

[在CODE上查看代码片](https://code.csdn.net/snippets/1615323)

1. #if CONFIG\_HAL\_BOARD == HAL\_BOARD\_PX4 || CONFIG\_HAL\_BOARD == HAL\_BOARD\_VRBRAIN
2. #define AP\_MAIN \_\_EXPORT ArduPilot\_main
3. #endif

        解析以后实际上是这样的：

1. #define AP\_HAL\_MAIN\_CALLBACKS(CALLBACKS) extern "C" { \
2. int \_\_EXPORT ArduPilot\_main(int argc, char\* const argv[]); \
3. int \_\_EXPORT ArduPilot\_main(int argc, char\* const argv[]) { \
4. hal.run(argc, argv, CALLBACKS); \
5. return 0; \
6. } \
7. }

        将这个宏替换到ArduCopter.cpp里的AP\_HAL\_MAIN\_CALLBACKS(&copter)，它就变成了：

1. int \_\_EXPORT ArduPilot\_main(int argc, char\* const argv[]);
2. int \_\_EXPORT ArduPilot\_main(int argc, char\* const argv[]) {
3. hal.run(argc, argv, &copter);
4. return 0;
5. }

        因此实际上这个工程的main函数就是ArduCopter.cpp里的ArduPilot\_main函数。那么这里可能又牵扯到了一个问题，ArduPilot\_main函数又是怎么调用的呢？如果像以前我们经常使用的单片机裸机系统，入口函数就是程序中函数名为main的函数，但是这个工程里边名字不叫main，而是ArduPilot\_main，所以这个也不像裸机系统那样去运行ArduPilot\_main那么简单。区别在于这是跑的Nuttx操作系统，这是一个类Unix的操作系统，它的初始化过程是由脚本去完成的。注意一个重要的词——脚本，如果你对Nuttx的启动过程不是很熟悉，可以查看我先前写的一些文章。而在这里需要注意两个脚本，一个是ardupilot/mk/PX4/ROMFS/init.d里的rcS，另一个是rc.APM，这个脚本在rcS里得到了调用，也就是说，rcS就是为Nuttx的启动文件。

        那么到底调用ArduPilot\_main的地方在哪里呢？查看rc.APM的最低端：

1. echo Starting ArduPilot $deviceA $deviceC $deviceD
2. if ArduPilot -d $deviceA -d2 $deviceC -d3 $deviceD start
3. then
4. echo ArduPilot started OK
5. else
6. sh /etc/init.d/rc.error
7. fi

        其中ArduPilot是一个内嵌的应用程序，由编译生成的builtin\_commands.c可知，这个应用程序的入口地址就是ArduPilot\_main。{"ArduPilot", SCHED\_PRIORITY\_DEFAULT, 4096, ArduPilot\_main},

1. {"px4flow", SCHED\_PRIORITY\_DEFAULT, CONFIG\_PTHREAD\_STACK\_DEFAULT, px4flow\_main}

        这样的命令有很多，在rcS里就开始调用的。至于这些内置的命令是怎么生成的，就要了解PX4Firmware的编译过程了。查看px4.targes.mk。 PX4\_MAKE = $(v)+ GIT\_SUBMODULES\_ARE\_EVIL=1 ARDUPILOT\_BUILD=1 $(MAKE) -C $(SKETCHBOOK) -f $(PX4\_ROOT)/Makefile.make EXTRADEFINES="$(SKETCHFLAGS) $(WARNFLAGS) $(OPTFLAGS) "'$(EXTRAFLAGS)' APM\_MODULE\_DIR=$(SKETCHBOOK) SKETCHBOOK=$(SKETCHBOOK) CCACHE=$(CCACHE) PX4\_ROOT=$(PX4\_ROOT) NUTTX\_SRC=$(NUTTX\_SRC) MAXOPTIMIZATION="-Os" UAVCAN\_DIR=$(UAVCAN\_DIR)

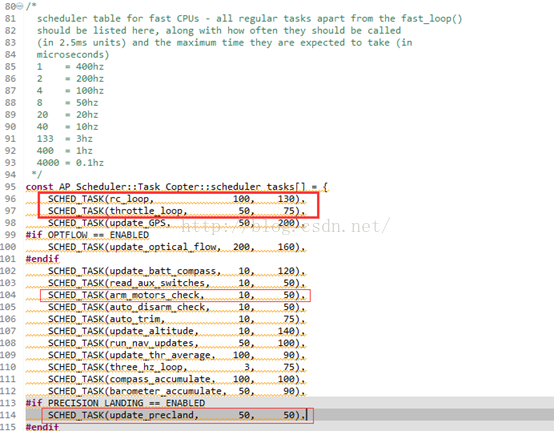
4、arducopter.cpp

        分析ardupilot这套代码首先就是拿arducopter.cpp开刀，

        Loop函数的设计框架既要准确，又要高效。总体设计是这样的：

        其一用一个计时器定时触发测量；其二所有测量过程都靠中断推进；其三在main函数里不断检查测量是否完成，完成就进行解算。测量过程还是比较耗时间的，基于这样的设计，测量和解算可以同时进行，不会浪费CPU时间在（等待）测量上。而通过计时器触发测量，最大限度保证积分间隔的准确。

         整个控制过程主要就集中在arducopter.cpp里面，首先就是scheduler\_tasks[]中是需要创建的task线程，如下图中，接收机的rc\_loop、arm\_motors\_check等，还记得上篇博文中介绍的解锁和上锁的操作么，就是在arm\_motors\_check函数中实现的，以固定的频率去采集遥控器信号。



1）setup函数

        然后就是setup函数，在该函数中做的事情还是比较全面的，其中内部调用了一个比较重要的函数init\_ardupilot()，该函数做了一系列的初始化，该初始化和上一篇博文介绍的不一样，上一篇主要是配置底层硬件的（如STM32、sensors），而此处的主要是飞行前的检测比那个初始化的晚；比如检测是否有USB连接、初始化UART、打印固件版本、初始化电源检测、初始化RSSI、注册mavlink服务、初始化log、初始化rc\_in/out（内部含有电调校准）、初始化姿态/位置控制器、初始化飞行模式、初始化aux\_switch、使能失控保护、配置dataflash最后打印"Ready to FLY "。接下来就几个比较重要的函数（上述标红）进行深入分析。

1. void Copter::init\_rc\_in()
2. {
3. //rc\_map 结合AC\_RCMapper.h
4. channel\_roll     = RC\_Channel::rc\_channel(rcmap.roll()-1);
5. channel\_pitch    = RC\_Channel::rc\_channel(rcmap.pitch()-1);
6. channel\_throttle = RC\_Channel::rc\_channel(rcmap.throttle()-1);
7. channel\_yaw      = RC\_Channel::rc\_channel(rcmap.yaw()-1);
8. // set rc channel ranges
9. channel\_roll->set\_angle(ROLL\_PITCH\_INPUT\_MAX);//4500
10. channel\_pitch->set\_angle(ROLL\_PITCH\_INPUT\_MAX);
11. channel\_yaw->set\_angle(4500);
12. channel\_throttle->set\_range(g.throttle\_min, THR\_MAX);//1000
14. channel\_roll->set\_type(RC\_CHANNEL\_TYPE\_ANGLE\_RAW);
15. channel\_pitch->set\_type(RC\_CHANNEL\_TYPE\_ANGLE\_RAW);
16. channel\_yaw->set\_type(RC\_CHANNEL\_TYPE\_ANGLE\_RAW);
17. //set auxiliary servo ranges
18. g.rc\_5.set\_range\_in(0,1000);
19. g.rc\_6.set\_range\_in(0,1000);
20. g.rc\_7.set\_range\_in(0,1000);
21. g.rc\_8.set\_range\_in(0,1000);
22. // set default dead zones
23. default\_dead\_zones();
24. // initialise throttle\_zero flag
25. ap.throttle\_zero = true;//注意这个，rc\_map好以后把油门配置为0
26. // true if the throttle stick is at zero, debounced, determines if pilot intends shut-down when not using motor interlock
27. }

1. // init\_rc\_out -- initialise motors and check if pilot wants to perform ESC calibration
2. void Copter::init\_rc\_out()
3. {
4. motors.set\_update\_rate(g.rc\_speed);
5. motors.set\_frame\_orientation(g.frame\_orientation);//配置机体类型（+/x）
6. motors.Init();                                              // motor initialisation
7. #if FRAME\_CONFIG != HELI\_FRAME
8. motors.set\_throttle\_range(g.throttle\_min, channel\_throttle->radio\_min, channel\_throttle->radio\_max);//配置油门最大值和最小值    motors.set\_hover\_throttle(g.throttle\_mid);
9. #endif
11. for(uint8\_t i = 0; i < 5; i++) {
12. delay(20);
13. read\_radio();
14. }
15. // we want the input to be scaled correctly
16. channel\_throttle->set\_range\_out(0,1000);
17. // check if we should enter esc calibration mode
18. esc\_calibration\_startup\_check();//电调校准，进入以后首先判断是否有pre-arm，如果没有则直接退出校准。校准过飞机的都知道这个
19. // enable output to motors
20. pre\_arm\_rc\_checks();
21. if (ap.pre\_arm\_rc\_check) {
22. enable\_motor\_output();//内部会调用motors.output\_min()函数sends minimum values out to the motors，待会介绍该函数
23. }
24. // refresh auxiliary channel to function map
25. RC\_Channel\_aux::update\_aux\_servo\_function();
26. // setup correct scaling for ESCs like the UAVCAN PX4ESC which
27. // take a proportion of speed.
28. hal.rcout->set\_esc\_scaling(channel\_throttle->radio\_min, channel\_throttle->radio\_max);
29. }

        简单介绍如何如何控制电机转动以及cork() and push()，并在此基础上测试了关于scheduler\_tasks[] 中的任务的执行频率是否可以达到要求。测试方法：在scheduler\_tasks[] 任务列表的throttle\_task中添加一个累加标志位counterflag，每执行一次throttle\_task任务就对齐加1，加到100时使电机转动，测试结果约为5S时电机转动，理论是50HZ的周期（不加执行时间是需要2S转动）再加上每次需要的执行时间以后还是比较理想的。

1. // output\_min - sends minimum values out to the motors
2. void AP\_MotorsMatrix::output\_min()
3. {
4. int8\_t i;
5. // set limits flags
6. limit.roll\_pitch = true;
7. limit.yaw = true;
8. limit.throttle\_lower = true;
9. limit.throttle\_upper = false;
10. // fill the motor\_out[] array for HIL use and send minimum value to each motor
11. hal.rcout->cork();
12. /\*Delay subsequent calls to write() going to the underlying hardware in
13. order to group related writes together. When all the needed writes are
14. done, call push() to commit the changes.
15. This method is optional: if the subclass doesn't implement it all calls
16. to write() are synchronous.\*/
17. for( i=0; i<AP\_MOTORS\_MAX\_NUM\_MOTORS; i++ ) {
18. // AP\_MOTORS\_MAX\_NUM\_MOTORS=8
19. if( motor\_enabled[i] ) {
20. rc\_write(i, \_throttle\_radio\_min);
21. //下面会解析rc\_write(uint8\_t chan, uint16\_t pwm)
22. }
23. }
24. hal.rcout->push();
25. /\*Push pending changes to the underlying hardware. All changes between a call to cork() and push() are pushed together in a single transaction.This method is optional: if the subclass doesn't implement it all calls to write() are synchronous.\*/
26. }
27. //由上述可知在通过HAL层配置数据时使用cork() and push()函数包装需要单次传输的数据。该方法就是为了实现对四个电机同时配置，避免由for语句产生的延时，也是强调同步（synchronous）。
28. 如下解析rc\_write(uint8\_t chan, uint16\_t pwm)
29. void AP\_Motors::rc\_write(uint8\_t chan, uint16\_t pwm)
30. {
31. if (\_motor\_map\_mask & (1U<<chan)) {
32. // we have a mapped motor number for this channel
33. chan = \_motor\_map[chan];// mapping to output channels
34. }
35. hal.rcout->write(chan, pwm);//通过HAL层直接输出配置电调
36. }

        Setup()函数的最后一句是fast\_loopTimer = AP\_HAL::micros()，获取micros()通过层层封装最终就是上篇博文中介绍的hrt，该时间会在下面的loop函数中使用。

2）loop函数

        该函数比较重要，fast\_loop和schedule\_task都是封装在该函数中的，下面主要讲述fast\_loop。

// Main loop - 400hz

1. void Copter::fast\_loop()
2. {
3. // IMU DCM Algorithm 里面有个update函数，这个函数就是读取解算好的角度信息
4. read\_AHRS();
5. // run low level rate controllers that only require IMU data
6. attitude\_control.rate\_controller\_run();
7. // send outputs to the motors library
8. motors\_output();
9. // Inertial Nav
10. read\_inertia();
11. // check if ekf has reset target heading
12. check\_ekf\_yaw\_reset();
13. // run the attitude controllers
14. update\_flight\_mode();
15. // update home from EKF if necessary
16. update\_home\_from\_EKF();
17. // check if we've landed or crashed
18. update\_land\_and\_crash\_detectors();
19. // log sensor health
20. if (should\_log(MASK\_LOG\_ANY)) {
21. Log\_Sensor\_Health();
22. }
23. }

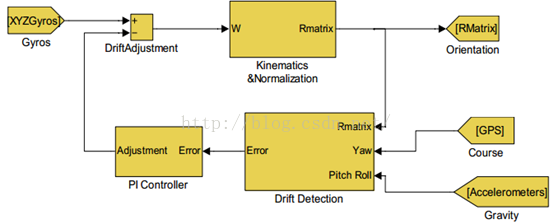
        read\_AHRS()内部使用ahrs.update更新AHRS数据。// run a full DCM update round

1. Void AP\_AHRS\_DCM::update(void)
2. {
3. // tell the IMU to grab some data
4. \_ins.update();//update gyro and accel values from backends具体实现过程详见源码
5. // ask the IMU how much time this sensor reading represents
6. delta\_t = \_ins.get\_delta\_time();
7. // Integrate the DCM matrix using gyro inputs
8. //使用陀螺仪数据更新DCM矩阵（方向余弦矩阵：direction-cosine-matrix ），使用刚刚测量出来的陀螺仪值、以及上个周期对陀螺仪的补偿值进行角度更新
9. matrix\_update(delta\_t);
10. // Normalize the DCM matrix 归一化处理
11. normalize();
12. // Perform drift correction
13. drift\_correction(delta\_t);
14. // paranoid check for bad values in the DCM matrix
15. check\_matrix();
16. // Calculate pitch, roll, yaw for stabilization and navigation
17. euler\_angles();
18. // update trig values including \_cos\_roll, cos\_pitch
19. update\_trig();
20. }

        到此为止，卡住了。不是因为不懂C++的缘故，而是因为理  论  知  识  太  欠  缺 了~~~~  
        所以还是好好的准备理论知识吧，大把大把的论文要看。我是看Mahony的和Paul的，现在主要是Paul的《Direction Cosine Matrix IMU: Theory》，讲的实在是太好了，搞无人机必看啊，这篇文章的最后给出了Mahony论文的下载地址。

六、高大上的理论

        关于上面的这个Normalize the DCM matrix（ 归一化处理）很有深度，值得了解一下其原理，在使用DCM控制和导航时，肯定存在积累数值的舍入误差、陀螺漂移误差、偏移误差、增益误差。它主要就是补偿抵消这几种误差（注意这几种误差 error）的；使用PI负反馈控制器检测这些误差，并在下一次产生之前就抵消这些误差（GPS检测yaw error，加速度计检测pitch和roll error）。在ardupilot源码中使用的Normalize算法就是来自2009年Paul的研究成果--**Direction Cosine Matrix IMU: Theory**。首先了解一下几个比较关键的概念。用下图先有个感性认识吧



1、陀螺漂移（Gyro drift）

        由于外干扰力矩（机械摩擦、振动等因素）引起的陀螺自转轴的缓慢进动：陀螺漂移。通常，干扰力矩分为两类，与之对应的陀螺漂移也分为两类：一类干扰力矩是系统性的，规律已知，它引起规律性漂移，因而是可以通过计算机加以补偿的；另一类是随机因素造成的，它引起随机漂移。在实际应用中，除了要尽可能减小随机因素的影响外，对实验结果还要进行统计处理，以期对随机漂移作出标定，并通过系统来进行补偿。但由于它是无规律的，很难达到。

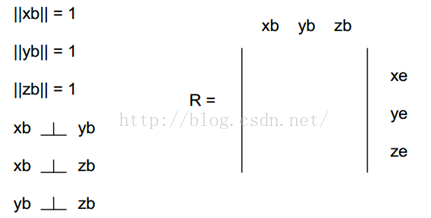
        平时所说的用加速计（静态特性好）修正陀螺仪的漂移，其实这个修正是利用加速度计修正陀螺仪计（动态特性好）算出的姿态，并估计出陀螺仪的漂移，在下一次做姿态解算时，陀螺仪的输出减去估计出的漂移后再做捷联姿态解算，以此不断循环。融合的方法一般用EKF，KF也是基于最优估计的。

2、误差来源

        在进行数值积分的过程中一定会引入数值误差，数值误差分为两类；一类是积分误差（来自于我们假设旋转速率在短时间内不变引起的），另一类是量化误差（主要来自于模数转换过程中引起的）。关于gyro误差产生的详细分析见：<http://www.crazepony.com/wiki/mpu6050.html>

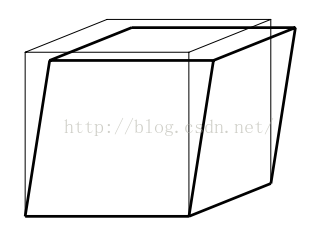
3、旋转矩阵的约束

        旋转矩阵就是改变方向不改变大小；旋转矩阵有9个元素，实际上是只有三个是独立的。由于旋转矩阵的正交性，意味着任何一对行或者列都是相互垂直的，并且每个行或者列的元素的平方和为1。所以在9个元素中有6个限制条件。



4、误差导致的结果

        旋转矩阵的关键性能之一就是正交性，即在某个参考坐标系下的三个方向的向量两两垂直，且向量长度在每个参考系下都是等长的。数值误差会违背该性能，比如旋转矩阵的行和列都是单位向量，其长度都是1，但是由于数值误差的原因导致其长度边长或变短；最终致使它们缩小到0或者增长到无穷大，最后的结果就是导致原本相互正交的现在变的倾斜了。如下图所示。

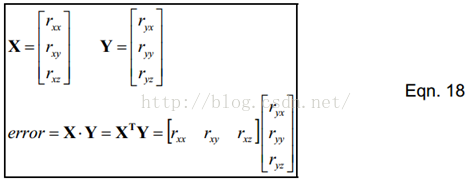


        那么问题来了，如何修正这个问题呢？可以使用如下方法（ Ardupilot源码中就是利用如下算法处理的errors）。

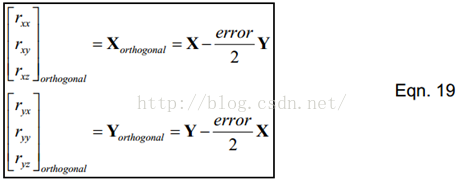
5、如何消除各种误差（积累数值的舍入误差、陀螺漂移误差、偏移误差、增益误差）

        其方法就是采样DCMIMU:Theory中提出的理论—强制正交化（也称为Renormalization）。

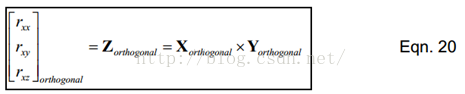
        首先计算矩阵中X、Y行的点积（dotproduct），理论上应该是0，但是由于各种errors的影响，所以点积的值不是0，而表示为error。



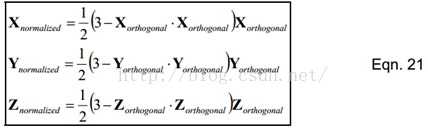
        然后把这个误差项分半交叉耦合到X、Y行上，如下公式。



        通过上述两个公式处理过以后，正交误差明显减小了很多，即R旋转矩阵的每个行和列的长度都非常接近1。接下来就是调整旋转矩阵的Z行，使其与X、Y正交，方法比较简单，直接使用X、Y的叉积（cross product）即可。



        最后一步就是放缩旋转矩阵的每一行使其长度等于1，现在用一种比较简单的方法实现，即使用泰勒展开。



        ardupilot中的源码实现如下：

1. Void AP\_AHRS\_DCM::normalize(void)
2. {
3. float error;
4. Vector3f t0, t1, t2;
5. error = \_dcm\_matrix.a \* \_dcm\_matrix.b;     // eq.18
6. t0 = \_dcm\_matrix.a - (\_dcm\_matrix.b \* (0.5f \* error));      // eq.19
7. t1 = \_dcm\_matrix.b - (\_dcm\_matrix.a \* (0.5f \* error));      //eq.19
8. t2 = t0 % t1;                            // c= a x b // eq.20
9. if (!renorm(t0, \_dcm\_matrix.a) ||
10. !renorm(t1, \_dcm\_matrix.b) ||
11. !renorm(t2, \_dcm\_matrix.c)) {
12. // Our solution is blowing up and we will force back
13. // to last euler angles
14. \_last\_failure\_ms = AP\_HAL::millis();
15. AP\_AHRS\_DCM::reset(true);
16. }
17. }

七、结论  
        本篇博文没有介绍多少关于源代码的，主要就时觉得理论太欠缺了，接下来还有四元数，控制中主要还是用它做运算，还有各种滤波。接下来就是主要看关于姿态估计的了，姿态估计算法实现主要就是在PX4Firmware/src/modules/attitude\_estimator\_q中（q：quaternion），首先介绍一些代码执行顺序，方便后期有个良好的逻辑框架阅读代码和习惯。

        1）  首先就是该文件中需要的头文件的包含；

        2）  然后是一个C++的引用定义extern"C" \_\_EXPORT int attitude\_estimator\_q\_main(int argc, char \*argv[])，可以根据这个attitude\_estimator\_q\_main进行追踪到函数原型（754）；

        3）  在attitude\_estimator\_q\_main函数中调用姿态估计的启动函数start()（775）；

        4）  详细介绍一下该启动函数，比较重要，源码中很多都是靠这种方法启动的，还记上次讲的sensor初始化么。源码如下。

1. int AttitudeEstimatorQ::start()
2. {
3. ASSERT(\_control\_task == -1);
4. /\* start the task \*//\*POSIX接口的任务启动函数\*/
5. \_control\_task = px4\_task\_spawn\_cmd("attitude\_estimator\_q",
6. SCHED\_DEFAULT,
7. SCHED\_PRIORITY\_MAX - 5,
8. 2100,
9. (px4\_main\_t)&AttitudeEstimatorQ::task\_main\_trampoline,
10. nullptr);
11. if (\_control\_task < 0) {
12. warn("task start failed");
13. return -errno;
14. }
15. return OK;
16. }

[http://static.blog.csdn.net/images/save_snippets.png](javascript:;)

        5） 然后是task\_main\_trampoline函数，内部跳转到task\_main()

        好了，就是它了，慢慢研究吧，把这个里面的过程都研究透吧~~~~至少好几天

### 姿态解算2\_源码姿态解算算法分析

**一、开篇**

        还是没能进入到源码部分研究，对姿态解算过程太过于模糊，所以主要开始研究一下关于姿态解算的过程和实现，本篇博文主要是以mahony的[**算法**](http://lib.csdn.net/base/datastructure)为基础理解姿态解算的过程，主要参考的论文就是William Premerlani and Paul Bizard的关于DCM的一篇经典论文《Direction Cosine Matrix IMU\_Theory》，一定要搞透这篇论文，没看过它都不敢称自己研究过飞控算法；然后接下来还有madgwick和mahony的论文需要研究，看英文的比较费时间，但是还是得慢慢的啃啊~~~

        然后就是结合国内的一本很不错的书籍《捷联惯性导航技术》，不需要细看，只需要了解其中关于姿态解算部分的即可。国内的课本嘛，大家都懂的，恨不得手把手的教你了。国外的论文就不一样了，继续啃吧。

**四、基本知识**

1、 陀螺仪和加速计（特性分析）

1）陀螺仪

         Gyro sensitivity、 operating range、offset、drift、calibrationandsaturation must be taken into account in the implementation of DCM。

灵敏度

        测量角速度，具有高动态特性，它是一个间接测量角度的器件其中一个关键参数就是gyro sensitivity（其单位是millivolts per degree persecond，把转速转换到电压值），测量范围越小气灵敏度越好。也就是说测量的是角度的导数，即角速度，要将角速度对时间积分才能得到角度。陀螺仪就是内部有一个陀螺，它的轴由于陀螺效应始终与初始方向平行，这样就可以通过与初始方向的偏差计算出旋转方向和角度。

偏移

        偏移就是在陀螺没有转动的时候却又输出，这个输出量的大小和供电电压以及温度有关，该偏移可以在陀螺仪上电时通过一小段时间的测量来修正。

漂移  
        它是由于在时间的积累下偏移和噪声相互影响的结果，例如有一个偏置（offset）0.1dps加在上面，于是测量出来是0.1dps，积分一秒之后，得到的角度是0.1度，1分钟之后是6度，还能忍受，一小时之后是360度，转了一圈，也就是说，陀螺仪在短时间内有很大的参考价值。  
白噪声  
        电信号的测量中，一定会带有白噪声，陀螺仪数据的测量也不例外。所以获得的陀螺仪数据中也会带有白噪声，而且这种白噪声会随着积分而累加。  
积分误差  
        对陀螺仪角速度的积分是离散的，长时间的积分会出现漂移的情况。所以要考虑积分误差的问题。

溢出

        就是转速超过了其测量的最大转速范围。关于这个问题的解决办法，在DCM IMU:Theory中给出来三种解决办法，不写了。

2）加速度计  
         加速度计的低频特性好，可以测量低速的静态加速度。在无人机上就是对重力加速度g的测量和分析。当把加速度计拿在手上随意转动时，看的是重力加速度在三个轴上的分量值。加速度计在自由落体时，其输出为0。为什么会这样呢？这里涉及到加速度计的设计原理：加速度计测量加速度是通过比力来测量(比力方程，秦永元的书中有介绍)，而不是通过加速度。加速度计仅仅测量的是重力加速度,而重力加速度与刚才所说的R坐标系（EarthFrame）是固连的,通过这种关系,可以得到加速度计所在平面与地面的角度关系。

        加速度计仅仅测量的是重力加速度，3轴加速度计输出重力加速度在加速度计所在机体坐标系3个轴上的分量大小。重力加速度的方向和大小是固定的。通过这种关系，可以得到加速度计所在平面与地面的角度关系。加速度计若是绕着重力加速度的轴转动，则测量值不会改变，也就是说加速度计无法感知这种水平旋转。

**Accelerometersare used for roll-pitch drift correction because they have zero drift**

        有关陀螺仪和加速度计和关系，姿态解算融合的原理，再把下面这个比喻放到这里一遍。  
        机体好似一条船，姿态就是航向（船头的方位），重力是灯塔，陀螺（角速度积分）是舵手，加速度计是瞭望手。舵手负责估计和把稳航向，他相信自己，本来船向北开的，就一定会一直往北开，觉得转了90度弯，那就会往东开。当然如果舵手很牛逼，也许能估计很准确，维持很长时间。不过只信任舵手，肯定会迷路，所以一般都有瞭望手来观察误差。  
        瞭望手根据地图灯塔方位和船的当前航向，算出灯塔理论上应该在船的X方位。然而看到实际灯塔在船的Y方位，那肯定船的当前航向有偏差了，偏差就是ERR=X-Y。舵手收到瞭望手给的ERR报告，觉得可靠，那就听个90%ERR，觉得天气不好、地图误差大，那就听个10%ERR，根据这个来纠正估算航向。

3）磁传感器

        可以测量磁场，在没有其他磁场的情况下，仅仅测量的是地球的磁场，而地磁也是和R坐标系固连的，通过这种关系，可以得到平面A和地平面的关系。(平面A：和磁场方向垂直的平面)，同样的，若是沿着磁场方向的轴旋转，测量值不会改变，无法感知这种旋转。

        综合考虑，加速度计和磁传感器都是极易受外部干扰的传感器，都只能得到2维的角度关系，但是测量值随时间的变化相对较小，结合加速度计和磁传感器可以得到3维的角度关系。陀螺仪可以积分得到三维的角度关系，动态性能好，受外部干扰小，但测量值随时间变化比较大。可以看出，它们优缺点互补，结合起来才能有好的效果。

4）关于数据融合

        现在有了三个传感器，都能在一定程度上测量角度关系，但是究竟相信谁？根据刚才的分析，应该是在短时间内更加相信陀螺仪，隔三差五的问问加速度计和磁传感器，角度飘了多少了？有一点必须非常明确，陀螺仪才是主角，加速度计和磁传感器仅仅是跑龙套的。其实加速度计无法对航向角进行修正，修正航向角需要磁力计。

参考crazypony的分析：<http://www.crazepony.com/wiki/mpu6050.html>和《DCM IMU:Theory》

**五、正文**

1、首先就是基于mahony算法的AHRS姿态解算的一套源码，理论基础是DCM IMU:Theory，很有参考价值。先自己分析一下，看看每一行具体是做什么的，是如何实现姿态解算和修正的。然后会给出相应的分析

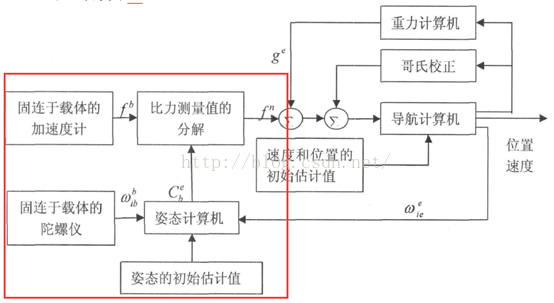
1. /\*
2. \*AHRS
3. \*/
4. // AHRS.c
5. // Quaternion implementation of the 'DCM filter' [Mayhony et al]. Incorporates the magnetic distortion
6. // compensation algorithms from my filter [Madgwick] which eliminatesthe need for a reference
7. // direction of flux (bx bz) to be predefined and limits the effect ofmagnetic distortions to yaw
8. // axis only.
9. // User must define 'halfT' as the (sample period / 2), and the filtergains 'Kp' and 'Ki'.
10. // Global variables 'q0', 'q1', 'q2', 'q3' are the quaternion elementsrepresenting the estimated
11. // orientation.  See my report foran overview of the use of quaternions in this application.
12. // User must call 'AHRSupdate()' every sample period and parsecalibrated gyroscope ('gx', 'gy', 'gz'),
13. // accelerometer ('ax', 'ay', 'ay') and magnetometer ('mx', 'my', 'mz')data.  Gyroscope units are
14. // radians/second, accelerometer and magnetometer units are irrelevantas the vector is normalised.
15. #include "stm32f10x.h"
16. #include "AHRS.h"
17. #include "Positioning.h"
18. #include <math.h>
19. #include <stdio.h>
20. /\* Private define------------------------------------------------------------\*/
21. #define Kp 2.0f                       // proportional gain governs rate of convergence toaccelerometer/magnetometer
22. #define Ki 0.005f          // integral gain governs rate of convergenceof gyroscope biases
23. #define halfT 0.0025f      // half the sample period
24. #define ACCEL\_1G 1000    //theacceleration of gravity is: 1000 mg
25. /\* Private variables---------------------------------------------------------\*/
26. static float q0 = 1, q1 = 0, q2 = 0, q3 = 0;        // quaternion elements representing theestimated orientation
27. static float exInt = 0, eyInt = 0, ezInt = 0;        // scaled integral error
28. /\* Public variables----------------------------------------------------------\*/
29. EulerAngle\_Type EulerAngle;       //unit: radian
30. u8InitEulerAngle\_Finished = 0;
31. float Magnetoresistor\_mGauss\_X = 0, Magnetoresistor\_mGauss\_Y = 0,Magnetoresistor\_mGauss\_Z = 0;//unit: milli-Gauss
32. float Accelerate\_mg\_X, Accelerate\_mg\_Y, Accelerate\_mg\_Z;//unit: mg
33. float AngularRate\_dps\_X, AngularRate\_dps\_Y, AngularRate\_dps\_Z;//unit:dps: degree per second
34. int16\_t Magnetoresistor\_X, Magnetoresistor\_Y, Magnetoresistor\_Z;
35. uint16\_t Accelerate\_X = 0, Accelerate\_Y = 0, Accelerate\_Z = 0;
36. uint16\_t AngularRate\_X = 0, AngularRate\_Y = 0, AngularRate\_Z = 0;
37. u8 Quaternion\_Calibration\_ok = 0;
38. /\* Private macro-------------------------------------------------------------\*/
39. /\* Private typedef-----------------------------------------------------------\*/
40. /\* Private function prototypes -----------------------------------------------\*/
41. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
42. \*Function Name  : AHRSupdate
43. \*Description    : None
44. \*Input          : None
45. \*Output         : None
46. \*Return         : None
47. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
48. void AHRSupdate(float gx, float gy, float gz, float ax, float ay, floataz, float mx, float my, float mz) {
49. float norm;
50. float hx, hy, hz, bx, bz;
51. float vx, vy, vz, wx, wy, wz;
52. float ex, ey, ez;
54. // auxiliary variables to reduce number of repeated operations
55. float q0q0 = q0\*q0;
56. float q0q1 = q0\*q1;
57. float q0q2 = q0\*q2;
58. float q0q3 = q0\*q3;
59. float q1q1 = q1\*q1;
60. float q1q2 = q1\*q2;
61. float q1q3 = q1\*q3;
62. float q2q2 = q2\*q2;
63. float q2q3 = q2\*q3;
64. float q3q3 = q3\*q3;
66. // normalise the measurements
67. norm = sqrt(ax\*ax + ay\*ay + az\*az);
68. ax = ax / norm;
69. ay = ay / norm;
70. az = az / norm;
71. norm = sqrt(mx\*mx + my\*my + mz\*mz);
72. mx = mx / norm;
73. my = my / norm;
74. mz = mz / norm;
76. // compute reference direction of magnetic field
77. hx = 2\*mx\*(0.5 - q2q2 - q3q3) + 2\*my\*(q1q2 - q0q3) + 2\*mz\*(q1q3 + q0q2);
78. hy = 2\*mx\*(q1q2 + q0q3) + 2\*my\*(0.5 - q1q1 - q3q3) + 2\*mz\*(q2q3 - q0q1);
79. hz = 2\*mx\*(q1q3 - q0q2) + 2\*my\*(q2q3 + q0q1) + 2\*mz\*(0.5 - q1q1 -q2q2);
80. bx = sqrt((hx\*hx) + (hy\*hy));
81. bz = hz;
83. // estimated direction of gravity and magnetic field (v and w)
84. vx = 2\*(q1q3 - q0q2);
85. vy = 2\*(q0q1 + q2q3);
86. vz = q0q0 - q1q1 - q2q2 + q3q3;
87. wx = 2\*bx\*(0.5 - q2q2 - q3q3) + 2\*bz\*(q1q3 - q0q2);
88. wy = 2\*bx\*(q1q2 - q0q3) + 2\*bz\*(q0q1 + q2q3);
89. wz = 2\*bx\*(q0q2 + q1q3) + 2\*bz\*(0.5 - q1q1 - q2q2);
91. // error is sum ofcross product between reference direction of fields and directionmeasured by sensors
92. ex = (ay\*vz - az\*vy) + (my\*wz - mz\*wy);
93. ey = (az\*vx - ax\*vz) + (mz\*wx - mx\*wz);
94. ez = (ax\*vy - ay\*vx) + (mx\*wy - my\*wx);
96. // integral error scaled integral gain
97. exInt = exInt + ex\*Ki\* (1.0f / sampleFreq);
98. eyInt = eyInt + ey\*Ki\* (1.0f / sampleFreq);
99. ezInt = ezInt + ez\*Ki\* (1.0f / sampleFreq);
100. // adjusted gyroscope measurements
101. gx = gx + Kp\*ex + exInt;
102. gy = gy + Kp\*ey + eyInt;
103. gz = gz + Kp\*ez + ezInt;
105. // integrate quaternion rate and normalize
106. q0 = q0 + (-q1\*gx - q2\*gy - q3\*gz)\*halfT;
107. q1 = q1 + (q0\*gx + q2\*gz - q3\*gy)\*halfT;
108. q2 = q2 + (q0\*gy - q1\*gz + q3\*gx)\*halfT;
109. q3 = q3 + (q0\*gz + q1\*gy - q2\*gx)\*halfT;
111. // normalise quaternion
112. norm = sqrt(q0\*q0 + q1\*q1 + q2\*q2 + q3\*q3);
113. q0 = q0 / norm;
114. q1 = q1 / norm;
115. q2 = q2 / norm;
116. q3 = q3 / norm;
117. }

2、上述算法的源码分析  
        先给一个简要的代码注释分析。[在CODE上查看代码片](https://code.csdn.net/snippets/1626113)

1. //陀螺仪、加速度计、磁力计数据融合
2. void AHRSupdate(float gx, float gy, float gz, float ax, float ay, float az, float mx, float my, float mz) {
3. float norm;
4. float hx, hy, hz, bx, bz;
5. float vx, vy, vz, wx, wy, wz; //v\*当前姿态计算得来的重力在三轴上的分量
6. float ex, ey, ez;
8. // auxiliary variables to reduce number of repeated operations
9. float q0q0 = q0\*q0;
10. float q0q1 = q0\*q1;
11. float q0q2 = q0\*q2;
12. float q0q3 = q0\*q3;
13. float q1q1 = q1\*q1;
14. float q1q2 = q1\*q2;
15. float q1q3 = q1\*q3;
16. float q2q2 = q2\*q2;
17. float q2q3 = q2\*q3;
18. float q3q3 = q3\*q3;
20. // normalise the measurements
21. norm = sqrt(ax\*ax + ay\*ay + az\*az);
22. ax = ax / norm;
23. ay = ay / norm;
24. az = az / norm;
25. norm = sqrt(mx\*mx + my\*my + mz\*mz);
26. mx = mx / norm;
27. my = my / norm;
28. mz = mz / norm;
30. // compute reference direction of magnetic field
31. hx = 2\*mx\*(0.5 - q2q2 - q3q3) + 2\*my\*(q1q2 - q0q3) + 2\*mz\*(q1q3 + q0q2);
32. hy = 2\*mx\*(q1q2 + q0q3) + 2\*my\*(0.5 - q1q1 - q3q3) + 2\*mz\*(q2q3 - q0q1);
33. hz = 2\*mx\*(q1q3 - q0q2) + 2\*my\*(q2q3 + q0q1) + 2\*mz\*(0.5 - q1q1 - q2q2);
34. bx = sqrt((hx\*hx) + (hy\*hy));
35. bz = hz;
37. // estimated direction of gravity and magnetic field (v and w)
38. //参考坐标n系转化到载体坐标b系的用四元数表示的方向余弦矩阵第三列即是。
39. //处理后的重力分量
40. vx = 2\*(q1q3 - q0q2);
41. vy = 2\*(q0q1 + q2q3);
42. vz = q0q0 - q1q1 - q2q2 + q3q3;
43. //处理后的mag，反向使用DCM得到
44. wx = 2\*bx\*(0.5 - q2q2 - q3q3) + 2\*bz\*(q1q3 - q0q2);
45. wy = 2\*bx\*(q1q2 - q0q3) + 2\*bz\*(q0q1 + q2q3);
46. wz = 2\*bx\*(q0q2 + q1q3) + 2\*bz\*(0.5 - q1q1 - q2q2);
48. // error is sum of cross product between reference direction of fields and direction measured by sensors 体现在加速计补偿和磁力计补偿，因为仅仅依靠加速计补偿没法修正Z轴的变差，所以还需要通过磁力计来修正Z轴。（公式28）。《四元数解算姿态完全解析及资料汇总》的作者把这部分理解错了，不是什么叉积的差，而叉积的计算就是这样的。计算方法是公式10。
49. ex = (ay\*vz - az\*vy) + (my\*wz - mz\*wy);
50. ey = (az\*vx - ax\*vz) + (mz\*wx - mx\*wz);
51. ez = (ax\*vy - ay\*vx) + (mx\*wy - my\*wx);
53. // integral error scaled integral gain
54. exInt = exInt + ex\*Ki\* (1.0f / sampleFreq);
55. eyInt = eyInt + ey\*Ki\* (1.0f / sampleFreq);
56. ezInt = ezInt + ez\*Ki\* (1.0f / sampleFreq);
57. // adjusted gyroscope measurements
58. //将误差PI后补偿到陀螺仪，即补偿零点漂移。通过调节Kp、Ki两个参数，可以控制加速度计修正陀螺仪积分姿态的速度。（公式16和公式29）
59. gx = gx + Kp\*ex + exInt;
60. gy = gy + Kp\*ey + eyInt;
61. gz = gz + Kp\*ez + ezInt;
63. // integrate quaternion rate and normalize
64. //一阶龙格库塔法更新四元数
65. q0 = q0 + (-q1\*gx - q2\*gy - q3\*gz)\*halfT;
66. q1 = q1 + (q0\*gx + q2\*gz - q3\*gy)\*halfT;
67. q2 = q2 + (q0\*gy - q1\*gz + q3\*gx)\*halfT;
68. q3 = q3 + (q0\*gz + q1\*gy - q2\*gx)\*halfT;
70. // normalise quaternion
71. norm = sqrt(q0\*q0 + q1\*q1 + q2\*q2 + q3\*q3);
72. q0 = q0 / norm;
73. q1 = q1 / norm;
74. q2 = q2 / norm;
75. q3 = q3 / norm;
76. }

3、在深入一点

        1）无人机控制中，主要就是姿态解算和姿态控制部分，该部分主要介绍姿态解算，下面是来一张比较好理解的框图。





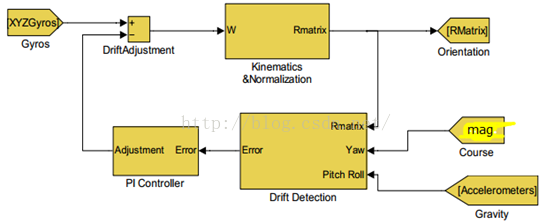
        由上面两幅图很容易理解整个控制和姿态解算了吧，在第四部分也介绍了关于陀螺仪、加速计、磁力计它们的作用，所以不再单独介绍了。不理解的往上翻翻再看看吧。

        2）在姿态解算过程中，到底用什么表示无人机的姿态呢？姿态表示的方法有很多种，比如欧拉角、四元数、DCM，各有的各的优势，比较常用的就是四元数，方便计算。上面的姿态解算算法也是基于四元数的。下面介绍一个它们三个的优缺点。

姿态解算方法的比较：

|  |  |  |  |
| --- | --- | --- | --- |
| 算法 | 优点 | 缺点 | PS |
| 欧拉角法（3参数） | 1、  通过欧拉角微分方程直接解算出pitch、roll、yaw  2、  概念直观，易于理解  3、  解算结果永远是正交的，无需再次正交化处理 | 1、  方程中有三角函数的运算，接超越函数有一定的困难  2、  当俯仰角接近90°时出现退化现象 | 1、  适应于水平姿态角变化不大的情况  2、  不适用于全姿态运载体 |
| 方向余弦法（9参数） | 1、  对姿态矩阵微分方程的求解，避免了欧拉角法中出现的退化现象  2、  可以全姿态运行 | 1、参数量过多，计算量大，实时性不好 | 很少在工程中使用 |
| 四元数法（4参数） | 1、  直接求解四元数微分方程  2、  只需要求解四个参数，计算量小  3、算法简单，易于操作 | 漂移比较严重 | 可以在实现过程中修正漂移，应用比较广泛 |

         3）废话不多说，进入正题。上述算法主要就是利用加速度计和磁力计修正陀螺仪的误差，算法是使用了PI反馈控制器实现反馈修正的。如下图：



        首先，东北天坐标系是n系（地理坐标系，参考坐标系），载体坐标系是b系，就是我们飞行器的坐标系。对于四元数法的姿态解算，需要求取的就是四元数的值；方向余弦矩阵（用于表示n系和b系的相对关系）中的元素本来应该是三角函数，这里由于使用四元数法，所以矩阵中的元素就成了四元数。所以姿态解算的任务就是求解由四元数构成的方向余弦矩阵nCb（nCb表示从b系到n的转换矩阵，同理，bCn表示从n系到b的转换矩阵，它们的关系是转置）。

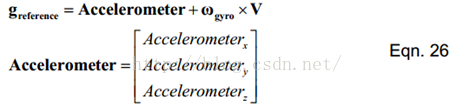
        显然，上述矩阵是有误差存在的。对于一个确定的向量n，用不同的坐标系表示时，他们所表示的大小和方向一定是相同的。但是由于这两个坐标系的转换矩阵存在误差，那么当一个向量经过这么一个有误差存在的旋转矩阵变换后，在另一个坐标系中肯定和理论值是有偏差的，我们通过这个偏差来修正这个旋转矩阵。这个旋转矩阵的元素是四元数，这就是说修正的就是四元数，于是乎姿态就这样被修正了，这才是姿态解算的原理。姿态解算求的是四元数，是通过修正旋转矩阵中的四元数来达到姿态解算的目的，而不要以为通过加速度计和地磁计来修正姿态，加速度计和地磁计只是测量工具和载体，通过这两个器件表征旋转矩阵的误差存在，然后通过算法修正误差，修正四元数，修正姿态。

加速度计修正pitch\_roll

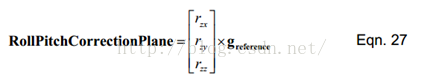
        加速度计可以修正pitch\_roll，但是我们必须要考虑离心加速度（centrifugalacceleration），离心加速度就等于旋转率向量（即gyro vector）和速度向量的叉积（没有原因，平均得来的并且还相当准确，速度可以用GPS获取）。我们假设飞机方向和X轴平行。



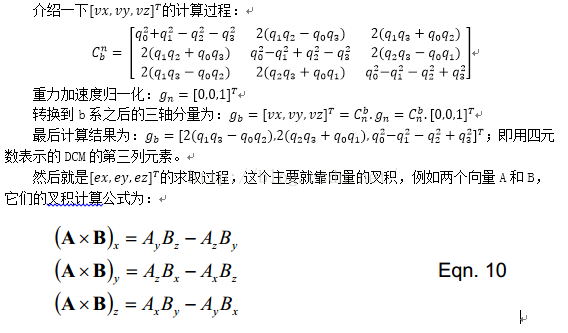
        在机体上测得的重力的为：



        Pitch\_roll的旋转修正向量是由DCM的Z行与归一化以后的重力参考向量的叉积。



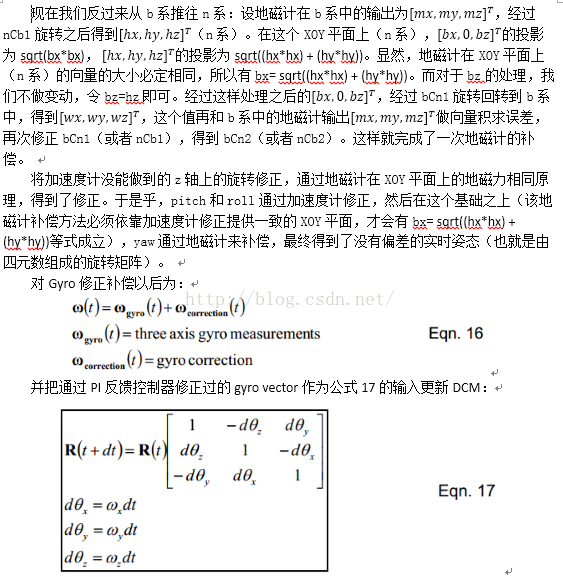
        在n系中，加速度计输出为，经过bCn（用四元数表示的转换矩阵）转换之后到b系中的值为；在b系中，加速度计的测量值为，现在和均表示在b系中的竖直向下的向量，由此，我们来做向量积（叉积），得到误差，利用这个误差来修正bCn矩阵，于是四元数就在这样一个过程中被修正了。但是，由于加速度计无法感知z轴上的旋转运动，所以还需要用地磁计来进一步补偿。



        PS：公式不好加，只能直接切图了~~~

        加速度计在静止时测量的是重力加速度，是有大小和方向的；同理，地磁计同样测量的是地球磁场的大小和方向，只不过这个方向不再是竖直向下，而是与x轴（或者y轴）呈一个角度，与z轴呈一个角度。记作，假设x轴对准北边，所以by=0，即。倘若知道bx和bz的精确值，那么就可以采用和加速度计一样的修正方法来修正。只不过在加速度计中，在n系中的参考向量是，变成了地磁计的。如果我们知道bx和bz的精确值，那么就可以摆脱掉加速度计的补偿，直接用地磁计和陀螺仪进行姿态解算，但是你看过谁只用陀螺仪和地磁计进行姿态解算吗？没有，因为没人会去测量当地的地磁场相对于东北天坐标的夹角，也就是bx和bz（插曲：关于这个bx和bz的理解：可以对比重力加速度的理解，就像vx vy vz似的，因为在每一处的归一化以后的重力加速度都是0 0 1然后旋转到机体坐标系，而地球每一处的地磁大小都不一样的，不能像重力加速度那样直接旋转得到了，只能用磁力计测量到的数据去强制拟合。）。那么现在怎么办？前面已经讲了，姿态解算就是求解旋转矩阵，这个矩阵的作用就是将b系和n正确的转化直到重合。现在我们假设nCb旋转矩阵是经过加速度计校正后的矩阵，当某个确定的向量（b系中）经过这个矩阵旋转之后（到n系），这两个坐标系在XOY平面上重合（参考DCM IMU:Theory的Drift cancellation部分），只是在z轴旋转上会存在一个偏航角的误差。下图表示的是经过nCb旋转之后的b系和n系的相对关系。可以明显发现加速度计可以把b系通过四元数法从任意角度拉到与n系水平的位置上，此时只剩下一个偏航角误差。这也是为什么加速度计无法修正偏航的原因。





        为什么在word中使用公式编译器编辑的公式没办法复制到CSDN中呢？？？？？？？？？？？？

        此方式在数学上没有任何问题，但是由于地磁传感器极易受到各种干扰，而此算法又将地磁传感器所指示的方向过度的融入到了姿态当中，导致实际使用中数据会非常不稳定。怪不得crazypony的算法中没有使用磁力计修正YAW，这应该就是原因所在吧。

**六、来一点ardupilot源码分析**

        就理解了一个通过gyro\_vector更新DCM的算法，这个应该是在renormalization直接就需要了解，可以前期没理解到底是干嘛的，现在补上；关于renormalization的算法可以参考上一篇博文。

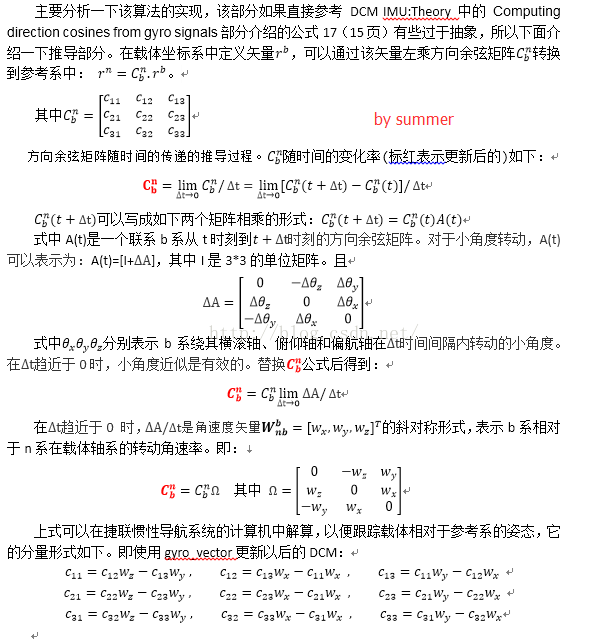
        姿态解算过程中不仅需要修正陀螺仪的各种errors，还需要实时的更新的DCM。上周一直没能理解的一个问题，在AP\_AHRS\_DCM.cpp中的matrix\_update(delta\_t)，就是实时更新DCM矩阵的，源码如下，这一部分研究了很久，需要的基础知识比较多。先上源码

1. // update the DCM matrix using only the gyros
2. Void AP\_AHRS\_DCM::matrix\_update(float \_G\_Dt)
3. {
4. \_omega.zero();
5. // average across first two healthy gyros. This reduces noise on
6. // systems with more than one gyro. We don't use the 3rd gyro
7. // unless another is unhealthy as 3rd gyro on PH2 has a lot more
8. // noise
9. uint8\_t healthy\_count = 0;
10. Vector3f delta\_angle;
11. for (uint8\_t i=0; i<\_ins.get\_gyro\_count(); i++) {
12. if (\_ins.get\_gyro\_health(i) && healthy\_count < 2) {
13. Vector3f dangle;
14. if (\_ins.get\_delta\_angle(i, dangle)) {
15. healthy\_count++;
16. delta\_angle += dangle;
17. }
18. }
19. }
20. if (healthy\_count > 1) {
21. delta\_angle /= healthy\_count;
22. }
23. if (\_G\_Dt > 0) {
24. \_omega = delta\_angle / \_G\_Dt;
25. \_omega += \_omega\_I;
26. \_dcm\_matrix.rotate((\_omega + \_omega\_P + \_omega\_yaw\_P) \* \_G\_Dt);
27. }
28. }

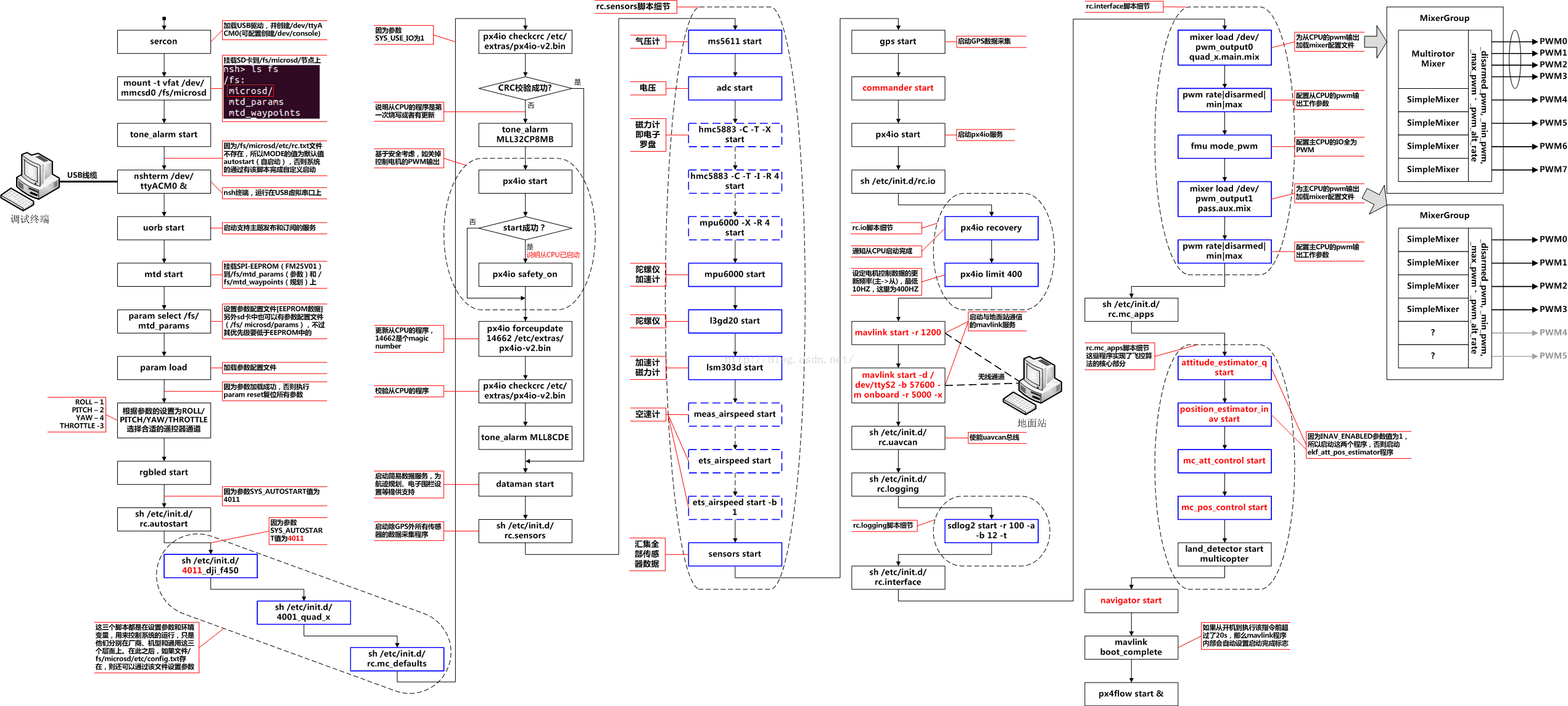
          首先就是通过陀螺仪获取两次gyro\_vector，然后取平均以降低误差；然后就是比较专业的算法实现\_dcm\_matrix.rotate((\_omega +\_omega\_P + \_omega\_yaw\_P) \* \_G\_Dt)了。进入到matrix3.cpp中有rotate()函数，该函数就是实现DCM更新的算法实现，算法主要是用陀螺仪的输出值与DCM矩阵的乘积再加回到DCM中去，处理过程中使用了离散化的概念，即dcm（k+1）=dcm（k）+增量，因为公式里有求导，必须离散化后才能计算机处理，感谢青龙的指导。

1. // apply an additional rotation from a body frame gyro vector
2. // to a rotation matrix.
3. template <typename T>
4. void Matrix3<T>::rotate(const Vector3<T> &g)
5. {
6. Matrix3<T> temp\_matrix;
7. temp\_matrix.a.x = a.y \* g.z - a.z \* g.y;
8. temp\_matrix.a.y = a.z \* g.x - a.x \* g.z;
9. temp\_matrix.a.z = a.x \* g.y - a.y \* g.x;
10. temp\_matrix.b.x = b.y \* g.z - b.z \* g.y;
11. temp\_matrix.b.y = b.z \* g.x - b.x \* g.z;
12. temp\_matrix.b.z = b.x \* g.y - b.y \* g.x;
13. temp\_matrix.c.x = c.y \* g.z - c.z \* g.y;
14. temp\_matrix.c.y = c.z \* g.x - c.x \* g.z;
15. temp\_matrix.c.z = c.x \* g.y - c.y \* g.x;
17. (\*this) += temp\_matrix;//增加累加到原始数据上
18. }

        公式太多，没办法还是上图吧~~~



        最后来一张神人做的图。



**六、总结**

        算法的理解过程比较艰难，千万不能闭门造车，要多多交流，思想的碰撞才能产生出火花，多谢各位群友的无私帮助。通过上述的分析以后，对整个姿态解算过程有了整体的框架理解，接下来会结合上面算法的分析去分析ardupilot中关于姿态解算的源码，应该会理解的快一点了吧，希望如此~~~  
        最近事情好多啊，愁死了，谁能帮我分担一点啊~~~~  
        昨天去三星见了一位智能家居group的总监，聊了以后才发现自己有多low啊，我的BLE怎么搞，大公司都在做垄断化、平台化，看来毕业只能硬着头皮进大公司了，希望能找个好工作，谁在外企啊，帮我介绍工作啊，啊啊， 啊， ，啊， 啊。。。。  
        写的语无伦次的，凑合看吧

### 姿态解算3\_源码姿态解算算法分析

**一、开篇**

        终于到ardupilot源代码的姿态解算了，有了前期关于mahony姿态解算算法的基础以后，理解源代码的姿态解算算法就快多了，所有的东西都在脑海中初步有了一个框架；首先要做什么，然后再做什么，再然后捏~~~反正容易上手的。

        2016.04.04日晚，别人都在嗨，而我却在实验室苦逼的工作着，今晚最大的收获就是发现了“新大陆”-----“北航可靠飞行控制研究组”，其喜悦之情绝不亚于哥伦布发现新大陆。他们才是专业的啊，看看他们毕业生的去向，不是研究所就是出国深造，好吧，人家才是专业搞科研的，不食人间烟火，那么问题来了：DJI的员工都是在哪招来的呢？！哎，我是俗人一个，经济基础决定上层建筑，好好学习才能挣大钱。最近他们开设了一门课程《多旋翼飞行器设计与控制》，课程体系安排的非常好，现在更新到第四讲了（听北航一个博士说只有PPT没有视频，感谢Mallin的帮助，成功打入内部），PPT也足够了，相当上档次啊，课程到2016.06.30结束，正好可以把无人机的整个架构理解完，找工作去~~~

        在下面的基础知识部分先分享一部分“北航”的研究成果，特别是气动方面的，以前一点概念都没有，只看着超跑的流线型非常炫酷，不知其原因，特此记录，大家共勉。

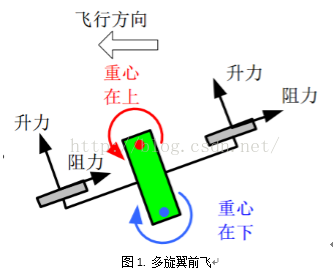
**四、基础知识（均来自北航可靠飞行控制研究组）**

1、无人机飞行的气动模型与分析

        1）多旋翼前飞情形：在下图中，因为螺旋桨的柔性，诱导的来流会产生阻力。

        如果多旋翼重心在桨盘平面下方，那么阻力形成的力矩会促使多旋翼俯仰角转向0度方向。

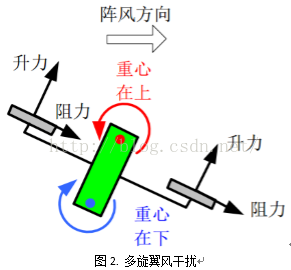
        如果多旋翼重心在桨盘平面上方，那么阻力形成的力矩会促使多旋翼俯仰角朝发散方向发展，直至翻转。因此，当多旋翼前飞时，重心在桨盘平面的下方会使前飞运动稳定。



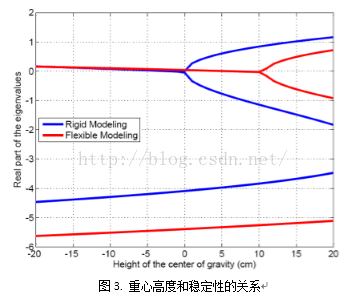
        2）多旋翼风干扰情形：在下图中，当阵风吹来，因为螺旋桨的柔性，诱导的来流会在产生阻力。

        如果多旋翼重心在下，那么阻力形成的力矩会促使多旋翼俯仰角朝发散的方向发展，直至翻转。

        如果多旋翼重心在上，那么阻力形成的力矩会促使多旋翼俯仰超0度方向发展。因此，当多旋翼受到外界风干扰时，重心在桨盘平面的上方可以抑制扰动。



        3）综上所述：无论重心在桨盘平面的上方或下方都不能使多旋翼稳定。因此需要通过反馈控制将多旋翼平衡。然而，如果重心在桨盘平面很靠上的位置，会使多旋翼某个运动模态很不稳定。因此，实际中建议将重心配置在飞行器桨盘周围，可以稍微靠下。这样控制器控制起来更容易些。



2、气动布局

        对外形进行设计主要是为了降低飞行时的阻力。按其产生的原因不同可分为

（1）摩擦阻力

（2）压差阻力

（3）诱导阻力

（4）干扰阻力

        要减少这些阻力，需要妥善考虑和安排各部件之间的相对位置关系，部件连接处尽量圆滑过渡，减少漩涡产生。



        因此它与物体的迎风面积有很大关系，迎风面积越大，压差阻力也越大。物体的形状也对压差阻力影响很大。如上图所示的三个物体，圆盘的压差阻力最大，球体次之，而流线体的最小，就压差阻力而言可以是平板压差阻力的1/20。

        设计建议：（法拉利、保驰捷等超跑的流线型车就是很好的榜样，宝马 Z4也可以，奔驰Smart就太low了，差点忘记特斯拉了）

（1）需要考虑多旋翼前飞时的倾角，减少最大迎风面积。

（2）并设计流线型机身。

（3）考虑和安排各部件之间的相对位置关系，部件连接处尽量圆滑过渡，飞机表面也要尽量光滑。

（4）通过CFD仿真（Computational Fluid Dynamics：计算流体动力学）计算阻力系数，不断优化。



        昨天见到了零度的四旋翼，外形设计的就是好，可惜只靠外形还是干不过DJI的，加油吧，零度。

**五、正文（源代码的姿态解算算法）**

1、进程入口：voidAttitudeEstimatorQ::task\_main()

        1）订阅所需要的topics，注意sensor\_combined，传感器数据都是靠它来的。

        在订阅时使用ORB\_ID(sensor\_combined)获取ID号，该ID号代表了从topic\_name到topicmetadata structure name之间的转换桥梁。在task\_main()的初始部分使用uORB模型的orb\_subscribe()函数获取在sensors.cpp中通过orb\_advertise()函数广播的传感器信息（sensor\_combined）。由此说明在使用之前需要通过orb\_advertise()函数之后才能在需要其数据的地方使用orb\_subscribe()获取。如果没有使用，订阅者可以订阅，但是接收不到有效数据。

        关于uOBR模型的不再赘述，详细介绍参看：<http://blog.csdn.net/freeape/article/details/46880637>

和<http://www.pixhawk.com/start?id=zh%2Fdev%2Fshared_object_communication&go>

        px4\_poll(fds,1, 1000)：配置阻塞时间，1ms读取一次sensor\_combined的数据。

        必须有了orb\_advertise()函数和orb\_subscribe()函数（通过它获取orb\_copy()函数中的参数handle）之后才可以使用orb\_copy(ORB\_ID(sensor\_combined),\_sensors\_sub, &sensors)函数获取sensors的实际有效数据。

        首先是读取gyro的数据：

1. float gyro[3];
2. for (unsigned j = 0; j < 3; j++)
3. {
4. if (sensors.gyro\_integral\_dt[i] > 0)
5. {
6. gyro[j] = (double)sensors.gyro\_integral\_rad[i \* 3 + j] / (sensors.gyro\_integral\_dt[i] / 1e6);
7. }
8. else{
9. /\* fall back to angular rate \*/
10. gyro[j] = sensors.gyro\_rad\_s[i \* 3 + j];
11. }
12. }

        然后以同样的方法读取accel和mag的数据。

1. Void DataValidatorGroup::put(unsigned index, uint64\_t timestamp, float val[3], uint64\_t error\_count, int priority)
2. {
3. DataValidator \*next = \_first;
4. unsigned i = 0;
5. while (next != nullptr) {
6. if (i == index) {
7. next->put(timestamp, val, error\_count, priority);//goto
8. break;
9. }
10. next = next->sibling();//下一组数据
11. i++;
12. }
13. }

        最终还是goto到put函数。

1. Void DataValidator::put(uint64\_t timestamp, float val[3], uint64\_t error\_count\_in, int priority\_in)
2. {
3. \_event\_count++;
4. if (error\_count\_in > \_error\_count) {
5. \_error\_density += (error\_count\_in - \_error\_count);
6. } else if (\_error\_density > 0) {
7. \_error\_density--;
8. }
9. \_error\_count = error\_count\_in;
10. \_priority = priority\_in;
11. for (unsigned i = 0; i < \_dimensions; i++) {//\_dimensions=3
12. if (\_time\_last == 0) {//时间变量已经初始化为0
13. \_mean[i] = 0;
14. \_lp[i] = val[i];
15. \_M2[i] = 0;
16. } else {
17. float lp\_val = val[i] - \_lp[i];
18. float delta\_val = lp\_val - \_mean[i];
19. \_mean[i] += delta\_val / \_event\_count;
20. \_M2[i] += delta\_val \* (lp\_val - \_mean[i]);
21. \_rms[i] = sqrtf(\_M2[i] / (\_event\_count - 1));
22. if (fabsf(\_value[i] - val[i]) < 0.000001f) {
23. \_value\_equal\_count++;
24. } else {
25. \_value\_equal\_count = 0;
26. }
27. }
28. // XXX replace with better filter, make it auto-tune to update rate
29. \_lp[i] = \_lp[i] \* 0.5f + val[i] \* 0.5f;
30. \_value[i] = val[i];
31. }
32. \_time\_last = timestamp;
33. }

        详细介绍使用方法：主要是将三类参数分别建立相应的 DataValidatorGroup类来对数据进行处理。

        DataValidatorGroup类： \_voter\_gyro、\_voter\_accel、\_voter\_mag

         调用方法：

         1）首先gyro、accel、mag每次读取数据都是三组三组的读取

         2）先将每组的数据（例如gyro将三个维度的的传感器数据put入（如\_voter\_gyro.put(...))）DataValidatorGroup中，并goto到DataValidator::put函数

         3）在DataValidator函数中计算数据的误差、平均值、并进行滤波。

    滤波入口的put函数：

         val=传感器读取的数据

         \_lp=滤波器的系数（lowpass value）

         初始化：由上图可知当第一次读到传感器数据时\_mean和\_M2置0，\_lp=val；

         lp\_val= val - \_lp

         delta\_val= lp\_val - \_mean

         \_mean= (平均值）每次数据读取时，每次val和\_lp的差值之和的平均值 \_mean[i] += delta\_val / \_event\_count

         \_M2= （均方根值）delta\_val \* (lp\_val - \_mean)的和

         \_rms= 均方根值sqrtf(\_M2[i] / (\_event\_count - 1))

         优化滤波器系数：\_lp[i]= \_lp[i] \* 0.5f + val[i] \* 0.5f

         \_value= val ：get\_best()函数的最后调用该结果(通过比较三组数据的confidence大小决定是否选取）。

    滤波器的confidence函数（信任度函数，貌似模糊控制理论有个隶属函数，应该类似的功能）：返回值是对上N次测量的验证的信任程度，其值在0到1之间，越大越好。返回值是返回上N次测量的误差诊断，用于get\_best函数选择最优值，选择的方法如下：

 Switch if:

         1)the confidence is higher and priority is equal or higher

         2)the confidence is no less than 1% different and the priority is higher

2、根据\_voter\_gyro、\_voter\_accel、\_voter\_mag三个参数的failover\_count函数判断是否存在数据获取失误问题，并通过mavlink协议显示错误原因。

3、根据\_voter\_gyro、\_voter\_accel、\_voter\_mag三个参数的get\_vibration\_factor函数判断是否有震动现象，返回值是float型的RSM值，其代表振动的幅度大小。

1. Float DataValidatorGroup::get\_vibration\_factor(uint64\_t timestamp)
2. {
3. DataValidator \*next = \_first;
4. float vibe = 0.0f;
5. /\* find the best RMS value of a non-timed out sensor \*/
6. while (next != nullptr) {
7. if (next->confidence(timestamp) > 0.5f) {
8. float\* rms = next->rms();
9. for (unsigned j = 0; j < 3; j++) {
10. if (rms[j] > vibe) {
11. vibe = rms[j];//获取最大值
12. }
13. }
14. }
15. next = next->sibling();
16. }
17. return vibe;//返回DataValidatorGroup中的三组中的最大的振动值
18. }

        将rms变量（原来put函数中的\_rms）传回主函数中和\_vibration\_warning\_threshold进行判断。

4、通过uORB模型获取vision和mocap的数据（视觉和mocap）

1. // Update vision and motion capture heading
2. bool vision\_updated = false;
3. orb\_check(\_vision\_sub, &vision\_updated);
4. bool mocap\_updated = false;
5. orb\_check(\_mocap\_sub, &mocap\_updated);
6. if (vision\_updated) {
7. orb\_copy(ORB\_ID(vision\_position\_estimate), \_vision\_sub, &\_vision);
8. math::Quaternion q(\_vision.q);
9. math::Matrix<3, 3> Rvis = q.to\_dcm();
10. math::Vector<3> v(1.0f, 0.0f, 0.4f);
11. // Rvis is Rwr (robot respect to world) while v is respect to world.
12. // Hence Rvis must be transposed having (Rwr)' \* Vw
13. // Rrw \* Vw = vn. This way we have consistency
14. \_vision\_hdg = Rvis.transposed() \* v;
15. }
16. if (mocap\_updated) {
17. orb\_copy(ORB\_ID(att\_pos\_mocap), \_mocap\_sub, &\_mocap);
18. math::Quaternion q(\_mocap.q);
19. math::Matrix<3, 3> Rmoc = q.to\_dcm();
20. math::Vector<3> v(1.0f, 0.0f, 0.4f);
21. // Rmoc is Rwr (robot respect to world) while v is respect to world.
22. // Hence Rmoc must be transposed having (Rwr)' \* Vw
23. // Rrw \* Vw = vn. This way we have consistency
24. \_mocap\_hdg = Rmoc.transposed() \* v;
25. }</span><span style="font-size:14px;">

5、位置加速度获取（position，注意最后在修正时会用到该处的\_pos\_acc）（484）

        首先检测是否配置了自动磁偏角获取，如果配置了则用当前的经纬度（[longitude and latitude](http://www.baidu.com/link?url=DU4aaxqzQ5yYjLR6rqfDNZU44kVHMm1EvsUAz80M_j0mKsrh8HJpxDKMYxVlUwtsQoYX4Q2ugZfJECsprukOR8Ya3893HlkA_nqeHC6KEs0uor49SnB9tNbiCYnYLO8D)）通过get\_mag\_declination(\_gpos.lat,\_gpos.lon)函数获取当前位置的磁偏角（magnetic declination），实现过程就是一系列的算算算，自己跟踪源码看吧，用地面站校准罗盘的应该比较熟悉这个。然后就是获取机体的速度，通过速度计算机体的加速度。

1. bool gpos\_updated;
2. orb\_check(\_global\_pos\_sub, &gpos\_updated);
3. if (gpos\_updated) {
4. orb\_copy(ORB\_ID(vehicle\_global\_position), \_global\_pos\_sub, &\_gpos);
5. if (\_mag\_decl\_auto && \_gpos.eph < 20.0f && hrt\_elapsed\_time(&\_gpos.timestamp) < 1000000) {
6. /\* set magnetic declination automatically \*/
7. \_mag\_decl = math::radians(get\_mag\_declination(\_gpos.lat, \_gpos.lon));
8. }
9. }
10. if (\_acc\_comp && \_gpos.timestamp != 0 && hrt\_absolute\_time() < \_gpos.timestamp + 20000 && \_gpos.eph < 5.0f && \_inited) {
11. /\* position data is actual \*/
12. if (gpos\_updated) {
13. Vector<3> vel(\_gpos.vel\_n, \_gpos.vel\_e, \_gpos.vel\_d);
14. /\* velocity updated \*/
15. if (\_vel\_prev\_t != 0 && \_gpos.timestamp != \_vel\_prev\_t) {
16. float vel\_dt = (\_gpos.timestamp - \_vel\_prev\_t) / 1000000.0f;
17. /\* calculate acceleration in body frame ：速度之差除时间\*/
18. \_pos\_acc = \_q.conjugate\_inversed((vel - \_vel\_prev) / vel\_dt);
19. }
20. \_vel\_prev\_t = \_gpos.timestamp;
21. \_vel\_prev = vel;
22. }
23. } else {
24. /\* position data is outdated, reset acceleration \*/
25. \_pos\_acc.zero();
26. \_vel\_prev.zero();
27. \_vel\_prev\_t = 0;
28. }

6、update函数（528行）

        接下来就是528行的updata函数了，非常重要，这个函数主要用于对四元数向量\_q进行初始化赋值或者更新。

        --------------------------------------------------------------------------------------------------------------------------

        首先判断是否是第一次进入该函数，第一次进入该函数先进入init函数初始化，源码如下。

1. bool AttitudeEstimatorQ::init()
2. {
3. // Rotation matrix can be easily constructed from acceleration and mag field vectors
4. // 'k' is Earth Z axis (Down) unit vector in body frame
5. Vector<3> k = -\_accel;
6. k.normalize();
7. // 'i' is Earth X axis (North) unit vector in body frame, orthogonal with 'k'
8. Vector<3> i = (\_mag - k \* (\_mag \* k));
9. i.normalize();
10. // 'j' is Earth Y axis (East) unit vector in body frame, orthogonal with 'k' and 'i'
11. Vector<3> j = k % i;
12. // Fill rotation matrix
13. Matrix<3, 3> R;
14. R.set\_row(0, i);
15. R.set\_row(1, j);
16. R.set\_row(2, k);
17. // Convert to quaternion
18. \_q.from\_dcm(R);
19. \_q.normalize();
20. if (PX4\_ISFINITE(\_q(0)) && PX4\_ISFINITE(\_q(1)) &&
21. PX4\_ISFINITE(\_q(2)) && PX4\_ISFINITE(\_q(3)) &&
22. \_q.length() > 0.95f && \_q.length() < 1.05f) {
23. \_inited = true;
24. } else {
25. \_inited = false;
26. }
27. return \_inited;
28. }

        初始化方法：

        k= -\_accel 然后归一化k，k为加速度传感器测量到加速度方向向量，由于第一次测量数据时无人机一般为平稳状态无运动状态，所以可以直接将测到的加速度视为重力加速度g，以此作为dcm旋转矩阵的第三行k（这个介绍过了）。

        i= （\_mag - k \* (\_mag \* k)) \_mag向量指向正北方，k\*(\_mag\*k) 正交correction值，对于最终的四元数归一化以后的范数可以在正负5%以内；感觉不如《DCM IMU:Theory》中提出的理论“强制正交化”修正的好，Renormalization算法在ardupilot的上层应用AP\_AHRS\_DCM中使用到了。

        j= k%i ：外积、叉乘。关于上面的Vector<3>k = -\_accel，Vector<3>相当于一个类型（int）定义一个变量k，然后把-\_accel负值给k，在定义\_accel时也是使用Vector<3>，属于同一种类型的，主要就是为了考虑其实例化过程（类似函数重载）。

        然后以模板的形式使用“Matrix<3, 3> R”建立3x3矩阵R，通过set\_row()函数赋值。

1. /\*\*
2. \* set row from vector
3. \*/
4. void set\_row(unsigned int row, const Vector<N> v) {
5. for (unsigned i = 0; i < N; i++) {
6. data[row][i] = v.data[i];
7. }
8. }

        第一行赋值i 向量指向北，第二行赋值j 向量指向东，第三行赋值k向量指向DOWN。然后就是把DCM转换为四元数\_q （使用from\_dcm），并归一化四元数，一定要保持归一化的思想。来一个比较牛掰的求范数的倒数的快速算法（mahony的算法实现用到的）：

1. float invSqrt(float x) {
2. float halfx = 0.5f \* x;
3. float y = x;
4. long i = \*(long\*)&y;
5. i = 0x5f3759df - (i>>1);
6. y = \*(float\*)&i;
7. y = y \* (1.5f - (halfx \* y \* y));
8. return y;
9. }

        具体为什么这么实现的还是看wiki连接吧：<https://en.wikipedia.org/wiki/Fast_inverse_square_root>

        其中DCM转四元数的算法如下，tr>0时比较好理解，别的情况被简写的看不透了。后续给出原始代码便于理解

1. tr = dcm.data[0][0] + dcm.data[1][1] + dcm.data[2][2]
2. 如果 tr>0
3. s=(tr+1)^0.5  \_q[0]= 0.5\*s
4. \_q[1]=(dcm(3,2)-dcm(2,3))\*s
5. \_q[2]=(dcm(1,3)-dcm(3,1))\*s
6. \_q[3]=(dcm(2,1)-dcm(1,2))\*s

        其他情况去看代码吧，不好解释。然后\_q 单位化结束初始化。PS：另外根据DCM求取四元数的算法是参考MartinJohn Baker的，就是上述的原始版，该算法在AP\_Math/quaternion.cpp中，链接：

<http://www.euclideanspace.com/maths/geometry/rotations/conversions/matrixToQuaternion/index.htm>

        源码如下。

1. void Quaternion::from\_rotation\_matrix(const Matrix3f &m)
2. {
3. const float &m00 = m.a.x;
4. const float &m11 = m.b.y;
5. const float &m22 = m.c.z;
6. const float &m10 = m.b.x;
7. const float &m01 = m.a.y;
8. const float &m20 = m.c.x;
9. const float &m02 = m.a.z;
10. const float &m21 = m.c.y;
11. const float &m12 = m.b.z;
12. float &qw = q1;
13. float &qx = q2;
14. float &qy = q3;
15. float &qz = q4;
16. float tr = m00 + m11 + m22;
17. if (tr > 0) {
18. float S = sqrtf(tr+1) \* 2;
19. qw = 0.25f \* S;
20. qx = (m21 - m12) / S;
21. qy = (m02 - m20) / S;
22. qz = (m10 - m01) / S;
23. } else if ((m00 > m11) && (m00 > m22)) {
24. float S = sqrtf(1.0f + m00 - m11 - m22) \* 2;
25. qw = (m21 - m12) / S;
26. qx = 0.25f \* S;
27. qy = (m01 + m10) / S;
28. qz = (m02 + m20) / S;
29. } else if (m11 > m22) {
30. float S = sqrtf(1.0f + m11 - m00 - m22) \* 2;
31. qw = (m02 - m20) / S;
32. qx = (m01 + m10) / S;
33. qy = 0.25f \* S;
34. qz = (m12 + m21) / S;
35. } else {
36. float S = sqrtf(1.0f + m22 - m00 - m11) \* 2;
37. qw = (m10 - m01) / S;
38. qx = (m02 + m20) / S;
39. qy = (m12 + m21) / S;
40. qz = 0.25f \* S;
41. }
42. }

        ---------------------------------------------------------------------------------------------------------------------------------

        如果不是第一次进入该函数，则判断是使用什么mode做修正的，比如vision、mocap、acc和mag（DJI精灵4用的视觉壁障应该就是这个vision），Hdg就是heading。

1. if (\_ext\_hdg\_mode > 0 && \_ext\_hdg\_good) {
2. if (\_ext\_hdg\_mode == 1) {
3. // Vision heading correction
4. // Project heading to global frame and extract XY component
5. Vector<3> vision\_hdg\_earth = \_q.conjugate(\_vision\_hdg);
6. float vision\_hdg\_err = \_wrap\_pi(atan2f(vision\_hdg\_earth(1), vision\_hdg\_earth(0)));
7. // Project correction to body frame
8. corr += \_q.conjugate\_inversed(Vector<3>(0.0f, 0.0f, -vision\_hdg\_err)) \* \_w\_ext\_hdg;
9. }
10. if (\_ext\_hdg\_mode == 2) {
11. // Mocap heading correction
12. // Project heading to global frame and extract XY component
13. Vector<3> mocap\_hdg\_earth = \_q.conjugate(\_mocap\_hdg);
14. float mocap\_hdg\_err = \_wrap\_pi(atan2f(mocap\_hdg\_earth(1), mocap\_hdg\_earth(0)));
15. // Project correction to body frame
16. corr += \_q.conjugate\_inversed(Vector<3>(0.0f, 0.0f, -mocap\_hdg\_err)) \* \_w\_ext\_hdg;
17. }
18. }

        \_ext\_hdg\_mode== 1、2时都是利用vision数据和mocap数据对gyro数据进行修正，下面的global frame就是所谓的earthframe。

         ---------------------------------------------------------------------------------------------------------------------------------

**\_ext\_hdg\_mode== 0利用磁力计修正。**

1. if (\_ext\_hdg\_mode == 0  || !\_ext\_hdg\_good) {
2. // Magnetometer correction
3. // Project mag field vector to global frame and extract XY component
4. Vector<3> mag\_earth = \_q.conjugate(\_mag); //b系到n系
5. float mag\_err = \_wrap\_pi(atan2f(mag\_earth(1), mag\_earth(0)) - \_mag\_decl);
6. // Project magnetometer correction to body frame
7. corr += \_q.conjugate\_inversed(Vector<3>(0.0f, 0.0f, -mag\_err)) \* \_w\_mag; //n系到b系
8. }

        \_w\_mag为mag的权重。PS：发现一个规律，在不太理解C++的情况下看代码的算法过程中经常性的不知道某行代码是做什么的，在哪定义的，比如这个Vector<3>，前面已经介绍过它了，只要有它这样的做前缀的，后面的变量就是类似int定义一个变量一样，几乎都在PX4Firmware/src/lib/mathlib/math/Quaternion.hpp中，进行实例展开的。磁力计数据为\_mag。mag\_earth= \_q.conjugate(\_mag)，这行代码的含义为先将归一化的\_q的旋转矩阵R（b系转n系）乘以\_mag向量（以自身机体坐标系为视角看向北方的向量表示），得到n系（NED坐标系）下的磁力计向量。如下就是conjugate函数，其过程就是实现从b系到n系的转换，使用左乘。

1. /\*\*
2. \* conjugation//b系到n系
3. \*/
4. Vector<3> conjugate(const Vector<3> &v) const {
5. float q0q0 = data[0] \* data[0];
6. float q1q1 = data[1] \* data[1];
7. float q2q2 = data[2] \* data[2];
8. float q3q3 = data[3] \* data[3];
9. return Vector<3>(
10. v.data[0] \* (q0q0 + q1q1 - q2q2 - q3q3) +
11. v.data[1] \* 2.0f \* (data[1] \* data[2] - data[0] \* data[3]) +
12. v.data[2] \* 2.0f \* (data[0] \* data[2] + data[1] \* data[3]),
14. v.data[0] \* 2.0f \* (data[1] \* data[2] + data[0] \* data[3]) +
15. v.data[1] \* (q0q0 - q1q1 + q2q2 - q3q3) +
16. v.data[2] \* 2.0f \* (data[2] \* data[3] - data[0] \* data[1]),
18. v.data[0] \* 2.0f \* (data[1] \* data[3] - data[0] \* data[2]) +
19. v.data[1] \* 2.0f \* (data[0] \* data[1] + data[2] \* data[3]) +
20. v.data[2] \* (q0q0 - q1q1 - q2q2 + q3q3)
21. );
22. }
23. mag\_err = \_wrap\_pi(atan2f(mag\_earth(1), mag\_earth(0)) - \_mag\_decl);

        只考虑Vector<3> mag\_earth中的前两维的数据mag\_earth(1)和mag\_earth(0)（即x、y，忽略z轴上的偏移），通过arctan（mag\_earth(1),mag\_earth(0)）得到的角度和前面根据经纬度获取的磁偏角做差值得到纠偏误差角度mag\_err ，\_wrap\_pi函数是用于限定结果-pi到pi的函数，源码如下。

1. \_\_EXPORT float \_wrap\_pi(float bearing)
2. {
3. /\* value is inf or NaN \*/
4. if (!isfinite(bearing)) {
5. return bearing;
6. }
7. int c = 0;
8. //大于pi则与2pi做差取补
9. while (bearing >= M\_PI\_F) {//M\_PI\_F==3.14159265358979323846f
10. bearing -= M\_TWOPI\_F;//M\_TWOPI\_F==2\*M\_PI\_F
11. if (c++ > 3) {
12. return NAN;//NaN==not a number
13. }
14. }
15. c = 0;
16. //小于-pi则与2pi做和取补
17. while (bearing < -M\_PI\_F) {
18. bearing += M\_TWOPI\_F;
19. if (c++ > 3) {
20. return NAN;
21. }
22. }
23. return bearing;
24. }
25. 类似的约束函数都在src/lib/geo/geo.c中，比如：
26. \_\_EXPORT float \_wrap\_180(float bearing);
27. \_\_EXPORT float \_wrap\_360(float bearing);
28. \_\_EXPORT float \_wrap\_pi(float bearing);
29. \_\_EXPORT float \_wrap\_2pi(float bearing);

         corr +=\_q.conjugate\_inversed(Vector<3>(0.0f, 0.0f, -mag\_err)) \* \_w\_mag;//n系到b系

         计算corr值等于单位化的旋转矩阵R（b系转n系）的转置（可以理解为 R（n系转b系））乘以（0,0，-mag\_err），相当于机体坐标系绕地理坐标系N轴（Z轴）转动arctan（mag\_earth(1), mag\_earth(0)）度。

         关于磁还需要更深入的了解，看相关论文吧，一大推~~~

         ------------------------------------------------------------------------------------------------------------------------------

        加速度计修正

        首先就是把归一化的n系重力加速度通过旋转矩阵R左乘旋转到b系，即k为归一化的旋转矩阵R（b-e）的第三行，代码如下。

1. // Accelerometer correction
2. // Project 'k' unit vector of earth frame to body frame
3. // Vector<3> k = \_q.conjugate\_inversed(Vector<3>(0.0f, 0.0f, 1.0f));// n系到b系
4. // Optimized version with dropped zeros
5. Vector<3> k(
6. 2.0f \* (\_q(1) \* \_q(3) - \_q(0) \* \_q(2)),
7. 2.0f \* (\_q(2) \* \_q(3) + \_q(0) \* \_q(1)),
8. (\_q(0) \* \_q(0) - \_q(1) \* \_q(1) - \_q(2) \* \_q(2) + \_q(3) \* \_q(3))
9. );

        上面这段代码是不是很熟悉，还记得mahony算法中的的计算过程么？！都是一样的，这里只是换种方式（C++）表达，不在赘述。

        下面这个比较重要：

**corr += (k %(\_accel - \_pos\_acc).normalized()) \* \_w\_accel。**

        {k%（\_accel“加速度计的测量值”-位移加速度）的单位化）<约等于重力加速度g>}\*权重。

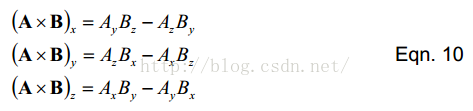
        关于这个“\_accel-\_pos\_acc”的含义：

        总的受到合力的方向（\_accel）减去机体加速度方向（\_pos\_acc）得到g的方向，即总加速度（加速度获取）减去机体运动加速度（第五部分）获取重力加速度，然后姿态矩阵的不是行就是列来与纯重力加速度来做叉积，算出误差。因为运动加速度是有害的干扰，必须减掉。算法的理论基础是[0,0,1]与姿态矩阵相乘。该差值获取的重力加速度的方向是导航坐标系下的z轴，加上运动加速度之后，总加速度的方向就不是与导航坐标系的天或地平行了，所以要消除这个误差，即“\_accel-\_pos\_acc”。然后叉乘z轴向量得到误差，进行校准 。

        关于%运算符在vector.hpp中介绍了其原型：

1. Vector<3> operator %(const Vector<3> &v) const {
2. return Vector<3>(
3. data[1] \* v.data[2] - data[2] \* v.data[1],
4. data[2] \* v.data[0] - data[0] \* v.data[2],
5. data[0] \* v.data[1] - data[1] \* v.data[0]
6. );
7. }
8. };

       “ %”其实就是求向量叉积，叉积公式如下。



        ---------------------------------------------------------------------------------------------------------------------------------

        \_gyro\_bias+= corr \* (\_w\_gyro\_bias \* dt);PI控制器中的I（积分）效果。然后对\_gyro\_bias做约束处理。

1. for (int i = 0; i < 3; i++) {
2. \_gyro\_bias(i) = math::constrain(\_gyro\_bias(i), -\_bias\_max, \_bias\_max);
3. }

        对偏移量进行约束处理，源码写的相当好啊，简单易懂，其函数原型是：

1. uint64\_t \_\_EXPORT constrain(uint64\_t val, uint64\_t min, uint64\_t max)
2. {
3. return (val < min) ? min : ((val > max) ? max : val);
4. }

        ---------------------------------------------------------------------------------------------------------------------------------

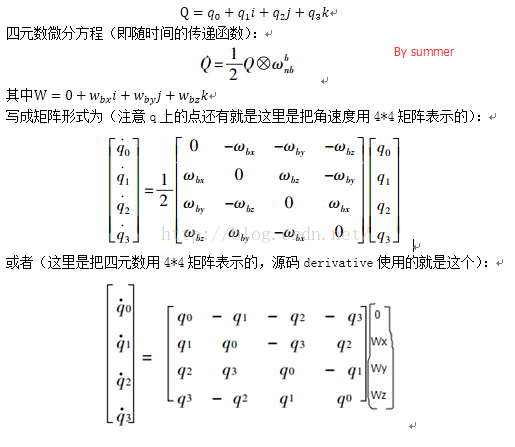
        最后就是使用修正的数据更新四元数，并把\_rates和\_gyro\_bias置零便于下次调用时使用。

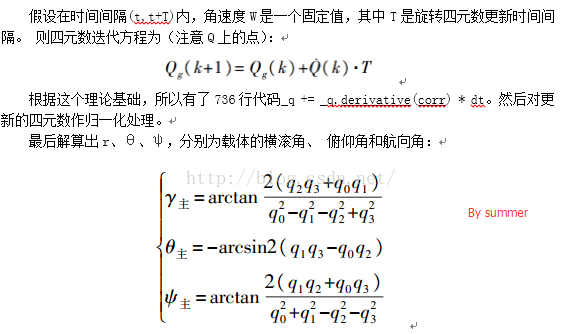
1. \_rates = \_gyro + \_gyro\_bias; //得到经过修正后的角速度
2. // Feed forward gyro
3. corr += \_rates;//PI控制器的体现
4. // Apply correction to state
5. \_q += \_q.derivative(corr) \* dt;//736
6. // Normalize quaternion
7. \_q.normalize();
8. if (!(PX4\_ISFINITE(\_q(0)) && PX4\_ISFINITE(\_q(1)) &&
9. PX4\_ISFINITE(\_q(2)) && PX4\_ISFINITE(\_q(3)))) {
10. // Reset quaternion to last good state
11. \_q = q\_last;
12. \_rates.zero();
13. \_gyro\_bias.zero();
14. return false;
15. }
16. return true;

        上面736行的\_q += \_q.derivative(corr) \* dt非常重要，又用到了微分方程离散化的思想。以前讲过DCM矩阵更新过程中也是用到了该思想。先看看代码，有点怪，怪就怪在derivative（衍生物）这个名字上，平时一大推的论文和期刊上面都是用的omga \*Q 的形式，而这里的代码实现确是用的Q \* omga的形式，所以构造的4\*4矩阵的每一列的符号就不一样了。

1. /\*\*\* derivative\*/
2. const Quaternion derivative(const Vector<3> &w) {
3. float dataQ[] = {
4. data[0], -data[1], -data[2], -data[3],
5. data[1],  data[0], -data[3],  data[2],
6. data[2],  data[3],  data[0], -data[1],
7. data[3], -data[2],  data[1],  data[0]
8. };
9. Matrix<4, 4> Q(dataQ);
10. Vector<4> v(0.0f, w.data[0], w.data[1], w.data[2]);
11. return Q \* v \* 0.5f;
12. }

**这一部分理论基础在《捷联惯性导航技术》中有详细介绍，关于DCM随时间传递的推导过程、四元数随时间传递的推导以及DCM、欧拉角、四元数之间的相互关系都有详细的介绍。**





        总结一下：corr包含磁力计修正、加速度计修正、（vision、mocap修正）、gyro中测量到的角速度偏转量，且因为corr为update函数中定义的变量，所以每次进入update函数中时会刷新corr变量的数据； \_rate也会刷新其中的数据，含义为三个姿态角的角速度（修正后）； \_q为外部定义的变量，在这个函数中只会+=不会重新赋值，如果计算出现错误会返回上一次计算出的\_q。

        ---------------------------------------------------------------------------------------------------------------------------------

7、将\_q转换成欧拉角euler并发布（532）

        终于从updata函数出来了，它还是相当重要的，主要就是它了，需要深入的理解透彻，下面就是些小角色了。Updata函出来就是直接用一更新的四元数（\_q）求出对于的欧拉角，以便在控制过程中实现完美的控制，控制还是需要用直接明了的欧拉角。上源码：

1. Vector<3> euler = \_q.to\_euler();
2. struct vehicle\_attitude\_s att = {};
3. att.timestamp = sensors.timestamp;
4. att.roll = euler(0);//获取的欧拉角赋值给roll、pitch、yaw
5. att.pitch = euler(1);
6. att.yaw = euler(2);
8. att.rollspeed = \_rates(0); //获取roll、pitch、yaw得旋转速度
9. att.pitchspeed = \_rates(1);
10. att.yawspeed = \_rates(2);
11. for (int i = 0; i < 3; i++) {
12. att.g\_comp[i] = \_accel(i) - \_pos\_acc(i);
13. //获取导航坐标系的重力加速度，前面已介绍过
14. }

        比较重要的就是如何由四元数获取欧拉角：\_q.to\_euler()

1. /\*\*
2. \* create Euler angles vector from the quaternion
3. \*/
4. Vector<3> to\_euler() const {
5. return Vector<3>(
6. atan2f(2.0f \* (data[0] \* data[1] + data[2] \* data[3]), 1.0f - 2.0f \* (data[1] \* data[1] + data[2] \* data[2])),
7. asinf(2.0f \* (data[0] \* data[2] - data[3] \* data[1])),
8. atan2f(2.0f \* (data[0] \* data[3] + data[1] \* data[2]), 1.0f - 2.0f \* (data[2] \* data[2] + data[3] \* data[3]))
9. );
10. }

        下面的就比较好理解了，不在赘述。

1. if (\_att\_pub == nullptr) {
2. \_att\_pub = orb\_advertise(ORB\_ID(vehicle\_attitude), &att);
3. } else {
4. orb\_publish(ORB\_ID(vehicle\_attitude), \_att\_pub, &att);
5. }
6. struct control\_state\_s ctrl\_state = {};
7. ctrl\_state.timestamp = sensors.timestamp;
8. /\* Attitude quaternions for control state \*/
9. ctrl\_state.q[0] = \_q(0);
10. ctrl\_state.q[1] = \_q(1);
11. ctrl\_state.q[2] = \_q(2);
12. ctrl\_state.q[3] = \_q(3);
13. /\* Attitude rates for control state \*/
14. ctrl\_state.roll\_rate = \_lp\_roll\_rate.apply(\_rates(0));
15. ctrl\_state.pitch\_rate = \_lp\_pitch\_rate.apply(\_rates(1));
16. ctrl\_state.yaw\_rate = \_lp\_yaw\_rate.apply(\_rates(2));
17. /\* Publish to control state topic \*/
18. if (\_ctrl\_state\_pub == nullptr) {
19. \_ctrl\_state\_pub = orb\_advertise(ORB\_ID(control\_state), &ctrl\_state);
20. } else {
21. orb\_publish(ORB\_ID(control\_state), \_ctrl\_state\_pub, &ctrl\_state);
22. }

六、总结  
        通过上面的分析对ardupilot源代码中的姿态解算算法有了深一步的了解，还有一部分就是关于mag的，还需要看一些论文，磁是最不容易处理的，极易受到外部干扰。还有就是加速度计的数据，对震动比较敏感，所以在无人机装机时需要在pixhawk下面安装减震板，做减震处理。  
        基本的姿态解算和飞行控制靠陀螺仪和加速计等肯定可以实现，代码也比较好写，可以基于mahony的那套算法，硬件部分就是选定芯片，确定通信接口；然后移植一套免费的OS即可。但是难得就是飞行时的稳定性和对震动和噪声的处理问题，这些都是最细节最重要的部分，也是最难的部分。  
        接下来就是关于姿态控制的了，至少一个多礼拜的时间才能搞明白吧；att\_estimate、pos\_estimate和att\_control、pos\_control这四个部分全部搞定需要下很大的功夫，慢慢搞吧，看论文吧，站在别人的肩膀上，加油~~~       
        感谢Mr、青龙等群友的帮助~~~

### 姿态解算4\_补充篇

**一、开篇**

        大家期待已久的第四篇来了，但是本篇可能比较水啊~~~见谅~~~

        首先，上一周没有什么收获，虽然看了不少的论文，但是却没有什么质的飞越~~~~

        看的论文都是关于姿态解算的，用的算法大部分也都是基于mahony算法的扩展，就当是深入理解一下姿态解算过程吧，稍微写写上周总结的一丁点的知识。

        昨天开了一整天的会议，最终决定姿态解算部分还是需要用EKF做，慢慢搞吧，但是我还是先把姿态控制部分解决了，感觉这个部分也相当重要。

**三、实验平台**

Software Version：PX4Firmware

Hardware Version：pixhawk

IDE：eclipse Juno （Windows）

**四、基础知识**

        研究多旋翼无人机前期主要需要了解其气动布局和复杂的动力学模型，然后就是姿态解算和控制器的设计。为了实现精确四旋翼飞行器的姿态估计，首先就是需要了解各传感器采集的数据和误差存在的原因，然后就是各种数学运算（都是基于一定的数学平台），比如DCM随时间的变化、四元数微分方程、四元数随时间的变化、重正交化等等。

        在理解构建四旋翼动力学模型时，需要了解它是属于四输入六输出的欠驱动系统，欠驱动主要就是输入只值靠四个电机的拉力实现控制角速度、角度、上升速度、下降速度、上升距离和下降距离。动力学模型还在研究中。

        实现四旋翼飞行器的自主飞行主要就是靠姿态解算和姿态控制。姿态控制是四旋翼飞行器实现各项工作的核心，位置控制建立在完善的姿态控制的基础上。前面已经介绍过，仅仅依靠陀螺仪积分获取角度值是不可靠的，由于陀螺仪由于其本身的固有的drift，积分带来的误差随时间的积累越来越大；而加速度计又容易受到噪声和震动的干扰。因此在此基础上使用各种算法对两种数据进行融合，以获取较为理想的姿态数据。比较常见的融合算法有EKF（Extended Kalman Filter）、UKF（Unscented Kalman Filter）、梯度下降法、CKF（Cubature Kalman Filter）、PF（Particle Filter）和CF（Complementary Filter）。

        其中mahony姿态解算算法主要就是基于CF设计的数据融合滤波器，该算法需要重点掌握其原理思想其分为AHRS（陀螺仪、加速度计、磁力计）和IMU（陀螺仪、加速度计）两种。

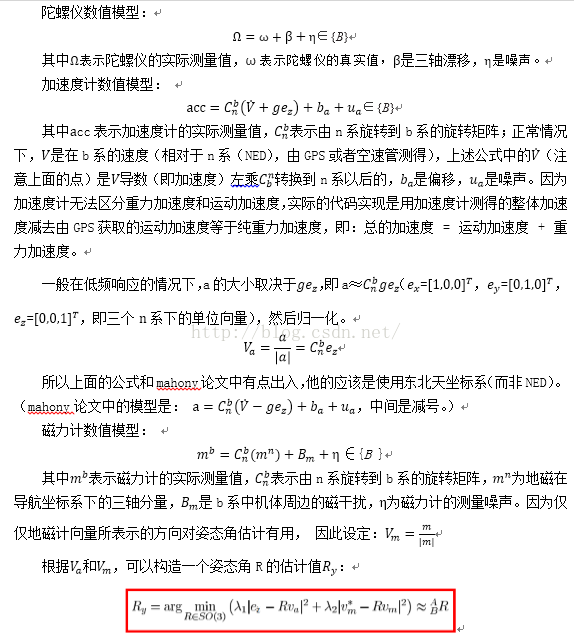
        目前研究多集中在非线性控制领域，由于非线性控制对模型准确性有较强的依赖，在模型误差存在的条件下，PID控制更加实用，所以现有的基于mahony互补滤波器设计的算法中大多数都是基于PID反馈控制器的。

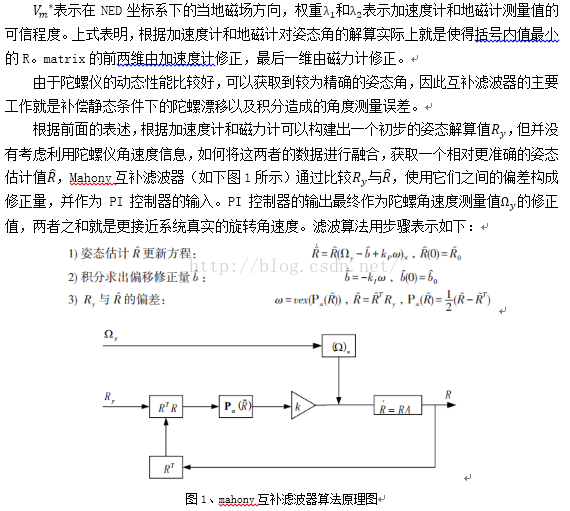
        接下来就是使用Simulink进行控制器建模仿真，也在研究中。

**五、正文**

1、传感器的数值模型（mahony互补滤波器）

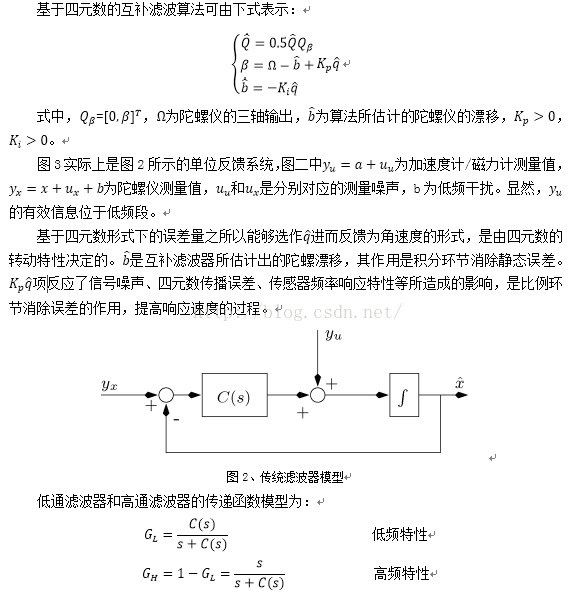
        首先还是贴图吧，word编辑的公式还是搞不进来。

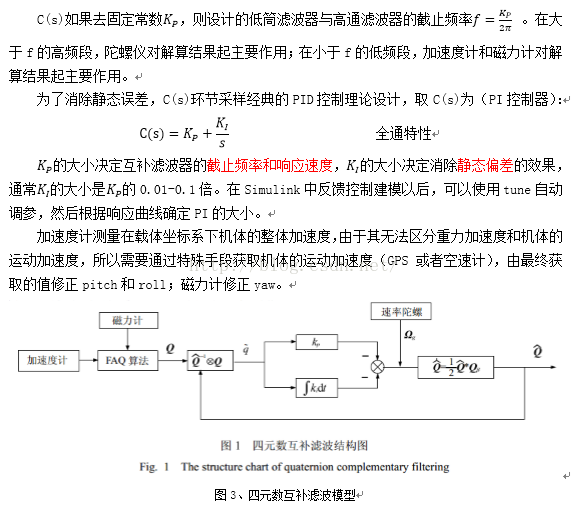




2、互补滤波器的设计（基于四元数）

        依旧是图片~~~~





        终于不是图片了~~~

        这里讲的互补滤波就是在短时间内采用陀螺仪得到的角度做为最优值，定时对加速度采样来的加速度值进行取平均值来校正陀螺仪的得到的角度。短时间内用陀螺仪比较准确，以它为主；长时间用加速度计比较准确，这时候加大它的比重，这就是互补了，不过加速度计要滤掉高频信号，陀螺仪要滤掉低频信号，互补滤波器就是根据传感器特性不同，通过不同的滤波器（高通或低通，互补的），然后再相加得到整个频带的信号。例如，加速度计测加速度值，其动态响应较慢，在高频时信号不可用，所以可通过低通滤波器抑制高频干扰；陀螺仪响应快，积分后可测倾角，不过由于零点漂移等，在低频段信号不好，通过高通滤波器可抑制低频干扰。将两者结合，就将陀螺仪和加速度计的优点融合起来，得到在高频和低频都较好的信号，互补滤波需要选择切换的频率点，即高通和低通的频率。

**六、在深入一点**

1、对mahony互补滤波算法和PX4Firmware源代码中的姿态解算算法的一点见解

        首先是最原始mahony滤波算法中的加速度计数据修正陀螺仪数据，源码如下。

1. 1.  // measured by Accelerate sensor
2. 2.    ax = ax / norm;
3. 3.    ay = ay / norm;
4. 4.    az = az / norm;
5. 5.
6. 6.  // estimated direction of gravity  (v)
7. 7.    vx = 2\*(q1q3 - q0q2);
8. 8.    vy = 2\*(q0q1 + q2q3);
9. 9.    vz = q0q0 - q1q1 - q2q2 + q3q3;
10. 10.
11. 11. // error is sum ofcross product between reference direction of fields and direction measured by Accelerate sensor
12. 12.   ex = ay\*vz - az\*vy ;
13. 13.   ey = az\*vx - ax\*vz ;
14. 14.   ez = ax\*vy - ay\*vx ;
15. 15.

        对上述代码的理解：[ax,ay,az] 表示由机体上的加速度计实际测量到的加速度数据，根据数据融合修正原理，加速度计在低频的情况下效果比较好，陀螺仪在高频情况下效果好；在低频下，忽略机体的运动加速度，即加速度计测量到的数据就是近似的纯重力加速度向量。[vx,vy,vz]表示由陀螺仪积分以后的姿态矩阵中获取的重力加速度向量（如何得到的不再赘述）。并且这两个向量都是在机体坐标系中，两个重力加速度向量做外积得到的向量就是所谓的误差向量（12、13、14行代码），且叉积的大小和陀螺仪的积分误差成正比，用该误差向量修正下一次的陀螺仪数据即可达到用加速度计修正陀螺仪数据误差的效果。

        然后就是PX4Firmware源码中的加速度计数据修正陀螺仪数据，源码如下。

1. 1.  // Accelerometer correction
2. 2.  // Project 'k' unit vector of earth frame to body frame
3. 3.  // Vector<3> k = \_q.conjugate\_inversed(Vector<3>(0.0f, 0.0f, 1.0f)); n系到
4. 4.  // b系
5. 5.  // Optimized version with dropped zeros
6. 6.      Vector<3> k(
7. 7.          2.0f \* (\_q(1) \* \_q(3) - \_q(0) \* \_q(2)),
8. 8.          2.0f \* (\_q(2) \* \_q(3) + \_q(0) \* \_q(1)),
9. 9.          (\_q(0) \* \_q(0) - \_q(1) \* \_q(1) - \_q(2) \* \_q(2) + \_q(3) \* \_q(3))
10. 10.     );
11. 11.  corr += (k % (\_accel - \_pos\_acc).normalized()) \* \_w\_accel;
12. 12.

        该算法中，Vector<3> k就是由陀螺仪积分以后的姿态矩阵中获取的重力加速度向量（等同于mahony算法中的[vx,vy,vz]）；这部分都类似，主要的不同就是下面的加速度计测量数据的处理，首先通过加速度计获取总的加速度向量，然后同过GPS或者空速管获取运动加速度向量，然后通过这两个向量之差求取纯重力加速度，即：总的加速度 =运动加速度 + 重力加速度（此处考虑了低频情况下的运动加速度，修正效果更好）。\_w\_accel代表权重，即类似于Kp。

        上述代码中的“%”运算符定义为叉积运算：

1. 1.  Vector<3> operator %(const Vector<3> &v) const {
2. 2.          return Vector<3>(
3. 3.                     data[1] \* v.data[2] - data[2] \* v.data[1],
4. 4.                     data[2] \* v.data[0] - data[0] \* v.data[2],
5. 5.                     data[0] \* v.data[1] - data[1] \* v.data[0]
6. 6.                 );
7. 7.      }
8. 8.  };

**七、总结**  
        没看多少东西，可能是因为清明放假太嗨了吧，心还没有收回来。好吧，我承认，是我太水~~~

        今天建了一个群，希望大家可以多多讨论技术相关问题，为祖国的无人机行业多做贡献。

### 姿态解算 5\_ECF/EKF/GD介绍

**一、开篇**

        很久没更新blog了， 最近研究的东西比较杂乱，也整理了很多东西，没有来的及更新，最近发现很多小伙伴都开始写blog了，在不更新就要“被落后了”。兄弟们，等等我啊~~~

        转了这么久又回到了姿态解算部分，最近研究了现有的几种关于姿态解算的实现算法，理论大都相通，请各位耐心看完。 希望阅读本部分的读者对姿态解算算法有一定基础的认识，该部分涉及了常用的三种姿态解算算法：Explicit Complementary Filter(ECF)、Extended Kalman Filter(EKF)、Gradient Descent Filter(GD)。本篇只做大致介绍，不结合代码分析，后续会结合代码分析算法单独出文章。

        现有的开源的姿态解算算法的代码实现在如下地址都可以找到，里面涉及了很多的姿态解算算法，慢慢的一个个的扣吧。

        链接：<https://developer.mbed.org/users/gke/code/UAVXArm-GKE/file/90292f8bd179/attitude.c>

**四、正文**

        经过近一段时间的研究，主要是关于多旋翼无人机姿态解算算法的研究，开源PX4源代码里面默认使用的是基于mahony的互补滤波算法（explicit complement filter）进行姿态解算的；还有一套备用姿态解算算法是基于kalman的EKF（extended kalman filter）；然后呢，其实还有一套PX4源代码中尚未涉及的基于madgwick的梯度下降算法（gradient descent）。

1.1、三套不同的算法各有优势可有优缺点。

        三套算法都可以使用，很多公司偏向于EKF，所以对EKF的要求还是比较高的，慢慢撸EKF吧，不然工作就愁了。小四轴用的比较多的还是互补滤波算法，尤其是大学生参加比赛也好，个人爱好也好，基于STM32自己制作小四轴的大多基于这套（我见过的很多都是用的CF，可能也有很多用的别的算法吧，不用太纠结这个）。梯度下降用的属于最少的了吧。最后出场的就是大头了：EKF；啥都别说了，好好学习它没错。

1.1.1、mahony算法

        Mahony算法即所谓的互补滤波算法，通过PID反馈控制器把误差量反馈补偿修正陀螺仪的误差。Mahony的互补滤波算法是基于IMU系统的，即陀螺仪和加速度计，过程中不涉及磁力计的修正。而我们使用过程中一般都是AHRS系统（即MARG：Magnetic、Angular Rate、Gravity），这个就需要寻求另外的算法或者基于mahony算法进行改进。PX4源代码中就是使用的改进的mahony算法，即加入了磁力计数据和GPS数据进行姿态解算。该算法是最简单和最容易实现的算法，PX4姿态解算默认的就是使用的这个。但是滤波器截止频率不好确定且与采样频率有关，Kp大截止频率就大。

        关于这一块，现在研究的比较多就是如何实现自适应调参。固定的参数不能获得所有情况下的最优运动姿态角，可以设计参数可调的自适应算法在不同运动状态下进行调节参数的大小。其参数调节规则为：正常运动状态情况下，Kp和Ki值取为系统初始化值；当运动体具有较大运动加速度或姿态变化剧烈时，应选择较小的Kp值（可取其初始化值的0.1倍），而Ki值应在同一数量级内适当取大一点。具体取值需根据实际应用系统选取。

        Mahony论文：《AComplementary Filter for Attitude Estimation of a Fixed-Wing UAV》和《NonlinearComplementary Filters on the Special Orthogonal Group》

        Mahony团队发表了太多的论文了，有兴趣的可以到researchgate查看：

        链接：<https://www.researchgate.net/profile/Robert_Mahony/publications/3>

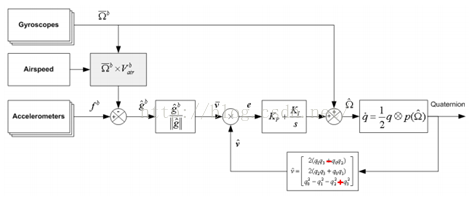


Fig.1 Explicit Complementary Filter

1.1.2 EKF算法

        EKF俗称扩展卡尔曼滤波算法，优点不言而喻。补充一句：Kalman还活着，还活着呢~~~

        EKF是出了名的计算量大，因为算法实现过程在不范是各种矩阵运算，对处理器的运算速度和精度要求很高对MCU的主频要求比较大，再加上系统的非线性，难以建立稳定可靠的状态方程以及合适的过程噪声协方差阵Q以及测量噪声协方差阵R。一言以蔽之，就是CPU计算负荷大。但是理解了KF以后，对整合滤波过程还是比较清晰的，很多公司都要求必须会EKF算法，优势就是在于处理过程中的预测和修正，其中一点就是可以估计陀螺仪的bias；在算法的计算处理过程中每一次都在不停的修正K和bias，相当于动态自适应算法了，主要就是调好Q、R。所以得把EKF搞懂，虽然比较难，但是必须知难而上。后续会根据直接KF求欧拉角和EKF求取四元数来更新blog，kalman filter相当有研究价值。

        相关论文：《A Double-Stage Kalman Filter for Orientation TrackingWith an Integrated Processor in 9-D IMU》

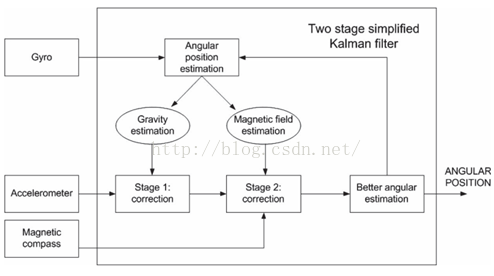


Fig.2 Extended Kalman Filter

1.1.3、梯度下降法

        The Gradient Descent是madgwick在2010搞出来的一套算法，其核心思想是将用加速度计和磁力计通过梯度下降法得到的姿态四元数与由陀螺仪积分得到的姿态，进行线性融合，得到最优的姿态。在算法实现过程中需要了解目标函数的建立，梯度下降法是根据目标函数求取误差向量的最小值的，这一点在madgwick的论文中尚未提及，应该是觉得默认大家都懂这个吧。

        该算法首先是只用陀螺仪数据根据一阶毕卡更新四元数，然后才是梯度下降法对加速度数据和磁力计数据进行处理以便求出最小的误差四元数，最后根据权重值（线性）把两者融合到一起得到最终的姿态四元数。使用梯度下降法就是对加速度数据和磁力计数据和相应参考向量求取误差量，这一部分和mahony\_base算法非常类似，mahony\_base是利用向量的叉乘求取量测值和参考向量的误差的，而梯度下降法是根据反梯度方向迭代求取最小误差的（理论基础就是构造目标函数使用梯度下降法求取误差向量的最小值）。过程中最重要就是参数β需要通过实验进行确定gyro的测量噪声（measurement error）。

        该算法适用于IMU和AHRS系统，其明显特点就是计算量小（computationally inexpensive），低频有效性（比如10hz）。Madgwick说该算法最后的精度比kalman\_base算法要好，我也不知道，没实际测试过；基于梯度下降的数据融合算法能够显著降低对处理器速度的要求。

        Madgwick论文：《An efficient orientation filter for inertial and inertial magneticsensor arrays》和《Estimation of IMU and MARG orientation using agradient descent algorithm》。梯度wiki：<https://en.wikipedia.org/wiki/Gradient_descent>

        算法代码试下：<http://www.x-io.co.uk/open-source-imu-and-ahrs-algorithms/>

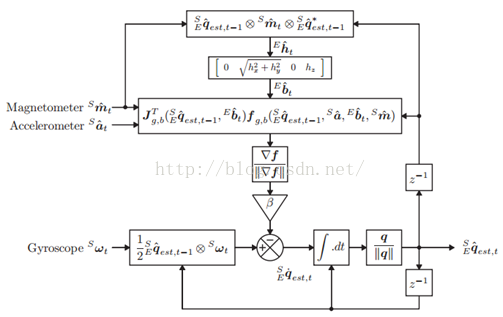


Fig.3 Complete Orientation Filter

**五、结论**

        上述三种算法都是以陀螺仪采集的角速度数据位主要控制量，通过不同的算法处理加速度数据和磁力计数据对由陀螺仪估计的姿态进行修正补偿。比如mahony\_base算法是通过构造PID反馈控制器实现对由陀螺仪估计的的姿态修正的，主要过程是基于重力或者重力和磁力参考向量修正陀螺仪数据，最后根据一阶毕卡（或多阶）更新姿态四元数；EKF是通过量测方程实现对由陀螺仪估计的姿态角修正的；Gradient descent是通过两种算法的融合来进行姿态解算的，首先是只用陀螺仪数据根据一阶毕卡更新四元数，然后才是梯度下降法对加速度数据和磁力计数据进行处理以便求出最小的误差四元数，最后根据权重值把两者融合到一起得到最终的姿态四元数。如果kalman算法是自适应的，那解算精度就比梯度下降算法强了。

        综上所述，姿态解算不过就是利用陀螺仪的高动态性能估计实时的姿态，但是由于陀螺仪固有的缺陷问题，导致随着时间的漂移实时的姿态估计的越来越不准确，所以呢，找一个也能估计姿态的sensor来修正一下，但是呢，总不能用两个陀螺仪吧，可以要找一个可以和陀螺仪固有缺陷相互补的sensor，那就是加速计了，无漂移，低频特性好。那么为何不直接只用加速度计进行姿态估计呢？原因就在于它的高频特性太烂，响应时间久，比如想实现一个3°角度变化量，多旋翼飞行速度很快，加速计响应时间来个上百ms的话，飞机早掉地上上升天了。这个也是为什么PX4里面对姿态控制部分分为两级PID控制的原因所在吧，内环控制角速度响应速度极快，外环控制角度响应速度相对较慢，但是有内环在接近实时的PID反馈修正，所以就不怕飞机升天了。

**六、附加题**

        经常关注群内的技术讨论有利于快速提升自我水平。所以，在群里水还是有很大好处的~~~好吧，就是找个借口而已~~~

        不知道如何添加附件，只能留网盘了，自己去下载吧。

        360云盘链接：<https://yunpan.cn/cReQVk9qMKTGT> （提取码：e25f）

学习px4代码也有一段时间了，所以想写一写，自己的一些学习心得吧，也算是笔记吧。

　　在px4这套代码中，每一个功能都是一个模块，例如姿态控制，也就是一个应用程序，我们可以把它添加到初始话脚本里，让它自启动。需要注意的就是在一个应用程序就是处理订阅的消息，然后发布处理过后的消息。这种消息机制就是uorb消息机制，可以找资料学习它具体实现的一个过程。

　　接下来，就逐步学习一下这个姿态控制的模块。

　　首先就是应用程序的入口，“extern "C" \_\_EXPORT int fw\_att\_control\_main(int argc, char \*argv[])”，在这个函数里就实现了，这个程序是否已近启动，如果没有启动就会注册函数来启动。

　　task\_main()这个函数就是整个姿态控制的关键，具体姿态控制的算法就是在这里面实现。刚开始就介绍说每个应用程序会订阅相应的消息，然后处理，最后发布。

所以刚开始就是一些消息的订阅：

[复制代码](javascript:void(0);)

\_att\_sp\_sub = orb\_subscribe(ORB\_ID(vehicle\_attitude\_setpoint));//姿态设定点

\_ctrl\_state\_sub = orb\_subscribe(ORB\_ID(control\_state));//飞机状态

\_accel\_sub = orb\_subscribe\_multi(ORB\_ID(sensor\_accel), 0);//加速度值

\_vcontrol\_mode\_sub = orb\_subscribe(ORB\_ID(vehicle\_control\_mode));//控制模式

\_params\_sub = orb\_subscribe(ORB\_ID(parameter\_update));//参数更新

\_manual\_sub = orb\_subscribe(ORB\_ID(manual\_control\_setpoint));//手动控制的设定

\_global\_pos\_sub = orb\_subscribe(ORB\_ID(vehicle\_global\_position));//车辆全球位置

\_vehicle\_status\_sub = orb\_subscribe(ORB\_ID(vehicle\_status));//飞机状态

\_vehicle\_land\_detected\_sub = orb\_subscribe(ORB\_ID(vehicle\_land\_detected));//着陆探测

[复制代码](javascript:void(0);)

接下来就是以check的方式获取订阅消息的值：

vehicle\_setpoint\_poll();

vehicle\_accel\_poll();

vehicle\_control\_mode\_poll();

vehicle\_manual\_poll();

vehicle\_status\_poll();

vehicle\_land\_detected\_poll();

接下来就是以阻塞等待方式检查参数是否更新，以及当前的飞机状态。阻塞等待这种获取订阅消息的方式，具体是怎么实现的可以查找资料进行学习。

这个if函数就是姿态控制运行的开始，如果姿态发生了改变，就运行这个if

if (fds[1].revents & POLLIN)

那么要知道姿态是否发生改变，那么就需要知道当前的姿态，这通过获取当前姿态的消息，然后得到四元数，转化为旋转矩阵，进而求得姿态角：

[复制代码](javascript:void(0);)

1 orb\_copy(ORB\_ID(control\_state), \_ctrl\_state\_sub, &\_ctrl\_state);

2

3

4 /\* get current rotation matrix and euler angles from control state quaternions \*/

5 math::Quaternion q\_att(\_ctrl\_state.q[0], \_ctrl\_state.q[1], \_ctrl\_state.q[2], \_ctrl\_state.q[3]);

6 \_R = q\_att.to\_dcm();

7

8 math::Vector<3> euler\_angles;

9 euler\_angles = \_R.to\_euler();

10 \_roll = euler\_angles(0);

11 \_pitch = euler\_angles(1);

12 \_yaw = euler\_angles(2);

[复制代码](javascript:void(0);)

由于姿态控制算法在一些模式下面是不会估算姿态设定点的，所以要确认这些标志。

\_att\_sp.fw\_control\_yaw = \_att\_sp.fw\_control\_yaw && \_vcontrol\_mode.flag\_control\_auto\_enabled;//判断垂尾，用于自主起飞。

判断是否安全故障，如果安全故障开启，就会设定降落伞。

接下来是襟翼的设定，分为手动控制和自动控制

[复制代码](javascript:void(0);)

1 float flap\_control = 0.0f;

2

3 /\* map flaps by default to manual if valid \*/

4 if (PX4\_ISFINITE(\_manual.flaps) && \_vcontrol\_mode.flag\_control\_manual\_enabled

5 && fabsf(\_parameters.flaps\_scale) > 0.01f) {

6 flap\_control = 0.5f \* (\_manual.flaps + 1.0f) \* \_parameters.flaps\_scale;

7

8 } else if (\_vcontrol\_mode.flag\_control\_auto\_enabled

9 && fabsf(\_parameters.flaps\_scale) > 0.01f) {

10 flap\_control = \_att\_sp.apply\_flaps ? 1.0f \* \_parameters.flaps\_scale : 0.0f;

11 }

12

13 // move the actual control value continuous with time, full flap travel in 1sec

14 if (fabsf(\_flaps\_applied - flap\_control) > 0.01f) {

15 \_flaps\_applied += (\_flaps\_applied - flap\_control) < 0 ? deltaT : -deltaT;

16

17 } else {

18 \_flaps\_applied = flap\_control;

19 }

[复制代码](javascript:void(0);)

然后是襟副翼的设定，分为手动控制和自动控制

[复制代码](javascript:void(0);)

1 if (PX4\_ISFINITE(\_manual.aux2) && \_vcontrol\_mode.flag\_control\_manual\_enabled

2 && fabsf(\_parameters.flaperon\_scale) > 0.01f) {

3 flaperon\_control = 0.5f \* (\_manual.aux2 + 1.0f) \* \_parameters.flaperon\_scale;

4

5 } else if (\_vcontrol\_mode.flag\_control\_auto\_enabled

6 && fabsf(\_parameters.flaperon\_scale) > 0.01f) {

7 flaperon\_control = \_att\_sp.apply\_flaps ? 1.0f \* \_parameters.flaperon\_scale : 0.0f;

8 }

9

10 // move the actual control value continuous with time, full flap travel in 1sec

11 if (fabsf(\_flaperons\_applied - flaperon\_control) > 0.01f) {

12 \_flaperons\_applied += (\_flaperons\_applied - flaperon\_control) < 0 ? deltaT : -deltaT;

13

14 } else {

15 \_flaperons\_applied = flaperon\_control;

16 }

[复制代码](javascript:void(0);)

接下来就是通过订阅的消息来判断是否能够控制姿态

也就是这一行代码：

1 if (\_vcontrol\_mode.flag\_control\_attitude\_enabled)

在这个if语句里面执行的主要有判断空速是否有效，如果无效设定空速为参数设定，如果有效设定空速为测量或者计算的空速。

通过vehicle\_global\_position来计算飞机的地面速度。

如果自问模式下面，我们需要通过遥控器来产生姿态设定点；

计算机体坐标系下飞机的速度；

准备姿态控制器运行需要的参数；

[复制代码](javascript:void(0);)

1 control\_input.roll = \_roll;

2 control\_input.pitch = \_pitch;

3 control\_input.yaw = \_yaw;

4 control\_input.roll\_rate = \_ctrl\_state.roll\_rate;

5 control\_input.pitch\_rate = \_ctrl\_state.pitch\_rate;

6 control\_input.yaw\_rate = \_ctrl\_state.yaw\_rate;

7 control\_input.speed\_body\_u = speed\_body\_u;

8 control\_input.speed\_body\_v = speed\_body\_v;

9 control\_input.speed\_body\_w = speed\_body\_w;

10 control\_input.acc\_body\_x = \_accel.x;

11 control\_input.acc\_body\_y = \_accel.y;

12 control\_input.acc\_body\_z = \_accel.z;

13 control\_input.roll\_setpoint = roll\_sp;

14 control\_input.pitch\_setpoint = pitch\_sp;

15 control\_input.yaw\_setpoint = yaw\_sp;

16 control\_input.airspeed\_min = \_parameters.airspeed\_min;

17 control\_input.airspeed\_max = \_parameters.airspeed\_max;

18 control\_input.airspeed = airspeed;

19 control\_input.scaler = airspeed\_scaling;

20 control\_input.lock\_integrator = lock\_integrator;

21 control\_input.groundspeed = groundspeed;

22 control\_input.groundspeed\_scaler = groundspeed\_scaler;

[复制代码](javascript:void(0);)

前面的判断就是为运行姿态控制器所准备的。

1 \_roll\_ctrl.control\_attitude(control\_input);

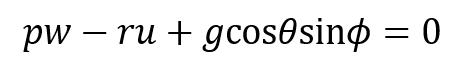
2 \_pitch\_ctrl.control\_attitude(control\_input);

3 \_yaw\_ctrl.control\_attitude(control\_input);

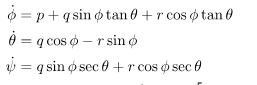
4 \_wheel\_ctrl.control\_attitude(control\_input);

上面就是计算目标与当前姿态的角度误差值,对于roll和pitch是计算角度误差，然后算出角速率，对于yaw速率的计算是，假设在没有侧向力的情况下,通过计算可以得到相应的yaw速率：

就是通过如下计算公式得到：

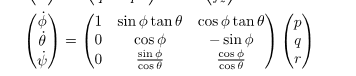


以及：

                                                                                                                                                                                                                                                       、

计算得到相应的yaw速率；

由于滚转，俯仰和偏航速率是在地面坐标系下，因此，要通过坐标转换转换到机体坐标系下，也就是：

                                                                                                                                                                           通过上述转换就把角速率转换到机体坐标系下了，接下来是通过误差，运用pid来控制姿态，下面的还需要研究一下。先这样吧。

      以上内容均属自己理解，有错误之处见谅，共同讨论。

### 姿态解算 6[\_Gradient Descent](http://blog.csdn.net/qq_21842557/article/details/51881355)

一、开篇

        在多旋翼进行姿态估计的过程中，最简单的就是直接使用gyro测量角速度进行积分求取欧拉角（RPY），但是由于gyro自身存在的bias和drift，导致直接测量过程随着时间的推荐变得越来越不精确。所以，许多研究者想到了使用加速度计和磁力计测量重力加速度和地球磁场强度对gyro的bias和drift进行补偿修正（不是eliminate gyro’s bias and drift，该叙述是错误的，加速度计和磁力计不可能排除gyro的bias和drift）。常用的方法就是CF、GD、EKF。

        但是由于磁力计测量地球磁场时极易被干扰（distortion），而且在补偿修正时磁场时无法修正RP的。所以，又有很多研究者想到了分开修正补偿求取欧拉角的方法，即通过加速度计和gyro一起求取RP，用磁力计和gyro一起修正补偿求取Yaw。该类解决方法中常见的就是two\_stage EKF、two\_stage GD、GD\_EKF。

        写了那么多就是为了引出GD~~~

Software Version：PX4Firmware

Hardware Version：pixhawk

IDE：eclipse Juno （Windows）

四、正文

1、写在前面

        该篇博文主要结合源代码介绍一下GD的实现过程，不再赘述理论的推导过程。说了那么多主要还是引出GD（gradient descent），关于该算法的介绍大家自行阅读madgwick的论文吧，该部分就不叙述算法推导过程的。  Madgwick论文：《An efficient orientation filter for inertial and inertial magneticsensor arrays》和《Estimation of IMU and MARG orientation using agradient descent algorithm》。梯度wiki：<https://en.wikipedia.org/wiki/Gradient_descent>。还有一个写的比较详细的关于GD介绍的blog，其中大部分都是论文翻译，英语不好的可以结合着看。链接：<http://blog.csdn.net/nemol1990/article/details/23102643>

2、源代码

1. IMUfilter::IMUfilter(double rate, double gyroscopeMeasurementError){
2. firstUpdate = 0;
3. //Estimated orientation quaternion elements with initial conditions.
4. SEq\_1 = 1;
5. SEq\_2 = 0;
6. SEq\_3 = 0;
7. SEq\_4 = 0;
8. //Sampling period (typical value is ~0.1s).
9. deltat = rate;
10. //Gyroscope measurement error (in degrees per second).
11. gyroMeasError = gyroscopeMeasurementError;
12. //Compute beta.
13. beta = sqrt(3.0 / 4.0) \* (PI \* (gyroMeasError / 180.0));
14. }
16. void IMUfilter::updateFilter(double w\_x, double w\_y, double w\_z, double a\_x, double a\_y, double a\_z) {
18. //Local system variables.
19. //Vector norm.
20. double norm;
21. //Quaternion rate from gyroscope elements.
22. double SEqDot\_omega\_1;
23. double SEqDot\_omega\_2;
24. double SEqDot\_omega\_3;
25. double SEqDot\_omega\_4;
26. //Objective function elements.
27. double f\_1;
28. double f\_2;
29. double f\_3;
30. //Objective function Jacobian elements.
31. double J\_11or24;
32. double J\_12or23;
33. double J\_13or22;
34. double J\_14or21;
35. double J\_32;
36. double J\_33;
37. //Objective function gradient elements.
38. double nablaf\_1;
39. double nablaf\_2;
40. double nablaf\_3;
41. double nablaf\_4;
43. //Auxiliary variables to avoid reapeated calcualtions.
44. double halfSEq\_1 = 0.5 \* SEq\_1;
45. double halfSEq\_2 = 0.5 \* SEq\_2;
46. double halfSEq\_3 = 0.5 \* SEq\_3;
47. double halfSEq\_4 = 0.5 \* SEq\_4;
48. double twoSEq\_1 = 2.0 \* SEq\_1;
49. double twoSEq\_2 = 2.0 \* SEq\_2;
50. double twoSEq\_3 = 2.0 \* SEq\_3;
52. //Compute the quaternion rate measured by gyroscopes.
53. SEqDot\_omega\_1 = -halfSEq\_2 \* w\_x - halfSEq\_3 \* w\_y - halfSEq\_4 \* w\_z;
54. SEqDot\_omega\_2 = halfSEq\_1 \* w\_x + halfSEq\_3 \* w\_z - halfSEq\_4 \* w\_y;
55. SEqDot\_omega\_3 = halfSEq\_1 \* w\_y - halfSEq\_2 \* w\_z + halfSEq\_4 \* w\_x;
56. SEqDot\_omega\_4 = halfSEq\_1 \* w\_z + halfSEq\_2 \* w\_y - halfSEq\_3 \* w\_x;
58. //Normalise the accelerometer measurement.
59. norm = sqrt(a\_x \* a\_x + a\_y \* a\_y + a\_z \* a\_z);
60. a\_x /= norm;
61. a\_y /= norm;
62. a\_z /= norm;
64. //Compute the objective function and Jacobian.
65. f\_1 = twoSEq\_2 \* SEq\_4 - twoSEq\_1 \* SEq\_3 - a\_x;
66. f\_2 = twoSEq\_1 \* SEq\_2 + twoSEq\_3 \* SEq\_4 - a\_y;
67. f\_3 = 1.0 - twoSEq\_2 \* SEq\_2 - twoSEq\_3 \* SEq\_3 - a\_z;
68. //J\_11 negated in matrix multiplication.
69. J\_11or24 = twoSEq\_3;
70. J\_12or23 = 2 \* SEq\_4;
71. //J\_12 negated in matrix multiplication
72. J\_13or22 = twoSEq\_1;
73. J\_14or21 = twoSEq\_2;
74. //Negated in matrix multiplication.
75. J\_32 = 2 \* J\_14or21;
76. //Negated in matrix multiplication.
77. J\_33 = 2 \* J\_11or24;
79. //Compute the gradient (matrix multiplication).
80. nablaf\_1 = J\_14or21 \* f\_2 - J\_11or24 \* f\_1;
81. nablaf\_2 = J\_12or23 \* f\_1 + J\_13or22 \* f\_2 - J\_32 \* f\_3;
82. nablaf\_3 = J\_12or23 \* f\_2 - J\_33 \* f\_3 - J\_13or22 \* f\_1;
83. nablaf\_4 = J\_14or21 \* f\_1 + J\_11or24 \* f\_2;
85. //Normalise the gradient.
86. norm = sqrt(nablaf\_1 \* nablaf\_1 + nablaf\_2 \* nablaf\_2 + nablaf\_3 \* nablaf\_3 + nablaf\_4 \* nablaf\_4);
87. nablaf\_1 /= norm;
88. nablaf\_2 /= norm;
89. nablaf\_3 /= norm;
90. nablaf\_4 /= norm;
92. //Compute then integrate the estimated quaternion rate.
93. SEq\_1 += (SEqDot\_omega\_1 - (beta \* nablaf\_1)) \* deltat;
94. SEq\_2 += (SEqDot\_omega\_2 - (beta \* nablaf\_2)) \* deltat;
95. SEq\_3 += (SEqDot\_omega\_3 - (beta \* nablaf\_3)) \* deltat;
96. SEq\_4 += (SEqDot\_omega\_4 - (beta \* nablaf\_4)) \* deltat;
98. //Normalise quaternion
99. norm = sqrt(SEq\_1 \* SEq\_1 + SEq\_2 \* SEq\_2 + SEq\_3 \* SEq\_3 + SEq\_4 \* SEq\_4);
100. SEq\_1 /= norm;
101. SEq\_2 /= norm;
102. SEq\_3 /= norm;
103. SEq\_4 /= norm;
105. }
106. void IMUfilter::reset(void) {
108. firstUpdate = 0;
109. //Estimated orientation quaternion elements with initial conditions.
110. SEq\_1 = 1;
111. SEq\_2 = 0;
112. SEq\_3 = 0;
113. SEq\_4 = 0;
114. }

3、重点来了

        3.1、首先就是初始化函数，包含四元数、采样周期、beta。源代码如下：

1. IMUfilter::IMUfilter(double rate, double gyroscopeMeasurementError){
2. firstUpdate = 0;
3. //Estimated orientation quaternion elements with initial conditions.
4. SEq\_1 = 1;
5. SEq\_2 = 0;
6. SEq\_3 = 0;
7. SEq\_4 = 0;
8. //Sampling period (typical value is ~0.1s).
9. deltat = rate;
10. //Gyroscope measurement error (in degrees per second).
11. gyroMeasError = gyroscopeMeasurementError;
12. //Compute beta.
13. beta = sqrt(3.0 / 4.0) \* (PI \* (gyroMeasError / 180.0));
14. }

        通过上述代码可以了解到，采样周期就是整体代码中该算法的执行频率的倒数。该部分比较重要的一点就是beta的计算，由上述可知beta = sqrt(3.0 / 4.0) \* (PI \* (gyroMeasError / 180.0))，主要就是通过gyro的芯片手册查找到gyroscopeMeasurementError，然后转换成弧度表示并乘一个权重。Beta的由来可以参考madgwick的论文的3.6节filter gain部分。论文中给出的介绍如下：Thefilter gain*β*represents all mean zerogyroscope measurement errors, expressed as the magnitude of a quaternionderivative. The sources of error include: sensor noise, signal aliasing, quantisation errors, calibration errors, sensor miss-alignment,sensor axis nonorthogonality and frequency response characteristics.建议大家还是好好阅读madgwick的论文，吃透它。

        3.2、然后捏，一系列的参数变量，一个个自己去对吧，前提是需要了解整个过程是什么样的，梯度下降法使用的就是梯度下降的理论，所以了解梯度下降就理解了该算法的精髓（梯度的作用和求取过程）。

1. //Local system variables.
2. //Vector norm.
3. double norm;
4. //Quaternion rate from gyroscope elements.
5. double SEqDot\_omega\_1;
6. double SEqDot\_omega\_2;
7. double SEqDot\_omega\_3;
8. double SEqDot\_omega\_4;
9. //Objective function elements.
10. double f\_1;
11. double f\_2;
12. double f\_3;
13. //Objective function Jacobian elements.
14. double J\_11or24;
15. double J\_12or23;
16. double J\_13or22;
17. double J\_14or21;
18. double J\_32;
19. double J\_33;
20. //Objective function gradient elements.
21. double nablaf\_1;
22. double nablaf\_2;
23. double nablaf\_3;
24. double nablaf\_4;
25. //Auxiliary variables to avoid reapeated calcualtions.
26. double halfSEq\_1 = 0.5 \* SEq\_1;
27. double halfSEq\_2 = 0.5 \* SEq\_2;
28. double halfSEq\_3 = 0.5 \* SEq\_3;
29. double halfSEq\_4 = 0.5 \* SEq\_4;
30. double twoSEq\_1 = 2.0 \* SEq\_1;
31. double twoSEq\_2 = 2.0 \* SEq\_2;
32. double twoSEq\_3 = 2.0 \* SEq\_3;

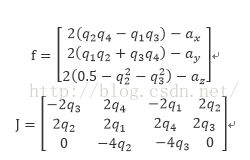
        3.3、然后捏，由w更新四元数，这个大家都很熟悉吧，一阶毕卡，不在赘述。

1. //Compute the quaternion rate measured by gyroscopes.
2. SEqDot\_omega\_1 = -halfSEq\_2 \* w\_x - halfSEq\_3 \* w\_y - halfSEq\_4 \* w\_z;
3. SEqDot\_omega\_2 = halfSEq\_1 \* w\_x + halfSEq\_3 \* w\_z - halfSEq\_4 \* w\_y;
4. SEqDot\_omega\_3 = halfSEq\_1 \* w\_y - halfSEq\_2 \* w\_z + halfSEq\_4 \* w\_x;
5. SEqDot\_omega\_4 = halfSEq\_1 \* w\_z + halfSEq\_2 \* w\_y - halfSEq\_3 \* w\_x;

        3.4、然后捏，构造目标函数（objectivefunction）和雅克比矩阵，并用它们求取梯度。

1. //Compute the objective function and Jacobian.
2. f\_1 = twoSEq\_2 \* SEq\_4 - twoSEq\_1 \* SEq\_3 - a\_x;
3. f\_2 = twoSEq\_1 \* SEq\_2 + twoSEq\_3 \* SEq\_4 - a\_y;
4. f\_3 = 1.0 - twoSEq\_2 \* SEq\_2 - twoSEq\_3 \* SEq\_3 - a\_z;
5. //J\_11 negated in matrix multiplication.
6. J\_11or24 = twoSEq\_3;
7. J\_12or23 = 2 \* SEq\_4;
8. //J\_12 negated in matrix multiplication
9. J\_13or22 = twoSEq\_1;
10. J\_14or21 = twoSEq\_2;
11. //Negated in matrix multiplication.
12. J\_32 = 2 \* J\_14or21;
13. //Negated in matrix multiplication.
14. J\_33 = 2 \* J\_11or24;

        看着上面的是不是比较晕呢？不用怕，结合下面的整体部分看着就不晕了。



        3.5、然后捏，求梯度：梯度公式为▽f=transpose(J)\*f

1. //Compute the gradient (matrix multiplication).
2. nablaf\_1 = J\_14or21 \* f\_2 - J\_11or24 \* f\_1;
3. nablaf\_2 = J\_12or23 \* f\_1 + J\_13or22 \* f\_2 - J\_32 \* f\_3;
4. nablaf\_3 = J\_12or23 \* f\_2 - J\_33 \* f\_3 - J\_13or22 \* f\_1;
5. nablaf\_4 = J\_14or21 \* f\_1 + J\_11or24 \* f\_2;

        3.6、然后捏，归一化，不管用什么算法进行姿态估计，都别忘记归一化（DCM还需要正交化）。

1. //Normalise the gradient.
2. norm = sqrt(nablaf\_1 \* nablaf\_1 + nablaf\_2 \* nablaf\_2 + nablaf\_3 \* nablaf\_3 + nablaf\_4 \* nablaf\_4);
3. nablaf\_1 /= norm;
4. nablaf\_2 /= norm;
5. nablaf\_3 /= norm;
6. nablaf\_4 /= norm;

        3.7、再然后捏，算法融合，即把一阶毕卡得到的姿态四元数和GD得到的姿态四元数融合，为什么这么做？为什么beta用作权重，移步madgwick的论文吧，里面有详细介绍，方法很巧妙。  
//Compute then integrate the estimated quaternion rate.

1. SEq\_1 += (SEqDot\_omega\_1 - (beta \* nablaf\_1)) \* deltat;
2. SEq\_2 += (SEqDot\_omega\_2 - (beta \* nablaf\_2)) \* deltat;
3. SEq\_3 += (SEqDot\_omega\_3 - (beta \* nablaf\_3)) \* deltat;
4. SEq\_4 += (SEqDot\_omega\_4 - (beta \* nablaf\_4)) \* deltat;
5. 最后，还是归一化，不写了~~~

        完了，有感觉么？~~~这种算法感觉比CF还要简单。

五、总结

        GD没什么东西，看懂理解了就OK了，主要是学习这样思想。就像开篇里面介绍的，现在大部分研究者都是集中精力实现各种姿态估计算法的解算算法（即杂交）。论文看的多了，感觉都一样，百分之50~60的内容都是在阐述一样古老的东西，在最后的最后才引出作者改变的一点东西，比如自适应算法，就是想一种方式实现对步长的取优变化（动态即自适应）。

        一部大部分时间还是在学习EKF，说简单它也简单就是五步法；说难也难，PQR的确定问题。还没有看五步法的由来，都是直接看应用案列，感觉这样理解的更快。I think the easiest way to understand the EKF is to start off with an example~~~~

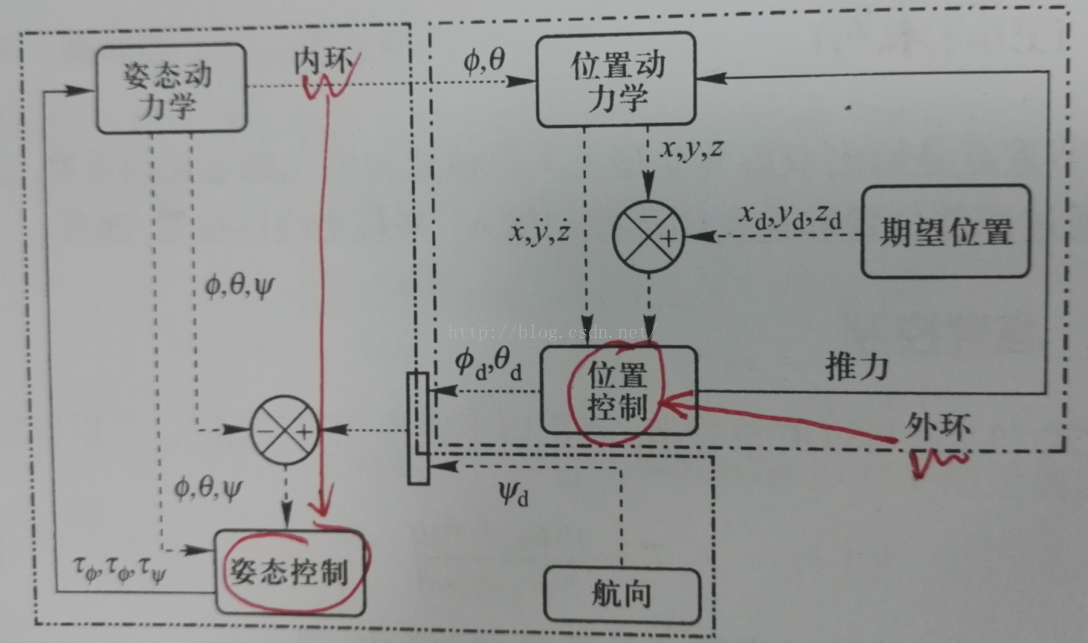
## 第四章 姿态控制

四、基础知识

1、写在前面

        无人机控制部分主要分为两个部分，姿态控制和位置控制部分

        下图阐释了PX4源码中的两个环路控制，分为姿态控制和位置控制。



        补充：关于Pixhawk原生固件中姿态(估计/控制)和位置(估计/控制)源码的应用问题

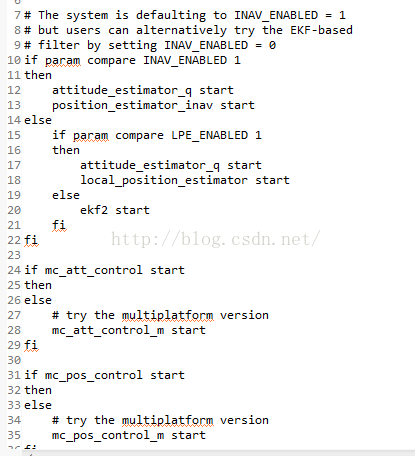
PX4Fireware原生固件中的modules中姿态估计有多种：Attitude\_estimator\_ekf、Attitude\_estimator\_q、ekf\_att\_pos\_estimator。

位置估计有：ekf\_att\_pos\_estimator、local\_position\_estimator、position\_estimator\_inav

姿态控制有：fw\_att\_control、mc\_att\_control、mc\_att\_control\_multiplatform、vtol\_att\_control

位置控制有：fw\_pos\_control\_l1、fw\_pos\_control\_l1、mc\_pos\_control\_multiplatform

        四旋翼用到以上哪些估计和控制算法呢？这部分在启动代码rc.mc\_app里面有详细的说明。



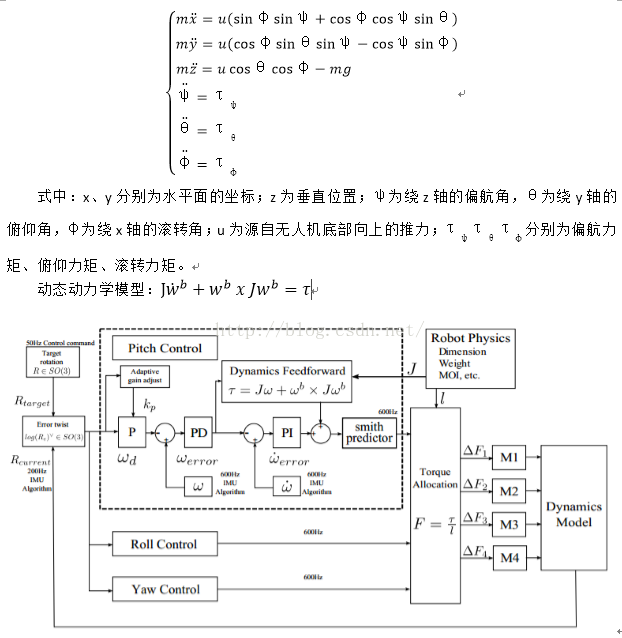
    默认的是：  
        姿态估计 Attitude\_estimator\_q  
        位置估计 position\_estimator\_inav  
        姿态控制 mc\_att\_control  
        位置控制 mc\_pos\_control

2、飞行控制（该部分属于理论概述）

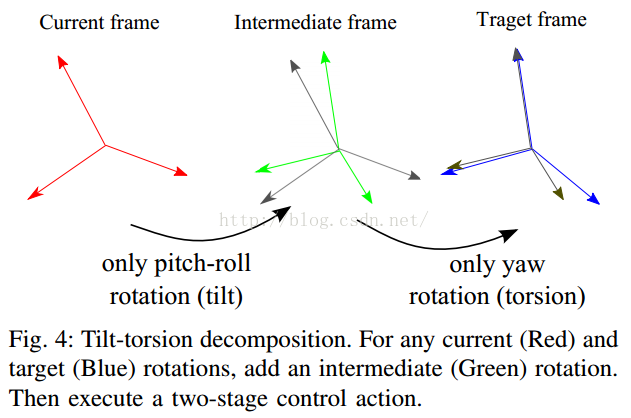
        飞行控制分为姿态控制和位置控制，该文章主讲姿态控制。

        所谓姿态控制，主要就是在前期姿态解算的基础上对四旋翼飞行器进行有效的飞行控制，以达到所需要的控制效果。在这种情况下，算法要学会如何连续地做决策，并且算法的评价应该根据其所做选择的长期质量来进行。举一个具体的例子，想想无人机飞行所面临的难题：每不到一秒，算法都必须反复地选择最佳的行动控制。控制过程还是以经典的PID反馈控制器为主（在控制环路中可以添加smith预测器）。那么如何实现控制呢?

        以四旋翼飞行器为例，主要就是通过改变旋翼的角速度来控制四旋翼无人机。每个旋翼产生一个推力（F1、F2、F3、F4）和一个力矩，其共同作用构成四旋翼无人机的主推力、偏航力矩、俯仰力矩和滚转力矩。在四旋翼无人机中，正对的一对旋翼旋转方向一致，另外一对与之相反，来抵消静态平稳飞行时的回转效应和气动力矩。升降以及RPY的实现不在赘述。控制对象就是四旋翼无人机，其动力学模型可以描述为：将其视为有一个力和三个力矩的三维刚体。如下给出了小角度变化条件下的四旋翼无人机的近似动力学模型：



        PS：PX4的姿态控制部分使用的是roll-pitch和yaw分开控制的（是为了解耦控制行为），即tilt和torsion两个环节。感性认识一下，如下图：



        根据经验所得，控制toll-pitch比控制yaw更容易实现。比如同样是实现10°的变化，roll-pitch需要60ms左右；但是yaw控制器却需要接近150ms。（上面两幅图是出自DJI某哥写的论文里面，仅作参考，结合理解Pixhawk）

    控制流程：

        1）、预处理：各参数的初始化。

        2）、稳定roll-pitch的角速度。

        3）、稳定roll-pitch的角度。

        4）、稳定yaw的角速度。

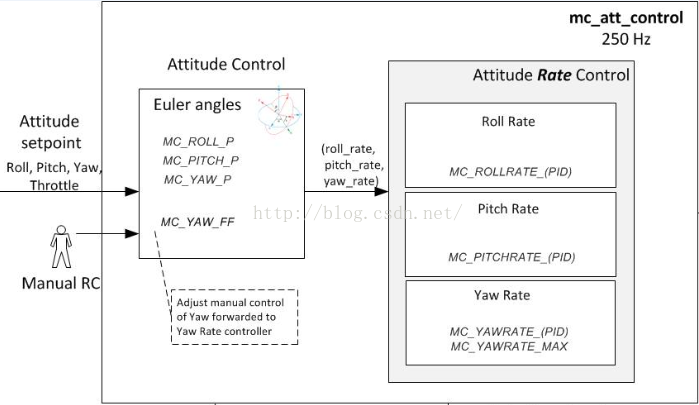
        5）、稳定yaw的角度。

        其中在第五步中有一个yaw的前馈控制（MC\_YAW\_FF）：There is MC\_YAW\_FF parameter that controls how much of userinput need to feed forward to yaw rate controller. 0 means very slow control,controller will start to move yaw only when sees yaw position error, 1 meansvery fast responsive control, but with some overshot, controller will move yawimmediately, always keeping yaw error near zero。This parameter is not critical and can be tuned in flight, inworst case yaw responce will be sluggish or too fast. Play with FF parameter toget comfortable responce. Valid range is 0…1. Typical value is 0.8…0.9. (Foraerial video optimal value may be much smaller to get smooth responce.) Yawovershot should be not more than 2-5%。

        摘自：<https://pixhawk.org/users/multirotor_pid_tuning>

3、 进入姿态控制源码的前期过程

        首先感性认识一下姿态控制部分的框架，控制部分分为内外环控制，内环控制角速度、外环控制角度。控制过程是先根据目标姿态（target）和当前姿态（current）求出偏差角，然后通过角速度来修正这个偏差角，最终到达目标姿态。



        和姿态解算算法的流程几乎类似，主要的代码流程首先就是按照C++语言的格式引用C语言的main函数，但是在该处变成了：

extern "C" \_\_EXPORT int mc\_att\_control\_main(int argc, char \*argv[])。

        然后捏：跳转到所谓的main函数，该部分有个要注意的点，如下代码所示：即mc\_att\_control::g\_control = new MulticopterAttitudeControl;//重点（934），new关键词应该不陌生吧，类似于C语言中的malloc，对变量进行内存分配的，即对姿态控制过程中使用到的变量赋初值。

1. int mc\_att\_control\_main(int argc, char \*argv[])
2. {
3. if (argc < 2) {
4. warnx("usage: mc\_att\_control {start|stop|status}");
5. return 1;
6. }
7. if (!strcmp(argv[1], "start")) {
8. if (mc\_att\_control::g\_control != nullptr) {
9. warnx("already running");
10. return 1;
11. }
12. mc\_att\_control::g\_control = new MulticopterAttitudeControl;//重点
13. if (mc\_att\_control::g\_control == nullptr) {
14. warnx("alloc failed");
15. return 1;
16. }
17. if (OK != mc\_att\_control::g\_control->start()) {//跳转
18. delete mc\_att\_control::g\_control;
19. mc\_att\_control::g\_control = nullptr;
20. warnx("start failed");
21. return 1;
22. }
23. return 0;
24. }

        然后捏：start函数

1. Int MulticopterAttitudeControl::start()
2. {
3. ASSERT(\_control\_task == -1);
4. /\* start the task \*/
5. \_control\_task = px4\_task\_spawn\_cmd("mc\_att\_control",
6. SCHED\_DEFAULT,
7. SCHED\_PRIORITY\_MAX - 5,
8. 1500,
9. (px4\_main\_t)&MulticopterAttitudeControl::task\_main\_trampoline,
10. nullptr);
11. if (\_control\_task < 0) {
12. warn("task start failed");
13. return -errno;
14. }
15. return OK;
16. }

        其中上面有个封装了nuttx自带的生成task的任务创建函数（他把优先级什么的做了重新的define，这么做是便于代码阅读）：px4\_task\_spawn\_cmd()，注意它的用法。其函数原型是：

1. px4\_task\_t px4\_task\_spawn\_cmd(const char \*name, int scheduler, int priority, int stack\_size, px4\_main\_t entry,
2. char \*const argv[])

        第一个参数是namespace，第二个参数是选择调度策略，第三个是任务优先级，第四个是任务的栈空间大小，第五个是任务的入口函数，最后一个一般是null。

        然后捏：

1. Void  MulticopterAttitudeControl::task\_main\_trampoline(int argc, char \*argv[])
2. {
3. mc\_att\_control::g\_control-**>**task\_main();
4. }

        再然后捏：终于到本体了。

1. Void MulticopterAttitudeControl::task\_main(){}

        比较讨厌的就是为什么要封装那么多层，应该是水平不够，还没有理解此处的用意。下面就是重点了。

五、重点

1、姿态控制源码\_订阅

        姿态控制的代码比姿态解算的代码少了不少，所以接下来分析应该会比较快。

        首先还是需要通过IPC模型uORB进行订阅所需要的数据。需要注意的一个细节就是在该算法处理过程中的有效数据的用途问题，最后处理过的数据最后又被改进程自己订阅了，然后再处理，再订阅，一直处于循环状态，这就是所谓的PID反馈控制器吧，最终达到所需求的控制效果，达到控制效果以后就把一系列的控制量置0（类似于idle），该任务一直在运行，随启动脚本启动的。

1. /\* \* do subscriptions \*/
2. \_v\_att\_sp\_sub = orb\_subscribe(ORB\_ID(vehicle\_attitude\_setpoint));
3. \_v\_rates\_sp\_sub = orb\_subscribe(ORB\_ID(vehicle\_rates\_setpoint));
4. \_ctrl\_state\_sub = orb\_subscribe(ORB\_ID(control\_state));
5. \_v\_control\_mode\_sub = orb\_subscribe(ORB\_ID(vehicle\_control\_mode));
6. \_params\_sub = orb\_subscribe(ORB\_ID(parameter\_update));
7. \_manual\_control\_sp\_sub = orb\_subscribe(ORB\_ID(manual\_control\_setpoint));
8. \_armed\_sub = orb\_subscribe(ORB\_ID(actuator\_armed));
9. \_vehicle\_status\_sub = orb\_subscribe(ORB\_ID(vehicle\_status));
10. \_motor\_limits\_sub = orb\_subscribe(ORB\_ID(multirotor\_motor\_limits));

        上面这些订阅到底订阅了哪些东西呢，顾名思义，根据ORB()中的参数的名称就是知道订阅的到底用于做什么的了。这套开源代码中最优越的地方时变量的命名很好，通俗易懂。

2、 参数初始化

        紧随上面的代码就是参数数据的获取，parameters主要就是我们前期定义的感兴趣的数据，在姿态控制中的这些数据都是私有数据（private），比如roll、pitch、yaw以及与它们对应的PID参数。注意区分\_params\_handles和\_params这两种数据结构（struct类型）。

1. /\* initialize parameters cache \*/
2. parameters\_update();
3. 函数原型欣赏：
4. int MulticopterAttitudeControl::parameters\_update()
5. {
6. float v;
7. /\* roll gains \*/
8. param\_get(\_params\_handles.roll\_p, &v);
9. \_params.att\_p(0) = v;
10. param\_get(\_params\_handles.roll\_rate\_p, &v);
11. \_params.rate\_p(0) = v;
12. param\_get(\_params\_handles.roll\_rate\_i, &v);
13. \_params.rate\_i(0) = v;
14. param\_get(\_params\_handles.roll\_rate\_d, &v);
15. \_params.rate\_d(0) = v;
16. param\_get(\_params\_handles.roll\_rate\_ff, &v);
17. \_params.rate\_ff(0) = v;
18. /\* pitch gains \*/
19. 省略
20. /\* yaw gains \*/
21. 省略
22. /\* angular rate limits \*/
23. param\_get(\_params\_handles.roll\_rate\_max, &\_params.roll\_rate\_max);
24. \_params.mc\_rate\_max(0) = math::radians(\_params.roll\_rate\_max);
25. param\_get(\_params\_handles.pitch\_rate\_max, &\_params.pitch\_rate\_max);
26. \_params.mc\_rate\_max(1) = math::radians(\_params.pitch\_rate\_max);
27. param\_get(\_params\_handles.yaw\_rate\_max, &\_params.yaw\_rate\_max);
28. \_params.mc\_rate\_max(2) = math::radians(\_params.yaw\_rate\_max);
29. /\* manual rate control scale and auto mode roll/pitch rate limits \*/
30. param\_get(\_params\_handles.acro\_roll\_max, &v);
31. \_params.acro\_rate\_max(0) = math::radians(v);
32. param\_get(\_params\_handles.acro\_pitch\_max, &v);
33. \_params.acro\_rate\_max(1) = math::radians(v);
34. param\_get(\_params\_handles.acro\_yaw\_max, &v);
35. \_params.acro\_rate\_max(2) = math::radians(v);
36. /\* stick deflection needed in rattitude mode to control rates not angles \*/
37. param\_get(\_params\_handles.rattitude\_thres, &\_params.rattitude\_thres);
38. \_actuators\_0\_circuit\_breaker\_enabled = circuit\_breaker\_enabled("CBRK\_RATE\_CTRL", CBRK\_RATE\_CTRL\_KEY);
39. return OK;
40. }

        重点分析一下上述代码：其中param\_get()函数比较重要，特别是内部使用的lock和unlock的使用（主要就是通过sem信号量控制对某一数据的互斥访问）。

1. Int param\_get(param\_t param, void \*val)
2. {
3. int result = -1;
4. param\_lock();
5. const void \*v = param\_get\_value\_ptr(param);
6. if (val != NULL) {
7. memcpy(val, v, param\_size(param));
8. result = 0;
9. }
10. param\_unlock();
11. return result;
12. }

        上述使用的\*lock和\*unlock通过sem实现互斥访问（进临界区），源码如下。

1. /\*\* lock the parameter store \*/
2. static void param\_lock(void)
3. {
4. //do {} while (px4\_sem\_wait(¶m\_sem) != 0);
5. }
6. /\*\* unlock the parameter store \*/
7. static void param\_unlock(void)
8. {
9. //px4\_sem\_post(¶m\_sem);
10. }

        上面是开源代码中的，代码里面把lock和unlock函数都写成空函数了，那还有屁用啊。应该是由于程序开发和版本控制不是一个人，有的程序开发到一半人走了，搞版本控制的，又找不到新的人来进行开发，搁置了忘记修改回来了吧；再或者别的什么意图。

        经过上述分析，该parameters\_update()函数主要就是获取roll、pitch、yaw的PID参数的。并对三种飞行模式（stablize、auto、acro）下的最大姿态速度做了限制。

3、NuttX任务使能

1. /\* wakeup source: vehicle attitude \*/
2. px4\_pollfd\_struct\_t fds[1];
3. fds[0].fd = \_ctrl\_state\_sub;
4. fds[0].events = POLLIN;

        注意上面的fd的赋值。随后进入任务的循环函数：while (!\_task\_should\_exit){}。都是一样的模式，在姿态解算时也是使用的该种方式。

4、阻塞等待数据

1. /\* wait for up to 100ms for data \*/
2. int pret = px4\_poll(&fds[0], (sizeof(fds) / sizeof(fds[0])), 100);
3. /\* timed out - periodic check for \_task\_should\_exit \*/
4. if (pret == 0) {
5. continue;
6. }
7. /\* this is undesirable but not much we can do - might want to flag unhappy status \*/
8. if (pret < 0) {
9. warn("mc att ctrl: poll error %d, %d", pret, errno);
10. /\* sleep a bit before next try \*/
11. usleep(100000);
12. continue;
13. }
14. perf\_begin(\_loop\_perf);

        首先是px4\_poll()配置阻塞时间100ms（uORB模型的函数API）。然后是打开MAVLINK协议，记录数据。如果poll失败，直接使用关键词continue从头开始运行（注意while和continue的组合使用）。其中的usleep(10000)函数属于线程级睡眠函数，使当前线程挂起。原文解释为：

        “Theusleep() function will cause the calling thread to be suspended from executionuntil either the number of real-time microseconds specified by the argument'usec' has elapsed or a signal is delivered to the calling thread。”

    上面最后一个perf\_begin(\_loop\_perf)，是一个空函数，带perf开头的都是空函数，它的作用主要是“Empty function calls forroscompatibility”。

5、重点来了（获取当前姿态Current）

        终于到了姿态控制器了，兴奋不？别只顾着兴奋了，好好理解一下。尤其是下面的几个\*poll函数，特别重要，后期算法中的很多数据都是通过这个几个\*poll()函数获取的，也是uORB模型，不理解这个后去会很晕的，别说没提醒啊；代码中没有一点冗余的部分，每一个函数、每一行都是其意义所在。

1. /\* run controller on attitude changes \*/
2. if (fds[0].revents & POLLIN) {
3. static uint64\_t last\_run = 0;
4. float dt = (hrt\_absolute\_time() - last\_run) / 1000000.0f;
5. last\_run = hrt\_absolute\_time();
6. /\* guard against too small (<2ms) and too large (>20ms) dt's \*/
7. if (dt < 0.002f) {
8. dt = 0.002f;
9. } else if (dt > 0.02f) {
10. dt = 0.02f;
11. }
12. /\* copy attitude and control state topics \*///获取当前姿态数据
13. orb\_copy(ORB\_ID(control\_state), \_ctrl\_state\_sub, &\_ctrl\_state);
14. /\* check for updates in other topics \*/
15. parameter\_update\_poll();
16. vehicle\_control\_mode\_poll();
17. arming\_status\_poll();
18. vehicle\_manual\_poll();
19. vehicle\_status\_poll();
20. vehicle\_motor\_limits\_poll();

         注意上面的revents，要与events区分开来，两者的区别如下：

    pollevent\_t events;  /\* The input event flags \*/

    pollevent\_t revents; /\* The output event flags \*/

        首先就是判断姿态控制器的控制任务是否已经使能，然后就是检测通过hrt获取时间精度的所需时间，并且约束在2ms至20ms以内。完了，orb\_copy()函数怎么用的忘记了。。。。

1. /\*\*
2. \* Fetch data from a topic.
3. \* This is the only operation that will reset the internal marker that
4. \* indicates that a topic has been updated for a subscriber. Once poll
5. \* or check return indicating that an updaet is available, this call
6. \* must be used to update the subscription.
7. \* @param meta    The uORB metadata (usually from the ORB\_ID() macro)
8. \*      for the topic.
9. \* @param handle  A handle returned from orb\_subscribe.
10. \* @param buffer  Pointer to the buffer receiving the data, or NULL
11. \*      if the caller wants to clear the updated flag without
12. \*      using the data.
13. \* @return    OK on success, ERROR otherwise with errno set accordingly.
14. \*/
15. int  orb\_copy(const struct orb\_metadata \*meta, int handle, void \*buffer)
16. {
17. return uORB::Manager::get\_instance()->orb\_copy(meta, handle, buffer);
18. }

        第三个参数就是为了保存通过orb\_subscribe()函数订阅获得的有效数据，该部分获取的是\_ctrl\_state，即控制姿态的数据，数据结构如下：（包含三轴加速度、三轴速度、三轴位置、空速、四元数、roll/pitch/yaw的速率）。记住这个copy的内容，后面会用到多次。

        然后就是检测数据是否已经更新，举一例说明问题。

1. /\* check for updates in other topics \*/
2. parameter\_update\_poll();
3. vehicle\_status\_poll();//注意这个，后面会用到内部的数据处理结果，即发布和订阅的ID问题。

        函数原型：

1. Void MulticopterAttitudeControl::parameter\_update\_poll()
2. {
3. bool updated;
4. /\* Check if parameters have changed \*/
5. orb\_check(\_params\_sub, &updated);
6. if (updated) {
7. struct parameter\_update\_s param\_update;
8. orb\_copy(ORB\_ID(parameter\_update), \_params\_sub, ¶m\_update);
9. parameters\_update();
10. }
11. }

        然后捏：飞行模式判断是否是MAIN\_STATE\_RATTITUD模式，该模式是一种新的飞行模式，只控制角速度，不控制角度，俗称半自稳模式（小舵量自稳大舵量手动），主要用在setpoint中，航点飞行。根据介绍，这个模式只有在pitch和roll都设置为Rattitude模式时才有意义，如果yaw也设置了该模式，那么就会自动被手动模式替代了。所以代码中只做了x、y阈值的检测。官方介绍：

* RATTITUDE The pilot's inputs are passed as roll, pitch, and yaw rate commands to the autopilot if they are greater than the mode's threshold. If not the inputs are passed as roll and pitch angle commands and a yaw rate command. Throttle is passed directly to the output mixer.

1. /\* Check if we are in rattitude（新的飞行模式，角速度模式，没有角度控制） mode and the pilot is above the threshold on pitch or roll (yaw can rotate 360 in normal att control).  If both are true don't  even bother running the attitude controllers \*/
2. if(\_vehicle\_status.main\_state == vehicle\_status\_s::MAIN\_STATE\_RATTITUDE){
3. if (fabsf(\_manual\_control\_sp.y) > \_params.rattitude\_thres ||
4. fabsf(\_manual\_control\_sp.x) > \_params.rattitude\_thres){
5. \_v\_control\_mode.flag\_control\_attitude\_enabled = false;
6. }
7. }

6、姿态控制（这才是重点）  
        确定飞行模式以后，根据前面的代码分析，在确定了飞行模式以后（判断当前飞行模式，通过最开始部分的\*poll函数获取，还记得它么？刚才提醒过了吧），再进行姿态控制。先来代码，然后详细分析。

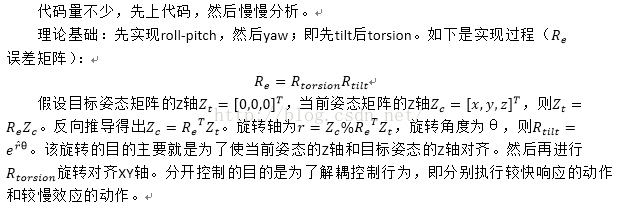
1. if (\_v\_control\_mode.flag\_control\_attitude\_enabled)
2. {
3. control\_attitude(dt);
4. /\* publish attitude rates setpoint \*/
5. \_v\_rates\_sp.roll = \_rates\_sp(0);
6. \_v\_rates\_sp.pitch = \_rates\_sp(1);
7. \_v\_rates\_sp.yaw = \_rates\_sp(2);
8. \_v\_rates\_sp.thrust = \_thrust\_sp;
9. \_v\_rates\_sp.timestamp = hrt\_absolute\_time();
10. if (\_v\_rates\_sp\_pub != nullptr) {
11. orb\_publish(\_rates\_sp\_id, \_v\_rates\_sp\_pub, &\_v\_rates\_sp);
13. } else if (\_rates\_sp\_id) {
14. \_v\_rates\_sp\_pub = orb\_advertise(\_rates\_sp\_id, &\_v\_rates\_sp);
15. }
16. //}
17. } else {
18. /\* attitude controller disabled, poll rates setpoint topic \*/
19. if (\_v\_control\_mode.flag\_control\_manual\_enabled) {
20. /\* manual rates control - ACRO mode \*/
21. \_rates\_sp = math::Vector<3>(\_manual\_control\_sp.y, -\_manual\_control\_sp.x, \_manual\_control\_sp.r).emult(\_params.acro\_rate\_max);
22. \_thrust\_sp = math::min(\_manual\_control\_sp.z, MANUAL\_THROTTLE\_MAX\_MULTICOPTER);
24. /\* publish attitude rates setpoint \*/
25. \_v\_rates\_sp.roll = \_rates\_sp(0);
26. \_v\_rates\_sp.pitch = \_rates\_sp(1);
27. \_v\_rates\_sp.yaw = \_rates\_sp(2);
28. \_v\_rates\_sp.thrust = \_thrust\_sp;
29. \_v\_rates\_sp.timestamp = hrt\_absolute\_time();
30. if (\_v\_rates\_sp\_pub != nullptr) {
31. orb\_publish(\_rates\_sp\_id, \_v\_rates\_sp\_pub, &\_v\_rates\_sp);
32. } else if (\_rates\_sp\_id) {
33. \_v\_rates\_sp\_pub = orb\_advertise(\_rates\_sp\_id, &\_v\_rates\_sp);
34. }
35. } else {
36. /\* attitude controller disabled, poll rates setpoint topic \*/
37. vehicle\_rates\_setpoint\_poll();
38. \_rates\_sp(0) = \_v\_rates\_sp.roll;
39. \_rates\_sp(1) = \_v\_rates\_sp.pitch;
40. \_rates\_sp(2) = \_v\_rates\_sp.yaw;
41. \_thrust\_sp = \_v\_rates\_sp.thrust;
42. }
43. }

        上面的代码中，初始就是control\_attitude(dt)，控制数据都是由它来获取的。该函数内部做了很多的处理，控制理论基本都是在这个里面体现的，所以需要深入研究理解它才可以进一步的研究后续的算法。它的内部会通过算法处理获得控制量（目标姿态Target），即\_rates\_sp，一个vector<3>变量，以便后续控制使用。好了，进入正题。

        首先是姿态控制(control\_attitude)，然后是速度控制(control\_attitude\_rates)，一个个来。

6.1、control\_attitude()函数(角度控制环)

        获取目标姿态Target



1. /\*\*
2. \* Attitude controller.
3. \* Input: 'vehicle\_attitude\_setpoint' topics (depending on mode)
4. \* Output: '\_rates\_sp' vector, '\_thrust\_sp'
5. \*/
6. Void MulticopterAttitudeControl::control\_attitude(float dt)
7. {
8. vehicle\_attitude\_setpoint\_poll();
9. \_thrust\_sp = \_v\_att\_sp.thrust;
10. /\* construct attitude setpoint rotation matrix \*/
11. math::Matrix<3, 3> R\_sp;
12. R\_sp.set(\_v\_att\_sp.R\_body);
13. /\* get current rotation matrix from control state quaternions \*/
14. math::Quaternion q\_att(\_ctrl\_state.q[0], \_ctrl\_state.q[1], \_ctrl\_state.q[2], \_ctrl\_state.q[3]);
15. math::Matrix<3, 3> R = q\_att.to\_dcm();
16. /\* all input data is ready, run controller itself \*/
17. /\* try to move thrust vector shortest way, because yaw response is slower than roll/pitch 约两倍\*/
18. math::Vector<3> R\_z(R(0, 2), R(1, 2), R(2, 2));
19. math::Vector<3> R\_sp\_z(R\_sp(0, 2), R\_sp(1, 2), R\_sp(2, 2));
20. /\* axis and sin(angle) of desired rotation \*/
21. math::Vector<3> e\_R = R.transposed() \* (R\_z % R\_sp\_z);
22. /\* calculate angle error \*/
23. float e\_R\_z\_sin = e\_R.length();
24. float e\_R\_z\_cos = R\_z \* R\_sp\_z;
25. /\* calculate weight for yaw control \*/
26. float yaw\_w = R\_sp(2, 2) \* R\_sp(2, 2);
27. /\* calculate rotation matrix after roll/pitch only rotation \*/
28. math::Matrix<3, 3> R\_rp;
29. if (e\_R\_z\_sin > 0.0f) {
30. /\* get axis-angle representation \*/
31. float e\_R\_z\_angle = atan2f(e\_R\_z\_sin, e\_R\_z\_cos);
32. math::Vector<3> e\_R\_z\_axis = e\_R / e\_R\_z\_sin;
33. e\_R = e\_R\_z\_axis \* e\_R\_z\_angle;
34. /\* cross product matrix for e\_R\_axis \*/
35. math::Matrix<3, 3> e\_R\_cp;
36. e\_R\_cp.zero();
37. e\_R\_cp(0, 1) = -e\_R\_z\_axis(2);
38. e\_R\_cp(0, 2) = e\_R\_z\_axis(1);
39. e\_R\_cp(1, 0) = e\_R\_z\_axis(2);
40. e\_R\_cp(1, 2) = -e\_R\_z\_axis(0);
41. e\_R\_cp(2, 0) = -e\_R\_z\_axis(1);
42. e\_R\_cp(2, 1) = e\_R\_z\_axis(0);
43. /\* rotation matrix for roll/pitch only rotation \*/
44. R\_rp = R \* (\_I + e\_R\_cp \* e\_R\_z\_sin + e\_R\_cp \* e\_R\_cp \* (1.0f - e\_R\_z\_cos));
45. } else {
46. /\* zero roll/pitch rotation \*/
47. R\_rp = R;
48. }
49. /\* R\_rp and R\_sp has the same Z axis, calculate yaw error \*/
50. math::Vector<3> R\_sp\_x(R\_sp(0, 0), R\_sp(1, 0), R\_sp(2, 0));
51. math::Vector<3> R\_rp\_x(R\_rp(0, 0), R\_rp(1, 0), R\_rp(2, 0));
52. e\_R(2) = atan2f((R\_rp\_x % R\_sp\_x) \* R\_sp\_z, R\_rp\_x \* R\_sp\_x) \* yaw\_w;
53. if (e\_R\_z\_cos < 0.0f) {
54. /\* for large thrust vector rotations use another rotation method:
55. \* calculate angle and axis for R -> R\_sp rotation directly \*/
56. math::Quaternion q;
57. q.from\_dcm(R.transposed() \* R\_sp);
58. math::Vector<3> e\_R\_d = q.imag();
59. e\_R\_d.normalize();
60. e\_R\_d \*= 2.0f \* atan2f(e\_R\_d.length(), q(0));
61. /\* use fusion of Z axis based rotation and direct rotation \*/
62. float direct\_w = e\_R\_z\_cos \* e\_R\_z\_cos \* yaw\_w;
63. e\_R = e\_R \* (1.0f - direct\_w) + e\_R\_d \* direct\_w;
64. }
65. /\* calculate angular rates setpoint \*/
66. \_rates\_sp = \_params.att\_p.emult(e\_R);
67. /\* limit rates \*/
68. for (int i = 0; i < 3; i++) {
69. \_rates\_sp(i) = math::constrain(\_rates\_sp(i), -\_params.mc\_rate\_max(i), \_params.mc\_rate\_max(i));
70. }
71. /\* feed forward yaw setpoint rate \*/
72. \_rates\_sp(2) += \_v\_att\_sp.yaw\_sp\_move\_rate \* yaw\_w \* \_params.yaw\_ff;
73. }

         详细分析：首先就是通过uORB模型检测姿态数据是否已经更新。检测到更新数据以后，把数据拷贝到当前，并通过\_thrust\_sp = \_v\_att\_sp.thrust把油门控制量赋值给控制变量。

        然后捏：构建姿态旋转矩阵（目标状态，所谓的TargetRotation）。

1. /\* construct attitude setpoint rotation matrix \*/
2. math::Matrix<3,3> R\_sp;
3. R\_sp.set(\_v\_att\_sp.R\_body);//不在赘述，在姿态解算时使用了同样的方法

        然后捏：通过控制四元数获取当前状态的旋转矩阵DCM，后面在计算误差以后旋转到b系时使用到了该处的DCM。即由姿态解算得到的有效姿态信息。

1. /\* get current rotation matrix from control state quaternions \*/
2. math::Quaternion q\_att(\_ctrl\_state.q[0], \_ctrl\_state.q[1], \_ctrl\_state.q[2], \_ctrl\_state.q[3]);
3. math::Matrix<3, 3> R = q\_att.to\_dcm();
4. 通过math库构建四元数；获取DCM的函数原型：无可厚非，都懂的
5. /\*\*\* create rotation matrix for the quaternion \*/
6. Matrix<3, 3> to\_dcm(void) const {
7. Matrix<3, 3> R;
8. float aSq = data[0] \* data[0];
9. float bSq = data[1] \* data[1];
10. float cSq = data[2] \* data[2];
11. float dSq = data[3] \* data[3];
12. R.data[0][0] = aSq + bSq - cSq - dSq;
13. R.data[0][1] = 2.0f \* (data[1] \* data[2] - data[0] \* data[3]);
14. R.data[0][2] = 2.0f \* (data[0] \* data[2] + data[1] \* data[3]);
15. R.data[1][0] = 2.0f \* (data[1] \* data[2] + data[0] \* data[3]);
16. R.data[1][1] = aSq - bSq + cSq - dSq;
17. R.data[1][2] = 2.0f \* (data[2] \* data[3] - data[0] \* data[1]);
18. R.data[2][0] = 2.0f \* (data[1] \* data[3] - data[0] \* data[2]);
19. R.data[2][1] = 2.0f \* (data[0] \* data[1] + data[2] \* data[3]);
20. R.data[2][2] = aSq - bSq - cSq + dSq;
21. return R;
22. }
23. };

        然后捏：取两个矩阵中的Z轴向量，即YAW-axis。

1. /\* all input data is ready, run controller itself \*/
2. /\* try to move thrust vector shortest way, because yaw response is slower than roll/pitch 这个地方应该知道旋转按照ZYX来进行的\*/
3. math::Vector<3> R\_z(R(0, 2), R(1, 2), R(2, 2));
4. math::Vector<3> R\_sp\_z(R\_sp(0, 2), R\_sp(1, 2), R\_sp(2, 2));

        然后捏：当前姿态的z轴和目标姿态的z轴的误差大小（即需要旋转的角度）并旋转到b系（即先对齐Z轴）。

1. /\* axis and sin(angle) of desired rotation \*/
2. math::Vector<3> e\_R = R.transposed() \* (R\_z % R\_sp\_z);

        R\_z%R\_sp\_z叉积，还记得这个么?在mahony算法中已经出现过一次了，就是求取误差的，本来应该z轴相互重合的，如果不是0就作为误差项。然后再左乘旋转矩阵旋转到b系。

        转置源码：

1. Matrix3<T> Matrix3<T>::transposed(void) const
2. {
3. return Matrix3<T>(Vector3<T>(a.x, b.x, c.x),
4. Vector3<T>(a.y, b.y, c.y),
5. Vector3<T>(a.z, b.z, c.z));
6. }

        然后捏：计算姿态角度误差（姿态误差），一个数学知识背景：由公式a×b=︱a︱︱b︱sinθ，a•b=︱a︱︱b︱cosθ，这里的R\_z和R\_sp\_z都是单位向量，模值为1，因此误差向量e\_R（a×b叉积就是误差）的模就是sinθ，点积就是cosθ。

1. /\* calculate angle error \*/
2. float e\_R\_z\_sin = e\_R.length();
3. float e\_R\_z\_cos = R\_z \* R\_sp\_z;

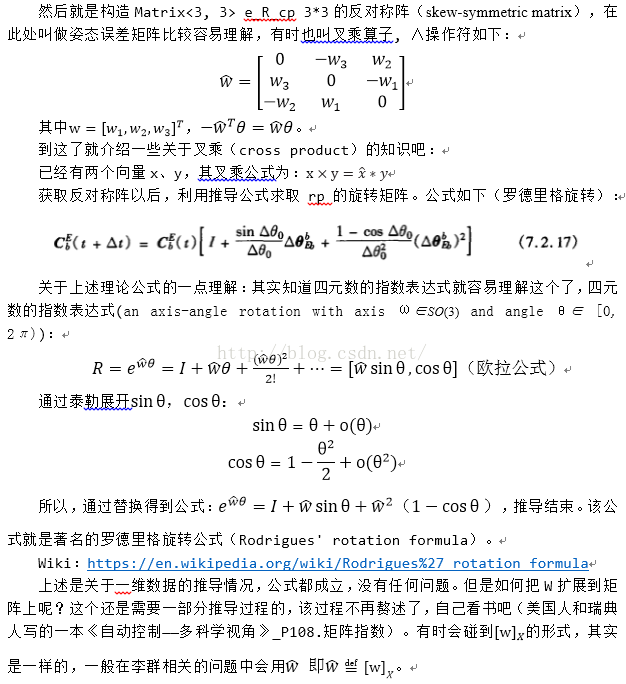
        然后捏：计算yaw的权重（不懂，谁帮忙解释一下原因。。跪谢）

1. /\* calculate weight for yaw control \*/
2. float yaw\_w = R\_sp(2, 2) \* R\_sp(2, 2);//不懂
3. 第一行的这个权重纯粹是因为如果不转动roll-pitch的话那应该是1，而如果转动的话，那个权重会平方倍衰减 （来自MR的解释）。

        然后捏：因为多轴的yaw响应一般比roll/pitch慢了接近一倍，因此将两者解耦(需要理解解耦的目的)，先补偿roll-pitch的变化，计算R\_rp。

1. /\* calculate rotation matrix after roll/pitch only rotation \*/
2. math::Matrix<3, 3> R\_rp;
3. if (e\_R\_z\_sin > 0.0f) {
4. /\* get axis-angle representation \*/
5. float e\_R\_z\_angle = atan2f(e\_R\_z\_sin, e\_R\_z\_cos);
6. math::Vector<3> e\_R\_z\_axis = e\_R / e\_R\_z\_sin;
7. e\_R = e\_R\_z\_axis \* e\_R\_z\_angle;//很大的用途，下面的R\_rp求取公式就是利用欧拉角计算的。
8. /\* cross product matrix for e\_R\_axis \*/
9. math::Matrix<3, 3> e\_R\_cp;
10. e\_R\_cp.zero();
11. e\_R\_cp(0, 1) = -e\_R\_z\_axis(2);
12. e\_R\_cp(0, 2) = e\_R\_z\_axis(1);
13. e\_R\_cp(1, 0) = e\_R\_z\_axis(2);
14. e\_R\_cp(1, 2) = -e\_R\_z\_axis(0);
15. e\_R\_cp(2, 0) = -e\_R\_z\_axis(1);
16. e\_R\_cp(2, 1) = e\_R\_z\_axis(0);
17. /\* rotation matrix for roll/pitch only rotation \*/
18. R\_rp = R \* (\_I + e\_R\_cp \* e\_R\_z\_sin + e\_R\_cp \* e\_R\_cp \* (1.0f - e\_R\_z\_cos));//罗德里格旋转公式：Rodrigues rotation formula
19. } else {
20. /\* zero roll/pitch rotation \*/
21. R\_rp = R;
22. }

        首先需要明确的就是上述处理过程中的DCM量都是通过欧拉角来表示的，这个主要就是考虑在控制时需要明确具体的欧拉角的大小，还有就是算法的解算过程是通过矩阵微分方程推导得到的（参考《惯性技术\_邓正隆》\_P148-P152以及《惯性导航\_秦永元》\_P342），并且在《惯性技术\_邓正隆》\_P154页介绍了姿态矩阵的实时解算方法。再判断两个z轴是否存在误差（e\_R\_z\_sin> 0.0f），若存在误差则通过反正切求出该误差角度值（atan2f(e\_R\_z\_sin,e\_R\_z\_cos)）；然后归一化e\_R\_z\_axis(e\_R /e\_R\_z\_sin该步计算主要就是利用e\_R\_z\_sin=e\_R.length()，往上看就是了，不会这么快就忘记了吧？！)。然后就是e\_R =e\_R\_z\_axis\* e\_R\_z\_angle了（主要就是为了误差向量用角度量表示）。



        然后捏：计算yaw的误差，该误差是roll\_pitch获取的z轴和目标姿态的z轴的误差。

1. /\* R\_rp and R\_sp has the same Z axis, calculate yaw error \*/
2. math::Vector<3> R\_sp\_x(R\_sp(0, 0), R\_sp(1, 0), R\_sp(2, 0));
3. math::Vector<3> R\_rp\_x(R\_rp(0, 0), R\_rp(1, 0), R\_rp(2, 0));
4. e\_R(2) = atan2f((R\_rp\_x % R\_sp\_x) \* R\_sp\_z, R\_rp\_x \* R\_sp\_x) \* yaw\_w;

        该部分同样是根据向量的叉积和点积求出误差角度的正弦和余弦，再反正切求出角度（又忘记了？回头看吧）。

        上面介绍的是在小角度变化时，如果是大角度变化时（大于90°，可能性比较小，还是集中在上面的算法吧）使用如何方法处理。

1. if (e\_R\_z\_cos < 0.0f) {
2. /\* for large thrust vector rotations use another rotation method:
3. \* calculate angle and axis for R->R\_sp rotation directly \*/
4. math::Quaternion q;
5. q.from\_dcm(R.transposed() \* R\_sp);
6. math::Vector<3> e\_R\_d = q.imag();
7. e\_R\_d.normalize();
8. e\_R\_d \*= 2.0f \* atan2f(e\_R\_d.length(), q(0));//不懂
9. /\* use fusion of Z axis based rotation and direct rotation \*/
10. float direct\_w = e\_R\_z\_cos \* e\_R\_z\_cos \* yaw\_w;
11. e\_R = e\_R \* (1.0f - direct\_w) + e\_R\_d \* direct\_w;
12. }

        上面这段代码比较好理解，主要就是由DCM获取四元数；然后把四元数的虚部取出赋值给e\_R\_d(e\_R\_d = q.imag());然后对其进行归一化处理；最后2行是先求出互补系数，再通过互补方式求取e\_R。

        然后捏：计算角速度变化的大小，并对其进行约束（constrain）。

1. /\* calculate angular rates setpoint \*/
2. \_rates\_sp = \_params.att\_p.emult(e\_R);
3. /\* limit rates \*/
4. for (int i = 0; i < 3; i++) {
5. \_rates\_sp(i) = math::constrain(\_rates\_sp(i), -\_params.mc\_rate\_max(i), \_params.mc\_rate\_max(i));
6. }
7. /\* feed forward yaw setpoint rate 因为yaw响应较慢，再加入一个前馈控制\*/
8. \_rates\_sp(2) += \_v\_att\_sp.yaw\_sp\_move\_rate \* yaw\_w \* \_params.yaw\_ff;
9. 上述代码中的一个emult(e\_R)的函数原型：
10. Matrix<Type, M, N> emult(const Matrix<Type, M, N> &other) const
11. {
12. Matrix<Type, M, N> res;
13. const Matrix<Type, M, N> &self = \*this;
14. for (size\_t i = 0; i < M; i++) {
15. for (size\_t j = 0; j < N; j++) {
16. res(i , j) = self(i, j)\*other(i, j);
17. }
18. }
19. return res;
20. }

        所以\_rates\_sp = \_params.att\_p.emult(e\_R)这句话的意思就是用att\_p的每一个元素和e\_R中对应位置的每一个元素相乘，结果赋值给\_rates\_sp角速度变量（该死的C++）。

6.2、control\_attitude(dt)返回以后

1. /\* publish attitude rates setpoint \*/
2. \_v\_rates\_sp.roll = \_rates\_sp(0);
3. \_v\_rates\_sp.pitch = \_rates\_sp(1);
4. \_v\_rates\_sp.yaw = \_rates\_sp(2);
5. \_v\_rates\_sp.thrust = \_thrust\_sp;
6. \_v\_rates\_sp.timestamp = hrt\_absolute\_time();
7. if (\_v\_rates\_sp\_pub != nullptr) {
8. orb\_publish(\_rates\_sp\_id, \_v\_rates\_sp\_pub, &\_v\_rates\_sp);
9. } else if (\_rates\_sp\_id) {
10. \_v\_rates\_sp\_pub = orb\_advertise(\_rates\_sp\_id, &\_v\_rates\_sp);
11. }

        上面这部分代码就通过control\_attitude(dt)经过一系列的算法处理过以后获取的目标内环角速度值，并通过uORB模型发布出去，包含roll-pitch-yaw、油门量和时间戳。

        该处正好可以再次深入理解一下uORB模型的一些理论。上述代码涉及了orb\_publish()和orb\_advertise()两个函数接口，通常第一次发布有效数据之前需要使用orb\_advertise()函数进行广播（类似topic register），它发布成功以后会返回一个handle供orb\_publish()发布时使用，即广播之后可以使用orb\_publish()进行发布新的数据。orb\_advertise()发布函数有第一个参数类似ID，返回值作为handle以便区分再次使用orb\_publish()时发布的是何种消息数据，即再次说明orb\_publish()需要在orb\_advertise()函数接口之后使用。通过查看orb\_advertise()函数的代码原型可以了解到，该函数的作用就类似于把需要后续发布的主题（topic）注册一下，然后才可以进行orb\_publish()。

        现在最不明了的就是这个数据发布出去以后在哪订阅了该数据呢或者说给谁用呢？？？自己发布，自己订阅，生生不息息，PX4里面有很多都是自己发布然后再自己订阅的，感谢群友我是肉包子的帮助。细节说明：在task\_main()的开头处就是订阅各种topics，其中就有一个\_v\_rates\_sp\_sub = orb\_subscribe(ORB\_ID(vehicle\_rates\_setpoint))订阅过程（735\_linenumber），它就是在该算法执行到最后时发布的控制量数据“\_v\_rates\_sp”（822），也就是按照前讲述的理论，自己订阅自己发布的topic，以实现循环控制。其中需要注意的就是发布时用的ID和订阅时用的不一致所迷惑了，其实它俩是一样的;因为在上述处理过程中是把ORB\_ID(vehicle\_rates\_setpoint)赋值给\_rates\_sp\_id它的（567），赋值过程在发布topic之前，即在vehicle\_status\_poll()函数内部（794）。

        前面的算法都是在flag\_control\_attitude\_enabled非零（姿态控制）的情况下实现的。紧接着，是在flag\_control\_attitude\_enabled为零时，即转变为flag\_control\_manual\_enabled：手动控制，方法类似，不在赘述。再接着，连手动控制都为使能时，只能poll了，并把控制量都置0。

        姿态控制结束。

        姿态速度控制开始。

7、姿态速度控制（角速度环）

        代码来也，先感性认识~~~~~~

1. if (\_v\_control\_mode.flag\_control\_rates\_enabled) {
2. control\_attitude\_rates(dt);
3. /\* publish actuator controls \*/
4. \_actuators.control[0] = (PX4\_ISFINITE(\_att\_control(0))) ? \_att\_control(0) : 0.0f;
5. \_actuators.control[1] = (PX4\_ISFINITE(\_att\_control(1))) ? \_att\_control(1) : 0.0f;
6. \_actuators.control[2] = (PX4\_ISFINITE(\_att\_control(2))) ? \_att\_control(2) : 0.0f;
7. \_actuators.control[3] = (PX4\_ISFINITE(\_thrust\_sp)) ? \_thrust\_sp : 0.0f;
8. \_actuators.timestamp = hrt\_absolute\_time();
9. \_actuators.timestamp\_sample = \_ctrl\_state.timestamp;
10. \_controller\_status.roll\_rate\_integ = \_rates\_int(0);
11. \_controller\_status.pitch\_rate\_integ = \_rates\_int(1);
12. \_controller\_status.yaw\_rate\_integ = \_rates\_int(2);
13. \_controller\_status.timestamp = hrt\_absolute\_time();
14. if (!\_actuators\_0\_circuit\_breaker\_enabled) {
15. if (\_actuators\_0\_pub != nullptr) {
16. orb\_publish(\_actuators\_id, \_actuators\_0\_pub, &\_actuators);
17. perf\_end(\_controller\_latency\_perf);
18. } else if (\_actuators\_id) {
19. \_actuators\_0\_pub = orb\_advertise(\_actuators\_id, &\_actuators);
20. }
21. }
22. /\* publish controller status \*/
23. if(\_controller\_status\_pub != nullptr) {
24. orb\_publish(ORB\_ID(mc\_att\_ctrl\_status),\_controller\_status\_pub, &\_controller\_status);
25. } else {
26. \_controller\_status\_pub = orb\_advertise(ORB\_ID(mc\_att\_ctrl\_status), &\_controller\_status);
27. }
28. }

        进入上述代码首先就是control\_attitude\_rates(dt)，该函数的输入是前面算法处理得到的\_rates\_sp控制量（目标姿态），输出是\_att\_control控制量。其函数原型是：

1. Void MulticopterAttitudeControl::control\_attitude\_rates(float dt)
2. {
3. /\* reset integral if disarmed \*/
4. if (!\_armed.armed || !\_vehicle\_status.is\_rotary\_wing) {
5. \_rates\_int.zero();
6. }
7. /\* current body angular rates \*/
8. math::Vector<3> rates;
9. rates(0) = \_ctrl\_state.roll\_rate;
10. rates(1) = \_ctrl\_state.pitch\_rate;
11. rates(2) = \_ctrl\_state.yaw\_rate;
12. /\* angular rates error \*/
13. math::Vector<3> rates\_err = \_rates\_sp - rates;//目标姿态-当前姿态
14. \_att\_control = \_params.rate\_p.emult(rates\_err) + \_params.rate\_d.emult(\_rates\_prev - rates) / dt + \_rates\_int + \_params.rate\_ff.emult(\_rates\_sp - \_rates\_sp\_prev) / dt;
15. \_rates\_sp\_prev = \_rates\_sp;
16. \_rates\_prev = rates;
17. /\* update integral only if not saturated on low limit and if motor commands are not saturated \*/
18. if (\_thrust\_sp > MIN\_TAKEOFF\_THRUST && !\_motor\_limits.lower\_limit && !\_motor\_limits.upper\_limit ) {
19. for (int i = 0; i < 3; i++) {
20. if (fabsf(\_att\_control(i)) < \_thrust\_sp) {
21. float rate\_i = \_rates\_int(i) + \_params.rate\_i(i) \* rates\_err(i) \* dt;
22. if (PX4\_ISFINITE(rate\_i) && rate\_i > -RATES\_I\_LIMIT && rate\_i < RATES\_I\_LIMIT &&
23. \_att\_control(i) > -RATES\_I\_LIMIT && \_att\_control(i) < RATES\_I\_LIMIT) {
24. \_rates\_int(i) = rate\_i;
25. }
26. }
27. }
28. }
29. }

        主要就是通过\_ctrl\_state数据结构（前面说过要记住它的吧，当前姿态信息）把需要的有效数据赋值给rates，然后通过rates进行一系列的算法处理。该过程中最最最需要注意的就是这个\_ctrl\_state变量的获取过程，其实还是通过uORB。前面也涉及过多次，比如control\_attitude()函数内部使用它构造状态四元数。

        如下非常重要。。。。。。打通姿态解算和姿态控制部分。

        数据获取过程：

        Quaterion\_CF姿态解算算法：（需要对代码有个整体把握，不然会很晕啊，还有就是关于姿态解算部分使用的CF时，在PX4Firmware/src/module/attitude\_estimator\_q中）。首先是通过姿态解算部分获取当前的姿态信息（Quaterion\_CF），获取之后通过uORB模型发布.

1. /\* publish to control state topic \*/(646)
2. orb\_publish\_auto(ORB\_ID(control\_state), &\_ctrl\_state\_pub, &ctrl\_state, &ctrl\_inst, ORB\_PRIO\_HIGH);

        Ekf2姿态解算算法：（还是需要对代码有个整体把握，不然还是会很晕啊，还有就是关于姿态解算部分使用的ekf2时，在PX4Firmware/src/module/ekf2中）。首先是通过姿态解算部分获取当前的姿态信息（ekf2），获取之后通过uORB模型发布：

1. // publish control state data(475)
2. if (\_control\_state\_pub == nullptr) {
3. \_control\_state\_pub = orb\_advertise(ORB\_ID(control\_state), &ctrl\_state);
4. } else {
5. orb\_publish(ORB\_ID(control\_state), \_control\_state\_pub, &ctrl\_state);
6. }

        关于到底使用哪种解算算法在启动脚本rc\_mc\_app里面涉及了关于姿态解算用什么算法的问题，里面给了一个宏，通过宏定义选取的。而且在使用四元数的互补算法和ekf2的算法里面都对结算到的姿态信息进行了发布处理，以便供姿态控制时订阅使用。

        然后再姿态控制中通过uORB模型订阅：

1. \_ctrl\_state\_sub = orb\_subscribe(ORB\_ID(control\_state));(736)
2. orb\_copy(ORB\_ID(control\_state), \_ctrl\_state\_sub, &\_ctrl\_state);(787)

        再然后就是姿态控制量（\_att\_control）的获取：获取原则是由预期姿态控制获取的角速度值与通过uORB获得的角速度值做差（该部分差值代表error=target-current，\_ctrl\_state应该是要控制的控制量）。rates\_err的获取就是通过经典的PD控制器了，然后再加个前馈。还未使用I控制器；在后面会单独使用。

1. /\* angular rates error \*/
2. math::Vector<3> rates\_err = \_rates\_sp - rates;
3. \_att\_control = \_params.rate\_p.emult(rates\_err) + \_params.rate\_d.emult(\_rates\_prev - rates) / dt + \_rates\_int + \_params.rate\_ff.emult(\_rates\_sp - \_rates\_sp\_prev) / dt;
4. \_rates\_sp\_prev = \_rates\_sp;
5. \_rates\_prev = rates;
6. I控制器的使用（注意使用条件）。
7. /\* update integral only if not saturated on low limit and if motor commands are not saturated \*/
8. if (\_thrust\_sp > MIN\_TAKEOFF\_THRUST && !\_motor\_limits.lower\_limit && !\_motor\_limits.upper\_limit ) {
9. for (int i = 0; i < 3; i++) {
10. if (fabsf(\_att\_control(i)) < \_thrust\_sp) {
11. float rate\_i = \_rates\_int(i) + \_params.rate\_i(i) \* rates\_err(i) \* dt;
12. if (PX4\_ISFINITE(rate\_i) && rate\_i > -RATES\_I\_LIMIT && rate\_i < RATES\_I\_LIMIT &&
13. \_att\_control(i) > -RATES\_I\_LIMIT && \_att\_control(i) < RATES\_I\_LIMIT) {
14. \_rates\_int(i) = rate\_i;
15. }
16. }
17. }
18. }

        其中fabsf()的函数原型是（取绝对值）：

1. float fabsf(float x)
2. {
3. return ((x < 0) ? -x : x);
4. }

    常用的几种取绝对值的函数：

        int abs(int i);         //处理int类型的取绝对值

        double fabs(double i); //处理double类型的取绝对值

        float fabsf(float i);  //处理float类型的取绝对值

        注意上面的fabsf(\_att\_control(i)) <\_thrust\_sp)这个判断项，符合就执行积分。这个做主要是为了安全考虑，当roll的变化值需要很大时，就停止积分项的累加以便防止积分项产生较大的误差。

        别看这个\_thrust\_sp单单的一个控制量，其实它可麻烦了，不对整体核心的解算和控制（姿态解算姿态控制、位置解算位置控制）有个深入理解的话，很难看懂这部分。下面详细介绍一下这个控制量的获取过程，耐心看，别晕了。介绍还是需要正向介绍，在看的时候可以反向看，比较容易理解。



        首先是\_v\_att\_sp\_sub =orb\_subscribe(ORB\_ID(vehicle\_attitude\_setpoint));（813），订阅所需的控制量。

        然后再attitude control里面处理：\_thrust\_sp =\_v\_att\_sp.thrust（653）

    上面是订阅拷贝和使用部分，下面就是发布部分。

        发布分为两个地方，一个是mc\_pos\_control和mavlink\_receiver.cpp。主要考虑前者。

        ID重定义：\_attitude\_setpoint\_id= ORB\_ID(vehicle\_attitude\_setpoint);（595）

        正式发布给mc\_att\_control: orb\_publish(\_attitude\_setpoint\_id,\_att\_sp\_pub,&\_att\_sp);(1932)

        为何称为正式发布呢？主要是因为在mc\_pos\_control里面根据不懂的模式进行了多次发布处理，比如idle状态下这个\_thrust\_sp就赋值为0发布出去。这个正式发布出来的才是我们飞行控制过程中需要考虑的控制量。

        补充mavlink\_receiver.cpp

        orb\_publish(ORB\_ID(vehicle\_attitude\_setpoint),\_att\_sp\_pub,&\_att\_sp);（951）

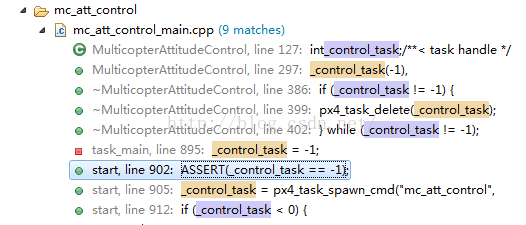
    现在发现这个规律了吧，任务间通信（IPC）都是靠的uORB，找不到来源就查ID吧。

8、发布控制量

1. /\* publish actuator controls \*/
2. \_actuators.control[0] = (PX4\_ISFINITE(\_att\_control(0))) ? \_att\_control(0) : 0.0f;
3. \_actuators.control[1] = (PX4\_ISFINITE(\_att\_control(1))) ? \_att\_control(1) : 0.0f;
4. \_actuators.control[2] = (PX4\_ISFINITE(\_att\_control(2))) ? \_att\_control(2) : 0.0f;
5. \_actuators.control[3] = (PX4\_ISFINITE(\_thrust\_sp)) ? \_thrust\_sp : 0.0f;
6. \_actuators.timestamp = hrt\_absolute\_time();
7. \_actuators.timestamp\_sample = \_ctrl\_state.timestamp;
8. \_controller\_status.roll\_rate\_integ = \_rates\_int(0);
9. \_controller\_status.pitch\_rate\_integ = \_rates\_int(1);
10. \_controller\_status.yaw\_rate\_integ = \_rates\_int(2);
11. \_controller\_status.timestamp = hrt\_absolute\_time();
12. if (!\_actuators\_0\_circuit\_breaker\_enabled) {
13. if (\_actuators\_0\_pub != nullptr) {
14. orb\_publish(\_actuators\_id, \_actuators\_0\_pub, &\_actuators);
15. perf\_end(\_controller\_latency\_perf);
16. } else if (\_actuators\_id) {
17. \_actuators\_0\_pub = orb\_advertise(\_actuators\_id, &\_actuators);
18. }
19. }
20. /\* publish controller status \*/
21. if(\_controller\_status\_pub != nullptr) {
22. orb\_publish(ORB\_ID(mc\_att\_ctrl\_status),\_controller\_status\_pub, &\_controller\_status);
23. } else {
24. \_controller\_status\_pub = orb\_advertise(ORB\_ID(mc\_att\_ctrl\_status), &\_controller\_status);
25. }
26. }
27. }
28. perf\_end(\_loop\_perf);
29. }
30. \_control\_task = -1;
31. return;

        PS：一个比较有趣的东西task handle：“\_control\_task”

        了解姿态控制任务的执行流么？可以参考这个task handle思考思考。



**六、结论**

        其实在mc\_att\_control里面就完全涵盖了姿态控制的内环和外环（即角速度控制、角度控制）。主要就是attitude control和attitude rate control两个部分，前者是控制角度后者是控制角速度并把控制量输入给mixer。在控制过程中是通过控制电机的速度以实现多旋翼的整体的rpy的速度，通过这个速度随时间的累加实现角度控制。

        attitude\_control 输入是体轴矩阵R和期望的体轴矩阵Rsp，角度环只是一个P控制，算出来之后输出的是期望的角速度值rate\_sp（这一段已经完成了所需要的角度变化，并将角度的变化值转换到了需要的角速度值）。并且把加速度值直接输出给attitude rate control，再经过角速度环的pid控制，输出值直接就给mixer，然后控制电机输出了。

        关于这些，主要还是需要理解这个控制过程：一方面是通过姿态解算部分获取的实时的姿态信息，并通过uORB模型把姿态信息发布出去；姿态控制部分订阅姿态解算得到的姿态信息。然后通过attitude control获取目标姿态和当前姿态的角度差值并经过算法处理得到对应的角速度值，并把这个角速度值输出给attitude rate control 最终获取到需求的控制量。输出给mixer。但是关于上述还是有一个迷惑的地方，就是在attitude control这个里面输出的是根据目标姿态计算的角速度值，然后再和attitude rate control 里面通过uORB获取的当前的角速度值做差得出角速度差值。。。。本身对这个比较懵逼。其实attitude control输出是需要达到这个误差角度时所需要的角速度值，用这个值与当前的角速度值做差，求出现在需要的角速度值而已。这个就是为什么控制角速度的原因，进而达到控制角度的效果。

## ****第四章 位置控制****

### 整体架构

位置控制是无人机飞控的核心算法之一，一方面根据commander中的flag标志位和Navigator中提供的航点信息进行控制（自主模式下），另一方面得到期望姿态角（setpoint）的四元数信息，给到姿态控制模块进行姿态控制。本文重点PX4飞控的位置控制的代码整体架构（mc\_pos\_control）,具体的控制算法将在后续文章中陆续奉上。   
位置控制模块的主函数：task\_main()   
1.订阅结构体orb\_subscribe（）。   
2.参数更新parameters\_update(true)   
3.while (!\_task\_should\_exit){}进入位置控制循环   
（1）加锁解锁控制参数重置   
if (\_control\_mode.flag\_armed && !was\_armed)则重置位置和高度sp，速度sp归零。   
\_control\_mode.flag\_armed=armed.armed;飞机启动后为真。while初始定义时bool was\_armed = false;初始化为假，意味着在飞机启动时进行位置和速度sp的重置。   
**注：根据加锁和解锁的判断，保证重新解锁后时位置参数被重置。**   
（2）获取当前位置和速度信息，作为反馈信息，进行之后的PID控制

if (PX4\_ISFINITE(\_local\_pos.x) &&

PX4\_ISFINITE(\_local\_pos.y) &&

PX4\_ISFINITE(\_local\_pos.z)) {

\_pos(0) = \_local\_pos.x;

\_pos(1) = \_local\_pos.y;

if (\_params.alt\_mode == 1 &&\_local\_pos.dist\_bottom\_valid) {

\_pos(2) = -\_local\_pos.dist\_bottom;

} else {

\_pos(2) = \_local\_pos.z;

}

}//更新当前位置坐标

if (PX4\_ISFINITE(\_local\_pos.vx) &&

PX4\_ISFINITE(\_local\_pos.vy) &&

PX4\_ISFINITE(\_local\_pos.vz)) {

\_vel(0) = \_local\_pos.vx;

\_vel(1) = \_local\_pos.vy;

if (\_params.alt\_mode == 1 && \_local\_pos.dist\_bottom\_valid) {

\_vel(2) = -\_local\_pos.dist\_bottom\_rate;

} else {

\_vel(2) = \_local\_pos.vz;

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22

注：\_params.alt\_mode 是MPC\_ALT\_MODE，默认为0，代表高度跟随，1代表地形跟随。地形跟随模式下，垂直方向上的参考基准为地面。   
（3）**位置sp**   
根据commander中不同的标志位选择不同的控制方式。   
control\_manual(dt);//手动模式   
control\_offboard(dt);//外部模式   
control\_auto(dt);//自动模式   
得到\_pos\_sp和yaw\_sp.   
在control\_auto中   
起飞时\_do\_reset\_alt\_pos\_flag = false;其它情况为真。貌似只有在manual中用到。从auto切换到manual时。   
进行起落架控制   
4.**速度SP**   
（1）非手动控制下，怠速模式下，不进行控制   
姿态角的俯仰、横滚为0，航向为当前航向，油门为0.

\_att\_sp.roll\_body = 0.0f;

\_att\_sp.pitch\_body = 0.0f;

\_att\_sp.yaw\_body = \_yaw;

\_att\_sp.thrust = 0.0f;

* 1
* 2
* 3
* 4

（2）手动控制，落地之后   
对位置和高度sp、三个方向上的积分量进行重置，同时姿态角sp为0，油门sp为0，航向角为当前角度。

\_att\_sp.roll\_body = 0.0f;

\_att\_sp.pitch\_body = 0.0f;

\_att\_sp.yaw\_body = \_yaw;

\_att\_sp.thrust = 0.0f;

* 1
* 2
* 3
* 4

注：**落地之后，为防止手动打杆翻机，将姿态角和油门都归零，航向角保持不变。**   
（3）正常情况的水平位置控制：   
①\_run\_pos\_control为真时，（初始化为真，只有在手动模式中和offboard中会赋值为假）   
**水平位置P控制，得到水平速度sp**

\_vel\_sp(0) = (\_pos\_sp(0) - \_pos(0)) \* \_params.pos\_p(0);

\_vel\_sp(1) = (\_pos\_sp(1) - \_pos(1)) \* \_params.pos\_p(1);

* 1
* 2

②目标跟随模式下   
目标水平速度：ft\_vel(\_pos\_sp\_triplet.current.vx, \_pos\_sp\_triplet.current.vy, 0);   
if (cos\_ratio > 0) {

ft\_vel \*= (cos\_ratio);//目标速度在无人setpoint速度方向上投影

ft\_vel += ft\_vel.normalized() \* 1.5f;}

* 1
* 2

实际上ft\_vel =ft\_vel \* (cos\_ratio)+ft\_vel \* (cos\_ratio).normalized() \* 1.5（比投影速度多一点）

\_vel\_sp(0) = fabs(ft\_vel(0)) > fabs(\_vel\_sp(0)) ? ft\_vel(0) : \_vel\_sp(0);

vel\_sp(1) = fabs(ft\_vel(1)) > fabs(\_vel\_sp(1)) ? ft\_vel(1) : \_vel\_sp(1);

* 1
* 2

二者取较大的速度   
③当前目标位置无效，速度有效时，速度sp就等于航点设定速度

\_vel\_sp(0) = \_pos\_sp\_triplet.current.vx;

\_vel\_sp(1) = \_pos\_sp\_triplet.current.vy;

* 1
* 2

（4）正常情况的高度位置控制

\_vel\_sp(2) = (\_pos\_sp(2) - \_pos(2)) \* \_params.pos\_p(2);

* 1

（5）速度限幅（在control auto 中也有进行限幅的控制）   
水平方向的和速度，不能大于水平方向上的最大速度，大于之后，对水平的xy方向上的速度进行等比例缩小。合速度等于最大速度。

float vel\_norm\_xy = sqrtf(\_vel\_sp(0) \* \_vel\_sp(0) +

\_vel\_sp(1) \* \_vel\_sp(1));

if (vel\_norm\_xy > \_params.vel\_max(0)) {

/\* note assumes vel\_max(0) == vel\_max(1) \*/

\_vel\_sp(0) = \_vel\_sp(0) \* \_params.vel\_max(0) / vel\_norm\_xy;

\_vel\_sp(1) = \_vel\_sp(1) \* \_params.vel\_max(1) / vel\_norm\_xy;

}

/\* make sure velocity setpoint is saturated in z\*/

if (\_vel\_sp(2) < -1.0f \* \_params.vel\_max\_up) {

\_vel\_sp(2) = -1.0f \* \_params.vel\_max\_up;

}

if (\_vel\_sp(2) > \_params.vel\_max\_down) {

\_vel\_sp(2) = \_params.vel\_max\_down;

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17

垂直方向上的速度，在最大上升速度和最小下降速度之间。   
（6）对于需要reset的情况，位置、高度、速度重置（在mission等自主模式中不会触发）

if (!\_control\_mode.flag\_control\_position\_enabled) {

\_reset\_pos\_sp = true;

}

if (!\_control\_mode.flag\_control\_altitude\_enabled) {

\_reset\_alt\_sp = true;

}

if (!\_control\_mode.flag\_control\_velocity\_enabled) {

\_vel\_sp\_prev(0) = \_vel(0);

\_vel\_sp\_prev(1) = \_vel(1);

\_vel\_sp(0) = 0.0f;

\_vel\_sp(1) = 0.0f;

control\_vel\_enabled\_prev = false;

}

if (!\_control\_mode.flag\_control\_climb\_rate\_enabled) {

\_vel\_sp(2) = 0.0f;

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17

（7）降落速度控制（见下一篇博客）   
（8）起飞速度控制（见下一篇博客）   
（9）加速度最大值对速度sp的限幅

水平：

math::Vector<2> acc\_hor;

acc\_hor(0) = (\_vel\_sp(0) - \_vel\_sp\_prev(0)) / dt;

acc\_hor(1) = (\_vel\_sp(1) - \_vel\_sp\_prev(1)) / dt;

if ((acc\_hor.length() > \_params.acc\_hor\_max) & !\_reset\_pos\_sp) {

acc\_hor.normalize();

acc\_hor \*= \_params.acc\_hor\_max;

math::Vector<2> vel\_sp\_hor\_prev(\_vel\_sp\_prev(0), \_vel\_sp\_prev(1));

math::Vector<2> vel\_sp\_hor = acc\_hor \* dt + vel\_sp\_hor\_prev;

\_vel\_sp(0) = vel\_sp\_hor(0);

\_vel\_sp(1) = vel\_sp\_hor(1);

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11

垂直：

float acc\_v = (\_vel\_sp(2) - \_vel\_sp\_prev(2)) / dt;

if ((fabsf(acc\_v) > 2 \* \_params.acc\_hor\_max) & !\_reset\_alt\_sp) {

acc\_v /= fabsf(acc\_v);

\_vel\_sp(2) = acc\_v \* 2 \* \_params.acc\_hor\_max \* dt + \_vel\_sp\_prev(2);

}

* 1
* 2
* 3
* 4
* 5
* 6

5.**油门sp**   
（1）积分项   
（2）加速度控制时，直接根据目标航点加速度得到油门值   
非加速度控制，利用PID参数得到thrust sp。（对于mission模式，都是非加速度控制）

if (\_control\_mode.flag\_control\_acceleration\_enabled && \_pos\_sp\_triplet.current.acceleration\_valid) {

thrust\_sp = math::Vector<3>(\_pos\_sp\_triplet.current.a\_x, \_pos\_sp\_triplet.current.a\_y, \_pos\_sp\_triplet.current.a\_z);

} else {

thrust\_sp = vel\_err.emult(\_params.vel\_p) + \_vel\_err\_d.emult(\_params.vel\_d) + thrust\_int;

}

* 1
* 2
* 3
* 4
* 5

（3）起飞、降落油门控制   
（4）最大加速度限幅   
水平：

if (thrust\_sp\_xy\_len > thrust\_xy\_max) {

float k = thrust\_xy\_max / thrust\_sp\_xy\_len;

thrust\_sp(0) \*= k;

thrust\_sp(1) \*= k;

saturation\_xy = true;

}

* 1
* 2
* 3
* 4
* 5
* 6

（5）重力补偿（垂直油门）

if (\_control\_mode.flag\_control\_climb\_rate\_enabled && !\_control\_mode.flag\_control\_velocity\_enabled) {

float att\_comp;

if (\_R(2, 2) > TILT\_COS\_MAX) {

att\_comp = 1.0f / \_R(2, 2);

} else if (\_R(2, 2) > 0.0f) {

att\_comp = ((1.0f / TILT\_COS\_MAX - 1.0f) / TILT\_COS\_MAX) \* \_R(2, 2) + 1.0f;

saturation\_z = true;

} else {

att\_comp = 1.0f;

saturation\_z = true;

}

thrust\_sp(2) \*= att\_comp;

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13

（6）总油门限幅，加速度超过最大值thrust\_abs > thr\_max   
垂直方向上的 推力向上，同时超过最大值 thr\_max时（0.9），水平方向上为0，垂直方向上等于最大值。   
垂直方向上的推力向上，没有超过最大值 thr\_max时，优先保证垂直推力，得到水平上的最大推力合力，对水平进行限幅。   
垂直方向上加速度向下时，则同时对三个方向上进行等比例限幅   
（7）更新积分项   
速度控制有效同时没有进行推力限幅时。   
由速度误差得到积分值，水平和垂直两个方向。   
6.**姿态角sp与四元数**   
（1）body\_z = -thrust\_sp / thrust\_abs;机体z轴与推力轴重合，   
求机体坐标系三个轴单位向量在地面坐标系下的坐标   
先求body\_z= -thrust\_sp / thrust\_abs;   
（2）机体坐标系y轴在水平面上的投影向量

y\_C(-sinf(\_att\_sp.yaw\_body), cosf(\_att\_sp.yaw\_body), 0.0f);

* 1

由偏航角求得，   
叉乘得到机体坐标系的x轴向量

body\_x = y\_C % body\_z;

* 1

（3）机体x轴和机体z轴叉乘的到真正的机体y轴向量。

body\_y = body\_z % body\_x;

* 1

（4）这三个向量组成旋转矩阵R=euler   
\_att\_sp.roll\_body = euler(0);\_att\_sp.pitch\_body = euler(1);   
得到四元数，给到姿态sp   
7.**手动控制下的姿态角sp**

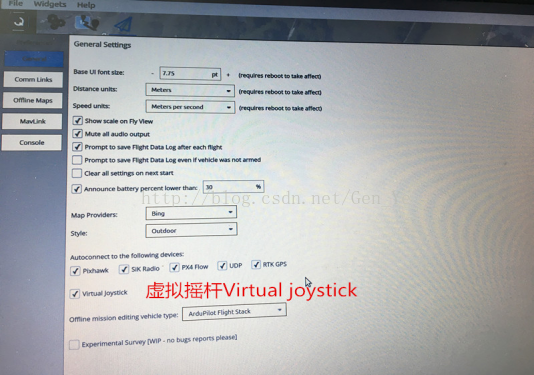
\_att\_sp.yaw\_sp\_move\_rate = \_manual.r \* yaw\_rate\_max;

yaw\_target = \_wrap\_pi(\_att\_sp.yaw\_body + \_att\_sp.yaw\_sp\_move\_rate \* dt);

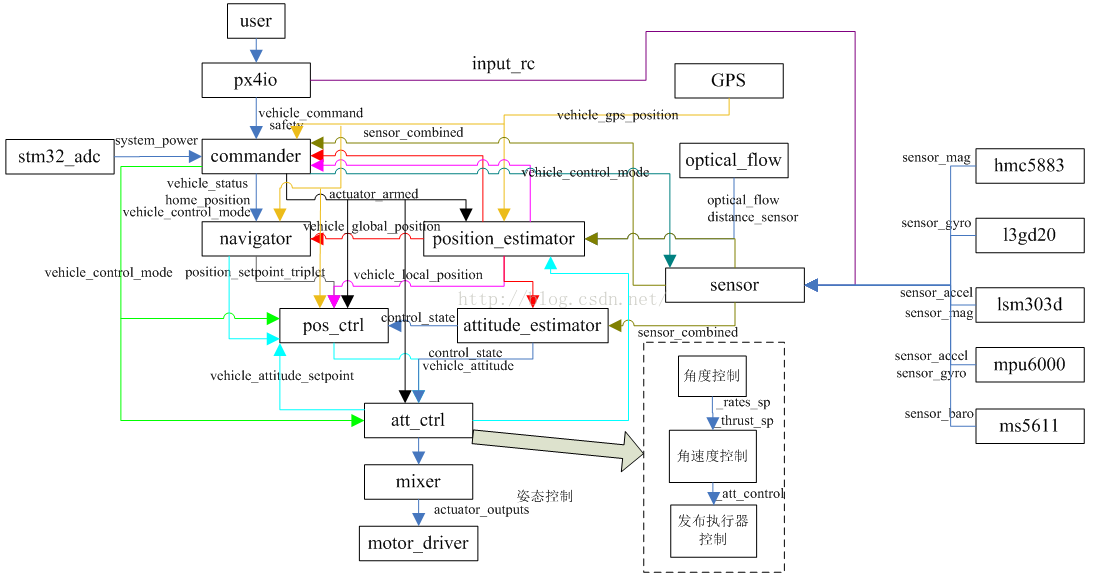
\_att\_sp.roll\_body = \_manual.y \* \_params.man\_roll\_max;

\_att\_sp.pitch\_body = -\_manual.x \* \_params.man\_pitch\_max;//x为正时，摇杆往前推，俯仰角为负

### ****Offboard位置控制****



## 第五章 源码模块关系图：



**系统命令程序**

[mavlink](http://link.zhihu.com/?target=https%3A//pixhawk.org/firmware/apps/mavlink) –通过串口发送和接收mavlink信息

[sdlog2](http://link.zhihu.com/?target=https%3A//pixhawk.org/firmware/apps/sdlog2) –保存系统日志/飞行数据到SD卡

[tests](http://link.zhihu.com/?target=https%3A//pixhawk.org/firmware/apps/tests) –测试系统中的测试程序

[top](http://link.zhihu.com/?target=https%3A//pixhawk.org/firmware/apps/top) –列出当前的进程和CPU负载

[uORB](http://link.zhihu.com/?target=https%3A//pixhawk.org/firmware/apps/uorb) – 微对象请求代理器-分发其他应用程序之间的信息

drivers –底层驱动程序

modules –算法模块

[mkblctrl](http://link.zhihu.com/?target=https%3A//pixhawk.org/firmware/apps/mkblctrl)–blctrl电子模块驱动

[esc\_calib](http://link.zhihu.com/?target=https%3A//pixhawk.org/firmware/apps/esc_calib) –ESC的校准工具

[fmu](http://link.zhihu.com/?target=https%3A//pixhawk.org/firmware/apps/fmu) –FMU引脚输入输出定义

[gpio\_led](http://link.zhihu.com/?target=https%3A//pixhawk.org/firmware/apps/gpio_led) –GPIO LED驱动

[gps](http://link.zhihu.com/?target=https%3A//pixhawk.org/firmware/apps/gps) –GPS接收器驱动

[pwm](http://link.zhihu.com/?target=https%3A//pixhawk.org/firmware/apps/pwm) –PWM的更新速率命令

[sensors](http://link.zhihu.com/?target=https%3A//pixhawk.org/firmware/apps/sensors) –传感器应用

[px4io](http://link.zhihu.com/?target=https%3A//pixhawk.org/firmware/apps/px4io) –[px4io](http://link.zhihu.com/?target=http%3A//translate.baiducontent.com/transpage%3Fcb%3DtranslateCallback%26ie%3Dutf8%26source%3Durl%26query%3D%252Fmodules%252Fpx4io%26from%3Den%26to%3Dzh%26token%3D%26monLang%3Dzh)驱动

[uavcan](http://link.zhihu.com/?target=https%3A//pixhawk.org/firmware/apps/uavcan) –[uavcan](http://link.zhihu.com/?target=http%3A//translate.baiducontent.com/transpage%3Fcb%3DtranslateCallback%26ie%3Dutf8%26source%3Durl%26query%3Dhttp%253A%252F%252Fuavcan.org%26from%3Den%26to%3Dzh%26token%3D%26monLang%3Dzh)驱动

**飞行安全状态机和导航**

[commander](http://link.zhihu.com/?target=https%3A//pixhawk.org/firmware/apps/commander) –主要飞行安全状态机

[navigator](http://link.zhihu.com/?target=https%3A//pixhawk.org/firmware/apps/navigator) –任务，失效保护和RTL导航仪

**估计姿态和位置**

[attitude\_estimator\_ekf](http://link.zhihu.com/?target=https%3A//pixhawk.org/firmware/apps/attitude_estimator_ekf) –基于EKF的姿态估计

[ekf\_att\_pos\_estimator](http://link.zhihu.com/?target=https%3A//pixhawk.org/firmware/apps/ekf_att_pos_estimator) –基于EKF的姿态和位置估计

[position\_estimator\_inav](http://link.zhihu.com/?target=https%3A//pixhawk.org/firmware/apps/position_estimator_inav)–惯性导航的位置估计

**multirotor姿态和位置控制器**

[mc\_att\_control](http://link.zhihu.com/?target=https%3A//pixhawk.org/firmware/apps/mc_att_control)–multirotor姿态控制器

[mc\_pos\_control](http://link.zhihu.com/?target=https%3A//pixhawk.org/firmware/apps/mc_pos_control) –multirotor位置控制器

**fixedwing姿态和位置控制器**

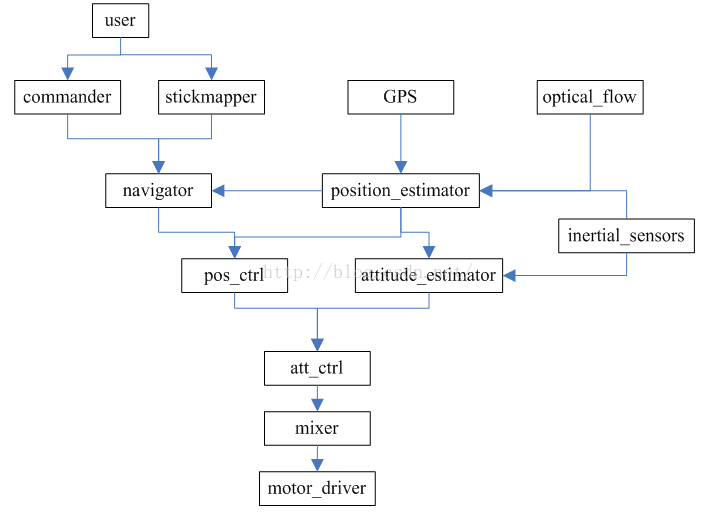
[fw\_att\_control](http://link.zhihu.com/?target=https%3A//pixhawk.org/firmware/apps/fw_att_control) –固定翼飞机的姿态控制

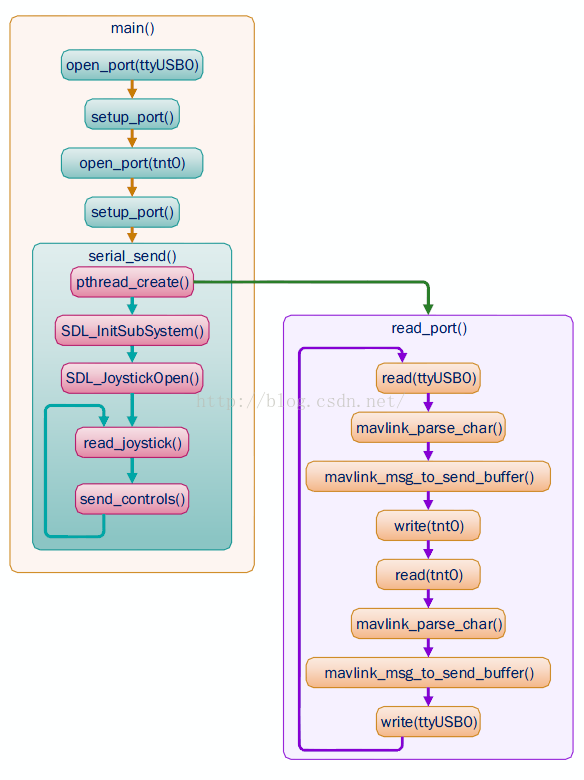
[fw\_pos\_control\_l1](http://link.zhihu.com/?target=https%3A//pixhawk.org/firmware/apps/fw_pos_control_l1) –固定翼位置控制器

**垂直起降姿态控制器**

[vtol\_att\_control](http://link.zhihu.com/?target=https%3A//pixhawk.org/firmware/apps/vtol_att_control) –垂直起降姿态控制器

## 第六章 源码流程图





## 第七章 PX4源码启动分析

**一、开篇**

       ardupilot和pixhawk原生代码到底有什么区别和联系？

答：单独的pixhawk原生代码是可以正常飞行的，其上电以后在rcS（nsh的启动脚本，其存储在ROMFS中，后面会详细介绍）中开启相应的进程对无人机进行姿态等复杂的控制，关于这个大家最先认识到的应该是各种飞行模式的确定，

ardupilot很多人称之为APM，但是这个APM的HAL层也可以使用pixhawk硬件，此APM非彼apm，官网有介绍。Ardupilot是基于pixhawk原生代码开发的上层应用，即有了常见的mian和loop函数，ardupilot把底层的PX4Firmware和NuttX等都抽象为HAL，开发者只需要关心上层应用，基于pixhawk原生设计上层控制逻辑。ardupilot其实是开源的、玩具级飞控，它为了兼容各种乱七八糟的板子、机型，有很多冗余的代码，这些冗余的代码让我们理解飞控本身增加了难度和学习成本。如果我们需要一款专业的飞控，完全可以去掉这些冗余的代码，只保留我们需要的精简过的功能，这样也就能大大降低它代码的复杂程度。从程序设计的方法论来看（面向对象）：就是要把一个个对象确立好，把问题的复杂程度控制住，才能设计出逻辑清晰的程序出来。

**三、实验平台**

Software Version：ArduCopter（Ver\_3.3）

Hardware Version：pixhawk

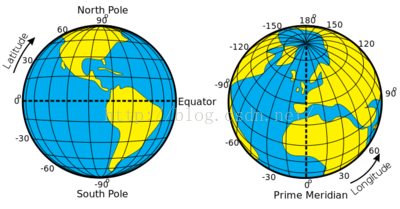
IDE：eclipse Juno （Windows）

**四、基本知识介绍**

1、关于坐标系

1）GeographicCoordinate System

        Represents position on earth with alongitude and latitude value (Geographic\_coordinate\_system). Additionally thealtitude may may be included. The altitude can be expressed as distance fromthe earth center or as altitude above the mean sea level. All in all, thisgives a spherical coordinate system.



2）Earth-Fixed frame

        Cartesian coordinate system at the center of the earth. Thepositive z axis goes through the north pole. The x and y axes are on theequatorial plane.

3）Body Fixed Frame

        The x axis points in forward (defined bygeometry and not by movement) direction. (= roll axis)

        The y axis points to the right(geometrically) (= pitch axis)

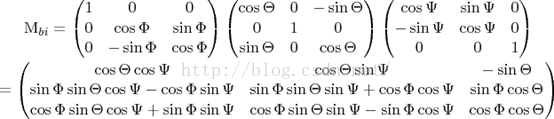
        The z axis points downwards (geometrically)(= yaw axis)

4）EulerAngles

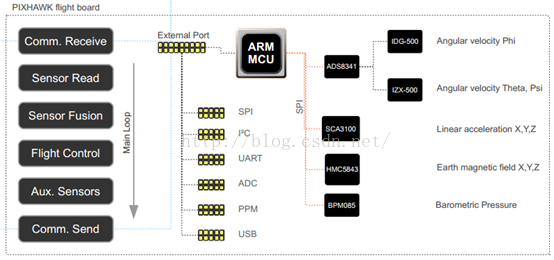
        Usually a conversion between a earth fixed“ground” frame and the body fixed “in-air” frame is described via Euler-Angles.There are multiple conventions of the Euler angles.

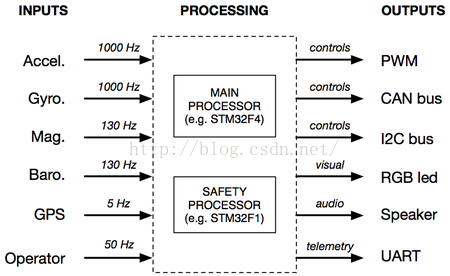
        The rotation order for the Tait-Bryan angles is Z Y’X” (see thefigure):  
rotation of  around Z (yaw)  
rotation of  around Y' (pitch)  
rotation of  around X” (roll)



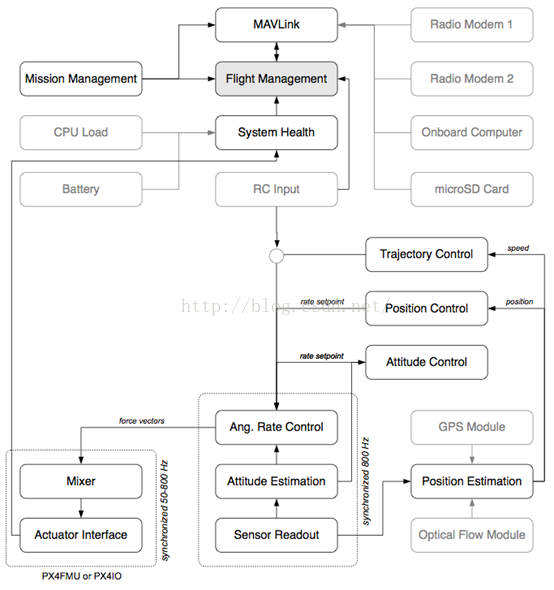


2、pixhawk的HAL





3、pixhawk的控制图



**五、正文**

1、关于两个控制器和任务优先级

        在PX4Firmware/src/modules中的mc\_att\_control：姿态控制器和mc\_pos\_control位置控制器（mc：multicopter），整个系统都是围绕着这两个控制器。

        mc\_att\_control – Multirotor attitude controller

        mc\_pos\_control – Multirotor position controller

        The PX4 firmware is organized in priority bands:

1) (interrupt level) fast sensordrivers

2) watchdog/system state monitors

3) actuator outputs (PWM outputdriver thread, IO comms sender thread)

4) attitude controller(s)

5) slow/blocking sensor drivers(must not block attitude controllers)

6) destination/positioncontroller(s)

7) default priority - generic usercode, shell commands, random crap, all RR scheduled

8) logger, parameter syncer

9) idle

2、  关于.mk文件（**必须深入理解**）

        了解整个代码运行过程的最简便方法就是通过其makefile了解文件的包含关系和调用关系。请仔细阅读arducopter中的.mk和modules/PX4Firmware中的.mk文件。.mk文件就是在[**Linux**](http://lib.csdn.net/base/linux)下编写的一些脚本文件（makefile），类似于编译过程中的条件编译，便于直接使用命令行进行编译链接。

        1）首先介绍上层应用的.mk。在ardupilot/mk/PX4/Tool中px4\_common.mk文件中是关于所有PX4平台的通用编译文件，config\_px4fmu\_v2\_APM.mk中针对pixhawk的特有硬件的编译文件。下面实例代码是config\_px4fmu\_v2\_APM.mk中的代码，.mk脚本主要目的是为了选择需要编译的**MODULES**。

1. # Makefile for the px4fmu-v2\_APM configuration
2. include $(SKETCHBOOK)/mk/PX4/px4\_common.mk
3. MODULES += drivers/lsm303d
4. MODULES += drivers/l3gd20
5. MODULES += drivers/mpu9250
6. MODULES += drivers/boards/px4fmu-v2
7. MODULES += drivers/pwm\_input
8. MODULES += modules/uavcan
9. MODULES += lib/mathlib
10. MODULES += drivers/px4io
11. MODULES += drivers/px4flow
12. MODULES += drivers/oreoled

        Makefile的文件夹，指定编译的规则，编译顺序：

ardupilot/ArduCopter/Makefile àmk/apm.mkà environ.mk, targets.mk, sketch\_sources.mk, board\_px4.mkà find\_tools.mk,**px4\_targets.mk。**

        尤其是上面的这个px4\_targets.mk文件，该文件中涉及了所有的需要编译的文件，最后使用命令行px4-v2编译整个工程，它是最终的编译过程的执行者，注意其中px4-v2规则的详细执行情况，它会去调用PX4Firmware的makefile。下面针对pixhawk硬件平台摘录比较重要一部分。

1. ………
2. /\*指定PX4Firmware、PX4NuttX、uavcan \*/
3. # these can be overridden in developer.mk
4. PX4FIRMWARE\_DIRECTORY ?= $(SKETCHBOOK)/modules/PX4Firmware
5. PX4NUTTX\_DIRECTORY ?= $(SKETCHBOOK)/modules/PX4NuttX
6. UAVCAN\_DIRECTORY ?= $(SKETCHBOOK)/modules/uavcan
7. PX4\_ROOT := $(shell cd $(PX4FIRMWARE\_DIRECTORY) && pwd)
8. NUTTX\_ROOT := $(shell cd $(PX4NUTTX\_DIRECTORY) && pwd)
9. NUTTX\_SRC := $(NUTTX\_ROOT)/nuttx/
10. UAVCAN\_DIR=$(shell cd $(UAVCAN\_DIRECTORY) && pwd)/
11. ………
12. /\*选择硬件平台\*/
13. # we have different config files for V1 and V2
14. PX4\_V1\_CONFIG\_FILE=$(MK\_DIR)/PX4/config\_px4fmu-v1\_APM.mk
15. PX4\_V2\_CONFIG\_FILE=$(MK\_DIR)/PX4/config\_px4fmu-v2\_APM.mk
16. PX4\_V4\_CONFIG\_FILE=$(MK\_DIR)/PX4/config\_px4fmu-v4\_APM.mk
17. ………
18. /\*针对pixhawk，请详细了解该部分的实现\*/
19. px4-v2: $(BUILDROOT)/make.flags CHECK\_MODULES $(MAVLINK\_HEADERS) $(PX4\_ROOT)/Archives/px4fmu-v2.export $(SKETCHCPP) module\_mk px4-io-v2
20. $(RULEHDR)/\* PX4\_ROOT  = ardupilot/modules/PX4Firmware\*/
21. $(v) rm -f $(PX4\_ROOT)/makefiles/$(PX4\_V2\_CONFIG\_FILE)
22. $(v) cp $(PX4\_V2\_CONFIG\_FILE) $(PX4\_ROOT)/makefiles/nuttx/
23. $(PX4\_MAKE) px4fmu-v2\_APM
24. $(v) /bin/rm -f $(SKETCH)-v2.px4
25. $(v) arm-none-eabi-size $(PX4\_ROOT)/Build/px4fmu-v2\_APM.build/firmware.elf
26. $(v) cp $(PX4\_ROOT)/Images/px4fmu-v2\_APM.px4 $(SKETCH)-v2.px4
27. $(v) $(SKETCHBOOK)/Tools/scripts/add\_git\_hashes.py $(HASHADDER\_FLAGS) "$(SKETCH)-v2.px4" "$(SKETCH)-v2.px4"
28. $(v) echo "PX4 $(SKETCH) Firmware is in $(SKETCH)-v2.px4"
29. /\*最后编译生成的\*.px4文件，直接下载到飞控板中即可实现飞行控制\*/
30. ………
31. px4-clean: clean CHECK\_MODULES px4-archives-clean px4-cleandep
32. $(v) /bin/rm -rf $(PX4\_ROOT)/makefiles/build $(PX4\_ROOT)/Build $(PX4\_ROOT)/Images/\*.px4 $(PX4\_ROOT)/Images/\*.bin
33. $(v) /bin/rm -rf $(PX4\_ROOT)/src/modules/uORB/topics $(PX4\_ROOT)/src/platforms/nuttx/px4\_messages
34. px4-cleandep: clean
35. $(v) find $(PX4\_ROOT)/Build -type f -name '\*.d' | xargs rm -f
36. $(v) find $(UAVCAN\_DIRECTORY) -type f -name '\*.d' | xargs rm -f
37. $(v) find $(SKETCHBOOK)/$(SKETCH) -type f -name '\*.d' | xargs rm -f
38. px4-v2-upload-solo: px4-v2
39. scp $(SKETCH)-v2.px4 root@10.1.1.10:/tmp/
40. ssh root@10.1.1.10 PYTHONUNBUFFERED=1 loadPixhawk.py /tmp/ArduCopter-v2.px4
41. ssh root@10.1.1.10 rm /tmp/ArduCopter-v2.px4;
42. ………
43. px4-v2-upload: px4-v2
44. $(RULEHDR)
45. $(v) $(PX4\_MAKE) px4fmu-v2\_APM upload
46. px4-upload: px4-v1-upload
47. px4-archives-clean:
48. $(v) /bin/rm -rf $(PX4\_ROOT)/Archives

        2）接下来就是PX4Firmware中的.mk。如果仔细了解了上面所讲述的，那么这部分就很easy了，不再细讲，自己去看代码吧。需要说明的是在PX4Firmware的makefile文件中有一个README.txt文本，该文本详细的介绍了PX4Firmware中的.mk的调用关系和作用。补充一点，PX4Firmware/makefile/nuttx中的config\_px4fmu-v2\_default.mk包含了所有基本的模块，从这里再一次验证了开篇讲述的ardupilot的代码是基于pixhawk原生代码PX4Firmware“阉割”而来的。所以在后期做开发的过程中编写应用程序只是按照ardupilot现有的子模块添加自己的就可以，参考mc\_att\_control实现。

        在PX4Firmware/src/modules中添加一个新的文件夹，命名为summer

        在summer文件夹中创建module.mk文件，并输入以下内容：

1. MODULE\_COMMAND = summer
2. SRCS = summer.cpp

        在summer文件夹中创建summer.c文件，编译需要实现的功能代码。

        注册新添加的应用到NuttShell中：PX4Firmware/makefiles/nuttx/config\_px4fmu-v2\_default.mk文件中添加如下内容：

1. MODULES += modules/summer

**任务的启动就是靠NuttShell的内建命令启动的，包含rcS，特别是rcS一定要了解，下面会详细介绍。**

3、关于编译链接库（上层应用库 libraries）

        在ardupilot/Arducopter中有一个make.inc文件，指定了编译ArduCopter时需要编译哪些依赖库，它对系统及硬件进行了抽象，以获取更好的可移植性。

<http://dev.ardupilot.com/wiki/apmcopter-programming-libraries/>

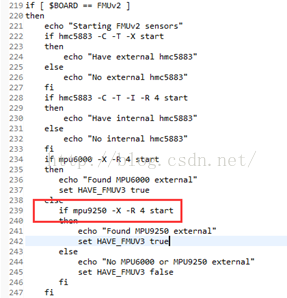
        如果想添加或者删除编译的文件，就在此处进行修改。如下是官方默认的。

1. LIBRARIES += AP\_Common
2. LIBRARIES += AP\_Menu
3. LIBRARIES += AP\_Param
4. LIBRARIES += StorageManager
5. LIBRARIES += GCS
6. LIBRARIES += GCS\_MAVLink
7. LIBRARIES += AP\_SerialManager
8. LIBRARIES += AP\_GPS
9. LIBRARIES += DataFlash
10. LIBRARIES += AP\_ADC
11. LIBRARIES += AP\_Baro
12. LIBRARIES += AP\_Compass
13. LIBRARIES += AP\_Math
14. LIBRARIES += AP\_InertialSensor
15. LIBRARIES += AP\_AccelCal
16. LIBRARIES += AP\_AHRS
17. LIBRARIES += AP\_NavEKF
18. ………

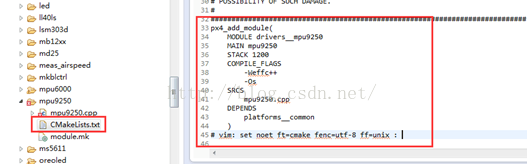
4、启动脚本（rcS：有两个，比较重要，任务分工）

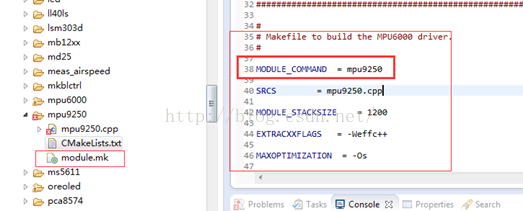
        所谓启动脚本其实是用shellscript写的启动文件以前写过比较简单的脚本文件，还是参考的鸟哥的那本书，还有就是[**linux**](http://lib.csdn.net/base/linux)启动文件也是用的这个写的，有兴趣的可以去Linux平台下查看一下，会写C的学这个很快，该死的C++什么时候能学会啊。

        1）其中之一就是负责较为底层driver里面的（比如mpu9250的register map、驱动），在使用命令px4-v2-upload以后，会在modules/PX4Firmware目录下生成build文件夹，内部是编译生成的文件（不编译没有），在modules/PX4Firmware/build/px4fmu-v2\_APM.build/romfs\_scratch/init.d文件下就是rcS和rc.APM，这个rcS是自启动脚本（setMODE autostart）。rcS脚本会挂载SD卡（这就是没有SD飞控板不能使用的原因所在），然后跳转rc.APM中，该脚本会在pixhawk系统中创建binfs，判断硬件类型，然后启动orb（对，就是那个uORB），然后启动各种传感器，然后咔咔咔，咔咔咔，完了，自己去看脚本吧。摘一部分鉴赏，关于shell脚本的编写自己百度吧，跟[**C语言**](http://lib.csdn.net/base/c)的风格很像。



        其中类似于mpu9250 start就启动了mpu9250模块的驱动，理解该部分需要结合modules/PX4Firmware/src/drivers/mpu9250中的CMakeLists.txt和module.mk这两个，其中CMakeLists.txt是使用Cmake写的，和module.mk类似，在最新的pixhawk原生代码中删除了make，使用清一色的Cmake。





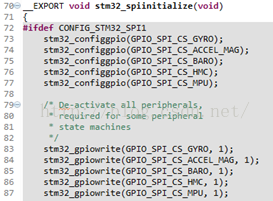
        上面标红的MODULE\_COMMAND = mpu9250就是为了以后可以直接在.mk文件直接使用的命令mpu9250。理解了上述以后，其他的driver中的所有的传感器模块、各种总线和modules/PX4Firmware/src/modules中的各种算法（mc\_att\_control、land\_detector等等）都是按照此类方法实现的，不在赘述，但是这个rcS只是负责启动driver中的，modules/PX4Firmware/src/modules需要靠下面的一个rcS启动，两个rcS各自分工。全部都是.mk啊，.mk，.mk~~~~

        下一阶段主要应该就是放在这个modules/PX4Firmware/src/modules内部了，整个控制响应过程全靠它了，尤其是commander，校磁什么的都在里面，各种calibration，见过2000+行的函数么？！怕么？ ~~~   168MHZ的主频呢，怕啥

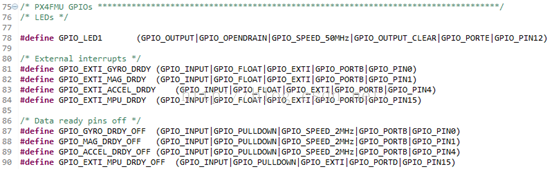
        PS：需要注意的是，在modules/PX4Firmware/src/drivers/boards/px4fmu-v2中有几个.c文件，其中px4fmu\_spi.c和board\_config.h。

**px4fmu\_spi.c 会在入口函数\_\_start()中被调用初始化SPI，下面会提到。**

1. \* Name: stm32\_spiinitialize
2. \* Description:
3. \*   Called to configure SPI chip select GPIO pins for the PX4FMU board.



        board\_config.h：如下的代码是不是很熟悉，用过单片机的肯定都见过，尤其是pixhawk使用的又是STM32，再加上ST公司的芯片在各个高校中的普及程度，虽然我没用过，我是菜鸟。



        2）另外一个rcS就是modules/PX4Firmware/ROMFS/px4fmu\_common/init.d中的rcS，PX4FMU的自启动文件，它主要是负责启动modules/PX4Firmware/modules里面的关于各自控制算法的模块。同时，该rcS会以脚本的形式启动同一级目录中的rc.mc\_apps、rc\_io等等。详细实现过程参见源代码。

        地址：<https://github.com/PX4/Firmware/tree/master/ROMFS/px4fmu_common/init.d>

        举一例说明问题，rc.mc\_apps：关于姿态估计、位置估计和姿态控制和位置控制的，源代码如下。

1. #!nsh
2. # Standard apps for multirotors:
3. # att & pos estimator, att & pos control.
4. # The system is defaulting to INAV\_ENABLED = 1
5. # but users can alternatively try the EKF-based
6. # filter by setting INAV\_ENABLED = 0
7. if param compare INAV\_ENABLED 1
8. then
9. attitude\_estimator\_q start  //姿态估计
10. position\_estimator\_inav start//位置估计
11. else
12. if param compare LPE\_ENABLED 1
13. then
14. attitude\_estimator\_q start
15. local\_position\_estimator start
16. else
17. ekf2 start
18. fi
19. fi
20. if mc\_att\_control start //姿态控制
21. then
22. else
23. # try the multiplatform version
24. mc\_att\_control\_m start
25. fi
26. if mc\_pos\_control start //位置控制
27. then
28. else
29. # try the multiplatform version
30. mc\_pos\_control\_m start
31. fi
32. # Start Land Detector
33. land\_detector start multicopter //启动落地检测

        应该能了解代码中的xxxxx start的含义了吧~~~~还不了解的话请拨打114，有专人为您解答

六、**再深入一点**

1、主控STM32F4的选择（肯定很多人不知道这个）

        在ardupilot/modules/PX4NuttX/nuttx/arch/arm/src/stm32中，主要是通过在stm32\_start.c中包含头文件<arch/board/board.h>，board.h中包含<stm32.h>并且配置了STM32的时钟（Clock），stm32.h中包含<chip.h>。另外，stm32.h中还包含了关于stm32的各种Peripherals的头文件，即各种外设（spi、can、uart、i2c等等）的驱动。

        在ardupilot/modules/PX4NuttX/nuttx/arch/arm/src/stm32文件下有头文件chip.h，内部通过条件编译执行进行主控MCU的选择。实例代码如下。

1. /\* STM32 F2 Family \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
2. #elif defined(CONFIG\_STM32\_STM32F20XX)
3. #  include "chip/stm32f20xxx\_pinmap.h"
4. /\* STM32 F3 Family \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
5. #elif defined(CONFIG\_STM32\_STM32F30XX)
6. #  include "chip/stm32f30xxx\_pinmap.h"
7. /\* STM32 F4 Family \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
8. #elif defined(CONFIG\_STM32\_STM32F40XX)
9. #  include "chip/stm32f40xxx\_pinmap.h"
10. #else
11. #  error "No pinmap file for this STM32 chip"
12. #endif

2、入口函数（\_\_start()）（这个就更不知道了吧）

        在ardupilot/modules/PX4NuttX/nuttx/arch/arm/src/stm32文件下有定义文件stm32\_start.c，内部有个上电/重启的函数入口**\_\_start(void)**。仔细看，一行行的看，别偷懒，代码附加列出了几个小问题，检查自己一下都会么？！！！

1. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
2. \* Public Functions
3. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
4. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
5. \* Name: \_start
6. \*
7. \* Description:
8. \*   This is the reset entry point.
9. \*
10. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
11. void \_\_start(void)
12. {
13. const uint32\_t \*src;
14. uint32\_t \*dest;
15. #ifdef CONFIG\_ARMV7M\_STACKCHECK
16. /\* Set the stack limit before we attempt to call any functions \*/
17. \_\_asm\_\_ volatile ("sub r10, sp, %0" : : "r" (CONFIG\_IDLETHREAD\_STACKSIZE - 64) : ); /\*限制栈大小，可以防止递归把栈内存浪费完了，知道原因么？！\*/
18. #endif
19. /\* Configure the uart so that we can get debug output as soon as possible \*/
20. stm32\_clockconfig();
21. stm32\_fpuconfig();
22. stm32\_lowsetup();
23. stm32\_gpioinit();
24. showprogress('A');
25. /\* Clear .bss.  We'll do this inline (vs. calling memset) just to be
26. \* certain that there are no issues with the state of global variables.
27. inline的好处。
28. \*/
29. for (dest = &\_sbss; dest < &\_ebss; )
30. {
31. \*dest++ = 0;
32. }
33. showprogress('B');
34. /\* Move the intialized data section from his temporary holding spot in
35. \* FLASH into the correct place in SRAM.  The correct place in SRAM is
36. \* give by \_sdata and \_edata.  The temporary location is in FLASH at the
37. \* end of all of the other read-only data (.text, .rodata) at \_eronly.
38. \* 知道上述所讲的几个segments的分布么？！\*/
39. for (src = &\_eronly, dest = &\_sdata; dest < &\_edata; )
40. {
41. \*dest++ = \*src++;
42. }
43. showprogress('C');
44. /\* Perform early serial initialization \*/
45. #ifdef USE\_EARLYSERIALINIT
46. up\_earlyserialinit();
47. #endif
48. showprogress('D');
49. /\* For the case of the separate user-/kernel-space（知道几比几么？） build, perform whatever
50. \* platform specific initialization of the user memory is required.
51. \* Normally this just means initializing the user space .data and .bss
52. \* segments.
53. \*/
54. #ifdef CONFIG\_NUTTX\_KERNEL
55. stm32\_userspace();
56. showprogress('E');
57. #endif
58. /\* Initialize onboard resources \*/
59. stm32\_boardinitialize();//初始化SPI（飞控板上的传感器都是通过SPI通信）和LEDs。
60. // 还记上面提到的px4fmu\_spi.c么？！
61. showprogress('F');
62. /\* Then start NuttX \*/
63. showprogress('\r');
64. showprogress('\n');
65. os\_start();//还记得上一篇博文的这个么？
66. /\* Shoulnd't get here \*/
67. for(;;);
68. }

        轻松一下：飞控板上为什么使用SPI而不用I2C?如下是官方原版解释，说白了就是速度问题，如下权当练练English~~~

        I2C was not intended by Philips as a high rate sensor bus - this is what SPI has been invented for. Running multiple sensors at a suitably high rateover I2C is not recommended. Our hardware is designed to rely on SPI for allcritical sensors, and is available globally. Given the individual cost of thediscovery kit and sensor boards and power supply, one of the available Pixhawkkits is actually not more expensive, and will save a lot of trouble duringdevelopment and operation.

3、 Init sensors

        飞控板是干嘛的？

        控制无人机飞行滴（逼格高点就是通过各种传感器检测无人机的实时姿态信息和位置信息并结合MCU输出控制信息）。

        那sensor使用前要干嘛呢？

        ！#￥@￥！@%￥！#%

        初始化，初始化，初始化~~~~

        上面那么多传感器必须得初始化以后才能使用啊，所以需要先初始化用到的所有的传感器，在哪配置呢，你找到没?! 在ardupilot/modules/PX4Firmware/src/modules/sensors文件中有定义文件sensor.c（关于飞控板中的各个sensor的初始化函数，以task的方式启动，初始化完成以后kill掉这个task，都是POSIX接口的API）。

        首先是sensors\_main函数

1. int sensors\_main(int argc, char \*argv[])
2. {
3. ………
4. if (OK != sensors::g\_sensors->start()) {  //2293 line number
5. delete sensors::g\_sensors;
6. sensors::g\_sensors = nullptr;
7. warnx("start failed");
8. return 1;
9. }
10. ………

        然后捏：start()函数

1. Int Sensors::start()
2. {
3. ASSERT(\_sensors\_task == -1);
4. /\* start the task \*/
5. /\*创建任务进程对sensors进行初始化以后，在task\_main()函数执行的最后会调用px4\_task\_exit()函数退出该任务进程\*//\* px4\_task\_spawn\_cmd()创建任务的接口函数是posix接口的，具体实现参见源码\*/
6. \_sensors\_task = px4\_task\_spawn\_cmd("sensors", SCHED\_DEFAULT,
7. SCHED\_PRIORITY\_MAX - 5,
8. 2000,
9. (px4\_main\_t)&Sensors::task\_main\_trampoline,
10. nullptr);
11. /\* wait until the task is up and running or has failed \*/
12. while (\_sensors\_task > 0 && \_task\_should\_exit) {
13. usleep(100);
14. }
15. if (\_sensors\_task < 0) {
16. return -ERROR;
17. }
18. return OK;
19. }

        然后捏：task\_main\_trampoline()函数

1. Void Sensors::task\_main\_trampoline(int argc, char \*argv[])
2. {
3. sensors::g\_sensors->task\_main();
4. }

        再然后捏：task\_main()函数，大牌终于出场了

Void Sensors::task\_main()

1. {
2. /\* start individual sensors \*/
3. int ret = 0;
4. do { /\* create a scope to handle exit with break \*//\*do\_while用的精妙\*/
5. ret = accel\_init();/\* return 0 \*/
6. if (ret) { break; }
7. ret = gyro\_init();
8. if (ret) { break; }
9. ret = mag\_init();
10. if (ret) { break; }
11. ret = baro\_init();
12. if (ret) { break; }
13. ret = adc\_init();
14. if (ret) { break; }
15. break;
16. } while (0);
17. ………//2059
18. /\*
19. \* do subscriptions  这就是所谓的IPC使用的uORB模型（ publish\_subscribe）吧 ，我也不懂
20. \*/
21. unsigned gcount\_prev = \_gyro\_count;
22. ………
23. \_gyro\_count = init\_sensor\_class(ORB\_ID(sensor\_gyro), &\_gyro\_sub[0],
24. &raw.gyro\_priority[0], &raw.gyro\_errcount[0]);
25. ………
26. /\* get a set of initial values \*/
27. accel\_poll(raw);//通过uORB模型获取acc值
28. gyro\_poll(raw);
29. mag\_poll(raw);
30. baro\_poll(raw);
31. diff\_pres\_poll(raw);
32. parameter\_update\_poll(true /\* forced \*/);
33. rc\_parameter\_map\_poll(true /\* forced \*/);
34. ………
35. /\* check vehicle status for changes to publication state \*/
36. vehicle\_control\_mode\_poll();
37. /\* the timestamp of the raw struct is updated by the gyro\_poll() method \*/
38. /\* copy most recent sensor data \*/
39. gyro\_poll(raw);
40. accel\_poll(raw);
41. mag\_poll(raw);
42. baro\_poll(raw);
43. ………
44. /\* check battery voltage \*/
45. adc\_poll(raw);
46. diff\_pres\_poll(raw);
47. /\* Inform other processes that new data is available to copy \*/
48. if (\_publishing && raw.timestamp > 0) {
49. orb\_publish(ORB\_ID(sensor\_combined), \_sensor\_pub, &raw);
50. }
51. /\* keep adding sensors as long as we are not armed,
52. \* when not adding sensors poll for param updates
53. \*/
54. if (!\_armed && hrt\_elapsed\_time(&\_last\_config\_update) > 500 \* 1000) {
55. \_gyro\_count = init\_sensor\_class(ORB\_ID(sensor\_gyro), &\_gyro\_sub[0],
56. &raw.gyro\_priority[0], &raw.gyro\_errcount[0]);
57. \_mag\_count = init\_sensor\_class(ORB\_ID(sensor\_mag), &\_mag\_sub[0],
58. &raw.magnetometer\_priority[0], &raw.magnetometer\_errcount[0]);
59. \_accel\_count = init\_sensor\_class(ORB\_ID(sensor\_accel), &\_accel\_sub[0],
60. &raw.accelerometer\_priority[0], &raw.accelerometer\_errcount[0]);
61. \_baro\_count = init\_sensor\_class(ORB\_ID(sensor\_baro), &\_baro\_sub[0],
62. &raw.baro\_priority[0], &raw.baro\_errcount[0]);
63. \_last\_config\_update = hrt\_absolute\_time();
64. } else {
65. /\* check parameters for updates \*/
66. parameter\_update\_poll();
67. /\* check rc parameter map for updates \*/
68. rc\_parameter\_map\_poll();
69. }
70. /\* Look for new r/c input data \*/
71. rc\_poll();
72. perf\_end(\_loop\_perf);
73. }
74. warnx("exiting.");
75. \_sensors\_task = -1;
76. px4\_task\_exit(ret);/\* px4\_task\_exit ()终止任务的接口函数也是posix接口的\*/
77. }

        PS：关于IPC使用的 uORB的简单介绍（摘自官网）。

        The micro Object Request Broker (uORB) application is used toshare data structures between threads and applications,Communications between processes / applications (e.g. sendingsensor values from the sensors app to the attitude filter app) is a key part ofthe PX4 software architecture. Processes (often called nodes in this context) exchange messagesover named buses, called topics.In PX4, a topic contains only one message type, e.g. the vehicle\_attitude topic transports a message containing the attitude struct (roll, pitch and yaw estimates). Nodes can publish a message on a bus/topic (“send” data)or subscribe toa bus/topic (“receive” data). They are not aware of who they are communicatingwith. There can be multiple publishers and multiple subscribers to a topic.This design pattern prevents locking issues and is very common in robotics. Tomake this efficient, there is always only one message on the bus and no queue iskept.

        详细介绍：<https://pixhawk.org/dev/shared_object_communication>

4、如何获取精确时间（timestamp）

        在控制过程中多数环节都是使用经典的PID控制算法，为了获取较为实时的响应最重要的时间变量，这就涉及如何获取高精度的时间问题。

        Pixhawk主控使用ST公司的STM32F4系列处理器，其主频达168MHZ，内部有高精度RTC，以RTC为基准获取精确计时。根据分析源码发现，在modules/PX4Firmware/src/drivers中有一个头文件drv\_hrt.h（High-resolutiontimer with callouts and timekeeping），内部对其作出了一部分的介绍，微妙（us）级的精确计时，在中断上下文中调用，与其他函数并行执行不会被堵塞。摘抄几个典型函数如下。

1. /\*\* Get absolute time.\*/
2. \_\_EXPORT extern hrt\_abstime hrt\_absolute\_time(void);
3. /\*\* Compute the delta between a timestamp taken in the past and now.
4. \* This function is safe to use even if the timestamp is updated by an interrupt during execution. \*/
5. \_\_EXPORT extern hrt\_abstime hrt\_elapsed\_time(const volatile hrt\_abstime \*then);
6. /\*\* Store the absolute time in an interrupt-safe fashion.
7. \* This function ensures that the timestamp cannot be seen half-written by an interrupt handler.\*/
8. \_\_EXPORT extern hrt\_abstime hrt\_store\_absolute\_time(volatile hrt\_abstime \*now);
9. /\* initialise a hrt\_call structure \*/
10. \_\_EXPORT extern void    hrt\_call\_init(struct hrt\_call \*entry);
11. /\* Initialise the HRT. \*/
12. \_\_EXPORT extern void    hrt\_init(void);

        针对上述**hrt\_absolute\_time**(**void**)函数做阐述，其他的都类似，它的原型在modules/PX4Firmware/unittests中的hrt.cpp。

hrt\_abstime hrt\_absolute\_time()

1. {
2. struct timeval te;
3. gettimeofday(&te, NULL); // get current time
4. hrt\_abstime us = static\_cast<uint64\_t>(te.tv\_sec) \* 1e6 + te.tv\_usec; // caculate us
5. return us;
6. }

        然后捏：直接进入[**操作系统**](http://lib.csdn.net/base/operatingsystem)（NuttX），轮到操作系统上场了，modules/PX4NuttX/nuttx/schedclock\_gettimeofday.c，自己跟踪进去看吧，不在赘述。

## 第八章 PX4的sensors校准源码分析

#### PX4源码分析8\_PX4的sensor校准

#### sensor的校准过程分为两部分，首先需要先通过地面站进行校准设置，然后通过校准数据的更新来获取最新的校准数据。

### 1.校准数据的设置：

commander（地面站）经过计算之后通过param\_set()来设置校准数据，路径为src/modules/commander下的各个calibration文件，上电后rcS脚本执行sensors start之后会执行commder start来启动commander.cpp，commander.cpp进而调用各个calibration文件，进而进行校准设置。

### 2.校准数据的获取：

在sensors.cpp中的task\_main()循环中还有一个poll函数是parameter\_update\_poll()，这个poll函数获取了各个sensor的offset和scale来供各个sensor驱动中measure校准时使用，它使用param\_get()来获得。

### 二.陀螺仪的校准：

#### 1.do\_gyro\_calibration()：

commander.cpp中commander\_main()中执行do\_gyro\_calibration()。

#### 2.初始化：

gyro\_calibration.cpp中do\_gyro\_calibration()初始化gyro\_scale\_zero中各个offset赋值为0，各个scale赋值为1，并将gyro\_scale\_zero各个值通过param\_set()赋值给对应参数值。

#### 3.gyro\_calibration\_worker()：

gyro\_calibration.cpp中do\_gyro\_calibration()执行gyro\_calibration\_worker()。

#### 4.校准：

gyro\_calibration.cpp中gyro\_calibration\_worker()中：   
（1）循环执行5000次。   
（2）每次循环执行px4\_poll()进行延时处理。   
（3）每次循环均依次对各个gyro进行如下校准。   
（3）orb\_check()检查更新。   
（4）orb\_copy()接受数据赋值相应坐标轴offset，将校准变量gyro\_scale[s]加等gyro测量值gyro\_report。   
（5）相应坐标轴offset分别除以校准计数值5000。

#### 5.赋值：

gyro\_calibration.cpp中执行param\_set\_no\_notification()将gyro\_scale[s]相应坐标轴的offset赋值给对应参数值。

### 三.加速度计的校准：

#### 1.do\_accel\_calibration()：

commander.cpp中commander\_main()中执行do\_accel\_calibration()。

#### 2.初始化：

accelerometer\_calibration.cpp中do\_accel\_calibration()初始化accel\_scale中各个offset赋值为0，各个scale赋值为1，并将accel\_scale各个值通过param\_set()赋值给对应参数值。

#### 3.校准：

accelerometer\_calibration.cpp中do\_accel\_calibration()执行do\_accel\_calibration\_measurements。   
（1）执行calibrate\_from\_orientation()进行六个方向测量accel的offset值并赋值accel\_ref[max\_accel\_sens][detect\_orientation\_side\_count][3]中：   
 [1] 根据方向的转换，循环六次。   
 [2] 每次循环执行accel\_calibration\_worker()。   
 [3] accel\_calibration\_worker()执行read\_accelerometer\_avg()。   
 [4] read\_accelerometer\_avg函数中：   
   a：循环执行750次。   
   b：每次循环执行px4\_poll()进行延时处理。   
   c：并对各个accel进行如下校准。   
   d：orb\_check()检查更新。   
   e：orb\_copy()接受数据赋值相应坐标轴offset，将校准变量accel\_sum[s]加等accel测量值arp。   
   f：相应坐标轴offset分别除以校准计数值750赋值给accel\_ref。   
（2）执行calculate\_calibration\_values()，根据accel\_ref计算偏移offsets和转移矩阵transform matrix，分别赋值accel\_offs和accel\_T。

#### 4.计算offset：

根据accel\_offs和accel\_T值结合board\_rotation\_t计算accel\_scale[s]相应坐标轴的offset。

#### 5.赋值：

accelerometer\_calibration.cpp中执行param\_set\_no\_notification()将accel\_scale[s]相应坐标轴的offset赋值给对应参数值。

### 四.地磁的校准：

#### 1.do\_mag\_calibration()：

commander.cpp中commander\_main()中执行do\_mag\_calibration()。

#### 2.校准：

mag\_calibration.cpp的do\_mag\_calibration()中   
（1）初始化mscale\_null中各个offset赋值为0，各个scale赋值为1。   
（2）执行mag\_calibrate\_all(mavlink\_log\_pub)同时对所有mag校准。   
（3）所有测量点x，y，z轴数据初始化。   
（4）订阅mag消息。   
（5）限制点数据测量间隔时间。   
（6）进行mag点数据进行采集，执行calibrate\_from\_orientation，赋值worker\_data。   
 [1] 对六个面分别进行检测，检测过程如下：   
 [2] 执行calibration\_worker指针函数，即mag\_calibration\_worker函数：   
   a：旋转检测：任意方向旋转即可，仅需要大致相交的两个轴，通过对陀螺数据积分检测是否旋转。   
   b：对该面进行采集calibration\_points\_perside个mag数据存储至worker\_data.x，worker\_data.y，worker\_data.z中。   
（7）球体拟合最小二乘mag数据，调用sphere\_fit\_least\_squares()得到mag偏移sphere\_x，sphere\_y，sphere\_z，sphere\_radius。   
（8）sphere\_x，sphere\_y，sphere\_z分别赋值mscale.x\_offset，mscale.y\_offset，mscale.z\_offset。   
（9）通过mscale.x\_offset，mscale.y\_offset，mscale.z\_offset对偏移参数赋值。

## 第八章 PX4源码总结

## 第九章 安全决策机制

以任务为导向，任务决策主要决定多旋翼下一步“去哪儿”，进一步，需要规划路径，使得整个过程能满足诸如：飞向航路点并沿航线飞，以及飞向航路点并避障等要求。  
健康管理和失效保护。以安全为导向， 失效保护主要决定多旋翼下一步“去哪儿”。多旋翼飞行器在飞行前或飞行中，可能会发生通信故障、传感器失效和动力系统异常等，这些意外会直接导致控制任务无法完成。这一部分包括安全问题的介绍、机载设备的健康评估、机载设备的健康监测、失效后的保护建议。完成任务模式切换，同时考虑电池电量、GPS等传感器是否正常工作等信息。

## 第十章 PX4内部进程通信

PX4 平台的封装提供了跨平台的无锁发布 -订阅模式( [publish–subscribe pattern](http://en.wikipedia.org/wiki/publish%E2%80%93subscribe%20pattern)) 对象请求处理器。取决于平台/操作系统，完成这个的后端可能是[uORB](http://www.pixhawk.com/dev/shared_object_communication),或者Linux上的ROS / DDS ([data delivery service](http://en.wikipedia.org/wiki/data%20delivery%20service))。封装可扩展，也可作为消息系统提供支持，比如[ØMQ](http://en.wikipedia.org/wiki/%20%C3%98MQ) 。所有这些后端的共同特点是，它们都允许发送数据到一个像如“位置”这样的语义订阅主题（topic），并使多个接收器跨进程的收到这一数据。

### IPC性能

发布（Publication）到订阅（subscription）之间的延迟：

uORB: 23 us @ 168 MHz STM32F4

 ROS: TBD

ROS2 / DDS: 185 us @ 1.6 GHz IntelPentium 4 and 2 GB RAM running Windows XP ([details](http://mnb.ociweb.com/mnb/MiddlewareNewsBrief-201004.html))

ZeroMQ: 170 us @ 1.6 GHz IntelPentium 4 and 2 GB RAM running Windows XP ([details](http://mnb.ociweb.com/mnb/MiddlewareNewsBrief-201004.html))

### Uorb原理

PX4应用程序框架在所有平台上代码兼容，(比如 Pixhawk / NuttX，又比如， SnapDragon / Linux)，独立并将中间设备作为传输层。



### 节点句柄（Node Handle）

节点句柄是每一个连接到中间设备的发布器（publisher）或者订阅器（subscriber）的核心数据结构（一个节点可以同时是发布器和订阅器）。一个节点是一个逻辑单元，每一个进程都可以有多于一个的节点（虽然这并不是典型的形式）。

1. px4::NodeHandle n();

### 数据结构

对于所有相关的中间设备实现，PX4根据储存在“msg”中的定义文件自动生成据结构。

### 发布（publication）

通过使用节点句柄通告（advertise）订阅主题（ topic）来创建一个新的发布（publication）。注意这个模板，是订阅主题的数据结构。

1. px4::Publisher<px4::px4\_rc\_channels>\* rc\_channels\_pub = n.advertise<px4\_rc\_channels>()

从这个指向 rc\_channels\_pub 的句柄的指针，可以用于发布数据

1. px4\_rc\_channels rc\_channels\_msg;
2. rc\_channels\_msg.data().timestamp\_last\_valid= px4::get\_time\_micros();
3. \_rc\_channels\_pub->publish(rc\_channels\_msg);

### 订阅（Subscription）

通过订阅一个订阅主题创建一个新的发布。注意模板，是订阅主题的数据结构。使用订阅共有三个选择：

1.           使用订阅主题的句柄，进行手动复制操作。

2.           作为函数的回调（callback），在订阅主题更新时，函数被调用。

3.           作为类方法的回调（相当于函数，但是是C++类型的）

单纯订阅

除非手动调用复制方法，否则数据不会被复制。

1. unsigned min\_interval =500;
2. \_sub\_rc\_chan = \_n.subscribe<px4\_rc\_channels>(min\_interval);

函数回调

每次订阅主题更新，则'rc\_channels\_callback\_function'被调用。

1. unsigned min\_interval =500;
2. \_n.subscribe<px4\_rc\_channels>(rc\_channels\_callback\_function, min\_interval);

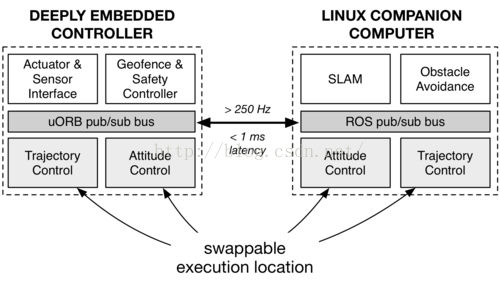
类方法回调

相当于前面函数的例子。

1. \_n.subscribe<px4\_rc\_channels>(&SubscriberExample::rc\_channels\_callback, **this**, min\_interval);

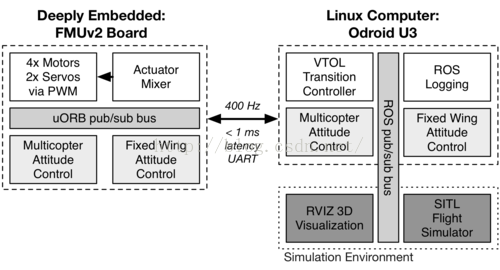
### 混合系统

了解决更高级别的挑战，比如基于视觉的避障或者复杂控制问题，一个运行嵌入式Linux的伴随计算机将非常有用。



### 集成ROS

PX4可以通过两种不同的API与ROS整合到一起：或者自然地将每一个应用作为ROS节点，或者通过[mavros](http://github.com/mavlink/mavros)在嵌入式自驾仪上唯一地运行



## 第十一章 进程间通讯开发方法

进程/应用之间的通信（例如将传感器应用的传感器数据发送到姿态滤波应用）是PX4软件[**架构**](http://lib.csdn.net/base/architecture)的关键部分。进程（通常又被叫做节点（node））通过命名为总线（buses）进行消息交换的过程被称作订阅主题（topics）。

在PX4中，一个订阅主题仅包含一个消息类型，比如vehicle\_attitude订阅主题传输一个包含姿态结构（滚转、俯仰和偏航估计）的消息。

节点可以在总线/订阅主题上发布（*publish）*一个消息（发送数据）或者订阅（*subscribe）*一个总线/订阅主题（接收数据） 。应用并不知道在与谁通信，1个订阅主题可以有多个发布器和订阅器。

这种设计模式阻止了锁定的问题（locking issues），在[**机器人**](http://lib.csdn.net/base/robot)领域非常常见。为了使这更为高效，通常在总线上只有一个消息，并且不会有队列。

这种发布/订阅机制由微对象请求处理器[microobject request broker (uORB)](http://www.pixhawk.com/firmware/apps/uorb)实现。

快速开始

这是一个简单但是完整的发布器（publisher） /订阅器（subscriber）组合，发布器发布一个叫做*random\_integer* 的订阅主题，并使用随机的整数更新订阅主题。订阅器检查并打印这些更新。

topic.h

1. /\* declare the topic \*/
2. ORB\_DECLARE(random\_integer);
3. /\* define the data structure that will be published where subscribers cansee it \*/
4. **struct** random\_integer\_data{
5. **int** r;
6. };

publisher.c

1. #include <topic.h>
2. /\* create topic metadata \*/
3. ORB\_DEFINE(random\_integer);
4. /\* file handle that will be used for publishing \*/
5. staticint topic\_handle;
6. **int** init()
7. {
8. /\* generate the initial data for first publication\*/
9. **struct** random\_integer\_data rd={ .r= random(),};
10. /\* advertise the topic and make the initialpublication \*/
11. topic\_handle = orb\_advertise(ORB\_ID(random\_integer),&rd);
12. }
13. **int** update\_topic()
14. {
15. /\* generate a new random number forpublication \*/
16. **struct** random\_integer\_data rd={ .r= random(),};
17. /\* publish the new data structure \*/
18. orb\_publish(ORB\_ID(random\_integer), topic\_handle,&rd);
19. }

subscriber.c

1. #include <topic.h>
2. /\* file handle that will be used forsubscribing \*/
3. staticint topic\_handle;
4. **int** init()
5. {
6. /\* subscribe to the topic \*/
7. topic\_handle = orb\_subscribe(ORB\_ID(random\_integer));
8. }
9. voidcheck\_topic()
10. {
11. **bool** updated;
12. **struct** random\_integer\_data rd;
13. /\* check to see whether the topic has updatedsince the last time we read it \*/
14. orb\_check(topic\_handle,&updated);
15. **if**(updated){
16. /\* make a local copy of the updated datastructure \*/
17. orb\_copy(ORB\_ID(random\_integer), topic\_handle,&rd);
18. printf("Random integer is now %d\n", rd.r);
19. }
20. }

发布过程

发布过程包含三个独立却相关联的动作：定义订阅主题，通告订阅主题，并且发布更新。

定义订阅主题

作为一种在元件之间提供通用接口的方式，系统定义了许多标准的订阅主题。如果一个发布器希望使用某一标准的订阅主题和其相关的[**数据结构**](http://lib.csdn.net/base/datastructure)，不需要再做额外的工作。

常规订阅主题

为了定义一个常规的订阅主题，发布器需要为订阅器建一个可用的头文件（上面的topic.h）。在这个头文件里必须包含：

* 将订阅主题的名字作为参数的一个**ORB\_DECLARE()** 宏实例。
* 一个用于描述将发布数据结构的结构定义。

订阅主题的名字应该是描述性的；PX4的命名规则是是使用下划线分开订阅主题的名字；组件的名字应该首先使用通用的词汇。

例如，原始传感器数据在**sensors\_raw**订阅主题中进行发布。

除了头文件，发布器必须包含一个**ORB\_DEFINE()**宏实例在源文件中，该文件可以在构建固件时被编译和链接（见publisher.c示例）。这个宏来构建由ORB使用的数据结构，从而唯一表示这个topic。

可选的订阅主题（optionaltopic）

如果一个由软件组件发布的订阅主题是可选的，并且可能不会出现在固件中，头文件可以使用 **ORB\_DECLARE\_OPTIONAL()**宏代替。以这种方式声明的订阅主题需要由发布器进行特别的处理，但是还有一些以下需要讨论的考虑，订阅器在处理可选订阅主题时必须要注意。

通告（advertise）订阅主题

在数据发布到一个订阅主题时，必须首先通告。发布器使用如下的API通告一个新的订阅主题：

1. /\*\*
2. \* Advertise as the publisher of atopic.
3. \*
4. \* This performs the initialadvertisement of a topic; it creates the topic
5. \* node in /obj if required andwrites the initial data.
6. \*
7. \* @param meta                 The uORB metadata (usually fromthe ORB\_ID() macro)
8. \*                             forthe topic. topic uORB元数据（通常来自ORB\_ID()宏）
9. \* @param data                 A pointer to the initial datato be published.指向将要publish初始数据的指针。
10. \*                             Fortopics published by interrupt handlers, the advertisement
11. \*                             mustbe performed from non-interrupt context.
12. \* @return           ERROR on error, otherwise returns a handle
13. \*                             thatcan be used to publish to the topic.
14. \*                             Ifthe topic in question is not known (due to an
15. \*                             ORB\_DEFINE\_OPTIONALwith no corresponding ORB\_DECLARE)
16. \*                             thisfunction will return -1 and set errno to ENOENT.
17. \*/
18. externint          orb\_advertise(conststruct orb\_metadata \*meta,constvoid\*data);

通告同时为订阅主题发布初始数据。

API的meta参数是由**ORB\_DEFINE()**宏产生的数据指针，通常使用**ORB\_ID()** 宏提供，该宏执行从订阅主题名到元数据结构名的转换。

需要注意的是，可以从中断句柄中发布实时更新，但是通告订阅主题必须在常规线程的情况下。

多个发布器

一个发布器一次只能通告一个订阅主题，然而发布器可能会关闭订阅主题句柄（这是个文件句柄，可以简单的传递到close（）函数），然后由另外一个发布器通告订阅主题。

发布更新

一个订阅主题被通告后，从通告那里返回的句柄可以使用如下的API，用于对订阅主题发布更新：

1. /\*\*
2. \* Publish new data to a topic.
3. \*
4. \* The data is atomically publishedto the topic and any waiting subscribers
5. \* will be notified.  Subscribers that are not waiting can checkthe topic
6. \* for updates using orb\_checkand/or orb\_stat.
7. \*
8. \* @handle           The handle returned from orb\_advertise.
9. \* @param meta                 The uORB metadata (usually fromthe ORB() macro)
10. \*                             forthe topic.
11. \* @param data                 A pointer to the data to bepublished.
12. \* @return           OK on success, ERROR otherwise with errno set accordingly.
13. \*/
14. externint          orb\_publish(conststruct orb\_metadata \*meta,**int** handle,constvoid\*data);

需要注意的是ORB并不缓存多个更新，当订阅器检查订阅主题时，只会看到最近的一个更新。

订阅

订阅一个订阅主题需要如下：

* 一个**ORB\_DEFINE()或者ORB\_DEFINE\_OPTIONAL()**宏（例如，在订阅器中包含的头文件）
* 在订阅主题中发布的数据结构定义。（通常来自相同的头文件）

假如这些条件都满足，订阅器使用下面的API来订阅一个订阅主题。

1. /\*\*
2. \* Subscribe to a topic.
3. \*
4. \* The returned value is a filedescriptor that can be passed to poll()
5. \* in order to wait for updates to atopic, as well as orb\_read,
6. \* orb\_check and orb\_stat.
7. \*
8. \* Subscription will succeed even ifthe topic has not been advertised;
9. \* in this case the topic will havea timestamp of zero, it will never
10. \* signal a poll() event, checkingwill always return false and it cannot
11. \* be copied. When the topic issubsequently advertised, poll, check,
12. \* stat and copy calls will react tothe initial publication that is
13. \* performed as part of theadvertisement.
14. \*
15. \* Subscription will fail if thetopic is not known to the system, i.e.
16. \* there is nothing in the systemthat has defined the topic and thus it
17. \* can never be published.
18. \*
19. \* @param meta                 The uORB metadata (usually fromthe ORB\_ID() macro)
20. \*                             forthe topic.
21. \* @return           ERROR on error, otherwise returns a handle
22. \*                             thatcan be used to read and check the topic for updates.
23. \*                             Ifthe topic in question is not known (due to an
24. \*                             ORB\_DEFINE\_OPTIONALwith no corresponding ORB\_DECLARE)
25. \*                             thisfunction will return -1 and set errno to ENOENT.
26. \*/
27. externint          orb\_subscribe(conststruct orb\_metadata \*meta);

订阅一个可选的订阅主题时，如果订阅主题不存在的话，将会失败。然而其他的所有订阅都会成功，并创建订阅主题，即使这并没有经发布器通告。这极大的降低了仔细安排系统启动顺序的需要。

对一项任务可以执行的订阅数目并没有具体的限制。

取消对一个订阅主题的订阅，使用如下的API：

1. /\*\*
2. \* Unsubscribe from a topic.
3. \*
4. \* @param handle     A handle returned from orb\_subscribe.
5. \* @return           OK on success, ERROR otherwise with errno set accordingly.
6. \*/
7. externint          orb\_unsubscribe(**int** handle);

从订阅主题中拷贝数据

订阅器并不是引用存储在ORB中的数据，或者与其他订阅器共享数据；而是在其请求下，将数据从ORB中拷贝出来存放在每一个订阅器对应的临时缓冲区中。这一拷贝就避免了锁定问题，并使得发布器和订阅器的API都变的简单。也允许订阅器及时根据自己的使用需求对数据进行修改。

当订阅器希望得到发布到订阅主题的最近数据拷贝时，使用如下的API：

1. /\*\*
2. \* Fetch data from a topic.
3. \*
4. \* @param meta                 The uORB metadata (usually fromthe ORB() macro)
5. \*                             forthe topic.
6. \* @param handle     A handle returned from orb\_subscribe.
7. \* @param buffer     Pointer to the buffer receiving the data.
8. \* @return           OK on success, ERROR otherwise with errno set accordingly.
9. \*/
10. externint          orb\_copy(conststruct orb\_metadata \*meta,**int** handle,**void**\*buffer);

复制可以保证数据是最新发布的。

更新检查

在订阅器调用**orb\_copy**函数后，可以使用如下的API检查从上一个时间开始，订阅主题是否接收到了新的发布。

1. /\*\*
2. \* Check whether a topic has beenpublished to since the last orb\_copy.
3. \*
4. \* This check can be used todetermine whether to copy from the topic when
5. \* not using poll(), or to avoid theoverhead of calling poll() when the
6. \* topic is likely to have updated.
7. \*
8. \* Updates are tracked on aper-handle basis; this call will continue to
9. \* return true until orb\_copy iscalled using the same handle. This interface
10. \* should be preferred over callingorb\_stat due to the race window between
11. \* stat and copy that can lead tomissed updates.
12. \*
13. \* @param handle     A handle returned from orb\_subscribe.
14. \* @param updated    Set to true if the topic has been publishedsince the
15. \*                             lasttime it was copied using this handle.
16. \* @return           OK if the check was successful, ERROR otherwise with
17. \*                             errnoset accordingly.
18. \*/
19. externint          orb\_check(**int** handle, **bool** \*updated);

当订阅主题在通告之前，已经被订阅，这个API将会返回没有更新，直到订阅主题被通告。

发布时间戳

订阅器可以使用如下的API检查订阅主题最近一次发布的时间：

1. /\*\*
2. \* Return the last time that thetopic was published.
3. \*
4. \* @param handle     A handle returned from orb\_subscribe.
5. \* @param time                 Returns the time that the topicwas published, or zero if it has
6. \*                             neverbeen published/advertised.
7. \* @return           OK on success, ERROR otherwise with errno set accordingly.
8. \*/
9. externint          orb\_stat(**int** handle,uint64\_t\*time);

当调用这个功能时，需要仔细检查，因为并不能保证在调用返回后订阅主题不会紧接着就发布。

等待更新

依赖于发布作为数据源的订阅器，其可以同时具有任意数量等待发布的订阅。这一功能通过使用 **poll()** 函数实现，与文件描述符等待数据具有相同的方式。这之所以可以工作，是因为订阅本身实际上也是文件描述符。

下面的例子展示的是，订阅器具有三个独立的订阅，等待到每一个的发布，当收到这些发布时进行回应。如果第二轮没有到任意订阅的更新，更新超时计数器，并发布给其他订阅器。

color\_counter.h

1. ORB\_DECLARE(color\_red);
2. ORB\_DECLARE(color\_green);
3. ORB\_DECLARE(color\_blue);
4. ORB\_DECLARE(color\_timeouts);
6. /\* structure published to color\_red, color\_green, color\_blue andcolor\_timeouts \*/
7. **struct** color\_update
8. {
9. **int** number;
10. };
11. color\_counter.c
12. #include <poll.h>
14. ORB\_DEFINE(color\_timeouts,**struct** color\_update);
16. **void**
17. subscriber(**void**)
18. {
19. **int**       sub\_red, sub\_green, sub\_blue;
20. **int**       pub\_timeouts;
21. **int**       timeouts=0;
22. **struct** color\_update cu;
24. /\* subscribe to color topics \*/
25. sub\_red = orb\_subscribe(ORB\_ID(color\_red));
26. sub\_green = orb\_subscribe(ORB\_ID(color\_green));
27. sub\_blue = orb\_subscribe(ORB\_ID(color\_blue));
29. /\* advertise the timeout topic \*/
30. cu.number=0;
31. pub\_timeouts = orb\_advertise(ORB\_ID(color\_timeouts),&cu);
33. /\* loop waiting for updates \*/
34. **for**(;;){
36. /\* wait for updates or a 1-second timeout \*/
37. **struct** pollfd fds[3]={
38. { .fd= sub\_red,   .events= POLLIN },
39. { .fd= sub\_green, .events= POLLIN },
40. { .fd= sub\_blue,  .events= POLLIN }
41. };
42. **int** ret= poll(fds,3,1000);
44. /\* check for a timeout \*/
45. **if**(ret==0){
46. puts("timeout");
47. cu.number=++timeouts;
48. orb\_publish(ORB\_ID(color\_timeouts), pub\_timeouts,&cu);
50. /\* check for color updates \*/
51. }**else**{
53. **if**(fds[0].revents& POLLIN){
54. orb\_copy(ORB\_ID(color\_red), sub\_red,&cu);
55. printf("red is now %d\n", cu.number);
56. }
57. **if**(fds[1].revents& POLLIN){
58. orb\_copy(ORB\_ID(color\_green), sub\_green,&cu);
59. printf("green is now %d\n", cu.number);
60. }
61. **if**(fds[2].revents& POLLIN){
62. orb\_copy(ORB\_ID(color\_blue), sub\_blue,&cu);
63. printf("blue is now %d\n", cu.number);
64. }
65. }
66. }
67. }

限制更新的速率

订阅器可能希望限制在订阅主题上接收更新的速率，这可以由下面的API完成。

1. /\*\*
2. \* Set the minimum interval betweenwhich updates are seen for a subscription.
3. \*
4. \* If this interval is set, thesubscriber will not see more than one update
5. \* within the period.
6. \*
7. \* Specifically, the first time anupdate is reported to the subscriber a timer
8. \* is started. The update willcontinue to be reported via poll and orb\_check, but
9. \* once fetched via orb\_copy anotherupdate will not be reported until the timer
10. \* expires.
11. \*
12. \* This feature can be used to pacea subscriber that is watching a topic that
13. \* would otherwise update tooquickly.
14. \*
15. \* @param handle     A handle returned from orb\_subscribe.
16. \* @param interval   An interval period in milliseconds.
17. \* @return           OK on success, ERROR otherwise with ERRNO set accordingly.
18. \*/
19. externint          orb\_set\_interval(**int** handle,unsigned interval);

速率限制是具体到某一个订阅的，一个订阅主题可能具有多于一个的订阅器，分别具有不同的速率限制。

## 第十二章 构建后台应用程序

在Unix和其他多任务计算机[**操作系统**](http://lib.csdn.net/base/operatingsystem)中，后台程序是指，作为后台进程运行的计算机，而不是由交互用户直接控制。

后台程序概念的主要好处是，后台程序可以直接启动，而不需要将其发送到精确的用户或者shell的后台（然而，这不适用于Nuttx），其状态可以在运行的时候，通过shell查询。也可以终止。

### 第一步: 创建一个小的标准应用

根据 [FirstOnboard Application Tutorial (Hello Sky)](http://www.pixhawk.com/dev/px4_simple_app)教程（见PX4源码开发人员文档（二）），这是一个基本程序（简化）：

1. ..
2. \_\_EXPORT **int** px4\_daemon\_app\_main(**int** argc, **char** \*argv[]);
3. ..
4. **int** px4\_daemon\_app\_main(**int** argc, **char** \*argv[])
5. {
6. **while** (**true**) {
7. warnx("Hello Daemon!\n");
8. sleep(1);
9. }
10. **return** 0;
11. }

这个应用的问题非常明显，如果不使用&启动，将会阻塞shell（Nuttx，并不如此，并且会出于small footprint和可靠性的原因，支持CTRL-Z / fg / bg）。为了回避这个问题，下面部分将应用转换为一个后台程序。

### 第二步: 创建后台进程管理函数

主函数由后台进程管理函数替代，旧的主函数的内容现在位于后台任务/进程中

1. #include <systemlib/systemlib.h>
3. ..
4. \_\_EXPORT **int** px4\_daemon\_app\_main(**int** argc, **char** \*argv[]);
5. ..
6. **int** mavlink\_thread\_main(**int** argc, **char** \*argv[]);
7. ..
8. **int** mavlink\_thread\_main(**int** argc, **char** \*argv[])
9. {
10. **while** (**true**) {
11. warnx("Hello Daemon!\n");
12. sleep(1);
13. **if** (thread\_should\_exit) **break**;
14. }
16. **return** 0;
17. }
18. ..
19. **int** px4\_daemon\_app\_main(**int** argc, **char** \*argv[])
20. {
21. **if** (argc < 1)
22. usage("missing command");
24. **if** (!strcmp(argv[1], "start")) {
26. **if** (thread\_running) {
27. warnx("daemon already running\n");
28. /\* this is not an error \*/
29. exit(0);
30. }
32. thread\_should\_exit = **false**;
33. daemon\_task = task\_spawn\_cmd("daemon",
34. SCHED\_RR,
35. SCHED\_PRIORITY\_DEFAULT,
36. 4096,
37. px4\_daemon\_thread\_main,
38. (argv) ? (**const** **char** \*\*)&argv[2] : (**const** **char** \*\*)NULL);
39. thread\_running = **true**;
40. exit(0);
41. }
43. usage("unrecognized command");
44. exit(1);
45. }

这将会启动一个新的任务，具有4096字节的堆栈，并传递非后台程序的具体指令行选项到后台主函数。典型的调用如下所示：

1. px4\_daemon\_app start

上面的代码没有报告状态，并且没有对多次调用后台进程进行保护。

第三步: 添加停止/状态指令以及安全保护

具有合适的启动/停止/状态建立和附加安全保护的完整px4\_daemon\_app代码如下：

1. /\*\*
2. \* @file px4\_daemon\_app.c
3. \* daemon application example for PX4 autopilot
4. \*
5. \* @author Example User <mail@example.com>
6. \*/
8. #include <stdio.h>
9. #include <stdlib.h>
10. #include <string.h>
11. #include <unistd.h>
13. #include <px4\_config.h>
14. #include <nuttx/sched.h>
16. #include <systemlib/systemlib.h>
17. #include <systemlib/err.h>
19. **static** **bool** thread\_should\_exit = **false**;     /\*\*< daemon exit flag \*/
20. **static** **bool** thread\_running = **false**;     /\*\*< daemon status flag \*/
21. **static** **int** daemon\_task;             /\*\*< Handle of daemon task / thread \*/
23. /\*\*
24. \* daemon management function.
25. \*/
26. \_\_EXPORT **int** px4\_daemon\_app\_main(**int** argc, **char** \*argv[]);
28. /\*\*
29. \* Mainloop of daemon.
30. \*/
31. **int** px4\_daemon\_thread\_main(**int** argc, **char** \*argv[]);
33. /\*\*
34. \* Print the correct usage.
35. \*/
36. **static** **void** usage(**const** **char** \*reason);
38. **static** **void**
39. usage(**const** **char** \*reason)
40. {
41. **if** (reason) {
42. warnx("%s\n", reason);
43. }
45. warnx("usage: daemon {start|stop|status} [-p <additional params>]\n\n");
46. }
48. /\*\*
49. \* The daemon app only briefly exists to start
50. \* the background job. The stack size assigned in the
51. \* Makefile does only apply to this management task.
52. \*
53. \* The actual stack size should be set in the call
54. \* to task\_create().
55. \*/
56. **int** px4\_daemon\_app\_main(**int** argc, **char** \*argv[])
57. {
58. **if** (argc < 2) {
59. usage("missing command");
60. **return** 1;
61. }
63. **if** (!strcmp(argv[1], "start")) {
65. **if** (thread\_running) {
66. warnx("daemon already running\n");
67. /\* this is not an error \*/
68. **return** 0;
69. }
71. thread\_should\_exit = **false**;
72. daemon\_task = px4\_task\_spawn\_cmd("daemon",
73. SCHED\_DEFAULT,
74. SCHED\_PRIORITY\_DEFAULT,
75. 2000,
76. px4\_daemon\_thread\_main,
77. (argv) ? (**char** \***const** \*)&argv[2] : (**char** \***const** \*)NULL);
78. **return** 0;
79. }
81. **if** (!strcmp(argv[1], "stop")) {
82. thread\_should\_exit = **true**;
83. **return** 0;
84. }
86. **if** (!strcmp(argv[1], "status")) {
87. **if** (thread\_running) {
88. warnx("\trunning\n");
90. } **else** {
91. warnx("\tnot started\n");
92. }
94. **return** 0;
95. }
97. usage("unrecognized command");
98. **return** 1;
99. }
101. **int** px4\_daemon\_thread\_main(**int** argc, **char** \*argv[])
102. {
104. warnx("[daemon] starting\n");
106. thread\_running = **true**;
108. **while** (!thread\_should\_exit) {
109. warnx("Hello daemon!\n");
110. sleep(10);
111. }
113. warnx("[daemon] exiting.\n");
115. thread\_running = **false**;
117. **return** 0;
118. }

代码[**测试**](http://lib.csdn.net/base/softwaretest)将会产生如下的输出：

1. nsh> px4\_daemon\_app start
2. [daemon] starting
3. Hello Daemon!

为了使用这一APP，只需在Firmware/makefiles/config\_px4fmu\_default.mk中，取消对这一示例部分的注释。

## 第十三章 构建最小应用程序

### 第一步: 准备源码文件

* 用UART1连接PX4FMU和计算机
* 安装PX4Toolchain
* 注册Github账户

为了方便管理代码，可以使用GIT 版本控制系统，在 GitHub上 fork和更新源码。

不注册GitHub的话，可以在PX4 console中输入下面的命令：

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

更新the git submodules，在 PX4 console上输入：

1. cd Firmware
2. git submodule init
3. git submodule update

文件准本完成，编译操作系统，输入：

1. make archives

当submodules或者NuttX 配置改变，重新编译。

在Firmware/src/examples/ 新建目录px4\_simple\_app，在目录中新建文件module.mk并添加如下内容:

MODULE\_COMMAND      = px4\_simple\_app

1. SRCS            = px4\_simple\_app.c

### 第二步: 编写最小程序

在px4\_simple\_app目录下创建px4\_simple\_app.c文件。

遵循[PX4 CodingStyle](http://www.pixhawk.com/dev/code_style) 编辑如下代码：

1. /\*\*
2. \* @file px4\_simple\_app.c
3. \* Minimal application example for PX4 autopilot.
4. \*/
6. #include <nuttx/config.h>
7. #include <stdio.h>
8. #include <errno.h>
10. \_\_EXPORT **int** px4\_simple\_app\_main(**int** argc, **char** \*argv[]);
12. **int** px4\_simple\_app\_main(**int** argc, **char** \*argv[])
13. {
14. printf("Hello Sky!\n");
15. **return** OK;
16. }

### 第三步: 在NuttShell中注册应用并build

完成了应用并可以运行，但是没有在NuttShell 命令中注册。为将应用编译到固件当中，将其填加到需要build的模块中。该配置位于：

[Firmware/makefiles/nuttx/config\_px4fmu-v2\_default.mk](https://github.com/PX4/Firmware/blob/master/makefiles/nuttx/config_px4fmu-v2_default.mk)

在上面的文件中的任意位置，为应用创建一行：

MODULES     += examples/px4\_simple\_app

然后编译，并清除已经build的应用：

1. make clean
2. make px4fmu-v2\_default -j4

如果没有注册新的app，只需要对新增的builds执行下述命令：

make px4fmu-v2\_default -j4

### 第四步:加载应用并测试

使能uploader 然后重置电路板:

1. make upload px4fmu-v2\_default -j4

在重置电路板之前显示如下一系列信息：

1. Generating /Users/user/src/Firmware/Images/px4fmu.px4
2. Loaded firmware for 5,0, waiting for the bootloader...

电路板充值并上载后，打印：

1. Found board 5,0 on /dev/tty.usbmodem1
2. erase...
3. program...
4. verify...
5. done, rebooting.

运行px4 Toolchain 下的TeraTerm，运行File/New connection，选择飞控所在的串口，点击OK。按回车，出现nsh>.

在NSH连接的情况下，切换到shell.如果没有看到输出，敲击Enter键，会得到NSH提示；

或者也可以，使用QGroundControl进行NuttShell (NSH)，使用Mini-USB连接Pixhawk，按照下图进行。点击QGroundControl上的终端输出，并敲击Enter键。

1. nsh>

输入 help，并敲击Enter键:

1. nsh> help
2. help usage:  help [-v] [<cmd>]
4. [           df          kill        mkfifo      ps          sleep
5. ?           echo        losetup     mkrd        pwd         test
6. cat         exec        ls          mh          rm          umount
7. cd          exit        mb          mount       rmdir       unset
8. cp          free        mkdir       mv          set         usleep
9. dd          help        mkfatfs     mw          sh          xd
11. Builtin Apps:
12. reboot
13. perf
14. top
15. ..
16. px4\_simple\_app
17. ..
18. sercon
19. serdis

px4\_simple\_app 现在成为可用的指令。输入px4\_simple\_app并敲击Enter键:

1. nsh> px4\_simple\_app
2. Hello Sky!

## 第十五章 添加自己的传感器

要想自己添加一个传感器的话，最好先搞明白已有的传感器的工作过程。

这里记录一下PX4中MPU6000加速度计陀螺仪的解读过程，从[mpu6000.cpp](https://github.com/PX4/Firmware/blob/v1.5.2/src/drivers/mpu6000/mpu6000.cpp#L35)出发，介绍从驱动注册到原始数据读取的过程。涉及到一些关于Linux设备驱动开发的知识。

在继续往下读之前有必要先感受一下PX4中[驱动的注册过程](http://blog.csdn.net/oqqenvy12/article/details/69676785)，以及关键的[设备驱动ID分配](http://blog.csdn.net/oqqenvy12/article/details/69568468)。

### 字符型设备

在NuttX操作系统中，MPU6000是以字符设备的形式存在的，这一点从MPU6000这个类的定义中可用看出来

class MPU6000 : public device::CDev{ }

* 1

MPU6000类以公有形式继承自CDev(character device)字符型设备，表明MPU6000可以看做成是字符型设备，可以进行如下的设备操作

const struct file\_operations CDev::fops = {

open : cdev\_open,

close : cdev\_close,

read : cdev\_read,

write : cdev\_write,

seek : cdev\_seek,

ioctl : cdev\_ioctl,

poll : cdev\_poll,

};

这里最值得关注的是file\_operations这个结构体，其定义位于fs.h，该文件中包含所有字符型设备的结构体和API。在Linux系统中，万物皆文件，所有的设备都被当做文件进行操作open、read、close等。

struct file\_operations

{

int (\*open)(FAR struct file \*filp);

int (\*close)(FAR struct file \*filp);

ssize\_t (\*read)(FAR struct file \*filp, FAR char \*buffer, size\_t buflen);

ssize\_t (\*write)(FAR struct file \*filp, FAR const char \*buffer, size\_t buflen);

off\_t (\*seek)(FAR struct file \*filp, off\_t offset, int whence);

int (\*ioctl)(FAR struct file \*filp, int cmd, unsigned long arg);

#ifndef CONFIG\_DISABLE\_POLL

int (\*poll)(FAR struct file \*filp, struct pollfd \*fds, bool setup);

#endif

};

按照这样的思路，大可以想象直接将传感器作为一个文件open然后read即可，思路是正确的，但是需要一些前提条件的：

1. 驱动注册。只有将设备注册到系统中才能进行文件操作的，而且怎么保证你打开的设备是你想要打开的？MPU6000的加速度计、陀螺仪算是两个设备了
2. 端口配置。MPU6000通过SPI总线连接，中断读取，要配置的东西还是有一些的。

### 传感器的硬件连接

board.h中介绍了Pixhawk飞控板的资源分配情况，包括STM32的时钟配置，各串口的引脚对应情况，I2C、CAN的连接以及本文着重要关注的SPI总线连接情况：

/\*

\* SPI

\*

\* There are sensors on SPI1, and SPI2 is connected to the FRAM.

\*/

#define GPIO\_SPI1\_MISO (GPIO\_SPI1\_MISO\_1|GPIO\_SPEED\_50MHz) // SPI1

#define GPIO\_SPI1\_MOSI (GPIO\_SPI1\_MOSI\_1|GPIO\_SPEED\_50MHz)

#define GPIO\_SPI1\_SCK (GPIO\_SPI1\_SCK\_1|GPIO\_SPEED\_50MHz)

#define GPIO\_SPI2\_MISO (GPIO\_SPI2\_MISO\_1|GPIO\_SPEED\_50MHz) // SPI2

#define GPIO\_SPI2\_MOSI (GPIO\_SPI2\_MOSI\_1|GPIO\_SPEED\_50MHz)

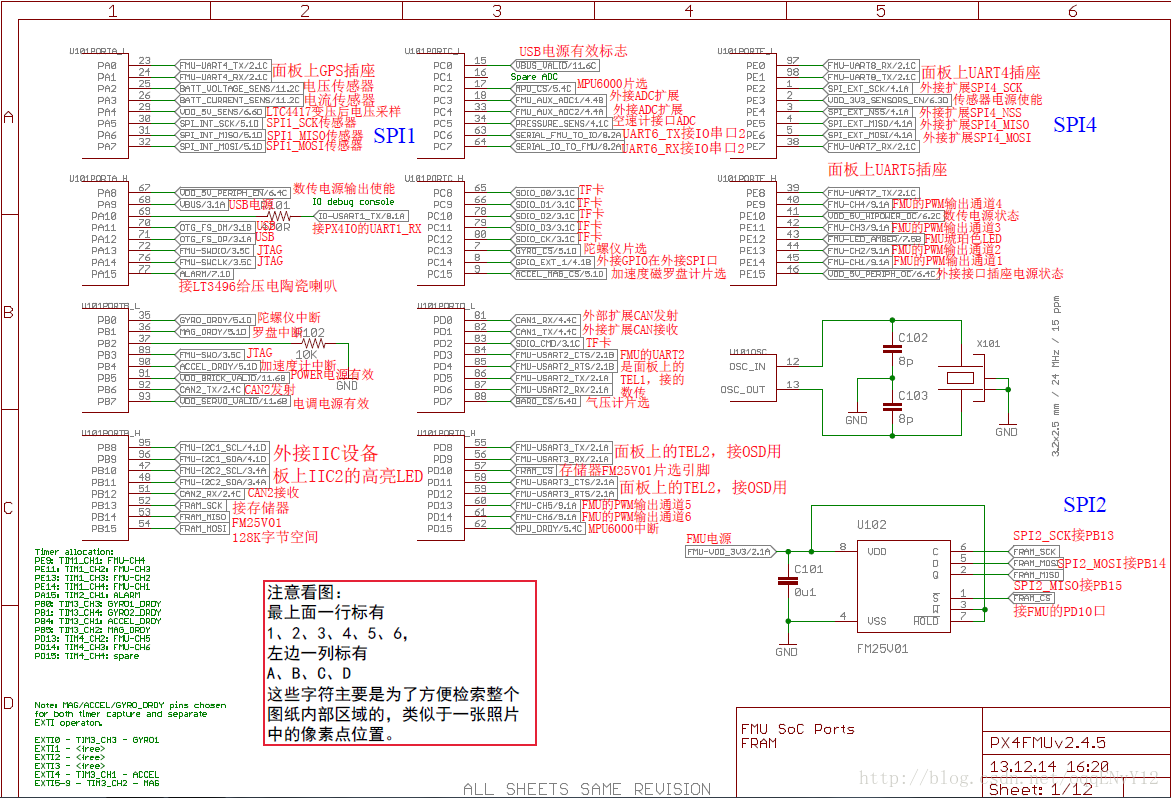
#define GPIO\_SPI2\_SCK (GPIO\_SPI2\_SCK\_2|GPIO\_SPEED\_50MHz)

#define GPIO\_SPI4\_MISO (GPIO\_SPI4\_MISO\_1|GPIO\_SPEED\_50MHz) // SPI4

#define GPIO\_SPI4\_MOSI (GPIO\_SPI4\_MOSI\_1|GPIO\_SPEED\_50MHz)

#define GPIO\_SPI4\_SCK (GPIO\_SPI4\_SCK\_1|GPIO\_SPEED\_50MHz)

Pixhawk飞控板引出了3个SPI总线接口



在文件board\_config.h中则是对相关引脚的功能配置，例如给PWM舵机输出引脚上拉、定时器配置、ADC定义以及关键的SPI总线设置。主要包括：

* 片选引脚配置

/\* SPI chip selects \*/

// SPI芯片片选引脚配置

#define GPIO\_SPI\_CS\_GYRO (GPIO\_OUTPUT|GPIO\_PUSHPULL|GPIO\_SPEED\_2MHz|GPIO\_OUTPUT\_SET|GPIO\_PORTC|GPIO\_PIN13)

#define GPIO\_SPI\_CS\_ACCEL\_MAG (GPIO\_OUTPUT|GPIO\_PUSHPULL|GPIO\_SPEED\_2MHz|GPIO\_OUTPUT\_SET|GPIO\_PORTC|GPIO\_PIN15)

#define GPIO\_SPI\_CS\_BARO (GPIO\_OUTPUT|GPIO\_PUSHPULL|GPIO\_SPEED\_2MHz|GPIO\_OUTPUT\_SET|GPIO\_PORTD|GPIO\_PIN7)

#define GPIO\_SPI\_CS\_FRAM (GPIO\_OUTPUT|GPIO\_PUSHPULL|GPIO\_SPEED\_2MHz|GPIO\_OUTPUT\_SET|GPIO\_PORTD|GPIO\_PIN10)

#define GPIO\_SPI\_CS\_HMC (GPIO\_OUTPUT|GPIO\_PUSHPULL|GPIO\_SPEED\_2MHz|GPIO\_OUTPUT\_SET|GPIO\_PORTC|GPIO\_PIN1)

#define GPIO\_SPI\_CS\_MPU (GPIO\_OUTPUT|GPIO\_PUSHPULL|GPIO\_SPEED\_2MHz|GPIO\_OUTPUT\_SET|GPIO\_PORTC|GPIO\_PIN2)

/////// 外部扩展SPI4的片选引脚

#define GPIO\_SPI\_CS\_EXT0 (GPIO\_OUTPUT|GPIO\_PUSHPULL|GPIO\_SPEED\_50MHz|GPIO\_OUTPUT\_SET|GPIO\_PORTE|GPIO\_PIN4)

#define GPIO\_SPI\_CS\_EXT1 (GPIO\_OUTPUT|GPIO\_PUSHPULL|GPIO\_SPEED\_50MHz|GPIO\_OUTPUT\_SET|GPIO\_PORTC|GPIO\_PIN14)

#define GPIO\_SPI\_CS\_EXT2 (GPIO\_OUTPUT|GPIO\_PUSHPULL|GPIO\_SPEED\_50MHz|GPIO\_OUTPUT\_SET|GPIO\_PORTC|GPIO\_PIN15)

#define GPIO\_SPI\_CS\_EXT3 (GPIO\_OUTPUT|GPIO\_PUSHPULL|GPIO\_SPEED\_50MHz|GPIO\_OUTPUT\_SET|GPIO\_PORTC|GPIO\_PIN13)

#define GPIO\_SPI\_CS\_LIS (GPIO\_OUTPUT|GPIO\_PUSHPULL|GPIO\_SPEED\_50MHz|GPIO\_OUTPUT\_SET|GPIO\_PORTE|GPIO\_PIN4)

* 以及SPI总线的宏定义

#define PX4\_SPI\_BUS\_SENSORS 1

#define PX4\_SPI\_BUS\_RAMTRON 2

#define PX4\_SPI\_BUS\_EXT 4

#define PX4\_SPI\_BUS\_BARO PX4\_SPI\_BUS\_SENSORS

一共三各SPI接口1、2、4，其中传感器连到SPI1上，铁电随机存储器FM25V01连到SPI2上，还有外部SPI4。

\*\*注意：\*\*FMUv3也就是常说的Pixhawk2.1的Cube中有两套IMU，用的就是SPI4，并且外接的两套IMU与Pixhawk上原有的两套IMU是相同的，Pixhawk2上多出来一套MPU9250九轴IMU不知道用上没有。

* 以及SPI1总线上的设备枚举

#define PX4\_SPIDEV\_GYRO 1

#define PX4\_SPIDEV\_ACCEL\_MAG 2

#define PX4\_SPIDEV\_BARO 3

#define PX4\_SPIDEV\_MPU 4

#define PX4\_SPIDEV\_HMC 5

#define PX4\_SPIDEV\_LIS 7

#define PX4\_SPIDEV\_BMI 8

### 与SPI操作相关的函数

#### spi.h

如同fs.h中包含了所有字符型设备的结构体和API，spi.h中是所有SPI设备驱动和API的定义。

需要注意spi\_ops\_s这个关键的指向函数的结构体，SPI协议的相关操作都可以从这里找到：select(片选)、setmode(时钟极性、相位)、setbit(8/16位)等等。

struct spi\_ops\_s

{

#ifndef CONFIG\_SPI\_OWNBUS

int (\*lock)(FAR struct spi\_dev\_s \*dev, bool lock);

#endif

void (\*select)(FAR struct spi\_dev\_s \*dev, enum spi\_dev\_e devid,

bool selected);

uint32\_t (\*setfrequency)(FAR struct spi\_dev\_s \*dev, uint32\_t frequency);

void (\*setmode)(FAR struct spi\_dev\_s \*dev, enum spi\_mode\_e mode);

void (\*setbits)(FAR struct spi\_dev\_s \*dev, int nbits);

uint8\_t (\*status)(FAR struct spi\_dev\_s \*dev, enum spi\_dev\_e devid);

#ifdef CONFIG\_SPI\_CMDDATA

int (\*cmddata)(FAR struct spi\_dev\_s \*dev, enum spi\_dev\_e devid, bool cmd);

#endif

uint16\_t (\*send)(FAR struct spi\_dev\_s \*dev, uint16\_t wd);

#ifdef CONFIG\_SPI\_EXCHANGE

void (\*exchange)(FAR struct spi\_dev\_s \*dev, FAR const void \*txbuffer,

FAR void \*rxbuffer, size\_t nwords);

#else

void (\*sndblock)(FAR struct spi\_dev\_s \*dev, FAR const void \*buffer,

size\_t nwords);

void (\*recvblock)(FAR struct spi\_dev\_s \*dev, FAR void \*buffer,

size\_t nwords);

#endif

int (\*registercallback)(FAR struct spi\_dev\_s \*dev, spi\_mediachange\_t callback,

void \*arg);

};

#### stm32\_spi.c

文件stm32\_spi.c中是spi协议的函数实现。

首先以SPI1为例，g\_sp1iops是一个spi\_ops\_s结构体，可以类似的理解为SPI这个类的实例，包含了所有的成员.select、setmode等，并且对应的完成了功能函数的实现，如spi\_setfrequency、spi\_setmode等。stm32\_spi1select的实现在后面的文件中会介绍。

#ifdef CONFIG\_STM32\_SPI1

static const struct spi\_ops\_s g\_sp1iops =

{

#ifndef CONFIG\_SPI\_OWNBUS

.lock = spi\_lock,

#endif

.select = stm32\_spi1select,

.setfrequency = spi\_setfrequency,

.setmode = spi\_setmode,

.setbits = spi\_setbits,

.status = stm32\_spi1status,

#ifdef CONFIG\_SPI\_CMDDATA

.cmddata = stm32\_spi1cmddata,

#endif

.send = spi\_send,

#ifdef CONFIG\_SPI\_EXCHANGE

.exchange = spi\_exchange,

#else

.sndblock = spi\_sndblock,

.recvblock = spi\_recvblock,

#endif

.registercallback = 0,

};

static struct stm32\_spidev\_s g\_spi1dev =

{

.spidev = { &g\_sp1iops },

.spibase = STM32\_SPI1\_BASE,

.spiclock = STM32\_PCLK2\_FREQUENCY,

#ifdef CONFIG\_STM32\_SPI\_INTERRUPTS

.spiirq = STM32\_IRQ\_SPI1,

#endif

#ifdef CONFIG\_STM32\_SPI\_DMA

.rxch = DMACHAN\_SPI1\_RX,

.txch = DMACHAN\_SPI1\_TX,

#endif

};

#endif

由结构体g\_spi1dev的第一个成员.spidev = { &g\_sp1iops }可以看出**结构体**g\_sp1iops是**属于**g\_spi1dev这个结构体的，因此一个spi设备可以由stm32\_spidev\_s这个结构体的实例表示。

本文件中有几个关键的函数需要注意：

* spi\_portinitialize

/\* 将所选的SPI端口初始化为其默认状态 \*/

static void spi\_portinitialize(FAR struct stm32\_spidev\_s \*priv)

{

/\* Configure CR1. Default configuration:

\* Mode 0: CPHA=0 and CPOL=0

\* Master: MSTR=1

\* 8-bit: DFF=0

\* MSB tranmitted first: LSBFIRST=0

\* Replace NSS with SSI & SSI=1: SSI=1 SSM=1 (prevents MODF error)

\* Two lines full duplex: BIDIMODE=0 BIDIOIE=(Don't care) and RXONLY=0

\*/

...

}

* up\_spiinitialize

/\* 初始化spi端口 \*/

FAR struct spi\_dev\_s \*up\_spiinitialize(int port)

{

// 以SPI1为例

#ifdef CONFIG\_STM32\_SPI1

if (port == 1) // 对应硬件配置中的宏定义

{

/\* Select SPI1 \*/

priv = &g\_spi1dev;

/\* Only configure if the port is not already configured \*/

if ((spi\_getreg(priv, STM32\_SPI\_CR1\_OFFSET) & SPI\_CR1\_SPE) == 0)

{

/\* Configure SPI1 pins: SCK, MISO, and MOSI \*/

// 配置SPI1的SCK、MISO、MOSI引脚

stm32\_configgpio(GPIO\_SPI1\_SCK);

stm32\_configgpio(GPIO\_SPI1\_MISO);

stm32\_configgpio(GPIO\_SPI1\_MOSI);

/\* Set up default configuration: Master, 8-bit, etc. \*/

// 设置SPI默认配置

spi\_portinitialize(priv); // 上面的函数

}

}

else

#endif

#ifdef CONFIG\_STM32\_SPI2

....

}

#### px4fmu\_spi.c

文件px4fmu\_spi.c中是一些Pixhawk飞控板特定的SPI函数

* stm32\_spiinitialize

/\* 为PX4FMU板配置SPI片选GPIO引脚 \*/

\_\_EXPORT void stm32\_spiinitialize(void)

{

#ifdef CONFIG\_STM32\_SPI1

px4\_arch\_configgpio(GPIO\_SPI\_CS\_GYRO); // PC13 L3GD20陀螺仪片选

px4\_arch\_configgpio(GPIO\_SPI\_CS\_ACCEL\_MAG); // PC15 LSM303D加速度计/磁力计片选

px4\_arch\_configgpio(GPIO\_SPI\_CS\_BARO); // PD7 MS5611气压计片选

px4\_arch\_configgpio(GPIO\_SPI\_CS\_HMC); // PC1 HMC5883磁力计片选 Pixhawk上木有HMC啊

px4\_arch\_configgpio(GPIO\_SPI\_CS\_MPU); // PC2 MPU6000 加速度计/陀螺仪片选

/\* De-activate all peripherals,

\* required for some peripheral

\* state machines

\*/

px4\_arch\_gpiowrite(GPIO\_SPI\_CS\_GYRO, 1);

px4\_arch\_gpiowrite(GPIO\_SPI\_CS\_ACCEL\_MAG, 1);

px4\_arch\_gpiowrite(GPIO\_SPI\_CS\_BARO, 1);

px4\_arch\_gpiowrite(GPIO\_SPI\_CS\_HMC, 1);

px4\_arch\_gpiowrite(GPIO\_SPI\_CS\_MPU, 1);

px4\_arch\_configgpio(GPIO\_EXTI\_GYRO\_DRDY);

px4\_arch\_configgpio(GPIO\_EXTI\_MAG\_DRDY);

px4\_arch\_configgpio(GPIO\_EXTI\_ACCEL\_DRDY);

px4\_arch\_configgpio(GPIO\_EXTI\_MPU\_DRDY);

#endif

#ifdef CONFIG\_STM32\_SPI2

...

}

* stm32\_spi1select

这个函数应该熟悉，是文件stm32\_spi.c中SPI1的结构体g\_sp1iops片选成员函数的实现。其作用是根据设备ID(devid)选中一个具体的SPI设备

\_\_EXPORT void stm32\_spi1select(FAR struct spi\_dev\_s \*dev, enum spi\_dev\_e devid, bool selected)

{

/\* SPI select is active low, so write !selected to select the device \*/

// SPI片选低电平有效。所以写!select就是选中了芯片

switch (devid) {

case PX4\_SPIDEV\_GYRO:

/\* Making sure the other peripherals are not selected \*/

px4\_arch\_gpiowrite(GPIO\_SPI\_CS\_GYRO, !selected);

px4\_arch\_gpiowrite(GPIO\_SPI\_CS\_ACCEL\_MAG, 1);

px4\_arch\_gpiowrite(GPIO\_SPI\_CS\_BARO, 1);

px4\_arch\_gpiowrite(GPIO\_SPI\_CS\_HMC, 1);

px4\_arch\_gpiowrite(GPIO\_SPI\_CS\_MPU, 1);

break;

case PX4\_SPIDEV\_ACCEL\_MAG:

/\* Making sure the other peripherals are not selected \*/

px4\_arch\_gpiowrite(GPIO\_SPI\_CS\_GYRO, 1);

px4\_arch\_gpiowrite(GPIO\_SPI\_CS\_ACCEL\_MAG, !selected);

px4\_arch\_gpiowrite(GPIO\_SPI\_CS\_BARO, 1);

px4\_arch\_gpiowrite(GPIO\_SPI\_CS\_HMC, 1);

px4\_arch\_gpiowrite(GPIO\_SPI\_CS\_MPU, 1);

break;

case PX4\_SPIDEV\_BARO:

...

}

#### px4fmu2\_init.c

文件px4fmu2\_init.c作用于系统配置和映射所有内存之后，在初始化任何设备之前。执行NSH的架构特定初始化。主要是SPI总线各设备的选择。

\_\_EXPORT int nsh\_archinitialize(void)

{

...

/\* Configure SPI-based devices \*/

// 配置基于SPI的设备

spi1 = px4\_spibus\_initialize(1);

if (!spi1) {

message("[boot] FAILED to initialize SPI port 1\n");

up\_ledon(LED\_AMBER);

return -ENODEV;

}

/\* Default SPI1 to 1MHz and de-assert the known chip selects. \*/

// 默认SPI1频率为1MHz，并取消断言已知芯片选择

SPI\_SETFREQUENCY(spi1, 10000000);

SPI\_SETBITS(spi1, 8);

SPI\_SETMODE(spi1, SPIDEV\_MODE3);

SPI\_SELECT(spi1, PX4\_SPIDEV\_GYRO, false);

SPI\_SELECT(spi1, PX4\_SPIDEV\_ACCEL\_MAG, false);

SPI\_SELECT(spi1, PX4\_SPIDEV\_BARO, false);

SPI\_SELECT(spi1, PX4\_SPIDEV\_MPU, false);

up\_udelay(20);

/\* Get the SPI port for the FRAM \*/

spi2 = px4\_spibus\_initialize(2);

...

}

它的调用常见的是

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Name: nsh\_archinitialize

\*

\* Description:

\* Perform architecture specific initialization for NSH.

\*

\* CONFIG\_NSH\_ARCHINIT=y :

\* Called from the NSH library

\*

\* CONFIG\_BOARD\_INITIALIZE=y, CONFIG\_NSH\_LIBRARY=y, &&

\* CONFIG\_NSH\_ARCHINIT=n :

\* Called from board\_initialize().

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifdef CONFIG\_NSH\_LIBRARY

int nsh\_archinitialize(void);

#endif

### MPU6000驱动分析

进入mpu6000.cpp文件。

这个文件主要分为3个部分：MPU6000类的实现(实例化、类成员函数定义)、MPU6000\_gyro类的实现(实例化、类成员函数定义)以及一些与shell命令相关的函数定义。

### MPU6000类和MPU6000\_gyro类

从两个类的定义可以看到，这两个类互为友元类，可以互相访问对方的私有成员函数。

class MPU6000 : public device::CDev

{

public:

MPU6000(device::Device \*interface, const char \*path\_accel, const char \*path\_gyro, enum Rotation rotation,

int device\_type);

...

protected:

...

friend class MPU6000\_gyro;

...

private:

...

MPU6000\_gyro \*\_gyro;

...

}

/\*\*

\* Helper class implementing the gyro driver node.

\*/

class MPU6000\_gyro : public device::CDev

{

public:

MPU6000\_gyro(MPU6000 \*parent, const char \*path);

...

protected:

friend class MPU6000;

...

private:

MPU6000 \*\_parent;

...

}

从代码中可以看出，该传感器的功能实现部分主要是在MPU6000类中实现的，包括传感器连接检测(MPU6000::probe)、设备初始化/**加速度计驱动注册**(MPU6000::init)、**加速度计的I/O通道管理**(MPU6000::ioctl)、**陀螺仪的I/O通道管理**(MPU6000::gyro\_ioctl)传感器自检(MPU6000::self\_test)、采样频率设置(MPU6000::\_set\_sample\_rate)、数据读取(MPU6000::measure)以及与数据预处理相关的一些操作(数据检验、低通滤波)等等。

而在MPU6000\_gyro这个类中，其实只做了一件事情，那就是完成**陀螺仪的驱动注册**(MPU6000\_gyro::init)。虽说MPU6000\_gyro类的成员函数中也包含了陀螺仪数据读取(MPU6000\_gyro::read)、陀螺仪的I/O通道管理(MPU6000\_gyro::ioctl)，但是其最终实现都是调用的MPU6000类成员函数

// MPU6000\_gyro实例

MPU6000\_gyro::MPU6000\_gyro(MPU6000 \*parent, const char \*path) :

CDev("MPU6000\_gyro", path), // 陀螺仪设备端口

\_parent(parent), // parent就是一个类MPU6000的对象

...

}

ssize\_t

MPU6000\_gyro::read(struct file \*filp, char \*buffer, size\_t buflen)

{

return \_parent->gyro\_read(filp, buffer, buflen);// 调用MPU6000::gyro\_read

}

int

MPU6000\_gyro::ioctl(struct file \*filp, int cmd, unsigned long arg)

{

switch (cmd) {

case DEVIOCGDEVICEID: // 获取设备ID

return (int)CDev::ioctl(filp, cmd, arg);

break;

default:

return \_parent->gyro\_ioctl(filp, cmd, arg);// 调用MPU6000::gyro\_ioctl

}

}

**总结**：主要的功能函数实现还是看MPU6000的类成员函数。

MPU6000加速度计陀螺仪传感器中的加速度计、陀螺仪端口不同。

因为读陀螺仪的数据和其他的数据不是一个端口，所以新建了MPU6000\_gyro这个Helper类。MPU6000类内完成加速度计的驱动注册，MPU6000\_gyro类内完成陀螺仪的驱动注册。分别注册到fs文件系统后，才能进行file\_operation相关的指令:open、read 、write。

### 驱动注册过程

以陀螺仪为例介绍一下PX4中如何将一个设备注册到NuttX的文件系统中。

int

MPU6000\_gyro::init()

{

int ret;

// do base class init

ret = CDev::init(); // 注册到fs中

/\* if probe/setup failed, bail now \*/

if (ret != OK) {

DEVICE\_DEBUG("gyro init failed");

return ret;

}

\_gyro\_class\_instance = register\_class\_devname(GYRO\_BASE\_DEVICE\_PATH); //注册节点

return ret;

}

关于这里注册的设备的具体信息，后面会讲到，现在可以简单理解成MPU6000的设备端口

先来看看CDev::init()字符设备初始化的过程

int CDev::init()

{

// base class init first

// 首先初始化基类

int ret = Device::init(); // 注册irq中断

if (ret != OK) {

goto out;

}

// now register the driver

// 现在注册驱动

if (\_devname != nullptr) {

ret = register\_driver(\_devname, &fops, 0666, (void \*)this); // 需要关注的是这个\_devname对应的设备

if (ret != OK) {

goto out;

}

\_registered = true;

}

out:

return ret;

}

看看设备是怎么初始化的：Device::init()

int

Device::init()

{

int ret = OK;

// If assigned an interrupt, connect it

if (\_irq) {

/\* ensure it's disabled \*/

up\_disable\_irq(\_irq);

/\* register \*/

// 注册中断

ret = register\_interrupt(\_irq, this);

if (ret != OK) {

\_irq = 0;

}

}

return ret;

}

/\*\*

\* Register an interrupt to a specific device.

\* 向特定设备注册中断。

\*

\* @param irq The interrupt number to register. 要注册的中断号码

\* @param owner The device receiving the interrupt. 接收中断的设备

\* @return OK if the interrupt was registered.

\*/

static int register\_interrupt(int irq, Device \*owner){

int ret = -ENOMEM;

// look for a slot where we can register the interrupt

for (unsigned i = 0; i < irq\_nentries; i++) {

if (irq\_entries[i].irq == 0) {

// great, we could put it here; try attaching it

ret = irq\_attach(irq, &interrupt);

if (ret == OK) {

irq\_entries[i].irq = irq;

irq\_entries[i].owner = owner;

}

break;

}

}

return ret;

}

关于这里为什么用irq(Interrupt Request, 中断请求)，能力有限，不得而知。

如果你了解这一块，烦请告知。

注册好中断以后，继续回到字符型设备的初始化函数中来：CDev::init()。现在注册驱动

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Name: register\_driver

\*

\* Description:

\* Register a character driver inode the pseudo file system.

\* 注册一个字符驱动程序inode到伪文件系统。

\*

\* Input parameters:

\* path - The path to the inode to create

\* fops - The file operations structure

\* mode - inmode priviledges (not used)

\* priv - Private, user data that will be associated with the inode.

\*

\* Returned Value:

\* Zero on success (with the inode point in 'inode'); A negated errno

\* value is returned on a failure (all error values returned by

\* inode\_reserve):

\*

\* EINVAL - 'path' is invalid for this operation

\* EEXIST - An inode already exists at 'path'

\* ENOMEM - Failed to allocate in-memory resources for the operation

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int register\_driver(FAR const char \*path, FAR const struct file\_operations \*fops,

mode\_t mode, FAR void \*priv)

{

FAR struct inode \*node;

...

}

笔者无力深究函数内部的实现过程，看函数的注释可以知道这里是注册了一个字符驱动程序inode(索引节点)到文件系统中。而下面这段驱动注册过程

ret = register\_driver(\_devname, &fops, 0666, (void \*)this);

* 1

就是将\_devname注册到了文件系统中，这个设备dev每个人可读写。

/\*

\* chmod指令用数字格式指定权限的改变

\* 每个Linux文件具有四种访问权限：可读(r)、可写(w)、可执行(x)和无权限(-)。

\* 例如 chmod 777 这里的777分别表示 owner group other

\* 模式 数字

\* rwx 7

\* rw- 6

\* r-x 5

\* r-- 4

\* -wx 3

\* -w- 2

\* --x 1

\* --- 0

\*

\* 所以代码中常见的666意思是模式为每个人可读和可写

\*/

关于inode的介绍，可以参考[这篇博客](http://www.cnblogs.com/itech/archive/2012/05/15/2502284.html)。对索引节点的一个简单理解是，通过它可以找到NuttX操作系统中不同文件（设备），inode中包含了文件除文件名外所有的元信息（文件创建者、创建日期、大小等），Unix/Linux系统内部不使用文件名，而使用inode号码来识别文件。

通过MPU6000\_gyro陀螺仪的实例化过程可以推测讨论的陀螺仪设备名称\_devname为path，关于path，继续往下看

MPU6000\_gyro::MPU6000\_gyro(MPU6000 \*parent, const char \*path) :

CDev("MPU6000\_gyro", path), // 陀螺仪设备端口

...

通过CDev::init()将字符型设备注册到了文件系统中，然后回到陀螺仪的驱动注册过程MPU6000\_gyro::init()，接下来需要将初始化生成的设备节点作为一个设备文件，对应用层开放，可以像访问一个文件一样访问

\_gyro\_class\_instance = register\_class\_devname(GYRO\_BASE\_DEVICE\_PATH); //注册节点

* 1

其中

#define GYRO\_BASE\_DEVICE\_PATH "/dev/gyro"

#define GYRO0\_DEVICE\_PATH "/dev/gyro0"

#define GYRO1\_DEVICE\_PATH "/dev/gyro1"

#define GYRO2\_DEVICE\_PATH "/dev/gyro2"

注册节点

int CDev::register\_class\_devname(const char \*class\_devname)

{

if (class\_devname == nullptr) {

return -EINVAL;

}

int class\_instance = 0;

int ret = -ENOSPC;

while (class\_instance < 4) {

char name[32];

snprintf(name, sizeof(name), "%s%d", class\_devname, class\_instance);

ret = register\_driver(name, &fops, 0666, (void \*)this); // 注册驱动

//这里相当于

// ret = register\_driver("/dev/gyro", &fops, 0666, (void \*)this);

if (ret == OK) { break; }

class\_instance++;

}

if (class\_instance == 4) {

return ret;

}

return class\_instance;

}

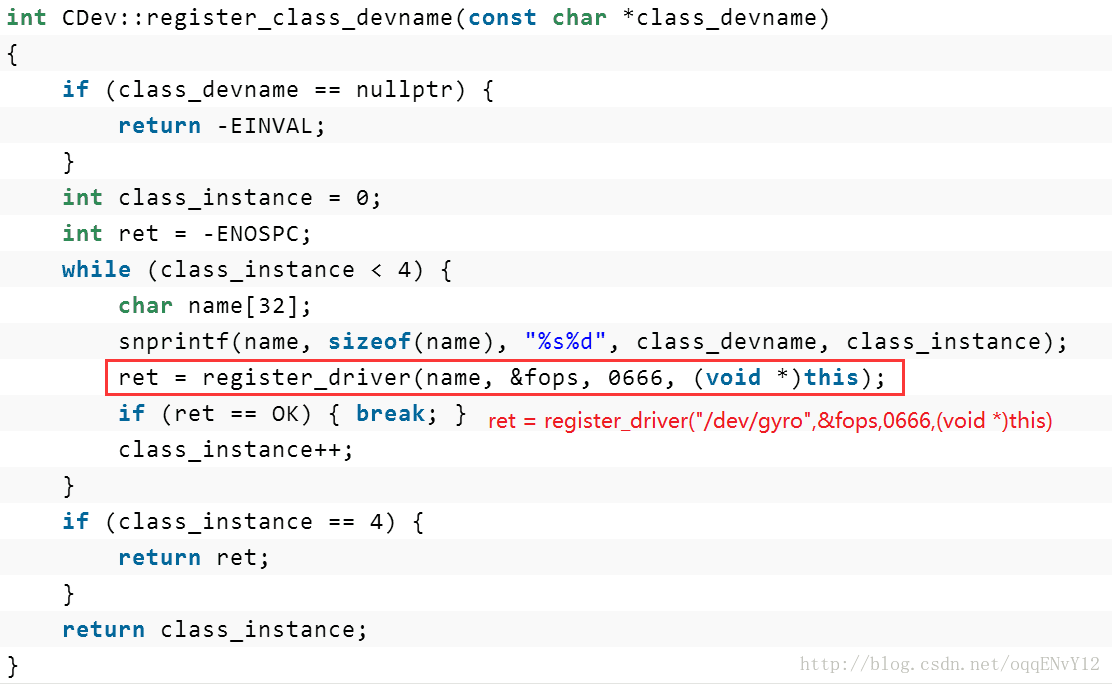
到这里/dev/gyro就可以作为一个文件设备open、read了。而关于调用的Device::init函数进行irq中断注册不是很明白，但是对于PX4中断传感器数据读取的话

驱动层是定时器，最底层听说是中断，然后驱动层的定时器直接去拿已有的数据即可

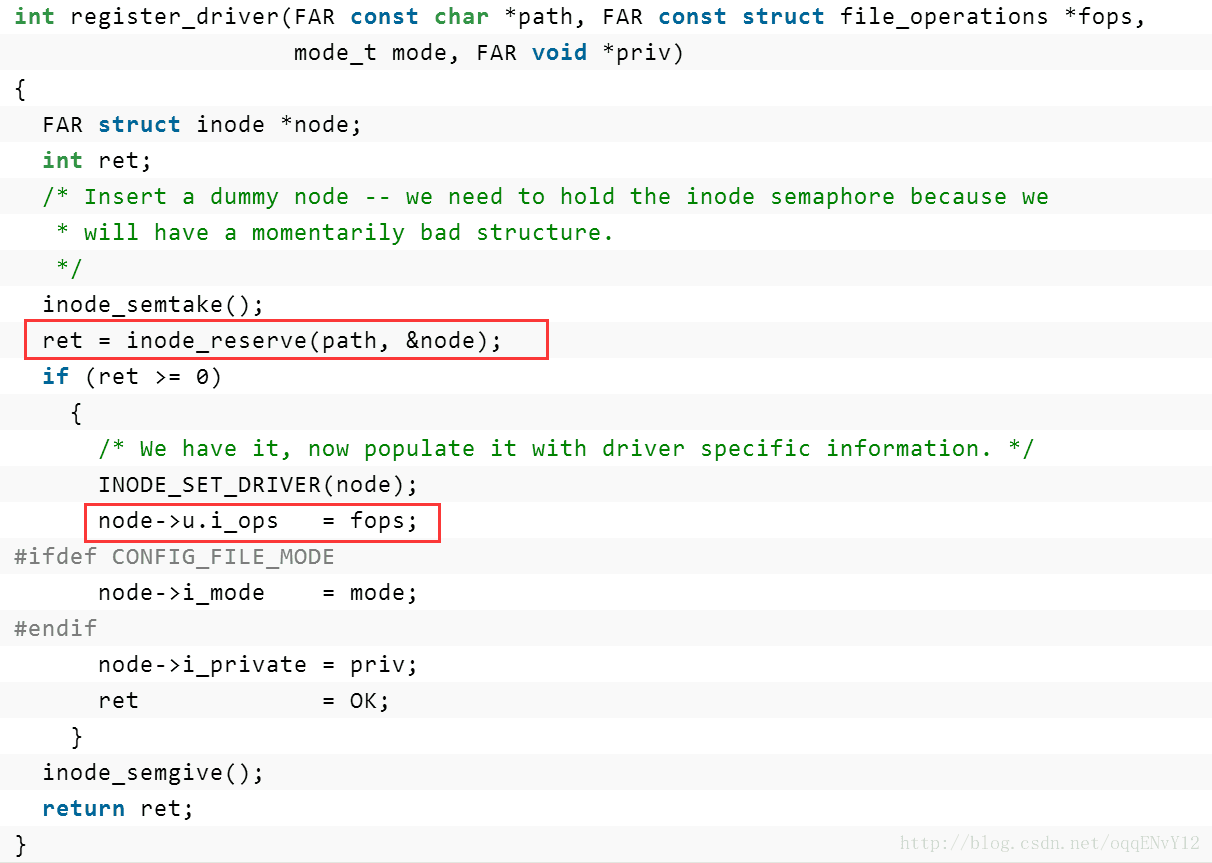
关于register\_class\_devname()这个函数作用应该就是将MPU6000中的陀螺仪这个设备/dev/gyro作为节点注册到了NuttX系统中。，并且作为一个SPI端口使用。

从下面几幅图可以更加直观的看到整个连接流程

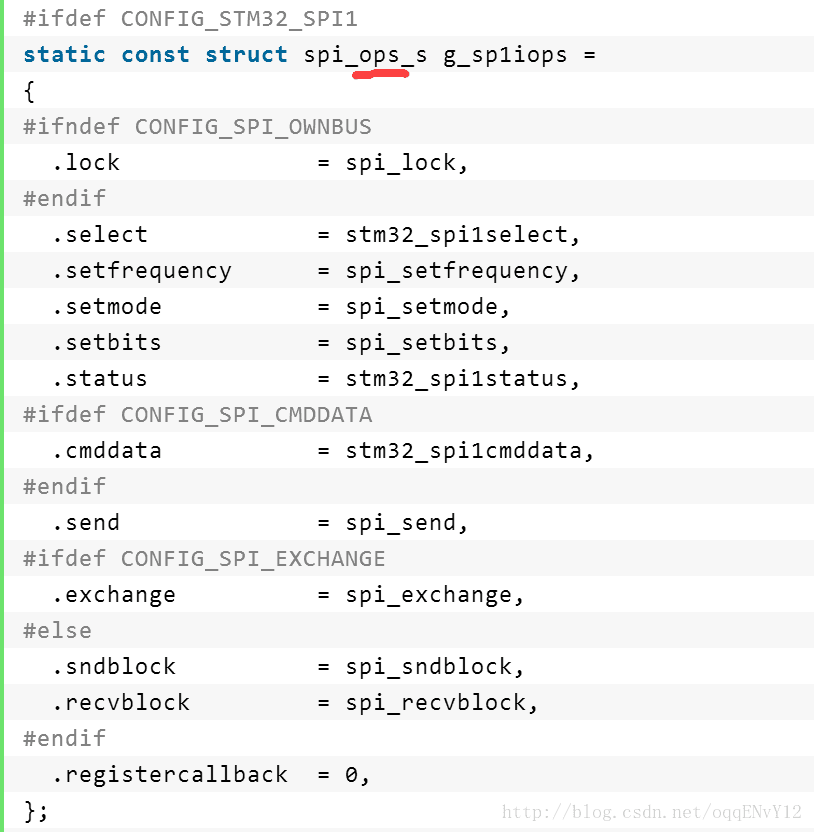
* 注册类设备名(直译)



* 注册驱动



* SPI节点操作



以上这一部分讲的实在是很业余，半猜半理解，对于设备与path的对应尚模棱两可，目的是将陀螺仪注册到文件系统中，后面要加传感器的话，笔者认为可以仿照着来。各取所需吧

就此打住了。

### MPU6000的启动过程

接下来主要分析mpu6000.cpp的内部逻辑，从程序启动到传感器读取的原始数据处理。进入主函数

int mpu6000\_main(int argc, char \*argv[])

{

enum MPU6000\_BUS busid = MPU6000\_BUS\_ALL;

int device\_type = 6000;

int ch;

bool external = false;

enum Rotation rotation = ROTATION\_NONE;

int accel\_range = 8;

/\* jump over start/off/etc and look at options first \*/

while ((ch = getopt(argc, argv, "T:XISsR:a:")) != EOF) {

switch (ch) {

case 'X':

busid = MPU6000\_BUS\_I2C\_EXTERNAL;

break;

case 'I':

busid = MPU6000\_BUS\_I2C\_INTERNAL;

break;

case 'S':

busid = MPU6000\_BUS\_SPI\_EXTERNAL;

break;

case 's':

busid = MPU6000\_BUS\_SPI\_INTERNAL;

break;

case 'T':

device\_type = atoi(optarg);

break;

case 'R':

rotation = (enum Rotation)atoi(optarg);

break;

case 'a':

accel\_range = atoi(optarg);

break;

default:

mpu6000::usage();

exit(0);

}

}

external = (busid == MPU6000\_BUS\_I2C\_EXTERNAL || busid == MPU6000\_BUS\_SPI\_EXTERNAL);

const char \*verb = argv[optind];

/\*

\* Start/load the driver.

\* 开始/加载驱动

\*/

if (!strcmp(verb, "start")) {

mpu6000::start(busid, rotation, accel\_range, device\_type, external);

}

if (!strcmp(verb, "stop")) {

mpu6000::stop(busid);

}

/\*

\* Test the driver/device.

\*/

if (!strcmp(verb, "test")) {

mpu6000::test(busid);

}

...

首先是对MPU6000传感器的硬件连接情况的判断，从传感器的启动脚本rc\_sensors可以初见端倪

if ver hwcmp PX4FMU\_V2

then

...

# Pixhawk的传感器启动

else

# FMUv2

if mpu6000 start

then

fi

if mpu9250 start

then

fi

if l3gd20 start

then

fi

if lsm303d start

then

fi

fi

fi

对于Pixhawk来说，MPU6000通过内部SPI总线连接。

关于Pixhawk上的MPU6000的总线配置全都是在驱动程序中写死了的，并没有在启动脚本中进行T:XISsR:a:的参数定义。

从这里可以看到：

enum MPU6000\_BUS busid = MPU6000\_BUS\_ALL;

int device\_type = 6000;

int ch;

bool external = false;

enum Rotation rotation = ROTATION\_NONE;

int accel\_range = 8;

* 1

接下来的command就跟其他的模块一样了，start、stop、test、reset……参数argv[optind]从启动脚本或者NSH传递到这里

* start → 打开驱动

void

start(enum MPU6000\_BUS busid, enum Rotation rotation, int range, int device\_type, bool external)

{

bool started = false;

for (unsigned i = 0; i < NUM\_BUS\_OPTIONS; i++) { //遍历

if (busid == MPU6000\_BUS\_ALL && bus\_options[i].dev != NULL) {

// this device is already started

// 设备已经打开了

continue;

}

if (busid != MPU6000\_BUS\_ALL && bus\_options[i].busid != busid) {

// not the one that is asked for

// 不是想要的总线

continue;

}

// 启动特定总线的驱动程序

started |= start\_bus(bus\_options[i], rotation, range, device\_type, external);

}

exit(started ? 0 : 1);

}

其中mpu6000\_bus\_option结构体列出了Pixhawk支持的所有总线配置，如下所示

struct mpu6000\_bus\_option {

enum MPU6000\_BUS busid;

const char \*accelpath;

const char \*gyropath;

MPU6000\_constructor interface\_constructor;

uint8\_t busnum;

MPU6000 \*dev;

} bus\_options[] = {

#if defined (USE\_I2C)

# if defined(PX4\_I2C\_BUS\_ONBOARD)

{ MPU6000\_BUS\_I2C\_INTERNAL, MPU\_DEVICE\_PATH\_ACCEL, MPU\_DEVICE\_PATH\_GYRO, &MPU6000\_I2C\_interface, PX4\_I2C\_BUS\_ONBOARD, NULL },

# endif

# if defined(PX4\_I2C\_BUS\_EXPANSION)

{ MPU6000\_BUS\_I2C\_EXTERNAL, MPU\_DEVICE\_PATH\_ACCEL\_EXT, MPU\_DEVICE\_PATH\_GYRO\_EXT, &MPU6000\_I2C\_interface, PX4\_I2C\_BUS\_EXPANSION, NULL },

# endif

#endif

#ifdef PX4\_SPIDEV\_MPU

{ MPU6000\_BUS\_SPI\_INTERNAL, MPU\_DEVICE\_PATH\_ACCEL, MPU\_DEVICE\_PATH\_GYRO, &MPU6000\_SPI\_interface, PX4\_SPI\_BUS\_SENSORS, NULL },

/\*内部SPI，加速度路径，陀螺仪路径，MPU6000的SPI接口，SPI1，null\*/

#endif

#if defined(PX4\_SPI\_BUS\_EXT)

{ MPU6000\_BUS\_SPI\_EXTERNAL, MPU\_DEVICE\_PATH\_ACCEL\_EXT, MPU\_DEVICE\_PATH\_GYRO\_EXT, &MPU6000\_SPI\_interface, PX4\_SPI\_BUS\_EXT, NULL },

#endif

};

然后启动特定总线的驱动程序

bool start\_bus(struct mpu6000\_bus\_option &bus, enum Rotation rotation, int range, int device\_type, bool external)

{

int fd = -1;

if (bus.dev != nullptr) {

warnx("%s SPI not available", external ? "External" : "Internal");

return false;

}

device::Device \*interface = bus.interface\_constructor(bus.busnum, device\_type, external);

/\*

\* 确定设备接口Interface(很重要)

\* busid = MPU6000\_BUS\_SPI\_INTERNAL

\* accelpath = MPU\_DEVICE\_PATH\_ACCEL(/dev/accel)

\* gyropath = MPU\_DEVICE\_PATH\_GYRO(/dev/gyro)

\* interface\_constructor = MPU6000的内部SPI片选(作为SPI类的加速度计实例)

\* busnum = PX4\_SPI\_BUS\_SENSORS(Pixhawk传感器连在SPI1上)

\* dev = 是否存在设备连接在此端口上

\*/

if (interface == nullptr) {

warnx("no device on bus %u", (unsigned)bus.busid);

return false;

}

if (interface->init() != OK) { // 设备初始化，向特定的设备注册中断请求

delete interface;

warnx("no device on bus %u", (unsigned)bus.busid);

return false;

}

bus.dev = new MPU6000(interface, bus.accelpath, bus.gyropath, rotation, device\_type); // 新建MPU6000类的实例

if (bus.dev == nullptr) {

delete interface;

return false;

}

if (OK != bus.dev->init()) { // MPU6000::init()

goto fail;

}

/\* set the poll rate to default, starts automatic data collection \*/

fd = open(bus.accelpath, O\_RDONLY);

if (fd < 0) {

goto fail;

}

// 注意ioctl：关于传感器轮询模式的配置，自动轮询/手动轮询；截止频率

if (ioctl(fd, SENSORIOCSPOLLRATE, SENSOR\_POLLRATE\_DEFAULT) < 0) {

goto fail;

}

if (ioctl(fd, ACCELIOCSRANGE, range) < 0) {

goto fail;

}

close(fd);

return true;

fail:

if (fd >= 0) {

close(fd);

}

if (bus.dev != nullptr) {

delete(bus.dev);

bus.dev = nullptr;

}

return false;

}

这个函数干的事情多了，确定MPU6000的最终设备ID，新建MU6000实例，主要是它调用了**MPU6000::init()**(前文中已经说明过其实现过程)。而MPU6000::init()是整个传感器功能的实现，函数中先将MPU6000注册到文件系统中，包括加速度计、陀螺仪两个设备，然后进行数据测量measure()。

* test → 驱动测试

/\*\*

\* Perform some basic functional tests on the driver;

\* make sure we can collect data from the sensor in polled

\* and automatic modes.

\*/

// 传感器作为一个文件设备，操作步骤

// open -> ioctl -> read -> close

void

test(enum MPU6000\_BUS busid)

{

struct mpu6000\_bus\_option &bus = find\_bus(busid);

accel\_report a\_report;

gyro\_report g\_report;

ssize\_t sz;

/\* get the driver \*/

int fd = open(bus.accelpath, O\_RDONLY);

if (fd < 0) {

err(1, "%s open failed (try 'mpu6000 start')", bus.accelpath);

}

/\* get the driver \*/

int fd\_gyro = open(bus.gyropath, O\_RDONLY);

if (fd\_gyro < 0) {

err(1, "%s open failed", bus.gyropath);

}

/\* reset to manual polling \*/

if (ioctl(fd, SENSORIOCSPOLLRATE, SENSOR\_POLLRATE\_MANUAL) < 0) {

err(1, "reset to manual polling");

}

/\* do a simple demand read \*/

sz = read(fd, &a\_report, sizeof(a\_report));

if (sz != sizeof(a\_report)) {

warnx("ret: %d, expected: %d", sz, sizeof(a\_report));

err(1, "immediate acc read failed");

}

warnx("single read");

warnx("time: %lld", a\_report.timestamp);

warnx("acc x: \t%8.4f\tm/s^2", (double)a\_report.x);

warnx("acc y: \t%8.4f\tm/s^2", (double)a\_report.y);

warnx("acc z: \t%8.4f\tm/s^2", (double)a\_report.z);

warnx("acc x: \t%d\traw 0x%0x", (short)a\_report.x\_raw, (unsigned short)a\_report.x\_raw);

warnx("acc y: \t%d\traw 0x%0x", (short)a\_report.y\_raw, (unsigned short)a\_report.y\_raw);

warnx("acc z: \t%d\traw 0x%0x", (short)a\_report.z\_raw, (unsigned short)a\_report.z\_raw);

warnx("acc range: %8.4f m/s^2 (%8.4f g)", (double)a\_report.range\_m\_s2,

(double)(a\_report.range\_m\_s2 / MPU6000\_ONE\_G));

/\* do a simple demand read \*/

sz = read(fd\_gyro, &g\_report, sizeof(g\_report));

if (sz != sizeof(g\_report)) {

warnx("ret: %d, expected: %d", sz, sizeof(g\_report));

err(1, "immediate gyro read failed");

}

warnx("gyro x: \t% 9.5f\trad/s", (double)g\_report.x);

warnx("gyro y: \t% 9.5f\trad/s", (double)g\_report.y);

warnx("gyro z: \t% 9.5f\trad/s", (double)g\_report.z);

warnx("gyro x: \t%d\traw", (int)g\_report.x\_raw);

warnx("gyro y: \t%d\traw", (int)g\_report.y\_raw);

warnx("gyro z: \t%d\traw", (int)g\_report.z\_raw);

warnx("gyro range: %8.4f rad/s (%d deg/s)", (double)g\_report.range\_rad\_s,

(int)((g\_report.range\_rad\_s / M\_PI\_F) \* 180.0f + 0.5f));

warnx("temp: \t%8.4f\tdeg celsius", (double)a\_report.temperature);

warnx("temp: \t%d\traw 0x%0x", (short)a\_report.temperature\_raw, (unsigned short)a\_report.temperature\_raw);

/\* reset to default polling \*/

if (ioctl(fd, SENSORIOCSPOLLRATE, SENSOR\_POLLRATE\_DEFAULT) < 0) {

err(1, "reset to default polling");

}

close(fd);

close(fd\_gyro);

/\* XXX add poll-rate tests here too \*/

reset(busid);

errx(0, "PASS");

}

从这个函数可以看出，PX4中的传感器数据读取是可以按照基本的文件操作方法实现的，

**open**→**ioctl**→**read**→**close**

印证了文章最开始的想法。

但是这样错太粗糙了，还是得进行数据预处理的。

### 数据处理过程

### 数据测量 [MPU6000::measure()](https://github.com/PX4/Firmware/blob/v1.5.2/src/drivers/mpu6000/mpu6000.cpp#L1805)

这个函数中是从传感器读数并进行数据处理并发布数据的过程。直接关系到传感器的最终读数。

#### read

数据读取过程由下面的代码实现

// sensor transfer at high clock speed

// 高速读取

if (sizeof(mpu\_report) != \_interface->read(MPU6000\_SET\_SPEED(MPUREG\_INT\_STATUS, MPU6000\_HIGH\_BUS\_SPEED),

(uint8\_t \*)&mpu\_report,

sizeof(mpu\_report))) {

return -EIO;

}

check\_registers(); // 寄存器数据检查

显然，使用的是read，将mpu6000作为文件进行读操作。

接下来就是数据处理了

#### 大小端处理

/\*

\* Convert from big to little endian

\* 从大端到小端

\*

\* Little-Endian就是低位字节排放在内存的低地址端，高位字节排放在内存的高地址端。

\* Big-Endian就是高位字节排放在内存的低地址端，低位字节排放在内存的高地址端。

\*/

report.accel\_x = int16\_t\_from\_bytes(mpu\_report.accel\_x); // 将测得的值mpu\_report传递给将发布的值report

report.accel\_y = int16\_t\_from\_bytes(mpu\_report.accel\_y);

report.accel\_z = int16\_t\_from\_bytes(mpu\_report.accel\_z);

report.temp = int16\_t\_from\_bytes(mpu\_report.temp);

report.gyro\_x = int16\_t\_from\_bytes(mpu\_report.gyro\_x);

report.gyro\_y = int16\_t\_from\_bytes(mpu\_report.gyro\_y);

report.gyro\_z = int16\_t\_from\_bytes(mpu\_report.gyro\_z);

mpu\_report为传感器测得值，report为最终采集到的值(需进一步处理)。

int16\_t int16\_t\_from\_bytes(uint8\_t bytes[])

{

union {

uint8\_t b[2];

int16\_t w;

} u;

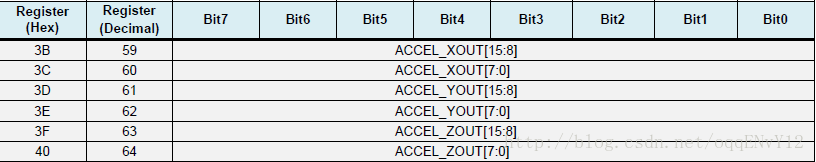
u.b[1] = bytes[0];

u.b[0] = bytes[1];

return u.w;

}

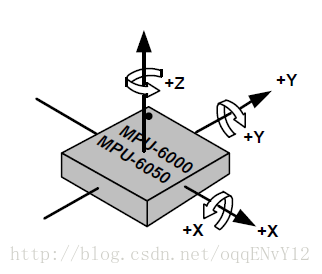
对于一个16位传感器MPU6000，其数据存储寄存器分为高八位、低八位，各占一个字节。



SPI协议的话，数据时一位一位地读取，因此以大端模式保存到2个含8个元素的数组uint8\_t b[2]中，但最终要处理的是一个16位的无符号整型数据int16\_t w，所以要进行大小端数据转换。

#### 交换XY轴

对于Pixhawk来说，MPU6000是放置在飞控板底面的，也就是绕其自身Y轴旋转了180°。



而对于PX4固件来说，其使用的是NED坐标系，要完成软硬件的匹配，在驱动层采用了交换XY轴的方法。

/\*

\* Swap axes and negate y

\*

\* 交换xy轴读数并将新的y轴读取取负

\*

\* 理由是 正放的话，MPU6K y向前，x向右, z向上。 但是

\* Pixhawk 中，传感器是倒置的， x向前, y向右，z向下。

\*/

int16\_t accel\_xt = report.accel\_y;

int16\_t accel\_yt = ((report.accel\_x == -32768) ? 32767 : -report.accel\_x);

int16\_t gyro\_xt = report.gyro\_y;

int16\_t gyro\_yt = ((report.gyro\_x == -32768) ? 32767 : -report.gyro\_x);

/\*

\* Apply the swap

\*/

report.accel\_x = accel\_xt;

report.accel\_y = accel\_yt;

report.gyro\_x = gyro\_xt;

report.gyro\_y = gyro\_yt;

接下来是一连串的数据处理过程：二阶低通滤波、设置分辨率/缩放因子、积分环节、自定义旋转，我们只需要关心最后的发布即可。

if (accel\_notify && !(\_pub\_blocked)) {

/\* log the time of this report \*/

perf\_begin(\_controller\_latency\_perf);

/\* publish it \*/

//////////////////////// 发布加速度计主题 //////////////////////

orb\_publish(ORB\_ID(sensor\_accel), \_accel\_topic, &arb);

}

if (gyro\_notify && !(\_pub\_blocked)) {

/\* publish it \*/

//////////////////////// 发布陀螺仪主题 //////////////////////

orb\_publish(ORB\_ID(sensor\_gyro), \_gyro->\_gyro\_topic, &grb);

}

### 自动测量[MPU6000::start()](https://github.com/PX4/Firmware/blob/v1.5.2/src/drivers/mpu6000/mpu6000.cpp#L1655)

调用此函数的话，启动自动测量模式。详情请查看MPU6000::ioctl()函数

void

MPU6000::start()

{

/\* make sure we are stopped first \*/

uint32\_t last\_call\_interval = \_call\_interval;

stop();

\_call\_interval = last\_call\_interval;

/\* discard any stale data in the buffers \*/

// 清空缓冲区

\_accel\_reports->flush();

\_gyro\_reports->flush();

if (!is\_i2c()) {

/\* start polling at the specified rate \*/

// 以指定的速率开始轮询

hrt\_call\_every(&\_call,

1000,

\_call\_interval - MPU6000\_TIMER\_REDUCTION,

(hrt\_callout)&MPU6000::measure\_trampoline, this); // 定时器

} else {

// 与I2C相关的所有都不用管，Pixhawk的MPU6000接在SPI总线上

#ifdef USE\_I2C

/\* schedule a cycle to start things \*/

work\_queue(HPWORK, &\_work, (worker\_t)&MPU6000::cycle\_trampoline, this, 1);

#endif

}

}

接下来就是循环测量更新了

void MPU6000::cycle() //循环

{

int ret = measure();

if (ret != OK) {

/\* issue a reset command to the sensor \*/

reset();

start();

return;

}

if (\_call\_interval != 0) {

work\_queue(HPWORK,

&\_work,

(worker\_t)&MPU6000::cycle\_trampoline,

this,

USEC2TICK(\_call\_interval - MPU6000\_TIMER\_REDUCTION));

}

}

#endif

void MPU6000::measure\_trampoline(void \*arg)

{

MPU6000 \*dev = reinterpret\_cast<MPU6000 \*>(arg);

/\* make another measurement \*/

dev->measure(); // 数据测量

}

### 寄存器检查

最后的话，提一下文件中对寄存器读数有效性检查的处理

首先，对于MPU6000来说，其内部关键的寄存器列表如下

// 使用的寄存器列表

const uint8\_t MPU6000::\_checked\_registers[MPU6000\_NUM\_CHECKED\_REGISTERS] = { MPUREG\_PRODUCT\_ID,

MPUREG\_PWR\_MGMT\_1, /\* 电源 \*/

MPUREG\_USER\_CTRL,

MPUREG\_SMPLRT\_DIV, /\* 频率 \*/

MPUREG\_CONFIG,

MPUREG\_GYRO\_CONFIG, /\* 加计量程 \*/

MPUREG\_ACCEL\_CONFIG, /\* 陀螺量程 \*/

MPUREG\_INT\_ENABLE,

MPUREG\_INT\_PIN\_CFG,

MPUREG\_ICM\_UNDOC1

//检查寄存器

void MPU6000::check\_registers(void)

{

uint8\_t v;

// the MPUREG\_ICM\_UNDOC1 is specific to the ICM20608 (and undocumented)

// 不是 ICM20608

if (\_checked\_registers[\_checked\_next] == MPUREG\_ICM\_UNDOC1 && !is\_icm\_device()) {

\_checked\_next = (\_checked\_next + 1) % MPU6000\_NUM\_CHECKED\_REGISTERS;

}

if ((v = read\_reg(\_checked\_registers[\_checked\_next], MPU6000\_HIGH\_BUS\_SPEED)) !=

\_checked\_values[\_checked\_next]) { // 从寄存器读取到的值和写入的值不同

// 如果我们得到错误的值，那么我们知道SPI总线或传感器问题很严重 。

// 我们将\_register\_wait设置为20，然后等连续看到20个良好的值。

// 再次认为传感器健康，

perf\_count(\_bad\_registers);

/\*

try to fix the bad register value. We only try to

fix one per loop to prevent a bad sensor hogging the

bus.

\*/

if (\_register\_wait == 0 || \_checked\_next == 0) {

// if the product\_id is wrong then reset the

// sensor completely

write\_reg(MPUREG\_PWR\_MGMT\_1, BIT\_H\_RESET); // 0x80

// after doing a reset we need to wait a long

// time before we do any other register writes

// or we will end up with the mpu6000 in a

// bizarre state where it has all correct

// register values but large offsets on the

// accel axes

\_reset\_wait = hrt\_absolute\_time() + 10000;

\_checked\_next = 0;

} else {

write\_reg(\_checked\_registers[\_checked\_next], \_checked\_values[\_checked\_next]); // 向寄存器写入检验后的值

\_reset\_wait = hrt\_absolute\_time() + 3000;

}

\_register\_wait = 20;

}

\_checked\_next = (\_checked\_next + 1) % MPU6000\_NUM\_CHECKED\_REGISTERS;

}

写一个寄存器，更新\_checked\_values

/\*\*

\* Write a register in the MPU6000, updating \_checked\_values

\*

\* @param reg The register to write.

\* @param value The new value to write.

\*/

void MPU6000::write\_checked\_reg(unsigned reg, uint8\_t value)

{

write\_reg(reg, value); // 写寄存器

for (uint8\_t i = 0; i < MPU6000\_NUM\_CHECKED\_REGISTERS; i++) {

if (reg == \_checked\_registers[i]) { // 寄存器列表中有这个寄存器

\_checked\_values[i] = value; // 将写入的寄存器值赋给\_checked\_values[i]

}

}

}

From 吴神

说mpu6000为什么要有个gyro class

**两个原因**

第一，有的传感器分两三块类型的数据，比如陀螺和加速度，但是这多种数据的到达时间是不一样的，就是DRDY引脚的高电平响应，所以要分开来采样

第二，传感器有不同的数据类型也决定了，有的时候有些部分是坏的，不能用。比如说LSM303D这个传感器，加速度计就容易坏掉，这种情况下为了不影响磁力计的输出，这个cpp也是把两个分开来的

过程有些凌乱，不应该。

笔者觉得一个真的懂这些的人是能够把问题的本质抽象出来的，可以以简单的流程图说明整个过程的，能让旁人一看就懂的，革命尚未成功。

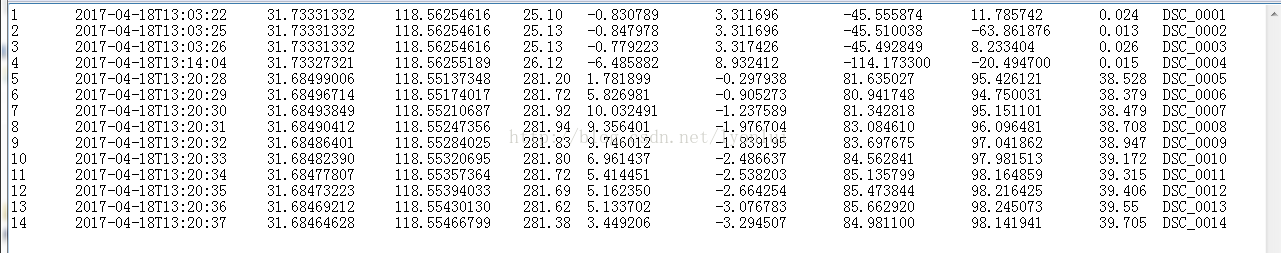
接下来就准备自定义

## 为相片添加POS数据

为单反拍摄照片批量加入地理位置信息（POS点）和EXIF信息

    很多航测朋友对于照片中没有POS点很是懊恼，单反同飞控是分离的，照片由单反生成，而POS点的数据由飞控生成。对于后期数据处理相当不利，尤其是目前对于倾斜摄影的朋友，地理位置信息一定要写到照片中去，而动辄几百上千的照片通过手工加入地理位置信息，是非常不现实的。所以我编写了一个程序，可以通过这个程序将飞控生成得POS点数据批量加入单反所拍摄的照片中。

下面以纵横的飞控生成的POS数据为列子：

数据是以.txt文本的形式进行保存的。

我们需要的数据是 第三列的经度，第四列的纬度，第五列的高度信息。

当然，我的程序需要POS点信息排序同照片的名字排序是都是増序，即POS信息第一条数据是DSC\_0001，假设单反拍摄的第一张照片DSC\_0078.JPG，那么这就要求DSC\_0002数据对应的照片名字为DSC\_0079.JPG，以此类推。

那么怎么批量为照片添加POS数据呢？

首先，我们得为一张照片添加POS点，然后才能为大量照片添加，首先我来介绍一下我们的主角exiv2

大部分exif的库都不是很好用，exiv2 ([http://www.exiv2.org](http://www.exiv2.org/)) 挺不错。但在windows下编译很烦。

我们可以通过命令

exiv2 -M"set Exif.GPSInfo.GPSLatitude 4/1 15/1 33/1" -M"set Exif.GPSInfo.GPSLatitudeRef N" image.jpg

这个命令就是为照片添加一个卑微4°15′33″的GPS信息

当我们要批量为照片添加GPS信息时，我们可以通过读入飞控生成的GPS信息然后循环使用3条命令将这些信息写入照片中：1.添加经度信息，2.添加纬度信息，3.添加高度信息。

                cmd\_Latitude =" -M\"set Exif.GPSInfo.GPSLatitude"+latitude\_value+"\""+"-M\"setExif.GPSInfo.GPSLatitudeRefN\" ";

cmd\_Longitude =" -M \"set Exif.GPSInfo.GPSLongitude "+Longitude\_value+"\""+" -M \"set Exif.GPSInfo.GPSLongitudeRef E\" ";

cmd\_GPSAltitude=" -M \"set Exif.GPSInfo.GPSAltitude "+Altitude\_value+"\" ";

qDebug((cmd\_Latitude+cmd\_Longitude+cmd\_GPSAltitude+fileName).toLatin1().data());

p.start("exiv2.exe"+cmd\_Latitude+cmd\_Longitude+cmd\_GPSAltitude+fileName);

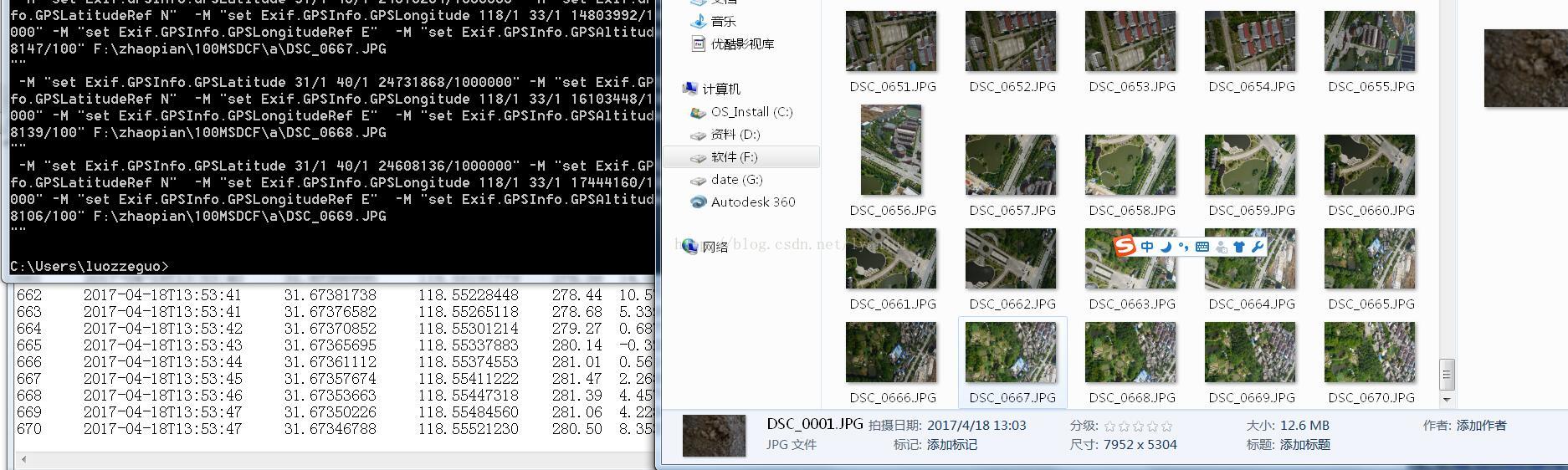
p.waitForStarted();

p.waitForFinished();

qDebug()<<QString::fromLocal8Bit(p.readAllStandardError());

以上就是程序的主要代码实现，代码是使用Qt编写的，上面是通过Qt调用exiv2来执行命令来将信息写入。

执行程序之后可以为照片添加GPS信息



exiv2还同样支持以下命令：

Exif.Image.ProcessingSoftware

Exif.Image.NewSubfileType

Exif.Image.SubfileType

Exif.Image.ImageWidth

Exif.Image.ImageLength

Exif.Image.BitsPerSample

Exif.Image.Compression

Exif.Image.PhotometricInterpretation

Exif.Image.Thresholding

Exif.Image.CellWidth

Exif.Image.CellLength

Exif.Image.FillOrder

Exif.Image.DocumentName

Exif.Image.ImageDescription

Exif.Image.Make

Exif.Image.Model

Exif.Image.StripOffsets

Exif.Image.Orientation

Exif.Image.SamplesPerPixel

Exif.Image.RowsPerStrip

Exif.Image.StripByteCounts

Exif.Image.XResolution

Exif.Image.YResolution

Exif.Image.PlanarConfiguration

Exif.Image.GrayResponseUnit

Exif.Image.GrayResponseCurve

Exif.Image.T4Options

Exif.Image.T6Options

Exif.Image.ResolutionUnit

Exif.Image.PageNumber

Exif.Image.TransferFunction

Exif.Image.Software

Exif.Image.DateTime

Exif.Image.Artist

Exif.Image.HostComputer

Exif.Image.Predictor

Exif.Image.WhitePoint

Exif.Image.PrimaryChromaticities

Exif.Image.ColorMap

Exif.Image.HalftoneHints

Exif.Image.TileWidth

Exif.Image.TileLength

Exif.Image.TileOffsets

Exif.Image.TileByteCounts

Exif.Image.SubIFDs

Exif.Image.InkSet

Exif.Image.InkNames

Exif.Image.NumberOfInks

Exif.Image.DotRange

Exif.Image.TargetPrinter

Exif.Image.ExtraSamples

Exif.Image.SampleFormat

Exif.Image.SMinSampleValue

Exif.Image.SMaxSampleValue

Exif.Image.TransferRange

Exif.Image.ClipPath

Exif.Image.XClipPathUnits

Exif.Image.YClipPathUnits

Exif.Image.Indexed

Exif.Image.JPEGTables

Exif.Image.OPIProxy

Exif.Image.JPEGProc

Exif.Image.JPEGInterchangeFormat

Exif.Image.JPEGInterchangeFormatLength

Exif.Image.JPEGRestartInterval

Exif.Image.JPEGLosslessPredictors

Exif.Image.JPEGPointTransforms

Exif.Image.JPEGQTables

Exif.Image.JPEGDCTables

Exif.Image.JPEGACTables

Exif.Image.YCbCrCoefficients

Exif.Image.YCbCrSubSampling

Exif.Image.YCbCrPositioning

Exif.Image.ReferenceBlackWhite

Exif.Image.XMLPacket

Exif.Image.Rating

Exif.Image.RatingPercent

Exif.Image.ImageID

Exif.Image.CFARepeatPatternDim

Exif.Image.CFAPattern

Exif.Image.BatteryLevel

Exif.Image.Copyright

Exif.Image.ExposureTime

Exif.Image.FNumber

Exif.Image.IPTCNAA

Exif.Image.ImageResources

Exif.Image.ExifTag

Exif.Image.InterColorProfile

Exif.Image.ExposureProgram

Exif.Image.SpectralSensitivity

Exif.Image.GPSTag

Exif.Image.ISOSpeedRatings

Exif.Image.OECF

Exif.Image.Interlace

Exif.Image.TimeZoneOffset

Exif.Image.SelfTimerMode

Exif.Image.DateTimeOriginal

Exif.Image.CompressedBitsPerPixel

Exif.Image.ShutterSpeedValue

Exif.Image.ApertureValue

Exif.Image.BrightnessValue

Exif.Image.ExposureBiasValue

Exif.Image.MaxApertureValue

Exif.Image.SubjectDistance

Exif.Image.MeteringMode

Exif.Image.LightSource

Exif.Image.Flash

Exif.Image.FocalLength

Exif.Image.FlashEnergy

Exif.Image.SpatialFrequencyResponse

Exif.Image.Noise

Exif.Image.FocalPlaneXResolution

Exif.Image.FocalPlaneYResolution

Exif.Image.FocalPlaneResolutionUnit

Exif.Image.ImageNumber

Exif.Image.SecurityClassification

Exif.Image.ImageHistory

Exif.Image.SubjectLocation

Exif.Image.ExposureIndex

Exif.Image.TIFFEPStandardID

Exif.Image.SensingMethod

Exif.Image.XPTitle

Exif.Image.XPComment

Exif.Image.XPAuthor

Exif.Image.XPKeywords

Exif.Image.XPSubject

Exif.Image.PrintImageMatching

Exif.Image.DNGVersion

Exif.Image.DNGBackwardVersion

Exif.Image.UniqueCameraModel

Exif.Image.LocalizedCameraModel

Exif.Image.CFAPlaneColor

Exif.Image.CFALayout

Exif.Image.LinearizationTable

Exif.Image.BlackLevelRepeatDim

Exif.Image.BlackLevel

Exif.Image.BlackLevelDeltaH

Exif.Image.BlackLevelDeltaV

Exif.Image.WhiteLevel

Exif.Image.DefaultScale

Exif.Image.DefaultCropOrigin

Exif.Image.DefaultCropSize

Exif.Image.ColorMatrix1

Exif.Image.ColorMatrix2

Exif.Image.CameraCalibration1

Exif.Image.CameraCalibration2

Exif.Image.ReductionMatrix1

Exif.Image.ReductionMatrix2

Exif.Image.AnalogBalance

Exif.Image.AsShotNeutral

Exif.Image.AsShotWhiteXY

Exif.Image.BaselineExposure

Exif.Image.BaselineNoise

Exif.Image.BaselineSharpness

Exif.Image.BayerGreenSplit

Exif.Image.LinearResponseLimit

Exif.Image.CameraSerialNumber

Exif.Image.LensInfo

Exif.Image.ChromaBlurRadius

Exif.Image.AntiAliasStrength

Exif.Image.ShadowScale

Exif.Image.DNGPrivateData

Exif.Image.MakerNoteSafety

Exif.Image.CalibrationIlluminant1

Exif.Image.CalibrationIlluminant2

Exif.Image.BestQualityScale

Exif.Image.RawDataUniqueID

Exif.Image.OriginalRawFileName

Exif.Image.OriginalRawFileData

Exif.Image.ActiveArea

Exif.Image.MaskedAreas

Exif.Image.AsShotICCProfile

Exif.Image.AsShotPreProfileMatrix

Exif.Image.CurrentICCProfile

Exif.Image.CurrentPreProfileMatrix

Exif.Image.ColorimetricReference

Exif.Image.CameraCalibrationSignature

Exif.Image.ProfileCalibrationSignature

Exif.Image.AsShotProfileName

Exif.Image.NoiseReductionApplied

Exif.Image.ProfileName

Exif.Image.ProfileHueSatMapDims

Exif.Image.ProfileHueSatMapData1

Exif.Image.ProfileHueSatMapData2

Exif.Image.ProfileToneCurve

Exif.Image.ProfileEmbedPolicy

Exif.Image.ProfileCopyright

Exif.Image.ForwardMatrix1

Exif.Image.ForwardMatrix2

Exif.Image.PreviewApplicationName

Exif.Image.PreviewApplicationVersion

Exif.Image.PreviewSettingsName

Exif.Image.PreviewSettingsDigest

Exif.Image.PreviewColorSpace

Exif.Image.PreviewDateTime

Exif.Image.RawImageDigest

Exif.Image.OriginalRawFileDigest

Exif.Image.SubTileBlockSize

Exif.Image.RowInterleaveFactor

Exif.Image.ProfileLookTableDims

Exif.Image.ProfileLookTableData

Exif.Image.OpcodeList1

Exif.Image.OpcodeList2

Exif.Image.OpcodeList3

Exif.Image.NoiseProfile

Exif.Photo.ExposureTime

Exif.Photo.FNumber

Exif.Photo.ExposureProgram

Exif.Photo.SpectralSensitivity

Exif.Photo.ISOSpeedRatings

Exif.Photo.OECF

Exif.Photo.SensitivityType

Exif.Photo.StandardOutputSensitivity

Exif.Photo.RecommendedExposureIndex

Exif.Photo.ISOSpeed

Exif.Photo.ISOSpeedLatitudeyyy

Exif.Photo.ISOSpeedLatitudezzz

Exif.Photo.ExifVersion

Exif.Photo.DateTimeOriginal

Exif.Photo.DateTimeDigitized

Exif.Photo.ComponentsConfiguration

Exif.Photo.CompressedBitsPerPixel

Exif.Photo.ShutterSpeedValue

Exif.Photo.ApertureValue

Exif.Photo.BrightnessValue

Exif.Photo.ExposureBiasValue

Exif.Photo.MaxApertureValue

Exif.Photo.SubjectDistance

Exif.Photo.MeteringMode

Exif.Photo.LightSource

Exif.Photo.Flash

Exif.Photo.FocalLength

Exif.Photo.SubjectArea

Exif.Photo.MakerNote

Exif.Photo.UserComment

Exif.Photo.SubSecTime

Exif.Photo.SubSecTimeOriginal

Exif.Photo.SubSecTimeDigitized

Exif.Photo.FlashpixVersion

Exif.Photo.ColorSpace

Exif.Photo.PixelXDimension

Exif.Photo.PixelYDimension

Exif.Photo.RelatedSoundFile

Exif.Photo.InteroperabilityTag

Exif.Photo.FlashEnergy

Exif.Photo.SpatialFrequencyResponse

Exif.Photo.FocalPlaneXResolution

Exif.Photo.FocalPlaneYResolution

Exif.Photo.FocalPlaneResolutionUnit

Exif.Photo.SubjectLocation

Exif.Photo.ExposureIndex

Exif.Photo.SensingMethod

Exif.Photo.FileSource

Exif.Photo.SceneType

Exif.Photo.CFAPattern

Exif.Photo.CustomRendered

Exif.Photo.ExposureMode

Exif.Photo.WhiteBalance

Exif.Photo.DigitalZoomRatio

Exif.Photo.FocalLengthIn35mmFilm

Exif.Photo.SceneCaptureType

Exif.Photo.GainControl

Exif.Photo.Contrast

Exif.Photo.Saturation

Exif.Photo.Sharpness

Exif.Photo.DeviceSettingDescription

Exif.Photo.SubjectDistanceRange

Exif.Photo.ImageUniqueID

Exif.Photo.CameraOwnerName

Exif.Photo.BodySerialNumber

Exif.Photo.LensSpecification

Exif.Photo.LensMake

Exif.Photo.LensModel

Exif.Photo.LensSerialNumber

Exif.Iop.InteroperabilityIndex

Exif.Iop.InteroperabilityVersion

Exif.Iop.RelatedImageFileFormat

Exif.Iop.RelatedImageWidth

Exif.Iop.RelatedImageLength

Exif.GPSInfo.GPSVersionID

Exif.GPSInfo.GPSLatitudeRef

Exif.GPSInfo.GPSLatitude

Exif.GPSInfo.GPSLongitudeRef

Exif.GPSInfo.GPSLongitude

Exif.GPSInfo.GPSAltitudeRef

Exif.GPSInfo.GPSAltitude

Exif.GPSInfo.GPSTimeStamp

Exif.GPSInfo.GPSSatellites

Exif.GPSInfo.GPSStatus

Exif.GPSInfo.GPSMeasureMode

Exif.GPSInfo.GPSDOP

Exif.GPSInfo.GPSSpeedRef

Exif.GPSInfo.GPSSpeed

Exif.GPSInfo.GPSTrackRef

Exif.GPSInfo.GPSTrack

Exif.GPSInfo.GPSImgDirectionRef

Exif.GPSInfo.GPSImgDirection

Exif.GPSInfo.GPSMapDatum

Exif.GPSInfo.GPSDestLatitudeRef

Exif.GPSInfo.GPSDestLatitude

Exif.GPSInfo.GPSDestLongitudeRef

Exif.GPSInfo.GPSDestLongitude

Exif.GPSInfo.GPSDestBearingRef

Exif.GPSInfo.GPSDestBearing

Exif.GPSInfo.GPSDestDistanceRef

Exif.GPSInfo.GPSDestDistance

Exif.GPSInfo.GPSProcessingMethod

Exif.GPSInfo.GPSAreaInformation

Exif.GPSInfo.GPSDateStamp

Exif.GPSInfo.GPSDifferential

为照片添加更多的EXIF信息。

因为不同飞控生成的数据可能不同，所以并没有将纵横的POS数据处理代码贴出了，对其他飞控POS数据有处理要求或者需要为照片添加其他EXIF数据的朋友可以发送邮件联系我邮箱：