第2部

第5章 姿勢制御のイメージをつかむ

まずはここから…PID制御飛行プログラム

藤原 大悟

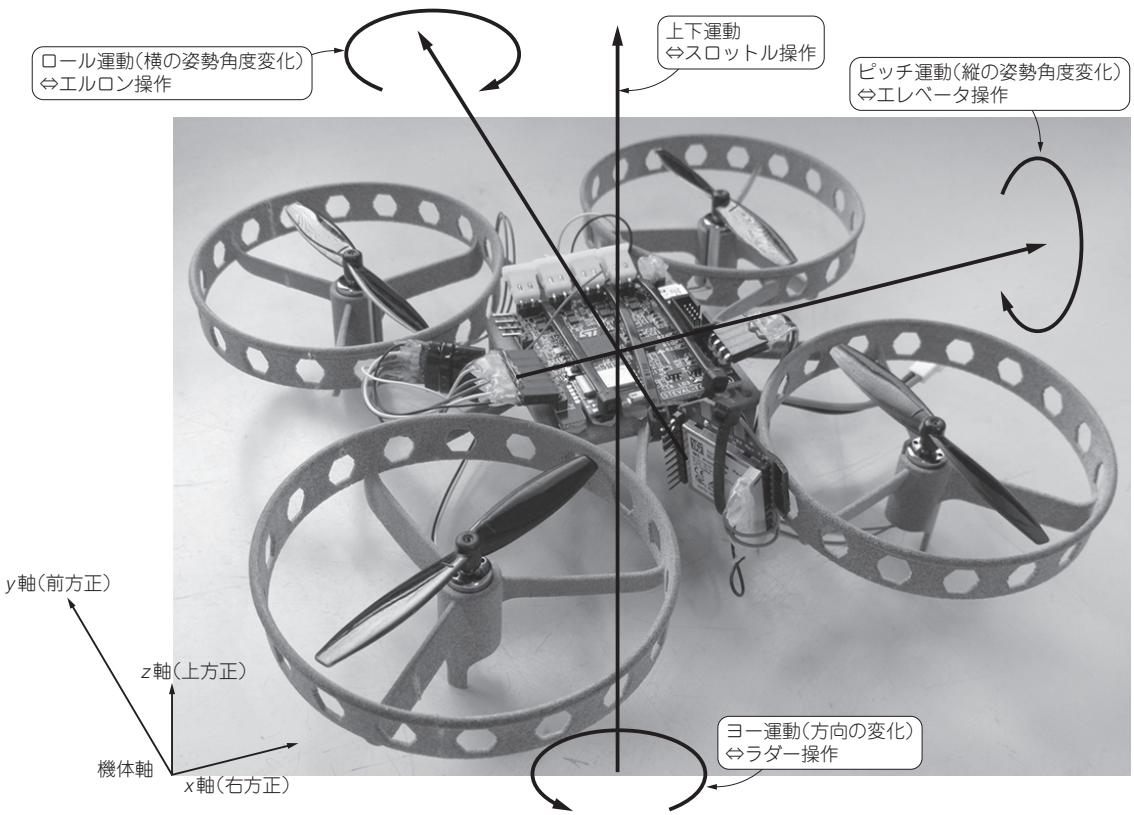

図1 ドローンの運動の定義
プロポの操作に対応した4つの運動がある

ST-DRONEのFCU(飛行制御ユニット基板)プログラム(ファームウェア)のソースコードは，リファレンス・デザイン(型名：STSW-FCU001)がGitHubにて公開されており，無償で入手可能です．これを読み解くだけでもドローンの制御について理解を深めることができます．

`https://www.st.com/ja/embedded-software/stsw-fcu001.html`

本章では，ドローンの姿勢推定，姿勢制御およびプロポと受信機の処理を，ソースコードや図を見ながら解説します(**図1**)．

ST社提供のプログラムのディレクトリ構成

● プロポを使うかスマホを使うかでディレクトリを選ぶ

ST_Drone_FCU_F401ディレクトリの中には，3つのサブディレクトリがあります．このサブディレクトリSTM32 FW Projectの下にFCUのソースコードがあります．

この下にはさらに3つのサブディレクトリがあり，操縦かんとしてプロポを使うか，またはスマートフォ

ンを利用するかによって使うサブディレクトリが異なります．今回はプロポを使うのでOfficial latest release 221117を使います(**図2**)．このOfficial latest release 221117より下は，1つのファイルと8つのサブ・ディレクトリで構成されています．

● 設定保存用ファイル

まず，ファイルSTEVAL_FCU001V1_ver1.iocは，STM32シリーズマイコン用のソースコードのひな形を生成するソフトウェア「CubeMX」が生成した設定保存用ファイルです．リファレンス・デザインのソースコードは一通り完成形になっているので，改めてCubeMXを使う必要はありませんが，機能拡張をしたい場合など必要があれば利用することになります．

● サブディレクトリの中身

▶4つのディレクトリは統合開発環境で使用する

開発時に利用する統合開発環境に応じて，統合開発環境で使用するファイルが入ったEWARM，MDK-ARM，SW4STM32，TrueSTUDIOの4つのサブディレクトリがあります．本稿では，統合開発環境としてAtollic TrueSTUDIOを利用するので，TrueSTUDIOディレクトリのみ使うことになります．残りの3つは使わないので消しても支障はありません．

▶Driversディレクトリ

Driversディレクトリには，CubeMXが生成したユーザ・プログラムとマイコンの間の橋渡しをするソースコードが入っています．従来ユーザ・プログラムがマイコンの各機能へアクセスする際にはマイコンのレジスタを直接読み書きしますが，CubeMXのひな形にユーザ・プログラムを実装する際は，直接レジスタにアクセスする代わりに，Driversディレクトリ内の関数へアクセスします．ハードウェアが抽象化されることで，ハードウェア依存度が下がり，移植性の高いプログラムを作ることができます．

▶Midelewaresディレクトリ

Midelewaresディレクトリには，その名の通り各種ミドルウェアのソースコードが入っています．使用したいミドルウェアをCubeMX上で選択すると，ここにそのソースコードが生成されます．今回直接触ることはありませんので詳細は省略します．

▶Inc，Srcディレクトリ

残る2つのIncとSrcディレクトリの中身が，ユーザ・プログラムのソースコードです．Incにはヘッダ・ファイル(拡張子が.h)が，Srcにはメイン・プログラム(拡張子が.c)のファイルがそれぞれ入ります．ドローンの制御に直接関係するプログラムはここに記述されています．

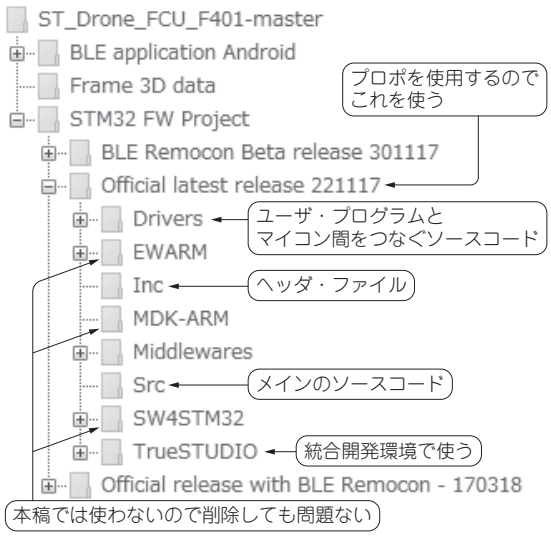

図2　FCUプログラムのディレクトリ構成は1つのファイルと8つのサブ・ディレクトリから成る

リファレンス・デザインはSTマイクロエレクトロニクスのホームページからのリンクでGitHubから入手できる．STSW-FCU001で検索

メインの流れ

ユーザ・プログラムのソースコードをファイルごとに見ていくことにします．各機能の結びつきやファイルとの対応関係は**表1**，**図3**を参照してください．

● main.cにはmain()関数/初期化/タイマ割り込み処理が記述されている

リスト1(pp.58-59)にmain.cの抜粋を示します．

表1　ST社から提供されているプログラムのうち特集に関係しそうなもの

メイン・ファイル	ヘッダ・ファイル	処理内容
main.c		メイン関数，初期化やタイマ割り込み
rc.c	rc.h	プロポと受信機のデータ取得
sensor_data.c		各センサからデータを読み取る
flight_control.c	flight_control.h	姿勢制御
motor.c		4つのモータへPWM信号を出力
quaternion.c		クォータニオンに関する各種計算(オイラー角への変換など)
ahrs.c		姿勢推定
	config_drone.h	センサ・データの軸の設定
debug.c		UARTによる飛行データの送信

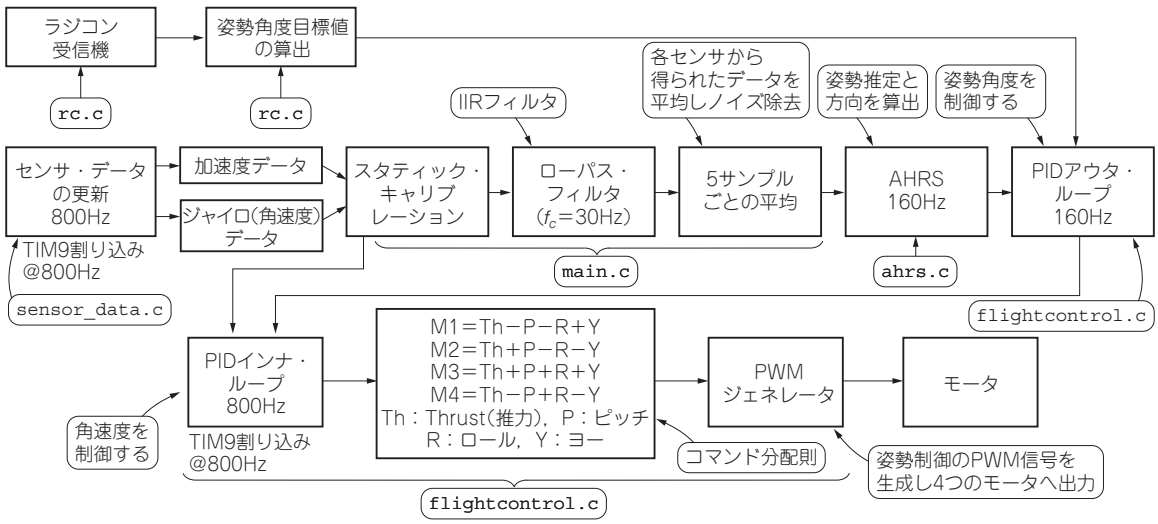

図3　プログラム・ファイルと対応する処理の関係

マイコン周辺機能の初期化のための独立した関数は462～724行目に入っていますが，ドローンの制御に特に重要なのはタイマTIM9で，665行目から始まるMX_TIM9_Init()関数で初期化が行われます．

TIM9のクロック・ソースは84MHzで，それをプリスケーラで1：(51+1)倍に下げ，(1999+1)カウントでタイムアウトとする設定となります．従って，タイムアウトの周期は約1.238ms(ミリ秒)，周波数にすると約808Hzです．これが制御の基本周期(周波数)となります．ただし，以降では端数を丸めて周波数を800Hz，周期を1.25msとして話を進めていきます．

● main()関数の流れ

▶161～318行目

マイコンの各種機能や制御関係の変数の初期化を行います．先ほどのMX_TIM9_Init()関数もこの中で呼び出します．

▶324行目

whileループによる繰り返し処理に入ります．

▶330～403行目

ここにあるif文は，先ほど解説した基本周期の5倍周期(6.25ms, 160Hz)ごとに処理が行われます(331行目の50Hzの記載は誤りと思われます)．ここでは，前半377行目までAHRSの計算が行われます．AHRSは，Attitude and Heading Reference Systemの略で，機体の姿勢と方向を算出する処理を意味します．処理の詳細は後ほど解説します．

▶336～365行目

IMUの加速度センサとジャイロ・センサから取得したデータを過去5サンプル分平均し，ノイズ除去を行います．

▶371行目

ahrs_fusion_ag()関数で機体姿勢の算出を行います．

▶374行目

QuaternionToEuler()関数で機体姿勢の表現方法をクォータニオンからオイラー角へ変換して，AHRSの周期処理が完了します．

▶380～399行目

機体の姿勢制御の準備を行います．

▶401～432行目

FlightControlPID_OuterLoop()関数で姿勢制御ルーチン(アウタ・ループ制御)を呼び出します．この関数そのものはflight_control.c内にあります．引き数のうち，euler_rc_filは操縦かんの操作量から算出した目標姿勢角度，euler_ahrsはAHRSで算出した現在の機体姿勢を格納しています．

この関数は，目標姿勢と現在姿勢から角速度目標値を算出するところまでの計算を行います．403行目までのif文を抜けると，途中で幾つかの処理を行った後，432行目PRINTF()関数で操縦かんの操作量やAHRSで算出した機体姿勢などをUARTから出力する処理を行います．これでwhileループおよびmain()関数の末尾となります．なお，PRINTF()関数はwhileループの直下にあるため，決められた一定周期ではなく，ループ最短周期で呼び出されることに気を付けてください

● HAL_TIM_PeriodElapsedCallback()関数の処理

731行目から始まるHAL_TIM_PeriodElapsed

Callback()関数は，TIM9のタイムアウトが生じるたびに呼び出される関数です．

▶733～771行目

FCU起動後2秒間で，IMUの加速度センサとジャイロ・センサのオフセット(0点ずれ)を取得します．この取得が完了する2秒以降から，773行目から始まるif文の処理が実行されるようになります．

▶779行目

ReadSensorRawData()関数では，IMU，磁気センサ，気圧センサからデータを読み取ります．この関数そのものは，sensor_data.cの中にあります．

ここで，センサ・データの軸の設定は，config_drone.h内の定数COORDINATE_SYSTEMに1～4の整数を与えて行います．初期設定3の場合，FCU基板上に描かれた白い矢印の方向がy軸，y軸に直角でFCU基板に平行かつUSBコネクタ側の方向がx軸，FCU基板に垂直かつ部品実装側の方向がz軸となります．もちろん，これらの軸はFCU，つまり機体胴体に固定されていて，機体とともに動きます．以後この直交3軸を機体軸，これら3軸により定まる座標系を「機体座標系」と呼びます．

▶781～813行目

操縦かんで特定のスティック操作が行われたときに(rc.cの199～202行目を参照)，再び前述のセンサのオフセット取得を行います．

▶815～820行目

センサ・データから取得したオフセット分を差し引きます．

▶823～825行目

加速度センサのデータをFIFOバッファに格納します．

▶828～849行目

ジャイロ・センサのデータをノイズ・フィルタ(IIRフィルタ)に通します．

▶852～854行目

フィルタ出力値をFIFOバッファに格納します．

▶857～870行目

ここにあるif文は，これらのFIFOバッファ(おのおの5個)がいっぱいになったときに，バッファ内のデータをAHRS計算用のFIFOバッファにコピーした上で，tim9_event_flagを1にします．これにより，先ほど解説したmain()関数における基本周期の5倍の周期で行う処理が起動されます．

▶873～875行目

ジャイロ・センサで取得した3軸角速度データの単位を[rad/s](ラジアン毎秒)に変換します．

▶877行目

オイラー角のうち，機体の方向(FCU基板上に描かれた矢印の水平面内の向き)であるeuler_ahrs.thzをz軸の角速度の数値積分により求めます．この処理はAHRSの一部です．

▶879～883行目

操縦かんのスロットル・スティックの操作量gTHRがしきい値MIN_THR未満である場合，構造体euler_rcとeuler_ahrsそれぞれの変数thzを0にクリアします．euler_rcは操縦かん操作から決まる姿勢の目標値が格納される構造体です．

▶886～889行目

プロポと受信機が正常に動作し，モータが動作可能状態となっているときFlight_ControlPID_innerLoop()関数を呼び出します．この関数そのものはflight_control.c内にあります．

この関数は，現在の機体角速度をmain()関数ループのFlightControlPID_OuterLoop()関数呼び出し時に求めた角速度目標値に追従させるため，4つのモータへ与えるPWM指令値を求めます．角速度制御，あるいはインナ・ループ制御と呼ばれ，ドローンの姿勢安定化に必要不可欠です．ST-DRONEでは，この処理を800Hzで行っているわけですが，ドローンのインナ・ループは一般にこのような高い周波数(短い周期)で処理を行います．

▶893行目

プロポと受信機が動作していないか，モータが動作可能状態となっていないときは，893行目でモータのPWM指令入力が0になります．

▶898行目

スロットル・スティックの操作量gTHRがしきい値MIN_THR未満である場合も，898行目でPWM指令入力が0になります．

▶901行目

最終的に決まった4つのモータのPWM指令入力は，901行目で呼び出されるset_motor_pwm()関数に渡されモータへ出力されます．この関数そのものはmotor.c内にあります．

プロポと受信機のプログラム

● プロポの操作と機体の動き

▶プロポ

プロポ(送信機)のスティック(操縦かん)は左右に1本ずつ2本あります．スティックは，上下左右に動き1本のスティックで上下と左右の2つの操作を行います．従って，操縦者は2本のスティックで4つの操作を同時に行うことになります．4つの操作は機体の運動に対応します．

▶エルロン

エルロン(AIL：Aileron，固定翼機の補助翼に由来)は，機体の横(左右，ロール)の姿勢変化に関連する

リスト1 main.cから抜粋したソースコード

```
/**
  ********************************************
  * File Name      : main.c
  * Description    : Main program body
  ********************************************
  *
  * COPYRIGHT(c) 2017 STMicroelectronics
      ⋮
int main(void)
{

  /* USER CODE BEGIN 1 */
  int16_t pid_interval, i;

  int mytimcnt = 0;
  acc_fil.AXIS_X = 0;
      ⋮
  /* Start timer */
  StartTimer(&tim);
  ch = 0;
  ch_flag = 0;

  /* USER CODE END 2 */



  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */

    if (tim9_event_flag == 1)
    {     // Timer9 event: frequency 50Hz
  tim9_event_flag = 0;

  count1++;

  acc_ahrs.AXIS_X = 0;
  acc_ahrs.AXIS_Y = 0;
  acc_ahrs.AXIS_Z = 0;
  gyro_ahrs.AXIS_X = 0;
  gyro_ahrs.AXIS_Y = 0;
  gyro_ahrs.AXIS_Z = 0;

  for(i=0;i<FIFO_Order;i++)
  {
    acc_ahrs.AXIS_X += acc_ahrs_FIFO[i].AXIS_X;
    acc_ahrs.AXIS_Y += acc_ahrs_FIFO[i].AXIS_Y;
    acc_ahrs.AXIS_Z += acc_ahrs_FIFO[i].AXIS_Z;
    gyro_ahrs.AXIS_X += gyro_ahrs_FIFO[i].AXIS_X;
    gyro_ahrs.AXIS_Y += gyro_ahrs_FIFO[i].AXIS_Y;
    gyro_ahrs.AXIS_Z += gyro_ahrs_FIFO[i].AXIS_Z;
  }

  acc_ahrs.AXIS_X *=FIFO_Order_Recip;
  acc_ahrs.AXIS_Y *=FIFO_Order_Recip;
  acc_ahrs.AXIS_Z *=FIFO_Order_Recip;
  gyro_ahrs.AXIS_X *=FIFO_Order_Recip;
  gyro_ahrs.AXIS_Y *=FIFO_Order_Recip;
  gyro_ahrs.AXIS_Z *=FIFO_Order_Recip;

  acc_fil_int.AXIS_X = acc_ahrs.AXIS_X;
  acc_fil_int.AXIS_Y = acc_ahrs.AXIS_Y;
  acc_fil_int.AXIS_Z = acc_ahrs.AXIS_Z;
  gyro_fil_int.AXIS_X = gyro_ahrs.AXIS_X;
  gyro_fil_int.AXIS_Y = gyro_ahrs.AXIS_Y;
  gyro_fil_int.AXIS_Z = gyro_ahrs.AXIS_Z;

  //PRINTF("%f %f %f %f\n", acc_ahrs.AXIS_X,
                  acc_ahrs.AXIS_Y, gyro_ahrs.AXIS_X,
                                   gyro_ahrs.AXIS_Y);

  // AHRS update, quaternion & true gyro data are
                                        stored in ahrs
  ahrs_fusion_ag(&acc_ahrs, &gyro_ahrs, &ahrs);

  // Calculate euler angle drone
  QuaternionToEuler(&ahrs.q, &euler_ahrs);

  // Get target euler angle from remote control
  GetTargetEulerAngle(&euler_rc, &euler_ahrs);


  if(gTHR<MIN_THR)
  {
    euler_ahrs_offset.thx = 0;
    euler_ahrs_offset.thy = 0;
  }

  Fly_origin.X_Degree = (int16_t)
                          (euler_ahrs.thx * 5730);
  Fly_origin.Y_Degree = (int16_t)
                          (euler_ahrs.thy * 5730);
  Fly_origin.Z_Degree = (int16_t)
                          (euler_ahrs.thz * 5730);


  if(gTHR<MIN_THR)
  {
    euler_rc.thz = 0;
    euler_ahrs.thz = 0;
  }

  euler_rc_fil.thx = euler_rc.thx;
  euler_rc_fil.thy = euler_rc.thy;
  euler_rc_fil.thz = euler_rc.thz;

  FlightControlPID_OuterLoop(&euler_rc_fil,
                        &euler_ahrs, &ahrs, &pid);

    }
      ⋮
    /* Added for debug on UART*/
    /* Remocon ELE, AIL, RUD, THR, Motor1_pwm,
                AHRS Euler angle x and y axis */
    PRINTF("%d\t%d\t%d\t%d\t%f\t%f\t%f\t%f\t%f\n",
    gELE, gAIL, gRUD, gTHR, motor_pwm.motor1_pwm,
    euler_ahrs.thx * 57.3, euler_ahrs.thy * 57.3,
       euler_rc.thx * 57.3, euler_rc.thy * 57.3);
      ⋮
  }
  /* USER CODE END 3 */

}
      ⋮
/* TIM9 init function */
void MX_TIM9_Init(void)
{

  TIM_ClockConfigTypeDef sClockSourceConfig;

  htim9.Instance = TIM9;
  htim9.Init.Prescaler = 51;
  htim9.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim9.Init.Period = 1999;
  htim9.Init.ClockDivision =
                          TIM_CLOCKDIVISION_DIV1;
  HAL_TIM_Base_Init(&htim9);

  sClockSourceConfig.ClockSource =
                        TIM_CLOCKSOURCE_INTERNAL;
  HAL_TIM_ConfigClockSource(&htim9,
                        &sClockSourceConfig);

}
      ⋮
/* USER CODE BEGIN 4 */
/*
 *  Handle Timer9 interrupt @ 800Hz
 *  Set the event flag and increase time index
 */
void HAL_TIM_PeriodElapsedCallback
                        (TIM_HandleTypeDef *htim)
{
  if(sensor_init_cali == 0)
  {
    sensor_init_cali_count++;

    if(sensor_init_cali_count > 800)
    {
  // Read sensor data and prepare for specific
                                  coodinate system
```

```
740   ReadSensorRawData(LSM6DSL_X_0_handle,
            LSM6DSL_G_0_handle, LIS2MDL_M_0_handle,
        LPS22HB_P_0_handle, &acc, &gyro, &mag, &pre);
741
742   acc_off_calc.AXIS_X += acc.AXIS_X;
743   acc_off_calc.AXIS_Y += acc.AXIS_Y;
744   acc_off_calc.AXIS_Z += acc.AXIS_Z;
745
746   gyro_off_calc.AXIS_X += gyro.AXIS_X;
747   gyro_off_calc.AXIS_Y += gyro.AXIS_Y;
748   gyro_off_calc.AXIS_Z += gyro.AXIS_Z;
749
750   if (sensor_init_cali_count >= 1600)
751   {
752     acc_offset.AXIS_X = acc_off_calc.AXIS_X *
                                          0.00125;
753     acc_offset.AXIS_Y = acc_off_calc.AXIS_Y *
                                          0.00125;
754     acc_offset.AXIS_Z = acc_off_calc.AXIS_Z *
                                          0.00125;
755
756     gyro_offset.AXIS_X = gyro_off_calc.AXIS_X *
                                          0.00125;
757     gyro_offset.AXIS_Y = gyro_off_calc.AXIS_Y *
                                          0.00125;
758     gyro_offset.AXIS_Z = gyro_off_calc.AXIS_Z *
                                          0.00125;
759
760     acc_off_calc.AXIS_X = 0;
761     acc_off_calc.AXIS_Y = 0;
762     acc_off_calc.AXIS_Z = 0;
763     gyro_off_calc.AXIS_X = 0;
764     gyro_off_calc.AXIS_Y = 0;
765     gyro_off_calc.AXIS_Z = 0;
766
767     sensor_init_cali_count = 0;
768     sensor_init_cali = 1;
769   }
770     }
771   }
772
773   if(sensor_init_cali == 1)
774   {
775     tim9_cnt++;
776     tim9_cnt2++;
777
778     // Read sensor data and prepare for specific
                                      coodinate system
779     ReadSensorRawData(LSM6DSL_X_0_handle,
            LSM6DSL_G_0_handle, LIS2MDL_M_0_handle,
        LPS22HB_P_0_handle, &acc, &gyro, &mag, &pre);
780
781     if (rc_cal_flag == 1)
782     {
          ⋮
813     }
814
815     acc.AXIS_X -= acc_offset.AXIS_X;
816     acc.AXIS_Y -= acc_offset.AXIS_Y;
817     acc.AXIS_Z -= (acc_offset.AXIS_Z - 1000);
818     gyro.AXIS_X -= gyro_offset.AXIS_X;
819     gyro.AXIS_Y -= gyro_offset.AXIS_Y;
820     gyro.AXIS_Z -= gyro_offset.AXIS_Z;
821
822     // Save filtered data to acc_FIFO
823     acc_FIFO[tim9_cnt2-1].AXIS_X = acc.AXIS_X;
824     acc_FIFO[tim9_cnt2-1].AXIS_Y = acc.AXIS_Y;
825     acc_FIFO[tim9_cnt2-1].AXIS_Z = acc.AXIS_Z;
826
827     // IIR Filtering on gyro
828     gyro_fil.AXIS_X = gyro_fil_coeff.b0*gyro.
    AXIS_X + gyro_fil_coeff.b1*gyro_x_pre[0].AXIS_X
           + gyro_fil_coeff.b2*gyro_x_pre[1].AXIS_X
829                  + gyro_fil_coeff.a1*
                              gyro_y_pre[0].AXIS_X
         + gyro_fil_coeff.a2*gyro_y_pre[1].AXIS_X;
830     gyro_fil.AXIS_Y = gyro_fil_coeff.b0*gyro.
    AXIS_Y + gyro_fil_coeff.b1*gyro_x_pre[0].AXIS_Y
           + gyro_fil_coeff.b2*gyro_x_pre[1].AXIS_Y
831                  + gyro_fil_coeff.a1*
                              gyro_y_pre[0].AXIS_Y
         + gyro_fil_coeff.a2*gyro_y_pre[1].AXIS_Y;
832     gyro_fil.AXIS_Z = gyro_fil_coeff.b0*gyro.
    AXIS_Z + gyro_fil_coeff.b1*gyro_x_pre[0].AXIS_Z
           + gyro_fil_coeff.b2*gyro_x_pre[1].AXIS_Z
833                  + gyro_fil_coeff.a1*
                              gyro_y_pre[0].AXIS_Z
         + gyro_fil_coeff.a2*gyro_y_pre[1].AXIS_Z;
834     // Shift IIR filter state
835     for(int i=1;i>0;i--)
836     {
837   gyro_x_pre[i].AXIS_X = gyro_x_pre[i-1].AXIS_X;
838   gyro_x_pre[i].AXIS_Y = gyro_x_pre[i-1].AXIS_Y;
839   gyro_x_pre[i].AXIS_Z = gyro_x_pre[i-1].AXIS_Z;
840   gyro_y_pre[i].AXIS_X = gyro_y_pre[i-1].AXIS_X;
841   gyro_y_pre[i].AXIS_Y = gyro_y_pre[i-1].AXIS_Y;
842   gyro_y_pre[i].AXIS_Z = gyro_y_pre[i-1].AXIS_Z;
843     }
844     gyro_x_pre[0].AXIS_X = gyro.AXIS_X;
845     gyro_x_pre[0].AXIS_Y = gyro.AXIS_Y;
846     gyro_x_pre[0].AXIS_Z = gyro.AXIS_Z;
847     gyro_y_pre[0].AXIS_X = gyro_fil.AXIS_X;
848     gyro_y_pre[0].AXIS_Y = gyro_fil.AXIS_Y;
849     gyro_y_pre[0].AXIS_Z = gyro_fil.AXIS_Z;
850
851     //  Save filtered data to gyro_FIFO
852     gyro_FIFO[tim9_cnt2-1].AXIS_X =
                                    gyro_fil.AXIS_X;
853     gyro_FIFO[tim9_cnt2-1].AXIS_Y =
                                    gyro_fil.AXIS_Y;
854     gyro_FIFO[tim9_cnt2-1].AXIS_Z =
                                    gyro_fil.AXIS_Z;
855
856
857     if(tim9_cnt2 == FIFO_Order)
858     {
859   tim9_cnt2 = 0;
860   tim9_event_flag = 1;
861   for(int i=0;i<FIFO_Order;i++)
862   {
863     acc_ahrs_FIFO[i].AXIS_X = acc_FIFO[i].AXIS_X;
864     acc_ahrs_FIFO[i].AXIS_Y = acc_FIFO[i].AXIS_Y;
865     acc_ahrs_FIFO[i].AXIS_Z = acc_FIFO[i].AXIS_Z;
866     gyro_ahrs_FIFO[i].AXIS_X = gyro_FIFO[i].AXIS_X;
867     gyro_ahrs_FIFO[i].AXIS_Y = gyro_FIFO[i].AXIS_Y;
868     gyro_ahrs_FIFO[i].AXIS_Z = gyro_FIFO[i].AXIS_Z;
869   }
870     }
871
872
873   gyro_rad.gx = gyro_fil.AXIS_X*COE_MDPS_TO_RADPS;
874   gyro_rad.gy = gyro_fil.AXIS_Y*COE_MDPS_TO_RADPS;
875   gyro_rad.gz = gyro_fil.AXIS_Z*COE_MDPS_TO_RADPS;
876
877   euler_ahrs.thz += gyro_rad.gz*PID_SAMPLING_TIME;
878
879   if(gTHR<MIN_THR)
880   {
881     euler_rc.thz = 0;
882     euler_ahrs.thz = 0;
883   }
884
885
886   if (rc_connection_flag && rc_enable_motor)
887   {   // Do PID Control
888     FlightControlPID_innerLoop(&euler_rc_fil,
              &gyro_rad, &ahrs, &pid, &motor_pwm);
889   }
890   else
891   {
892     // set motor output zero
893     set_motor_pwm_zero(&motor_pwm);
894   }
895
896   if(gTHR<MIN_THR)
897   {
898     set_motor_pwm_zero(&motor_pwm);
899   }
900
901   set_motor_pwm(&motor_pwm);
            /* To comment ifwant to debug remocon
             calibrationswitching off the motors */
902   }
903 }
```

センサ・データの読み込み

800サンプル（約1s間）にわたる各軸の加速度と角速度のデータを平均し，オフセット（0点ずれ）を算出する

センサ初期化完了後に実行

加速度データをFIFOバッファに格納

センサ・データの0点ずれを補正

角速度をノイズ・フィルタ（IIRフィルタ）に通す

角速度データをFIFOバッファに格納

AHRS計算（main()関数のループから呼び出し）に渡すためのデータを作成

機体方向のオイラー角を算出（Z軸角速度を積分）

角速度の単位を［rad/s］へ変換

ラジコン・プロポ受信器が正常動作し，かつモータが動作可能状態であれば，インナ・ループ角速度制御の計算を実行

モータへPWM信号を出力

操作量の呼び名です．

▶エレベータ

エレベータ（ELE：Elevator，固定翼機の昇降舵に由来）は，機体の縦（機首上げ下げ，ピッチ）の姿勢変化に関連する操作量の呼び名です．

▶ラダー

ラダー（RUD：Rudder，固定翼機の方向舵に由来）は，機体の方向（機首の向き，ヨー）の変化に関連する操作量の呼び名です．

▶スロットル

スロットル（THR：Throttle）は，モータ出力に関連する操作量の呼び名です．

● プロポの操作方法でメジャーな「モード1」

エルロン，エレベータ，ラダー，スロットルの計4つの操作量がスティックの4つの動きにどのように結び付くかは，プロポの機種によります．日本国内で多く用いられる「モード1」と呼ばれるプロポの場合は，左のスティックの上下がエレベータ，左右がラダー，右のスティックの上下がスロットル，左右がエルロンに割り当てられています（**図1**，**図4**）．

● スティック操作量はPWMパルス信号の時間幅に対応

スティックの操作量は，受信機からPWMパルス幅にエンコードされて各チャネルに出力されます．つまり，パルスのONの時間幅がスティックの操作量に対応します．筆者が今回使用するプロポ/受信機の場合，特別な設定変更がなければ，チャネル番号と操縦操作の対応関係は，1がエルロン，2がエレベータ，3がスロットル，4がラダーで，PWMパルス幅はいずれもスティックが中央のときに約1520μs，スティックを上下あるいは左右に最大に振ったときにそこから約±420μsの変化となるようです．

スティックの操作方向とパルス幅が長く/短くなる方向の対応関係もまた，使用するプロポ/受信機により異なります．筆者が今回使用するプロポ/受信機の場合，既に解説したサーボ・リバースの設定を行うと，スティックを下または右へ操作するとパルス幅が短くなり（−），スティックを上または左へ操作するとパルス幅が長くなります（+）．

図4　モード1ではスティック操作があらかじめ割り当てられている

● ヘッダ・ファイルrc.hに記述されている定数

プロポと受信機のデータ取得に関する処理は，rc.h（**リスト2**）とrc.c（**リスト3**）に記述されています．まずは，rc.h（**リスト2**）で定義される定数について解説します．

▶13～41行目

使用するプロポ/受信機に応じて設定する定数です．ラジコン受信機から出力されたパルスは，インプット・キャプチャ機能を用いてマイコンに取り込まれます．取り込まれたパルス幅は，0.25μsを1LSBとする整数値としてレジスタに入ります．AIL，ELE，THR，RUDの各チャネルの最大値，中央値（THRを除く），最小値および振幅（片振幅）を設定します．今回使用するプロポに合わせて筆者が設定した値は**表2**に示す通りです．

▶47～48行目

PITCH_MAX_DEGとROLL_MAX_DEGは，それぞれエレベータ，エルロンのスティック操作量に対応する姿勢角度の最大値を設定する定数で，デフォルト値は20°です．よりアグレッシブに機体を動かしたい場合はこの値を大きくします．

▶54行目

YAW_MAX_DEGは，ラダー・スティック操作時の単位時間の機体方向の変化量で，この値を大きくするほど機体の回転が速くなります．デフォルト値は

表2　rc.h内に記述されている定数の設定

定数名	値
AIL_LEFT，ELE_TOP，THR_TOP，RUD_LEFT	7760
AIL_MIDDLE，ELE_MIDDLE，RUD_MIDDLE	6080
AIL_RIGHT，ELE_BOTTOM，THR_BOTTOM，RUD_RIGHT	4400
RC_FULLSCALE	1680

リスト2　プロポと受信機のデータ取得に関する処理rc.hから抜粋したソースコード

```
#ifndef _RC_H_
#define _RC_H_
   :
/* Definition for R/C Timing (1 LSB = 250us) */

// Definition for AIL(Roll) Channel
#define AIL_LEFT    8088
   :
#define AIL_MIDDLE  5862
                  /* Calibrated Devo7E remocon */
   :
#define AIL_RIGHT   4126
// Definition for ELE(Pitch) Channel
#define ELE_BOTTOM  4472
   :
#define ELE_MIDDLE  6151
                  /* Calibrated Devo7E remocon */
   :
#define ELE_TOP     8088
// Definition for THR Channel
   :
#define THR_BOTTOM  4450
                  /* Calibrated Devo7E remocon */
   :
#define THR_TOP     8080
```

エルロン（AIL）のスティック位置とインプット・キャプチャされるレジスタ値［LSB］の対応関係の定義

エレベータ（ELE）に関する同上の定義

スロットル（ELE）に関する同上の定義

```
// Definition for RUD(Yaw) Channel
#define RUD_LEFT    8088
   :
#define RUD_MIDDLE  6397
                  /* Calibrated Devo7E remocon */
   :
#define RUD_RIGHT   4216

#define RC_FULLSCALE        1800
#define RC_CAL_THRESHOLD    1200
   :
// Maximum roll/pitch 35deg
#define PITCH_MAX_DEG   20
#define ROLL_MAX_DEG    20
   :
#define YAW_MAX_DEG  (120.0*
                        SENSOR_SAMPLING_TIME)
#define YAW_MIN_RAD  0.0872

#define EULER_Z_TH   600
   :
#endif /* _RC_H_ */
```

ラダー（RUD）に関する同上の定義

振幅［LSB］

エレベータ・エルロンの最大操作量に対するピッチ／ロールの姿勢角度目標値［°］

ラダー・スティック（最大）操作量に対するヨー角度変化量

リスト3　プロポと受信機のデータ取得に関する処理rc.cから抜粋したソースコード

```
/* Introduction for remote control module:
   This module provide interface to work with
                    remote control receiver signals
      :
/* Global R/C data */
int16_t gAIL, gELE, gTHR, gRUD;
      :
void HAL_TIM_IC_CaptureCallback
                              (TIM_HandleTypeDef *htim)
{
      :
}
      :
/* Update global variables of R/C data */
void update_rc_data(int32_t idx)
{
  switch (idx)
  {
 case 0: gAIL = rc_t[0] - ail_center; break;
 case 1: gELE = rc_t[1] - ele_center; break;
 case 2: gTHR = (rc_t[2] > THR_BOTTOM) ?
                (rc_t[2] - THR_BOTTOM) : 0; break;
 case 3: gRUD = rc_t[3] - rud_center; break;
 default: break;
  }

  if ( (gTHR == 0) && (gELE < - RC_CAL_THRESHOLD)
                   && (gAIL > RC_CAL_THRESHOLD) &&
                      (gRUD < - RC_CAL_THRESHOLD))
  {
 rc_cal_flag = 1;
  }

  if ( (gTHR == 0) && (gELE < - RC_CAL_THRESHOLD)
                   && (gAIL < - RC_CAL_THRESHOLD)
                    && (gRUD > RC_CAL_THRESHOLD))
  {
 rc_enable_motor = 1;
 fly_ready = 1;
  }
}

/*
 * Convert RC received gAIL, gELE, gRUD
 */
```

エレベータ・スティック操作量からピッチ姿勢角度目標値を作る

マイコンのインプット・キャプチャ機能を用いてラジコン受信器から受けたPWM信号のパルス幅を計測する（途中で関数update_rc_data()を呼び出し）

gAIL，gELE，gTHR，gRUDに操作量［LSB］を代入

特定のスティック操作（ELEとTHRが下端，AILが右端，RUDが左端）があったとき，モータを動作可能状態とする

特定のスティック操作（ELEとTHRが下端，AILが左端，RUDが右端）があったとき，センサ・データのオフセットを再取得する処理を起動する

```
void GetTargetEulerAngle(EulerAngleTypeDef
      *euler_rc, EulerAngleTypeDef *euler_ahrs)
{
 t1 = gELE;
 if (t1 > RC_FULLSCALE)
  t1 = RC_FULLSCALE;
 else if (t1 < -RC_FULLSCALE)
  t1 = - RC_FULLSCALE;
 euler_rc->thx = -t1 * max_pitch_rad /
                                  RC_FULLSCALE;

 t1 = gAIL;
 if (t1 > RC_FULLSCALE)
  t1 = RC_FULLSCALE;
 else if (t1 < -RC_FULLSCALE)
  t1 = - RC_FULLSCALE;
 euler_rc->thy = -t1 * max_roll_rad /
                                  RC_FULLSCALE;

 t1 = gRUD;
 if (t1 > RC_FULLSCALE)
  t1 = RC_FULLSCALE;
 else if (t1 < -RC_FULLSCALE)
  t1 = - RC_FULLSCALE;

 if(rc_z_control_flag == 1)
 {
   if(t1 > EULER_Z_TH)
   {
  euler_rc->thz = euler_rc->thz + max_yaw_rad;
   }
   else if(t1 < -EULER_Z_TH)
   {
  euler_rc->thz = euler_rc->thz - max_yaw_rad;
   }
 }
 else
 {
   if(t1 > -EULER_Z_TH&&t1 < EULER_Z_TH)
   {
     rc_z_control_flag = 1;
   }
 }
}
```

エレベータ・スティック操作量からピッチ姿勢角度目標値を作る

エルロン・スティック操作量からロール姿勢角度目標値を作る

ラダー・スティック操作量から方向目標値を作る

120°/sです．機体方向の回転は，ラダー・スティックの中央からの操作量が57行目のEULER_Z_THで設定した値を超えた場合のみ発生し，回転速度は一定です．

● メイン・ファイルrc.cに記述されている処理内容

次に，メインの処理を行っているrc.c（**リスト3**）を見ていきます．

▶73行目

ここで定義される変数gAIL，gELE，gTHR，gRUDはスティック操作量を格納する変数です．gAIL，gELE，gRUDは，スティック中央を0，gTHRはスティックが一番下のときを0とし，いずれも，PWMパルス幅0.25μs当たり1LSBとなる値が入ります．

▶113～171行目

パルス幅を計測するインプット・キャプチャは，113～171行目の関数HAL_TIM_IC_CautureCallback()関数で行っていて，その中で，188～209行目のupdate_rc_data()関数を呼び出し，先ほど解説した4つの変数（gAILなど）に値を書き込んでいます．また，update_rc_data()関数において，特定のスティック操作を検出する処理を行っています．

▶199～202行目

スロットルとエレベータがともに一番下，エルロンが一番左，ラダーが一番右となるようスティックを操作したとき，センサ・データのオフセットを再取得する処理を実行するようフラグrc_cal_flagを1にします．

▶204～208行目

スロットルとエレベータがともに一番下，エルロンが一番右，ラダーが一番左となるようスティックを操作したとき，モータを動作可能状態とするよう，フラグrc_enable_motorを1にします．このスティック操作を行うまではモータは回転しません．FCUの電源を投入して，すぐにモータが回り出して危険な状態になることを防ぐ安全装置の役割を果たしています．

▶214～254行目

ここにあるGetTargetEulerAngle()関数は，スティック操作量から姿勢制御で使う姿勢の目標値を作ります．ここで作られた目標値はflight_control.c内の姿勢制御を行うFlightControlPID_OuterLoop()関数で利用されます．

姿勢推定

● センサ値から姿勢を推定する基本メカニズム

AHRSは，角速度を計測するジャイロ・センサと重力加速度を含む加速度を計測する加速度センサを用いて，縦と横の傾きや方向を推定する計算を行うものです．

基本的な計算方法は，ジャイロ・センサで得られる角速度を積分して角度を計算するというものですが，ジャイロ・センサの角速度計測値にはさまざまな誤差があり，積分すると一般には誤差によって時間とともに積分値が発散していきます．

加速度センサの情報から重力方向は分かるので，それを基準にして，ジャイロ・センサの角速度の積分値を発散しないように補正します．ただし，加速度センサは重力加速度の他に機体の運動によって生じる加速度も含むので，常に正確な重力方向が測れるわけではありません．

● 複数のセンサ・データから新たな推定値を得る「センサ・フュージョン」

上述の通り，ジャイロ・センサと加速度センサのいずれも，姿勢計測において無視できない誤差を持つので，互いの短所を補って姿勢を「推定」することになります．このように，2つ以上のセンサの情報を融合させて，単一のセンサでは取得が困難な情報を構築し推定値を得ることをセンサ・フュージョンと呼びます．

AHRSの中には，磁気センサを用いて地磁気を測り，磁方位を一緒に推定するものもありますが，ST-DRONEのFCUには磁気センサは付いているものの，ソースコード中では磁気センサは用いていません．加速度センサで測る重力方向は方位に関する情報を含まないので，ST-DRONEのAHRSで得る方位（機体の方向）はジャイロ・センサで得られる角速度の数値積分のみで推定しています．

● 計算方法「Complementary filter」

AHRSの姿勢推定には幾つかの計算方法があります．ST-DRONEのFCUのソースコードでは，その中でも比較的シンプルなComplementary filterと呼ばれる手法を用いています．姿勢の情報のうち，低周波数成分を加速度センサで，高周波数成分をジャイロ・センサでそれぞれ構築するというものです．

具体的な計算方法としては，加速度センサにより得られる加速度のみから計算した（誤差を含んだ）姿勢を目標値として，ジャイロ・センサの角速度の積分で得られる姿勢がこの目標値に追従するようにフィード

バック制御します．本手法の利点は，計算が簡単で処理負荷が小さいことです．ブロック線図で示すと**図5**のようになっています．

● 数学的には4元数（クォータニオン）で表現する

姿勢は，4元数（以下，クォータニオン）と呼ばれる方法で数学的に表現します．姿勢は3つの角度で表すことができますが，あえて変数を4つに増やすことで，角速度の積分によって姿勢を得る際に生じる特異な状況を回避することができ，AHRSではよく使われます．

▶定義

ドローンが地上に着地している状態での機体座標系を基準座標系とし，それを1本の単位ベクトル$[l\ m\ n]^{\mathrm{T}}$まわりに角度$\bar{\theta}$だけ回転させて飛行中の機体座標系と各軸の向きが一致したとすると，クォータニオン$\boldsymbol{q}$は次式で表されます．

$$\boldsymbol{q}=\begin{bmatrix}q_0\\q_1\\q_2\\q_3\end{bmatrix}=\begin{bmatrix}\cos\dfrac{\bar{\theta}}{2}\\ l\sin\dfrac{\bar{\theta}}{2}\\ m\sin\dfrac{\bar{\theta}}{2}\\ n\sin\dfrac{\bar{\theta}}{2}\end{bmatrix}\quad\cdots\cdots(1)$$

また，ノルムは$||\boldsymbol{q}||=1$となります．

● 姿勢推定の処理を行っている ahrs_fusion_ag() 関数

図5とahrs.c（**リスト4**）の25行目から始まるahrs_fusion_ag()関数の内容を併せて見ていきます．

▶加速度センサ

加速度センサは，機体軸3軸それぞれの加速度を測ることができるセンサで，計測値には重力加速度も含まれます．**図5**ではベクトル$\vec{g}$がそれに当たり，ソースコード中ではaxf，ayf，azfに対応します．また，加速度ベクトルは長さ1に正規化して用います．

▶ジャイロ・センサ

ジャイロ・センサは，機体軸3軸それぞれを軸とする回転の角速度を測ります．**図5**では$\boldsymbol{\omega}$というベクトルがそれに当たり，ソースコード中ではgxf，gyf，gzfに対応します．単位は[rad/s]です．

▶73～75行目

クォータニオンを用いると，長さ1に正規化した重力加速度ベクトルは，ソースコードの73～75行目のvx，vy，vzに対応し，**図5**で言えば一番下の$\vec{g}'=\cdots$のブロック内の式で書くことができます．

▶77～79行目

次に，$\vec{g}'$に対する$\vec{g}$の誤差ベクトル$\boldsymbol{e}$を求めます．ソースコード中では，77～79行目のex，ey，ezです．**図5**では，単純な引き算をしているように見えますが，実際にはそうではなく，2つのベクトルの外積$\vec{g}\times\vec{g}'$をとっています．これは，$\vec{g}'$を目標値である$\vec{g}$に追従（一致）させるための回転軸を得ることに相当します．

▶82～84行目

このようにして得た$\boldsymbol{e}$を用いて，PI（比例/積分）制御を行います．ソースコードの82～84行目は$\boldsymbol{e}$に積分ゲインk_i（AHRS_KI）を掛けて数値積分しています．

▶87～89行目

その数値積分値と$\boldsymbol{e}$に比例ゲインK_p（ahrs_kp）を掛けた値を足して，$\boldsymbol{\omega}$に対して補正項として加算

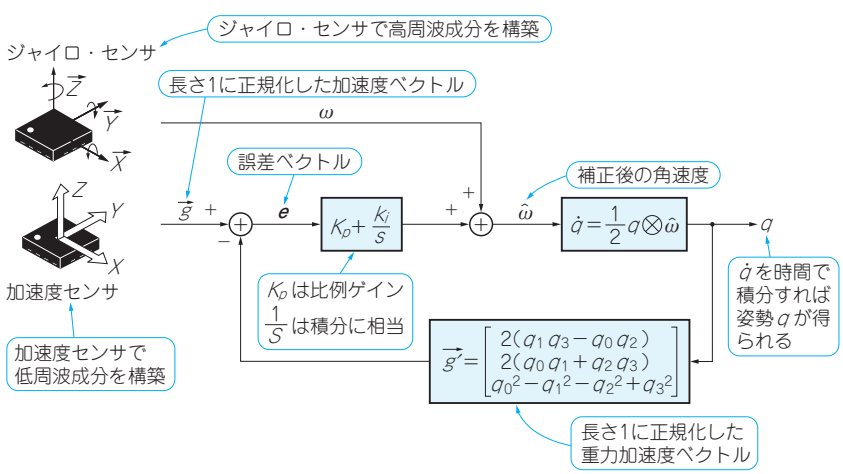

図5　姿勢推定用Complementary filterの計算方法
低周波数成分を加速度センサで，高周波数成分をジャイロ（角速度）センサで構築する

リスト4　姿勢推定の処理を行っているahrs.cから抜粋したソースコード

```
1
2   //#include "matrix.h"
3   #include "ahrs.h"
4   //#include "magnet_cal.h"
5   #include "basic_math.h"
6   #include "flight_control.h"
7
8
9   float offset[3];
10  float cor[3][3];
11
12  float q0 = 1, q1 = 0, q2 = 0, q3 = 0;
13  float gx_off, gy_off, gz_off;
14  float mx_mag, my_mag, mz_mag;
15  float wbx = 0, wby = 0, wbz = 0;
16  float by = 1, bz = 0;
17  float exInt = 0, eyInt = 0, ezInt = 0;
18
19  int count;
20  int ahrs_init_flag = 0;
21  int acc_over = 0;
22  extern int16_t gTHR;
23  float ahrs_kp;
24
25  void ahrs_fusion_ag(AxesRaw_TypeDef_Float *acc,
                        AxesRaw_TypeDef_Float *gyro,
                          AHRS_State_TypeDef *ahrs)
26  {
27    float axf,ayf,azf,gxf,gyf,gzf;
28    float norm;
29    float vx, vy, vz;
30    float ex, ey, ez;
31    float q0q0, q0q1, q0q2, q0q3, q1q1, q1q2,
                              q1q3, q2q2, q2q3, q3q3;
32    float halfT;
33
34
35    if(gTHR<MIN_THR)
36    {
37      ahrs_kp = AHRS_KP_BIG;
38    }
39    else
40    {
41      ahrs_kp = AHRS_KP_NORM;
42    }
43
44    axf = acc->AXIS_X;
45    ayf = acc->AXIS_Y;
46    azf = acc->AXIS_Z;
47
48    // mdps convert to rad/s
49    gxf = gyro->AXIS_X * COE_MDPS_TO_RADPS;
50    gyf = gyro->AXIS_Y * COE_MDPS_TO_RADPS;
51    gzf = gyro->AXIS_Z * COE_MDPS_TO_RADPS;
52
53    // auxiliary variables to reduce number of
                                    repeated operations
54    q0q0 = q0*q0;
55    q0q1 = q0*q1;
56    q0q2 = q0*q2;
57    q0q3 = q0*q3;
58    q1q1 = q1*q1;
59    q1q2 = q1*q2;
60    q1q3 = q1*q3;
61    q2q2 = q2*q2;
62    q2q3 = q2*q3;
63    q3q3 = q3*q3;
64
65    // normalise the accelerometer measurement
66    norm = invSqrt(axf*axf+ayf*ayf+azf*azf);
67
68    axf = axf * norm;
69    ayf = ayf * norm;
70    azf = azf * norm;
71
72    // estimated direction of gravity and flux
                                          (v and w)
73    vx = 2*(q1q3 - q0q2);
74    vy = 2*(q0q1 + q2q3);
75    vz = q0q0 - q1q1 - q2q2 + q3q3;
76
77    ex = (ayf*vz - azf*vy);
78    ey = (azf*vx - axf*vz);
79    ez = (axf*vy - ayf*vx);
80
81    // integral error scaled integral gain
82    exInt = exInt + ex*AHRS_KI*
                                  SENSOR_SAMPLING_TIME;
83    eyInt = eyInt + ey*AHRS_KI*
                                  SENSOR_SAMPLING_TIME;
84    ezInt = ezInt + ez*AHRS_KI*
                                  SENSOR_SAMPLING_TIME;
85
86    // adjusted gyroscope measurements
87    gxf = gxf + ahrs_kp*ex + exInt;
88    gyf = gyf + ahrs_kp*ey + eyInt;
89    gzf = gzf + ahrs_kp*ez + ezInt;
90
91    // integrate quaternion rate and normalise
92    halfT = 0.5*SENSOR_SAMPLING_TIME;
93    q0 = q0 + (-q1*gxf - q2*gyf - q3*gzf)*halfT;
94    q1 = q1 + (q0*gxf + q2*gzf - q3*gyf)*halfT;
95    q2 = q2 + (q0*gyf - q1*gzf + q3*gxf)*halfT;
96    q3 = q3 + (q0*gzf + q1*gyf - q2*gxf)*halfT;
97
98    // normalise quaternion
99    norm = invSqrt(q0 * q0 + q1 * q1 + q2 * q2
                                     + q3 * q3);
100   q0 *= norm;
101   q1 *= norm;
102   q2 *= norm;
103   q3 *= norm;
104
105   ahrs->q.q0 = q0;
106   ahrs->q.q1 = q1;
107   ahrs->q.q2 = q2;
108   ahrs->q.q3 = q3;
109
110 }
```

機体姿勢を表すクォータニオン

3軸の加速度

(49行目～)
3軸の加速度

姿勢の推定誤差，すなわち，[vx，vy，vz]を[axf，ayf，azf]へ近づけるための回転軸ベクトルを求める

後の計算で繰り返し使う量をあらかじめ計算しておく

加速度ベクトルの長さを1に正規化

姿勢クォータニオンの前回値から，重力加速度ベクトルを推定する

姿勢クォータニオンを更新

(99行目～)
更新したクォータニオンについて，ノルムが1になるように修正する

[vx，vy，vz]を[axf，ayf，azf]へ近づけるため，PI制御の要領で角速度[gxf，gyf，gzf]を補正する

し，補正後の角速度$\hat{\omega}$を得ます．角速度$\hat{\omega}$とクォータニオンの時間微分（単位時間当たりの変化量）$\dot{q}$の関係は，ブロック線図の右側にある$\dot{q}=\cdots$のブロック内の式で表されます．記号の上のドットは時間微分（d/dt）を表します．$\dot{q}=\frac{dq}{dt}$です．⊗という見慣れない演算子がありますが，行列の積の形で書き直すと次式となります．

$$\dot{q}=\frac{1}{2}q\otimes\hat{\omega}=\frac{1}{2}\begin{bmatrix}q_0\\q_1\\q_2\\q_3\end{bmatrix}\otimes\begin{bmatrix}\hat{\omega}_x\\\hat{\omega}_y\\\hat{\omega}_z\end{bmatrix}=\frac{1}{2}\begin{bmatrix}-q_1&-q_2&-q_3\\q_0&-q_3&q_2\\q_3&q_0&-q_1\\-q_2&q_1&q_0\end{bmatrix}\begin{bmatrix}\hat{\omega}_x\\\hat{\omega}_y\\\hat{\omega}_z\end{bmatrix}\quad\cdots\cdots(2)$$

また，$\hat{\omega}_x$，$\hat{\omega}_y$，$\hat{\omega}_z$はそれぞれ，ソースコード中の87～89行目の左辺のgxf，gyf，gzfです．

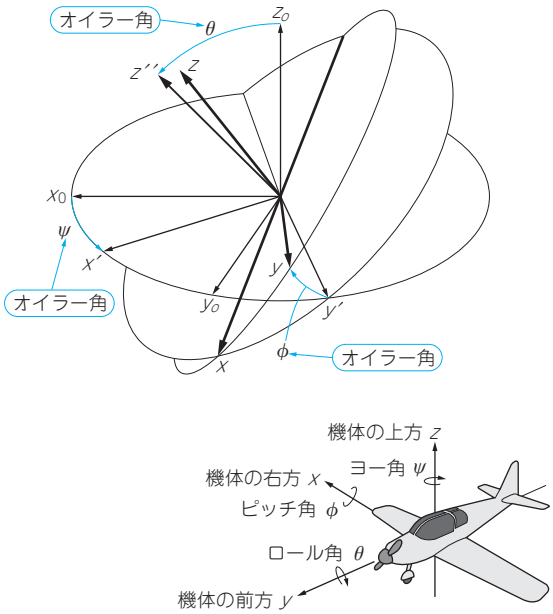

図6　オイラー角の定義を機体の座標系に対応させる

▶92～96行目

後は，$\dot{q}$を時間で数値積分すれば姿勢qが得られます．このあたりの計算はソースコード中の92～96行目に書かれています．数値積分には，さまざまな手法がありますが，ST-DRONEのFCUのソースコードでは，最も単純なオイラー法が用いられています．1処理周期ごとに時間微分とサンプリング時間を掛けた値を加算していくという方法です．

▶99～103行目

$\dot{q}$を数値積分してqを得た場合，$||q|| = 1$とならないことがあります．それを修正するのが99～103行目の処理です．以上のプロセスにより，機体の姿勢推定の1回の処理が完了します．

直感的に姿勢をイメージするための四元数－オイラー角変換

● クォータニオンは直感的でない

クォータニオンには，積分計算時の特異点を回避できるという利点があるものの，4つの成分の値を見ても機体の姿勢をイメージしにくいです．そこで一般には，より直感的に機体の姿勢をイメージできる，オイラー角（Euler angles）と呼ばれる3つの角度で姿勢を表します．

まず，地上に固定された基準座標系を(x_o, y_o, z_o)とします．これをz_o軸まわりに角度ψ回転させた座標系を(x', y', z_o)，y'軸まわりに角度θ回転させた座標系を(x, y', z'')，x軸まわりに角度ϕ回転させた座標系を(x, y, z)とし，これが機体座標系と一致するとき，

リスト5　クォータニオンからオイラー角へ変換する処理が記述されたquaternion.cから抜粋したソースコード

```
1   #include "quaternion.h"
2   #include <math.h>
      :
                          クォータニオンをピッチ・ロールのオイラー角へ変換する関数
75  /*
76   * Convert Quaternion to Euler Angle
77   */
78  void QuaternionToEuler(QuaternionTypeDef *qr,
                           EulerAngleTypeDef *ea)
79  {
80   float q0q0, q1q1, q2q2, q3q3;
81   float dq0, dq1, dq2;
82   float dq1q3, dq0q2, dq1q2;
83   float dq0q1, dq2q3, dq0q3;
84
85   q0q0 = qr->q0*qr->q0;
86   q1q1 = qr->q1*qr->q1;
87   q2q2 = qr->q2*qr->q2;
88   q3q3 = qr->q3*qr->q3;
89   dq0 = 2*qr->q0;
90   dq1 = 2*qr->q1;
91   dq2 = 2*qr->q2;
92   dq1q2 = dq1 * qr->q2;
93   dq1q3 = dq1 * qr->q3;
94   dq0q2 = dq0 * qr->q2;
95   dq0q3 = dq0 * qr->q3;
96   dq0q1 = dq0 * qr->q1;
97   dq2q3 = dq2 * qr->q3;
                                   thx：ピッチのオイラー角
98
99   ea->thx = atan2(dq0q1+dq2q3,
                     q0q0+q3q3-q1q1-q2q2);
100  ea->thy = asin(dq0q2-dq1q3);
                                   thy：ロールのオイラー角
      :
116 }
```

3つの回転角度ϕ，θ，ψを「オイラー角」と呼びます．

● 機体の座標系とオイラー角を対応付ける

ST-DRONEにFCUを部品実装面を上にして取り付けた場合，y軸が機体の前方，x軸が機体の右方，z軸が機体の上方となります．このとき，x軸まわりに姿勢を変化させる運動をピッチング（ピッチ），y軸まわりに姿勢を変化させる運動をローリング（ロール），z軸まわりに姿勢（方向）を変化させる運動をヨーイング（ヨー）と呼び，オイラー角と対応させて，ϕをピッチ角，θをロール角，ψをヨー角と呼びます（図6）．

ソースコード中ではϕがthx，θがthy，ψがthzという変数名で表されています．クォータニオンとオイラー角ϕ，θの間には次の関係式が成り立ちます．

$$\phi = \tan^{-1}\frac{2(q_0q_1 + q_2q_3)}{q_0^2 - q_1^2 - q_2^2 + q_3^2} \quad \cdots\cdots(3)$$

$$\theta = \sin^{-1}\{2(q_0q_2 - q_1q_3)\} \quad \cdots\cdots(4)$$

● クォータニオンからオイラー角へ変換するQuaternionToEuler()関数

上式は，quaternion.cの78～116行目のQuaternionToEuler()関数に実装されていて，ahrs_fusion_ag()関数の1回の処理が終わるごとにmain.cの374行目で呼び出され，都度，クォータニ

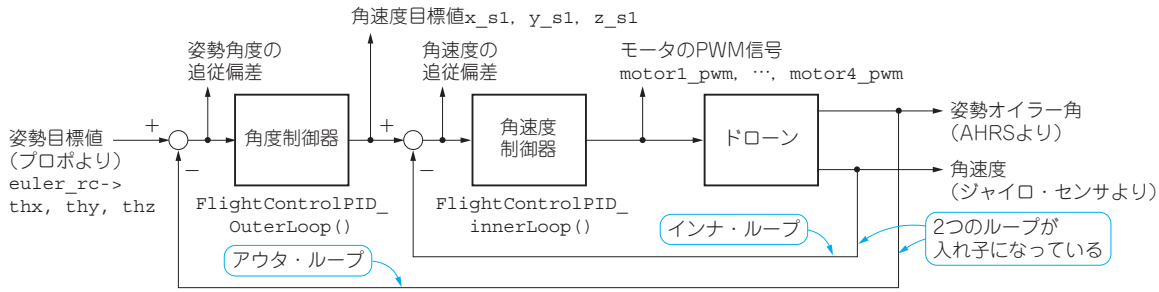

図7　インナ・ループとアウタ・ループで姿勢角と角速度を制御する

オンからオイラー角への変換が行われるようになっています(**リスト5**).

なお，航空機の力学では，x軸が機体の前方，y軸が機体の右方，z軸が機体の下方となるようにとり，地上に固定する基準座標系もz_0軸が鉛直下方となるようにとることが多いです．この場合，ϕがロール角，θがピッチ角となり，また，ヨー角ψは正の回転方向が逆になります．

姿勢制御

● 角速度と姿勢角をフィードバック制御する

姿勢制御とは，機体の傾きや方向が望み通りになるように，4つのモータに与えるPWM信号を決めてプロペラの回転速度を調整する計算処理です．ST-DRONEのFCUのソースコードに実装されている姿勢制御系は，おおまかに描くと**図7**のブロック線図のようになっています．

センサなどで得られた機体の角速度や姿勢角度をもとに，制御入力であるモータのPWM指令値が計算され，それにより機体が運動し角速度や姿勢角度が変化する，という信号ループが構成されています．このような制御手法をフィードバック制御と呼びます．ブロック線図を見ると，2つのループが入れ子になっていることが分かります．このうち，角速度を制御する内側のループをインナ・ループ，姿勢角度を制御する外側のループをアウタ・ループと呼びます．

● 角度制御器と角速度制御器の仕事

▶角度制御器

角度制御器は，姿勢角度の目標値と現在の姿勢角度から，機体姿勢が目標値に追従するために必要な角速度を求めます．次に，求めた角速度を目標値として角速度制御器に渡します．

▶角速度制御器

角速度制御器は，角度制御器から受け取った角速度目標値と現在の角速度から，機体角速度が目標値に追従するよう4つのモータのPWM指令値を求め，各モータへ与えます．

● 角度制御器と角速度制御器はPID制御で目標値を追従

角速度制御器，角度制御器ともに制御手法は「PID制御」と呼ばれる手法を用いて実装されています．これは，目標値から制御量(角速度または角度)を引いた追従偏差を求め，その値に比例ゲインを乗算した値と，追従偏差を時間積分した値に積分ゲインを掛けた値および追従偏差を時間微分した値に微分ゲインを掛けた値の和をとり，それを制御器の出力とする手法で，ドローンに限らず自動制御の場面で広く適用されています．

PIDは，比例(Proportional)，積分(Integral)，微分(Differential)の頭文字をとったものです．比例/積分/微分の各ゲインは多くの場合定数で，制御系設計における設計パラメータとなります．つまり，各ゲインの値を変えることで，安定性や制御性能が変わってきます．また，3つのゲインのうち一部を0にすることもあります．例えば，微分ゲインを0にした場合は，Dを外してPI制御と呼ばれることがあります．

● ソースコードに対応する処理内容

姿勢制御のソースコード`flight_control.c`を見ていきます．ヘッダ・ファイル`flight_control.h`には各種パラメータの値が入っているので，そちらもあわせて見てください(**リスト6**，**リスト7**).

154～191行目の`FlightControlPID_OuterLoop()`関数は，アウタ・ループ角度制御器の実装です．166～172行目はピッチ角，75～181行目はロール角，184～190行目はヨー角の制御です．

▶ピッチ角の制御内容

ピッチ角，ロール角，ヨー角の処理内容はほとんど同じなので，ここでは，ピッチ角の制御を例に解説します．

・166行目

リスト6　姿勢制御を行う処理が記述されたflight_control.hから抜粋したソースコード

```
1   #ifndef _FLIGHT_CONTROL_H_
2   #define _FLIGHT_CONTROL_H_
        :
28  #define ROLL_PID_KP1            3
29  #define ROLL_PID_KI1            0
31  #define ROLL_PID_KP2            100
                                /* test minidrone */
35  #define ROLL_PID_KI2            100
                                /* test minidrone */
38  #define ROLL_PID_KD2            10
                              //(x/PID_SAMPLING_TIME)
40  #define ROLL_PID_I1_LIMIT       2.0       //5 degree
42  #define ROLL_PID_I2_LIMIT       20.0
        :
46  #define PITCH_PID_KP1           ROLL_PID_KP1
47  #define PITCH_PID_KI1           ROLL_PID_KI1
48  #define PITCH_PID_KP2           ROLL_PID_KP2
49  #define PITCH_PID_KI2           ROLL_PID_KI2
50  #define PITCH_PID_KD2           ROLL_PID_KD2
51  #define PITCH_PID_I1_LIMIT      ROLL_PID_I1_LIMIT
52  #define PITCH_PID_I2_LIMIT      ROLL_PID_I2_LIMIT
        :
61  #define YAW_PID_KP1             4
62  #define YAW_PID_KI1             0
63  #define YAW_PID_KP2             1000
64  #define YAW_PID_KI2             0
65  #define YAW_PID_KD2             0
                              //(x/PID_SAMPLING_TIME)
66  #define YAW_PID_I1_LIMIT        2.0       //5 degree
68  #define YAW_PID_I2_LIMIT        2
69
70  #define PID_SAMPLING_TIME       0.00125
71
72  #define D_FILTER_COFF           0.025f
        :
88  #define FIFO_Order              5
89  #define MID_FIFO                (FIFO_Order>>1)
90  #define FIFO_Order_Recip        (1.0/FIFO_Order)
        :
117 #endif
```

ロール（Y軸）角速度制御の制御ゲイン
ロール姿勢角度制御の制御ゲイン
ヨー（Z軸）角速度制御の制御ゲイン
ヨー角度制御の制御ゲイン
インナ・ループのサンプリング時間［s］
角速度追従偏差の微分値に適用するノイズ・フィルタの係数
ロール角度・角速度制御の追従偏差積分値に対する制限値
ヨー角度・角速度制御の追従偏差積分値に対する制限値
ピッチ角度・角速度制御の制御ゲインなどはロールと同じ

追従偏差，つまりプロポの操作量より定められた角度目標値euler_rc->thxからAHRSで推定されたピッチ角euler_ahrs->thxを引いた値を求めます．

• 167行目

積分制御の準備として追従偏差を数値積分し，値をpid_x_integ1に格納します．数値積分に使用するサンプリング時間pid->tsは，FlightControl PID_OuterLoop()関数の呼び出し周期と合わせるべきなので，正しくはpid->tsのところに5を掛ける修正が必要です．ただし，後述のように積分制御のゲインは0なので，影響はありません．

• 158 ～ 163行目

積分値は，158 ～ 163行目に書かれているようにスロットル操作量がしきい値以下になると0に初期化されます．

• 168 ～ 171行目

積分値が±pid->x_i1_limitの範囲を超えないよう制限を設けています．このような制限方法をリミッタや飽和と呼びます．

• 172行目

比例/積分制御の計算を行って，制御器出力pid->x_s1を求めています．実際には，積分ゲインは0になっているので比例制御のみです．

193 ～ 273行目のFlightControlPID_innerLoop()関数は，インナ・ループ角速度制御器の実装です．前半の207 ～ 220行目はピッチ角速度，223 ～ 236行目はロール角速度，239 ～ 250行目はヨー角速度の制御です．

▶ピッチ角速度の制御内容

ピッチ角速度，ロール角速度，ヨー角速度の処理内容はほとんど同じなので，ここでは，ピッチ角速度の制御を例に解説します．

基本的な流れとしては，角度制御器と同様で，違いとしては微分制御があるところと，追従偏差の微分値にノイズ・フィルタを適用しているところ，そしてPID制御の最終的な計算結果にもリミッタを適用しているところです．

• 207行目

角速度の追従偏差を求めています．

• 208行目

207行目で求めたものを数値積分しています．

• 209 ～ 212行目

数値積分した値に対するリミッタです．

• 213 ～ 214行目

追従偏差の数値微分です．

• 215 ～ 216行目

数値微分の値をノイズ・フィルタに通しています．

1次遅れと呼ばれるフィルタを実装しています．このフィルタは，よく使われており高周波数成分の信号の振幅を小さくする働きがあります．D_FILTER_COFFは，折点周波数を指定する係数です．おおむね，これを2π×サンプリング周期[s]で割った値が折点周波数[Hz]となります．初期設定は0.025なので，折点周波数は約3.2Hzです．

• 217行目

PID制御の計算を行い，出力pid->x_s2が決められます．

• 219 ～ 220行目

出力が過大にならないようにするリミッタです．こちらは，積分値に対するリミッタとは目的が少々異なり，制御器の出力が過大になり機体の姿勢が崩れて墜落する，といった危険を避ける目的で入れていて，自

リスト7　姿勢制御を行う処理が記述された flight_control.c から抜粋したソースコード

```
1   #include "flight_control.h"
2   #include "rc.h"
3   #include <math.h>
      :
154 void FlightControlPID_OuterLoop
    (EulerAngleTypeDef *euler_rc, EulerAngleTypeDef
          *euler_ahrs, AHRS_State_TypeDef *ahrs,
                    P_PI_PIDControlTypeDef *pid)
155 {
156   float error;
157
158   if(gTHR<MIN_THR)
159   {
160     pid_x_integ1 = 0;
161     pid_y_integ1 = 0;
162     pid_z_integ1 = 0;
163   }
164
165   //x-axis pid
166   error = euler_rc->thx - euler_ahrs->thx;
167   pid_x_integ1 += error*pid->ts;
168   if(pid_x_integ1 > pid->x_i1_limit)
169     pid_x_integ1 = pid->x_i1_limit;
170   else if(pid_x_integ1 < -pid->x_i1_limit)
171     pid_x_integ1 = -pid->x_i1_limit;
172   pid->x_s1 =  pid->x_kp1*error +
                          pid->x_ki1*pid_x_integ1;
173
174   //y-axis pid
175   error = euler_rc->thy - euler_ahrs->thy;
176   pid_y_integ1 += error*pid->ts;
177   if(pid_y_integ1 > pid->y_i1_limit)
178     pid_y_integ1 = pid->y_i1_limit;
179   else if(pid_y_integ1 < -pid->y_i1_limit)
180     pid_y_integ1 = -pid->y_i1_limit;
181   pid->y_s1 =  pid->y_kp1*error +
                          pid->y_ki1*pid_y_integ1;
182
183   //z-axis pid
184   error = euler_rc->thz - euler_ahrs->thz;
185   pid_z_integ1 += error*pid->ts;
186   if(pid_z_integ1 > pid->z_i1_limit)
187     pid_z_integ1 = pid->z_i1_limit;
188   else if(pid_z_integ1 < -pid->z_i1_limit)
189     pid_z_integ1 = -pid->z_i1_limit;
190   pid->z_s1 =  pid->z_kp1*error +
                          pid->z_ki1*pid_z_integ1;
191 }
192
193 void FlightControlPID_innerLoop
    (EulerAngleTypeDef *euler_rc, Gyro_Rad *gyro_rad,
     AHRS_State_TypeDef *ahrs, P_PI_PIDControlTypeDef
            *pid, MotorControlTypeDef *motor_pwm)
194 {
195   float error, deriv;
196
197   if(gTHR<MIN_THR)
198   {
199     pid_x_integ2 = 0;
200     pid_y_integ2 = 0;
201     pid_z_integ2 = 0;
202   }
203
204   dt_recip = 1/pid->ts;
205
206   //X Axis
207   error = pid->x_s1 - gyro_rad->gx;
208   pid_x_integ2 += error*pid->ts;
209   if(pid_x_integ2 > pid->x_i2_limit)
210     pid_x_integ2 = pid->x_i2_limit;
211   else if(pid_x_integ2 < -pid->x_i2_limit)
212     pid_x_integ2 = -pid->x_i2_limit;
213   deriv = (error - pid_x_pre_error2)*dt_recip;
214   pid_x_pre_error2 = error;
215   deriv = pid_x_pre_deriv +
          (deriv - pid_x_pre_deriv)*D_FILTER_COFF;
216   pid_x_pre_deriv = deriv;
217   pid->x_s2 = pid->x_kp2*error +
     pid->x_ki2*pid_x_integ2 + pid->x_kd2*deriv;
218
219   if(pid->x_s2 > MAX_ADJ_AMOUNT)  pid->x_s2 =
                                  MAX_ADJ_AMOUNT;
220   if(pid->x_s2 < -MAX_ADJ_AMOUNT)  pid->x_s2 =
                                 -MAX_ADJ_AMOUNT;
221
222   //Y Axis
223   error = pid->y_s1 - gyro_rad->gy;
224   pid_y_integ2 += error*pid->ts;
225   if(pid_y_integ2 > pid->y_i2_limit)
226     pid_y_integ2 = pid->y_i2_limit;
227   else if(pid_y_integ2 < -pid->y_i2_limit)
228     pid_y_integ2 = -pid->y_i2_limit;
229   deriv = (error - pid_y_pre_error2)*dt_recip;
230   pid_y_pre_error2 = error;
231   deriv = pid_y_pre_deriv +
          (deriv - pid_y_pre_deriv)*D_FILTER_COFF;
232   pid_y_pre_deriv = deriv;
233   pid->y_s2 = pid->y_kp2*error +
     pid->y_ki2*pid_y_integ2 + pid->y_kd2*deriv;
234
235   if(pid->y_s2 > MAX_ADJ_AMOUNT)  pid->y_s2 =
                                  MAX_ADJ_AMOUNT;
236   if(pid->y_s2 < -MAX_ADJ_AMOUNT)  pid->y_s2 =
                                 -MAX_ADJ_AMOUNT;
237
238   //Z Axis
239   error = pid->z_s1 - gyro_rad->gz;
240   pid_z_integ2 += error*pid->ts;
241   if(pid_z_integ2 > pid->z_i2_limit)
242     pid_z_integ2 = pid->z_i2_limit;
243   else if(pid_z_integ2 < -pid->z_i2_limit)
244     pid_z_integ2 = -pid->z_i2_limit;
245   deriv = (error - pid_z_pre_error2)*dt_recip;
246   pid_z_pre_error2 = error;
247   pid->z_s2 = pid->z_kp2*error +
     pid->z_ki2*pid_z_integ2 + pid->z_kd2*deriv;
248
249   if(pid->z_s2 > MAX_ADJ_AMOUNT_YAW)
                 pid->z_s2 = MAX_ADJ_AMOUNT_YAW;
250   if(pid->z_s2 < -MAX_ADJ_AMOUNT_YAW)
               pid->z_s2 = -MAX_ADJ_AMOUNT_YAW;
251
252
253 #ifdef MOTOR_DC
254
255   motor_thr = 0.33333f*gTHR + 633.333f;
                //Remocon Devo7E >> 630 to 1700
256
257 #endif
258
259 #ifdef MOTOR_ESC
      :
266 #endif
267
268   motor_pwm->motor1_pwm = motor_thr - pid->x_s2
           - pid->y_s2 + pid->z_s2 + MOTOR_OFF1;
269   motor_pwm->motor2_pwm = motor_thr + pid->x_s2
           - pid->y_s2 - pid->z_s2 + MOTOR_OFF2;
270   motor_pwm->motor3_pwm = motor_thr + pid->x_s2
           + pid->y_s2 + pid->z_s2 + MOTOR_OFF3;
271   motor_pwm->motor4_pwm = motor_thr - pid->x_s2
           + pid->y_s2 - pid->z_s2 + MOTOR_OFF4;
272
273 }
```

アウタ・ループ姿勢角度制御
インナ・ループ角速度制御
追従偏差の積分値が過大にならないようリミッタをかけて飽和させる
追従偏差の積分を求める
スロットルを下げているとき（着陸中など）は追従偏差の積分値をリセットする
追従偏差を求める
166～172行目はピッチ姿勢角の制御
206～220行目はX軸（ピッチ）角速度の制御
比例（P）制御
積分（I）制御
微分（D）制御
ロール姿勢角の制御
ヨー角の制御
追従偏差の微分を求める
PI制御により角速度目標値を求める
PID制御の計算を行う
追従偏差の微分値をノイズフィルタに通す
Y軸（ロール）角速度の制御
Z軸（ヨー）角速度の制御（～250行目）
推力指令値を求める
制御出力が過大にならないようリミッタをかけて飽和させる
推力指令値および角速度制御器の出力をコマンド分配則に通して，4つのモータのPWM値へ変換する

動制御を行う際は一般によく使われます．

● 積分値にリミッタが必要な理由…目標値追従性の悪化を防ぐ

制御理論の詳細は省きますが，一般に積分制御は風などの一定値の外乱がある環境下や一定値の目標値（ステップ目標値と呼ぶ）が与えられた場合に目標値に定常偏差なく追従させるために不可欠です．

しかし，その一方で次のような弊害もあります．積分は追従偏差が0でなければ常に増減するため，過大な値をとる可能性があります．機体姿勢が目標値を通り越して（オーバシュートと呼ぶ）追従偏差の符号が反転した場合は，できればすぐに制御器出力の符号も反転させたいところです．しかし，追従偏差の積分値の絶対値が大きくなりすぎると，積分値の符号が反転するまでに時間がかかってしまい，制御器出力の符号がなかなか反転せず，目標値追従性を悪化させることがあります．このような現象をワインドアップと呼び，積分制御を実装する場合はこのようなことが起きないようにアンチ・ワインドアップの実装もあわせて行います．ここまでの解説でお分かりだと思いますが，積分値にリミッタが設けられているのは，アンチ・ワインドアップのためです．

● 数値微分は高周波ノイズの増幅に注意が必要

角速度制御の追従偏差には，ジャイロで測った角速度`gyro_rad->gx`があり，高周波成分のノイズを含んでいます．このような信号を数値微分すると，ノイズを増幅し微分制御器の出力が激しく振動することになり，制御性能の劣化はもちろんモータおよび関連回路の故障につながることがあります．そのため，追従偏差の数値微分の値を，ノイズ・フィルタに通し，ノイズ成分を小さくしています．

● 1次遅れフィルタはローパス特性

1次遅れ系は，通した信号の振幅が0.7倍（－3dB）となる周波数を折点周波数と呼び，これよりも低い周波数は振幅がほとんど変化せず，高い周波数は周波数が10倍になるごとに振幅が0.1倍になるという特性があります．つまり，折点周波数を低くするとフィルタの効きが強くなり，高くするとフィルタの効きが弱まります．

ただし副作用として，折点周波数付近から高い周波数において信号の位相に遅れが生じ，最大90°遅れます．位相遅れは制御系に安定性や性能の劣化をもたらす場合があるので注意が必要です．

● 制御器出力は機体の状態に合わせて4つのモータへ入力する

角速度制御器の計算結果はどのようにモータへ入力すればよいでしょうか．ST-DRONEは，4枚のプロペラそれぞれの回転速度の組み合わせにより姿勢や方向を制御するので，制御器出力をこれら4つのプロペラのモータに適切に分配しなくてはなりません．

例えば，スロットルを上げた場合は4つ全てのモータの回転速度を上げる，右進のため機体を右に傾けるには1番と2番のプロペラ回転速度を下げ，3番と4番は回転速度を上げる，といったようにします．x，y，z各軸の角速度制御器の出力`pid->x_s2`，`y_s2`，`z_s2`，およびスロットル操作に対応する出力`motor_thr`（255行目で計算）を各モータのPWM値`motor1_pwm`，…，`motor4_pwm`に換算する式は行列計算で書くと次式となります．

$$\begin{bmatrix} \text{motor1\_pwm} \\ \text{motor2\_pwm} \\ \text{motor3\_pwm} \\ \text{motor4\_pwm} \end{bmatrix} = C_{\text{dist}} \begin{bmatrix} \text{motor\_thr} \\ \text{x\_s2} \\ \text{y\_s2} \\ \text{z\_s2} \end{bmatrix} + \begin{bmatrix} \text{motor\_OFF1} \\ \text{motor\_OFF2} \\ \text{motor\_OFF3} \\ \text{motor\_OFF4} \end{bmatrix} \quad \cdots\cdots(5)$$

$$C_{\text{dist}} = \begin{bmatrix} 1 & -1 & -1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \quad \cdots\cdots(6)$$

このような計算式は，コマンド分配則と呼ばれます．ソースコードは，268～271行目です．`MOTOR_OFF1`，…，`MOTOR_OFF4`は微調整用の定数で，初期設定は0になっています．

◆参考・引用*文献◆

(1) Getting started with the STSW-FCU001 reference design firmware for mini drones，UM2329，STマイクロエレクトロニクス．
`https://www.st.com/resource/enuser_manual/dm00456644.pdf`

ふじわら・だいご