Px4 平台之我见

Author：jingwenyi

第一章 各传感器数据的更新流程

第一节 imu 数据更新代码流程

```
void Copter::loop()
{
    fast_loop();
}--->
void Copter::fast_loop()
{
    read_AHRS();
}--->
void Copter::read_AHRS(void)
{
//这个地方调用了 AP_AHRS_NavEKF 中的 updata,而不是 AP_AHRS_DCM 的 updata
    ahrs.update();
}--->
void AP_AHRS_NavEKF::update(void)
{
    AP_AHRS_DCM::update(); //这里直接调用了 AP_AHRS_DCM 的 updata
}--->
AP_AHRS_DCM::update(void)
{
    //跟新 imu 的值
    _ins.update();
}--->
void AP_InertialSensor::update(void)
{
    for (uint8_t i=0; i<_backend_count; i++) {
        _backends[i]->update(); //imu 更新
    }
}-->
bool AP_InertialSensor_PX4::update(void)
{
    _get_sample();//读取 imu 的数据

    //将加速度计的值付给 AP_InertialSensor 结构体_accel 成员
    _publish_accel(_accel_instance[k], accel, false);
    //将加速度计对时间积分的速度付给 AP_InertialSensor 结构体
_delta_velocity
    _publish_delta_velocity(_accel_instance[k],
         _delta_velocity_accumulator[k], _delta_velocity_dt[k]);
```

```
    //将陀螺仪的值付给 AP_InertialSensor 结构体_gyro 成员
    _publish_gyro(_gyro_instance[k], gyro, false);
    //将 ekf 需要使用的值保持到 AP_InertialSensor 结构体_delta_angle
    _publish_delta_angle(_gyro_instance[k],
                             _delta_angle_accumulator[k]);
}
_get_sample 简单分析
void AP_InertialSensor_PX4::_get_sample()
{
    //获取陀螺仪数据
    bool gyro_valid = _get_gyro_sample(i,gyro_report);
    //获取加速度计数据
    bool accel_valid = _get_accel_sample(i,accel_report);
    .....
    //处理采样的数据，并保存到 AP_InertialSensor_PX4
    _new_gyro_sample(i,gyro_report);
    _new_gyro_sample(i,gyro_report);
    .......
}

-->AP_InertialSensor_PX4::_new_accel_sample(......)
{

    //将加速度的数据加入到_accel_in
    _accel_in[i] = _accel_filter[i].apply(accel);
    // 将加速度对时间积分算出速度，供 ekf 使用
    Vector3f delVel = Vector3f(accel.x, accel.y, accel.z) * dt;
    _delta_velocity_accumulator[i] += delVel;
        _delta_velocity_dt[i] += dt;
}
-->AP_InertialSensor_PX4::_new_gyro_sample(.....)
{
    //将陀螺仪的数据记录到_gyro_in 中
    _gyro_in[i] = _gyro_filter[i].apply(gyro);
    //计算角度的累积值供 ekf 使用
//http://www.sage.unsw.edu.au/snap/publications/tian_etal2010b.pdf
    Vector3f delConing = ((_delta_angle_accumulator[i]+
                  _last_delAng[i]*(1.0f/6.0f)) % delAng) * 0.5f;
    _delta_angle_accumulator[i] += delAng + delConing;
}
```

第二节 gps 数据更新代码流程

```
//启动一个跟新 gps 的任务
{ SCHED_TASK(update_GPS),              8,     200 }--->
void Copter::update_GPS(void)
{
    gps.update(); //更新 gps 数据
    //将 gps 数据写入 flash
    DataFlash.Log_Write_GPS(gps, i, current_loc.alt);
}--->
Void AP_GPS::update(void)
{
    update_instance(i);
}-->
AP_GPS::update_instance(uint8_t instance)
{
    bool result = drivers[instance]->read();
}-->
AP_GPS_PX4::read(void)//jwy gps read
{
    //从串口中读
    if (OK == orb_copy(ORB_ID(vehicle_gps_position), _gps_sub, &_gps_pos)) {

    if (_gps_pos.fix_type >= 2) { //保存位置数据
                        state.location.lat = _gps_pos.lat;
                        state.location.lng = _gps_pos.lon;
                        state.location.alt = _gps_pos.alt/10;

                        state.ground_speed = _gps_pos.vel_m_s;
                        state.ground_course_cd = int32_t(double(_gps_pos.cog_rad) / M_PI *
18000. +.5);

                        // convert epoch timestamp back to gps epoch - evil hack until we get the
genuine
                        // raw week information (or APM switches to Posix epoch ;-) )
                        uint64_t epoch_ms = uint64_t(_gps_pos.time_utc_usec/1000.+.5);
                        uint64_t gps_ms = epoch_ms - DELTA_POSIX_GPS_EPOCH +
LEAP_MS_GPS_2014;
                        state.time_week = uint16_t(gps_ms / uint64_t(MS_PER_WEEK));
                        state.time_week_ms = uint32_t(gps_ms -
uint64_t(state.time_week)*MS_PER_WEEK);

                        if (_gps_pos.time_utc_usec == 0) {
                          // This is a work-around for
```

```
https://github.com/PX4/Firmware/issues/1474
                        // reject position reports with invalid time, as APM adjusts it's clock
after the first lock has been aquired
                        state.status = AP_GPS::NO_FIX;
                    }
                }
                if (_gps_pos.fix_type >= 3) {//保存速度信息
                    state.have_vertical_velocity = _gps_pos.vel_ned_valid;
                    state.velocity.x = _gps_pos.vel_n_m_s;
                    state.velocity.y = _gps_pos.vel_e_m_s;
                    state.velocity.z = _gps_pos.vel_d_m_s;
                }
}
```

第三节 rc 数据处理流程

```
//启动一个 rc 任务
{ SCHED_TASK(rc_loop),                  4,     130 }-->
void Copter::rc_loop()
{
    read_radio(); //读取遥控器传来的 rc 值
    read_control_switch();//切换飞行模式
}--->
//获取信道的 pwm 值
void Copter::read_radio()
{
    .....
    //获得所有信道的 pwm 值，这里最后调用的 PX4RCInput 中的 read
    RC_Channel::set_pwm_all();
    ........
}
//切换信道
void Copter::read_control_switch()
{
    int8_t switch_position;
    //模式切换的信道为信道 5，一共可配置 6 种模式，六种模式对应的 pwm 值
如下
    if      (g.rc_5.radio_in < 1231) switch_position = 0;
    else if (g.rc_5.radio_in < 1361) switch_position = 1;
    else if (g.rc_5.radio_in < 1491) switch_position = 2;
```

```
    else if (g.rc_5.radio_in < 1621) switch_position = 3;
    else if (g.rc_5.radio_in < 1750) switch_position = 4;
    else switch_position = 5;
    ......
    //进行模式切换，这里需要说明的是，这个地方只选择了第几种模式，至于
该模式下是那种飞行模式，需要在 misson 软件中配置
    if (set_mode(flight_modes[switch_position]))
    ......
}-->
bool Copter::set_mode(uint8_t mode)
{

}
```

到了这里下面就不用分析了，基本都懂。

Rc 在驱动中的处理过程

Rc 的接口为

```
class AP_HAL::RCInput {
};
```

其实现类为

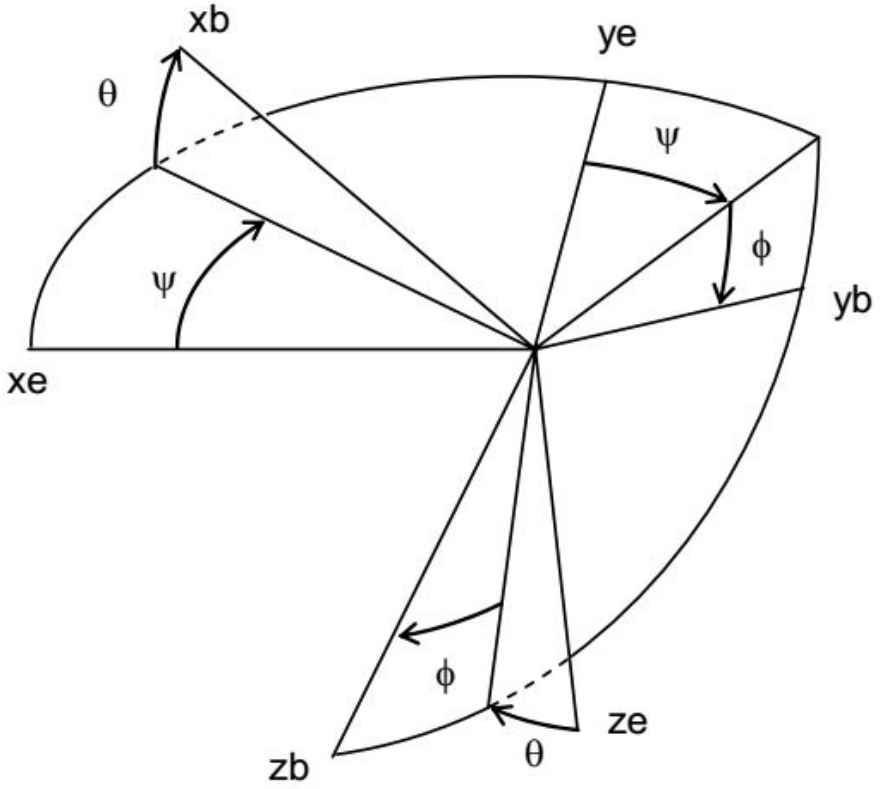
```
class PX4::PX4RCInput : public AP_HAL::RCInput {
    ......
    //为上次提供了两个读_rcin 中信道值的函数
    uint16_t read(uint8_t ch);
    uint8_t read(uint16_t* periods, uint8_t len);
    void _timer_tick(void);
    ......
    private:
    struct rc_input_values _rcin;
};
//系统刚启动时，启动了这个定时器
void PX4RCInput::_timer_tick(void)
{
    //读取遥控器的 pwm 数据保存到 18 个信道中_rcin 的
    //values[RC_INPUT_MAX_CHANNELS]变量中
    orb_copy(ORB_ID(input_rc), _rc_sub, &_rcin);
}
```

第二章 代码分析基础知识

第一节 3X3 矩阵操作

1、如何通过角度值求 DCM 矩阵

飞机和地球坐标系建立如下图



其中 xe、ye、ze 是地球坐标系统，ze 指向地心，xe 指向正东方，ye 指向正北方；xb、yb、zb 为机体坐标系；

第一步：假设我站在机体坐标中，我需要通过先绕 Xb 轴旋转ϕ，再旋转 Yb 轴旋转θ，最后绕 Zb 轴旋转ψ，回到地球坐标系；先求出每次旋转的矩阵。

如果绕机体 X 轴旋转的角度为ϕ，那么

$$L(\phi)=\begin{bmatrix}1 & 0 & 0\\ 0 & \cos\phi & -\sin\phi\\ 0 & \sin\phi & \cos\phi\end{bmatrix};$$

如果绕机体 Y 轴旋转的角度为θ，那么

$$L(\theta)=\begin{bmatrix}\cos\theta & 0 & \sin\theta\\ 0 & 1 & 0\\ -\sin\theta & 0 & \cos\theta\end{bmatrix};$$

如果绕机体 Z 轴旋转的角度为ψ，那么

$$L(\psi)\begin{bmatrix}\cos\psi & -\sin\psi & 0\\ \sin\psi & \cos\psi & 0\\ 0 & 0 & 1\end{bmatrix};$$

第二步：由于站在机体坐标上需要按照 X->Y->Z 轴的顺序，经过 3 次旋转，才能回到地球坐标系；反过来如果站在地球坐标系，则需要经过 Z->Y->X 的三次旋转才能到达机体坐标系；因为我们可以列出从地球坐标系到机体坐标系的 DCM 矩阵。

$$L(\phi,\theta,\psi)=L(\psi)*L(\theta)*L(\phi);$$

计算得：

$$L(\phi,\theta,\psi)=\begin{bmatrix}\cos\theta\cos\psi & \sin\phi\sin\theta\cos\psi-\cos\phi\sin\psi & \cos\phi\sin\theta\cos\psi+\sin\phi\sin\psi\\ \cos\theta\sin\psi & \sin\phi\sin\theta\sin\psi+\cos\phi\cos\psi & \cos\phi\sin\theta\sin\psi-\sin\phi\cos\psi\\ -\sin\theta & \sin\phi\cos\theta & \cos\phi\cos\theta\end{bmatrix}$$

这个公式表示的是地轴系与体坐标的方向余弦的关系，该公式对应的矩阵值为

$$\mathbf{R}=\begin{bmatrix}r_{xx} & r_{xy} & r_{xz}\\ r_{yx} & r_{yy} & r_{yz}\\ r_{zx} & r_{zy} & r_{zz}\end{bmatrix}=\text{rotation matrix}$$

本文解析的代码在 matrix3.cpp 中
我进行转换下
设

$$L(\phi,\theta,\psi)=\begin{bmatrix}a\\ b\\ c\end{bmatrix}=\begin{bmatrix}a_x & a_y & a_z\\ b_x & b_y & b_z\\ c_x & c_y & c_z\end{bmatrix};$$

以上整个求解过程是对 matrix3.cpp 代码中 from_euler 函数的解析

```
template <typename T>
void Matrix3<T>::from_euler(float roll, float pitch, float yaw)
{
    float cp = cosf(pitch);//pitch 表示俯仰相对于地球坐标系的角度值
    float sp = sinf(pitch);
    float sr = sinf(roll);//roll 表示横滚相对于地球坐标系的角度值
    float cr = cosf(roll);
    float sy = sinf(yaw);//yaw 表示偏航相对于地球坐标的角度值
    float cy = cosf(yaw);

    a.x = cp * cy;
    a.y = (sr * sp * cy) - (cr * sy);
    a.z = (cr * sp * cy) + (sr * sy);
    b.x = cp * sy;
    b.y = (sr * sp * sy) + (cr * cy);
    b.z = (cr * sp * sy) - (sr * cy);
    c.x = -sp;
    c.y = sr * cp;
    c.z = cr * cp;
}
```

2、如何通过 DCM 矩阵值来求角度值

数学公式：

$$\theta = -\arcsin(r_{zx})$$
$$\phi = \operatorname{atan}2(r_{zy}, r_{zz})$$
$$\psi = \operatorname{atan}2(r_{yx}, r_{xx})$$

函数 to_euler 式通过上面的 3 个公式求出对应的角度的

```
template <typename T>
void Matrix3<T>::to_euler(float *roll, float *pitch, float *yaw) const
{
    if (pitch != NULL) {
        *pitch = -safe_asin(c.x);
    }
    if (roll != NULL) {
        *roll = atan2f(c.y, c.z);
    }
    if (yaw != NULL) {
        *yaw = atan2f(b.x, a.x);
    }
}
```

3、从地球坐标系到体坐标系，按照 Z->X->Y 轴的顺序旋转可以得到

$$L(\phi,\theta,\psi) = L(\psi) * L(\phi) * L(\theta)$$

计算得：

$$L(\phi,\theta,\psi) = \begin{bmatrix} \cos\theta\cos\psi - \sin\phi\sin\theta\sin\psi & -\sin\phi\sin\psi & \cos\theta\cos\psi + \sin\phi\cos\theta\sin\psi \\ \cos\theta\sin\psi + \sin\phi\sin\theta\cos\psi & \cos\phi\cos\psi & \sin\theta\sin\psi - \sin\phi\cos\theta\cos\psi \\ -\cos\phi\sin\theta & \sin\phi & \cos\phi\cos\theta \end{bmatrix}$$

对应的函数为

```
template <typename T>
void Matrix3<T>::from_euler312(float roll, float pitch, float yaw)
{
    float c3 = cosf(pitch);
    float s3 = sinf(pitch);
    float s2 = sinf(roll);
    float c2 = cosf(roll);
    float s1 = sinf(yaw);
    float c1 = cosf(yaw);

    a.x = c1 * c3 - s1 * s2 * s3;
    b.y = c1 * c2;
    c.z = c3 * c2;
    a.y = -c2*s1;
    a.z = s3*c1 + c3*s2*s1;
    b.x = c3*s1 + s3*s2*c1;
    b.z = s1*s3 - s2*c1*c3;
    c.x = -s3*c2;
    c.y = s2;
}
```

4、根据陀螺仪的角度值，来计算当前机体的姿态 DCM 矩阵

其使用的方法是：机体坐标的每个轴的向量与 g(陀螺仪改变的角度向量)求叉积，这里求的是角度改变后，姿态在各个方向上的变化量，所以最后使用了矩阵的加法

```
template <typename T>
void Matrix3<T>::rotate(const Vector3<T> &g)  //jwy
{
    Matrix3<T> temp_matrix;
    //a1 = axg
    temp_matrix.a.x = a.y * g.z - a.z * g.y;
    temp_matrix.a.y = a.z * g.x - a.x * g.z;
    temp_matrix.a.z = a.x * g.y - a.y * g.x;
```

```
    //b1 = bxg
    temp_matrix.b.x = b.y * g.z - b.z * g.y;
    temp_matrix.b.y = b.z * g.x - b.x * g.z;
    temp_matrix.b.z = b.x * g.y - b.y * g.x;
    //c1 = cxg
    temp_matrix.c.x = c.y * g.z - c.z * g.y;
    temp_matrix.c.y = c.z * g.x - c.x * g.z;
    temp_matrix.c.z = c.x * g.y - c.y * g.x;

    (*this) += temp_matrix;//使用矩阵的加法，把角度的变化量统计进去
}
```

只求 XY 轴上的叉积

```
template <typename T>
void Matrix3<T>::rotateXY(const Vector3<T> &g)
{
    Matrix3<T> temp_matrix;
    temp_matrix.a.x = -a.z * g.y;
    temp_matrix.a.y = a.z * g.x;
    temp_matrix.a.z = a.x * g.y - a.y * g.x;
    temp_matrix.b.x = -b.z * g.y;
    temp_matrix.b.y = b.z * g.x;
    temp_matrix.b.z = b.x * g.y - b.y * g.x;
    temp_matrix.c.x = -c.z * g.y;
    temp_matrix.c.y = c.z * g.x;
    temp_matrix.c.z = c.x * g.y - c.y * g.x;

    (*this) += temp_matrix;
}
```

对 XY 轴上的叉积进行取反操作

```
template <typename T>
void Matrix3<T>::rotateXYinv(const Vector3<T> &g)
{
    Matrix3<T> temp_matrix;
    temp_matrix.a.x =   a.z * g.y;
    temp_matrix.a.y = - a.z * g.x;
    temp_matrix.a.z = - a.x * g.y + a.y * g.x;
    temp_matrix.b.x =   b.z * g.y;
    temp_matrix.b.y = - b.z * g.x;
    temp_matrix.b.z = - b.x * g.y + b.y * g.x;
    temp_matrix.c.x =   c.z * g.y;
    temp_matrix.c.y = - c.z * g.x;
    temp_matrix.c.z = - c.x * g.y + c.y * g.x;
```

```
    (*this) += temp_matrix;
}
```

矩阵的正交化（请参考 Starlino_DCM_Tutorial_01.pdf）

```
template <typename T>
void Matrix3<T>::normalize(void)
{
    float error = a * b;
    Vector3<T> t0 = a - (b * (0.5f * error));
    Vector3<T> t1 = b - (a * (0.5f * error));
    Vector3<T> t2 = t0 % t1;
    a = t0 * (1.0f / t0.length());
    b = t1 * (1.0f / t1.length());
    c = t2 * (1.0f / t2.length());
}
```

矩阵的乘法运算

```
template <typename T>
Matrix3<T> Matrix3<T>::operator *(const Matrix3<T> &m) const
{
    Matrix3<T> temp (Vector3<T>(a.x * m.a.x + a.y * m.b.x + a.z * m.c.x,
                                a.x * m.a.y + a.y * m.b.y + a.z * m.c.y,
                                a.x * m.a.z + a.y * m.b.z + a.z * m.c.z),
                     Vector3<T>(b.x * m.a.x + b.y * m.b.x + b.z * m.c.x,
                                b.x * m.a.y + b.y * m.b.y + b.z * m.c.y,
                                b.x * m.a.z + b.y * m.b.z + b.z * m.c.z),
                     Vector3<T>(c.x * m.a.x + c.y * m.b.x + c.z * m.c.x,
                                c.x * m.a.y + c.y * m.b.y + c.z * m.c.y,
                                c.x * m.a.z + c.y * m.b.z + c.z * m.c.z));
    return temp;
}
```

转置矩阵

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix}^T = \begin{bmatrix} a & b & c \end{bmatrix};$$

```
template <typename T>
Matrix3<T> Matrix3<T>::transposed(void) const
{
    return Matrix3<T>(Vector3<T>(a.x, b.x, c.x),
                      Vector3<T>(a.y, b.y, c.y),
                      Vector3<T>(a.z, b.z, c.z));
}
```

清空矩阵

```
template <typename T>
void Matrix3<T>::zero(void)
{
    a.x = a.y = a.z = 0;
    b.x = b.y = b.z = 0;
    c.x = c.y = c.z = 0;
}
```

5、向量操作的相关定义函数

向量的内积操作

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} * V$$

```
template <typename T>
Vector3<T> Matrix3<T>::operator *(const Vector3<T> &v) const
{
    return Vector3<T>(a.x * v.x + a.y * v.y + a.z * v.z,
                      b.x * v.x + b.y * v.y + b.z * v.z,
                      c.x * v.x + c.y * v.y + c.z * v.z);
}
```

只求 XY 轴的

```
template <typename T>
Vector2<T> Matrix3<T>::mulXY(const Vector3<T> &v) const
{
    return Vector2<T>(a.x * v.x + a.y * v.y + a.z * v.z,
                      b.x * v.x + b.y * v.y + b.z * v.z);
}
```

转置向量的的乘积

$$\begin{bmatrix} a & b & c \end{bmatrix} * V$$

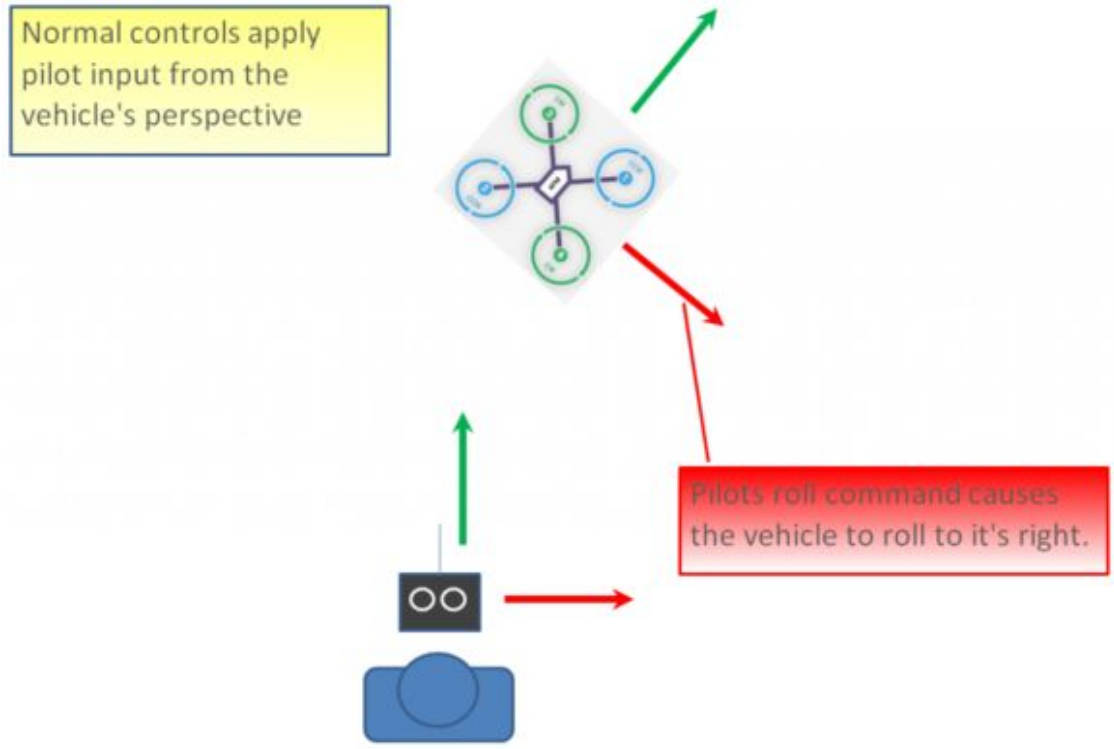
```
template <typename T>
Vector3<T> Matrix3<T>::mul_transpose(const Vector3<T> &v) const
{
    return Vector3<T>(a.x * v.x + b.x * v.y + c.x * v.z,
                      a.y * v.x + b.y * v.y + c.y * v.z,
                      a.z * v.x + b.z * v.y + c.z * v.z);
}
```

第二节 飞行器的三种模式介绍

正常模式、简单模式、超简单模式

以下关于这三种模式的理解，来自网上。

1、正常模式



在不用简单和超简单的情况下，飞手的发射机摇杆输入是对不断旋转着的飞行器进行操作的。 拿上方图示举例，当飞手进行向右（红色）的 roll 的控制的时候，模型会向它自己的右侧横滚。

如果飞手和飞行器在同一方向，控制起来就相对简单，但是如果飞行器面对着飞手，没有经验的飞手就会感觉控制全都反了。 换句话说，飞手向右控制 roll，从飞手的视角来看模型是向左移动的。

总结：正常模式就是飞机的头朝着哪个方向，那个方向就是前方，遥控器输入的 roll 和 pitch 值不需要进行投影。

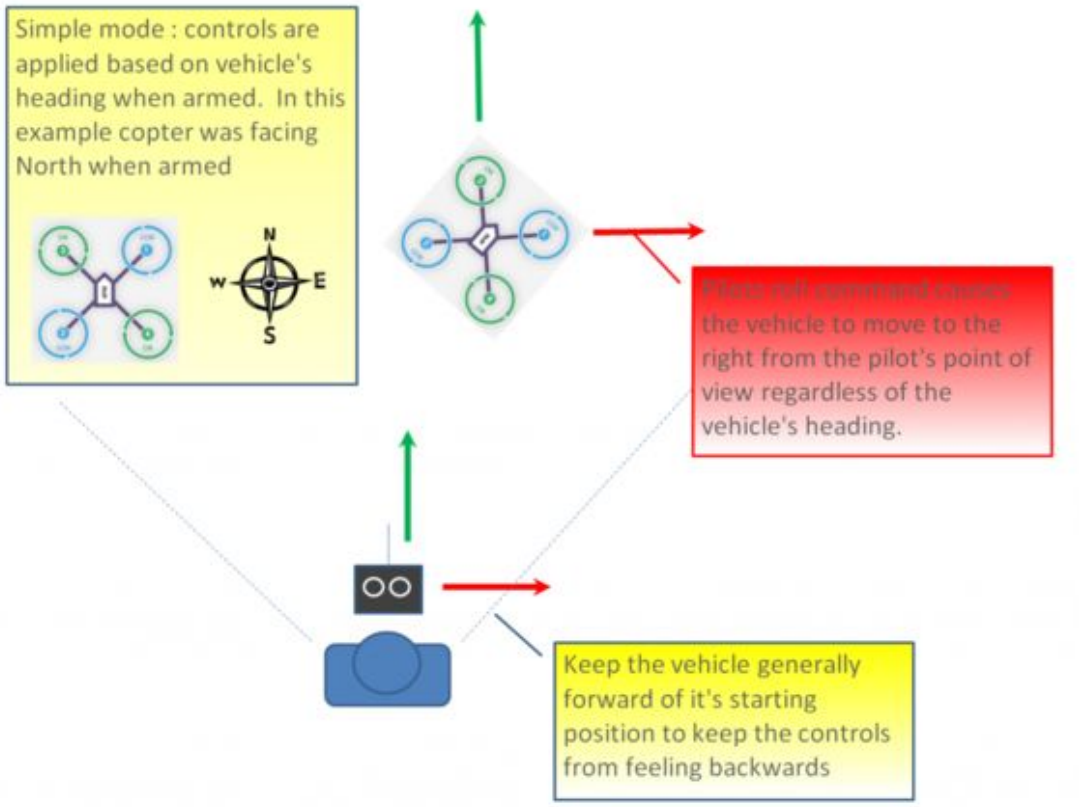
（设 $roll_{in}$ 为遥控器输入的横滚值， $pitch_{in}$ 为遥控器输入的俯仰值， $roll_b$ 为机体坐标系的横滚值， $pitch_b$ 为机体坐标系的俯仰值， $roll_e$ 为地球坐标系统的横滚值， $pitch_e$ 为地球坐标系的俯仰值，以下同此）

在正常模式下，不需要把角度转成地球坐标系

$roll_b = roll_{in}$；

$pitch_b = pitch_{in}$；

2、简单模式



和 别的系统（MultiWii 等）的“无忧（carefree）”模式差不多，这个模式可以让你飞飞行器的时候，就像它一直是它解锁时的方向一样，不用管它 现在转到了什么方向。 如果你向前推 pitch 摇杆，飞行器就会飞离你，向后推 pitch 摇杆，飞行器就会朝家的方向飞回来。 你甚至可以操作 yaw 任意旋转飞行器，但是用摇杆控制飞行器移动时是和起飞时一样的。
通常，解锁时你应该站在模型的后面，模型的机头指向正前方。 在飞行中应保持模型在起飞位置的前面，因为如果它飞到了你身后，你就会感觉所有操控都反了。
如上所述，在飞行器飞得太远了看不清头的朝向的紧急情况下，简单模式也是非常有用的。

总结：简单模式就是飞行器起飞时头部对准的方向始终为机体坐标系的 pitch 轴，也就是说启动的时候就定死了机体坐标系，所以遥控器需要传入的 roll 和 pitch 值需要转到机体坐标系，在转到地球坐标中。

（设ψ_b为机体坐标系中简单模式偏航角度，ψ_e为机体坐标系先对应地球坐标系

的 Z 轴投影角度）
简单模式下的坐标变换

$$roll_b = roll_{in} * \cos\psi_b - pitch_{in} * \sin\psi_b ;$$

$$pitch_b = roll_{in} * \sin\psi_b + pitch_{in} \cos\psi_b ;$$

在上篇 3X3 矩阵操作中，我们得到体坐标系到地球坐标系的绕 Z 轴的旋转矩阵是

$$L(\psi)\begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

去掉 Z 轴相关的

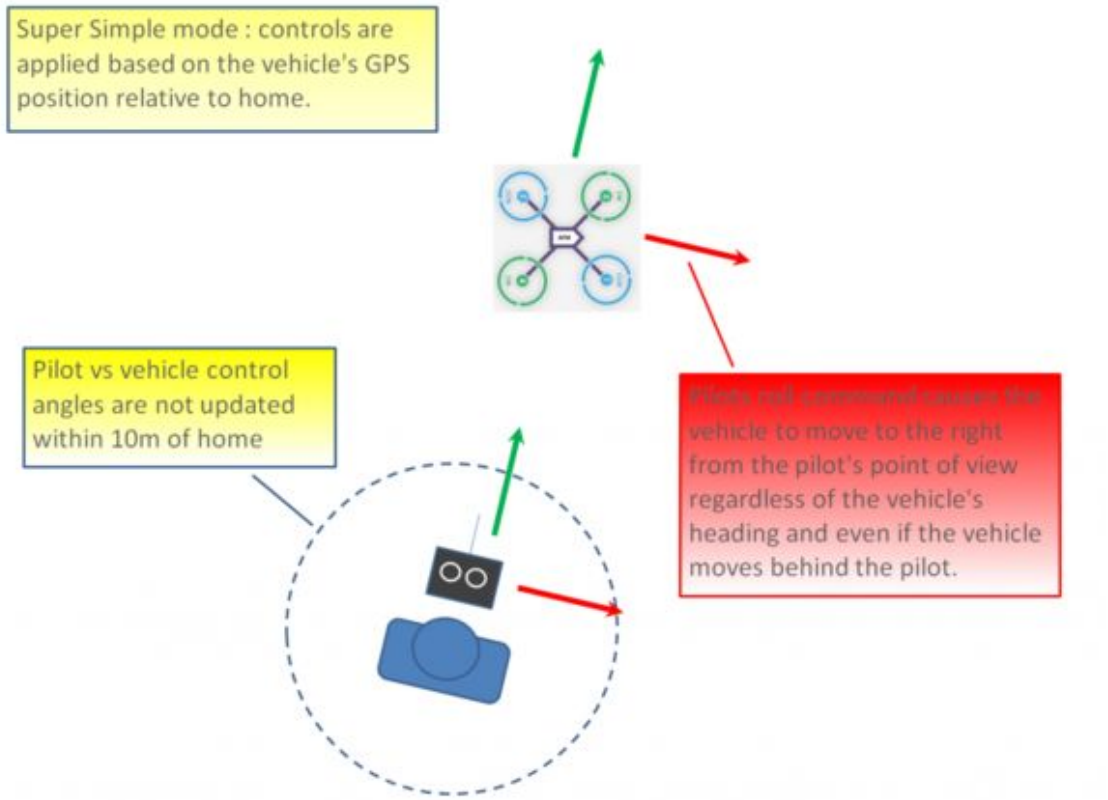
$$L(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi \\ \sin\psi & \cos\psi \end{bmatrix}$$

所以有

$$roll_e = roll_b * \cos\psi_e + pitch_b * \sin\psi_e ;$$

$$pitch_e = -roll_b * \sin\psi_e + pitch_b * \cos\psi_e ;$$

3、超简单模式



超简单模式和简单模式是基本相同的，除了它用的是模型的位置与家的位置相关联，不是用的模型最初解锁时的头的方向。 这就是说不用管模型在哪，也不用管模型朝向哪，只要向后拉 pitch 就会让飞行器朝家的方向飞。
相较简单模式的优点就在于即使飞行器飞到了飞手或是家的后面，飞手还可以用自己的视角来控制。
如果向右拉满 roll，模型就以飞手为中心顺时针绕圈飞（尽管因为“时滞”每圈半径都有可能会增长一点）。
缺点是这个模式需要 GPS 定位，所以你要确保在起飞之前 GPS 已经定位。
模型在家 10m 以内时，方向是不会更新的，所以要避免在家附近飞。
在起飞时要确保控制是正确的，和简单模式一样，你应该在解锁时站在模型后面，飞手和模型所朝方向也应是一样的。

总结：超简单模式，又名无头模式超简单模式的投影公式同简单模式，只是旋转的角度不一致而已。

4、代码分析

这三种模式的代码在 arducopter.cpp 中 update_simple_mode 函数

```
void Copter::update_simple_mode(void)
{
    .....
    if (ap.simple_mode == 0 || !ap.new_radio_frame) {
        return; //没有配置简单模式就是正常模式，直接退出
    }
```

```
.....
 if (ap.simple_mode == 1) {
     //飞机的头部始终执行起飞时所执行的方向:简单模式
      // rotate roll, pitch input by -initial simple heading (i.e. north facing)
      rollx = channel_roll->control_in*simple_cos_yaw -
                        channel_pitch->control_in*simple_sin_yaw;
      pitchx = channel_roll->control_in*simple_sin_yaw +
                        channel_pitch->control_in*simple_cos_yaw;
 }else{
     //无头模式：超简单模式
      rollx = channel_roll->control_in*super_simple_cos_yaw -
                          channel_pitch->control_in*super_simple_sin_yaw;
      pitchx = channel_roll->control_in*super_simple_sin_yaw +
                          channel_pitch->control_in*super_simple_cos_yaw;
 }

//通过 dcm 矩阵，把 roll pitch 的值转到地球坐标系统
 channel_roll->control_in = rollx*ahrs.cos_yaw() + pitchx*ahrs.sin_yaw();
 channel_pitch->control_in = -rollx*ahrs.sin_yaw() + pitchx*ahrs.cos_yaw();
}
```

第三节 22 状态 EKF 滤波

（声明：该节原创乔智杰）

状态方程

状态选取

姿态四元数 $\begin{bmatrix} q_0 & q_1 & q_2 & q_3 \end{bmatrix}^T$

NED 速度 $\begin{bmatrix} V_n & V_e & V_d \end{bmatrix}^T$

NED 位置 $\begin{bmatrix} p_n & p_e & p_d \end{bmatrix}^T$

姿态角增量偏差 $\begin{bmatrix} \delta\phi_{bias} & \delta\theta_{bias} & \delta\psi_{bias} \end{bmatrix}^T$

速度增量偏差 δV_{zbias}

NE 方向风速 $\begin{bmatrix} V_{wind|n} & V_{wind|e} \end{bmatrix}^T$

NED 方向磁场强度 $\begin{bmatrix} m_n & m_e & m_d \end{bmatrix}^T$

BODY 坐标系中的磁场强度 $\begin{bmatrix} m_x & m_y & m_z \end{bmatrix}^T$

注意，下面为两组时变参数，而非观测量

体坐标系中测量的姿态角增量$\begin{bmatrix} \delta\phi & \delta\theta & \delta\psi \end{bmatrix}^T$

体坐标系中测量的速度增量$\begin{bmatrix} \delta V_x & \delta V_y & \delta V_z \end{bmatrix}^T$

姿态四元数更新 $\begin{bmatrix} q_0 & q_1 & q_2 & q_3 \end{bmatrix}^T$

$$\begin{bmatrix} q_0(k+1) \\ q_1(k+1) \\ q_2(k+1) \\ q_3(k+1) \end{bmatrix} = \begin{bmatrix} q_0(k) \\ q_1(k) \\ q_2(k) \\ q_3(k) \end{bmatrix} + \frac{1}{2}\Omega(q)\omega \cdot dt$$

其中，$\Omega(q) = \begin{bmatrix} -q_1 & -q_2 & -q_3 \\ q_0 & -q_3 & q_2 \\ q_3 & q_0 & -q_1 \\ -q_2 & q_1 & q_0 \end{bmatrix}$，$\begin{bmatrix} \delta\phi \\ \delta\theta \\ \delta\psi \end{bmatrix} = \begin{bmatrix} \delta\phi_{measure} \\ \delta\theta_{measure} \\ \delta\psi_{measure} \end{bmatrix} - \begin{bmatrix} \delta\phi_{bias} \\ \delta\theta_{bias} \\ \delta\psi_{bias} \end{bmatrix}$

速度更新

$$\begin{bmatrix} V_n(k+1) \\ V_e(k+1) \\ V_d(k+1) \end{bmatrix} = \begin{bmatrix} V_n(k) \\ V_e(k) \\ V_d(k) \end{bmatrix} + \begin{bmatrix} g_n(k) \\ g_e(k) \\ g_d(k) \end{bmatrix} \cdot dt + R_{B\to W}\delta V$$

其中，$\delta V = \begin{bmatrix} \delta V_x \\ \delta V_y \\ \delta V_z \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ \delta V_{zbias} \end{bmatrix}$，注意$\begin{bmatrix} \delta V_x & \delta V_y & \delta V_z \end{bmatrix}^T$是时变参数，$\delta V_{zbias}$是状态变量。

位置更新

$$\begin{bmatrix} p_n(k+1) \\ p_e(k+1) \\ p_d(k+1) \end{bmatrix} = \begin{bmatrix} p_n(k) \\ p_e(k) \\ p_d(k) \end{bmatrix} + \begin{bmatrix} V_n(k) \\ V_e(k) \\ V_d(k) \end{bmatrix} \cdot dt$$

姿态角偏差更新

$$\begin{bmatrix} \delta\phi(k+1) \\ \delta\theta(k+1) \\ \delta\psi(k+1) \end{bmatrix} = \begin{bmatrix} \delta\phi(k) \\ \delta\theta(k) \\ \delta\psi(k) \end{bmatrix}$$

速度偏差更新

$$\delta V_z(k+1) = \delta V_z(k)$$

NED 速度更新

$$\begin{bmatrix} V_n(k+1) \\ V_e(k+1) \end{bmatrix} = \begin{bmatrix} V_n(k) \\ V_n(k) \end{bmatrix}$$

NED 方向磁场强度更新

$$\begin{bmatrix} m_n(k+1) \\ m_e(k+1) \\ m_d(k+1) \end{bmatrix} = \begin{bmatrix} m_n(k) \\ m_e(k) \\ m_d(k) \end{bmatrix}$$

BODY 坐标系中的磁场强度更新

$$\begin{bmatrix} m_x(k+1) \\ m_y(k+1) \\ m_z(k+1) \end{bmatrix} = \begin{bmatrix} m_x(k) \\ m_y(k) \\ m_z(k) \end{bmatrix}$$

观测量

NED 速度$\begin{bmatrix} V_n & V_e & V_d \end{bmatrix}^T$

NED 位置 $\begin{bmatrix} p_n & p_e & p_d \end{bmatrix}^T$

空速 *airspeed*

BODY 坐标系中的磁场强度 $\begin{bmatrix} m_x & m_y & m_z \end{bmatrix}^T$

光流传感器测得的视线角速率 $\begin{bmatrix} \omega_x^{los} & \omega_y^{los} \end{bmatrix}^T$

到地面的距离 *range*

空速的输出方程

$$airspeed = \sqrt{\left(V_n - V_{wn}\right)^2 + \left(V_e - V_{we}\right)^2 + V_d^2}$$

BODY 坐标系中的磁场强度的输出方程

$$\begin{bmatrix} m_x \\ m_y \\ m_z \end{bmatrix} = R_{W\to B} \begin{bmatrix} m_n \\ m_e \\ m_d \end{bmatrix}$$

到地面的距离 *range* 的输出方程

$$range = \frac{p_{td} - p_d}{R_{B\to W}}$$

其中， p_{td} 为地形在当地 NED 坐标系 D 向的坐标值， p_d 为飞行器在当地 NED 坐标系 D 向的坐标值。

光流传感器测得的视线角速率 $\begin{bmatrix} \omega_x^{los} & \omega_y^{los} \end{bmatrix}^T$ 的输出方程

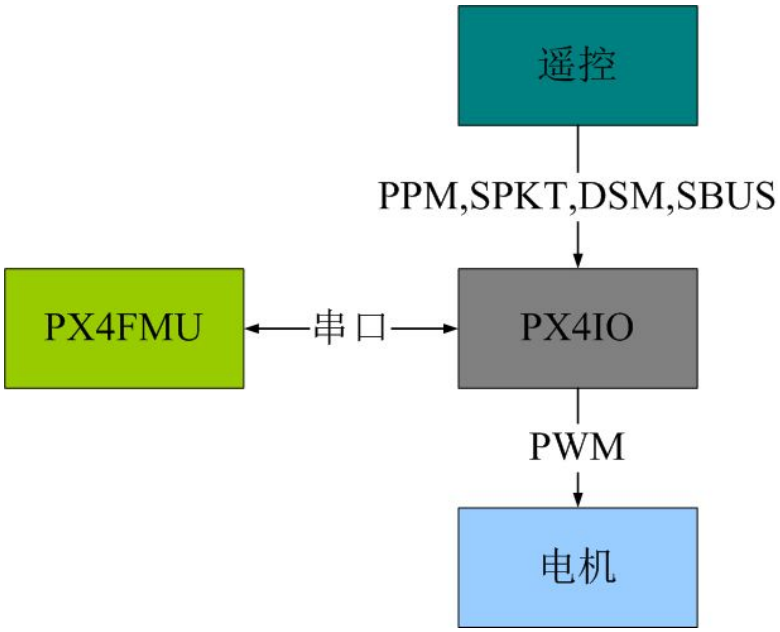
$$\begin{bmatrix} \omega_x^{los} \\ \omega_y^{los} \end{bmatrix} = \frac{1}{rang} \begin{bmatrix} V_{ry} \\ -V_{rx} \end{bmatrix}$$

其中， $\begin{bmatrix} V_{rx} \\ V_{ry} \\ V_{rz} \end{bmatrix} = R_{B\to W} \begin{bmatrix} V_n \\ V_e \\ V_d \end{bmatrix}$

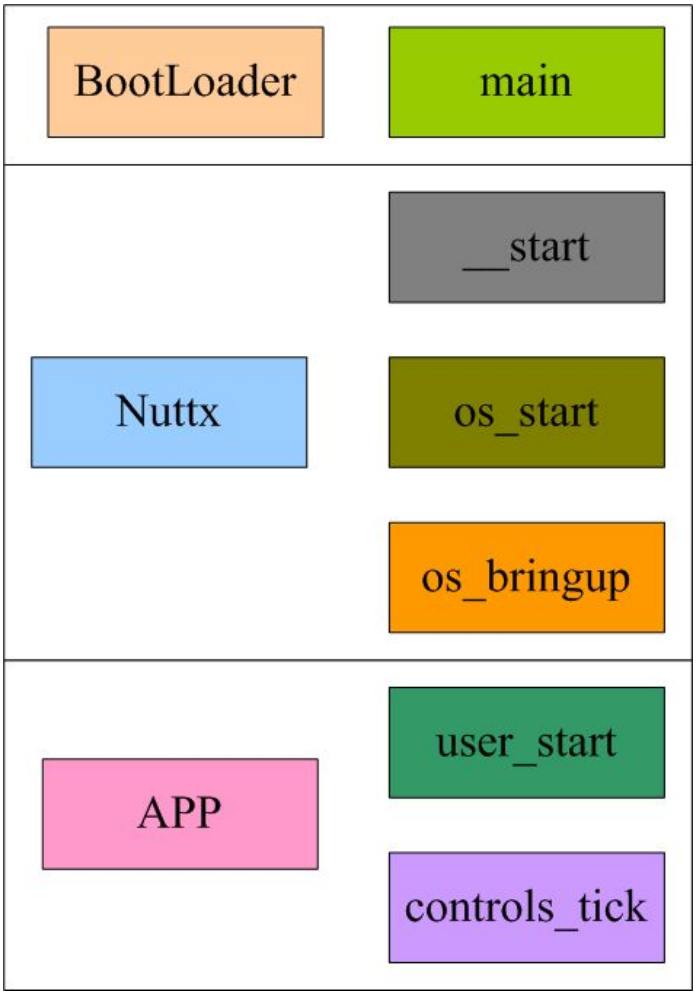
第四节 Pxio

（声明：本节原创吕元宙）

PX4IO(STM32F103)为 PIXHAWK 中专用于处理输入输出的部分,输入为支持的各类遥控器(PPM,SPKT/DSM,SBUS),输出为电调的 PWM 驱动信号,它与 PX4FMU(STM32F427)通过串口进行通信,概括图如下:



PX4IO 与 PX4FMU 一样,在大的软件架构上分为 Bootloader,OS 和 APP 三个部分,OS 为 Nuttx,如下图

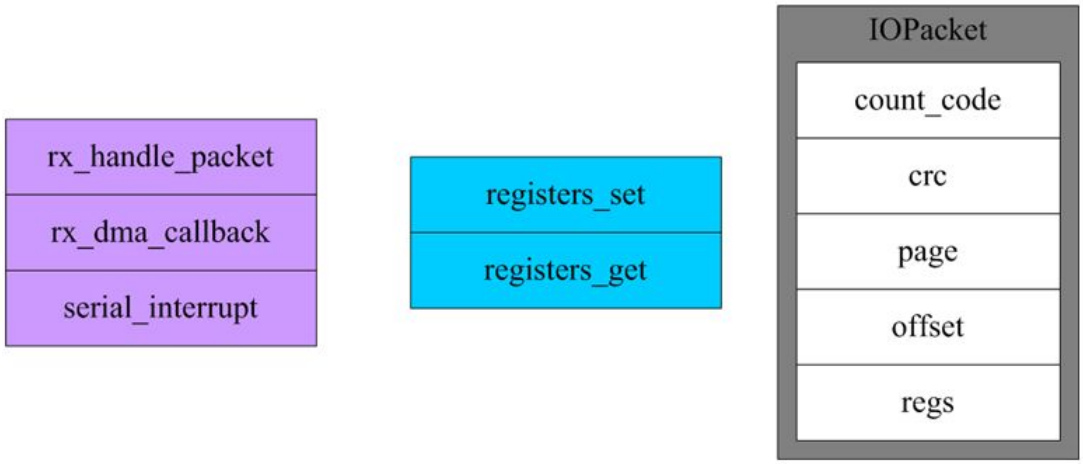


其中:

1. main 负责是否升级芯片,不升级则进入 OS

2. __start 负责 STM32 芯片的底层初始化,包括时钟,FPU,GPIO 等单元的初始化
3. os_start 负责 OS 的底层初始化,包括各种队列和进程结构的初始化
4. os_bringup 负责 OS 基本进程的启动以及用户进程的启动,用户启动入口由(CONFIG_USER_ENTRYPOINT)宏进行指定
5. user_start 负责 px4io 基础环境的初始化,包括 PWM,串口,ADC 等资源的初始化,最后运行一个死循环,用于处理遥控器输入,与 PX4FMU 通信的内容
6. controls_tick 负责处理遥控器的输入内容,包括
   - SBUS 的处理 sbus_input
   - SPKT/DSM 的处理 dsm_port_input
   - PPM 的处理 ppm_input

PX4IO 底层处理的内容如下图



1. 紫色为 PX4IO 的底层串口 IO 操作,流程为当 PX4IO 收到 PX4FMU 的串口数据后会运行 serial_interrupt, serial_interrupt 负责收发 DMA 的操作,如果收到一个完整的包,则调用 rx_dma_callback 进行处理, rx_dma_callback 首先调用 rx_handle_packet 解析包中的内容,判断为写寄存器还是读寄存器,处理完成后由 rx_dma_callback 发送回包给 PX4FMU
2. 蓝色为包操作,只提供 registers_set 写操作和 registers_get 读操作
3. IOPacket 为协议包,包括以下几部分
   - count_code 标记包的读写,错误,长度等信息
   - crc 为包的效验码
   - page 为数据页
   - offset 为数据偏移量
   - regs 为数据内容

数据表格如下:

PAGE	定义	描述
0	PX4IO_PAGE_CONFIG	配置信息,包含协议版本,硬件版本,最大 RC 数量,最大 ADC 数量等信息
1	PX4IO_PAGE_STATUS	状态信息,包含 PPM,SBUS 等通道是否有效等信息
2	PX4IO_PAGE_ACTUATORS	

3	PX4IO_PAGE_SERVOS	
4	PX4IO_PAGE_RAW_RC_INPUT	RC 输入状态,包括信号强度等
5	PX4IO_PAGE_RC_INPUT	RC 输入值
6	PX4IO_PAGE_RAW_ADC_INPUT	ADC 采样值
7	PX4IO_PAGE_PWM_INFO	
50	PX4IO_PAGE_SETUP	
51	PX4IO_PAGE_CONTROLS	
52	PX4IO_PAGE_MIXERLOAD	
53	PX4IO_PAGE_RC_CONFIG	RC 配置信息,例如最大值,最小值,死区等
54	PX4IO_PAGE_DIRECT_PWM	
55	PX4IO_PAGE_FAILSAFE_PWM	
56	PX4IO_PAGE_SENSORS	
57	PX4IO_PAGE_TEST	
106	PX4IO_PAGE_CONTROL_MIN_PWM	
107	PX4IO_PAGE_CONTROL_MAX_PWM	
108	PX4IO_PAGE_DISARMED_PWM	

第五节 kalman 滤波

（本章内容来源与博客 lizipl 的专栏）

1、应用前提

要应用 kalman Filter,首先要有三个前提假设：

当前状态的概率分布必须是上一状态和将要执行的控制量的线性函数，再叠加一个高斯噪声。表达式如下：

$$x_t = A_t x_{t-1} + B_t u_t + \varepsilon_t .$$

其中 x_t 和 x_{t-1} 是状态变量，如果系统有多个自由度的话，它们表示状态向量。这样

的话根据高斯分布

$$p(x) = \det(2\pi\Sigma)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right\}$$

可以得到状态转移概率

$$p(x_t \mid u_t, x_{t-1})$$

$$= \det(2\pi R_t)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(x_t - A_t x_{t-1} - B_t u_t)^T R_t^{-1}(x_t - A_t x_{t-1} - B_t u_t)\right\}$$

其中$A_t x_{t-1} + B_t u_t$表示上一状态的均值，R_t表示方差。

对状态的测量必须是状态的线性函数叠加高斯噪声

$$z_t = C_t x_t + \delta_t$$

$$p(z_t \mid x_t) = \det(2\pi Q_t)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(z_t - C_t\, x_t)^T\, Q_t^{-1}\, (z_t - C_t\, x_t)\right\}$$

其含义与上类似，不再赘述。

初始状态分布为高斯分布

$$bel(x_0) = p(x_0) = \det(2\pi\Sigma_0)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}\,(x_0-\mu_0)^T\Sigma_0^{-1}(x_0-\mu_0)\right\}$$

2、算法想想介绍

Kalman Filter 五条黄金公式：

Algorithm Kalman_filter($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$):

$\bar{\mu}_t = A_t\, \mu_{t-1} + B_t\, u_t$

$\bar{\Sigma}_t = A_t\, \Sigma_{t-1}\, A_t^T + R_t$

$K_t = \bar{\Sigma}_t\, C_t^T (C_t\, \bar{\Sigma}_t\, C_t^T + Q_t)^{-1}$

$\mu_t = \bar{\mu}_t + K_t(z_t - C_t\, \bar{\mu}_t)$

$\Sigma_t = (I - K_t\, C_t)\, \bar{\Sigma}_t$

return μ_t, Σ_t

这五条公式基本上就是 Kalman Filter 的主要内容了，它的本质就是通过预测结合测量来估计当前系统的状态。举个 lizi，假如我们要估计一架飞行器的姿态，可以通过 IMU

来实时测量，但是测量值有一定的风险是不准确的，所以并不能完全依赖传感器。任何一个满足物理规律的系统应当是连续的，所以我们还可以通过上一状态来预测当前状态。Kalman Filter 正是结合这两条进行状态估计，到底是相信哪一个多一点，还要根据 Kt 来决定，我们定义 Kt 为卡尔曼增益，它是根据 测量和预测的协方差来计算的。

下面逐条解释：

line 2： 首先通过上一状态最优值和将要施加的控制量来预测当前状态，由假设一可以得到：

$$\bar{\mu}_t = A_t\ \mu_{t-1} + B_t\ u_t$$

因为我们只是求均值，而高斯噪声均值为 0，所以可省去最后一项。

line 3： 除了预测均值之外，我们还需要预测值的协方差来计算 Kalman 增益。

$$\bar{\Sigma}_t = A_t\ \Sigma_{t-1}\ A_t^T + R_t$$

根据假设 2，这条公式可以很容易得到。

*line 4：*准备工作完成之后，需要根据预测值的协方差 $\bar{\Sigma}_t$，测量值和状态

的比例系数，测量值的协方差来计算 Kalman 增益。

$$K_t = \bar{\Sigma}_t\ C_t^T (C_t\ \bar{\Sigma}_t\ C_t^T + Q_t)^{-1}$$

具体证明需要用到假设中的高斯分布公式，因为我们只是应用，所以就不在 blog 中讨论啦，感兴趣的小伙伴可以看书中 3.2.4 节 Mathematical Derivation of the KF，里面讲的很详细，分享一下下载链接

http://download.csdn.net/detail/lizilpl/8632071。

*line 5：*这一行可以说是 Kalman Filter 的精华了，现在我们有了对状态的预测值和协方差，同时也收集到了对状态的测量值。这时就可以通过 kalman 增益来计算状态估计值了。

$$\mu_t = \bar{\mu}_t + K_t(z_t - C_t\,\bar{\mu}_t)$$

增益越大，表明我们越相信测量值。

line 6： 根据 *line3*，预测当前状态需要用到上一状态的协方差，所以我们还需要计算当前状态的协方差用于下一次迭代。它同样要根据 Kalman 增益来计算：

$$\Sigma_t = (I - K_t\,C_t)\,\bar{\Sigma}_t$$

相信到这里，大家应该对 kalman Filter 的原理有了一个大致的了解，算法中，从初始状态开始，不断计算当前状态的均值和方差来迭代，直至系统结束。

3、扩展 kalman 的应用前提

与 kalman Filter 只能应用于线性系统不同,Extended Kalman Filter 可以用于非线性系统中。：

当前状态的概率分布是关于上一状态和将要执行的控制量的二元函数，再叠加一个高斯噪声，测量值同样是关于当前状态的函数叠加高斯噪声。具体表达式如下：

$$\begin{aligned} x_t &= g(u_t, x_{t-1}) + \varepsilon_t \\ z_t &= h(x_t) + \delta_t\ . \end{aligned}$$

$g(u_t, x_{t-1})$ 和 $h(x_t)$ 可以是非线性的函数。

为了用经典卡尔曼滤波器的思想来解决非线性系统中的状态估计问题，首先要做的就是把 $g(u_t, x_{t-1})$ 和 $h(x_t)$ 用泰勒级数展开，将其线性化，只取一次项为一阶 EKF 滤波。具体如下：

$$
\begin{aligned}
g(u_t, x_{t-1}) &\approx g(u_t, \mu_{t-1}) + \underbrace{g'(u_t, \mu_{t-1})}_{=: G_t} (x_{t-1} - \mu_{t-1}) \\
&= g(u_t, \mu_{t-1}) + G_t \, (x_{t-1} - \mu_{t-1})
\end{aligned}
$$

$$
\begin{aligned}
h(x_t) &\approx h(\bar{\mu}_t) + \underbrace{h'(\bar{\mu}_t)}_{=: H_t} (x_t - \bar{\mu}_t) \\
&= h(\bar{\mu}_t) + H_t \, (x_t - \bar{\mu}_t)
\end{aligned}
$$

$g(u_t, x_{t-1})$ 在上一状态估计的最优值处取一阶导数，$h(x_t)$ 在当前时刻预测值处取一阶导数，得到 G 和 H 分别相当于 Kalman Filter 中的 A 和 C。

4、ekf 算法详细介绍

Extended Kalman Filter 五条黄金公式：

1: **Algorithm Extended_Kalman_filter**($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$)**:**
2: $\bar{\mu}_t = g(u_t, \mu_{t-1})$
3: $\bar{\Sigma}_t = G_t \, \Sigma_{t-1} \, G_t^T + R_t$
4: $K_t = \bar{\Sigma}_t \, H_t^T (H_t \, \bar{\Sigma}_t \, H_t^T + Q_t)^{-1}$
5: $\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$
6: $\Sigma_t = (I - K_t \, H_t) \, \bar{\Sigma}_t$
7: return μ_t, Σ_t

将其线性化之后，EFK 与 FK 就非常类似了，所以我就不再赘述，可参考上一篇 blog：卡尔曼滤波器的原理及应用 http://blog.csdn.net/lizilpl/article/details/45268471 里面对这五条公式有较为详细的介绍。

第三章 px4 各种模式的分析

第一节 悬停模式

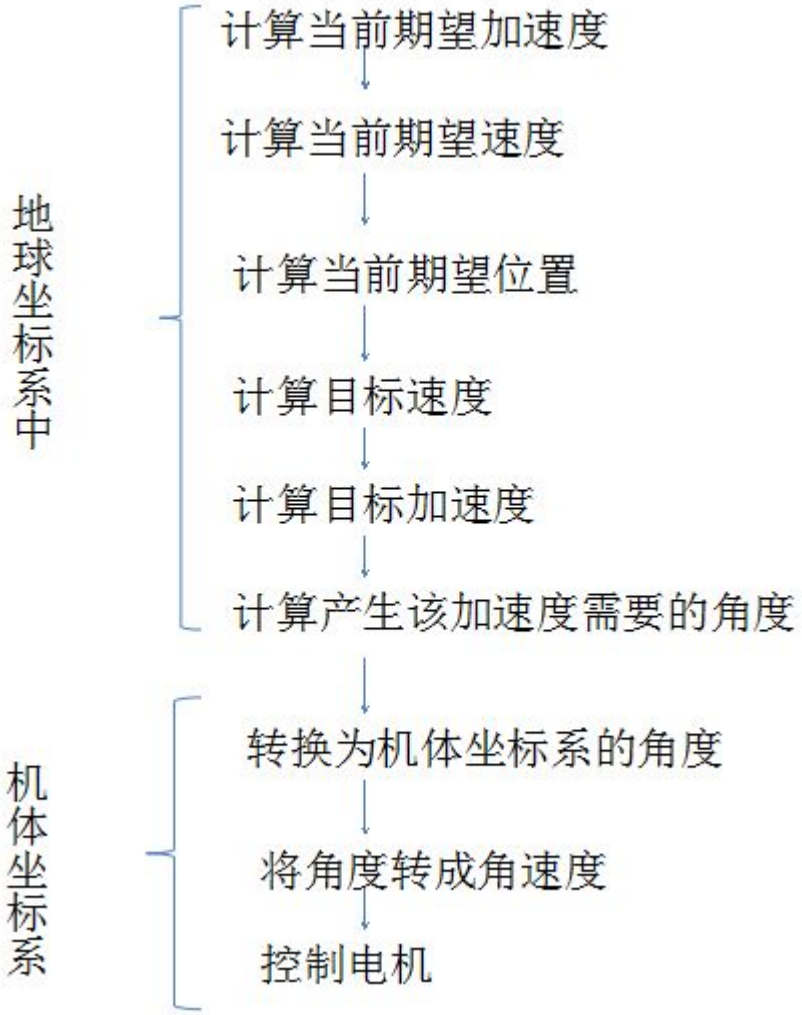
1、悬停代码流程

悬停的代码在 control_loiter.cpp 中的 loiter_run()函数

```
void Copter::loiter_run()
{
    .......
    /*
    悬停的时候由于没有俯仰和横滚的遥控值，所以向前和向右的期望加速度都
    为 0
    */
    wp_nav.clear_pilot_desired_acceleration();
    .......
    //这里设置了目的位置为当前位置
    wp_nav.loiter_soften_for_landing();
    ......
    //设置为悬停模式
    loiter_state = Loiter_Flying;
    .......
    switch (loiter_state) {
        ....
        case Loiter_Flying://悬停
        wp_nav.update_loiter(ekfGndSpdLimit, ekfNavVelGainScaler);
        attitude_control.angle_ef_roll_pitch_rate_ef_yaw(wp_nav.get_roll(),
                                        wp_nav.get_pitch(), target_yaw_rate);
        if (sonar_enabled && (sonar_alt_health >= SONAR_ALT_HEALTH_MAX))
        {
            arget_climb_rate = get_surface_tracking_climb_rate(target_climb_rate,
                                    pos_control.get_alt_target(), G_Dt);
         }
         pos_control.set_alt_target_from_climb_rate(target_climb_rate,
                                    G_Dt, false);
         pos_control.update_z_controller();//定高
         break;
        ......
    }
}
```

2、悬停 xy 方向控制思路

悬停代码流程图



3、加速度的初始化

原理分析：

设飞机可提供的最大速度为$V_{\max}$，最大加速为$a_{\max}$；飞机从速度为 0 增加到最大速度需要的时间为T_s，

则: $T_s = \frac{v_{\max}}{a_{\max}}$;

设当最近一次从传感器中获取的速度（通过 ekf 滤波）为$\vec{v_1}$，在 x 和 y 轴的分量分别为$\vec{v_{(1).x}}$和$\vec{v_{(1).y}}$，假设速度是从 0 在T_s时间内增加到$\vec{v_1}$，设此时的加速度为$\vec{a_1}$，分在 x 和 y 轴的加速度分别为$\vec{a_{(1).x}}$和$\vec{a_{(1).y}}$，

那么， $\vec{a}_{(1).x} = \frac{\vec{v}_{(1).x}}{T_s} = \frac{v_{(1).x} * \vec{a}_{\max}}{v_{\max}}$，

$\vec{a}_{(1).y} = \frac{\vec{v}_{(1).y}}{T_s} = \frac{v_{(1).y} * \vec{a}_{\max}}{v_{\max}}$；

代码如下：

```
void AC_WPNav::init_loiter_target()
{
    //读取从 gps 中获取当前位置和速度
    const Vector3f& curr_pos = _inav.get_position();
    const Vector3f& curr_vel = _inav.get_velocity();
    /*
    *设置悬停时的最大水平速度为_loiter_speed_cms，以及计算_leash（漂移控
    *制距离）值
    */
     _pos_control.set_speed_xy(_loiter_speed_cms);
    /*
    *设置悬停时的最大水平加速度为_loiter_accel_cmss，以及计算_leash（漂移
    *控制距离）值
    */
    _pos_control.set_accel_xy(_loiter_accel_cmss);
    //设置目标位置为当前位置
    _pos_control.set_xy_target(curr_pos.x, curr_pos.y);
    //设置悬停时的期望速度为当前速度
    _pos_control.set_desired_velocity_xy(curr_vel.x, curr_vel.y);
    /*
    *设置悬停时的期望加速度，在地球坐标系中
    */
    _loiter_desired_accel.x =
                        (_loiter_accel_cmss)*curr_vel.x/_loiter_speed_cms;
    _loiter_desired_accel.y =
              (_loiter_accel_cmss)*curr_vel.y/_loiter_speed_cms;
    /*
    *由于没有 roll、pitch 遥感值，所以
    */
    _pilot_accel_fwd_cms = 0;//前后加速度为 0
    _pilot_accel_rgt_cms = 0;//左右加速为 0
}
```

在悬停第一次初始化了加速度后，整个过程中都没有再次复位加速度。

4、计算悬停时的期望速度

在每次执行 update_flight_mode()代码之前都执行 read_inertia()到 gps 传感器中读取当前速度和位置。

原理分析：设上一次（n-1）次悬停的期望加速度为$\vec{a}_{(n-1)loiter}$，roll 和 pitch 传入的期望加速度为$\vec{a}_{(n)\text{desire}}$，由于在悬停是 roll 和 pitch 都为 0,

所以有：　　$\vec{a}_{(n)\text{desire}}$=0；

设，第 n 次期望加速度和（n-1）的加速度的差值为$\vec{a}_{dif}$，

则：　　$\vec{a}_{dif}=\vec{a}_{(n)desire}-\vec{a}_{(n-1)loiter}$；

设，第（n-1）到当前的时间d_t，加速度的变化率为d_a（默认配置为 1000），那么在d_t的时间内，加速度的最大变化量为a_{change}，

$$a_{\text{change}}=d_a*d_t;$$

我们这里需要判断$|\vec{a}_{dif}|$与a_{change}的大小，如果$|\vec{a}_{dif}|>\ a_{\text{change}}$，则表示在$d_t$时间内加速度不可能增加到$|\vec{a}_{dif}|$大，需要让$|\vec{a}_{dif}|=\ a_{\text{change}}$。

设当前悬停的期望加速度为$\vec{a}_{(n)loiter}$，

$$\vec{a}_{(n)loiter}=\vec{a}_{(n-1)loiter}+\vec{a}_{dif};$$

知道了当前悬停是期望的加速度，可以求出当前悬停期望的速度，设当前期望的速度为$\vec{v}_n$，n-1 次的实际速度可以从 gps 中获取到计为$\vec{v}_{(n-1)true}$，

则，　　$\vec{v}_{n.x}=\vec{v}_{(n-1)true.x}+\vec{a}_{(n)loiter.x}*d_t$，

$$\vec{v}_{n.y}=\vec{v}_{(n-1)true.y}+\vec{a}_{(n)loiter.y}*d_t;$$

考虑飞机本身的阻力，阻力系数为ρ，该系数的大小为最大加速度在d_t内累积的速度飞机相对地面最大速度的比值（暂不知道为何如此）

$$\rho=\frac{a_{\max}*d_t}{v_{gnd}};$$

那么，$|\vec{v}_n|=|\vec{v}_n|*(1-\rho)$.

期望速度不能大于相对于地面的最大速度，如果$|\vec{v}_n|>\ v_{gnd}$,则$|\vec{v}_n|=\ v_{gnd}$。

代码如下：

```
void AC_WPNav::calc_loiter_desired_velocity(float nav_dt, float ekfGndSpdLimit)
{
    //求飞机相对地面的最大速度
    float gnd_speed_limit_cms = min(_loiter_speed_cms,ekfGndSpdLimit*100.0f);
    gnd_speed_limit_cms = max(gnd_speed_limit_cms, 10.0f);
    .....
    /*
    *求当前期望加速度，由于在悬停时 roll 和 pitch 的值为 0，所有期望加速度
    *也为 0
    */
    Vector2f desired_accel;
    desired_accel.x = (_pilot_accel_fwd_cms*_ahrs.cos_yaw() -
                       _pilot_accel_rgt_cms*_ahrs.sin_yaw());//=0
    desired_accel.y = (_pilot_accel_fwd_cms*_ahrs.sin_yaw() +
                       _pilot_accel_rgt_cms*_ahrs.cos_yaw());//=0
    //求第 n 期望加速度和 n-1 期望加速度的差值
    Vector2f des_accel_diff = (desired_accel - _loiter_desired_accel);
    /*
    *判断在 dt 时间内是否可以增加到需要的加速度，如果不可以，同最大可改
    *变的加速度为期望加速度的差值
    */
    float des_accel_change_total = pythagorous2(des_accel_diff.x, des_accel_diff.y);
    float accel_change_max = _loiter_jerk_max_cmsss * nav_dt;
    if (_loiter_jerk_max_cmsss > 0.0f
            && des_accel_change_total > accel_change_max
            && des_accel_change_total > 0.0f) {
        des_accel_diff.x = accel_change_max *
                           des_accel_diff.x/des_accel_change_total;
        des_accel_diff.y = accel_change_max *
                                           des_accel_diff.y/des_accel_change_total;
     }
    //设置当前的期望加速度
    _loiter_desired_accel += des_accel_diff;

    //获取从传感器读出来的速度
    const Vector3f &desired_vel_3d = _pos_control.get_desired_velocity();
    Vector2f desired_vel(desired_vel_3d.x,desired_vel_3d.y);

    //求出第 n 次的期望速度
    desired_vel.x += _loiter_desired_accel.x * nav_dt;
    desired_vel.y += _loiter_desired_accel.y * nav_dt;
```

```
    //考虑阻力
    float desired_speed = desired_vel.length();

    if (!is_zero(desired_speed)) {
       Vector2f desired_vel_norm = desired_vel/desired_speed;
       float drag_speed_delta = -_loiter_accel_cmss*nav_dt*desired_speed
                                                    /gnd_speed_limit_cms;

       if (_pilot_accel_fwd_cms == 0 && _pilot_accel_rgt_cms == 0) {
            drag_speed_delta = min(drag_speed_delta,
                                                    -_loiter_accel_min_cmss*nav_dt);
       }

       desired_speed = max(desired_speed+drag_speed_delta,0.0f);
       desired_vel = desired_vel_norm*desired_speed;
    }
    //如果期望速度大于对地最大速度，就设期望速度为对地速度
    float horizSpdDem = sqrtf(sq(desired_vel.x) + sq(desired_vel.y));
        if (horizSpdDem > gnd_speed_limit_cms) {
              desired_vel.x = desired_vel.x * gnd_speed_limit_cms / horizSpdDem;
              desired_vel.y = desired_vel.y * gnd_speed_limit_cms / horizSpdDem;
    }
    //设置本次悬停是需要的速度供下面的代码使用
    _pos_control.set_desired_velocity_xy(desired_vel.x,desired_vel.y);
}
```

5、通过期望速度计算期望位置

```
void AC_PosControl::desired_vel_to_pos(float nav_dt)
{
    .....
    _pos_target.x += _vel_desired.x * nav_dt;
    _pos_target.y += _vel_desired.y * nav_dt;
    .....
}
```

6、设置每次移动的最大距离

原理分析：从一个点运动到另一个点，有两种方式，一种是是通过匀速直线运动；另一种是先加速后减速。设飞机的最大速度为v_{max}，最大加速度为a_{max}

匀加速运动：$S_1 = v_{max}t$；

先加速后减速：$S_2 = \frac{a_{max}t^2}{2} + \frac{v_{max}^2}{2a_{max}}$;

假设 $S_1 = S_2$;

通过 $v_{\max} t = \frac{a_{\max} t^2}{2} + \frac{v_{\max}^2}{2a_{\max}}$，可得 $(v_{\max} - a_{\max} t)^2 = 0$；我们设 $kp = \frac{1}{t}$;

当 $v_{\max} <= a_{\max} * \frac{1}{kp}$，则 $S_1 >= S_2$；

当 $v_{\max} > a_{\max} * \frac{1}{kp}$，则 $S_1 < S_2$；

这里我有点搞不懂，同样的距离 S_2 所用的时间比 S_1 多了一倍，为啥默认的要选择用公式 S_2。用户可以通过调试 kp 着参数来控制每次移动的最大距离。

代码如下

```
float AC_PosControl::calc_leash_length(float speed_cms, float accel_cms, float kP) const
{
    .....
    if(speed_cms <= accel_cms / kP) {
            leash_length = speed_cms / kP;
        }else{
            leash_length = (accel_cms / (2.0f*kP*kP)) +
                    (speed_cms*speed_cms / (2.0f*accel_cms));
        }
    .......
}
```

7、计算出从当前位置到期望位置需要的目标速度

```
void AC_PosControl::pos_to_rate_xy(xy_mode mode, float dt, float ekfNavVelGainScaler)
{
    //从 gps 中获得当前的位置，通过目标位置与当前位置差距可以求出目标速
    //度
    Vector3f curr_pos = _inav.get_position();

    // S = a_max t^2 / 2 这个公式,求前半段加速的长度

    linear_distance = _accel_cms/(2.0f*kP*kP);
    //如果期望位置与时间位置的差大于二倍的 linear_distance，则用先加速后减
```

//速的方式 $S_2 = \frac{a_{\max} t^2}{2} + \frac{v_{\max}^2}{2a_{\max}}$，变形为 $S_2 - \frac{a_{\max} t^2}{2} = \frac{v_t^2}{2a_{\max}}$ 用这个公式求出需

//要的速度 v_t;否则就通过匀速直线运动 $v_t = \frac{S}{t} = S * kp$

```
    if (_distance_to_target > 2.0f*linear_distance) {
                float vel_sqrt = safe_sqrt(2.0f*_accel_cms*
                                           (_distance_to_target-linear_distance));
                _vel_target.x = vel_sqrt * _pos_error.x/_distance_to_target;
                _vel_target.y = vel_sqrt * _pos_error.y/_distance_to_target;
          }else{
                _vel_target.x = _p_pos_xy.kP() * _pos_error.x;
                _vel_target.y = _p_pos_xy.kP() * _pos_error.y;
       }
    //最终的目标速度还要加上之前计算的期望速度
          _vel_target.x += _vel_desired.x;
          _vel_target.y += _vel_desired.y;
}
```

8、根据目标速度求出目标加速度

```
void AC_PosControl::rate_to_accel_xy(float dt, float ekfNavVelGainScaler)
{
    //从 gps 中获得当前的速度
    const Vector3f &vel_curr = _inav.get_velocity();

    //求出从上次计算完目标速度到本次计算产生的加速度
    _accel_feedforward.x = (_vel_target.x - _vel_last.x)/dt;
    _accel_feedforward.y = (_vel_target.y - _vel_last.y)/dt;

    //计算目标速度与现在实际速度的差速度
    _vel_error.x = _vel_target.x - vel_curr.x;
    _vel_error.y = _vel_target.y - vel_curr.y;

    //  更新和获得 pi 的速度
    _pi_vel_xy.set_input(_vel_error);
    vel_xy_p = _pi_vel_xy.get_p();

    if ((!_limit.accel_xy && !_motors.limit.throttle_upper)) {
                vel_xy_i = _pi_vel_xy.get_i();
    } else {
          vel_xy_i = _pi_vel_xy.get_i_shrink();
    }
```

```
    //计算目标加速度（这里的分析放在 pid 中区分析）
    _accel_target.x = _accel_feedforward.x +
                (vel_xy_p.x + vel_xy_i.x) * ekfNavVelGainScaler;
        _accel_target.y = _accel_feedforward.y +
                (vel_xy_p.y + vel_xy_i.y) * ekfNavVelGainScaler;


}
```

9、根据目标加速度求出机体坐标系 xy 轴上的加速度，在求出机体需要的角度

```
void AC_PosControl::accel_to_lean_angles(float dt, float ekfNavVelGainScaler)
{
    //根据上次目标加速度和本次目标加速度，求出差异加速度
    Vector2f accel_in(_accel_target.x, _accel_target.y);
    Vector2f accel_change = accel_in-_accel_target_jerk_limited;
    float accel_change_length = accel_change.length();
    //上次目标加速度加上差异加速度就得到当前目标加速度
    _accel_target_jerk_limited += accel_change;
    //进行滤波处理
    _accel_target_filter.set_cutoff_frequency(min(_accel_xy_filt_hz,
5.0f*ekfNavVelGainScaler));
    Vector2f accel_target_filtered =
            _accel_target_filter.apply(_accel_target_jerk_limited, dt);
    //把加速度分配到机体坐标系的 xy 轴上
    /*
    在上篇 3X3 矩阵操作中，我们得到体坐标系到地球坐标系的绕 Z 轴的旋转
    矩阵是
```

$$L(\psi)\begin{bmatrix}\cos\psi & -\sin\psi & 0\\ \sin\psi & \cos\psi & 0\\ 0 & 0 & 1\end{bmatrix}$$

去掉 Z 轴相关的

$$L(\psi)=\begin{bmatrix}\cos\psi & -\sin\psi\\ \sin\psi & \cos\psi\end{bmatrix}$$

所以有

$roll_e = roll_b * \cos\psi_e + pitch_b * \sin\psi_e$；

$pitch_e = -roll_b * \sin\psi_e + pitch_b * \cos\psi_e$;

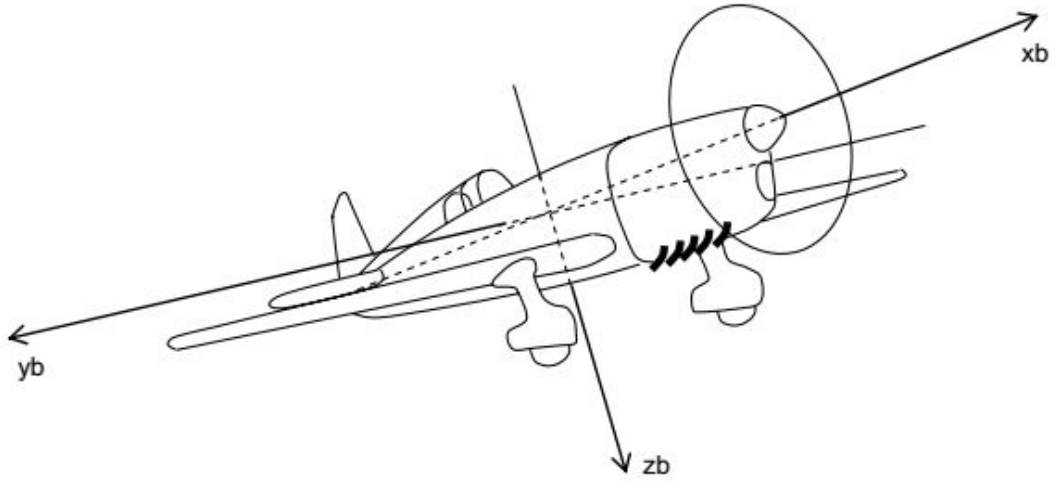
反之，亦然，地球坐标系到机体坐标系

$roll_b = roll_e * \cos\psi_b + pitch_e * \sin\psi_b$；

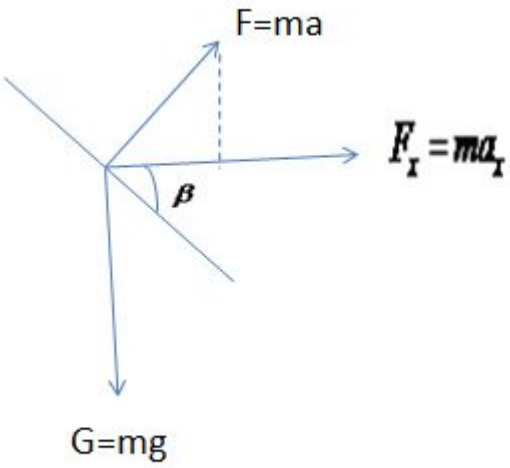
$pitch_b = -roll_e * \sin\psi_b + pitch_e * \cos\psi_b$;

其实，$\psi_b = \psi_e$。

机体方向的定义见图 X 轴为 pitch,Y 轴为 roll




```
*/
accel_forward = accel_target_filtered.x*_ahrs.cos_yaw() +
                        accel_target_filtered.y*_ahrs.sin_yaw();
accel_right = -accel_target_filtered.x*_ahrs.sin_yaw() +
                        accel_target_filtered.y*_ahrs.cos_yaw();
/*
*求出产生需要的加速度，机体相对于地球坐标的角度
```



*飞机需要向前运动，会有一个 F 的力来平衡重力，并且在 X 轴上的分量为向前需要的
*加速度和质量的乘积，β 角为飞行向前的倾斜角度

$$\tan\beta = -\frac{a_x}{g},$$

*则：$\beta = \arctan(-\frac{a_x}{g})$

*在 y 轴方向由于考虑到向 X 的倾斜太大，如果再向 Y 轴倾斜很大的角度，飞机很容易
*侧翻，所以给 Y 的加速度乘了一个 $\cos\beta$ 的分量（只是这么理解的）

$\delta = \arctan(\frac{a_y * \cos\beta}{g})$，这里的方向是由 $\cos\beta$ 来确定的。

```
*/
_pitch_target = constrain_float(atanf(-accel_forward/(GRAVITY_MSS *
                100))*(18000/M_PI_F),-lean_angle_max, lean_angle_max);
float cos_pitch_target = cosf(_pitch_target*M_PI_F/18000);
_roll_target = constrain_float(atanf(accel_right*cos_pitch_target/(GRAVITY_MSS *
                                100))*(18000/M_PI_F), -lean_angle_max, lean_angle_max);
```

}

10、将地球坐标系的角度转换成机体坐标系的角速度

```
void AC_AttitudeControl::angle_ef_roll_pitch_rate_ef_yaw(.....)
{
    //计算目标角度和 dcm 角度差
    _angle_ef_target.x = constrain_float(roll_angle_ef,
                                -_aparm.angle_max, _aparm.angle_max);
    angle_ef_error.x = wrap_180_cd_float(_angle_ef_target.x -
                                _ahrs.roll_sensor);

    _angle_ef_target.y = constrain_float(pitch_angle_ef,
                            -_aparm.angle_max, _aparm.angle_max);
    angle_ef_error.y = wrap_180_cd_float(_angle_ef_target.y -
                            _ahrs.pitch_sensor);

    .....
    //通过 Z 轴的 w 计算 Z 轴的角度差
    update_ef_yaw_angle_and_error(_rate_ef_desired.z, angle_ef_error,
                        AC_ATTITUDE_RATE_STAB_YAW_OVERSHOOT_ANGLE_MAX);
    //将地球坐标系的角度差转成体坐标的角度差
```

```
    frame_conversion_ef_to_bf(angle_ef_error, _angle_bf_error);
    //更新机体坐标的 w
    update_rate_bf_targets();
    //计算机体期望 w
    _rate_ef_desired.x = 0;
    _rate_ef_desired.y = 0;
    frame_conversion_ef_to_bf(_rate_ef_desired, _rate_bf_desired);

    //计算机体的目标 w
    _rate_bf_target += _rate_bf_desired;

}
```

（这部分后续分析.....）

第四章 ekf 和 dcm 对数据的处理

第一节 ekf 和 dcm 代码总体流程介绍

```
void Copter::loop()--->
void Copter::fast_loop()
{
    read_AHRS();
}-->
void Copter::read_AHRS(void)
{
    ahrs.update();//这个地方调用了 AP_AHRS_NavEKF 中的 updata,而不是
AP_AHRS_DCM 的 updata
}--->
void AP_AHRS_NavEKF::update(void)
{
    AP_AHRS_DCM::update(); //这里直接调用了 AP_AHRS_DCM 的 updata


    EKF.UpdateFilter(); //更新滤波器状态
}
```

先看 dcm 矩阵的更新

Dcm 状态更新的算法主要在 DCMDraft2.pdf 和 Starlino_DCM_Tutorial_01.pdf 中有很详细的说明。

```
Void AP_AHRS_DCM::update(void)
{
    //这里更新 imu(加速度计和陀螺仪)的值，具体请看第一章相关的流程分析
    _ins.update();
    //用陀螺仪更新 dcm 矩阵的值
    matrix_update(delta_t);
    //这里考虑到经过 dcm 变换后，矩阵各个轴可能不在垂直，所以要正交化
    normalize();
    //漂移修正
    drift_correction(delta_t);
    //检查 dcm 矩阵
    check_matrix();
    //计算欧拉角
    euler_angles();
}
```

Ekf 滤波代码流程

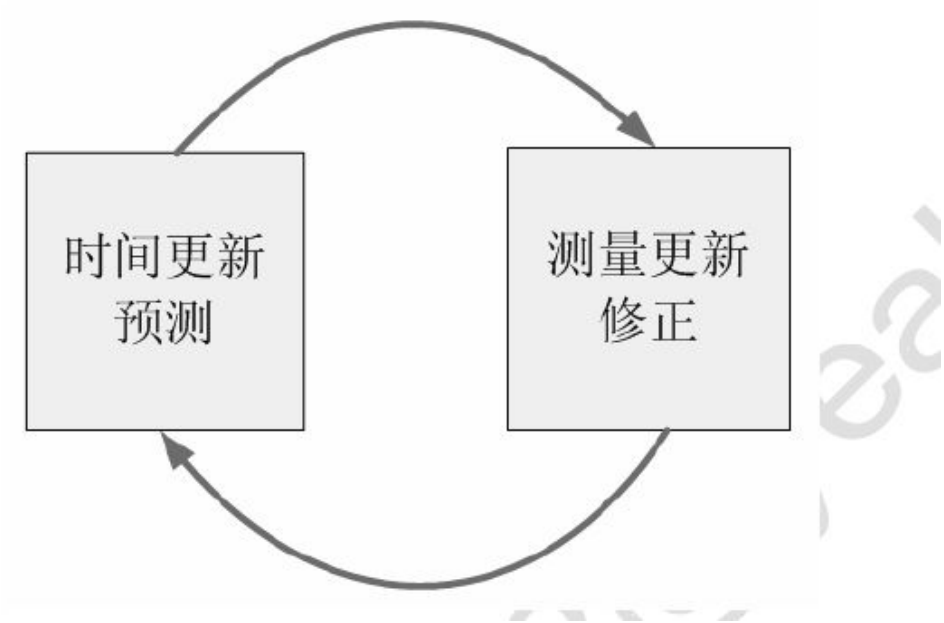
```
void NavEKF::UpdateFilter()
{
```

```
    //初始四元素
    correctedDelAngQuat.initialise();
    //读取加速度计和陀螺仪的值
    readIMUData();
    //通过 gps、空速传感器和高度计判断是否在地面上
    SetFlightAndFusionModes();
    //检查启动状态
    performArmingChecks();
    //根据 imu 测量的数据，更新四元素 、速度、位置的状态
    UpdateStrapdownEquationsNED();

    //这里保存了连续 50 次的状态方程数据
    StoreStates();

    //利用 gps 数据和气压计数据更新状态方程
    SelectVelPosFusion();
    //利用指南针数据更新状态方程
    SelectMagFusion();
    //利用光流传感器更新状态方程
    SelectFlowFusion();
    //利用空速传感器更新状态方程
    SelectTasFusion();

}
```

第二节 Ekf 状态方程分析

ekf 算法

（该节主要来源 KalmanFilter.v2.pdf，具体原理推导，请查看原文）
卡尔曼滤波器的算法分成两个循环往复的步骤：时间更新和测量更新，概况如下图。



第一步、时间更新

将系统的时间向前移动一格。卡尔曼滤波器按照先验估计给出新的时间点的估计值。涉及的公式如下

一、预测系统状态

$$\hat{x}_t^- = a\hat{x}_{t-1} + bu_{t-1}$$

二、预测系统均方差

$$p_t^- = a^2 p_{t-1} + Q$$

第二步、测量更新

将新时间点的实际测量值加入到算法中，滤波器按照此值修正卡尔曼混合系统的修正值，并给出后验估计。涉及的公式如下

一、修正卡尔曼混合系数

$$k_t = \frac{hp_t^-}{h^2 p_t^- + R}$$

二、修正系统状态

$$\hat{x}_t = (1 - hk)\hat{x}_t^- + kz_t$$

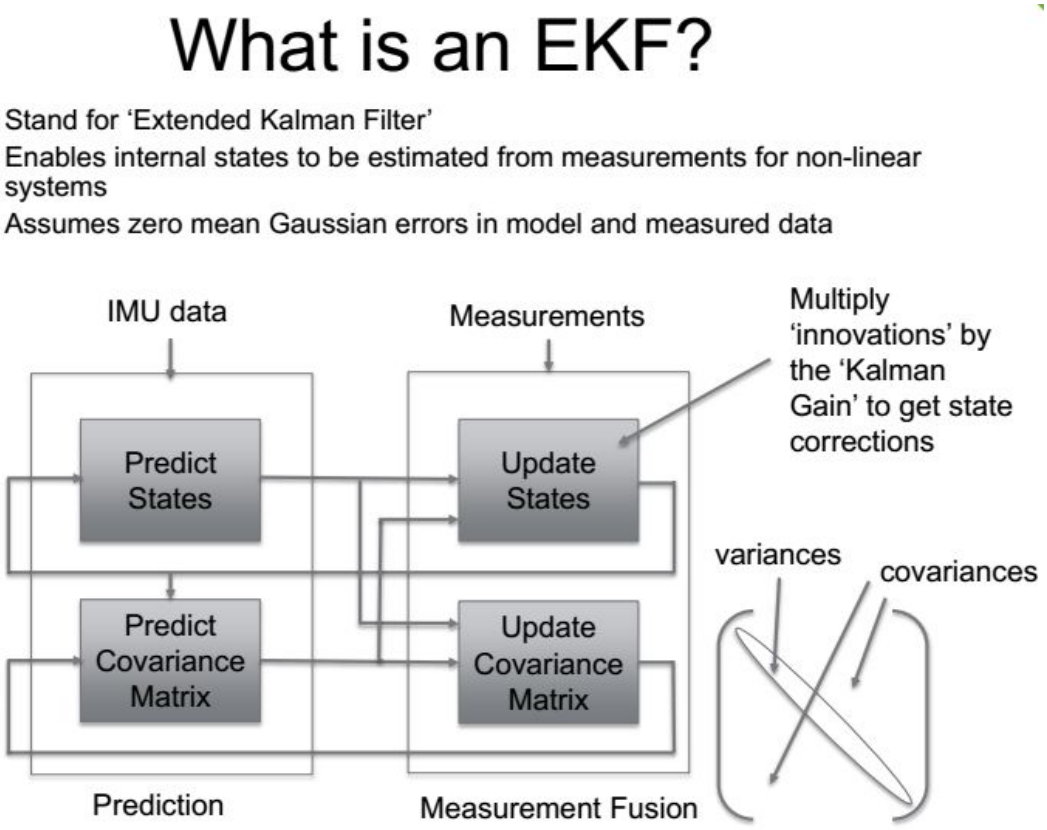
三、修正系统均方差

$$p_t = (1 - hk_t)p_t^-$$

px4 ekf 算法详解

（以下图片来源于 Application_of_Data_Fusion_to_Aerial_Robotics.pdf）

算法原理



Ekf 22 维状态变量

$$X=[\delta q_0,\delta q_1,\delta q_2,\delta q_3,\delta v_x,\delta v_y,\delta v_z,\delta p_x,\delta p_y,\delta p_z,\delta g_x,\delta g_y,\delta g_z,\delta a_1,$$
$$\delta w_x,\delta w_y,\delta em_x,\delta em_y,\delta em_z,\delta bm_x,\delta bm_y,\delta bm_z]$$

状态表结构体

（22 状态变量对应状态表结构体的前 22 位）

```
struct state_elements {
        Quaternion    quat;              // 0..3       四元素
        Vector3f      velocity;          // 4..6       速度值
        Vector3f      position;          // 7..9       位置值
        Vector3f      gyro_bias;         // 10..12     陀螺仪偏移误差
        float         accel_zbias1;      // 13         加速度计 1 Z 轴的偏移误差
        Vector2f      wind_vel;          // 14..15     风的速度
        Vector3f      earth_magfield; // 16..18        地磁场值
        Vector3f      body_magfield;     // 19..21     机体磁场值
        float         accel_zbias2;      // 22         加速度计 2 Z 轴的偏移误差
        Vector3f      vel1;              // 23 .. 25   IMU1 加速度计积分的速度
        float         posD1;             // 26         IMU1 加速度计积分的高度
        Vector3f      vel2;              // 27 .. 29    IMU2 加速度计积分的速度
```

```
        float       posD2;          // 30       IMU2 计速度计积分的高度
        Vector3f    omega;          // 31 .. 33
    } &state;
```

状态方程

$$x_t = Fx_{t-1} + Gu_t$$

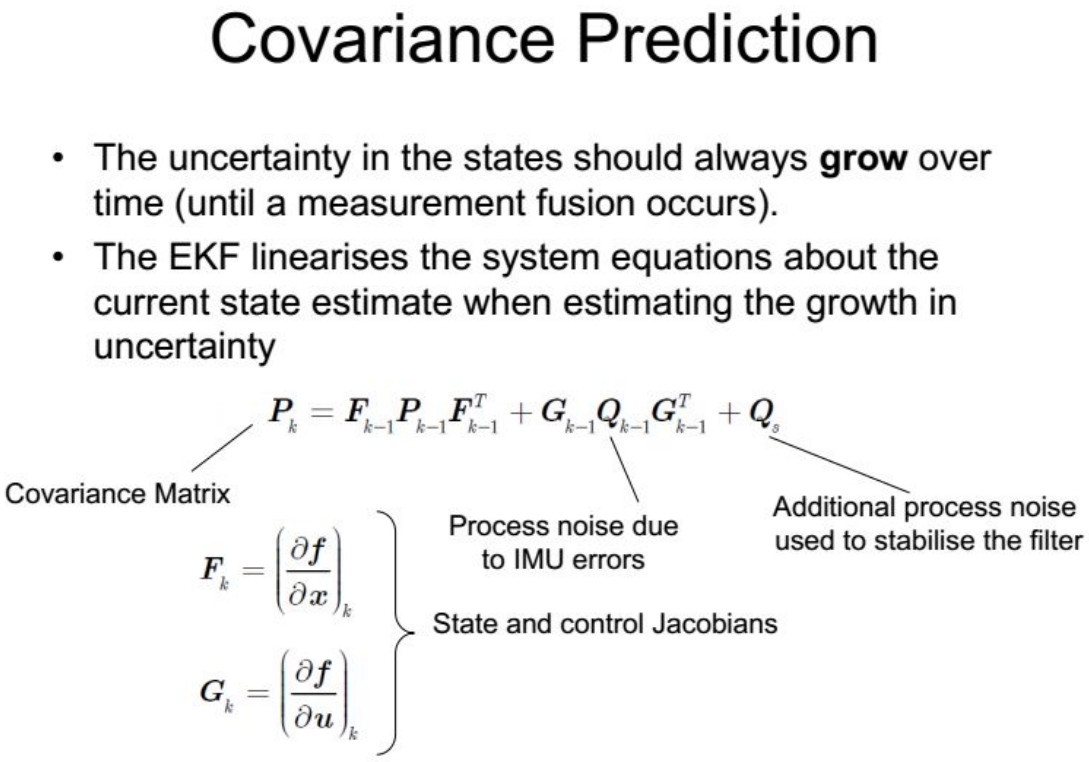
量测方程

$$z_t = Hx_t$$

Px4 ekf 代码流程如下

一、预测系统均方差

相关代码在 CovariancePrediction()函数中实现



二、融合测量数据、更新均方差

相关代码在 FuseVelPosNED()、FuseMagnetometer()、FuseOptFlow()、FuseAirspeed()、FuseSideslip()中

Measurement Fusion

Updates the state estimates and covariance matrix using measurements.

The covariance will always **decrease** after measurements are fused provided new information is gained.

Kalman Gain: $\boldsymbol{K} = \boldsymbol{P}_k^- \boldsymbol{H}_k \left[\boldsymbol{H}_k \boldsymbol{P}_k^- \boldsymbol{H}_k^T + \boldsymbol{R}_k \right]^{-1}$

$\boldsymbol{H}_k = \left(\frac{\partial \boldsymbol{z}_p}{\partial \boldsymbol{x}} \right)_k$

Measurement covariance

Innovation: $\nu = \boldsymbol{z} - \boldsymbol{z}_p$

Predicted measurement

Covariance Update: $\boldsymbol{P}_k^+ = \left[\boldsymbol{I} - \boldsymbol{K} \boldsymbol{H}_k \right] \boldsymbol{P}_k^-$

State Update: $\boldsymbol{x}_k^+ = \boldsymbol{x}_k^- + \boldsymbol{K} \nu$

Actual measurement

计算 F_k、Q_k、G_k 和 H_k

$Q_k =$

$$
\begin{array}{cccccccccccccccccccccc}
q_{0,0} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & q_{1,1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & q_{2,2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & q_{4,4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & q_{5,5} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & q_{6,6} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}
$$

其中

（sq 表示"平方",dvxCov 是加速度计在 x 轴的方差,dvyCov 加速度计在 y 轴的方差,dvzCov 加速度计在 z 轴的方差,daxCov 陀螺仪在 x 轴上的方差，dayCov 陀螺仪在 y 轴上的方差，dazCov 陀螺仪在 z 轴上的方差）

$q_{0,0}$ = daxCov=sq(dt*_gyrNoise);

$q_{1,1}$ = dayCov=sq(dt*_gyrNoise);

$q_{2,2}$ = dazCov=sq(dt*_gyrNoise) + sq(dt*0.03f*yawRateFilt);

$q_{4,4}$ = dvxCov=sq(dt*_accNoise);

$q_{5,5}$ = dvyCov=sq(dt*_accNoise);

$q_{6,6}$ = dvzCov=sq(dt*_accNoise);

由于 H_k 是估计测量值对 22 维状态的雅克比矩阵 $\left\{\frac{\partial \overline{z_t}}{\partial x}\right\}$，不同的传感器其雅克比矩阵不同，在具体的传感器数据融合中分析。

F 和 G 矩阵的计算

从 Application_of_Data_Fusion_to_Aerial_Robotics.pdf 文档中，我们可以找到 F 和 G 矩阵的计算原理

$F = \frac{\partial x_t}{\partial x_{t-1}}$，要求 F 矩阵，我们需要向找到 x_t 中每一项对于 x_{t-1} 的更新方程

影响 t 时刻四元数的值，有 t-1 时刻的四元素和陀螺仪的角度的积分值以及陀螺仪的误差；影响 t 时刻速度，有 t-1 时刻的速度值和速度的的增量值，以及加速度计的误差；影响 t 时刻的位置的有 t-1 时刻的位置和速度，根据这些，我们可以把 F 矩阵列出来

$$
F=\begin{matrix}
1 & F_{0,1} & F_{0,2} & F_{0,3} & 0 & 0 & 0 & 0 & 0 & 0 & F_{0,10} & F_{0,11} & F_{0,12} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
F_{1,0} & 1 & F_{1,2} & F_{1,3} & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{q_0}{2} & F_{1,11} & F_{1,12} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
F_{2,0} & F_{2,1} & 1 & F_{2,3} & 0 & 0 & 0 & 0 & 0 & 0 & F_{2,10} & -\frac{q_0}{2} & F_{2,12} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
F_{3,0} & F_{3,1} & F_{3,2} & 1 & 0 & 0 & 0 & 0 & 0 & 0 & F_{3,10} & F_{3,11} & -\frac{q_0}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
F_{4,0} & F_{4,1} & F_{4,2} & F_{4,3} & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & F_{4,13} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
F_{5,0} & F_{5,1} & F_{5,2} & F_{5,3} & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & F_{5,13} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
F_{6,0} & F_{6,1} & F_{6,2} & F_{6,3} & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & F_{6,13} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & dt & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & dt & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & dt & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{matrix}
$$

首先看最四元数的更新

（该资料来自网上《四元数解算姿态完全解析及资料汇总》）

一阶龙格库塔法更新四元数

3. 四元數微分

四元數微分，已知一四元數 $Q = Cos[\frac{\theta}{2}] + \hat{n} \cdot Sin[\frac{\theta}{2}]$，對時間微分

$$\frac{dQ}{dt} = -\frac{1}{2}Sin[\frac{\theta}{2}] \cdot \frac{d\theta}{dt} + \frac{d\hat{n}}{dt} \cdot Sin[\frac{\theta}{2}] + \hat{n} \cdot \frac{1}{2}Cos[\frac{\theta}{2}] \cdot \frac{d\theta}{dt}$$

已知 $\hat{n} \cdot \hat{n} = -1$, $\frac{d\hat{n}}{dt} = 0$, $\frac{d\theta}{dt} = \omega_{Eb}^{E}$, E 為地理座標系, b 為飛行器坐標系

$$\frac{dQ}{dt} = \frac{1}{2}\hat{n} \cdot \omega_{Eb}^{E} \cdot (Cos\left[\frac{\theta}{2}\right] + \hat{n} \cdot Sin[\frac{\theta}{2}]) = \frac{1}{2}\vec{\omega}_{Eb}^{E} \cdot Q$$

因為陀螺儀在飛行器上測到的角速度為 $\vec{\omega}_{Eb}^{b} = \omega_x\hat{i} + \omega_y\hat{j} + \omega_z\hat{k}$，故將 $\vec{\omega}_{Eb}^{E}$ 轉換成 $\vec{\omega}_{Eb}^{b}$ 會較為方便

$$\vec{\omega}_{Eb}^{b} = Q^*\vec{\omega}_{Eb}^{E} \cdot Q \Rightarrow Q \cdot \vec{\omega}_{Eb}^{b} = Q \cdot Q^* \cdot \vec{\omega}_{Eb}^{E} \cdot Q = \vec{\omega}_{Eb}^{E} \cdot Q$$
$$\Rightarrow \frac{dQ}{dt} = \frac{1}{2}\vec{\omega}_{Eb}^{E} \cdot Q = \frac{1}{2}Q \cdot \vec{\omega}_{Eb}^{b}$$

將 $\frac{dQ}{dt} = \frac{1}{2}Q \cdot \vec{\omega}_{Eb}^{b}$ 展開

$$\frac{dQ}{dt} = \frac{1}{2}(q_0 + q_1\hat{i} + q_2\hat{j} + q_3\hat{k}) \cdot (\omega_x\hat{i} + \omega_y\hat{j} + \omega_z\hat{k}) = \frac{1}{2}\begin{pmatrix} 0 & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & 0 & \omega_z & -\omega_y \\ \omega_y & -\omega_z & 0 & \omega_x \\ \omega_z & \omega_y & -\omega_x & 0 \end{pmatrix} \cdot \begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{pmatrix}$$

整理為

$$\frac{dQ}{dt} = \Omega_b \cdot Q$$

其中

$$\Omega_b = \frac{1}{2}\begin{pmatrix} 0 & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & 0 & \omega_z & -\omega_y \\ \omega_y & -\omega_z & 0 & \omega_x \\ \omega_z & \omega_y & -\omega_x & 0 \end{pmatrix}$$

4. 更新四元數

使用一階 Runge-Kutta 更新四元數，假設有一微分方程

$$\frac{dX}{dt} = f[X[t], \omega[t]]$$

則其解為

$$X[t + \Delta t] = X[t] + \Delta t \cdot f[X[t], \omega[t]]$$

其中 Δt 為取樣週期，將套用至四元數

$$Q[t + \Delta t] = Q[t] + \Delta t \cdot \Omega_b[t] \cdot Q[t]$$

展開上式

$$\begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{pmatrix}_{t+\Delta t} = \begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{pmatrix}_{t} + \frac{\Delta t}{2}\begin{pmatrix} -\omega_x \cdot q_1 - \omega_y \cdot q_2 - \omega_z \cdot q_3 \\ +\omega_x \cdot q_0 - \omega_y \cdot q_3 + \omega_z \cdot q_2 \\ +\omega_x \cdot q_3 + \omega_y \cdot q_0 - \omega_z \cdot q_1 \\ -\omega_x \cdot q_2 + \omega_y \cdot q_1 + \omega_z \cdot q_0 \end{pmatrix}$$

只需利用角速度即可更新四元數

px4 考虑了陀螺仪偏移误差的问题，所以在该方程的后面添加了一项

$$\begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{pmatrix}_{t+\Delta t} = \begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{pmatrix}_{t} + \frac{1}{2}(w*\Delta t)\begin{Bmatrix} -q_1 & -q_2 & -q_3 \\ q_0 & -q_3 & q_2 \\ q_3 & q_0 & -q_1 \\ -q_2 & q_1 & q_0 \end{Bmatrix}^T - g_{bias}\begin{Bmatrix} -q_1 & -q_2 & -q_3 \\ q_0 & -q_3 & q_2 \\ q_3 & q_0 & -q_1 \\ -q_2 & q_1 & q_0 \end{Bmatrix}^T$$

为了与代码一致，这个公式要做一点小小的变化

$$\begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{pmatrix}_{t+\Delta t} = \begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{pmatrix}_{t} + \frac{1}{2}(w*\Delta t - g_{bias})\begin{Bmatrix} -q_1 & -q_2 & -q_3 \\ q_0 & -q_3 & q_2 \\ q_3 & q_0 & -q_1 \\ -q_2 & q_1 & q_0 \end{Bmatrix}^T - \frac{1}{2}g_{bias}\begin{Bmatrix} -q_1 & -q_2 & -q_3 \\ q_0 & -q_3 & q_2 \\ q_3 & q_0 & -q_1 \\ -q_2 & q_1 & q_0 \end{Bmatrix}^T$$

其中

$$w = \begin{pmatrix} w_x & w_y & w_z \end{pmatrix}$$

根据上面的关系，我们可以展开成与 q0,q1,q2,q3,相关的方程

Q0(t+△t) = q0+q1*(1/2)*(dax_b-dax)+q2*(1/2)*(day_b-day)+q3*(1/2)*(daz_b-daz) +
(1/2)*dax_b*q1+(1/2)*day_b*q2+(1/2)*daz_b*q3;
Q1(t+△t) = q0*(1/2)*(dax-dax_b)+q1+q2*(1/2)*(daz-daz_b)+q3*(1/2)*(day_b-day) +
(1/2)*dax_b*q0+(1/2)*day_b*q3-(1/2)*daz_b*q2;
Q2(t+△t) = q0*(1/2)*(day-day_b)+q1*(1/2)*(daz_b-daz)+q2+q3*(1/2)*(dax-dax_b)+
(-1/2)*dax_b*q3-(1/2)*day_b*q0+(1/2)*day_z*q1;
Q3(t+△t) = q0*(1/2)*(daz-daz_b)+q1*(1/2)*(day-day_b)+q2*(1/2)*(dax_b-dax)+q3+
(1/2)*dax_b*q2-(1/2)*day_b*q1-(1/2)*daz_b*q0;

速度的更新

在 px4 的 matlab 中有

```
% define the velocity update equations
vNew = [vn;ve;vd] + [gn;ge;gd]*dt + Tbn*dVelCor;
```

速度的更新=上一时刻的速度+加速度计对时间的积分+加速度计的偏移误差

$$\begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}_{t+\Delta t} = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}_{t} + 2(a*\Delta t)*(C_b^R)^T - a_{bias}*(C_b^R)^T$$

为了代码中一致，这里做一个小小的变化

$$\begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}_{t+\Delta t} = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}_{t} + 2(a*\Delta t - a_{bias})*(C_b^R)^T + a_{bias}*(C_b^R)^T$$

式中的“2”不知道 why。

其中 $e_{bias} = a_{bias} = \begin{pmatrix} 0 & 0 & a_{bias.z} \end{pmatrix}$，这里吧 X 和 Y 轴的偏移误差忽略掉了

$a=\begin{pmatrix}a_x & a_y & a_z\end{pmatrix}$，其中第二项是要把速度的增量分配到各个轴上去。

$$C_b^R=\begin{bmatrix}q_0^2+q_1^2-q_2^2-q_3^2 & 2(q_1q_2-q_0q_3) & 2(q_1q_3+q_0q_2)\\ 2(q_1q_2+q_0q_3) & q_0^2-q_1^2+q_2^2-q_3^2 & 2(q_2q_3-q_0q_1)\\ 2(q_1q_3-q_0q_2) & 2(q_2q_3+q_0q_1) & q_0^2-q_1^2-q_2^2+q_3^2\end{bmatrix} \quad (9.2.34)$$

V.x(t+△t) = v.x +2*dvx*(q0*q0 + q1*q2 - q2*q2 - q3*q3)+2*dvy*(-2*q0*q3 + 2*q1*q2)+
2*(dvz - dvz_b)*(2*q0*q2 + 2*q1*q3)-a.z*2*(q0*q2+q1*q3);
V.y(t+△t) = v.y+ 2*dvx*(2*q0*q3+2*q1*q2)+2*dvy*(q0*q0-q1*q1+q2*q2-q3*q3)
2*(dvz - dvz_b)*(-2*q0*q1+2*q2*q3)+a.z*2(q0*q1-q2*q3);

V.z(t+△t) = v.z+2*dvx* (-2*q0*q2+2*q1*q3)+2*dvy*(2*q0*q1+2*q2*q3)+
2*(dvz - dvz_b)*(q0*q0-q1*q1-q2*q2+q3*q3) - a.z(q0*q0-q1*q1-q2*q2+q3*q3);
继续变形为
V.x(t+△t) = q0*(2*q2*(dvz - dvz_b) + 2*dvx*q0 - 2*dvy*q3)+
q1*(2*q3*(dvz - dvz_b) + 2*dvx*q1 + 2*dvy*q2)+
q2*(2*q0*(dvz - dvz_b) - 2*dvx*q2 + 2*dvy*q1)-
q3*(- 2*q1*(dvz - dvz_b) + 2*dvx*q3 + 2*dvy*q0)
+ v.x-a.z*2*(q0*q2+q1*q3);
V.y(t+△t) = q0*(- 2*q1*(dvz - dvz_b) + 2*dvx*q3 + 2*dvy*q0)-
q1*(2*q0*(dvz - dvz_b) - 2*dvx*q2 + 2*dvy*q1)+
q2*(2*q3*(dvz - dvz_b) + 2*dvx*q1 + 2*dvy*q2)+
q3*(2*q2*(dvz - dvz_b) + 2*dvx*q0 - 2*dvy*q3)
+v.y+a.z*2(q0*q1-q2*q3);

V.z(t+△t) = q0*(2*q0*(dvz - dvz_b) - 2*dvx*q2 + 2*dvy*q1)+
q1*(- 2*q1*(dvz - dvz_b) + 2*dvx*q3 + 2*dvy*q0)-
q2*(2*q2*(dvz - dvz_b) + 2*dvx*q0 - 2*dvy*q3)+
q3*(2*q3*(dvz - dvz_b) + 2*dvx*q1 + 2*dvy*q2)
+v.z- a.z(q0*q0-q1*q1-q2*q2+q3*q3);

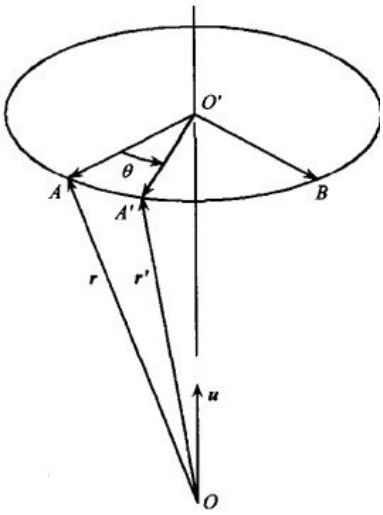
（C_b^R 的是旋转矩阵，求发如下，来自《四元数解算姿态完全解析及资料汇总》）

四元数与姿态阵间的关系——摘自《惯性导航》-秦永元 P292-297

9.2.2 四元数与姿态阵间的关系

设有参考坐标系 R，坐标轴为 x_0、y_0、z_0，坐标轴方向的单位向量为 $\boldsymbol{i}_0$、$\boldsymbol{j}_0$、$\boldsymbol{k}_0$。刚体相对 R 系作定点转动，定点为 O。取坐标系 b 与刚体固联，b 系的坐标轴为 x、y、z，坐标轴方向的单位向量为 $\boldsymbol{i}$、$\boldsymbol{j}$、$\boldsymbol{k}$。假设初始时刻 b 系与 R 系重合。为了便于分析刚体的空间角位置，在刚体上取一点 A，转动点 O 至该点引位置向量 $\boldsymbol{OA}$，如图 9.2.1 所示。则该位置向量的空间位置实际上描述了刚体的空间角位置。

设刚体以$\boldsymbol{\omega}=\omega_x\boldsymbol{i}+\omega_y\boldsymbol{j}+\omega_z\boldsymbol{k}$相对 R 系旋转，初始时刻位置向量处于$\boldsymbol{OA}=\boldsymbol{r}$，经过时间 t 后位置向量处于$\boldsymbol{OA}'=\boldsymbol{r}'$。根据欧拉定理，仅考虑刚体在 0 时刻和 t 时刻的角位置时，刚体从 A 位置转到 A'位置的转动可等效成绕瞬轴 $\boldsymbol{u}$（单位向量）转过θ角一次完成。这样，位置向量做圆锥运动，A 和 A'位于同一圆上，$\boldsymbol{r}$ 和 $\boldsymbol{r}'$位于同一圆锥面上。



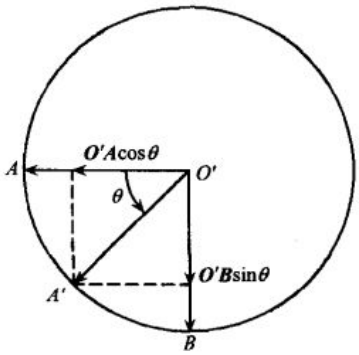


图 9.2.1　刚体的等效旋转

下面分析 $\boldsymbol{r}'$与 $\boldsymbol{r}$ 的关系。在圆上取一点 B，使$\angle AO'B=90°$，由图得

$$\begin{aligned}\boldsymbol{OO}' &=(\boldsymbol{r}\cdot\boldsymbol{u})\boldsymbol{u}\\ \boldsymbol{O'A} &=\boldsymbol{r}-\boldsymbol{OO}'=\boldsymbol{r}-(\boldsymbol{r}\cdot\boldsymbol{u})\boldsymbol{u}\\ \boldsymbol{O'B} &=\boldsymbol{u}\times\boldsymbol{O'A}\\ &=\boldsymbol{u}\times\boldsymbol{r}-(\boldsymbol{r}\cdot\boldsymbol{u})\boldsymbol{u}\times\boldsymbol{u}=\boldsymbol{u}\times\boldsymbol{r}\\ \boldsymbol{O'A'} &=\boldsymbol{O'A}\cos\theta+\boldsymbol{O'B}\sin\theta\\ &=\boldsymbol{r}\cos\theta-(\boldsymbol{r}\cdot\boldsymbol{u})\boldsymbol{u}\cos\theta+\boldsymbol{u}\times\boldsymbol{r}\sin\theta\end{aligned}$$

所以

$$\begin{aligned}\boldsymbol{r}' &=\boldsymbol{OO}'+\boldsymbol{O'A'}=\boldsymbol{r}\cos\theta+(1-\cos\theta)\\ &\times(\boldsymbol{r}\cdot\boldsymbol{u})\boldsymbol{u}+\boldsymbol{u}\times\boldsymbol{r}\sin\theta\end{aligned}$$

由三重矢积计算公式：

$$\begin{aligned}\boldsymbol{u}\times(\boldsymbol{u}\times\boldsymbol{r}) &=\boldsymbol{u}(\boldsymbol{u}\cdot\boldsymbol{r})-(\boldsymbol{u}\cdot\boldsymbol{u})\boldsymbol{r}\\ &=(\boldsymbol{r}\cdot\boldsymbol{u})\boldsymbol{u}-\boldsymbol{r}\end{aligned}$$

即

$$(\boldsymbol{r}\cdot\boldsymbol{u})\boldsymbol{u}=\boldsymbol{r}+\boldsymbol{u}\times(\boldsymbol{u}\times\boldsymbol{r})$$

所以

$$\begin{aligned}\boldsymbol{r}' &=\boldsymbol{r}\cos\theta+(1-\cos\theta)[\boldsymbol{r}+\boldsymbol{u}\times(\boldsymbol{u}\times\boldsymbol{r})]+\boldsymbol{u}\times\boldsymbol{r}\sin\theta\\ &=\boldsymbol{r}+\boldsymbol{u}\times\boldsymbol{r}\sin\theta+(1-\cos\theta)\boldsymbol{u}\times(\boldsymbol{u}\times\boldsymbol{r})\end{aligned}$$

将上式向 R 系内投影：

$$\begin{aligned}\boldsymbol{r}'^{R} &=\boldsymbol{r}^{R}+(\boldsymbol{u}\times\boldsymbol{r})^{R}\sin\theta\\ &+(1-\cos\theta)[\boldsymbol{u}\times(\boldsymbol{u}\times\boldsymbol{r})]^{R}\end{aligned}$$

记

$$\boldsymbol{r}'^{R}=\begin{bmatrix}r'_x\\ r'_y\\ r'_z\end{bmatrix},\qquad \boldsymbol{r}^{R}=\begin{bmatrix}r_x\\ r_y\\ r_z\end{bmatrix},\qquad \boldsymbol{u}^{R}=\begin{bmatrix}l\\ m\\ n\end{bmatrix}$$

又根据叉乘关系表达式：

$$(\boldsymbol{u}\times\boldsymbol{r})^R=\begin{bmatrix}0&-n&m\\n&0&-l\\-m&l&0\end{bmatrix}\begin{bmatrix}r_x\\r_y\\r_z\end{bmatrix}$$

记

$$\boldsymbol{U}=\begin{bmatrix}0&-n&m\\n&0&-l\\-m&l&0\end{bmatrix}\tag{9.2.23}$$

则

$$(\boldsymbol{u}\times\boldsymbol{r})^R=\boldsymbol{U}\boldsymbol{r}^R$$
$$[\boldsymbol{u}\times(\boldsymbol{u}\times\boldsymbol{r})]^R=\boldsymbol{U}\cdot\boldsymbol{U}\boldsymbol{r}^R$$

所以

$$\begin{aligned}\boldsymbol{r}'^R&=\boldsymbol{r}^R+\boldsymbol{U}\boldsymbol{r}^R\sin\theta+(1-\cos\theta)\boldsymbol{U}\cdot\boldsymbol{U}\boldsymbol{r}^R\\&=\left(\boldsymbol{I}+2\boldsymbol{U}\sin\frac{\theta}{2}\cos\frac{\theta}{2}+2\sin^2\frac{\theta}{2}\boldsymbol{U}\cdot\boldsymbol{U}\right)\boldsymbol{r}^R\end{aligned}\tag{9.2.24}$$

令

$$\boldsymbol{D}=\boldsymbol{I}+2\boldsymbol{U}\sin\frac{\theta}{2}\cos\frac{\theta}{2}+2\sin^2\frac{\theta}{2}\boldsymbol{U}\cdot\boldsymbol{U}\tag{9.2.25}$$

则式(9.2.24)可写成：

$$\boldsymbol{r}'^R=\boldsymbol{D}\boldsymbol{r}^R\tag{9.2.26}$$

记初始时刻的刚体固联坐标系为 b_0，由于初始时刻刚体固联坐标系与参考坐标系重合，所以

$$\boldsymbol{r}^R=\boldsymbol{r}^{b_0}\tag{9.2.27}$$

而在转动过程中，位置向量和 b 系都同刚体固联，所以位置向量和 b 系的相对角位置始终不变，即有

$$\boldsymbol{r}^{b_0}=\boldsymbol{r}'^b\tag{9.2.28}$$

将式(9.2.28)代入式(9.2.27)，得

$$\boldsymbol{r}^R=\boldsymbol{r}'^b\tag{9.2.29}$$

将式(9.2.29)代入式(9.2.26)，得

$$\boldsymbol{r}'^R=\boldsymbol{D}\boldsymbol{r}'^b$$

该式说明 $\boldsymbol{D}$ 即为 b 系至 R 系的坐标变换矩阵，根据式(9.2.25)和式(9.2.23)

$$\boldsymbol{C}_b^R=\boldsymbol{D}=\boldsymbol{I}+2\boldsymbol{U}\sin\frac{\theta}{2}\cos\frac{\theta}{2}+2\sin^2\frac{\theta}{2}\boldsymbol{U}\cdot\boldsymbol{U}\tag{9.2.30a}$$

即

$$\boldsymbol{C}_b^R=\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}+2\cos\frac{\theta}{2}\begin{bmatrix}0&-n\sin\frac{\theta}{2}&m\sin\frac{\theta}{2}\\n\sin\frac{\theta}{2}&0&-l\sin\frac{\theta}{2}\\-m\sin\frac{\theta}{2}&l\sin\frac{\theta}{2}&0\end{bmatrix}$$

$$+2\begin{bmatrix} -(m^2+n^2)\sin^2\frac{\theta}{2} & lm\sin^2\frac{\theta}{2} & ln\sin^2\frac{\theta}{2} \\ lm\sin^2\frac{\theta}{2} & -(l^2+n^2)\sin^2\frac{\theta}{2} & mn\sin^2\frac{\theta}{2} \\ ln\sin^2\frac{\theta}{2} & mn\sin^2\frac{\theta}{2} & -(m^2+l^2)\sin^2\frac{\theta}{2} \end{bmatrix} \tag{9.2.30b}$$

令

$$\begin{cases} q_0 = \cos\dfrac{\theta}{2} \\ q_1 = l\sin\dfrac{\theta}{2} \\ q_2 = m\sin\dfrac{\theta}{2} \\ q_3 = n\sin\dfrac{\theta}{2} \end{cases} \tag{9.2.31}$$

并以 q_0、q_1、q_2、q_3 构造四元数：

$$\boldsymbol{Q} = q_0 + q_1\boldsymbol{i}_0 + q_2\boldsymbol{j}_0 + q_3\boldsymbol{k}_0 = \cos\frac{\theta}{2} + (l\boldsymbol{i}_0 + m\boldsymbol{j}_0 + n\boldsymbol{k}_0)\sin\frac{\theta}{2}$$

$$= \cos\frac{\theta}{2} + \boldsymbol{u}^R\sin\frac{\theta}{2} \tag{9.2.32}$$

则可得如下结论：

（1）四元数 $\boldsymbol{Q}=\cos\dfrac{\theta}{2}+\boldsymbol{u}^R\sin\dfrac{\theta}{2}$ 描述了刚体的定点转动，即当只关心 b 系相对 R 系的角位置时，可认为 b 系是由 R 系经过无中间过程的一次性等效旋转形成的，$\boldsymbol{Q}$ 包含了这种等效旋转的全部信息：$\boldsymbol{u}^R$ 为旋转瞬轴和旋转方向，θ 为转过的角度。

（2）四元数可确定出 b 系至 R 系的坐标变换矩阵。将式(9.2.31)代入式(9.2.30)，得

$$\boldsymbol{C}_b^R = \begin{bmatrix} 1-2(q_2^2+q_3^2) & 2(q_1q_2-q_0q_3) & 2(q_1q_3+q_0q_2) \\ 2(q_1q_2+q_0q_3) & 1-2(q_1^2+q_3^2) & 2(q_2q_3-q_0q_1) \\ 2(q_1q_3-q_0q_2) & 2(q_2q_3+q_0q_1) & 1-2(q_1^2+q_2^2) \end{bmatrix} \tag{9.2.33}$$

由于 $\|\boldsymbol{Q}\| = q_0^2+q_1^2+q_2^2+q_3^2 = \cos^2\dfrac{\theta}{2}+(l^2+m^2+n^2)\sin^2\dfrac{\theta}{2}=1$，所以可进一步推得如下结论：

（1）描述刚体旋转的四元数是规范化四元数。

（2）式(9.2.33)可写成：

$$\boldsymbol{C}_b^R = \begin{bmatrix} q_0^2+q_1^2-q_2^2-q_3^2 & 2(q_1q_2-q_0q_3) & 2(q_1q_3+q_0q_2) \\ 2(q_1q_2+q_0q_3) & q_0^2-q_1^2+q_2^2-q_3^2 & 2(q_2q_3-q_0q_1) \\ 2(q_1q_3-q_0q_2) & 2(q_2q_3+q_0q_1) & q_0^2-q_1^2-q_2^2+q_3^2 \end{bmatrix} \tag{9.2.34}$$

如果参考坐标系 R 是导航坐标系 n，刚体固联坐标系 b 为机体坐标系，则坐标变换阵 $\boldsymbol{C}_b^R$ 就是姿态矩阵 $\boldsymbol{C}_b^n$，而由姿态矩阵可计算出航向角和姿态角。

设运载体的航向角为 Ψ（习惯上以北偏东为正），俯仰角为 θ，横滚角为 γ，取地理坐标系 g 为导航坐标系，并规定 x_g、y_g、z_g 的指向依次为东、北、天，则机体坐标系 b 与导航坐标系 n（即地理坐标系 g）的关系如图 1.2.3 所示。

由该图可得三次基本旋转对应的坐标变换阵为

$$\boldsymbol{C}_n^1 = \boldsymbol{C}_g^1 = \begin{bmatrix} \cos\Psi & -\sin\Psi & 0 \\ \sin\Psi & \cos\Psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{9.2.37}$$

$$\begin{aligned} \boldsymbol{C}_1^b &= \boldsymbol{C}_2^b \boldsymbol{C}_1^2 = \begin{bmatrix} \cos\gamma & 0 & -\sin\gamma \\ 0 & 1 & 0 \\ \sin\gamma & 0 & \cos\gamma \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix} \\ &= \begin{bmatrix} \cos\gamma & \sin\theta\sin\gamma & -\cos\theta\sin\gamma \\ 0 & \cos\theta & \sin\theta \\ \sin\gamma & -\sin\theta\cos\gamma & \cos\theta\cos\gamma \end{bmatrix} \end{aligned} \tag{9.2.38}$$

$$\begin{aligned} \boldsymbol{C}_n^b &= \boldsymbol{C}_1^b \cdot \boldsymbol{C}_n^1 \\ &= \begin{bmatrix} \cos\gamma\cos\Psi + \sin\gamma\sin\Psi\sin\theta & -\cos\gamma\sin\Psi + \sin\gamma\cos\Psi\sin\theta & -\sin\gamma\cos\theta \\ \sin\Psi\cos\theta & \cos\Psi\cos\theta & \sin\theta \\ \sin\gamma\cos\Psi - \cos\gamma\sin\Psi\sin\theta & -\sin\gamma\sin\Psi - \cos\gamma\cos\Psi\sin\theta & \cos\gamma\cos\theta \end{bmatrix} \end{aligned} \tag{9.2.39}$$

记 $\boldsymbol{C}_b^n = \begin{bmatrix} T_{11} & T_{12} & T_{13} \\ T_{21} & T_{22} & T_{23} \\ T_{31} & T_{32} & T_{33} \end{bmatrix}$，由于 n 系至 b 系的旋转过程中坐标系始终保持直角坐标系，所以 $\boldsymbol{C}_b^n$ 为正交矩阵

$$\boldsymbol{C}_n^b = (\boldsymbol{C}_b^n)^{\mathrm{T}} = \begin{bmatrix} T_{11} & T_{21} & T_{31} \\ T_{12} & T_{22} & T_{32} \\ T_{13} & T_{23} & T_{33} \end{bmatrix} \tag{9.2.40}$$

比较式(9.2.39)和式(9.2.40)，得

$$\begin{cases} \theta = \arcsin(T_{32}) \\ \gamma_{主} = \arctan\left(-\dfrac{T_{31}}{T_{33}}\right) \\ \Psi_{主} = \arctan\left(\dfrac{T_{12}}{T_{22}}\right) \end{cases} \tag{9.2.41}$$

航向角和横滚角的真值按表 9.2.1 和表 9.2.2 确定。

表 9.2.1 航向角Ψ真值表

T_{22}	T_{12}	Ψ
→0	+	90°
→0	−	−90°
+	+	$\Psi_{主}$
+	−	$\Psi_{主}$
−	+	$\Psi_{主}+180°$
−	−	$\Psi_{主}-180°$

表 9.2.2 横滚角γ真值表

$\gamma_{主}$	T_{33}	γ
+	+	$\gamma_{主}$
−		
+	−	$\gamma_{主}-180°$
−	−	$\gamma_{主}+180°$

上述分析说明：如果表征 n 系至 b 系的旋转四元数 $\boldsymbol{Q}$ 已确定，则按式(9.2.33)或式(9.2.34)可计算出姿态阵 $\boldsymbol{C}_b^n$，再按式(9.2.41)和表 9.2.1 及表 9.2.2 可确定出运载体的航向角、俯仰角和横滚角，因此，四元数 $\boldsymbol{Q}$ 包含了所有的姿态信息，捷联惯导中的姿态更新实质上是如何计算四元数 $\boldsymbol{Q}$。

位置的更新

Px4 matlab 中关于位置更新的描述

```
% define the position update equations
pNew = [pn;pe;pd] + [vn;ve;vd]*dt;
```

根据上面的求出的四元数、速度和位置的公式，我们可以都其中的每一项进行求偏导，就得到了 F 对应的每项的值

$$F_{0,1}=\frac{\partial q_0(t+\Delta t)}{\partial q_1}=\text{dax\_b/2 - dax/2;} \qquad F_{0,2}=\frac{\partial q_0(t+\Delta t)}{\partial q_2}=\text{day\_b/2 - day/2;}$$

$$F_{0,3}=\frac{\partial q_0(t+\Delta t)}{\partial q_1}=\text{daz\_b/2 - daz/2;} \qquad F_{0,10}=\frac{\partial q_0(t+\Delta t)}{\partial dax\_b}=\text{q1/2;}$$

$$F_{0,11}=\frac{\partial q_0(t+\Delta t)}{\partial day\_b}=\text{q2/2;} \qquad F_{0,12}=\frac{\partial q_0(t+\Delta t)}{\partial daz\_b}=\text{q3/2;}$$

$$F_{1,0}=\frac{\partial q_1(t+\Delta t)}{\partial q_0}=\text{dax/2 - dax\_b/2;} \qquad F_{1,2}=\frac{\partial q_1(t+\Delta t)}{\partial q_2}=\text{daz/2 - daz\_b/2;}$$

$F_{1,3}=\frac{\partial q_1(t+\Delta t)}{\partial q_3}$=day_b/2 - day/2；　　$F_{1,10}=\frac{\partial q_1(t+\Delta t)}{\partial dax\_b}=-\frac{q_0}{2}$

$F_{1,11}=\frac{\partial q_1(t+\Delta t)}{\partial day\_b}$=q3/2；　　$F_{1,12}=\frac{\partial q_1(t+\Delta t)}{\partial daz\_b}$=-q2/2；

$F_{2,0}=\frac{\partial q_2(t+\Delta t)}{\partial q_0}$=day/2 - day_b/2；　　$F_{2,1}=\frac{\partial q_2(t+\Delta t)}{\partial q_1}$=daz_b/2 - daz/2；

$F_{2,3}=\frac{\partial q_2(t+\Delta t)}{\partial q_3}$=dax/2 - dax_b/2;　　$F_{2,10}=\frac{\partial q_2(t+\Delta t)}{\partial dax\_b}$=-q3/2；

$F_{2,11}=\frac{\partial q_2(t+\Delta t)}{\partial day\_b}=-\frac{q_0}{2}$；　　$F_{2,12}=\frac{\partial q_2(t+\Delta t)}{\partial daz\_b}$=q1/2；

$F_{3,0}=\frac{\partial q_3(t+\Delta t)}{\partial q_0}$= daz/2 - daz_b/2；　　$F_{3,1}=\frac{\partial q_3(t+\Delta t)}{\partial q_1}$=day/2 - day_b/2；

$F_{3,2}=\frac{\partial q_3(t+\Delta t)}{\partial q_2}$=dax_b/2 - dax/2；　　$F_{3,10}=\frac{\partial q_3(t+\Delta t)}{\partial dax\_b}$=q2/2；

$F_{3,11}=\frac{\partial q_3(t+\Delta t)}{\partial day\_b}$=-q1/2；　　$F_{3,12}=\frac{\partial q_3(t+\Delta t)}{\partial daz\_b}=-\frac{q_0}{2}$；

$F_{4,0}=\frac{\partial v_x(t+\Delta t)}{\partial q_0}$= 2*q2*(dvz - dvz_b) + 2*dvx*q0 - 2*dvy*q3；

$F_{4,1}=\frac{\partial v_x(t+\Delta t)}{\partial q_1}$= 2*q3*(dvz - dvz_b) + 2*dvx*q1 + 2*dvy*q2；

$F_{4,2}=\frac{\partial v_x(t+\Delta t)}{\partial q_2}$= 2*q0*(dvz - dvz_b) - 2*dvx*q2 + 2*dvy*q1；

$F_{4,3}=\frac{\partial v_x(t+\Delta t)}{\partial q_3}$= -（-　 2*q1*(dvz - dvz_b) + 2*dvx*q3 + 2*dvy*q0）；

$F_{4,13}=\frac{\partial v_x(t+\Delta t)}{\partial dvz\_b}$=-2*(q0*q2+q1*q3)；

$F_{5,0}=\frac{\partial v_y(t+\Delta t)}{\partial q_0}$= -2*q1*(dvz - dvz_b) + 2*dvx*q3 + 2*dvy*q0；

$F_{5,1}=\frac{\partial v_y(t+\Delta t)}{\partial q_1}$=-（2*q0*(dvz - dvz_b) - 2*dvx*q2 + 2*dvy*q1）；

$F_{5,2}=\frac{\partial v_y(t+\Delta t)}{\partial q_2}$=2*q3*(dvz - dvz_b) + 2*dvx*q1 + 2*dvy*q2；

$F_{5,3}=\frac{\partial v_y(t+\Delta t)}{\partial q_3}$ = 2*q2*(dvz - dvz_b) + 2*dvx*q0 - 2*dvy*q3；

$F_{5,13}=\frac{\partial v_y(t+\Delta t)}{\partial dvz\_b}$=2(q0*q1-q2*q3)；

$F_{6,0}=\frac{\partial v_z(t+\Delta t)}{\partial q_0}$=2*q0*(dvz - dvz_b) - 2*dvx*q2 + 2*dvy*q1；

$F_{6,1}=\frac{\partial v_z(t+\Delta t)}{\partial q_1}$ = - 2*q1*(dvz - dvz_b) + 2*dvx*q3 + 2*dvy*q0；

$F_{6,2}=\frac{\partial v_z(t+\Delta t)}{\partial q_2}$ = -（2*q2*(dvz - dvz_b) + 2*dvx*q0 - 2*dvy*q3）；

$F_{6,3}=\frac{\partial v_z(t+\Delta t)}{\partial q_3}$ = 2*q3*(dvz - dvz_b) + 2*dvx*q1 + 2*dvy*q2；

$F_{6,13}=\frac{\partial v_z(t+\Delta t)}{\partial dvz\_b}=-(sq(q_0)-sq(q_1)-sq(q_2)+sq(q_3))$；

$G=\frac{\partial x_t}{\partial u_t}$，要求 G 矩阵，我们需要向找到 x_t 中每一项对于 u_t 的更新方程

我们知道控制量是通过控制电机的转速来改变飞机的姿态和速度和方向的；所以 G 矩阵中相关与控制量相关的就是四元数和速度了。

$G_k =$

$$
\begin{matrix}
\frac{q_1}{2} & \frac{q_2}{2} & \frac{q_3}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-\frac{q_0}{2} & \frac{q_3}{2} & -\frac{q_2}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-\frac{q_3}{2} & -\frac{q_0}{2} & \frac{q_1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{q_2}{2} & -\frac{q_1}{2} & -\frac{q_0}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & a_1 & a_2 & a_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -b_1 & -b_2 & b_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -c_1 & c_2 & c_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{matrix}
$$

其中

a1=SG[1] + SG[2] - SG[3] - SG[4]=sq(q3)+sq(q2)-sq(q1)-sq(q0);

a2=SG[7] - 2*q0*q3=2*q1*q2 - 2*q0*q3;

a3=SG[6] + 2*q0*q2=2*q1*q3 + 2*q0*q2;

b1=SG[7] + 2*q0*q3=2*q1*q2 + 2*q0*q3;

b2=SG[1] - SG[2] + SG[3] - SG[4]=sq(q3) - sq(q2) + sq(q1) -sq(q0);

b3= SG[5] - 2*q0*q1=2*q2*q3 - 2*q0*q1;

c1=SG[6] - 2*q0*q2=2*q1*q3 - 2*q0*q2;

c2=SG[5] + 2*q0*q1=2*q2*q3+2*q0*q1;

c3=SG[1] - SG[2] - SG[3] + SG[4]=sq(q3)-sq(q2)-sq(q1)+sq(q0);

从上面的 F 矩阵求解过程可知

$$\begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{pmatrix}_{t+\Delta t} += \{q_0, q_1, q_2, q_3\} * (-\frac{1}{2}) * \begin{Bmatrix} -q_1 & -q_2 & -q_3 & 0 \\ q_0 & -q_3 & q_2 & 0 \\ q_3 & q_0 & -q_1 & 0 \\ -q_2 & q_1 & q_0 & 0 \end{Bmatrix}^T ;$$

$$\begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}_{t+\Delta t} += \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}^T * (-(C_b^R)^T) ;$$

其中

$$\begin{matrix} a_1 & a_2 & a_3 \\ -b_1 & -b_2 & b_3 \\ -c_1 & c_2 & c_3 \end{matrix} = -C_b^R ;$$

第三节 空速传感器 ekf 代码分析

1、Ekf 的 22 维状态变量：

$$X = [\delta q_0, \delta q_1, \delta q_2, \delta q_3, \delta v_x, \delta v_y, \delta v_z, \delta p_x, \delta p_y, \delta p_z, \delta g_x, \delta g_y, \delta g_z, \delta a_1,$$
$$\delta w_x, \delta w_y, \delta em_x, \delta em_y, \delta em_z, \delta bm_x, \delta bm_y, \delta bm_z]$$

该状态方程的变量取自 state_elements 结构体的 0-21 项

```
struct state_elements {
        Quaternion    quat;              // 0..3        四元素
        Vector3f      velocity;          // 4..6        速度值
        Vector3f      position;          // 7..9        位置值
        Vector3f      gyro_bias;         // 10..12      陀螺仪的值
        float         accel_zbias1;      // 13          加速度计值
        Vector2f      wind_vel;          // 14..15      风的速度
        Vector3f      earth_magfield; // 16..18         地磁场值
        Vector3f      body_magfield;     // 19..21      机体磁场值
        float         accel_zbias2;      // 22
        Vector3f      vel1;              // 23 .. 25
        float         posD1;             // 26
        Vector3f      vel2;              // 27 .. 29
        float         posD2;             // 30
        Vector3f      omega;             // 31 .. 33
    } &state;
```

2、空速传感器 ekf 原理分析

空速传感器的雅克比矩阵

设机体在任意时刻的速度标量值为 f(x,y,z)

则：$f(x,y,z)=\sqrt{x^2+y^2+z^2}=|V|$

求出 x,y,z 的偏导数
X 偏导数：

$$\frac{\partial f}{\partial x}=\frac{x}{\sqrt{x^2+y^2+z^2}}=\frac{x}{|V|}$$

Y 的偏导数为

$$\frac{\partial f}{\partial y}=\frac{y}{|V|}$$

Z 的偏导数

$$\frac{\partial f}{\partial z}=\frac{z}{|V|}$$

由于空速传感器是检测的水平方向上的速度，所以空速传感器的值应于 x,y 轴的速度大小相等，方向相反。
由上面的分析可能可以求出状态方程当前的雅克比矩阵

$$J=[0,0,0,0,\frac{v_n-v_{wn}}{|V|},\frac{v_e-v_{we}}{|V|},\frac{v_d}{|V|},0,0,0,0,0,0,0,-\frac{v_n-v_{wn}}{|V|},-\frac{v_e-v_{we}}{|V|},0,0,0,0,0,0]$$

Kalman filter 的五条黄金公式

（1）、通过上一状态最优值（u_{t-1}）和将要施加的控制量（u_t）来预测当前状态$\bar{u}_t$.

$$\overline{u_t}=A_t u_{t-1}+B_t u_t$$

（2）、用上一状态的协方差（$\sum_{t-1}$ ）和高斯噪声（预测差值的方差）来计算预测值的协方差

$$\overline{\sum_t}=A_t\sum_{t-1}A_t^T+R_t$$

（3）、计算 kalman 增益，（Q_t为测量精度的方差：高斯噪声，C_t为测量值和状态值的比例系数）

$$K_t=\frac{\overline{\sum_t}C_t^T}{C_t\overline{\sum}C_t^T+Q_t}$$

（4）、用测量值计算状态估计值

$$u_t=\overline{u_t}+K_t(z_t-C_t\overline{u_t})$$

（5）、计算当前状态的协方差用于下次迭代

$$\sum_t=(I-K_tC_t)\overline{\sum_t}$$

根据 kalman 公式来迭代计算 u_t 和 $\sum_t$

速度的估计值为 $\overline{z_t}$ =VtasPred

$$\overline{z_t} = \sqrt{(v_n - v_{gpsoffset.y} - v_{wn})^2 + (v_e - v_{gpsoffset.x} - v_{we})^2 + v_d{}^2} \qquad (1.1)$$

设 $A_t = I$,上一状态的协方差为 $\sum_{t-1} = P_{t-1}$，测量误差 $R_t = 0$，带入的（2）式，

$$\overline{\sum_t} = IP_{t-1}I^T = P_{t-1} \qquad (1.2)$$

设 $C_t = J_t$， $Q_t = R\_TAS$

$$K_t = \frac{\overline{\sum_t} J_t}{J_t \overline{\sum} J_t^T + Q_t} = \frac{P_{t-1}J_t}{J_t P_{t-1} J_t^T + Q_t} \qquad (1.3)$$

设测量速度为 z_t =VtasMeas，革新值为 innovVtas=VtasPred - VtasMeas;

EKF 状态矩阵的最优估计值=EKF 上一次的状态值+kalman 增益*革新值 （1.4）
states[j] = states[j] - Kfusion[j] * innovVtas;

计算当前状态的协方差，由（1.2）和（1.3）式可得

$$\sum_t = (I - K_t C_t)\overline{\sum_t} = (I - K_t J_t)P_{t-1} \qquad (1.5)$$

3、代码流程分析

```
void NavEKF::FuseAirspeed()
{
    .....
    //求测量精度的方差
    const float R_TAS = sq(constrain_float(_easNoise, 0.5f, 5.0f) * constrain_float(EAS2TAS, 0.9f, 10.0f));
    //计算预测的空气速度
    VtasPred = pythagorous3((ve - gpsVelGlitchOffset.y - vwe) , (vn - gpsVelGlitchOffset.x - vwn) , vd);
    //计算雅克比矩阵
    SH_TAS[0] = 1.0f/VtasPred;
    SH_TAS[2] = (SH_TAS[0]*(2*vn - 2*vwn))/2;   //y 轴的速度
    SH_TAS[1] = (SH_TAS[0]*(2*ve - 2*vwe))/2;   //x 轴的速度

    for (uint8_t i=0; i<=21; i++) H_TAS[i] = 0.0f;
    //机体速度值
    H_TAS[4] = SH_TAS[2];
    H_TAS[5] = SH_TAS[1];
```

```
H_TAS[6] = vd*SH_TAS[0];
//风的速度
H_TAS[14] = -SH_TAS[2];
H_TAS[15] = -SH_TAS[1];
```

//求 $J_t P_{t-1} J_t^T + Q_t$ 的值

```
float temp = (R_TAS + SH_TAS[2]*(P[4][4]*SH_TAS[2] + P[5][4]*SH_TAS[1] - P[14][4]*SH_TAS[2] - P[15][4]*SH_TAS[1] + P[6][4]*vd*SH_TAS[0])
                    + SH_TAS[1]*(P[4][5]*SH_TAS[2] + P[5][5]*SH_TAS[1] - P[14][5]*SH_TAS[2] - P[15][5]*SH_TAS[1] + P[6][5]*vd*SH_TAS[0])
                    - SH_TAS[2]*(P[4][14]*SH_TAS[2] + P[5][14]*SH_TAS[1] - P[14][14]*SH_TAS[2] - P[15][14]*SH_TAS[1] + P[6][14]*vd*SH_TAS[0])
                    - SH_TAS[1]*(P[4][15]*SH_TAS[2] + P[5][15]*SH_TAS[1] - P[14][15]*SH_TAS[2] - P[15][15]*SH_TAS[1] + P[6][15]*vd*SH_TAS[0])
                    + vd*SH_TAS[0]*(P[4][6]*SH_TAS[2] + P[5][6]*SH_TAS[1] - P[14][6]*SH_TAS[2] - P[15][6]*SH_TAS[1] + P[6][6]*vd*SH_TAS[0]));

//根据公式（1.3）求 kalman 增益
Kfusion[0] = SK_TAS*(P[0][4]*SH_TAS[2] - P[0][14]*SH_TAS[2] + P[0][5]*SH_TAS[1] - P[0][15]*SH_TAS[1] + P[0][6]*vd*SH_TAS[0]);
        Kfusion[1] = SK_TAS*(P[1][4]*SH_TAS[2] - P[1][14]*SH_TAS[2] + P[1][5]*SH_TAS[1] - P[1][15]*SH_TAS[1] + P[1][6]*vd*SH_TAS[0]);
        Kfusion[2] = SK_TAS*(P[2][4]*SH_TAS[2] - P[2][14]*SH_TAS[2] + P[2][5]*SH_TAS[1] - P[2][15]*SH_TAS[1] + P[2][6]*vd*SH_TAS[0]);
        Kfusion[3] = SK_TAS*(P[3][4]*SH_TAS[2] - P[3][14]*SH_TAS[2] + P[3][5]*SH_TAS[1] - P[3][15]*SH_TAS[1] + P[3][6]*vd*SH_TAS[0]);
        Kfusion[4] = SK_TAS*(P[4][4]*SH_TAS[2] - P[4][14]*SH_TAS[2] + P[4][5]*SH_TAS[1] - P[4][15]*SH_TAS[1] + P[4][6]*vd*SH_TAS[0]);
        Kfusion[5] = SK_TAS*(P[5][4]*SH_TAS[2] - P[5][14]*SH_TAS[2] + P[5][5]*SH_TAS[1] - P[5][15]*SH_TAS[1] + P[5][6]*vd*SH_TAS[0]);
        Kfusion[6] = SK_TAS*(P[6][4]*SH_TAS[2] - P[6][14]*SH_TAS[2] + P[6][5]*SH_TAS[1] - P[6][15]*SH_TAS[1] + P[6][6]*vd*SH_TAS[0]);
        Kfusion[7] = SK_TAS*(P[7][4]*SH_TAS[2] - P[7][14]*SH_TAS[2] + P[7][5]*SH_TAS[1] - P[7][15]*SH_TAS[1] + P[7][6]*vd*SH_TAS[0]);
        Kfusion[8] = SK_TAS*(P[8][4]*SH_TAS[2] - P[8][14]*SH_TAS[2] + P[8][5]*SH_TAS[1] - P[8][15]*SH_TAS[1] + P[8][6]*vd*SH_TAS[0]);
        Kfusion[9] = SK_TAS*(P[9][4]*SH_TAS[2] - P[9][14]*SH_TAS[2] + P[9][5]*SH_TAS[1] - P[9][15]*SH_TAS[1] + P[9][6]*vd*SH_TAS[0]);
        Kfusion[10] = SK_TAS*(P[10][4]*SH_TAS[2] - P[10][14]*SH_TAS[2] + P[10][5]*SH_TAS[1] - P[10][15]*SH_TAS[1] + P[10][6]*vd*SH_TAS[0]);
        Kfusion[11] = SK_TAS*(P[11][4]*SH_TAS[2] - P[11][14]*SH_TAS[2] +
```

```
P[11][5]*SH_TAS[1] - P[11][15]*SH_TAS[1] + P[11][6]*vd*SH_TAS[0]);
        Kfusion[12] = SK_TAS*(P[12][4]*SH_TAS[2] - P[12][14]*SH_TAS[2] +
P[12][5]*SH_TAS[1] - P[12][15]*SH_TAS[1] + P[12][6]*vd*SH_TAS[0]);
        // this term has been zeroed to improve stability of the Z accel bias
        Kfusion[13] = 0.0f;//SK_TAS*(P[13][4]*SH_TAS[2] - P[13][14]*SH_TAS[2] +
P[13][5]*SH_TAS[1] - P[13][15]*SH_TAS[1] + P[13][6]*vd*SH_TAS[0]);
        Kfusion[14] = SK_TAS*(P[14][4]*SH_TAS[2] - P[14][14]*SH_TAS[2] +
P[14][5]*SH_TAS[1] - P[14][15]*SH_TAS[1] + P[14][6]*vd*SH_TAS[0]);
        Kfusion[15] = SK_TAS*(P[15][4]*SH_TAS[2] - P[15][14]*SH_TAS[2] +
P[15][5]*SH_TAS[1] - P[15][15]*SH_TAS[1] + P[15][6]*vd*SH_TAS[0]);
        // zero Kalman gains to inhibit magnetic field state estimation
        if (!inhibitMagStates) {
            Kfusion[16] = SK_TAS*(P[16][4]*SH_TAS[2] - P[16][14]*SH_TAS[2] +
P[16][5]*SH_TAS[1] - P[16][15]*SH_TAS[1] + P[16][6]*vd*SH_TAS[0]);
            Kfusion[17] = SK_TAS*(P[17][4]*SH_TAS[2] - P[17][14]*SH_TAS[2] +
P[17][5]*SH_TAS[1] - P[17][15]*SH_TAS[1] + P[17][6]*vd*SH_TAS[0]);
            Kfusion[18] = SK_TAS*(P[18][4]*SH_TAS[2] - P[18][14]*SH_TAS[2] +
P[18][5]*SH_TAS[1] - P[18][15]*SH_TAS[1] + P[18][6]*vd*SH_TAS[0]);
            Kfusion[19] = SK_TAS*(P[19][4]*SH_TAS[2] - P[19][14]*SH_TAS[2] +
P[19][5]*SH_TAS[1] - P[19][15]*SH_TAS[1] + P[19][6]*vd*SH_TAS[0]);
            Kfusion[20] = SK_TAS*(P[20][4]*SH_TAS[2] - P[20][14]*SH_TAS[2] +
P[20][5]*SH_TAS[1] - P[20][15]*SH_TAS[1] + P[20][6]*vd*SH_TAS[0]);
            Kfusion[21] = SK_TAS*(P[21][4]*SH_TAS[2] - P[21][14]*SH_TAS[2] +
P[21][5]*SH_TAS[1] - P[21][15]*SH_TAS[1] + P[21][6]*vd*SH_TAS[0]);
        } else {
            for (uint8_t i=16; i<=21; i++) {
                Kfusion[i] = 0.0f;
            }
        }
    //用预测的空气速度减去真实的空气速度得到速度的革新值
    innovVtas = VtasPred - VtasMeas;
     //更新 ekf 状态矩阵
     for (uint8_t j=0; j<=21; j++)
     {
            states[j] = states[j] - Kfusion[j] * innovVtas;
      }
    //根据公式(1.5)计算当前协方差
    for (uint8_t i = 0; i<=21; i++)
    {
        for (uint8_t j = 0; j<=3; j++) KH[i][j] = 0.0f;
        for (uint8_t j = 4; j<=6; j++)
        {
            KH[i][j] = Kfusion[i] * H_TAS[j];
        }
```

```
            for (uint8_t j = 7; j<=13; j++) KH[i][j] = 0.0f;
            for (uint8_t j = 14; j<=15; j++)
            {
                    KH[i][j] = Kfusion[i] * H_TAS[j];
             }
            for (uint8_t j = 16; j<=21; j++) KH[i][j] = 0.0f;
        }
        for (uint8_t i = 0; i<=21; i++)
        {
             for (uint8_t j = 0; j<=21; j++)
             {
                    KHP[i][j] = 0;
                    for (uint8_t k = 4; k<=6; k++)
                    {
                        KHP[i][j] = KHP[i][j] + KH[i][k] * P[k][j];
                    }
                   for (uint8_t k = 14; k<=15; k++)
                    {
                            KHP[i][j] = KHP[i][j] + KH[i][k] * P[k][j];
                        }
                }
            }
            for (uint8_t i = 0; i<=21; i++)
            {
                    for (uint8_t j = 0; j<=21; j++)
                    {
                        P[i][j] = P[i][j] - KHP[i][j];
                     }
            }
}
```

第三节 磁力计 ekf 算法分析

1、以下是我在网上看到的一篇关于四元素和旋转矩阵相关的文章，代码中运用到了其中的公式，贴在这里

Quaternions and other representations of rotations
[edit] Qualitative description of the advantages of quaternions

The representation of a rotation as a quaternion (4 numbers) is more compact than the representation as an orthogonal matrix (9 numbers). Furthermore, for a given axis and angle, one can easily construct the corresponding quaternion, and conversely, for a given quaternion one can

easily read off the axis and the angle. Both of these are much harder with matrices or Euler angles.

In computer games and other applications, one is often interested in "smooth rotations", meaning that the scene should slowly rotate and not in a single step. This can be accomplished by choosing a curve such as the spherical linear interpolation in the quaternions, with one endpoint being the identity transformation 1 (or some other initial rotation) and the other being the intended final rotation. This is more problematic with other representations of rotations.

When composing several rotations on a computer, rounding errors necessarily accumulate. A quaternion that's slightly off still represents a rotation after being normalised— a matrix that's slightly off may not be orthogonal anymore and is harder to convert back to a proper orthogonal matrix.

Quaternions also avoid a phenomenon called gimbal lock which can result when, for example in pitch/yaw/roll rotational systems, the pitch is rotated 90° up or down, so that yaw and roll then correspond to the same motion, and a degree of freedom of rotation is lost. In a gimbal-based aerospace inertial navigation system, for instance, this could have disastrous results if the aircraft is in a steep dive or ascent.

[edit] Conversion to and from the matrix representation
[edit] From a quaternion to an orthogonal matrix

The orthogonal matrix corresponding to a rotation by the unit quaternion z = a + bi + cj + dk (with |z| = 1) is given by

$$\begin{pmatrix} a^2+b^2-c^2-d^2 & 2bc-2ad & 2bd+2ac \\ 2bc+2ad & a^2-b^2+c^2-d^2 & 2cd-2ab \\ 2bd-2ac & 2cd+2ab & a^2-b^2-c^2+d^2 \end{pmatrix}.$$

[edit] From an orthogonal matrix to a quaternion

Finding the quaternion (q0 + qxi + qyj + qzk) that corresponds to a rotation matrix Qij can be numerically unstable if the trace (sum of the diagonal elements) of the rotation matrix is zero or very small. A robust method is to choose the diagonal element with the largest absolute value (Quu). The value of

$$r = \sqrt{1 + Q_{uu} - Q_{vv} - Q_{ww}}$$

will certainly be a real number[dubious - discuss], where uvw is an even permutation of xyz (i.e. xyz, yzx or zxy). If r is zero the matrix is the identity matrix, and the quaternion must be the identity quaternion (1, 0, 0, 0). Otherwise the quaternion can be calculated as follows:[citation needed]

$$\begin{aligned} q_0 &= \frac{Q_{wv} - Q_{vw}}{2r} \\ q_u &= r/2 \\ q_v &= \frac{Q_{uv} + Q_{vu}}{2r} \\ q_w &= \frac{Q_{wu} + Q_{uw}}{2r} \end{aligned}$$

Beware the vector convention: There are two conventions for rotation matrices: one assumes row vectors on the left; the other assumes column vectors on the right; the two conventions generate matrices that are the transpose of each other. The above matrix assumes column vectors on the right. In general, a matrix for vertex transpose is ambiguous unless the vector convention is also mentioned. Historically, the column-on-the-right convention comes from mathematics and classical mechanics, whereas row-vector-on-the-left comes from computer graphics, where typesetting row vectors was easier back in the early days.

(Compare the equivalent general formula for a 3 × 3 rotation matrix in terms of the axis and the angle.)

[edit] Fitting quaternions

The above section described how to recover a quaternion q from a 3 × 3 rotation matrix Q. Suppose, however, that we have some matrix Q that is not a pure rotation — due to round-off errors, for example — and we wish to find the quaternion q that most accurately represents Q. In that case we construct a symmetric 4×4 matrix

$$K = \frac{1}{3}\begin{bmatrix} Q_{xx} - Q_{yy} - Q_{zz} & Q_{yx} + Q_{xy} & Q_{zx} + Q_{xz} & Q_{yz} - Q_{zy} \\ Q_{yx} + Q_{xy} & Q_{yy} - Q_{xx} - Q_{zz} & Q_{zy} + Q_{yz} & Q_{zx} - Q_{xz} \\ Q_{zx} + Q_{xz} & Q_{zy} + Q_{yz} & Q_{zz} - Q_{xx} - Q_{yy} & Q_{xy} - Q_{yx} \\ Q_{yz} - Q_{zy} & Q_{zx} - Q_{xz} & Q_{xy} - Q_{yx} & Q_{xx} + Q_{yy} + Q_{zz} \end{bmatrix},$$

and find the eigenvector (x, y, z, w) corresponding to the largest eigenvalue (that value will be 1 if and only if Q is a pure rotation). The quaternion so obtained will correspond to the rotation closest to the original matrix Q[dubious – discuss][3]

2、Ekf 的 22 维状态变量：

$$X=[\delta q_0,\delta q_1,\delta q_2,\delta q_3,\delta v_x,\delta v_y,\delta v_z,\delta p_x,\delta p_y,\delta p_z,\delta g_x,\delta g_y,\delta g_z,\delta a_1,$$
$$\delta w_x,\delta w_y,\delta em_x,\delta em_y,\delta em_z,\delta bm_x,\delta bm_y,\delta bm_z]$$

该状态方程的变量取自 state_elements 结构体的 0-21 项

```
struct state_elements {
        Quaternion    quat;              // 0..3        四元素
        Vector3f      velocity;          // 4..6        速度值
        Vector3f      position;          // 7..9        位置值
        Vector3f      gyro_bias;         // 10..12      陀螺仪的值
        float         accel_zbias1;      // 13          加速度计值
        Vector2f      wind_vel;          // 14..15      风的速度
        Vector3f      earth_magfield;    // 16..18      地磁场值
        Vector3f      body_magfield;     // 19..21      机体磁场值
        float         accel_zbias2;      // 22
        Vector3f      vel1;              // 23 .. 25
        float         posD1;             // 26
        Vector3f      vel2;              // 27 .. 29
        float         posD2;             // 30
        Vector3f      omega;             // 31 .. 33
    } &state;
```

22 状态中与指南针相关的状态为四元素和地磁场的值，机体磁场值

3、磁力计 ekf 原理分析

从以上文章可知，一个三维向量转到另一个三维向量用四元素表示的旋转矩阵为 R_{dcm}

$$R_{dcm}=\begin{matrix} q_0^2+q_1^2-q_2^2-q_3^2 & 2q_1q_2-2q_0q_3 & 2q_1q_3+2q_0q_2 \\ 2q_1q_2+2q_0q_3 & q_0^2-q_1^2+q_2^2-q_3^2 & 2q_2q_3-2q_0q_1 \\ 2q_1q_3-2q_0q_2 & 2q_2q_3+2q_0q_1 & q_0^2-q_1^2-q_2^2+q_3^2 \end{matrix} \quad (2.1)$$

设估计磁感值为 M_p=Mag_Pred，地磁场的磁感值为 M，机体磁场相对于地磁场的相对磁感值为 $M_{B\to e}$

$$M_p=[m_x \quad m_y \quad m_z]$$

$$M = [magN \quad magE \quad magD]$$

$$M_{b\to e} = [magXbias \quad magYbias \quad magZbias]^T$$

$$M_p = M * R_{dcm} + M_{b\to e} \qquad (2.2)$$

这里计算的 M_p 相当于 kalman 中的 $\overline{z_t}$，代码是分 X、Y、Z 分别融合进状态表

X 轴的融合
从（2.1）（2.2）式可以得到

$$m_x = (q_0^{\ 2} + q_1^{\ 2} - q_2^{\ 2} - q_3^{\ 2})magN + 2(q_1q_2 + q_0q_3)magE + 2(q_1q_3 - q_0q_2)magD + magXbias$$

$$\frac{\partial m_x}{\partial q_0} = 2q_0magN + 2q_3magE - 2q_2magD$$

$$\frac{\partial m_x}{\partial q_1} = 2q_1magN + 2q_2magE + 2q_3magD$$

$$\frac{\partial m_x}{\partial q_2} = -2q_2magN + 2q_1magE - 2q_0magD$$

$$\frac{\partial m_x}{\partial q_3} = -2q_3magN + 2q_0magE + 2q_1magD$$

$$\frac{\partial m_x}{\partial magN} = q_0^{\ 2} + q_1^{\ 2} - q_2^{\ 2} - q_3^{\ 2}$$

$$\frac{\partial m_x}{\partial magE} = 2(q_1q_2 + q_0q_3)$$

$$\frac{\partial m_x}{magD} = 2(q_1q_3 - q_0q_2)$$

由于现在做的是机体 X 轴上的融合，所以

$$\frac{\partial X}{\partial bm_x} = 1$$

$$\frac{\partial X}{\partial bm_y} = 0$$

$$\frac{\partial X}{\partial bm_z} = 0$$

X 轴上的 jacobian 矩阵为

$$J_x=[\frac{\partial m_x}{\partial q_0},\frac{\partial m_x}{\partial q_1},\frac{\partial m_x}{\partial q_2},\frac{\partial m_x}{\partial q_3},0,0,0,0,0,0,0,0,0,0,0,0,\frac{\partial m_x}{\partial magN},\frac{\partial m_x}{\partial magE},\frac{\partial m_x}{\partial magD},\frac{\partial X}{\partial bm_x},0,0]$$

（2.3）

设 kalman 中的 $A_t=I$， $R_t=0$, $Q_t=R\_MAG$， $C_t=J_x$

$$K_t=\frac{\overline{\sum_t}C_t^T}{C_t\overline{\sum}C_t^T+Q_t}=\frac{P_{t-1}J_x{}^T}{J_xP_{t-1}J_x{}^T+Q_t}$$ （2.4）

计算革新值

革新值=$\overline{z_t}-z_t$ （2.5）

更新状态方程

EKF 状态矩阵的最优估计值=EKF 上一次的状态值+kalman 增益*革新值； （2.6）

计算当前状态协方差

$$\sum_t=(I-K_tC_t)\overline{\sum_t}=(I-K_tJ_x)P_{t-1}$$ （2.7）

Y 轴和 Z 轴的融合与 X 轴的原理一致

4、代码分析

```
void NavEKF::FuseMagnetometer()
{
    if (obsIndex == 0)//融合 X 轴
    {
        //用四元素求出旋转矩阵，见（2.1）
        DCM[0][0] = q0*q0 + q1*q1 - q2*q2 - q3*q3;
        DCM[0][1] = 2*(q1*q2 + q0*q3);
        DCM[0][2] = 2*(q1*q3-q0*q2);
        DCM[1][0] = 2*(q1*q2 - q0*q3);
        DCM[1][1] = q0*q0 - q1*q1 + q2*q2 - q3*q3;
        DCM[1][2] = 2*(q2*q3 + q0*q1);
        DCM[2][0] = 2*(q1*q3 + q0*q2);
        DCM[2][1] = 2*(q2*q3 - q0*q1);
        DCM[2][2] = q0*q0 - q1*q1 - q2*q2 + q3*q3;
        //求出预测指南针的值，见（2.2）
        MagPred[0] = DCM[0][0]*magN + DCM[0][1]*magE  + DCM[0][2]*magD +
    magXbias;
        MagPred[1] = DCM[1][0]*magN + DCM[1][1]*magE  + DCM[1][2]*magD +
    magYbias;
        MagPred[2] = DCM[2][0]*magN + DCM[2][1]*magE  + DCM[2][2]*magD +
    magZbias;
        //把第一次计算的值记录起来，供 Y、Z 轴使用
        SH_MAG[0] = 2*magD*q3 + 2*magE*q2 + 2*magN*q1;
        SH_MAG[1] = 2*magD*q0 - 2*magE*q1 + 2*magN*q2;
        SH_MAG[2] = 2*magD*q1 + 2*magE*q0 - 2*magN*q3;
```

```
SH_MAG[3] = sq(q3);
SH_MAG[4] = sq(q2);
SH_MAG[5] = sq(q1);
SH_MAG[6] = sq(q0);
SH_MAG[7] = 2*magN*q0;
SH_MAG[8] = 2*magE*q3;
for (uint8_t i=0; i<=21; i++) H_MAG[i] = 0;
//求出四元素的偏导值
H_MAG[0] = SH_MAG[7] + SH_MAG[8] - 2*magD*q2;
H_MAG[1] = SH_MAG[0];
H_MAG[2] = 2*magE*q1 - 2*magD*q0 - 2*magN*q2;
H_MAG[3] = SH_MAG[2];
//求出地球坐标的偏导值
H_MAG[16] = SH_MAG[5] - SH_MAG[4] - SH_MAG[3] + SH_MAG[6];
H_MAG[17] = 2*q0*q3 + 2*q1*q2;
H_MAG[18] = 2*q1*q3 - 2*q0*q2;
//求出机体坐标的偏导值，由于 Y 和 Z 与 X 轴垂直，所以偏导都为 0
H_MAG[19] = 1;
H_MAG[20] = 0;
H_MAG[21] = 0;
//计算 kalman 增益的分母见（2.4），这里我对之前的代码进行了整理，这样可以更好的看出规律
float temp =R_MAG-
-(2*magD*q0 - 2*magE*q1 + 2*magN*q2)*(P[19][2]*1 + P[1][2]*SH_MAG[0] + P[3][2]*SH_MAG[2] - P[16][2]*(SH_MAG[3] + SH_MAG[4] - SH_MAG[5] - SH_MAG[6])+ P[17][2]*(2*q0*q3 + 2*q1*q2) - P[18][2]*(2*q0*q2 - 2*q1*q3) - P[2][2]*(2*magD*q0 - 2*magE*q1 + 2*magN*q2) + P[0][2]*(SH_MAG[7] + SH_MAG[8] - 2*magD*q2))
+(SH_MAG[7] + SH_MAG[8] - 2*magD*q2)*(P[19][0]*1 + P[1][0]*SH_MAG[0] + P[3][0]*SH_MAG[2] - P[16][0]*(SH_MAG[3] + SH_MAG[4] - SH_MAG[5] - SH_MAG[6]) + P[17][0]*(2*q0*q3 + 2*q1*q2)- P[18][0]*(2*q0*q2 - 2*q1*q3) - P[2][0]*(2*magD*q0 - 2*magE*q1 + 2*magN*q2) + P[0][0]*(SH_MAG[7] + SH_MAG[8] - 2*magD*q2))
+SH_MAG[0]*(P[19][1]*1 + P[1][1]*SH_MAG[0] + P[3][1]*SH_MAG[2] - P[16][1]*(SH_MAG[3] + SH_MAG[4] - SH_MAG[5] - SH_MAG[6]) + P[17][1]*(2*q0*q3 + 2*q1*q2) - P[18][1]*(2*q0*q2 - 2*q1*q3) - P[2][1]*(2*magD*q0 - 2*magE*q1 + 2*magN*q2) + P[0][1]*(SH_MAG[7] + SH_MAG[8] - 2*magD*q2))
+SH_MAG[2]*(P[19][3]*1 + P[1][3]*SH_MAG[0] + P[3][3]*SH_MAG[2] - P[16][3]*(SH_MAG[3] + SH_MAG[4] - SH_MAG[5] - SH_MAG[6]) + P[17][3]*(2*q0*q3 + 2*q1*q2) - P[18][3]*(2*q0*q2 - 2*q1*q3)- P[2][3]*(2*magD*q0 - 2*magE*q1 + 2*magN*q2) + P[0][3]*(SH_MAG[7] + SH_MAG[8] - 2*magD*q2))
-(SH_MAG[3] + SH_MAG[4] - SH_MAG[5] - SH_MAG[6])*(P[19][16]*1 +
```

P[1][16]*SH_MAG[0] + P[3][16]*SH_MAG[2] - P[16][16]*(SH_MAG[3] + SH_MAG[4] - SH_MAG[5] - SH_MAG[6]) + P[17][16]*(2*q0*q3 + 2*q1*q2) - P[18][16]*(2*q0*q2 - 2*q1*q3) - P[2][16]*(2*magD*q0 - 2*magE*q1 + 2*magN*q2) + P[0][16]*(SH_MAG[7] + SH_MAG[8] - 2*magD*q2))

+(2*q0*q3 + 2*q1*q2)*(P[19][17]*1 + P[1][17]*SH_MAG[0] + P[3][17]*SH_MAG[2] - P[16][17]*(SH_MAG[3] + SH_MAG[4] - SH_MAG[5] - SH_MAG[6]) + P[17][17]*(2*q0*q3 + 2*q1*q2) - P[18][17]*(2*q0*q2 - 2*q1*q3) - P[2][17]*(2*magD*q0 - 2*magE*q1 + 2*magN*q2) + P[0][17]*(SH_MAG[7] + SH_MAG[8] - 2*magD*q2))

-(2*q0*q2 - 2*q1*q3)*(P[19][18] + P[1][18]*SH_MAG[0] + P[3][18]*SH_MAG[2] - P[16][18]*(SH_MAG[3] + SH_MAG[4] - SH_MAG[5] - SH_MAG[6]) + P[17][18]*(2*q0*q3 + 2*q1*q2) - P[18][18]*(2*q0*q2 - 2*q1*q3) - P[2][18]*(2*magD*q0 - 2*magE*q1 + 2*magN*q2) + P[0][18]*(SH_MAG[7] + SH_MAG[8] - 2*magD*q2))

1*(P[19][19]*1 + P[1][19]*SH_MAG[0] + P[3][19]*SH_MAG[2] - P[16][19]*(SH_MAG[3] + SH_MAG[4] - SH_MAG[5] - SH_MAG[6]) + P[17][19]*(2*q0*q3 + 2*q1*q2) - P[18][19]*(2*q0*q2 - 2*q1*q3) - P[2][19]*(2*magD*q0 - 2*magE*q1 + 2*magN*q2) + P[0][19]*(SH_MAG[7] + SH_MAG[8] - 2*magD*q2))

//计算 kalman 增益

Kfusion[0] = SK_MX[0]*(P[0][19] + P[0][1]*SH_MAG[0] + P[0][3]*SH_MAG[2] + P[0][0]*SK_MX[3] - P[0][2]*SK_MX[2] - P[0][16]*SK_MX[1] + P[0][17]*SK_MX[5] - P[0][18]*SK_MX[4]);

Kfusion[1] = SK_MX[0]*(P[1][19] + P[1][1]*SH_MAG[0] + P[1][3]*SH_MAG[2] + P[1][0]*SK_MX[3] - P[1][2]*SK_MX[2] - P[1][16]*SK_MX[1] + P[1][17]*SK_MX[5] - P[1][18]*SK_MX[4]);

Kfusion[2] = SK_MX[0]*(P[2][19] + P[2][1]*SH_MAG[0] + P[2][3]*SH_MAG[2] + P[2][0]*SK_MX[3] - P[2][2]*SK_MX[2] - P[2][16]*SK_MX[1] + P[2][17]*SK_MX[5] - P[2][18]*SK_MX[4]);

Kfusion[3] = SK_MX[0]*(P[3][19] + P[3][1]*SH_MAG[0] + P[3][3]*SH_MAG[2] + P[3][0]*SK_MX[3] - P[3][2]*SK_MX[2] - P[3][16]*SK_MX[1] + P[3][17]*SK_MX[5] - P[3][18]*SK_MX[4]);

Kfusion[4] = SK_MX[0]*(P[4][19] + P[4][1]*SH_MAG[0] + P[4][3]*SH_MAG[2] + P[4][0]*SK_MX[3] - P[4][2]*SK_MX[2] - P[4][16]*SK_MX[1] + P[4][17]*SK_MX[5] - P[4][18]*SK_MX[4]);

Kfusion[5] = SK_MX[0]*(P[5][19] + P[5][1]*SH_MAG[0] + P[5][3]*SH_MAG[2] + P[5][0]*SK_MX[3] - P[5][2]*SK_MX[2] - P[5][16]*SK_MX[1] + P[5][17]*SK_MX[5] - P[5][18]*SK_MX[4]);

Kfusion[6] = SK_MX[0]*(P[6][19] + P[6][1]*SH_MAG[0] + P[6][3]*SH_MAG[2] + P[6][0]*SK_MX[3] - P[6][2]*SK_MX[2] - P[6][16]*SK_MX[1] + P[6][17]*SK_MX[5] - P[6][18]*SK_MX[4]);

Kfusion[7] = SK_MX[0]*(P[7][19] + P[7][1]*SH_MAG[0] + P[7][3]*SH_MAG[2] + P[7][0]*SK_MX[3] - P[7][2]*SK_MX[2] - P[7][16]*SK_MX[1] + P[7][17]*SK_MX[5] - P[7][18]*SK_MX[4]);

```
    Kfusion[8] = SK_MX[0]*(P[8][19] + P[8][1]*SH_MAG[0] + P[8][3]*SH_MAG[2] + P[8][0]*SK_MX[3] - P[8][2]*SK_MX[2] - P[8][16]*SK_MX[1] + P[8][17]*SK_MX[5] - P[8][18]*SK_MX[4]);
    Kfusion[9] = SK_MX[0]*(P[9][19] + P[9][1]*SH_MAG[0] + P[9][3]*SH_MAG[2] + P[9][0]*SK_MX[3] - P[9][2]*SK_MX[2] - P[9][16]*SK_MX[1] + P[9][17]*SK_MX[5] - P[9][18]*SK_MX[4]);
    Kfusion[10] = SK_MX[0]*(P[10][19] + P[10][1]*SH_MAG[0] + P[10][3]*SH_MAG[2] + P[10][0]*SK_MX[3] - P[10][2]*SK_MX[2] - P[10][16]*SK_MX[1] + P[10][17]*SK_MX[5] - P[10][18]*SK_MX[4]);
    Kfusion[11] = SK_MX[0]*(P[11][19] + P[11][1]*SH_MAG[0] + P[11][3]*SH_MAG[2] + P[11][0]*SK_MX[3] - P[11][2]*SK_MX[2] - P[11][16]*SK_MX[1] + P[11][17]*SK_MX[5] - P[11][18]*SK_MX[4]);
     Kfusion[12] = SK_MX[0]*(P[12][19] + P[12][1]*SH_MAG[0] + P[12][3]*SH_MAG[2] + P[12][0]*SK_MX[3] - P[12][2]*SK_MX[2] - P[12][16]*SK_MX[1] + P[12][17]*SK_MX[5] - P[12][18]*SK_MX[4]);
     // this term has been zeroed to improve stability of the Z accel bias
     Kfusion[13] = 0.0f;//SK_MX[0]*(P[13][19] + P[13][1]*SH_MAG[0] + P[13][3]*SH_MAG[2] + P[13][0]*SK_MX[3] - P[13][2]*SK_MX[2] - P[13][16]*SK_MX[1] + P[13][17]*SK_MX[5] - P[13][18]*SK_MX[4]);
     // zero Kalman gains to inhibit wind state estimation
     if (!inhibitWindStates) {
          Kfusion[14] = SK_MX[0]*(P[14][19] + P[14][1]*SH_MAG[0] + P[14][3]*SH_MAG[2] + P[14][0]*SK_MX[3] - P[14][2]*SK_MX[2] - P[14][16]*SK_MX[1] + P[14][17]*SK_MX[5] - P[14][18]*SK_MX[4]);
          Kfusion[15] = SK_MX[0]*(P[15][19] + P[15][1]*SH_MAG[0] + P[15][3]*SH_MAG[2] + P[15][0]*SK_MX[3] - P[15][2]*SK_MX[2] - P[15][16]*SK_MX[1] + P[15][17]*SK_MX[5] - P[15][18]*SK_MX[4]);
    } else {
          Kfusion[14] = 0.0f;
          Kfusion[15] = 0.0f;
       }
      // zero Kalman gains to inhibit magnetic field state estimation
      if (!inhibitMagStates) {
          Kfusion[16] = SK_MX[0]*(P[16][19] + P[16][1]*SH_MAG[0] + P[16][3]*SH_MAG[2] + P[16][0]*SK_MX[3] - P[16][2]*SK_MX[2] - P[16][16]*SK_MX[1] + P[16][17]*SK_MX[5] - P[16][18]*SK_MX[4]);
          Kfusion[17] = SK_MX[0]*(P[17][19] + P[17][1]*SH_MAG[0] + P[17][3]*SH_MAG[2] + P[17][0]*SK_MX[3] - P[17][2]*SK_MX[2] - P[17][16]*SK_MX[1] + P[17][17]*SK_MX[5] - P[17][18]*SK_MX[4]);
          Kfusion[18] = SK_MX[0]*(P[18][19] + P[18][1]*SH_MAG[0] + P[18][3]*SH_MAG[2] + P[18][0]*SK_MX[3] - P[18][2]*SK_MX[2] - P[18][16]*SK_MX[1] + P[18][17]*SK_MX[5] - P[18][18]*SK_MX[4]);
          Kfusion[19] = SK_MX[0]*(P[19][19] + P[19][1]*SH_MAG[0] + P[19][3]*SH_MAG[2] + P[19][0]*SK_MX[3] - P[19][2]*SK_MX[2] - P[19][16]*SK_MX[1] +
```

```
P[19][17]*SK_MX[5] - P[19][18]*SK_MX[4]);
                Kfusion[20] = SK_MX[0]*(P[20][19] + P[20][1]*SH_MAG[0] + P[20][3]*SH_MAG[2] + P[20][0]*SK_MX[3] - P[20][2]*SK_MX[2] - P[20][16]*SK_MX[1] + P[20][17]*SK_MX[5] - P[20][18]*SK_MX[4]);
                Kfusion[21] = SK_MX[0]*(P[21][19] + P[21][1]*SH_MAG[0] + P[21][3]*SH_MAG[2] + P[21][0]*SK_MX[3] - P[21][2]*SK_MX[2] - P[21][16]*SK_MX[1] + P[21][17]*SK_MX[5] - P[21][18]*SK_MX[4]);
            } else {
                for (uint8_t i=16; i<=21; i++) {
                    Kfusion[i] = 0.0f;
                }
            }

        }
        else if (obsIndex == 1)
        {
            //Y 轴的融合
        }
        else if (obsIndex == 2)
        {
            //Z 轴的融合
        }

        //计算革新值
         innovMag[obsIndex] = MagPred[obsIndex] - magData[obsIndex];

        //更新状态方程
        states[j] = states[j] - Kfusion[j] * innovMag[obsIndex];
        //计算当前协方差
        for (uint8_t i = 0; i<=21; i++) {
            for (uint8_t j = 0; j<=3; j++) {
                KH[i][j] = Kfusion[i] * H_MAG[j];
            }
            for (uint8_t j = 4; j<=15; j++) {
                KH[i][j] = 0.0f;
            }
            if (!inhibitMagStates) {
                for (uint8_t j = 16; j<=21; j++) {
                    KH[i][j] = Kfusion[i] * H_MAG[j];
                }
            } else {
                for (uint8_t j = 16; j<=21; j++) {
                    KH[i][j] = 0.0f;
                }
```

```
            }
        }
        for (uint8_t i = 0; i<=21; i++) {
            for (uint8_t j = 0; j<=21; j++) {
                KHP[i][j] = 0;
                for (uint8_t k = 0; k<=3; k++) {
                    KHP[i][j] = KHP[i][j] + KH[i][k] * P[k][j];
                }
                if (!inhibitMagStates) {
                    for (uint8_t k = 16; k<=21; k++) {
                        KHP[i][j] = KHP[i][j] + KH[i][k] * P[k][j];
                    }
                }
            }
        }
        for (uint8_t i = 0; i<=21; i++) {
            for (uint8_t j = 0; j<=21; j++) {
                P[i][j] = P[i][j] - KHP[i][j];
            }
        }
    }
}
```

第四节 速度和位置的 kalman filter 算法

1、速度和位置的融合为什么用 Kalman Filter，二不用 ekf

前两节分析的空速传感器和磁力计，用的是 Extended Kalman Filter ，而本节分析的速度位置的融合所用的是 Kalman Filter。怎么这么说呢？求 Extended Kalman Filter 的思路是用泰勒级数展开，取一阶偏导（高阶偏导舍去），按照 Kalman Filter 的思路求解。前两节都求了雅克比偏导数，就是泰勒级数展开的一阶偏导。

速度和位置的融合的传感器是 gps、气压计和光流传感器，它们测试的值直接是各个方向的速度和位置，正好满足 Kalman Filter 的三个前提条件：

当前状态的概率分布必须是上一状态和将要执行的控制量的**线性函数**，再叠加一个高斯噪声；

对状态的测量必须是状态的**线性函数**叠加高斯噪声；

初始状态分布为高斯分布；

2、Kalman Filter 的 34 维状态变量：

```
struct state_elements {
        Quaternion  quat;           // 0..3     四元素
        Vector3f    velocity;       // 4..6     速度值
```

```
    Vector3f    position;           // 7..9        位置值
    Vector3f    gyro_bias;          // 10..12      陀螺仪的值
    float       accel_zbias1;       // 13          加速度计 1 Z 轴的值
    Vector2f    wind_vel;           // 14..15      风的速度
    Vector3f    earth_magfield; // 16..18          地磁场值
    Vector3f    body_magfield;    // 19..21        机体磁场值
    float       accel_zbias2;       // 22          加速度计 2 Z 轴的值
    Vector3f    vel1;               // 23 .. 25    IMU1 加速度计积分的速度
    float       posD1;              // 26          IMU1 加速度计积分的高度
    Vector3f    vel2;               // 27 .. 29    IMU2 加速度计积分的速度
    float       posD2;              // 30          IMU2 计速度计积分的高度
    Vector3f    omega;              // 31 .. 33
} &state;
```

State 34 个状态中跟速度和位置相关的有 velocity、position、accel_zbias1、accel_zbias2、vel1、posD1、vel2、posD2。

3、Kalman filter 的五条黄金公式

（6）、通过上一状态最优值（u_{t-1}）和将要施加的控制量（u_t）来预测当前状态 $\bar{u}_t$.

$$\overline{u_t} = A_t u_{t-1} + B_t u_t$$

（7）、用上一状态的协方差（$\sum_{t-1}$ ）和高斯噪声（预测差值的方差）来计算预测值的协方差

$$\overline{\sum_t} = A_t \sum_{t-1} A_t^T + R_t$$

（8）、计算 kalman 增益，（Q_t 为测量精度的方差：高斯噪声，C_t 为测量值和状态值的比例系数）

$$K_t = \frac{\overline{\sum_t} C_t^T}{C_t \overline{\sum} C_t^T + Q_t}$$

（9）、用测量值计算状态估计值

$$u_t = \overline{u_t} + K_t(z_t - C_t \overline{u_t})$$

（10）、计算当前状态的协方差用于下次迭代

$$\sum_t = (I - K_t C_t)\overline{\sum_t}$$

在这里我对 kalman filter 的黄金五公式进行简化

设 $A_t = I$， $B_t = 0$， $C_t = I$， $R_t = 0$， $\sum_{t-1} = P_{t-1}$

简化的 kalman filter 如下

（1）、$\overline{u_t} = u_{t-1}$ （3.1）

（2）、$\overline{\sum_t} = \sum_{t-1} = P_{t-1}$ （3.2）

（3）、$K_t = \frac{P_{t-1}}{P_{t-1} + Q_t}$ （3.3）

（4）、$u_t = \overline{u_t} + K_t(z_t - C_t\overline{u_t}) = u_{t-1} - K_t(u_{t-1} - z_t)$ （3.4）

（5）、$\sum_t = (I - K_t)P_{t-1}$ （3.5）

4、代码分析

（分析代码的前提是在 gps 和光流传感器至少一个有用的情况下，以下式中的$\overline{u_t}$就等于$\overline{z_t}$）

代码中分速度的融合和 XY 轴位置的融合以及高度的融合，所以有 3 个$\overline{u_t}$，分别是 statesAtVelTime、statesAtPosTime、statesAtHgtTime

在 gps 和光流传感器至少一个有效的情况下，3 个$\overline{u_t}$的获取

```
void NavEKF::SelectVelPosFusion()
{
    .....
    readGpsData();//读 gps 的值
    ....
    readHgtData();//读气压计的值
    ......
    FuseVelPosNED();//进行速度、位置、高度的融合
    ......
    //用 kalman filter  计算的修正值对四元素、速度和位置进行修正
    if (gpsUpdateCount < gpsUpdateCountMax) {
        gpsUpdateCount ++;
        for (uint8_t i = 0; i <= 9; i++) {
            states[i] += gpsIncrStateDelta[i];
        }
    }
    if (hgtUpdateCount < hgtUpdateCountMax) {
        hgtUpdateCount ++;
        for (uint8_t i = 0; i <= 9; i++) {
            states[i] += hgtIncrStateDelta[i];
        }
    }
}

void NavEKF::readGpsData()//读 gps 的值
{
```

```
    .......
    //获取估计速度（其实就是最近一次的速度）
    RecallStates(statesAtVelTime, (imuSampleTime_ms -
                                        constrain_int16(_msecVelDelay, 0, 500)));
    //获取估计位置（其实就是最近一次的位置）
    RecallStates(statesAtPosTime, (imuSampleTime_ms -
                                        constrain_int16(_msecPosDelay, 0, 500)));
    ......
    //获得 gps 速度测量值
    velNED = _ahrs->get_gps().velocity();
    ......
    //获取 gps 位置德测量值
    gpsPosNE = location_diff(EKF_origin, gpsloc);
    ....
}

void NavEKF::readHgtData()//读气压计的值
{
    .......
    //气压计对高度测量值
    hgtMea = _baro.get_altitude();
    .....
    //获取估计的高度（其实就是最近一次的高度）
    RecallStates(statesAtHgtTime, (imuSampleTime_ms - msecHgtDelay));
    ......
}
void NavEKF::FuseVelPosNED()
{
    //imu 速度积分的融合
    Vector3f velInnov;//通过权重对 imu1 和 imu2 的混合积分的速度
    Vector3f velInnov1; //imu1 积分的速度
    Vector3f velInnov2; //imu2  积分的速度
    //是否参与融合标识位
    /*
    **    fuseData[0]: X 轴的速度
    **    fuseData[1]: Y 轴的速度
    **    fuseData[2]: Z 轴的速度
    **    fuseData[3]: X 轴的位置
    **    fuseData[4]: Y 轴的位置
    **    fuseData[5]:  高度
    */
    bool fuseData[6] = {false,false,false,false,false,false};
    Vector6 R_OBS; //测量精度的方差

    Vector6 observation;//测量值，即 z_t
```

```
//进入对速度、位置和高度的融合
if (fuseVelData || fusePosData || fuseHgtData) {
    //gps 可用的情况下 constPosMode = false，constVelMode 始终等于 false,所以
    //statesAtPosTime 和 statesAtVelTime 在这里不会被更新
    if (constPosMode) {
        statesAtPosTime = state;
    } else if (constVelMode) {
        statesAtVelTime = state;
    }
    //计算 z_t
    if (!constPosMode && !constVelMode) {
        //从 gps 中获得测量值
        observation[0] = velNED.x + gpsVelGlitchOffset.x;
        observation[1] = velNED.y + gpsVelGlitchOffset.y;
        observation[2] = velNED.z;
        observation[3] = gpsPosNE.x + gpsPosGlitchOffsetNE.x;
        observation[4] = gpsPosNE.y + gpsPosGlitchOffsetNE.y;
    }
    observation[5] = -hgtMea; //从高度计获得测量值
    //计算 gps 速度 测量精度方差
    if (gpsSpdAccuracy > 0.0f) {
        R_OBS[0] = sq(constrain_float(gpsSpdAccuracy, _gpsHorizVelNoise, 50.0f));
        R_OBS[2] = sq(constrain_float(gpsSpdAccuracy, _gpsVertVelNoise, 50.0f));
    } else {
        R_OBS[0] = sq(constrain_float(_gpsHorizVelNoise, 0.05f, 5.0f)) +
                                            sq(gpsNEVelVarAccScale * accNavMag);
        R_OBS[2] = sq(constrain_float(_gpsVertVelNoise,   0.05f, 5.0f)) +
                                            sq(gpsDVelVarAccScale   * accNavMag);
    }
    R_OBS[1] = R_OBS[0];
    //gps 位置测量精度方差
    R_OBS[3] = sq(constrain_float(_gpsHorizPosNoise, 0.1f, 10.0f)) + sq(posErr);
    R_OBS[4] = R_OBS[3];
    //高度计测量精度方差
    R_OBS[5] = sq(constrain_float(_baroAltNoise, 0.1f, 10.0f));
    //检查 imu 的数据是否可用
    if (useGpsVertVel && fuseVelData && (imuSampleTime_ms -
                                        lastHgtMeasTime) <   (2 * msecHgtAvg)) {
        float hgtErr   = statesAtHgtTime.position.z - observation[5];
        float velDErr = statesAtVelTime.velocity.z - observation[2];
        if ((hgtErr*velDErr > 0.0f) &&
            (sq(hgtErr) > 9.0f * (P[9][9] + R_OBS_DATA_CHECKS[5])) &&
            (sq(velDErr) > 9.0f * (P[6][6] + R_OBS_DATA_CHECKS[2]))) {
```

```
                badIMUdata = true;
            } else {
                badIMUdata = false;
            }
        }
        //测试位置的测试值是否参与融合
        if (fusePosData) {
            //位置革新值= 上一次状态值- 测量值
            innovVelPos[3] = statesAtPosTime.position.x - observation[3];
            innovVelPos[4] = statesAtPosTime.position.y - observation[4];
            //计算方差的可用的革新值
            varInnovVelPos[3] = P[7][7] + R_OBS_DATA_CHECKS[3];
            varInnovVelPos[4] = P[8][8] + R_OBS_DATA_CHECKS[4];
            //计算位置革新值是否连续
            float accelScale =    (1.0f + 0.1f * accNavMag);
            float maxPosInnov2 = sq(_gpsPosInnovGate * _gpsHorizPosNoise) +
                    sq(0.005f * accelScale * float(_gpsGlitchAccelMax) * sq(0.001f *
                    float(imuSampleTime_ms - lastPosPassTime)));
            posTestRatio = (sq(innovVelPos[3]) + sq(innovVelPos[4])) / maxPosInnov2;
            posHealth = ((posTestRatio < 1.0f) || badIMUdata);
            //imu 取样时间和 gps 位置取样时间，是否超时
            posTimeout = (((imuSampleTime_ms - lastPosPassTime) > gpsRetryTime) ||
                                PV_AidingMode == AID_NONE);
            if (posHealth || posTimeout || constPosMode) {
                posHealth = true;//同意融合
                if (posTimeout || (maxPosInnov2 > sq(float(_gpsGlitchRadiusMax)))) {//超时
                    decayGpsOffset();
                    ResetPosition();//重置位置
                    ResetVelocity(); //重置速度
                    fusePosData = false;//不同意融合
                }
            } else {
                posHealth = false;//不同意融合
            }
        }
        //测试速度的测量值是否参与融合
        if (fuseVelData) {
            for (uint8_t i = 0; i<=imax; i++) {
                stateIndex     = i + 4;
                //速度的革新值= 上一次速度状态值-观察值
                velInnov[i]    = statesAtVelTime.velocity[i] - observation[i]; // blended
                velInnov1[i] = statesAtVelTime.vel1[i] - observation[i]; // IMU1
                velInnov2[i] = statesAtVelTime.vel2[i] - observation[i]; // IMU2
                //计算方差的可用革新值
```

```
            varInnovVelPos[i] = P[stateIndex][stateIndex] + R_OBS_DATA_CHECKS[i];
            //计算测试精度的方差
            float R_hgt;
            if (i == 2) {
               R_hgt = sq(constrain_float(_gpsVertVelNoise, 0.05f, 5.0f));
            } else {
               R_hgt = sq(constrain_float(_gpsHorizVelNoise, 0.05f, 5.0f));
            }
            //分别计算 imu 积分的速度的误差
            K1 += R_hgt / (R_hgt + sq(velInnov1[i]));
            K2 += R_hgt / (R_hgt + sq(velInnov2[i]));
            //计算革新速度的方差和，和革新方差的和
            innovVelSumSq += sq(velInnov[i]);
            varVelSum += varInnovVelPos[i];
        }
        //计算 IMU1  参与融合的权重
        if (vehicleArmed) {
            IMU1_weighting = 1.0f * (K1 / (K1 + K2)) + 0.0f * IMU1_weighting;
        } else {
            IMU1_weighting = 0.5f;
        }
        if (clipRateFilt1 > 0.1f || clipRateFilt2 > 0.1f) {
             if (clipRateFilt1 > clipRateFilt2) {
                 IMU1_weighting = 0.0f;
              } else {
                 IMU1_weighting = 1.0f;
              }
        }
        //计算速度是否连续
        velTestRatio = innovVelSumSq / (varVelSum * sq(_gpsVelInnovGate));
        velHealth = ((velTestRatio < 1.0f)    || badIMUdata);
        //imu 采集数据和 gps 速度采集是否超时
        velTimeout = (((imuSampleTime_ms - lastVelPassTime) > gpsRetryTime) ||
                                                PV_AidingMode == AID_NONE);
         if (velHealth || velTimeout || constVelMode) {
               velHealth = true;//同样融合
        } else if (velTimeout && !posHealth && PV_AidingMode == AID_ABSOLUTE)
        {
            ResetVelocity();//重置速度
            fuseVelData =    false;//不同意融合
        }else {
            velHealth = false;//不容易融合
        }
    }
```

```
//测试高度的测量值是否参与融合
if (fuseHgtData) {
    //计算高度的革新值
    innovVelPos[5] = statesAtHgtTime.position.z - observation[5];
    //计算革新值协方差的合理革新值(P[9][9]是高度方差)
    //R_OBS_DATA_CHECKS[5]是测量精度的方差
    varInnovVelPos[5] = P[9][9] + R_OBS_DATA_CHECKS[5];
    //如果革新值的的平方除以高度革新范围的平方大于方差，表示不连续
    hgtTestRatio = sq(innovVelPos[5]) / (sq(_hgtInnovGate) * varInnovVelPos[5]);
    hgtHealth = ((hgtTestRatio < 1.0f) || badIMUdata);
    //计算 imu 的采样与高度采样是否超时
    hgtTimeout = (imuSampleTime_ms - lastHgtPassTime) > hgtRetryTime;
    if (hgtHealth || hgtTimeout || constPosMode) {
          hgtHealth = true;//hgtHealth 值同样融合
        if (hgtTimeout) { //如果超时
            //从气压计中读出高度值，重置高度
            ResetHeight();
            useHgtData = false; //fuseHgtData 不同意融合
         }
    }else {
        hgtHealth = false;//hgtHealth 不同意融合
    }
}
//通过之前的测试，指定那些数据可以融合
if (fuseVelData && useGpsVertVel && velHealth &&
                !constPosMode && PV_AidingMode == AID_ABSOLUTE) {
     fuseData[0] = true;
     fuseData[1] = true;
     fuseData[2] = true;
 }
 if (fuseVelData && _fusionModeGPS == 1 && velHealth &&
                        !constPosMode && PV_AidingMode == AID_ABSOLUTE) {
       fuseData[0] = true;
       fuseData[1] = true;
 }
if ((fusePosData && posHealth && PV_AidingMode ==
                        AID_ABSOLUTE) || constPosMode) {
       fuseData[3] = true;
       fuseData[4] = true;
 }
 if ((fuseHgtData && hgtHealth) || constPosMode) {
       fuseData[5] = true;
 }
 if (constVelMode) {
```

```
        fuseData[0] = true;
        fuseData[1] = true;
    }
//开始融合需要融合的值
for (obsIndex=0; obsIndex<=5; obsIndex++) {
    if (fuseData[obsIndex]) {
        stateIndex = 4 + obsIndex;//算出需要更新 Kalman 增益的哪一位
        if (obsIndex <= 2)
        {
            //速度革新值，状态矩阵中的 4、5、6 位
            innovVelPos[obsIndex] = statesAtVelTime.velocity[obsIndex] -
                                                observation[obsIndex];
            R_OBS[obsIndex] *= sq(gpsNoiseScaler);
        }else if (obsIndex == 3 || obsIndex == 4) {
            //位置 革新值 ，状态矩阵中的 7、8 位
            innovVelPos[obsIndex] = statesAtPosTime.position[obsIndex-3] -
                                observation[obsIndex];
             R_OBS[obsIndex] *= sq(gpsNoiseScaler);
        } else {
            //高度革新值，状态矩阵中的 9 位
            innovVelPos[obsIndex] = statesAtHgtTime.position[obsIndex-3] -
                                        observation[obsIndex];
        }

        //计算上面（3.3）式的分母 K_t = P_{t-1} / (P_{t-1} + Q_t)

        varInnovVelPos[obsIndex] = P[stateIndex][stateIndex] + R_OBS[obsIndex];
                                SK = 1.0f/varInnovVelPos[obsIndex];
        //kalman 增益的 0--12 位
        for (uint8_t i= 0; i<=12; i++) {
            Kfusion[i] = P[i][stateIndex]*SK;
        }
        //计算加速度计的 kalman 增益，P[13][stateIndex]是加速度和位置的协方差
        if ((obsIndex == 5 || obsIndex == 2) &&
                    prevTnb.c.z > 0.5f && !getTakeoffExpected()) {
            Kfusion[13] = constrain_float(P[13][stateIndex]*SK,-1.0f,0.0f);
         } else {
            Kfusion[13] = 0.0f;
         }
        if (!inhibitWindStates) {
            Kfusion[14] = P[14][stateIndex]*SK;
            Kfusion[15] = P[15][stateIndex]*SK;
          } else {
             Kfusion[14] = 0.0f;
```

```
        Kfusion[15] = 0.0f;
    }
if (!inhibitMagStates) {
        for (uint8_t i = 16; i<=21; i++) {
            Kfusion[i] = P[i][stateIndex]*SK;
        }
 } else {
        for (uint8_t i = 16; i<=21; i++) {
            Kfusion[i] = 0.0f;
    }
}
Kfusion[26] = Kfusion[9];    // IMU1 posD
Kfusion[30] = Kfusion[9];    // IMU2 posD
for (uint8_t i = 0; i<=2; i++) {
        Kfusion[i+23] = Kfusion[i+4]; // IMU1 velNED
        Kfusion[i+27] = Kfusion[i+4]; // IMU2 velNED
 }
......
//更新状态方程
if (obsIndex == 5){//把由 imu 加速度计统计的高度融合到 imu 相关的项中
      //以 imu 统计的高度计算革新值
      float hgtInnov1 = statesAtHgtTime.posD1 - observation[obsIndex];
      float hgtInnov2 = statesAtHgtTime.posD2 - observation[obsIndex];、
      //融合了与 imu 相关的项（由于篇幅原因，不写了）
}
else if (obsIndex == 0 || obsIndex == 1 || obsIndex == 2) {
      //把加速度计统计的速度融合到 imu 加速度计积分的速度和高度中
      for (uint8_t i = 23; i<=26; i++) {
          states[i] = states[i] - Kfusion[i] * velInnov1[obsIndex];
      }
      for (uint8_t i = 27; i<=30; i++) {
          states[i] = states[i] - Kfusion[i] * velInnov2[obsIndex];
      }
}

//对普通项进行正常的融合
for (uint8_t i = 0; i<=21; i++) {
    if (i != 13) {
        if ((i <= 3 && highRates) || i >= 10 || constPosMode || constVelMode) {
            //融合与速度和位置无关的项
            states[i] = states[i] - Kfusion[i] * innovVelPos[obsIndex];
        } else {
        if (obsIndex == 5) {
            //计算高度修正值
```

```
                        hgtIncrStateDelta[i] -= Kfusion[i] * innovVelPos[obsIndex] *
                                                                hgtUpdateCountMaxInv;
                      } else {
                        //计算速度修正值
                        gpsIncrStateDelta[i] -= Kfusion[i] * innovVelPos[obsIndex] *
                                                    gpsUpdateCountMaxInv;
                    }
                   }
                  }
               }


            }
          }
      }
}
```

第五章 mavlink 与地面站的通信

第一节 mavlink 协议

（声明：原创吕元宙）

内容一栏表

Mavlink 包描述

基础枚举类型

Heartbeat,心跳包

Sys_status,系统状态

System_time

Set_mode,设置飞行模式

Param_request_read,获取参数信息

param_request_list,获取所有的参数信息

Param_value,参数信息

Param_set,设置参数值

Gps_raw_int,gps 原始数据包

Raw_imu,1 号 imu 原始数据

Scaled_pressure,气压计原始数据

Attitude,姿态信息

Local_position_ned，ned 框架的相对位置

Global_position_int，绝对位置

Rc_channels_sscaled，rc 输入通道转换信息

Rc_channels_raw，rc 输入通道原始信息

Servo_output_raw，rc 输入通道信息

Mission_write_partial_list 写入任务队列

MISSION_ITEM,写入任务项
MISSION_REQUEST,查询指定任务项
MISSION_SET_CURRENT,设置当前运行的任务项
MISSION_CURRENT,当前运行任务号
MISSION_REQUEST_LIST,查询任务总数
MISSION_COUNT,任务总数
MISSION_CLEAR_ALL,清除所有任务
MISSION_ACK,任务应答包
NAV_CONTROLLER_OUTPUT,发送导航信息
REQUEST_DATA_STREAM,设置数据流
RC_CHANNELS_OVERRIDE,设置 RC 输入值
VFR_HUD,HUD 信息
COMMAND_LONG,执行指定命令
COMMAND_ACK,命令应答包
SET_POSITION_TARGET_LOCAL_NED,设置飞行的相对目标
SET_POSITION_TARGET_GLOBAL_INT,设置飞行的绝对目标
OPTICAL_FLOW,光流传感器数据
RADIO,数传状态
SCALED_IMU2,2 号 IMU 原始数据
LOG_REQUEST_LIST,查询 log 列表
LOG_ENTRY,log 列表信息
LOG_REQUEST_DATA,申请发送
LOG_DATA,发送 log 数据
LOG_ERASE,擦除所有 log 信息
LOG_REQUEST_END,停止发送 log
POWER_STATUS,电源状态
SCALED_IMU3,3 号 IMU 原始数据
SENSOR_OFFSETS,传感器校准数据
MEMINFO,内存信息
FENCE_STATUS,未实现
AHRS,姿态解算信息
HWSTATUS,硬件状态信息
RANGEFINDER,测距器信息
EKF_STATUS_REPORT,EKF 状态信息
PID_TUNING,内环 PID 状态
VIBRATION,机体震动状态
STATUSTEXT,发送文字信息
附录 1:MAV_CMD

MAVLINK 包描述

MAVLINK 包分为 9 个部分,每个部分以 1 个字节为单位,如下

	位置	长度	取值范围	描述
STX	0	1	254	指示包头,固定为 254
LEN	1	1	0-255	包的长度

SEQ	2	1	0-255	包的顺序号
SYS	3	1	7	SYS ID 号(固定为 7)
COMP	4	1	1	COMP ID 号(固定为 1)
MSG	5	1	0-255	MSG ID 号
PAYLOAD	6	n	0-255	数据,由 MSG ID 号进行指定
CKA	n+7	1	0-255	校验和低字节
CKB	n+8	1	0-255	校验和高字节

基础枚举类型

MAV_POWER_STATUS		
BRICK_VALID	1	电池电源是否有效
SERVO_VALID	2	伺服电机电源是否有效
USB_CONNECTED	4	USB 电源是否有效
PERIPH_OVERCURRENT	8	总线 BUS 是否过流
PERIPH_HIPOWER_OVERCURRENT	16	SERIAL4/5 接口是否过流
CHANGED	32	

MAV_MODE_FLAG		
CUSTOM_MODE_ENABLED	1	启用定制模式
TEST_ENABLED	2	启用测试模式
AUTO_ENABLED	4	启用自动模式
GUIDED_ENABLED	8	启用跟随模式
STABILIZE_ENABLED	16	启用自稳模式
HIL_ENABLED	32	启用 HIL 模式
MANUAL_INPUT_ENABLED	64	启用手动输入模式
SAFETY_ARMED	128	启用安全模式

MAV_STATE		
UNINIT	0	未初始化
BOOT	1	启动
CALIBRATING	2	校准
STANDBY	3	待机
ACTIVE	4	激活
CRITICAL	5	紧急(仍可操控)
EMERGENCY	6	危机(不可操控)
POWEROFF	7	关机

MAV_SYS_STATUS

SENSOR_3D_GYRO	0x01	陀螺仪
SENSOR_3D_ACCEL	0x02	加速计
SENSOR_3D_MAG	0x04	罗盘
SENSOR_ABSOLUTE_PRESSURE	0x08	绝对气压计
SENSOR_DIFFERENTIAL_PRESSURE	0x10	相对气压计
SENSOR_GPS	0x20	GPS
SENSOR_OPTICAL_FLOW	0x40	光流计
SENSOR_VISION_POSITION	0x80	视觉位置
SENSOR_LASER_POSITION	0x100	激光位置
SENSOR_EXTERNAL_GROUND_TRUTH	0x200	外部观察
SENSOR_ANGULAR_RATE_CONTROL	0x400	角速度控制
SENSOR_ATTITUDE_STABILIZATION	0x800	姿态稳定
SENSOR_YAW_POSITION	0x1000	偏航位置
SENSOR_Z_ALTITUDE_CONTROL	0x2000	高度控制
SENSOR_XY_POSITION_CONTROL	0x4000	XY 平面控制
SENSOR_MOTOR_OUTPUTS	0x8000	马达输出
SENSOR_RC_RECEIVER	0x10000	RC 输入
SENSOR_3D_GYRO2	0x20000	2 号陀螺仪
SENSOR_3D_ACCEL2	0x40000	2 号加速计
SENSOR_3D_MAG2	0x80000	2 号罗盘
GEOFENCE	0x100000	地理围栏
AHRS	0x200000	姿态解算
TERRAIN	0x400000	

MAV_AUTOPILOT		
ARDUPILOTMEGA	3	Ardupilot 驾驶仪

MAV_TYPE		
QUADROTOR	2	四轴
HEXAROTOR	13	六轴
OCTOROTOR	14	八轴

EKF_STATUS_FLAGS		
ATTITUDE	1	姿态正常
VELOCITY_HORIZ	2	速度(水平方向) 正常
VELOCITY_VERT	4	速度(垂直方向) 正常
POS_HORIZ_REL	8	水平位置(相对) 正常
POS_HORIZ_ABS	16	水平位置(绝对) 正常

POS_VERT_ABS	32	垂直位置(绝对) 正常
POS_VERT_AGL	64	垂直位置(相对大地) 正常
CONST_POS_MODE	128	固定位置正常
PRED_POS_HORIZ_REL	256	预测水平(相对)位置正常
PRED_POS_HORIZ_ABS	512	预测水平(绝对)位置正常

autopilot_modes		
STABILIZE	0	自稳模式
ALT_HOLD	2	定高模式
AUTO	3	航点模式
GUIDED	4	跟随模式
RTL	6	返航模式
CIRCLE	7	环绕模式
LAND	9	降落模式
POSHOLD	16	定点模式

MAV_PARAM_TYPE		
CHAR	0	字符
UINT8_T	1	无符号 8 位
INT8_T	2	有符号 8 位
UINT16_T	3	无符号 16 位
INT16_T	4	有符号 16 位
UINT32_T	5	无符号 32 位
INT32_T	6	有符号 32 位
UINT64_T	7	无符号 64 位
INT64_T	8	有符号 64 位
FLOAT	9	float
DOUBLE	10	double

MAV_CMD		
NAV_WAYPOINT	16	普通航点
NAV_LOITER_UNLIM	17	悬停模式
NAV_LOITER_TURNS	18	环绕模式
NAV_LOITER_TIME	19	定时悬停
NAV_RETURN_TO_LAUNCH	20	RTL 模式
NAV_LAND	21	降落模式
NAV_TAKEOFF	22	起飞模式
NAV_SPLINE_WAYPOINT	82	曲线航点

NAV_GUIDED_ENABLE	92	引导模式
CONDITION_DELAY	112	延时
CONDITION_YAW	115	设定航向
DO_CHANGE_SPEED	178	更改航点速度
DO_SET_HOME	179	设置 HOME 点
DO_SET_RELAY	181	设置继电器
DO_REPEAT_RELAY	182	设置继电器
DO_SET_SERVO	183	设置伺服
DO_REPEAT_SERVO	184	设置伺服
DO_SET_ROI	201	设置兴趣区域
DO_DIGICAM_CONFIGURE	202	相机配置
DO_DIGICAM_CONTROL	203	相机操作
DO_SET_CAM_TRIGG_DIST	206	相机触发
DO_FENCE_ENABLE	207	地理围栏
DO_PARACHUTE	208	降落伞
DO_MOTOR_TEST	209	电机测试
DO_GRIPPER	211	夹子
DO_GUIDED_LIMITS	222	引导限制
PREFLIGHT_CALIBRATION	241	校准传感器
PREFLIGHT_SET_SENSOR_OFFSETS	242	设置罗盘偏移
PREFLIGHT_REBOOT_SHUTDOWN	246	重启飞控
MISSION_START	300	开始任务
COMPONENT_ARM_DISARM	400	解锁加锁
REQUEST_AUTOPILOT_CAPABILITIES	520	发送版本号

MAV_DATA_STREAM		
ALL	0	所有数据流
RAW_SENSORS	1	IMU 和 GPS 数据流
EXTENDED_STATUS	2	GPS,控制和 AUX 数据流
DC_CHANNELS	3	RC 和 SERVO 数据流
RAW_CONTROLLER	4	姿态,位置和导航数据流
POSITION	6	局部和全局位置数据流
EXTRA1	10	保留
EXTRA2	11	保留
EXTRA3	12	保留

MAV_RESULT		
ACCEPTED	0	执行成功

TEMPORARILY_REJECTED	1	暂时拒绝
DENIED	2	永久拒绝
UNSUPPORTED	3	不支持
FAILED	4	失败

MAV_FRAME		
GLOBAL	0	WGS84(x:lat y:lon z:基于海平面的正值)
LOCAL_NED	1	x:north y:east z:down
MISSION	2	
GLOBAL_RELATIVE_ALT	3	WGS84(x:lat y:lon z:基于 Home 的正值)
LOCAL_ENU	4	x:east y:north z:up
GLOBAL_INT	5	WGS84(x:lat *1.0e-7 y:lon *1.0e-7 z:基于海平面的正值)
GLOBAL_RELATIVE_ALT_INT	6	WGS84(x:lat *1.0e-7 y:lon *1.0e-7 z:基于 Home 的正值)
LOCAL_OFFSET_NED	7	基于当前局部坐标框架的差值
BODY_NED	8	基于机体 NED 框架(前后左右)
BODY_OFFSET_NED	9	基于机体 NED 框架(东南西北)
GLOBAL_TERRAIN_ALT	10	
GLOBAL_TERRAIN_ALT_INT	11	

MAV_MISSION_RESULT		
MAV_MISSION_ACCEPTED	0	处理完成
MAV_MISSION_ERROR	1	处理失败
MAV_MISSION_UNSUPPORTED_FRAME	2	不支持的坐标框架
MAV_MISSION_UNSUPPORTED	3	不支持的命令
MAV_MISSION_NO_SPACE	4	没有空间存储任务
MAV_MISSION_INVALID	5	其中一个参数无效
MAV_MISSION_INVALID_PARAM1	6	参数 1 无效
MAV_MISSION_INVALID_PARAM2	7	参数 2 无效
MAV_MISSION_INVALID_PARAM3	8	参数 3 无效
MAV_MISSION_INVALID_PARAM4	9	参数 4 无效
MAV_MISSION_INVALID_PARAM5_X	10	参数 5 无效
MAV_MISSION_INVALID_PARAM6_Y	11	参数 6 无效
MAV_MISSION_INVALID_PARAM7	12	参数 7 无效
MAV_MISSION_INVALID_SEQUENCE	13	航点序列号错误

MAV_MISSION_DENIED	14	拒绝处理

HEARTBEAT,心跳包

MSG 编号	名字	输入输出
0	HEARTBEAT	IO
描述	确认和地面站之间的通信是否持续 可以设置超时失控保护	
成员	类型	描述
custom_mode	uint32_t	定制模式(autopilot_modes)
type	uint8_t	机架类型(MAV_TYPE)
autopilot	uint8_t	自动驾驶仪型号(MAV_AUTOPILOT)
base_mode	uint8_t	基础模式(MAV_MODE_FLAG)
system_status	uint8_t	系统状态(MAV_STATE)
mavlink_version	uint8_t	MAVLINK 版本(固定为 3)

ardupilot 的发送处理流程为:

1. 设置 custom_mode 为飞行模式 autopilot_modes 中的一项
2. 设置 type 为机架类型 MAV_TYPE 中的一项
3. 设置 autopilot 为自动驾驶仪类型 MAV_AUTOPILOT 中的一项
4. 设置 base_mode 为基础模式 MAV_MODE_FLAG 中项的混合,必选
   MAV_MODE_FLAG_STABILIZE_ENABLED
   MAV_MODE_FLAG_MANUAL_INPUT_ENABLED
   MAV_MODE_FLAG_CUSTOM_MODE_ENABLED
   当处于 ATUO,RTL,GUIDED,CIRCLE,POSHOLD 模式时需选中 MAV_MODE_FLAG_GUIDED_ENABLED 模式
   当处于电机解锁状态时需选中 MAV_MODE_FLAG_SAFETY_ARMED 模式
5. 设置 system_status 为系统状态 MAV_STATE 中的一项

ardupilot 的接收处理流程为:

1. 将 failsafe 类中的 last_heartbeat_ms 设置为当前时间
2. pmTest1 心跳包计数器自加

SYS_STATUS,系统状态

MSG 编号	名字	输入输出
1	SYS_STATUS	0
描述	发送传感器,电池和 CPU 信息	
成员	类型	描述
onboard_control_sensors_present	uint32_t	传感器是否在位(MAV_SYS_STATUS_SENSOR)
onboard_control_sensors_enabled	uint32_t	传感器是否开启(MAV_SYS_STATUS_SENSOR)
onboard_control_sensors_health	uint32_t	传感器是否健康(MAV_SYS_STATUS_SENSOR)
load	uint16_t	主循环使用率

voltage_battery	uint16_t	电池电压,单位为 mv
current_battery	int16_t	电池电流,单位为 ma
drop_rate_comm	uint16_t	保留为 0
errors_comm	uint16_t	保留为 0
errors_count1	uint16_t	保留为 0
errors_count2	uint16_t	保留为 0
errors_count3	uint16_t	保留为 0
errors_count4	uint16_t	保留为 0
battery_remaining	int8_t	剩余电量,单位为%

ardupilot 的发送处理流程为:

1. 设置 onboard_control_sensors_present 为 MAV_SYS_STATUS_SENSOR 中项的混合,默认选择

SENSOR_3D_GYRO	SENSOR_3D_ACCEL	SENSOR_ABSOLUTE_PRESSURE
SENSOR_ANGULAR_RATE_CONTROL	SENSOR_ATTITUDE_STABILIZATION	SENSOR_YAW_POSITION
SENSOR_Z_ALTITUDE_CONTROL	SENSOR_XY_POSITION_CONTROL	SENSOR_MOTOR_OUTPUTS
AHRS		

   - 当罗盘可用时选中 SENSOR_3D_MAG
   - 当 GPS 可用时选中 SENSOR_GPS
   - 当光流计可用时选中 SENSOR_OPTICAL_FLOW
   - 当 RC 接收可用时选中 SENSOR_RC_RECEIVER
   - 当超声波可用时选中 SENSOR_LASER_POSITION
   - 当 AHRS 未初始化完成和未校准完成时去掉 SENSOR_3D_GYRO 和 SENSOR_3D_ACCEL
2. 设置 onboard_control_sensors_enabled 和 onboard_control_sensors_present 相等
   - 当不为 ALT_HOLD,AUTO,GUIDED,RTL,CIRCLE,LAND,POSHLOD 等飞行模式时去掉 SENSOR_Z_ALTITUDE_CONTROL 和 SENSOR_XY_POSITION_CONTROL
   - 当安全开关未按下时,去掉 SENSOR_MOTOR_OUTPUTS
   - 当 AHRS 未初始化完成和未校准完成时去掉 SENSOR_3D_GYRO 和 SENSOR_3D_ACCEL
3. 设置 onboard_control_sensors_health 和 onboard_control_sensors_present 相等
   - 当气压计不健康时去掉 SENSOR_ABSOLUTE_PRESURE
   - 当罗盘不健康时去掉 SENSOR_3D_MAG
   - 当 GPS 未定位时去掉 SENSOR_GPS
   - 当光流计不健康时去掉 SENSOR_OPTICAL_FLOW
   - 当 RC 接收不健康时去掉 SENSOR_RC_RECEIVER
   - 当陀螺仪不健康时去掉 SENSOR_3D_GYRO
   - 当加速计不健康时去掉 SENSOR_3D_ACCEL
   - 当 AHRS 不健康时去掉 AHRS

SYSTEM_TIME

MSG 编号	名字	输入输出

2	SYSTEM_TIME	0
描述	反馈系统时间	
成员	**类型**	**描述**
time_unix_usec	uint64_t	GPS 时间,用毫秒表示
time_boot_ms	uint32_t	系统开机时间,用毫秒表示

SET_MODE,设置飞行模式

MSG 编号	**名字**	**输入输出**
11	SET_MODE	I
描述	设置飞行模式为指定模式	
成员	**类型**	**描述**
custom_mode	uint32_t	目标定制模式(autopilot_modes)
target_system	uint8_t	保留
base_mode	uint8_t	目标基础模式(MAV_MODE_FLAG)

ardupilot 的接收处理流程为:

1. 检查 base_mode 是否配置了 MAV_MODE_FLAG 中的 CUSTOM_MODE_ENABLED 项
   - 使用 Copter::set_mode 根据 custom_mode 配置目标定制模式
2. 如果没有再检查是否配置了 MAV_MODE_FLAG 中的 DECODE_POSITION_SAFETY 项
   - 如果 custom_mode 为 0 则关闭安全开关检查
   - 如果 custom_mode 为 0 则开启安全开关检查
3. 根据执行结果返回 ACK 包

PARAM_REQUEST_READ,获取参数信息

MSG 编号	**名字**	**输入输出**
20	PARAM_REQUEST_READ	I
描述	根据指定参数获取参数信息	
成员	**类型**	**描述**
param_index	int16_t	参数序列号
target_system	uint8_t	保留
target_component	uint8_t	保留
param_id	char[16]	参数 ID

ardupilot 的接收处理流程为:

1. 根据 param_index 查找参数
   - param_index 不为-1 按序列号进行查找
   - 否则按 param_id 进行字符串查找
2. 根据查找结果将参数封装成 PARAM_VALUE 包并发送返回

PARAM_REQUEST_LIST,获取所有参数信息

MSG 编号	名字	输入输出
21	PARAM_REQUEST_LIST	I
描述	收到该包后将所有参数顺序发出	
成员	类型	描述
target_system	uint8_t	保留
target_component	uint8_t	保留

ardupilot 的接收处理流程为:

1. 发送固件版本号
2. 发送 PX4 版本号和 Nuttx 版本号
3. 发送飞行器框架类型
4. 发送 sysid 号
5. 配置待发参数队列,配置后按照一定的频率发送参数队列
   - 设队列指向第一个参数
   - 设队列序列号为 0
   - 设队列计数器为参数总数

PARAM_VALUE,参数信息

MSG 编号	名字	输入输出
22	PARAM_VALUE	0
描述	发送参数信息	
成员	类型	描述
param_value	float	参数值
param_count	uint16_t	参数总数
param_index	uint16_t	参数序列号
param_id	char[16]	参数名字
param_type	uint8_t	参数类型

PARAM_SET,设置参数值

MSG 编号	名字	输入输出
23	PARAM_SET	I
描述	根据参数信息设置新的参数值	
成员	类型	描述
param_value	float	目标参数值
target_system	uint8_t	保留
target_component	uint8_t	保留
param_id	char[16]	目标参数名
param_type	uint8_t	目标参数类型

ardupilot 的接收处理流程为:

1. 使用 param_id 历遍所有参数名进行匹配
2. 有匹配项则设置其值为 param_value,并发送 PARAM_VALUE 包返回修改后的参数

GPS_RAW_INT,GPS 原始数据

MSG 编号	名字	输入输出
24	GPS_RAW_INT	0
描述	发送 GPS 原始信息	
成员	**类型**	**描述**
time_usec	uint64_t	系统时间
lat	int32_t	纬度
lon	int32_t	经度
alt	int32_t	高度
eph	uint16_t	HDOP
epv	uint16_t	VDOP,保留为 65536
vel	uint16_t	地速
cog	uint16_t	航线?
fix_type	uint8_t	GPS 锁定类型
satellites_visible	uint8_t	搜星数

ardupilot 的发送处理流程为:

1. 当 GPS 处于锁定状态时,如果无新数据则不发送数据包
2. 当 GPS 处于非锁定状态时,则按 1HZ 的频率发送数据包

RAW_IMU,1 号 imu 原始数据

MSG 编号	名字	输入输出
27	RAW_IMU	0
描述	发送 1 号 IMU 的信息	
成员	**类型**	**描述**
time_usec	uint64_t	系统时间
xacc	int16_t	加速度 X
yacc	int16_t	加速度 Y
zacc	int16_t	加速度 Z
xgyro	int16_t	角速度 X
ygyro	int16_t	角速度 Y
zgyro	int16_t	角速度 Z
xmag	int16_t	罗盘 X
ymag	int16_t	罗盘 Y
zmag	int16_t	罗盘 Z

SCALED_PRESSURE,气压计原始数据

MSG 编号	名字	输入输出
29	SCALED_PRESSURE	0
描述	发送气压计原始数据	
成员	**类型**	**描述**
time_boot_ms	uint32_t	系统时间
press_abs	float	绝对气压值
press_diff	float	相对气压值
temperature	int16_t	温度

ATTITUDE,姿态信息

MSG 编号	名字	输入输出
30	ATTITUDE	0
描述	发送姿态信息	
成员	**类型**	**描述**
time_boot_ms	uint32_t	系统时间
roll	float	横滚角
pitch	float	俯仰角
yaw	float	偏航角
rollspeed	float	横滚角速度
pitchspeed	float	俯仰角速度
yawspeed	float	偏航角速度

LOCAL_POSITION_NED,NED 框架的相对位置

MSG 编号	名字	输入输出
32	LOCAL_POSITION_NED	0
描述	发送 NED 框架的局部位置信息	
成员	**类型**	**描述**
time_boot_ms	uint32_t	系统时间
x	float	X 位置
y	float	Y 位置
z	float	Z 位置
vx	float	X 速度
vy	float	Y 速度
vz	float	Z 速度

ardupilot 的发送处理流程为:

1. 检查是否能获得 NED 的相对位置和速度,如果不能则不发送数据包

GLOBAL_POSITION_INT,绝对位置

MSG 编号	名字	输入输出
33	GLOBAL_POSITION_INT	0
描述	发送绝对位置信息	
成员	类型	描述
time_boot_ms	uint32_t	系统时间
lat	int32_t	纬度
lon	int32_t	经度
alt	int32_t	高度
relative_alt	int32_t	相对地面高度
vx	int16_t	经度速度
vy	int16_t	纬度速度
vz	int16_t	高度速度
hdg	uint16_t	航向角

ardupilot 的发送处理流程为:

1. 当 GPS 处于锁定状态时, time_boot_ms 为最后一次获取的 GPS 时间
2. 当 GPS 不为锁定状态时, time_boot_ms 为系统时间

RC_CHANNELS_SCALED,rc 输入通道转换信息

MSG 编号	名字	输入输出
34	RC_CHANNELS_SCALED	0
描述	发送 RC 输入通道转后的信息	
成员	类型	描述
time_boot_ms	uint32_t	系统时间
chan1_scaled	int16_t	通道 1
chan2_scaled	int16_t	通道 2
chan3_scaled	int16_t	通道 3
chan4_scaled	int16_t	通道 4
chan5_scaled	int16_t	通道 5
chan6_scaled	int16_t	通道 6
chan7_scaled	int16_t	通道 7
chan8_scaled	int16_t	通道 8
port	uint8_t	端口,保留为 0
rssi	uint8_t	接收信号强度

ardupilot 的发送处理流程为:

1. chan1_scaled 为 RC1 的 servo_out 输出
2. chan2_scaled 为 RC2 的 servo_out 输出

3. chan3_scaled 为 RC3 的 radio_out 输出
4. chan4_scaled 为 RC4 的 servo_out 输出
5. chan5_scaled 为 RC1 的 norm_output 输出
6. chan6_scaled 为 RC2 的 norm_output 输出
7. chan7_scaled 为 RC3 的 norm_output 输出
8. chan8_scaled 为 RC4 的 norm_output 输出

RC_CHANNELS_RAW,RC 输入通道原始信息

MSG 编号	名字	输入输出
35	RC_CHANNELS_RAW	0
描述	RC 输入通道的原始信息	
成员	类型	描述
time_boot_ms	uint32_t	系统时间
chan1_raw	uint16_t	通道 1
chan2_raw	uint16_t	通道 2
chan3_raw	uint16_t	通道 3
chan4_raw	uint16_t	通道 4
chan5_raw	uint16_t	通道 5
chan6_raw	uint16_t	通道 6
chan7_raw	uint16_t	通道 7
chan8_raw	uint16_t	通道 8
port	uint8_t	端口,保留为 0
rssi	uint8_t	接收信号强度

ardupilot 的发送处理流程为:

1. 从 hal.rcin 中读取各个通道的输入并保存到 chanx_raw 通道中

SERVO_OUTPUT_RAW,rc 输入通道信息

MSG 编号	名字	输入输出
36	SERVO_OUTPUT_RAW	0
描述	发送 RC 输出通道的信息(电机驱动通道)	
成员	类型	描述
time_boot_ms	uint32_t	系统时间
servo1_raw	uint16_t	通道 1
servo2_raw	uint16_t	通道 2
servo3_raw	uint16_t	通道 3
servo4_raw	uint16_t	通道 4
servo5_raw	uint16_t	通道 5
servo6_raw	uint16_t	通道 6

servo7_raw	uint16_t	通道 7
servo8_raw	uint16_t	通道 8
port	uint8_t	端口

ardupilot 的发送处理流程为:

1. 从 hal. rcout 中读取各个通道的输入并保存到 chanx_raw 通道中

MISSION_WRITE_PARTIAL_LIST,写入任务队列

MSG 编号	**名字**	**输入输出**
38	MISSION_WRITE_PARTIAL_LIST	I
描述	写入任务队列,接收到该包后使用 MISSION_ITEM 包对指定的序列进行写入	
成员	**类型**	**描述**
start_index	int16_t	开始序列号
end_index	int16_t	结束序列号
target_system	uint8_t	保留
target_component	uint8_t	保留

ardupilot 的接收处理流程为:

1. 检查 start_index 是否大于最大命令数
2. 检查 end_index 是否大于最大命令数
3. 检查 start_index 是否大于 end_index
4. 设置 waypoint_timelast_receive 为当前时间
5. 设置 waypoint_timelast_request 为 0
6. 设置 waypoint_receiving 为 true
7. 设置 waypoint_request_i 为 start_index
8. 设置 waypoint_request_last 为 end_index

MISSION_ITEM,写入任务项

MSG 编号	**名字**	**输入输出**
39	MISSION_ITEM	I
描述	写入任务项,包含具体的任务信息	
成员	**类型**	**描述**
param1	float	参数 1
param2	float	参数 2
param3	float	参数 3
param4	float	参数 4
x	float	参数 5
y	float	参数 6
z	float	参数 7

seq	uint16_t	序列号
command	uint16_t	命令
target_system	uint8_t	保留
target_component	uint8_t	保留
frame	uint8_t	地理框架
current	uint8_t	
autocontinue	uint8_t	

ardupilot 的接收处理流程为:

1. 首先根据 seq 和 command 转换为 cmd 命令,并将参数赋给 cmd
2. 如果 current 为 2 时,使用 handle_guided_request 处理 cmd,并返回 ACK 包(ACCEPTED)
3. 如果 current 为 3 时,使用 handle_change_alt_request 处理 cmd,并返回 ACK 包(ACCEPTED)
4. 如果 waypoint_receiving 为 false,则返回 ACK 包(ERROR)
5. 如果 seq 不等于 waypoint_request_i,则返回 ACK 包(INVALID_SEQUENCE)
6. 如果 seq 小于最大命令数,则进行替换检查,成功则返回 ACK 包(ACCEPTED),失败则返回 ACK 包(ERROR)
7. 如果 seq 等于最大命令数,则进行插入检查, 成功则返回 ACK 包(ACCEPTED),失败则返回 ACK 包(ERROR)
8. 更新 waypoint_timelast_receive 为当前时间
9. 更新 waypoint_request_i 累加 1
10. 如果 waypoint_request_i 大于等于 waypoint_request_last
    - 发送 ACK 包(ACCEPTED)
    - 发送文字(flight plan received)
    - 设置 waypoint_receiving 为 false
11. 如果 waypoint_request_i 小于 waypoint_request_last,发送 MISSION_REQUEST 包

MISSION_REQUEST,查询指定任务项

MSG 编号	**名字**	**输入输出**
40	MISSION_REQUEST	IO
描述	查询指定任务项并使用 MISSION_ITEM 包返回	
成员	**类型**	**描述**
seq	uint16_t	序列号
target_system	uint8_t	保留
target_component	uint8_t	保留

ardupilot 的接收处理流程为:

1. 根据 seq 从 eeprom 中读取 cmd,失败则返回一个 ACK 包(ERROR)
2. 将 cmd 转换为一个 MISSION_ITEM 包,失败则返回一个 ACK 包(ERROR)
3. 如果 cmd 为当前执行的命令,则将 MISSION_ITEM 包中的 current 设为 1,否则为 0
4. 设置 MISSION_ITEM 包中的 autocontinue 为 1
5. 设置 MISSION_ITEM 包中的 seq 为当前的 seq

6. 设置 MISSION_ITEM 包中的 command 为 cmd 包的 id
7. 将 MISSION_ITEM 包发送返回

MISSION_SET_CURRENT,设置当前运动的任务项

MSG 编号	名字	输入输出
41	MISSION_SET_CURRENT	I
描述	设置当前运行的任务为指定任务项,并使用 MISSION_CURRENT 包返回	
成员	类型	描述
seq	uint16_t	序列号
target_system	uint8_t	保留
target_component	uint8_t	保留

ardupilot 的接收处理流程为:

1. 停止当前执行的命令
2. 如果 seq 为 0,则设 seq 为 1
3. 如果当前状态不为 MISSION_RUNNING
   - 根据 nav_cmd_loaded 的状态查找所有的 cmd
   - 从 eeprom 中获取 index 指定的 cmd
   - 如果该 cmd 为导航命令,则将 nav_cmd_loaded 设为 true
   - 如果该 cmd 为执行命令,则将 index 设为 cmd.index 加 1,继续进行查找
   - 如果查找失败,则设置_nav_cmd.index 为 AP_MISSION_CMD_INDEX_NONE
   - 返回查找结果
4. 根据 nav_cmd_loaded 的状态查找所有的 cmd
   - 从 eeprom 中获取 index 指定的 cmd
   - 如果该 cmd 为导航命令,则设 nav_cmd_loaded 为 true,并执行该命令
   - 如果该 cmd 为执行命令,则检查当前 do_cmd_loaded 状态,为 false 则执行该命令
5. 将当前运行的 nav_cmd. index 通过 MISSION_CURRENT 包发送返回

MISSION_CURRENT,当前运动任务号

MSG 编号	名字	输入输出
42	MISSION_CURRENT	O
描述	发送当前正在运行的任务号	
成员	类型	描述
seq	uint16_t	命令序列号

MISSION_REQUEST_LIST,查询任务总数

MSG 编号	名字	输入输出
43	MISSION_REQUEST_LIST	I
描述	接收到这个包后使用 MISSION_COUNT 包返回任务总数	
成员	类型	描述

target_system	uint8_t	保留
target_component	uint8_t	保留

ardupilot 的接收处理流程为:

1. 将命令总数通过 MISSION_COUNT 包返回

MISSION_COUNT,任务总数

MSG 编号	**名字**	**输入输出**
44	MISSION_COUNT	IO
描述	发送任务总数	
成员	**类型**	**描述**
count	uint16_t	任务数量
target_system	uint8_t	保留
target_component	uint8_t	保留

ardupilot 的接收处理流程为:

1. 比较任务数量是否大于 eeprom 的最大存储任务数,如果大于则返回一个 ACK 包(NO_SPACE)
2. 如果任务总数大于包的任务数量,则设任务总数为包的任务数量
3. 设置 waypoint_timelast_receive 为当前时间
4. 设置 waypoint_receiving 为 true
5. 设置 waypoint_request_i 为 0
6. 设置 waypoint_request_last 为包的任务数量
7. 设置 waypoint_timelast_request 为 0

MISSION_CLEAR_ALL,清除所有任务

MSG 编号	**名字**	**输入输出**
45	MISSION_CLEAR_ALL	I
描述	清除所有任务	
成员	**类型**	**描述**
target_system	uint8_t	保留
target_component	uint8_t	保留

ardupilot 的接收处理流程为:

1. 清除所有的航点,根据执行结果返回一个 ACK 包(ACCEPTED, TEMPORARILY_REJECTED)

MISSION_ACK,任务应答包

MSG 编号	**名字**	**输入输出**
47	MISSION_ACK	O
描述	反馈任务包执行结果	
成员	**类型**	**描述**

target_system	uint8_t	保留
target_component	uint8_t	保留
type	uint8_t	任务包处理结果(MAV_RESULT)

NAV_CONTROLLER_OUTPUT,发送导航信息

MSG 编号	名字	输入输出
62	NAV_CONTROLLER_OUTPUT	O
描述	发送导航信息	
成员	类型	描述
nav_roll	float	目标横滚角
nav_pitch	float	目标俯仰角
alt_error	float	高度差
aspd_error	float	保留为 0
xtrack_error	float	保留为 0
nav_bearing	int16_t	目标航向角
target_bearing	int16_t	航向误差角
wp_dist	uint16_t	距离目标地点剩余距离

REQUEST_ADTA_STREAM,设置数据流

MSG 编号	名字	输入输出
66	REQUEST_DATA_STREAM	I
描述	设置传感器,RC 通道等数据流的输出使能和频率	
成员	类型	描述
req_message_rate	uint16_t	数据流速率
target_system	uint8_t	目标系统
target_component	uint8_t	目标组件
req_stream_id	uint8_t	数据流 ID
start_stop	uint8_t	使能

ardupilot 的接收处理流程为:

1. 如果 start_stop 为 0,将频率设为 0
2. 如果 start_stop 为 1,将频率设为 req_message_rate
3. 根据 req_stream_id 的数据流类型设置对应数据的输出频率,频率由 req_message_rate 决定

RC_CHANNELS_OVERRIDE,设置 RC 输入值

MSG 编号	名字	输入输出
70	RC_CHANNELS_OVERRIDE	I
描述	覆盖 RC 输入通道上的数据	

成员	类型	描述
chan1 _raw	uint16_t	通道 1
chan2_raw	uint16_t	通道 2
chan3 _raw	uint16_t	通道 3
chan4 _raw	uint16_t	通道 4
chan5 _raw	uint16_t	通道 5
chan6 _raw	uint16_t	通道 6
chan7 _raw	uint16_t	通道 7
chan8 _raw	uint16_t	通道 8
target_system	uint8_t	目标系统
target_component	uint8_t	目标组件

ardupilot 的接收处理流程为:

1. 将通道数值写入 RCIN 中
2. 设置 last_heartbeat_ms 为当前时间

VFR_HUD,HUD 信息

MSG 编号	名字	输入输出
74	VFR_HUD	0
描述	发送 HUD 信息	
成员	**类型**	**描述**
airspeed	float	空速(GPS 的地速)
groundspeed	float	地速(GPS 的地速)
alt	float	高度
climb	float	爬升率
heading	int16_t	航向角
throttle	uint16_t	油门

COMMAND_LONG,执行指定命令

MSG 编号	名字	输入输出
76	COMMAND_LONG	I
描述	执行指定命令	
成员	**类型**	**描述**
param1	float	参数 1
param2	float	参数 2
param3	float	参数 3
param4	float	参数 4
param5	float	参数 5

param6	float	参数 6
param7	float	参数 7
command	uint16_t	命令号
target_system	uint8_t	保留
target_component	uint8_t	保留
confirmation	uint8_t	

ardupilot 的接收处理流程为:

1. 根据 command 中的命令执行对应命令
2. 根据执行结果返回 ACK 包

COMMAND_ACK,命令应答包

MSG 编号	**名字**	**输入输出**
77	COMMAND_ACK	IO
描述	反馈命令执行结果	
成员	**类型**	**描述**
command	uint16_t	命令号(MAV_CMD)
result	uint8_t	执行结果(MAV_RESULT)

ardupilot 的接收处理流程为:

1. 设置 command_ack_counter 自加 1

SET_POSITION_TARGET_LOCAL_NED,设置飞行的相对目标

MSG 编号	**名字**	**输入输出**
84	SET_POSITION_TARGET_LOCAL_NED	I
描述	设置飞行的相对目标	
成员	**类型**	**描述**
time_boot_ms	uint32_t	系统时间
x	float	NED 位置 X
y	float	NED 位置 Y
z	float	NED 位置 Z
vx	float	NED 速度 X
vy	float	NED 速度 Y
vz	float	NED 速度 Z
afx	float	NED 加速度 X
afy	float	NED 加速度 Y
afz	float	NED 加速度 Z
yaw	float	偏航角
yaw_rate	float	偏航角速率

type_mask	uint16_t	使能标志
target_system	uint8_t	保留
target_component	uint8_t	保留
coordinate_frame	uint8_t	坐标框架

ardupilot 的接收处理流程为:

1. 如果当前飞行模式不为 GUIDED 和 AUTO 模式则直接返回
2. 根据 type_mask 来决定是否使用其中的位置,速度和加速度项

SET_POSITION_TARGET_GLOBAL_INT,设置飞行的绝对目标

MSG 编号	**名字**	**输入输出**
86	SET_POSITION_TARGET_GLOBAL_INT	I
描述	设置飞行的绝对目标	
成员	**类型**	**描述**
time_boot_ms	uint32_t	系统时间
lat_int	int32_t	WGS84 位置 X
lon_int	int32_t	WGS84 位置 Y
alt	float	高度
vx	float	NED 速度 X
vy	float	NED 速度 Y
vz	float	NED 速度 Z
afx	float	NED 加速度 X
afy	float	NED 加速度 Y
afz	float	NED 加速度 Z
yaw	float	偏航角
yaw_rate	float	偏航角速率
type_mask	uint16_t	使能标志
target_system	uint8_t	保留
target_component	uint8_t	保留
coordinate_frame	uint8_t	坐标框架

ardupilot 的接收处理流程为:

1. 如果当前飞行模式不为 GUIDED 和 AUTO 模式则直接返回
2. 根据 type_mask 来决定是否使用其中的位置,速度和加速度项

OPTICAL_FLOW,光流传感器数据

MSG 编号	**名字**	**输入输出**
100	OPTICAL_FLOW	O
描述	发送光流传感器的数据	

成员	类型	描述
time_usec	uint64_t	系统时间
flow_comp_m_x	float	X 轴上移动的距离
flow_comp_m_y	float	Y 轴上移动的距离
ground_distance	float	对地距离
flow_x	int16_t	X 轴上移动的像素
flow_y	int16_t	Y 轴上移动的像素
sensor_id	uint8_t	传感器 ID
quality	uint8_t	光流品质

ardupilot 的发送处理流程为:

1. 设置 flow_x(y)为 flowRate
2. 设置 flow_comp_m_x(y)为 bodyRate

RADIO,数据状态

MSG 编号	名字	输入输出
166/109	RADIO_STATUS/RADIO	I
描述	发送数传的传输状态	
成员	**类型**	**描述**
rxerrors	uint16_t	收到的错包量
fixed	uint16_t	收到的可纠正错包量
rssi	uint8_t	本地信号强度
remrssi	uint8_t	远端信号强度
txbuf	uint8_t	剩余的发射缓冲空间
noise	uint8_t	后台噪音强度
remnoise	uint8_t	远端后台噪音强度

ardupilot 的接收处理流程为:

1. 根据 txbuf 的数值来调整所有数据流的传输的频率

SCALED_IMU2,2 号 imu 原始数据

MSG 编号	名字	输入输出
116	SCALED_IMU2	O
描述	发送 2 号 IMU 的原始数据	
成员	**类型**	**描述**
time_boot_ms	uint32_t	系统时间
xacc	int16_t	加速计 X
yacc	int16_t	加速计 Y
zacc	int16_t	加速计 Z
xgyro	int16_t	陀螺仪 X

ygyro	int16_t	陀螺仪 Y
zgyro	int16_t	陀螺仪 Z
xmag	int16_t	罗盘 X
ymag	int16_t	罗盘 Y
zmag	int16_t	罗盘 Z

LOG_REQUEST_LIST,查询 log 列表

MSG 编号	**名字**	**输入输出**
117	LOG_REQUEST_LIST	I
描述	接收该包后返回一个 LOG_ENTRY 包描述 log 列表信息	
成员	**类型**	**描述**
start	uint16_t	日志开始序列号
end	uint16_t	日志结束序列号
target_system	uint8_t	保留
target_component	uint8_t	保留

ardupilot 的接收处理流程为:

1. 设置_log_num_logs 为 log 数量
2. 设置_log_next_list_entry 为 start,设置_log_last_list_entry 为 end
3. 设置_log_listing 为 true
4. 根据_log_next_list_entry 从存储器中获得 log 的大小和时间
5. 根据获得的信息返回一个 LOG_ENTRY 包

LOG_ENTRY,log 列表信息

MSG 编号	**名字**	**输入输出**
118	LOG_ENTRY	O
描述	发送 log 列表的信息	
成员	**类型**	**描述**
time_utc	uint32_t	UTC 时间
size	uint32_t	log 大小
id	uint16_t	log 开始序列号
num_logs	uint16_t	log 总数
last_log_num	uint16_t	log 结束序列号

LOG_REQUEST_DATA,申请发送 log

MSG 编号	**名字**	**输入输出**
119	LOG_REQUEST_DATA	I
描述	申请发送 log 数据,数据根据该包中的信息从存储器中读取	
成员	**类型**	**描述**

ofs	uint32_t	日志偏移量
count	uint32_t	字节数
id	uint16_t	日志 ID 号
target_system	uint8_t	保留
target_component	uint8_t	保留

ardupilot 的接收处理流程为:

1. 设置_log_listing 为 false
2. 检查_log_sending 是否为发送状态或者 id 号是否和当前发送的 log 号不符,如果其中一项符合则
   - 设置_log_num_data 为 id
   - 设置 log_data_size 为 log 的大小
   - 设置_log_data_page 为 log 的起始位置
3. 设置_log_data_offset 为 log 的偏移位置
4. 设置_log_data_remaining 为 log 的待发送大小
5. 设置_log_sending 为 true
6. 从存储器中获取数据并通过 LOG_DATA 包进行发送

LOG_DATA,发送 log 数据

MSG 编号	**名字**	**输入输出**
120	LOG_DATA	0
描述	发送 log 数据,每次最大 90 个字节	
成员	**类型**	**描述**
ofs	uint32_t	log 偏移
id	uint16_t	log 序列号
count	uint8_t	log 字节数
data	uint8_t[90]	log 数据

LOG_ERASE,擦除所有 log 信息

MSG 编号	**名字**	**输入输出**
121	LOG_ERASE	I
描述	擦除存储器上的所有 log	
成员	**类型**	**描述**
target_system	uint8_t	保留
target_component	uint8_t	保留

ardupilot 的接收处理流程为:

1. 擦除所有 log

LOG_REQUEST_END,停止发送 log

MSG 编号	名字	输入输出
122	LOG_REQUEST_END	I
描述	停止发送 log 信息	
成员	类型	描述
target_system	uint8_t	保留
target_component	uint8_t	保留

ardupilot 的接收处理流程为:

1. 设置_log_sending 为 false

POWER_STATUS,电源状态

MSG 编号	名字	输入输出
125	POWER_STATUS	O
描述	发送系统各个电源的当前状态	
成员	类型	描述
Vcc	uint16_t	系统 5V 状态
Vservo	uint16_t	伺服电机电源状态
flags	uint16_t	电源状态

SCALED_IMU3，3 号 imu 原始数据

MSG 编号	名字	输入输出
129	SCALED_IMU3	O
描述	发送 3 号 IMU 的原始数据	
成员	类型	描述
time_boot_ms	uint32_t	系统时间
xacc	int16_t	加速计 X
yacc	int16_t	加速计 Y
zacc	int16_t	加速计 Z
xgyro	int16_t	陀螺仪 X
ygyro	int16_t	陀螺仪 Y
zgyro	int16_t	陀螺仪 Z
xmag	int16_t	罗盘 X
ymag	int16_t	罗盘 Y
zmag	int16_t	罗盘 Z

SENSOR_OFFSETS,传感器校准数据

MSG 编号	名字	输入输出
150	SENSOR_OFFSETS	O
描述	1 号 IMU 传感器的校准数据	

成员	类型	描述
mag_declination	float	磁偏角
raw_press	int32_t	气压计的气压值
raw_temp	int32_t	气压计的温度
gyro_cal_x	float	陀螺仪 X 校正量
gyro_cal_y	float	陀螺仪 Y 校正量
gyro_cal_z	float	陀螺仪 Z 校正量
accel_cal_x	float	加速度 X 校正量
accel_cal_y	float	加速度 Y 校正量
accel_cal_z	float	加速度 Z 校正量
mag_ofs_x	int16_t	罗盘 X 校正量
mag_ofs_y	int16_t	罗盘 Y 校正量
mag_ofs_z	int16_t	罗盘 Z 校正量

MEMINFO,内存信息

MSG 编号	名字	输入输出
152	MEMINFO	0
描述	当前系统的内存使用情况	
成员	类型	描述
brkval	uint16_t	堆头
freemem	uint16_t	剩余内存容量

FENCE_STATUS,未实现

MSG 编号	名字	输入输出
162	FENCE_STATUS	0
描述	未实现	
成员	类型	描述

AHRS,姿态解算信息

MSG 编号	名字	输入输出
163	AHRS	0
描述	姿态解算信息	
成员	类型	描述
omegaIx	float	陀螺仪积分 X
omegaIy	float	陀螺仪积分 Y
omegaIz	float	陀螺仪积分 Z
accel_weight	float	保留为 0
renorm_val	float	保留为 0

error_rp	float	imu 和 gps 之间的差值
error_yaw	float	imu 和罗盘/GPS 之间的差值

HWSTATUS,硬件状态信息

MSG 编号	**名字**	**输入输出**
165	HWSTATUS	0
描述	描述当前硬件状态	
成员	**类型**	**描述**
Vcc	uint16_t	系统 5V 电压值
I2Cerr	uint8_t	I2C 传输失败计数器

RANGEFINDER,测距器信息

MSG 编号	**名字**	**输入输出**
173	RANGEFINDER	0
描述	测距器状态信息	
成员	**类型**	**描述**
distance	float	测距值
voltage	float	电压值

EKF_STATUS_REPORT,EKF 状态信息

MSG 编号	**名字**	**输入输出**
193	EKF_STATUS_REPORT	0
描述	描述 EKF 当前工作状态信息	
成员	**类型**	**描述**
velocity_variance	float	速度方差
pos_horiz_variance	float	位置水平方差
pos_vert_variance	float	位置垂直方差
compass_variance	float	罗盘方差
tas_variance	float	空速方差
flags	uint16_t	EKF 各组件工作状态

PID_TUNING,内环 pid 状态

MSG 编号	**名字**	**输入输出**
194	PID_TUNING	0
描述	描述当前内环 PID 的状态信息	
成员	**类型**	**描述**
desired	float	期望角速率
achieved	float	实际角速率

FF	float	PID 最大值
P	float	P 值
I	float	I 值
D	float	D 值
axis	uint8_t	轴通道

VIBRATION,机体震动状态

MSG 编号	**名字**	**输入输出**
241	VIBRATION	0
描述	描述当前机体的震动状态	
成员	**类型**	**描述**
time_usec	uint64_t	系统时间
vibration_x	float	X 轴震动等级
vibration_y	float	Y 轴震动等级
vibration_z	float	Z 轴震动等级
clipping_0	uint32_t	加速计 1 的 G 值临界计数器
clipping_1	uint32_t	加速计 2 的 G 值临界计数器
clipping_2	uint32_t	加速计 3 的 G 值临界计数器

STATUSTEXT,发送文字信息

MSG 编号	**名字**	**输入输出**
253	STATUSTEXT	0
描述	发送文字信息,最大为 50 个字节	
成员	**类型**	**描述**
severity	uint8_t	安全等级
text	char[50]	字符信息

附录 1：MAV_CMD

编号	**名字**
16	NAV_WAYPOINT
成员	**描述**
param_1	停留时间
param_2	
param_3	
param_4	
param_5	纬度
param_6	经度
param_7	高度

编号	**名字**

17	NAV_LOITER_UNLIM
成员	**描述**
param_1	
param_2	
param_3	旋转方向
param_4	
param_5	纬度
param_6	经度
param_7	高度

编号	**名字**
18	NAV_LOITER_TURNS
成员	**描述**
param_1	转圈次数
param_2	
param_3	任务半径,旋转方向使用正负进行区分
param_4	
param_5	纬度
param_6	经度
param_7	高度

编号	**名字**
19	NAV_LOITER_TIME
成员	**描述**
param_1	时间
param_2	
param_3	旋转方向
param_4	
param_5	纬度
param_6	经度
param_7	高度

编号	**名字**
20	NAV_RETURN_TO_LAUNCH
成员	**描述**
param_1	
param_2	
param_3	
param_4	
param_5	纬度
param_6	经度
param_7	高度

编号	名字
21	NAV_LAND
成员	**描述**
param_1	
param_2	
param_3	
param_4	
param_5	纬度
param_6	经度
param_7	高度

编号	名字
22	NAV_TAKEOFF
成员	**描述**
param_1	
param_2	
param_3	
param_4	
param_5	纬度
param_6	经度
param_7	高度

编号	名字
82	NAV_SPLINE_WAYPOINT
成员	**描述**
param_1	时间
param_2	
param_3	
param_4	
param_5	纬度
param_6	经度
param_7	高度

编号	名字
92	NAV_GUIDED_ENABLE
成员	**描述**
param_1	使能
param_2	
param_3	
param_4	
param_5	
param_6	
param_7	

编号	名字
112	CONDITION_DELAY
成员	**描述**
param_1	时间
param_2	
param_3	
param_4	
param_5	
param_6	
param_7	

编号	名字
115	CONDITION_YAW
成员	**描述**
param_1	目标航向角
param_2	偏航角速率
param_3	正反旋转
param_4	相对绝对
param_5	
param_6	
param_7	

编号	名字
177	DO_JUMP
成员	**描述**
param_1	序列号
param_2	重复次数
param_3	
param_4	
param_5	
param_6	
param_7	

编号	名字
178	DO_CHANGE_SPEED
成员	**描述**
param_1	速度类型
param_2	目标速度
param_3	油门
param_4	
param_5	
param_6	

param_7	

编号	**名字**
179	DO_SET_HOME
成员	**描述**
param_1	是否使用当前位置
param_2	
param_3	
param_4	
param_5	纬度
param_6	经度
param_7	高度

编号	**名字**
181	DO_SET_RELAY
成员	**描述**
param_1	继电器号
param_2	使能
param_3	
param_4	
param_5	
param_6	
param_7	

编号	**名字**
182	DO_REPEAT_RELAY
成员	**描述**
param_1	继电器号
param_2	开启次数
param_3	开启时间
param_4	
param_5	
param_6	
param_7	

编号	**名字**
183	DO_SET_SERVO
成员	**描述**
param_1	伺服输出通道
param_2	输出值
param_3	
param_4	
param_5	

param_6	
param_7	

编号	**名字**
184	DO_REPEAT_SERVO
成员	**描述**
param_1	伺服输出通道
param_2	输出值
param_3	开启次数
param_4	开启时间
param_5	
param_6	
param_7	

编号	**名字**
201	DO_SET_ROI
成员	**描述**
param_1	兴趣点类型
param_2	
param_3	
param_4	
param_5	纬度
param_6	经度
param_7	高度

编号	**名字**
202	DO_DIGICAM_CONFIGURE
成员	**描述**
param_1	
param_2	
param_3	
param_4	
param_5	
param_6	
param_7	

编号	**名字**
203	DO_DIGICAM_CONTROL
成员	**描述**
param_1	
param_2	
param_3	
param_4	

param_5	
param_6	
param_7	

编号	**名字**
206	DO_SET_CAM_TRIGG_DIST
成员	**描述**
param_1	
param_2	
param_3	
param_4	
param_5	
param_6	
param_7	

编号	**名字**
207	DO_FENCE_ENABLE
成员	**描述**
param_1	使能
param_2	
param_3	
param_4	
param_5	
param_6	
param_7	

编号	**名字**
208	DO_PARACHUTE
成员	**描述**
param_1	
param_2	
param_3	
param_4	
param_5	
param_6	
param_7	

编号	**名字**
209	DO_MOTOR_TEST
成员	**描述**
param_1	
param_2	
param_3	

param_4	
param_5	
param_6	
param_7	

编号	**名字**
211	DO_GRIPPER
成员	**描述**
param_1	
param_2	
param_3	
param_4	
param_5	
param_6	
param_7	

编号	**名字**
222	DO_GUIDED_LIMITS
成员	**描述**
param_1	
param_2	
param_3	
param_4	
param_5	
param_6	
param_7	

编号	**名字**
241	PREFLIGHT_CALIBRATION
成员	**描述**
param_1	
param_2	
param_3	
param_4	
param_5	
param_6	
param_7	

编号	**名字**
242	PREFLIGHT_SET_SENSOR_OFFSETS
成员	**描述**
param_1	
param_2	

param_3	
param_4	
param_5	
param_6	
param_7	

编号	**名字**
246	PREFLIGHT_REBOOT_SHUTDOWN
成员	**描述**
param_1	
param_2	
param_3	
param_4	
param_5	
param_6	
param_7	

编号	**名字**
300	MISSION_START
成员	**描述**
param_1	
param_2	
param_3	
param_4	
param_5	
param_6	
param_7	

编号	**名字**
400	COMPONENT_ARM_DISARM
成员	**描述**
param_1	
param_2	
param_3	
param_4	
param_5	
param_6	
param_7	

编号	**名字**
520	REQUEST_AUTOPILOT_CAPABILITIES
成员	**描述**
param_1	

param_2	
param_3	
param_4	
param_5	
param_6	
param_7	

第六章 Pid 控制

第七章 相关实验测试

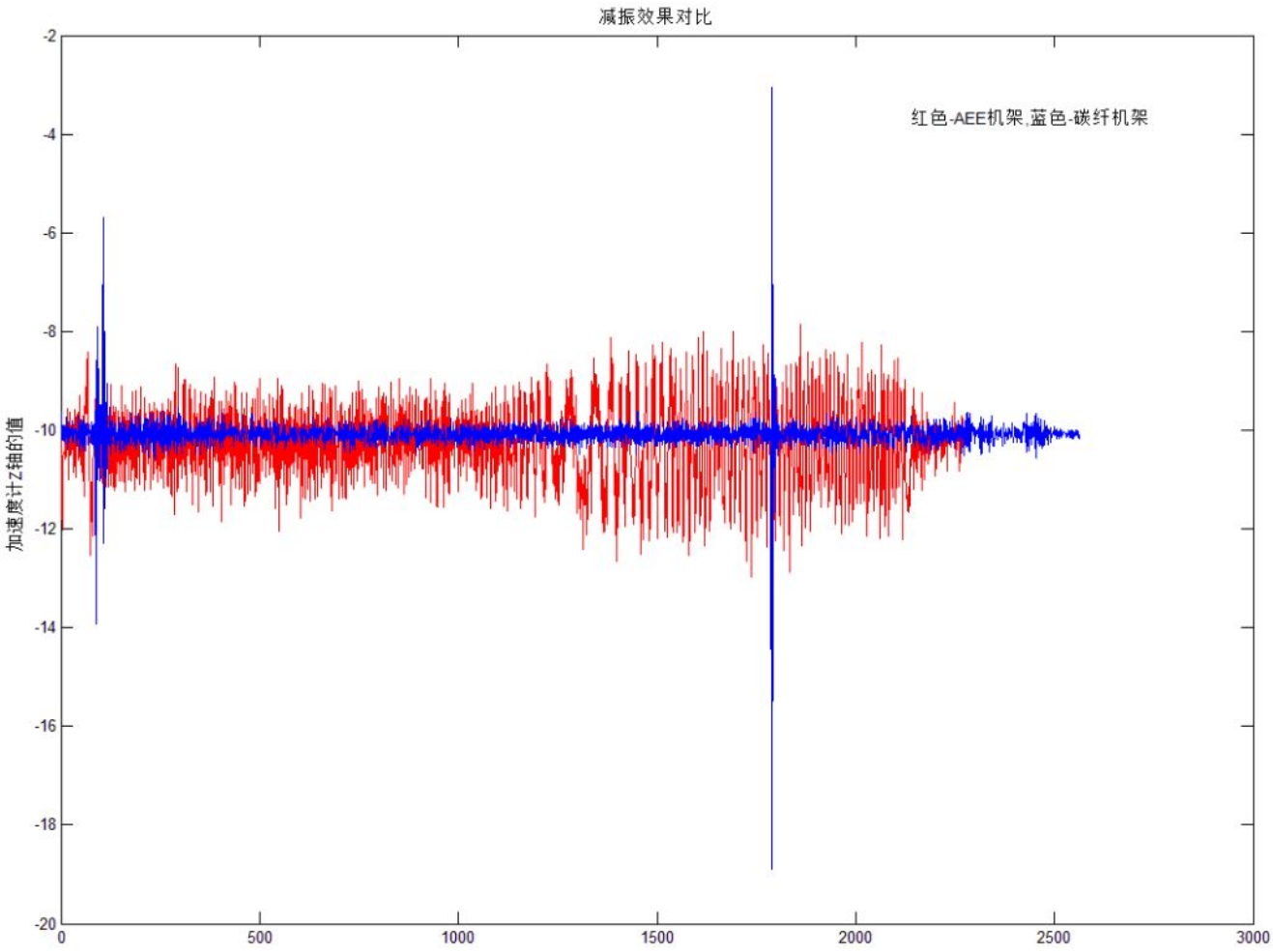
实验一 IMU 减振实验报告分析

（本实验作者：邹荣）

本次实验主要是研究机架、桨叶以及电机对 IMU 传感器振动干扰的影响情况。实验中分别对 AP10 机架、采购的碳纤机架、Tmotor 桨叶、大疆桨叶、AP10 自带电机以及大疆电机分别进行对比实验。

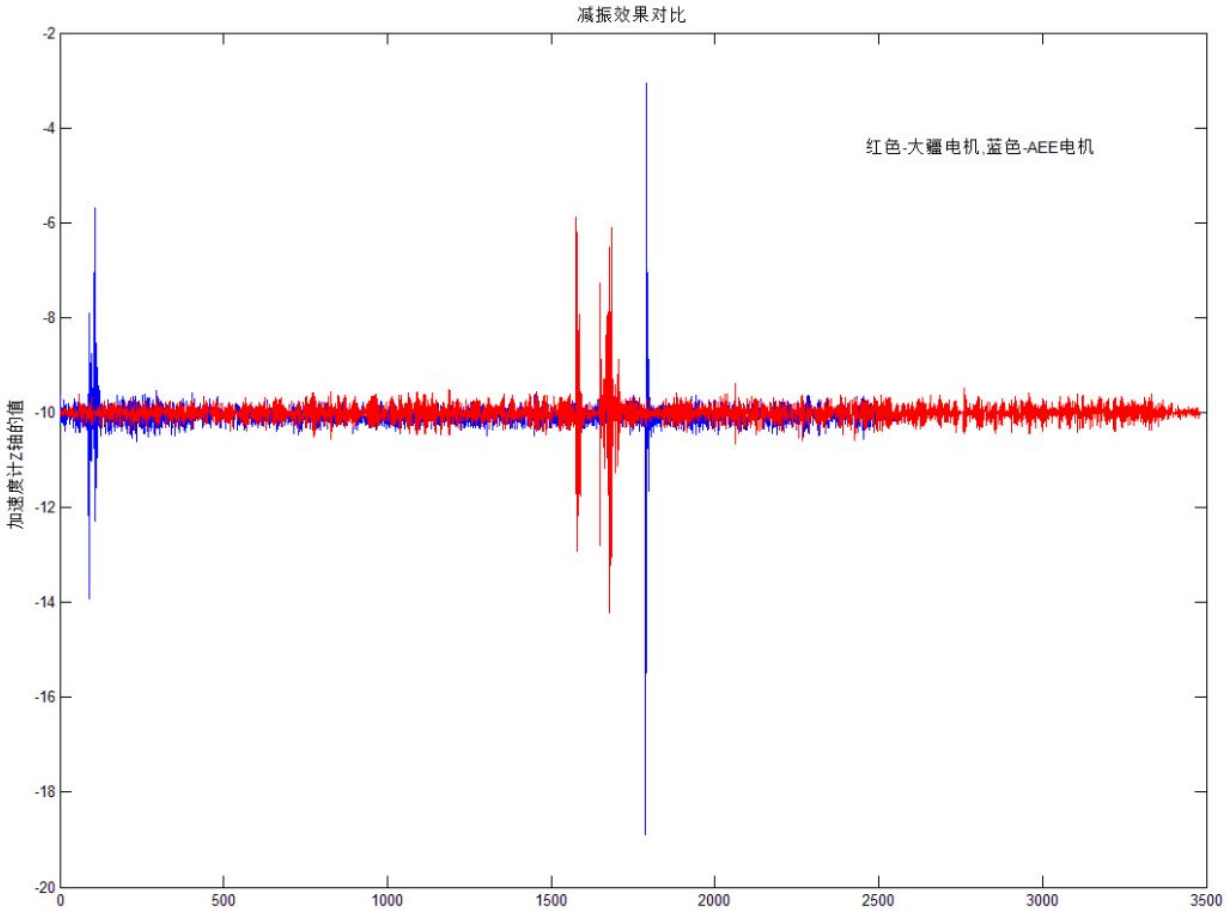
本次实验分别进行了如下 6 次对比测试：

第一次实验：不带桨叶，采用 AP10 机架和碳纤机架开电机进行对比测试(电机、电调、主板都采用的是 AP10 的配置)，记录 IMU 的加速度计 Z 轴的原始数据对比如下：



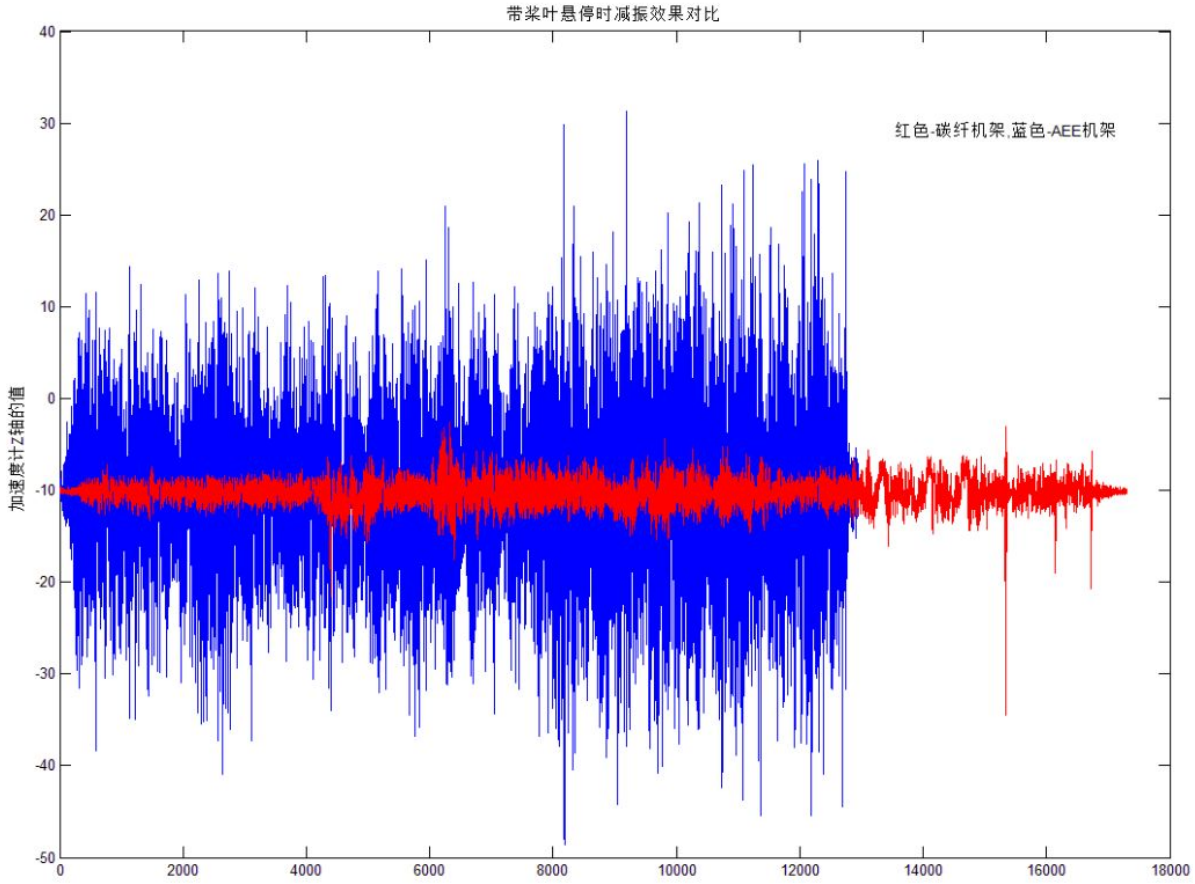
从图中可以看出：采用碳纤机架可以明显的降低高频噪声对 IMU 加速度计的干扰。

第二次实验：不带桨叶，分别对大疆电机和 AP10 自带电机进行对比测试（其他配置均一致，都采用碳纤机架和 AP10 自带电调以及主板），记录 IMU 的加速度计 Z 轴的原始数据对比如下：



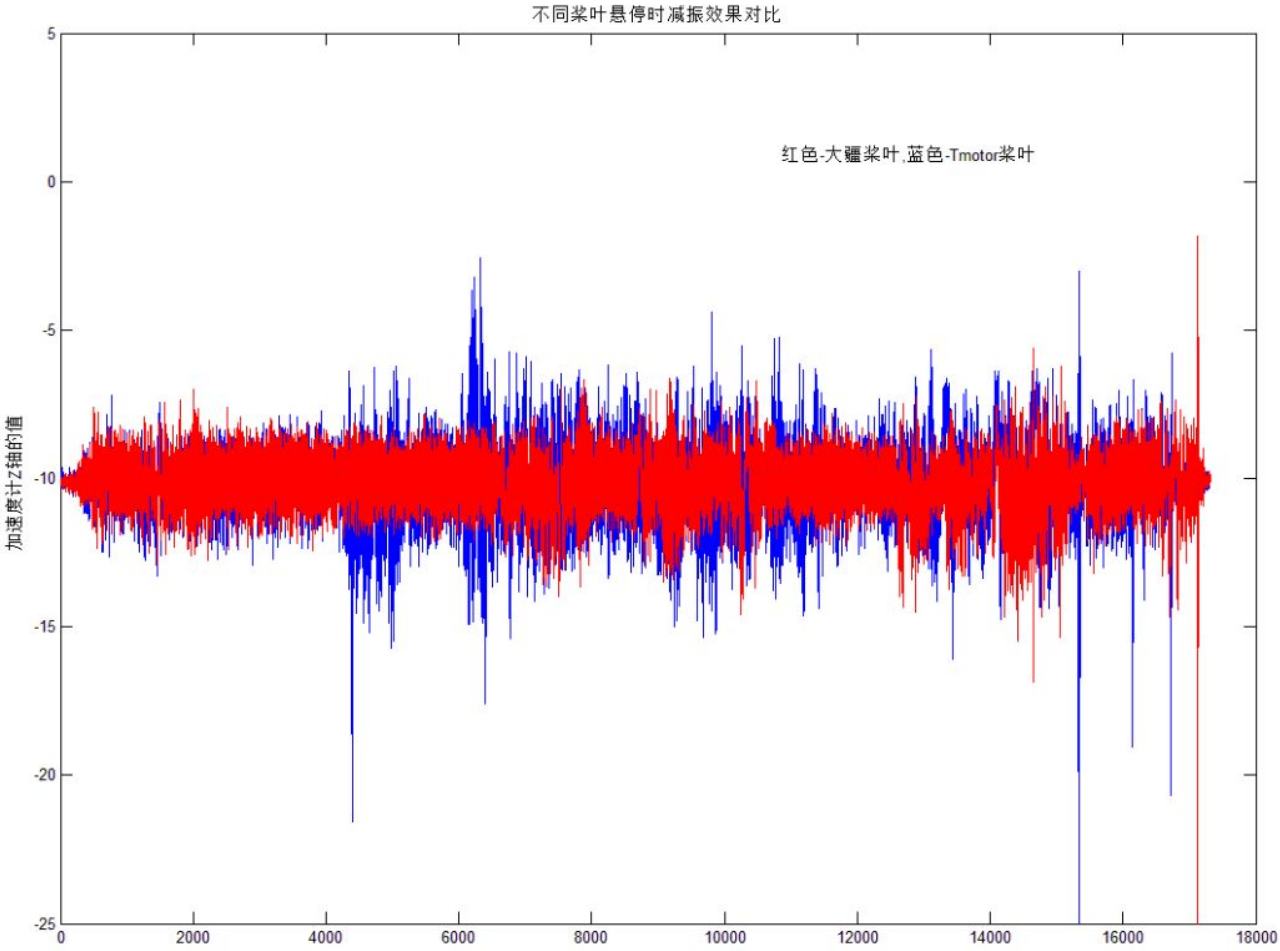
从图中可以看出，平稳状态下大疆电机对 IMU 造成的高频噪声比 AP10 自带的电机会更小一些（图中变换比较大的部分是由于油门打杆导致的）。

第三次实验：带 Tmotor 桨叶悬停飞行，分别对 AP10 机架和碳纤机架进行对比测试（其他配置均一致，都采 AP10 自带的电机、电调以及主板），记录 IMU 的加速度计 Z 轴的原始数据对比如下：



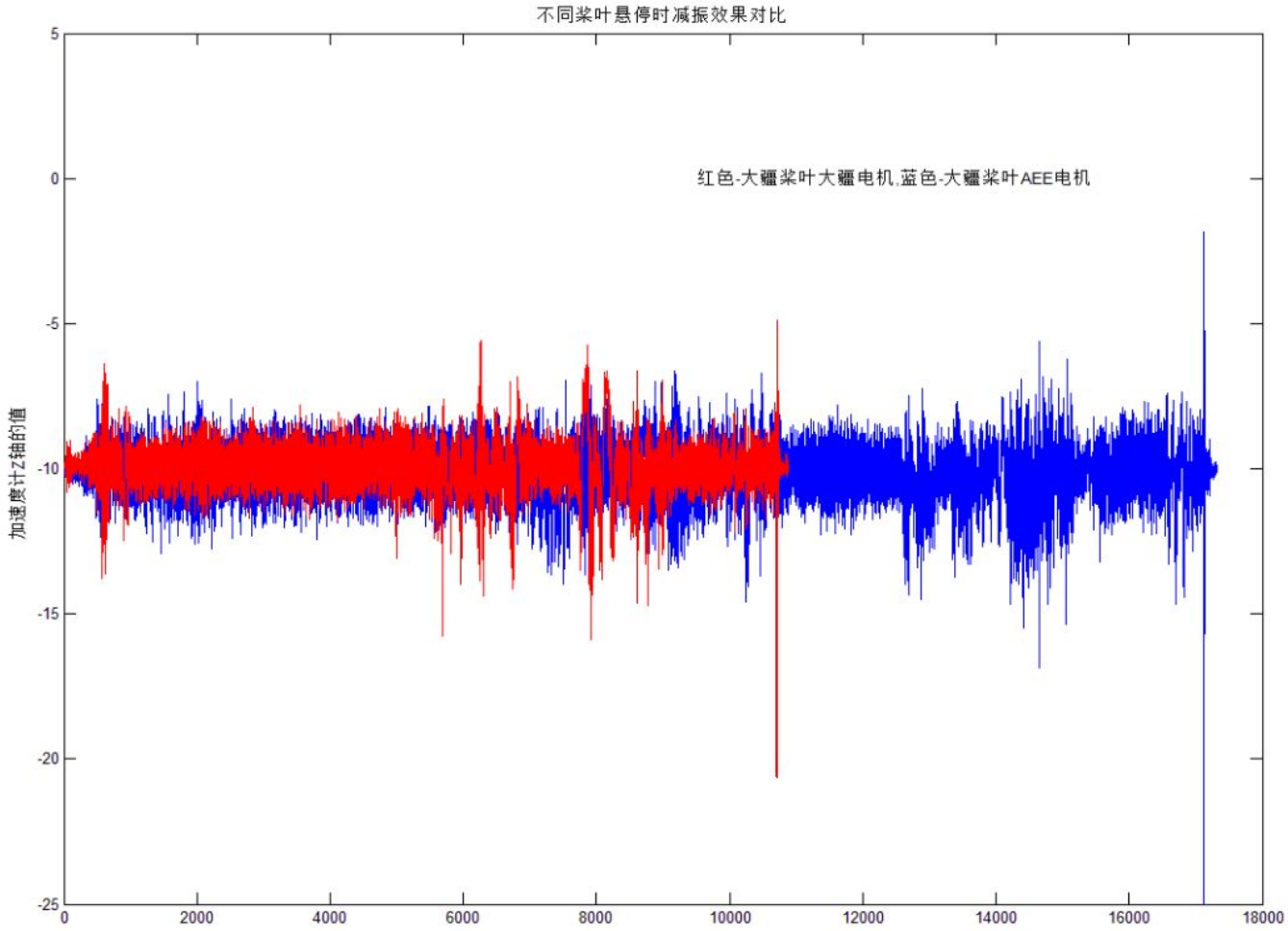
从图中可以看出：在悬停飞行时，采用碳纤机架对比 AP10 机架可以明显的降低高频噪声对 IMU 加速度计原始数据的干扰。

第四次实验：悬停飞行，分别对大疆桨叶和 Tmotor 桨叶进行对比测试（其他配置均一致，都采碳纤机架，AP10 自带的电机、电调以及主板），记录 IMU 的加速度计 Z 轴的原始数据对比如下：



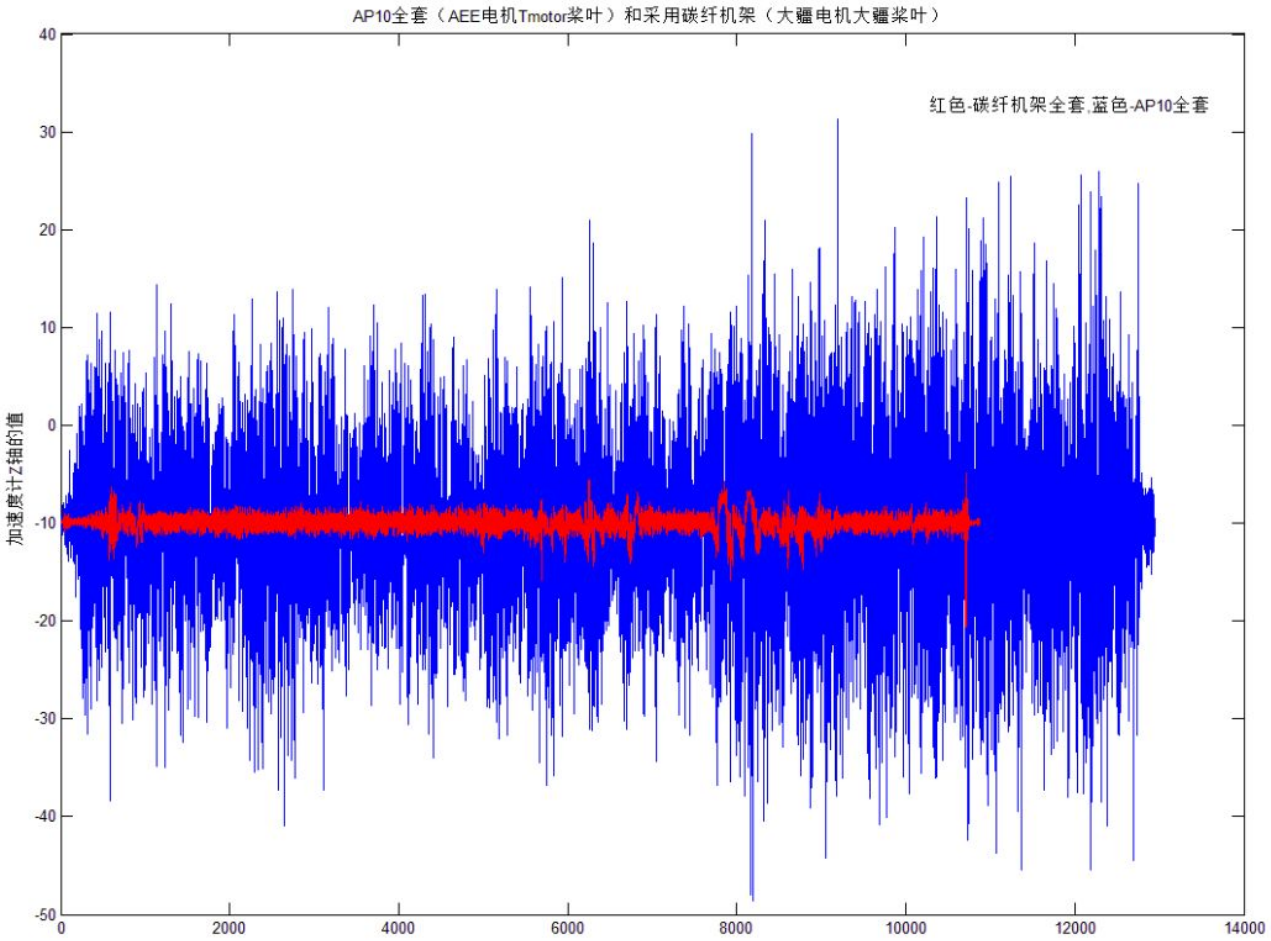
从图中可知：大疆桨叶相对 Tmotor 桨叶对 IMU 的加速度计 Z 轴产生的高频噪声干扰相对来说会更小一些。

第五次实验：带大疆桨叶悬停飞行，分别对大疆电机和 AP10 自带的电机进行对比测试（其他配置均一致，都采碳纤机架、AP10 自带的电调以及主板），记录 IMU 的加速度计 Z 轴的原始数据对比如下：



从图中可知：大疆电机相对 AP10 自带的电机对 IMU 的加速度计 Z 轴产生的高频噪声干扰相对来说会更小一些。

第六次实验：悬停飞行，分别对 AP10 飞机（AP10 自带的电机、电调、主板以及 Tmotor 桨叶）和采用碳纤机架组装的飞机（AP10 主板、AP10 自带电调、大疆电机以及大疆桨叶）进行对比测试，记录 IMU 的加速度计 Z 轴的原始数据对比如下：



从图中可知：用碳纤机架组装的飞机相比 AP10 飞机来说，减振效果会好 5 倍左右。

总结：从整个实验分析来看，机架、电机、桨叶对 IMU 振动噪声都会带来不同程度的影响。