APM 四旋翼飞行控制分为两级，内层 AHRS(航向姿态参考系统)利用 IMU(陀螺仪和加速度计)数据快速更新迭代(1s 更新 7000-8000 次)；外层控制包含电机、气压计、数传、遥控、地面站等，更新的比较慢。

APM 中 AHRS 更新原理：利用陀螺仪角速度数据更新 DCM 矩阵(方向余弦)；归一化矩阵；然后利用加速度计和 gps 数据进行误差修正；最后从方向余弦阵中算出欧拉角，进行控制。对应的入口函数为：ArduCopter.pde 文件中的 read_AHRS(),即函数 ahrs.update()的调用。

```
AP_AHRS_DCM::update(void)
{
    float delta_t;

    // tell the IMU to grab some data
    _ins.update();
```

读数据

```
    // ask the IMU how much time this sensor reading represents
    delta_t = _ins.get_delta_time();

    // if the update call took more than 0.2 seconds then discard it,
    // otherwise we may move too far. This happens when arming motors
    // in ArduCopter
    if (delta_t > 0.2f) {
        memset(&_ra_sum[0], 0, sizeof(_ra_sum));
        _ra_deltat = 0;
        return;
    }

    // Integrate the DCM matrix using gyro inputs
    matrix_update(delta_t);
```

更新方向余弦阵

```
    // Normalize the DCM matrix
    normalize();
```

矩阵归一化

```
    // Perform drift correction
    drift_correction(delta_t);
```

误差修正

```
    // paranoid check for bad values in the DCM matrix
    check_matrix();
    // Calculate pitch, roll, yaw for stabilization and navigation
    euler_angles();
```

从矩阵中算出飞控需要的俯仰，翻滚，偏航角

```
    // update trig values including _cos_roll, cos_pitch
    update_trig();
} ?end update ?
```

以下为 4 个姿态控制步骤的具体分析。

1. DCM 矩阵更新

```
AP_AHRS_DCM::matrix_update(float _G_Dt)
{
    // note that we do not include the P terms in _omega. This is
    // because the spin_rate is calculated from _omega.length(),
    // and including the P terms would give positive feedback into
    // the _P_gain() calculation, which can lead to a very large P
    // value
    _omega.zero();

    // average across all healthy gyros. This reduces noise on systems
    // with more than one gyro
    uint8_t healthy_count = 0;
    for (uint8_t i=0; i<_ins.get_gyro_count(); i++) {
        if (_ins.get_gyro_health(i)) {
            _omega += _ins.get_gyro(i);
            healthy_count++;
        }
    }
    if (healthy_count > 1) {
        _omega /= healthy_count;
    }
```

以上两个for循环为读取所有的陀螺仪数据，然后求平均值

```
    _omega += _omega_I;
```

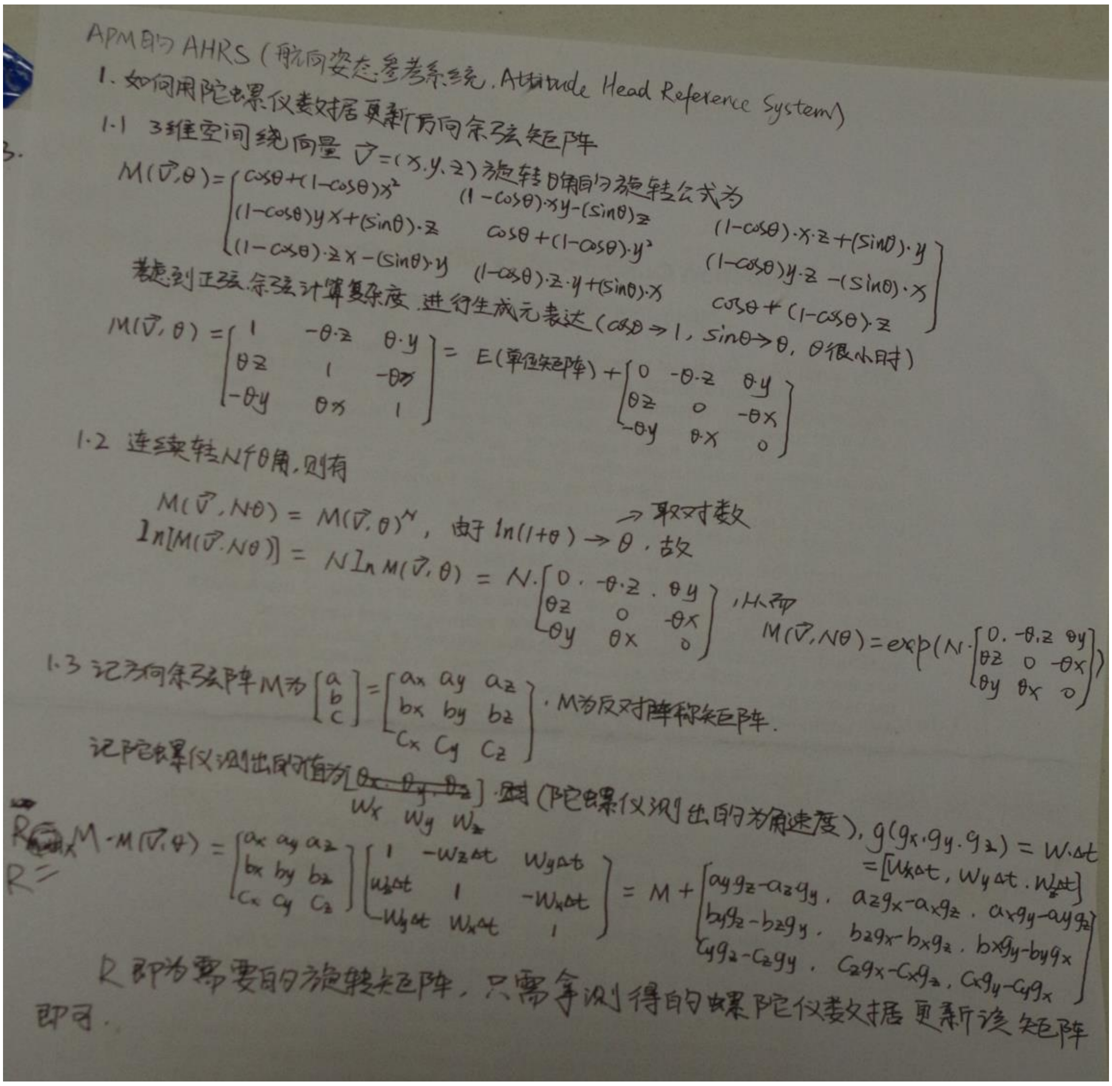
PI控制器，此处用积分器来修正陀螺仪数据

```
    _dcm_matrix.rotate((_omega + _omega_P + _omega_yaw_P) * _G_Dt);
```

参数中包含了用比例控制器修正陀螺仪数据，然后更新DCM矩阵

```
} ?end matrix_update ?
```

DCM 矩阵更新的数学公式推导：



APM的 AHRS（航向姿态参考系统，Attitude Head Reference System）

1. 如何用陀螺仪数据更新方向余弦矩阵

1.1 3维空间绕向量 $\vec{v}=(x,y,z)$ 旋转θ角的旋转公式为

$$M(\vec{v},\theta)=\begin{bmatrix} \cos\theta+(1-\cos\theta)x^2 & (1-\cos\theta)\cdot xy-(\sin\theta)z & (1-\cos\theta)\cdot x\cdot z+(\sin\theta)\cdot y \\ (1-\cos\theta)yx+(\sin\theta)\cdot z & \cos\theta+(1-\cos\theta)\cdot y^2 & (1-\cos\theta)y\cdot z-(\sin\theta)\cdot x \\ (1-\cos\theta)\cdot zx-(\sin\theta)\cdot y & (1-\cos\theta)\cdot z\cdot y+(\sin\theta)\cdot x & \cos\theta+(1-\cos\theta)\cdot z \end{bmatrix}$$

考虑到正弦.余弦计算复杂度.进行生成元表达（$\cos\theta\to 1$，$\sin\theta\to\theta$，θ很小时）

$$M(\vec{v},\theta)=\begin{bmatrix} 1 & -\theta\cdot z & \theta\cdot y \\ \theta z & 1 & -\theta x \\ -\theta y & \theta x & 1 \end{bmatrix}=E(\text{单位矩阵})+\begin{bmatrix} 0 & -\theta\cdot z & \theta\cdot y \\ \theta z & 0 & -\theta x \\ -\theta y & \theta x & 0 \end{bmatrix}$$

1.2 连续转N个θ角，则有

$M(\vec{v},N\theta)=M(\vec{v},\theta)^N$，由于 $\ln(1+\theta)\to\theta$，故 $\xrightarrow{\text{取对数}}$

$$\ln[M(\vec{v},N\theta)]=N\ln M(\vec{v},\theta)=N\cdot\begin{bmatrix} 0 & -\theta\cdot z & \theta y \\ \theta z & 0 & -\theta x \\ \theta y & \theta x & 0 \end{bmatrix}$$，从而

$$M(\vec{v},N\theta)=\exp\left(N\cdot\begin{bmatrix} 0 & -\theta z & \theta y \\ \theta z & 0 & -\theta x \\ \theta y & \theta x & 0 \end{bmatrix}\right)$$

1.3 记方向余弦阵M为 $\begin{bmatrix} a \\ b \\ c \end{bmatrix}=\begin{bmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ C_x & C_y & C_z \end{bmatrix}$，M为反对称矩阵.

记陀螺仪测出的值为 $[\cancel{\theta_x\ \theta_y\ \theta_z}]$ $[W_x\ W_y\ W_z]$（陀螺仪测出的为角速度），$g(g_x,g_y,g_z)=W\cdot\Delta t=[W_x\Delta t, W_y\Delta t, W_z\Delta t]$

$$R=M\cdot M(\vec{v},\theta)=\begin{bmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ C_x & C_y & C_z \end{bmatrix}\begin{bmatrix} 1 & -W_z\Delta t & W_y\Delta t \\ W_z\Delta t & 1 & -W_x\Delta t \\ -W_y\Delta t & W_x\Delta t & 1 \end{bmatrix}=M+\begin{bmatrix} a_yg_z-a_zg_y, & a_zg_x-a_xg_z, & a_xg_y-a_yg_x \\ b_yg_z-b_zg_y, & b_zg_x-b_xg_z, & b_xg_y-b_yg_x \\ C_yg_z-C_zg_y, & C_zg_x-C_xg_z, & C_xg_y-C_yg_x \end{bmatrix}$$

R即为需要的旋转矩阵，只需拿测得的螺陀仪数据更新该矩阵即可.

2. DCM 矩阵的归一化

   旋转矩阵为正交矩阵，但用陀螺仪数据的迭代更新过程容易引起数值计算误差，故每次迭代后都需正交化处理。

   归一化原理：利用正交矩阵的特性，两行点乘结果为 0，减小误差；再利用前两行叉乘的结果作为第三行。

```
AP_AHRS_DCM::normalize(void)
{
    float error;
    Vector3f t0, t1, t2;
                                          矩阵前两行点乘结果作为误差(本应该为0)
    error = _dcm_matrix.a * _dcm_matrix.b;                          // eq.18

    t0 = _dcm_matrix.a - (_dcm_matrix.b * (0.5f * error));         // eq.19 消除矩阵第一行误差
    t1 = _dcm_matrix.b - (_dcm_matrix.a * (0.5f * error));         // eq.19
    t2 = t0 % t1;                                   // c= a x b // eq.20    消除矩阵第二行误差
                          前两行叉乘结果作为第三行
    if (! renorm(t0, _dcm_matrix.a) ||
        ! renorm(t1, _dcm_matrix.b) ||
        ! renorm(t2, _dcm_matrix.c)) {            让每一行向量的范数为1
        // Our solution is blowing up and we will force back
        // to last euler angles
        _last_failure_ms = hal.scheduler->millis();
        AP_AHRS_DCM::reset(true);
    }
} ?end normalize ?
```

DCM 矩阵归一化的数学表达：

矩阵的规一化：

$$R=\begin{Bmatrix} r_{xx} & r_{xy} & r_{xz} \\ r_{yx} & r_{yy} & r_{yz} \\ r_{zx} & r_{zy} & r_{zz} \end{Bmatrix}=\begin{Bmatrix} X \\ Y \\ Z \end{Bmatrix}$$

1. $E_{error} = X^T \cdot Y = r_{xx}r_{yx} + r_{xy}r_{yy} + r_{xz}r_{yz}$

2. $\begin{pmatrix} r_{xx} \\ r_{xy} \\ r_{xz} \end{pmatrix}_{new} = \begin{pmatrix} r_{xx} \\ r_{xy} \\ r_{xz} \end{pmatrix} - \frac{E_{error}}{2} \cdot Y$; $\begin{pmatrix} r_{zx} \\ r_{zy} \\ r_{zz} \end{pmatrix}_{new} = X_{new} \times Y_{new}$

$\begin{pmatrix} r_{yx} \\ r_{yy} \\ r_{yz} \end{pmatrix}_{new} = \begin{pmatrix} r_{yx} \\ r_{yy} \\ r_{yz} \end{pmatrix} - \frac{E_{error}}{2} \cdot X$;

3. 使向量范数为一，

$$X' = \frac{X_{new}}{|X_{new}|}, \quad Y' = \frac{Y_{new}}{|Y_{new}|}, \quad Z' = \frac{Z_{new}}{|Z_{new}|},$$

$$R' = \begin{pmatrix} X' \\ Y' \\ Z' \end{pmatrix}$$

或者用 $\begin{cases} X' = \frac{1}{2}(3 - X_{new} \cdot X_{new}) X_{new} \\ Y' = \frac{1}{2}(3 - Y_{new} \cdot Y_{new}) Y_{new} \\ Z' = \frac{1}{2}(3 - Z_{new} \cdot Z_{new}) Z_{new} \end{cases}$ 使范数为1.

3. DCM 矩阵的误差修正

修正原理：利用 gps 获取的速度数据和加速度计获取的加速度数据叉乘作为误差；然后根据误差计算 PI 控制器的比例数值和积分数值。

代码比较多，如下：

```
AP_AHRS_DCM::drift_correction(float deltat)
{
    Vector3f velocity;
    uint32_t last_correction_time;

    // perform yaw drift correction if we have a new yaw reference
    // vector
    drift_correction_yaw();

    // rotate accelerometer values into the earth frame
    for (uint8_t i=0; i<_ins.get_accel_count(); i++) {
        if (_ins.get_accel_health(i)) {
            _accel_ef[i] = _dcm_matrix * _ins.get_accel(i
            // integrate the accel vector in the earth frame between GPS readings
            _ra_sum[i] += _accel_ef[i] * deltat;
        }
    }
```

消除偏航角误差

for循环为读取加速度计数据，然后把加速度数据旋转到地球坐标系。读取所有的加速度计数据，然后求平均

……

……

```
    if (_gps.last_fix_time_ms() == _ra_sum_start) {
        // we don't have a new GPS fix - nothing more to do
        return;
    }
    velocity = _gps.velocity();    利用gps传感器计算当前的飞行速度
    last_correction_time = _gps.last_fix_time_ms();
    if (_have_gps_lock == false) {
        // if we didn't have GPS lock in the last drift
        // correction interval then set the velocities equal
        _last_velocity = velocity;
    }
    _have_gps_lock = true;

    // keep last airspeed estimate for dead-reckoning purposes
    Vector3f airspeed = velocity - _wind;
    airspeed.z = 0;
    _last_airspeed = airspeed.length();
}
```

……

```
// equation 9: get the corrected acceleration vector in earth frame. Units
// are m/s/s
Vector3f GA_e;
GA_e = Vector3f(0, 0, -1.0f);

bool using_gps_corrections = false;
float ra_scale = 1.0f/(_ra_deltat*GRAVITY_MSS);

if (_flags.correct_centrifugal && (_have_gps_lock || _flags.fly_forward)) {
    float v_scale = gps_gain.get() * ra_scale;
    Vector3f vdelta = (velocity - _last_velocity) * v_scale;
    GA_e += vdelta;
    GA_e.normalize();     利用gps数据，计算地球坐标系下的
    if (GA_e.is_inf()) {  合加速度，可参考下边的数学公式
        // wait for some non-zero acceleration information
        _last_failure_ms = hal.scheduler->millis();
        return;
    }
    using_gps_corrections = true;
}
```

……

```
Vector3f GA_b[INS_MAX_INSTANCES];
int8_t besti = -1;
float best_error = 0;
for (uint8_t i=0; i<_ins.get_accel_count(); i++) {
    if (!_ins.get_accel_health(i)) {
        // only use healthy sensors
        continue;
    }
    _ra_sum[i] *= ra_scale;

    // get the delayed ra_sum to match the GPS lag
    if (using_gps_corrections) {
        GA_b[i] = ra_delayed(i, _ra_sum[i]);
    } else {
        GA_b[i] = _ra_sum[i];
    }
    if (GA_b[i].is_zero()) {
        // wait for some non-zero acceleration information
        continue;
    }     利用加速度计获取机身坐标系的合加速度，然后转换
    GA_b[i].normalize();到地球坐标系(前面读取时已经做过坐标系转换)
    if (GA_b[i].is_inf()) {
        // wait for some non-zero acceleration information
        continue;
    }
```

```
        error[i] = GA_b[i] % GA_e;
        float error_length = error[i].length();
        if (besti == -1 || error_length < best_error) {
            besti = i;
            best_error = error_length;
        }
    } ?end for uint8_ti=0;i<_ins.get... ?
```

向量叉乘，作为两种方式获取的加速度误差

......

```
    if (! _ins.healthy()) {
        error[besti].zero();
    } else {
        // convert the error term to body frame
        error[besti] = _dcm_matrix.mul_transpose(error[besti]);
    }
```

把误差向量转换到机身坐标系

......

```
    // base the P gain on the spin rate
    float spin_rate = _omega.length();

    // sanity check _kp value
    if (_kp < AP_AHRS_RP_P_MIN) {
        _kp = AP_AHRS_RP_P_MIN;
    }

    // we now want to calculate _omega_P and _omega_I. The
    // _omega_P value is what drags us quickly to the
    // accelerometer reading.
    _omega_P = error[besti] * _P_gain(spin_rate) * _kp;
    if (_flags.fast_ground_gains) {
        _omega_P *= 8;
    }
```

计算比例控制器数值

......

```
    // accumulate some integrator error
    if (spin_rate < ToRad(SPIN_RATE_LIMIT)) {
        _omega_I_sum += error[besti] * _ki * _ra_deltat;
        _omega_I_sum_time += _ra_deltat;
    }

    if (_omega_I_sum_time >= 5) {
        // limit the rate of change of omega_I to the hardware
        // reported maximum gyro drift rate. This ensures that
        // short term errors don't cause a buildup of omega_I
        // beyond the physical limits of the device
        float change_limit = _gyro_drift_limit * _omega_I_sum_time;
        _omega_I_sum.x = constrain_float(_omega_I_sum.x, -change_limit, change_limit);
        _omega_I_sum.y = constrain_float(_omega_I_sum.y, -change_limit, change_limit);
        _omega_I_sum.z = constrain_float(_omega_I_sum.z, -change_limit, change_limit);
        _omega_I += _omega_I_sum;
        _omega_I_sum.zero();
        _omega_I_sum_time = 0;
    }
```

计算积分控制器数值

每隔5s把累计的积分控制器数值作用于陀螺仪的输出

误差修正的数学公式推导：

2. 如何用加速度计和gps消除陀螺仪的误差.

2.1 两个向量的误差($\vec{A}$, $\vec{B}$)，用叉乘表示.

$$E_{error} = \vec{A} \times \vec{B} = |\vec{A}| \cdot |\vec{B}| \cdot \sin\theta .$$

如果$\vec{A}$, $\vec{B}$都规一化后(向量模为1)，θ越小，表明两个向量越相似，误差越小.

2.2 漂移误差修正原理

首先修正偏航角. 此处主要为俯仰角和翻滚角修正.

$$g_e - a_e(t) = R(t) \cdot A_b(t) \quad ①$$

地球坐标系下加速度 = 旋转矩阵 乘以 机身坐标系加速度

$$A_b(t) = g_b(t) - a_b(t) \quad ②$$

机身加速度 = 机身重力加速度与机动加速度之差

差 → 即加速度计测得的值.

根据①式.有 $\int_{t_1}^{t_2} R(t)A_b(t)dt = (t_2-t_1)g_e - (t_2-t_1)a_e(t)$

即 $\int_{t_1}^{t_2} R(t)A_b(t)dt = (t_2-t_1)g_e - (V_{t_2}-V_{t_1})$ ③

③式左边为测得的加速度与旋转矩阵乘积后积分;(由于矩阵不够精确 故有误差)
右边为重力加速度(常量).与gps测得的速度之差.(比较准确)
故③式不是数值计算上严格相等.误差产生.

$$E_e(\text{地球坐标系}) = \frac{\int_{t_1}^{t_2} R(t)A_b(t)dt \times [(t_2-t_1)g_e - (V_{t_2}-V_{t_1})]}{\left|\int_{t_1}^{t_2} R(t)A_b(t)dt\right| \cdot \left|(t_2-t_1)g_e - (V_{t_2}-V_{t_1})\right|}$$

即左.右两边规一化后.叉乘结果为误差向量.

$\dot{R}\ E_b(\text{机身坐标系}) = E_e$. 故 $\underline{E_b = R^T E_e}$ ④ (R^T 为矩阵的转置.正交矩阵有 $R^T = R^{-1}$).

2.3 代码中的体现 drift_correction( ) 函数中.

采用③式的变形为

$$\frac{1}{(t_2-t_1)g_e} \cdot R(t) \cdot A_b(t) \cdot \Delta t = \frac{1}{(t_2-t_1)g_e}(V_{t_2}-V_{t_1}) - \underset{\text{即}(0,\,0,\,-1)}{1} \quad ⑤$$

2.4 在地球坐标系中计算.然后再转换到机身坐标系原因.
gps测得的数据更新慢(1秒5-6次),而加速度计数据更新快(1秒7000-8000次).
故③式左侧适合积分.右侧不适合。在机身坐标系下运算.得对③式的右侧积分.

2.5 ④式的具体化.代码中体现.

$$E_b = \begin{pmatrix} a_x & b_x & c_x \\ a_y & b_y & c_y \\ a_z & b_z & c_z \end{pmatrix}\begin{pmatrix} e_{ex} \\ e_{ey} \\ e_{ez} \end{pmatrix} = \begin{bmatrix} a_x e_{ex} + b_x e_{ey} + c_x e_{ez} \\ a_y e_{ex} + b_y e_{ey} + c_y e_{ez} \\ a_z e_{ex} + b_y e_{ey} + c_y e_{ez} \end{bmatrix}$$

2.6 用pid调整(主要是pI)

I 积分调整. 每5S调整一次. 变量 $\_omga\_I = \underbrace{E_b}_{\text{机身坐标系误差向量}} \cdot \underbrace{\Delta t}_{\text{积分时间}} \cdot \underbrace{k_i}_{\text{积分常数,为经验值}}$

P 比例调整, 每次gps更新数据.才调整.(1s,5-6次)

$\_omga\_P = \underbrace{E_b}_{\text{机身坐标系误差向量}} \cdot \underbrace{k_p}_{\text{比例调整常数.根据经验获得}}$

_omga_P 和 _omga_I. 用于修正 _omga_(陀螺仪数据), 相加即可作为下一次旋转矩阵更新的输入

$$\underbrace{R_{matrix}}_{\text{每秒7000-8000次}} = R_{matrix}[(\underbrace{\_omga\_}_{\text{每秒7000-8000次更新}} + \underbrace{\_omga\_I}_{\text{5秒更新一次}} + \underbrace{\_omga\_P}_{\text{1秒5-6次}})\Delta t]$$

另外 _omga_P 和 _omga_I 还可以用apm的地面站软件连接APM进行远程修改.

4. 由 DCM 矩阵求俯仰，翻滚，偏航角

```
AP_AHRS_DCM::euler_angles(void)
{
    _body_dcm_matrix = _dcm_matrix;
    _body_dcm_matrix.rotateXYinv(_trim);
    _body_dcm_matrix.to_euler(&roll, &pitch, &yaw);

    update_cd_values();
}
```

对应的数学公式为：

3. 用旋转矩阵求欧拉角.

空间旋转.

1.绕z轴转ψ(yaw,偏航). 2.绕y轴θ(pitch 俯仰) 3.绕x轴φ(roll 翻滚)

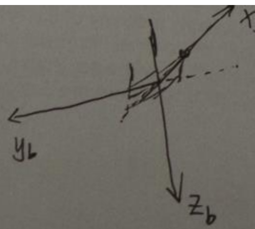
$$\begin{pmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix} \quad \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{pmatrix}$$

①→②→③

$$\theta_e = R\theta_b = R_z \cdot R_y \cdot R_x \theta_b = \begin{pmatrix} \cos\theta\cos\psi, & -\sin\psi, & \cos\psi\sin\theta \\ \sin\psi\cos\theta, & \cos\psi, & \sin\psi\sin\theta \\ -\sin\theta, & 0, & \cos\theta \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{pmatrix}$$

$\begin{pmatrix} e: \text{earth} \\ b: \text{body} \end{pmatrix}$

$$= \begin{pmatrix} \cos\theta\cos\psi, & \sin\phi\sin\theta\cos\psi - \sin\psi\cos\phi, & \cos\psi\cos\phi\sin\theta + \sin\psi\sin\phi \\ \cos\theta\sin\psi, & \cancel{\sin\theta\sin\phi}\ \sin\theta\sin\psi\cos\phi + \cos\phi\cos\psi, & \sin\psi\sin\theta\cos\phi - \sin\phi\cos\psi \\ -\sin\theta, & \cos\theta\sin\psi, & \cos\theta\cos\phi \end{pmatrix}$$



坐标系：机头面向自己,坐左右为机翼.

$$R\begin{pmatrix} R_{xx}, R_{xy}, R_{xz} \\ R_{yx}, R_{yy}, R_{yz} \\ R_{zx}, R_{zy}, R_{zz} \end{pmatrix} \rightarrow \text{欧拉角}.$$

$$\begin{cases} \theta = -\arcsin R_{zx} \\ \phi = \arctan2(R_{zy}, R_{zz}) \\ \psi = \arctan2(R_{yx}, R_{xx}) \end{cases}$$

欧拉角→R,直接套矩阵即可.