郑州大学毕业设计

算法说明和源程序

题　　目	基于UWB和惯性导航融合的室内定位方法研究
院　　系	电气工程学院
专　　业	轨道交通信号与控制
班　　级	2015-2
学生姓名	卢鸿浩
学　　号	20150270228
指导教师	辛健斌　职称　副教授

2019年 06 月 10 日

目录

一 融合算法结构

1.整体结构

判断接收到的是 IMU/UWB 数据？

是 IMU

判断是否小于 IMU 初始化步数？

是：进行初始化，陀螺仪求平均

否：判断是否静止和是否转弯；

姿态解算；

判断是否加速；

选用双互补滤波器中其中一个给出的加速度值；

速度位置解算

是 UWB

判断基站数目合适？

是：是否静止？

是：启动对 UWB 定位固定点情况滤波。

判断是否小于 IMU 初始化步数？

是：INS 从 UWB 获取初始位置（IMU 初始化步数要足够长，使 UWB 能提供初始位置）

否：判断 UWB 数据好坏？

好：进行融合滤波

结果去修正 INS 的速度和位置；

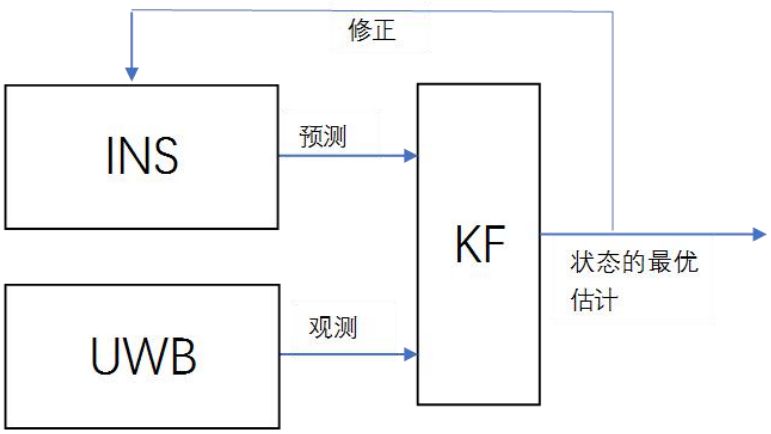
机器人方向修正，除去非前进方向的速度；

坏：剔除

显示轨迹

都不是：本轮数据判断

2. 融合滤波结构



松散融合的执行方式

融合定位的 KF 算法

Input: x_0, P_0, y_k,R,Q

for **k=1:** ∞

1 $\hat{x}_{k|k-1} = A\hat{x}_k + Bu$

2 $P_{k|k-1} = AP_{k-1}A^T + Q$

3 $K_k = P_{k|k-1}H(HP_{k|k-1}H^T + R)^{-1}$

4 $\hat{x}_k = \hat{x}_{k|k-1} + K_k(y_k - \hat{x}_{k|k-1})$

5 $P_k = P_{k|k-1} - K_k P_{k|k-1}$

end

其中：

$$x = \begin{bmatrix} P_x \\ P_y \\ V_x \\ V_y \end{bmatrix}$$

$$u = \begin{bmatrix} a_x \\ a_y \end{bmatrix}$$

$$A = \begin{bmatrix} 1 & 0 & T & 0 \\ 0 & 1 & 0 & T \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$B = \begin{bmatrix} T^2 & 0 \\ 0 & T^2 \\ T & 0 \\ 0 & T \end{bmatrix}$$

$$y = \begin{bmatrix} P_{x_{UWB}} \\ P_{y_{UWB}} \end{bmatrix}$$

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

$$\begin{cases} Q_f = diag([0.1 \quad 0.1 \quad 0.05 \quad 0.05]) \\ R_f = diag([3 \quad 3 \quad 3]) \end{cases}$$

$$\begin{cases} Q = diag([10 \quad 10]) \\ R = diag([0 \quad 0]) \end{cases}$$

二 源程序

Fusion_with_KF（主程序）

```
% INS 和 UWB 融合
% 融合采用直接的卡尔曼滤波的方法
% 此程序是线下开发程序，导入采集的数据后进行算法开发验证；稍加改动可以改为线上的运行的程序

% 从数据中导入 w_record_all，a_record_all,uwb_record，g，a_sen，a_zero_drift;w_cor(角速度零点漂移),N_g_cor(校正步数)
% g_local = 9.7966;
% a_record_all 记录[ a; j; k; T; T2; t;]； uwb_record 记录[k; j; T; T2; Y(:, k); 4; d'; t;]

   a_scale = [0.9997 0.9987 0.9922]'; a_offset = [0.113 0.156 0.345]';   % 10 号 的灵敏度漂移和零点漂移
   a_scale = [ 1.0066 0.9943    0.9837]'; a_offset = [0.156893 -0.098563 0.562545]'; % 5 号 的漂移

% 融合滤波
global X1_fu
global Y_fu    % uwb 的观测误差
global F_fu
global H_fu
global P_fu
global Q_fu
global R_fu    % uwb 的观测噪声
```

```
global I_fu

global DCMg

global q    % 四元数 q = [q0 q1 q2 q3]

global N_g_cor % 校正步数

global roll    % 横滚角

global pitch    % 俯仰角

global yaw
global integral_e_a
global yaw_1
global error_w_record

global a_ave    % 加速度，检测静止

global state
global Kp1
global Kp2
Kp1 = 2;
Kp2 = 0.2;

state =0;

error_w_record = [0 0 0]';
state_count = 0;

global DCMg1

global q1    % 四元数 q = [q0 q1 q2 q3]

global DCMg2
global q2

q = [1 0 0 0]';
integral_e_a = [0 0 0]';
integral_e_a_record = [0 0 0]';
q_record_1 = [1 0 0 0]';
a_T_record_1 = [0 0 0]';
a_T_record_2 = [0 0 0]';

h = 1;    % INS 计数

c = 2;    % UWB 计数
```

```
velocity_ins_1 = [0 0 0]';
velocity_ins_2 = [0 0 0]';
velocity_ins_o = [0 0 0]';
position_ins_1 = [0 0 0]';
position_ins_2 = [0 0 0]';
position_ins_o = [0 0 0]';
velocity_ins_b = [0 0 0]';
position_ins_b = [0 0 0]';
velocity_ins_f = [0 0 0]';
position_ins_f = [0 0 0]';

a_record = a_record_all(1:3, :);
w_record = w_record_all(1:3, :);

w3_filter(1) = w_record(3, 1);
yaw_1(1) = 0;
yaw_record = 0;
det_acc = 0;
eula_record = [0 0 0]';

u1 =5; u2 = 6;
Y = uwb_record(u1:u2, :);
record_a2(1) = 0;

P_s = [100 0;0 100];
X_s(:, 1:2) = uwb_record(u1:u2, 1:2);
X1_s = X_s;

uwb_s = 0;    % 记录开始静止时，uwb 定位数据点的个数


Initial_KF;
Initial_kf_fuse;

count_a = 0;
detect_a = 0;
yaw1_t = 0;
X_fu_record = [0 0 0 0 0 0 0]';
yaw1 = 0;

for ii = 1:3    % 预设了阶数为 2 阶，200 是采样频率，2 是截止频率

    a_record_lf(ii, :) = butter1array(a_record(ii,:), 200, 2);    % a1 是算加速度，2hz

    a_record_lf2(ii, :) = butter1array(a_record(ii,:), 200, 2);    % a2 算姿态
```

```
end
t = 1;
while t < length(a_record(1,:)-1)
    t = t + 1;
    T = a_record_all(6, t);

    a1 = a_record_lf(:, t);
    a2 = a_record_lf2(:,t);

%       a1 = a_record_lf(:, t);
%       a2 = a_record_lf2(:,t);
    if t>115
    w = w_record(:, t);
    else
        w= [0 0 0]';
    end

    a1 = a1 - a_offset;
%       a1 = a1 .* a_scale;
    a2 = a2 - a_offset;
%       a2 = a2 .* a_scale;
        record_a2 = [record_a2 (a2(1)^2+ a2(2)^2+ a2(3)^2)^0.5];

    % ****初始化,校准
    if t <= N_g_cor
        a_T_1 = [0 0 0]';
        for i = 1:3
            a_initial(i) = mean(a_record(i, 2:N_g_cor));
        end
%           Quaternion_with_gyr( T, a_initial, w, t);
            Quaternion_with_PID1( T, a_initial, w, t);
            Quaternion_with_PID2( T, a_initial, w, t);
%           q_record_1 = [q_record_1 q];
        a_ave = a_record(1:2, t);
        yaw1(t) =    0;
    else
        h = h + 1;
        w = w - w_cor;
        if abs(w) < 0.001
            w = [0 0 0]';
        end

        % 门限值静止检测法
```

```
        if t > (N_g_cor + 20)    %  检测静止
            if ~mod(t,50)
                for i = 1: 2
                    a_ave(i) = mean( a_record(i, (t-20):t) );    %  取平均
                end
            end
            a_stand_threshold = 0.15;
            if (a_ave(1) - a_stand_threshold)<a_record(1, t) & (a_ave(1) +
a_stand_threshold)>a_record(1, t)...
                    & (a_ave(2) - a_stand_threshold)<a_record(2, t) & (a_ave(2) +
a_stand_threshold)>a_record(2, t)
                state_count = state_count + 1;
            else
                state_count = 0;
            end
            if state_count > 19
                state = 0;
            else
                state = 1;
                Kp1 = 2;
                Kp2 = 0.2;
            end
        end

        if w_record(3, t) > 0.4    %  检测到转弯
            Kp1 = 0;
            Kp2 = 0;
        end

        Quaternion_with_PID1(T, a1, w, t);
        Quaternion_with_PID2(T, a1, w, t);



        yaw1(t) = yaw1(t-1) + w(3)*T; %  航向角,yaw
        yaw1_t = yaw1(t);

        eula = [pitch    roll yaw]';
        eula_record = [eula_record eula];
```

```
% 加速度转化到全局坐标系下，并除去重力
    a_T_1 = DCMg1 * (a1) - g;
    a_T_2 = DCMg2 * (a1) - g;

    a_T_11 = (DCMg1 * (a2) - g);
    a_T_22 = (DCMg2 * (a2) - g);
    a_T_record_1 = [a_T_record_1 a_T_11];    % record v or a_T
    a_T_record_2 = [a_T_record_2 a_T_22];    % record v or a_T

    if (abs(a_T_22(1)) < 0.1) && (abs(a_T_22(2)) < 0.1)
        a_T = a_T_1;
    else
        a_T = a_T_2;
        if abs(a_T_22(1)) > 0.15
            detect_a = 1;
        elseif abs(a_T_22(2)) > 0.15
            detect_a = 2;
        end
    end
    a_T_record = [a_T_record a_T];


    % 得到速度<-加速度积分
    velocity_ins_1(:,h) = velocity_ins_1(:,h-1) + a_T_1 * T;
    velocity_ins_2(:,h) = velocity_ins_2(:,h-1) + a_T_2 * T;
    velocity_ins_b(:,h) = velocity_ins_b(:,h-1) + a_T * T;
    velocity_ins_f(:,h) = velocity_ins_f(:,h-1) + a_T * T;
    if detect_a == 1
        if   ( a_T_record_2(1,h) * a_T_record_2(1,h-1) ) < 0   % 检测第一次过零点
            count_a = 700;
            detect_a = 0;
            velocity_ins_b(:,h-1) = velocity_ins_b(:,h-50);
            velocity_ins_f(:,h) = velocity_ins_b(:,h);
        end
    elseif detect_a == 2
        if   ( a_T_record_2(2,h) * a_T_record_2(2,h-1) ) < 0   % 检测第一次过零点
            count_a =700;
            detect_a = 0;
            velocity_ins_b(:,h-1) = velocity_ins_b(:,h-50);
            velocity_ins_f(:,h) = velocity_ins_b(:,h) * 1.2;
        end
```

```
        end
        if count_a > 0
            count_a = count_a-1;
            velocity_ins_b(:,h) = velocity_ins_b(:,h-1);
        end

        velocity_ins_o(1,h) = cos(yaw1_t) * ( velocity_ins_f(1,h)*cos(yaw1_t) +
velocity_ins_f(2,h)*sin(yaw1_t));
        velocity_ins_o(2,h) = sin(yaw1_t) * ( velocity_ins_f(1,h)*cos(yaw1_t) +
velocity_ins_f(2,h)*sin(yaw1_t));
        % %   是否加入方位修正
        if t < 20000
            velocity_ins_f(: ,h) = velocity_ins_o(:,h);
        end

        if state == 0
            velocity_ins_1(:,h) = [0 0 0]';
            velocity_ins_2(:,h) = [0 0 0]';
            velocity_ins_b(:,h) = [0 0 0]';
            velocity_ins_o(:,h) = [0 0 0]';
            velocity_ins_f(:,h) = [0 0 0]';
        end
        % 得到位移<-速度积分
        displacement_T_1 = velocity_ins_1(:,h) * T;
        displacement_T_2 = velocity_ins_2(:,h) * T;
        displacement_T_b = velocity_ins_b(:,h) * T;
        displacement_T_o = velocity_ins_o(:,h) * T;
        displacement_T_f = velocity_ins_f(:,h) * T;

        % 位置
        position_ins_1(:,h) = position_ins_1(:,h-1) + displacement_T_1;
        position_ins_2(:,h) = position_ins_2(:,h-1) + displacement_T_2;
        position_ins_b(:,h) = position_ins_b(:,h-1) + displacement_T_b;
        position_ins_o(:,h) = position_ins_o(:,h-1) + displacement_T_o;
        position_ins_f(:,h) = position_ins_f(:,h-1) + displacement_T_f;

        % ****INS 积分，结束

%           %   法一   % 预测 时间更新
```

```
%               F_fu1 = [eye(3) diag([T T T]); zeros(3) eye(3)];
%               P_fu1 = F_fu1 * P_fu1 * F_fu1' + Q_fu1;    % 过程噪声更新


%           %   法二   % 预测 时间更新
%               F_fu = [eye(3) diag([T T T]); zeros(3) eye(3)];
%               P_fu = F_fu * P_fu * F_fu' + Q_fu;    % 过程噪声更新

            % [P V]' = [1 T;0 1] * [P V]' + [T^2 T] * a;

        end

% UWB*****************
    if ( uwb_record(12, c) == a_record_all(8, t) ) && (c < length(uwb_record(1,:)))
            % % uwb_record 记录[k; j; T; T2; Y(:, k); 4; d'; t;];    a_record_all 记录[ a; j; k; T; T2;
t;];

            %   检查 UWB 和 INS 采样点在时间上重合


            % T = 0.005;    % 时间间隔为 0.556s
            T = uwb_record(3, c);
            Y(:,c) =    uwb_record(u1:u2, c);    % Y 为观察值

            position_error = Y(:, c) - position_ins_f(1:2, h);

            if ( (t <= N_g_cor)    || (state == 0) )

              % 静止时，速度为零，获取位置

                 % 预测 时间更新

                F_s = [1 0; 0 1]; H_s = [1 0;0 1]; R_s = [10 0;0 10]; Q_s = [0 0;0 0]; I_s = [1 0;0
1]; % R_s = [5e-5 0;0 5e-5];
                X1_s(:,c) = F_s * X_s(:,c-1);
                P_s = F_s * P_s * F_s' + Q_s; % Q 为 预测噪声协方差

                % 修正 测量更新
```

```
Y_s(:,c) = Y(:,c);    % Y 为观察值

K_s = (P_s*H_s') * pinv((H_s*P_s*H_s') + R_s);       % R 为 观测噪声协方差，
```

源于传感器的测量（测量精度及热噪声等的影响）； K 为 卡尔曼增益

```
X_s(:,c) = X1_s(:,c) + K_s * ( Y_s(:,c) - H_s * X1_s(:,c) );
P_s = (I_s - K_s*H_s) * P_s;

if uwb_s < 1
    X_s(:, c) = uwb_record(u1:u2, c);
    X1_s(:, c) = X_s(:, c);
end

uwb_s = uwb_s + 1;    % 记录开始静止时，uwb 定位数据点的个数


position_ins_f(1:2, h) = X_s(:,c);    % 初始的状态值

position_ins_f(3,h) = 0;
velocity_ins_f(:,h) = [0 0 0]';

X_fu(1:3) = position_ins_f(:, h);    % 位置

X_fu(4:6) = velocity_ins_f(:, h);    % 速度

Y_fu(:,c) = [Y(:,c); 0;];
%                 position_ins_f(:,h) = Y_fu(:,c);
```

```
elseif    ( position_error(1)^2 + position_error(2)^2 < (1^2)) % 检测 uwb 定位不稳定的
```

情况

```
% 静止滤波    记录 c

X_s(:, c) = uwb_record(u1:u2, c);
X1_s(:, c) = X_s(:, c);
uwb_s = 0;


% 法二----------------------------------------------------------

% 直接把 UWB 定位数据来作为观测
```

```
% 融合滤波(f:fuse) F_fu = [eye(3) diag([T T T]); zeros(3) eye(3)];  H_fu =
[eye(3) zeros(3)];
% 观测 时间更新
Y_fu(:,c) = [Y(:,c); 0];

T_f = T;
% 预测 时间更新
F_fu = [eye(3) diag([T T T]); zeros(3) eye(3)];
P_fu = F_fu * P_fu * F_fu' + Q_fu;  % 过程噪声更新

% 修正 测量更新
X1_fu(:,c) = [position_ins_f(:,h); velocity_ins_f(:,h)];
K_fu = P_fu * H_fu' * pinv(H_fu * P_fu * H_fu' + R_fu);  % 卡尔曼增益

X_fu = X1_fu(:, c) + K_fu * ( Y_fu(:,c) - H_fu * X1_fu(:,c) );  % 状态估计

P_fu = ( I_fu - K_fu * H_fu ) * P_fu;  % 协方差矩阵更新
X_fu_record = [X_fu_record [X_fu; c]];

% 融合滤波结果送回，校正 INS 的 速度，位置

position_ins_f(:,h) = X_fu(1:3);  % 位置

velocity_ins_f(:,h) = X_fu(4:6);  % 速度
%-------------------------------------------------------------
else  % 照应-检测 uwb 定位不稳定的情况

Y_fu(:,c) = [Y(:,c); 0];
X_fu(1:3) = position_ins_f(:, h);  % 位置

X_fu(4:6) = velocity_ins_f(:, h);  % 速度
X_fu_record = [X_fu_record [X_fu; c]];

% 静止滤波  记录 c
X_s(:, c) = uwb_record(u1:u2, c);
```

```
            X1_s(:, c) = X_s(:, c);
            uwb_s = 0;
        end

        y2 = [y2 [t state]'];
        Display_trajectory(uwb_record(u1:u2, c),2);
        Display_trajectory(position_ins_f(1:2,h),1);
        c = c + 1;
    end
     % *************************

     % *************************
End
```

```
function Quaternion_with_PID1(T, a, w, t)

% 传入时间间隔、加速度、角速度、时间段计数

global DCMg1
global q1    % 四元数 q = [q0 q1 q2 q3]
global N_g_cor % 校正步数
global roll    % 横滚角
global pitch    % 俯仰角
global yaw    % 翻滚角
global integral_e_a
global error_w_record
global error_a_record
global state
global Kp1

Ki = 0.01;
Kd = 0.2;

halfT = 0.5 * T;    % 时间间隔

if state==0
    Kp1=4;
end

% 初始化
if t <= N_g_cor
    if t < N_g_cor
        q1 = [1 0 0 0]';
%         q1 = [0.9997 0 0 -0.0262]';
    end
else
    norm_a = (a(1)^2 + a(2)^2 + a(3)^2)^0.5;
    a = a / norm_a;    % 单位化
```

```
    % 理论重力加速度，由四元数表示的 DCMb*[0 0 1]'得到，即 DCMb 的第三列
    v(1) = 2 * (q1(2)*q1(4) - q1(1)*q1(3));
    v(2) = 2 * (q1(1)*q1(2) + q1(3)*q1(4));
    v(3) = q1(1)*q1(1) - q1(2)*q1(2) - q1(3)*q1(3) + q1(4)*q1(4);
    v = v';
    % 构建 PI 控制器，求误差补偿
%       error_a = a - v;
    error_a = cross(a, v);  % 误差 <- 对实际 g 与理论 g 求叉乘
    error_a_record(:, t) = error_a;
    integral_e_a = integral_e_a + Ki * error_a * T;  % 积分项
    error_w = Kp1 * error_a + integral_e_a + Kd * (error_a - error_a_record(:,t-1));  % 补偿值
<- 比例项 + 积分项
%       error_w = Kp * error_a + integral_e_a;  % 补偿值 <- 比例项 + 积分项
    error_w_record = [error_w_record error_w];
    w = w + error_w;
    w = w;
    % 四元数微分方程求解(法)，更新----------
    q1(1) = q1(1) + (-q1(2)*w(1) - q1(3)*w(2) - q1(4)*w(3)) * halfT;
    q1(2) = q1(2) + (q1(1)*w(1) + q1(3)*w(3) - q1(4)*w(2)) * halfT;
    q1(3) = q1(3) + (q1(1)*w(2) - q1(2)*w(3) + q1(4)*w(1)) * halfT;
    q1(4) = q1(4) + (q1(1)*w(3) + q1(2)*w(2) - q1(3)*w(1)) * halfT;
    % 四元数单位化
    norm_q = (q1(1)^2 + q1(2)^2 + q1(3)^2 + q1(4)^2)^0.5;
    q1 = q1 / norm_q;
end
% if ~mod(t-1,70)
    % 由四元数求 DCMb
    % 全局转到体坐标系
    DCMb(1,1) = q1(1)*q1(1) + q1(2)*q1(2) - q1(3)*q1(3) - q1(4)*q1(4);
    DCMb(2,1) = 2 * (q1(2)*q1(3) - q1(1)*q1(4));
```

```
        DCMb(3,1) = 2 * (q1(2)*q1(4) + q1(1)*q1(3));
        DCMb(1,2) = 2 * (q1(2)*q1(3) + q1(1)*q1(4));
        DCMb(2,2) = q1(1)*q1(1) - q1(2)*q1(2) + q1(3)*q1(3) - q1(4)*q1(4);
        DCMb(3,2) = 2 * (q1(4)*q1(3) - q1(1)*q1(2));
        DCMb(1,3) = 2 * (q1(2)*q1(4) - q1(1)*q1(3));
        DCMb(2,3) = 2 * (q1(4)*q1(3) + q1(1)*q1(2));
        DCMb(3,3) = q1(1)*q1(1) - q1(2)*q1(2) - q1(3)*q1(3) + q1(4)*q1(4);
        DCMg1 = DCMb';

%      if ~mod(t-1,30)
%            Display_axis(DCMg);
%      end

    roll = atan( DCMg1(3,2)/DCMg1(3,3) );
    pitch = asin( -DCMg1(3,1) );
    yaw = atan( DCMg1(2,1)/DCMg1(1,1) );


% % %  由方程：四元数表示的 DCM =  欧拉角表示的 DCM，来反解出欧拉角

% roll = atan2( 2*(q(1)*q(2) + q(3)*q(4)), (1 - 2*q(2)*q(2) - 2*q(3)*q(3)) )* 57.3; %  横滚角,roll

% pitch = asin( -2*q(2)*q(4) + 2*q(1)*q(3) ) * 57.3; %  俯仰角,pitch

% yaw = atan2( 2*(q(2)*q(3) + q(1)*q(4)), (q(1)*q(1)+q(2)*q(2)-q(3)*q(3)-q(4)*q(4)) ) * 57.3; %
航向角,yaw


end
```

```
function Quaternion_with_PID2( T, a, w, t)

% 传入时间间隔、加速度、角速度、时间段计数

global DCMg2
global q2    % 四元数 q = [q0 q1 q2 q3]
global N_g_cor % 校正步数
global roll    % 横滚角
global pitch    % 俯仰角
global yaw    % 翻滚角
global integral_e_a
global error_w_record
global error_a_record
global state
global Kp2

 Ki = 0;    % 积分参数，Ki 太大的话会产生很大的震荡
 Kd = 0.05;

halfT = 0.5 * T;    % 时间间隔

if state==0
    Kp2=4;
end

% 初始化
if t <= N_g_cor
    if t < N_g_cor
        q2 = [1 0 0 0]';
%         q2 = [0.9997 0 0 -0.0262]';
    end
else
    norm_a = (a(1)^2 + a(2)^2 + a(3)^2)^0.5;
    a = a / norm_a;    % 单位化
```

```
    % 理论重力加速度，由四元数表示的 DCMb*[0 0 1]'得到，即 DCMb 的第三列
    v(1) = 2 * (q2(2)*q2(4) - q2(1)*q2(3));
    v(2) = 2 * (q2(1)*q2(2) + q2(3)*q2(4));
    v(3) = q2(1)*q2(1) - q2(2)*q2(2) - q2(3)*q2(3) + q2(4)*q2(4);
    v = v';
    % 构建 PI 控制器，求误差补偿
%       error_a = a - v;
    error_a = cross(a, v);  % 误差 <- 对实际 g 与理论 g 求叉乘
    error_a_record(:, t) = error_a;
    integral_e_a = integral_e_a + Ki * error_a * T;  % 积分项
    error_w = Kp2 * error_a + integral_e_a + Kd * (error_a - error_a_record(:,t-1));  % 补偿值
<- 比例项 + 积分项
%       error_w = Kp * error_a + integral_e_a;  % 补偿值 <- 比例项 + 积分项
    error_w_record = [error_w_record error_w];
    w = w + error_w;
    w = w;
    % 四元数微分方程求解(法)，更新----------
    q2(1) = q2(1) + (-q2(2)*w(1) - q2(3)*w(2) - q2(4)*w(3)) * halfT;
    q2(2) = q2(2) + (q2(1)*w(1) + q2(3)*w(3) - q2(4)*w(2)) * halfT;
    q2(3) = q2(3) + (q2(1)*w(2) - q2(2)*w(3) + q2(4)*w(1)) * halfT;
    q2(4) = q2(4) + (q2(1)*w(3) + q2(2)*w(2) - q2(3)*w(1)) * halfT;
    % 四元数单位化
    norm_q = (q2(1)^2 + q2(2)^2 + q2(3)^2 + q2(4)^2)^0.5;
    q2 = q2 / norm_q;
end
% if ~mod(t-1,70)
    % 由四元数求 DCMb
    % 全局转到体坐标系
    DCMb(1,1) = q2(1)*q2(1) + q2(2)*q2(2) - q2(3)*q2(3) - q2(4)*q2(4);
    DCMb(2,1) = 2 * (q2(2)*q2(3) - q2(1)*q2(4));
```

```
        DCMb(3,1) = 2 * (q2(2)*q2(4) + q2(1)*q2(3));
        DCMb(1,2) = 2 * (q2(2)*q2(3) + q2(1)*q2(4));
        DCMb(2,2) = q2(1)*q2(1) - q2(2)*q2(2) + q2(3)*q2(3) - q2(4)*q2(4);
        DCMb(3,2) = 2 * (q2(4)*q2(3) - q2(1)*q2(2));
        DCMb(1,3) = 2 * (q2(2)*q2(4) - q2(1)*q2(3));
        DCMb(2,3) = 2 * (q2(4)*q2(3) + q2(1)*q2(2));
        DCMb(3,3) = q2(1)*q2(1) - q2(2)*q2(2) - q2(3)*q2(3) + q2(4)*q2(4);
        DCMg2 = DCMb';

%       if ~mod(t-1,30)
%             Display_axis(DCMg);
%       end

%       roll = atan( DCMg(3,2)/DCMg(3,3) );
%       pitch = asin( -DCMg(3,1) );
%       yaw = atan( DCMg(2,1)/DCMg(1,1) );

% % % 由方程：四元数表示的 DCM = 欧拉角表示的 DCM，来反解出欧拉角

% roll = atan2( 2*(q(1)*q(2) + q(3)*q(4)), (1 - 2*q(2)*q(2) - 2*q(3)*q(3)) )* 57.3; % 横滚角,roll

% pitch = asin( -2*q(2)*q(4) + 2*q(1)*q(3) ) * 57.3; % 俯仰角,pitch

% yaw = atan2( 2*(q(2)*q(3) + q(1)*q(4)), (q(1)*q(1)+q(2)*q(2)-q(3)*q(3)-q(4)*q(4)) ) * 57.3; %
航向角,yaw


end
```

```
function Initial_KF()


% 对 uwb 的数据进行先进行一次滤波
global X
global X1
global Y
global F
global H
global P
global Q
global R
global I


% UWB 滤波
T = 0.6;
F = [1 0 T 0;0 1 0 T;0 0 1 0;0 0 0 1];  % t-1 时刻到 t 时刻的状态 X 的转移矩阵，g_t 是时间间
隔
H = [1 0 0 0;0 1 0 0];      % H 为测量矩阵
P = diag([10 10 10 10]);
% Q = diag([1e-6 1e-7 1e-7 1e-6]);  % Q 为过程噪声
Q = diag([1e-3 1e-3 1e-3 1e-3]);  % Q 为过程噪声
% R = [1e-6 0;0 1e-6];  % R 为观测噪声
R = diag([15e-1 15e-1]);  % R 为观测噪声
% R = diag([0.1e-1 0.1e-1]);  % R 为观测噪声
I = eye(4);
X(:,1) = [0 0 0 0]';
Y(:,1) = [0 0]';

end
```

```
function Initial_kf_fuse()


% 融合滤波

global X1_fu

global Y_fu    % uwb 的观测误差

global F_fu
global H_fu
global P_fu
global Q_fu

global R_fu    % uwb 的观测噪声

global I_fu


% 融合滤波

T_uwb = 0;

% F_fu    % t-1 时刻到 t 时刻的状态 X 的转移矩阵，g_t 是时间间隔

H_fu = [eye(3) zeros(3)];    % 测量转换矩阵

P_fu = eye(6);    % 协方差矩阵的初始化


% UWB 频率高

Q_fu = diag([0.1 0.1 0.01    5e-2 5e-2 5e-2]);

R_fu = diag([3 3 3]);    % R 为观测噪声


I_fu = eye(6);
clear X_fu
clear Y_fu

X_fu(:,1) = [0 0 0 0 0 0]';    % 状态向量,位置误差，速度误差，角度误差

Y_fu(:,1) = [0 0 0]';
```

```
function [ y ] = butter1array( x, fs, fc)

%  二阶巴特沃斯滤波器，输入一组数组 x，输出滤波后的数组 y

% fs 为采样频率，fc 为截止频率

%  wc 是截止频率，T 是采样周期


size_x = size(x);

% order = 1;  % 滤波器阶数

order = 2;  % 滤波器阶数

%  order = 3;  % 滤波器阶数

[a1,b1] = design_butterws_filter(2, order, fs, fc);
for n = 1 : size_x(2)
    if n <= order
        y(n) = x(n);
    else

%           y(n) = a1(1) * x(n) + a1(2) * x(n-1) - b1(2) * y(n-1);  % 1 阶

        y(n) = a1(1) * x(n) + a1(2) * x(n-1) + a1(3) * x(n-2) - b1(2) * y(n-1) - b1(3) * y(n-2); %
2 阶

%           y(n) = a1(1) * x(n) + a1(2) * x(n-1) + a1(3) * x(n-2) + a1(4) * x(n-3) - b1(2) * y(n-1) -
b1(3) * y(n-2) - b1(4) * y(n-3); % 3 阶    % 3 阶

    end
end
end
```

```
function [num,den] = design_butterws_filter( which , n, fs, fc )

%   设计 butterworth filter，which 选择离散化方法
%   n: order of the filter
%   fs: simple frequency
%   fc: boundary frequency

if which == 1

% mathod 1: impluse invariance transformation
[b,a] = butter(n, 2*pi*fc,'s');   % 模拟滤波器分子，分母系数向量

[num1,den1] = impinvar(b,a,fs);   % 脉冲相应不变法求得数字滤波器的分子分母向量
num = num1;
den = den1;
else

% mathod 2: bilinear transformation
[b,a] = butter(n, 2*fs*tan(2*pi*fc/(2*fs)), 's');   % 预扭曲频率

[num2,den2] = bilinear(b,a,fs);   % 双线性变换法求得数字滤波器的分子分母向量
num = num2;
den = den2;
end

end
```