quaternion product

$$\boxed{C(r) = C(q)C(p)}$$

Let p and q are quaternions.

$$p = \begin{bmatrix} p_0 & \bar{p} \end{bmatrix}, \quad q = \begin{bmatrix} q_0 & \bar{q} \end{bmatrix}$$

where

$$\bar{p} = \begin{bmatrix} p_1 & p_2 & p_3 \end{bmatrix}, \quad \bar{q} = \begin{bmatrix} q_1 & q_2 & q_3 \end{bmatrix}$$

- quaternion product

$$r = p \otimes q = p_0 q_0 - \bar{p} \cdot \bar{q} + p_0 \bar{q} + q_0 \bar{p} + \bar{p} \times \bar{q}$$

- matrix form: $r = p \otimes q$

$$\begin{bmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \end{bmatrix} = \begin{bmatrix} p_0 & -p_1 & -p_2 & -p_3 \\ p_1 & p_0 & -p_3 & p_2 \\ p_2 & p_3 & p_0 & -p_1 \\ p_3 & -p_2 & p_1 & p_0 \end{bmatrix} \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & q_3 & -q_2 \\ q_2 & -q_3 & q_0 & q_1 \\ q_3 & q_2 & -q_1 & q_0 \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix}$$

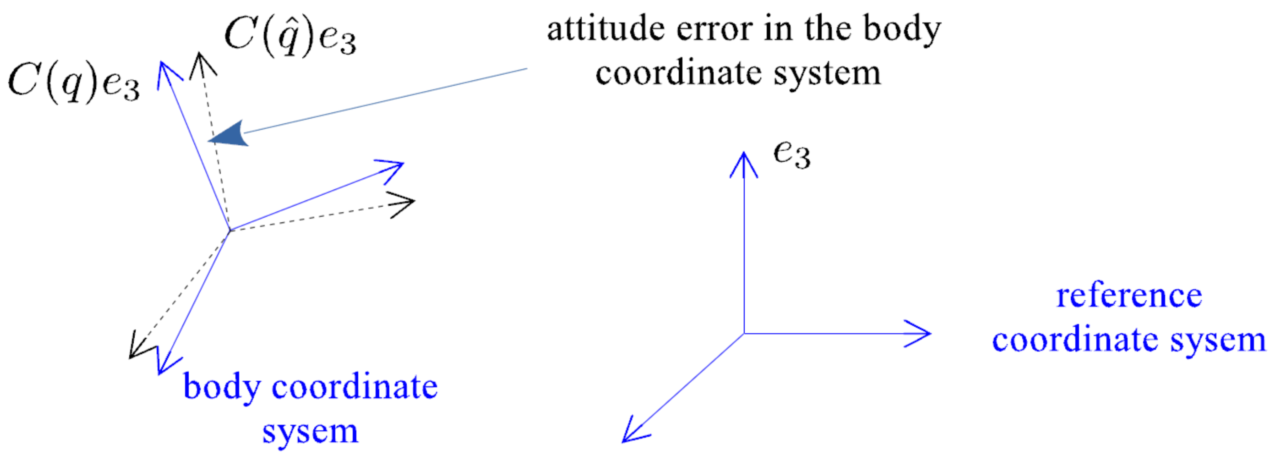
multiplicative quaternion error

- multiplicative quaternion error

true quaternion: q, estimated quaternion $\hat{q}$, attitude error quaternion q_e

$$q = \hat{q} \otimes q_e$$

$$C(q) = C(q_e)C(\hat{q})$$



- assumption: attitude error is small

$$q_e = \begin{bmatrix} \cos\frac{\theta}{2} \\ e\sin\frac{\theta}{2} \end{bmatrix} \xrightarrow{\theta \text{ is small}} q_e \approx \begin{bmatrix} 1 \\ \bar{q}_e \end{bmatrix} \qquad (\bar{q}_e \text{ is small})$$

attitude error propagation (1)

- gyroscope (yg)

$$y_g = \omega + v_g \ (\omega\text{: angular velocity, } v_g\text{: gyroscope measurement noise})$$

- quaternion dynamics

$$\begin{aligned} \dot{q} &= \tfrac{1}{2}\Omega(\omega)q = \tfrac{1}{2}q \otimes \begin{bmatrix} 0 \\ \omega \end{bmatrix} \\ \dot{\hat{q}} &= \tfrac{1}{2}\Omega(y_g)\hat{q} = \tfrac{1}{2}q \otimes \begin{bmatrix} 0 \\ y_g \end{bmatrix} \end{aligned}$$

$$\Omega(\omega) = \begin{bmatrix} 0 & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & 0 & \omega_z & -\omega_y \\ \omega_y & -\omega_z & 0 & \omega_x \\ \omega_z & \omega_y & -\omega_x & 0 \end{bmatrix}$$

- attitude error dynamics

$$q = \hat{q} \otimes q_e$$

$$\dot{q} = \dot{\hat{q}} \otimes q_e + \hat{q} \otimes \dot{q}_e$$

$$\frac{1}{2}q \otimes \begin{bmatrix} 0 \\ \omega_b \end{bmatrix} = \frac{1}{2}\hat{q} \otimes \begin{bmatrix} 0 \\ y_g \end{bmatrix} \otimes q_e + \hat{q} \otimes \dot{q}_e$$

attitude error propagation (2)

Premultiplying $\hat{q}^*$ to both sides, we have

$$\dot{q}_e = \frac{1}{2} q_e \otimes \begin{bmatrix} 0 \\ \omega_b \end{bmatrix} - \frac{1}{2} \begin{bmatrix} 0 \\ y_g \end{bmatrix} \otimes q_e$$

Using the fact

$$a \otimes b = b \otimes a + 2 \begin{bmatrix} 0 \\ \bar{a} \times \bar{b} \end{bmatrix}$$

we can obtain

$$\dot{q}_e = \frac{1}{2} q_e \otimes \begin{bmatrix} 0 \\ (\omega_b - y_g) \end{bmatrix} + \begin{bmatrix} 0 \\ (\bar{q}_e \times y_g) \end{bmatrix}$$

- vector part equation

$$\boxed{\dot{\bar{q}}_e \approx -y_g \times \bar{q}_e - \frac{1}{2} v_g = -[y_g \times] \bar{q}_e - \frac{1}{2} v_g}$$

Kalman filter

- state x

$$x = q_e \quad \Rightarrow \quad \dot{x} = Ax + w \qquad (A = -[y_g \times],\ w = -\tfrac{1}{2} v_g)$$

- discretization with the sampling period T

$$x_{k+1} = \exp(AT) x_k + w_k$$

where Q_d is given by

$$\begin{aligned} \int_0^T \exp(Ar) Q(r) \exp(Ar)'\, dr &\approx \int_0^T (I + Ar) Q(r) (I + Ar)'\, dr \\ &\approx \int_0^T Q(r) + ArQ(r) + Q(r)A'r\, dr \\ &\approx Q(0)T + \frac{T^2}{2} AQ(0) + \frac{T^2}{2} Q(0)A' + \frac{1}{3} AQA'T^3 \end{aligned}$$

- gyroscope noise

$$Q(0) = 0.25 r_g I_3$$

울산대학교
UNIVERSITY OF ULSAN

Kalman filter: measurement equation (1)

- measurement equation

$$
\begin{aligned}
y_a &= C(q)\tilde{g} + v_a = C(q_e)C(\hat{q})\tilde{g} + v_a \\
y_m &= C(q)\tilde{m} + v_m = C(q_e)C(\hat{q})\tilde{m} + v_m
\end{aligned}
$$

- C(qe)

$$
C(q) = \begin{bmatrix} 2q_0^2 + 2q_1^2 - 1 & 2q_1q_2 + 2q_0q_3 & 2q_1q_3 - 2q_0q_2 \\ 2q_1q_2 - 2q_0q_3 & 2q_0^2 + 2q_2^2 - 1 & 2q_2q_3 + 2q_0q_1 \\ 2q_1q_3 + 2q_0q_2 & 2q_2q_3 - 2q_0q_1 & 2q_0^2 + 2q_3^2 - 1 \end{bmatrix}
$$

- when the rotation is small $\Rightarrow q_{e,0} \approx 1,\ \bar{q}_1, \bar{q}_2, \bar{q}_3 \approx 0$

$$
q_e \approx \begin{bmatrix} 1 \\ \bar{q}_e \end{bmatrix}
$$

$$
\begin{aligned}
C(q_e) &\approx \begin{bmatrix} 1 & 2\bar{q}_3 & -2\bar{q}_2 \\ -2\bar{q}_3 & 1 & 2\bar{q}_1 \\ 2\bar{q}_2 & -2\bar{q}_1 & 1 \end{bmatrix} \\
&= I - 2\begin{bmatrix} 0 & -\bar{q}_3 & \bar{q}_2 \\ \bar{q}_3 & 0 & -\bar{q}_1 \\ -\bar{q}_2 & \bar{q}_1 & 0 \end{bmatrix} = (I - 2[\bar{q}_e \times])
\end{aligned}
$$

Kalman filter: measurement equation (2)

- measurement equation

$$\begin{aligned} y_a - C(\hat{q})\tilde{g} &= 2K(C(\hat{q})\tilde{g})q_e + v_a \\ y_m - C(\hat{q})\tilde{m} &= 2K(C(\hat{q})\tilde{m})q_e + v_m \end{aligned}$$

- In the first Kalman filter, only ya is used as measurement
  - ym is not used

matlab code (1) (attitude1.m in 3dattitude2.zip)

- initialization

```
load('3dsim.mat');
R2D = 180 / pi;
N = size(ya,2);
T = 0.01;
qhat = zeros(4,N);
eulerhat = zeros(3,N);
qhat(:,1) = q(:,1);                    % assume we know initial value
P = 0 * eye(3);
eulerhat(:,1) = quaternion2euler(q(:,1));
x = zeros(3,1);
gtilde = [0 ; 0 ; 9.8];
R = ra * eye(3);
```

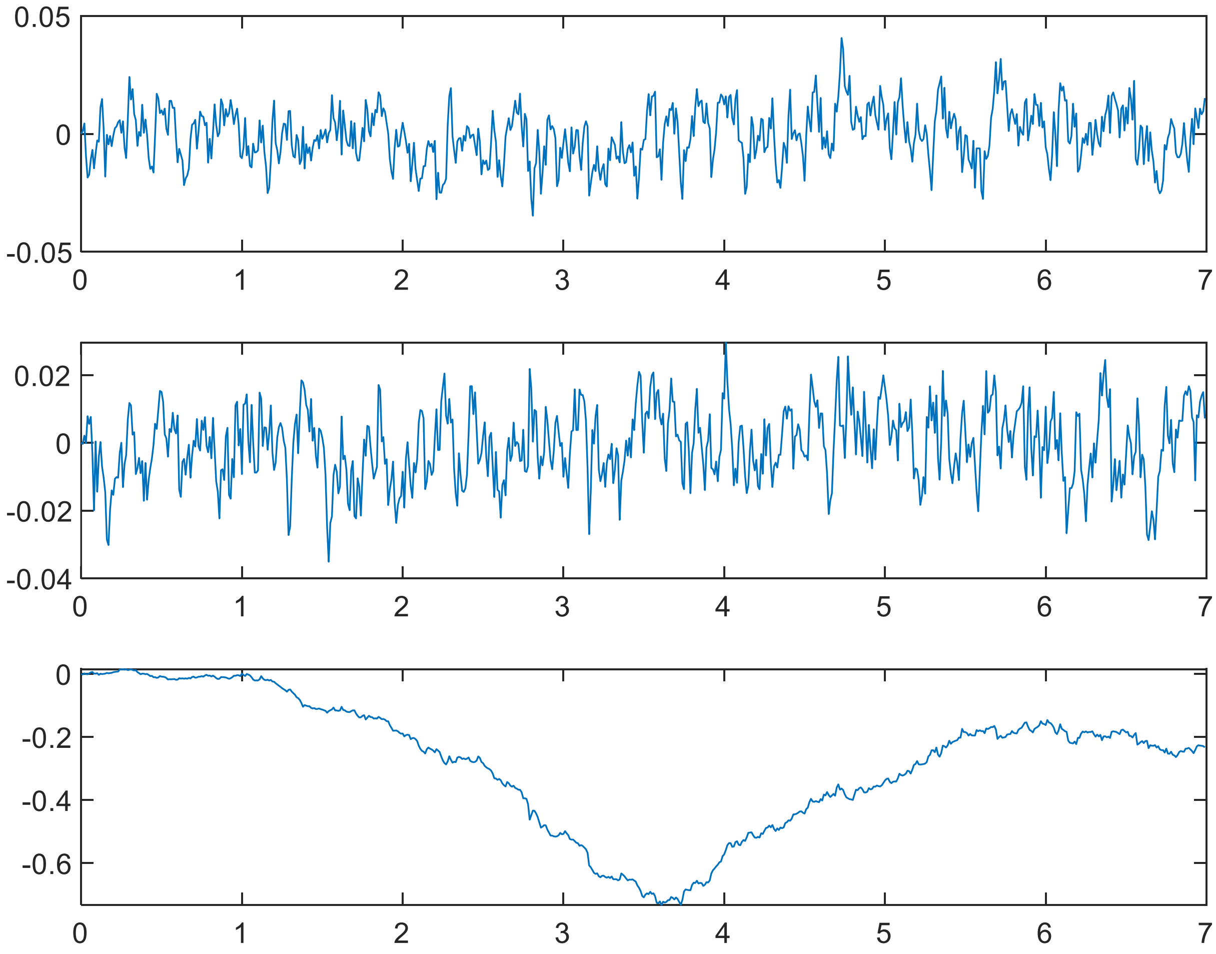
matlab code (2)

```
for i = 2:N
    wx = yg(1,i-1);
    wy = yg(1,i-1);
    wz = yg(1,i-1);
    Omega = [ 0 , -wx, -wy, -wz ; ...
          wx , 0 , wz , -wy ; wy , -wz, 0, wx ; wz , wy , -wx , 0 ];
    qhat(:,i) = (eye(4) + Omega * T) * qhat(:,i-1);        % integration
    qhat(:,i) = qhat(:,i) / norm(qhat(:,i));

    A = -vec2product(yg(:,i-1));
    phi = eye(3) + A*T;
    Qd = 0.25 * rg * (eye(3) * T + 0.5 * T^2 * (A + A') + (1/3) * A * A' * T^3);
```

matlab code (3)

```
    P = phi * P * phi' + Qd;
    Cq = quaternion2dcm(qhat(:,i));
    H = 2 * vec2product(Cq * gtilde);
    K = P * H' * inv(H * P * H' + R);

    z = ya(:,i) - Cq * gtilde;
    x = K * z;
    qe = [ 1 ; x ];
    qhat(:,i) = quaternionmul(qhat(:,i),qe);
    qhat(:,i) = qhat(:,i) / norm(qhat(:,i));

    eulerhat(:,i) = quaternion2euler(qhat(:,i));

    P = ( eye(3) - K * H ) * P;
    P = (P + P')/2;
end
```

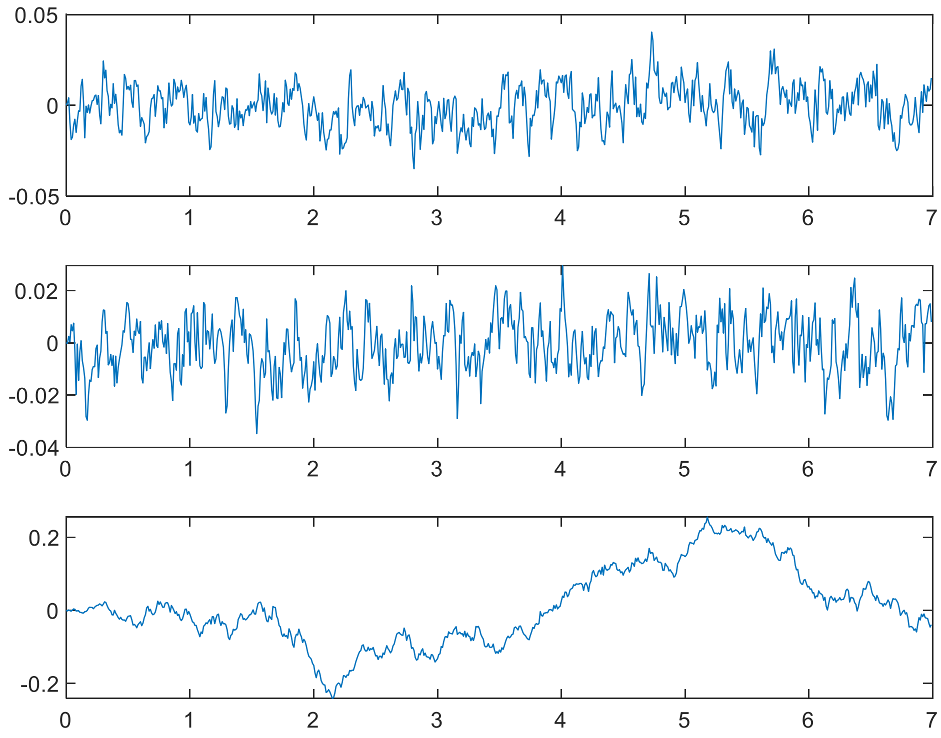
estimation error in Euler angles



attitude estimation (3dsim.mat)

- write Kalman filter code, which uses both ya and ym

```
alpha = 50 * (pi/180);
mtilde = [ cos(alpha) ; 0 ; -sin(alpha) ];
```



Kalman filter (1)

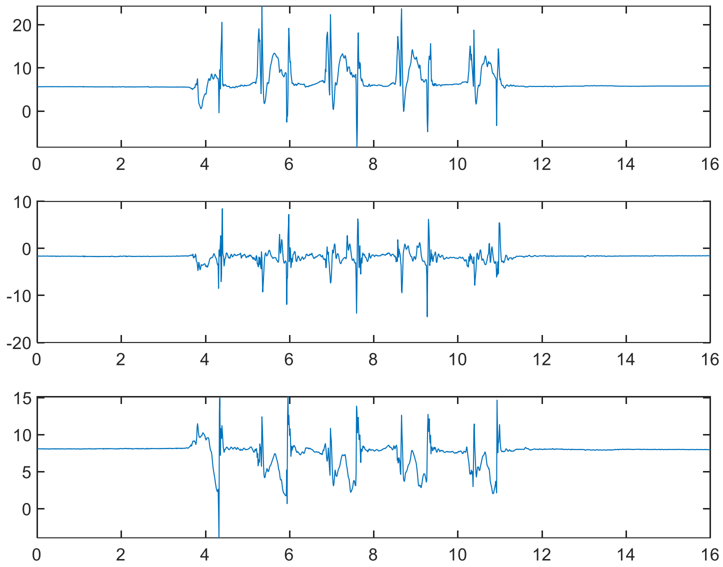
```
load('3dsim.mat');
R2D = 180 / pi;
N = size(ya,2);
T = 0.01;
qhat = zeros(4,N);
eulerhat = zeros(3,N);
qhat(:,1) = q(:,1);
eulerhat(:,1) = quaternion2euler(q(:,1));
x = zeros(3,1);
P = 0 * eye(3);
gtilde = [0 ; 0 ; 9.8];
alpha = 50 * (pi/180);
mtilde = [ cos(alpha) ; 0 ; -sin(alpha) ];
R = [ ra * eye(3) , zeros(3,3) ; zeros(3,3) , rm * eye(3) ];
for i = 2:N
    wx = yg(1,i-1);
    wy = yg(1,i-1);
    wz = yg(1,i-1);
```

Kalman filter (2)

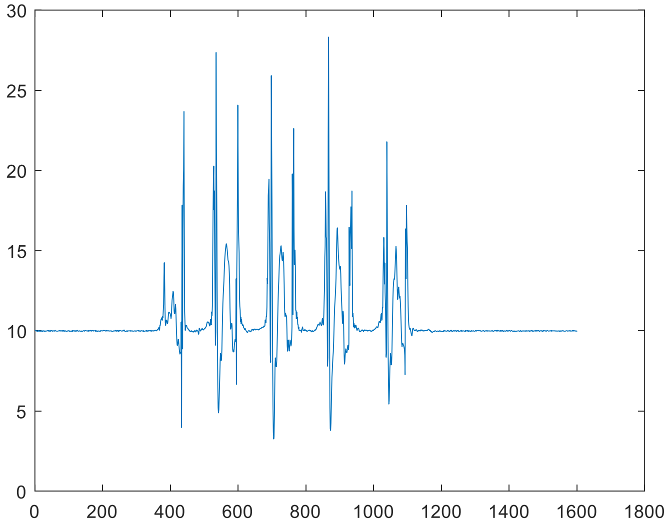
```
    A = -vec2product(yg(:,i-1));
    phi = eye(3) + A*T;
    Qd = 0.25 * rg * (eye(3) * T + 0.5 * T^2 * (A + A') + (1/3) * A * A' * T^3);
    P = phi * P * phi' + Qd;
    Cq = quaternion2dcm(qhat(:,i));
    H = [ 2 * vec2product(Cq * gtilde) ; 2 * vec2product(Cq * mtilde) ];
    K = P * H' * inv(H * P * H' + R);

    z = [ ya(:,i) - Cq * gtilde ; ym(:,i) - Cq * mtilde];
    x = K * z;
    qe = [ 1 ; x ];
    qhat(:,i) = quaternionmul(qhat(:,i),qe);
    qhat(:,i) = qhat(:,i) / norm(qhat(:,i));

    eulerhat(:,i) = quaternion2euler(qhat(:,i));

    P = ( eye(3) - K * H ) * P;
    P = (P + P')/2;
end
```

walking data

- foot.mat : inertial sensor is on the shoe
  - walking data



accelerometer



norm

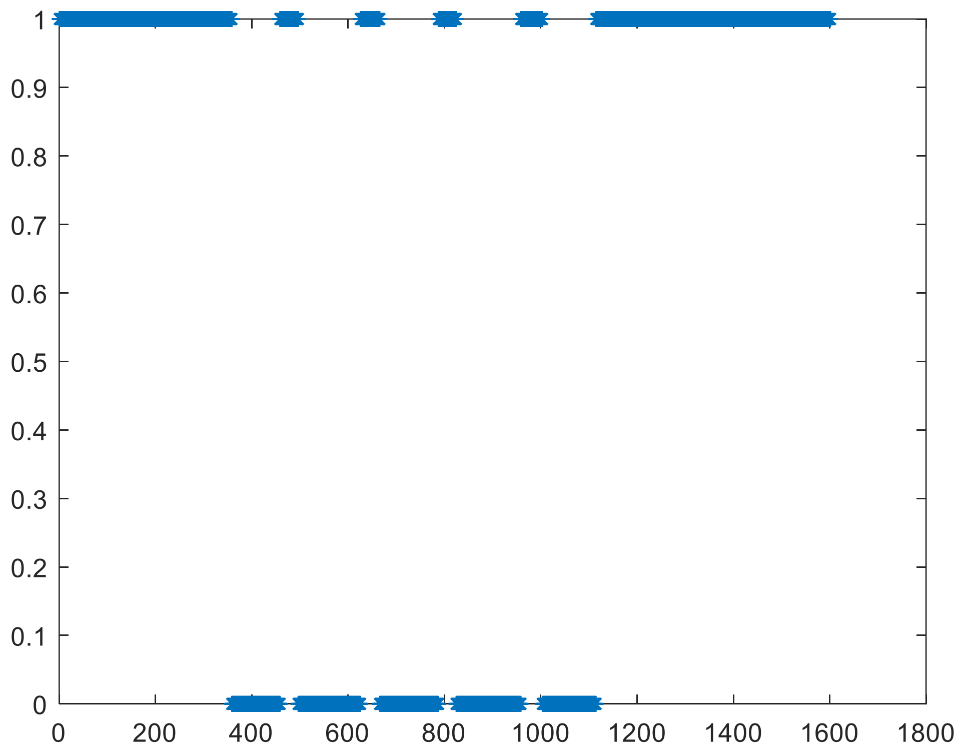
울산대학교 UNIVERSITY OF ULSAN

find zero external acceleration intervals

```
yanorm = zeros(1,N);
zerovel2 = zeros(1,N);
for i = 1:N
    yanorm(i) = norm(ya(:,i));
    if ( (yanorm(i) < 9.8+0.5) &&
(yanorm(i) > 9.8 -0.5) )
        zerovel2(i) = 1;
    end
end
```

```
zerovel = zerovel2;
M = 10;
for i = M+1:N-M
    if ( sum(zerovel2(i-M:i+M)) ==
2*M+1 )
        zerovel(i) = 1;
    else
        zerovel(i) = 0;
    end
end

plot(zerovel,'*')
```

zero velocity interval



Kalman filter code

- Write a matlab code, where measurement update is done only if zerovel(i) = 1

```
load('walking.mat');
N = size(ya,2);
ra = 0.005;
rg = 0.001;
```

Kalman filter (1)

```
qhat = zeros(4,N);
eulerhat = zeros(3,N);
qhat(:,1) = quaternionya(ya(:,1));
eulerhat(:,1) = quaternion2euler(q(:,1));
x = zeros(3,1);
P = 0 * eye(3);
gtilde = [0 ; 0 ; 9.8];
R = ra * eye(3);
for i = 2:N
    wx = yg(1,i-1);
    wy = yg(1,i-1);
    wz = yg(1,i-1);
    Omega = [ 0 , -wx, -wy, -wz ; wx , 0 , wz , -wy ; wy , -wz, 0, wx ; wz , wy , -
wx , 0 ];
    qhat(:,i) = (eye(4) + Omega * T) * qhat(:,i-1);
    qhat(:,i) = qhat(:,i) / norm(qhat(:,i));
```

Kalman filter (2)

```
A = -vec2product(yg(:,i-1));
phi = eye(3) + A*T;
Qd = 0.25 * rg * (eye(3) * T + 0.5 * T^2 * (A + A') + (1/3) * A * A' * T^3);
P = phi * P * phi' + Qd;
Cq = quaternion2dcm(qhat(:,i));
H = 2 * vec2product(Cq * gtilde);
```

Kalman filter (3)

```
    if ( zerovel(i) == 1 )
        K = P * H' * inv(H * P * H' + R);
         z = ya(:,i) - Cq * gtilde;
        x = K * z;
        qe = [ 1 ; x ];
        qhat(:,i) = quaternionmul(qhat(:,i),qe);
        qhat(:,i) = qhat(:,i) / norm(qhat(:,i));
         P = ( eye(3) - K * H ) * P;
        P = (P + P')/2;
    end
    eulerhat(:,i) = quaternion2euler(qhat(:,i));
end
```