```
#include "quaternion.h"
#include <math.h>

EulerAngleTypeDef ea_pre;

void QuaternionNorm(QuaternionTypeDef *q)
{
    float norm;

    norm = invSqrt(q->q0*q->q0 + q->q1*q->q1 + q->q2*q->q2 + q->q3*q->q3);
    q->q0 *= norm;
    q->q1 *= norm;
    q->q2 *= norm;
    q->q3 *= norm;
}

/*
 * Quaternion Multiplay - qo = qa * qb
 * note - qo can be different from qa/qb, or the same as qa/qb
 */
void QuaternionMult(QuaternionTypeDef *qa, QuaternionTypeDef *qb, QuaternionTypeDef *qo)
{
    float q0, q1, q2, q3;

    q0 = qa->q0*qb->q0 - qa->q1*qb->q1 - qa->q2*qb->q2 - qa->q3*qb->q3;
    q1 = qa->q0*qb->q1 + qa->q1*qb->q0 + qa->q2*qb->q3 - qa->q3*qb->q2;
    q2 = qa->q0*qb->q2 - qa->q1*qb->q3 + qa->q2*qb->q0 + qa->q3*qb->q1;
    q3 = qa->q0*qb->q3 + qa->q1*qb->q2 - qa->q2*qb->q1 + qa->q3*qb->q0;
    qo->q0 = q0; qo->q1 = q1; qo->q2 = q2; qo->q3 = q3;
}


/*
 * This function calculate the vector rotaton via quatornion
 * qr - rotation quaternion
 * qv - vector to rotate (qv->q0 = 0)
 * qo - output vector (qo->q0 = 0)
 * qo = qr' * qv * qr
 */
void QuaternionRotation(QuaternionTypeDef *qr, QuaternionTypeDef *qv, QuaternionTypeDef
*qo)
{
    float q0q0, q1q1, q2q2, q3q3;
    float dq0, dq1, dq2;
    float dq1q2, dq1q3, dq0q2, dq0q3;
    float dq0q1, dq2q3;

    q0q0 = qr->q0*qr->q0;
    q1q1 = qr->q1*qr->q1;
    q2q2 = qr->q2*qr->q2;
```

```
    q3q3 = qr->q3*qr->q3;
    dq0 = 2*qr->q0;
    dq1 = 2*qr->q1;
    dq2 = 2*qr->q2;
    dq1q2 = dq1 * qr->q2;
    dq1q3 = dq1 * qr->q3;
    dq0q2 = dq0 * qr->q2;
    dq0q3 = dq0 * qr->q3;
    dq0q1 = dq0 * qr->q1;
    dq2q3 = dq2 * qr->q3;

    qo->q0 = 0;
    qo->q1 = (q0q0+q1q1-q2q2-q3q3)*qv->q1 + (dq1q2+dq0q3)*qv->q2 + (dq1q3-dq0q2)*qv->q3;
    qo->q2 = (dq1q2-dq0q3)*qv->q1 + (q0q0+q2q2-q1q1-q3q3)*qv->q2 + (dq0q1+dq2q3)*qv->q3;
    qo->q3 = (dq0q2+dq1q3)*qv->q1 + (dq2q3-dq0q1)*qv->q2 + (q0q0+q3q3-q1q1-q2q2)*qv->q3;
}

void QuaternionConj(QuaternionTypeDef *qa, QuaternionTypeDef *qo)
{
    qo->q0 = qa->q0;
    qo->q1 = -qa->q1;
    qo->q2 = -qa->q2;
    qo->q3 = -qa->q3;
}

/*
 * Convert Quaternion to Euler Angle
 */
void QuaternionToEuler(QuaternionTypeDef *qr, EulerAngleTypeDef *ea)
{
    float q0q0, q1q1, q2q2, q3q3;
    float dq0, dq1, dq2;
    float dq1q3, dq0q2, dq1q2;
    float dq0q1, dq2q3, dq0q3;

    q0q0 = qr->q0*qr->q0;
    q1q1 = qr->q1*qr->q1;
    q2q2 = qr->q2*qr->q2;
    q3q3 = qr->q3*qr->q3;
    dq0 = 2*qr->q0;
    dq1 = 2*qr->q1;
    dq2 = 2*qr->q2;
    dq1q2 = dq1 * qr->q2;
    dq1q3 = dq1 * qr->q3;
    dq0q2 = dq0 * qr->q2;
    dq0q3 = dq0 * qr->q3;
    dq0q1 = dq0 * qr->q1;
    dq2q3 = dq2 * qr->q3;

    ea->thx = atan2(dq0q1+dq2q3, q0q0+q3q3-q1q1-q2q2);
```

クォータニオンをピッチ・ロールのオイラー角へ変換する関数

thx:ピッチのオイラー角

```
100     ea->thy = asin(dq0q2-dq1q3);
101
102     /* This part is removed to manage angle >90deg */
103 //    if(ea->thx > MAX_RAD || ea->thx < -MAX_RAD)
104 //      ea->thx = ea_pre.thx;
105 //    if(ea->thy > MAX_RAD || ea->thy < -MAX_RAD)
106 //      ea->thy = ea_pre.thy;
107 //
108 //    ea_pre.thx = ea->thx;
109 //    ea_pre.thy = ea->thy;
110
111
112
113
114     //ea->thz = atan2(dq1q2+dq0q3, q0q0+q1q1-q2q2-q3q3);
115
116 }
```

thy:ロールのオイラー角