```c
/**
 *    ||          ____  _ __
 * +------+      / __ )(_) /_______________ _____  ___
 * | 0xBC |     / __  / / __/ ___/ ___/ __ `/_  / / _ \
 * +------+    / /_/ / / /_/ /__/ /  / /_/ / / /_/  __/
 *  ||  ||    /_____/_/\__/\___/_/   \__,_/ /___/\___/
 *
 * Crazyflie Firmware
 *
 * Copyright (C) 2011-2012 Bitcraze AB
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, in version 3.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 *
 *
 */
#include "stm32f10x_conf.h"
#include <math.h>

#include "sensfusion6.h"
#include "imu.h"
#include "param.h"

//#define MADWICK_QUATERNION_IMU

#ifdef MADWICK_QUATERNION_IMU
  #define BETA_DEF     0.01f    // 2 * proportional gain
#else // MAHONY_QUATERNION_IMU
    #define TWO_KP_DEF  (2.0f * 0.4f) // 2 * proportional gain
    #define TWO_KI_DEF  (2.0f * 0.001f) // 2 * integral gain
#endif

#ifdef MADWICK_QUATERNION_IMU
  float beta = BETA_DEF;     // 2 * proportional gain (Kp)
#else // MAHONY_QUATERNION_IMU
  float twoKp = TWO_KP_DEF;    // 2 * proportional gain (Kp)
  float twoKi = TWO_KI_DEF;    // 2 * integral gain (Ki)
  float integralFBx = 0.0f;
  float integralFBy = 0.0f;
  float integralFBz = 0.0f;  // integral error terms scaled by Ki
#endif

float q0 = 1.0f;
float q1 = 0.0f;
float q2 = 0.0f;
float q3 = 0.0f;  // quaternion of sensor frame relative to auxiliary frame

static bool isInit;

// TODO: Make math util file
static float invSqrt(float x);

void sensfusion6Init()
{
  if(isInit)
    return;

  isInit = TRUE;
```

```c
}

bool sensfusion6Test(void)
{
  return isInit;
}


#ifdef MADWICK_QUATERNION_IMU
// Implementation of Madgwick's IMU and AHRS algorithms.
// See: http://www.x-io.co.uk/open-source-ahrs-with-x-imu
//
// Date     Author          Notes
// 29/09/2011 SOH Madgwick    Initial release
// 02/10/2011 SOH Madgwick  Optimised for reduced CPU load
void sensfusion6UpdateQ(float gx, float gy, float gz, float ax, float ay, float az, float dt)
{
  float recipNorm;
  float s0, s1, s2, s3;
  float qDot1, qDot2, qDot3, qDot4;
  float _2q0, _2q1, _2q2, _2q3, _4q0, _4q1, _4q2 ,_8q1, _8q2, q0q0, q1q1, q2q2, q3q3;

  // Rate of change of quaternion from gyroscope
  qDot1 = 0.5f * (-q1 * gx - q2 * gy - q3 * gz);
  qDot2 = 0.5f * (q0 * gx + q2 * gz - q3 * gy);
  qDot3 = 0.5f * (q0 * gy - q1 * gz + q3 * gx);
  qDot4 = 0.5f * (q0 * gz + q1 * gy - q2 * gx);

  // Compute feedback only if accelerometer measurement valid (avoids NaN in accelerometer normalisation)
  if(!((ax == 0.0f) && (ay == 0.0f) && (az == 0.0f)))
  {
    // Normalise accelerometer measurement
    recipNorm = invSqrt(ax * ax + ay * ay + az * az);
    ax *= recipNorm;
    ay *= recipNorm;
    az *= recipNorm;

    // Auxiliary variables to avoid repeated arithmetic
    _2q0 = 2.0f * q0;
    _2q1 = 2.0f * q1;
    _2q2 = 2.0f * q2;
    _2q3 = 2.0f * q3;
    _4q0 = 4.0f * q0;
    _4q1 = 4.0f * q1;
    _4q2 = 4.0f * q2;
    _8q1 = 8.0f * q1;
    _8q2 = 8.0f * q2;
    q0q0 = q0 * q0;
    q1q1 = q1 * q1;
    q2q2 = q2 * q2;
    q3q3 = q3 * q3;

    // Gradient decent algorithm corrective step
    s0 = _4q0 * q2q2 + _2q2 * ax + _4q0 * q1q1 - _2q1 * ay;
    s1 = _4q1 * q3q3 - _2q3 * ax + 4.0f * q0q0 * q1 - _2q0 * ay - _4q1 + _8q1 * q1q1 + _8q1 * q2q2 + _4q1
* az;
    s2 = 4.0f * q0q0 * q2 + _2q0 * ax + _4q2 * q3q3 - _2q3 * ay - _4q2 + _8q2 * q1q1 + _8q2 * q2q2 + _4q2
* az;
    s3 = 4.0f * q1q1 * q3 - _2q1 * ax + 4.0f * q2q2 * q3 - _2q2 * ay;
    recipNorm = invSqrt(s0 * s0 + s1 * s1 + s2 * s2 + s3 * s3); // normalise step magnitude
    s0 *= recipNorm;
    s1 *= recipNorm;
    s2 *= recipNorm;
    s3 *= recipNorm;

    // Apply feedback step
    qDot1 -= beta * s0;
```

```
    qDot2 -= beta * s1;
    qDot3 -= beta * s2;
    qDot4 -= beta * s3;
  }

  // Integrate rate of change of quaternion to yield quaternion
  q0 += qDot1 * dt;
  q1 += qDot2 * dt;
  q2 += qDot3 * dt;
  q3 += qDot4 * dt;

  // Normalise quaternion
  recipNorm = invSqrt(q0*q0 + q1*q1 + q2*q2 + q3*q3);
  q0 *= recipNorm;
  q1 *= recipNorm;
  q2 *= recipNorm;
  q3 *= recipNorm;
}
#else // MAHONY_QUATERNION_IMU
// Madgwick's implementation of Mayhony's AHRS algorithm.
// See: http://www.x-io.co.uk/open-source-ahrs-with-x-imu
//
// Date     Author      Notes
// 29/09/2011 SOH Madgwick    Initial release
// 02/10/2011 SOH Madgwick  Optimised for reduced CPU load
void sensfusion6UpdateQ(float gx, float gy, float gz, float ax, float ay, float az, float dt)
{
  float recipNorm;
  float halfvx, halfvy, halfvz;
  float halfex, halfey, halfez;
  float qa, qb, qc;

  gx = gx * M_PI / 180;
  gy = gy * M_PI / 180;
  gz = gz * M_PI / 180;

  // Compute feedback only if accelerometer measurement valid (avoids NaN in accelerometer normalisation)
  if(!((ax == 0.0f) && (ay == 0.0f) && (az == 0.0f)))
  {
    // Normalise accelerometer measurement
    recipNorm = invSqrt(ax * ax + ay * ay + az * az);
    ax *= recipNorm;
    ay *= recipNorm;
    az *= recipNorm;

    // Estimated direction of gravity and vector perpendicular to magnetic flux
    halfvx = q1 * q3 - q0 * q2;
    halfvy = q0 * q1 + q2 * q3;
    halfvz = q0 * q0 - 0.5f + q3 * q3;

    // Error is sum of cross product between estimated and measured direction of gravity
    halfex = (ay * halfvz - az * halfvy);
    halfey = (az * halfvx - ax * halfvz);
    halfez = (ax * halfvy - ay * halfvx);

    // Compute and apply integral feedback if enabled
    if(twoKi > 0.0f)
    {
      integralFBx += twoKi * halfex * dt;  // integral error scaled by Ki
      integralFBy += twoKi * halfey * dt;
      integralFBz += twoKi * halfez * dt;
      gx += integralFBx;  // apply integral feedback
      gy += integralFBy;
      gz += integralFBz;
    }
    else
    {
```

```c
      integralFBx = 0.0f; // prevent integral windup
      integralFBy = 0.0f;
      integralFBz = 0.0f;
    }

    // Apply proportional feedback
    gx += twoKp * halfex;
    gy += twoKp * halfey;
    gz += twoKp * halfez;
  }

  // Integrate rate of change of quaternion
  gx *= (0.5f * dt);   // pre-multiply common factors
  gy *= (0.5f * dt);
  gz *= (0.5f * dt);
  qa = q0;
  qb = q1;
  qc = q2;
  q0 += (-qb * gx - qc * gy - q3 * gz);
  q1 += (qa * gx + qc * gz - q3 * gy);
  q2 += (qa * gy - qb * gz + q3 * gx);
  q3 += (qa * gz + qb * gy - qc * gx);

  // Normalise quaternion
  recipNorm = invSqrt(q0 * q0 + q1 * q1 + q2 * q2 + q3 * q3);
  q0 *= recipNorm;
  q1 *= recipNorm;
  q2 *= recipNorm;
  q3 *= recipNorm;
}
#endif

void sensfusion6GetEulerRPY(float* roll, float* pitch, float* yaw)
{
  float gx, gy, gz; // estimated gravity direction

  gx = 2 * (q1*q3 - q0*q2);
  gy = 2 * (q0*q1 + q2*q3);
  gz = q0*q0 - q1*q1 - q2*q2 + q3*q3;

  *yaw = atan2(2*q1*q2 - 2*q0*q3, 2*q0*q0 + 2*q1*q1 - 1) * 180 / M_PI;
  *pitch = atan(gx / sqrt(gy*gy + gz*gz)) * 180 / M_PI;
  *roll = atan(gy / sqrt(gx*gx + gz*gz)) * 180 / M_PI;
}

//---------------------------------------------------------------------------------------------------
// Fast inverse square-root
// See: http://en.wikipedia.org/wiki/Fast_inverse_square_root
float invSqrt(float x)
{
  float halfx = 0.5f * x;
  float y = x;
  long i = *(long*)&y;
  i = 0x5f3759df - (i>>1);
  y = *(float*)&i;
  y = y * (1.5f - (halfx * y * y));
  return y;
}


PARAM_GROUP_START(sensorfusion6)
#ifdef MADWICK_QUATERNION_IMU
PARAM_ADD(PARAM_FLOAT, beta, &beta)
#else // MAHONY_QUATERNION_IMU
PARAM_ADD(PARAM_FLOAT, kp, &twoKp)
PARAM_ADD(PARAM_FLOAT, ki, &twoKi)
#endif
```

```
PARAM_GROUP_STOP(sensorfusion6)
```