```
/*********************************************************************************************
********************
* Razor AHRS Firmware
* 9 Degree of Measurement Attitude and Heading Reference System
* for Sparkfun "9DOF Razor IMU" (SEN-10125 and SEN-10736)
* and "9DOF Sensor Stick" (SEN-10183, 10321 and SEN-10724)
*
* Released under GNU GPL (General Public License) v3.0
* Copyright (C) 2013 Peter Bartz [http://ptrbrtz.net]
* Copyright (C) 2011-2012 Quality & Usability Lab, Deutsche Telekom Laboratories, TU Berlin
*
* Infos, updates, bug reports, contributions and feedback:
*     https://github.com/ptrbrtz/razor-9dof-ahrs
*
*
* History:
*   * Original code (http://code.google.com/p/sf9domahrs/) by Doug Weibel and Jose Julio,
*     based on ArduIMU v1.5 by Jordi Munoz and William Premerlani, Jose Julio and Doug Weib
el. Thank you!
*
*   * Updated code (http://groups.google.com/group/sf_9dof_ahrs_update) by David Malik (dav
id.zsolt.malik@gmail.com)
*     for new Sparkfun 9DOF Razor hardware (SEN-10125).
*
*   * Updated and extended by Peter Bartz (peter-bartz@gmx.de):
*     * v1.3.0
*       * Cleaned up, streamlined and restructured most of the code to make it more compreh
ensible.
*       * Added sensor calibration (improves precision and responsiveness a lot!).
*       * Added binary yaw/pitch/roll output.
*       * Added basic serial command interface to set output modes/calibrate sensors/synch
stream/etc.
*       * Added support to synch automatically when using Rovering Networks Bluetooth modul
es (and compatible).
*       * Wrote new easier to use test program (using Processing).
*       * Added support for new version of "9DOF Razor IMU": SEN-10736.
*       --> The output of this code is not compatible with the older versions!
*       --> A Processing sketch to test the tracker is available.
*     * v1.3.1
*       * Initializing rotation matrix based on start-up sensor readings -> orientation OK
right away.
*       * Adjusted gyro low-pass filter and output rate settings.
*     * v1.3.2
*       * Adapted code to work with new Arduino 1.0 (and older versions still).
*     * v1.3.3
*       * Improved synching.
*     * v1.4.0
*       * Added support for SparkFun "9DOF Sensor Stick" (versions SEN-10183, SEN-10321 and
 SEN-10724).
*     * v1.4.1
*       * Added output modes to read raw and/or calibrated sensor data in text or binary fo
rmat.
*       * Added static magnetometer soft iron distortion compensation.
*     * v1.4.2
*       * (No core firmware changes)
*     * v1.5
*       * Added support for "9DoF Razor IMU M0": SEN-14001.
*     * v1.5.1
*       * Added ROS-compatible output mode.
*     * v1.5.2
*       * Added ROS-compatible input mode to set calibration parameters.
*     * v1.5.3
*       * Fixed the problem where commands were ignored by the M0 depending on how they wer
e sent.
*     * v1.5.4
*       * Attempts to fix random nan problems in orientation computations.
*       * Added an option to get yaw/pitch/roll from the M0 DMP.
*       * Added a command to enable/disable the use of magnetometers for yaw computation.
*     * v1.5.5
*       * Added various options to determine the most stable configuration for the M0.
*     * v1.5.6
*       * Removed some unnecessary math error checking.
*       * Set back gyro full-scale range to the maximum for the M0.
*       * Increased startup delay to try to get a correct initial orientation for the M0.
*     * v1.5.7
*       * Calibration data are now also used to compute the initial orientation.
*
* TODOs:
*   * Allow optional use of Flash/EEPROM for storing and reading calibration values.
*   * Use self-test and temperature-compensation features of the sensors.
*********************************************************************************************
*******************/

/*
  "9DoF Razor IMU M0" hardware versions: SEN-14001

  Arduino IDE : Follow the same instructions as for the default firmware on
  https://learn.sparkfun.com/tutorials/9dof-razor-imu-m0-hookup-guide
  and use an updated version of SparkFun_MPU-9250-DMP_Arduino_Library from
  https://github.com/lebarsfa/SparkFun_MPU-9250-DMP_Arduino_Library"
*/

/*
  "9DOF Razor IMU" hardware versions: SEN-10125 and SEN-10736

  ATMega328@3.3V, 8MHz

  ADXL345  : Accelerometer
  HMC5843  : Magnetometer on SEN-10125
  HMC5883L : Magnetometer on SEN-10736
  ITG-3200 : Gyro

  Arduino IDE : Select board "Arduino Pro or Pro Mini (3.3v, 8Mhz) w/ATmega328"
*/

/*
  "9DOF Sensor Stick" hardware versions: SEN-10183, SEN-10321 and SEN-10724

  ADXL345  : Accelerometer
  HMC5843  : Magnetometer on SEN-10183 and SEN-10321
  HMC5883L : Magnetometer on SEN-10724
  ITG-3200 : Gyro
*/

/*
  Axis definition (differs from definition printed on the board!):
    X axis pointing forward (towards the short edge with the connector holes)
    Y axis pointing to the right
    and Z axis pointing down.

  Positive yaw   : clockwise
  Positive roll  : right wing down
  Positive pitch : nose up

  Transformation order: first yaw then pitch then roll.
*/

/*
  Serial commands that the firmware understands:

  "#c<params>" - SET _c_alibration parameters. The available options are:
    [a|m|g|c|t] _a_ccelerometer, _m_agnetometer, _g_yro, magnetometerellipsoid_c_enter, mag
netometerellipsoid_t_ransform.
    [x|y|z] x,y or z.
    [m|M|X|Y|Z] _m_in or _M_ax (accel or magnetometer), X, Y, or Z of transform (magnetomet
erellipsoid_t_ransform).
```

```
  "#p" - PRINT current calibration values.


  "#o<params>" - Set OUTPUT mode and parameters. The available options are:

      // Streaming output
      "#o0" - DISABLE continuous streaming output. Also see #f below.
      "#o1" - ENABLE continuous streaming output.

      // Angles output
      "#ob" - Output angles in BINARY format (yaw/pitch/roll as binary float, so one output
 frame
              is 3x4 = 12 bytes long).
      "#ot" - Output angles in TEXT format (Output frames have form like "#YPR=-142.28,-5.3
8,33.52",
              followed by carriage return and line feed [\r\n]).
      "#ox" - Output angles and linear acceleration and rotational
              velocity. Angles are in degrees, acceleration is
              in units of 1.0 = 1/256 G (9.8/256 m/s^2). Rotational
              velocity is in rad/s. (Output frames have form like
              "#YPRAG=-142.28,-5.38,33.52,0.1,0.1,1.0,0.01,0.01,0.01",
              followed by carriage return and line feed [\r\n]).

      // Sensor calibration
      "#oc" - Go to CALIBRATION output mode.
      "#on" - When in calibration mode, go on to calibrate NEXT sensor.

      // Sensor data output
      "#osct" - Output CALIBRATED SENSOR data of all 9 axes in TEXT format.
                One frame consist of three lines - one for each sensor: acc, mag, gyr.
      "#osrt" - Output RAW SENSOR data of all 9 axes in TEXT format.
                One frame consist of three lines - one for each sensor: acc, mag, gyr.
      "#osbt" - Output BOTH raw and calibrated SENSOR data of all 9 axes in TEXT format.
                One frame consist of six lines - like #osrt and #osct combined (first RAW,
then CALIBRATED).
                NOTE: This is a lot of number-to-text conversion work for the little 8MHz c
hip on the Razor boards.
                In fact it's too much and an output frame rate of 50Hz can not be maintaine
d. #osbb.
      "#oscb" - Output CALIBRATED SENSOR data of all 9 axes in BINARY format.
                One frame consist of three 3x3 float values = 36 bytes. Order is: acc x/y/z
, mag x/y/z, gyr x/y/z.
      "#osrb" - Output RAW SENSOR data of all 9 axes in BINARY format.
                One frame consist of three 3x3 float values = 36 bytes. Order is: acc x/y/z
, mag x/y/z, gyr x/y/z.
      "#osbb" - Output BOTH raw and calibrated SENSOR data of all 9 axes in BINARY format.
                One frame consist of 2x36 = 72 bytes - like #osrb and #oscb combined (first
 RAW, then CALIBRATED).

      // Error message output
      "#oe0" - Disable ERROR message output.
      "#oe1" - Enable ERROR message output.
      "#oec" - Output ERROR COUNT.
      "#oem" - Output MATH ERROR COUNT.


  "#I" - Toggle INERTIAL-only mode for yaw computation.


  "#f" - Request one output frame - useful when continuous output is disabled and updates a
re
         required in larger intervals only. Though #f only requests one reply, replies are
still
         bound to the internal 20ms (50Hz) time raster. So worst case delay that #f can add
 is 19.99ms.
```

```
  "#s<xy>" - Request synch token - useful to find out where the frame boundaries are in a c
ontinuous
         binary stream or to see if tracker is present and answering. The tracker will send
         "#SYNCH<xy>\r\n" in response (so it's possible to read using a readLine() function
).
         x and y are two mandatory but arbitrary bytes that can be used to find out which r
equest
         the answer belongs to.


  ("#C" and "#D" - Reserved for communication with optional Bluetooth module.)

  Newline characters are not required. So you could send "#ob#o1#s", which
  would set binary output mode, enable continuous streaming output and request
  a synch token all at once.

  The status LED will be on if streaming output is enabled and off otherwise.

  Byte order of binary output is little-endian: least significant byte comes first.
*/



/*****************************************************************/
/*********** USER SETUP AREA! Set your options here! *************/
/*****************************************************************/

// HARDWARE OPTIONS
/*****************************************************************/
// Select your hardware here by uncommenting one line!
//#define HW__VERSION_CODE 10125 // SparkFun "9DOF Razor IMU" version "SEN-10125" (HMC5843
magnetometer)
//#define HW__VERSION_CODE 10736 // SparkFun "9DOF Razor IMU" version "SEN-10736" (HMC5883L
 magnetometer)
#define HW__VERSION_CODE 14001 // SparkFun "9DoF Razor IMU M0" version "SEN-14001"
//#define HW__VERSION_CODE 10183 // SparkFun "9DOF Sensor Stick" version "SEN-10183" (HMC58
43 magnetometer)
//#define HW__VERSION_CODE 10321 // SparkFun "9DOF Sensor Stick" version "SEN-10321" (HMC58
43 magnetometer)
//#define HW__VERSION_CODE 10724 // SparkFun "9DOF Sensor Stick" version "SEN-10724" (HMC58
83L magnetometer)


// OUTPUT OPTIONS
/*****************************************************************/
// Set your serial port baud rate used to send out data here!
#define OUTPUT__BAUD_RATE 57600
#if HW__VERSION_CODE == 14001
// Set your port used to send out data here!
#define LOG_PORT SERIAL_PORT_USBVIRTUAL
#else
// Set your port used to send out data here!
#define LOG_PORT Serial
#endif // HW__VERSION_CODE

// Sensor data output interval in milliseconds
// This may not work, if faster than 20ms (=50Hz)
// Code is tuned for 20ms, so better leave it like that
#define OUTPUT__DATA_INTERVAL 20  // in milliseconds

// Output mode definitions (do not change)
#define OUTPUT__MODE_CALIBRATE_SENSORS 0 // Outputs sensor min/max values as text for manua
l calibration
#define OUTPUT__MODE_ANGLES 1 // Outputs yaw/pitch/roll in degrees
#define OUTPUT__MODE_SENSORS_CALIB 2 // Outputs calibrated sensor values for all 9 axes
#define OUTPUT__MODE_SENSORS_RAW 3 // Outputs raw (uncalibrated) sensor values for all 9 ax
es
```

```
#define OUTPUT__MODE_SENSORS_BOTH 4 // Outputs calibrated AND raw sensor values for all 9 a
xes
#define OUTPUT__MODE_ANGLES_AG_SENSORS 5 // Outputs yaw/pitch/roll in degrees + linear acce
l + rot. vel
// Output format definitions (do not change)
#define OUTPUT__FORMAT_TEXT 0 // Outputs data as text
#define OUTPUT__FORMAT_BINARY 1 // Outputs data as binary float

// Select your startup output mode and format here!
int output_mode = OUTPUT__MODE_SENSORS_RAW;
int output_format = OUTPUT__FORMAT_TEXT;

// Select if serial continuous streaming output is enabled per default on startup.
#define OUTPUT__STARTUP_STREAM_ON true  // true or false

// If set true, an error message will be output if we fail to read sensor data.
// Message format: "!ERR: reading <sensor>", followed by "\r\n".
boolean output_errors = false;  // true or false

// Bluetooth
// You can set this to true, if you have a Rovering Networks Bluetooth Module attached.
// The connect/disconnect message prefix of the module has to be set to "#".
// (Refer to manual, it can be set like this: SO,#)
// When using this, streaming output will only be enabled as long as we’re connected. That
way
// receiver and sender are synchronzed easily just by connecting/disconnecting.
// It is not necessary to set this! It just makes life easier when writing code for
// the receiving side. The Processing test sketch also works without setting this.
// NOTE: When using this, OUTPUT__STARTUP_STREAM_ON has no effect!
#define OUTPUT__HAS_RN_BLUETOOTH false  // true or false


// SENSOR CALIBRATION
/*****************************************************************/
// How to calibrate? Read the tutorial at http://dev.qu.tu-berlin.de/projects/sf-razor-9dof
-ahrs
// Put MIN/MAX and OFFSET readings for your board here!
// For the M0, only the extended magnetometer calibration seems to be really necessary if D
EBUG__USE_DMP_M0 is set to true...
// Accelerometer
// "accel x,y,z (min/max) = X_MIN/X_MAX  Y_MIN/Y_MAX  Z_MIN/Z_MAX"
float ACCEL_X_MIN = -288;
float ACCEL_X_MAX = 260;
float ACCEL_Y_MIN = -282;
float ACCEL_Y_MAX = 281;
float ACCEL_Z_MIN = -290;
float ACCEL_Z_MAX = 310;

// Magnetometer (standard calibration mode)
// "magn x,y,z (min/max) = X_MIN/X_MAX  Y_MIN/Y_MAX  Z_MIN/Z_MAX"
float MAGN_X_MIN = -600;
float MAGN_X_MAX = 600;
float MAGN_Y_MIN = -600;
float MAGN_Y_MAX = 600;
float MAGN_Z_MIN = -600;
float MAGN_Z_MAX = 600;

// Magnetometer (extended calibration mode)
// Set to true to use extended magnetometer calibration (compensates hard & soft iron error
s)
boolean CALIBRATION__MAGN_USE_EXTENDED = false;
float magn_ellipsoid_center[3] = {0, 0, 0};
float magn_ellipsoid_transform[3][3] = {{0, 0, 0}, {0, 0, 0}, {0, 0, 0}};

// Gyroscope
// "gyro x,y,z (current/average) = .../OFFSET_X  .../OFFSET_Y  .../OFFSET_Z
float GYRO_AVERAGE_OFFSET_X = 0.02;
float GYRO_AVERAGE_OFFSET_Y = 0.0;
float GYRO_AVERAGE_OFFSET_Z = 0.0;

/*
// Calibration example:

// "accel x,y,z (min/max) = -277.00/264.00  -256.00/278.00  -299.00/235.00"
float ACCEL_X_MIN = -277;
float ACCEL_X_MAX = 264;
float ACCEL_Y_MIN = -256;
float ACCEL_Y_MAX = 278;
float ACCEL_Z_MIN = -299;
float ACCEL_Z_MAX = 235;

// "magn x,y,z (min/max) = -511.00/581.00  -516.00/568.00  -489.00/486.00"
float MAGN_X_MIN = -511;
float MAGN_X_MAX = 581;
float MAGN_Y_MIN = -516;
float MAGN_Y_MAX = 568;
float MAGN_Z_MIN = -489;
float MAGN_Z_MAX = 486;

// Extended magn
boolean CALIBRATION__MAGN_USE_EXTENDED = true;
float magn_ellipsoid_center[3] = {91.5, -13.5, -48.1};
float magn_ellipsoid_transform[3][3] = {{0.902, -0.00354, 0.000636}, {-0.00354, 0.9, -0.005
99}, {0.000636, -0.00599, 1}};

// Extended magn (with Sennheiser HD 485 headphones)
//boolean CALIBRATION__MAGN_USE_EXTENDED = true;
//float magn_ellipsoid_center[3] = {72.3360, 23.0954, 53.6261};
//float magn_ellipsoid_transform[3][3] = {{0.879685, 0.000540833, -0.0106054}, {0.000540833
, 0.891086, -0.0130338}, {-0.0106054, -0.0130338, 0.997494}};

//"gyro x,y,z (current/average) = -40.00/-42.05  98.00/96.20  -18.00/-18.36"
float GYRO_AVERAGE_OFFSET_X = -42.05;
float GYRO_AVERAGE_OFFSET_Y = 96.20;
float GYRO_AVERAGE_OFFSET_Z = -18.36;
*/


// DEBUG OPTIONS
/*****************************************************************/
// When set to true, gyro drift correction will not be applied
boolean DEBUG__NO_DRIFT_CORRECTION = false;
// Print elapsed time after each I/O loop
#define DEBUG__PRINT_LOOP_TIME false

#define DEBUG__PRINT_LOOP_TIME_2 false

#define DEBUG__ADD_LOOP_DELAY false

#define DEBUG__LOOP_DELAY 1
// Set to true to enable auto-calibration features of the M0 (does not apply to magnetomete
rs)
#define DEBUG__USE_DMP_M0 false
// Set to true to disable the use of the DCM algorithm
#define DEBUG__USE_ONLY_DMP_M0 false

#define DEBUG__ENABLE_FIFO_M0 false

#define DEBUG__ENABLE_INTERRUPT_M0 false

#define DEBUG__USE_DEFAULT_GYRO_FSR_M0 false

#define DEBUG__USE_DEFAULT_ACCEL_FSR_M0 false
```

```
/*************************************************************/
/***************** END OF USER SETUP AREA!  *******************/
/*************************************************************/









// Check if hardware version code is defined
#ifndef HW__VERSION_CODE
  // Generate compile error
  #error YOU HAVE TO SELECT THE HARDWARE YOU ARE USING! See "HARDWARE OPTIONS" in "USER SET
UP AREA" at top of Razor_AHRS.ino!
#endif

#if HW__VERSION_CODE == 14001
// MPU-9250 Digital Motion Processing (DMP) Library
#include <SparkFunMPU9250-DMP.h>

// Danger - don't change unless using a different platform
#define MPU9250_INT_PIN 4
#define SD_CHIP_SELECT_PIN 38
#define MPU9250_INT_ACTIVE LOW

MPU9250_DMP imu; // Create an instance of the MPU9250_DMP class

#if DEBUG__USE_ONLY_DMP_M0 == true
#undef DEBUG__USE_DMP_M0
#define DEBUG__USE_DMP_M0 true
#endif // DEBUG__USE_ONLY_DMP_M0

#if DEBUG__USE_DMP_M0 == true
#undef DEBUG__ENABLE_FIFO_M0
#define DEBUG__ENABLE_FIFO_M0 true
#endif // DEBUG__USE_DMP_M0

#if DEBUG__USE_ONLY_DMP_M0 == true
float initialmagyaw = -10000;
float initialimuyaw = -10000;
#endif // DEBUG__USE_ONLY_DMP_M0
#else
#include <Wire.h>
#endif // HW__VERSION_CODE

#define GRAVITY 256.0f // "1G reference" used for DCM filter and accelerometer calibration

// Sensor calibration scale and offset values
float ACCEL_X_OFFSET = ((ACCEL_X_MIN + ACCEL_X_MAX) / 2.0f);
float ACCEL_Y_OFFSET = ((ACCEL_Y_MIN + ACCEL_Y_MAX) / 2.0f);
float ACCEL_Z_OFFSET = ((ACCEL_Z_MIN + ACCEL_Z_MAX) / 2.0f);
float ACCEL_X_SCALE = (GRAVITY / (ACCEL_X_MAX - ACCEL_X_OFFSET));
float ACCEL_Y_SCALE = (GRAVITY / (ACCEL_Y_MAX - ACCEL_Y_OFFSET));
float ACCEL_Z_SCALE = (GRAVITY / (ACCEL_Z_MAX - ACCEL_Z_OFFSET));

float MAGN_X_OFFSET = ((MAGN_X_MIN + MAGN_X_MAX) / 2.0f);
float MAGN_Y_OFFSET = ((MAGN_Y_MIN + MAGN_Y_MAX) / 2.0f);
float MAGN_Z_OFFSET = ((MAGN_Z_MIN + MAGN_Z_MAX) / 2.0f);
float MAGN_X_SCALE = (100.0f / (MAGN_X_MAX - MAGN_X_OFFSET));
float MAGN_Y_SCALE = (100.0f / (MAGN_Y_MAX - MAGN_Y_OFFSET));
float MAGN_Z_SCALE = (100.0f / (MAGN_Z_MAX - MAGN_Z_OFFSET));
```

```
#if HW__VERSION_CODE == 14001
#define GYRO_SCALED_RAD(x) (x) // Calculate the scaled gyro readings in radians per second
#else
// Gain for gyroscope (ITG-3200)
#define GYRO_GAIN 0.06957 // Same gain on all axes
#define GYRO_SCALED_RAD(x) (x * TO_RAD(GYRO_GAIN)) // Calculate the scaled gyro readings in
 radians per second
#endif // HW__VERSION_CODE

// DCM parameters
#define Kp_ROLLPITCH 0.02f
#define Ki_ROLLPITCH 0.00002f
#define Kp_YAW 1.2f
#define Ki_YAW 0.00002f

// Stuff
#define STATUS_LED_PIN 13  // Pin number of status LED
#define TO_RAD(x) (x * 0.01745329252)  // *pi/180
#define TO_DEG(x) (x * 57.2957795131)  // *180/pi

// Sensor variables
float accel[3] = {0, 0, 0}; // Actually stores the NEGATED acceleration (equals gravity, if
 board not moving).
float accel_min[3] = {0, 0, 0};
float accel_max[3] = {0, 0, 0};

float magnetom[3] = {0, 0, 0};
float magnetom_min[3] = {0, 0, 0};
float magnetom_max[3] = {0, 0, 0};
float magnetom_tmp[3] = {0, 0, 0};

float gyro[3] = {0, 0, 0};
float gyro_average[3] = {0, 0, 0};
int gyro_num_samples = 0;

// DCM variables
float MAG_Heading = 0;
float Accel_Vector[3]= {0, 0, 0}; // Store the acceleration in a vector
float Gyro_Vector[3]= {0, 0, 0}; // Store the gyros turn rate in a vector
float Omega_Vector[3]= {0, 0, 0}; // Corrected Gyro_Vector data
float Omega_P[3]= {0, 0, 0}; // Omega Proportional correction
float Omega_I[3]= {0, 0, 0}; // Omega Integrator
float Omega[3]= {0, 0, 0};
float errorRollPitch[3] = {0, 0, 0};
float errorYaw[3] = {0, 0, 0};
float DCM_Matrix[3][3] = {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}};
float Update_Matrix[3][3] = {{0, 1, 2}, {3, 4, 5}, {6, 7, 8}};
float Temporary_Matrix[3][3] = {{0, 0, 0}, {0, 0, 0}, {0, 0, 0}};

// Euler angles
float yaw = 0;
float pitch = 0;
float roll = 0;

// DCM timing in the main loop
unsigned long timestamp = 0;
unsigned long timestamp_old = 0;
float G_Dt = 0; // Integration time for DCM algorithm

// More output-state variables
boolean output_stream_on = false;
boolean output_single_on = false;
int curr_calibration_sensor = 0;
boolean reset_calibration_session_flag = true;
int num_accel_errors = 0;
int num_magn_errors = 0;
int num_gyro_errors = 0;
int num_math_errors = 0;
```

```
void read_sensors() {
#if HW__VERSION_CODE == 14001
  loop_imu();
#else
  Read_Gyro(); // Read gyroscope
  Read_Accel(); // Read accelerometer
  Read_Magn(); // Read magnetometer
#endif // HW__VERSION_CODE
}

// Should be called after every #ca calibration command
void recalculateAccelCalibration() {
  ACCEL_X_OFFSET = ((ACCEL_X_MIN + ACCEL_X_MAX) / 2.0f);
  ACCEL_Y_OFFSET = ((ACCEL_Y_MIN + ACCEL_Y_MAX) / 2.0f);
  ACCEL_Z_OFFSET = ((ACCEL_Z_MIN + ACCEL_Z_MAX) / 2.0f);
  ACCEL_X_SCALE = (GRAVITY / (ACCEL_X_MAX - ACCEL_X_OFFSET));
  ACCEL_Y_SCALE = (GRAVITY / (ACCEL_Y_MAX - ACCEL_Y_OFFSET));
  ACCEL_Z_SCALE = (GRAVITY / (ACCEL_Z_MAX - ACCEL_Z_OFFSET));
}

// Should be called after every #cm calibration command
void recalculateMagnCalibration() {
  MAGN_X_OFFSET = ((MAGN_X_MIN + MAGN_X_MAX) / 2.0f);
  MAGN_Y_OFFSET = ((MAGN_Y_MIN + MAGN_Y_MAX) / 2.0f);
  MAGN_Z_OFFSET = ((MAGN_Z_MIN + MAGN_Z_MAX) / 2.0f);
  MAGN_X_SCALE = (100.0f / (MAGN_X_MAX - MAGN_X_OFFSET));
  MAGN_Y_SCALE = (100.0f / (MAGN_Y_MAX - MAGN_Y_OFFSET));
  MAGN_Z_SCALE = (100.0f / (MAGN_Z_MAX - MAGN_Z_OFFSET));
}

// Apply calibration to raw sensor readings
void compensate_sensor_errors() {
    // Compensate accelerometer error
    accel[0] = (accel[0] - ACCEL_X_OFFSET) * ACCEL_X_SCALE;
    accel[1] = (accel[1] - ACCEL_Y_OFFSET) * ACCEL_Y_SCALE;
    accel[2] = (accel[2] - ACCEL_Z_OFFSET) * ACCEL_Z_SCALE;

    // Compensate magnetometer error
    if (CALIBRATION__MAGN_USE_EXTENDED)
    {
      for (int i = 0; i < 3; i++)
        magnetom_tmp[i] = magnetom[i] - magn_ellipsoid_center[i];
      Matrix_Vector_Multiply(magn_ellipsoid_transform, magnetom_tmp, magnetom);
    }
    else
    {
      magnetom[0] = (magnetom[0] - MAGN_X_OFFSET) * MAGN_X_SCALE;
      magnetom[1] = (magnetom[1] - MAGN_Y_OFFSET) * MAGN_Y_SCALE;
      magnetom[2] = (magnetom[2] - MAGN_Z_OFFSET) * MAGN_Z_SCALE;
    }

    // Compensate gyroscope error
    gyro[0] -= GYRO_AVERAGE_OFFSET_X;
    gyro[1] -= GYRO_AVERAGE_OFFSET_Y;
    gyro[2] -= GYRO_AVERAGE_OFFSET_Z;
}

// Read every sensor and record a time stamp
// Init DCM with unfiltered orientation
// TODO re-init global vars?
void reset_sensor_fusion() {
  float temp1[3] = {0, 0, 0};
  float temp2[3] = {0, 0, 0};
  float xAxis[3] = {1, 0, 0};

  read_sensors();
  compensate_sensor_errors();
  timestamp = millis();

  // GET PITCH
  // Using y-z-plane-component/x-component of gravity vector
  pitch = -atan2(accel[0], sqrt(accel[1] * accel[1] + accel[2] * accel[2]));

  // GET ROLL
  // Compensate pitch of gravity vector
  Vector_Cross_Product(temp1, accel, xAxis);
  Vector_Cross_Product(temp2, xAxis, temp1);
  // Normally using x-z-plane-component/y-component of compensated gravity vector
  // roll = atan2(temp2[1], sqrt(temp2[0] * temp2[0] + temp2[2] * temp2[2]));
  // Since we compensated for pitch, x-z-plane-component equals z-component:
  roll = atan2(temp2[1], temp2[2]);

  // GET YAW
  Compass_Heading();
  yaw = MAG_Heading;

  // Init rotation matrix
  init_rotation_matrix(DCM_Matrix, yaw, pitch, roll);
}

// Reset calibration session if reset_calibration_session_flag is set
void check_reset_calibration_session()
{
  // Raw sensor values have to be read already, but no error compensation applied

  // Reset this calibration session?
  if (!reset_calibration_session_flag) return;

  // Reset acc and mag calibration variables
  for (int i = 0; i < 3; i++) {
    accel_min[i] = accel_max[i] = accel[i];
    magnetom_min[i] = magnetom_max[i] = magnetom[i];
  }

  // Reset gyro calibration variables
  gyro_num_samples = 0;  // Reset gyro calibration averaging
  gyro_average[0] = gyro_average[1] = gyro_average[2] = 0.0f;

  reset_calibration_session_flag = false;
}

void turn_output_stream_on()
{
  output_stream_on = true;
  digitalWrite(STATUS_LED_PIN, HIGH);
}

void turn_output_stream_off()
{
  output_stream_on = false;
  digitalWrite(STATUS_LED_PIN, LOW);
}

// Blocks until another byte is available on serial port
char readChar()
{
  while (LOG_PORT.available() < 1) { } // Block
  return LOG_PORT.read();
}

void setup()
{
  // Init serial output
  LOG_PORT.begin(OUTPUT__BAUD_RATE);
  // Init status LED
  pinMode (STATUS_LED_PIN, OUTPUT);
  digitalWrite(STATUS_LED_PIN, LOW);

  // Init sensors
  delay(50);  // Give sensors enough time to start
#if HW__VERSION_CODE == 14001
#if DEBUG__ENABLE_INTERRUPT_M0 == true
  // Set up MPU-9250 interrupt input (active-low)
  pinMode(MPU9250_INT_PIN, INPUT_PULLUP);
#endif // DEBUG__ENABLE_INTERRUPT_M0
  initIMU();
#else
  I2C_Init();
  Accel_Init();
  Magn_Init();
  Gyro_Init();
#endif // HW__VERSION_CODE

  // Read sensors, init DCM algorithm
#if HW__VERSION_CODE == 14001
  delay(400);  // Give sensors enough time to collect data
#else
  delay(20);  // Give sensors enough time to collect data
#endif // HW__VERSION_CODE
  reset_sensor_fusion();

  // Init output
#if (OUTPUT__HAS_RN_BLUETOOTH == true) || (OUTPUT__STARTUP_STREAM_ON == false)
  turn_output_stream_off();
#else
  turn_output_stream_on();
#endif
}

// Main loop
void loop()
{
  // Read incoming control messages
 #if HW__VERSION_CODE == 14001
  // Compatibility fix : if bytes are sent 1 by 1 without being read, available() might nev
er return more than 1...
  // Therefore, we need to read bytes 1 by 1 and the command byte needs to be a blocking re
ad...
  if (LOG_PORT.available() >= 1)
  {
    if (LOG_PORT.read() == '#') // Start of new control message
    {
      int command = readChar(); // Commands
#else
  if (LOG_PORT.available() >= 2)
  {
    if (LOG_PORT.read() == '#') // Start of new control message
    {
      int command = LOG_PORT.read(); // Commands
#endif // HW__VERSION_CODE
      if (command == 'f') // request one output _f_rame
        output_single_on = true;
      else if (command == 's') // _s_ynch request
      {
        // Read ID
        byte id[2];
        id[0] = readChar();
        id[1] = readChar();

        // Reply with synch message
        LOG_PORT.print("#SYNCH");
        LOG_PORT.write(id, 2);
        LOG_PORT.println();
      }
      else if (command == 'o') // Set _o_utput mode
      {
        char output_param = readChar();
        if (output_param == 'n')  // Calibrate _n_ext sensor
        {
          curr_calibration_sensor = (curr_calibration_sensor + 1) % 3;
          reset_calibration_session_flag = true;
        }
        else if (output_param == 't') // Output angles as _t_ext
        {
          output_mode = OUTPUT__MODE_ANGLES;
          output_format = OUTPUT__FORMAT_TEXT;
        }
        else if (output_param == 'b') // Output angles in _b_inary format
        {
          output_mode = OUTPUT__MODE_ANGLES;
          output_format = OUTPUT__FORMAT_BINARY;
        }
        else if (output_param == 'c') // Go to _c_alibration mode
        {
          output_mode = OUTPUT__MODE_CALIBRATE_SENSORS;
          reset_calibration_session_flag = true;
        }
        else if (output_param == 'x') // Output angles + accel + rot. vel as te_x_t
        {
          output_mode = OUTPUT__MODE_ANGLES_AG_SENSORS;
          output_format = OUTPUT__FORMAT_TEXT;
        }
        else if (output_param == 's') // Output _s_ensor values
        {
          char values_param = readChar();
          char format_param = readChar();
          if (values_param == 'r')  // Output _r_aw sensor values
            output_mode = OUTPUT__MODE_SENSORS_RAW;
          else if (values_param == 'c')  // Output _c_alibrated sensor values
            output_mode = OUTPUT__MODE_SENSORS_CALIB;
          else if (values_param == 'b')  // Output _b_oth sensor values (raw and calibrated
)
            output_mode = OUTPUT__MODE_SENSORS_BOTH;

          if (format_param == 't') // Output values as _t_text
            output_format = OUTPUT__FORMAT_TEXT;
          else if (format_param == 'b') // Output values in _b_inary format
            output_format = OUTPUT__FORMAT_BINARY;
        }
        else if (output_param == '0') // Disable continuous streaming output
        {
          turn_output_stream_off();
          reset_calibration_session_flag = true;
        }
        else if (output_param == '1') // Enable continuous streaming output
        {
          reset_calibration_session_flag = true;
          turn_output_stream_on();
        }
        else if (output_param == 'e') // _e_rror output settings
        {
          char error_param = readChar();
          if (error_param == '0') output_errors = false;
          else if (error_param == '1') output_errors = true;
          else if (error_param == 'c') // get error _c_ount
          {
            LOG_PORT.print("#AMG-ERR:");
            LOG_PORT.print(num_accel_errors); LOG_PORT.print(",");
            LOG_PORT.print(num_magn_errors); LOG_PORT.print(",");
            LOG_PORT.println(num_gyro_errors);
          }
          else if (error_param == 'm') // get _m_ath error count
          {
            LOG_PORT.print("#MATH-ERR:");
            LOG_PORT.println(num_math_errors);
          }
        }
      }
      else if (command == 'p') // Set _p_rint calibration values
      {
         LOG_PORT.print("ACCEL_X_MIN:");LOG_PORT.println(ACCEL_X_MIN);
         LOG_PORT.print("ACCEL_X_MAX:");LOG_PORT.println(ACCEL_X_MAX);
         LOG_PORT.print("ACCEL_Y_MIN:");LOG_PORT.println(ACCEL_Y_MIN);
         LOG_PORT.print("ACCEL_Y_MAX:");LOG_PORT.println(ACCEL_Y_MAX);
         LOG_PORT.print("ACCEL_Z_MIN:");LOG_PORT.println(ACCEL_Z_MIN);
         LOG_PORT.print("ACCEL_Z_MAX:");LOG_PORT.println(ACCEL_Z_MAX);
         LOG_PORT.println("");
         LOG_PORT.print("MAGN_X_MIN:");LOG_PORT.println(MAGN_X_MIN);
         LOG_PORT.print("MAGN_X_MAX:");LOG_PORT.println(MAGN_X_MAX);
         LOG_PORT.print("MAGN_Y_MIN:");LOG_PORT.println(MAGN_Y_MIN);
         LOG_PORT.print("MAGN_Y_MAX:");LOG_PORT.println(MAGN_Y_MAX);
         LOG_PORT.print("MAGN_Z_MIN:");LOG_PORT.println(MAGN_Z_MIN);
         LOG_PORT.print("MAGN_Z_MAX:");LOG_PORT.println(MAGN_Z_MAX);
         LOG_PORT.println("");
         LOG_PORT.print("MAGN_USE_EXTENDED:");
         if (CALIBRATION__MAGN_USE_EXTENDED)
           LOG_PORT.println("true");
         else
           LOG_PORT.println("false");
         LOG_PORT.print("magn_ellipsoid_center:[");LOG_PORT.print(magn_ellipsoid_center[0],
4);LOG_PORT.print(",");
         LOG_PORT.print(magn_ellipsoid_center[1],4);LOG_PORT.print(",");
         LOG_PORT.print(magn_ellipsoid_center[2],4);LOG_PORT.println("]");
         LOG_PORT.print("magn_ellipsoid_transform:[");
         for(int i = 0; i < 3; i++){
           LOG_PORT.print("[");
           for(int j = 0; j < 3; j++){
             LOG_PORT.print(magn_ellipsoid_transform[i][j],7);
             if (j < 2) LOG_PORT.print(",");
           }
           LOG_PORT.print("]");
           if (i < 2) LOG_PORT.print(",");
         }
         LOG_PORT.println("]");
         LOG_PORT.println("");
         LOG_PORT.print("GYRO_AVERAGE_OFFSET_X:");LOG_PORT.println(GYRO_AVERAGE_OFFSET_X);
         LOG_PORT.print("GYRO_AVERAGE_OFFSET_Y:");LOG_PORT.println(GYRO_AVERAGE_OFFSET_Y);
         LOG_PORT.print("GYRO_AVERAGE_OFFSET_Z:");LOG_PORT.println(GYRO_AVERAGE_OFFSET_Z);
      }
          else if (command == 'c') // Set _i_nput mode
      {
        char input_param = readChar();
        if (input_param == 'a')  // Calibrate _a_ccelerometer
        {
          char axis_param = readChar();
          char type_param = readChar();
          float value_param = LOG_PORT.parseFloat();
          if (axis_param == 'x')  // x value
          {
            if (type_param == 'm')
              ACCEL_X_MIN = value_param;
            else if (type_param == 'M')
              ACCEL_X_MAX = value_param;
          }
          else if (axis_param == 'y')  // y value
          {
            if (type_param == 'm')
              ACCEL_Y_MIN = value_param;
            else if (type_param == 'M')
              ACCEL_Y_MAX = value_param;
          }
          else if (axis_param == 'z')  // z value
          {
            if (type_param == 'm')
              ACCEL_Z_MIN = value_param;
            else if (type_param == 'M')
              ACCEL_Z_MAX = value_param;
          }
          recalculateAccelCalibration();
        }
        else if (input_param == 'm')  // Calibrate _m_agnetometer (basic)
        {
          //disable extended magnetometer calibration
          CALIBRATION__MAGN_USE_EXTENDED = false;
          char axis_param = readChar();
          char type_param = readChar();
          float value_param = LOG_PORT.parseFloat();
          if (axis_param == 'x')  // x value
          {
            if (type_param == 'm')
              MAGN_X_MIN = value_param;
            else if (type_param == 'M')
              MAGN_X_MAX = value_param;
          }
          else if (axis_param == 'y')  // y value
          {
            if (type_param == 'm')
              MAGN_Y_MIN = value_param;
            else if (type_param == 'M')
              MAGN_Y_MAX = value_param;
          }
          else if (axis_param == 'z')  // z value
          {
            if (type_param == 'm')
              MAGN_Z_MIN = value_param;
            else if (type_param == 'M')
              MAGN_Z_MAX = value_param;
          }
          recalculateMagnCalibration();
        }
        else if (input_param == 'c')  // Calibrate magnetometerellipsoid_c_enter (extended)
        {
          //enable extended magnetometer calibration
          CALIBRATION__MAGN_USE_EXTENDED = true;
          char axis_param = readChar();
          float value_param = LOG_PORT.parseFloat();
          if (axis_param == 'x')  // x value
              magn_ellipsoid_center[0] = value_param;
          else if (axis_param == 'y')  // y value
              magn_ellipsoid_center[1] = value_param;
          else if (axis_param == 'z')  // z value
              magn_ellipsoid_center[2] = value_param;
        }
        else if (input_param == 't')  // Calibrate magnetometerellipsoid_t_ransform (extend
ed)
        {
          //enable extended magnetometer calibration
          CALIBRATION__MAGN_USE_EXTENDED = true;
          char axis_param = readChar();
          char type_param = readChar();
          float value_param = LOG_PORT.parseFloat();
          if (axis_param == 'x')  // x value
          {
            if (type_param == 'X')
              magn_ellipsoid_transform[0][0] = value_param;
            else if (type_param == 'Y')
```

```
              magn_ellipsoid_transform[0][1] = value_param;
            else if (type_param == 'Z')
              magn_ellipsoid_transform[0][2] = value_param;
          }
          else if (axis_param == 'y')  // y value
          {
            if (type_param == 'X')
              magn_ellipsoid_transform[1][0] = value_param;
            else if (type_param == 'Y')
              magn_ellipsoid_transform[1][1] = value_param;
            else if (type_param == 'Z')
              magn_ellipsoid_transform[1][2] = value_param;
          }
          else if (axis_param == 'z')  // z value
          {
            if (type_param == 'X')
              magn_ellipsoid_transform[2][0] = value_param;
            else if (type_param == 'Y')
              magn_ellipsoid_transform[2][1] = value_param;
            else if (type_param == 'Z')
              magn_ellipsoid_transform[2][2] = value_param;
          }
        }
        else if (input_param == 'g')  // Calibrate _g_yro
        {
          char axis_param = readChar();
          float value_param = LOG_PORT.parseFloat();
          if (axis_param == 'x')  // x value
              GYRO_AVERAGE_OFFSET_X = value_param;
          else if (axis_param == 'y')  // y value
              GYRO_AVERAGE_OFFSET_Y = value_param;
          else if (axis_param == 'z')  // z value
              GYRO_AVERAGE_OFFSET_Z = value_param;
        }
      }
          else if (command == 'I') // Toggle _i_nertial-only mode for yaw computation
          {
                DEBUG__NO_DRIFT_CORRECTION = !DEBUG__NO_DRIFT_CORRECTION;
#if DEBUG__USE_ONLY_DMP_M0 == true
                // Update reference for yaw...
                initialmagyaw = -MAG_Heading;
                initialimuyaw = imu.yaw*PI/180.0f;
#endif // DEBUG__USE_ONLY_DMP_M0
          }
#if OUTPUT__HAS_RN_BLUETOOTH == true
      // Read messages from bluetooth module
      // For this to work, the connect/disconnect message prefix of the module has to be se
t to "#".
      else if (command == 'C') // Bluetooth "#CONNECT" message (does the same as "#o1")
        turn_output_stream_on();
      else if (command == 'D') // Bluetooth "#DISCONNECT" message (does the same as "#o0")
        turn_output_stream_off();
#endif // OUTPUT__HAS_RN_BLUETOOTH == true
    }
    else
    { } // Skip character
  }

  // Time to read the sensors again?
  if((millis() - timestamp) >= OUTPUT__DATA_INTERVAL)
  {
#if DEBUG__PRINT_LOOP_TIME_2 == true
    LOG_PORT.print("loop time (ms) = ");
    LOG_PORT.println(millis() - timestamp);
#endif // DEBUG__PRINT_LOOP_TIME_2
    timestamp_old = timestamp;
    timestamp = millis();
    if (timestamp > timestamp_old)
```

```
      G_Dt = (float) (timestamp - timestamp_old) / 1000.0f; // Real time of loop run. We us
e this on the DCM algorithm (gyro integration time)
    else G_Dt = 0;

    // Update sensor readings
    read_sensors();

    if (output_mode == OUTPUT__MODE_CALIBRATE_SENSORS)  // We're in calibration mode
    {
      check_reset_calibration_session();  // Check if this session needs a reset
      if (output_stream_on || output_single_on) output_calibration(curr_calibration_sensor)
;
    }
    else if (output_mode == OUTPUT__MODE_ANGLES)  // Output angles
    {
      // Apply sensor calibration
      compensate_sensor_errors();

#if DEBUG__USE_ONLY_DMP_M0 == true
          Euler_angles_only_DMP_M0();
#else
      // Run DCM algorithm
      Compass_Heading(); // Calculate magnetic heading
      Matrix_update();
      Normalize();
      Drift_correction();
      Euler_angles();
#endif // DEBUG__USE_ONLY_DMP_M0

      if (output_stream_on || output_single_on) output_angles();
    }
    else if (output_mode == OUTPUT__MODE_ANGLES_AG_SENSORS)  // Output angles + accel + rot
. vel
    {
      // Apply sensor calibration
      compensate_sensor_errors();

#if DEBUG__USE_ONLY_DMP_M0 == true
          Euler_angles_only_DMP_M0();
#else
      // Run DCM algorithm
      Compass_Heading(); // Calculate magnetic heading
      Matrix_update();
      Normalize();
      Drift_correction();
      Euler_angles();
#endif // DEBUG__USE_ONLY_DMP_M0

      if (output_stream_on || output_single_on) output_both_angles_and_sensors_text();
    }
    else  // Output sensor values
    {
      if (output_stream_on || output_single_on) output_sensors();
    }

    output_single_on = false;

#if DEBUG__PRINT_LOOP_TIME == true
    LOG_PORT.print("loop time (ms) = ");
    LOG_PORT.println(millis() - timestamp);
#endif
  }
#if DEBUG__PRINT_LOOP_TIME == true
  else
  {
    LOG_PORT.println("waiting...");
  }
#else
```

```
#if DEBUG__ADD_LOOP_DELAY == true
  else
  {
        delay(DEBUG__LOOP_DELAY);
  }
#endif // DEBUG__ADD_LOOP_DELAY
#endif
}
```