//This code is used for the Co-Coach, a device developed by Tess Ernest, Ward de Groot, Willem Schaeffers en Willem Gelden from the University of Technology Eindhoven at the faculty of Industrial Design.

//This code code is based on the MPU6050_DPM6 example from the MPU 6050 library from Jeff Rowberg.

//06/09/2016

// I2C device class (I2Cdev) demonstration Arduino sketch for MPU6050 class using DMP (MotionApps v2.0)

// 6/21/2012 by Jeff Rowberg <jeff@rowberg.net>

// Updates should (hopefully) always be available at https://github.com/jrowberg/i2cdevlib

//

// Changelog:

// 2013-05-08 - added seamless Fastwire support

// - added note about gyro calibration

// 2012-06-21 - added note about Arduino 1.0.1 + Leonardo compatibility error

// 2012-06-20 - improved FIFO overflow handling and simplified read process

// 2012-06-19 - completely rearranged DMP initialization code and simplification

// 2012-06-13 - pull gyro and accel data from FIFO packet instead of reading directly

// 2012-06-09 - fix broken FIFO read sequence and change interrupt detection to RISING

// 2012-06-05 - add gravity-compensated initial reference frame acceleration output

// - add 3D math helper file to DMP6 example sketch

// - add Euler output and Yaw/Pitch/Roll output formats

// 2012-06-04 - remove accel offset clearing for better results (thanks Sungon Lee)

// 2012-06-01 - fixed gyro sensitivity to be 2000 deg/sec instead of 250

// 2012-05-30 - basic DMP initialization working

/* ==

```
// I2Cdev and MPU6050 must be installed as libraries, or else the .cpp/.h files
// for both classes must be in the include path of your project
#include "I2Cdev.h"

#include "MPU6050_6Axis_MotionApps20.h"
//#include "MPU6050.h" // not necessary if using MotionApps include file

// Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE implementation
// is used in I2Cdev.h
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
#include "Wire.h"
#endif
```

```
// class default I2C address is 0x68
// specific I2C addresses may be passed as a parameter here
// AD0 low = 0x68 (default for SparkFun breakout and InvenSense evaluation board)
// AD0 high = 0x69
MPU6050 mpu;
//MPU6050 mpu(0x69); // <-- use for AD0 high

/* =========================================================================
   NOTE: In addition to connection 3.3v, GND, SDA, and SCL, this sketch
   depends on the MPU-6050's INT pin being connected to the Arduino's
   external interrupt #0 pin. On the Arduino Uno and Mega 2560, this is
   digital I/O pin 2.
 * ========================================================================= */

/* =========================================================================
   NOTE: Arduino v1.0.1 with the Leonardo board generates a compile error
   when using Serial.write(buf, len). The Teapot output uses this method.
   The solution requires a modification to the Arduino USBAPI.h file, which
   is fortunately simple, but annoying. This will be fixed in the next IDE
   release. For more info, see these links:

   http://arduino.cc/forum/index.php/topic,109987.0.html
   http://code.google.com/p/arduino/issues/detail?id=958
 * ========================================================================= */


// uncomment "OUTPUT_READABLE_YAWPITCHROLL" if you want to see the yaw/
// pitch/roll angles (in degrees) calculated from the quaternions coming
// from the FIFO. Note this also requires gravity vector calculations.
// Also note that yaw/pitch/roll angles suffer from gimbal lock (for
```

```
// more info, see: http://en.wikipedia.org/wiki/Gimbal_lock)
#define OUTPUT_READABLE_YAWPITCHROLL

// MPU control/status vars
bool dmpReady = false;  // set true if DMP init was successful
uint8_t mpuIntStatus;   // holds actual interrupt status byte from MPU
uint8_t devStatus;      // return status after each device operation (0 = success, !0 = error)
uint16_t packetSize;    // expected DMP packet size (default is 42 bytes)
uint16_t fifoCount;     // count of all bytes currently in FIFO
uint8_t fifoBuffer[64]; // FIFO storage buffer
boolean increaseAllowed = true;

//digital pins for the buttons
int button1 = 3;
int button2 = 4;
int button3 = 5;

//state of the buttons
int buttonState1 = 0;
int buttonState2 = 0;
int buttonState3 = 0;

// we need this to prevent complications concerning the buttons
long lastDebounceTime2 = 0;
long lastDebounceTime3 = 0;
long debounceDelay = 250;

//statistics
int stat1 = 0;
int stat2 = 0;
int stat3 = 0;
```

```
int stat4 = 0;

// orientation/motion vars
Quaternion q;           // [w, x, y, z]        quaternion container
VectorInt16 aa;         // [x, y, z]           accel sensor measurements
VectorInt16 aaReal;     // [x, y, z]           gravity-free accel sensor measurements
VectorInt16 aaWorld;    // [x, y, z]           world-frame accel sensor measurements
VectorFloat gravity;    // [x, y, z]           gravity vector
float euler[3];         // [psi, theta, phi]    Euler angle container
float ypr[3];           // [yaw, pitch, roll]   yaw/pitch/roll container and gravity vector

// ================================================================
// ===               INTERRUPT DETECTION ROUTINE                ===
// ================================================================

volatile bool mpuInterrupt = false;     // indicates whether MPU interrupt pin has gone high
void dmpDataReady() {
  mpuInterrupt = true;
}
// ================================================================
// ===                      INITIAL SETUP                       ===
// ================================================================

void setup() {
  //set buttons as input
  pinMode(button1, INPUT);
  pinMode(button2, INPUT);
  pinMode(button3, INPUT);

  // join I2C bus (I2Cdev library doesn't do this automatically)
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
```

```
  Wire.begin();
  TWBR = 24; // 400kHz I2C clock (200kHz if CPU is 8MHz)
#elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
  Fastwire::setup(400, true);
#endif
  // initialize serial communication
  // (115200 chosen because it is required for Teapot Demo output, but it's
  // really up to you depending on your project)
  Serial.begin(115200);
  //while (!Serial); // wait for Leonardo enumeration, others continue immediately

  // NOTE: 8MHz or slower host processors, like the Teensy @ 3.3v or Ardunio
  // Pro Mini running at 3.3v, cannot handle this baud rate reliably due to
  // the baud timing being too misaligned with processor ticks. You must use
  // 38400 or slower in these cases, or use some kind of external separate
  // crystal solution for the UART timer.

  // initialize device
  // Serial.println(F("Initializing I2C devices..."));
  mpu.initialize();
  // verify connection
  // Serial.println(F("Testing device connections..."));
  //Serial.println(mpu.testConnection() ? F("MPU6050 connection successful") : F("MPU6050
connection failed"));
  // wait for ready
  //Serial.println(F("\nSend any character to begin DMP programming and demo: "));
  //  while (Serial.available() && Serial.read()); // empty buffer
  //  while (!Serial.available());                 // wait for data
  //  while (Serial.available() && Serial.read()); // empty buffer again
  // load and configure the DMP
  //Serial.println(F("Initializing DMP..."));
```

```
 devStatus = mpu.dmpInitialize();
 // supply your own gyro offsets here, scaled for min sensitivity
 mpu.setXGyroOffset(220);
 mpu.setYGyroOffset(76);
 mpu.setZGyroOffset(-85);
 mpu.setZAccelOffset(1788); // 1688 factory default for my test chip
 // make sure it worked (returns 0 if so)
 if (devStatus == 0) {
  // turn on the DMP, now that it's ready
  //Serial.println(F("Enabling DMP..."));
  mpu.setDMPEnabled(true);
  // enable Arduino interrupt detection
  //Serial.println(F("Enabling interrupt detection (Arduino external interrupt 0)..."));
  attachInterrupt(0, dmpDataReady, RISING);
  mpuIntStatus = mpu.getIntStatus();
  // set our DMP Ready flag so the main loop() function knows it's okay to use it
  //Serial.println(F("DMP ready! Waiting for first interrupt..."));
  dmpReady = true;
  // get expected DMP packet size for later comparison
  packetSize = mpu.dmpGetFIFOPacketSize();
 } else {
  // ERROR!
  // 1 = initial memory load failed
  // 2 = DMP configuration updates failed
  // (if it's going to break, usually the code will be 1)
  //Serial.print(F("DMP Initialization failed (code "));
  //Serial.print(devStatus);
  //Serial.println(F(")"));
 }
}
// ================================================================
```

```
// ===                MAIN PROGRAM LOOP                ===
// ================================================================
void loop() {
  // state of the buttons
  buttonState1 = digitalRead(button1);
  buttonState2 = digitalRead(button2);
  buttonState3 = digitalRead(button3);
  // if programming failed, don't try to do anything
  if (!dmpReady) return;
  // wait for MPU interrupt or extra packet(s) available
  while (!mpuInterrupt && fifoCount < packetSize) {
    // other program behavior stuff here
    // if you are really paranoid you can frequently test in between other
    // stuff to see if mpuInterrupt is true, and if so, "break;" from the
    // while() loop to immediately process the MPU data

  }
  // reset interrupt flag and get INT_STATUS byte
  mpuInterrupt = false;
  mpuIntStatus = mpu.getIntStatus();
  // get current FIFO count
  fifoCount = mpu.getFIFOCount();
  // check for overflow (this should never happen unless our code is too inefficient)
  if ((mpuIntStatus & 0x10) || fifoCount == 1024) {
    // reset so we can continue cleanly
    mpu.resetFIFO();
    Serial.println(F("FIFO overflow!"));
    // otherwise, check for DMP data ready interrupt (this should happen frequently)
  } else if (mpuIntStatus & 0x02) {
    // wait for correct available data length, should be a VERY short wait
    while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();
```

```
    // read a packet from FIFO
    mpu.getFIFOBytes(fifoBuffer, packetSize);
    // track FIFO count here in case there is > 1 packet available
    // (this lets us immediately read more without waiting for an interrupt)
    fifoCount -= packetSize;
#ifdef OUTPUT_READABLE_YAWPITCHROLL
    // display Euler angles in degrees
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    mpu.dmpGetGravity(&gravity, &q);
    mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);

    if ( (millis() - lastDebounceTime2) > debounceDelay)  { //debounce button
      if (buttonState2 == HIGH) {
        stat1 = stat1 + 1; //if the button is pressed, add 1 to statistic 1
        Serial.print(stat1);//print statistics in serial port
        Serial.print(',');
        Serial.print(stat2);
        Serial.print(',');
        Serial.print(stat3);
        Serial.print(',');
        Serial.print(stat4);
        Serial.println(',');
        lastDebounceTime2 = millis();
      }
    }
    if ( (millis() - lastDebounceTime3) > debounceDelay) {
      if (buttonState3 == HIGH) {
        stat2 = stat2 + 1;
        Serial.print(stat1);
        Serial.print(',');
        Serial.print(stat2);
```

```
      Serial.print(',');
      Serial.print(stat3);
      Serial.print(',');
      Serial.print(stat4);
      Serial.println(',');
      lastDebounceTime3 = millis();
    }
   }
   if (increaseAllowed == true && buttonState1 == HIGH) {
    if (ypr[2] * 180 / M_PI > 25) {
      stat3 = stat3 + 1; // if button 1 is pressed and the divice is tilted 25 degrees to one side, add 1 to statistic 3
      increaseAllowed = false;
      Serial.print(stat1);
      Serial.print(',');
      Serial.print(stat2);
      Serial.print(',');
      Serial.print(stat3);
      Serial.print(',');
      Serial.print(stat4);
      Serial.println(',');
    }
    if (ypr[2] * 180 / M_PI < -25) {
      stat4 = stat4 + 1;
      increaseAllowed = false;
      Serial.print(stat1);
      Serial.print(',');
      Serial.print(stat2);
      Serial.print(',');
      Serial.print(stat3);
      Serial.print(',');
```

```
      Serial.print(stat4);
      Serial.println(',');
     }
    }

    // the device has to be in its neutral posistion again before statistics 3 and 4 can increase again
    else {
     if ((ypr[2] * 180 / M_PI < 25) && (ypr[2] * 180 / M_PI > -25)) {
      increaseAllowed = true;
     }
    }
#endif
  }
}
```