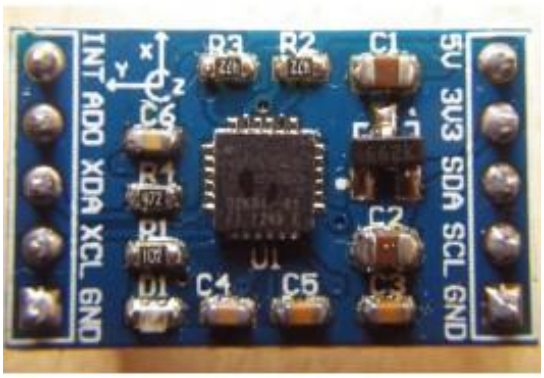
Module gyroscope et accéléromètre MPU6050 6 axes + Arduino



Composants utilisés

- 1 module gyroscope et accéléromètre MPU 6050
- 1 carte Arduino Uno
- Une petite plaquette de montage (prototype shield).
- Câbles de branchement mâle femelle Dupont

Principe de fonctionnement
Le gyroscope - accéléromètre MPU-6050 comporte 6 axes. (Fabricant : http://www.lctech-inc.com).
Sa puce MEMS est très précise avec une conversion analogique-digitale sur 16 bits simultanée sur chaque canal, et une interface I2C (400 kHz).

La lecture des mesures brutes de ce capteur est facile.
Le capteur contient un registre FIFO de 1024 octets que le micro-contrôleur Arduino peut lire, étant prévenu par un signal d'interruption.

Le module fonctionne en esclave sur le **bus I2C** vis à vis de l'Arduino (pins SDA, SLC) mais il peut aussi contrôler un autre dispositif en aval avec AUX-DA et AUX-CL, par exemple un magnétomètre 3 axes (mesure du champ magnétique terrestre) pour une orientation absolue dans l'espace (boussole).
Sa consommation est faible, 3.9 mA avec les 6 capteurs activés.

Le capteur possède un DMP (Digital Motion Processor) capable de faire des calculs rapides

directement sur la puce à partir des mesures brutes du capteur (mais c'est malheureusement mal documenté).
Il est donc plus simple de traiter les mesures brutes sur sa carte Arduino.

Mesures réalisées par le capteur

-Le **gyroscope** retourne une **vitesse angulaire de rotation** selon 3 axes, 0 si pas de rotation (degrés/seconde).

Il ne donne pas directement un angle d'orientation (degrés).
Un **angle** s'obtient par intégration dans le temps, en faisant attention au cumul des erreurs de dérive.

- **L'accéléromètre** retourne une **force ou une accélération** (m^2/s), la pesanteur terrestre seule si le module est fixe.
On peut la supprimer (par soustraction) dans le code si on ne veut que les accélérations.

La **vitesse** (m/s) peut s'en déduire par une première intégration dans le temps (à une vitesse initiale V0 près), en faisant attention au cumul des erreurs de dérive.

La **position** de déplacement (m) peut s'en déduire par une seconde intégration dans le temps.
Un filtrage des mesures brutes s'impose si on ne veut pas cumuler des petites erreurs et faire dériver la position exacte.

Pilotage d'un servomoteur selon la position du capteur
Les changements de positions mesurés peuvent être utilisés pour **stabiliser à l'horizontale** une nacelle photo ou vidéo (compensation de tangage et roulis), ou un avion / hélicoptère radiocommandés par exemple (guimbal).
La stabilisation d'une caméra (sur 3 axes) nécessite 3 servos et l'inversion des mouvements.

Filtrage des mesures brutes
L'accéléromètre doit être filtré si on veut piloter un servo de manière fluide sans vibrations.

- On peut utiliser une simple **moyenne glissante** de 40 mesures de 12 ms chacune par exemple, pour lisser les micro fluctuations
- ou utiliser un filtre de Kalman (méthode plus complexe).

Applications
- Pilotage automatique et stabilisation en radio modélisme
- Jouets, sports (Segway...)

- Reconnaissance de gestes
- Télécommandes 3D
- Capteur de mouvement à porter sur soi
- Stabilisation de camera vidéo, nacelle photo (guimbal)
- Commandes par mouvements
- Détecteur de chocs

Lecture des données brutes du capteur
Les échelles de mesures suivantes sont programmables :

- 3 axes de gyroscope , échelles de ±250, ±500, ±1000, et ±2000 degrés/sec
- 3 axes d'accéléromètre , pleine échelle de ±2g, ±4g, ±8g et ±16g.

Exemple de programme Arduino
Ce code fournit en sortie plusieurs données calculées.
https://github.com/jrowberg/i2cdevlib/t ... no/MPU6050

- **les quaternions (w x y z)**.
  3 quaternions permettent de définir un axe, et le 4ème décrit la rotation autour de cet axe, ceci permet d'éviter les limitations au delà de 180°.
  Voir l'effet de guimbal lock ici http://en.wikipedia.org/wiki/Gimbal_lock
- **les angles d'Euler** (psi, téta, phi).
  Les angles d'Euler sont les rotations autour des axes X Y et Z.
- **les angles yaw , pitch, roll**.
  YAW = direction(> 0 virage à droite)
  PITCH = tangage (> 0 bascule en avant)
  ROLL = roulis (> 0 à gauche)
  C'est la sortie la plus utile (guimbal)

  Attention au phénomène de guimbal lock (perte d'un degré de liberté quand deux axes de cardan sont alignés).
- **l'accélération**, selon les axes du capteur
- **l'accélération sans la pesanteur** (yaw selon l'orientation initiale, sans magnétomètre).
- **le format brut**

Câblage
4 fils suffisent au montage (grâce au Bus i2c)

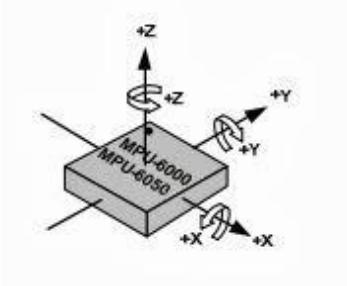
Le capteur est monté sur un PCB avec 2 x 5 pins --> Carte Arduino Uno

- 5V
- 3V3 --> 3.3V (ne pas utiliser le 5v)
- SDA -->A4 (uno)
- SCL -->A5 (uno)
- GND --> GND

- INT --> pin2 si on utilise les interruptions, pas nécessaire
- ADO
- XDA
- XCL
- GND

Rappel : Brochage du bus S2I de la carte UNO A4 =SDA, A5= SCL, et pour la carte Mega 20= SDA, 21=SCL.

Orientation spatiale des 3 axes
La puce comporte 3 axes (XY dans le sens anti-horaire vu de dessus)
x = selon le longueur, gauche > droite
y = selon la largeur , avant > arrière
z = selon la hauteur, bas > haut



Attention, les axes xy sont inversés sur la sérigraphie du PCB. (X = y et Y = -x)

Exemple de code : un détecteur de chocs
L'accéléromètre mesure Ax, Ay, Az, on calcule avec ça la pesanteur terrestre, on ne garde que les accélérations, et on affiche l'amplitude des chocs (sismographe, alarme, etc..).
CODE:

```
//====== MPU-6050  ========================================
// Demo mesure des valeurs brutes Acceleration - Gyro
// tiptopboards.com
// Modifié 30 08 2013
//
//=========================================================
// Source :  10/7/2011 by Jeff Rowberg <jeff@rowberg.net>
//  https://github.com/jrowberg/i2cdevlib
//=========================================================
#include "Wire.h"  // Arduino Wire library
#include "I2Cdev.h"  //Installer ces 2 librairies
#include "MPU6050.h"
// AD0 low = 0x68 (default for InvenSense evaluation board)
// AD0 high = 0x69
MPU6050 accelgyro;

int16_t ax, ay, az;  //mesures brutes
int16_t gx, gy, gz;

int16_t ax_moy, ay_moy, az_moy;  //moyennes
int16_t gx_moy, gy_moy, gzv;

#define LED_PIN 13   //Clignotte pour indiquer le bon fonctionnement
bool blinkState = false;

void setup() {
   Wire.begin();  //I2C bus
   //Serial.begin(9600);
    Serial.begin(38400);

   // initialize device
   Serial.println("Initialisation I2C...");
   accelgyro.initialize();
```

```
    // verify connection
    Serial.println("Test de la conection du dispositif ...");
    Serial.println(accelgyro.testConnection() ? "MPU6050 connection reussie" :
"MPU6050 connection echec");
    pinMode(LED_PIN, OUTPUT);  //Allume la led
}

void loop() {
    // Lire les données brutes acceleration / gyro
    accelgyro.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
    ax_moy = ax;
    // On peut aussi utiliser ces méthodes
    //accelgyro.getAcceleration(&ax, &ay, &az);
    //accelgyro.getRotation(&gx, &gy, &gz);

    // Affichage accel/gyro x/y/z
    Serial.print("a/g:\t");
    Serial.print(ax); Serial.print("\t");
    Serial.print(ay); Serial.print("\t");
    Serial.print(az); Serial.print("\t");
    Serial.print(gx); Serial.print("\t");
    Serial.print(gy); Serial.print("\t");
    Serial.print(gz); Serial.print("\t");
    // x > 0 si bascule en avant (tanguage)
    // y > 0 si roulis du côté gauche
    // z > 0 si orienté en haut

    //== module détecteur de chocs ou d'accélérations (sans 1g)
    float module = (float)ax*ax + (float)ay*ay + (float)az*az;
    module = abs(sqrt(module)/1000-16);   //retirer 1g et absolu
    String etoiles = "******************************";
    String nb_etoiles = etoiles.substring(0,(int)module);
    Serial.print(module);
    Serial.print(" ");
    Serial.println(nb_etoiles);  //module

    delay (100);
    blinkState = !blinkState;   //LED témoin d'activité clignotte
    digitalWrite(LED_PIN, blinkState);
}
```

Exemple de code : les rotations
Mesure des angles de roulis, de tangage et de lacet.
CODE:

```
//=== demo MPU 6050 roulis tangage et lacet ====================
// tiptopboards.com
// Rolland modifié 30 08 2013
//
//===============================================================
// 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
//
===============================================================
/*I2Cdev device library code is placed under the MIT license
Copyright (c) 2012 Jeff Rowberg

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:
The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.
*/

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

// 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
```

```
// 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;

/*
============================================================================
    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_QUATERNION" if you want to see the actual
// quaternion components in a [w, x, y, z] format (not best for parsing
// on a remote host such as Processing or something though)
//#define OUTPUT_READABLE_QUATERNION

// uncomment "OUTPUT_READABLE_EULER" if you want to see Euler angles
```

```
// (in degrees) calculated from the quaternions coming from the FIFO.
// Note that Euler angles suffer from gimbal lock (for more info, see
// http://en.wikipedia.org/wiki/Gimbal_lock)
//#define OUTPUT_READABLE_EULER

// 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

// uncomment "OUTPUT_READABLE_REALACCEL" if you want to see acceleration
// components with gravity removed. This acceleration reference frame is
// not compensated for orientation, so +X is always +X according to the
// sensor, just without the effects of gravity. If you want acceleration
// compensated for orientation, us OUTPUT_READABLE_WORLDACCEL instead.
//#define OUTPUT_READABLE_REALACCEL

// uncomment "OUTPUT_READABLE_WORLDACCEL" if you want to see acceleration
// components with gravity removed and adjusted for the world frame of
// reference (yaw is relative to initial orientation, since no magnetometer
// is present in this case). Could be quite handy in some cases.
//#define OUTPUT_READABLE_WORLDACCEL

// uncomment "OUTPUT_TEAPOT" if you want output that matches the
// format used for the InvenSense teapot demo
//#define OUTPUT_TEAPOT



#define LED_PIN 13 // (Arduino is 13, Teensy is 11, Teensy++ is 6)
bool blinkState = false;

// 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

// 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

// packet structure for InvenSense teapot demo
uint8_t teapotPacket[14] = { '$', 0x02, 0,0, 0,0, 0,0, 0,0, 0x00, 0x00, '\r', '\n' };



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

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



// ================================================================
// ===                      INITIAL SETUP                       ===
// ================================================================

void setup() {
    // join I2C bus (I2Cdev library doesn't do this automatically)
    Wire.begin();

    // 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);
    //Serial.begin(9600);
```

```
    //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();

    // 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(")"));
    }

    // configure LED for output
    pinMode(LED_PIN, OUTPUT);
}



// ================================================================
// ===                    MAIN PROGRAM LOOP                     ===
// ================================================================

void loop() {
    // 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_QUATERNION
      // display quaternion values in easy matrix form: w x y z
      mpu.dmpGetQuaternion(&q, fifoBuffer);
      Serial.print("quat\t");
      Serial.print(q.w);
      Serial.print("\t");
      Serial.print(q.x);
      Serial.print("\t");
      Serial.print(q.y);
      Serial.print("\t");
      Serial.println(q.z);
   #endif

   #ifdef OUTPUT_READABLE_EULER
      // display Euler angles in degrees
      mpu.dmpGetQuaternion(&q, fifoBuffer);
```

```
    mpu.dmpGetEuler(euler, &q);
    Serial.print("euler\t");
    Serial.print(euler[0] * 180/M_PI);
    Serial.print("\t");
    Serial.print(euler[1] * 180/M_PI);
    Serial.print("\t");
    Serial.println(euler[2] * 180/M_PI);
#endif

#ifdef OUTPUT_READABLE_YAWPITCHROLL
    // display Euler angles in degrees
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    mpu.dmpGetGravity(&gravity, &q);
    mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
    Serial.print("ypr\t");
    Serial.print(ypr[0] * 180/M_PI);
    Serial.print("\t");
    Serial.print(ypr[1] * 180/M_PI);
    Serial.print("\t");
    Serial.print(ypr[2] * 180/M_PI);
    Serial.print("\t");

#endif

#ifdef OUTPUT_READABLE_REALACCEL
    // display real acceleration, adjusted to remove gravity
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    mpu.dmpGetAccel(&aa, fifoBuffer);
    mpu.dmpGetGravity(&gravity, &q);
    mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
    Serial.print("areal\t");
    Serial.print(aaReal.x);
    Serial.print("\t");
    Serial.print(aaReal.y);
    Serial.print("\t");
    Serial.println(aaReal.z);
#endif

#ifdef OUTPUT_READABLE_WORLDACCEL
    // display initial world-frame acceleration, adjusted to remove gravity
    // and rotated based on known orientation from quaternion
```

```
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            mpu.dmpGetAccel(&aa, fifoBuffer);
            mpu.dmpGetGravity(&gravity, &q);
            mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
            mpu.dmpGetLinearAccelInWorld(&aaWorld, &aaReal, &q);
            Serial.print("aworld\t");
            Serial.print(aaWorld.x);
            Serial.print("\t");
            Serial.print(aaWorld.y);
            Serial.print("\t");
            Serial.println(aaWorld.z);
        #endif

        #ifdef OUTPUT_TEAPOT
            // display quaternion values in InvenSense Teapot demo format:
            teapotPacket[2] = fifoBuffer[0];
            teapotPacket[3] = fifoBuffer[1];
            teapotPacket[4] = fifoBuffer[4];
            teapotPacket[5] = fifoBuffer[5];
            teapotPacket[6] = fifoBuffer[8];
            teapotPacket[7] = fifoBuffer[9];
            teapotPacket[8] = fifoBuffer[12];
            teapotPacket[9] = fifoBuffer[13];
            Serial.write(teapotPacket, 14);
            teapotPacket[11]++; // packetCount, loops at 0xFF on purpose
        #endif

        // blink LED to indicate activity
        blinkState = !blinkState;
        digitalWrite(LED_PIN, blinkState);
    }
}
```

Références

Arduino Playground MPU-6050
http://playground.arduino.cc/Main/MPU-6050