

DOCUMENTATION TECHNIQUE DES TESTS EN ELECTRONIQUE

Test 1 : Gyroscope et accéléromètre

S'orienter dans l'espace, savoir distinguer le haut, le bas, la gauche, la droite, l'avant et l'arrière, est une compétence naturelle chez l'être humain.

Mais comment faire pour qu'un robot ou un système automatisé reproduise cette capacité ?

C'est là qu'interviennent les capteurs inertIELS : des composants électroniques capables de « sentir » les mouvements et rotations d'un objet dans l'espace. Nous avons choisi d'explorer cette technologie avec un capteur MPU6050, un module combinant :

- un **accéléromètre 3 axes**, pour détecter les accélérations (y compris la gravité),
- un **gyroscope 3 axes**, pour mesurer les vitesses de rotation autour de ces 3 axes.

Ce capteur sera placé dans la paume d'une main et connecté à un microcontrôleur ATmega328P.

À chaque geste : lever la main, la baisser, la tourner, les informations seront affichées en temps réel sur un écran LCD.

✓ En quoi ce test relève d'une importance capitale ?

Ce test constitue une **immersion concrète** dans les systèmes embarqués et la robotique, où les capteurs inertIELS comme ceux que l'on retrouve dans les drones, smartphones ou robots sont essentiels pour mesurer la position, la vitesse et la stabilité du système. En combinant un accéléromètre (qui mesure les accélérations linéaires) et un gyroscope (qui mesure la vitesse de rotation), on peut reconstruire en temps réel la trajectoire et l'orientation d'un objet dans l'espace : c'est ce que l'on appelle la fusion de données, un processus indispensable dans les applications à 3D.

Grâce à la simplicité du module MPU6050 (interface I2C standard) et la restitution des résultats sur un écran LCD, ce test offre une approche pédagogique efficace : il permet de visualiser directement comment un geste manuel se traduit en direction détectée, sans complexité inutile.

Ce compromis entre **technique**, **accessibilité** et **interactivité** en fait un excellent point de départ pour quiconque souhaite comprendre la détection de mouvement, l'électronique embarquée et l'orientation spatiale, avant de creuser des solutions plus complexes (filtres avancés, modèles AHRS, etc.).

1. Objectifs de ce test

Ce test vise à développer et valider plusieurs compétences techniques et pratiques essentielles dans le domaine des systèmes embarqués et de la détection de mouvement. Plus précisément, il s'agit de :

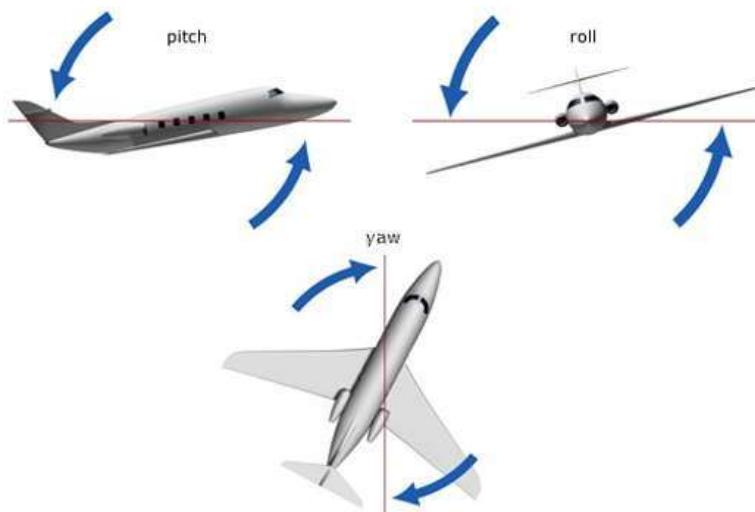
- Identifier et comprendre le fonctionnement d'un capteur inertiel combiné (MPU6050) intégrant un accéléromètre et un gyroscope, et savoir exploiter ses données pour mesurer l'orientation et les mouvements dans l'espace.
- Mettre en œuvre la **communication I2C** entre le capteur et le microcontrôleur ATmega328P, en maîtrisant **l'échange de données numériques** via ce protocole standardisé.
- Programmer un microcontrôleur Arduino pour lire les données brutes du capteur, appliquer un traitement logiciel (filtrage, calibration) et calculer les angles d'orientation (yaw, pitch, roll).
- Développer une **logique de détection** de direction capable d'interpréter les variations d'angles et d'accélération pour déterminer les mouvements de la main dans les six directions principales (haut, bas, gauche, droite, avant, arrière).
- Assurer **l'affichage en temps réel** des résultats sur un écran LCD I2C, permettant une visualisation claire et intuitive des mouvements détectés.

- Comprendre l'**importance de la calibration et du filtrage** pour améliorer la précision et la fiabilité des mesures issues du capteur inertiel.
- Acquérir une démarche méthodique de test et validation, en vérifiant le bon fonctionnement du système dans des conditions réelles d'utilisation.

2. Comprendre les orientations : roulis, tangage et lacet.

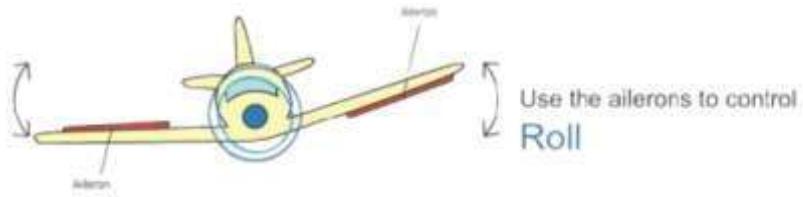
Pour décrire précisément comment un objet (comme ta main équipée du capteur MPU6050) s'oriente et se déplace dans l'espace, on utilise trois angles fondamentaux appelés roulis (roll), tangage (pitch) et lacet (yaw).

Ces trois mouvements correspondent à des rotations autour de trois axes perpendiculaires, et permettent de définir la position et l'orientation d'un objet dans un espace à trois dimensions.



a) Le roulis (Roll)

C'est la **rotation** autour de l'**axe longitudinal**, c'est-à-dire un mouvement d'inclinaison latérale, comme si ta main penchait vers la droite ou la gauche.

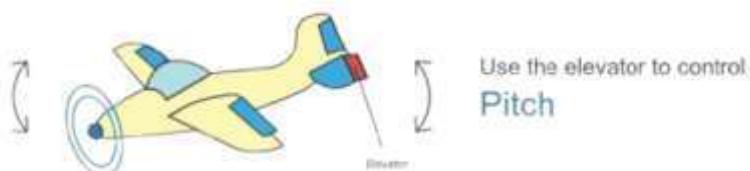


Démonstration pratique : imagine que tu tiens ta main devant toi et que tu la fais basculer sur le côté, comme si tu voulais verser un verre d'eau. Ce mouvement est un roulis.

Importance dans notre système : le roulis permet de détecter si la main s'incline sur la gauche ou la droite, ce qui est essentiel pour comprendre les mouvements latéraux.

b) Le tangage (Pitch)

C'est la **rotation** autour de l'**axe transversal**, correspondant à un mouvement de bascule avant-arrière.



Démonstration pratique : imagine que tu fais un signe de tête pour dire « oui », en penchant ta main vers l'avant ou l'arrière. Ce mouvement est un tangage.

Importance dans notre système : le tangage permet de détecter si la main se penche vers l'avant ou vers l'arrière, ce qui correspond aux mouvements de translation avant/arrière.

c) Le lacet (Yaw)

C'est la **rotation** autour de l'**axe vertical**, c'est-à-dire un mouvement de rotation horizontale, comme si ta main tournait sur elle-même à plat.

Démonstration pratique : imagine que tu tournes ta main vers la gauche ou la droite sans la pencher, juste en la faisant pivoter sur elle-même. Ce mouvement est un lacet.

Importance dans notre système : le lacet permet de détecter les rotations horizontales, ce qui est crucial pour comprendre les mouvements de rotation autour de l'axe vertical.

3. Pourquoi ces angles sont-ils importants dans notre système ?

Le MPU6050 mesure ces trois rotations grâce à son gyroscope, et les angles correspondants (roll, pitch, yaw) sont calculés pour déterminer précisément **l'orientation de la main** dans l'espace ainsi que son accélération.

En combinant ces données avec celles de l'accéléromètre (qui mesure les accélérations linéaires), le système peut détecter non seulement la **position**, mais aussi le **mouvement et la direction** de la main.

Cette compréhension des trois axes de rotation est essentielle pour :

- Interpréter correctement les gestes (lever, baisser, tourner la main).
- Détecter les directions de mouvement (haut, bas, gauche, droite, avant, arrière).
- Améliorer la précision des mesures en filtrant et fusionnant les données des capteurs.

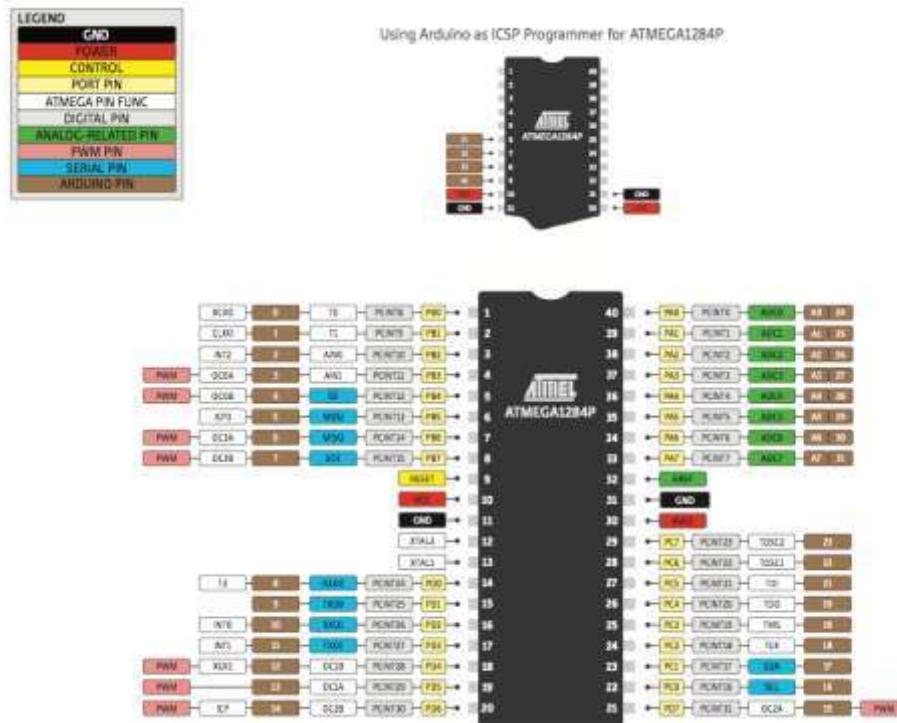
4. Choix du matériel pour ce test

Pour réaliser ce projet de détection d'orientation et de mouvement, le choix du matériel est crucial afin d'assurer un fonctionnement fiable, précis et adapté à une intégration finale. Nous avons opté pour des composants permettant à la fois simplicité, compacité et performance.

- Microcontrôleur : ATmega328P seul

Plutôt que d'utiliser une carte Arduino UNO complète, nous avons choisi d'employer le microcontrôleur ATmega328P en version « puce seule ». Cette approche présente plusieurs avantages :

- **Miniaturisation** : le microcontrôleur seul occupe beaucoup moins de place, facilitant l'intégration dans un boîtier ou un montage personnalisé.
- **Le coût réduit** : le prix du microcontrôleur nu est nettement inférieur à celui d'une carte Arduino complète.
- **La personnalisation du circuit** : cela permet de sélectionner précisément les composants périphériques nécessaires (oscillateur, alimentation, reset) et d'optimiser le montage.
- **Une approche pédagogique** : cette méthode offre une meilleure compréhension du fonctionnement matériel d'un système embarqué en réalisant soi-même le circuit minimal.





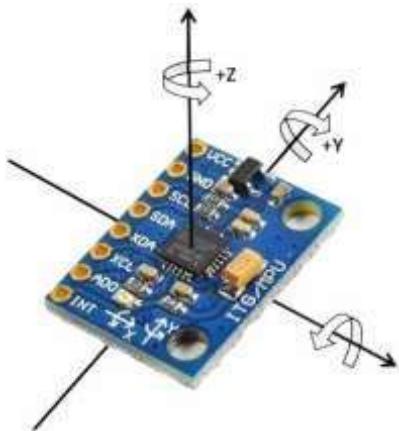
Composants nécessaires pour faire fonctionner l'ATmega328P

- ✓ Un **cristal oscillateur** 16 MHz avec des condensateurs pour fournir une horloge stable.



- ✓ **Résistance pull-up** sur la broche RESET pour éviter les réinitialisations intempestives.
 - ✓ **Alimentation 5 V** régulée avec condensateurs de découplage.
 - ✓ **Convertisseur USB-série externe** (comme un module FTDI) pour programmer le microcontrôleur via USB.
 - ✓ **Câblage** soigné pour relier le microcontrôleur aux capteurs et à l'écran.
- **Capteur inertiel : MPU6050**

Le choix du capteur s'est porté sur le **module MPU6050**, qui combine un accéléromètre 3 axes et un gyroscope 3 axes dans un seul composant.



Pourquoi le MPU6050 ?

- *Mesure complète du mouvement* : il fournit à la fois les **accélérations linéaires** et les **vitesses angulaires**, indispensables pour déterminer l'orientation dans l'espace.
- *Interface I2C* : la communication via un bus I2C standard simplifie le câblage et permet de connecter plusieurs périphériques sur les mêmes lignes SDA et SCL.
- *Intégration du DMP* : le MPU6050 intègre un **processeur de mouvement numérique (DMP)** qui réalise la **fusion des données** pour fournir des angles d'orientation stables.
- *Compatibilité avec ATmega328P* : fonctionne sous une alimentation de 3,3 V à 5 V, compatible avec le microcontrôleur utilisé.

NB : Particularités à prendre en compte

- Le module MPU6050 fonctionne généralement sous 3,3 V, mais la plupart des modules GY-521 intègrent un régulateur 3,3 V et des résistances pull-up compatibles 5 V, ce qui facilite l'utilisation avec un ATmega328P alimenté en 5 V.
- Il faut veiller à connecter SDA (données) et SCL (horloge) aux broches A4 et A5 du microcontrôleur.
- L'adresse I2C par défaut est 0x68, modifiable en connectant la broche AD0 à 3,3 V (adresse 0x69).

- **Affichage : écran LCD I2C 16x2**

Pour visualiser les résultats en temps réel, un écran LCD 16 colonnes x 2 lignes avec interface I2C a été choisi.



Avantages de l'écran LCD I2C

- **Câblage simplifié** : utilise seulement 4 fils (VCC, GND, SDA, SCL) pour la communication et l'alimentation.
- **Compatibilité I2C** : partage le bus I2C avec le MPU6050, réduisant le nombre de connexions nécessaires.
- **Affichage clair et lisible** : Permet de visualiser directement la direction détectée sans matériel supplémentaire.

5. Architecture et principe de fonctionnement

Ce système utilise le capteur MPU6050 pour détecter comment un objet (comme une main) bouge et s'oriente dans l'espace.

Voici comment fonctionnera le système de notre test :

✓ **Acquisition des données :**

Le MPU6050 contient deux capteurs importants :

- **Accéléromètre 3 axes** : il mesure les accélérations sur trois directions (avant/arrière, gauche/droite, haut/bas). Cela permet de savoir si la main accélère ou change d'inclinaison.
- **Gyroscope 3 axes** : il mesure la vitesse à laquelle la main tourne autour de ces trois axes.

Ces deux capteurs travaillent ensemble pour donner une image complète du mouvement.

✓ **Prétraitement des données :**

Les mesures brutes du capteur peuvent contenir du bruit (petites erreurs ou fluctuations).

Pour améliorer la qualité :

- ❖ On applique un **filtre passe-bas** qui lisse les données de l'accéléromètre, en gardant les tendances importantes et en supprimant les petites perturbations.
- ❖ Le capteur utilise aussi une fonction appelée DMP (*Digital Motion Processor*) qui prépare les données et envoie une alerte (interruption) quand elles sont prêtes à être lues.

Cela permet au microcontrôleur de récupérer les données au bon moment, sans perte d'information.

✓ **Fusion et calcul des orientations :**

Le DMP du MPU6050 combine les données de l'**accéléromètre** et du **gyroscope** pour calculer la position exacte de la main dans l'espace. Il utilise une méthode mathématique appelée **quaternion** pour représenter l'orientation sans ambiguïté.

Un **quaternion** est une structure mathématique utilisée pour représenter des rotations dans l'espace 3D.

À partir de ces quaternions, on calcule trois angles simples à comprendre :

- Roll (roulis) : inclinaison gauche-droite.

- Pitch (tangage) : inclinaison avant-arrière.
- Yaw (lacet) : rotation sur place à gauche ou à droite.

Le **système** corrige aussi l'accélération mesurée pour enlever l'effet de la gravité, afin de détecter uniquement les mouvements réels.

✓ Détection de la direction

Pour savoir dans quelle **direction** la main se déplace, le système :

- Surveille **l'accélération sur l'axe Z** pour détecter un mouvement vers le haut ou le bas.
- Analyse les **variations des angles roll, pitch et yaw** pour détecter les mouvements avant/arrière, gauche/droite et les rotations.
- Si aucun mouvement significatif n'est détecté, l'état est considéré comme étant stable.

✓ Affichage et retour utilisateur

- La **direction détectée** est affichée en temps réel sur un écran LCD I2C 16×2, facile à lire et à comprendre.
- Les données peuvent aussi être envoyées sur un moniteur série (ordinateur) pour suivre les mesures, calibrer le système et améliorer la précision.

6. Liste du matériel

Composants	Spécifications principales	Rôle dans le test
ATmega328P	Horloge 16 MHz, 32 kB Flash	Microcontrôleur principal, traitement des données et contrôle du système

MPU6050 (GY-521)	Interface I2C, DMP intégré, alimentation 3 V– 5 V	Capteur inertiel combinant accéléromètre et gyroscope MEMS
Écran LCD 16×2 I ² C	Adresse I2C 0x27 ou 0x3F	Interface utilisateur pour affichage en temps réel des directions détectées
Module FTDI	Interface USB-TTL, 6 broches	Programmation du microcontrôleur et communication série pour debug
Veroboard ou PCB	24×16 bandes, format Eurorack	Support mécanique pour montage et soudure des composants
Fer à souder + étain	Pointe fine 0,4 mm	Assemblage des composants électroniques par soudure
Multimètre	Mesure de continuité, tension, résistance	Vérification des connexions et tests électriques
Breadboard + Dupont	Montage sans soudure, prototypage rapide	Réalisation de prototypes et tests avant soudure définitive

- Microcontrôleur : ATmega328P – PU (version en boîtier DIP28)
- Alimentation
 - Régulateur 5V (ex : AMS1117 ou module 7805)
 - Régulateur AMS1117
 - Régulateur 7805
 - Source : 7 à 12V
 - Condensateurs de filtrage :
 - 1 × 10 µF électrolytique (entre VCC et GND pour l'alimentation générale)

- $2 \times 100 \text{ nF}$ céramique (entre VCC et GND, et AVCC et GND pour le découplage)
- Oscillateur
 - Quartz 16 MHz
 - 2 condensateurs de 22 pF
- Broche Reset
 - Résistance $10 \text{ k}\Omega$ (entre RESET et $+5V$ en pull-up)
 - Bouton poussoir pour forcer le RESET à GND
- Connexion pour la programmation
 - Une carte Arduino UNO
 - Un convertisseur USB-TTL
- Test visuel du programme
 - Une LED (pour le clignotement du test Blynk)
 - Une résistance $220\Omega - 330\Omega$ (en série avec la LED pour sa protection)

7. Réalisation du circuit

A. Test du microcontrôleur ATmega328P

Avant d'intégrer l'ATmega328P à ton circuit définitif, il est conseillé de le tester sur une breadboard (plaquette d'essai).

- Montage minimal : placer le microcontrôleur sur la breadboard avec :
 - une alimentation 5 V
 - un cristal 16 MHz

- deux condensateurs de 22 pF
- une résistance de 10 kΩ entre RESET et Vcc

Relie aussi les broches d'alimentation (Vcc, GND, AVcc).

- Test de fonctionnement : téléverse un code simple (ex. blink LED) pour vérifier que la puce fonctionne correctement.

B. Gravure du bootloader

Le bootloader permet à l'ATmega328P d'être programmé comme un Arduino classique.

Si tu utilises une puce neuve, il faut graver ce bootloader :

- Utilise une carte Arduino comme programmateur ou un programmateur externe.
- Branchements : Connecte les broches MOSI, MISO, SCK, RESET, Vcc et GND de l'Arduino à celles de l'ATmega328P.
- Gravure :
 - Dans l'IDE Arduino, sélectionne : Arduino as ISP, puis Burn Bootloader.
 - Choisis le bon modèle selon ton montage :
 - ATmega328 on a breadboard (8 MHz internal clock) si tu utilises l'oscillateur interne
 - ou Arduino Uno si tu utilises un cristal 16 MHz

C. Conception du schéma sous KiCad

Ce circuit a pour but de mesurer les mouvements à l'aide du capteur MPU-6050 (accéléromètre et gyroscope).

- Ces mesures sont traitées et formatées par le microcontrôleur ATmega328P.
- Elles sont ensuite affichées sur un écran LCD.

- Tous les composants communiquent via le protocole I2C, ce qui permet une réduction du nombre de connexions nécessaires.

D. Schéma, composants et assemblage

Le schéma a été réalisé dans KiCad. Les composants ont été choisis depuis la librairie KiCad officielle. Les connexions sont nommées de façon explicite pour faciliter la lecture.

❖ Liste des composants

- ATmega328P
- MPU-6050
- LCD I2C
- Quartz, condensateurs et résistances

Le quartz génère l'horloge du microcontrôleur. Les condensateurs C1 et C2 sont typiquement de 22 pF, et la résistance de 1 kΩ.

- Bouton poussoir : il permet de réinitialiser manuellement le microcontrôleur.

❖ Fonctionnement global

À la **mise sous tension** :

- Le microcontrôleur initialise le capteur MPU-6050 via le module I2C.
- Il lit en boucle les **valeurs d'accélérations et de rotation**.
- Les données sont traitées et converties dans un format lisible.
- Les résultats sont envoyés à l'**écran LCD I2C** pour affichage.

❖ Alimentation

- Tous les composants sont alimentés en +5V (VCC).
- Le MPU-6050 possède une broche VLOGIC pour s'adapter aux logiques de 3.3V ou 5V.
- L'alimentation peut être fait :
 - par un adaptateur
 - ou par une batterie

❖ Protocole de communication utilisé

Les capteurs d'accélération et de déplacement doivent transmettre des données précises et lisibles.

Les broches analogiques de communication I2C assurent cela :

- **SDA** : reçoit, traite et transmet les données des capteurs vers l'écran LCD.
- **SCL** : génère un signal d'horloge pour synchroniser l'envoi et la réception des données numériques.

E. Soudure des composants

➤ Préparation :

- Préchauffer le fer à souder (300 - 350 °C)
- Nettoyer la panne
- Préparer les composants et le veroboard

➤ Soudure des composants traversants (THT) :

- Placer chaque composant
- Souder une patte, ajuster, puis souder les autres

- Couper l'excédent
- Soudure des composants SMD (si applicable) :
 - Utiliser de la pâte à souder et du flux
 - Fixer un coin, puis souder les autres broches
(drag soldering ou point par point)

NB : Utiliser du fil à souder fin (0,38 mm recommandé), éviter les ponts de soudure, vérifier que chaque joint soit brillant et conique, inspecter visuellement et corriger les défauts éventuels.

F. Programmation et test du circuit assemblé

- Connexion du convertisseur USB-série : brancher le module FTDI (ou équivalent) aux broches RX, TX, Vcc, GND (et DTR/RESET si présent) de l'ATmega328P.
- Téléversement du code : utiliser l'IDE Arduino pour charger un programme de test (ex. blink ou lecture capteur).
- Test fonctionnel :

Vérifier le bon fonctionnement :

- du microcontrôleur
- de la communication I2C (MPU6050 + LCD)
- de l'alimentation

Vidéo de démonstration

- <https://vimeo.com/1093499371?share=copy>

8. Explication détaillée du code MPU6050 + LCD I2C avec ATmega328P

✓ Rappels sur les objectifs du projet

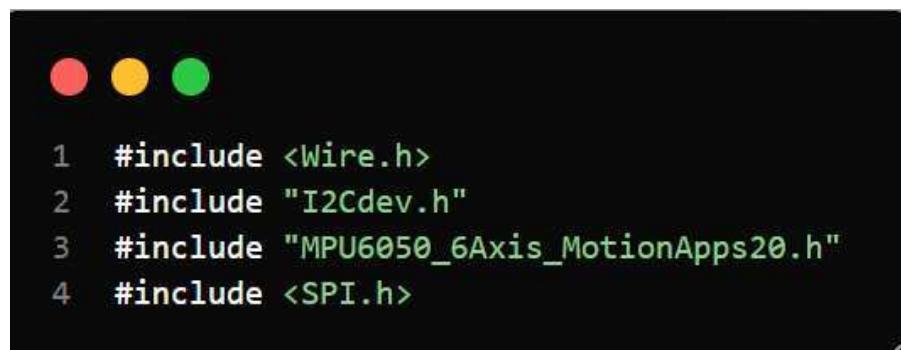
Ce projet a pour but de :

- ✓ Lire l'orientation et les mouvements d'un objet à l'aide du capteur MPU6050 (gyroscope et accéléromètre).
- ✓ Afficher ces mouvements sur un petit écran LCD relié en I2C.
- ✓ Utiliser un microcontrôleur ATmega328P (le cœur des cartes Arduino Uno par exemple).

✓ Les bibliothèques utilisées

Avant de pouvoir écrire notre programme, on utilise des *bibliothèques* qui contiennent des morceaux de code déjà écrits par d'autres personnes, pour simplifier notre travail.

Voici celles qu'on utilise :



```
1 #include <Wire.h>
2 #include "I2Cdev.h"
3 #include "MPU6050_6Axis_MotionApps20.h"
4 #include <SPI.h>
```

Pourquoi ces bibliothèques ?

- ❖ **Wire.h** : permet de communiquer avec des appareils via le protocole I2C (deux fils seulement : SDA et SCL).
- ❖ **LiquidCrystal_I2C.h** : gère l'affichage des messages sur notre écran LCD.

- ❖ **I2Cdev.h et MPU6050_6Axis_MotionApps20.h** : facilitent l'utilisation du capteur MPU6050, notamment de son DMP (Digital Motion Processor) qui calcule lui-même l'orientation.

✓ Cration des objets

Pour utiliser l'écran et le capteur, on cre des *objets* :



```
1 LiquidCrystal_I2C lcd(0x27, 16, 2);
2 MPU6050 mpu;
```

- 0x27 est l'adresse la plus frquente pour les écrans LCD I2C.
- mpu est l'objet qui nous permettra de piloter le capteur MPU6050.

✓ Gestion des interruptions

Une interruption permet de dire au microcontrôleur :

« Attention ! Le capteur a des nouvelles données ! »

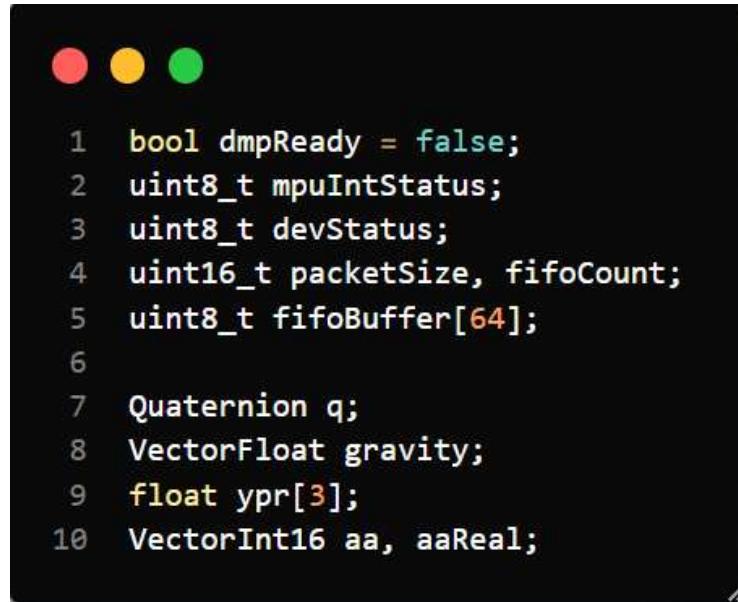


```
1 #define INTERRUPT_PIN 2
2 volatile bool mpuInterrupt = false;
3 void dmpDataReady() { mpuInterrupt = true; }
```

- On utilise la broche 2 de l'ATmega328P pour détecter les nouvelles données du MPU6050.

✓ Variables globales

Pour stocker les données du capteur :



```

1  bool dmpReady = false;
2  uint8_t mpuIntStatus;
3  uint8_t devStatus;
4  uint16_t packetSize, fifoCount;
5  uint8_t fifoBuffer[64];
6
7  Quaternion q;
8  VectorFloat gravity;
9  float ypr[3];
10 VectorInt16 aa, aaReal;

```

- Quaternion et YPR (Yaw, Pitch, Roll) permettent de décrire l'orientation dans l'espace.
- aaReal : accélération mesurée sans l'effet de la gravité (vraie accélération de l'objet).

On utilise aussi des variables pour :

- ✓ Comparer les mouvements d'une lecture à l'autre.
- ✓ Filtrer les données pour éviter les fausses détections dues à de petits mouvements parasites :

```
1 float prevYaw = 0, prevPitch = 0, prevRoll = 0;
2 const float seuilAngle = 5.0;           // Seuil en degrés pour rotations
3 const int   seuilAccZ   = 600;          // À ajuster après calibration (ex. 500-1200)
4 float accZ_filtered = 0;
5 const float alpha = 0.3; // coefficient filtre passe-bas
6 String direction = "";
```

✓ Fonction setup() — Initialisation

C'est ici qu'on prépare tout au démarrage :

```
1 void setup() {
2     Wire.begin();
3     lcd.begin(16, 2);
4     lcd.init();
5     lcd.backlight();
6     lcd.clear();
7     lcd.setCursor(0,0);
8     lcd.print("Init MPU6050...");
9     delay(1000);
10    Serial.begin(115200);
11    mpu.initialize();
12    pinMode(INTERRUPT_PIN, INPUT);
```

- ✓ Le bus I2C démarre.
- ✓ L'écran LCD s'allume.
- ✓ La communication série (USB vers PC) commence pour qu'on puisse afficher des messages de debug.
- ✓ Le capteur MPU6050 est initialisé.

✓ Calibrage du capteur

Le capteur doit être calibré pour donner des mesures précises :

```
1 // Offsets calibrés (à ajuster pour ton module)
2     mpu.setXAccelOffset(874);
3     mpu.setYAccelOffset(-853);
4     mpu.setZAccelOffset(-21);
5     mpu.setXGyroOffset(-59);
6     mpu.setYGyroOffset(20);
7     mpu.setZGyroOffset(-57);
8
9     devStatus = mpu.dmpInitialize();
10    if (devStatus == 0) {
11        mpu.CalibrateAccel(6);
12        mpu.CalibrateGyro(6);
13        mpu.setDMPEnabled(true);
14        attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), dmpDataReady, RISING);
15        mpuIntStatus = mpu.getIntStatus();
16        dmpReady = true;

```

```
1 packetSize = mpu.dmpGetFIFOPacketSize();
2         lcd.clear();
3         lcd.setCursor(0,0);
4         lcd.print("MPU6050 prêt!");
5         delay(1000);
6     } else {
7         lcd.clear();
8         lcd.setCursor(0,0);
9         lcd.print("Erreur DMP:");
10        lcd.print(devStatus);
11        while (1);
12    }
13 }
```

- Si tout fonctionne bien, le DMP est activé.
- L'interruption est attachée : le microcontrôleur sera prévenu à chaque nouveau paquet de données.

✓ Fonction `loop()` - Lecture continue des données

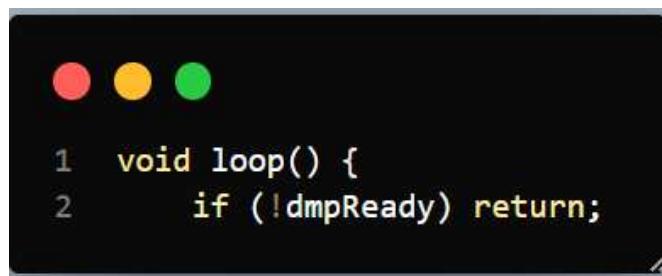
Dans Arduino, la fonction `loop()` est exécutée en boucle infinie. Tant qu'on alimente le microcontrôleur, cette fonction tourne.

Ici, le rôle principal de la boucle est :

- D'attendre que le capteur MPU6050 fournisse de nouvelles données.
- De lire ces données.
- De calculer l'orientation et les mouvements.
- D'afficher le résultat.

1. Vérification que le DMP est prêt

Au tout début de la boucle :



```
1 void loop() {
2     if (!dmpReady) return;
```

- Si le capteur MPU6050 n'a pas encore terminé sa phase d'initialisation, on quitte tout de suite la boucle et on ne fait rien.
- Tant que `dmpReady` est faux, on attend.
- `return` signifie « sortir immédiatement de la boucle actuelle ».

2. Vérifier s'il y a des nouvelles données

Ensuite :



```
1 if (!mpuInterrupt && fifoCount < packetSize) return;
```

- ✓ mpuInterrupt devient vrai quand une nouvelle donnée est prête (grâce à l'interruption sur la broche 2).
- ✓ fifoCount contient le nombre de paquets disponibles dans la mémoire tampon interne du capteur appelée FIFO (First In First Out = files d'attente).
- ✓ Tant qu'il n'y a pas de nouvelles données à lire (mpuInterrupt == false) et que le FIFO n'a pas assez de données pour former un paquet complet (fifoCount < packetSize), on ne fait rien et on attend le prochain tour de boucle.

3. Lecture de la mémoire FIFO

Dès qu'on a des nouvelles données prêtes :



```
1 mpuInterrupt = false;
2 mpuIntStatus = mpu.getIntStatus();
3 fifoCount     = mpu.getFIFOCount();
```

On récupère le nombre de bytes actuellement présents dans le FIFO.

Puis on vérifie qu'il n'y a pas d'erreur :

```
1 // Gestion overflow FIFO
2     if ((mpuIntStatus & 0x10) || fifoCount == 1024) {
3         mpu.resetFIFO();
4         lcd.clear();
5         lcd.setCursor(0,0);
6         lcd.print("Overflow FIFO!");
7         return;
8 }
```

- Si le FIFO déborde (c'est-à-dire qu'il est plein à 1024 bytes sans avoir été lu à temps), on remet à zéro la mémoire FIFO avec mpu.resetFIFO() pour repartir proprement.
- On affiche sur l'écran LCD un message d'alerte : "Overflow FIFO!"

4. Extraction du paquet de données

Quand tout va bien, on peut lire les données :

```
1
2 // Lecture du FIFO
3 while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();
4 mpu.getFIFOBytes(fifoBuffer, packetSize);
5 fifoCount -= packetSize;
6
```

- On lit un paquet complet de données depuis la FIFO.
- fifoBuffer est un tableau dans lequel on stocke ces données brutes.

5. Calcul des angles d'orientation

Une fois qu'on a les données, on peut les convertir en valeurs compréhensibles :

```
1 // Calcul des angles
2 mpu.dmpGetQuaternion(&q, fifoBuffer);
3 mpu.dmpGetGravity(&gravity, &q);
4 mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
5 float yaw = ypr[0] * 180/M_PI;
6 float pitch = ypr[1] * 180/M_PI;
7 float roll = ypr[2] * 180/M_PI;
```

- Quaternions (q) : format mathématique interne pour représenter l'orientation.
- Gravité (gravity) : vecteur calculé à partir des quaternions.
- Yaw-Pitch-Roll (ypr) : ce sont les trois angles que l'on comprend facilement :
 - Yaw : rotation horizontale (vers la droite ou gauche, comme une boussole).
 - Pitch : basculement avant/arrière.
 - Roll : basculement gauche/droite.

6. Calcul de l'accélération linéaire (mouvement réel sans gravité)

En plus des angles, on récupère aussi l'accélération subie par l'objet :

```
1 // Accélération linéaire
2     mpu.dmpGetAccel(&aa, fifoBuffer);
3     mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
4
5     float acc = sqrt(pow(aaReal.x, 2) + pow(aaReal.y, 2) + pow(aaReal.z, 2));
```

- aa = accélération brute.
- aaReal = accélération linéaire réelle (on retire l'effet de la gravité).

Puis on applique un filtrage pour lisser les variations :

```
1
2 // Filtrage passe-bas simple sur accZ
3 accZ_filtered = alpha * aaReal.z + (1 - alpha) * accZ_filtered;
```

On applique un filtre passe-bas simple (type moyenne pondérée) pour lisser les valeurs trop brusques.

7. Détection de la direction du mouvement

Maintenant qu'on a les mesures, on peut deviner dans quelle direction l'objet a bougé :

```
1 void detectDirection(float yaw, float pitch, float roll, float accZf) {  
2     float dPitch = pitch - prevPitch;  
3     float dRoll = roll - prevRoll;  
4     float dYaw = yaw - prevYaw;  
5  
6     if (abs(accZf) > seuilAccZ) {  
7         direction = (accZf > 0) ? "Haut" : "Bas";  
8     }  
9     else if (abs(dPitch) > abs(dRoll) && abs(dPitch) > seuilAngle) {  
10        direction = (dPitch > 0) ? "Avant" : "Arriere";  
11    }  
12    else if (abs(dRoll) > seuilAngle) {  
13        direction = (dRoll > 0) ? "Droite" : "Gauche";  
14    }  
15    else if (abs(dYaw) > seuilAngle) {  
16        direction = (dYaw > 0) ? "Rot Droite" : "Rot Gauche";  
17    }  
18    else {  
19        direction = "Stable";  
20    }  
}
```

- Si l'accélération sur l'axe Z est forte : l'objet est monté ou descendu.
- Sinon, on regarde les variations d'inclinaison (Pitch, Roll) pour savoir si on penche vers l'avant/arrière ou droite/gauche.
- Enfin, si c'est une simple rotation (Yaw), on l'indique aussi.
- Si aucune variation significative n'est détectée : l'objet est stable.

```
1 detectDirection(yaw, pitch, roll, accZ_filtered);
```

Donc on a utilisé une fonction pour cette détermination et on l'a fait appeler dans `loop()`.

8. Affichage des résultats

Enfin, on affiche sur l'écran LCD et sur le port série :



L'utilisateur voit directement la direction détectée en texte simple.

Un résumé visuel du cycle loop()

Étape	Action	Condition
1	Vérifie si le DMP est prêt	if (!dmpReady)
2	Attend des nouvelles données	if (!mpuInterrupt)
3	Vérifie l'absence d'erreur FIFO	if (fifoCount == 1024)
4	Lit les données FIFO	mpu.getFIFOBytes()
5	Calcule Yaw, Pitch, Roll	mpu.dmpGetYawPitchRoll()
6	Calcule accélération filtrée	accZ_filtered = ...

7	Déetecte la direction	detectDirection()
8	Affiche le résultat	lcd.print()

En résumé, ce programme :

- Utilise un capteur MPU6050 pour détecter les mouvements.
- Calcule automatiquement les angles grâce au DMP.
- Filtre les données pour éviter les erreurs.
- Affiche le résultat sur un écran LCD.
- Fonctionne en temps réel grâce aux interruptions.

CONCLUSION

Ce projet de détection d'orientation et de mouvement à l'aide du capteur MPU6050 et du microcontrôleur ATmega328P constitue une introduction concrète aux systèmes embarqués intelligents. Il montre comment des données brutes issues de capteurs peuvent être filtrées, fusionnées, interprétées, puis affichées en temps réel pour donner un retour visuel clair à l'utilisateur.

Grâce à une architecture simple et accessible (protocole I2C, LCD I2C, filtrage logiciel, détection de gestes), ce projet démontre les bases essentielles pour comprendre :

- la communication capteur-microcontrôleur,
- le traitement des signaux inertIELS,
- la visualisation de données embarquées.

Il peut servir de base pédagogique pour les débutants, mais également de point de départ pour des systèmes plus avancés comme les interfaces gestuelles, les dispositifs portables, les robots autonomes ou les systèmes d'assistance.

Enfin, cette réalisation met en avant l'importance d'une documentation rigoureuse, d'une approche expérimentale structurée et d'une maîtrise progressive des outils matériels et logiciels du domaine.