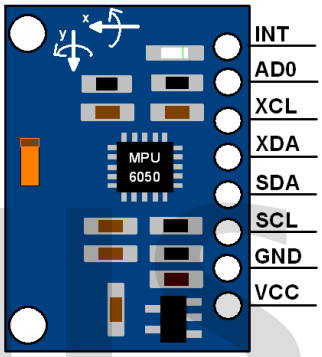
EPISODE 4

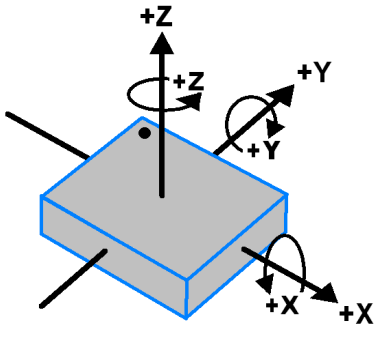
GY-521 sensor module

The GY-521 module is a breakout board for the MPU-6050 MEMS (Micro-electromechanical systems) that features a 3-axis gyroscope, a 3-axis ac-celerometer, a digital motion processor (DMP), and a temperature sensor. The digital motion processor can be used to process complex algorithms directly on the board. Usually, the DMP processes algorithms that turn the raw values from the sensors into stable position data. The sensor values are retrieved by using the I2C serial data bus, which requires only two wires (SCL and SDA).



3-Axis Gyroscope

The MPU6050 consists of a 3-axis Gyroscope with Micro Electro Mechanical System(MEMS) technology. It is used to detect rotational velocity along the X, Y, Z axes.

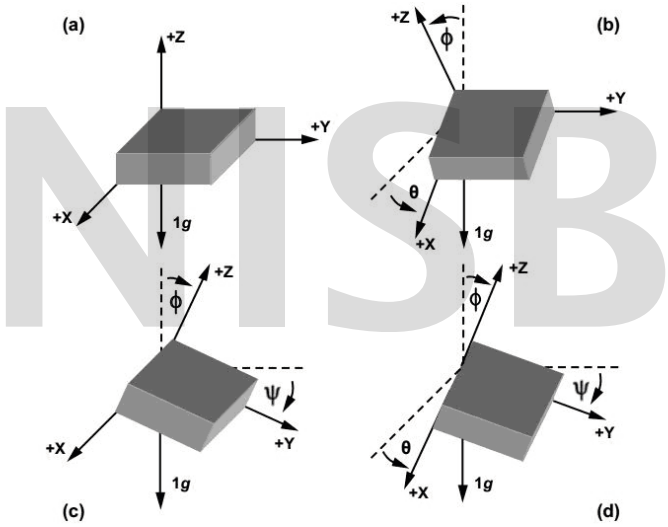


MPU-6050
Orientation & Polarity of Rotation

- When the gyros are rotated about any of the sense axes, the Coriolis Effect causes a vibration that is detected by a MEM inside the module.
- The resulting signal is amplified, demodulated, and filtered to produce a voltage that is proportional to the angular rate.
- This voltage is digitised using 16-bit ADC to sample each axis.
- The full-scale range of outputs are ±250, ±500, ±1000, ±2000.
- It measures the angular velocity along each axis in degree per second unit.

3-Axis Accelerometer

It is used to detect angle of tilt or inclination along the X, Y and Z axes.



- Acceleration along the axes deflects the movable mass.
- The displacement of the moving mass unbalances the differential capacitor which results in sensor output. The output amplitude is proportional to acceleration.
- 16-bit ADC is used to get digitised output.
- The full-scale range of acceleration are ±2g, ±4g, ±8g, ±16g.
- It is measured in g (gravity force) unit.

- When a device is placed on a flat surface it will measure 0g on X and Y axis and +1g on Z axis.

DMP (Digital Motion Processor)

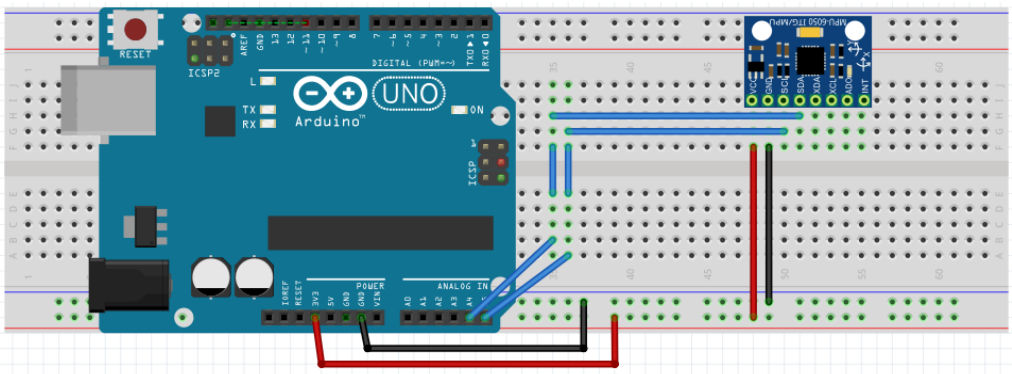
The embedded Digital Motion Processor (DMP) is used to compute motion processing algorithms. It takes data from gyroscope, accelerometer and additional 3rd party sensor such as magnetometer and processes the data. It provides motion data like roll, pitch, yaw angles, landscape and portrait sense etc. It minimises the processes of the host in computing motion data. The resulting data can be read from DMP registers.

On-chip Temperature Sensor

On-chip temperature sensor output is digitised using ADC. The reading from the temperature sensor can be read from the sensor data register.

Wiring Scheme

- VCC (The breakout board has a voltage regulator. Therefore, you can connect the board to 3.3V and 5V sources.)
- GND (Ground pin)
- SCL (Serial Clock Line of the I2C protocol.)
- SDA (Serial Data Line of the I2C protocol.)
- XDA (Auxiliary data => I2C master serial data for connecting the module to external sensors.)
- XCL (Auxiliary clock => I2C master serial clock for connecting the module to external sensors.)
- AD0 (If this pin is LOW, the I2C address of the board will be 0x68. Otherwise, if the pin is HIGH, the address will be 0x69.)
- INT (Interrupt digital output)



Wire Library

This library allows you to communicate with I2C / TWI devices. On the Arduino UNO, SDA (data line) is on analog input pin 4, and SCL (clock line) is on analog input pin 5. On the Arduino Mega, SDA is digital pin 20 and SCL is 21.

- **begin({address})**
- **requestFrom({address},{quantity},{stop})**
- **beginTransmission({address})**
- **endTransmission({stop})**
- **write({val/data/string})**
- **byte available()**
- **byte read()**
- **setClock(frequency)**
- **onReceive(handler)**
- **onRequest(handler)**

PROGRAM_1 [GY-521 basics]

```
#include<Wire.h>
const int MPU=0x68;
int16_t AcX,AcY,AcZ,Tmp,GyX,GyY,GyZ;
void setup(){
  Wire.begin();
  Wire.beginTransmission(MPU);
  Wire.write(0x6B);
  Wire.write(0);
```

```
  Wire.endTransmission(true);
  Serial.begin(9600);
}
void loop(){
  Wire.beginTransmission(MPU);
  Wire.write(0x3B);
  Wire.endTransmission(false);
  Wire.requestFrom(MPU,12,true);
  AcX=Wire.read()<<8|Wire.read();
  AcY=Wire.read()<<8|Wire.read();
  AcZ=Wire.read()<<8|Wire.read();
  GyX=Wire.read()<<8|Wire.read();
  GyY=Wire.read()<<8|Wire.read();
  GyZ=Wire.read()<<8|Wire.read();

  Serial.print("Accelerometer: ");
  Serial.print("X = "); Serial.print(AcX);
  Serial.print(" | Y = "); Serial.print(AcY);
  Serial.print(" | Z = "); Serial.println(AcZ);

  Serial.print("Gyroscope: ");
  Serial.print("X = "); Serial.print(GyX);
  Serial.print(" | Y = "); Serial.print(GyY);
  Serial.print(" | Z = "); Serial.println(GyZ);
  Serial.println(" ");
  delay(333);
}
```


PROGRAM_1.1 [GY-521 thrust vectoring]

```
#include "I2Cdev.h"
#include "MPU6050_6Axis_MotionApps20.h"
#include <Servo.h>
```

```
MPU6050 mpu;

Servo pitchServo;

Servo yawServo;// Declare servo object


#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 devStatus;      // return status after each device op-
eration (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

VectorFloat gravity;    // [x, y, z]            gravity vector

float ypr[3];           // [yaw, pitch, roll]    yaw/pitch/roll
container and gravity vector


void setup() {

  pitchServo.attach(9); // Init servo

  yawServo.attach(10);

  pitchServo.write(90);

  yawServo.write(90);

  Serial.begin(115200);
```

```
  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) {
      // Calibration Time: generate offsets and calibrate our MPU6050
```

```
    mpu.CalibrateAccel(6);
    mpu.CalibrateGyro(6);
    mpu.PrintActiveOffsets();
    // turn on the DMP, now that it's ready
    Serial.println(F("Enabling DMP..."));
    mpu.setDMPEnabled(true);
    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);
}


void loop() {
  // if programming failed, don't try to do anything
  if (!dmpReady) return;
  // wait for MPU interrupt or extra packet(s) available
  while (fifoCount < packetSize) {
```

```
    if (fifoCount < packetSize) {
        // try to get out of the infinite loop
        fifoCount = mpu.getFIFOCount();
    }
  }
  // get current FIFO count
  fifoCount = mpu.getFIFOCount();
  if(fifoCount < packetSize){}
    // check for overflow (this should never happen unless our code is too  inefficient)
  else if (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)
    }
    // read a packet from FIFO
  while(fifoCount >= packetSize){ // Lets catch up to NOW, someone is using the dreaded delay()!
    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;
  }
        // 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);
            yawServo.write(map(ypr[0]  *  180/M_PI,  -180,  180,  0,
180));
            Serial.print("\t");
            Serial.print(ypr[1] * 180/M_PI);
            pitchServo.write(map(ypr[1]  *  180/M_PI,  -180,  180,
180, 0));
            Serial.print("\t");
            Serial.println(ypr[2] * 180/M_PI);
               //pitchServo.write(map(ypr[2] * 180/M_PI, -180, 180,
0, 180));

            // blink LED to indicate activity
            blinkState = !blinkState;
            digitalWrite(LED_PIN, blinkState);
            delay(100);
}
```