Cheerful Maker
Makey

CATEGORY

GY-521 gyro and accel

Dual GY-521 - 두개 사용하기 (왼쪽 달팽이관 + 오른쪽 달팽이관)

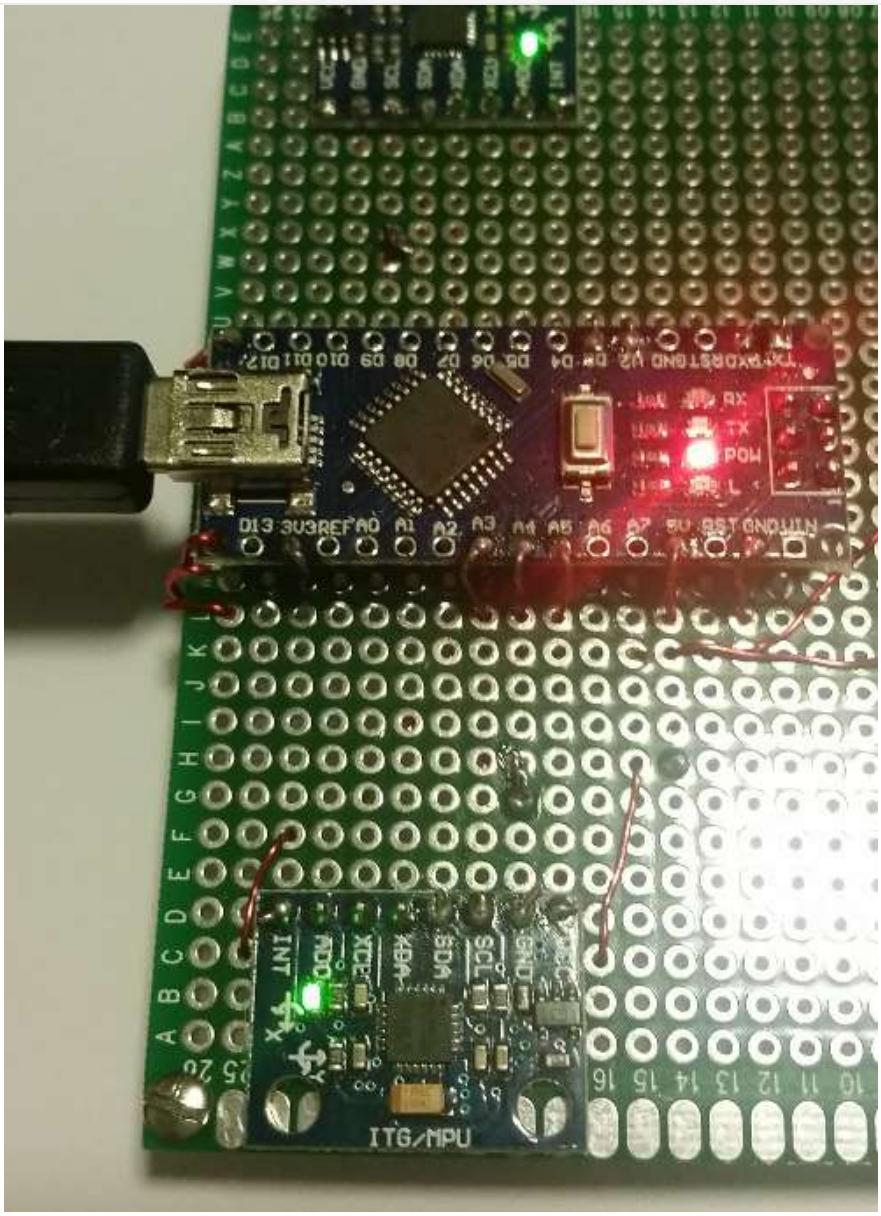
Cheerful Maker | 2016.01.31 18:06

불과 1년전에만 하더라도 6 DOF accelerometer 나 gyroscope 은 각기 대략 $40 이상되는 고가 부품이었는데, 요즘은 이 모든 기능을 갖추고 사용도 더 편리한 보드를 불과 $3 (삼천원)정도로 살 수 있으니 세상이 너무 빨리 바뀌었다. 심심해서라도 두개를 사용해도 전혀 아깝지 않은 수준.

Arduino Nano ($3 짜리 한개) + GY-521 6-Axis accelerometer / gyroscope ($3 짜리 두개)를 사용하는
동물과 비교하자면 두개의 귓속 달팽이관의 기능을 하는 센서를 추가하는 실험입니다.

아주 쉽게 따라올 수 있도록 회로그림도 추가하였습니다.

My Image



CIRCUIT IMAGE

(현재 인터넷에 돌아다니는 (Wrong) 잘못된 회로도. 여기서 수정할 부분들)

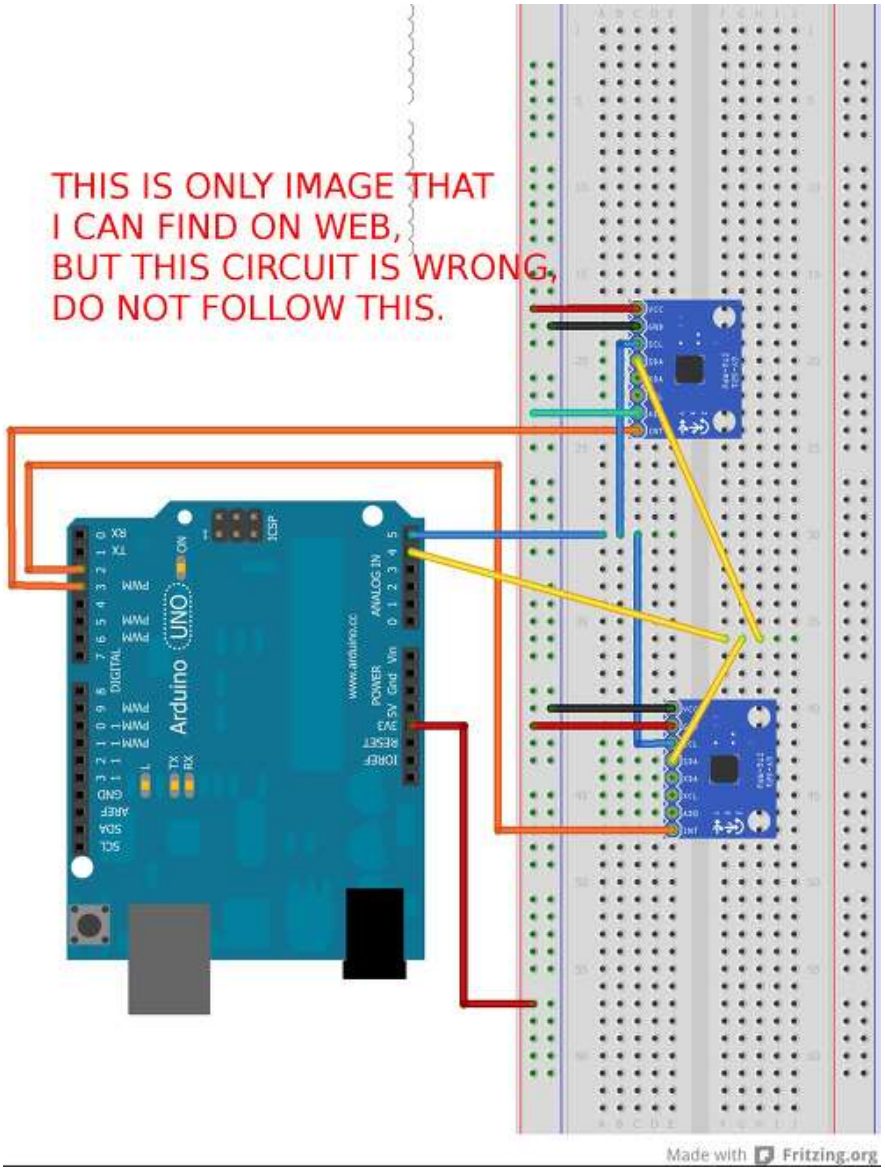
교정 1. VCC, GND 를 병렬로 각기 따로 연결해 줘야함. 그림에는 GND 로 연결이 없음.

교정 2. 한개의 AD0를 GND 로 연결해 주고 (0x68)
다른 한개의 AD0를 3V3 로 연결해 줘야 (0x69) 로 각기 다른 I2C를 인식됩니다.

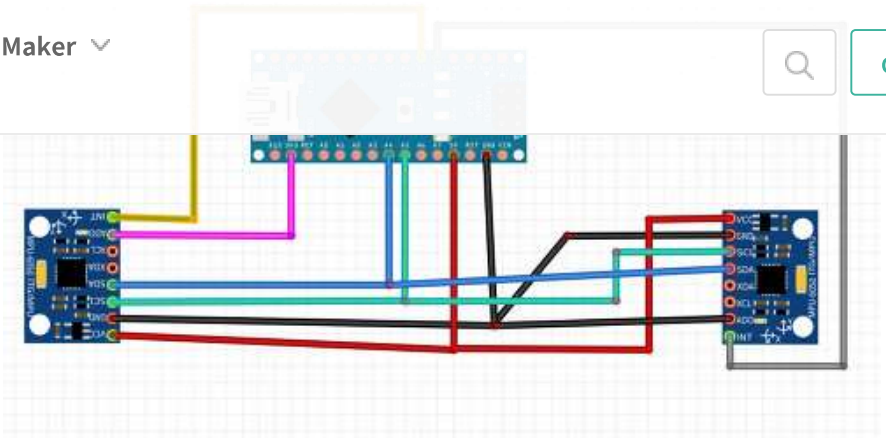
(Following circuit image from google image search is WRONG, needs to fix as follow)
Fix needed from bottom images
1. VCC, GND must be connected to arduino seperately.

Cheerful Maker
Makey

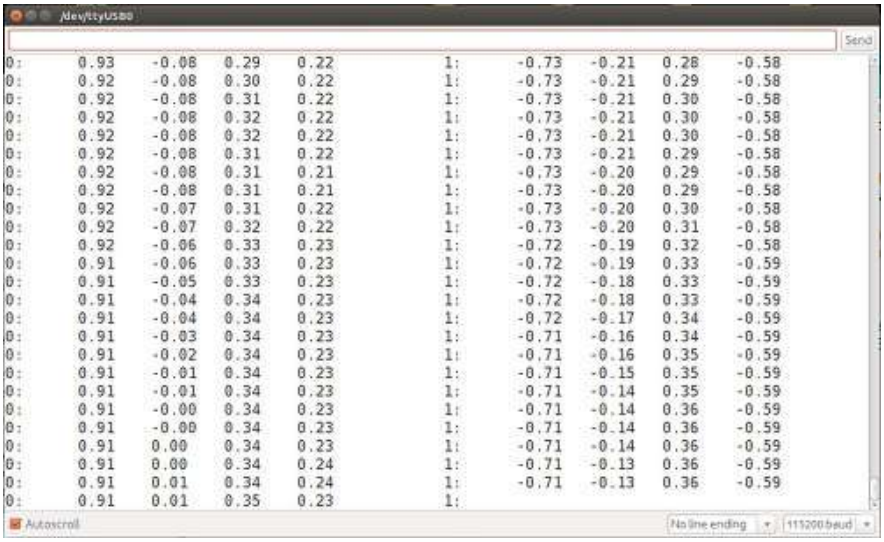
Without these settings, two chips will not be recognized as a seperate chips.



Fixed Circuit Image (교정된 회로도) - click to enlarge



실험결과 : (두개의 센서에서 실시간 자료받는 스크린)



```
0:    0.93   -0.08   0.29   0.22        1:   -0.73   -0.21   0.28   -0.58
0:    0.92   -0.08   0.30   0.22        1:   -0.73   -0.21   0.29   -0.58
0:    0.92   -0.08   0.31   0.22        1:   -0.73   -0.21   0.30   -0.58
0:    0.92   -0.08   0.32   0.22        1:   -0.73   -0.21   0.30   -0.58
0:    0.92   -0.08   0.32   0.22        1:   -0.73   -0.21   0.30   -0.58
0:    0.92   -0.08   0.31   0.22        1:   -0.73   -0.21   0.29   -0.58
0:    0.92   -0.08   0.31   0.21        1:   -0.73   -0.20   0.29   -0.58
0:    0.92   -0.08   0.31   0.21        1:   -0.73   -0.20   0.29   -0.58
0:    0.92   -0.07   0.31   0.22        1:   -0.73   -0.20   0.30   -0.58
0:    0.92   -0.07   0.32   0.22        1:   -0.73   -0.20   0.31   -0.58
0:    0.92   -0.06   0.33   0.23        1:   -0.72   -0.19   0.32   -0.58
0:    0.91   -0.06   0.33   0.23        1:   -0.72   -0.19   0.33   -0.59
0:    0.91   -0.05   0.33   0.23        1:   -0.72   -0.18   0.33   -0.59
0:    0.91   -0.04   0.34   0.23        1:   -0.72   -0.18   0.33   -0.59
0:    0.91   -0.04   0.34   0.23        1:   -0.72   -0.17   0.34   -0.59
0:    0.91   -0.03   0.34   0.23        1:   -0.71   -0.16   0.34   -0.59
0:    0.91   -0.02   0.34   0.23        1:   -0.71   -0.16   0.35   -0.59
0:    0.91   -0.01   0.34   0.23        1:   -0.71   -0.15   0.35   -0.59
0:    0.91   -0.01   0.34   0.23        1:   -0.71   -0.14   0.35   -0.59
0:    0.91   -0.00   0.34   0.23        1:   -0.71   -0.14   0.36   -0.59
0:    0.91   -0.00   0.34   0.23        1:   -0.71   -0.14   0.36   -0.59
0:    0.91    0.00   0.34   0.23        1:   -0.71   -0.14   0.36   -0.59
0:    0.91    0.00   0.34   0.24        1:   -0.71   -0.13   0.36   -0.59
0:    0.91    0.01   0.34   0.24        1:   -0.71   -0.13   0.36   -0.59
0:    0.91    0.01   0.35   0.23        1:
```

여기 두 센서에서 나오는 값이 큰 차이가 있는 이유는 두 부품이 같은 방향을 향해 있지 않기때문. - Parts' different physical orientation makes different sensor readings.

(*) 코드는 아래와 같습니다.

```
// Modified by Henry Kim

// Original Code is from the person below,
// I2C device class (I2Cdev) demonstration Arduino sketch for MPU6050 class using DMP (MotionApps v2.0)
// 6/21/2012 by Jeff Rowberg <jeff@rowberg.net>


// Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE implementation
```

```
// 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_1(0x68); //Primer GY-521
MPU6050 mpu_2(0x69); //Segundo GY-521

/* =========================================================================
  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 UNO's
  external interrupt digital I/O pin 2.
 * ========================================================================= */

// 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


// MPU control/status vars
bool    dmpReady = false; // set true if DMP init was successful

uint8_t  mpuIntStatus_1;   // holds actual interrupt status byte from MPU
uint8_t  devStatus_1;     // return status after each device operation (0 = success, !0 = error)
uint16_t packetSize_1;    // expected DMP packet size (default is 42 bytes)
uint16_t fifoCount_1;     // count of all bytes currently in FIFO
uint8_t  fifoBuffer_1[256];// FIFO storage buffer

uint8_t  mpuIntStatus_2;   // holds actual interrupt status byte from MPU
uint8_t  devStatus_2;     // return status after each device operation (0 = success, !0 = error)
uint16_t packetSize_2;    // expected DMP packet size (default is 42 bytes)
```

```
String  tempStr;

// orientation/motion vars
Quaternion q1;          // [w, x, y, z]      quaternion container
Quaternion q2;          // [w, x, y, z]      quaternion container

uint8_t testigo = 1;

boolean mpu1_listo;
boolean mpu2_listo;

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

volatile bool mpuInterrupt_1 = false;     // indicates whether MPU interrupt pin has gone high
volatile bool mpuInterrupt_2 = false;     // indicates whether MPU interrupt pin has gone high

// Interrupt for MPU 1
void dmp_1_DataReady() {
  mpuInterrupt_1 = true;
}
// Interrupt for  MPU 2
void dmp_2_DataReady() {
  mpuInterrupt_2 = 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)
```

```
  // initialize device
  Serial.println(F("Initializing I2C devices..."));
  mpu_1.initialize();
  mpu_2.initialize();

  // verify connection
  Serial.println(F("Testing device connections..."));
  Serial.println(mpu_1.testConnection() ? F("MPU6050_1 connection successful") : F("MPU6050_1 connection failed"));
  Serial.println(mpu_2.testConnection() ? F("MPU6050_2 connection successful") : F("MPU6050_2 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_1 = mpu_1.dmpInitialize();
  devStatus_2 = mpu_2.dmpInitialize();

  // make sure it worked (returns 0 if so)
  if (devStatus_1 == 0) {
    // turn on the DMP, now that it's ready
    Serial.println(F("Enabling DMP..."));
    mpu_1.setDMPEnabled(true);

    // enable Arduino interrupt detection
    Serial.println(F("Enabling interrupt detection (Arduino external interrupt 0)..."));
    attachInterrupt(0, dmp_1_DataReady, RISING); // Utilizamos la primera interrupción externa (número 0) que está en el pin digital 2
                                                 // Cuando la interrupción tiene lugar invoca la función "dmp_1_DataReady"
                                                 // RISING dispara la interrupción cuando el pin pasa de valor alto (HIGH) a bajo (LOW)
    mpuIntStatus_1 = mpu_1.getIntStatus();

    // set our DMP Ready flag so the main loop() function knows it's okay to use it
```

```
    // get expected DMP packet size for later comparison
    packetSize_1 = mpu_1.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 1 Initialization failed (code "));
    Serial.print(devStatus_1);
    Serial.println(F(")"));
  }

  if (devStatus_2 == 0) {
    // turn on the DMP, now that it's ready
    Serial.println(F("Enabling DMP..."));
    mpu_2.setDMPEnabled(true);

    // enable Arduino interrupt detection
    Serial.println(F("Enabling interrupt detection (Arduino external interrupt 0)..."));
    attachInterrupt(1, dmp_2_DataReady, RISING); // Utilizamos la segunda interrupción externa (número 1)
que está en el pin digital 3
                                 // Cuando la interrupción tiene lugar invoca la función "dmp_1_DataReady"
                                 // RISING dispara la interrupción cuando el pin pasa de valor alto (HIGH) a bajo
(LOW)
    mpuIntStatus_2 = mpu_2.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_2 = mpu_2.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 2 Initialization failed (code "));
    Serial.print(devStatus_2);
```

```
// ================================================================
// ===               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_1 && fifoCount_1 < packetSize_1)
    || (!mpuInterrupt_2 && fifoCount_2 < packetSize_2) ){
    // other program behavior stuff here
    //delay (1000);
    // .
    // 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
    // .
  }

  mpu1_listo=(!(!mpuInterrupt_1 && fifoCount_1 < packetSize_1) );
  mpu2_listo=(!(!mpuInterrupt_2 && fifoCount_2 < packetSize_2) );

  if (mpu1_listo) {

    if ((!mpu2_listo)||(testigo == 1)) {

      // reset interrupt flag and get INT_STATUS byte
      mpuInterrupt_1 = false;
      mpuIntStatus_1 = mpu_1.getIntStatus();

      // get current FIFO count
      fifoCount_1 = mpu_1.getFIFOCount();

      // check for overflow (this should never happen unless our code is too inefficient)
      if ((mpuIntStatus_1 & 0x10) || fifoCount_1 == 1024) {
```

```
      // otherwise, check for DMP data ready interrupt (this should happen frequently)

    } else if (mpuIntStatus_1 & 0x02) {

      // wait for correct available data length, should be a VERY short wait
      while (fifoCount_1 < packetSize_1) fifoCount_1 = mpu_1.getFIFOCount();

      // read a packet from FIFO
      if (packetSize_1 >= 64) Serial.println("mierda");
      mpu_1.getFIFOBytes(fifoBuffer_1, packetSize_1);

      // track FIFO count here in case there is > 1 packet available
      // (this lets us immediately read more without waiting for an interrupt)
      fifoCount_1 -= packetSize_1;

      #ifdef OUTPUT_READABLE_QUATERNION
        // display quaternion values in easy matrix form: w x y z
        mpu_1.dmpGetQuaternion(&q1, fifoBuffer_1);
        tempStr = "  0:\t";
        tempStr += q1.w;
        tempStr += "\t";
        tempStr += q1.x;
        tempStr += "\t";
        tempStr += q1.y;
        tempStr += "\t";
        tempStr += q1.z;
        tempStr += "   ";
      #endif
    }
  }
}

if (mpu2_listo) {

  if ((!mpu1_listo)||(testigo == 2)){

    // reset interrupt flag and get INT_STATUS byte
    mpuInterrupt_2 = false;
```

```
    mpuIntStatus_2 = mpu_2.getIntStatus();
    // get current FIFO count
    fifoCount_2 = mpu_2.getFIFOCount();

    // check for overflow (this should never happen unless our code is too inefficient)
    if ((mpuIntStatus_2 & 0x10) || fifoCount_2 == 1024) {
      // reset so we can continue cleanly
      mpu_2.resetFIFO();
      Serial.println(F("FIFO 2 overflow!"));

      // otherwise, check for DMP data ready interrupt (this should happen frequently)
    } else if (mpuIntStatus_2 & 0x02) {
      // wait for correct available data length, should be a VERY short wait
      while (fifoCount_2 < packetSize_2) fifoCount_2 = mpu_2.getFIFOCount();

      // read a packet from FIFO
      if (packetSize_2 >= 64) Serial.println("mierda");
      mpu_2.getFIFOBytes(fifoBuffer_2, packetSize_2);

      // track FIFO count here in case there is > 1 packet available
      // (this lets us immediately read more without waiting for an interrupt)
      fifoCount_2 -= packetSize_2;

      #ifdef OUTPUT_READABLE_QUATERNION
        // display quaternion values in easy matrix form: w x y z
        mpu_2.dmpGetQuaternion(&q2, fifoBuffer_2);
        tempStr += "\t1:\t";
        tempStr += q2.w;
        tempStr += "\t";
        tempStr += q2.x;
        tempStr += "\t";
        tempStr += q2.y;
        tempStr += "\t";
        tempStr += q2.z;
        Serial.println(tempStr);
       #endif
     }
   }
}

 if (testigo==1) {
```

```
    testigo = 1;
    mpu_2.resetFIFO();
  }
}
```


'GY-521 gyro and accel' 카테고리의 다른 글

1개의 GY-521을 Arduino Nano에 연결하는 경우. (0)

Dual GY-521 - 두개 사용하기 (왼쪽 달팽이관 + ... (0)