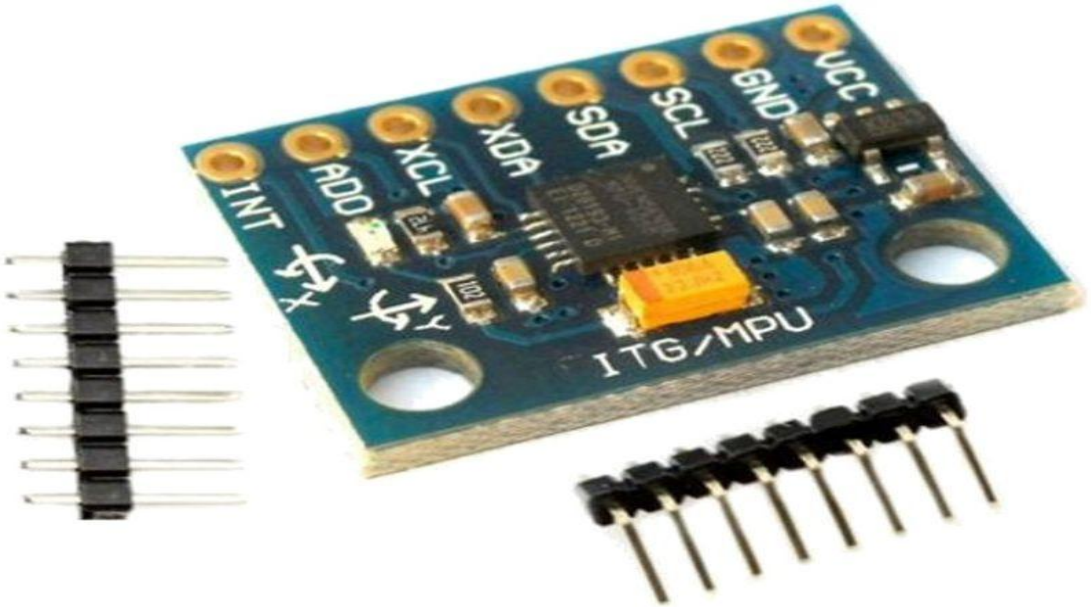
I.I.S.S. FOSSATI – DA PASSANO
LA SPEZIA

Relazione sul dispositivo GY-521(mpu-6050) con funzionalità di giroscopio/accelerometro.

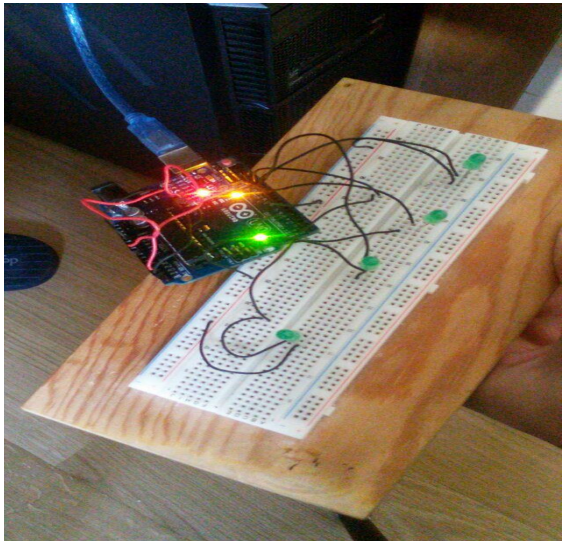


INDICE:

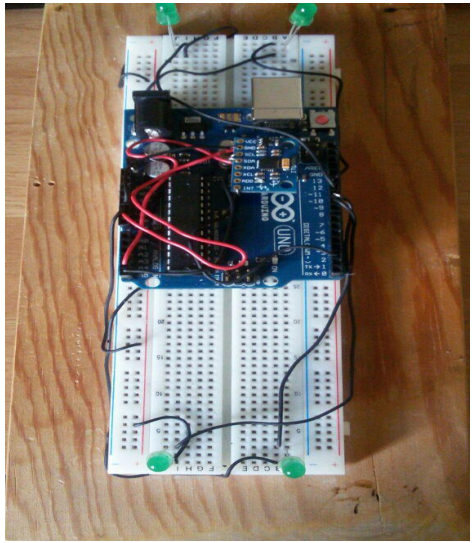
INTRODUZIONE

La relazione che segue è stata stesa da Nardi,Giambrocono,Mele,Labella,classe 3R, ha lo scopo di illustrare attraverso immagini la realizzazione di un circuito che permette, a seconda dell'inclinazione del chip GY-521, l'accensione di led ad esso collegati.

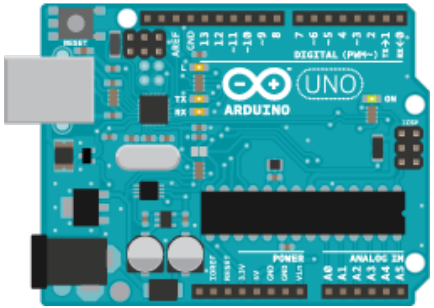
PRIMA



DOPO



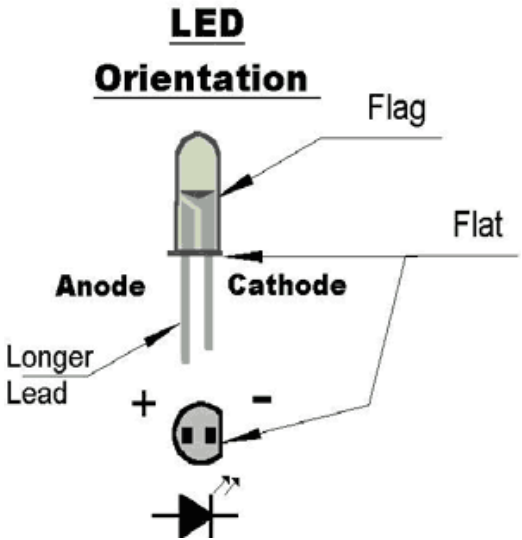
MATERIALE UTTILIZZATO:



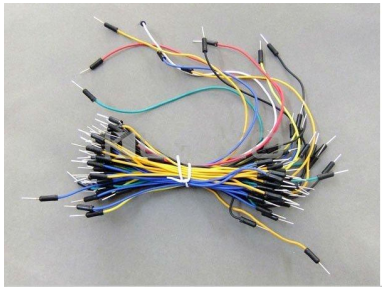
Arduino uno:

Arduino uno è una scheda elettronica di piccole dimensioni, con un microcontrollore e circuiteria di contorno, che permette la realizzazione di molti progetti. L'Arduino necessita di programmazione MicroC, derivato del linguaggio C.

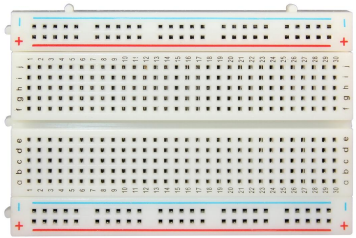
Led:



CAVI DI RAME RIGIDO (d=0.25mm):



BREADBOARD:



SPECIFICHE GY-521:

Ecco alcune caratteristiche del sensore **MPU-6050**:

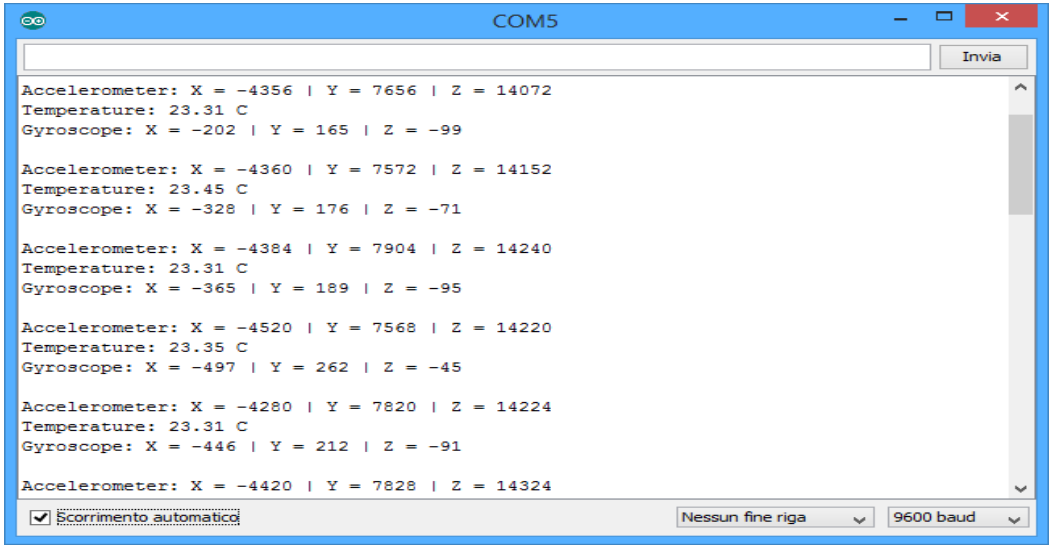
- Chip con Convertitore AD a 16 bit Integrato
- Range di misura giroscopio: ±250, 500, 1000 e 2000°/s
- Range di misura accelerometro: +2, +4 , +8 , +16 g
- Interfaccia: I²C
- Alimentazione: da 3V a 5VPotete trovare il datasheet del MPU-6050

GY-521(**MPU-6050):**

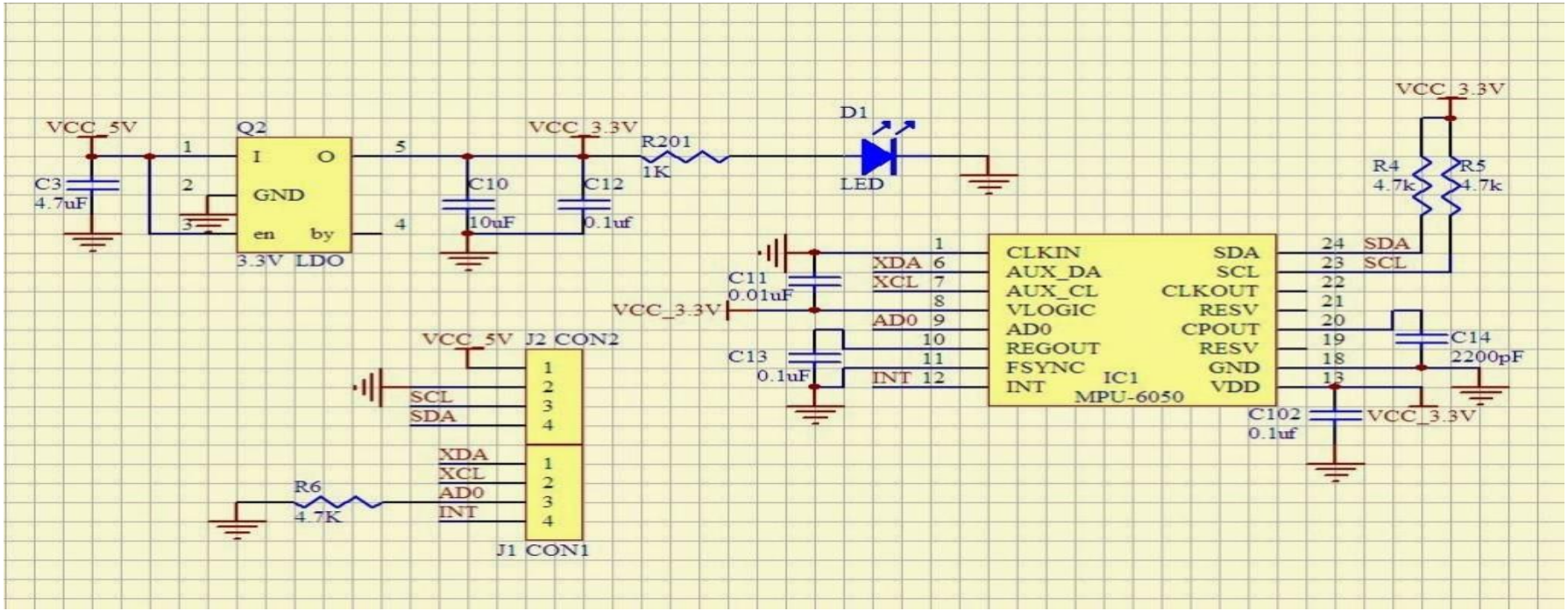
Il sensore **InvenSense MPU-6050** contiene, in un singolo integrato, un accelerometro MEMS a 3 assi ed un giroscopio MEMS a 3 assi. Con lo giroscopio possiamo misurare l' accelerazione angolare di un corpo su di un proprio asse, mentre con l'accelerometro possiamo misurare l' accelerazione di un corpo lungo una direzione. È molto preciso, in quanto ha un convertitore AD (da analogico a digitale) da 16 bit per ogni canale. Perciò cattura i canali x, y e z contemporaneamente. Il sensore possiede un protocollo di comunicazione standard I²C, quindi facile da interfacciare con il mondo arduino.

PROCEDIMENTO OPERATIVO

La realizzazione del progetto è avvenuta in 2 diverse fasi:
-Realizzazione del circuito;
-Programmazione dell'arduino.
Dopo la compilazione del programma e il caricamento su arduino ,sulla porta seriale del programma si poteva osservare questo solo (in caso di esatto funzionamento del programma):



Ecco lo schema elettrico del modulo **GY-521** per chi vuole costruirselo da solo:



PROGRAMMA ESECUTIVO IN C.

```
#include "I2Cdev.h"
#include "MPU6050_6Axis_MotionApps20.h"
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
    #include "Wire.h"
#endif
MPU6050 mpu;
#define OUTPUT_READABLE_YAWPITCHROLL
#define LED_PIN 13 // (Arduino is 13, Teensy is 11, Teensy++ is 6)
#define A 9
#define B 10
#define C 11
#define D 12
#define sens 0.05
bool blinkState = false;
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];
```

```
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
uint8_t teapotPacket[14] = { '$', 0x02, 0,0, 0,0, 0,0, 0,0, 0x00, 0x00, '\r', '\n' };
volatile bool mpuInterrupt = false;     // indicates whether MPU interrupt pin has gone high
void dmpDataReady() {
    mpuInterrupt = true;
}
void setup() {
    // 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
```

```
Serial.begin(115200);
    while (!Serial);
Serial.println(F("Initializing I2C devices..."));
    mpu.initialize();
 Serial.println(F("Testing device connections..."));
    Serial.println(mpu.testConnection() ? F("MPU6050 connection successful") : F("MPU6050
connection failed"));
 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
 Serial.println(F("Initializing DMP..."));
    devStatus = mpu.dmpInitialize();
 mpu.setXGyroOffset(220);
    mpu.setYGyroOffset(76);
    mpu.setZGyroOffset(-85);
    mpu.setZAccelOffset(1788); // 1688 factory default for my test chip
 if (devStatus == 0) {
        // turn on the DMP, now that it's ready
        Serial.println(F("Enabling DMP..."));
        mpu.setDMPEnabled(true);
```

```
Serial.println(F("Enabling interrupt detection (Arduino external interrupt 0)..."));
        attachInterrupt(0, dmpDataReady, RISING);
        mpuIntStatus = mpu.getIntStatus();
 Serial.println(F("DMP ready! Waiting for first interrupt..."));
        dmpReady = true;

        // get expected DMP packet size for later comparison
        packetSize = mpu.dmpGetFIFOPacketSize();
    } else {
            Serial.print(F("DMP Initialization failed (code "));
        Serial.print(devStatus);
        Serial.println(F(")"));
    }
 pinMode(LED_PIN, OUTPUT);
    pinMode(A, OUTPUT);
    pinMode(B, OUTPUT);
    pinMode(C, OUTPUT);
    pinMode(D, 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 (!mpuInterrupt && fifoCount < packetSize) {
}
 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();
```

```
 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
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[1] * 180/M_PI);
 Serial.print("\t");
            Serial.println(ypr[2] * 180/M_PI);
        #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);
 if(ypr[1]>sens && ypr[2]>sens)
       digitalWrite(A,HIGH);
       else
       digitalWrite(A,LOW);
```

```
 if(ypr[1]<sens && ypr[2]<sens)
      digitalWrite(B,HIGH);
      else
      digitalWrite(B,LOW);

      //led X
       if(ypr[1]>sens && ypr[2]<sens)
      digitalWrite(C,HIGH);
      else
      digitalWrite(C,LOW);

      //led X
       if(ypr[1]<sens && ypr[2]>sens)
      digitalWrite(D,HIGH);
      else
      digitalWrite(D,LOW);

}
}
```

COLLEGAMENTI CON ALTRE MATERIE:

INFORMATICA: la programmazione dell'arduino è stata realizzata in linguaggio "microC", che è derivato dal linguaggio "c" come il c++ che viene utilizzato in campo informatico.

INGLESE: il legame con la lingua inglese è indiscutibile in quanto, nel campo della programmazzione la maggior parte dei comandi dei vari linguaggi è espressa in tale lingua.

ITALIANO: l'italiano, in quanto nostra lingua è stata utilizzata nella compilazione della relazione che ha seguito l'uso della tecnica strutturale.

SISTEMI E RETI: tale materia è stata trattata in quanto arduino ha una porta seriale RS-232 che rende il transito di dati fra il collegamento di questi dispositivi e una porta USB-PLUG.

TECNOLOGIE:la stesura della relazione si è avvalsa della compressione delle foto argomento trattato nel corso dell'anno in tecnologie;inoltre ad esse si richiama il posizionamento di arduino su un chip megaAVR

Conclusione:

La costruzione del circuito è stato un lavoro veramente complesso, ma, dividendo il compito razionalmente, per ciò che riguarda il programma e la costruzione del circuito, siamo riusciti a collaborare risolvendo i problemi che di volta in volta si sono presentati.