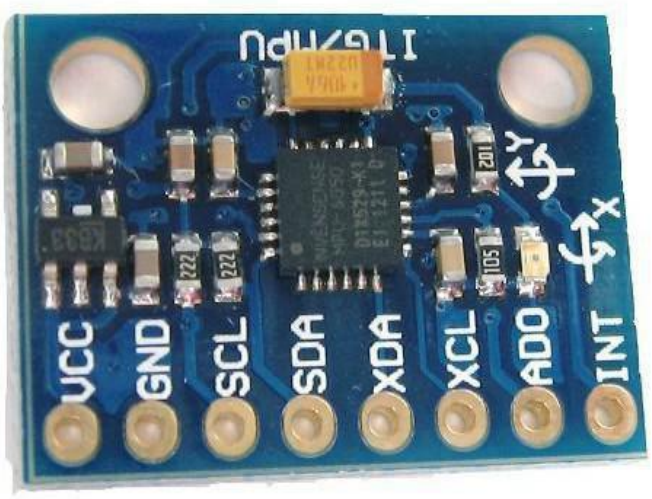
MPU-6050 Module

DESCRIPTION:

The MPU6050 contains both a 3-Axis Gyroscope and a 3-Axis accelerometer allowing measurements of both independently, but all based around the same axes, thus eliminating the problems of cross-axis errors when using separate devices.

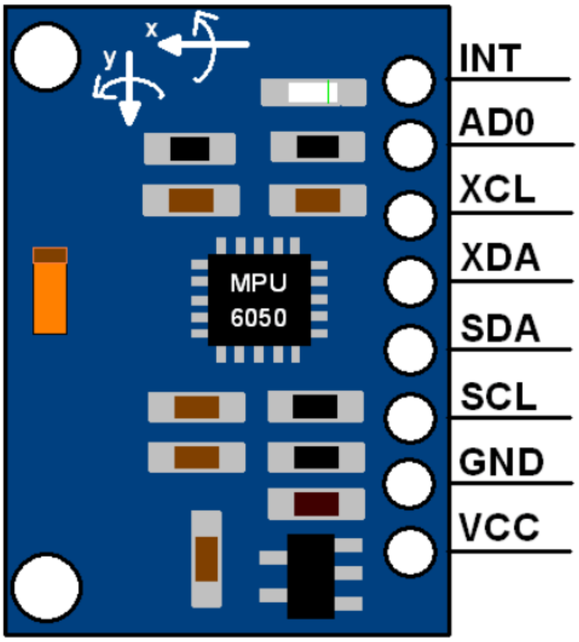


Specification：

- Accelerometer ranges: ±2, ±4, ±8, ±16g
- Gyroscope ranges: ± 250, 500, 1000, 2000 °/s
- Voltage range: 3.3V - 5V (the module include a low drop-out voltage regulator)

This simple module contains everything required to interface to the Arduino and other controllers via I2C (use the Wire Arduino library) and give motion sensing information for 3 axes - X, Y and Z.

MPU-6050 Module:



PIN CONFIGURATION:

INT: Interrupt digital output pin.

AD0: I2C Slave Address LSB pin. This is 0th bit in 7-bit slave address of device. If connected to VCC then it is read as logic one and slave address changes.

XCL: Auxiliary Serial Clock pin. This pin is used to connect other I2C interface enabled sensors SCL pin to MPU-6050.

XDA: Auxiliary Serial Data pin. This pin is used to connect other I2C interface enabled sensors SDA pin to MPU-6050.

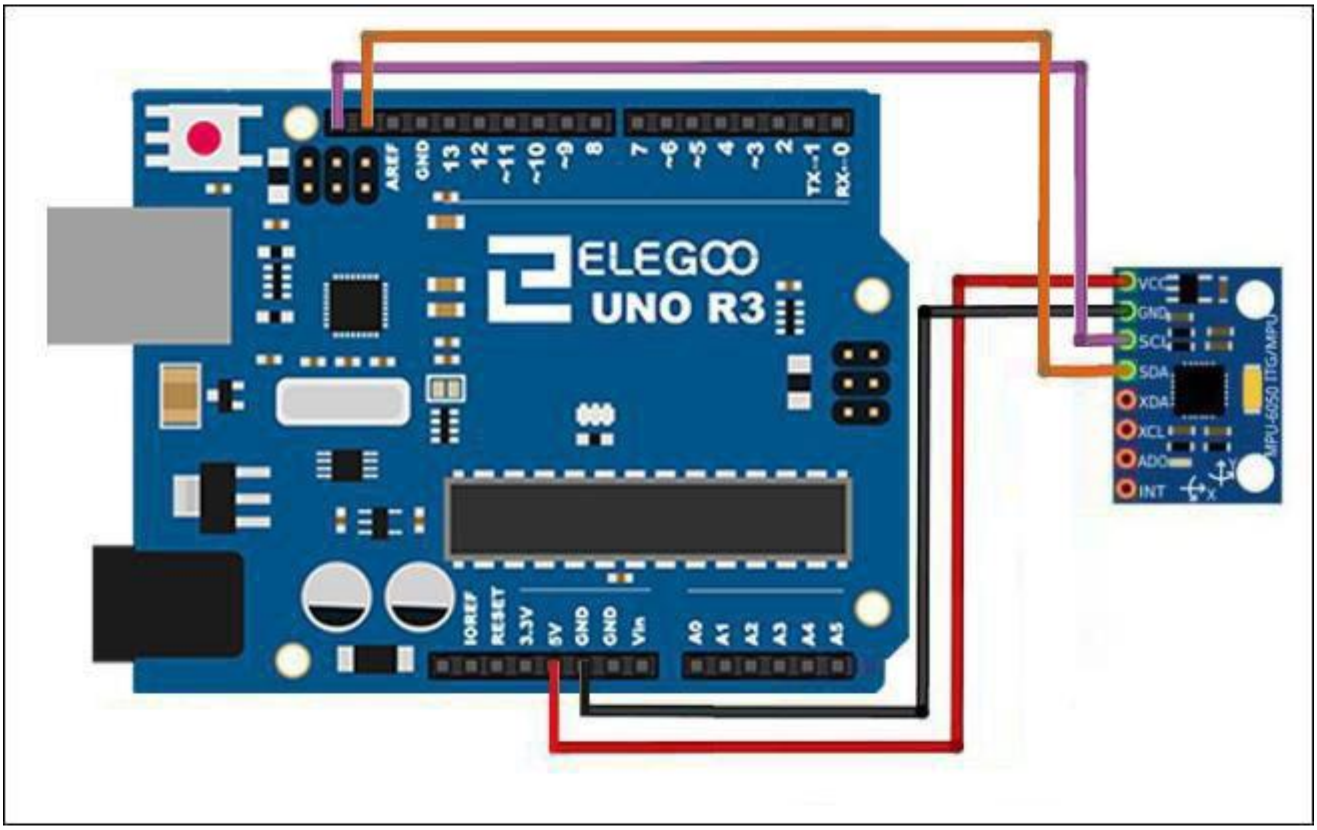
SCL: Serial Clock pin. Connect this pin to microcontrollers SCL pin.

SDA: Serial Data pin. Connect this pin to microcontrollers SDA pin.

GND: Ground pin. Connect this pin to ground connection.

VCC: Power supply pin. Connect this pin to +5V DC supply.

Example:



Code:

```
#include "Wire.h"
#include "I2Cdev.h"
#include "MPU6050_6Axis_MotionApps20.h"
//instantiate a MPU6050 object, the object name is mpu
MPU6050 mpu(0x68);
//statement MPU6050 control and state variable
bool dmpReady = false;    //set true if DMP init was successful
uint8_t mpuIntStatus;      //This variable is used to save the state when MPU6050 stop working
uint8_t devStatus;          //Return to equipment status, 0 for success, others for 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

//state direction and movement of variables:
Quaternion q;                    //quaternion variable W,X,Y,Z
VectorFloat gravity;       //gravity vector X，Y, Z
float ypr[3];                    // [yaw, pitch, roll] yaw/pitch/roll container and gravity vector

volatile bool mpuInterrupt = false;         // indicates whether MPU interrupt pin has gone high
void dmpDataReady()
{
    mpuInterrupt = true;
}
void setup()
{
    Serial.begin(9600); //Open the serial port and set the baud rate to 115200, upload the program to the Arduino IDE and observe th situation of the serial port

    //add the bus sequence of I2C
    Wire.begin();

    //Initial setup MPU6050
    Serial.println("Initializing I2C devices...");
    mpu.initialize();

    //verify connection
    Serial.println("Testing device connections...");
    Serial.println(mpu.testConnection() ? "MPU6050 connection successful":
```

```
"MPU6050 connection failed");

  delay(2); //delay 2ms

  //upload and configure DMP digital motion processing engine
  Serial.println("Initializing DMP...");
  devStatus = mpu.dmpInitialize(); //Return to DMP status, 0 for success, others for
error

  // if return to 0
  if (devStatus == 0)
  {
    // make DMP digital motion processing engine
    Serial.println("Enabling DMP...");
    mpu.setDMPEnabled(true);

    //Enabling the Arduino interrupt detection
    Serial.println("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("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("DMP Initialization failed (code ");
     Serial.print(devStatus);
     Serial.println(")");
  }
}
void loop()
{
  float alpha, omiga; //state two floating-point variables, alpha and omiga

  //if MPU6050 DMP status to error, the program stop working
  if (!dmpReady)
     return;

  // wait for MPU interrupt or extra packet(s) available
  if (!mpuInterrupt && fifoCount < packetSize)
     return;

  // 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("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;

     mpu.dmpGetQuaternion(&q, fifoBuffer);
     mpu.dmpGetGravity(&gravity, &q);
     mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);    //take three axis angle from the
DMP. they are Yaw, Pitch and Roll. put them into the succession of the array.Units:
radian
     alpha=-ypr[2] * 180/M_PI;

     omiga=mpu.getRotationX()/16.4; //configuration is 16. plus or minus2000°/s,
65536/4000
```

```
        Serial.print("Alpha ");

        Serial.print(alpha);

        Serial.print("\tOmiga ");

        Serial.println(omiga);


    }

}
```