CEG5205

AI Sensors & Virtual/ Augmented Reality Technologies

Lab 1 – Sensing Signal Processing; Interfacing to IoT; Python Basics

Lecturer: prof. Chengkuo Lee (elelc@nus.edu.sg)

Lab1 GA: Xinge Guo

Revolution of Internet-of-Things (IoT)

By 2020, there is
50 billion devices
connected to the internet

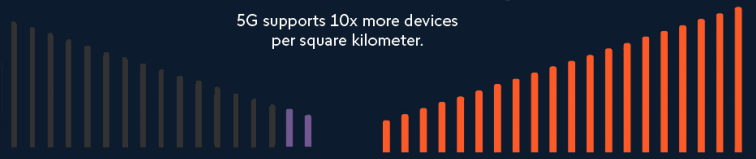
In the coming years,
40% of total data created
will be from sensors.

5G Revolution

In the coming years, the partnership between 5G and the IoT could bring about a boom in smart technologies

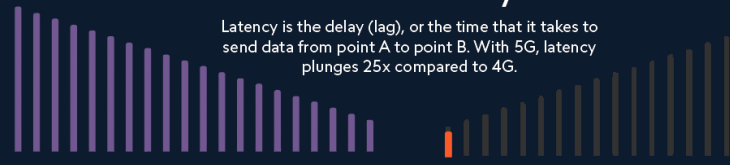
Increased connectivity

5G supports 10x more devices per square kilometer.



Lower latency

Latency is the delay (lag), or the time that it takes to send data from point A to point B. With 5G, latency plunges 25x compared to 4G.



Transport

Use cases:

Autonomous Vehicles | Smart Cities

Manufacturing

Use cases:

Augmented/ Virtual Reality | Smart Factories

Healthcare

Use cases:

Remote Robotic Surgery | Patient Monitoring

Sources: Allied Market Research; Samsung Insights; Swissquote; CNN; Gartner; Bitstew

IoT Empowered by Artificial Intelligence - AIoT

Programmable intelligence enabling devices to learn, reason, and process information like humans

4 Major Future AIoT Segments

Together, AI and IoT merge to create AIoT – a smart, connected network of devices that seamlessly communicate over powerful 5G networks – unleashing the power of data better and faster than ever

Smart Wearables

Monitor, track, and learn users' habits, health, etc.

Smart watches | AR/VR goggles | Wireless earbuds

Smart Cities

More convenient places, better urban planning, increased public safety

Smart energy grids | Smart streetlights | Smart public transportation

Smart Home

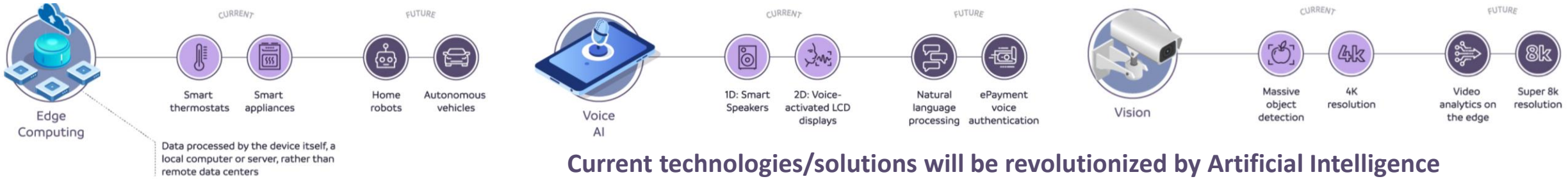
Learn and develop automated home "support" for users

Smart speakers | IoT appliances | Smart thermostats

Smart Industry

Optimized operations, logistics, and supply chain. Foresee and prevent errors.

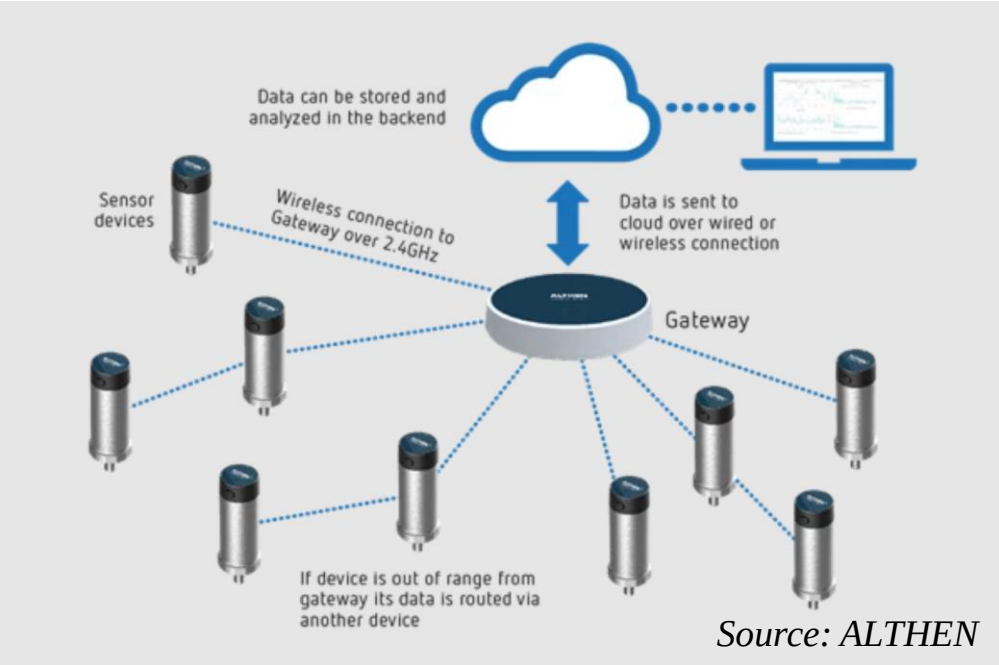
Autonomous manufacturing robots | Automated supply chain management | Predictive maintenance sensors



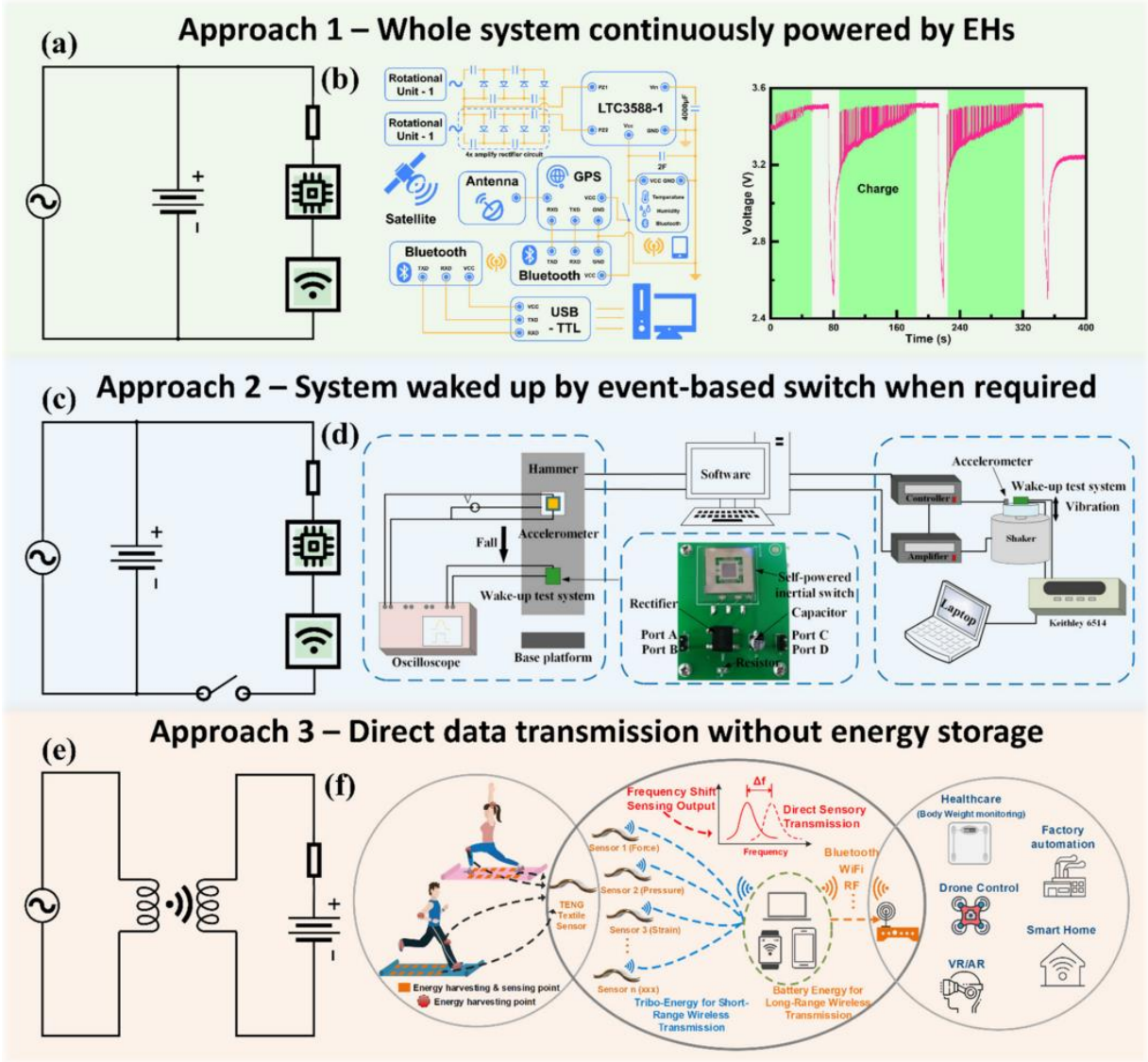
Current technologies/solutions will be revolutionized by Artificial Intelligence

Sources: CISCO; TSMC

Typical IoT System Structures

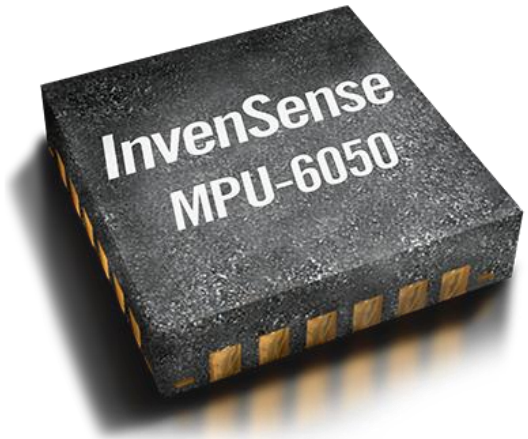


Source: ALTHEN

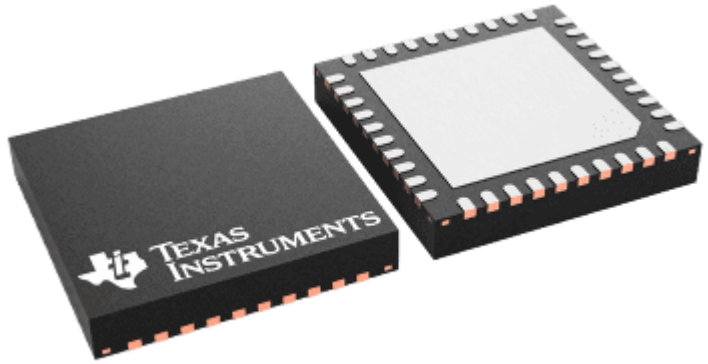


Nanomaterials, vol. 11, no. 11, 2975, 2021

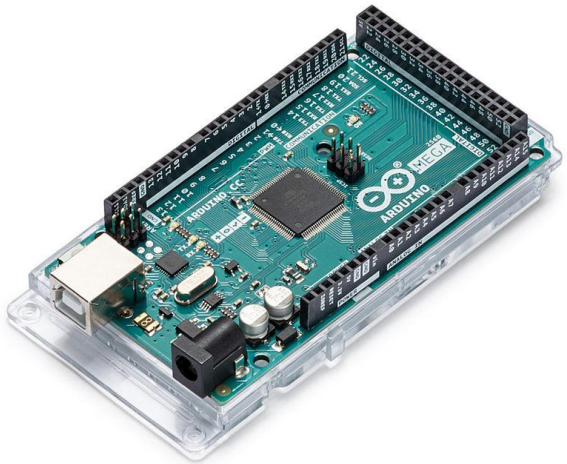
Devices Applied in Lab1



MPU-6050 Six-Axis
(Gyroscope & Accelerometer)
MEMS MotionTracking™ Devices



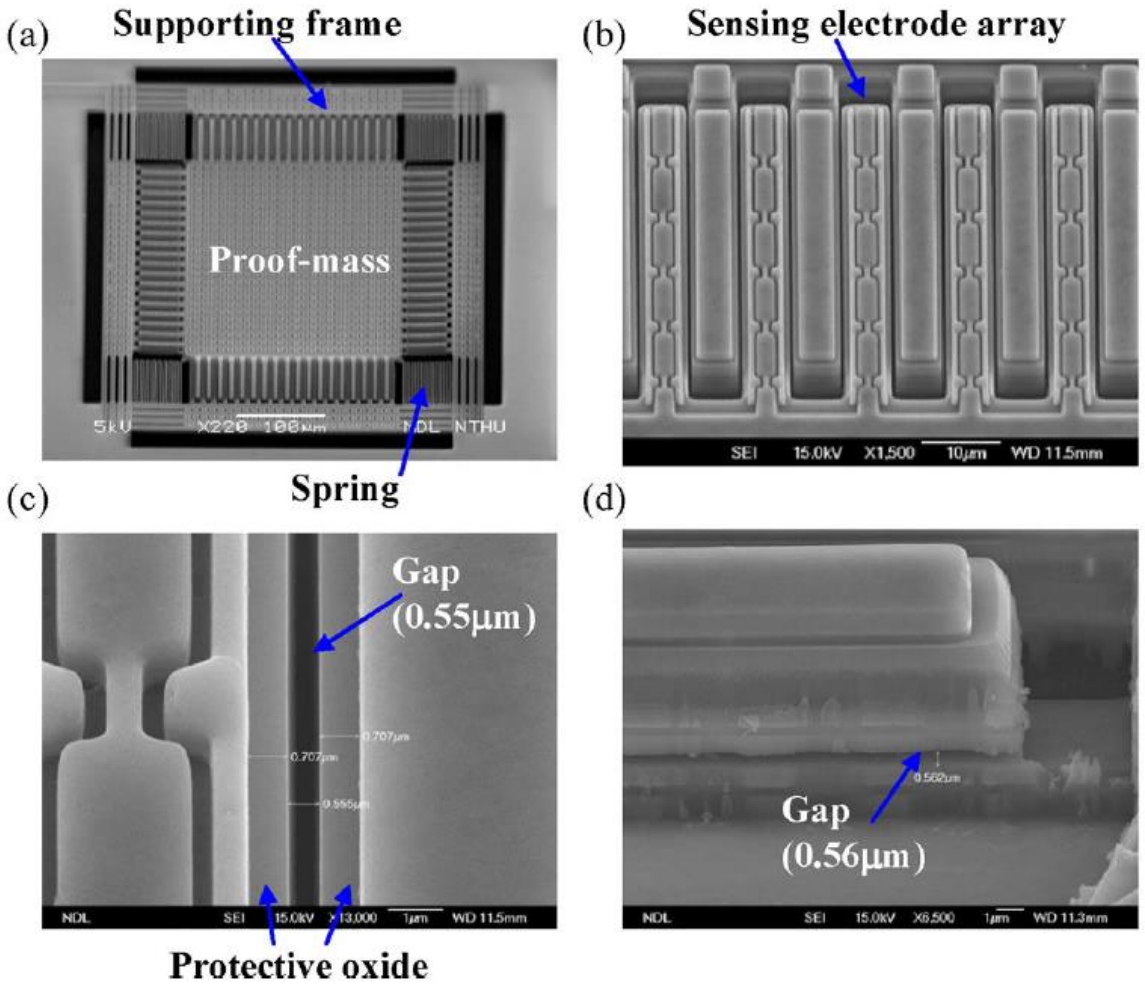
CC2530
Zigbee and IEEE 802.15.4 wireless MCU



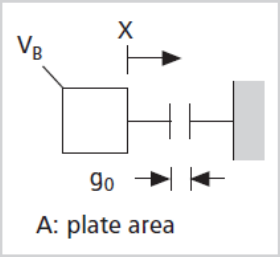
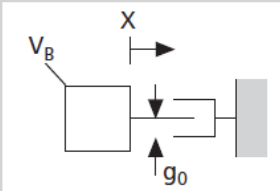
Arduino Mega 2560

MEMS Inertial sensor - Accelerometer

MEMS Inertial sensor - Accelerometer



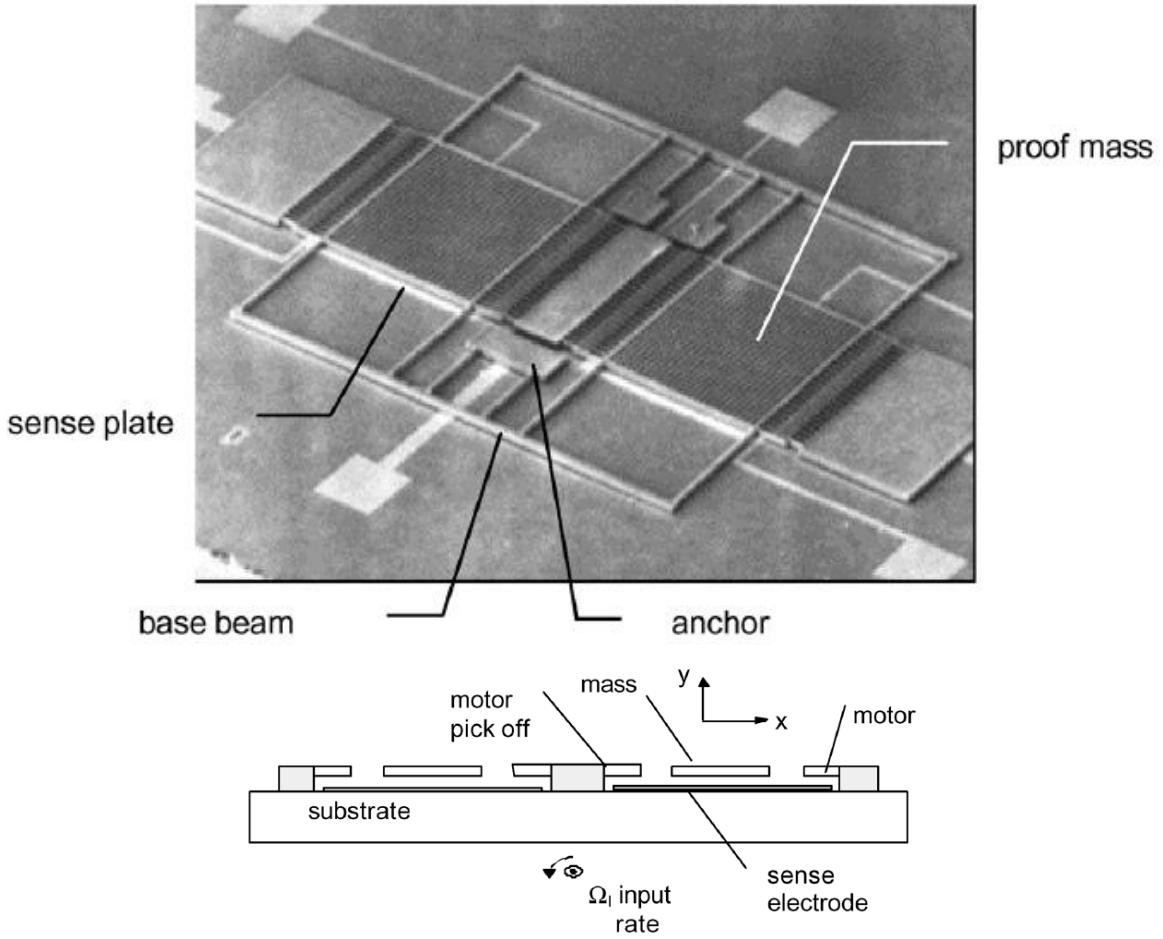
JMEMS, Vol 21, 1329-1337, 2012

		Capacitance	Charge transduction	Force transduction	Spring constant
		C(x)	ΔQ(Δx)	ΔF(Δv)	k_E
Parallel-plate capacitor	 A: plate area	$\frac{\varepsilon_0 A}{g_0 - x}$	$\frac{\varepsilon_0 A}{g_0^2} V_B \Delta x$	$\frac{\varepsilon_0 A}{g_0^2} V_B \Delta v$	$-\frac{\varepsilon_0 A}{g_0^3} V_B^2$
Comb-finger capacitor	 w: comb width l: comb overlap @ x=0	$\frac{\varepsilon_0 w(l+x)}{g_0}$	$\frac{\varepsilon_0 w}{g_0} V_B \Delta x$	$\frac{\varepsilon_0 w}{g_0} V_B \Delta v$	0

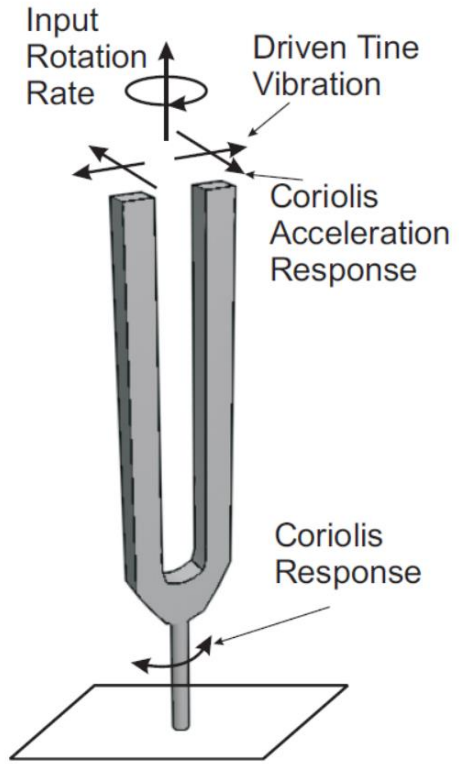
IEEE Comm. Mag., 100-109, 2013

MEMS Inertial sensor - Gyroscope

MEMS Inertial sensor - Gyroscope

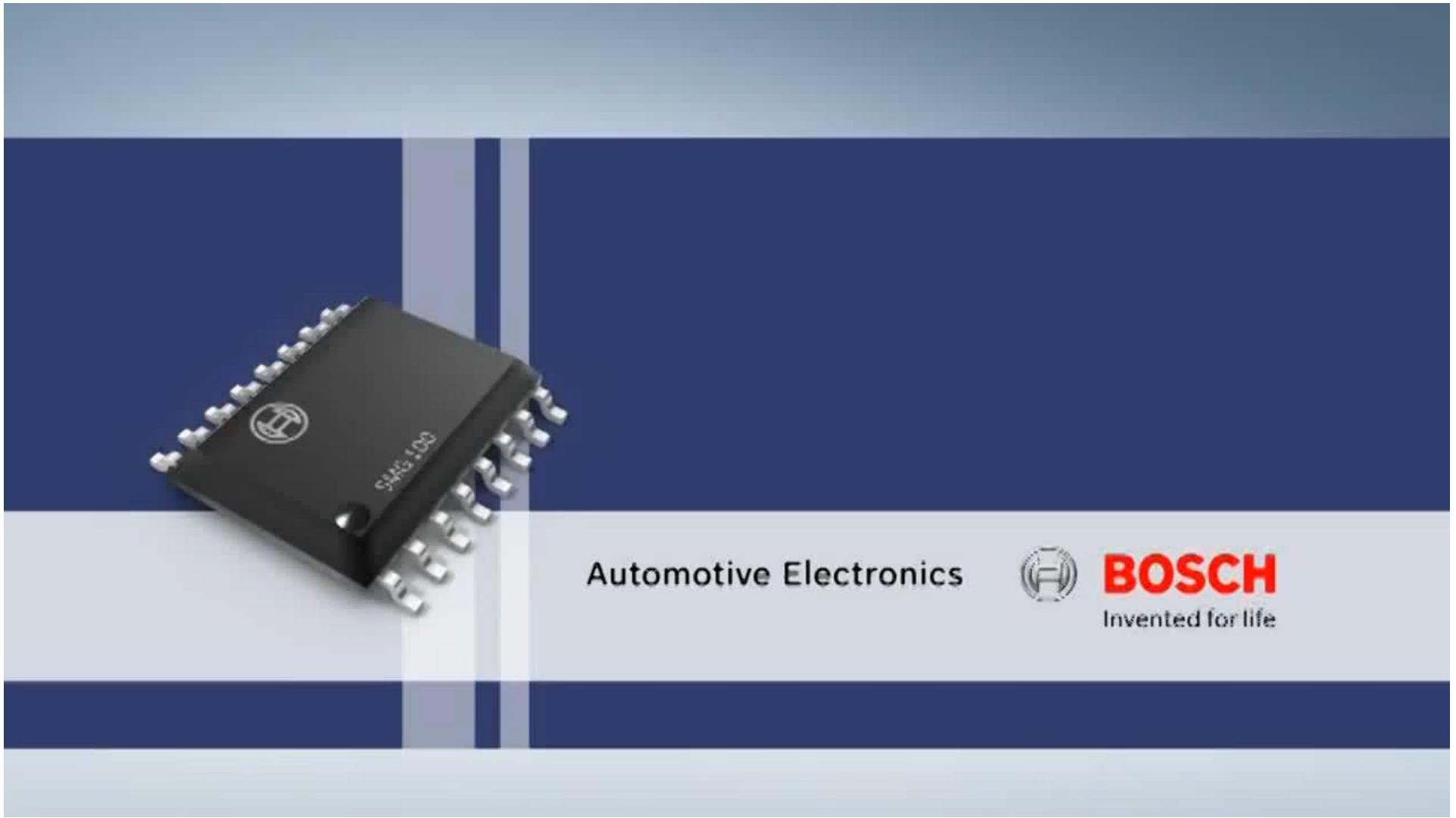


JMEMS, Vol 15, 479-491, 2006

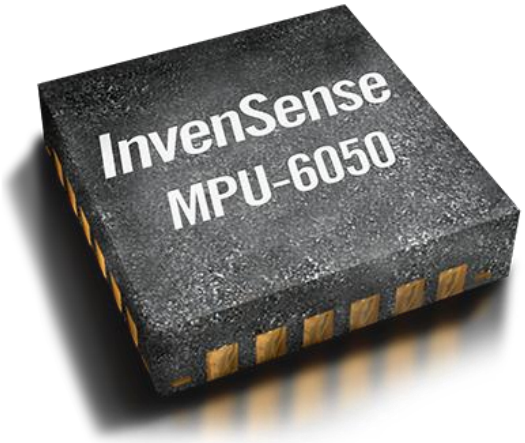


Coriolis force and tuning fork structure

MEMS Inertial sensor - Gyroscope



MPU-6050 Inertial Sensor



Two datasheet files are provided and can be downloaded through Canvas:

1. PS-MPU-6000A: Product Specification Datasheet

In this file, you can find:

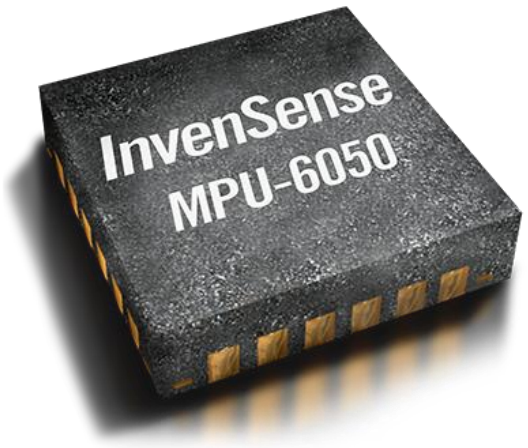
Accelerometer Specifications: Details such as range, sensitivity, noise, linearity, and other related parameters.
Gyroscope Specifications: Information on range, sensitivity, bias stability, etc.
Electrical Specifications: Details on power supply requirements, current consumption, voltage levels, etc.
Interface and Communication Protocols: Information on the supported communication interfaces, such as I2C, SPI, UART, etc., including pin configurations.
Typical Application Circuits: Example circuits or diagrams showing how the IMU can be integrated into a system.

2. RM-MPU-6000A: Register Map and Descriptions Datasheet

In this file, you can find:

Register Map: A table detailing the addresses and functions of each register within the IMU.
Register Addresses: The memory locations that are used to store or retrieve specific data from the sensor.
Bit Definitions: A description of individual bits within the register and their specific functions or meanings.
Default Values: The initial values that might be contained in each register upon power-up or reset.
Access Type: Specifies whether the register is read-only, write-only, or read/write.
Detailed Register Descriptions: A section providing an in-depth explanation of each register's functionality.
Functionality: Describes the specific purpose and operation of each register.
Usage Scenarios: Explains how the register can be used in different applications or under various conditions.

MPU-6050 Inertial Sensor



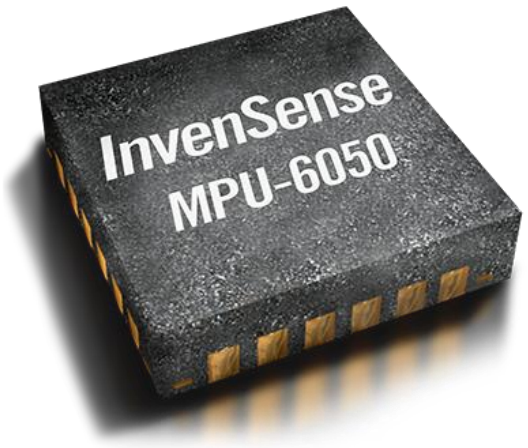
Accelerometer Features

The triple-axis MEMS accelerometer in MPU-60X0 includes a wide range of features:

- Digital-output triple-axis accelerometer with a programmable **full-scale range of ±2g, ±4g, ±8g, and ±16g**
- Integrated 16-bit ADCs enable simultaneous sampling of accelerometers while requiring no external multiplexer
- Accelerometer normal operating current: 500μA
- Low power accelerometer mode current: 10μA at 1.25Hz, 20μA at 5Hz, 60μA at 20Hz, 110μA at 40Hz
- Orientation detection and signaling
- Tap detection
- User-programmable interrupts
- Free-fall interrupt
- High-G interrupt
- Zero Motion/Motion interrupt
- User self-test

Sources:InvenSense

MPU-6050 Inertial Sensor



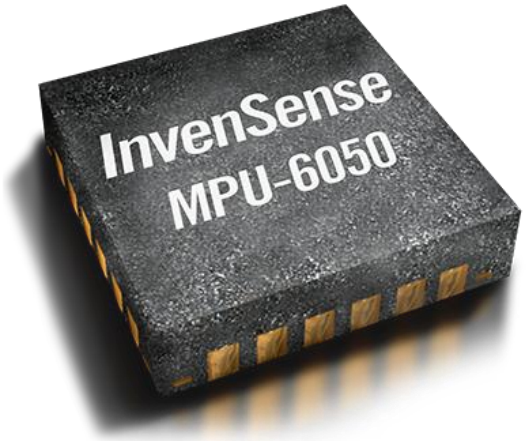
Gyroscope Features

The triple-axis MEMS gyroscope in the MPU-60X0 includes a wide range of features:

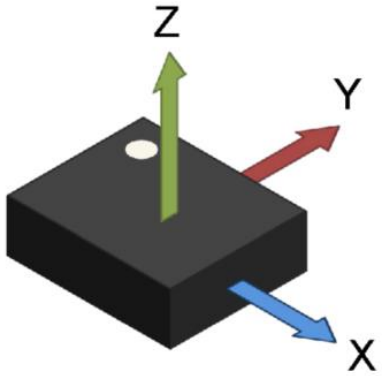
- Digital-output X-, Y-, and Z-Axis angular rate sensors (gyroscopes) with a user-programmable fullscale range of **±250, ±500, ±1000, and ±2000°/sec**
- External sync signal connected to the FSYNC pin supports image, video and GPS synchronization
- Integrated 16-bit ADCs enable simultaneous sampling of gyros
- Enhanced bias and sensitivity temperature stability reduces the need for user calibration
- Improved low-frequency noise performance
- Digitally-programmable low-pass filter
- Gyroscope operating current: 3.6mA
- Standby current: 5μA
- Factory-calibrated sensitivity scale factor
- User self-test

Sources:InvenSense

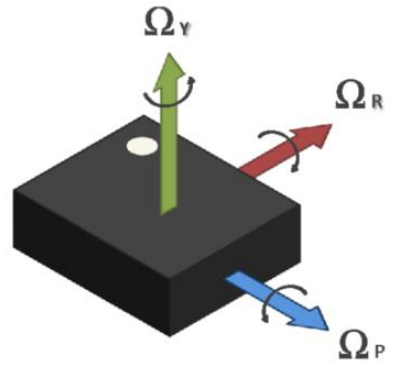
MPU-6050 Inertial Sensor



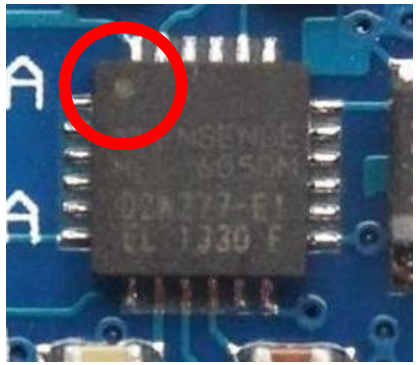
MPU-6050 Six-Axis
(Gyroscope & Accelerometer)
MEMS MotionTracking™ Devices



Direction of detectable acceleration (top view)



Direction of detectable angular rate (top view)

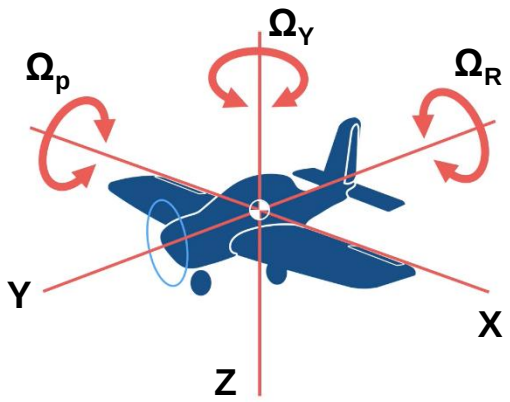


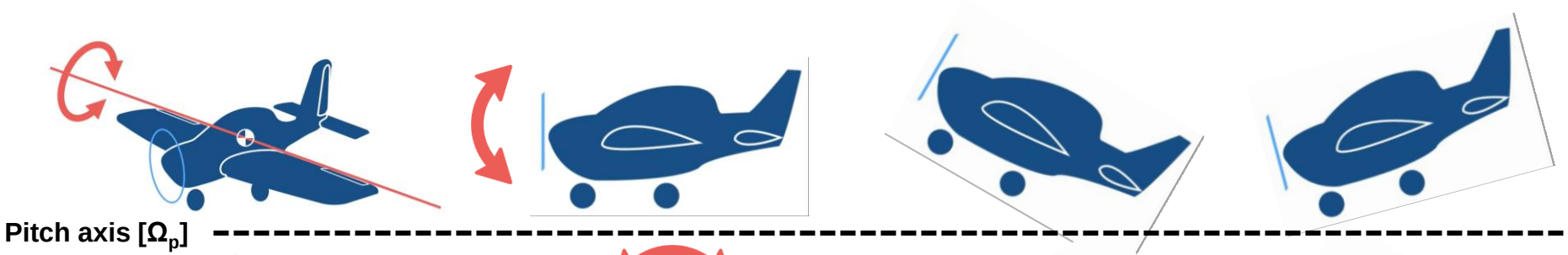
3-axis acceleration: X, Y, Z axis.

3-axis angular rate:
Pitch axis [Ω_p]: along X axis;
Roll axis [Ω_R]: along Y axis;
Yaw axis [Ω_Y]: along Z axis;

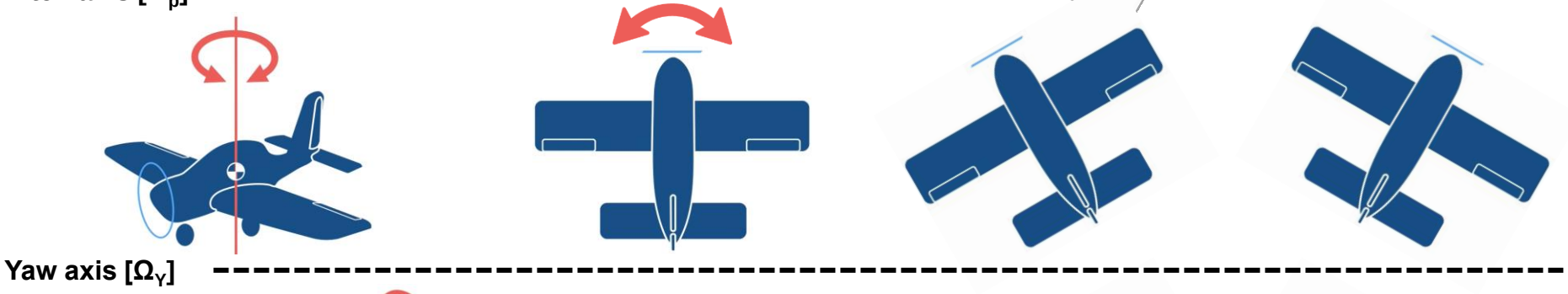
Sources:InvenSense

Three Axis for the Gyroscope

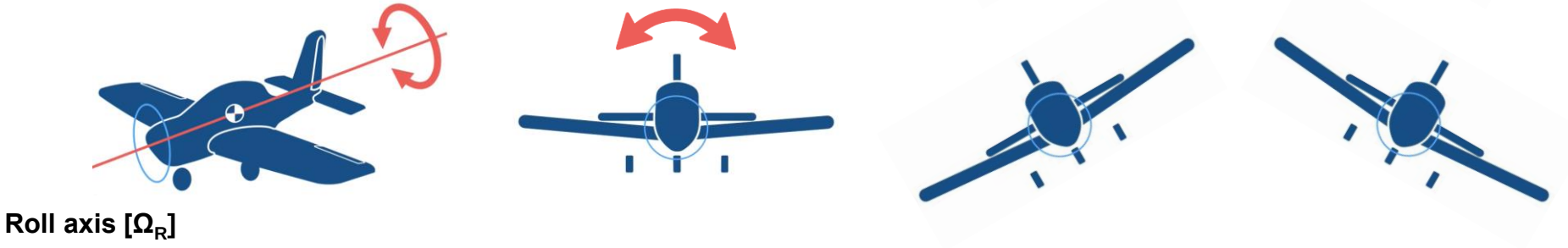




Pitch axis [Ω_p]



Yaw axis [Ω_Y]

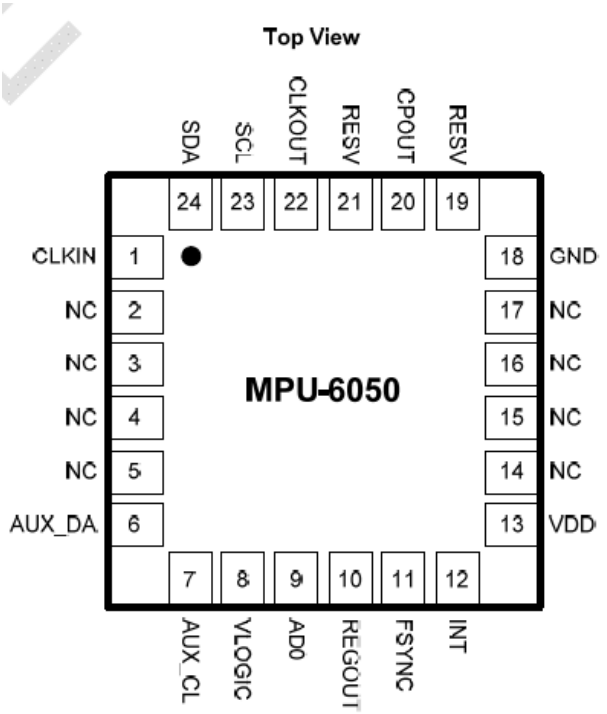


Roll axis [Ω_R]

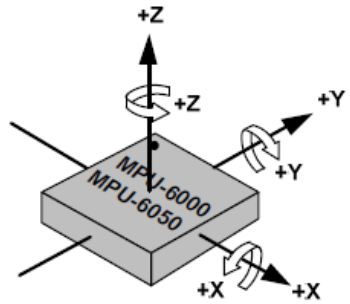
MPU-6050 Inertial Sensor

Pin Out and Signal Description

Pin Number	MPU-6000	MPU-6050	Pin Name	Pin Description
1	Y	Y	CLKIN	Optional external reference clock input. Connect to GND if unused.
6	Y	Y	AUX_DA	I^2C master serial data, for connecting to external sensors
7	Y	Y	AUX_CL	I^2C Master serial clock, for connecting to external sensors
8	Y		/CS	SPI chip select (0=SPI mode)
8		Y	VLOGIC	Digital I/O supply voltage
9	Y		AD0 / SDO	I^2C Slave Address LSB (AD0); SPI serial data output (SDO)
9		Y	AD0	I^2C Slave Address LSB (AD0)
10	Y	Y	REGOUT	Regulator filter capacitor connection
11	Y	Y	FSYNC	Frame synchronization digital input. Connect to GND if unused.
12	Y	Y	INT	Interrupt digital output (totem pole or open-drain)
13	Y	Y	VDD	Power supply voltage and Digital I/O supply voltage
18	Y	Y	GND	Power supply ground
19, 21	Y	Y	RESV	Reserved. Do not connect.
20	Y	Y	CPOUT	Charge pump capacitor connection
22	Y	Y	CLKOUT	System clock output
23	Y		SCL / SCLK	I^2C serial clock (SCL); SPI serial clock (SCLK)
23		Y	SCL	I^2C serial clock (SCL)
24	Y		SDA / SDI	I^2C serial data (SDA); SPI serial data input (SDI)
24		Y	SDA	I^2C serial data (SDA)
2, 3, 4, 5, 14, 15, 16, 17	Y	Y	NC	Not internally connected. May be used for PCB trace routing.



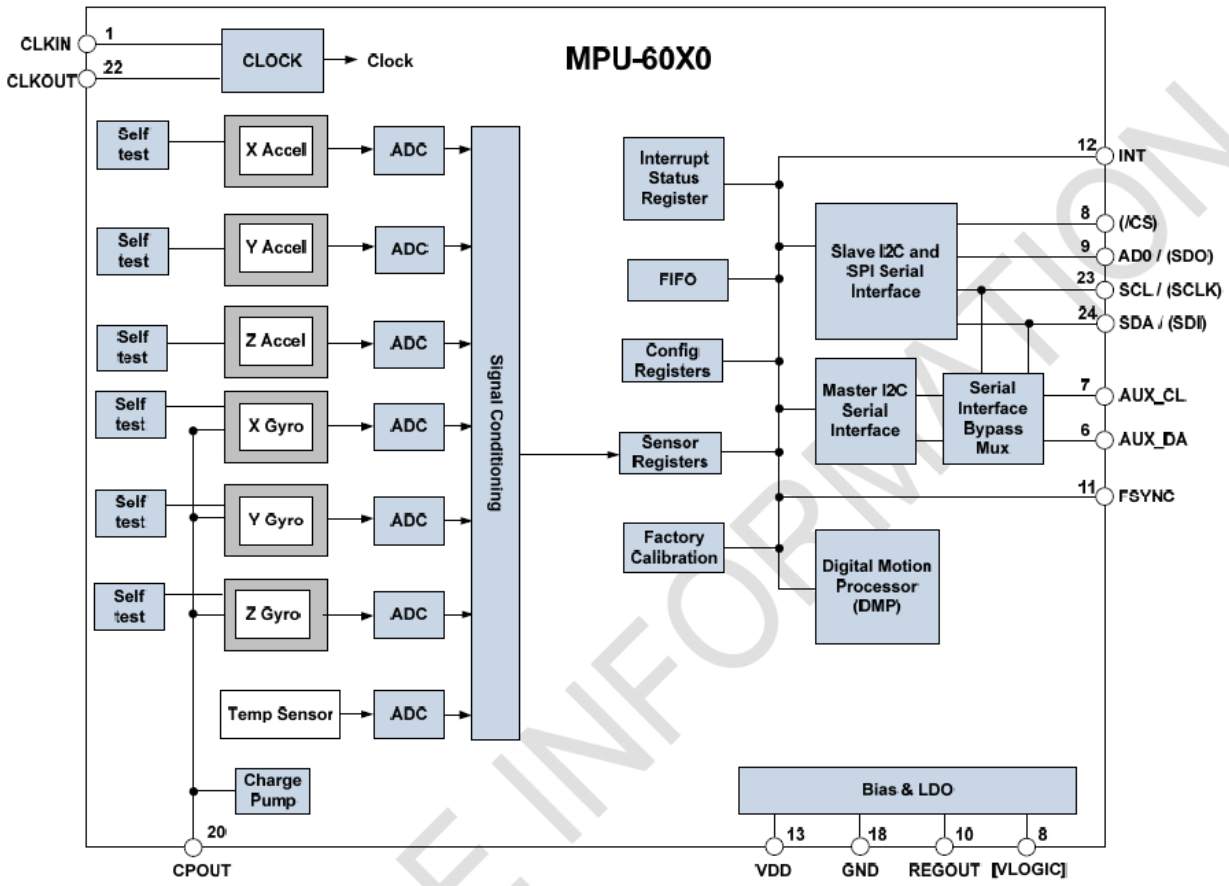
QFN Package
24-pin, 4mm x 4mm x 0.9mm



Orientation of Axes of Sensitivity and Polarity of Rotation

Sources:InvenSense

MPU-6050 Inertial Sensor

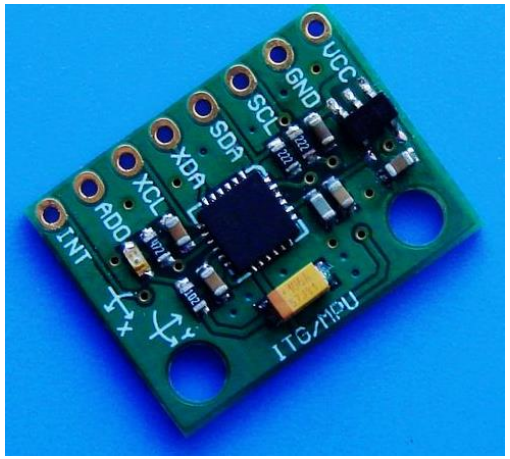


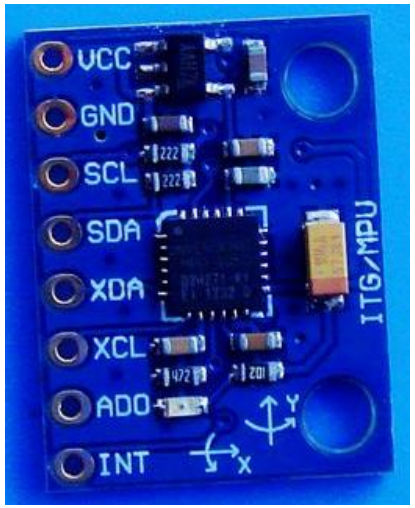
The MPU-60X0 is comprised of the following key blocks and functions:

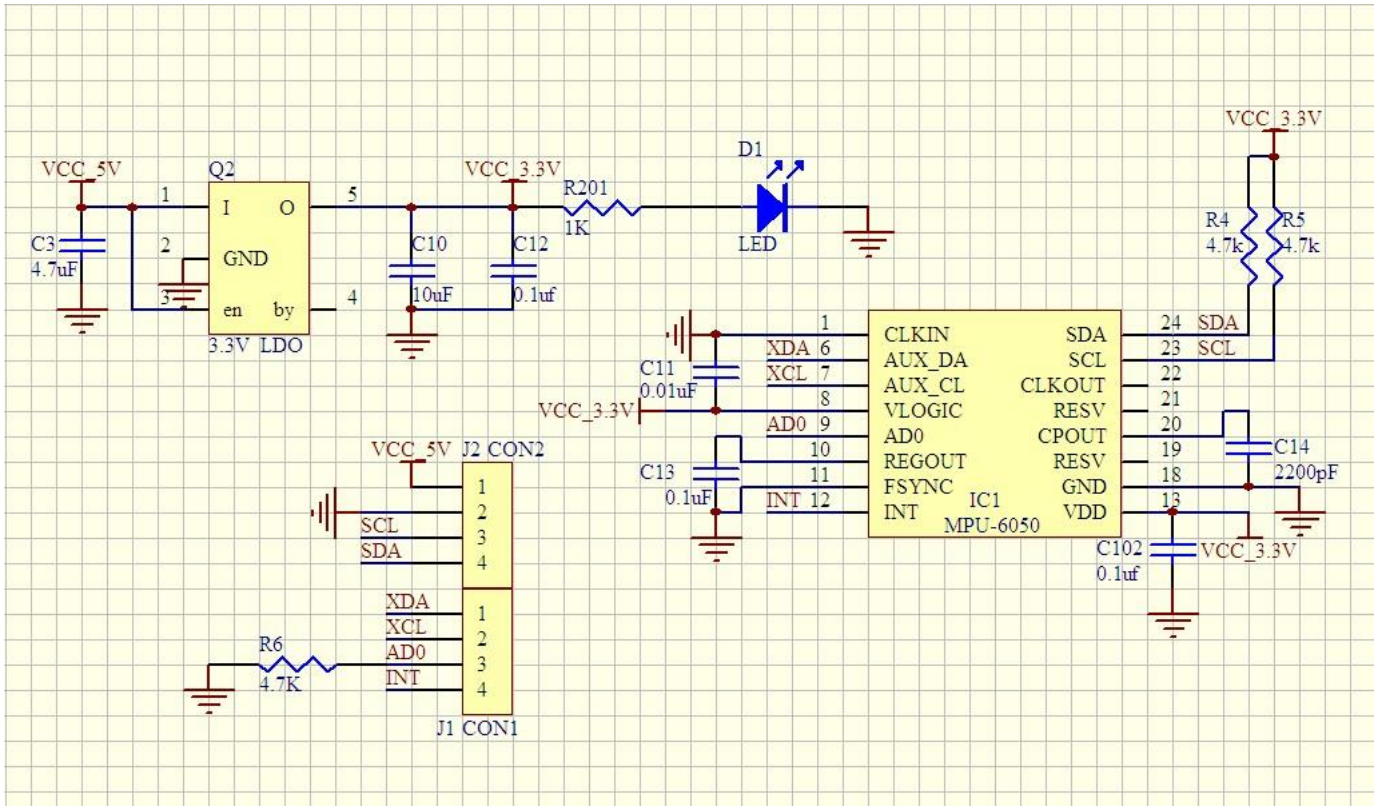
- Three-axis MEMS rate gyroscope sensor with 16-bit ADCs and signal conditioning
- Three-axis MEMS accelerometer sensor with 16-bit ADCs and signal conditioning
- Digital Motion Processor (DMP) engine
- Primary I2C and SPI (MPU-6000 only) serial communications interfaces
- Auxiliary I2C serial interface for 3rd party magnetometer & other sensors
- Clocking
- Sensor Data Registers
- FIFO
- Interrupts
- Digital-Output Temperature Sensor
- Gyroscope & Accelerometer Self-test
- Bias and LDO
- Charge Pump

Sources:InvenSense

Link the Inertial Sensor with Arduino

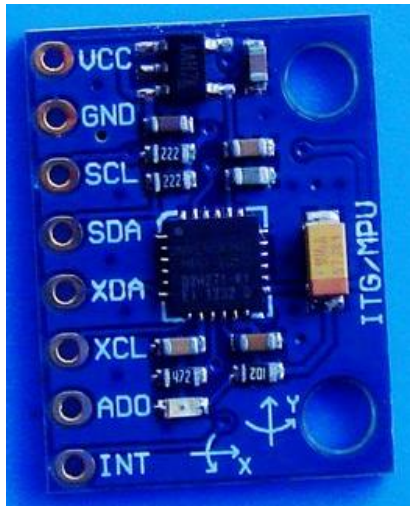


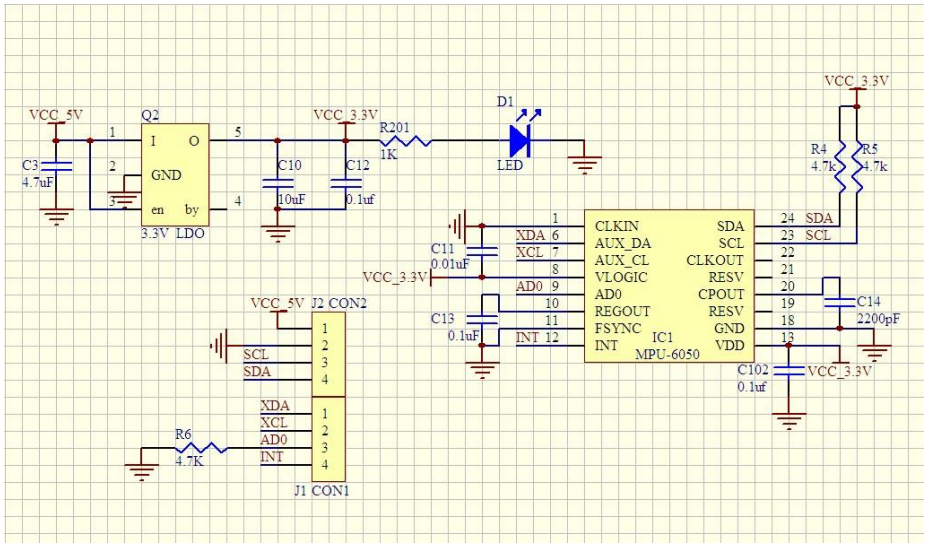




Commercial Module for MPU6050 based on PCB

Link the Inertial Sensor with Arduino





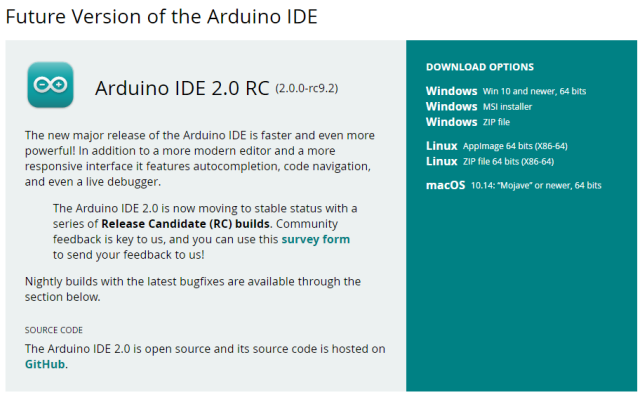
Pin Number	MPU-6000	MPU-6050	Pin Name	Pin Description
1	Y	Y	CLKIN	Optional external reference clock input. Connect to GND if unused.
6	Y	Y	AUX_DA	I^2C master serial data, for connecting to external sensors
7	Y	Y	AUX_CL	I^2C Master serial clock, for connecting to external sensors
8	Y		/CS	SPI chip select (0=SPI mode)
8		Y	VLOGIC	Digital I/O supply voltage
9	Y		AD0 / SDO	I^2C Slave Address LSB (AD0); SPI serial data output (SDO)
9		Y	AD0	I^2C Slave Address LSB (AD0)
10	Y	Y	REGOUT	Regulator filter capacitor connection
11	Y	Y	FSYNC	Frame synchronization digital input. Connect to GND if unused.
12	Y	Y	INT	Interrupt digital output (totem pole or open-drain)
13	Y	Y	VDD	Power supply voltage and Digital I/O supply voltage
18	Y	Y	GND	Power supply ground
19, 21	Y	Y	RESV	Reserved. Do not connect.
20	Y	Y	CPOUT	Charge pump capacitor connection
22	Y	Y	CLKOUT	System clock output
23	Y		SCL / SCLK	I^2C serial clock (SCL); SPI serial clock (SCLK)
23		Y	SCL	I^2C serial clock (SCL)
24	Y		SDA / SDI	I^2C serial data (SDA); SPI serial data input (SDI)
24		Y	SDA	I^2C serial data (SDA)
2, 3, 4, 5, 14, 15, 16, 17	Y	Y	NC	Not internally connected. May be used for PCB trace routing.

Based on the datasheet and circuit, we can understand the functions for each pin

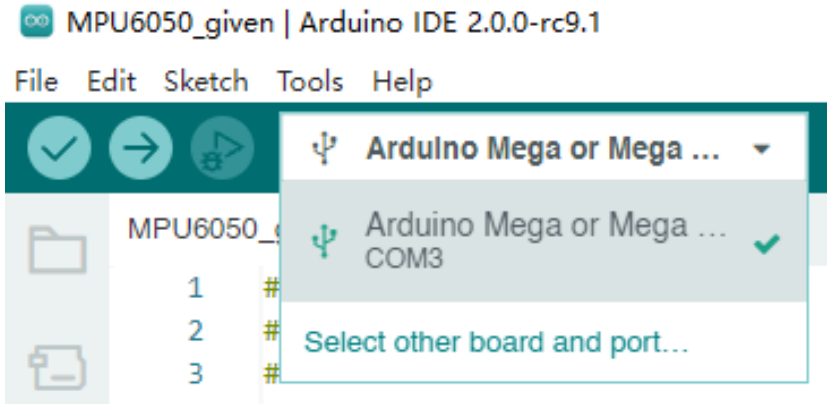
Link the Inertial Sensor with Arduino

Preparation for the Arduino board

Download the newest Arduino IDE through:
https://www.arduino.cc/en/software

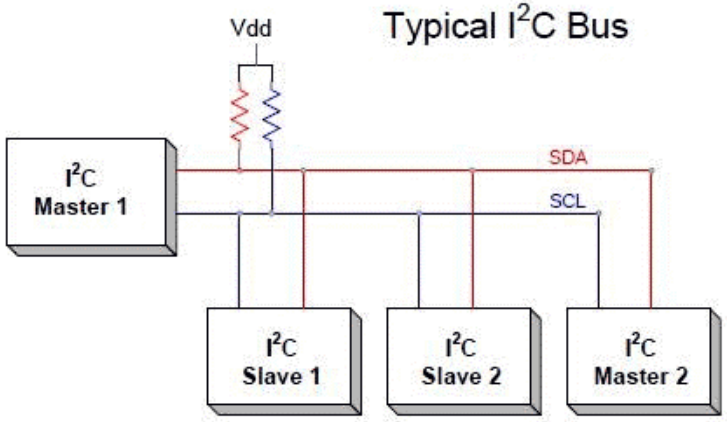


Link the Arduino Mega with computer, and choose the correct port:



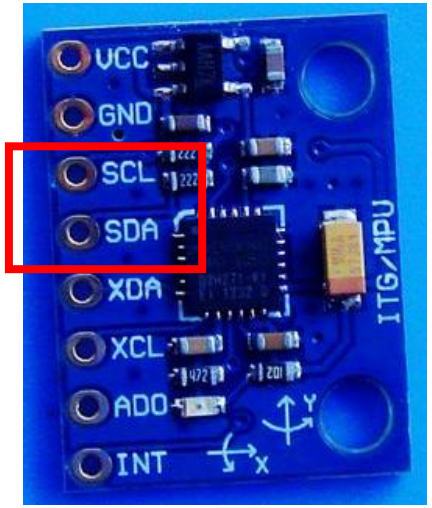
Use Arduino to Read the Output

1. Obtained from data sheet – MPU6050 supports I²C interface



This is the reason why the module can have six different signal outputs (3-axis acceleration and 3-axis angular acceleration) with only one output port.

2. Find the output pins on the PCB to read the I²C sensing signal

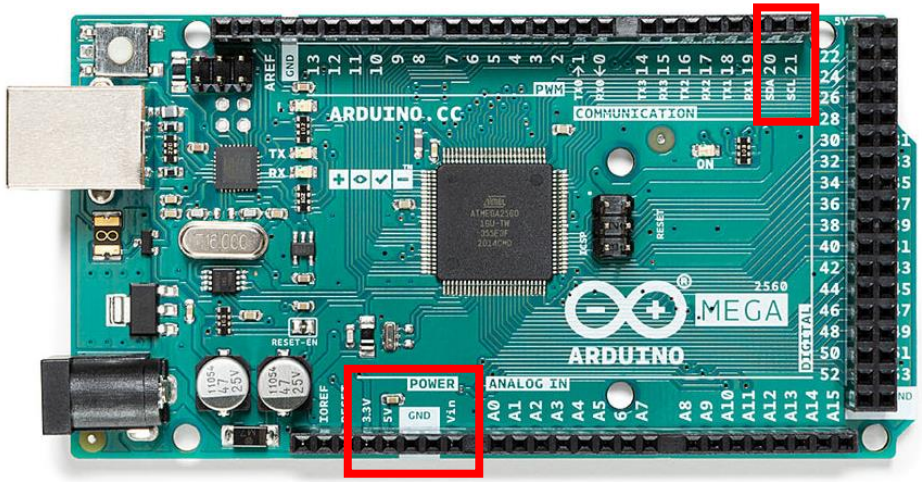


SDA: I²C serial data (sensing data)
SCL: I²C serial clock

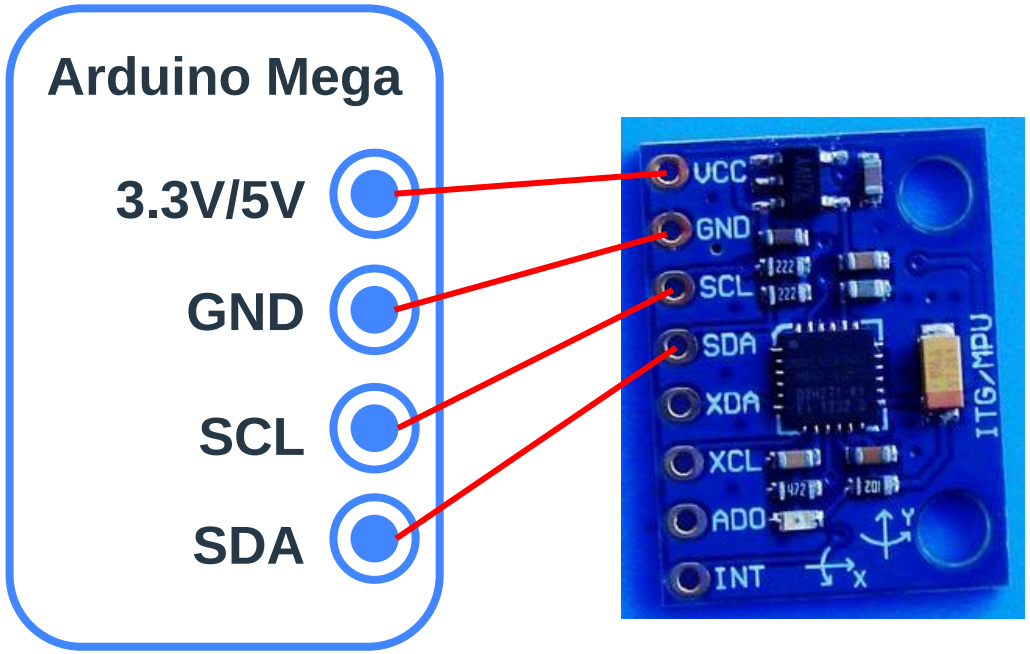
Use Arduino to Read the Output

3. Link the device with analog input on the Arduino board

Sensing data



Power supply

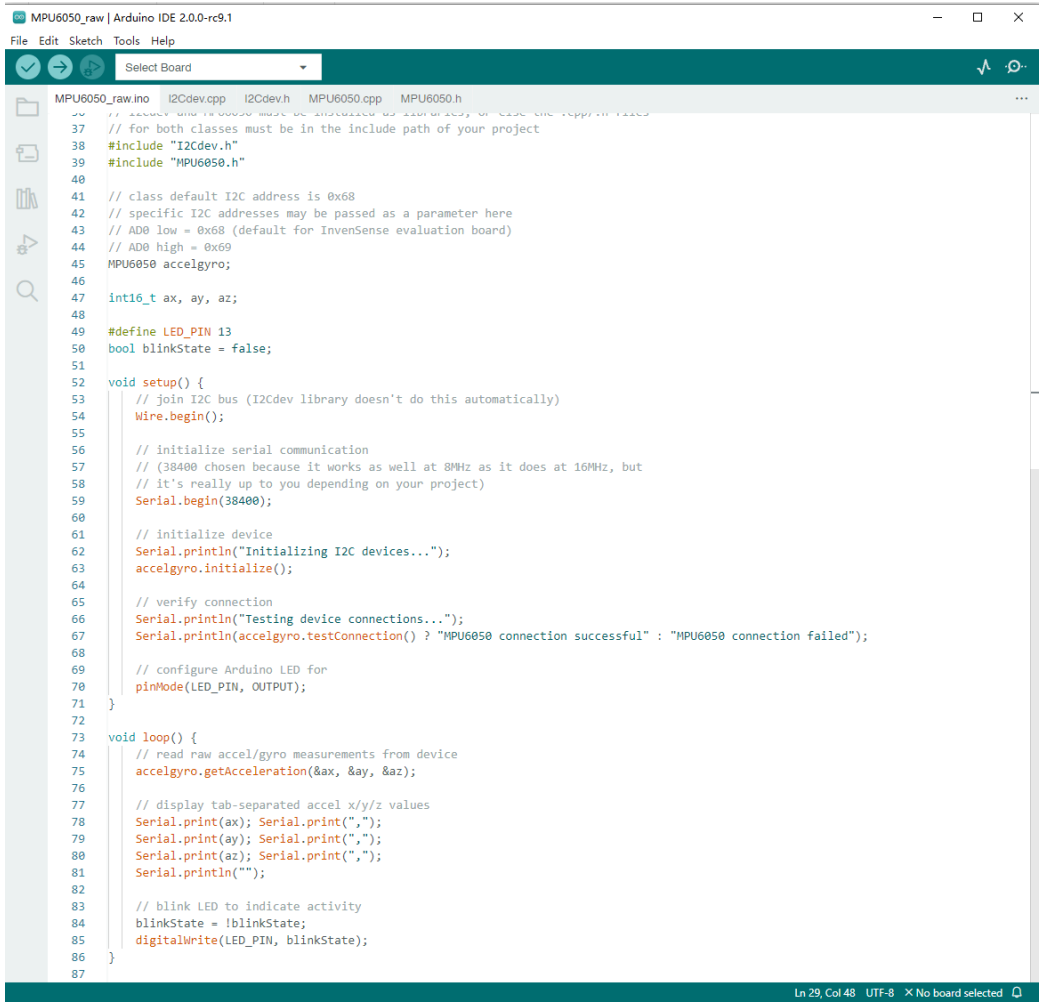


Note: The VCC should not be linked with the inertial sensor when uploading the code to the Ardunio

Sources: Arduino, Github

Use Arduino to Read the Output

4. Upload the code to the Arduino mega and try to see the output through the “Serial Plotter” (the code is provided and can be downloaded through Canvas, in a zip file named “MPU6050_raw”)



Sources: Arduino, Github

Use Arduino to Read the Output

The accelerations on three axes can be obtained

Use Arduino to Read the Output

```
// Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE implementation
// is used in I2Cdev.h
#include "Wire.h"

// 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.h"

// class default I2C address is 0x68
// specific I2C addresses may be passed as a parameter here
// AD0 low = 0x68 (default for InvenSense evaluation board)
// AD0 high = 0x69
MPU6050 accelgyro;

int16_t ax, ay, az;
```

- The *wire.h* and *I2Cdev.h* contain the basic information like the address for reading the MPU6050 IMU
- The *MPU6050.h* contain all the functions can be used for fully programming the MPU6050 IMU.

The MPU-60X0 is comprised of the following key blocks and functions:

- Three-axis MEMS rate gyroscope sensor with 16-bit ADCs and signal conditioning
- Three-axis MEMS accelerometer sensor with 16-bit ADCs and signal conditioning

Use Arduino to Read the Output

Reference > Language > Functions > Communication > Wire

Wire

[Communication]

Description

This library allows you to communicate with I2C/TWI devices. On the Arduino boards with the R3 layout (1.0 pinout), the SDA (data line) and SCL (clock line) are on the pin headers close to the AREF pin. The Arduino Due has two I2C/TWI interfaces SDA1 and SCL1 are near to the AREF pin and the additional one is on pins 20 and 21.

As a reference the table below shows where TWI pins are located on various Arduino boards.

Board	I2C/TWI pins
UNO, Ethernet	A4 (SDA), A5 (SCL)
Mega2560	20 (SDA), 21 (SCL)
Leonardo	20 (SDA), 21 (SCL), SDA1, SCL1

As of Arduino 1.0, the library inherits from the Stream functions, making it consistent with other read/write libraries. Because of this, `send()` and `receive()` have been replaced with `read()` and `write()`.

Recent versions of the Wire library can use timeouts to prevent a lockup in the face of certain problems on the bus, but this is not enabled by default (yet) in current versions. It is recommended to always enable these timeouts when using the Wire library. See the Wire.setWireTimeout function for more details.

Note: There are both 7 and 8-bit versions of I2C addresses. 7 bits identify the device, and the eighth bit determines if it's being written to or read from. The Wire library uses 7 bit addresses throughout. If you have a datasheet or sample code that uses 8-bit address, you'll want to drop the low bit (i.e. shift the value one bit to the right), yielding an address between 0 and 127. However the addresses from 0 to 7 are not used because are reserved so the first address that can be used is 8. Please note that a pull-up resistor is needed when connecting SDA/SCL pins. Please refer to the examples for more information. MEGA 2560 board has pull-up resistors on pins 20 and 21 onboard.

The Wire library implementation uses a 32 byte buffer, therefore any communication should be within this limit. Exceeding bytes in a single transmission will just be dropped.

To use this library:

```
#include <Wire.h>
```

https://www.arduino.cc/reference/en/language/functions/communication/wire/

Functions

begin()
end()
requestFrom()
beginTransmission()
endTransmission()
write()
available()
read()
setClock()
onReceive()
onRequest()
setWireTimeout()
clearWireTimeoutFlag()
getWireTimeoutFlag()

Reference > Language > Functions > Communication > Wire > Begin

begin()

Description

This function initializes the Wire library and join the I2C bus as a controller or a peripheral. This function should normally be called only once.

Syntax

```
Wire.begin()
```

```
Wire.begin(address)
```

Parameters

- *address*: the 7-bit slave address (optional); if not specified, join the bus as a controller device.

Returns

None.

Only the "begin" function will be used in this lab, you are encouraged to explore other functions

Use Arduino to Read the Output

```
73  void loop() {
74      // read raw accel/gyro measurements from device
75      accelgyro.getAcceleration(&ax, &ay, &az);
76
77      // display tab-separated accel x/y/z values
78      Serial.print(ax); Serial.print(",");
79      Serial.print(ay); Serial.print(",");
80      Serial.print(az); Serial.print(",");
81      Serial.println("");
82
83      // blink LED to indicate activity
84      blinkState = !blinkState;
85      digitalWrite(LED_PIN, blinkState);
86  }
```

```
        // ACCEL_*OUT_* registers
        void getMotion9(int16_t* ax, int16_t* ay, int16_t* az, int16_t* gx,
int16_t* gy, int16_t* gz, int16_t* mx, int16_t* my, int16_t* mz);
        void getMotion6(int16_t* ax, int16_t* ay, int16_t* az, int16_t* gx,
int16_t* gy, int16_t* gz);
        void getAcceleration(int16_t* x, int16_t* y, int16_t* z);
        int16_t getAccelerationX();
        int16_t getAccelerationY();
        int16_t getAccelerationZ();
```

More functions can be found in the given "MPU6050.h" file

Sources: Github

Build wireless communication

Paring two CC2530 module

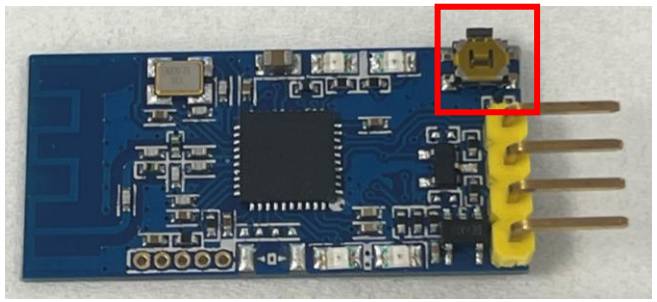
a. Link the CC2530 to the Arduino Board and USB-TTL module

GND – GND
RX – TX
TX – RX
VCC – VCC

Link the RX on the CC2530 to the TX on the Arduino board/ USB-TTL module

b. Hold down the button and, while keeping it pressed, plug in the USB-TTL interface to power the CC2530.

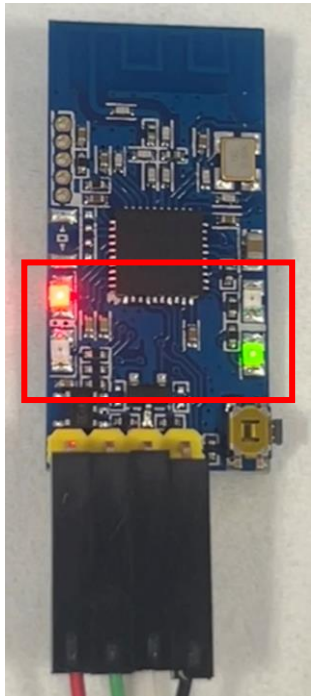
Note: need to hold the button before power it



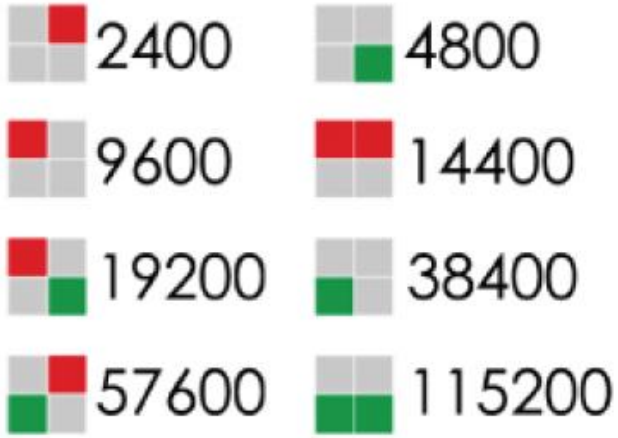
Build wireless communication

Paring two CC2530 module

c. Continue to hold the button until the device is powered, and watch for the 4 LED lights on it to flash in sequence. Once this happens, release the button, and you will enter the CC2530's setup mode.



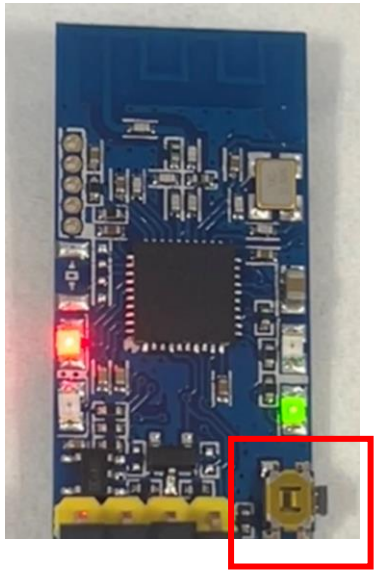
d. Press the button to set the CC2530's baud rate. Different LED patterns represent different baud rates:



Build wireless communication

Paring two CC2530 module

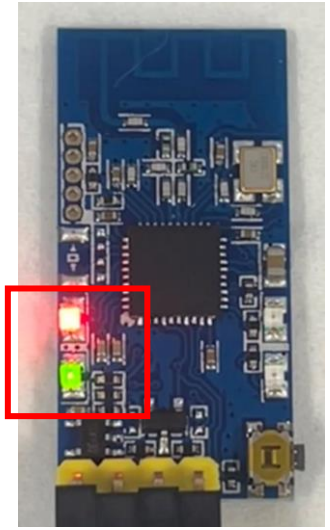
e. After setting the baud rate, press and hold the button again until the LED lights flash in sequence, to move to the next step of the setup.



f. Press the button to set the CC2530's channel.

Different LED patterns represent different channels.

Make sure both CC2530 devices have the same LED pattern, so they can communicate on the same channel.



Build wireless communication

Paring two CC2530 module

g. After setting the channel, press and hold the button again until the LED lights flash in sequence to proceed to the next step of the setup.

h. Press the button to set the CC2530's transmission mode:

Peer-to-peer A

Peer-to-peer B

Broadcast

To ensure better communication quality, set one CC2530 as endpoint A and the other as endpoint B.

The cc2530 modules can have the highest accuracy when only paired with each other.

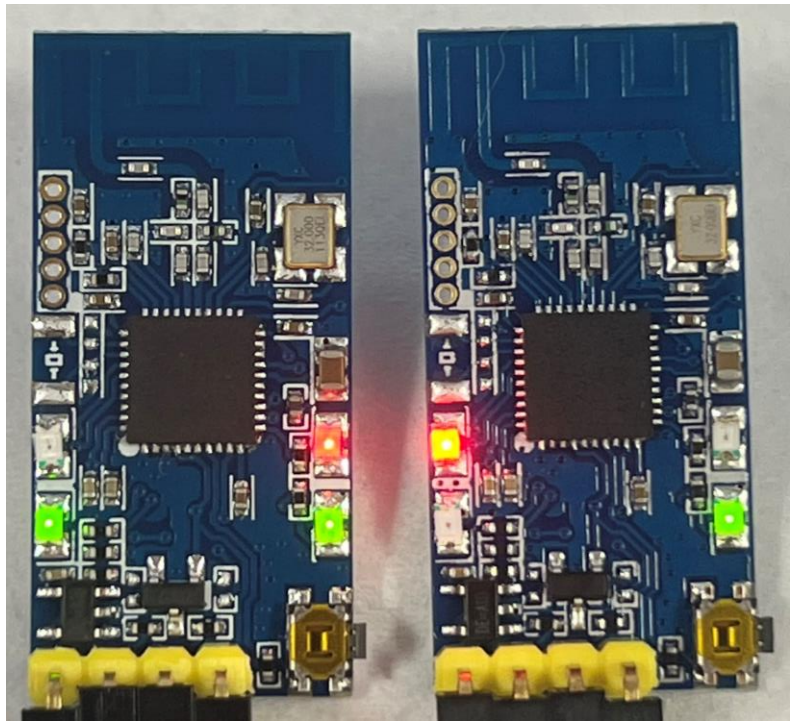
If it's set to the "Broadcast" mode, the signal can be received by every other cc2530 module in the same channel, and the bit error rate is around 1%.

Note that A and B do not designate which one is the receiver or transmitter; they simply indicate that they can communicate with each other. You can choose either one to act as the sender or receiver.

i. Press and hold the button until the LEDs blink and circulate to finish the setting, the LED will keep turning on for 2 seconds after releasing the button.

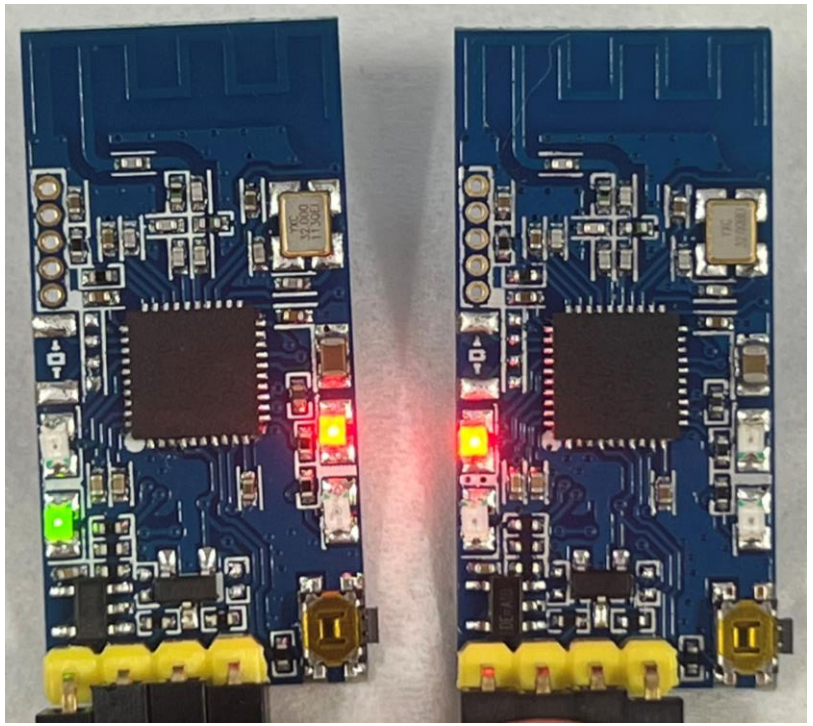
Build wireless communication

Paring two CC2530 module



Transmitter **Receiver**

The two CC2530 modules are paired successfully

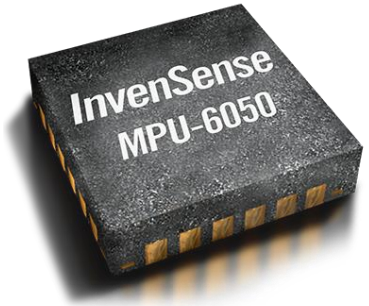


Transmitter **Receiver**

The two CC2530 modules are not paired

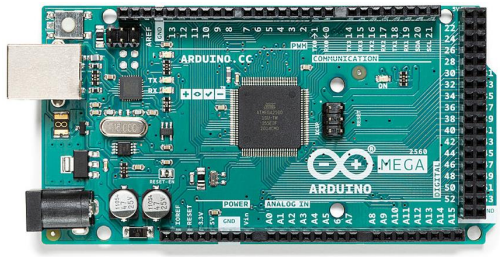
Note: the pattern might be different caused by you use A/B port as the transmitter/receiver, but there should at least be one green light for each module

Current IoT System Overview



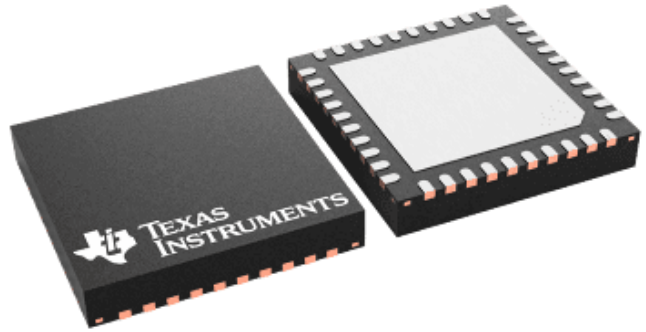
Inertial sensor

Any sensor placed in the environment, on human bodies, etc.



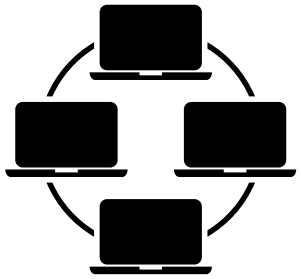
MCU

Data processing

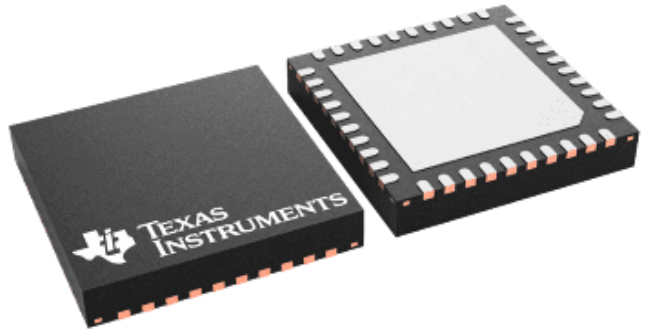


Wireless Module

Data communication



Data analysis & Computing



Wireless Module

Use Python to read the received data

Install “pyserial” to access the serial port through python:

```
C:\Users\admin>pip install pyserial
Looking in indexes: https://pypi.org/simple, https://pypi.ngc.nvidia.com
Collecting pyserial
  Downloading pyserial-3.5-py2.py3-none-any.whl (90 kB)
     ---------------------------------------- 90.6/90.6 kB ? eta 0:00:00
Installing collected packages: pyserial
Successfully installed pyserial-3.5

[notice] A new release of pip available: 22.2.1 -> 22.2.2
[notice] To update, run: python.exe -m pip install --upgrade pip
```

Use python to open the corresponding port:

```python
import serial

Data = serial.Serial()
Data.baudrate = 115200 # set baudrate
Data.port = 'COM3' # set port number
Data.timeout = 0.01
Data.open() # open the port
```

Find the port that linked with cc2530:

a. **Open Device Manager (Start → Control Panel → Hardware and Sound → Device Manager)**

b. **Look in the Device Manager list, open the category "Ports" and find the matching COM Port**

c. **Take the number in the bracket behind the port description**

You can explore more functions for pySerial through:
https://pythonhosted.org/pyserial/index.html

Use Python to read the received data

```python
import serial

Data = serial.Serial()
Data.baudrate = 115200 # set baudrate
Data.port = 'COM3' # set port number
Data.timeout = 0.01
Data.open() # open the port

while True:
    print(Data)
```

```
Serial<id=0x13ceee49780, open=True>(port='COM3', baudrate=115200, bytesize=8, parity='N', stopbits=1, timeout=0.01, xonxoff=False, rtscts=False, dsrdtr=False)
Serial<id=0x13ceee49780, open=True>(port='COM3', baudrate=115200, bytesize=8, parity='N', stopbits=1, timeout=0.01, xonxoff=False, rtscts=False, dsrdtr=False)
Serial<id=0x13ceee49780, open=True>(port='COM3', baudrate=115200, bytesize=8, parity='N', stopbits=1, timeout=0.01, xonxoff=False, rtscts=False, dsrdtr=False)
Serial<id=0x13ceee49780, open=True>(port='COM3', baudrate=115200, bytesize=8, parity='N', stopbits=1, timeout=0.01, xonxoff=False, rtscts=False, dsrdtr=False)
Serial<id=0x13ceee49780, open=True>(port='COM3', baudrate=115200, bytesize=8, parity='N', stopbits=1, timeout=0.01, xonxoff=False, rtscts=False, dsrdtr=False)
Serial<id=0x13ceee49780, open=True>(port='COM3', baudrate=115200, bytesize=8, parity='N', stopbits=1, timeout=0.01, xonxoff=False, rtscts=False, dsrdtr=False)
Serial<id=0x13ceee49780, open=True>(port='COM3', baudrate=115200, bytesize=8, parity='N', stopbits=1, timeout=0.01, xonxoff=False, rtscts=False, dsrdtr=False)
Serial<id=0x13ceee49780, open=True>(port='COM3', baudrate=115200, bytesize=8, parity='N', stopbits=1, timeout=0.01, xonxoff=False, rtscts=False, dsrdtr=False)
Serial<id=0x13ceee49780, open=True>(port='COM3', baudrate=115200, bytesize=8, parity='N', stopbits=1, timeout=0.01, xonxoff=False, rtscts=False, dsrdtr=False)
Serial<id=0x13ceee49780, open=True>(port='COM3', baudrate=115200, bytesize=8, parity='N', stopbits=1, timeout=0.01, xonxoff=False, rtscts=False, dsrdtr=False)
Serial<id=0x13ceee49780, open=True>(port='COM3', baudrate=115200, bytesize=8, parity='N', stopbits=1, timeout=0.01, xonxoff=False, rtscts=False, dsrdtr=False)
Serial<id=0x13ceee49780, open=True>(port='COM3', baudrate=115200, bytesize=8, parity='N', stopbits=1, timeout=0.01, xonxoff=False, rtscts=False, dsrdtr=False)
Serial<id=0x13ceee49780, open=True>(port='COM3', baudrate=115200, bytesize=8, parity='N', stopbits=1, timeout=0.01, xonxoff=False, rtscts=False, dsrdtr=False)
```

```python
import serial

Data = serial.Serial()
Data.baudrate = 115200 # set baudrate
Data.port = 'COM3' # set port number
Data.timeout = 0.01
Data.open() # open the port

while True:
    signal = Data.readline()
    print(signal)
```

Use "readline" to read the data from the port

```
b'-4100,-152,15780,\r\n'
b'-3880,-196,15800,\r\n'
b'-4096,-168,15896,\r\n'
b'-3984,-236,16088,\r\n'
b'-4056,-260,15852,\r\n'
b'-3928,-164,15860,\r\n'
b'-4220,-244,15984,\r\n'
```

Received three channel output data

Use Python to read the received data

```
b'-4100,-152,15780,\r\n'
b'-3880,-196,15800,\r\n'
b'-4096,-168,15896,\r\n'
b'-3984,-236,16088,\r\n'
b'-4056,-260,15852,\r\n'
b'-3928,-164,15860,\r\n'
b'-4220,-244,15984,\r\n'
```

We only need the input number for further data analysis

Method 1: Use the "I/O" function in python

```python
import serial
import io

Data = serial.Serial()
Data.baudrate = 115200 # set baudrate
Data.port = 'COM4' # set port number
Data.timeout = 0.01
Data.open() # open the port

sio = io.TextIOWrapper(io.BufferedRWPair(Data, Data, 1), encoding='ascii', newline='\r')

while True:
    signal = sio.readline()
    print(signal)
```

```
-3976,-124,15940,

-3932,-236,16056,

-3972,-92,16012,

-3932,-276,15932,

-4000,-180,15908,
```

https://docs.python.org/3/library/io.html#io.TextIOWrapper

Use Python to read the received data

Method 2

```
// display tab-separated accel x/y/z values
Serial.print(ax); Serial.print(",");
Serial.print(ay); Serial.print(",");
Serial.print(az); Serial.print(",");
Serial.println("");
```

We previously include “,” to help us to divide the data from each channel

```python
import serial

Data = serial.Serial()
Data.baudrate = 38400 # set baudrate
Data.port = 'COM3' # set port number
Data.timeout = 0.01
Data.open() # open the port

def dataProcess(data):
    data = str(data)
    dataSet = []
    datapoint = ''
    for i in data:
        if i.isdigit() or i == '-':
            datapoint += i
        elif i == ",":
            dataSet.append(datapoint)
            datapoint = ''
    return dataSet

while True:
    signal = Data.readline()
    signal = dataProcess(signal)
    print(signal)
```

```
['-3048', '1912', '-15576']
['-3204', '1972', '-15676']
['-3200', '2036', '-15580']
['-3096', '1936', '-15436']
['-3100', '1876', '-15656']
['-3112', '1980', '-15392']
['-3144', '1900', '-15616']
['-3104', '1964', '-15628']
['-3268', '2088', '-15656']
['-3172', '2016', '-15660']
['-3212', '2012', '-15620']
['-3216', '1984', '-15592']
['-3200', '2024', '-15624']
['-3192', '1984', '-15516']
['-3132', '2000', '-15512']
['-3184', '1996', '-15480']
```

Use Python to read the received data

Based on the previous method, we further need to convert the data to the “float” type for processing

```python
def canFloat(data):
    try:
        float(data)
        return True
    except:
        return False
```

```python
import serial

Data = serial.Serial()
Data.baudrate = 38400 # set baudrate
Data.port = 'COM3' # set port number
Data.timeout = 0.01
Data.open() # open the port

def canFloat(data):
    try:
        float(data)
        return True
    except:
        return False

def dataProcess(data):
    data = str(data)
    dataSet = []
    datapoint = ''
    for i in data:
        if i.isdigit() or i == '-':
            datapoint += i
        elif i == "," and canFloat(datapoint):
            dataSet.append(float(datapoint))
            datapoint = ''
    return dataSet

while True:
    signal = Data.readline()
    signal = dataProcess(signal)
    print(signal)
```

```
[-3644.0, 632.0, -15512.0]
[-3588.0, 556.0, -15632.0]
[-3652.0, 676.0, -15544.0]
[-3672.0, 636.0, -15512.0]
[-3648.0, 716.0, -15532.0]
[-3624.0, 692.0, -15648.0]
[-3520.0, 656.0, -15624.0]
[-3604.0, 736.0, -15484.0]
[-3560.0, 704.0, -15632.0]
[-3548.0, 632.0, -15608.0]
[-3668.0, 592.0, -15528.0]
[-3492.0, 628.0, -15588.0]
[-3572.0, 652.0, -15568.0]
[-3632.0, 716.0, -15564.0]
[-3560.0, 712.0, -15552.0]
[-3600.0, 708.0, -15404.0]
[-3608.0, 560.0, -15564.0]
```

Use Python to read the received data

However, the data directly printed in the terminal is hard for us to observe its changes and trends, so we can try to visualize the sensed data:

PyQtGraph

Scientific Graphics and GUI Library for Python

Documentation - GitHub Repository - Mailing list

Install from PyPI:

```
pip install pyqtgraph
```

or via conda:

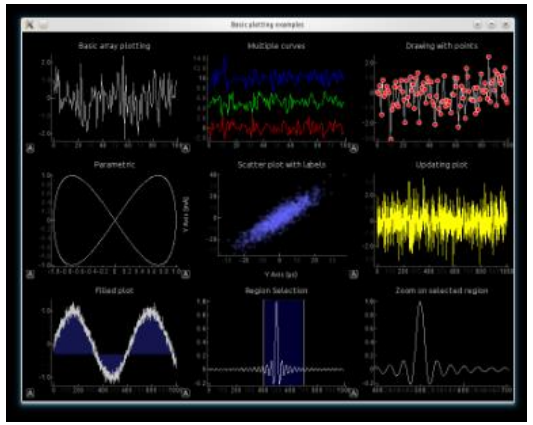
```
conda install -c conda-forge pyqtgraph
```

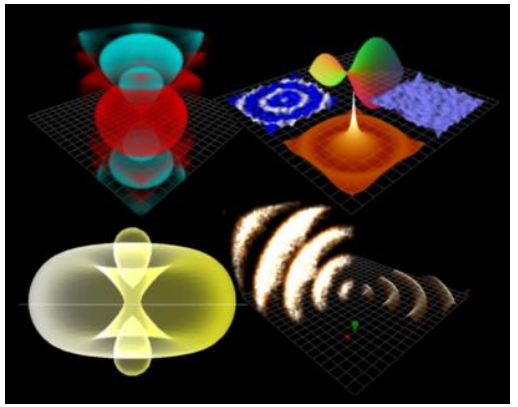
or from source on GitHub:

```
git clone https://github.com/pyqtgraph/pyqtgraph
cd pyqtgraph
pip install .
```

recent changes - older releases

PyQtGraph is a pure-python graphics and GUI library built on PyQt / PySide and numpy. It is intended for use in mathematics / scientific / engineering applications. Despite being written entirely in python, the library is very fast due to its heavy leverage of NumPy for number crunching and Qt's GraphicsView framework for fast display. PyQtGraph is distributed under the MIT open-source license.





Here, we use "pyqtgraph" function as an example: https://www.pyqtgraph.org/

You can also try to draw the figure through other functions

Use Python to read the received data

Use “pyqtgraph” to timely draw the sensed data:

```
import serial
import pyqtgraph as pg
import array
import numpy as np
import math

Data = serial.Serial()
Data.baudrate = 38400 # set baudrate
Data.port = 'COM3' # set port number
Data.timeout = 0.01
Data.open() # open the port
app = pg.mkQApp()
win = pg.GraphicsLayoutWidget()
win.setWindowTitle('demo')
win.resize(1600, 900)
xLength = 300
```

More functions are required to be imported

“array” and “numpy” are other two commonly used functions for data analysis, you can explore by yourself

Open the port same as before

Initialize the window for the drawing

Window title: demo

Window size: 1600 x 900 pixels

Data points for x-axis: 300

Use Python to read the received data

Use “pyqtgraph” to timely draw the sensed data:

```
17
18      fig1 = win.addPlot()
19      fig1.showGrid(x=True, y=True)
20      fig1.setRange(xRange=[0, xLength], padding=0)
21      fig1.setLabel(axis='left', text='g')
22      fig1.setLabel(axis='bottom', text='x / point')
23      fig1.setTitle('acceleration')
24
25      curve1 = fig1.plot()
26      curve2 = fig1.plot()
27      curve3 = fig1.plot()
28
29      data = [np.zeros(xLength).__array__('d'),
30              np.zeros(xLength).__array__('d'),
31              np.zeros(xLength).__array__('d')]
32
```

Add one figure to the Window we set before

Open the x-axis and the y-axis

Set the label for the x-axis and the y-axis

Set the title for the figure

More parameters can be set, which can be found at: https://www.pyqtgraph.org/

Initialize the curves to be drown

Here we include three curves for the x-, y-, and z- axis accelerations

Create a dynamic array to store the data we want to draw

Use Python to read the received data

Use “pyqtgraph” to timely draw the sensed data:

```
32
33  def plotData():
34
35      global signal
36      signal = Data.readline()
37      signal = dataProcess(signal)
38
39      if (len(signal) == 3):
40          for i in range(len(data)):
41
42              if len(data[i]) < xLength:
43                  data[i].append(signal[i])
44              else:
45                  data[i][:-1] = data[i][1:]
46                  data[i][-1] = signal[i]
47
48          curve1.setData(data[0], pen=pg.mkPen('g', width=3))
49          curve2.setData(data[1], pen=pg.mkPen('r', width=3))
50          curve3.setData(data[2], pen=pg.mkPen('b', width=3))
51
```

Use the function we defined before to obtain the float-type data received from the CC2530

What will happen if we didn’t include the condition: “if (len(signal) == 3):” ?

Add the new sensed data to the dynamic array we created, and if the data points stored in the dynamic array exceed the length we set, remove the previous data

Put the data into corresponding curves

Set the color and line width for the curves

Use Python to read the received data

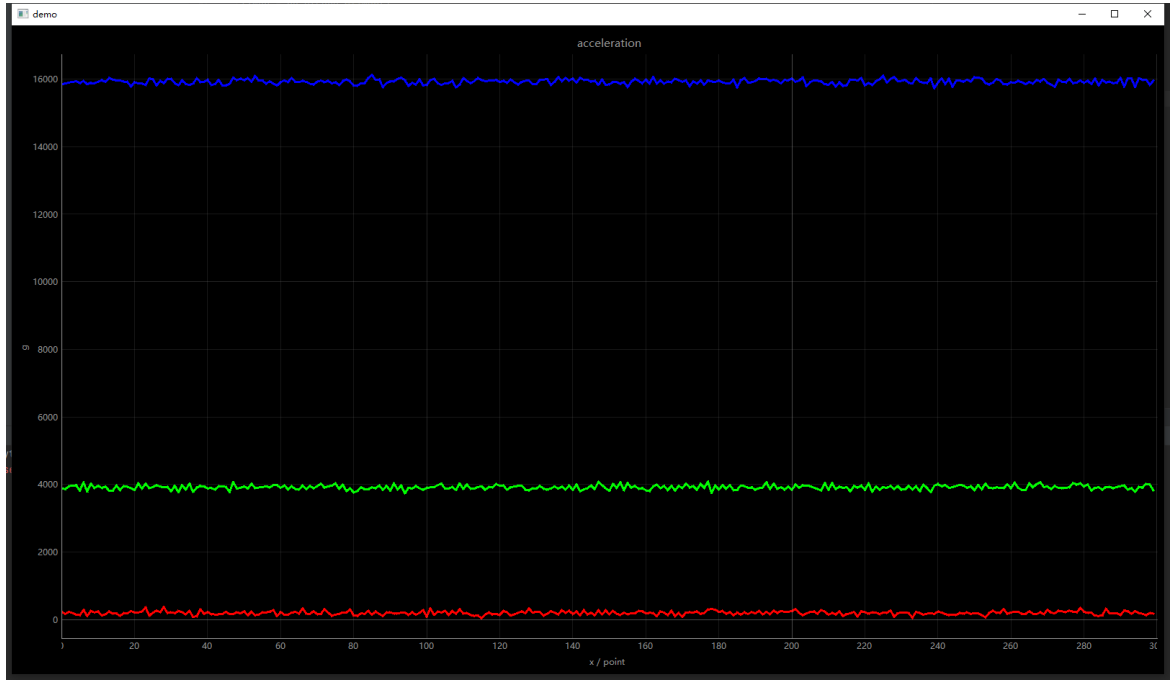
Use “pyqtgraph” to timely draw the sensed data:

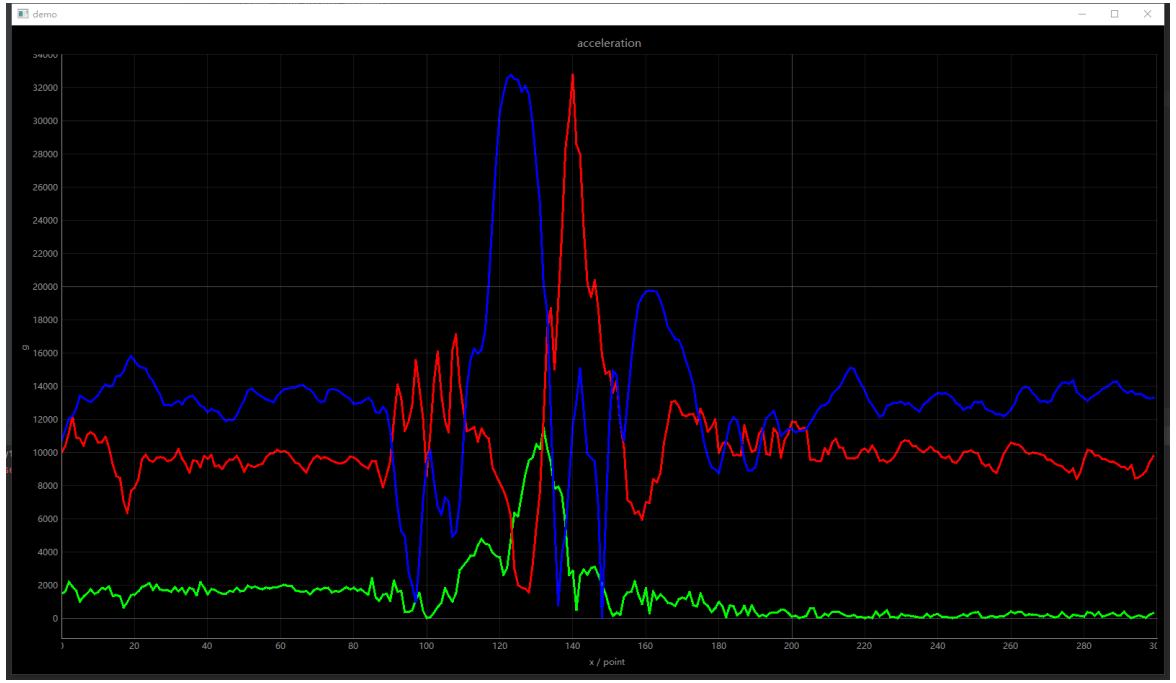
```
104
105 timer = pg.QtCore.QTimer()
106 timer.timeout.connect(plotData)
107 win.show()
108 timer.start(1)
109
110 app.exec()
111
```

Continuously call the plotData functions

Set the time interval as 1ms (can be changed)

Only stop the progress when we close the window





Use Python to read the received data

Besides, we can save all the sensing data for further analysis like machine learning:

```
33
34      ax = array.array('d')
35      ay = array.array('d')
36      az = array.array('d')
37
38      def plotData():
39
40          global signal
41          signal = Data.readline()
42          signal = dataProcess(signal)
43
44          if (len(signal) == 3):
45              for i in range(len(data)):
46
47                  if len(data[i]) < xLength:
48                      data[i].append(signal[i])
49                  else:
50                      data[i][:-1] = data[i][1:]
51                      data[i][-1] = signal[i]
52
53              ax.append(signal[0])
54              ay.append(signal[1])
55              az.append(signal[2])
56
57              curve1.setData(data[0], pen=pg.mkPen('g', width=3))
58              curve2.setData(data[1], pen=pg.mkPen('r', width=3))
59              curve3.setData(data[2], pen=pg.mkPen('b', width=3))
60
```

Create arrays to store the data obtained

Store the read data in the “signal” to each array

Use Python to read the received data

Save the data into files:

```
104
105 timer = pg.QtCore.QTimer()
106 timer.timeout.connect(plotData)
107 win.show()
108 timer.start(1)
109
110 app.exec()
111
112 try:
113     input("Press Enter to Stop\n")
114 except (Exception, KeyboardInterrupt):
115     pass
116
117 np.savetxt("x_acc.txt", ax)
118 np.savetxt("y_acc.txt", ay)
119 np.savetxt("z_acc.txt", az)
120
121 app.close()
```

In this lab, we save the data directly in the “txt” file, you can also try to save it into “csv” file or others.

The saving speed will vary according to the file types, normally, saving in “txt” is faster than “csv”

Stop the whole process manually after closing the window