Robótica Aerea
Módulo IV, Tema 1: AHRS with Arduino

Xin Chen

Ingeniería de Robótica Software
Escuela de Ingeniería de Fuenlabrada
Área de Ingeniería Aeroespacial

Universidad Rey Juan Carlos
GISAT

November 21, 2023

Contenidos

What is Arduino?

Arduino is an **open-source prototyping** (creación de prototipos de código abierto) platform based on **easy-to-use** hardware and software.

- The Arduino Board is able to **read inputs**, which are then turned into **outputs** to **actuate** correspondingly.
- The Sketch contains the **instructions** that will be sent to the **microcontroller** on the board.
- An Arduino sketch is written in the Arduino programming language that is derived from **C and C++**. The Arduino programming language is also called Arduino API (application programming interface, interfaz de programación de aplicaciones).
- The Arduino Software is an **integrated development environment** (IDE, entorno de desarrollo integrado) to **compile** (compilar) and **upload** (cargar) the sketch to the microcontroller.
- **Advantages**: Inexpensive, Cross-platform (compatible con múltiples plataformas), Simple and Clear Programming Environment, Open Source (código abierto).

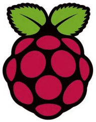
Figure: Arduino Logo

Figure: Arduino IDE

Arduino vs Raspberry Pi

Raspberry Pi

- **General Computer** with **operating system**.
- Capable of running **more complex tasks**.
- Can run **multiple programs**.
- Relatively **more difficult** to use.



Arduino

- **Microcontroller motherboard** (placa base), a **simple computer** that establishes communication between CPU and memory and provides some peripheral connections.
- Capable of doing **simple tasks**, like reading sensor measurement, control LED brightness, control simple actuators such as motors and servos, etc.
- Can run only **one program at a time**, in a **repetitive** manner.
- **Easy** to use.

Preparation

- Download the "sketchbook" folder from Aula Virtual.
- Launch Arduino IDE
  - Press the Windows logo key, type "Arduino", and press Enter.
  - **Select Board**: select Arduino Mega2560 board and the correspondent serial Port.
  - Change Arduino Sketchbook location: **File** → **Preference** → **Sketchbook location**: "sketchbook/arduino".
  - You can only modify or add Arduino sketches in the "/sketchbook/arduino" folder.
- Launch Processing Software
  - Download the **Processing Software** from here.
  - Extract the folder that contains the Processing Software to "Escritorio".
  - Open Terminal Window:
    - Press the Windows logo key or click the "Show Applications" button at the bottom-left corner of your screen (at the bottom of the dash bar).
    - Type "Terminal" and press Enter.
  - Drag the file "/Escritorio/processing-4.3/processing" to the Terminal window.
  - Click on anywhere in the Terminal window.
  - Press Enter.
  - Change Processing Sketchbook location: **File** → **Preference** → **Sketchbook folder**: "sketchbook/processing".
  - You can only modify or add Processing sketches in the "/sketchbook/processing" folder.

Universidad Rey Juan Carlos GISAT

Arduino Sketch basics

- In the Arduino project, a program is referred to as a SKETCH. It has the .ino extension, and is always stored in a folder of the same name.
- The **absolute minimum requirement** of an Arduino sketch is the use of two functions: void setup() and void loop().
  - void setup() - this function executes **only once**, when the Arduino is powered on. Here we define things such as the mode of a pin (input or output), the baud rate of serial communication or the initialization of a library.
  - void loop() - the code here is executed **over and over again**.
  - The "void" indicates that **nothing is returned** on execution.
  - The above functions are **always required** in an Arduino sketch, but you are of course able to add several more functions.
- ARDUINO LIBRARIES (librerías de Arduino) are an extension of the standard Arduino API, and consists of thousands of libraries, both official and contributed by the community. To use a library, you need to include it at the top of your code:
  - The way with **angle brackets** (corchetes angulares, #include <Wire.h>) includes header files (archivos de cabecera) for the **standard Arduino library**. The complier would search for the source code (código fuente) in the library folder in the **Arduino software directory**.
  - The way with **quotes** (comillas, #include "FreeIMU.h") includes **programmer-defined header files**. The complier would search for the source code in the **current project directory** or in the library folder in the **Arduino sketchbook directory**.

Universidad Rey Juan Carlos GISAT

Sketch build process

Arduino **sketch** undergoes several processes to be successfully **uploaded to the Arduino** board by the **Arduino software**.

- **Pre-Processing** (preprocesamiento): The Arduino software first performs some **minor transformations** to to turn your sketch into a **C/C++ program**.
- **Compilation** (compilación):
  - **Dependencies** (dependencias) of the sketch are located.
  - The code is then passed to a **C/C++ compiler** (e.g, avr-gcc) that turns the human readable code into **machine-readable instructions** (.o object files, archivos de objetos).
  - Then your code gets combined with (linked against) the **standard Arduino libraries**. The result is a **single Intel hex file** (archivo hexadecimal de Intel), which contains the specific bytes that need to be written to the **program memory of the microcontroller** on the Arduino board.
- **Uploading**: The hex file is uploaded (transmitted) to Arduino board through **USB or serial connection**.

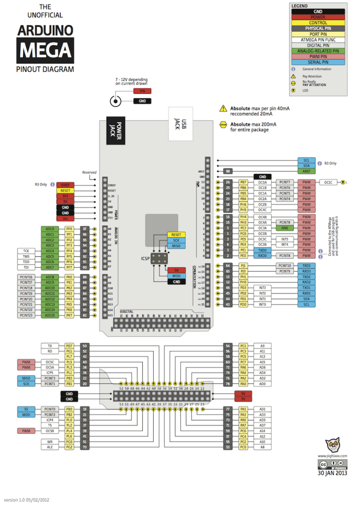
Arduino IDE workflow

- **Install your board** - this means installing the right "package" for your board. Without the package, you can simply not use your board. Installing is done directly in the IDE, and is a quick and easy operation.
- **Create a new sketch** - a sketch is your main program file. Here we write a set of instructions we want to execute on the microcontroller.
- **Compile sketch** (compilar) - the code we write is not exactly how it looks like when uploaded to our Arduino: compiling code means that we check it for errors, and convert it into a binary file (1s and 0s). If something fails, you will get this in the **error console** (consola de errores).
- **Upload sketch** (actualizar) - once the compilation is successful, the code can be uploaded to your board. In this step, we connect the board to the computer physically, and select the right serial port.
- **Serial Monitor** (Monitor Serie) - this tool allows to **visualize data** sent from your board to your computer. It also allows to **send data** from the computer to the board. It can be used as a **debugging tool** (herramienta de debugging/depuración).

Resources

- The Getting Started with Arduino page provides a short introduction to hardware, software tools, and the Arduino API.
- The Arduino Tutorials page provides a complete list of guides for using Arduino.
- The Arduino Language Reference page, as a user manual, provides a list of the basic elements of the language (structures, variables and functions).
- The Library page lists some build-in libraries and some of these additional libraries that can be installed to your Arduino IDE. .
- The modules provided by AVR-Libc is also available at the AVR Libc Home Page page.
- The Arduino Playground is a wiki where all users can share the source code, circuit diagrams, tutorials, etc.

Arduino Mega 2560

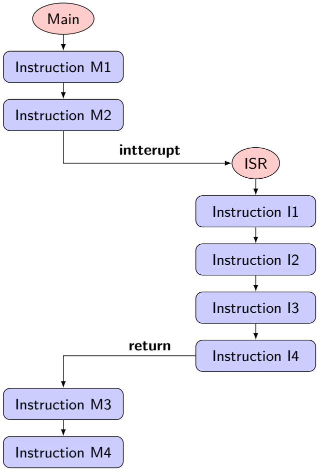


- 8-bit **microcontroller** ATmega2560 from Atmel
- 16 MHz device **clock**
- 7 V–12 V recommended **input voltage**
  - DC power jack (conector de alimentación)
  - VIN pin
  - USB
- 5 V **operating voltage** (voltaje de operación)
- 5 V and 3.3 V regulated power supply (fuente de alimentación regulada) for **external** components
- **maximum current**: 40 mA per pin and 200 mA for entire board
- **recommend current**: 20 mA per pin
- 54 **digital** I/O pins (pines de entrada/salida digitales), 16 **analogue** input pins (pines de entrada de señal analógica)

Universidad Rey Juan Carlos GISAT

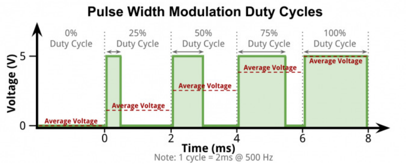
Digital pins

- pin0 - pin53
- **Input**: HIGH, $> 3\,\text{V}$; LOW, $< 3\,\text{V}$.
- **Output**: HIGH, $5\,\text{V}$; LOW, $0\,\text{V}$.
- **External Interrupts** (Interrupción): an **interrupt signal** alerts the processor that an event needs **immediate attention**. The processor **stops** whatever it was doing, **executes** a small chunk of code that is called **Interrupt Service Routine** (ISR, rutina de servicio de interrupción), and then **returns back** to what it was doing before.
  - pin2 (INT0), pin3 (INT1), pin21 (INT2), pin20 (INT3), pin19 (INT4), pin28 (INT5)
  - the interrupt **signal mode**: RISING (flanco acendente), FALLING (flanco descendente), CHANGE, LOW



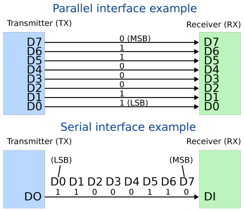
Digital pins

- **Pulse Width Modulation** (PWM, modulación por ancho de pulsos) is a technique to **simulate an average analogue voltage** (voltaje analógico promedio) by switching the digital signal between **on and off** to create a square wave.
- If you repeat this on-off pattern **fast enough** with an LED for example, the result is as if the signal is a **steady average voltage** controlling the **brightness of the LED**.
- **Duty cycle** (ciclo de trabajo) is the fraction of one period in that the digital signal is **HIGH**.
- pin2 - pin13, pin44 - pin46.
- 500 Hz frequency, 2 ms period for one cycle.
- 8-bit duty cycle: a call to analogWrite() is on a scale between 0 and 255 $= 2^8 - 1$, such that analogWrite(255) requests a 100% duty cycle and analogWrite(127) requests a 50% duty cycle.



Communication protocol

- A COMMUNICATION PROTOCOL (protocolo de comunicaciones) is a system of **rules and conventions** that allows two or more entities of a communication system to **transmit information**.
- In SERIAL COMMUNICATION, **bits** are sent **sequentially** by the **transmitter** (TX, transmisora), with the data being received and assembled one bit at a time by the **receiver** (RX, receptora). From TX to RX, the data transmission takes place on a **single wire**.



- In PARALLEL COMMUNICATION, **multiple bits** of a data packet are transmitted **simultaneously** by sending each bit over a single wire.
- As analogy to driving roads, a **parallel** interface is a 8-lane **mega-highway**, while a **serial** interface is a **two-lane rural road**.
- Over the same amount of time, the **mega-highway** potentially gets **more people** to their destinations, but that **two-lane road** serves its purpose and costs less.
- **Parallel** communication is **fast**, **straightforward**, and relatively **easy** to implement. But it requires many **more input-output lines**.
- In **Arduino**, **serial** communication is adopted since input-output lines are precious and few.

Universidad Rey Juan Carlos GISAT

Digital pins as UART serial ports

- Universal Asynchronous Receiver-Transmitter (UART, Transmisor-Receptor Asíncrono Universal) sends data bits one by one, typically from the **least significant** (menos significativo) to the **most significant** (más significativo), and framed by **start and stop bits** (bit de inicio, bit de parada). By default, Arduino uses **8N1** (8 bits, no parity, 1 stop bit) serial protocols.

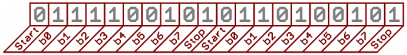
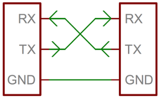

0 1 1 1 1 0 0 1 0 1 0 1 1 0 1 0 0 1 0 1
Start b0 b1 b2 b3 b4 b5 b6 b7 Stop Start b0 b1 b2 b3 b4 b5 b6 b7 Stop

Figure: A device transmitting two characters: ‘O’ with ASCII value of 79 (HEX 4F, BIN 01001111) and ‘K’ with ASCII value of 75 (HEX 4B, BIN 01001011).

- Asynchronous means that there is **no clock signal to synchronize data transmission** between the transmitter and the receiver.
- For this reason, both transmitter and the receiver need to have the **same** baud rate (tasa de baudios), which specifies **how fast** data is sent over a serial line, usually expressed in units of **bits-per-second** (bps). Common standard baud rates are 1200, 2400, 4800, 9600, 19200, 38400, 57600, and 115200 bps.

Digital pins as UART serial ports

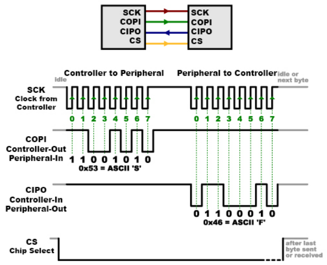
- **UART bus** consists of just **two wires** - one for sending data and another for receiving.
- Therefore, each device should have **two serial pins**: the receiver RX and the transmitter TX. Note that those **RX and TX** labels are with respect to the device itself. So the RX from one device should go to the TX of the other, and vice-versa.



- Mega2560 has 4 sets of serial ports: **Serial**, 0 (RX), 1(TX); **Serial1**, 19 (RX), 18(TX); **Serial2**: 17 (RX), 16(TX); **Serial3**: 15 (RX), 14(TX).
- UART is the communication protocol used to **communicate between the computer and the Arduino board** through the **USB cable**. The USB port is connected to **pin0 and pin1** of the serial ports set **Serial**.
- Arduino API provides the Serial library for UART implementation (e.g., Serial1.begin, **Serial.print**, Serial3.read, Serial2.write).

Digital pins as SPI serial ports

- SERIAL PERIPHERAL INTERFACE (SPI, interfaz periférica serial) is a **synchronous** serial data protocol used by one **controller device** to communicate with one or more **peripheral devices** (dispositivos periféricos).
- It's a SYNCHRONOUS (sincrónico) data bus, which means that it has a separate **SCK** (serial clock) line that keeps data transmission and reception in perfect sync.
- The clock is an **oscillating signal** that tells the receiver exactly when to **sample the bits** on the data line. When the receiver detects the **rising or falling edge** (flanco ascendente o descendente), it will immediately read the next bit from the data line.

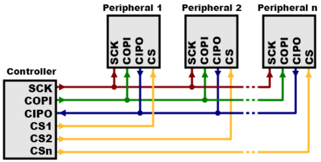


- Most SPI chips use **MSB first data order**.
- In SPI, the device that **generates the clock** is called the CONTROLLER (master, controlador), and the other device is called the PERIPHERAL (slave, periférico).
- The data is sent **from the controller** to a peripheral on the **COPI** (Controller Out Peripheral In, or MOSI) line. The data is sent **from a peripheral** to the controller on the **CIPO** (Controller In Peripheral Out, or MISO) line.
- The controller can use the **CS** (Chip Select, or SS for Slave Select) line to activate (LOW) and deactivate (HIGH) a peripheral device.

Universidad Rey Juan Carlos GISAT

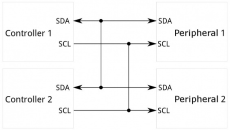
Digital pins as SPI serial ports

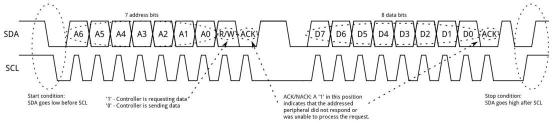
- By using the CS line, **multiple peripherals** can be connected to one controller. In general, each peripheral will need a separate CS line. To talk to a particular peripheral, the controller will make that peripheral's CS line low and keep the rest of CS lines high.
- Mega2560 has 1 set of SPI ports: SCK (pin52 or ICSP-3), CIPO (pin50 or ICSP-1), COPI (pin51 or ICSP-4), CS (pin53).
- Arduino API provides the SPI library for SPI implementation (e.g., SPI.transfer, SPI.beginTransaction).



Digital pins as I2C serial ports

- Inter-Integrated Circuit (I2C, circuito inter-integrado) is a **synchronous** protocol has only two lines: a **SCL serial clock line** that the controller board pulses at a regular interval and a **SDA serial data line** over which data is transmitted between the controller and the peripheral.
- The I2C protocol allows the communication among multiple peripherals and multiple controllers.
  - Each device has an **unique address**.
  - In the **address frame** (trama de dirección), a **read/write bit** indicates whether this is a read or write operation. Therefore, both controller and peripheral devices can take turns to communicate over a **single data line**.
- Mega2560 has 1 set of I2C ports: SCL (pin21), SDA (pin20).
- Arduino API provides the Wire library for I2C implementation (e.g., Wire.begin, Wire.read, Wire.write).




7 address bits
8 data bits
SDA
SCL
A6
A5
A4
A3
A2
A1
A0
R/W
ACK
D7
D6
D5
D4
D3
D2
D1
D0
ACK
Start condition: SDA goes low before SCL
'1' - Controller is requesting data
'0' - Controller is sending data
ACK/NACK: A '1' in this position indicates that the addressed peripheral did not respond or was unable to process the request.
Stop condition: SDA goes high after SCL

Analogue Input and Memory

- Arduino Mega2560 contains **16 analogue input pins** (pinA0 - pinA15) that are connected to **10-bit analogue to digital converters** (ADC, conversión analógica-digital). These converters map input **voltages** between $0\,\mathrm{V}$ and $5\,\mathrm{V}$ into **integer values** between 0 and $1023 = 2^{10} - 1$, with the resolution being $5\,\mathrm{V}/1023 \approx 4.9\,\mathrm{mV}$.
- Arduino Mega2560 has three types of memory:
  - The 8kB **Static Random-Access Memory** (SRAM, memoria estática de acceso aleatorio) is used to **create and manipulate variables**. It is **volatile** (volátil) memory because the data is lost when power is removed.
  - The 256kB **Flash memory** (memoria flash) is primarily used to **store Arduino sketch**. It is **non-volatile** memory because the data is not erase when power is removed, so that the instructions for the microcontroller are executed as soon as the board is powered.
  - The 4kB **Electrically Erasable Programmable Read-Only Memory** (EEPROM, memoria de solo lectura programable y borrable eléctricamente) is **non-volatile** and **users can read and write** these bytes.

Universidad Rey Juan Carlos GISAT

Example 1: Trigger LED

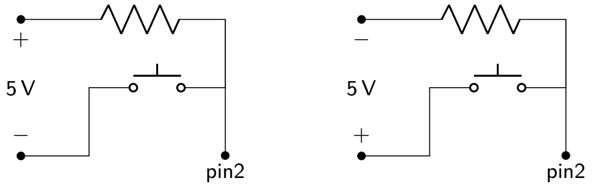
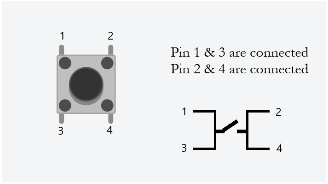


Figure: Left, pull-up resistor: press (presionar), LOW; release (soltar), HIGH. Right, pull-down resistor: press, HIGH; release, LOW.



Example 1: Trigger LED

- **Digital I/O**
  - pin2, Input, interrupt pin;
  - pin13, Output, connected to the internal LED.
- **Interrupt**
  - Look for the “attachInterrupt” in the Arduino Language Reference page
  - Understand the **syntax** (sintaxis). What are the inputs of the function? How to use this function?

    attachInterrupt(digitalPinToInterrupt(pin), ISR, mode)
- Construct the electronic **circuit**. Type the example **sketch** in Arduino IDE. **Compile**. **Upload**. ¡Voilà!
- **Understand** every line of the code. If you have question, first ask **Google**.
- Questions to answer
  - What does the blink function do?
  - Why can the variable *state* be accessed in both *loop()* and *blink()*?
  - What are the three input entries for the attachInterrupt function?
  - What are these three data types: *const*? *byte*?
  - The resistor is a pull-up or pull-down resistor?
  - When the button is not pressed, does the digital input pin read HIGH or LOW?
  - Try with different interrupt modes (CHANGE, RISING, FALLING). Would the LED change its status for pressing or releasing the button?

Example 2: Sweep Servo

- **Digital I/O**
  - pin12, Pulse Width Modulation (PWM) pin with Duty Cycle
- **Servo library** to control the **servo step motor**.
- Questions to answer
  - How to include a library in the sketch? What is the difference between *#include <xxx.h>* and *#include "xxx.h"*?
  - In the sketch, which one is the class of the servo motor? Which is the object of servo motor?
  - In the sketch, how to use one function provided by the class?
  - If you need to control several motors, what changes do you need to make for the circuit and for the sketch?

Exercise 1: Servo and potentiometer

- **Analogue input** and “analogRead()” function.
- You can not use the Map function.
- Questions to answer.
  - What are the three pins of a potentiometer?
  - What does an analogue input do? Arduino Mega 2560 map voltages between ____________ into integer values between ______.
  - Is there analogue output pin?
  - How can we imitate an analogue output pin using digital pins?
  - What are the differences between the data types *int* and *float*?
  - What is the problem of the division operation between integers? How can you force the division to perform floating point arithmetic?

Exercise 2: Servo and USB as UART port

- **UART** port and **USB**
- **Serial monitor**
- Questions to answer.
  - Which library is used for UART communication protocol?
  - To use the UART ports, do we need to creat an object from a class?
  - There are four sets of serial port in Arduino Mega, to which set is the USB port connected?
  - How to set up the baud rate so that Arduino Mega and Serial Monitor can communicate successfully using UART protocol?
  - Following which convention is the data type *char* codified and stored as binary data?
  - Which of the following function sends data from the Arduino board to the USB port as human-readable ASCII text? *Serial.print()* or *Serial.write()*?
  - What are the binary data sent from Arduino board to the USB by *Serial.write(4)* and *Serial.write("4")*?
  - How does *Serial.read()* decode and interpret the data arriving at the USB port of Arduino Mega?
  - How does the Serial Monitor of Arduino IDE decode and interpret the data arriving at the USB port of the computer?

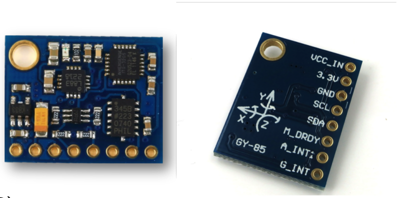
Exercise 3: Barometer with SPI and I2C protocol

- **SPI** and **I2C** communication protocols
- Questions to answer.
  - What is the difference between *#include <xxx.h>* and *#include "xxx.h"*? Where is the file "bmp280.h" stored?
  - SPI and I2C protocol.
    - For each protocol, how many pins are connected?
    - What is each pin used for?
    - Does the protocol support multiple peripherals? Why?
    - Does the protocol support multiple controllers? Why?
    - For one-to-one communication, how many data lines are required for this protocol? Why?
  - What is the difference between SPI and software SPI?
  - How to
    - include the library?
    - create the object?
    - initiate the barometer?
    - acquire data from the sensor?
    - send the data to the USB port?

Acrónimos

AHRS	Attitude and Heading Reference System Sistemas de Referencia de Actitud y Rumbo
API	Application Programming Interface Interfaz de programación de aplicaciones
CIPO/COPI	Controller In Peripheral Out, Controller Out Peripheral In Salida de Periférico y Entrada a Controlador, Salida de Controlador y Entrada a Periférico
IDE	Integrated Development Environment Entorno de Desarrollo Integrado
ISR	Interrupt Service Routine Rutina de Servicio de Interrupción
LSB/MSB	Least/Most Significant Bit Bit menos/más significativo
MISO/MOSI	Master In Slave Out, Master Out Slave In Salida de Esclavo y Entrada a Master, Salida de Master y Entrada a Esclavo
PWM	Pulse Width Modulation Modulación por ancho de pulsos
RX/TX	Receiver/Transmitter Receptor/Transmisor
UART	Universal Asynchronous Receiver/Transmitter Transmisor-Receptor Asíncrono Universal

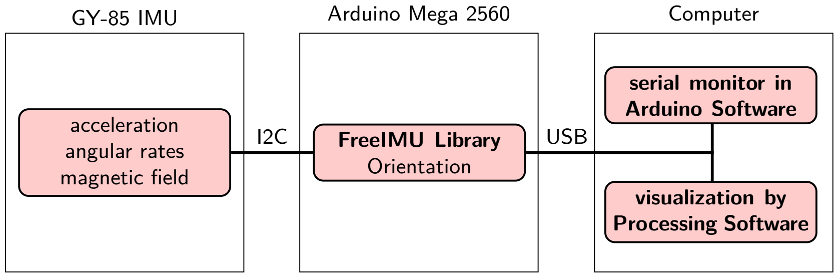
MARG GY-85 9 DOF IMU



- 9 DOF IMU (MARG)
- MEMS

Sensor	MEMS	Precision
Accelerometer	ADXL345	13-bit
Gyroscope	ITG3205	16-bit
Magnetometer	HMC5883L	12-bit

IMU - Arduino - Computer



MARG GY-85	Arduino Mega
VCC_IN	3.3V
GND	GND
SCL	SCL (PIN 21)
SDA	SDA (PIN 20)

FreeIMU library

FreeIMU library is an Open Source project which aims to develop libraries to use an **IMU** based on Arduino platform to perform orientation sensing.
This library contains four parts:

- **Python** scripts for **calibration**.
- **Libraries** for **Arduino**
- **Example ketches** for **Arduino**.
- **Processing** sketches for **visualization**.

FreeIMU library for Arduino

Library files: “ARD_SKETCH_LOC/libraries/FreeIMU”

- **FreeIMU.cpp**: sensor fusion algorithms to obtain orientation.
- **CommunicationUtils.cpp**: wrapper function that decides the format of the data transmitted between the computer and the Arduino.
- **calibration.h**: calibration parameters for accelerometer and the magnetometer (scaling factors and the offsets).

Arduino example Sketch: “ARD_SKETCH_LOC/freeimu”

- **FreeIMU_quaternion.pde**: The orientation information of the platform is sent to the USB port in the form of quaternions.
- **FreeIMU_raw.pde**: The un-calibrated data from the accelerometer, gyro, and magnetometer are sent to the USB port.
- **FreeIMU_serial.ino**: According to the commands received from the computer to the Arduino through the USB port, the Arduino board will send corresponding data in correct format. These data will later be accessed by **Processing for visualization**, or by a python code developed for **calibration**.

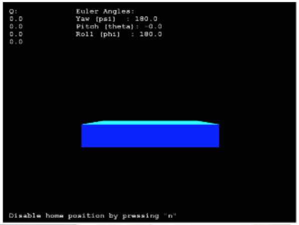
FreeIMU library for Processing

Processing is a flexible **open source** software sketchbook and a language to code within the context of the **visual arts**.
In this lab we use it to visualize the orientation of the platform. These scripts are placed in "PROC_SKETCH_LOC/FreeIMU":

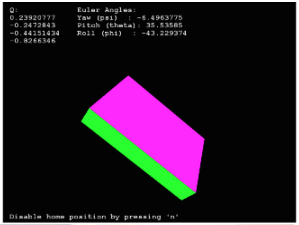
- **FreeIMU_cube.pde**
- **Artificial_horizon.pde**

Task 1

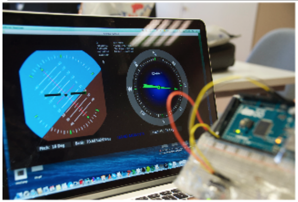
- Read and understand FreeIMU library
- Understand communication between IMU, Arduino board and Processing

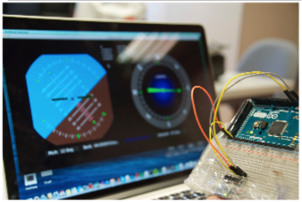


```
Q:                Euler Angles:
0.0               Yaw (psi)    : 180.0
0.0               Pitch (theta): -0.0
0.0               Roll (phi)   : 180.0
0.0

Disable home position by pressing 'n'
```



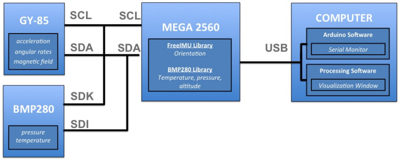
```
Q:                Euler Angles:
0.23920777        Yaw (psi)    : -6.4963775
-0.2472843        Pitch (theta): 35.53585
-0.44151434       Roll (phi)   : -43.229374
-0.8266346

Disable home position by pressing 'n'
```

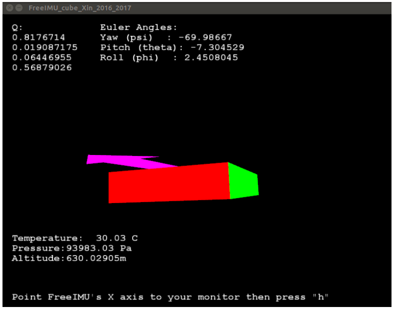




Task 2

- Display information from the barometer and the IMU in Processing. **Detailed steps are explained in the notes**.

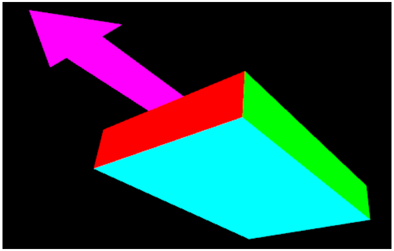




```
Q:              Euler Angles:
0.8176714       Yaw (psi)   : -69.98667
0.019087175     Pitch (theta): -7.304529
0.06446955      Roll (phi)   : 2.4508045
0.56879026

Temperature:  30.03 C
Pressure:93983.03 Pa
Altitude:630.02905m

Point FreeIMU's X axis to your monitor then press "h"
```

Task 3

- Modify the *FreeIMU_cube.pde* to add an arrow on the cube



Summary

Arduino Lab Sessions

- Hardware specifications and capabilities.
- Programming Arduino
  - Read example code.
  - Ask proper questions. **Use common sense and logic to guess the answer and verify with tests?**
  - Learn by yourself.
  - **!!!Syntax!!!!**
- Develop the capability of understanding and implementing a hardware-oriented programming.

Inertia Navigation System

- What is Inertia Navigation System?
  - IMU for measurements;
  - Arduino for navigation;
  - Computer for visualization.
- **Communication protocol.**
- Be decisive
  - Capability to grasp the high-level architecture of data acquisition and data communication.
  - Implement without being lost in the details.

Acrónimos

IMU	Inertial Measurement Unit
	Unidad de medición inercial
INS	Inertial Navigation System
	Sistema de navegación inercial
MEMS	MicroElectroMechanical System
	Sistemas MicroElectroMecánicos