Lab 1: Arduino

Part 1: Introduction to the microcontroller

1. Introduction

In the control and management of most electronic instruments, it is customary to delegate tasks to a computer specialized in such a task. Even though it is possible to find 100% analogue devices among medium-aged instruments, this fact is a chimaera today due to their high cost of development and maintenance.

The digitization of any measure has its cost as a loss of precision; However, these shortcomings are mitigated by the outstanding resolution of the ADCs (analogue-digital converter) and the high transfer speeds. Along with the high performance of the built-in computers.

These computers (or microcontrollers) have specific programming for the task they control, which can be modifiable to adapt to increases in the number of sensors or actuators (scalability), bug correction, etc. Thus, they provide us with modularity that did not possess the analogue instruments.

2. Objectives

This practice aims to introduce a low-cost microcontroller. The Arduino Nano RP2040 Connect is a development board in Nano format, based on the RP2040 microcontroller. It features a Wi-Fi / Bluetooth® module, a 6-axis IMU (Inertial Measurement Unit) with machine learning capabilities, a microphone and a built-in RGB.

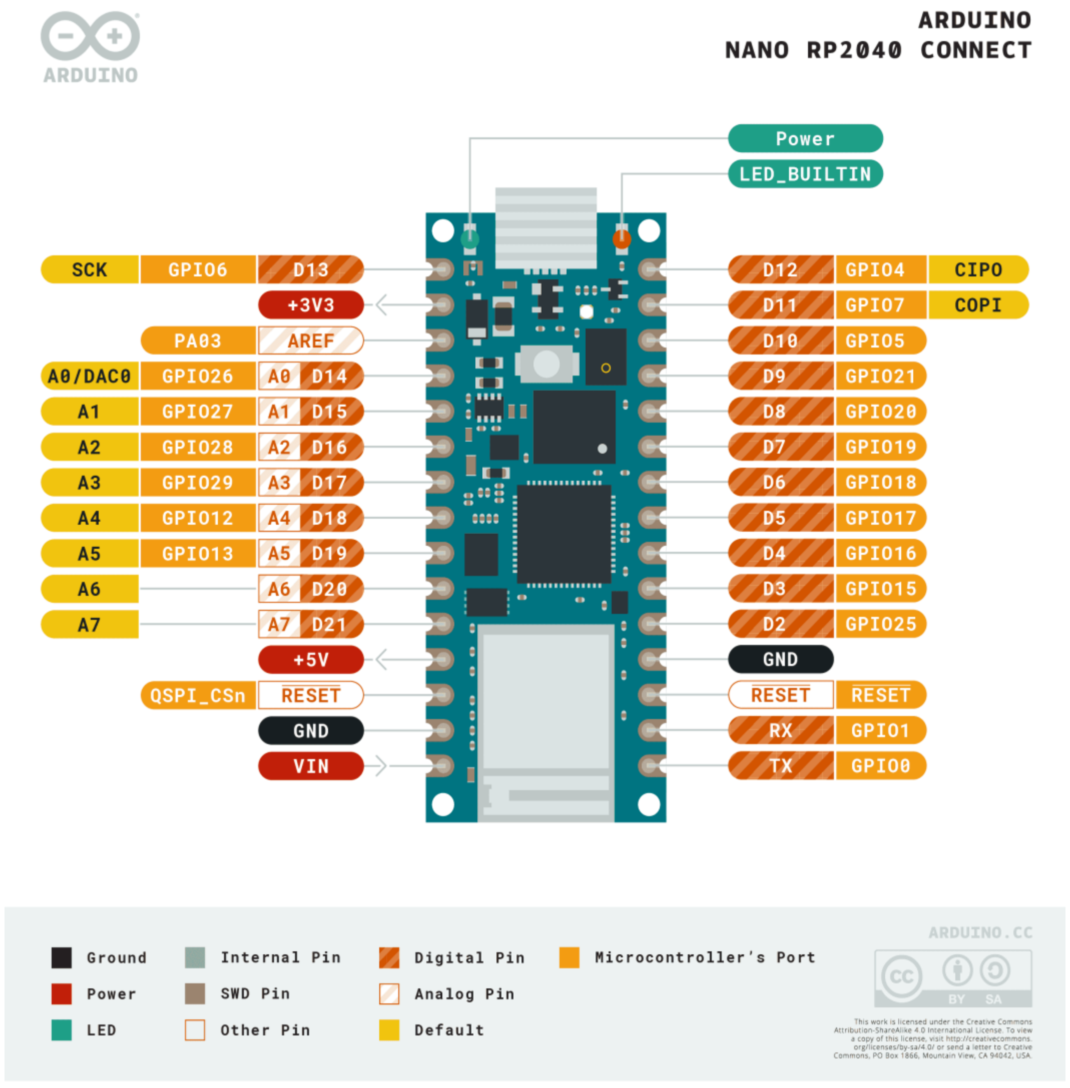
As a second objective, it is intended to introduce the programming environment and language of this board and the basic structure of a program. The Arduino Nano RP2040 is provided with various sensors to design your own circuits on his prototyping board and interconnect them with the Arduino. Some programmes will then be carried out to check their operation.

3. Insights into the Arduino Kit

The feature packed **Arduino Nano RP2040 Connect** brings the new **Raspberry Pi RP2040** microcontroller to the Nano form factor. Make the most of the dual core **32-bit Arm® Cortex®-M0+** to make Internet of Things projects with Bluetooth® and WiFi connectivity thanks to the **U-blox® Nina W102** module. Dive into real-world projects with the onboard accelerometer, gyroscope, RGB LED and microphone.

3.1 The Arduino nano RP2040 PCB

The PCB (Printed Circuit Board) used in this practice can be seen below:

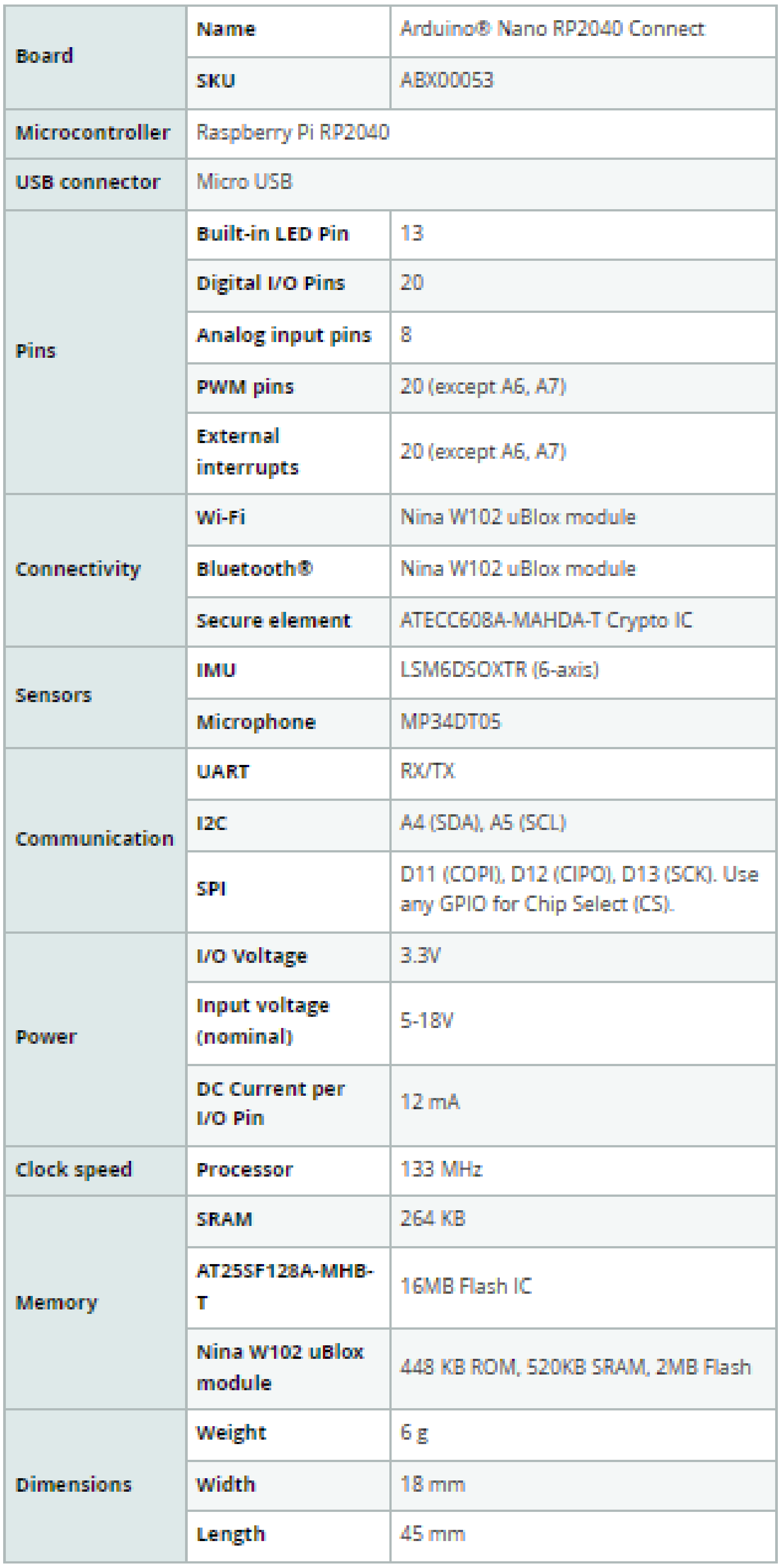


The Arduino® Nano RP2040 Connect is a development board in Nano format, based on the RP2040 microcontroller. It features a Wi-Fi / Bluetooth® module, a 6-axis IMU (Inertial Measurement Unit) with machine learning capabilities, a microphone and a built-in RGB.

The full datasheet is available as a downloadable PDF from the link below:

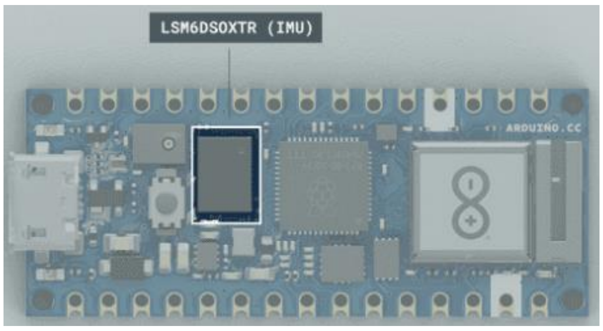
Arduino_nano_rp2040_datasheet

3.2 Technical specifications



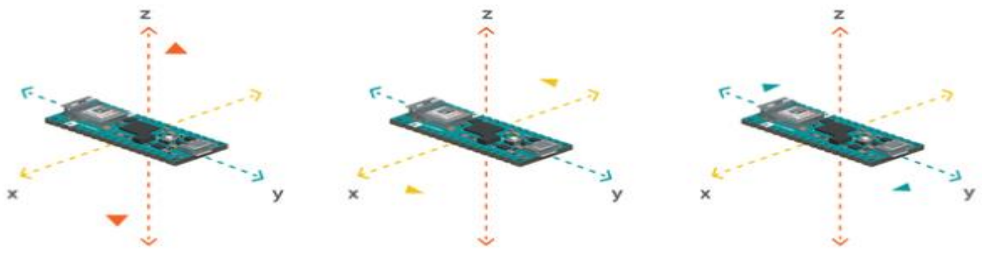
Board	Name	Arduino® Nano RP2040 Connect
	SKU	ABX00053
Microcontroller	Raspberry Pi RP2040	
USB connector	Micro USB	
Pins	Built-in LED Pin	13
	Digital I/O Pins	20
	Analog input pins	8
	PWM pins	20 (except A6, A7)
	External interrupts	20 (except A6, A7)
Connectivity	Wi-Fi	Nina W102 uBlox module
	Bluetooth®	Nina W102 uBlox module
	Secure element	ATECC608A-MAHDA-T Crypto IC
Sensors	IMU	LSM6DSOXTR (6-axis)
	Microphone	MP34DT05
Communication	UART	RX/TX
	I2C	A4 (SDA), A5 (SCL)
	SPI	D11 (COPI), D12 (CIPO), D13 (SCK). Use any GPIO for Chip Select (CS).
Power	I/O Voltage	3.3V
	Input voltage (nominal)	5-18V
	DC Current per I/O Pin	12 mA
Clock speed	Processor	133 MHz
Memory	SRAM	264 KB
	AT25SF128A-MHB-T	16MB Flash IC
	Nina W102 uBlox module	448 KB ROM, 520KB SRAM, 2MB Flash
Dimensions	Weight	6 g
	Width	18 mm
	Length	45 mm

3.2 IMU Module

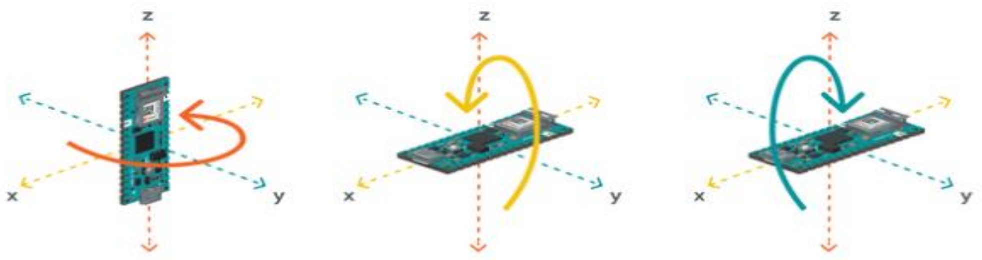


The LSM6DSOXTR from STM is an IMU (**Inertial Measurement Unit**) that features a **3D digital accelerometer** and a **3D digital gyroscope**. An IMU is a component that measures movement such as specific force, angular rate or orientation. It does that by combining a gyroscope and an accelerometer sensor. In addition to measuring the raw movement data, the IMU mounted on the Nano RP2040 Connect has a **machine learning core**, which is useful for any motion detection projects, such as free fall, step detector, step counter, pedometer. This allows to do advanced movement detection on the IMU itself. This frees the microcontroller from this task so it can do other things like sending data to a network, interacting with other sensors or operating actuators. By using the machine learning core specific activity patterns such as walking, jogging or biking can be recognized. The patterns that can be recognized by the IMU depend on the machine learning model that is used. To access the data from the LSM6DSOX module, we need to install the **LSM6DSOX library**, which comes with examples that can be used directly with the Nano RP2040 Connect. It can be installed directly from the library manager through the IDE of your choice. To use it, we need to include at the top of the sketch. This module also features an **embedded temperature sensor**.

Accelerometer



Gyroscope



3.3 - The programming environment

The Arduino IDE allows you to write programs and load them to your Arduino.

Download and install **Version: Arduino IDE 1.8.19** of the IDE from:

https://www.arduino.cc/en/software

3.4 Programming language

Most microcontrollers are programmed using an imperative programming language similar to C. An introductory program for an Arduino board is divided into five main parts. The basic structure of a program is as follows (it is recommended to keep this order in mind):

Element	Example
1. Library inclusion	#include <LiquidCristal.h>
2. Definition of constants	#define PI 3.141592
3. Global variables	boolean LedState = true;
4. Setup function	void setup() {...}
5. Loop function	void loop(){…}
6. Other functions	int miFun(int val){…}

1. **Library inclusions** commonly refer to standard peripherals, to provide the communication interfaces needed to control these external devices. When installing the environment, the libraries can be consulted in the path "C:\Program Files (x86)\Arduino\libraries" or whenever you installed it (OS dependent).

2. **Constants** are elements whose value cannot be modified during code execution. Concerning the microcontroller, they do not occupy memory as their value is substituted at compile time.

3. **Global variables** are usually used to represent the pins to which the peripherals are connected since these values must be used, at least, in the following two functions (setup and loop). Further global variables can be defined for other purposes. For both global and local variables, Arduino provides the following types:

Type	Description	Examples of Values
boolean	Logical value	true, false.
char	Character (ASCII). Integer 8 bits.	‘a’, ‘A’, ‘.’, …
byte	Integer of 8 bits (unsigned)	0 – 255
int	Integer of 16 bits.	[-215, 215-1]
long	Integer of 32 bits.	[-231, 231-1]
float	Float of 16 bits.	3.15, -1.3, …
double	Float of 32 bits.	3.15, -1.3, …

You can also work with compound types, such as arrays and character strings: Examples:

Array: int v [10] = {1,2,3,4,5,6,7,8,9,10}; || String: char cad [50] = "Hello World!";

In addition, various modifiers can be applied to the variables to obtain different results:

Modifier	Result
unsigned	Applicable to integers. Unsigned number.
const	It means that the value of the variable cannot be modified.
volatile	Directs the compiler to load the variable from RAM and not from a storage register. A variable must be declared volatile whenever its value can be modified by something beyond the section of code in which it appears, such as a concurrent thread of execution.
static	They are not created and destroyed every time the code block in which they are defined is called, but their value is saved for subsequent calls.

4. **Setup function:** This function is executed only once in the program and it corresponds to the device initialization. In this function, it will be necessary to establish the operating modes of each of the peripherals and initialize the pins to be used.

5. **Loop function:** This function is executed repeatedly in our program, therefore this function will be the right place to carry out the readings and / or writings on the peripherals.

6. **Other functions:** it is recommended that you encapsulate certain functionalities aside to make the code more readable and reusable.

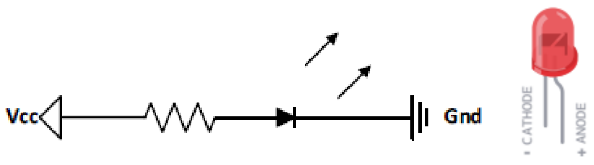
*You can consult a summary of all the operators and functions contemplated in the Arduino basic library in the appendix of this lab.

3.4 Basic elements

(You can skip this section if you are familiar with these electronic components. They are not going to be used in this lab as external components, we are only using the LEDs integrated on the board).

LED

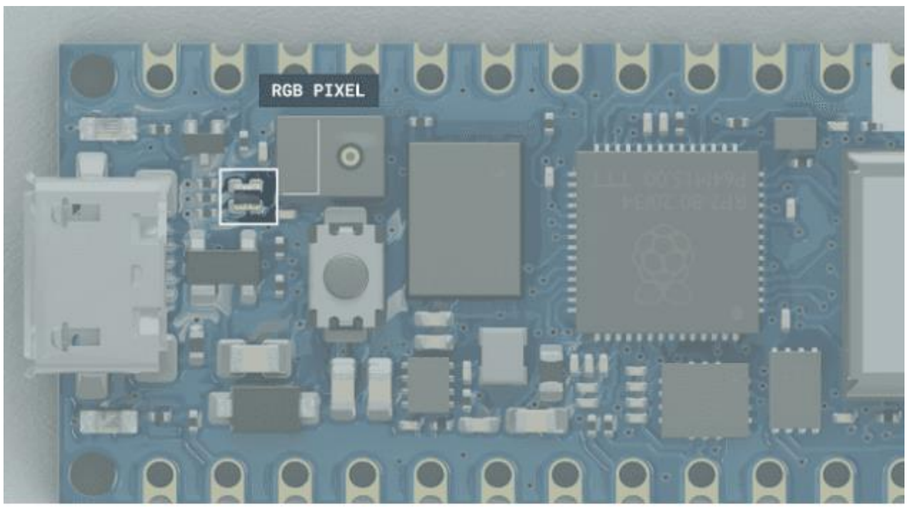
A passive electronic component that emits light. It is made up of an anode and a cathode. When connecting it to a power supply, consider the voltage required by the led diode (usually around 2-3V, depending on the color). For safety measures, use a resistor to mitigate the overvoltage. The connection of an LED is as follows:



RGB LED

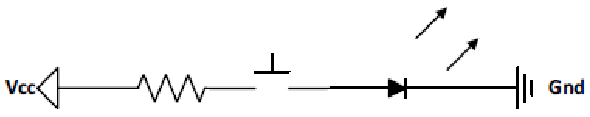
Component similar to the conjunction of three LEDs of different colours: red, green and blue. It has four pins that correspond to the three diodes located inside and the ground. To control and set it in a circuit, it is done identically to a common led diode: use a resistor at each colour component's input to avoid overvoltage. The ground is usually the longest pin.

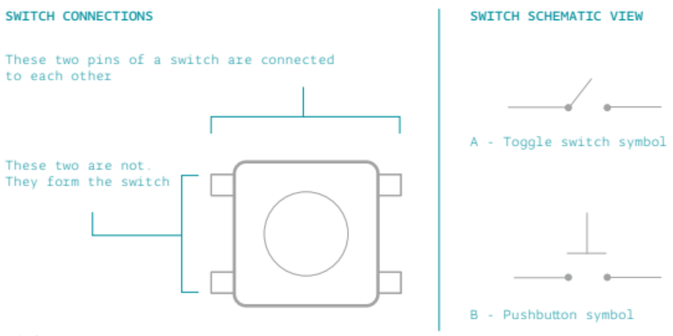
The figure shows where the RGB-LED is placed on the Arduino PCB.



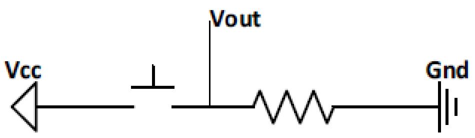
Interruptor / Switch

Device that allows diverting or interrupting the course of an electric current. To connect it in series within the previous circuit, insert it between the resistor and the LED:



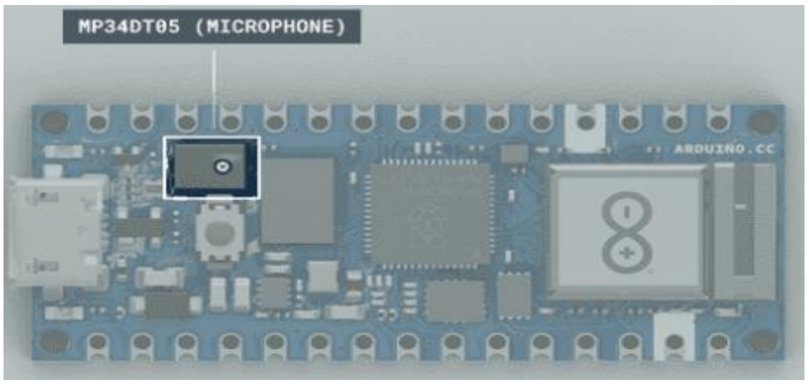

SWITCH CONNECTIONS
These two pins of a switch are connected to each other
These two are not. They form the switch
SWITCH SCHEMATIC VIEW
A - Toggle switch symbol
B - Pushbutton symbol

The switch is to be connected as a digital input of the microcontroller. A connection must be created between the power supply pin of the microcontroller with the digital input pin. Keep in mind to use a pull-down resistor to avoid inconsistencies in the rest state (this resistance is usually 10K):



3.5 Additional elements on Arduino nano rp2040

Microphone



Microphones are components that convert physical sound into digital data. Microphones are commonly used in mobile terminals, speech recognition systems or even gaming and virtual reality input devices. The MP34DT05 sensor is an ultra-compact microphone that use PDM (Pulse-Density Modulation) to represent an analog signal with a binary signal. The sensor's range of different values are the following:

- Signal-to-noise ratio: 64dB
- Sensitivity: -26dBFS ±3dB

- Temperature range: -40 to 85°C

If you want to read more about the MP34DT05 sensor you can take a look at the datasheet.

This Arduino includes many other features such as Wi-Fi or Bluetooth. However, such features will not be explained here as they are not necessary for this lab.

Serial port

Used for communication between the Arduino board and a computer or other devices. All Arduino boards have at least one serial port (also known as a UART or USART), and some have several. During this practice you will have to look for information (on the internet) on how to use the serial port to see values collected by the Arduino.

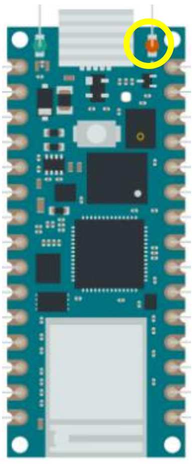
Part 2: Exercises

Part 2.1: Introductory exercises. First contact with the sensors.

During the exercises you will have to look for information on the internet about how to configure the Arduino. The idea is that you learn by doing. The literature is very extensive and you will find many examples. A good place to start is the Arduino official page and especially for Arduino nano.

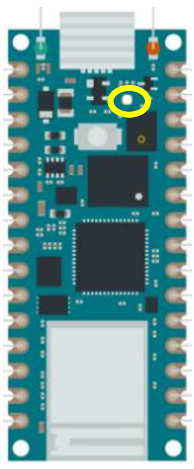
Exercise 0: Test that the development environment is working as expected by loading the Blink sketch which blinks the on-board LED:

1. Connect your Arduino Nano RP2040 Connect using a micro-USB cable to the host PC and let any drivers load.
2. The factory default code should run. The green power LED should be solid, the built-in LED should be flashing orange and the RGB LED should cycle through red/green/blue.
3. Run the Arduino IDE
4. From the file menu open File > Examples > Basics > Blink to load the code into the editor.
5. Open Tools > Board. Go to the Board Manager and install Arduino Mbed OS Nano Boards. Close the Board Manager
6. Open Tools > Board > Arduino Mbed OS Nano Boards > Arduino Nano RP2040 Connect to set the board type.
7. Select the port from Tools > Port > Com9 (Arduino Nano RP2040 Connect) or whatever port your device connected to (you can find the port name on the lower right corner of your IDE).
8. Click the Upload button to transfer the code to the Arduino Nano RP2040 Connect.
9. Now the orange LED next to the USB connector should blink once every second, meaning your code was successfully uploaded.



Exercise 1: Internal RGB

Implement the necessary code to make the RGB led go from red to blue, passing through green every half second. Create a new Sketch named ***Exercise1_RGB***. Load the sketch into the Arduino nano and display it. You can get inspired by reading Appendix 1 at the end of this document. You can also search for the information on the Internet (https://www.arduino.cc/reference/en/, you can use this tutorial for inspiration https://docs.arduino.cc/tutorials/nano-rp2040-connect/rp2040-web-server-rgb). For this exercise you will need to include the library WiFiNINA.h. To install the Library, go to Sketch > Include Library > Manage Libraries and search for WiFiNINA.



Exercise 2: Temperature sensor

In this exercise we are going to make use of the temperature sensor. Create a new Sketch named ***Exercise2_Temperature.*** You can reuse at the beginning the sketch from Exercise 1. For this you will have to make use of the IMU module. You can find here the information on how to use it. https://docs.arduino.cc/tutorials/nano-rp2040-connect/rp2040-01-technical-reference

Implement a code that displays the temperature in degrees Celsius through the UART serial port. Also implement a function in which, depending on the temperature, the RGB led will be red if the temperature is higher than 32 degrees, green if the temperature is between 20 and 36 degrees and blue, for temperature values lower than 25 degrees. To open the Serial Port console go to Tools > Serial Monitor. (You will realise that the thermometer gives a temperature higher than the environment and would need to be adjusted for production)

Exercise 3: Microphone

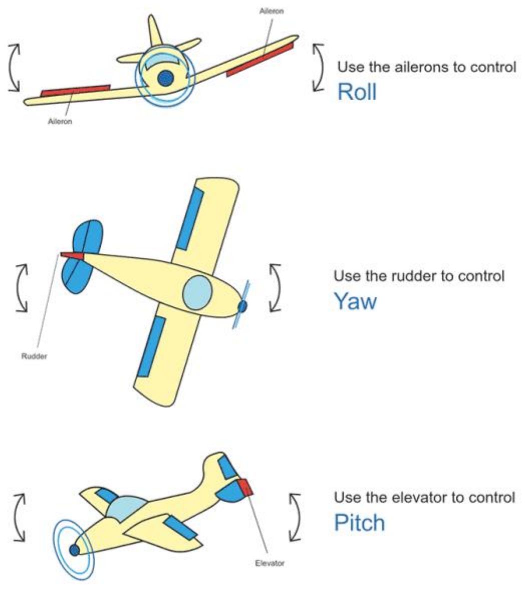
In this exercise we will make use of the microphone. Create a new Sketch named ***Exercise3_Microphone.*** The goal of the exercise is to try and understand the interactions with the microphone. It is acutally already solved in this tutorial. Please read it and try. Ask whatever you do not understand. You can see the microphone signal with Tools > Serial Plotter. (You may get some warnings while compiling, but it should work anyway).

Part 2.2: Posture Detector. Accelerometer

In part 1, it was possible to work with a low-cost microcontroller to manage certain control elements such as LEDs. Every electronic instrument consists of a multitude of elements much more complex than LEDs. This series of more complex components can come together to comprise the core of the instrument itself. Special attention is required to those in charge of collecting samples or obtaining specific information from the medium among the additional elements. These elements are called sensors. For example, an electronic or medical instrument is usually made up of a multitude of sensors.

Exercise 4

In this exercise, the objective is to develop a device that detects that a person's posture is correct. Create a new Sketch named ***Exercise4_Posture.*** The device should also include a sensor that measures the temperature of the room to check that we are in an ideal temperature to work. You will have to use the IMU module to develop all the above. The serial port will be used to display the information on the screen. You could also include some kind of alarm in case the temperature is too high or the position is wrong. For example, by turning on the RGB led. To detect the posture, you can consider that the correct posture is 90º with respect to the floor or a table (considering one axis). To make things easier, you can use the Madgwick library. This library wraps the official implementation of the **MadgwickAHRS** algorithm to get the orientation of an object based on accelerometer and gyroscope readings. To do this you must focus on the y-axis (pitch). We will consider that the ideal position for the back will be to have the back at 90º with respect to the horizontal. Therefore, you must choose a threshold for which the position of the back is incorrect (for example less than 80º).


Aileron
Aileron
Use the ailerons to control
Roll
Rudder
Use the rudder to control
Yaw
Elevator
Use the elevator to control
Pitch

For the exercise you can look at the example at: https://github.com/arduino-libraries/MadgwickAHRS/blob/master/examples/Visualize101/Visualize101.ino.

- The example is using the CurieIMU – you should change the library to use the IMU from nano as in the previous exercises.
- Declare the Magdwick filter and set the rate of the filter to be 104.00 Hz (which is the sample rate of the nano IMU).
- You can ignore the extra rate variables of the accelerometer and gyroscope for this exercise, as well as their range.
- You can also ignore the use of the micros() function to set the reading rate of the IMU because we are using the same sample rate as the reading rate of IMU (104.00 Hz). In case you would like to reduce the reading (like in the example to 25 Hz) use the micro function.
- To read the accelerometer and gyroscope of the IMU, follow the instructions on https://docs.arduino.cc/tutorials/nano-rp2040-connect/rp2040-01-technical-reference . You can use those values then to feed the Madgwick filter on: filter.updateIMU(gx, gy, gz, ax, ay, az); You can find more details on the IMU-functions on https://www.arduino.cc/reference/en/libraries/arduino_lsm6dsox/

Apendix I: Operators and Functions

Basic operators and functions:

- Arithmetic operators : +, -, *, /, %, =
- Comparators: <, >, <=, >=, ==, !=
- Booleans: &&, ||, !
- Bit-level operators: &, |, ~, ^, <<, >>
- Compound operators: ++, --, +=, -=, *=, /=, &=, |=
- Conversion functions:
- char **char**(x), byte **byte**(x), int **int**(x), word **word**(h, l), long **long**(x), float **float**(x).
- Size: int **sizeof**(x)

I/O:

- Digital:
  - void **pinMode**(numPin, INPUT|OUTPUT)
  - void **digitalWrite**(numPin, HIGH|LOW)
  - HIGH|LOW **digitalRead**(numPin)
- Analogue:
  - void **analogReference**(DEFAULT|INTERNAL|EXTERNAL)
  - int **analogRead**(numPin)
  - void **analogWrite**(numPin, valor)

Time control:

- unsigned long **millis**(): time in milliseconds from the beginning
- unsigned long **micros**(): time in microseconds from the beginning
- void **delay**(ms): pause execution in milliseconds.
- void **delayMicroseconds**(us): pause execution in microseconds.

Calculation:

- typeX **min**(typeX, typeX)
- typeX **max**(typeX, typeX)
- typeXunsigned **abs**(typeX)
- x|a|b **constrain**(x, a, b): return x if ε [a, b], a if x<a, b if x>b
- double **pow**(base, exponent)
- double **sqrt**(num)

Trigonometry:

- double **sin**(float)
- double **cos**(float)
- double **tan**(float)

Random numbers:

- void **randomSeed**(semilla)
- long **random**(max)
- long **random**(min, max)

Bits and Bytes:

- byte **lowByte**(x)
- byte **highByte**(x)
- int **bitRead**(x, n)
- void **bitWrite**(x, n, b)
- void **bitSet**(x, n)
- void **bitClear**(x, n)
- int **bit**(n)

External interrupts: Arduino UNO only has 2 external interrupts (0: associated with digital pin 2; 1: associated with digital pin 3)

- void **attachInterrupt**(numInter, funcion, LOW|CHANGE|RISING|FALLING)
- void **detachInterrupt**(numInterrupcion)

Global interrupts:

- void **interrupts**()
- void **noInterrupts**()

Part 3: Introduction to MicroPython

1. Introduction

The objective of this practice is to further explore the functionality that the Arduino nano rp2040 can offer. For this we will explore how to use pre-trained machine learning models.

1.1 MicroPython

Since its launch in 1991, the Python programming language – named after the famous comedy troupe Monty Python, rather than the snake – has grown to become one of the most popular in the world. Its popularity, though, doesn't mean there aren't improvements that could be made – particularly if you're working with a microcontroller. The Python programming language was developed for computer systems like desktops, laptops, and servers. Microcontroller boards like Raspberry Pi Pico are smaller, simpler, and with considerably less memory – meaning they can't run the same Python language as their bigger counterparts. That's where MicroPython comes in. Originally developed by Damien George and first released in 2014, MicroPython is a Python-compatible programming language developed specifically for microcontrollers. It includes many of the features of mainstream Python, while adding a range of new ones designed to take advantage of the facilities available on Raspberry Pi Pico and other microcontroller boards. If you've programmed with Python before, you'll find MicroPython immediately familiar. If not, don't worry: it's a friendly language to learn!

A MicroPython program, just as with a standard Python program, normally runs top-to-bottom: it goes through each line in turn, running it through the interpreter before moving on to the next, just as if you were typing them line-by-line into the Shell. A program that just runs through a list of instructions line-by-line wouldn't be very clever, though – so MicroPython, just like Python, has its own way of controlling the sequence in which its programs run: *indentation*.

If this is the first time you use Python, please check: https://www.w3schools.com/python/default.asp

1.2 OpenMV

Arduino have made some unusual choices with MicroPython, in particular the use of OpenMV for programming the Arduino boards. This is unusual as OpenMV is normally used with cameras and machine vision, and not for generic microcontroller boards. It does however provide a way to install MicroPython and upload new programs.

Part 2: Exercises

Exercise 0 – SET UP

To work with micropython we need to use a non-Arduino IDE. **Please follow the instructions in this tutorial and configure the Arduino Nano board to be detected by OpenMV**.

The second step in the configuration is to update the Nina firmware in your arduino board to version 1.4.8. **We will need to use the Arduino IDE for this step. https://docs.arduino.cc/tutorials/nano-rp2040-connect/rp2040-upgrading-nina-firmware**. You will see that there are three options offered for the update: (Option 1) over the Arduino Cloud, (Option 2) using the Arduino IDE (https://docs.arduino.cc/software/ide-v2/tutorials/ide-v2-fw-cert-uploader#firmware-updater) or (Option 3) with a firmware loader. We recommend you to use Option 2, but you can also try Option 1. If you choose Option 2, you can go directly to the second link. Follow the instructions to update the firmware (not to upload certificates). Please be aware that to work again with the board in the Arduino IDE after being connected to the OpenMV, you will first have to:

- Go to OpenMV, stop the execution of the program (red button) and close the Open MV
- Open Arduino IDE and double-tap again on the reset-buttonm on the board to enter bootloader

Finally, please complete the block of exercises related to the wifi connection in this tutorial (you will have to change again to OpenMV and doble-tap on the board again). It is necessary to understand how the wifi module works to complete the rest of the exercises.

You can enter the name and password to the available WiFi. If you run into trouble (only eduroam available, need of extra certificates) you can make a hotspot of your own computer. Change the SSID and key according to your hotspot.

You can try all the examples from the WiFi-part of the tutorial. Open the serial terminal of OpenMV to see the results.

Exercise 1 – MACHINE LEARNING MODELS

Next we are going to reuse a machine learning model to start developing our music detector. **Please follow this tutorial and modify the code so that instead of detecting vibration, the Arduino detects: no movement, slow dance and fast dance. The strings to be printed should be "dance|NO_DANCING", "dance|SLOW_DANCING" and "dance|FAST_DANCING".** Before starting with the tutorial, download the ML libraries and store them into the Arduino flash memory (the disk that opens when doube-tapping, maybe called "NO NAME":

- lsm6dsox.py
- lsm6dsox_vibration_monitoring.ucf

Warnings: if you copy-paste the example, you may get an "ENOENT" error. Check that the editor has your code and not the library open. Go to the window of your code and remove the last trailing end-of-line. Run the example again.