
UNIVERSITÀ
POLITECNICA
DELLE MARCHE

FACOLTÀ DI INGEGNERIA

*Corso di Laurea in Ingegneria Elettronica*

# Studio e sviluppo di funzionalità di sensing ambientale su piattaforma Edge Computing

# Study and development of Edge Computing-oriented ambient sensing functionalities

*Tesi di laurea di:*

**LUCA LIBERATORE**

*Relatore:*

Prof.ssa **SUSANNA SPINSANTE**

*Correlatore*

Prof.ssa **STEFANIA CECCHI**

---

*Anno Accademico 2020-2021*

# Abstract

La tesi verte sullo studio e lo sviluppo di una piattaforma, quale Arduino NANO 33 BLE Sense, per la raccolta e l'analisi di parametri ambientali. In particolare, si focalizza l'attenzione sulle modalità di implementazione e funzionamento di una rete neurale che possa operare a bordo della Scheda, secondo il paradigma Edge Computing, per riconoscere opportuni comandi vocali e restituire in risposta i dati acquisiti dai sensori. A seguito delle prove sperimentali condotte, le capacità di interpretazione e risposta si mostrano notevolmente ridotte a seguito di un aumento del carico di lavoro. Considerando una singola *keyword*, alla cui giusta interpretazione segue un prestabilito compito da svolgere, la Board si mostra, per la maggior parte dei casi, capace di svolgere correttamente le proprie funzionalità. Nella particolare situazione in cui le parole chiave da interpretare siano due e quindi, i compiti da svolgere e i sensori da interrogare siano diversi, il comportamento complessivo subisce un notevole, se non totale, degrado delle prestazioni. Le performance sono ottimali quando, invece di assumere come comando di input la parola acquisita dal microfono PDM integrato, si fa riferimento ad un gesto rilevato all'accelerometro a bordo della Scheda, essendo un sensore in grado di riconoscere semplici movimenti traslatori della Scheda stessa. Ulteriori miglioramenti, per quanto concerne lo sfruttamento più efficiente delle potenzialità dell'Arduino, si potrebbero ottenere cercando di arginare i limiti strutturali di progettazione del device stesso, in modo da riuscire a implementare un modello di Machine Learning su dispositivi embedded che sia il più fluido e performante possibile.

# Indice

# Introduzione

L'obiettivo della presente tesi confluisce in un approccio pragmatico alla gestione e all'utilizzo di periferiche elettroniche, per l'analisi e la visualizzazione di informazioni digitali inerenti parametri ambientali. Avvalendosi della metodologia dell'Edge computing e sfruttando gli strumenti messi a disposizione da dispositivi dalle ridotte dimensioni fisiche, come trattato nel Capitolo 1, l'utilizzatore è in grado di interfacciarsi nel modo più prossimo alla valutazione delle grandezze misurabili. A tal proposito si è scelta come piattaforma hardware l'Arduino NANO 33 BLE Sense, le cui caratteristiche principali sono presentate nel Capitolo 2. Nel successivo Capitolo vengono valutate le caratteristiche di precisione e calibrazione dei sensori, al fine di ottenere un quadro generale delle capacità di performance della Scheda stessa. Disponendo di un modulo Bluetooth tipo Low Energy, nel Capitolo 4 vengono illustrate le caratteristiche di trasferimento dati tramite l'utilizzo del suddetto protocollo. Il capitolo finale, che è il punto di arrivo della tesi, mira a sfruttare le capacità di un sistema Machine Learning integrato (embedded) sulla board Arduino, con lo scopo di valutare e interpretare correttamente i comandi vocali impartiti da un utente, al fine di rispondere correttamente alle richieste ricevute.

# Capitolo 1. Edge Computing

## 1.1 Introduzione all'Edge Computing

La diffusione sempre maggiore dell'Internet of Things (IoT) ha portato all'introduzione di un nuovo paradigma per l'elaborazione dati: l'Edge Computing, la cui dissonanza principale, rispetto a sistemi preesistenti, si esplicita nella metodologia di gestione ed elaborazione dei dati collezionati. Le informazioni raccolte dai dispositivi elettronici vengono direttamente processate all'estremità della rete cui fanno parte, evitando di delegare l'onere computazionale a componenti che non rappresentino la fonte dei dati stessi. In questo è insita la differenza fondamentale tra il sopra citato Edge Computing e, il più anziano, Cloud Computing; infatti, nel secondo l'elaborazione dei dati avviene esternamente ai dispositivi che li producono, come si verifica nel particolare caso dei moderni dispositivi di interazione vocale per le abitazioni. La tipologia di approccio del Cloud Computing, da un lato permette di avere a disposizione una potenza di calcolo più elevata, ma dall'altro evidenzia anche il suo principale svantaggio, ovvero la disponibilità di banda che, in talune circostanze, può compromettere l'efficienza e la prontezza delle operazioni richieste, a causa dell'enorme flusso di dati da condividere. Di conseguenza, si osserva una graduale sostituzione del Cloud Computing con l'Edge Computing, il quale, oltre che permettere una risposta più veloce alle richieste ricevute, garantisce una maggior riservatezza delle informazioni, in quanto circoscritte nella struttura dei dispositivi stessi. Lo scopo della trattazione è raccogliere e analizzare le informazioni tramite dispositivi elettronici in grado di svolgere funzionalità di sensing ambientale, secondo le modalità dell'Edge Computing, e rispondere prontamente alle richieste di interazione ed elaborazione da parte di un utente.

## 1.2 Smart Sensor

I protagonisti dell'Edge Computing sono i sensori, i quali svolgono il ruolo di generare una grande varietà di dati sotto forma di misurazione di grandezze dell'ambiente circostante, come ad esempio temperatura, pressione, quantità di luce ed altri. Vengono definiti smart sensors i sensori in grado di raccogliere queste informazioni, elaborarle e renderle disponibili come output in modo da essere utilizzate da un soggetto terzo. La combinazione di componenti hardware e software permette l'elaborazione, lo storage e l'invio delle misure effettuate. Gli smart sensors solitamente includono molteplici funzioni e componenti, come: trasduzione, sensing, microcontrollore (MCU) con un convertitore analogico-digitale (ADC) integrato, interfaccia input/output (I/0) e alcune funzionalità interne come la valutazione dell'integrità strutturale del sensore stesso. Molto frequentemente gli smart sensors vedono i propri componenti integrati nello stesso circuito della scheda (PCB), garantendo un costo minore della produzione e un miglioramento dell'affidabilità e delle performance. In alcuni casi però questa caratteristica compromette la validità dei dati raccolti, come nella particolare situazione di misurazioni di temperatura, dove il surriscaldamento delle componenti influenza negativamente il sensore di temperatura. La Figura 1.1 sottostante descrive la struttura classica di uno smart sensor, dove compaiono un display e un'interfaccia di comunicazione, che può essere wireless o cablata.

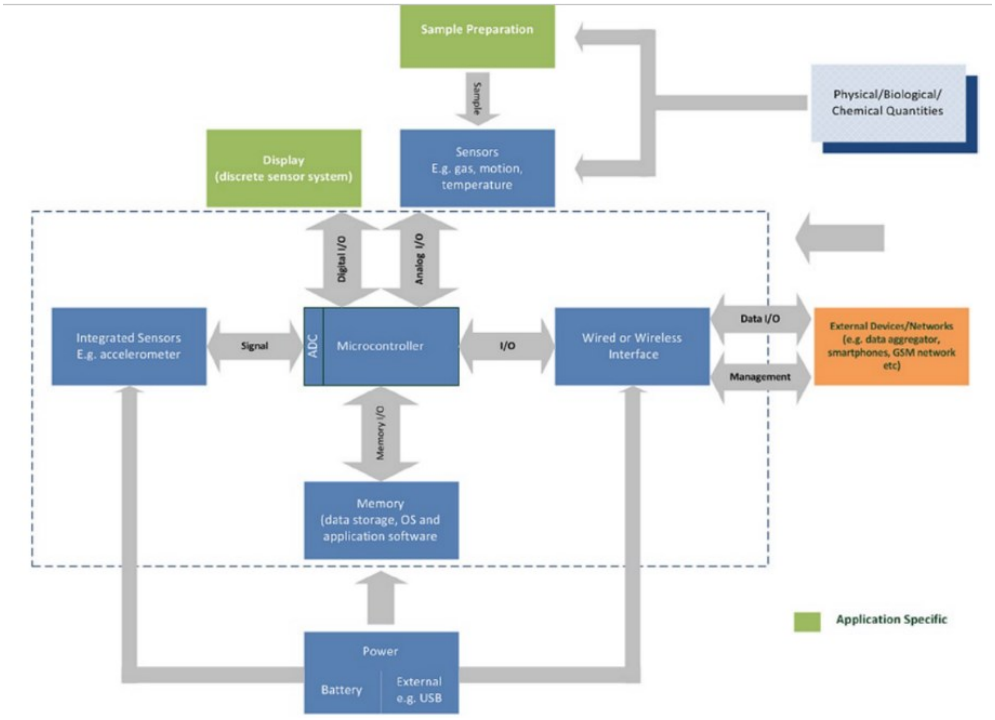



*Figura 1.1 Smart sensor*

Le sensor platforms sono un sottoinsieme degli smart sensors, includono un microcontrollore, un'interfaccia di comunicazione, una memoria e sensori che possono essere esterni o integrati. Queste schede sono molto utilizzate per la loro versatilità, garantita dalla possibilità di essere testate e riprogrammate con grande facilità tramite un'ambiente di sviluppo integrato (IDE). L'aspetto negativo è il costo, maggiore di quello degli smart sensors, e lo spreco di risorse, causato dalla natura non specifica di questa tipologia di schede che lascia inutilizzate alcune caratteristiche proprie nel caso di un'applicazione pratica particolare.

Le sensor platforms sono molto diffuse, tra le più famose si annoverano Arduino, Shimmer e i vari smartphones. Il progetto di Arduino fu sviluppato in Italia nel 2005, per poter garantire, anche ai non addetti ai lavori, la creazione di semplici prototipi avvalendosi dell'omonimo IDE e del proprio linguaggio di programmazione open source: un misto tra il C e l'Assembly. La Board più famosa della famiglia Arduino è Arduino UNO (in Figura 1.2). Anche smartphones e tablet includono una grande varietà di sensori che permettono di migliorare la qualità dell'esperienza utente nel loro utilizzo, i più noti sono accelerometro, giroscopio e sensore di luce ambientale.

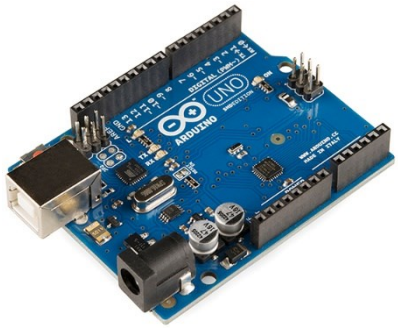



*Figura 1.2 Arduino UNO*

Prima di procedere oltre, è necessario esplicitare l'importante differenza concettuale tra **microprocessore**, **microcomputer** e **microcontrollore**, spesso confusi tra loro. Un microprocessore (CPU) è l'unità centrale di elaborazione, implementata su un singolo chip; mentre un microcomputer è un dispositivo provvisto sia di CPU che di una interfaccia I/O, memoria e timer. In particolare, una CPU si divide in cinque parti:

- Unità Logica ed Aritmetica (ALU), responsabile delle operazioni di calcolo sui dati;
- Unità di Controllo (CU), controlla la transazione di istruzioni da e verso il processore;

- Registro, memoria veloce (RAM) su interrogazione della CPU;
- Bus, un collegamento tra processore memoria e periferiche;
- Memoria, dove la quantità di dati viene salvata.

Un microcontrollore (MCU) è un dispositivo elettronico integrato su un circuito elettrico, nato come alternativa al microprocessore, è utilizzato in sistemi embedded e presenta diversi pin di ingresso e uscita, attraverso i quali è possibile interagire con il mondo esterno. I microcontrollori in genere si suddividono in famiglie che hanno in comune la CPU, ma diverse configurazioni input-output. Una volta individuata la CPU di interesse, si procede a cercare l'elemento della famiglia che meglio soddisfa le altre esigenze, come la memoria ad esempio. Vi sono tre tipologie di memoria contenute all'interno del MCU:

- RAM (Random Access Memory), in cui sono salvati dati che vengono modificati durante l'esecuzione del programma. I dati in essa contenuti vengono persi definitivamente una volta che il dispositivo viene spento.
- ROM (Read Only Memory), rappresenta la memoria non volatile, ovvero i dati salvati non vengono persi allo spegnimento del device.
- EEPROM (Electrically Erasable Programmable Read-Only Memory), è una memoria non volatile dove i dati vengono salvati e cancellati elettricamente.

Solitamente i microcontrollori sono dotati di oscillatori al quarzo utilizzati per generare precisi impulsi nel tempo, con lo scopo di ottenere un calcolo ottimale del tempo trascorso tra due eventi attraverso l'ausilio di un timer. Legato al concetto di timer vi è l'interrupt, cioè una chiamata verso l'MCU a seguito di un particolare evento, come può essere l'avvenimento di un overflow nel conteggio del tempo da parte del timer. Il watchdog timer opera separatamente e garantisce che il programma non giri a vuoto all'infinito.

Nei dispositivi embedded, il sistema di comunicazione principale per interfacciarsi con periferiche esterne è la GPIO (General Purpose input-output), un insieme di pin digitali o analogici che possono essere programmati come input o come output, come accade per l'accensione di un LED.

# 1.3 Protocolli di Comunicazione

I protocolli di comunicazione più usati sono tre, basati tutti su una comunicazione di tipo seriale e sono: **UART**, **I2C** e **SPI**. I collegamenti sono illustrati nella Figura 1.3 sottostante.

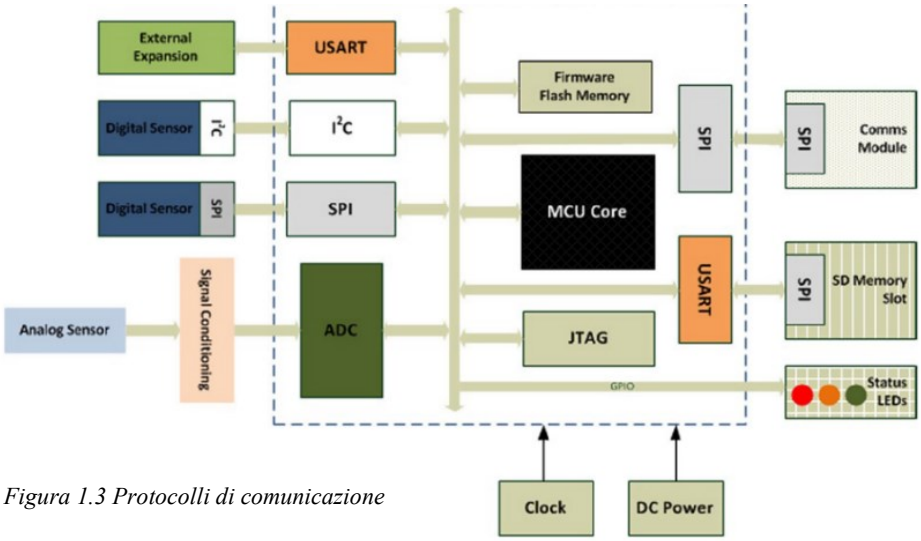



*Figura 1.3 Protocolli di comunicazione*

La comunicazione seriale è un protocollo di comunicazione utilizzato nelle telecomunicazioni che prevede l'invio di un bit alla volta in ordine sequenziale lungo il canale di trasmissione, diversamente dalla comunicazione parallela, di più difficile implementazione, in cui più bit vengono trasmessi contemporaneamente sul bus.

L'**UART** (Universal Asynchronous Receiver/Transmitter) supporta la trasmissione seriale bidirezionale, in modo asincrono, ovvero non dipende da un segnale di clock sincronizzato tra due dispositivi, ma da un bit di inizio (1) e uno di fine (0). La comunicazione deve avvenire a un preciso baud rate (rate trasmissione dei simboli) concorde tra i due dispositivi, solitamente di default posto a 115200. La trasmissione avviene in tre modalità differenti: simplex, i dati vengono trasmessi in una direzione sola; half-duplex, dati trasmessi in entrambe le direzioni ma non in contemporanea; full-duplex, i dati possono venire trasmessi e ricevuti nello stesso momento.

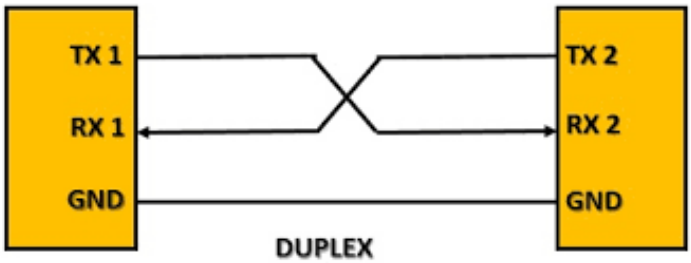



*Figura 1.4 Schema UART*

Nel collegamento fisico si utilizzano due linee di comunicazione: TX per la trasmissione, RX per la ricezione e una linea di terra (GND). I vantaggi di questo tipo di comunicazione sono la semplicità di operazione e implementazione, uniti alla non necessità di un clock per la sincronizzazione, a discapito di un basso rate di trasmissione. La velocità di trasmissione può essere notevolmente migliorata con il protocollo **USART**, che prevede l'ausilio di un clock per la sincronizzazione, che permette di evitare la necessità di specificare un preciso baud rate di comunicazione. Le velocità possono raggiungere i 4 Mbps.

La comunicazione **I2C** (Inter Integrated Circuit) è un bus seriale bidirezionale sincrono con due collegamenti fisici (fili), in cui vi sono almeno un master e uno o più slave. Le due linee di comunicazione sono l'SDA (serial data line acceptance port) e l'SCL (serial clock line), la prima per la trasmissione e la ricezione dei dati, la seconda per la sincronizzazione temporale, a cui va aggiunto un collegamento a terra (GND) e l'alimentazione (Vdd). Il dispositivo che funge da master inizia e termina la conversazione stabilendo il clock, tutti gli altri dispositivi sono gli slaves, i quali vengono comandati a ricevere o trasmettere i dati secondo le disposizioni del master.

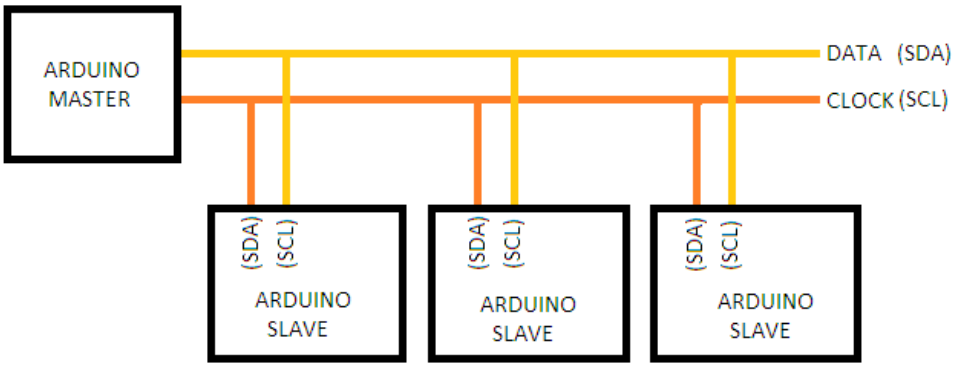



*Figura 1.5 Collegamento Master-Slave*

La flessibilità che offre la modalità di trasmissione I2C e la capacità di stabilire una comunicazione tra più dispositivi tramite solo 2 fili, garantiscono un importante pregio di questo protocollo, che viene meno quando si giunge ad un elevato numero di dispositivi.

In ultima analisi, vi è il protocollo **SPI** (Serial Peripheral Interface), cioè un sistema di comunicazione seriale sincrono e full-duplex, tra un microcontrollore e un circuito integrato, o tra più microcontrollori. È il protocollo più veloce tra quelli analizzati, permette velocità fino a 100 Mbps; le linee di clock e di dati vengono condivise tra più

dispositivi, i cui indirizzi di identificazione sono univoci. La comunicazione avviene tramite quattro porte:

- MOSI (Master Data Output, Slave Data Input);
- MISO (master data input, slave data output);
- SCLK (clock signale) dipende dal master;
- NSS (Slave enabled signal) segnale emesso dal master al fine di scegliere lo slave con cui comunicare.

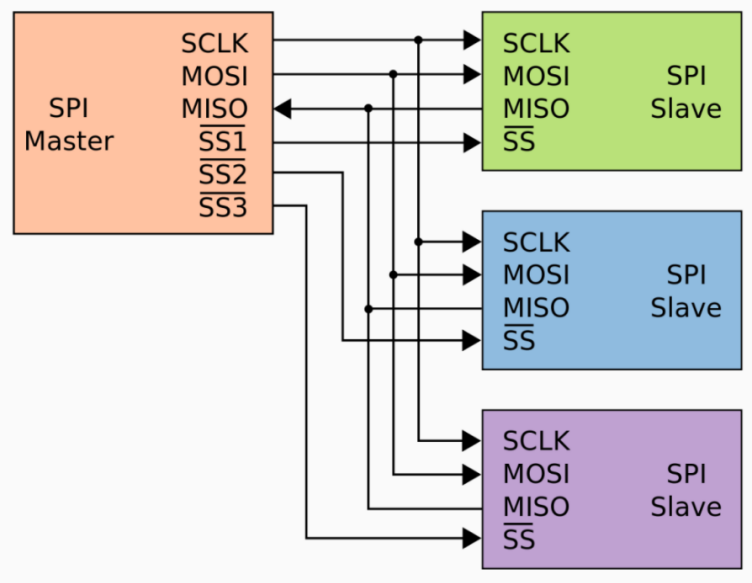



*Figura 1.6 Protocollo SPI*

La trasmisssione dei dati sul bus si basa su un registro a scorrimento (shift register), di dimensione solitamente a 8 bit di cui sono dotati i singoli dispositivi, che siano master o slave. Lo schema di collegamento è rappresentato in Figura 1.6. Il vantago dell'SPI rispetto all'I2C risiede nella non necessità di implementare un indirizzo ogni qual volta si voglia comunicare con uno slave, rispetto all'UART invece, non prevede bit di inizio e fine conversazione, permettendo una trasmissione delle informazioni continuativa senza interruzioni. Inoltre, è garantita la capacità di trasmettere e ricevere i dati nello stesso momento, avendo separati i collegamenti MISO e MOSI.

Gli svantaggi, confrontando con l'I2C e l'UART, sono rispettivamente: l'assenza di un meccanismo che testimoni la ricezione dei dati per l'uno, e la mancanza di sistemi di rilevamento d'errore per l'altro (bit di parità), oltre un numero maggiore di collegamenti fisici previsti dal protocollo.

La caratteristica fondamentale di un sensore è la capacità di comunicare ad un altro dispositivo o ad un utente il risultato della misurazione effettuata. La trasmissione dei dati può avvenire, tramite collegamento fisico con le tecniche viste sopra, oppure tramite un display o via wireless. Una delle comunicazioni fisiche più conosciute è quella **USB**, che comprende quattro versioni in base alla velocità (da 1.0 fino a 4.0), permettendo a un singolo host USB di collegare fino a 127 device. Una volta che un device è collegato ad un host USB, quest'ultimo legge il rate del dispositivo tra le informazioni e, se supportato, instaura una comunicazione che può basarsi anche su sub-functions, come può essere una webcam con funzioni di video e audio. È diffuso l'utilizzo della comunicazione USB per connettere sensori o schede integrate con sensori al PC, per acquisizione di dati, programmazione e alimentazione, come avviene per la serie Arduino.

# Capitolo 2. Arduino

## 2.1 Arduino Nano 33 BLE Sense

La scheda Arduino nano 33 BLE sense (Figura 2.1) è una Board evoluzione dell'Arduino Nano, dalle dimensioni minime (45 mm x 18 mm), ma con un microcontrollore più performante: l'nRF52840, basato sull'architettura Arm® Mbed™ OS. E' caratterizzata da una memoria di programma (flash memory) trentadue volte più potente dell'Arduino Uno, che permette di gestire programmi, detti "sketch", più lunghi e con più variabili, avendo anche una memoria RAM di 256KB (128 volte più grande). Il vantaggio principale della scheda è la capacità di eseguire applicazioni per l'Edge Computing, grazie alla grande quantità di sensori integrati quali:

- IMU LSM9DS1, giroscopio accelerometro e magnetometro;
- Microfono MP34DT05;
- Sensore gesti, prossimità e colori APDS9960;
- Barometro LPS22HB;
- Sensore di temperatura e umidità HTS221;
- NINA-B3 series, BLE (bluetooth low energy).

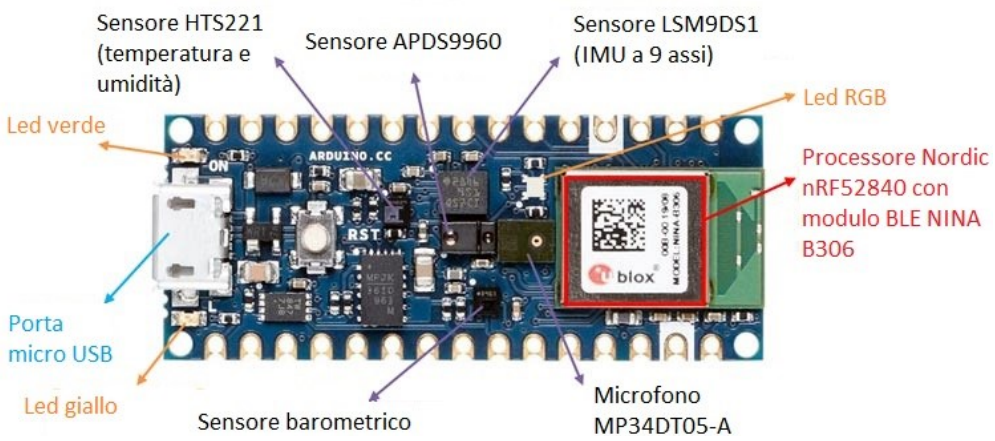



*Figura 2.1 Arduino Nano 33 BLE Sense*

La memoria RAM è volatile, diversamente dalla flash memory e dalla EEPROM (non presente nel BLE 33 sense). La EEPROM è quel tipo di memoria riservata ai programmatori per salvare informazioni a lungo termine all'interno della scheda. Tale spazio di archiviazione è da utilizzare con cautela perché limitato e soggetto ad un numero finito di scritture. Il connettore micro-USB e la tensione di lavoro a 3.3 V (non più 5V come nell'Arduino classico) sono una novità di questo modello, mentre il Bluetooth Low Energy permette di sfruttare la scheda come client o server. La Board è dotata di un led integrato, il numero 13, di quattordici pin ingresso/uscita digitali e otto pin analogici, una connessione per ogni tipologia di trasmissione (UART, SPI, I2C) e, per finire, pin di ingresso per l'alimentazione e collegamenti a terra (GND), come mostrato in Figura 2.2.

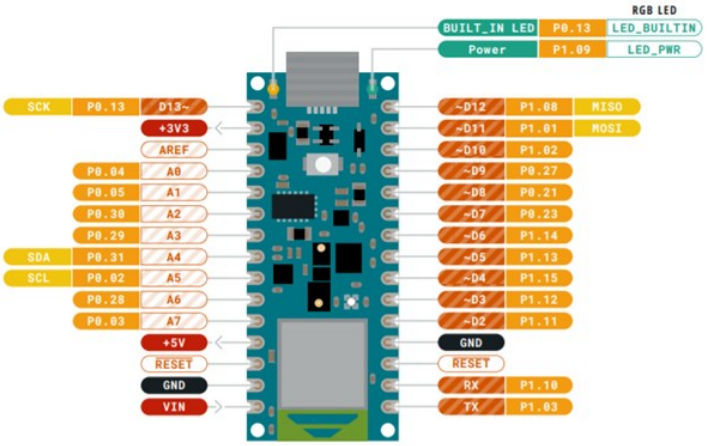



*Figura 2.2 Schema dei PIN*

Per la programmazione della scheda si utilizza l'Arduino Desktop IDE, il quale consente una scrittura del codice offline successivamente implementato nella scheda tramite il cavo micro USB, che funge anche da alimentatore. Una volta collegato l'Arduino, bisogna selezionare la scheda dal menù dell'IDE e installare i driver. Si procede quindi a conFigurare la porta dove è collegato, tramite la sezione "tools" dal menù. Le librerie, per il corretto funzionamento dei sensori, vanno importate selezionando "sketch"→ "importa librerie" e cercando il nome della libreria da scaricare. Lo sketch Arduino è composto essenzialmente da due insiemi che racchiudono gran parte del codice:

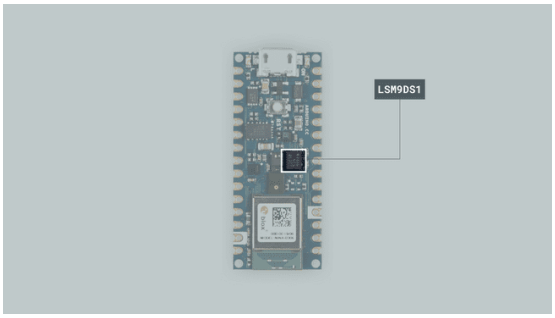
- **setup()** utilizzato per inizializzare variabili, come impostare lo stato dei pin, e per il setting delle comunicazioni, come ad esempio le seriali;
- **loop()** rappresenta le istruzioni principali del programma ripetute in modo ciclico.

Le librerie vanno incluse prima del "setup", tramite il comando *"#include"* seguito dal nome della libreria.

## 2.2 Sketch di base

### 2.2.1 IMU

L'IMU, acronimo per "inertial measurement unit", è un LSM9DS1, costruita da ST Microelectronics e fornita di accelerometro, giroscopio e magnetometro, a cui corrisponde la libreria ArduinoLSM9DS1, che contiene gli esempi per un utilizzo di base dei sensori. L'IMU è connessa al microcontrollore tramite il protocollo I2C e ritorna valori in *float*, ovvero numeri con virgola che occupano 4 byte di spazio di memoria, diversamente dal tipo *int,* che gestisce variabili di numeri interi (2 byte). Una volta inclusa la libreria, i sensori vengono inizializzati ai valori impostati di fabbrica. La posizione sull'Arduino è indicata nella Figura 2.3.




- **Accelerometer** range is set at [-4, +4]g -/+0.122 mg.
- **Gyroscope** range is set at [-2000, +2000] dps +/-70 mdps.
- **Magnetometer** range is set at [-400, +400] uT +/-0.014 uT.
- **Accelerometer** Output data rate is fixed at 104 Hz.
- **Gyroscope** Output data rate is fixed at 104 Hz.
- **Magnetometer** Output data rate is fixed at 20 Hz.

*Figura 2.3 IMU*

L'**accelerometro**, la cui orientazione degli assi è mostra in Figura 2.4, è uno strumento elettromeccanico che ha lo scopo di misurare le forze di accelerazione, che siano statiche, come nel caso della gravità, o dinamiche, come durante il movimento della scheda. Ci sono due modalità di funzionamento per l'accelerometro e il giroscopio: solamente l'accelerometro attivo, o entrambi i sensori attivi. Per passare da una modalità all'altra è necessaria un'operazione di scrittura al registro CTRL_REG6_XL (20h) per la prima, mentre sul registro CTRL_REG1_G (10h) per la seconda. La sensibilità nella misurazione dell'accelerazione lineare può essere determinata, per esempio, applicando un'accelerazione di 1 g al dispositivo, cioè puntando l'asse di misurazione verso terra, per poi girare il sensore di 180 gradi, sottrarre al valore più grande quello più piccolo e dividere per due. Il sensore, in una posizione orizzontale di stabilità, dovrebbe misurare 0 g in entrambi gli assi X e Y, mentre 1 g nella direzione dell'asse Z; una deviazione dal valore ideale è chiamata zero-g offset, il quale può variare anche in base alla temperatura.

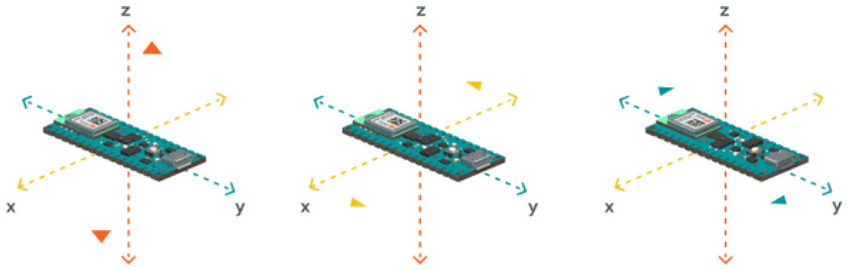
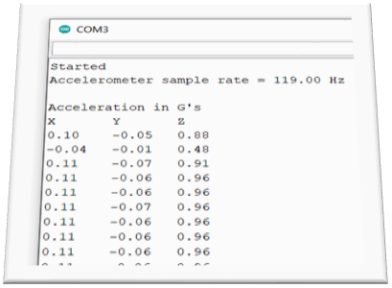



*Figura 2.4 Accelerometro*

Di seguito uno sketch esempio in cui viene interrogato ripetutamente il sensore, che ritorna a schermo, tramite il monitor seriale inizializzato a 9600 baud, il valore dell'accelerazione in G per ogni asse.

```
#include <Arduino_LSM9DS1.h>

void setup() {
  Serial.begin(9600);
  while (!Serial); //aspetta il collegamento seriale
  Serial.println("Started");

  if (!IMU.begin()) {  //ritorna 1 se avviata con successo, altrimenti 0
    Serial.println("Failed to initialize IMU!");
    while (1);
  }

  Serial.print("Accelerometer sample rate = ");
  Serial.print(IMU.accelerationSampleRate());  //ritorna la frequenza di campionamento
  Serial.println(" Hz");
  Serial.println();
  Serial.println("Acceleration in G's");
  Serial.println("X\tY\tZ");
}

void loop() {
  float x, y, z;

  if (IMU.accelerationAvailable()) {  //domanda se un un nuovo campione di accelerazione è
                                      //disponibile, ritorna 1 se si, 0 altrimenti
    IMU.readAcceleration(x, y, z);  //intorroga l'IMU e ritorna l'accelerazione in g
                                    //salva i valori nelle variabili float x y z
    Serial.print(x);                //che rappresentano gli assi e li invia in seriale
    Serial.print('\t');
    Serial.print(y);
    Serial.print('\t');
    Serial.println(z);
  }
}
```



Il **giroscopio** (Figura 2.5) misura l'orientamento e la velocità angolare di un oggetto. È un dispositivo più avanzato dell'accelerometro, in quanto può misurare l'inclinazione e l'orientamento laterale di un oggetto, mentre un accelerometro può misurare solo il suo movimento lineare. I sensori giroscopici sono anche chiamati "Sensori di velocità angolare". Misurata in gradi al secondo, la velocità angolare è la variazione dell'angolo di rotazione dell'oggetto per unità di tempo. Nel modulo LSM9DS1, il giroscopio può operare in tre modalità differenti: power-down, low-power and normal mode. La sensibilità, anche in questo caso, può essere leggermente influenzata dalla temperatura.

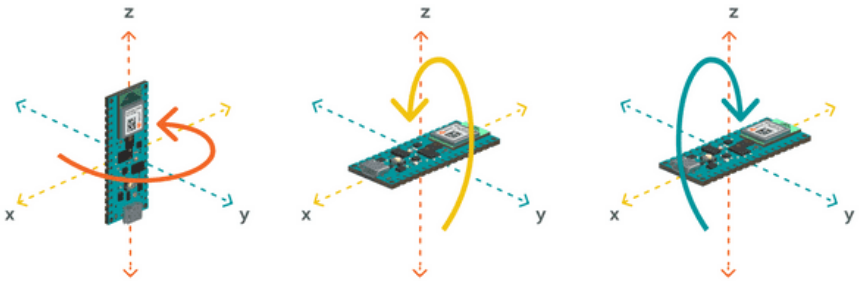



*Figura 2.5 Giroscopio*

L'esempio classico, di utilizzo e lettura tramite seriale, del giroscopio si presenta come segue.

```
#include <Arduino_LSM9DS1.h>

void setup() {
  Serial.begin(9600);
  while (!Serial);
  Serial.println("Started");

  if (!IMU.begin()) {
    Serial.println("Failed to initialize IMU!");
    while (1);
  }
  Serial.print("Gyroscope sample rate = ");
  Serial.print(IMU.gyroscopeSampleRate());  //Returns the gyroscope sample rate
  Serial.println(" Hz");
  Serial.println();
  Serial.println("Gyroscope in degrees/second");
  Serial.println("X\tY\tZ");
}

void loop() {
  float x, y, z;

  if (IMU.gyroscopeAvailable()) {  //Query if new gyroscope data sample is available
                                   //if no new gyroscope data sample is available
                                  //1 if new gyroscope data sample is available
    IMU.readGyroscope(x, y, z); //Query the IMU gyroscope sensor and
                                //returns the rotational speed in dps (degrees per second)
    Serial.print(x);           //x,y,z - the float variables where the gyroscope values will be stored
    Serial.print('\t');       //1 on success, 0 on failure
    Serial.print(y);
    Serial.print('\t');
    Serial.println(z);
  }
}
```

Il **magnetometro** è un dispositivo che misura il campo magnetico: la sua direzione, forza e il cambiamento relativo in una particolare posizione (Figura 2.6). La misura delle componenti del campo, lungo tre direzioni indipendenti, permette di definire unicamente il vettore campo magnetico nel punto in cui si effettua la misura. Il sensore magnetico ha tre modalità di funzionamento: power-down (default), continuous-conversion mode e single-conversion mode; per passare dal power-down alle altre modalità bisogna scrivere sul registro CTRL_REG3_M (22h), settando il bit MD. Lo Zero-gauss level offset rappresenta la deviazione dal reale valore di campo magnetico se nessun campo è presente. La misurazione può essere leggermente influenzata da temperatura, stress meccanico (del circuito in cui è il sensore integrato) e dal tempo.

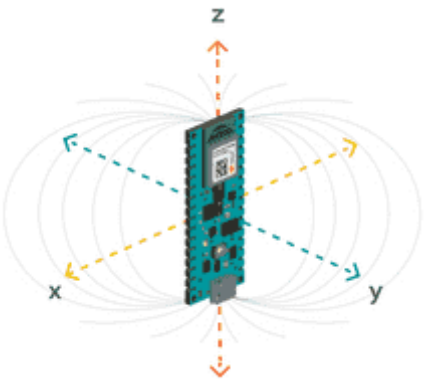



*Figura 2.6 Magnetometro*

Nella figura sottostante l'esempio di codice.

```
#include <Arduino_LSM9DS1.h>

void setup() {
  Serial.begin(9600);
  while (!Serial);
  Serial.println("Started");

  if (!IMU.begin()) {
    Serial.println("Failed to initialize IMU!");
    while (1);
  }
  Serial.print("Magnetic field sample rate = ");
  Serial.print(IMU.magneticFieldSampleRate());
  Serial.println(" uT");
  Serial.println();
  Serial.println("Magnetic Field in uT");
  Serial.println("X\tY\tZ");
}

void loop() {
  float x, y, z;

  if (IMU.magneticFieldAvailable()) { //Query if new magnetic field data sample is available
                                      //return 1 if new magnetic fielddata sample is available
    IMU.readMagneticField(x, y, z); //Query the magnetometer and returns the magnetic field measured in uT
                                    //x,y,z - the float variables where the magnetic field values will be stored
    Serial.print(x);                //return 1 on success, 0 on failure
    Serial.print('\t');
    Serial.print(y);
    Serial.print('\t');
    Serial.println(z);
  }
}
```

## 2.2.2 Microfono PDM

Il microfono digitale è un MP34DT05, fabbricato dalla ST Microelectronics, low-power e omnidirezionale, che produce un'uscita rappresentata con la modulazione digitale PDM (Pulse-density modulation). Il microfono integrato consente di catturare e analizzare il suono in tempo reale e può anche essere utilizzato per creare un interfaccia ad interazione vocale per un progetto (da includere nel codice tramite la propria libreria PDM.h). L'MP34DT05 è un microfono digitale a bassa distorsione, dotato di un rapporto segnale rumore di 64 dB e capace di operare in un intervallo di temperatura che va dai -40 gradi Celsius ai +85 gradi Celsius. È collocato nella parte bassa dell'Arduino (Figura 2.7)

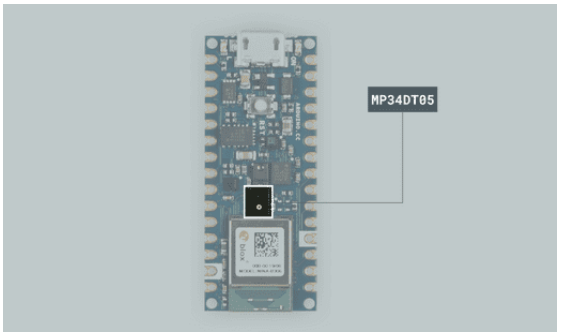



*Figura 2.7Microfono PDM*

La modulazione PDM è una forma generalizzata della PWM (pulse-width modulation) per applicazioni audio. Si tratta di una tecnica che permette di trasmettere l'informazione associata ad un segnale dato, detto segnale modulante. Nella trasmissione tramite modulazione PWM si usa come supporto un segnale a onda quadra (Figura 2.8), di frequenza e ampiezza fissate, nel quale la larghezza, ovvero la durata degli impulsi, è proporzionale al livello del segnale modulante. La durata di ciascun impulso può essere espressa con il concetto di duty cicle, spesso espresso in percentuale, che rappresenta il rapporto tra il tempo in cui il segnale è a valore alto e la durata del segnale. In ogni caso l'informazione risiede nella durata degli impulsi, che hanno tutti la stessa ampiezza.

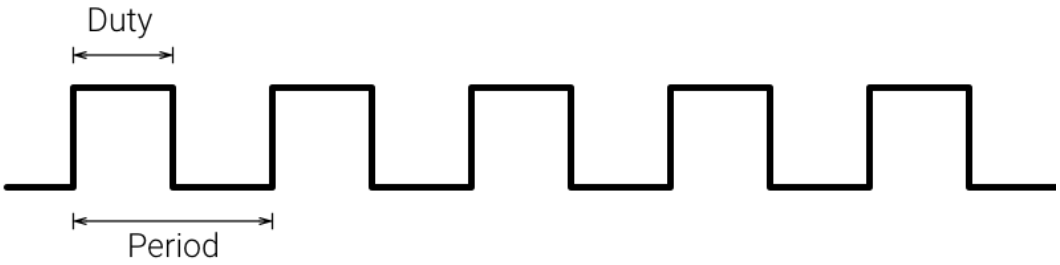



*Figura 2.8 Onda quadra*

Nel codice sottostante, a seguito di uno schiocco di dita viene quantificata, tramite seriale, la quantità di luce ambientale presente nell'ambiente.

```
#include <Arduino_APDS9960.h>
#include <PDM.h>

short sampleBuffer[512];        // Buffer to read samples into, each sample is 16-bits
volatile int samplesRead;      // Number of audio samples read (volatile -> interrupt)

void setup() {
  Serial.begin(9600);
  while (!Serial);

  if (!APDS.begin()) {
  Serial.println("Errore APDS9960");
  }

  PDM.onReceive(onPDMdata);                       // Configure the data receive callback (interrupt)

  if (!PDM.begin(1, 16000)) {                   //initialize the PDM interface
    Serial.println("Errore PDM");              //channels = 1 for mono, =2 for stereo, frequency 16KHz for nano 33 BLE
    while (1);
  }
}

void loop() {

  if (samplesRead) {                                  // Wait for samples to be read
       for (int i = 0; i < samplesRead; i++) {       //Print samples to the serial monitor or plotter
       //Serial.println(sampleBuffer[i]);
       if(sampleBuffer[i]>1000){
        lum();
       }
    }
     samplesRead = 0;    // Clear the read count
  }
}
void onPDMdata() {
  //Get the number of bytes available for reading from the PDM interface. This is data that has already arrived and was stored in the PDM receive buffer.
  int bytesAvailable = PDM.available();        // Query the number of available bytes
  PDM.read(sampleBuffer, bytesAvailable);     // Read into the sample buffer (= array to store the PDM data into)
  samplesRead = bytesAvailable / 2;          // 16-bit, 2 bytes per sample
}

void lum(){

  while (! APDS.colorAvailable()) {           // check if a color reading is available
    delay(5);
  }
  int r, g, b, a;
  APDS.readColor(r, g, b, a);       // read the color
  Serial.print("luminosità = ");           // print the values
  Serial.println(a);
  delay(1000);

}
```

## 2.2.3 Sensore RGB, Gesti e Prossimità

Il sensore di gesti è un APDS9960 (Figura 2.9), rileva il gesto, il colore, l'illuminazione dell'ambiente e la vicinanza di un oggetto. Viene prodotto dalla Broadcom ed è supportato dalla propria libreria APDS. Le letture dei gesti si basano sul rilevamento del movimento, tramite quattro fotodiodi all'interno del sensore, di una mano o un oggetto. La distanza da un punto viene calcolata tramite la quantità di luce infrarossa riflessa da un ostacolo. Il colore è dato come valore a 16 bit per le componenti R, G, B. L'interfacciamento e il controllo del sensore avvengono tramite un bus I2C a 7-bit, che garantisce accesso alla gestione del sensore e ai dati di output. Lo standard di trasmissione fornisce tre tipo di operazioni: lettura, scrittura e un protocollo combinato.

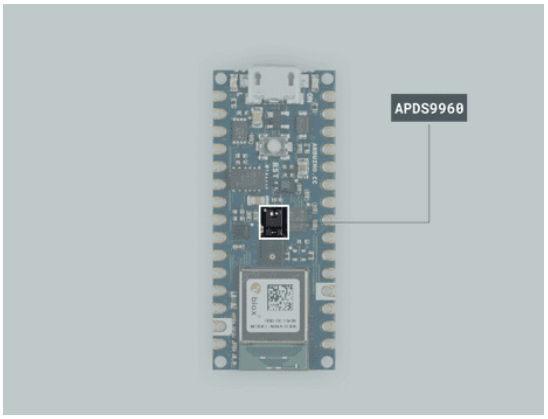



*Figura 2.9 Sensore RGB, Gesti e Prossimità*

La luce bianca è in realtà composta da tutti i colori dell'arcobaleno, infatti contiene tutte le lunghezze d'onda, ed è descritta come luce policromatica. Il sensore di colore è separato in quattro diversi canali: luce rossa, verde, blu e bianca; ognuno di essi ha un filtro di blocco UV e IR e un convertitore di dati raccolti dedicato per leggere tutte le informazioni contemporaneamente, come mostrato in Figura 2.10.

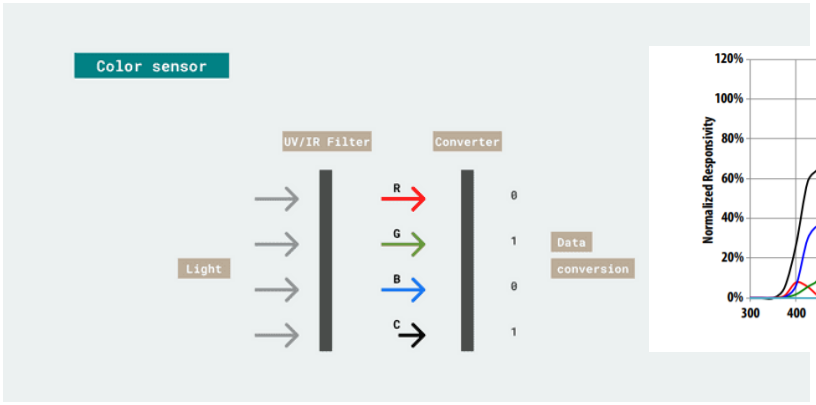



*Figura 2.10 sensore RGB*

Di seguito gli sketch base per l'utilizzo del sensore nelle tre diverse modalità: rilevamento colori, gesti e prossimità. Nelle ultime modalità, oltre ai quattro fotodiodi direzionali, il sensore è provvisto di compensazione di offset e interrupt tramite I2C.

Esempio **RGB** – rilevazione del colore e invio in seriale del risultato

```
#include <Arduino_APDS9960.h>

void setup() {
  Serial.begin(9600);
  while (!Serial);

  if (!APDS.begin()) {  //Initialize the APDS gesture sensor
    Serial.println("Error initializing APDS9960 sensor.");  //Returns 1 on success, 0 on failure
  }
}

void loop() {
  // Check if a color reading is available from the senor.
  while (! APDS.colorAvailable()) {  //This function also enables the color sensor when called for the first time
    delay(5);  //Returns 1 if a color has been read, otherwise 0
  }

  int r, g, b;

  APDS.readColor(r, g, b);  // read the color from the sensor
   //This function requires 3 or 4 integer variables as arguments where the read color will be stored
  // APDS.readColor(r, g, b, a); //a - is the ambient light intensity
  Serial.print("r = ");
  Serial.println(r);
  Serial.print("g = ");
  Serial.println(g);
  Serial.print("b = ");
  Serial.println(b);
  Serial.println();
  delay(1000);
}
```

Spesso in un programma c'è la necessità di confrontare il valore di una variabile con una serie di valori fino a trovare quello corrispondente. Nel linguaggio di programmazione per Arduino esiste il meccanismo switch/case che controlla il flusso del programma, permettendo di specificare codice differente che dovrebbe essere eseguito a seconda della condizione riscontrata. In particolare, *switch* confronta il valore della variabile al valore specificato dalla clausola *case*. Questa metodologia di approccio è presente nel prossimo esempio, che riguarda il sensore gesti per la rilevazione di un movimento della mano verso l'alto, basso, sinistra o destra.

```
Sintassi:
switch(var){
        case label1:
                //istruzioni
                break;
        case label2:
                //istruzioni
                break;
        default:
                //istruzioni
}
```

Esempio **SENSORE GESTI**

```
#include <Arduino_APDS9960.h>

void setup() {
  Serial.begin(9600);
  while (!Serial);
  if (!APDS.begin()) { //Initialize the APDS gesture sensor returns 1 on success, 0 on failure
    Serial.println("Error initializing APDS9960 sensor!");
  }
    Serial.println("Detecting gestures ...");
}
void loop() {
  if (APDS.gestureAvailable()) {//Checks if the sensor has detected gestures
    //Returns 1 if a gesture has been detected, otherwise 0
    int gesture = APDS.readGesture(); //Read the gesture detected from the sensor
    //Returns the detected gesture
    switch (gesture) {
      case GESTURE_UP:
        Serial.println("Detected UP gesture");
        break;

      case GESTURE_DOWN:
        Serial.println("Detected DOWN gesture");
        break;

      case GESTURE_LEFT:
        Serial.println("Detected LEFT gesture");
        break;

      case GESTURE_RIGHT:
        Serial.println("Detected RIGHT gesture");
        break;

      default:
        // ignore
        break;
```

Esempio **SENSORE PROSSIMITA'**

```
#include <Arduino_APDS9960.h>

void setup() {
  Serial.begin(9600);
  while (!Serial);

  if (!APDS.begin()) {
    Serial.println("Error initializing APDS9960 sensor!");
  }
}

void loop() {
  // check if a proximity reading is available
  if (APDS.proximityAvailable()) { //Returns 1 if a proximity has been read, otherwise 0
   //Retrieve the proximity read from the sensor
    int proximity = APDS.readProximity();  // The detected proximity that may range from 0 to 255
   //where 0 is the closest and 255 is the farthest, returns -1 in case of error.
    Serial.println(proximity);
  }
  // wait a bit before reading again
  delay(100);
}
```

## 2.2.4 Barometro

Il barometro è un LPS22HB, un sensore digitale ultracompatto a 24-bit, prodotto dalla ST e supportato dall'omonima libreria Arduino, che restituisce come output, tramite collegamenti I2C e SPI, la pressione atmosferica in kPa. Il sensore (Figura 2.11) è costituito da una membrana di silicone e può operare efficacemente a temperature comprese tra -40 e +85 gradi.

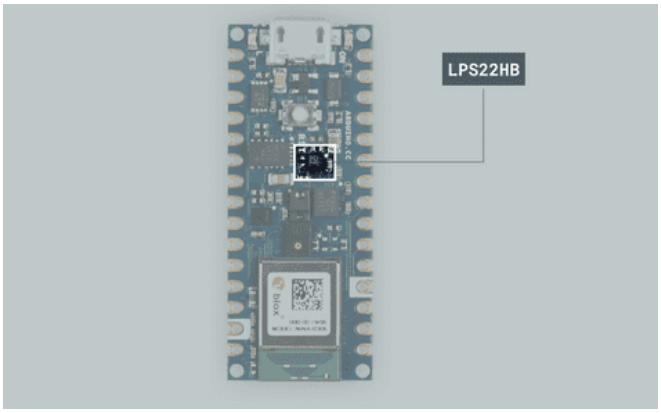



*Figura 2.11 Barometro*

Oltre a essere utilizzato per applicazioni GPS e previsioni meteo a breve termine, questo sensore permette di calcolare, in modo approssimativo, l'altitudine sul livello del mare attraverso la misurazione della pressione atmosferica. Ciò è reso possibile, nei moderni strumenti di misurazione, grazie ad un diaframma resistivo (Figura 2.12) che si deforma a seguito della pressione atmosferica: maggiore è la pressione maggiore sarà il movimento e di conseguenza il valore numerico misurato.

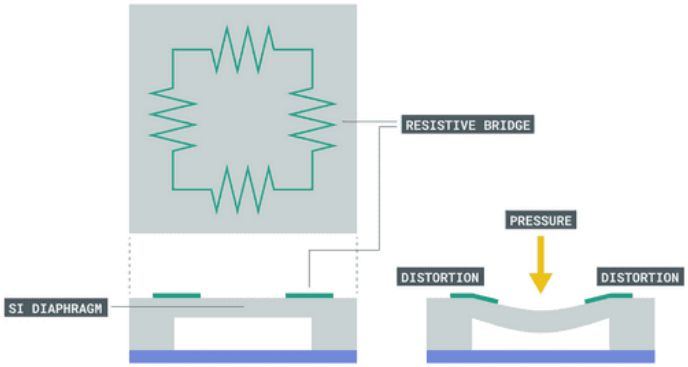



*Figura 2.12 Diaframma resistivo*

Una volta ottenuto il risultato numerico (“P”), in kPa, si utilizza la seguente formula matematica per il calcolo approssimato dell’altitudine (“H”) dell’ambiente in cui è stata fatta la misurazione:

```
H = 44330 * [1 - (P/p0)^(1/5.255) ]
```

dove “p0” è la pressione al livello del mare (101.325 kPa).

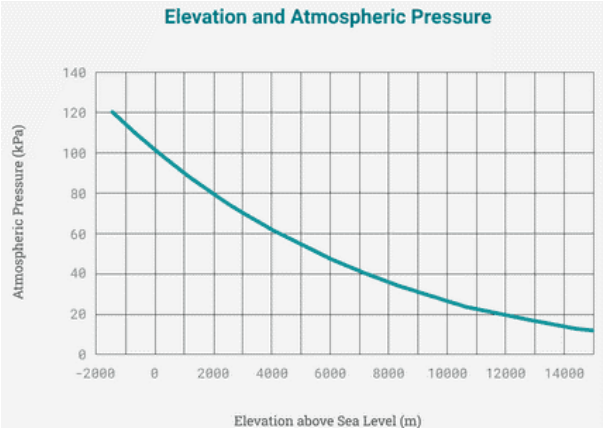



*Figura 2.13 Curva pressione-altitudine*

Nella Figura 2.13 soprastante è rappresentata la curva pressione-altitudine, mentre sotto lo sketch esempio di lettura del sensore, che invia in seriale il valore della pressione in kPa, da visualizzare tramite il monitor seriale.

```
#include <Arduino_LPS22HB.h>

void setup() {
  Serial.begin(9600);
  while (!Serial);
// Initialize the pressure sensor
  if (!BARO.begin()) { // Returns 1 on success, 0 on failure
    Serial.println("Failed to initialize pressure sensor!");
    while (1);
  }
}

void loop() {
  // read the sensor value
  float pressure = BARO.readPressure();

  // print the sensor value
  Serial.print("Pressure = ");
  Serial.print(pressure);
  Serial.println(" kPa");

  // print an empty line
  Serial.println();

  // wait 1 second to print again
  delay(1000);
}
```

## 2.2.5 Sensore Temperatura e Umidità

Il sensore che misura la temperatura e l'umidità relativa è un HTS221, sempre di produzione della ST, ultracompatto e dotato di un condensatore planare dielettrico in grado di rilevare variazioni di umidità relativa e temperatura. I risultati vengono inviati in seriale come output digitale, con il supporto della libreria ArduinoHTS221. La temperatura misurabile varia da -40 a +120 °C, con una precisione del mezzo grado nell'intervallo 15-40 gradi; mentre l'umidità relativa offre un range che spazia dallo 0% al 100%. Le applicazioni del sensore in questione sono molte e vanno dai condizionatori, ai frigoriferi, ai dispositivi indossabili fino alla già nota smart home.

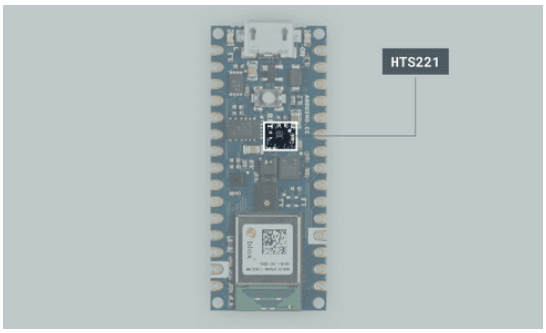



*Figura 2.14 Sensore temperatura e umidità*

L'inaffidabilità delle misurazioni raccolte dal sensore, che in genere sono sovrastime del valore reale, è dovuta al riscaldamento della scheda stessa, a causa del lavoro del processore. Per ovviare a questa problematica si può procedere in due modi: ridurre al minimo il lavoro svolto dal processore, in modo da avere un minor surriscaldamento, oppure tagliare il collegamento DC-DC, che consente la conversione da 5V a 3V. In quest'ultimo caso però viene meno la possibilità di programmare la scheda tramite USB, situazione che può essere recuperata ristabilendo il collegamento.

```
#include <Arduino_HTS221.h>

void setup() {
  Serial.begin(9600);
  while (!Serial);
// Initialize the humidity and temperature sensor
  if (!HTS.begin()) { // returns 1 on success, 0 on failure
    Serial.println("Failed to initialize humidity temperature sensor!");
    while (1);
  }
}

void loop() {
  // read all the sensor values
  float temperature = HTS.readTemperature(); //Read the temperature sensor's temperature value (Celsuis)
  float humidity    = HTS.readHumidity(); // Read the sensor's measured relative humidity value
   //returns the humidity value from the sensor as a percentage
  // print each of the sensor values
  Serial.print("Temperature = ");
  Serial.print(temperature);
  Serial.println(" °C");

  Serial.print("Humidity    = ");
  Serial.print(humidity);
  Serial.println(" %");

  // print an empty line
  Serial.println();

  // wait 1 second to print again
  delay(1000);
```

## 2.2.6 Esempio applicazione di più sensori

Nello sketch sottostante è rappresentato un programma che, una volta avviato, richiede all'utente, tramite il monitor seriale, di selezionare un sensore per ottenerne il corrispondente dato in formato numerico.

```
#include <Arduino_LSM9DS1.h>     //IMU
#include <Arduino_LPS22HB.h>    //BAROMETRO
#include <Arduino_APDS9960.h> //SENSORE DI GESTI

int sel=0;

void setup() {
  Serial.begin(9600);
  while(!Serial);
  Serial.println("Comunicazione Seriale inziata");
  //inizializz IMU
  if (!IMU.begin()) {
    Serial.println("Errore IMU!");
    while (1);
  }
  // inizializz BAROMETRO
  if (!BARO.begin()) {
    Serial.println("Errore Barometro!");
    while (1);
  }
   // inizializz SENSORE DI GESTI
  if (!APDS.begin()) {
    Serial.println("Error sensore di gesti!");
  }
}

void loop() {
float x, y, z, a, b, c;

  Serial.println("\n\nSelezionare sensore");
  Serial.println("1) IMU");
  Serial.println("2) BAROMETRO");
  Serial.println("3) SENSORE DI PROSSIMITA'");

  if(Serial.available()>0){
  sel=Serial.read(); // legge come char
  int n;
  n=int(sel); //char->int
  n-=48; //coversione ASCII
  if(sel != '\n'){
  Serial.print("\n\nHai selezionato il numero: ");
  Serial.println(n, DEC);
  }

  switch(n){
//accelerometro e giroscopio
    case 1:
    {
    if (IMU.accelerationAvailable()) {
    IMU.readAcceleration(x, y, z);
    Serial.println("\n\naccelerometro lungo assi x y z");
    Serial.print(x);
    Serial.print('\t');
    Serial.print(y);
    Serial.print('\t');
    Serial.println(z);
    }
    if (IMU.gyroscopeAvailable()) {
    IMU.readGyroscope(a, b, c);
    Serial.println("\n\ngiroscopio lungo assi x y z");
    Serial.print(a);
    Serial.print('\t');
    Serial.print(b);
    Serial.print('\t');
    Serial.println(c);
    }
    delay(2000);
    break;
    }
//barometro
    case 2:
    {
    float pressione = BARO.readPressure();
    Serial.println("\n\npressione in KPa");
    Serial.println(pressione);
    delay(2000);
    break;
    }
//sensore di gesti
    case 3:
    {
    int j=1; int k=0;
    while(k<j){
    if (APDS.proximityAvailable()) {
    int pro = APDS.readProximity();
    Serial.print("\n\nil sensore (range 0-255) misura:\n");
    Serial.println(pro);
    k++;
    }
    }
    delay(2000);
    break;
    }

    default:
    break;
    }
  }

delay(5000);
}
```

# Capitolo 3. Comportamento e Calibrazione dei Sensori

## 3.1 Temperatura e Umidità

Per raccogliere i dati generati dai sensori, al fine di studiarne il comportamento, è opportuno l'utilizzo di un software che consente di salvare in automatico su file i valori che Arduino scrive sulla seriale. Si tratta del tool: CoolTerm, cioè di un terminale di porta seriale. Tra le caratteristiche di CoolTerm si annoverano: la capacità di connessioni multiple su più porte seriali, la visualizzazione dei dati ricevuti in formato testuale o esadecimale, la possibilità di salvare in formato testuale le informazioni ricevute (in Figura 3.1).

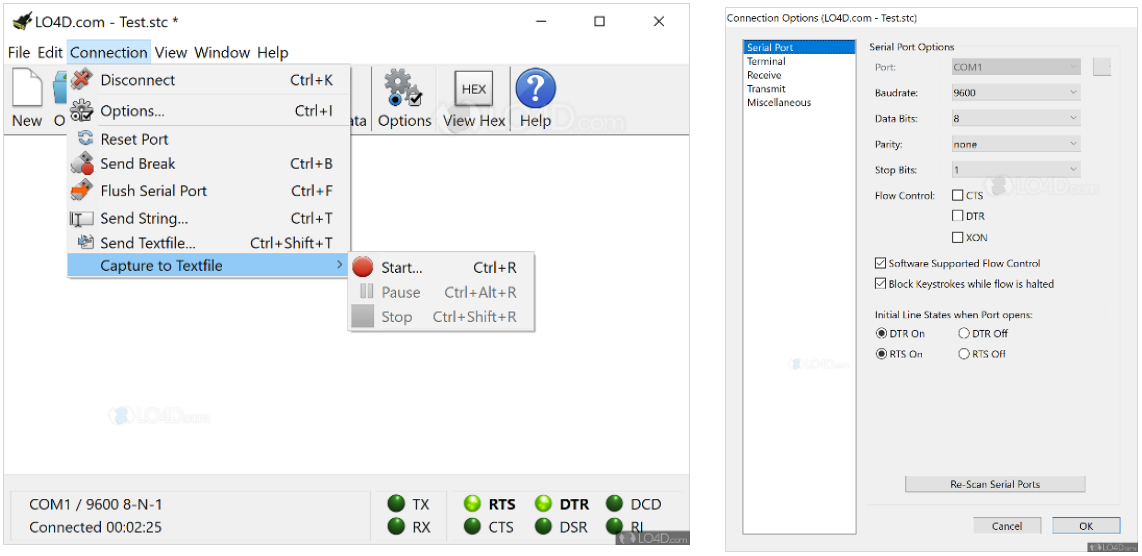
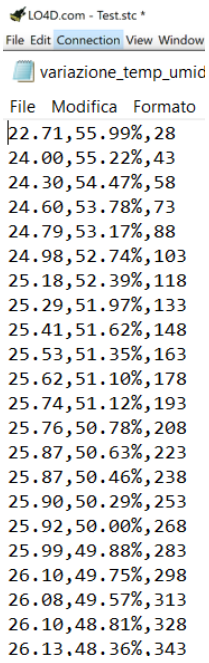



*Figura 3.1 CoolTerm*

Nel primo test relativo al sensore HTS221 si osserva l'**evoluzione** delle misurazioni effettuate **nell'arco di 45 minuti,** per constatare il comportamento del sensore al passare del tempo, cioè a seguito di un'operatività prolungata. Una volta raccolti i dati di temperatura e umidità, i quali vengono salvati in un file di testo separati da un punto (come nell'immagine sotto), tramite l'applicativo sopra presentato, si caricano su Excel per la stesura di un grafico che ne evidenzi l'andamento generale.




Di seguito viene mostrato il codice utilizzato per la raccolta dei dati e l'analisi del comportamento del sensore, tramite due grafi (Figura 3.2-3), al trascorrere del tempo.

```
#include <Arduino_HTS221.h>
unsigned long t;

void setup() {
  Serial.begin(9600);
  while (!Serial);

  if (!HTS.begin()) {
    Serial.println("Erroe!");
    while (1);
  }
  delay(20000);
}
```

```
void loop() {

  float grad = HTS.readTemperature();
  float hum = HTS.readHumidity();

  t=millis()/1000;
  Serial.print(grad);
  Serial.print(',');
  Serial.print(hum);
  Serial.print('%');
  Serial.print(',');
  Serial.println(t);

  delay(15000);
}
```

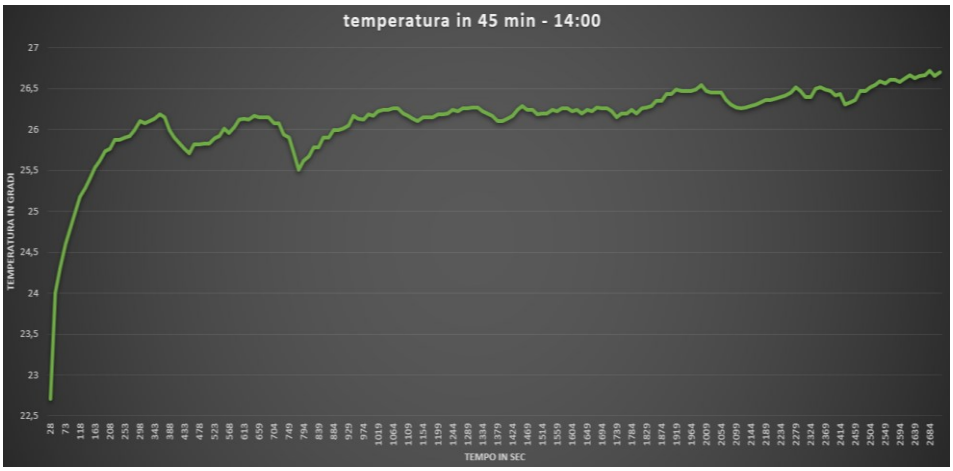



*Figura 3.2 grafico temperatura*

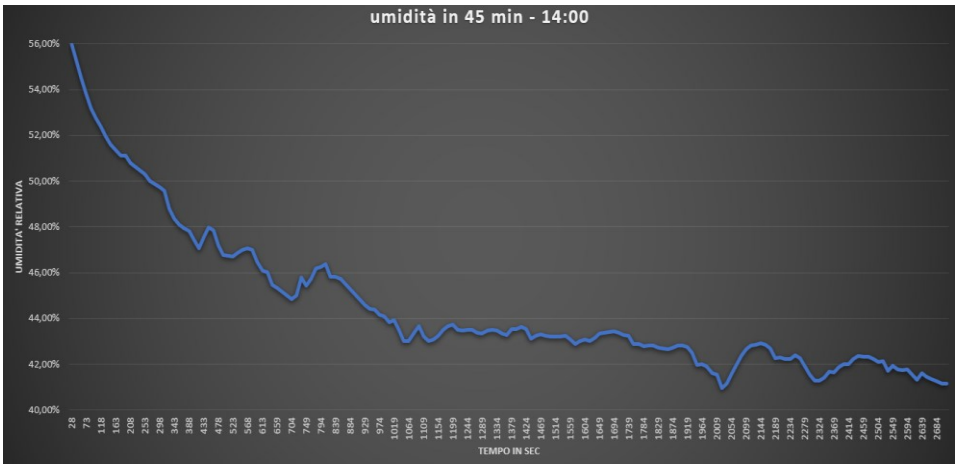



*Figura 3.3 grafico umidità relativa*

In entrambi i casi si osserva un andamento che si modifica al variare del tempo. Considerando l'umidità, il valore decresce; per la temperatura, a causa del surriscaldamento interno della scheda, aumenta. In generale, il valore calcolato dal sensore integrato per quanto concerne la temperatura, facendo un confronto con misurazioni di altri strumenti, è di qualche grado superiore.

Nello sketch sottostante si effettuano **misure ripetute** a distanza di 12 ore, alla massima velocità del dispositivo, **nell'arco di un minuto** (Figure 3.4-6). Raccolti i dati tramite CoolTerm e salvati in un file di testo, si è utilizzato Matlab per la creazione di istogrammi con lo scopo di studiarne la distribuzione di probabilità, tramite funzioni per il calcolo della media e della varianza. Il risultato delle misure effettuate è rappresentato come titolo degli istogrammi.

```
#include <Arduino_HTS221.h>

#define rit 20000
#define t_max 60000
unsigned long t;

void setup() {
  Serial.begin(9600);
  delay(rit);  //per aprire monitor seriale senza perdere dati
  while (!Serial);

 if (!HTS.begin()) {
    Serial.println("Errore sensore!");
    while (1);
  }
}
void loop() {
  float x, y, z;

    while(t<=t_max){    //calcola per 60 sec
      float temp = HTS.readTemperature();
      float hum = HTS.readHumidity();
      Serial.print(temp);
      Serial.print(',');
      Serial.print(hum);
      Serial.println('%');

  t=millis()-rit; //compenso delay del setup
}
}
```

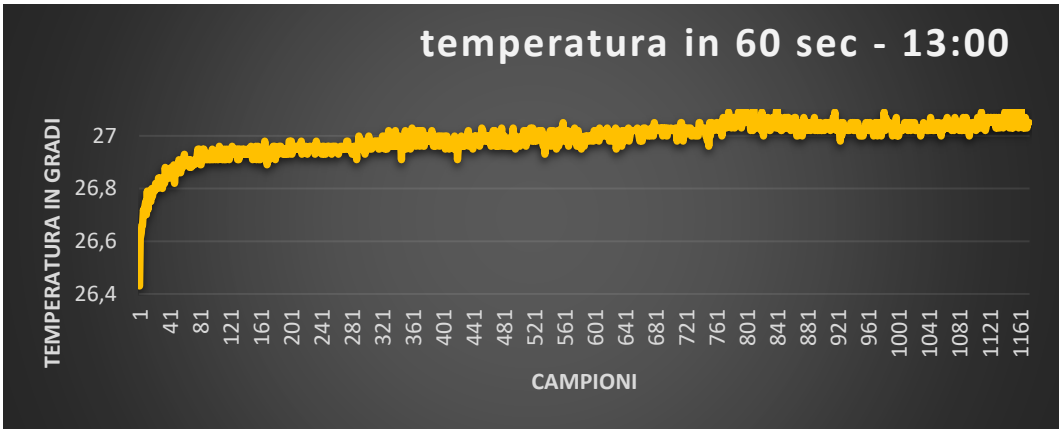
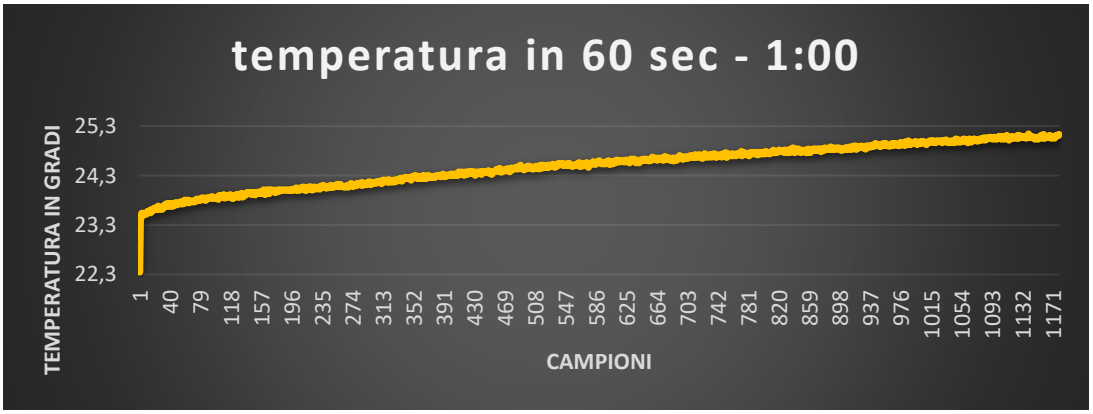



*Figura 3.4 Andamento temperatura in 60 secondi*



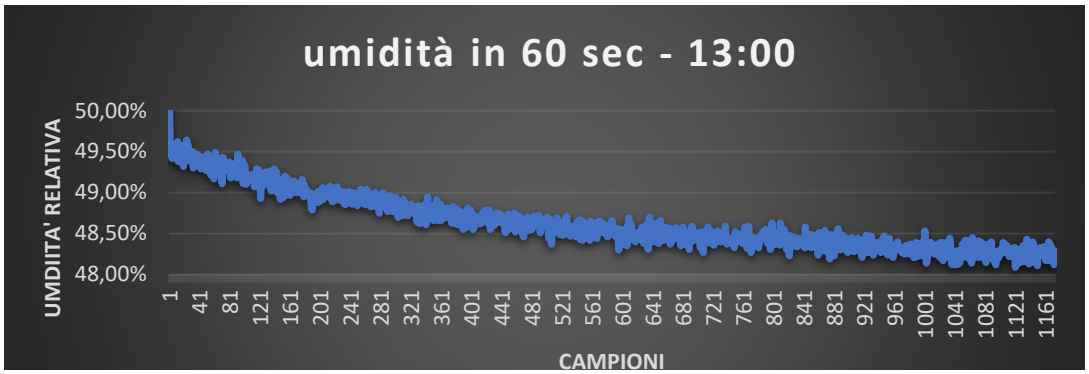
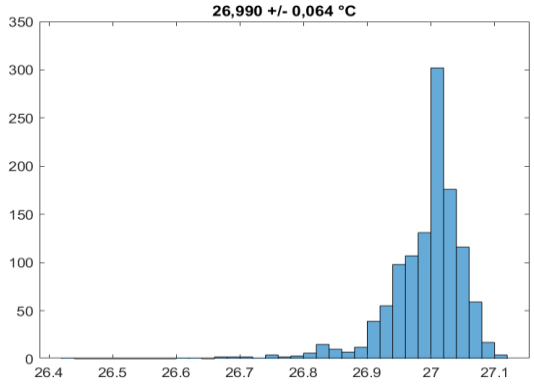
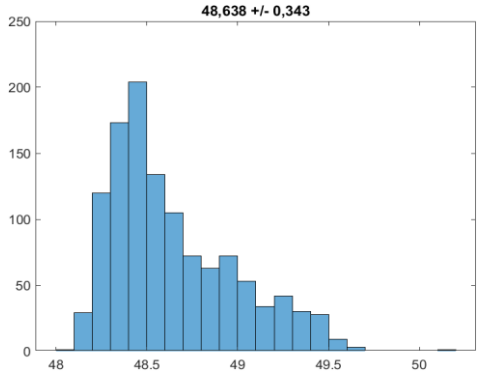


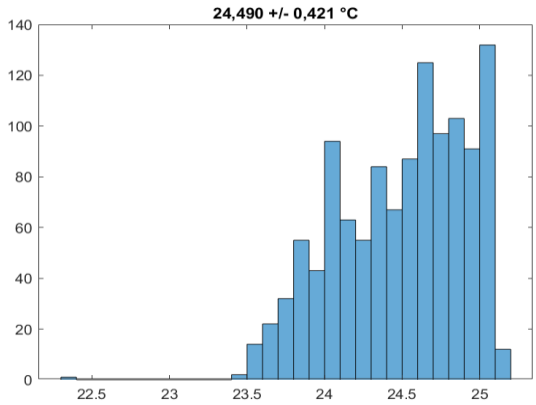

*Figura 3.5 Andamento umidità in 60 secondi*

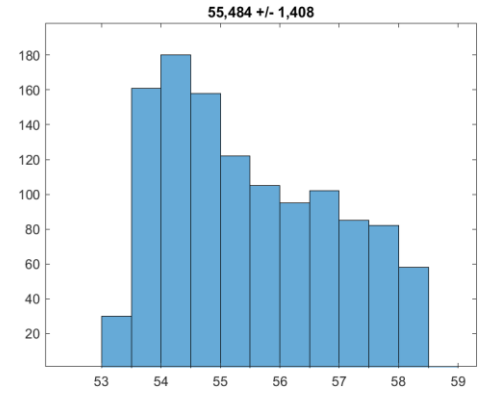
<u>temperatura in gradi</u>

<u>umidità in %</u>



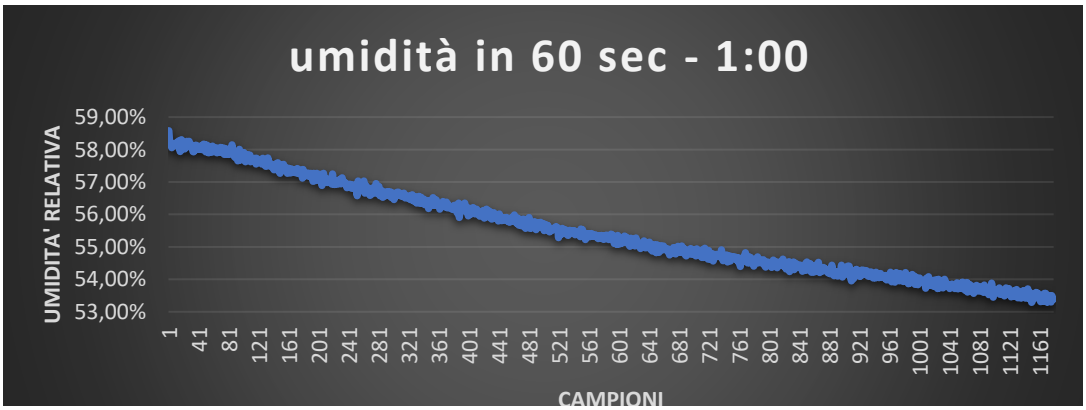




*Figura 3.6 Andamento umidità in 60 secondi*

## 3.2 Accelerazione

La valutazione della sensibilità nella misurazione da parte dell'accelerometro è svolta ponendo la Board ferma lungo ogni asse cartesiano, attraverso tre prove ripetute della durata di un minuto. Idealmente ci si aspetta un valore pari a 1 g, nelle condizioni in cui l'orientamento di uno degli assi dell'Arduino Nano sia parallelo a quello dell'accelerazione gravitazionale. Nel solito procedimento, tramite il supporto del software CoolTerm, si sono svolte le misurazioni per poi essere importate e studiate, tramite il software di elaborazione Matlab, per il calcolo della media e della varianza. Il codice utilizzato è stato il seguente:

```
#include <Arduino_LSM9DS1.h>
#define rit 20000
#define t_max 60000
unsigned long t;

void setup() {
  Serial.begin(9600);
  delay(rit);  //per aprire monitor seriale senza perdere dati
  while (!Serial);

  if (!IMU.begin()) {
    Serial.println("Erroe IMU!");
    while (1);
   }
}
void loop() {
  float x, y, z;

  //calcola per 60 sec
  while(t<=t_max){
  if (IMU.accelerationAvailable()) {
    IMU.readAcceleration(x, y, z);
    Serial.print(x);
    Serial.print(',');
    Serial.print(y);
    Serial.print(',');
    Serial.println(z);
    }
  t=millis()-rit; //compenso delay del setup
} }
```

Con circa 6953 campioni prelevati in un minuto e quindi, 116 campioni al secondo, si ottiene una frequenza di campionamento di 116 Hz che rende possibili, rispettando il teorema di Nyquist, misurazioni di accelerazioni che varino con una frequenza massima di 58 Hz. Per ogni asse sono riportati i valori riscontrati. In verde i risultati finali.
Per le ascisse:

ACC_X1: x = (-1,000 +/- 0,002)*G = -9,810 +/- 0,020 m/s^2
y = -0,1700*G = -1,6677 m/s^2
z = (-0.021 +/- 0,005) *G = 0, 206 +/- 0,049 m/s^2

ACC_X2: x = (-1,000 +/- 0,002) *G = -9,810 +/- 0,020 m/s^2
y = (0,010 +/- 0,001) *G = 0,098 +/- 0,010 m/s^2
z = (-0,010 +/- 0,006) *G = 0,098 +/- 0,059 m/s^2

ACC_X3: x = (-1,000 +/- 0,002) *G = -9,810 +/- 0,020 m/s^2
y = (0,013 +/- 0,005) *G = 0,128 +/- 0,049 m/s^2
z = (-0.014 +/- 0.007) *G = 0,137 +/- 0,069 m/s^2

Nella tabella sottostante si considerano prima le misurazioni ottenute ponendo l'asse Y parallelo all'accelerazione gravitazionale, in seguito l'asse Z, con tre tentativi per asse:

| ASSE | media | | deviazione std | | media*G | | dev std*G |
|---|---|---|---|---|---|---|---|
| x= | 0,010 | +/- | 0,010 | = | 0,098 | +/- | 0,098 |
| y= | 0,945 | +/- | 0,005 | = | 9,270 | +/- | 0,049 |
| z= | -0,090 | +/- | 0,050 | = | -0,883 | +/- | 0,491 |
| x= | 0,120 | +/- | 0,001 | = | 1,177 | +/- | 0,010 |
| y= | 0,940 | +/- | 0,001 | = | 9,221 | +/- | 0,010 |
| z= | -0,040 | +/- | 0,004 | = | -0,392 | +/- | 0,039 |
| x= | -0,002 | +/- | 0,004 | = | -0,020 | +/- | 0,039 |
| y= | 0,950 | +/- | 0,001 | = | 9,320 | +/- | 0,010 |
| z= | -0,030 | +/- | 0,005 | = | -0,294 | +/- | 0,049 |
| | | | | | | | |
| x= | 0,000 | +/- | 0,001 | = | 0,000 | +/- | 0,009 |
| y= | -0,0400 | +/- | 0,0004 | = | -0,3924 | +/- | 0,0039 |
| z= | 0,979 | +/- | 0,004 | = | 9,604 | +/- | 0,039 |
| x= | 0,000 | +/- | 0,001 | = | 0,000 | +/- | 0,010 |
| y= | -0,0400 | +/- | 0,0004 | = | -0,3924 | +/- | 0,0039 |
| z= | 0,980 | +/- | 0,005 | = | 9,614 | +/- | 0,049 |
| x= | 0,004 | +/- | 0,005 | = | 0,039 | +/- | 0,049 |
| y= | -0,0500 | +/- | 0,0005 | = | -0,4905 | +/- | 0,0049 |
| z= | 0,980 | +/- | 0,004 | = | 9,614 | +/- | 0,039 |

I risultati finali sono dati da una media delle medie per ogni asse (prese dalla tabella sopra) e come deviazione standard quella maggiore riscontrata (sempre dalla tabella sopra), da cui si ottiene:

$$X = (-1{,}000 \ +/- \ 0{,}002) * G = -9{,}810 \ +/-0{,}020 \frac{m}{s^2}$$

$$Y = (0{,}945 \ +/-0{,}005) * G = 9{,}270 \ +/-0{,}049 \frac{m}{s^2}$$

$$\mathrm{Z} = (0{,}980 \ +/-0{,}005) * G = 9{,}614 \ +/-0{,}049 \frac{m}{s^2}$$

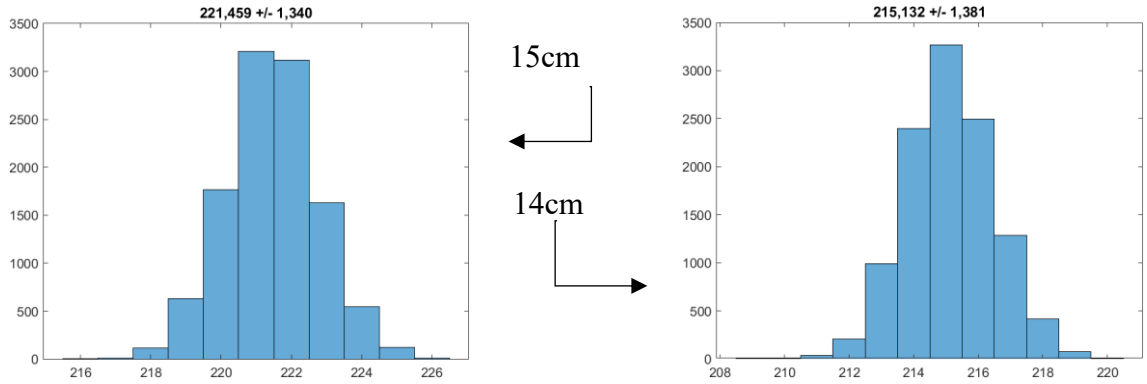
Si osserva una buona precisione del sensore per quanto riguarda l'asse delle ascisse, molto vicino infatti all'accelerazione di gravità pari a 9,80665 metri su secondo quadro, una minor precisione lungo l'asse Z e la massima imprecisione per l'asse Y.

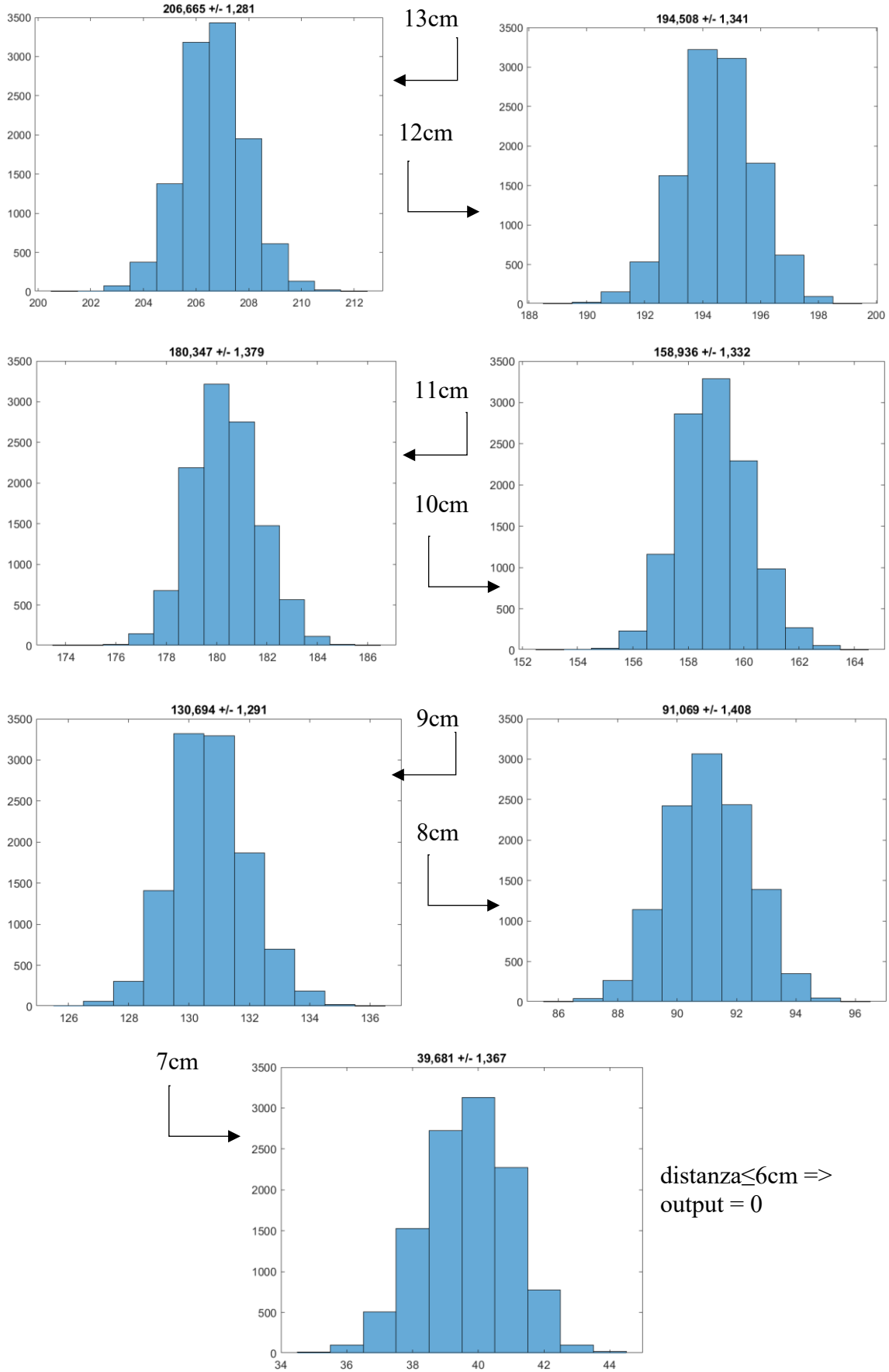
# 3.3 Sensore di Prossimità

Il sensore di prossimità, una volta effettuata la misurazione, restituisce in output un valore compreso tra 0 e 255, dove 0 esprime la massima vicinanza e 255 l'estremo opposto, -1 si ha nel caso in cui si siano verificati degli errori interni al sensore. Lo scopo della calibrazione è, tramite il supporto di uno strumento che mantenga un oggetto ad una distanza fissa dal sensore, stabilire un legame tra l'uscita numerica del sensore e la distanza reale misurata in cm. Per tale scopo si è utilizzato un programma che, per circa un minuto, esegue misurazioni continue; i valori in uscita, tramite il software CoolTerm sono stati salvati su un file di testo, importati sul programma di elaborazione Matlab e analizzati tramite istogrammi, medie e varianze. Lo sketch utilizzato è riportato di seguito:

```
#include <Arduino_APDS9960.h>

#define rit 10000
#define t_max 60000
unsigned long t;

void setup() {
  Serial.begin(9600);
  delay(rit);  //per aprire monitor seriale senza perdere dati
  while (!Serial);

  if (!APDS.begin()) {
    Serial.println("Error initializing APDS9960 sensor!");
  }
}


void loop() {

  //calcola per 60 sec
  while(t<=t_max){
  if (APDS.proximityAvailable()) {
    int proximity = APDS.readProximity();
    Serial.println(proximity);
  }
  t=millis()-rit; //compenso delay del setup
}
}
```

I risultati ottenuti sono rappresentati nelle figure sottostanti: si osserva un andamento gaussiano delle misure effettuate. Media e varianza sono indicate come titolo dei grafici, ai quali è associata la distanza reale corrispondente in centimetri.




206,665 +/- 1,281
13cm
194,508 +/- 1,341
12cm
180,347 +/- 1,379
11cm
158,936 +/- 1,332
10cm
130,694 +/- 1,291
9cm
91,069 +/- 1,408
8cm
7cm
39,681 +/- 1,367
distanza≤6cm =>
output = 0

Stabilito il legame tra l'uscita numerica del sensore e la distanza reale, viene implementato uno sketch in cui il sensore, tramite i dati e le analisi elencate sopra, ritorna sul monitor seriale la distanza reale misurata in centimetri. Il codice fa uso di una funzione: "f_prox", che viene richiamata all'interno del "loop" ed è definita alla fine dello sketch.

```
void loop() {
  f_prox();
  delay(100);
}

void f_prox(){
int prox;
do{
  if (APDS.proximityAvailable()) {          //verifica se sono presenti campioni
      prox = APDS.readProximity();         //ritorna un numero da 0 a 255 (0->vicino, 255->lontano)
      if(prox>225){
        Serial.println("maggiore di 15cm");
        delay(1000);
      }
      if(prox>218 && prox<=225){
        Serial.println("15cm");
        delay(1000);
      }
       if(prox>211 && prox<=218){
        Serial.println("14cm");
        delay(1000);
      }
       if(prox>200 && prox<=211){
        Serial.println("13cm");
        delay(1000);
      }
      if(prox>187 && prox<=200){
        Serial.println("12cm");
        delay(1000);
      }
      if(prox>169 && prox<=187){
        Serial.println("11cm");
        delay(1000);
      }
       if(prox>144 && prox<=169){
        Serial.println("10cm");
        delay(1000);
      }
      if(prox>110 && prox<=144){
        Serial.println("9cm");
        delay(1000);
      }
      if(prox>65 && prox<=110){
        Serial.println("8cm");
        delay(1000);
      }
      if(prox>38 && prox<=65){
        Serial.println("7cm");
        delay(1000);
      }
  }
}while(prox>38);
    Serial.println("minore di 6");
    Serial.print("\n\n");
    delay(1000);
 }
```

## 3.4 Considerazioni finali sui sensori valutati

In definitiva, riassumendo l'analisi dei comportamenti dei vari sensori considerati, si può constatare che:

- il sensore HTS221 ha un comportamento che è ampiamente influenzato dal surriscaldamento delle componenti dell'Arduino, tanto da influenzarne significativamente i risultati di misura a seguito di un'attività prolungata;
- il sensore LSM9DS1 permette un calcolo ottimale dell'accelerazione solamente per l'asse delle ascisse, mentre fornisce misure sempre meno veritiere lungo l'asse Z e Y rispettivamente;
- il sensore APDS9960 garantisce di ottenere, con una discreta precisione, la distanza di ostali da esso.

# Capitolo 4. Bluetooth Low Energy

## 4.1 Introduzione al BLE

Il Bluetooth Low Energy è una tecnologia wireless distribuita da Bluetooth SIG, che ha come principale differenza rispetto al Bluetooth classico il minor consumo di energia e costo (Figura 4.1).

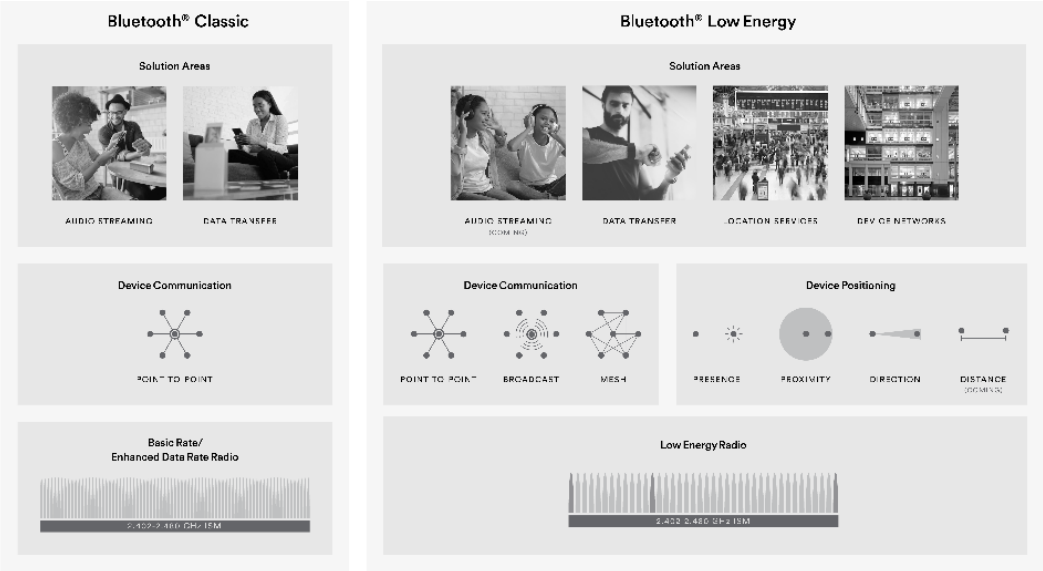



*Figura 4.1 BL e BLE*

Entrambi trasmettono intorno alla frequenza 2.4 GHz su bande non licenziate, ma il BLE usa un tipo di modulazione più semplice. Le modalità di comunicazione supportate sono molteplici: punto-punto, broadcast e mesh. Aspetto fondamentale da considerare è che BL e BLE operano in modo differente e quindi sono incompatibili tra di loro. Con un bit-rate di 1 Mbps e un consumo molto inferiore al BL classico, il nuovo modello di trasmissione, i cui albori risalgono ad un progetto Nokia del 2004, è uno strumento fondamentale che si integra perfettamente nel nuovo paradigma dell'Internet of Things. Si stima che nel 2018 i dispositivi dotati di Bluetooth siano stati circa 4 miliardi, presenti in tutti quei dispositivi per assistenza sanitaria (come i cardiofrequenzimetri), fitness (gli smartwatch) e localizzazione. Il basso costo, la facilità di implementazione e il ridotto

consumo sono i principi cardine del Bluetooth LE. Diversamente, il BL classico copre altri tipi di ruoli, infatti è principalmente utilizzato per trasmissioni di tipo audio, come nelle cuffie wireless ad esempio. Il Bluetooth LE utilizzato, come visto, per le trasmissioni di piccole quantità di dati, deve la propria capacità di ridotto consumo alla modalità di comunicazione che consiste nel permanere nella condizione di sleep mode per la maggior parte del tempo.
L'architettura dello standard prevede due tipi di device (Figura 4.2):

- il **server** (detto anche dispositivo centrale), si occupa di segnalare la propria presenza al client;
- il **client** (dispositivo periferico), scannerizza le vicinanze alla ricerca di un server a cui ci si vuole collegare. Stabilita la connessione si procede al trasferimento dei dati. Questo tipo di comunicazione è *detta point-to-point communication*.

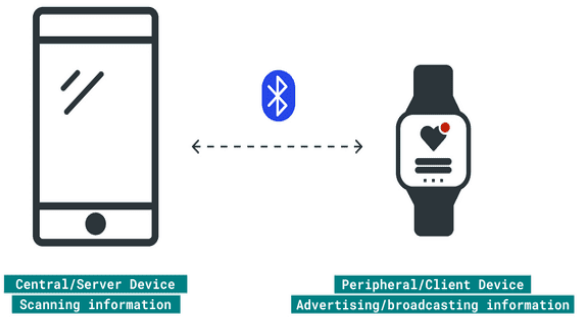



*Figura 4.2 Server e Client*

In generale i device BLE segnalano la propria presenza agli altri dispositivi attraverso il GAP (General Advertising Profile), dove i pacchetti inviati contengono nome, i servizi forniti e altre informazioni. Per comprendere le modalità di comunicazione tra dispositivi, si faccia riferimento alla Figura 4.3.

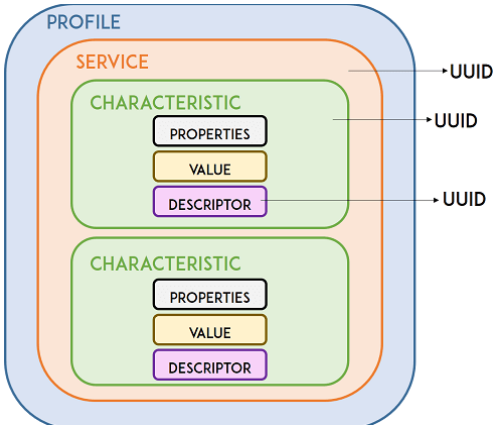



*Figura 4.3 Architettura BLE*

Il livello più alto della gerarchia è il profilo, costituito da uno o più servizi, ogni servizio contiene almeno una caratteristica e costituisce un insieme di informazioni; ci sono vari servizi predefiniti dalla SIG, come ad esempio il livello di batteria di un dispositivo. La caratteristica appartiene al servizio e contiene le informazioni, si articola in: proprietà, suggerisce come devono essere letti i dati; valore, lungo fino a 512 bytes; descrittore, fa parte dei metadata. Per fare un esempio, in uno smartwatch la misura del battito cardiaco e le ore di sonno totali calcolate rientrano in due caratteristiche differenti, ma entrambi appartenenti ad un unico servizio comune: "health service". Ogni servizio, caratteristica e descrittore hanno un UUID (Universally Unique Identifier), ovvero un identificatore univoco a 16 o 32-bit per le versioni ufficiali Bluetooth, mentre di 128-bit per quelle non ufficiali che possono essere create in modo randomico.

Il dispositivo centrale può compiere quattro operazioni con la caratteristica:

- READ, chiede al dispositivo periferico di mandare il valore della caratteristica;
- WRITE, modifica il valore della caratteristica;
- NOTIFY AND INDICATE, comanda al dispositivo periferico di aggiornare continuamente il valore della caratteristica, senza doverlo interrogare. Nel caso specifico di *Notify*, una volta che il device periferico aggiorna il valore della caratteristica esso è automaticamente inviato al dispositivo centrale (questo tipo di modalità è utilizzata con gli accelerometri). Invece, per *Indicate* l'unica differenza è rappresentata dal messaggio di acknowledgement da parte del dispositivo centrale al proprio dispositivo periferico.

## 4.2 Applicazioni del BLE con Arduino

L'Arduino NANO 33 BLE SENSE è basato sul modulo NINA-B3 Series, ovvero uno standalone Bluetooth 5 low energy, come mostrato in Figura 4.4.

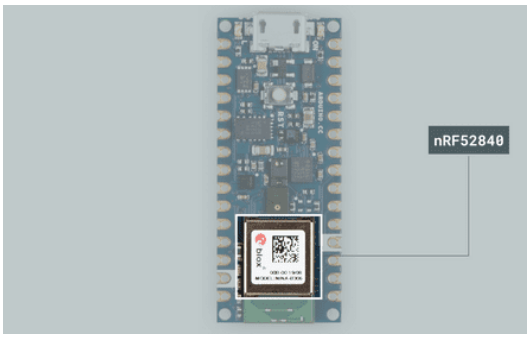



*Figura 4.4 Modulo BLE*

Per la creazione di un server BLE nell'IDE Arduino si seguono i seguenti passi:

i. *Creazione del Servizio: BLEService name(UUID)*
ii. *Creazione della Caratteristica: BLECharacteristic name(UUID)*
iii. *Inizializzazione del BLE: BLE.begin()*
iv. *Assegnazione del nome al dispositivo: BLE.setLocalName("name")*
v. *Imposizione dell'UUID del Servizio per la segnalazione: BLE.setAdvertisedService(bleService)*
vi. *Aggiunta della Caratteristica al Servizio: bleService.addCharacteristic(bleCharacteristic)*
vii. *Aggiunta del servizio: BLE.addService(service)*
viii. *(facoltativo) Scrittura del valore della Caratteristica: bleCharacteristic.writeValue(value)*
ix. *Segnalazione di presenza per essere individuato da altri dispositiv: BLE.advertise()*
x. *Interrogazione del client: BLEDevice central = BLE.central()*
xi. *Verifica della connessione di un dispositivo: bleDevice.connected()*

### 4.2.1 Accelerometro BLE

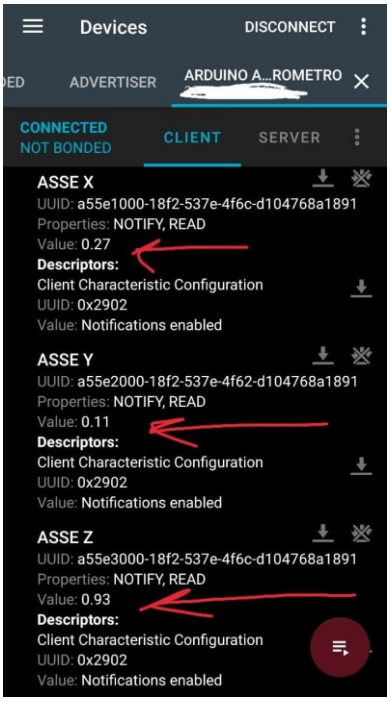
Uno degli utilizzi più conosciuti per il Bluetooth LE è nelle applicazioni di dispositivi indossabili, come può essere il caso di un accelerometro. Nella stesura del codice, all'interno dell'area riservata alle variabili globali, si enunciano il Servizio e le tre Caratteristiche, una per ogni asse cartesiano, con i rispettivi identificativi univoci UUID. All'interno del setup, dopo aver inizializzato IMU, BLE e PIN integrato come output, si assegna un nome al dispositivo, si aggiungono le Caratteristiche al Servizio e si procede con la segnalazione, ai device circostanti, della propria presenza. All'interno del loop si verifica che il client sia connesso, fatto ciò, se ne ha riscontro sul monitor seriale e sulla scheda fisica; infatti, il LED rimane acceso per tutta la durata della connessione del client. Stabilita la connessione, si richiama una funzione che svolge sostanzialmente il ruolo del classico sketch per la lettura del sensore, tali valori vengono trasformati in stringhe e scritti sulla caratteristica, in modo da essere visualizzati da un dispositivo remoto. Quest'ultima operazione è dovuta alla tipologia dell'app di visualizzazione: nRF Connect (Figura 4.5).

*Figura 4.5 nRF Connect logo*

Un'applicazione che permette la visualizzazione dei valori passati in formato file di testo o numeri esadecimali, perciò, per una più rapida visualizzazione, è preferibile passare i numeri come se fossero stringhe di testo. **nRF Connect**, per dispositivi mobili, è un potente tool prodotto dalla Nordic Semiconductor che ha varie funzionalità:

- Scansiona alla ricerca di device Bluetooth LE;
- Illustra e analizza le Caratteristiche e i Servizi;
- Legge e scrive sulle Caratteristiche;
- Attiva e disattiva le notifiche;
- Tiene registrazione degli eventi di log.

Una volta aperta l'app da smartphone, si procede a scansionare i dispositivi nelle vicinanze alla ricerca di un device BLE. Effettuata la connessione ci si sposta nella sezione corrispondente al dispositivo a cui si è collegati, dove possono essere letti (spuntata l'icona delle 3 frecce verso il basso bianche) i valori delle caratteristiche aggiornati in tempo reale. Un riscontro visivo si ha anche sul monitor seriale, come mostrato nelle figure. La Figura 4.6 mostra l'interfaccia di nRF Connect, in cui le frecce rosse indicano i valori (in tempo reale) della caratteristica; mentre l'altra ciò che viene visualizzato sul monitor seriale, ovvero gli stessi valori letti dalle Caratteristiche, ma passati sotto forma di float.




```
Seriale avviata
BLE acceso...
Connesso al client: 63:f7:
X       Y       Z
0.29    0.16    0.91


X       Y       Z
0.29    0.16    0.91


X       Y       Z
0.28    0.16    0.91


X       Y       Z
0.29    0.16    0.91
```

*Figura 4.6 nRF Connect App*

Il codice utilizzato è riportato nella pagina successiva.

```
// invio dati accelerometro tramite BLE, con riscontro su seriale e LED BUILTIN

#define UUID_S "19B1A600-E8F2-537E-4F6C-D104768A1219"     //UUID custom del servizio e delle caratteristiche
#define UUID_X "A55E1000-18F2-537E-4F6C-D104768A1891"
#define UUID_Y "A55E2000-18F2-537E-4F62-D104768A1891"
#define UUID_Z "A55E3000-18F2-537E-4F6C-D104768A1891"

#include <Arduino_LSM9DS1.h>
#include <ArduinoBLE.h>

BLEService acc_service(UUID_S);                                                  //crea un nuovo servizio BLE
BLEStringCharacteristic acc_charact_x(UUID_X, BLERead | BLENotify, 10);         //crea una nuova caratteristica per ogni a
BLEStringCharacteristic acc_charact_y(UUID_Y, BLERead | BLENotify, 10);        //a55e1 == asse X e così via
BLEStringCharacteristic acc_charact_z(UUID_Z, BLERead | BLENotify, 10);

float x,y,z;                                                                   //variabili globali per l'accelerometro


void setup() {

  Serial.begin(9600);                    //avvia comunicazione seriale
  while (!Serial);
  Serial.println("Seriale avviata");

  if (!IMU.begin()) {                  //avvia l'IMU
    Serial.println("Errore IMU!");
    while (1);
  }
 if(!BLE.begin()){                   //inizializza il device BLE
  Serial.println("Errore BLE!");
 while(1);
 }
pinMode(13, OUTPUT);              //setta il pin integrato come output

BLE.setLocalName("Arduino accelerometro");             //imposta il nome del dispositivo BLE

BLE.setAdvertisedService(acc_service);               //associa l'UUID quando segnala la sua presenza al servizio fornito

acc_service.addCharacteristic(acc_charact_x);      //aggiunge le caratteristice al servizio
acc_service.addCharacteristic(acc_charact_y);
acc_service.addCharacteristic(acc_charact_z);
BLE.addService(acc_service);                    //aggiunge il servizio al dispositivo BLE

BLE.advertise();                              //il dispositivo BLE segnala la sua presenza
Serial.println("BLE acceso...");
}


void loop() {

  BLEDevice central = BLE.central();              //attende un client e lo assegna

  if (central) {                                 //se il client è connesso
    Serial.print("Connesso al client: ");
    Serial.println(central.address());        //indirizzo MAC del client
    while (central.connected()) {            //finchè il client è connesso ritorna 1
      digitalWrite(13, HIGH);               //accende il led
      delay(500);
      acc_funz();                          //chiama la funzione per calcolare l'accelerazione lungo gli assi
      String x_w, y_w, z_w;             //creo le stringhe da inviare tramite BLE
      x_w=String(x, 2);                //converto i float in stringhe (2 = numeri dopo virgola)
      y_w=String(y, 2);               //https://www.arduino.cc/reference/en/language/variables/data-types/stringobject/
      z_w=String(z, 2);

      acc_charact_x.writeValue(x_w);//scrive il valore all'interno delle caratteristiche
      acc_charact_y.writeValue(y_w);
      acc_charact_z.writeValue(z_w);

    }
    Serial.print("Client disconnesso: ");  //segnala la disconnessione del client
    Serial.println(central.address());    //con relativo indirizzo MAC
    digitalWrite(13, LOW);               //spenge il led
  }
}

void acc_funz() {

  if (IMU.accelerationAvailable()) {     //verifica se sono disponibili nuovi campioni
    IMU.readAcceleration(x, y, z);      //legge i valori e li assegna nelle variabili

      Serial.println("X\tY\tZ");      //manda i valori tramite seriale
      Serial.print(x);
      Serial.print('\t');
      Serial.print(y);
      Serial.print('\t');
      Serial.println(z);
      Serial.println("\n");
   }
}
```

## 4.2.2 Menù BLE

Nel programma sottostante si utilizza la comunicazione BLE per trasmettere dei dati, i quali vengono richiesti tramite interazione gestuale per mezzo del sensore gesti integrato nell'Arduino. A seguito di un movimento rilevato dal sensore, si inviano tramite Bluetooth LE i valori acquisiti di pressione, temperatura e umidità.

```
/* obiettivo: nodo (cheda Arduino) che misura i parametri ambientali, ma allo stesso tempo
 * in grado di riconoscere alcune forme di interazione utente, come i gesti
 * e l'interazione vocale.
 NOTA: il sensore APDS rimane attivo per rilevare gesti -> non si possono chiamare funzioni che lo usano*/

#define UUID_SERVICE   "19B1A600-E8F2-531E-4F6C-D100368A1219"
#define UUID_C_TEMP    "CA190000-18F2-537E-4F6C-D104768A1891"
#define UUID_C_PRESS   "9A0A0255-18F2-537E-4F62-D104768A1891"
#define UUID_C_HUM     "AFA00AFA-18F2-537E-4F6C-D104768A1891"


#include <Arduino_HTS221.h>
#include <Arduino_APDS9960.h>
#include <ArduinoBLE.h>
#include <Arduino_LPS22HB.h>


BLEService ambiente(UUID_SERVICE);                                          //crea un nuovo servizio BLE
BLEStringCharacteristic c_temp(UUID_C_TEMP, BLERead | BLENotify, 10);       //crea una nuova caratteristica: temperatura
BLEStringCharacteristic c_hum(UUID_C_HUM, BLERead | BLENotify, 5);          //umidità relativa
BLEStringCharacteristic c_press(UUID_C_PRESS, BLERead | BLENotify, 6);      //sensore di pressione


void setup() {

  Serial.begin(9600);                                                   //avvia comunicazione seriale
  while (!Serial);
  Serial.println("Seriale avviata");

  if (!BARO.begin()) {
   Serial.println("Errore sensore di pressione!");
   while (1);
 }

 if (!HTS.begin()) {                                                  //avvia sensore temperatura e umidità
   Serial.println("Erroe sensore di temperatura e umdità!");
   while (1);
 }

 if (!APDS.begin())                                                 //avvia sensore di gesti
 Serial.println("Errore sensore APDS!");
 Serial.println("Rilevando gesti ...\n");

if(!BLE.begin()){                   //inizializza il device BLE
 Serial.println("Errore BLE!");
 while(1);
 }
pinMode(13, OUTPUT);              //setta il pin integrato come output

BLE.setLocalName("Arduino ambiente");                //imposta il nome del dispositivo BLE

BLE.setAdvertisedService(ambiente);                 //associa l'UUID quando segnala la sua presenza al servizio fornito

ambiente.addCharacteristic(c_temp);               //aggiunge le caratteristice al servizio
ambiente.addCharacteristic(c_press);
ambiente.addCharacteristic(c_hum);
BLE.addService(ambiente);                      //aggiunge il servizio al dispositivo BLE


BLE.advertise();                            //il dispositivo BLE segnala la sua presenza
 Serial.println("BLE acceso...");
}

void loop() {

  BLEDevice central = BLE.central();              //attende un client e lo assegna

  if (central) {                                 //se il client è connesso
    Serial.print("Connesso al client: ");
    Serial.println(central.address());         //indirizzo MAC del client
    while (central.connected()) {             //finchè il client è connesso ritorna 1
      digitalWrite(13, HIGH);                //accende il led
```

```
        if (APDS.gestureAvailable()) {    //controlla se sono stati individuati dei gesti e li legge
           int gesture = APDS.readGesture();

           switch (gesture) {
             case GESTURE_UP:
             {
             Serial.println("UP");
             f_temp();
             break;
             }

           case GESTURE_DOWN:
             {
             Serial.println("DOWN");
             f_hum();
             break;
             }
            case GESTURE_LEFT:
              {
              Serial.println("LEFT");
              f_bar();
              break;
              }

            case GESTURE_RIGHT:
              {
              Serial.println("RIGHT");
              Serial.print("vuoto\n\n");
              break;
              }

             default:
             break;
          }
        }
    }
    Serial.print("Client disconnesso: ");  //segnala la disconnessione del client
    Serial.println(central.address());    //con relativo indirizzo MAC
    digitalWrite(13, LOW);               //spenge il led
  }
}

void f_temp(){

  float temp = HTS.readTemperature();  //legge la temperatura (gradi Celsius)
  Serial.print("Temperatura = ");
  Serial.print(temp);
   Serial.print("Temperatura = ");
   Serial.print(temp);
   Serial.println(" °C\n");
   String t = String(temp, 2);
   c_temp.writeValue(t);         //scrivo sulla caratteristica
   delay(1000);

}

 void f_hum(){

   float hum = HTS.readHumidity();            //legge l'umidità relativa (%)
   Serial.print("Umidità relativa    = ");
   Serial.print(hum);
   Serial.println(" %\n");
   String h = String(hum);
   c_hum.writeValue(h);                 //scrivo sulla caratteristica
   delay(1000);

 }

 void f_bar(){
   float pres = BARO.readPressure();    //legge la pressione
   String p = String(pres, 2);
   Serial.print("Pressione = ");
   Serial.print(pres);
   Serial.println(" kPa\n");
   c_press.writeValue(p);          //scrivo sulla caratteristica
   delay(1000);
 }
```

# Capitolo 5. Embedded Machine Learning

## 5.1 Introduzione al Machine Learning

Quando si parla di Machine Learning è bene sottolineare che in realtà vi sono tre concetti chiave da differenziare: **Intelligenza Artificiale** (IA), **Machine Learning** (ML) e **Deep Learning** (DL). In particolare, per IA si intende quella scienza che si occupa di rendere gli oggetti "smart", ciò si manifesta sostanzialmente quando l'intelligenza umana è utilizzata dalle macchine, cioè in tutti quei dispositivi capaci di "pensare". La nascita dell'Intelligenza Artificiale risale a metà del Novecento, quando un ricercatore della IBM, Arthur Lee Samuels, sviluppò il primo programma di ML per il gioco della dama. In sintesi, l'IA si basa sul fornire ad un computer un insieme di dati, non processati, con i quali iniziare ad imparare ed allenarlo a prevedere in base alle osservazioni passate. Il termine fu coniato dallo stesso ricercatore e spiegato in un paper, pubblicato nel 1959, sull'IBM Journal of Research and Development. L'intelligenza Artificiale è caratterizzata da tre sottoinsiemi principali dai nomi:

- Reasoning, cioè la capacità di fare deduzione partendo dai dati disponibili, più esplicitamente è la capacità di riempire gli spazi vuoti quando mancano i dati;
- Natural Language Processing (NLP), rappresenta l'abilità dei computer di capire sia il linguaggio testuale che quello umano. Questa importante caratteristica permette a persone, che non hanno una preparazione adeguatamente complessa al sistema con cui si stanno relazionando, di comunicare in modo semplice;
- Planning, ovvero la competenza di un sistema intelligente nel predisporsi come obiettivi degli stadi intermedi da ottemperare per raggiungere il traguardo finale richiesto, attraverso stadi intermedi fatti di azioni.

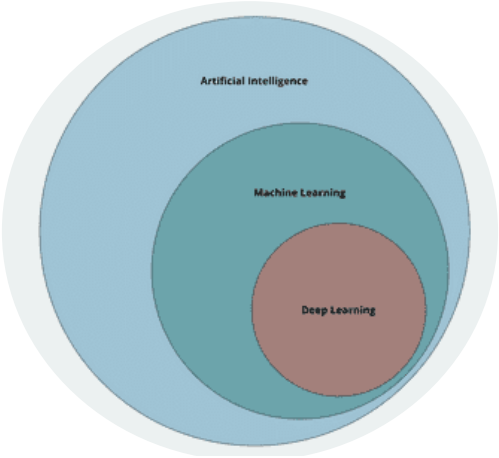



*Figura 5.1 Sottoinsiemi del ML*

Il DL invece è una tecnica per implementare il Machine Learning tramite l'utilizzo delle Reti Neurali Artificiali (ANN), cioè algoritmi dotati di modelli matematici per l'elaborazione dei dati in grado di imparare dalle informazioni fornite in ingresso. Lo scopo, attraverso la suddivisione di rete neurali in più substrati, è la riproduzione su scala artificiale del funzionamento del cervello umano. Le applicazioni principali delle reti neurali e del DL riguardano il riconoscimento di immagini e della voce. Una rete neurale solitamente si suddivide in tre livelli: un *input layer*, uno o più *hidden layer* e un *output layer*. I dati vengono passati attraverso il primo strato (input layer), modificati dagli strati intermedi (hidden layer) e fuoriescono dall'ultimo strato (output layer). Il termine DL è utilizzato quando ci sono più strati nascosti dentro una rete neurale. Il DL è una tecnica di ML che utilizza una gestione gerarchica delle reti neurali, viene utilizzato ampiamente nella produzione industriale basata sull'IoT con lo scopo, ad esempio, di prevedere quando un macchinario avrà un malfunzionamento, oppure utilizzato per il tracciamento di persone sospettate di reati.

Le modalità di apprendimento della rete neurale si suddividono in due tipi principali: **Supervised learning** e **Unsupervised learning**. Nel primo caso si tratta di un learning classificato, ovvero i dati in ingresso vengono suddivisi in categorie che fanno riferimento ad un insieme comune, etichettato da una label. Un esempio può essere quello di fornire in ingresso a una rete neurale un insieme di immagini di animali, in cui ogni foto è caratterizzata da una label che ne indica la specie, in modo da permettere al sistema la distinzione tra un animale e l'altro. Essendo l'algoritmo allenato da esempi preconfezionati, le proprie performance sono valutate da un insieme di dati completamente nuovi che fungono da test, banalmente i *test data*. Se il sistema è soggetto a *overfitting*, cioè predisposto unicamente a lavorare con i dati fornitigli da una stessa fonte e non applicabile a nuovi e sconosciuti dati, la rete neurale non è di buona qualità e il modo per aggirare questo problema è tramite il test con dati mai visti prima. Gli utilizzi di questo approccio spaziano dal rilevamento di frodi, all'analisi del rischio

fino al riconoscimento del segnale vocale umano. Nell'Unsupervised learning i training data, cioè le informazioni date in ingresso per esercitare la rete neurale, non sono suddivisi per label predefinite differenti, ma è la rete a riconoscere degli schemi simili o differenti nei dati forniti in ingresso. Un esempio di applicazione di questo modello è la gestione delle e-mail spam; infatti, l'assegnazione nella casella dello spam delle email non volute si basa su valutazioni di molti fattori che non sono stati precedentemente sottoposti a categorizzazione.
Un'alternativa alla prima tipologia di ML è il **Reinforcement learning**, un modello di apprendimento comportamentale in cui il sistema non impara da un insieme di dati campionati, ma lo fa tramite una procedura basata su tentativi ed errori. Dunque, una sequenza di decisioni giuste va a rinforzare il modello stesso. La robotica è il campo in cui questo modello è preponderante; infatti, prendendo il caso di un robot che cerca di salire delle scale, esso a seguito di cadute andrà a ricalibrare le azioni fino al raggiungimento del risultato: il robot impara da una successione di risultati positivi dopo un allenamento basato su "trial and error".
La differenza fondamentale, da parte del ML, con la programmazione tradizionale (Figura 5.2) è insita nell'output, che prima era sostanzialmente l'uscita del programma una volta interagito con i dati, ora è l'ingresso, insieme ai dati che producono il programma stesso.

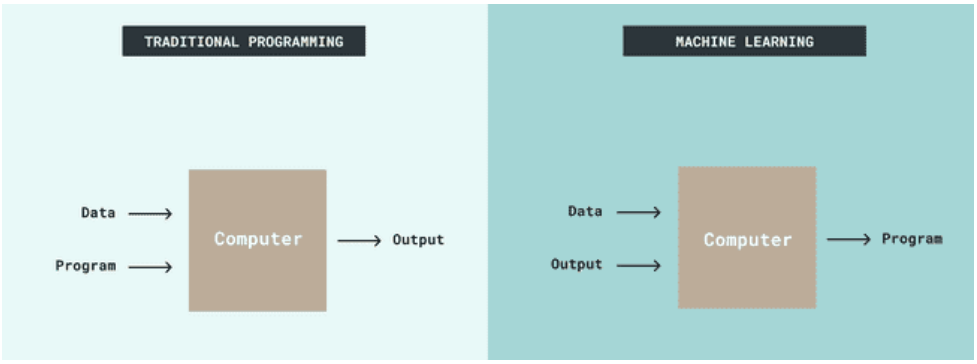



*Figura 5.2 Programmazione e ML*

I sistemi Machine Learning hanno quattro tipi di output differenti: Regression, utilizzato per predire valori; Classification, assegna le label ai dati; Clustering, suddivide un insieme di dati simili in sottocategorie; Sequence prediction, si basa su un valore precedente per predirne il successivo. Anche se non ce ne accorgiamo, durante la navigazione su siti e-commerce, spulciando tra i prodotti e leggendo le recensioni, stiamo fornendo dati ad un modello di ML, infatti tutti i suggerimenti di prodotti che potrebbero interessarci sono frutto di quell'analisi di informazioni, basate sulla nostra ricerca, che hanno prodotto dei risultati tramite l'utilizzo di reti neurali. Questo concetto appena espresso,

dà esito ad un ulteriore suddivisione del Machine Learning, ovvero nella sua variante *online*, la quale si adatta continuamente ai dati forniti in ingresso, e quella *offline*, che una volta allenata e messa in funzione, applica solamente ciò che ha imparato, senza potersi migliorare a seguito dell'utilizzo (offline machine learning models). In entrambi i casi la precisione delle previsioni e le interpretazioni, da parte del modello neurale, sono unicamente conseguenza dell'allenamento della rete stessa e della sua automazione; per questo motivo, le reti che si affidano al modello online sono più efficienti, essendo più versatili e sempre aggiornate. Negli ultimi anni vi è stato un cospicuo aumento di investimenti orientati al ML, a seguito di un interesse generale, questo è accaduto per molteplici fattori: processori moderni dalle capacità elaborative molto superiori al passato, la capacità di storage di grandi quantità di dati è divenuta meno costosa e più veloce, il ML è anche divenuto Open Source per molti aspetti e in ultimo, ma non meno importante, l'analisi dei risultati è diventata di più facile interpretazione tanto da non essere una prerogativa dei soli ricercatori. Come già accennato, la precisione del modello di Machine Learning è tanto più ottima tanto più sono i dati forniti, per questo motivo si introduce il concetto di *Big data*, cioè grandi volumi di dati provenienti da innumerevoli fonti differenti e disponibili con grande velocità, che quindi rappresentano il caso reale di applicazione. Da questo punto di vista il Cloud ha dato una grandissima mano; infatti, la velocità e la quantità di dati sono garantiti, inoltre offre la possibilità di riservare sforzo elaborativo di calcolo senza dover avere fisicamente gli strumenti per farlo, come una gran quantità di GPUs (Graphics Processing Units). Nella creazione di un modello di apprendimento, oltre al ML, rivestono un ruolo molto importante anche la statistica e il *Data Mining*; con quest'ultimo si intende il processo di esplorazione ed elaborazione dei Big Data, al fine di trovare dei pattern comuni. In sintesi, lo scopo di questa operazione è spiegare, interpretare i dati e suddividerli in gruppi senza operare alcuna previsione, ma fornendo esclusivamente il terreno fertile per poi poter rimandare questo compito a terzi.

## 5.2 Edge Impulse

Edge Impulse è una piattaforma online per la creazione di embedded device intelligenti, tramite l'utilizzo del Machine Learning Supervisionato (provvisto di label) e basato sulla metodologia di apprendimento offline. Le moderne tecnologie hanno portato a grandi passi in avanti sull'architettura dei microprocessori e sulla stesura di più efficienti algoritmi, e queste novità hanno permesso di fare operare sistemi abbastanza sofisticati di ML sui microcontrollori a minor capacità elaborativa. Questo ha portato alla nascita dell'Embedded Machine Learning, noto anche come TinyML, i cui vantaggi principali possono essere riassunti come segue:

- Larghezza di banda, gli algoritmi di ML estraggono importanti informazioni tramite gli embedded device evitando problemi di limitazione di banda;
- Latenza, notevolmente ridotta in quanto dipendente unicamente dal tempo di risposta dei mini-device e non dalla velocità della rete network;
- Economicità, evitando di trasmettere i dati al Cloud per poi essere processati, ma facendolo esclusivamente alla fonte si ha un risparmio nei costi;
- Affidabilità, vi è un'indipendenza dalla connessione al Cloud e quindi, sono di fatto più affidabili nel funzionamento;
- Privacy, evitando l'invio dei dati al Cloud si garantisce una maggior protezione delle informazioni.

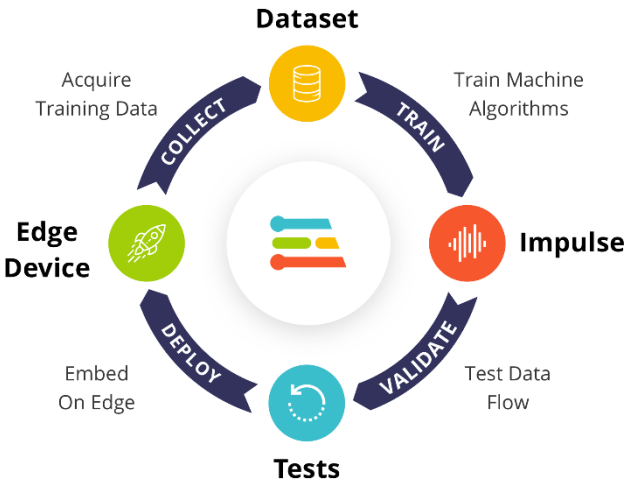



*Figura 5.3 Schema d'implementazione ML*

Edge Impulse (Figura 5.3) supporta una grande quantità di schede, che spaziano dai prodotti della Nordic Semiconductor, alla Raspberry fino alla casa di Arduino, tra i quali proprio la scheda di interesse: l'Arduino Nano 33 BLE Sense. Una volta registratisi sul sito, effettuato l'accesso e creato un nuovo progetto, bisogna collegare la Board con il

firmware aggiornato (insieme ad altri device utilizzati per la raccolta dei dati, come gli smartphones) al progetto, per mezzo del pulsante “Connect a new device”.

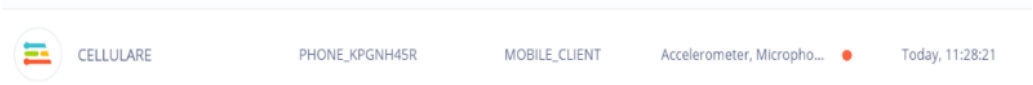



*Figura 5.4 dispositivo di registrazione*

## 5.2.1 Riconoscimento Vocale

Lo scopo di questo progetto è lo svolgimento di funzionalità di sensing ambientale da parte dell’Arduino tramite interazione vocale. Si richiede la quantità di luce presente nella stanza per mezzo della parola “luminosità”, campionata con l’ausilio del microfono PDM integrato e con il supporto del sensore APDS. La risposta, tramite seriale, avviene solamente dopo che la rete neurale avrà elaborato il segnale vocale in ingresso.

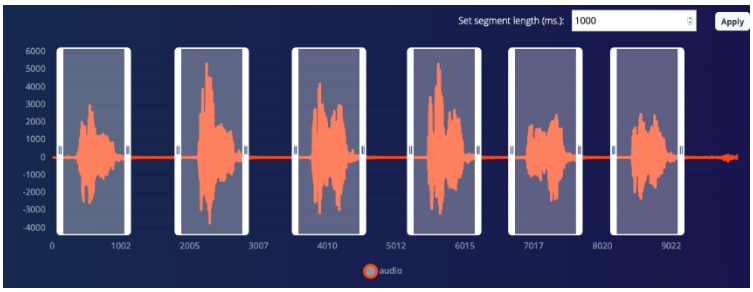



*Figura 5.5 Campioni audio registrati*

Come primo step, nella sezione **Data acquisition**, una volta collegato l’Arduino, si sceglie il sensore da utilizzare (microfono), si stabilisce la frequenza di campionamento (16 kHz) e la finestra di registrazione (10 secondi). Infine, essendo un ML supervisionato, si inserisce il nome della label che ci si appresta a registrare e si avvia con il comando: “start sampling”. I campioni di registrazione, intervallati da piccole pause, vanno opportunamente filtrati da finestre, di durata di un secondo, posizionate durante la pronuncia della parola (“split sample”) come in Figura 5.5. Collezionata una quantità sufficiente di dati, bisogna aggiungere, per una migliore prestazione del prodotto ML, dei segnali di rumore e dei segnali di parole a caso, diverse dalla keyword, in modo tale da permettere al sistema la distinzione tra le diverse fonti audio. Una volta raccolti i dati, si dovrebbe avere una situazione simile a quella mostrata in Figura 5.6. Prima di procedere è ben effettuare un ribilanciamento dei campioni raccolti, tramite il pulsante

“Rebalance dataset” nella sezione della DashBoard della piattaforma. Questa procedura permetterà di suddividere i dati in due gruppi: training data (80%) e test data (20%).

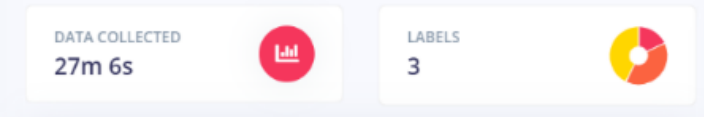



*Figura 5.6 Dataset*

I primi serviranno essenzialmente ad allenare la rete neurale, i secondi a testare i risultati di questa operazione, in pratica a determinare l’accuratezza della rete stessa e verranno utilizzati solamente alla fine del processo. La successiva operazione, l’**Impulse design**, si occupa di elaborare il segnale tramite tecniche di *signal processing* (MFCC) e di creare un blocco neurale (Keras) per catalogare i dati, cioè per distinguere i tre diversi insiemi di sorgenti vocali. L’**MFCC** mostra per ogni singolo campione il suo spettrogramma; in realtà si tratta di un particolare spettrogramma che mette in risalto le frequenze comuni della voce umana. Prima di cominciare ad allenare la rete neurale, bisogna produrre il blocco MFCC il quale mostra una visione 3D dell’intero dataset raccolto differenziato per colori, che rappresentano le diverse label di appartenenza (Figura 5.7). La rappresentazione grafica 3D è interattiva: ci si può spostare e aumentare lo zoom, inoltre cliccando su ogni singolo campione lo si può riascoltare. Questa rappresentazione consente di evidenziare la differenza tra i vari campioni raccolti in base alla posizione.

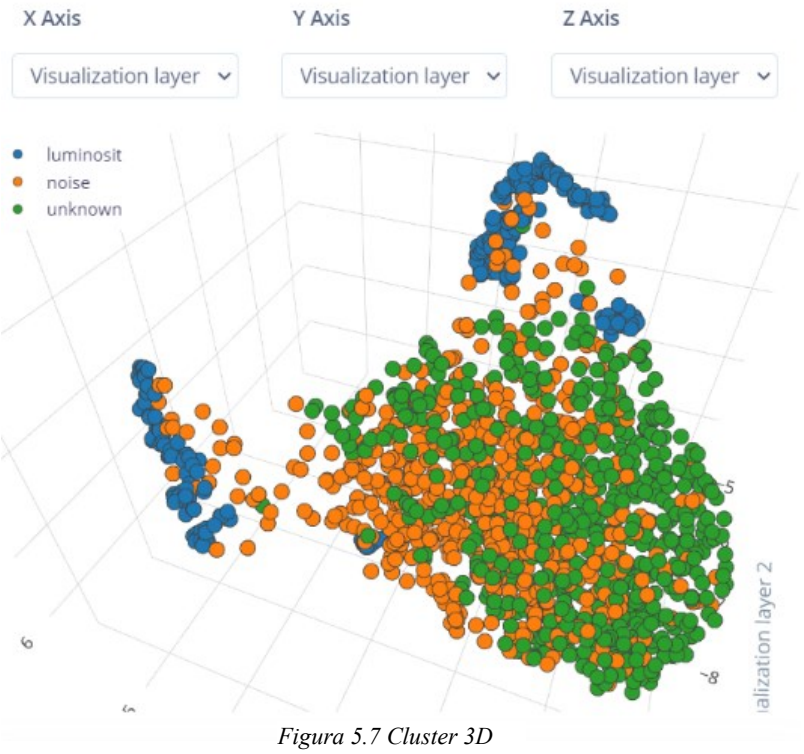



*Figura 5.7 Cluster 3D*

L’**NN Classifier** è il luogo dove inizia il vero e proprio allenamento della rete neurale. Tramite cento cicli di training e quattro layer intermedi, la rete inizierà ad imparare a seguito di esecuzioni di algoritmi matematici. È importante sottolineare che eventuali nuovi campioni da fornire alla rete vengono unicamente assimilati solo dopo aver riallenato la rete (**Retrain model**). Il risultato dell’apprendimento produce una visualizzazione 3D simile alla precedente, dove in rosso sono evidenziati i campioni interpretati in modo errato, in verde quelli giusti (Figura 5.8).

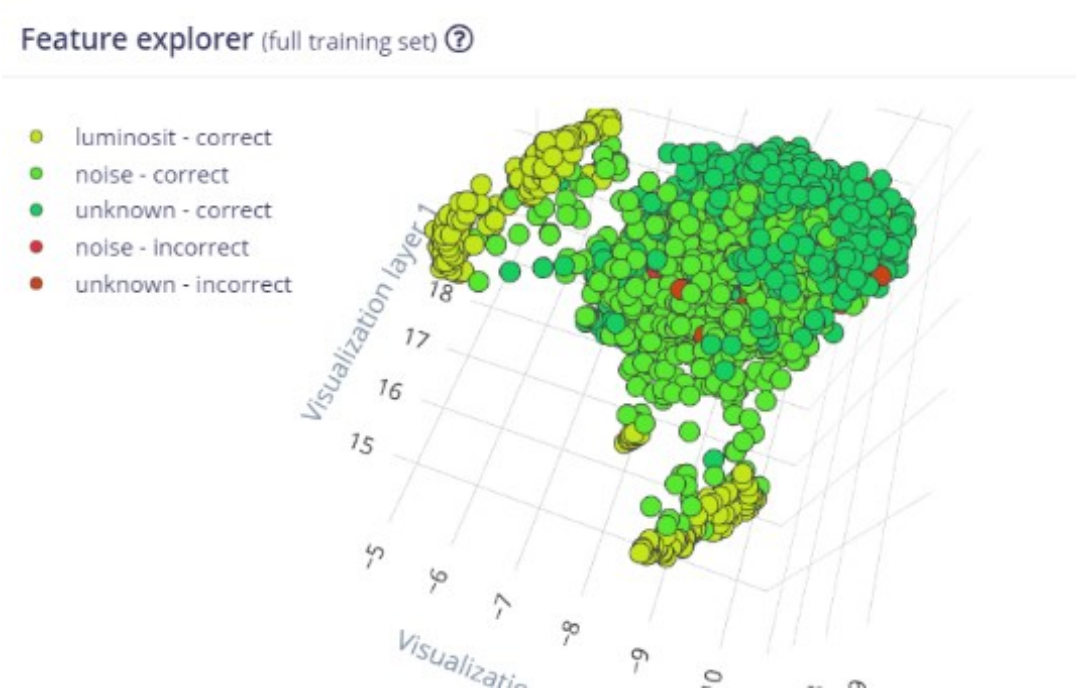



*Figura 5.8 Rappresentazione 3D degli errori di classificazione*

In più è fornita una valutazione sull’efficienza della rete neurale stessa, cioè la capacità di classificare correttamente i campioni, espressa in % tramite tabella (Figura 5.9).
L’ultimo step da eseguire è il **Model testing**, cioè quell’operazione che mostra la vera precisione del sistema neurale creato: vengono forniti alla rete i dati della sezione “test”, cioè dei campioni audio mai visti prima che devono essere correttamente catalogati. A seguito di questa verifica viene fornita una misura più veritiera delle capacità del sistema creato (Figura 5.10).


*Figura 5.9 Precisione di interpretazione della rete*

Inoltre, vengono anche mostrate le performance sul device embedded, suddivise in utilizzo della RAM e tempo di deduzione della rete. Per il **deployment** sulla scheda Arduino, nell'omonima sezione si dà la possibilità di scaricare il codice sotto forma di libreria C++, assembly o libreria Arduino. Ovviamente sarà l'ultimo caso quello di interesse.


*Figura 5.10 Test finale della rete neurale*

La libreria andrà aggiunta all'IDE Arduino e il programma completo sarà disponibile tra gli esempi, come segue.

```
// If your target is limited in memory remove this macro to save 10K RAM
#define EIDSP_QUANTIZE_FILTERBANK   0

/**
 * Define the number of slices per model window. E.g. a model window of 1000 ms
 * with slices per model window set to 4. Results in a slice size of 250 ms.
 * For more info: https://docs.edgeimpulse.com/docs/continuous-audio-sampling
 */
#define EI_CLASSIFIER_SLICES_PER_MODEL_WINDOW 3

/* Includes ---------------------------------------------------------------- */
#include <PDM.h>
#include <arduino_nano_33_sense_-_riconoscimento_vocale_inference.h>
#include <Arduino_APDS9960.h>

/** Audio buffers, pointers and selectors */
typedef struct {
    signed short *buffers[2];
    unsigned char buf_select;
    unsigned char buf_ready;
    unsigned int buf_count;
    unsigned int n_samples;
} inference_t;

static inference_t inference;
static bool record_ready = false;
static signed short *sampleBuffer;
static bool debug_nn = false; // Set this to true to see e.g. features generated from the raw signal
static int print_results = -(EI_CLASSIFIER_SLICES_PER_MODEL_WINDOW);
void setup()
{
    // put your setup code here, to run once:
    Serial.begin(115200);
     if (!APDS.begin()) {
    Serial.println("Error initializing APDS9960 sensor.");
    }

    Serial.println("Edge Impulse Inferencing Demo");

    // summary of inferencing settings (from model_metadata.h)
    ei_printf("Inferencing settings:\n");
    ei_printf("\tInterval: %.2f ms.\n", (float)EI_CLASSIFIER_INTERVAL_MS);
    ei_printf("\tFrame size: %d\n", EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE);
    ei_printf("\tSample length: %d ms.\n", EI_CLASSIFIER_RAW_SAMPLE_COUNT / 16);
    ei_printf("\tNo. of classes: %d\n", sizeof(ei_classifier_inferencing_categories) /
                                        sizeof(ei_classifier_inferencing_categories[0]));

    run_classifier_init();
    if (microphone_inference_start(EI_CLASSIFIER_SLICE_SIZE) == false) {
        ei_printf("ERR: Failed to setup audio sampling\r\n");
        return;
    }
}

/**
 * @brief      Arduino main function. Runs the inferencing loop.
 */
void loop()
{
    bool m = microphone_inference_record();
    if (!m) {
        ei_printf("ERR: Failed to record audio...\n");
        return;
    }

    signal_t signal;
    signal.total_length = EI_CLASSIFIER_SLICE_SIZE;
    signal.get_data = &microphone_audio_signal_get_data;
    ei_impulse_result_t result = {0};
```

```
    EI_IMPULSE_ERROR r = run_classifier_continuous(&signal, &result, debug_nn);
    if (r != EI_IMPULSE_OK) {
        ei_printf("ERR: Failed to run classifier (%d)\n", r);
        return;
    }

    if(result.classification[0].value > 0.7){
        // check if a color reading is available
        while (! APDS.colorAvailable()) {
          delay(5);
        }
        int r, g, b, a;

        // read the color
        APDS.readColor(r, g, b, a);
       Serial.print("a = ");
       Serial.println(a);

         }

    if (++print_results >= (EI_CLASSIFIER_SLICES_PER_MODEL_WINDOW)) {
        // print the predictions

//          ei_printf("Predictions ");
//          ei_printf("(DSP: %d ms., Classification: %d ms., Anomaly: %d ms.)",
//            result.timing.dsp, result.timing.classification, result.timing.anomaly);
//        ei_printf(": \n");
//      for (size_t ix = 0; ix < EI_CLASSIFIER_LABEL_COUNT; ix++) {
//          ei_printf("    %s: %.5f\n", result.classification[ix].label,
//                    result.classification[ix].value);
//      }
#if EI_CLASSIFIER_HAS_ANOMALY == 1
        ei_printf("    anomaly score: %.3f\n", result.anomaly);
#endif

        print_results = 0;
    }
}

void ei_printf(const char *format, ...) {
    static char print_buf[1024] = { 0 };

    va_list args;
    va_start(args, format);
    int r = vsnprintf(print_buf, sizeof(print_buf), format, args);


    va_end(args);

    if (r > 0) {
        Serial.write(print_buf);
    }
}

/**
 * @brief      PDM buffer full callback
 *             Get data and call audio thread callback
 */
static void pdm_data_ready_inference_callback(void)
{
    int bytesAvailable = PDM.available();

    // read into the sample buffer
    int bytesRead = PDM.read((char *)&sampleBuffer[0], bytesAvailable);

    if (record_ready == true) {
        for (int i = 0; i<bytesRead>> 1; i++) {
            inference.buffers[inference.buf_select][inference.buf_count++] = sampleBuffer[i];

            if (inference.buf_count >= inference.n_samples) {
                inference.buf_select ^= 1;
                inference.buf_count = 0;
                inference.buf_ready = 1;
            }
        }
    }
}
```

```
static bool microphone_inference_start(uint32_t n_samples)
{
    inference.buffers[0] = (signed short *)malloc(n_samples * sizeof(signed short));

    if (inference.buffers[0] == NULL) {
        return false;
    }

    inference.buffers[1] = (signed short *)malloc(n_samples * sizeof(signed short));

    if (inference.buffers[0] == NULL) {
        free(inference.buffers[0]);
        return false;
    }

    sampleBuffer = (signed short *)malloc((n_samples >> 1) * sizeof(signed short));

    if (sampleBuffer == NULL) {
        free(inference.buffers[0]);
        free(inference.buffers[1]);
        return false;
    }

    inference.buf_select = 0;
    inference.buf_count = 0;
    inference.n_samples = n_samples;
    inference.buf_ready = 0;

    // configure the data receive callback
    PDM.onReceive(&pdm_data_ready_inference_callback);
    // optionally set the gain, defaults to 20
    PDM.setGain(80);
    PDM.setBufferSize((n_samples >> 1) * sizeof(int16_t));
    if (!PDM.begin(1, EI_CLASSIFIER_FREQUENCY)) {
        ei_printf("Failed to start PDM!");
    }
    record_ready = true;
    return true;
}

static bool microphone_inference_record(void)
{
    bool ret = true;
    if (inference.buf_ready == 1) {
        ei_printf(
            "Error sample buffer overrun. Decrease the number of slices per model window "
            "(EI_CLASSIFIER_SLICES_PER_MODEL_WINDOW)\n");
        ret = false;
    }
    while (inference.buf_ready == 0) {
        delay(1);
    }
    inference.buf_ready = 0;
    return ret;
}
        delay(1);
    }
    inference.buf_ready = 0;
    return ret;
}

/**
 * Get raw audio signal data
 */
static int microphone_audio_signal_get_data(size_t offset, size_t length, float *out_ptr)
{
    numpy::int16_to_float(&inference.buffers[inference.buf_select ^ 1][offset], out_ptr, length);

    return 0;
}

/**
 * @brief      Stop PDM and release buffers
 */
static void microphone_inference_end(void)
{
    PDM.end();
    free(inference.buffers[0]);
    free(inference.buffers[1]);
    free(sampleBuffer);
}

#if !defined(EI_CLASSIFIER_SENSOR) || EI_CLASSIFIER_SENSOR != EI_CLASSIFIER_SENSOR_MICROPHONE
#error "Invalid model for current sensor."
#endif
```

Riga di codice particolarmente interessante è quella, compresa all'interno del loop, che recita come segue: "*if(result.classification[0].value > 0.7*". Questa indica con quale certezza espressa dalla rete bisogna assegnare quella decisione come vera: la soglia è fissata a 0.7, in cui 1 è la sicurezza massima nel riconoscimento della parola, mentre 0 l'opposto. Quindi, nel caso preso in considerazione, verranno riconosciute come parole chiavi solo quelle espressioni vocali in cui la rete dà una corrispondenza oltre il 70%. Fatta questa considerazione, è facile dedurre come lo spostamento della soglia vada, in un certo modo, ad influenzare le performance della rete stessa. Proprio per questo motivo di seguito è indicata una tabella in cui si descrive il comportamento della rete con soglie differenti e con keyword uguali e simili a quella corretta. Per ogni parola vengono effettuati venti tentativi e si ha una classificazione in base a come la rete riconosce il segnale vocale.

*Tabella: test della rete neurale per il riconoscimento di keywords*

| **soglia 0,5** | **parola** | **riconosciuta** |
|---|---|---|
| **Parola test** | luminosità | sconosciuta |
| luminosità | **18** | **2** |
| velleità | **10** | **10** |
| rumorosità | **15** | **5** |
| **soglia 0,7** | | |
| luminosità | **20** | **0** |
| velleità | **3** | **17** |
| rumorosità | **9** | **11** |
| **soglia 0,9** | | |
| luminosità | **16** | **4** |
| velleità | **6** | **14** |
| rumorosità | **2** | **18** |

Dalla tabella è facile notare come una soglia posta a "0,5" sia la peggiore delle impostazioni; mentre, confrontando le ultime due, si è spinti ad adottare la soglia più conservativa: quella intermedia tra tutte quelle prese in considerazione, ovvero "0,7".

Lo step successivo è di constatare come l'Arduino e la rete neurale si comportino aggiungendo un'altra parola chiave: "gradi". Quindi, oltre il sensore APDS viene richiamato anche l'HTS221 per recuperare il valore numerico della temperatura. Come già fatto si raccoglie una quantità di keyword sufficiente, da dover essere aggiunte a quelle precedenti sulla piattaforma Edge Impulse (Figura 5.11).

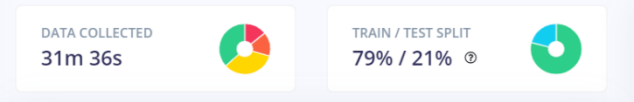



*Figura 5.11 Dataset*

Successivamente, si osserva come avviene la distribuzione grafica delle diverse label: parole sconosciute, rumore, luminosità e gradi (Figura 5.12a-b).

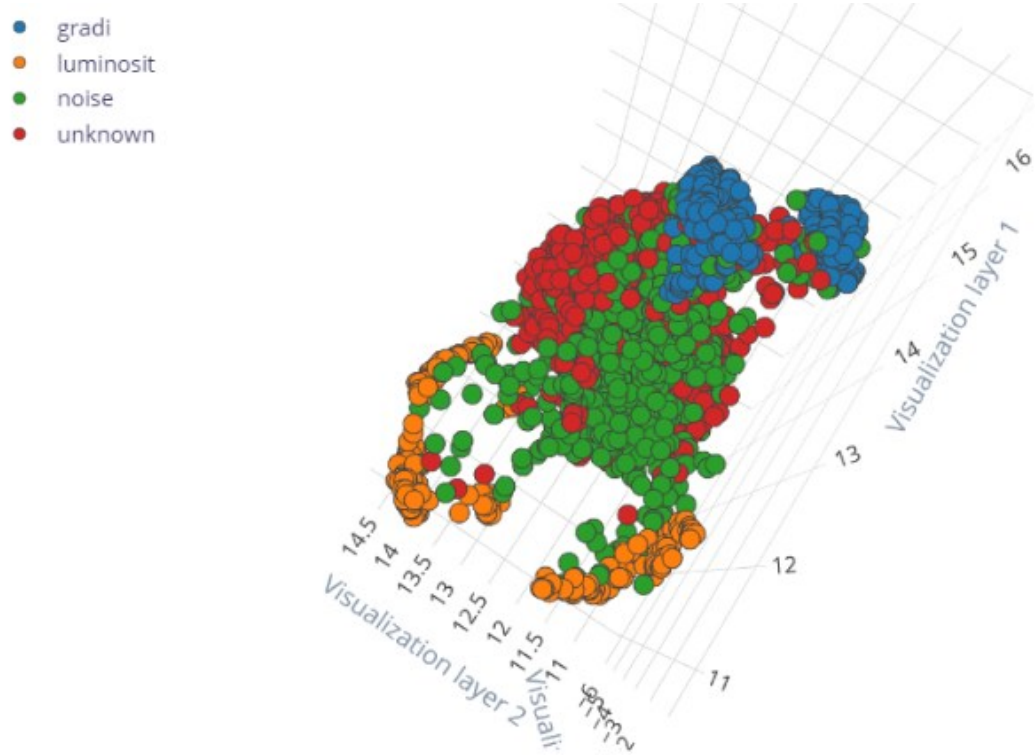



*Figura 5.12a Cluster 3D*

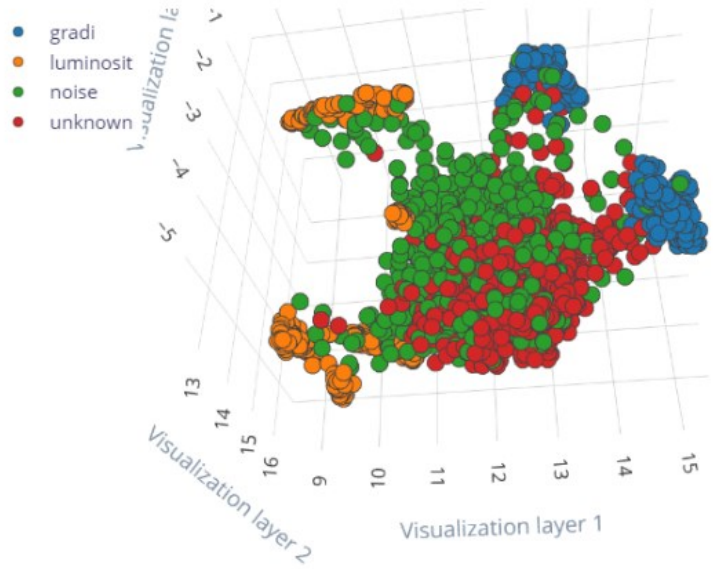



*Figura 5.12b Cluster 3D*

Infine, la rete neurale viene riallenata con i nuovi campioni (Figura 5.13) e testata con i "test data", con indicata relativa precisione (Figura 5.14).

**Confusion matrix** (validation set)

| | GRADI | LUMINOSIT | NOISE | UNKNOWN |
|---|---|---|---|---|
| GRADI | **96.6%** | 0% | 0% | 3.4% |
| LUMINOSIT | 1.8% | **96.4%** | 0% | 1.8% |
| NOISE | 0.8% | 1.6% | **87.4%** | 10.2% |
| UNKNOWN | 2.2% | 0% | 8.0% | **89.9%** |
| F1 SCORE | 0.94 | 0.96 | 0.89 | 0.89 |

**Feature explorer** (full training set)

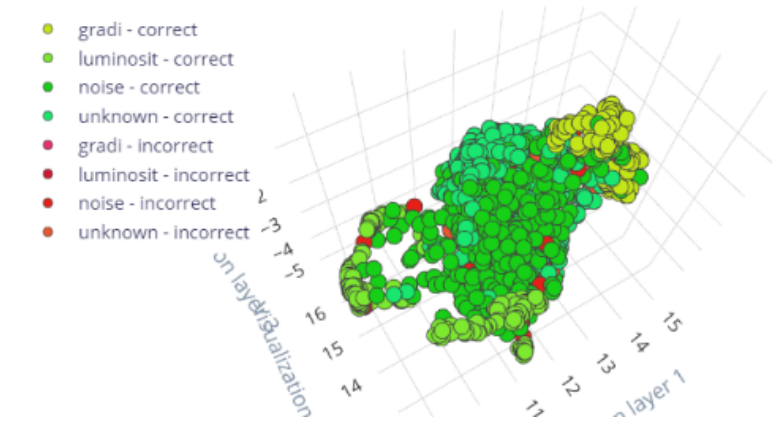
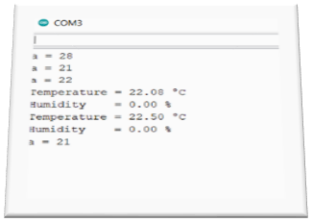



*Figura 5.13 Rappresentazione 3D degli errori di classificazione e relativa precisione percentuale*


*Figura 5.14 Precisione della rete*

Purtroppo, nonostante il modello di Machine Learning abbia delle ottime prestazioni, una volta implementato il codice, con la stessa procedura del caso precedente, sull'Arduino Nano si hanno prestazioni fortemente degradate tanto da non permettere un'adeguata analisi delle prestazioni, a causa di limiti strutturali della Board stessa. Di seguito viene mostrato il codice implementato mostrando esclusivamente la parte più significativa, escludendo l'inizializzazione dei sensori, cioè quella relativa alla soglia delle due keyword. Infine, il riscontro visivo tramite seriale del funzionamento dell'embedded ML.

```
//gradi
if(result.classification[0].value > 0.7){
    float temperature = HTS.readTemperature();
    float humidity    = HTS.readHumidity();

    // print each of the sensor values
    Serial.print("Temperature = ");
    Serial.print(temperature);
    Serial.println(" °C");

    Serial.print("Humidity    = ");
    Serial.print(humidity);
    Serial.println(" %");
}

//luminosità
if(result.classification[1].value > 0.7){
    // check if a color reading is available
    while (! APDS.colorAvailable()) {
      delay(5);
    }
    int r, g, b, a;

    // read the color
    APDS.readColor(r, g, b, a);

    // print the values
    Serial.print("a = ");
    Serial.println(a);
     }
```



## 5.2.2 Riconoscimento Movimenti

Scopo del progetto in questione è quello, tramite l'utilizzo dell IMU, di riconoscere l'esecuzione di due gesti da parte della Board e dell'assegnazione ad "anomalia" qualsiasi altro movimento che non rientri nelle due label. Come al solito, si procede a collegare fisicamente la Board e a raccogliere i dati, selezionando come sensore quello dell'accelerometro. Dopo avere raccolto i vari campioni di informazione, tramite movimenti verticali e orizzantali opportunamente catalogati, ci si ritrova nella condizione di Figura 5.15.

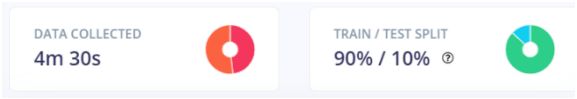



*Figura 5.15 Dataset*

Quindi, come anticipato, oltre alla classica rete neurale (Keras) e al blocco di elaborazione spettrale, si aggiunge un'ulteriore rete neurale per il rilevamento delle anomalie: **K-means Anomaly Detection**. La disposizione dei campioni raccolti appare decisamente ben distribuita e distinguibile (Figura 5.16). Si procede ad allenare la rete con i campioni disponibili, il cui risultato è di una precisione pari all'intero, con prestazioni su device che implicano un'occupazione massima della RAM di 1,5 kB e un tempo di deduzione pari a 1ms.

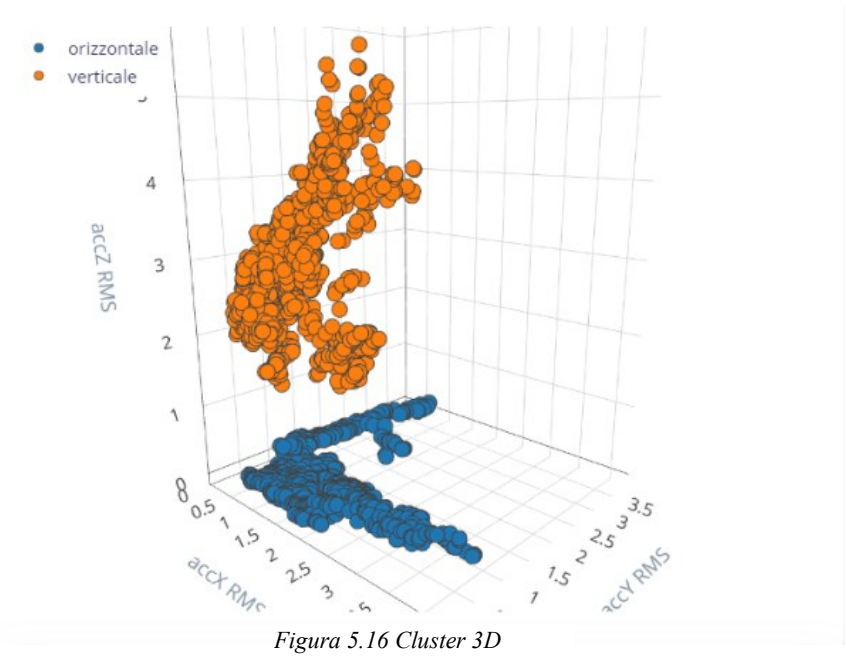
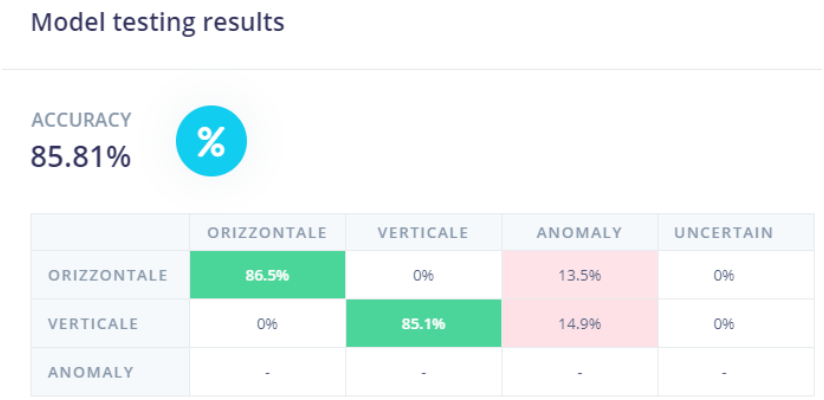



*Figura 5.16 Cluster 3D*

La condizione di perfetta accuratezza del modello può far pensare ad una situazione di overfitting, infatti nella prova test della rete le performance sono ridotte, ma comunque rimangono significativamente positive (Figura 5.17).




| | ORIZZONTALE | VERTICALE | ANOMALY | UNCERTAIN |
|---|---|---|---|---|
| ORIZZONTALE | 86.5% | 0% | 13.5% | 0% |
| VERTICALE | 0% | 85.1% | 14.9% | 0% |
| ANOMALY | - | - | - | - |

*Figura 5.17 Test rete neurale*

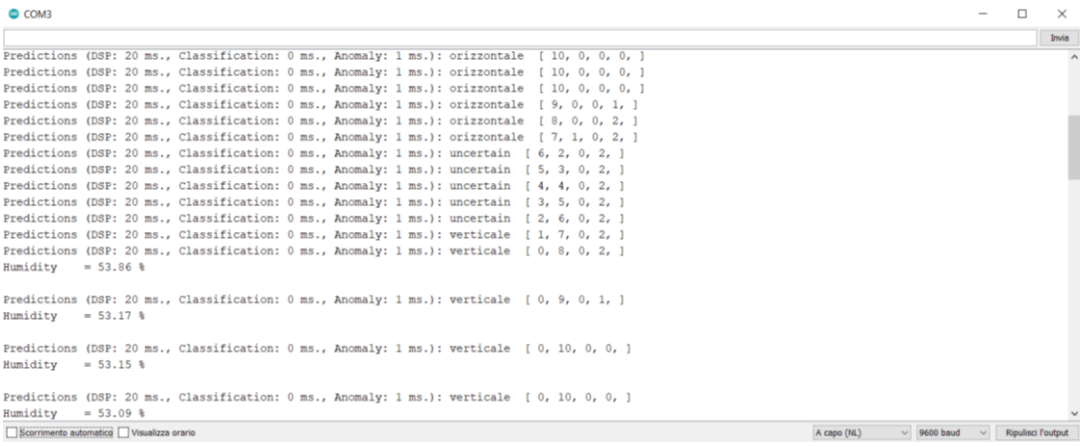
Ad ogni modo se si volesse osservare il comportamente della rete con nuovi dati raccolti sul momento, basta recarsi nella sezione **Live classification**.
Il codice sottostante riporta lo sviluppo del progetto, nel quale ad un movimento orizzontale è associata l'accensione del LED integrato, mentre a quello verticale la lettura del sensore HTS221 per ottenere il valore dell'umidità. In entrambi i casi il riscontro visivo avviene con il supporto del monitor seriale.

```
#include <ARDUINO_NANO_33_SENSE_-_RICONOSCIMENTO_GESTI_inferencing.h>
#include <Arduino_LSM9DS1.h>
#include <Arduino_HTS221.h>
/* Constant defines ------------------------------------------------------- */
#define CONVERT_G_TO_MS2    9.80665f

/* Private variables ------------------------------------------------------- */
static bool debug_nn = false; // Set this to true to see e.g. features generated from the raw signal
static uint32_t run_inference_every_ms = 200;
static rtos::Thread inference_thread(osPriorityLow);
static float buffer[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE] = { 0 };
static float inference_buffer[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE];

/* Forward declaration */
void run_inference_background();

/**
* @brief      Arduino setup function
*/
void setup()
{
    // put your setup code here, to run once:
    Serial.begin(115200);
    Serial.println("Edge Impulse Inferencing Demo");
    pinMode(13, OUTPUT);

    if (!IMU.begin()) {
        ei_printf("Failed to initialize IMU!\r\n");
    }
    else {
```

```
        ei_printf("IMU initialized\r\n");
    }

    if (!HTS.begin()) {
    Serial.println("Failed to initialize humidity temperature sensor!");
    while (1);
    }

    if (EI_CLASSIFIER_RAW_SAMPLES_PER_FRAME != 3) {
        ei_printf("ERR: EI_CLASSIFIER_RAW_SAMPLES_PER_FRAME should be equal to 3 (the 3 sensor axes)\n");
        return;
    }

    inference_thread.start(mbed::callback(&run_inference_background));
}

void ei_printf(const char *format, ...) {
   static char print_buf[1024] = { 0 };

   va_list args;
   va_start(args, format);
   int r = vsnprintf(print_buf, sizeof(print_buf), format, args);
   va_end(args);

   if (r > 0) {
      Serial.write(print_buf);
   }
}

void run_inference_background()
{
    // wait until we have a full buffer
    delay((EI_CLASSIFIER_INTERVAL_MS * EI_CLASSIFIER_RAW_SAMPLE_COUNT) + 100);

    // This is a structure that smoothens the output result
    // With the default settings 70% of readings should be the same before classifying.
    ei_classifier_smooth_t smooth;
    ei_classifier_smooth_init(&smooth, 10 /* no. of readings */, 7 /* min. readings the same */, 0.8 /* min. confidence */, 0.3 /* max anomaly */)

    while (1) {
        // copy the buffer
        memcpy(inference_buffer, buffer, EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE * sizeof(float));

        // Turn the raw buffer in a signal which we can the classify
        signal_t signal;
        int err = numpy::signal_from_buffer(inference_buffer, EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE, &signal);
        if (err != 0) {
            ei_printf("Failed to create signal from buffer (%d)\n", err);
            return;
        }

        // Run the classifier
        ei_impulse_result_t result = { 0 };

        err = run_classifier(&signal, &result, debug_nn);
        if (err != EI_IMPULSE_OK) {
            ei_printf("ERR: Failed to run classifier (%d)\n", err);
            return;
        }

        //led
        if(smooth.count[0]>7){
          digitalWrite(13, HIGH);
        }else{
          digitalWrite(13, LOW);
        }

        //humidity
        if(smooth.count[1]>7){
          float humidity    = HTS.readHumidity();
          Serial.print("Humidity    = ");
          Serial.print(humidity);
          Serial.println(" %\n");
         }

        // print the predictions
        ei_printf("Predictions ");
        ei_printf("(DSP: %d ms., Classification: %d ms., Anomaly: %d ms.)",
            result.timing.dsp, result.timing.classification, result.timing.anomaly);
        ei_printf(": ");

        // ei_classifier_smooth_update yields the predicted label
        const char *prediction = ei_classifier_smooth_update(&smooth, &result);
        ei_printf("%s ", prediction);
        // print the cumulative results
        ei_printf(" [ ");
        for (size_t ix = 0; ix < smooth.count_size; ix++) {
            ei_printf("%u", smooth.count[ix]);
            if (ix != smooth.count_size + 1) {
                ei_printf(", ");
```

```
            }
            else {
              ei_printf(" ");
            }        }
        ei_printf("]\n");
        delay(run_inference_every_ms);    }
    ei_classifier_smooth_free(&smooth);}
void loop()
{
    while (1) {
        // Determine the next tick (and then sleep later)
        uint64_t next_tick = micros() + (EI_CLASSIFIER_INTERVAL_MS * 1000);
        // roll the buffer -3 points so we can overwrite the last one
        numpy::roll(buffer, EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE, -3);
        // read to the end of the buffer
        IMU.readAcceleration(
            buffer[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE - 3],
            buffer[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE - 2],
            buffer[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE - 1]
        );
        buffer[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE - 3] *= CONVERT_G_TO_MS2;
        buffer[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE - 2] *= CONVERT_G_TO_MS2;
        buffer[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE - 1] *= CONVERT_G_TO_MS2;
        // and wait for next tick
        uint64_t time_to_wait = next_tick - micros();
        delay((int)floor((float)time_to_wait / 1000.0f));
        delayMicroseconds(time_to_wait % 1000);    }}
#if !defined(EI_CLASSIFIER_SENSOR) || EI_CLASSIFIER_SENSOR != EI_CLASSIFIER_SENSOR_ACCELEROMETER
#error "Invalid model for current sensor"
#endif
```

Sul monitor seriale si ottiene:




## 5.2.3 Considerazioni Finali

L'utilizzo dell'Arduino Nano 33 BLE Sense come strumento di Embedded Machine Learning è un'importante innovazione e passo in avanti della tecnologia. Infatti, permette una vasta gamma di applicazioni al fine di migliorare la qualità della vita. Nel presente lavoro di tesi si sono in particolare implementate delle modalità utili ad agevolare l'interazione dell'uomo con uno scenario di tipo smart home, interrogando i sensori ambientali presenti sulla board mediante l'interazione vocale.

# Bibliografia


*[1] Jie Cao Quan Zhang Weisong Shi, "Edge Computing: A Primer," 2018, Springer.*

*[2] S.Spinsante, "Sensori e Trasduttori", 2020, corso di Laurea Magistrale Ingegneria Elettroinca UNIVPM*

*[3] Judith Hurwitz Daniel Kirsch, "Machine Learning for dummies,"2018, IBM*

*[4] UART vs I2C vs SPI – Communication Protocols and Uses - Lastest Open Tech From Seeed (seeedstudio.com)*

*[5] Protocolli di comunicazione su SBC: UART, I2C, SPI - Moreware Blog*

*[6] USART vs UART: Know the difference - EDN*

*[7] Arduino Nano 33 BLE Sense con intestazioni — Arduino Official Store*

*[8] https://www.youtube.com/playlist?list=PL9_01HM23dGEDNNfR6BtlDWD8DDoAcLOT*

*[9] https://www.alberti-porro.edu.it/wordpress/wp-content/uploads/2014/01/ProgrammareArduino.pdf*

*[10]Bluetooth Technology Overview | Bluetooth® Technology Website*

*[11]https://forum.arduino.cc/t/ble-very-weak-signal/631751/21*

*[12]Nordic Semiconductor | Specialists in Low Power Wireless - nordicsemi.com*

*[13]*https://www.edgeimpulse.com/

*[14]Edge Impulse with the Nano 33 BLE Sense | Arduino Documentation | Arduino Documentation*

*[15]Wikipedia*