UNIVERSITÀ
DEGLI STUDI
DI BERGAMO

Dipartimento di
INGEGNERIA GESTIONALE, DELL'INFORMAZIONE E
DELLA PRODUZIONE

Corso di laurea in
Ingegneria Informatica

Progetto di Sensori

Classificazione di movimenti tramite Arduino Nano 33 BLE

Progetto svolto da:
Filippo Bordogna
(1058427)
Paolo Olivieri
(1057959)
Gabriele Morè
(1058401)

Anno Accademico
2023/2024

Indice

Elenco delle figure

Elenco dei listati

1 | Introduzione

Il progetto da noi svolto si propone di sfruttare l'*Arduino Nano 33 BLE* per sviluppare un sistema in grado di classificare movimenti del corpo umano in ambito fitness, in particolare di distinguere un esercizio per il bicipite da uno per le spalle.

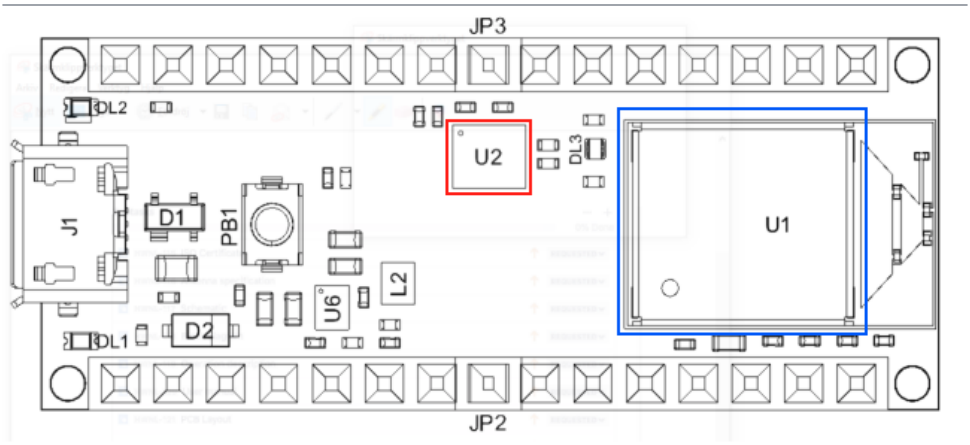
Al fine del raggiungimento dell'obbiettivo sopra citato, sono stati impiegati i dati provenienti dall'accelerometro, dal giroscopio e dal magnetometro integrati nel microcontrollore indossabile. Questi sensori consentono di rilevare, rispettivamente, le variazioni di accelerazione, orientamento e campo magnetico, fornendo così informazioni dettagliate sui movimenti eseguiti dall'individuo.
Il trasferimento dei dati avviene tramite *Bluetooth Low Energy (BLE)* verso una base station esterna che ha il compito di elaborare i dati e classificare i movimenti [2].

2 | Arduino

Come anticipato nell'introduzione, è stato utilizzato il microcontrollore Arduino Nano 33 BLE di cui ne vengono sfruttati i 3 sensori integrati nel modulo IMU (Figura 2.1 in rosso):

- Accelerometro: misura le accelerazioni lineari proprie;
- Giroscopio: misura le velocità angolari;
- Magnetometro: misura il campo magnetico.

Per la trasmissione dei dati, invece, è stato utilizzato il modulo bluetooth BLE (Figura 2.1 in blu), una tecnologia di trasmissione dati basata su frequenze radio a 2.4 GHz. La banda è la stessa utilizzata dalla tecnologia Bluetooth standard, ma lo standard BLE risulta ottimizzato attraverso tempi di connessione effettivi molto brevi consentendo un alto risparmio energetico.



Board topology Top

Ref.	Description	Ref.	Description
U1	NINA-B306 Module Bluetooth® Low Energy 5.0 Module	U6	MP2322GQH Step Down Converter
U2	LSM9DS1TR Sensor IMU	PB1	IT-1185AP1C-160G-GTR Push button
DL1	Led L	DL2	Led Power

Figura 2.1: Modulo IMU LSM9DS1 (in rosso) e modulo BLE (in blu) [1]

Per programmare il microcontrollore, è stato utilizzato `Arduino IDE`.

2.1. Arduino IDE

2.1.1. Inizializzazione

Per poter sfruttare il modulo BLE integrato, è necessario dichiarare un servizio e le caratteristiche relative ai dati che vogliamo inviare. Oltre a queste variabili, vengono dichiarati come oggetti anche i sensori che si intende attivare ed utilizzare.

```
1  BLEService customService("19B10000-E8F2-537E-4F6C-D104768A1214"); //
       Definizione del servizio BLE personalizzato
2
3  // Definizione delle caratteristice per gli assi dell'accelerometro
4  BLEFloatCharacteristic xAccelCharacteristic("19B10001-E8F2-537E-4F6C-
       D104768A1214", BLERead | BLENotify);
5  BLEFloatCharacteristic yAccelCharacteristic("19B10002-E8F2-537E-4F6C-
       D104768A1214", BLERead | BLENotify);
6  BLEFloatCharacteristic zAccelCharacteristic("19B10003-E8F2-537E-4F6C-
       D104768A1214", BLERead | BLENotify);
7
8  // Definizione delle caratteristice per gli assi del giroscopio
9  BLEFloatCharacteristic xGyroscopeCharacteristic("19B10021-E8F2-537E-4
       F6C-D104768A1214", BLERead | BLENotify);
10 BLEFloatCharacteristic yGyroscopeCharacteristic("19B10022-E8F2-537E-4
       F6C-D104768A1214", BLERead | BLENotify);
11 BLEFloatCharacteristic zGyroscopeCharacteristic("19B10023-E8F2-537E-4
       F6C-D104768A1214", BLERead | BLENotify);
12
13 // Definizione delle caratteristice per gli assi del magnetometro
14 BLEFloatCharacteristic xMagCharacteristic("19B10031-E8F2-537E-4F6C-
       D104768A1214", BLERead | BLENotify);
15 BLEFloatCharacteristic yMagCharacteristic("19B10032-E8F2-537E-4F6C-
       D104768A1214", BLERead | BLENotify);
16 BLEFloatCharacteristic zMagCharacteristic("19B10033-E8F2-537E-4F6C-
       D104768A1214", BLERead | BLENotify);
17
18 BLEIntCharacteristic timeStampCharacteristic("19B10041-E8F2-537E-4F6C-
       D104768A1214", BLERead | BLENotify); // Definizione della
```

```
    caratteristica per il timestamp di misurazione
19
20 // DICHIARAZIONE DEI SENSORI UTILIZZATI
21 Nano33BLEAccelerometerData accelerometerData; // Accelerometro
22 Nano33BLEGyroscopeData gyroscopeData; // Giroscopio
23 Nano33BLEMagneticData magneticData; // Magentometro
```

Codice 2.1: Dichiarazione servizio, caratteristiche BLE e sensori

Come mostrato nel codice, le caratteristiche corrispondono ai valori misurati che intendiamo inviare alla base station. Viene quindi creata una caratteristica specifica per ogni asse (XYZ) di ciascun sensore utilizzato ed una relativa all'istante temporale nel quale viene effettivamente campionato il dato misurato dai vari sensori.

2.1.2. Setup

Nella funzione di `setup` dell'Arduino, che viene eseguita una sola volta all'inizializzazione del microcontrollore, sono state aggiunte al servizio dichiarato in precedenza le varie caratteristiche che si vogliono inviare come mostrato nel codice seguente.

```
1  BLE.setLocalName("ArduinoNano33"); // Impostazione del nome del
    dispositivo BLE
2
3  BLE.setAdvertisedService(customService); // Aggiunta del servizio
    BLE
4
5  // Aggiunta delle caratteristiche dell'accelerometro al servizio BLE
6  customService.addCharacteristic(xAccelCharacteristic);
7  customService.addCharacteristic(yAccelCharacteristic);
8  customService.addCharacteristic(zAccelCharacteristic);
9
10 // Aggiunta delle caratteristiche del giroscopio al servizio BLE
11 customService.addCharacteristic(xGyroscopeCharacteristic);
12 customService.addCharacteristic(yGyroscopeCharacteristic);
13 customService.addCharacteristic(zGyroscopeCharacteristic);
14
15 // Aggiunta delle caratteristiche del magnetometro al servizio BLE
16 customService.addCharacteristic(xMagCharacteristic);
17 customService.addCharacteristic(yMagCharacteristic);
18 customService.addCharacteristic(zMagCharacteristic);
```

```
19
20  //caratteristiche temporali
21  customService.addCharacteristic(timeStampCharacteristic); //
     Aggiunta della caratteristica temporale al servizio BLE
22
23  BLE.addService(customService); // Aggiunta del servizio BLE
24
25  BLE.advertise(); // Pubblicazione del servizio BLE
26  Serial.println("In attesa di una connessione BLE...");
```

Codice 2.2: Sezione di `setup` - aggiunta caratteristiche

Per poter sfruttare i sensori sopra esposti, è necessario inizializzarli.

```
1  // Inizializzazione dell'accelerometro
2  Accelerometer.begin();
3
4  // Inizializzazione del giroscpio
5  Gyroscope.begin();
6
7  // Inizializzazione del magnetometro
8  Magnetic.begin();
```

Codice 2.3: Funzione di `setup` - inizializzazione

2.1.3. Loop

Nella funzione `loop`, contenente la parte di codice che viene ripetuta in modo ciclico dall'Arduino, viene effettuata la lettura ed il trasferimento dei dati.

Verificato che il microcontrollore sia connesso alla base station, viene effettuato un ciclo di lettura dei valori misurati dai sensori, uno per ogni asse, che poi vengono aggiunti alla relativa caratteristica in modo da essere inviati alla base station. Oltre alle nove caratteristiche dei sensori, ne viene aggiunta una riguardante l'istante temporale delle misurazioni. Per ottenere dati sincronizzati, oltre ad effettuare un nuovo ciclo di lettura non prima di $10ms$ dal precedente, si controlla che ogni sensore abbia effettuato una nuova misurazione.

```
1 void loop() {
2  static auto lastCheck = millis();
3  BLEDevice central = BLE.central(); // Attesa della connessione al
    canale BLE da parte di un altro dispositivo
```

```
4
5   if (central) { // Connessione eseguita
6     Serial.print("Connesso a: ");
7     Serial.println(central.address());
8
9     while (central.connected()) {
10        if (millis() - lastCheck >= 10 && Accelerometer.pop(
    accelerometerData) && Gyroscope.pop(gyroscopeData) && Magnetic.pop(
    magneticData)) {
11        lastCheck = millis();
12
13        // Lettura del valore dei  3 assi dell'accelerometro
14        float xAccelValue = accelerometerData.x;
15        float yAccelValue = accelerometerData.y;
16        float zAccelValue = accelerometerData.z;
17
18        // Lettura del valore dei  3 assi del giroscopio
19        float xGyroscopeValue = gyroscopeData.x;
20        float yGyroscopeValue = gyroscopeData.y;
21        float zGyroscopeValue = gyroscopeData.z;
22
23        // Lettura del valore dei  3 assi del magnetometro
24        float xMagValue = magneticData.x;
25        float yMagValue = magneticData.y;
26        float zMagValue = magneticData.z;
27
28        // Invio sul canale BLE dei valori dei 3 assi dell'
    accelerometro
29        xAccelCharacteristic.writeValue(xAccelValue);
30        yAccelCharacteristic.writeValue(yAccelValue);
31        zAccelCharacteristic.writeValue(zAccelValue);
32
33        // Invio sul canale BLE dei valori dei 3 assi del giroscopio
34        xGyroscopeCharacteristic.writeValue(xGyroscopeValue);
35        yGyroscopeCharacteristic.writeValue(yGyroscopeValue);
36        zGyroscopeCharacteristic.writeValue(zGyroscopeValue);
37
38        // Invio sul canale BLE dei valori dei 3 assi del magnetometro
39        xMagCharacteristic.writeValue(xMagValue);
40        yMagCharacteristic.writeValue(yMagValue);
41        zMagCharacteristic.writeValue(zMagValue);
```

```
42
43          timeStampCharacteristic.writeValue(lastCheck); // Invia il
    valore del timestamp
44        }
45      }
46      Serial.println("Connessione persa");
47    }
48 }
```

Codice 2.4: Funzione di `loop`

3 | Matlab

Per la ricezione dei dati inviati da Arduino mediante Bluetooth Low Energy (BLE), abbiamo utilizzato il software Matlab installato su un computer.

L'implementazione è stata suddivisa in due fasi:

- Addestramento del modello
- Classificazione e visualizzazione dei movimenti.

Nella prima fase è stato addestrato un albero decisionale al riconoscimento di tre classi (Spalle, Bicipite, Altro).
Nella seconda fase è stato utilizzato il modello precedentemente ottenuto al fine di effettuare la classificazione in tempo reale dei dati ricavati dal microcontrollore, i quali, poi, sono stati filtrati in modo da rappresentarne graficamente le caratteristiche.

Un'operazione comune alle due fasi precedentemente descritte è il trasferimento dei dati tramite `BLE` tra microcontrollore e `Matlab`.
Si procede ad effettuare la connessione tra i due dispositivi. Ad ogni dato inviato corrisponde una caratteristica (vedi Capitolo 2) alla quale è necessario effettuare la `subscribe`.
Essendo i dati inviati come array di `4 Byte`, è necessario effettuarne la conversione, come rappresentato di seguito, al fine di utilizzarli consistentemente in `Matlab`.

```
1    dati_acc_iniz(1)=typecast(uint8(data_acc_X),'single');
2    dati_acc_iniz(2)=typecast(uint8(data_acc_Y),'single');
3    dati_acc_iniz(3)=typecast(uint8(data_acc_Z),'single');
```

Codice 3.1: Esempio: conversione dei dati grezzi dell'accelerometro

Per assicurarsi di effettuare la classificazione di una Gesture che rappresenta effettivamente un esercizio, si è impostata una soglia sul modulo dell'accelerazione, sotto la quale non viene rilevato l'inizio del movimento.

3.1. Addestramento del modello

I dati impiegati nella fase di addestramento sono di natura grezza, poiché forniscono una quantità di informazioni maggiore rispetto a quelli sottoposti a filtraggio. Nello specifico, vengono conservate la componente gravitazionale e la dinamica, sia lenta che veloce.

I dati vengono suddivisi in *Gesture*, ognuna delle quali corrisponde a 20 dati campionati di accelerometro e giroscopio. Viene utilizzato un database di 300 *Gesture*, 100 per ogni classe che intendiamo riconoscere: *Spalle*, *Bicipite* e *Altro*.

Vengono calcolate la media e la deviazione standard dei dati campionati per ogni *Gesture*, le quali rappresentano le feature utilizzate per l'addestramento dell'albero.

```
for ind = 1:100
  featureBicipite1(ind,:) = mean(trainingBicipite{ind});
  featureBicipite2(ind,:) = std(trainingBicipite{ind});

  featureSpalle1(ind,:) = mean(trainingSpalle{ind});
  featureSpalle2(ind,:) = std(trainingSpalle{ind});

  featureAltro1(ind,:) = mean(trainingAltro{ind});
  featureAltro2(ind,:) = std(trainingAltro{ind});
end

% Features
X = [featureBicipite1,featureBicipite2;
    featureSpalle1,featureSpalle2;
    featureAltro1,featureAltro2];

% Labels - 1: Bicipite, 2: Spalle, 3: Altro
Y = [ones(100,1);2*ones(100,1);3*ones(100,1)];
```

Codice 3.2: Features ed etichette di addestramento

Mediante la tecnica `Holdout`, il dataset viene suddiviso in set di addestramento (training) e di validazione (test), rispettivamente 80% e 20% del totale dei dati.

Il Codice 3.3 implementa la creazione e l'addestramento di un modello (`ensMdl`) che sfrutta la tecnica dell'*ensemble* tramite *bagging* (Bootstrap Aggregating) con alberi decisionali come learner di base. In particolare, viene utilizzato un template per la

creazione di una foresta composta da 20 alberi (`NumLearningCycles`), specificando per ciascuno un massimo numero di suddivisioni decisionali pari a 399 (`MaxNumSplits`). Le classi del modello sono identificate come 1 (bicipite), 2 (spalle) e 3 (altro).

```
template = templateTree(...
    'MaxNumSplits', 399);

ensMdl = fitcensemble(...
    XTrain, ...
    YTrain, ...
    'Method', 'Bag', ...
    'NumLearningCycles', 20, ...
    'Learners', template, ...
    'ClassNames', [1; 2; 3]);
```

Codice 3.3: Addestramento del modello

3.2. Classificazione e visualizzazione dei movimenti

In questa fase vengono letti i dati campionati dall'Arduino e, utilizzando l'albero decisionale addestrato nella fase precedente, viene effettuata la classificazione *real-time*. I dati vengono divisi in *Gesture* in modo da essere consistenti con i dati di addestramento.

Successivamente, questi vengono filtrati e rappresentati graficamente.

3.2.1. Classificazione

Le features estratte dai dati campionati da accelerometro e giroscopio sono media e deviazione standard. L'operazione di predizione, date le *features*, restituisce le probabilità di appartenenza di ogni *Gesture* a ciascuna classe.

```
feature1 = mean(dataMovimentoClassificazione{j});
feature2 = std(dataMovimentoClassificazione{j});
features = [feature1 feature2];
[y,Prob] = predict(modello,features);
```

Codice 3.4: Fase di classificazione del movimento in real-time

```
In attesa di dati...
# Gesture1
Bicipite: 0.75, Spalle: 0, Altro: 0.25
# Gesture2
Bicipite: 0.7, Spalle: 0.25, Altro: 0.05
# Gesture3
Spalle: 0.95, Bicipite: 0.05, Altro: 0
# Gesture4
Altro: 0.75, Bicipite: 0.2, Spalle: 0.05
# Gesture5
Bicipite: 0.7, Spalle: 0.05, Altro: 0.25
>>
```

Figura 3.1: Esempio di risultato ottenuto dalla classificazione

3.2.2. Filtraggio

Offset Vengono prima rimossi gli offset di accelerometro e giroscopio calcolati in modo empirico. Nello specifico viene sottratta la media dei dati campionati mantenendo l'Arduino fermo con specifici orientamenti di cui sono note le misure attese.
Successivamente, viene applicato un filtro passa banda del quarto ordine (*filtro di Butterworth*) con frequenze di taglio $f_{low} = 0.3Hz$ e $f_{high} = 2.0Hz$, per poter mantenere il più piatto possibile il modulo della risposta in frequenza nella banda passante desiderata.

```
%default method BUTTERWORTH
fpb=designfilt('bandpassiir', 'FilterOrder', 4, 'HalfPowerFrequency1',
    f_low, 'HalfPowerFrequency2',f_high,'SampleRate', 1/dt);
% FILTRAGGIO ACCELERAZIONI
dataMovimentoAcc_filt{i}=filtfilt(fpb,dataMovimentoAcc{i});
% FILTRAGGIO GYRO
dataMovimentoGyro_filt{i}=filtfilt(fpb,dataMovimentoGyro{i});
```

Codice 3.5: Filtraggio delle accelerazioni e del giroscopio

Compensazione della gravità e filtraggio Dato che l'accelerometro misura le accelerazioni lineari proprie, si è reso necessario compensare la forza di gravità, la quale viene sempre misurata dal sensore.
Tramite *filtro di Kalman* (funzione `ahrsfilter`), viene effettuato il sensor fusion tra accelerometro, giroscopio e magnetometro in modo da ottenere una stima dell'orientamento del microcontrollore. Si procede, quindi, alla compensazione andando a sottrarre

alle accelerazioni misurate la componente gravitazionale scomposta sugli assi in base all'orientamento calcolato.

```
%SENSOR FUSION
fuse = ahrsfilter('SampleRate',1/dt);
q{i} = fuse(dataMovimentoAcc{i},dataMovimentoGyro{i},dataMovimentoMag{
    i});
% COMPENSAZIONE GRAVITA'
gframe = rotateframe(q{i},gravity);
dataMovimentoAcc_comp{i} = dataMovimentoAcc{i}.*9.81 - gframe;
```

Codice 3.6: Sensor fusion e compensazione della gravità

Le accelerazioni compensante vengono filtrate utilizzando un filtro passa banda del quarto ordine, con frequenze di taglio $f_{low} = 0.6Hz$ e $f_{high} = 1.5Hz$, analogamente al Codice 3.5.

Visualizzazione grafica Si procede, infine, alla visualizzazione di tre tipologie di accelerazioni per ciascuna *gesture* rilevata:

- Accelerazioni non filtrate e non compensate ottenute direttamente dall'Arduino;
- Accelerazioni filtrate con il filtro passa banda;
- Accelerazioni compensate e filtrate.

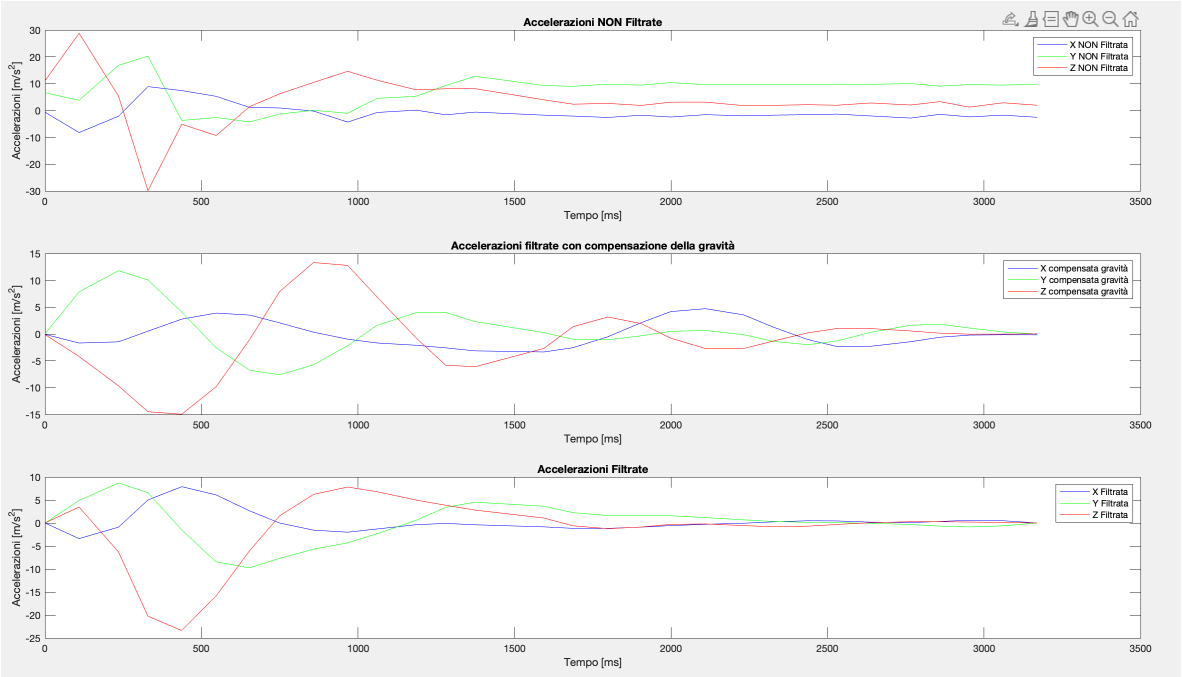


Figura 3.2: Accelerazioni relative alla Gesture riconosciuta come "Bicipite"

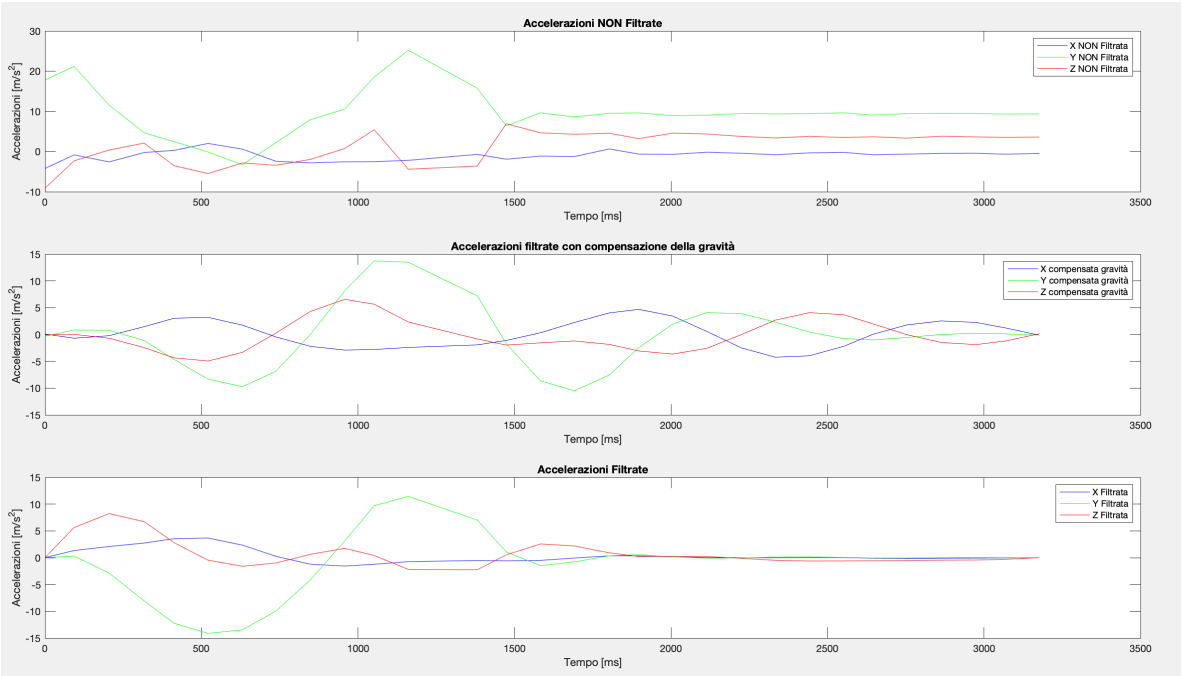


Figura 3.3: Accelerazioni relative alla Gesture riconosciuta come "Spalla"

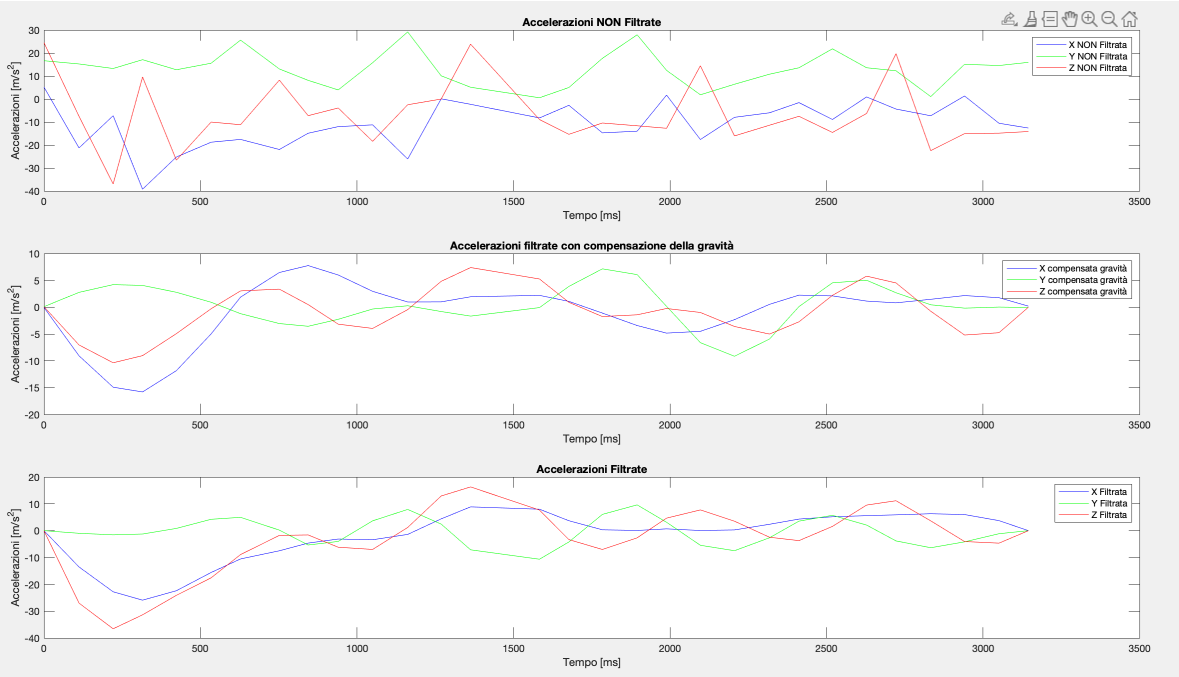


Figura 3.4: Accelerazioni relative alla Gesture riconosciuta come "Altro"

Dai grafici è possibile mettere in evidenza le differenze e le somiglianze tra le Gesture che rappresentano un esercizio per bicipite (Figura 3.2) ed uno per spalle (Figura 3.3).

- La gesture Bicipite ha prevalentemente accelerazioni sull'asse z (rosso) mentre la gesture Spalla ha prevalentemente accelerazioni sull'asse y (verde);
- Entrambe le gesture hanno una componente dell'accelerazione sull'asse x molto contenuta;
- Entrambe le gesture hanno due picchi di accelerazione, uno negativo (movimento di andata) ed uno positivo (movimento di ritorno).

4 | Conclusione

I risultati ottenuti, sebbene basati su un numero ristretto di gesture, si sono dimostrati soddisfacenti.

Sviluppi futuri Il progetto svolto offre un buon punto di partenza per una serie di sviluppi futuri come:

- l'espansione del set di gesture identificabili, introducendo nuovi movimenti per una maggiore versatilità;
- una migliore calibrazione dei sensori tramite l'utilizzo di metodi più rigorosi rispetto a quello utilizzati nell'applicazione;
- l'utilizzo di regressori più significativi, oltre alla media e alla varianza, oppure modelli più sofisticati per l'impiego in ambiti più complessi;
- l'integrazione del toolbox `MATLAB Support Package for Arduino Hardware` per un'implementazione più efficiente.

Bibliografia

[1] Arduino, Datasheet. Arduino datasheet, 2024. URL https://docs.arduino.cc/resources/datasheets/ABX00030-datasheet.pdf.

[2] Wikipedia. Bluetooth low energy, 2024. URL https://it.wikipedia.org/wiki/Bluetooth_Low_Energy.