UNIVERSITÀ DEGLI STUDI
DI GENOVA

DIPARTIMENTO DI INGEGNERIA NAVALE, ELETTRICA, ELETTRONICA E DELLE TELECOMUNICAZIONI

CORSO DI STUDIO IN INGEGNERIA ELETTRONICA E TECNOLOGIE DELL'INFORMAZIONE

Tesi di laurea triennale

Luglio 2024

# Progetto e implementazione di un sistema embedded per il monitoraggio della qualità della corsa di un atleta

# Design and development of an embedded system for monitoring the running quality of an athlete

Candidato: Incerti Simone

Relatore: Prof. Riccardo Berta

Correlatore: Dott. Matteo Fresta

# Sommario

Il presente elaborato si propone di descrivere la progettazione e l'implementazione di un sistema embedded per il monitoraggio della qualità della corsa di un atleta. Il sistema, inizialmente, raccoglie i dati rilevati dai sensori di una scheda elettronica Arduino, per poi addestrare un algoritmo di machine learning in grado di identificare diversi tipi di corsa. A tal fine, è stato utilizzato un dispositivo Arduino Nano 33 BLE Sense, che raccoglie i valori relativi all'accelerometro, al giroscopio e al magnetometro durante la corsa. Nel nostro caso, l'attenzione è stata rivolta a diversi tipi di corsa, quali la corsa ottimale e la corsa affaticata. I dati acquisiti vengono trasmessi a un dispositivo mobile connesso alla scheda elettronica tramite tecnologia Bluetooth Low Energy (BLE), utilizzando un'applicazione sviluppata con il Framework Flutter. Infine, per la raccolta dei dati, questi vengono inviati a un API Framework chiamato Measurify. Il mio obiettivo era affrontare un caso d'uso legato a una pratica sportiva comune, non necessariamente agonistica, dimostrando che con componenti hardware e software a basso costo è possibile ottenere risultati utili alla progettazione di un prototipo più strutturato e multi-sensore, capace di analizzare qualitativamente i singoli tipi di corsa.

# Indice

# 1 Introduzione

L'avvento dell'Internet of Things (IoT) ha rivoluzionato numerosi settori, incluso quello sportivo, grazie alla connessione di una vasta rete di dispositivi intelligenti. Questa trasformazione tecnologica ha significativamente migliorato la capacità di raccogliere, elaborare e utilizzare i dati in tempo reale, aprendo nuove prospettive per l'allenamento personalizzato, la prevenzione degli infortuni e l'ottimizzazione delle prestazioni atletiche. Nel mondo dello sport, l'IoT si applica attraverso l'uso di sensori avanzati e dispositivi indossabili, come i moderni smartwatch, che forniscono dati in tempo reale, come il monitoraggio della frequenza cardiaca e l'analisi dell'accelerazione. Questa tecnologia permette agli allenatori di adattare gli allenamenti alle esigenze degli atleti, migliorandone l'efficacia.

La mia ricerca in questo contesto, mira a sviluppare un sistema che applichi il machine learning nel contesto della attività della corsa.

In questo progetto sono state prese in considerazione i seguenti tipi di corsa:

- **Corsa Ottimale:** è una condizione in cui il corridore raggiunge l'equilibrio perfetto tra velocità, efficienza e comfort. In questa fase, la tecnica è corretta: il corpo è rilassato, la postura è eretta, i piedi atterrano sotto il centro di gravità e il movimento delle braccia è fluido. La respirazione è regolare e controllata, permettendo un adeguato apporto di ossigeno ai muscoli. Questo stato consente di mantenere un ritmo costante e sostenibile per lunghi periodi, minimizzando il rischio di infortuni e ottimizzando le performance.

Figura 1: Corsa ottimale

- **Corsa Affaticata:** è una condizione in cui il corridore supera i propri limiti di resistenza fisica e mentale. I segni di affaticamento includono una postura compromessa, con le spalle che si incurvano e i passi che diventano meno efficienti e più pesanti. La frequenza cardiaca aumenta e la respirazione diventa affannosa e meno efficace. I muscoli possono iniziare crampi o dolori, e il rischio di infortuni aumenta significativamente. Durante questa fase, la performance cala drasticamente e il corridore può trovarsi costretto a rallentare o fermarsi per recuperare.

Figura 2: Corsa affaticata

L'obiettivo primario è creare un sistema sofisticato in grado di riconoscere con precisione il tipo di corsa di un atleta, in relazione anche al suo livello di abilità. Utilizzando dati provenienti da sensori integrati in una cintura indossata dall'atleta, il machine learning sarà addestrato per identificare i vari tipi di corsa. Questo approccio non solo migliorerà la comprensione delle prestazioni, ma consentirà anche una personalizzazione più accurata degli allenamenti, adattandoli al livello specifico di competenza di ciascun individuo.

## 1.1 Flusso di lavoro

Il progetto è suddiviso in diverse fasi operative. Di seguito è riportato uno schema riassuntivo che illustra il processo di funzionamento del sistema embedded per il rilevamento dei tipi di corsa.

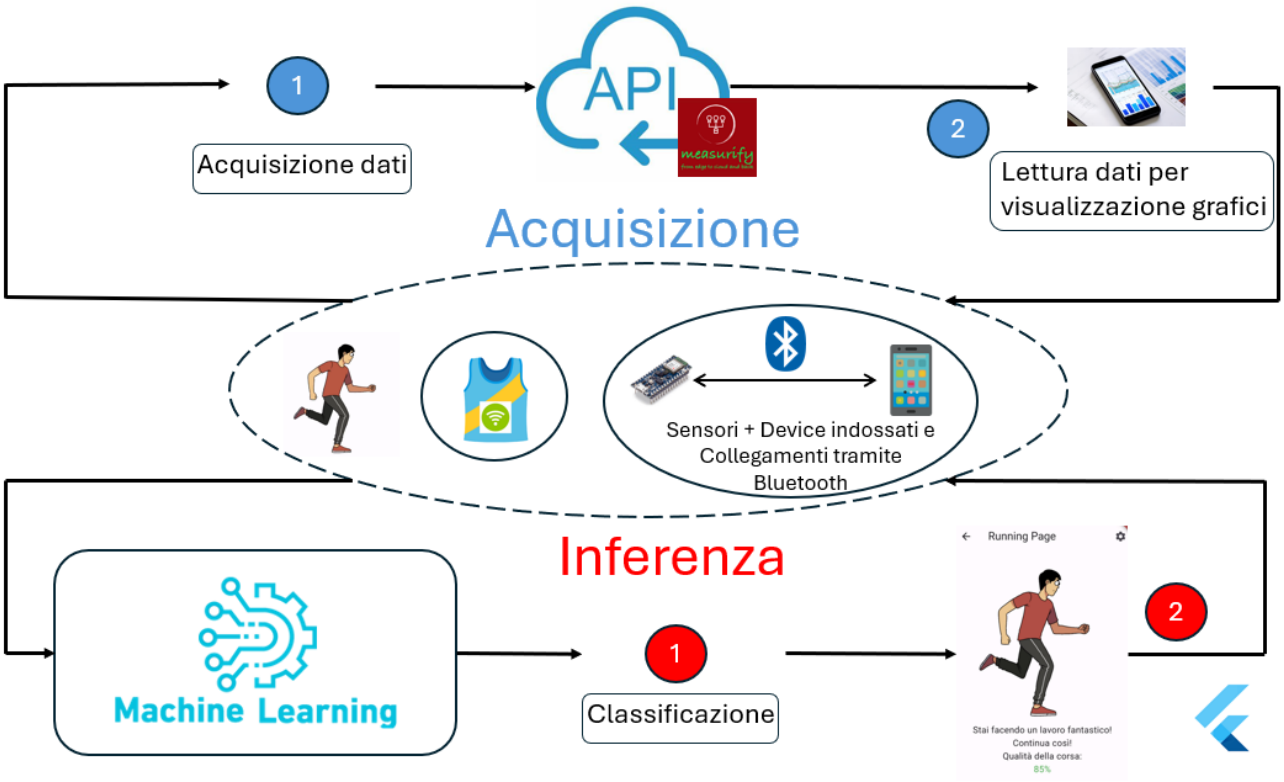



Figura 3: Schema riassuntivo

Per attuare questo approccio innovativo, viene utilizzato un sensore specializzato in grado di raccogliere dati cruciali da un accelerometro, un magnetometro e un giroscopio. Questo dati, una volta acquisiti, vengono trasmessi tramite Bluetooth Low Energy (BLE) a un dispositivo mobile dedicato. Tale dispositivo, tramite un'applicazione, consente di raccogliere i dati e di inviarli tramite API ai servizi cloud di Measurify.

In figura 3, è possibile vedere l'architettura del sistema. Questa figura è suddivisa in due sezioni principali:

- La parte superiore (Blu) rappresenta l'acquisizione dei dati. Include i passaggi dalla raccolta dei dati tramite sensori indossabili, la trasmissione via BLE al dispositivo mobile, l'invio ai servizi cloud di Measurify tramite API, e la visualizzazione dei dati attraverso l'applicazione mobile.
- La parte inferiore (Arancione) rappresenta l'inferenza. Utilizza i dati raccolti per addestrare un modello di machine learning che può classificare diversi tipi di corsa. I risultati della classificazione vengono poi utilizzati per fornire feedback agli atleti tramite l'applicazione Flutter.

# 2 Metodi e strumenti utilizzati

## 2.1 Programmazione della scheda Arduino

Il sistema in questione è una scheda elettronica sviluppata da Arduino che unisce la potenza di un microcontrollore alle funzionalità del Bluetooth Low Energy (BLE), insieme a una serie di sensori integrati che permettono di rilevare e misurare diverse grandezze ambientali e fisiche, suddivise in tre categorie: IMU (Inertial Measurement Unit), Ambiente e Orientamento. Questa combinazione di caratteristiche rende l'Arduino Nano 33 BLE Sense [1] la scelta ideale per progetti che richiedono il monitoraggio del movimento e dell'orientamento nello spazio, l'interazione con dispositivi esterni tramite connessione Bluetooth e l'elaborazione dei dati a bordo. Inoltre, grazie alle sue dimensioni ridotte (18 x 45 mm), può essere facilmente integrato nelle attrezzature tecniche sportive. Per questo scopo, il laboratorio Elios Lab ha sviluppato una scheda di supporto dotata di batteria a bottone per garantire un'alimentazione autonoma.

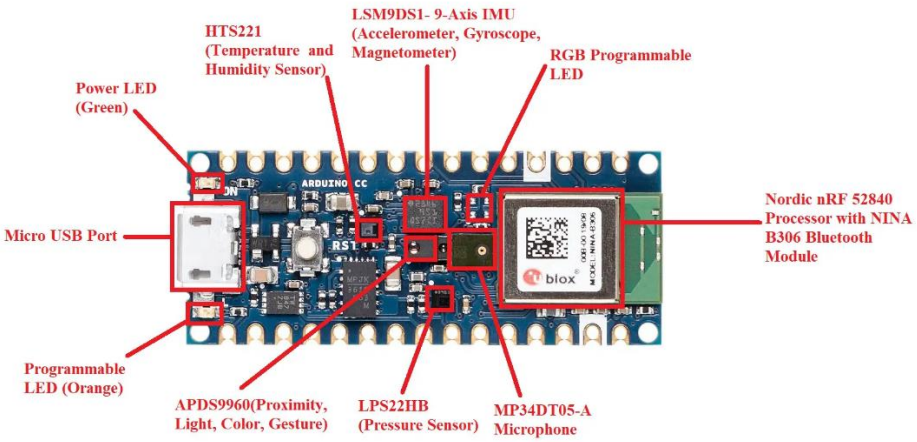



Figura 4: Arduino Nano 33 BLE Sense

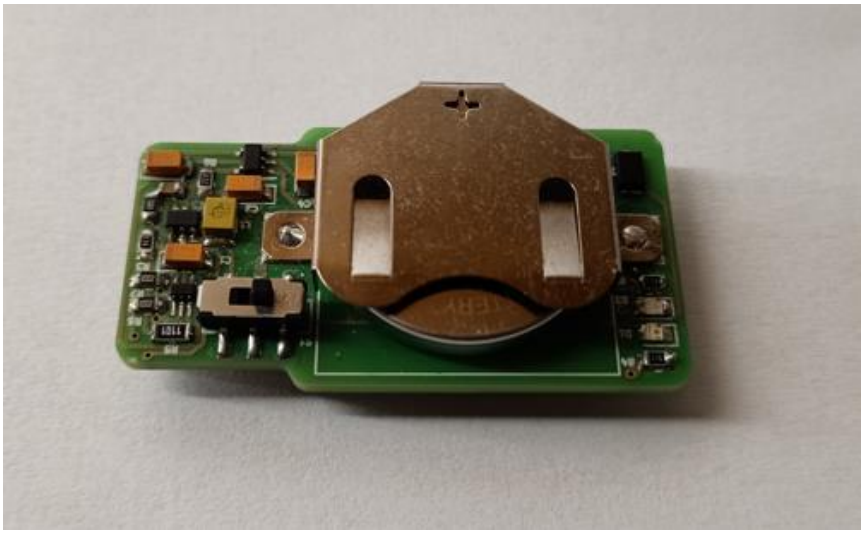


Figura 5: Arduino con batteria custom

L'IDE Arduino [2] consente di implementare e caricare il codice sulla scheda per gestire la connessione Bluetooth con il dispositivo mobile e l'invio dei dati. I valori trasmessi riguardano la caratteristica IMU, l'unica utilizzata in questo progetto, che può fornire informazioni sull'accelerazione lineare, sulla velocità angolare e sul campo magnetico, permettendo di tracciare il movimento nelle tre dimensioni dello spazio XYZ. I valori rilevati su ciascuno dei tre assi sono rappresentati da 9 numeri float; tuttavia, la connessione BLE impone un limite di trasmissione dati di 20 byte per pacchetto; quindi, si è deciso di convertire i valori da float32 a int16 per ridurre la dimensione della finestra di invio da 36 byte (9 valori x 4 byte = 36 byte) a 18 byte ( 9 valori x 2 byte = 18 byte). Questo è possibile grazie all'aggiunta nel codice di costanti moltiplicative per ciascuno dei nove valori. Un altro aspetto fondamentale è la scelta del periodo di campionamento, nel nostro caso pari a 50 millisecondi. È importante sottolineare che i valori del magnetometro, sebbene raccolti e inviati dalla scheda, non vengono presi in considerazione in quanto non rilevanti per la generazione del modello di machine learning e quindi per lo scopo della tesi.

```
19  #define ACC_MULTIPLIER 8192
20  #define GYR_MULTIPLIER 16.384
21  #define MAG_MULTIPLIER 81.92
22  #define TEMP_MULTIPLIER 100
23  #define HUM_MULTIPLIER 100
24  #define PRES_MULTIPLIER 100
25
26
27  String name = "Mesurify-Meter";
28
29  Madgwick filter;
30
31  const char* service_uuid = "8e7c2dae-0000-4b0d-b516-f525649c49ca";
32  const char* sampling_period_uuid = "8e7c2dae-0001-4b0d-b516-f525649c49ca";
33  const char* imu_uuid = "8e7c2dae-0002-4b0d-b516-f525649c49ca";
34  const char* environment_uuid = "8e7c2dae-0003-4b0d-b516-f525649c49ca";
35  const char* orientation_uuid = "8e7c2dae-0004-4b0d-b516-f525649c49ca";
36
37  BLEService service(service_uuid);
38
39  // Loop delay
40  int sampling_period = 50;
41  int heartbit_period = 5000;
42  unsigned long sampling_previousMillis = 0;
43  unsigned long heartbit_previousMillis = 0;
44  BLECharacteristic samplingPeriodCharacteristic(sampling_period_uuid, BLEWrite | BLERead, sizeof(int));
```

Figura 6: Codice Arduino raccoglimento dataset

In seguito alla gestione della raccolta dei dati, è stato sviluppato un nuovo codice per l'Arduino che consente di classificare in tempo reale la qualità della corsa. Questo codice, basato sul modello fornito dall'Elios Lab, utilizza un sistema binario per determinare la qualità della corsa: restituisce un valore di 0 se la corsa è ottimale e un valore di 1 se la corsa risulta affaticata.

Il codice utilizza l'IMU per raccogliere dati di accelerazione e velocità angolare, esegue un modello di machine learning tramite TensorFlow Lite per classificare il movimento, e comunica i risultati tramite BLE.

All'inizio del codice, in figura 7, vengono inclusi i file delle librerie necessarie per TensorFlow Lite, il supporto per l'IMU e il BLE. Vengono anche definiti moltiplicatori per l'accelerazione e la velocità angolare dell'IMU, e la dimensione dell'arena dei tensori per TensorFlow Lite.

```
1   #include <Arduino_BMI270_BMM150.h>  //#include <Arduino_LSM9DS1.h>
2   #include <ArduinoBLE.h>
3   #include "model.h"
4   #include <TensorFlowLite.h>
5   #include <tensorflow/lite/micro/all_ops_resolver.h>
6   #include "tensorflow/lite/micro/micro_log.h"
7   #include <tensorflow/lite/micro/micro_interpreter.h>
8   #include <tensorflow/lite/schema/schema_generated.h>
9
10  // Constants
11  #define ACC_MULTIPLIER 8192
12  #define GYR_MULTIPLIER 16.384
13  constexpr int kTensorArenaSize = 20 * 1024;
```

Figura 7: Inclusione delle librerie e definizione delle costanti

Successivamente, in figura 8, viene inizializzato un servizio e una caratteristica BLE per comunicare con un dispositivo esterno e vengono definite delle variabili per memorizzare dati di accelerazione e velocità angolare dell'IMU.

```
23  BLEService service("8e7c2dae-0000-4b0d-b516-f525649c49ca");
24  BLECharacteristic tiredCharacteristic("8e7c2dae-0001-4b0d-b516-f525649c49ca", BLENotify, sizeof(int16_t));
25
26  float acceleration[3];
27  float angular_speed[3];
28  int counter = 0;
```

Figura 8: Variabili globali e oggetti BLE

A questo punto, all'interno della funzione setup( ), tramite le funzioni init_sensors( ), init_BLE( ) e init_tflite( ) vengono inizializzati l'IMU, il Bluetooth e l'interprete di TensorFlow Lite.

```
94  void setup() {
95    Serial.begin(9600);
96    Serial.println("Initializing sensors...");
97    init_sensors();
98
99    Serial.println("Initializing BLE...");
100   init_BLE();
101
102   Serial.println("Initializing TensorFlow Lite...");
103   init_tflite();
104 }
```

Figura 9: Setup

Nel loop principale, se il dispositivo è connesso via BLE:

- Legge i dati dell'IMU fino a 80 campioni, ognuno a distanza di 50 ms.
- Esegue l'inferenza del modello TensorFlow Lite sui dati acquisiti.
- Invia i risultati dell'inferenza al dispositivo connesso tramite BLE.
- Misura il tempo di esecuzione dell'inferenza.

All'interno del ciclo principale, in figura 10, viene chiamata la funzione manageRawValues( ), la quale si occupa di leggere i dati di accelerazione e velocità angolare dall'IMU. Questi dati vengono quindi memorizzati nelle variabili globali.

```
106  void loop() {
107    if (BLE.connected()) {
108
109      while (counter < 80) {
110        if (BLE.connected()) {
111          manageRawValues();
112          tflInputTensor->data.f[counter * 6 + 0] = acceleration[0];
113          tflInputTensor->data.f[counter * 6 + 1] = acceleration[0];
114          tflInputTensor->data.f[counter * 6 + 2] = acceleration[0];
115          tflInputTensor->data.f[counter * 6 + 3] = angular_speed[0];
116          tflInputTensor->data.f[counter * 6 + 4] = angular_speed[0];
117          tflInputTensor->data.f[counter * 6 + 5] = angular_speed[0];
118          counter++;
119          delay(50);
120        }
121      }
122      counter = 0;
123      unsigned long startTime = micros();
124
125      if (tflInterpreter->Invoke() != kTfLiteOk) {
126        Serial.println("Inference failed");
127        while (1)
128          ;
129      }
130
131      unsigned long endTime = micros();
132      unsigned long elapsedTime = endTime - startTime;
133
134      Serial.print("Model output: ");
135      Serial.println(tflOutputTensor->data.f[0], 6);
136      Serial.println(tflOutputTensor->data.f[1], 6);
137      if (tflOutputTensor->data.f[0] > 0.50) {
138        Serial.println("Corsa Ottimale");
139        tiredCharacteristic.writeValue((int16_t)0, sizeof(int16_t));
140      }
141      if (tflOutputTensor->data.f[1] > 0.50) {
142        Serial.println("Corsa Affaticata");
143        tiredCharacteristic.writeValue((int16_t)1, sizeof(int16_t));
144      }
145
146      Serial.print("Inference time: ");
147      Serial.print(elapsedTime);
148      Serial.println(" microseconds");
149    }
150  }
```

Figura 10: Loop

## 2.2 Applicazione Flutter

Flutter [3] è un framework open-source creato da Google per lo sviluppo di applicazioni cross-platform. Questo significa che è possibile utilizzare un singolo codice sorgente per creare applicazioni che funzionano su diverse piattaforme, come Android, iOS, web e desktop. L'obiettivo principale di Flutter è offrire un metodo efficiente per costruire interfacce utente moderne e reattive. Flutter è scritto principalmente in Dart, un linguaggio di programmazione moderno sviluppato anch'esso da Google. Dart è progettato per essere facile da apprendere, con un ambiente di sviluppo efficiente e una sintassi pulita.

Dopo aver configurato l'Arduino Nano 33 BLE Sense per acquisire dati dai suoi sensori, si stabilisce una connessione Bluetooth con l'Arduino. Questo può essere fatto utilizzando il codice Flutter che gestisce la connettività BLE e la comunicazione dati. Una volta stabilita la connessione, l'app Flutter rimane in attesa di ricevere i dati dall'Arduino. L'Arduino legge i dati dai suoi sensori e li trasmette all'app Flutter tramite la connessione BLE. Questi dati nel nostro caso includono i valori dell'accelerometro, giroscopio e magnetometro lungo i tre assi.

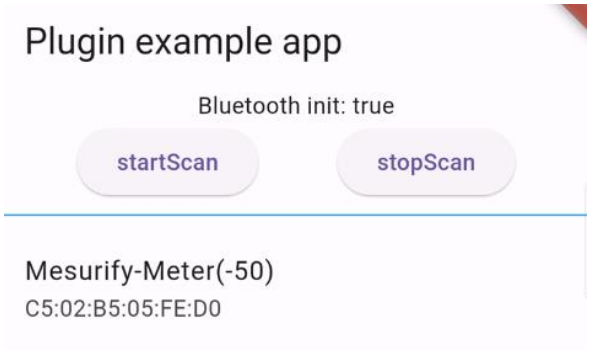



Figura 11: Connessione Bluetooth con l'Arduino

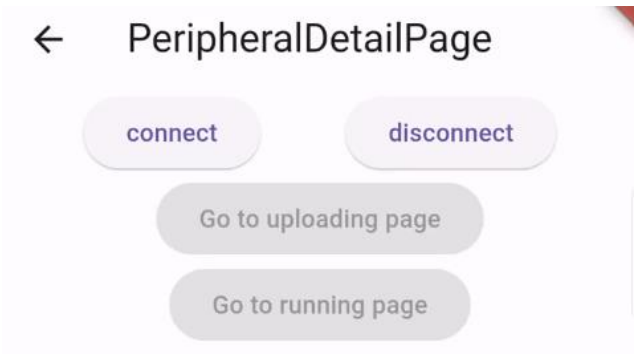



Figura 12: Peripheral Detail Page

In questo caso viene utilizzata la libreria quick_blue, un plugin BluetoothLE multipiattaforma per Flutter. L'applicazione Flutter, si chiama Smart Collector, e include alcune pagine iniziali dedicate alla connessione del dispositivo di misura.

La pagina principale, per la raccolta dei dati, è la Start Page, figura 13, dove si sceglie il tag da analizzare ( tipo di corsa), si identifica la misurazione con un nome, si seleziona la funzione IMU per raccogliere i dati relativi alle grandezze considerate e si avvia la raccolta dati attraverso il pulsante "Start". Un contatore in basso a sinistra conferma l'arrivo dei dati. Premendo il pulsante "Start" apparirà un diagramma a scorrimento che permette di visualizzare i valori raccolti in tempo reale. Premendo il tasto "Stop and Send", si interrompe la misurazione e, se tutto è andato a buon fine, appare il messaggio "Values sent correctly", indicando che i dati raccolti sono stati inviati al servizio cloud.


Figura 13: Start Page

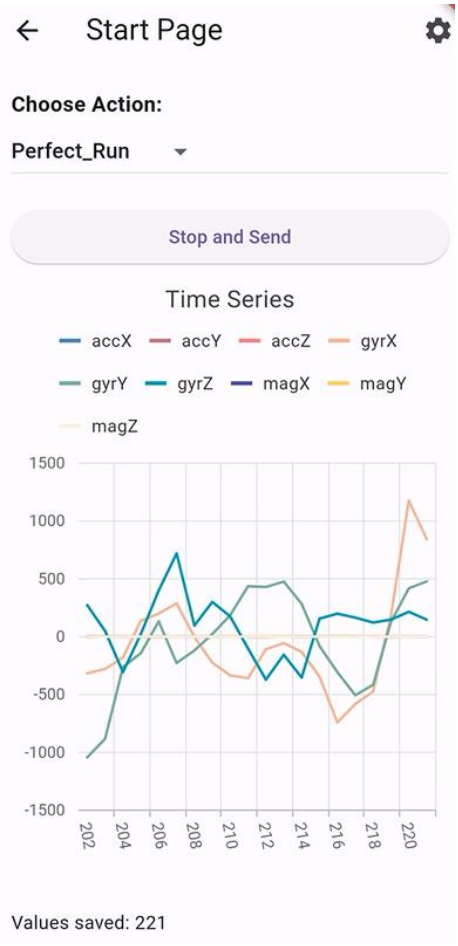



Figura 14: Start Page durante la misurazione

All'interno della Start Page, figura 13, è presente anche un pulsante "Go to chart page", che se premuto apre una nuova pagina chiamata Chart Page, in figura 15. Questa pagina offre la possibilità di consultare lo storico completo delle misurazioni effettuate dal tenant specifico, permettendo di confrontare graficamente le differenze con le serie temporali precedenti.

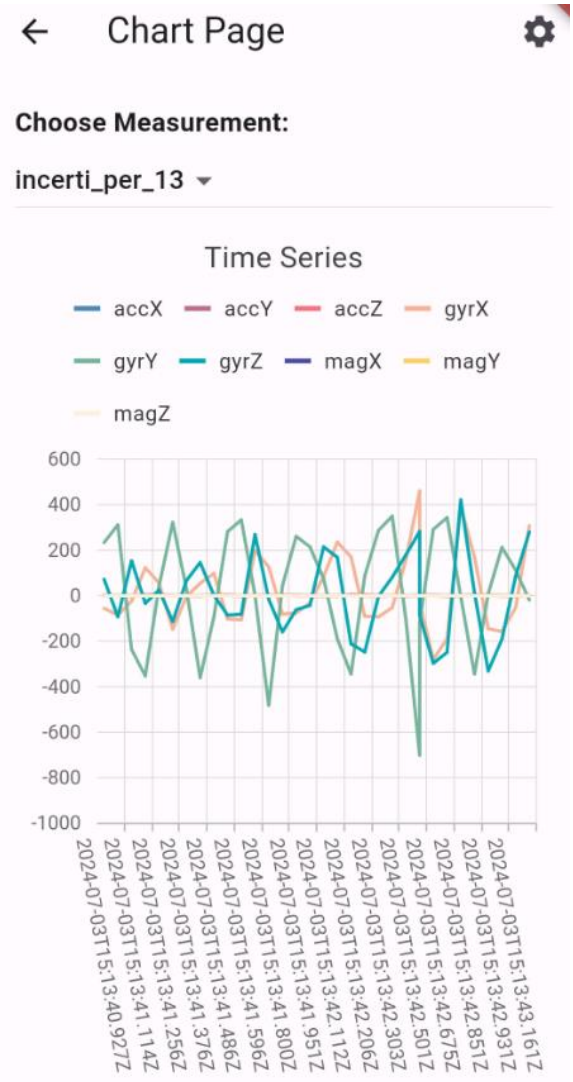



Figura 15: Chart Page

All'interno della Peripheral Detail Page, figura 12, è possibile accedere a una nuova pagina chiamata Running Page, figura 16, tramite il pulsante "Go to running page". In questa pagina, è possibile monitorare la corsa in tempo reale, poiché essa mostra una percentuale della qualità della corsa basata sulle predizioni dell'Arduino. Nel caso in cui vengano rilevate più classificazioni di corsa affaticata, la pagina notificherà al corridore di prendersi una pausa, mostrando la qualità della corsa raggiunta.

All'interno del codice Flutter, viene creato un array inizialmente vuoto. Dentro a questo array vengono salvati i valori binari basati sulle predizioni dell'Arduino. La qualità della corsa viene calcolata in base al numero degli '1', ossia quando la predizione del modello è stata corsa Affaticata, presenti all'interno di questo array. La percentuale degli '1' presenti nell'array di lunghezza 20 viene usata per calcolare la percentuale della qualità della corsa. In particolare, se tutte le valutazioni del modello sono corsa Ottimale, quindi venti '0', la qualità della corsa è al 100%. Una volta che la percentuale è scesa al di sotto del 25% il dispositivo su cui è installata l'applicazione vibrerà per un breve istante e l'immagine cambierà suggerendo all'atleta di fare una pausa.


Figura 16: Running page durante la corsa

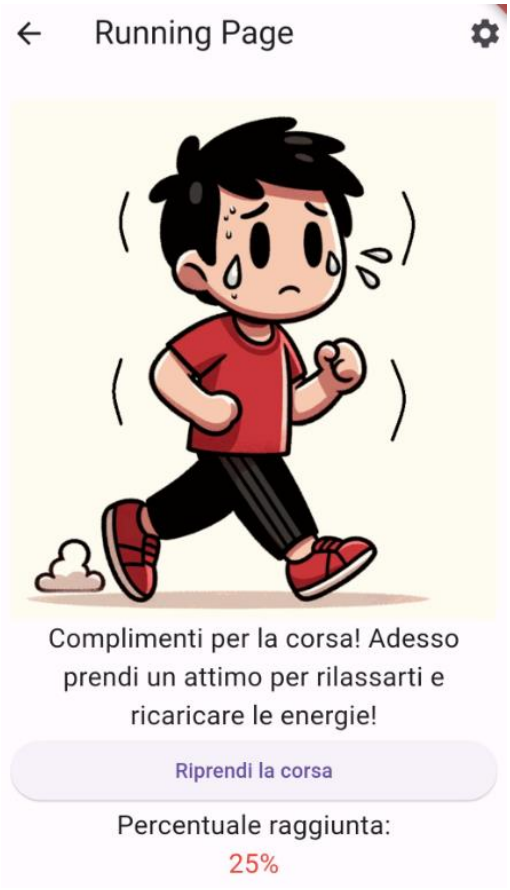



Figura 17: Running page durante la pausa

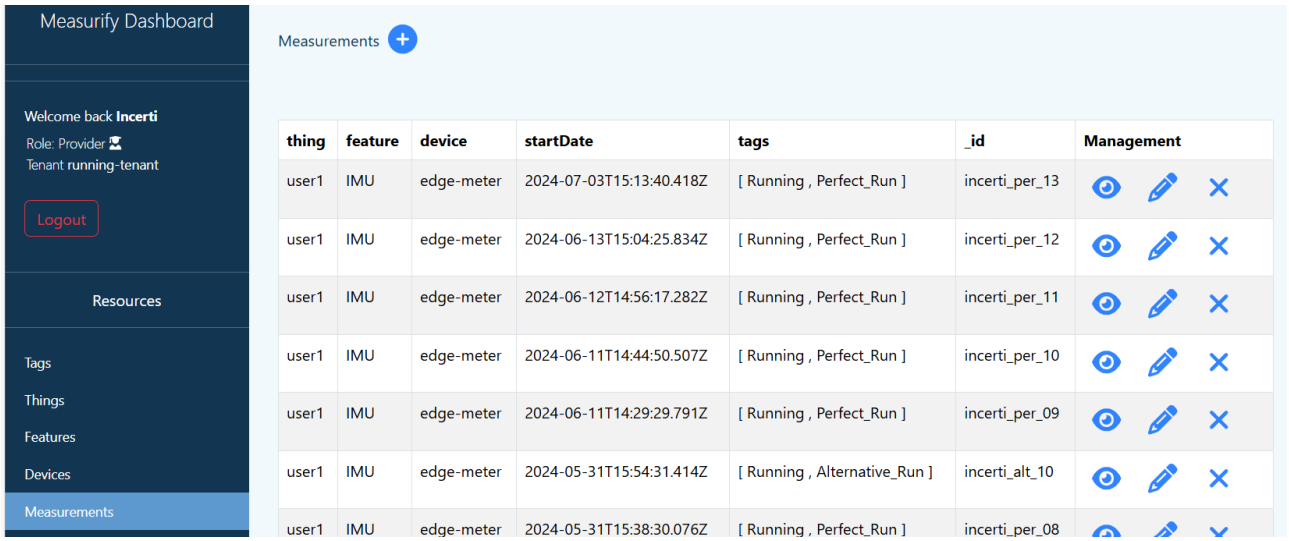
## 2.3 Measurify

Measurify [4] è un'API cloud (di tipo RESTful) sviluppata dall'Elios Lab dell'Università di Genova per la raccolta e l'elaborazione di dati provenienti da varie sorgenti, inclusi sensori IoT. Questa API è stata fondamentale come server remoto per memorizzare i dati e successivamente creare un modello di machine learning in grado di classificare autonomamente i principali tipi di corsa.

Gli elementi essenziali nell'ambiente IoT includono: Thing, Feature, Device, Measurement e Tag:

- **Thing**: il soggetto (persona o ambiente) per il quale viene effettuata una misurazione.
- **Feature**: la quantità (tipicamente fisica) che viene misurata tramite un dispositivo.
- **Device**: uno strumento (hardware o software) che fornisce misure di una certa Feature riguardante una Thing.
- **Measurement**: una misura effettuata da un dispositivo per una certa quantità su una Thing specifica.
- **Tag**: un'etichetta che può essere aggiunta a una misurazione, utile per distinguere un gruppo di misurazioni da un altro.

Nel caso specifico, il soggetto della misurazione (Thing) sarà l'atleta, che utilizzerà i sensori caratteristici dell'IMU (Feature) tramite una scheda Arduino (Device). Il risultato finale consisterà in una serie di misurazioni caratterizzate da questi tre elementi e contraddistinte da una StartDate e da uno dei 2 possibili Tag: Perfect_Run e Alternative_Run.




| thing | feature | device | startDate | tags | _id | Management |
|---|---|---|---|---|---|---|
| user1 | IMU | edge-meter | 2024-07-03T15:13:40.418Z | [ Running , Perfect_Run ] | incerti_per_13 | |
| user1 | IMU | edge-meter | 2024-06-13T15:04:25.834Z | [ Running , Perfect_Run ] | incerti_per_12 | |
| user1 | IMU | edge-meter | 2024-06-12T14:56:17.282Z | [ Running , Perfect_Run ] | incerti_per_11 | |
| user1 | IMU | edge-meter | 2024-06-11T14:44:50.507Z | [ Running , Perfect_Run ] | incerti_per_10 | |
| user1 | IMU | edge-meter | 2024-06-11T14:29:29.791Z | [ Running , Perfect_Run ] | incerti_per_09 | |
| user1 | IMU | edge-meter | 2024-05-31T15:54:31.414Z | [ Running , Alternative_Run ] | incerti_alt_10 | |
| user1 | IMU | edge-meter | 2024-05-31T15:38:30.076Z | [ Running , Perfect_Run ] | incerti_per_08 | |

Figura 18: Dashboard di Measurify [5]

Inoltre, una volta completata una misurazione, sarà possibile visualizzare i dati ottenuti in tempo reale tramite il tool Visualize Timeseries. Questo strumento consente di rappresentare graficamente tutte e 9 le variabili coinvolte, offrendo la possibilità di focalizzarsi su una o più di esse e di selezionare il numero di campioni da analizzare rispetto al totale registrato. Questa funzione di Measurify è stata cruciale nell'analisi dei tipi di corsa, effettuata dopo la raccolta dei dati, e ha giocato un ruolo fondamentale nella suddivisione dei campioni per l'addestramento del modello di machine learning.

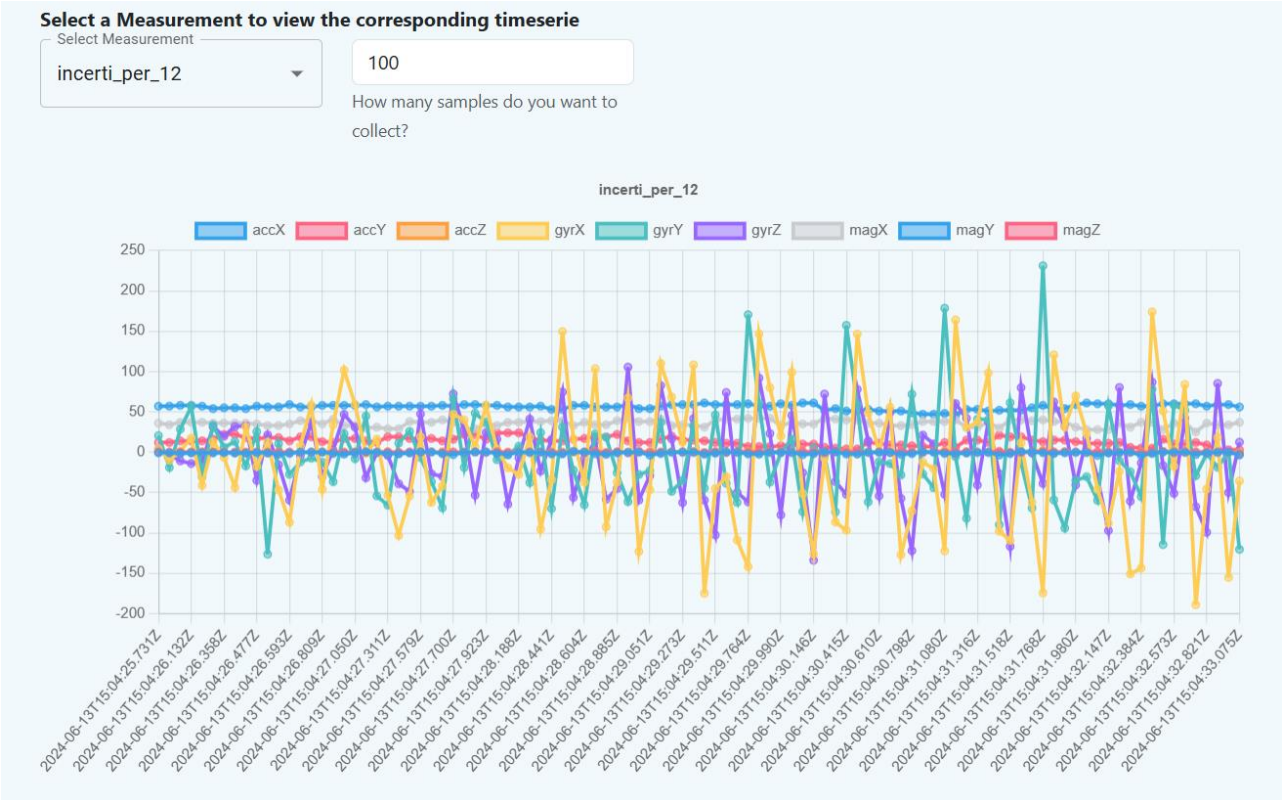



Figura 19: Visualizzazione serie temporali

# 3 Sperimentazione e risultati

## 3.1 Raccolta del dataset

Dopo aver completato lo sviluppo degli strumenti necessari per le misurazioni, sia dal punto di vista hardware che software, si è passati a testare le variabili in gioco utilizzando la strumentazione disponibile. Il primo passo è stato organizzare tutto il materiale nel modo più efficace possibile. Successivamente, il codice sorgente dell'applicazione sviluppata con Flutter è stato caricato su uno smartphone Android compatibile e ci si è assicurati che la connessione con il dispositivo Arduino funzionasse correttamente.

È stata quindi utilizzata una cintura con un taschino dedicato a contenere l'Arduino Nano, limitandone al minimo le oscillazioni e i movimenti durante la corsa, seguendo solo il movimento del corpo. Per soddisfare queste esigenze, la scheda è stata posizionata all'altezza della pancia, con l'asse delle ascisse parallelo al terreno. Infine, lo smartphone è stato collocato in una fascia da braccio portacellulare.

Figura 20: Cintura

Figura 21: Fascia da braccio

Dopo questa fase preliminare, è stato possibile passare alla fase di testing e raccolta dati vera e propria. I test sono stati condotti su due tags: Perfect_Run e Alternative_Run. Il procedimento seguito è stato il seguente: attivazione della scheda all'inizio della corsa dopo averla posizionata correttamente; connessione della scheda allo smartphone; selezione del tag da analizzare e assegnazione di un nome unico alla misurazione per evitare confusione nelle serie temporali da analizzare; registrazione dei dati; al termine della corsa, interruzione della registrazione e invio dei dati al server remoto. In totale sono state raccolte un totale di 12 corse di tipo Ottimale e 10 di tipo Affaticato per una durata media di 10 minuti ciascuna.

## 3.2 Addestramento del modello di Machine Learning

La finestra di campioni inviata all'algoritmo di Machine Learning per l'addestramento è stata impostata su una lunghezza di 80 campioni. Ogni serie temporale è stata quindi suddivisa in blocchi di 80 campioni, a ciascuno dei quali è stata assegnata la rispettiva etichetta. Ogni campione è formato da 6 assi ( 3 per l'accelerazione e 3 per il giroscopio), con un periodo di campionamento di 50 ms, il che significa una predizione ogni 4 secondi.

Il passo finale ha visto il Laboratorio Elios Lab scrivere il codice necessario per creare un modello di classificazione basato su una Rete Neurale Convoluzionale (CNN). Questa scelta è stata motivata dal consumo di memoria accettabile della CNN, essenziale per il successivo caricamento su Arduino per il riconoscimento in tempo reale.

Figura 22: Test

Il modello, sviluppato dall'Elios Lab utilizzando il dataset da me fornito, ha raggiunto un'accuratezza superiore al 90%, dimostrando ottimi risultati.

Per testare il corretto funzionamento dell'applicazione Flutter, sono andato a correre e ho verificato che tutto funzionava correttamente.

# 4 Contributo personale e considerazioni conclusive

Ho scelto questo progetto per il desiderio di unire gli studi universitari degli ultimi tre anni con una mia grande passione sportiva: la corsa. La possibilità di monitorare un atleta durante l'allenamento, concentrandomi sulle diverse tecniche di corsa, mi ha affascinato fin dall'inizio. Inoltre, ho avuto la possibilità di testare e utilizzare personalmente tutti gli strumenti necessari per il funzionamento del sistema.

Il percorso non è stato privo di difficoltà. Comprendere il sistema nella sua interezza e collegare le varie fasi, in particolare per quanto riguarda Flutter, è stato impegnativo, ma sono riuscito a risolvere il problema grazie all'aiuto del Laboratorio Elios Lab. Anche l'interpretazione e la raccolta dei dati hanno presentato delle sfide, così come le prove sul campo. Nonostante tutto, il risultato finale è promettente, poiché potrebbe permettere agli atleti di perfezionare la loro tecnica di corsa in futuro.

Il mio contributo personale all'evoluzione della piattaforma ha incluso diverse attività chiave. Ho corretto il codice e migliorato l'applicazione in generale, aggiungendo un grafico real-time visibile sull'applicazione Flutter durante la raccolta dei dataset. Ho creato un dataset di corsa in condizioni ottimali e di corsa affaticata per un totale di 12 corse Ottimali e 10 corse Affaticate.

Inoltre, ho sviluppato uno script per l'Arduino che classifica la qualità della corsa in real-time (binario: 0: Ottimale, 1: Affaticato). Ho anche creato una pagina per l'applicazione smartphone che monitora la corsa e, in caso di multiple classificazioni di corsa affaticata, notifica il corridore di riposarsi. Questa pagina mostra una percentuale della qualità di corsa in real-time basandosi sulle predizioni dell'Arduino.

Infine, ho studiato e analizzato le serie temporali per interpretare i dati raccolti e supportare il laboratorio nell'implementazione del modello di riconoscimento delle tecniche di corsa.

Questo progetto rappresenta un punto di partenza per future applicazioni. In particolare, potrebbe essere utile aumentare il numero di sensori utilizzati e gestirli meglio nelle varie fasi di raccolta dei dati. Questo permetterebbe di analizzare con maggiore precisione non solo le tecniche di corsa, ma anche le posizioni corrette da assumere durante l'allenamento, sia a livello di principiante per imparare, sia a livello agonistico per migliorare le prestazioni. Più dati diversi vengono raccolti, più ampio sarà il dataset, e il sistema potrà aggiungere un livello maggiore di affidabilità ed efficienza.

# 5 Riferimenti bibliografici


[1] Arduino Nano 33 BLE Sense Rev2: Nano 33 BLE Sense Rev2 | Arduino Documentation

[2] Arduino IDE: Software | Arduino

[3] Flutter: Flutter - Build apps for any screen

[4] Measurify: measurify.org

[5] DB Measurify: DB Dashboard (elioslab.net)

# 6 Ringraziamenti

Desidero esprimere la mia più sincera gratitudine a tutte le persone che mi hanno supportato e guidato durante la realizzazione di questa tesi.

Innanzitutto, un sentito ringraziamento al mio relatore, il Prof. Riccardo Berta, per avermi concesso l'opportunità di realizzare questo progetto e per il suo prezioso supporto.

Un ringraziamento speciale va anche al mio correlatore, il Dott. Matteo Fresta, per avermi seguito con costanza e dedizione durante tutto il percorso di tesi. La sua disponibilità e competenza sono state per me un punto di riferimento costante.

Vorrei inoltre ringraziare la mia famiglia per il loro sostegno incondizionato, la loro pazienza e il loro incoraggiamento in ogni momento. Senza il loro affetto e il loro supporto, questo traguardo non sarebbe stato possibile.

Un grazie di cuore va anche ai miei amici, per avermi supportato e motivato durante questo percorso. La vostra pazienza e il vostro incoraggiamento hanno reso più leggero questo viaggio.

A tutti voi, grazie di cuore.